قابلیت دید ضعیف پاره‌خط در محیط‌های مختلف

توسطِ

مجتبی نوری بایگی

رساله‌ی ارائه‌شده به عنوانِ بخشی از ملزومات برای دریافتِ درجه‌ی دکتری نرم‌افزار کامپیوتر

زیرِ نظرِ

دکتر محمد قدسی

بهمن ماه ۱۳۹۳

دانشکده‌ی مهندسی کامپیوتر

دانشگاهِ صنعتیِ شریف

تهران

قابلیت دید ضعیف پاره‌خط در محیط‌های مختلف

چکیده

تعیین منظره یا فضای قابل دید یک ناظر کاربردهای متعددی در هندسه‌ی محاسباتی و گرافیک کامپیوتری از جمله نورپردازی، مجموعه‌ی نگهبان و بازی‌های کامپیوتری دارد. این مسئله دارای گونه‌های مختلفی است که براساس نوع ناظر و فضای هدف طبقه‌بندی می‌شوند. مسائل قابلیت دید به شکل طبیعی در زمینه‌هایی که ابزارها و الگوریتم‌های هندسی استفاده می‌شوند، دیده می‌شود. همچنین راه‌حل مسائل قابلیت دید به عنوان عناصر سازنده در راه‌حل سایر مسائل هندسی، همچون یافتن کوتاه‌ترین مسیر یا مسائل مسیریابی مورد استفاده قرار می‌گیرد.

در این پایان‌نامه، ما به بررسی قابلیت دید ضعیف پاره‌خط می‌پردازیم. مسائلی که بدان‌ها پرداخته‌ایم عبارتند از: قابلیت دید پاره‌خط درون چندضلعی‌های ساده، قابلیت دید پاره‌خط درون چندضلعی‌های حفره‌دار، تعیین اندازه‌ی چندضلعی قابل دید پاره خط، و نگهداری قابلیت دید یک پاره‌خط در حال حرکت.

واژه‌های کلیدی: هندسه‌ی محاسباتی، قابلیت دید، چندضلعی قابل دید، قابلیت دید پاره‌خط، اندازه‌ی قابلیت دید

فهرستِ مطالب

۱ مقدمه **۱**

۱–۱ انگیزه‌ی پژوهش ۱

۱–۲ نتیجه‌های به دست آمده ۲
۱–۲–۱ محاسبه‌ی چندضلعی قابل دید ضعیف در چندضلعی‌های ساده (۱) . . . ۲
۱–۲–۲ محاسبه‌ی چندضلعی قابل دید ضعیف در چندضلعی‌های ساده (۲) . . . ۳
۱–۲–۳ محاسبه‌ی چندضلعی قابل دید ضعیف در چندضلعی‌های حفره‌دار . . . ۳
۱–۲–۴ اندازه‌ی چندضلعی قابل دید ضعیف ۴
۱–۲–۵ قابلیت دید پاره‌خط متحرک ۴
۱–۲–۶ نمودار قطبی ۵

۱–۳ طرح رساله ۵

۲ مفاهیم و تعاریف **۶**

۲–۱ قابلیت دید ۷

۲–۲ تجزیه قابلیت دید ۹

۲–۳ قابلیت دید پاره‌خط ۱۱

۲–۴ جستجوی دامنه‌ای ۱۳

۲–۵ دوگانگی نقطه-خط ۱۴

۳ قابلیت دید پاره‌خط در چندضلعی ساده ۱۵

۳-۱ الگوریتم خطی برای محاسبه WVP ۱۵

۳-۲ الگوریتم پرس‌وجوی محاسبه‌ی چندضلعی قابل دید ضعیف ۱۶
۳-۲-۱ محاسبه‌ی درخت کوتاه‌ترین مسیر به شکل حساس به خروجی ۱۶
۳-۲-۲ محاسبه‌ی چندضلعی قابل دید ضعیف به شکل حساس به خروجی . . ۱۸
۳-۲-۳ بهبود الگوریتم ۲۰

۳-۳ خلاصه ۲۴

۴ الگوریتم دوم محاسبه‌ی WVP ۲۵

۴-۱ چندضلعی قابل دید جزئی ۲۵

۴-۲ محاسبه‌ی درخت کوتاه‌ترین مسیر جزئی $SPT_L(p)$ ۲۶

۴-۳ محاسبه‌ی $WVP_L(pq)$ ۲۹

۴-۴ محاسبه‌ی $WVP_L(pq)$ برای پاره‌خط‌های گسترش یافته ۳۳

۴-۵ محاسبه‌ی WVP با استفاده از مثلث‌بندی متوازن ۳۳
۴-۵-۱ تحلیل زمان پرس‌وجو ۳۸

۴-۶ خلاصه ۳۸

۵ محاسبه‌ی WVP در چندضلعی‌های حفره‌دار ۴۰

۵-۱ محاسبه‌ی قابلیت دید از طریق قطرهای برشی ۴۱

۵-۲ الگوریتم ۴۲

۵-۳ بهبود الگوریتم ۴۴

۵-۴ محاسبه‌ی مرز $WVP(pq)$ ۴۸

۵-۵ خلاصه ۴۸

۶ اندازه‌ی چندضلعی قابل دید ضعیف ۵۰

۶-۱ مقدمه ۵۰

۶-۲ کاربردها و انگیزه‌ها ۵۱

۶-۳ اندازه‌ی دقیق قابلیت دید ضعیف ۵۲
۶-۳-۱ الگوریتم با بهترین زمان پرس‌وجو ۵۴
۶-۳-۲ کاهش هزینه‌های پیش‌پردازش ۵۴
۶-۳-۳ برقراری توازن بین هزینه‌ی پیش‌پردازش و زمان پرس‌وجو ۵۷

۶-۴ اندازه‌ی تقریبی قابلیت دید ضعیف ۵۷
۶-۴-۱ امتحان قابلیت دید ضعیف ۵۸
۶-۴-۲ نمونه‌برداری تصادفی ۵۸

۶-۵ خلاصه ۶۰

۷ قابلیت دید پاره‌خط متحرک ۶۱

۷-۱ مقدمه ۶۱

۷-۲ قابلیت دید پاره‌خط ایستا ۶۱

۷-۳ قابلیت دید پاره‌خط متحرک ۶۲
۷-۳-۱ رخداد گذر ۶۴
۷-۳-۲ رخداد تقاطع ۶۶

۷-۴ محاسبات محلی برای پردازش رخدادها ۶۸
۷-۴-۱ نقشه توپولوژیکی ۶۸
۷-۴-۲ قابلیت دید ایستا ۷۰
۷-۴-۳ قابلیت دید پویا ۷۲

۷-۵ خلاصه ۷۲

۸ نتیجه‌گیری ۷۴

۸-۱ نتایج به دست آمده ۷۴

۸-۲ پژوهش‌های آینده و مسائل باز ۷۴

فهرستِ اشکال

۲-۱ چند ضلعی ساده و حفره‌دار ۶
۲-۲ قابلیت دید در چندضلعی ساده و حفره‌دار ۷
۲-۳ چندضلعی قابل دید از نقطه و پاره‌خط ۸
۲-۴ uu' یک پاره‌خط بحرانی از چندضلعی است. ۹
۲-۵ تجزیه قابلیت دید چند ضلعی ساده. ۱۰
۲-۶ گراف دوگان جهت‌دار متناظر با تجزیه قابلیت دید. ۱۱
۲-۷ قابلیت دید قوی ۱۲
۲-۸ قابلیت دید ضعیف ۱۲
۲-۹ دوگان پرتوی. ۱۴

۳-۱ الگوریتم خطی محاسبه‌ی چندضلعی قابل دید ضعیف ۱۶
۳-۲ درخت کوتاه‌ترین مسیر نقطه‌ی p و انواع یال‌های آن ۱۷
۳-۳ حساس به خروجی نبودن الگوریتم ۱۹
۳-۴ موقعیت pq درون چندضلعی ۲۱
۳-۵ تغییر ساختار $SPT(p)$ با حرکت p ۲۲
۳-۶ تغییر اطلاعات بحرانی رئوس نسبت به p با حرکت آن در چندضلعی ۲۳
۳-۷ تغییر عدد بحرانی رئوس حرکت نقطه‌ی مبداء در چندضلعی ۲۴

۴-۱ قابلیت دید ضعیف جزئی پاره‌خط ۲۶
۴-۲ انواع یال‌ها در درخت کوتاه‌ترین مسیر ۲۷
۴-۳ تغییر ناحیه‌ی قابلیت دید ریشه درخت کوتاه‌ترین مسیر ۲۸
۴-۴ محل قرارگیری pq در محاسبه‌ی $WVP_L(pq)$ ۳۰
۴-۵ تغییرا وضعیت بحرانی در حرکت بین ناحیه‌های قابلیت دید ۳۱
۴-۶ عدد بحرانی راس‌ها ۳۲
۴-۷ قابلیت دید جزئی برای پاره‌خط‌های گسترش یافته ۳۳
۴-۸ مثلث‌بندی متوازن ۳۴
۴-۹ حالت ابتدایی در محاسبه‌ی $WVP_L(pq)$ ۳۴

۴-۱۰ درخت متناظر با مثلث‌بندی متوازن. ۳۵
۴-۱۱ محاسبه‌ی بازگشتی قابلیت دید پاره‌خط (۱) ۳۶
۴-۱۲ محاسبه‌ی بازگشتی قابلیت دید پاره‌خط (۲) ۳۶
۴-۱۳ بدترین حالت در محاسبه‌ی $WVP(pq)$ ۳۷
۴-۱۴ نمودار $\kappa \log^{۲}(n/\kappa)$ برای چند مقدار n. ۳۸

۵-۱ تبدیل چندضلعی حفره‌دار به چندضلعی ساده ۴۱
۵-۲ محاسبه‌ی $WVP(pq)$ درون یک چندضلعی حفره‌دار ۴۲
۵-۳ مثالی از حد بالای h' ۴۳
۵-۴ ایجاد ساختار پرتوافکنی برای خطوط عبوری از یک نقطه ۴۵
۵-۵ خطوط بحرانی از یک راس چندضلعی ۴۵
۵-۶ رابطه‌ی پاره‌خط‌های بحرانی و نقاط قابل دید ۴۷
۵-۷ قابلیت دید یکسان نقاط داخل سلول‌ها ۴۸

۶-۱ ساده‌سازی مسیر در چندضلعی ساده ۵۲
۶-۲ مشخص شدن قابلیت دید ضعیف پاره‌خط توسط ناحیه‌های دو سر آن ۵۳
۶-۳ محاسبه‌ی اندازه‌ی چندضلعی قابل دید ضعیف ۵۵
۶-۴ شمارش قابلیت دید در فضای دوگان ۵۶

۷-۱ یک ناظر پاره‌خط بین تعدادی شیء محدب. ۶۲
۷-۲ ناظر درون تجزیه‌ی قابلیت دید. ۶۳
۷-۳ رخداد گذر (بالا) و رخداد تقاطع (پایین). ۶۴
۷-۴ آرایش فضای دوگان قبل و بعد از رخداد گذر. ۶۵
۷-۵ وضعیت صفحه اصلی و صفحه دوگان در رخداد تقاطع. ۶۷
۷-۶ نقشه‌ی توپولوژیکی برای مجموعه‌ای از پاره‌خط‌ها. ۶۹
۷-۷ همسایگی‌های مختلف در نقشه‌ی توپولوژیکی. ۶۹
۷-۸ خط بحرانی ایجاد شده توسط سلول C در نقشه‌ی توپولوژیکی. ۷۰
۷-۹ تغییرات ممکن در نقشه‌ی توپولوژیکی. ۷۱

۸-۱ قابلیت دید آلفا ۷۵
۸-۲ سایه‌های نرم ۷۵
۸-۳ موزه‌ی هنر با وجود نگهبان‌های پاره‌خطی ۷۶

فصلِ ۱

مقدمه

یکی از مسائلی که به طور گسترده در زمینه‌های مختلف مرتبط با کامپیوتر، مانند گرافیک کامپیوتری و مسیریابی روبات‌ها مطرح می‌گردد، قابلیت دید اشیاء مختلف نسبت به هم است. دو شیء یک‌دیگر را می‌بینند و نسبت به هم قابل دید هستند، اگر خطی وجود داشته باشد به طوری که دو شیء را به هم وصل کند و در میان راه به مانعی برخورد نکند. اگر یکی از اشیاء را منبع نور بدانیم، قابلیت دید منبع نور نسبت به یک شیء دیگر به معنی رسیدن روشنایی به آن شیء است و در سایه قرار گرفتن معادل با قابل دید نبودن خواهد بود. این تناظر بین مسئله‌ی قابلیت دید که بیشتر در هندسه‌ی محاسباتی مورد بررسی قرار می‌گیرد و مسئله‌ی محاسبه‌ی سایه‌ها که در گرافیک کامپیوتری مطرح می‌گردد، باعث ایجاد کاربرد مستقیم مسئله‌ی قابلیت دید در گرافیک کامپیوتری و ترسیم فضاهای مجازی در کامپیوتر می‌شود. همچنین خود مفهوم قابلیت دید در کاربردهای مهمی مانند روباتیک، معماری و ارتباط راه دور استفاده می‌گردد. در این رساله، نتایجی را که در زمینه‌ی قابلیت دید در مدت تحقیقات کسب نموده‌ایم ارائه می‌نماییم.

۱-۱ انگیزه‌ی پژوهش

تعیین ناحیه‌ی قابل دید از یک ناظر واقع در یک فضای دلخواه، یکی از مسائل قدیمی در علوم کامپیوتر است که کاربردهای متعددی از جمله گرافیک و متحرک سازی دارد. در این مساله، برای یک ناظر نقطه‌ای، هدف یافتن تمام نقاطی از فضای مورد نظر است که با قرار دادن یک منبع نور در آن نقطه روشن می‌شوند.

دو نقطه درون یک چندضلعی درصورتی نسبت به هم قابل دید هستند که پاره‌خط متصل کننده‌ی آن دو کاملا درون چندضلعی قرار داشته باشد. به همین شکل، چندضلعی قابل دید یک نقطه q درون یک چندضلعی $\mathcal{P}$، یا $V(q)$، عبارت است از مجموعه نقاط $\mathcal{P}$ که از q قابل دید هستند. مطالعات فراوانی بر روی محاسبه‌ی چندضلعی قابل دید درون یک چندضلعی ساده انجام شده است. در یک چندضلعی ساده با n رأس، $V(q)$ را می‌توان در زمان $O(n)$ به دست آورد [۱۸].

هرچند این الگوریتم بهینه است، ولی حساس به خروجی نیست و برای اجرای آن به زمان $O(n)$ احتیاج است. Bose و دیگران [۶] نشان دادند که با پیش‌پردازش چندضلعی ساده در زمان $O(n^3 \log n)$ و

حافظه $O(n^۳)$، چندضلعی قابل دید را می‌توان در زمان حساس به خروجی $O(\log n + |V(q)|)$ محاسبه کرد. این نتیجه توسط Aronov و دیگران [۲] گسترش یافت، به‌گونه‌ای که با پیش‌پردازش زمانی $O(n^۲ \log n)$ و حافظه $O(n^۲)$، چندضلعی قابل دید نقطه در زمان $O(\log^۲ n + |V(q)|)$ قابل محاسبه است.

برای گونه‌های دیگر این مسئله برحسب نوع ناظر و نیز محیط مورد نظر، الگوریتم‌های مختلفی ارائه شده است. یکی از معروف‌ترین این گونه‌ها، قابلیت دید یک پاره‌خط است. قابلیت دید یک پاره‌خط نسبت به یک نقطه به صورت قابلیت دید قوی و قابلیت دید ضعیف تعریف می‌گردد. نقطه‌ی a نسبت به پاره‌خط pq قابل دید قوی است اگر a همه‌ی نقاط pq را ببیند، و قابل دید ضعیف است اگر حداقل یک نقطه از پاره‌خط pq را ببیند.

۱-۲ نتیجه‌های به دست آمده

هدف این رساله، بررسی مسئله‌ی قابلیت دید پاره‌خط در محیط‌های مختلف است. این مسائل شامل قابلیت دید پاره‌خط درون چندضلعی‌های ساده و حفره‌دار، و محاسبه‌ی اندازه‌ی این چندضلعی می‌باشد. همچنین مسئله‌ی نگهداری قابلیت دید پاره خط متحرک مورد بررسی قرار گرفته است. در زیر این نتایج به همراه توضیحات مختصری عنوان می‌شوند.

۱-۲-۱ محاسبه‌ی چندضلعی قابل دید ضعیف در چندضلعی‌های ساده

در وهله‌ی اول به بررسی مسئله‌ی محاسبه‌ی چندضلعی قابل دید پاره‌خط درون چندضلعی‌های ساده، به شکل حساس به خروجی می‌پردازیم. در الگوریتم ارائه شده، چندضلعی $\mathcal{P}$ با n راس را در زمان $O(n^۳ \log n)$ و با صرف فضای $O(n^۳)$ پیش‌پردازش می‌کنیم. سپس با ورود یک پاره‌خط جستجو درون $\mathcal{P}$، چندضلعی قابل دید ضعیف آن را در زمان $O(\log n + |\mathrm{WVP}(pq)|)$ گزارش می‌کنیم، که $|\mathrm{WVP}(pq)|$ اندازه‌ی چندضلعی قابل دید ضعیف می‌باشد. بهترین الگوریتمی که پیش از آن برای این مسئله با هزینه‌ی پیش‌پردازش یکسان وجود داشت، الگوریتمی است که توسط Bose و دیگران ارائه شده است [۶] و پاسخ را در زمان $O(|\mathrm{WVP}(pq)| \log n)$ به دست می‌دهد. برای مقایسه‌ی این نتیجه با کارهای مشابه به جدول ۱-۱ مراجعه کنید.

صورت اولیه‌ی این نتیجه در کنفرانس Canadian Conference on Computational Geometry 2011 ارائه شده است. همچنین این الگوریتم به همراه الگوریتم محاسبه‌ی چندضلعی قابل دید ضعیف در چندضلعی‌های حفره‌دار در مجله‌ی International Journal of Computational Mathematics چاپ شده است.

ارائه کنندگان	زمان پیش‌پردازش	حافظه	زمان پرس‌وجو
Bose و همکاران (۲۰۰۰)	$O(n^3 \log n)$	$O(n^3)$	$O(k \log n)$
Aronov و همکاران (۲۰۰۲)	$O(n^2 \log n)$	$O(n^2)$	$O(k \log^2 n)$
الگوریتم اول ما (فصل ۳)	$O(n^3 \log n)$	$O(n^3)$	$O(k + \log n)$
الگوریتم دوم ما (فصل ۴)	$O(n^2 \log n)$	$O(n^2)$	$O(\kappa \log^2(\frac{n}{\kappa}) + k)$

جدولِ ۱–۱: الگوریتم‌های محاسبه‌ی چندضلعی قابل دید ضعیف پاره‌خط درون چندضلعی ساده. n برابر با اندازه‌ی چندضلعی ساده، k اندازه‌ی خروجی، و κ مقداری وابسته به خروجی و حداکثر برابر k است.

ارائه کنندگان	زمان پیش‌پردازش	حافظه	زمان پرس‌وجو
Suri و O'Rourke (۱۹۸۶)	–	–	$O(n^2 \log n + k)$
الگوریتم ما (فصل ۵)	$O(n^2 \log n)$	$O(n^2)$	$O(nh' \log n + k)$

جدولِ ۱–۲: الگوریتم‌های محاسبه‌ی چندضلعی قابل دید ضعیف پاره‌خط درون چندضلعی حفره‌دار. n اندازه‌ی چندضلعی حفره‌دار، h تعداد حفره‌ها، h' تعداد حفره‌های قابل دید، و $k = O(n^2 h^2)$ اندازه‌ی خروجی می‌باشد.

۱–۲–۲ الگوریتم دوم برای محاسبه‌ی چندضلعی قابل دید ضعیف در چندضلعی‌های ساده

در الگوریتم دوم، به کاهش زمان پیش‌پردازش الگوریتم قبل پرداختیم. این کاهش زمان پیش‌پردازش به افزایش زمان پرس‌وجو منجر شده است. به طور دقیق‌تر، با هزینه‌ی پیش‌پردازش در زمان $O(n^2 \log n)$ و حافظه‌ی $O(n^2)$، قادر خواهیم بود که چندضلعی قابل دید ضعیف را در زمان $O(\kappa \log^2(\frac{n}{\kappa}) + |\mathrm{WVP}(pq)|)$ گزارش کنیم، که در آن $|\mathrm{WVP}(pq)|$ اندازه‌ی خروجی و κ عددی وابسته به پاره‌خط ورودی و کمتر از $|\mathrm{WVP}(pq)|$ می‌باشد. بهترین الگوریتم موجود با این هزینه‌ی پیش‌پردازش متعلق به Aronov و دیگران [۲] است که جواب را در زمان $O(|\mathrm{WVP}(pq)| \log^2 n)$ به دست می‌دهد. برای مقایسه‌ی این نتیجه با کارهای مشابه به جدول ۱–۱ مراجعه کنید.

۱–۲–۳ محاسبه‌ی چندضلعی قابل دید ضعیف در چندضلعی‌های حفره‌دار

در این کار که در واقع گسترش نتایج به دست آمده در چندضلعی‌های ساده است، به پیش‌پردازش چندضلعی حفره‌دار برای پاسخ به مسئله‌ی محاسبه‌ی چندضلعی قابل دید ضعیف می‌پردازیم. الگوریتم بهینه‌ای که برای این مسئله وجود دارد [۴۲]، پاسخ را در زمان $O(n^2 \log n + k')$ به دست می‌دهد که در آن $k' = O(n^4)$. برای یک چندضلعی حفره‌دار با h حفره و تعداد کل n راس، الگوریتم توسعه داده شده توسط ما دارای زمان پیش‌پردازش $O(n^2 \log n)$ و حافظه‌ی $O(n^2)$ است. در زمان پرس‌وجو، $\mathrm{WVP}(pq)$ را می‌توان در زمان $O(nh' \log n + k)$ به دست آورد. در این جا h' مقداری حساس به خروجی و حداکثر

	زمان پیش‌پردازش	حافظه	زمان پرس‌وجو
ابتدایی	–	$O(n)$	$O(n)$
دقیق	$O(n^{۷})$	$O(n^{۶})$	$O(\log n)$
دقیق	$O(n^{۴+\epsilon})$	$O(n^{۴+\epsilon})$	$O(\log^{۵} n)$
دقیق	$O(n^{۳+\epsilon})$	$O(n^{۳})$	$O(n^{۱/۲+\epsilon})$
تقریبی	$O(n^{۲})$	$O(n^{۲})$	$O(n^{۱/۲+\epsilon})$

جدولِ ۱–۳: خلاصه‌ای از نتایج به دست آمده در مسئله‌ی شمارش قابلیت دید ضعیف. بخش اول الگوریتم‌های دقیق به دست آمده را نشان می‌دهد و بخش دوم نتیجه‌ی الگوریتم تقریبی را بیان می‌کند.

$1 + \min(h, k)$ است، و $k = O(n^{۲}h^{۲})$ اندازه چندضلعی خروجی می‌باشد. تا جایی که ما اطلاع داریم این بهترین الگوریتم پرس‌وجوی ارائه شده برای این مسئله می‌باشد. برای مقایسه‌ی این نتیجه با کارهای مشابه به جدول ۱–۲ مراجعه کنید.

۱–۲–۴ اندازه‌ی چندضلعی قابل دید ضعیف

در این مسئله به محاسبه‌ی اندازه‌ی چندضلعی قابل دید یک پاره خط درون یک چندضلعی ساده می‌پردازیم. این کار را می‌توان با استفاده از الگوریتم‌های محاسبه‌ی چندضلعی قابل دید ضعیف در زمان $O(n)$ و محاسبه‌ی اندازه‌ی آن انجام داد. برای حل این مسئله در زمان زیرخطی چند الگوریتم دقیق و یک الگوریتم تقریبی ارائه کرده‌ایم. این الگوریتم‌های به دست آمده را می‌توان در جدول ۱–۳ مشاهده کرد.

۱–۲–۵ قابلیت دید پاره‌خط متحرک

آخرین مسئله‌ای که درباره‌ی قابلیت دید پاره‌خط به آن پرداخته‌ایم، قابلیت دید پاره‌خط در صفحه‌ای شامل n شیء است که این اشیاء می‌توانند پاره‌خط و یا هر شیء محدبی باشند. در این کار، الگوریتمی برای محاسبه‌ی قابلیت دید پاره‌خط بر اساس قابلیت دید نقطه ارائه می‌کنیم و سپس نشان می‌دهیم که با حرکت کردن پاره‌خط، چه رخدادهایی اتفاق می‌افتند. پردازش این رخدادها و نگهداری قابلیت دید از دیگر کارهایی است که به آن پرداخته‌ایم.

در حالت ایستا، الگوریتمی ارائه می‌کنیم که قابلیت دید پاره‌خط را با پیش‌پردازش $O(m + n \log n)$ و در زمان پرس‌وجوی $O((n+k)\log n)$ به‌دست می‌دهد، که n تعداد اشیاء حاضر در صحنه، $m = O(n^{۲})$ اندازه‌ی گراف قابلیت دید اشیاء، و k تعداد تغییرات قابلیت دید بر روی پاره‌خط می‌باشد. در حالت پویا، یعنی زمانی که ناظر در صفحه حرکت می‌کند، نشان داده‌ایم که چگونه رخدادهای گذر و تقاطع، ساختار قابلیت دید پاره‌خط را تغییر می‌دهند و نحوه‌ی تشخیص این رخدادها و پردازش آن‌ها را توضیح داده‌ایم. پردازش هر کدام از این رخدادها در زمان $O(\log n)$ امکان‌پذیر می‌باشد.

نتایج به دست آمده از این مطالعات در کنفرانس‌های EuroCG'09 و International Conference on Computational Science and Applications 2009 ارائه شده‌اند.

۱-۲-۶ نمودار قطبی

علاوه بر مطالعات بالا که در زمینه‌ی پدیداری پاره‌خط می‌باشند، پژوهش دیگری در زمینه‌ی نمودار قطبی جنبشی صورت گرفته است که در کنفرانس CCCG'08 و کنفرانس CSICC'08 ارائه گردیده است، که به علت جدا بودن موضوع این پژوهش از سایر مطالعات، از آوردن آن در این گزارش خودداری شده است.

۱-۳ طرح رساله

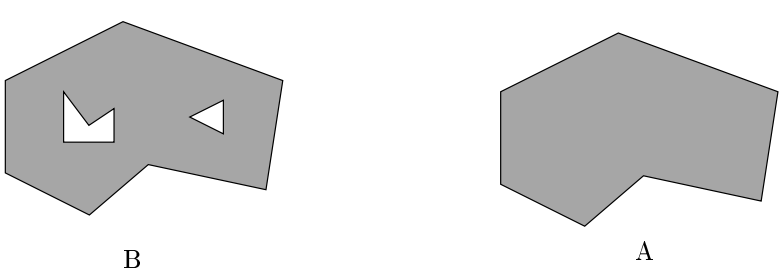
در ادامه رساله، ابتدا در فصل ۲ مفاهیم و تعاریف اصلی مورد نیاز در زمینه قابلیت دید و همچنین داده‌ساختارهایی که در مسائل حل شده مورد استفاده قرار گرفته‌اند، به اختصار توضیح داده می‌شوند. در فصل ۳ به مسئله‌ی قابلیت دید یک پاره‌خط درون چندضلعی ساده می‌پردازیم و الگوریتم اول به دست آمده را ارائه می‌کنیم. در فصل ۴ دوباره همین مسئله را بررسی می‌کنیم و الگوریتم دوم که با افرایش زمان پرس‌وجو، به کاهش زمان پیش‌پردازش دست یافته‌ایم، بیان می‌کنیم. در ادامه در فصل ۵ مسئله‌ی قابلیت دید پاره‌خط را در چندضلعی‌های حفره‌دار مطرح کرده و به بیان الگوریتمی حساس به خروجی برای آن می‌پردازیم. در فصل ۶ مسئله‌ی به دست آوردن اندازه چندضلعی قابل دید پاره‌خط درون چندضلعی‌های ساده را مطرح می‌کنیم. در این فصل چند الگوریتم دقیق و یک الگوریتم تقریبی برای این مسئله بیان می‌کنیم. در فصل ۷ به مسئله‌ی قابلیت دید پاره‌خط متحرک در بین تعدادی شیء می‌پردازیم. نشان می‌دهیم که چگونه با حرکت پاره‌خط، قابلیت دید پاره‌خط را به شکلی بهینه نگهداری کنیم. در نهایت، در فصل ۸ به کارهای آتی که در ادامه به آن‌ها خواهیم پرداخت اشاره می‌کنیم.

فصلِ ۲

مفاهیم و تعاریف

یک چندضلعی $\mathcal{P}$ در یک صفحه که با رئوس v_1، v_2، ... v_n مشخص شده است، شامل تمام نقاط صفحه است که توسط مرز چندضلعی محدود شده‌اند. مرز چندضلعی $\mathcal{P}$ شامل زنجیره پاره‌خط‌های v_iv_{i+1} و نیز v_nv_1 است. چندضلعی $\mathcal{P}$ ساده است هرگاه پاره‌خط‌های مرز چندضلعی فقط در نقاط انتهایی هم‌دیگر را قطع کنند و هیچ دو نقطه‌ای از مجموعه نقاط v_i هم‌پوشانی نداشته باشند. نقاط v_i رئوس چندضلعی $\mathcal{P}$ را تشکیل می‌دهند. چندضلعی $\mathcal{P}$ حفره‌دار نامیده می‌شود هرگاه مرز بیرونی آن یک چندضلعی ساده باشد و علاوه بر آن تعدادی چندضلعی ساده که با هم و با مرز بیرونی تقاطع ندارند از داخل چندضلعی بیرونی کنده شده باشند. هر کدام از این چندضلعی‌ها یک حفره در چندضلعی بیرونی ایجاد می‌کند. در شکل ۲-۱ یک چندضلعی ساده و یک چندضلعی حفره‌دار نشان داده شده است.

هر چندضلعی ساده $\mathcal{P}$ با دنباله رئوس آن و هر چندضلعی حفره‌دار با چندضلعی ساده بیرونی و چندضلعی‌های ساده داخلی آن مشخص می‌شود. تعداد رئوس هر چندضلعی برابر با تعداد رئوس مرز بیرونی و نیز حفره‌های داخلی آن است. پاره‌خط‌های متصل کننده رئوس مجاور نیز اضلاع چندضلعی را تشکیل می‌دهند.



شکلِ ۲-۱: A) چندضلعی ساده، B) چندضلعی حفره‌دار.

شکلِ ۲–۲: A) قابلیت دید در چندضلعی ساده، B) قابلیت دید در چندضلعی حفره‌دار.

۲–۱ قابلیت دید

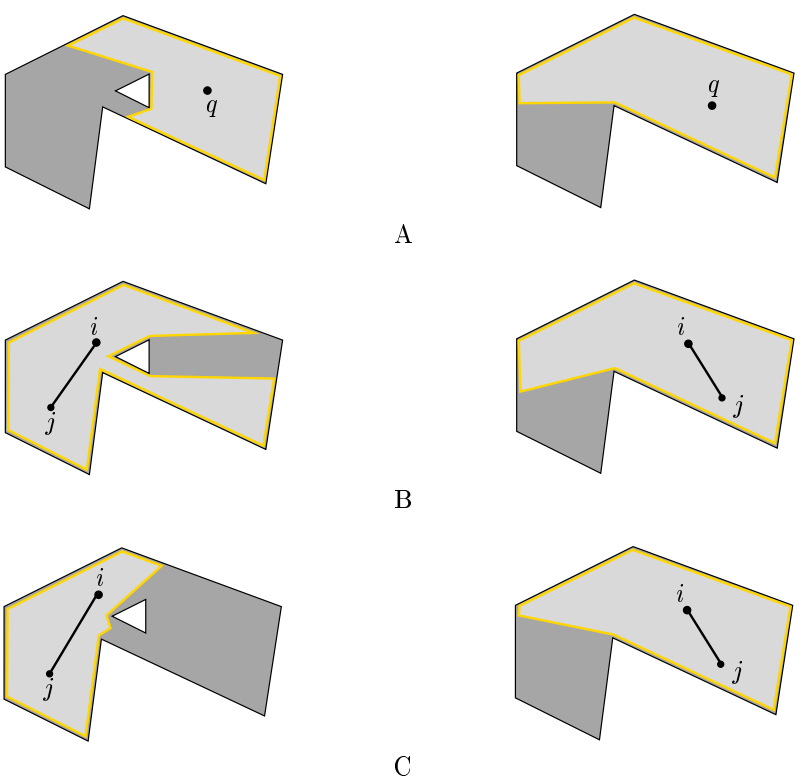
دو نقطه در داخل یک چندضلعی هم‌دیگر را می‌بینند یا از یک‌دیگر قابل دید هستند، هرگاه پاره‌خط متصل کننده آن‌ها به‌طور کامل درون چندضلعی قرار بگیرد و از بیرون چندضلعی یا از درون حفره‌های آن عبور نکند. در شکل ۲–۲ نقاط i و j از هم‌دیگر قابل دید هستند در حالیکه نقاط i و k و نیز j و k از هم‌دیگر قابل دید نمی‌باشند.

مجموعه‌ی تمام نقاطی که از نقطه‌ی q واقع در درون چندضلعی $\mathcal{P}$ قابل دید هستند، چندضلعی قابل دید q نامیده می‌شود و با $V(q)$ نشان داده می‌شود. چندضلعی قابل دید از یک نقطه برای چندضلعی‌های ساده و حفره‌دار در قسمت A از شکل ۲–۳ نشان داده شده است.

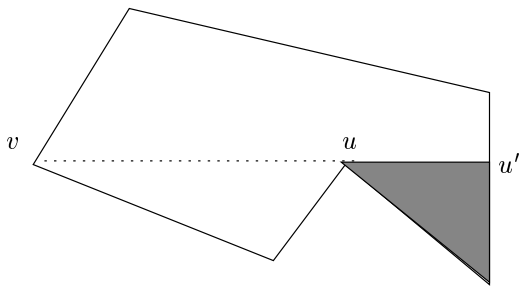
برای یک پاره‌خط که به‌طور کامل درون چندضلعی $\mathcal{P}$ قرار دارد دو نوع چندضلعی قابل دید تعریف می‌شود: چندضلعی قابل دید ضعیف برای پاره خط ij از $\mathcal{P}$ شامل تمام نقاطی از $\mathcal{P}$ است که از حداقل یک نقطه‌ی این پاره‌خط قابل دید باشند. این نوع از قابلیت دید در قسمت B از شکل ۲–۳ نشان داده شده است. نوع دیگر قابلیت دید برای پاره خط ij که قابلیت دید قوی نامیده می‌شود شامل تمام نقاطی از $\mathcal{P}$ است که از هر کدام از نقاط این پاره‌خط قابل دید باشند. در قسمت C از شکل ۲–۳ این نوع از قابلیت دید نشان داده شده است. قابلیت دید برای یک چندضلعی داخل $\mathcal{P}$ نیز به‌صورت مشابه قابل تعریف است.

اکثر الگوریتم‌های مربوط به محاسبه چندضلعی قابل دید برای پاره‌خط یا چندضلعی‌های درون $\mathcal{P}$ مبتنی بر محاسبه چندضلعی قابل دید نقطه‌ای می‌باشند. به عنوان مثال قابلیت دید قوی در چندضلعی‌های ساده از اشتراک چندضلعی‌های قابل دید نقاط دو سر پاره‌خط بدست می‌آید. برای محاسبه قابلیت دید ضعیف نیز با حرکت کردن از یک انتهای پاره‌خط به انتهای دیگر و توسعه‌ی چندضلعی قابل دید نقطه‌ای در طول مسیر بدست می‌آید.

بدیهی است که با داشتن رئوس $V(q)$ می‌توان آن‌را به‌طور دقیق مشخص کرد. علاوه‌بر آن، داشتن دنباله رئوس یا یال‌هایی از $\mathcal{P}$ که توسط q دیده می‌شوند، برای مشخص کردن دقیق $V(q)$ کافی است. برای این‌کار دنباله رئوس $\mathcal{P}$ در تناظر با دنباله مزبور پیمایش می‌شوند و $V(q)$ که ممکن است در برخی موارد فقط شامل بخشی از یک ضلع از $\mathcal{P}$ باشد به‌طور دقیق مشخص می‌گردد. به همین جهت در هنگام محاسبه $V(q)$ یا نمایش آن اصراری بر دقیق بودن $V(q)$ نداریم و فقط تعیین دنباله رئوس یا یال‌هایی که به‌طور کامل یا بخش‌هایی از آن‌ها از q قابل دید هستند مورد نظر است.



شکلِ ۲–۳: چندضلعی قابل دید در چندضلعی‌های ساده و حفره‌دار: A) چندضلعی قابل دید از نقطه q، B) چندضلعی قابل دید ضعیف از پاره‌خط ij، C) چندضلعی قابل دید قوی از پاره خط ij.



شکلِ ۲–۴: uu' یک پاره‌خط بحرانی از چندضلعی است.

۲–۲ تجزیه قابلیت دید

در چندضلعی ساده $\mathcal{P}$ ناحیه‌ی قابلیت دید R شامل یک مجموعه بیشینه از نقاط به هم پیوسته از $\mathcal{P}$ است که هر دو نقطه از این ناحیه دنباله یکسانی از رئوس $\mathcal{P}$ را می‌بینند. به عبارت دیگر چندضلعی قابل دید تمام نقاط یک ناحیه قابلیت دید از رئوس یکسانی از $\mathcal{P}$ تشکیل شده است.

چنان‌چه تمام ناحیه‌های قابلیت دید $\mathcal{P}$ شناسایی و مشخص گردند و برای هر ناحیه، دنباله رئوس قابل دید از آن تعیین گردد، در زمان پرس‌وجو کافیست تا ناحیه قابلیت دید که نقطه پرس‌وجوی q درون آن قرار دارد مشخص شود. در این صورت دنباله رئوس مربوط به $V(q)$ که برابر با مجموعه رئوس متناظر با آن ناحیه قابلیت دید است مهیا بوده و از روی آن‌ها $V(q)$ ساخته می‌شود.

تجزیه قابلیت دید روشی برای شناسایی ناحیه‌های قابلیت دید است. برای این‌که ناحیه‌های قابلیت دید مشخص شوند باید مرزهای این ناحیه‌ها و در واقع شرایطی که باعث تفاوت دید در دو نقطه می‌شود مشخص شود. چنانچه در شکل ۲–۴ نشان داده شده است، هر جا که یک رأس انعکاسی[1] وجود دارد باعث تفاوت دید در برخی از نقاط چندضلعی می‌شود. در این شکل نقاطی که در زیر پاره‌خط uu' قرار دارند راس v را نمی‌بینند، در حالی‌که نقاط بالای این پاره‌خط می‌توانند رأس v را ببینند.

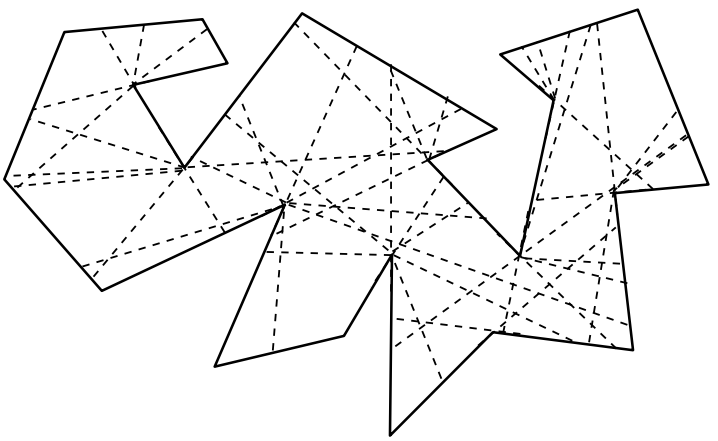
بنابراین پاره خط uu' جزء پاره‌خط‌های تعیین کننده مرز ناحیه‌های قابلیت دید است. این نوع پاره‌خط‌ها، یال محدودیت بحرانی[2]، و یا به طور خلاصه پاره‌خط بحرانی و یا خط بحرانی نامیده می‌شوند. هر دو رأس u و v که هم‌دیگر را می‌بینند و u یک رأس انعکاسی است، یک پاره‌خط بحرانی در چندضلعی به‌وجود می‌آورند که از امتداد دادن پاره‌خط vu از طرف u تا رسیدن به مرز چندضلعی به‌دست می‌آید.

به‌راحتی قابل بررسی است که اضلاع چندضلعی همراه با پاره‌خط‌های بحرانی آن مرزهای ناحیه‌های قابلیت دید را تشکیل می‌دهند. بنابراین برای تعیین ناحیه‌های قابلیت دید، تمام پاره‌خط‌های بحرانی $\mathcal{P}$ مشخص می‌شود. زیرتقسیم حاصل از این پاره‌خط‌ها و اضلاع چندضلعی، ناحیه‌های قابلیت دید را مشخص می‌کند. به این روش تجزیه قابلیت دید گفته می‌شود. در شکل ۲–۵ تجزیه قابلیت دید یک چندضلعی ساده نشان داده شده است.

---

[1] Reflex Vertex

[2] Critical Constraint Edge



شکلِ ۲–۵: تجزیه قابلیت دید چند ضلعی ساده.

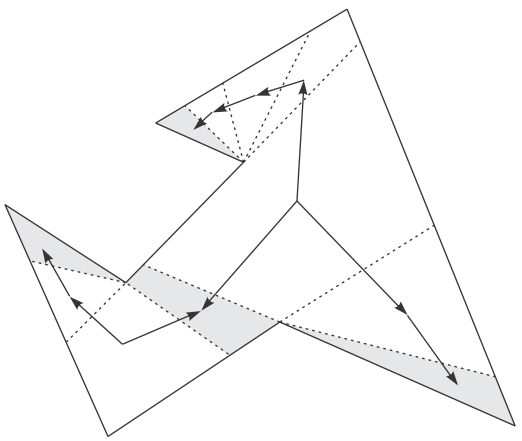
در صورت داشتن ناحیه‌های قابلیت دید و دنباله رئوس قابل دید از نقاط هر ناحیه، می‌توان $V(q)$ را به ازای هر نقطه پرس‌وجوی q محاسبه کرد. برای این کار ابتدا به عنوان پیش‌پردازش یک داده ساختار مکان‌یابی بر روی ناحیه‌های قابلیت دید ایجاد می‌کنیم. سپس در زمان پرس‌وجو با استفاده از داده‌ساختار مکان‌یابی، ناحیه در برگیرنده نقطه پرس‌وجو پیدا می‌شود. دنباله رئوس قابل دید از نقطه پرس‌وجو برابر با رئوس قابل دید مربوط به این ناحیه قابلیت دید است. با یک بار پیمایش این دنباله رئوس و اضلاع $V(q)$ بطور دقیق مشخص می‌شود.

با توجه به اینکه تعداد پاره‌خط‌های بحرانی $O(n^2)$ است، چنانچه این پاره‌خط‌ها در $O(n^4)$ نقطه تلاقی داشته باشند تعداد ناحیه‌های دیداری $O(n^4)$ خواهد شد که نتیجه‌ای بدیهی از فرمول اویلر برای گراف‌های مسطح است. در [۲۸] نشان داده شده است که در یک چندضلعی ساده تعداد ناحیه‌های قابلیت دید $O(n^3)$ است.

دو ناحیه قابلیت دید مجاور که یک پاره‌خط بحرانی بین آنها قرار گرفته است رئوس یکسانی از $\mathcal{P}$ را می‌بینند، بجز یک راس که مبدا پاره‌خط بحرانی بین آنها است. نقاط یکی از ناحیه‌های قابلیت دید این راس را می‌بینند ولی نقاط ناحیه قابلیت دید دیگر این راس را نمی‌بینند.

بنابراین نیازی به نگهداری رئوس قابل دید از تمام ناحیه‌های قابلیت دید نیست و می‌توان از اطلاعات پاره‌خط‌های بحرانی و نیز رئوس قابل دید برخی از ناحیه‌ها، رئوس قابل دید سایر ناحیه‌ها را تعیین کرد. برای این منظور گراف دوگان جهت‌دار مربوط به تجزیه قابلیت دید ساخته می‌شود. در این گراف هر راس متناظر با یک ناحیه قابلیت دید است و بین رئوس متناظر با دو ناحیه مجاور یک یال جهت‌دار وجود دارد. جهت یال به طرف راس متناظر با ناحیه‌ای است که تعداد رئوس قابل دید آن کمتر است. برچسب این یال نیز همان راسی است که در یکی از این دو ناحیه قابلیت دید دیده نمی‌شود ولی در ناحیه قابلیت دید دیگر دیده می‌شود و در واقع راس مبدا پاره‌خط بحرانی بین دو ناحیه است.

در این گراف، به علت منطق قابلیت دید در چندضلعی‌های ساده، دور وجود ندارد. همچنین رئوسی که درجه خروجی آنها صفر است متناظر با ناحیه‌هایی هستند که تعداد رئوس قابل دید آنها از تمام



شکلِ ۲–۶: گراف دوگان جهت‌دار متناظر با تجزیه قابلیت دید.

ناحیه‌های مجاور آنها کمتر است. به این ناحیه‌ها چاهک گفته می‌شود. در شکل ۲–۶ گراف دوگان جهت‌دار متناظر با تجزیه قابلیت دید یک چندضلعی ساده نشان داده شده است.

اگر فقط رئوس قابل دید از چاهک‌ها نگهداری شوند برای یک ناحیه غیر چاهک کافیست از راس متناظر با آن ناحیه تا رسیدن به راس متناظر با یک چاهک مسیری در گراف دوگان پیموده شود و برچسب‌های میسر به رئوس قابل دید چاهک افزوده گردند. به این ترتیب نیازی به نگهداری رئوس قابل دید ناحیه‌های غیر چاهک نیست و می‌توان آنها را از روی گراف دوگان جهت دار و رئوس قابل دید از چاهک‌ها ساخت. در [۲۸] اثبات شده است که تعداد چاهک‌ها در یک چندضلعی ساده $O(n^2)$ است.

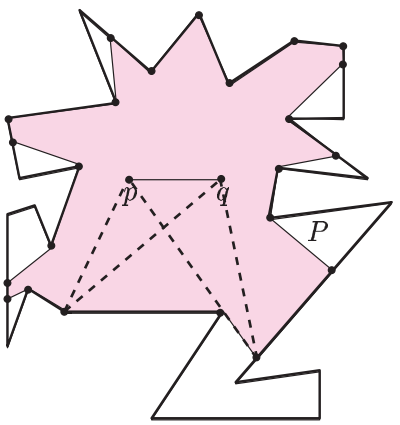
۲–۳ قابلیت دید پاره‌خط

با توجه به این که موضوع اصلی این پایان‌نامه ارائه‌ی الگوریتم‌های جدید برای محاسبه‌ی قابلیت دید پاره‌خط در چندضلعی‌ها می‌باشد، در این بخش به بیان تعاریف و کارهای گذشته در این زمینه می‌پردازیم.

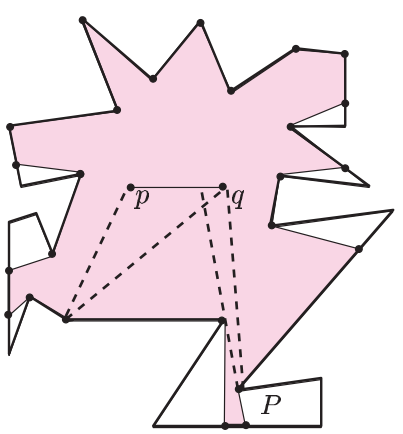
قابلیت دید یک پاره‌خط نسبت به یک نقطه به صورت قابلیت دید قوی و قابلیت دید ضعیف تعریف می‌گردد. نقطه‌ی a نسبت به پاره‌خط pq قابل دید قوی است اگر a همه‌ی نقاط pq را ببیند (شکل ۲–۷)، و قابل دید ضعیف است اگر حداقل یک نقطه از پاره‌خط pq را ببیند (شکل ۲–۸).

در یک چندضلعی ساده، هر گاه یک نقطه از دو سر یک پاره‌خط واقع در درون آن چندضلعی قابل دید باشد، آنگاه از تمام نقاط آن پاره‌خط قابل دید است. برعکس این مطلب نیز بطور بدیهی صادق است. بنابراین چندضلعی قابل دید قوی برای یک پاره‌خط واقع در درون یک چندضلعی ساده از اشتراک چندضلعی‌های قابل دید دو سر آن پاره‌خط به‌دست می‌آید.

چندضلعی قابل دید ضعیف یک پاره‌خط درون یک چندضلعی، به صورت مجموعه‌ی همه‌ی نقاطی که حداقل از یک نقطه‌ی پاره‌خط قابل دید باشند تعریف می‌شود. بر خلاف چندضلعی قابل دید قوی، محاسبه‌ی چندضلعی قابل دید ضعیف، چه در چندضلعی‌های ساده و چه در چندضلعی‌های حفره‌دار، به



شکلِ ۲–۷: قابلیت دید قوی.



شکلِ ۲–۸: قابلیت دید ضعیف.

راحتی قابل انجام نیست. فرض کنید $\mathcal{P}$ یک چندضلعی ساده و pq یک پاره‌خط در $\mathcal{P}$ باشد. اولین الگوریتم غیربدیهی برای این مسئله توسط ElGindy [۱۸]، Lee و Lin [۳۲]، و Chazelle و Guibas [۹] با زمان اجرای $O(n \log n)$ ارائه شد. Guibas و همکاران [۲۶] نشان دادند که مسئله در صورتی که مثلث‌بندی $\mathcal{P}$ از قبل موجود باشد، در زمان $O(n)$ قابل حل است. چون چندضلعی ساده $\mathcal{P}$ را می‌توان در زمان خطی مثلث‌بندی کرد [۱۰]، الگوریتم Guibas و همکارنش در زمان خطی قابل استفاده است.

این مسئله در حالت پرس‌وجو هم مورد توجه قرار گرفته است. در [۶] نشان داده شده است که یک چندضلعی ساده $\mathcal{P}$ با n راس را می‌توان در زمان $O(n^3 \log n)$ پیش‌پردازش کرد و داده ساختاری به اندازه‌ی $O(n^3)$ ایجاد کرد به نحوی که با دریافت یک پاره‌خط در زمان پرس‌وجو، چندضلعی قابل دید ضعیف پاره‌خط را در زمان $O(k \log n)$ به دست آورد. این نتیجه در [۲] بهبود داده شد و نشان داده شد که با پیش‌پردازش چندضلعی در زمان $O(n^2 \log n)$ و ایجاد داده ساختاری به اندازه‌ی $O(n^2)$، چندضلعی قابل دید ضعیف را می‌توان در زمان $O(k \log^2 n)$ به دست آورد.

حل مساله چندضلعی قابل دید ضعیف در داخل چندضلعی حفره‌دار به مراتب پیچیده‌تر است. Suri و O'Rourke [۴۲] ثابت کرده‌اند که در بدترین حالت، اندازه‌ی چندضلعی قابل دید ضعیف یک پاره‌خط در داخل یک چندضلعی حفره‌دار $\Omega(n^4)$ است، و الگوریتمی را ارائه کرده‌اند که می‌تواند در زمان $O(n^4)$ این چندضلعی را محاسبه کند. آن‌ها همچنین نشان داده‌اند که این چندضلعی، از اجتماع حداکثر $O(n^2)$ مثلث تشکیل شده است و می‌توان در زمان $O(n^2)$ این مثلث‌ها را به دست آورد. برای تعیین دقیق رئوس چندضلعی و ترتیب آن‌ها، به زمان $O(n^4)$ نیاز خواهیم داشت.

۲-۴ جستجوی دامنه‌ای

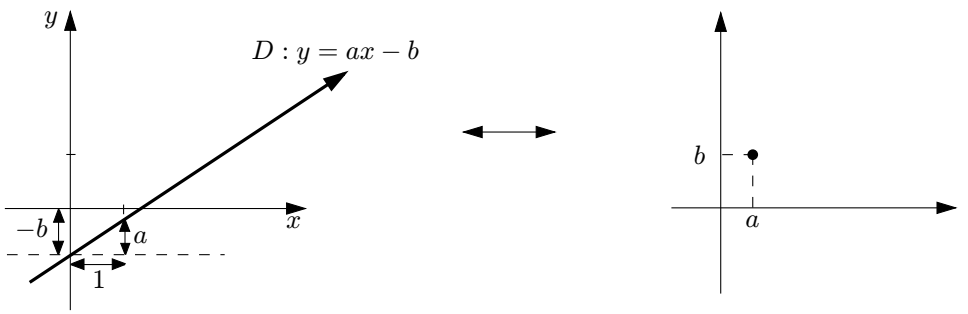
فرض کنید که P مجموعه‌ای از n نقطه در فضای d-بعدی باشد. در مسئله‌های جستجوی دامنه، می‌توان نقاط قرار گرفته درون یک ناحیه‌ی R در فضا را به دست آورد و یا تعداد آن‌ها را شمارش کرد. در فصل ۶ از این داده‌ساختار، هنگامی که P مجموعه نقاط درون صفحه و R یک نیم‌صفحه است استفاده می‌کنیم.

Chazelle و دیگران [۱۱] داده‌ساختاری را معرفی کردند که با استفاده از پیش‌پردازش در زمان $O(n^{1-1/d+\epsilon})$ و حافظه‌ی $O(n)$، می‌توان پرس‌وجوها را در زمان $O(n^{1-1/d+\epsilon})$ پاسخ داد. در اینجا ϵ هر مقدار کوچک مثبت می‌تواند باشد. آن‌ها همچنین توازنی بین هزینه‌ی پیش‌پردازش و زمان پرس‌وجو به دست آوردند و نشان دادند که می‌توان با استفاده از حافظه‌ی (Om) و زمان پیش‌پردازش $O(m^{1+\epsilon})$، که $n \leq m \leq n^d$، به پرس‌وجوهای داده شده در زمان $O(\frac{n^{1+\epsilon}}{m^{1/d}})$ پاسخ گفت.

در یک کار دیگر [۱۲]، Chazelle نشان داد که با استفاده از برش‌ها[3]، می‌توان داده‌ساختاری به اندازه‌ی $O(n^{d+\epsilon})$ ایجاد کرد که به پرس‌وجوها در زمان $O(\log n)$ پاسخ دهد. این داده‌ساختار را می‌توان در زمان $O(n^{d+\epsilon})$ به دست آورد.

نتیجه‌ی جستجوی دامنه‌ای به شکل اجتماع تعداد مجموعه‌ی کانونی به دست می‌آید. با بررسی دوباره این مجموعه‌ها می‌توان جستجوی دامنه‌ای را دوباره روی آن‌ها اجرا کرد [۱۶] و به عبارت دیگر

[3] Cutting



شکلِ ۲–۹: دوگان پرتوی.

جستجوی چندمرحله‌ای بر روی مجموعه نقاط اولیه انجام داد. استفاده از داده‌ساختارهای جستجو برای انجام جستجوی چندمرحله‌ای به هزینه‌ی پیش‌پردازش اضافی احتیاج ندارد و تنها زمان پاسخ‌گویی به پرس‌وجو با یک عامل لگاریتمی اضافه می‌گردد.

۲–۵ دوگانگی نقطه-خط

معمولاً نمایش نقطه در صفحه آسان‌تر از نمایش خط است. از طرف دیگر، گاهی بیان یا پیاده‌سازی یک الگوریتم که بر روی مجموعه‌ای از نقاط عمل می‌کند، ساده‌تر عمل کردن بر روی مجموعه‌ای از خطوط می‌باشد. یکی از ابزارهای مورد استفاده برای مواجهه‌ی بهتر با مسائل، انتقال مسئله به فضای دوگان[۴] می‌باشد. در ادامه یکی از دوگان‌های مطرح برای تبدیل خط به نقطه (و برعکس) را معرفی می‌کنیم. صفحه‌ای که داده اصلی در آن قرار دارد، صفحه‌ی اولیه[۵]، و صفحه‌ای که شامل تصویر داده اصلی است، صفحه‌ی دوگان[۶] نام دارد. در یکی از پرکاربردترین تبدیل‌ها، دوگان نقطه $P(a,b)$ در صفحه‌ی اولیه، خط $L : x/a + y/b = ۱$ در صفحه‌ی دوگان است و برعکس (شکل ۲–۹).

ما از مفهموم دوگانگی نقطه-خط در فصل ۷ و در هنگام پردازش رخدادهای قابلیت دید استفاده خواهیم کرد.

[4] Dual Space

[5] Primal

[6] Dual

فصلِ ۳

قابلیت دید پاره‌خط در چندضلعی ساده

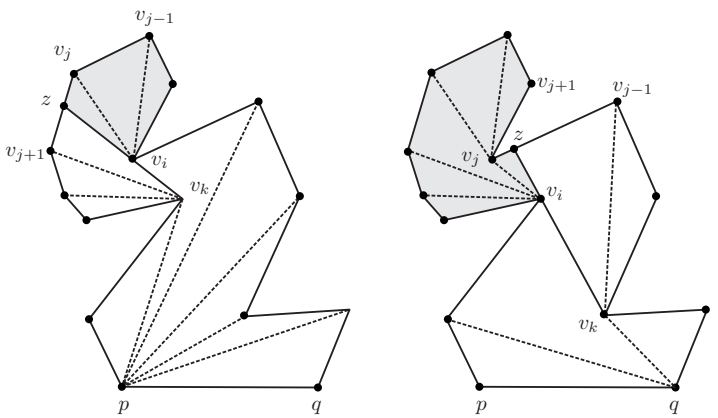
همان‌طور که قبلا گفته شد، الگوریتم خطی $O(n)$ برای محاسبه‌ی چندضلعی قابل دید ضعیف $\mathrm{WVP}(pq)$ ارائه شده است [۲۶]. هرچند این الگوریتم بهینه است، اما زمان اجرای آن وابسته به اندازه‌ی خروجی نیست و به عبارت دیگر این الگوریتم حساس به خروجی نیست. در این فصل نشان می‌دهیم که چگونه این مسئله را به صورت حساس به خروجی حل نماییم. برای یک چندضلعی ساده $\mathcal{P}$ با n راس، نشان می‌دهیم که می‌توان با پیش‌پردازش این چندضلعی در زمان $O(n^۳ \log n)$ و ایجاد داده‌ساختاری به اندازه‌ی $O(n^۳)$، چندضلعی قابل دید ضعیف یک پاره‌خط داده شده را در زمان $O(\log n + |\mathrm{WVP}(pq)|)$ به دست آورد، که در آن $|\mathrm{WVP}(pq)|$ اندازه‌ی خروجی می‌باشد.

در ابتدا الگوریتم خطی محاسبه‌ی چندضلعی قابل دید ضعیف پاره‌خط را بیان می‌کنیم و سپس نشان می‌دهیم که چگونه با تغییر این الگوریتم، جواب را به صورت حساس به خروجی به دست آوریم.

۳-۱ الگوریتم خطی برای محاسبه WVP

در این بخش الگوریتم خطی Guibas و دیگران [۲۶] را برای محاسبه $\mathrm{WVP}(pq)$ یک پاره‌خط pq درون چندضلعی ساده $\mathcal{P}$ با n رأس، آن‌گونه که در [۲۴] شرح داده شده است، بیان می‌کنیم. در بخش‌های بعد نشان خواهیم داد که چگونه از این الگوریتم برای محاسبه‌ی چندضلعی قابل دید ضعیف به شکل حساس به خروجی استفاده می‌کنیم. برای سادگی، فرض می‌کنیم که pq یک ضلع از $\mathcal{P}$ می‌باشد. در بخش‌های بعد نشان خواهیم داد که pq می‌تواند هر پاره خط درون چندضلعی باشد.

فرض کنید که $\mathrm{SPT}(p)$ درخت کوتاه‌ترین مسیر با ریشه‌ی p در $\mathcal{P}$ باشد. در مرحله‌ی اول، $\mathrm{SPT}(p)$ را با الگوریتم عمق اول پیمایش کرده و در هر رأس v_i در $\mathrm{SPT}(p)$، نوع گردش آن را بررسی می‌کنیم (شکل ۳-۱ را ببینید). چنانچه مسیر $\mathrm{SP}(p, v_j)$ در رأس v_i یک گردش به راست انجام دهد، در میان فرزندان v_i در درخت، رأس با بزرگ‌ترین اندیس j را پیدا می‌کنیم. سپس نقطه تقاطع $v_jv_{j+۱}$ و v_kv_i به نام z را پیدا می‌کنیم، که v_k پدر v_i در $SPT(p)$ است. با توجه به این که راسی بین v_j و $v_{j+۱}$ وجود ندارد، این کار را می‌توان در $O(۱)$ انجام داد. در نهایت مرز پادساعت‌گرد $\mathcal{P}$ از v_i تا z را، با اضافه کردن



شکلِ ۳-۱: دو مرحله‌ی الگوریتم محاسبه‌ی چندضلعی قابل دید ضعیف. در شکل سمت چپ، کوتاه‌ترین مسیر از p به v_j یک اولین گردش به راست را در v_i انجام می‌دهد. در شکل سمت راست، کوتاه‌ترین مسیر از q به v'_j اولین گردش به چپ را در v'_i انجام می‌دهد.

پاره‌خط v_iz حذف می‌کنیم.

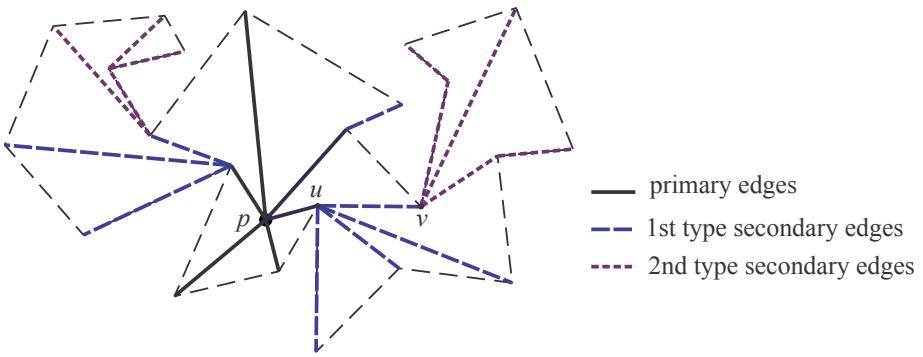
بخش باقی‌مانده از $\mathcal{P}$ را با $\mathcal{P}'$ نشان می‌دهیم. در مرحله‌ی دوم، کارهای انجام شده را برای q، این بار در $\mathcal{P}'$، تکرار می‌کنیم، با این تفاوت که با رسیدن به هر راس $\mathrm{SPT}(q)$، بررسی می‌کنیم که آیا این رأس اولین گردش به چپ را انجام داده است یا خیر. در صورت وجود گردش به چپ، چندضلعی را در آن راس به شیوه قبل برش می‌دهیم. بعد از اتمام این مرحله، بخش باقی‌مانده از $\mathcal{P}'$ برابر با $\mathrm{WVP}(pq)$ خواهد بود.

۳-۲ الگوریتم پرس‌وجوی محاسبه‌ی چندضلعی قابل دید ضعیف

در این بخش نشان می‌دهیم که چگونه الگوریتم خطی ارائه شده در بخش ۳-۱ را تغییر دهیم به نحوی که بتوانیم WVP را به صورت حساس به خروجی محاسبه کنیم. یکی از بخش‌های مهم این کار، محاسبه‌ی درخت کوتاه‌ترین مسیر است. بنابراین در ابتدا نشان می‌دهیم که چگونه می‌توان این درخت را به شکل حساس به خروجی محاسبه کرد. در ادامه در بخش ۳-۲-۲ صورت ابتدایی الگوریتم خود را بیان می‌کنیم، و در نهایت در بخش ۳-۲-۳ الگوریتم ارائه شده را بهبود داده و نتیجه‌ی نهایی را به دست می‌دهیم.

۳-۲-۱ محاسبه‌ی درخت کوتاه‌ترین مسیر به شکل حساس به خروجی

درخت کوتاه‌ترین مسیر یک نقطه درون یک چندضلعی ساده‌ی $\mathcal{P}$ با n راس را می‌توان در زمان $O(n)$ محاسبه کرد [۲۷]. استفاده از این الگوریتم، ما را مجبور به صرف زمان $O(n)$ در پرس‌وجو می‌کند، که این مخالف حساس به خروجی بودن نتیجه است. برای رفع این مشکل، در این بخش نشان می‌دهیم که چگونه



شکلِ ۳-۲: درخت کوتاه‌ترین مسیر نقطه‌ی p و انواع یال‌های آن: یال‌هایی که مستقیما به راس p متصل هستند (یال‌های اولیه)، یال‌هایی که به یک یال اولیه متصل هستند (یال‌های ثانویه نوع اول)، و یال‌های باقیمانده (یال‌های ثانویه نوع دوم).

یک چندضلعی ساده را پیش‌پردازش کنیم به طوری که در زمان پرس‌وجو، با دریافت یک نقطه‌ی p درون چندضلعی، هر بخش از درخت $\mathrm{SPT}(p)$ را به شکل حساس به خروجی محاسبه کنیم.

یک درخت کوتاه‌ترین مسیر SPT از دو نوع یال تشکیل شده است: یال‌های اولیه و یال‌های ثانویه. یال‌های اولیه، یال‌هایی هستند که از ریشه‌ی درخت شروع می‌شوند و ریشه را به راس‌های قابل دید آن متصل می‌کنند. یال‌های ثانویه دیگر یال‌های درخت هستند که دو راس چندضلعی را به هم متصل می‌کنند (شکل ۳-۲ را ببینید). علاوه بر این، دو نوع یال ثانویه قابل تفکیک از هم هستند: یال ثانویه‌ی نوع اول (به طور خلاصه، یال نوع اول) یک یال ثانویه است که به یک یال اولیه متصل شده است، و یال ثانویه‌ی نوع دوم (به اختصار، یال نوع دوم) یک یال ثانویه است که به یال اولیه‌ای متصل نیست. در ادامه نشان می‌دهیم که چگونه این یال‌ها را در زمان پیش‌پردازش محاسبه و ذخیره کنیم، به شکلی که بتوان آن‌ها را در زمان پرس‌وجو به صورت بهینه بازیابی کرد.

یال‌های اولیه $\mathrm{SPT}(p)$ را می‌توانیم با الگوریتم محاسبه‌ی قابلیت دید Bose و دیگران [۶] به دست آوریم. به بیان دقیق‌تر، با زمان پیش‌پردازش $O(n^3 \log n)$ و فضای $O(n^3)$، می‌توان لیست مرتب شده‌ی راس‌های قابل دید از یک نقطه‌ی پرس‌وجوی q را در زمان $O(\log n)$ محاسبه کرد.

برای محاسبه‌ی یال‌های ثانویه SPT، در زمان پیش‌پردازش همه‌ی مقادیر ممکن یال‌های ثانویه‌ی هر راس چندضلعی را محاسبه و ذخیره می‌کنیم. با داشتن این مقادیر، در زمان پرس‌وجو، لیست یال‌های مرتبط با راس مورد نظر را پیدا کرده و به عنوان یال‌های درخت کوتاه‌ترین مسیر در آن راس گزارش می‌کنیم.

هر راس v در $\mathcal{P}$ دارای $O(n)$ پدر ممکن در SPT است. به ازای هر کدام از این پدرها، v می‌تواند $O(n)$ یال نوع دوم در SPT داشته باشد. بنابراین، برای هر راس v، به حافظه‌ی $O(n^2)$ برای نگهداری همه‌ی یال‌های ممکن نوع دوم نیاز خواهیم داشت. محاسبه و ذخیره‌ی این یال‌های احتمالی را می‌توان در زمان $O(n^2 \log n)$ انجام داد. در زمان پرس‌وجو، وقتی به راس v می‌رسیم، از این داده‌های ذخیره شده استفاده می‌کنیم و لیست یال‌های نوع دوم درخت در این راس را بازیابی می‌کنیم. با توجه به این که درخت n راس دارد، محاسبه‌ی این داده‌ها برای همه‌ی راس‌های $\mathcal{P}$ به زمان و حافظه‌ی $O(n^3 \log n)$ و $O(n^3)$

احتیاج خواهد داشت.

حال به بررسی محاسبه‌ی یال‌های نوع اول می‌پردازیم. تعداد یال‌های نوع اول که از یک راس خارج می‌شوند $O(n)$ بوده و وابسته به پدر آن راس در درخت می‌باشد. می‌دانیم که پدر یک یال نوع اول در SPT، ریشه‌ی درخت می‌باشد. از آنجا که ریشه‌ی درخت می‌تواند در هر یک از $O(n^۳)$ ناحیه‌ی قابلیت دید قرار بگیرد (بخش ۲-۲ را ببینید)، برای محاسبه‌ی همه‌ی حالت‌های ممکن یال‌های نوع اول درخت، باید همه‌ی این ناحیه‌ها را به عنوان ناحیه‌ی ریشه‌ی درخت در نظر بگیریم. با توجه به این امر، یال‌های نوع اول را می‌توان در زمان $O(n^۴ \log n)$ و حافظه‌ی $O(n^۴)$ محاسبه و نگهداری کرد.

لم ۱ با انجام یک پیش‌پردازش بر روی چندضلعی ساده‌ی $\mathcal{P}$ با n رأس در زمان $O(n^۴ \log n)$ و ایجاد ساختاری به اندازه‌ی $O(n^۴)$، می‌توانیم درخت کوتاه‌ترین مسیر $\mathrm{SPT}(p)$ هر نقطه‌ی پرس‌وجوی p را در زمان $O(\log n + k)$ گزارش کنیم، که k اندازه‌ی بخشی از درخت است که گزارش می‌شود.

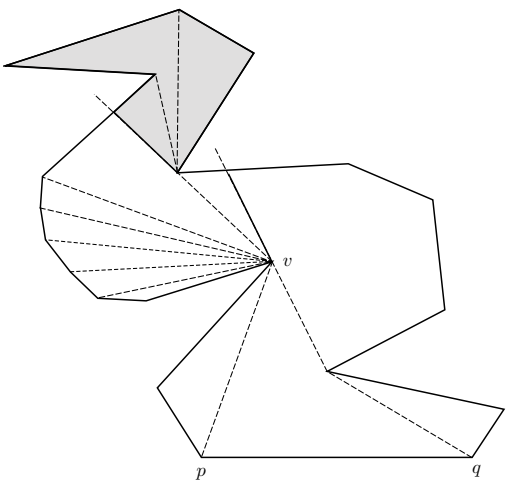
در بخش ۳-۲-۳ نشان می‌دهیم که چگونه می‌توان مقدار زمان پیش‌پردازش و حافظه‌ی این الگوریتم را به میزان خطی کاهش داد.

۳-۲-۲ محاسبه‌ی چندضلعی قابل دید ضعیف به شکل حساس به خروجی

الگوریتم خطی ارائه شده در بخش ۳-۱ برای محاسبه‌ی چندضلعی قابل دید ضعیف در چندضلعی‌های ساده، حساس به خروجی نیست. به مثال شکل ۳-۳ توجه کنید. در این مثال، همان‌طور که در بخش ۳-۱ بیان کرده‌ایم، ابتدا در $\mathrm{SPT}(p)$ با پیمایش عمق اول حرکت می‌کنیم و در هر راس، جهت گردش را بررسی می‌کنیم. راس v نشان داده شده در شکل را در نظر بگیرید. هنگامی که در کوتاه‌ترین مسیر $\mathrm{SP}(p, v)$ حرکت می‌کنیم، باید همه‌ی فرزندان v را بررسی کنیم. این کار به زمان $O(n)$ احتیاج دارد. در مرحله‌ی دوم و در زمان پیمایش $\mathrm{SPT}(q)$، همه‌ی فرزندان v حذف می‌گردند. بنابراین زمان مصرف شده در پردازش راس‌ها در $\mathrm{SPT}(p)$ اضافه بوده و حساس به خروجی بودن الگوریتم از بین می‌رود.

برای به دست آوردن یک الگوریتم حساس به خروجی، ابتدا داده‌ساختار بیان شده در بخش قبل برای درخت کوتاه‌ترین مسیر را می‌سازیم. بنابراین برای هر نقطه درون چند ضلعی، SPT را می‌توانیم در زمان پرس‌وجو محاسبه کنیم. برای هر راس چندضلعی، در زمان پیش‌پردازش یک سری اطلاعات اضافی محاسبه و ذخیره می‌کنیم. در یک چندضلعی ساده، یک راس v را بحرانی چپ (به اختصار LC) نسبت به نقطه‌ی p می‌خوانیم، چنانچه وقتی در $\mathrm{SPT}(p)$ و در مسیر $\mathrm{SP}(p, v)$ حرکت می‌کنیم، این مسیر اولین گردش به چپ خود را در v انجام دهد. به عبارت دیگر، هر کوتاه‌ترین مسیر از p به یک راس غیر LC، یک زنجیره‌ی محدب است که در همه‌ی راس‌ها تنها گردش به راست انجام می‌دهد. وضعیت بحرانی یک راس به این گفته می‌شود که آن راس LC هست یا نه. چنانچه وضعیت بحرانی همه‌ی راس‌های چندضلعی را نسبت به یک نقطه‌ی p بدانیم، می‌گوییم که اطلاعات بحرانی نقطه‌ی p را داریم. توجه داشته باشید که با توجه به این که در مرحله‌ی اول الگوریتم گردش به راست را در راس‌ها چک می‌کنیم، احتیاجی به ذخیره‌ی وضعیت گردش به راست در زمان پیش‌پردازش نداریم.

ایده‌ی اصلی این است که الگوریتم بخش ۳-۱ را تغییر دهیم و آن را حساس به خروجی کنیم. کلیات الگوریتم ما به این شکل است: در مرحله‌ی اول درخت $\mathrm{SPT}(p)$ را به صورت عمق اول پیمایش



شکلِ ۳-۳: در مرحله‌ی اول الگوریتم همه‌ی فرزندان v در $\mathrm{SPT}(p)$ پردازش می‌شوند. این راس‌ها که در $\mathrm{WVP}(pq)$ قرار ندارند می‌توانند هزینه‌ی اضافی $O(n)$ را ایجاد کنند.

می‌کنیم. در هر راس بررسی می‌کنیم که آیا این راس نسبت به q بحرانی چپ هست یا خیر. اگر چنین باشد، مطمئن خواهیم بود که فرزندان این راس نسبت به pq قابل دید ضعیف نخواهند بود. بنابراین پردازش این راس و زیردرخت آن را متوقف می‌کنیم تا در مرحله‌ی دوم دوباره به آن برسیم. در غیر این صورت، پردازش عادی راس را ادامه می‌دهیم و بررسی می‌کنیم که آیا یک گردش به راست در این راس صورت گرفته است یا نه (الگوریتم بخش ۳-۱ را ببینید). مرحله‌ی اول را تا بررسی همه‌ی راس‌های درخت ادامه می‌دهیم. در مرحله‌ی دوم، درخت $\mathrm{SPT}(q)$ را پیمایش کرده و الگوریتم را به شیوه‌ی قبل دنبال می‌کنیم.

لم ۲ همه‌ی راس‌های پیمایش شده در $\mathrm{SPT}(p)$ و $\mathrm{SPT}(q)$ راس‌های $\mathrm{WVP}(pq)$ هستند.

اثبات ۱ فرض کنید که در حین پیمایش $\mathrm{SPT}(p)$، راس v را ملاقات کنیم و $v \notin \mathrm{WVP}(pq)$. فرض کنید که u پدر v در مسیر $\mathrm{SP}(pv)$ باشد. بنابراین u یا یکی از پدران آن باید LC نسبت به q باشند، زیرا اگر چنین نبود الگوریتم v را به عنوان یکی از راس‌های $\mathrm{WVP}(pq)$ تشخیص می‌داد. بنابراین ممکن نیست که در حین پیمایش $\mathrm{SPT}(p)$ به راس v رسیده باشیم. استدلال مشابهی برای $\mathrm{SPT}(q)$ برقرار است. بنابراین لم ثابت می‌گردد.

در مرحله‌ی پیش‌پردازش، اطلاعات بحرانی یک راس از هر ناحیه‌ی قابلیت دید را محاسبه کرده و این اطلاعات را به آن ناحیه نسبت می‌دهیم. در زمان پرس‌وجو و با دریافت پاره‌خط pq، ناحیه‌های قابلیت دید p و q را پیدا می‌کنیم. با استفاده از اطلاعات بحرانی این دو ناحیه، الگوریتم را اجرا کرده و $\mathrm{WVP}(pq)$ را محاسبه می‌نماییم.

با توجه به این که تعداد ناحیه‌ها در تجزیه‌ی قابلیت دید $O(n^3)$ ناحیه است، برای نگهداری اطلاعات بحرانی ناحیه‌ها به فضای $O(n^4)$ احتیاج داریم. برای هر ناحیه، درخت کوتاه‌ترین مسیر SPT را برای یک نقطه دلخواه درون ناحیه محاسبه می‌کنیم و با پیمایش درخت، اطلاعات بحرانی هر راس را نسبت

به آن ناحیه به دست می‌آوریم. در هر ناحیه، یک آرایه به اندازه $O(n)$ را برای نگهداری این اطلاعات بحرانی راس‌ها در نظر می‌گیریم. همچنین داده‌ساختار توضیح داده شده در بخش ۳-۲-۱ برای محاسبه‌ی SPT را در زمان $O(n^4 \log n)$ و حافظه‌ی $O(n^4)$ محاسبه می‌کنیم. همچنین یک داده‌ساختار مکان‌یابی بر روی ناحیه‌های قابلیت دید می‌سازیم.

در زمان پرس‌وجو، ناحیه‌های قابلیت دید p و q را با داده‌ساختار مکان‌یابی در زمان $O(\log n)$ مشخص می‌کنیم. با توجه به این که زمان مورد نیاز برای پردازش هر راس برابر $O(1)$ است، و با در نظر گرفتن لم ۲، زمان پرس‌وجو برابر $O(\log n + |\mathrm{WVP}(pq)|)$ خواهد بود.

لم ۳ با پیش‌پردازش چندضلعی ساده $\mathcal{P}$ با n رأس در زمان $O(n^4 \log n)$ و ایجاد ساختاری به اندازه‌ی $O(n^4)$، می‌توانیم چندضلعی قابل دید ضعیف $\mathrm{WVP}(pq)$ را در زمان $O(\log n + |\mathrm{WVP}(pq)|)$ گزارش کنیم.

تا کنون فرض کرده‌ایم که pq یک راس چندضلعی است. نشان می‌دهیم که pq می‌تواند هر پاره‌خط درون چندضلعی $\mathcal{P}$ در نظر گرفته شود.

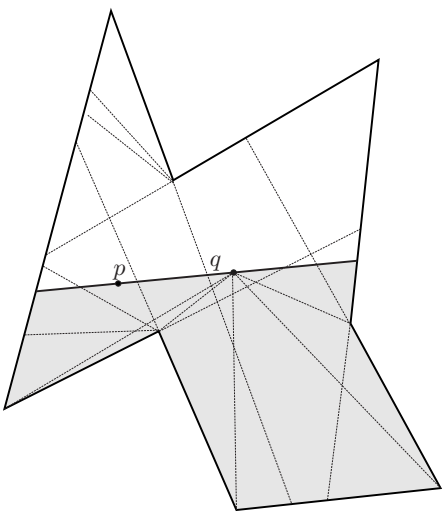
لم ۴ فرض کنید که pq یک پاره خط درون چند ضلعی ساده $\mathcal{P}$ باشد. می‌توانیم $\mathcal{P}$ را به دو چندضلعی $\mathcal{P}_1$ و $\mathcal{P}_2$ تجزیه کنیم به شکلی که در هر کدام از این دو چندضلعی، pq یک یال چندضلعی باشد. علاوه بر این، اطلاعات بحرانی $\mathcal{P}_1$ و $\mathcal{P}_2$ را می‌توان از اطلاعات بحرانی $\mathcal{P}$ به دست آورد.

اثبات ۲ نقاط برخورد خط حامل pq را با مرزهای $\mathcal{P}$ در نظر می‌گیریم. بر اساس این نقاط برخورد، $\mathcal{P}$ را به دو چندضلعی ساده‌ی $\mathcal{P}_1$ و $\mathcal{P}_2$ که در هر کدام pq یکی از یال‌ها است تقسیم می‌کنیم. ناحیه‌های قابلیت دید $\mathcal{P}_1$ و $\mathcal{P}_2$ زیرمجموعه‌هایی از ناحیه‌های قابلیت دید $\mathcal{P}$ هستند. بنابراین، اطلاعات بحرانی و همچنین یال‌های SPT مربوط به این ناحیه‌ها را خواهیم داشت. همچنین یال‌های اولیه‌ی p و q را می‌توانیم به دو دسته متعلق به $\mathcal{P}_1$ و متعلق به $\mathcal{P}_2$ تقسیم کنیم. مثال شکل ۳-۴ را ببینید.

۳-۲-۳ بهبود الگوریتم

در این بخش زمان و حافظه‌ی پیش‌پردازش به دست آمده در لم ۳ را بهبود می‌بخشیم. برای این منظور، قسمت‌هایی از الگوریتم ارائه شده در بخش ۳-۲-۲ که احتیاج به زمان $O(n^4 \log n)$ و حافظه‌ی $O(n^4)$ دارند را بازنگری می‌کنیم. این بخش‌ها عبارتند از اطلاعات بحرانی و یال‌های ثانویه نوع اول. نشان خواهیم داد که تنها کافی است که اطلاعات بحرانی و یال‌های نوع اول مربوط به ناحیه‌های قابلیت دید چاهک را محاسبه و نگهداری کنیم (برای تعریف ناحیه‌ی چاهک به بخش ۲-۲ مراجعه کنید).

برای یک نقطه‌ی پرس‌وجوی p که درون ناحیه‌ای غیر چاهک قرار داشته باشد، یال‌های درجه اول $\mathrm{SPT}(p)$ را می‌توان از یال‌های اولیه‌ی ناحیه‌های چاهک به دست آورد (لم ۵). همچنین اطلاعات بحرانی ناحیه‌های قابلیت دید چندضلعی را می‌توان از اطلاعات بحرانی ناحیه‌های چاهک استخراج کرد (لم ۶). در نتیجه، با توجه به این که در یک چندضلعی ساده تعداد ناحیه‌های چاهک $O(n^2)$ است، زمان و حافظه‌ی پیش‌پردازش الگوریتم ما به $O(n^3 \log n)$ و $O(n^3)$ کاهش پیدا خواهد کرد.

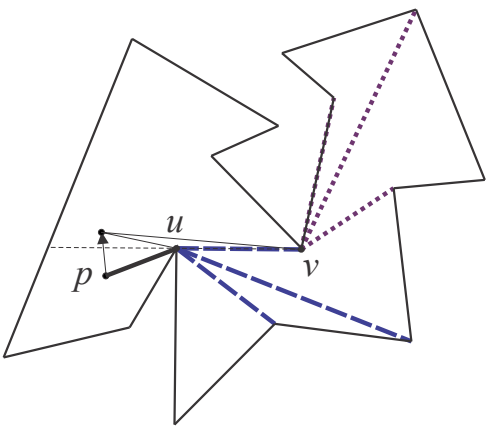


شکلِ ۳-۴: اگر پاره‌خط پرس‌وجوی pq درون چندضلعی $\mathcal{P}$ قرار داشت، چندضلعی را بر روی خط حاملِ pq به دو چندضلعی ساده تقسیم می‌کنیم. خطوط نقطه‌چین در عکس، بعضی از خطوط بحرانی چندضلعی‌های جدید را نشان می‌دهد.

در زمان پرس‌وجو، چنانچه هر دو نقطه‌ی p و q متعلق به ناحیه‌های چاهک باشند، اطلاعات لازم هر دو ناحیه را در اختیار داریم و می‌توانیم الگوریتم را به شکل عادی دنبال کنیم. اما چنانچه یکی از این دو نقطه به یک ناحیه‌ی غیرچاهک تعلق داشته باشد، لم‌های ۵ و ۶ نشان می‌دهند که یال‌های ثانویه و اطلاعات بحرانی راس‌های چندضلعی نسبت به این نقطه را می‌توانیم در زمان $O(\log n + |\mathrm{WVP}(pq)|)$ استخراج کنیم.

لم ۵ فرض کنید که برای یک ناحیه‌ی قابلیت دید $\mathcal{V}$، یال‌های ثانویه‌ی نوع اول محاسبه شده‌اند. برای یک ناحیه‌ی قابلیت دید همسایه که یال مشترکی با $\mathcal{V}$ دارد، می‌توانیم لیست یال‌های ثانویه‌ی نوع اول را در زمان ثابت از لیست مربوط به ناحیه‌ی $\mathcal{V}$ به دست آوریم.

اثبات ۳ وقتی نقطه‌ی دید p از مرز مشترک دو ناحیه‌ی قابلیت دید همسایه عبور می‌کند، یک راس نسبت به p قابل دید می‌شود و یا از دید خارج می‌گردد. به عنوان مثال، در شکل ۳-۵ وقتی p از مرز مشخص شده توسط u و v عبور می‌کند، یک یال نوع اول u تبدیل به یک یال اولیه p می‌شود، و همه‌ی یال‌های شروع شده از v تبدیل به یال‌های نوع اول می‌گردند. می‌توان مشاهده کرد که هیچ یال دیگری در این حرکت تغییر نمی‌کند. با توجه به این که این تغییرها شامل موارد زیر است، می‌توان آن‌ها را در زمان ثابت پردازش و اعمال کرد: حذف یک یال ثانویه از u (uv)، اضافه کردن یک یال اولیه (pv)، و انتقال یک اشاره‌گر به آرایه (یال‌های v) از یال‌های نوع دوم uv به یال‌های نوع اول pv. توجه کنید که موقعیت دقیق این تغییرات در لیست‌های متناظرشان مشخص است. تنها یال شرکت کننده در این تغییرات (یعنی یال متناظر با مرز ناحیه‌ی قطع شده) را می‌توان در زمان پیش‌پردازش محاسبه کرد. بنابراین زمان صرف شده در زمان پرس‌وجو $O(۱)$ خواهد بود.



شکلِ ۳–۵: وقتی p وارد یک ناحیه‌ی قابلیت دید جدید می‌شود ساختار ترکیبیاتی $\mathrm{SPT}(p)$ را می‌توان در زمان ثابت به‌روز کرد.

لم ۶ با در اختیار داشتن اطلاعات بحرانی راس‌ها برای یک ناحیه‌ی قابلیت دید، می‌توان این اطلاعات را برای ناحیه‌ی همسایه‌ی آن در زمان ثابت به دست آورد.

اثبات ۴ فرض کنید که اطلاعات بحرانی نقطه‌ی p را در اختیار داریم و p از خط بحرانی مشخص شده توسط uv عبور می‌کند، که u و v دو راس $\mathcal{P}$ هستند. به یاد داشته باشید که منظور ما از اطلاعات بحرانی یک نقطه، وضعیت بحرانی چپ بودن راس‌های چندضلعی نسبت به این نقطه می‌باشد. همان‌طور که در بخش ۳–۲–۲ اشاره شد، احتیاجی به نگهداری وضعیت گردش به راست راس‌ها نیست و این وضعیت در زمان پرس‌وجو بررسی می‌شود.

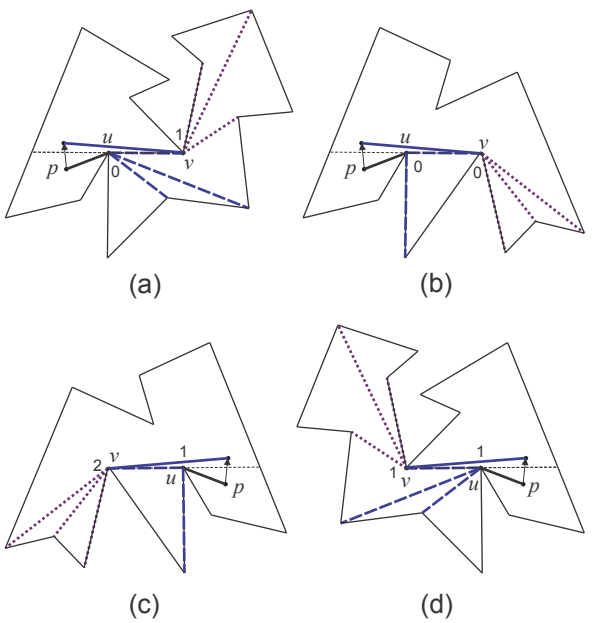
در عبور از مرز مشترک دو ناحیه، تنها راس‌هایی که به طور مستقیم تحت تاثیر قرار می‌گیرند، u و v هستند. با توجه به وضعیت بحرانی u و v نسبت به p، امکان رخ دادن چهار حالت وجود دارد (شکل ۳–۶ را ببینید). در سه حالت اول، وضعیت بحرانی v تغییر نخواهد کرد. اما در حالت چهارم، وضعیت بحرانی v تغییر می‌کند: قبل از برخورد با مرز، مسیر $\mathrm{SP}(p,v)$ یک گردش چپ در u انجام می‌دهد، بنابراین هم u و هم v نسبت به p بحرانی چپ هستند. اما بعد از برخورد، u بر روی $\mathrm{SP}(p,v)$ قرار ندارد و v دیگر بحرانی چپ نیست. به این خاطر، وضعیت بحرانی همه‌ی فرزندان v در $\mathrm{SPT}(p)$ هم ممکن است تغییر کند.

برای بررسی کردن این حالت‌ها، از یک روش به‌روزرسانی کندرو[1] برای پخش این تغییرات در زیردرخت استفاده می‌کنیم. به این منظور، روش ذخیره‌ی اطلاعات بحرانی هر راس نسبت به نقطه‌ی p را تغییر می‌دهیم. در هر راس چندضلعی، دو مقدار جدید را ذخیره می‌کنیم: عدد بحرانی[2] که عبارت است از تعداد راس‌های LC که در مسیر $\mathrm{SP}(p,v)$ دیده می‌شوند (شامل خود v)، و عدد بدهی[3] که عدد بحرانی‌ای است که باید در زیر درخت آن راس پخش گردد. اگر یک راس LC باشد به این معنی است که

[1] Lazy Updating

[2] Critical Number

[3] Debt Number



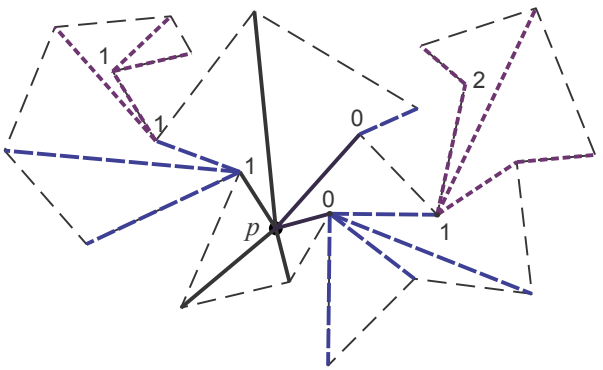
شکلِ ۳–۶: تغییرات اطلاعات بحرانی v نسبت به p، وقتی که p بین دو ناحیه‌ی همسایه حرکت می‌کند.

عدد بحرانی آن بزرگتر از صفر است (شکل ۳–۷ را ببینید). همچنین اگر یک راس عدد بدهی غیر صفر داشته باشد، عدد بحرانی فرزندان آن در $\mathrm{SPT}(p)$ باید با این عدد جمع شود. محاسبه و ذخیره‌ی این اعداد در کنار اطلاعات بحرانی تغییری در زمان و حافظه‌ی پیش‌پردازش نخواهد داشت. همچنین، در زمان پرس‌وجو و در حین پیمایش درخت، می‌توانیم عدد بحرانی راس‌ها را در زمان ثابت به‌روز کنیم.

با داشتن این اطلاعات جدید، باید عددهای بحرانی و بدهی را در حالت‌های سوم و چهارم شکل ۳–۶ تغییر دهیم. به عنوان مثال حالت چهارم را در نظر بگیرید. وقتی v نسبت به p قابل دید می‌شود، دیگر نسبت به p بحرانی چپ نیست. بنابراین عدد بحرانی v به ۰ تغییر می‌کند. اما به جای کاهش عدد بحرانی فرزندان v، عدد بدهی v را به ۱− کاهش می‌دهیم که نشان دهنده‌ی این است که عدد بحرانی راس‌های زیر درخت v باید یک واحد کاهش پیدا کنند. اعمال این کاهش در زمان پرس‌وجو و در هنگام پیمایش $\mathrm{SPT}(p)$ اتفاق می‌افتد. به طور مشابه، وقتی p در مسیر عکس حرکت کند و v نسبت به p غیر قابل دید گردد، با اضافه کردن ۱ به عدد بدهی، اعمال تغییرات را به شکل مشابهی به زمان پرس‌وجو منتقل می‌کنیم.

در زمان پیش‌پردازش، اطلاعات بحرانی و یال‌های درجه‌ی اول همه‌ی ناحیه‌های چاهک را محاسبه می‌کنیم. با توجه به لم‌های ۵ و ۶، دو ناحیه‌ی همسایه اطلاعات بحرانی و یال‌های درجه‌ی اول یکسانی دارند، مگر احتمالا در یک راس. این راس را در زمان پیش‌پردازش مشخص می‌کنیم. همچنین گراف جهت‌دار دوگانه‌ی ناحیه‌های قابلیت دید را می‌سازیم. در این گراف، هر راس معادل یک ناحیه‌ی قابلیت دید است و یک یال بین دو راس گراف معادل کاهش یا افزایش یک راس قابل دید در حرکت بین دو ناحیه‌ی قابلیت دید همسایه است.

در زمان پرس‌وجو، ناحیه‌ی قابلیت دید شامل p را پیدا می‌کنیم و بر روی گراف دوگانه، از این



شکلِ ۳–۷: تغییرات اطلاعات بحرانی v نسبت به p، وقتی که p بین دو ناحیه‌ی همسایه حرکت می‌کند.

ناحیه به یکی از چاهک‌ها حرکت می‌کنیم. با توجه به این که هر یال گراف متناظر با یک راس قابل دید نسبت به p و در نتیجه نسبت به pq است، تعداد یال‌های طی شده در این مسیر $O(|\mathrm{WVP}(pq|)$ خواهد بود. حال اطلاعات ناحیه‌ی چاهک به دست آمده را مبنا قرار داده و با برگشت از این ناحیه‌ی چاهک به ناحیه‌ی ابتدایی، اطلاعات بحرانی و یال‌های درجه‌ی اول ناحیه‌های میانی را به دست می‌آوریم. در نهایت با رسیدن به ناحیه‌ی ابتدایی، اطلاعات لازم مربوط به راس p را خواهیم داشت. پروسه‌ی مشابهی را برای نقطه‌ی q انجام می‌دهیم. حال با در اختیار داشتن اطلاعات p و q، با اجرای الگوریتم بخش ۳–۲–۲ می‌توانیم $\mathrm{WVP}(pq)$ را محاسبه کنیم. در نتیجه قضیه زیر را خواهیم داشت.

قضیه ۱ با پیش‌پردازش چندضلعی ساده‌ی $\mathcal{P}$ با n رأس در زمان $O(n^3 \log n)$ و ایجاد ساختاری به اندازه‌ی $O(n^3)$، می‌توانیم چندضلعی قابل دید ضعیف $\mathrm{WVP}(pq)$ را در زمان $O(\log n + |\mathrm{WVP}(pq)|)$ محاسبه و گزارش کنیم.

۳–۳ خلاصه

در این فصل به بررسی مسئله‌ی محاسبه‌ی چندضلعی قابل دید ضعیف یک پاره‌خط درون چندضلعی‌های ساده پرداختیم. نشان دادیم که با پیش‌پردازش یک چندضلعی ساده با n راس در زمان $O(n^3 \log n)$ و ایجاد ساختاری به اندازه‌ی $O(n^3)$، در زمان پرس‌وجو و با دریافت یک پاره‌خط pq، چندضلعی قابل دید ضعیف $\mathrm{WVP}(pq)$ را می‌توانیم در زمان $O(\log n + |\mathrm{WVP}(pq)|)$ محاسبه کنیم.

فصلِ ۴

الگوریتم دوم محاسبه‌ی WVP

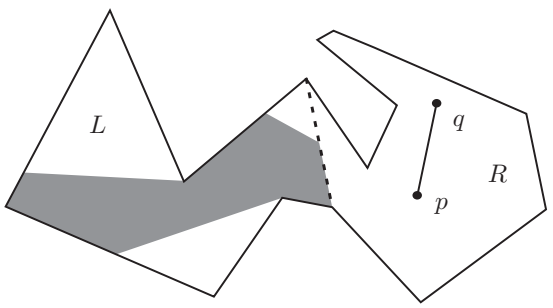
در این فصل دومین الگوریتمی که برای مسئله‌ی محاسبه‌ی چندضلعی قابل دید ضعیف پاره‌خط ارائه کرده‌ایم را شرح خواهیم داد. در یک چندضلعی ساده‌ی $\mathcal{P}$ با اندازه‌ی n، این الگوریتم به $O(n^2 \log n)$ زمان پیش‌پردازش و $O(n^2)$ حافظه نیاز دارد و $\mathrm{WVP}(pq)$ را به‌ازای هر پاره‌خط پرس‌وجوی pq واقع در چندضلعی، در زمان $O(\log^2 n + |\mathrm{WVP}(pq)|)$ محاسبه می‌کند.

این الگوریتم به صورت بازگشتی و با تقسیم متوالی چندضلعی ساده کار می‌کند. در ابتدا مفهوم قابلیت دید ضعیف جزئی را تعریف می‌کنیم و نشان می‌دهیم که چگونه آن را در زمان مناسب به دست آوریم. سپس از مثلث‌بندی متوازن استفاده کرده و بر اساس آن، روشی برای محاسبه‌ی چندضلعی قابل دید ضعیف به شکل بازگشتی ارائه می‌کنیم.

۴-۱ چندضلعی قابل دید جزئی

فرض کنید که چندضلعی ساده‌ی $\mathcal{P}$ به‌وسیله‌ی قطر e به دو بخش $\mathcal{L}$ و $\mathcal{R}$ تقسیم شده است. برای یک پاره‌خط $pq \in \mathcal{R}$، چندضلعی قابل دید ضعیف جزئی $\mathrm{WVP}_L(pq)$ (و یا $\mathrm{PWVP}_L(pq)$) را به شکل $\mathrm{WVP}(pq) \cap L$ تعریف می‌کنیم. به عبارت دیگر، $\mathrm{WVP}_L(pq)$ برابر است با بخشی از $\mathcal{P}$ که قابل دید ضعیف از طریق e باشد. در این بخش نشان می‌دهیم که چگونه $\mathrm{WVP}_L(pq)$ را به شکل حساس به خروجی محاسبه کنیم. سپس در بخش ۴-۵ از این نتیجه استفاده می‌کنیم و $\mathrm{WVP}(pq)$ را به دست می‌آوریم.

برای محاسبه‌ی $\mathrm{WVP}_L(pq)$، از الگوریتم Guibas و دیگران [۲۷] که در بخش ۳-۱ توضیح داده شد استفاده می‌کنیم. ایده‌ی کلی الگوریتم به این شکل است که تجزیه‌ی قابلیت دید چندضلعی را محاسبه می‌کنیم و به ازای هر سلول این تجزیه، ساختار درخت کوتاه‌ترین مسیر را به دست می‌آوریم. با توجه به این که تعداد ناحیه‌های قابلیت دید $O(n^3)$ است، هزینه‌ی پیش‌پردازشی که انجام خواهیم داد بالا خواهد بود.



شکلِ ۴-۱: قابلیت دید جزئی پاره‌خط pq به عنوان بخشی از زیرچندضلعی $\mathcal{L}$ که توسط pq قابل دید است تعریف می‌شود.

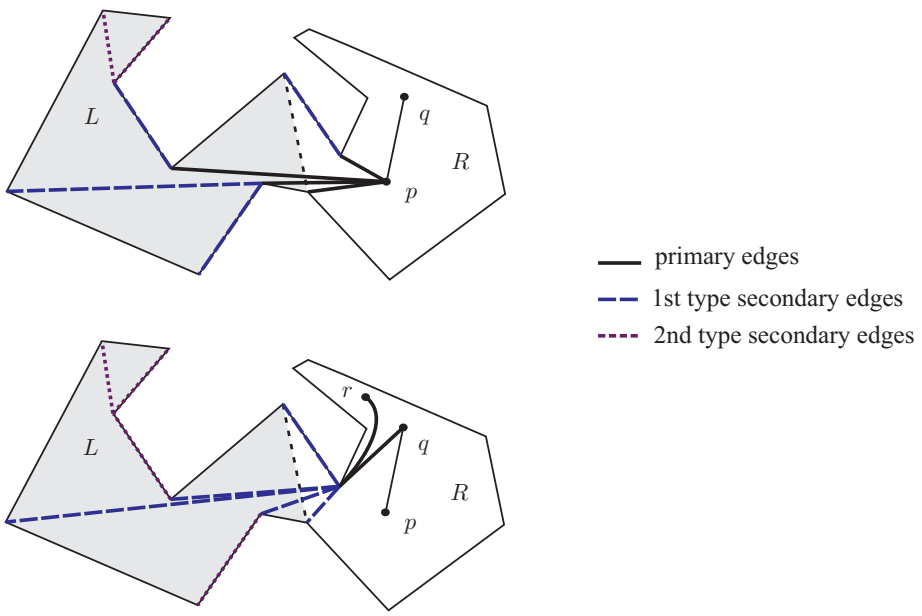
برای رفع این مسئله، تنها خطوط بحرانی‌ای را در نظر می‌گیریم که با قطر e برخورد کرده‌اند. تعداد این خطوط بحرانی برابر با $O(n)$ است و پیچیدگی تجزیه‌ی قابلیت دید تشکیل شده توسط این خطوط برابر با $O(n^2)$ است [۲]. تجزیه‌ی قابلیت دید تشکیل شده توسط این خطوط بحرانی را می‌توان در زمان $O(n^2)$ محاسبه کرد. به این تجزیه، تجزیه‌ی قابلیت دید جزئی $\mathcal{P}$ نسبت به e می‌گوییم. در ادامه‌ی این بخش نشان می‌دهیم که چگونه الگوریتم خطی Guibas و دیگران را تغییر دهیم به شکلی که $\mathrm{WVP}_L(pq)$ را بتوانیم در زمان حساس به خروجی محاسبه کنیم. در ابتدا نشان می‌دهیم که درخت کوتاه‌ترین مسیر چگونه قابل محاسبه است و سپس الگوریتم محاسبه‌ی $\mathrm{WVP}_L(pq)$ را بیان می‌کنیم.

۲-۴ محاسبه‌ی درخت کوتاه‌ترین مسیر جزئی $\mathrm{SPT}_L(p)$

درخت کوتاه‌ترین مسیر جزئی $\mathrm{SPT}_L(p)$ را به عنوان زیرمجموعه‌ای از $\mathrm{SPT}(p)$ تعریف می‌کنیم که به راس‌های درون $\mathcal{L}$ ختم می‌گردد. به عبارت دیگر، $\mathrm{SPT}_L(p)$ مجموعه‌ی همه‌ی کوتاه‌ترین مسیرها از p به راس‌های $\mathcal{L}$ می‌باشد. در این بخش نشان می‌دهیم که چگونه چندضلعی ساده‌ی $\mathcal{P}$ را پیش‌پردازش کنیم به شکلی که به ازای هر نقطه‌ی داده شده $p \in \mathcal{R}$، هر بخش دلخواه از $\mathrm{SPT}_L(p)$ به صورت حساس به خروجی قابل پیمایش باشد.

درخت کوتاه‌ترین مسیر $\mathrm{SPT}_L(p)$ شامل دو نوع یال است: یال‌های اولیه که ریشه‌ی p را به راس‌های قابل دیدش متصل می‌کنند، و یال‌های ثانویه که دو راس $\mathrm{SPT}_L(p)$ را به هم متصل می‌کنند. توجه داشته باشید که اگر نقطه‌ی p از یک خط بحرانی عبور کند و آن خط بحرانی با e برخورد نداشته باشد، در این صورت با این برخورد p ساختار $\mathrm{SPT}_L(p)$ تغییر نخواهد کرد. بنابراین می‌توانیم یال‌های اولیه‌ی خم شده داشته باشیم که p را به یک راس قابل دید از e متصل می‌کند (شکل ۴-۲ را ببینید). همچنین دو نوع یال ثانویه قابل تفکیک از هم هستند: یک یال ثانویه‌ی نوع اول (به اختصار یال نوع اول) یال ثانویه‌ای است که به یک یال اولیه متصل است، و یال ثانویه‌ی نوع دوم (به اختصار یال نوع دوم) یک یال ثانویه است که به هیچ یال اولیه‌ای متصل نیست.

یال‌های اولیه $\mathrm{SPT}_L(p)$ را می‌توانیم با الگوریتم محاسبه‌ی قابلیت دید جزئی Aronov [۲] به دست



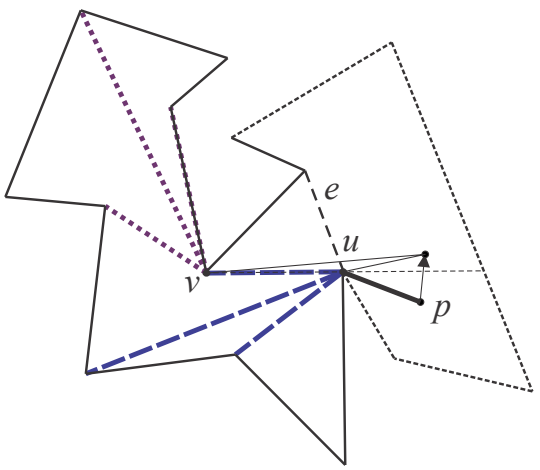
شکلِ ۴-۲: درخت SPT_L برای نقاط مختلف در $\mathcal{R}$. با توجه به این که q و r در ناحیه‌های یکسان قابلیت دید نسبت به $\mathcal{L}$ قرار گرفته‌اند، $\mathrm{SPT}_L(q)$ و $\mathrm{SPT}_L(r)$ ساختار یکسانی دارند.

آوریم. به بیان دقیق‌تر، با زمان پیش‌پردازش $O(n^۲ \log n)$ و فضای $O(n^۲)$، می‌توانیم لیست مرتب شده‌ی راس‌های قابل دید از p در $\mathcal{L}$ را در زمان $O(\log n)$ محاسبه کنیم.

همچنین باید لیست یال‌های ثانویه‌ی هر راس $\mathrm{SPT}_L(p)$ را محاسبه کنیم. یک راس r درون $\mathrm{SPT}_L(p)$ می‌تواند $O(n)$ یال نوع دوم احتمالی داشته باشد. بسته به این که پدر r چه راسی است، یک زیرلیستی[1] از این یال‌های احتمالی در $\mathrm{SPT}_L(p)$ مشارکت خواهند کرد. برای این که همه‌ی حالت‌های ممکن یال‌های نوع دوم r را ذخیره کنیم، همه‌ی این زیرلیست‌ها، یا به بیان بهتر شروع و پایان زیرلیست‌ها را، برای همه‌ی پدرهای احتمالی r محاسبه و ذخیره می‌کنیم. با توجه به این که یک راس $O(n)$ پدر احتمالی دارد، این محاسبات برای همه‌ی راس‌های چندضلعی در زمان $O(n^۲ \log n)$ و حافظه‌ی $O(n^۲)$ قابل انجام است. با داشتن این اطلاعات، در زمان پرس‌وجو، یال‌های نوع دوم هر راس در زمان ثابت قابل بازیابی خواهد بود.

ساختار مشابهی را برای یال‌های نوع اول می‌سازیم. پدر یک یال نوع اول ریشه‌ی درخت است. با توجه به این که ریشه‌ی درخت می‌تواند در هر یک از $O(n^۲)$ ناحیه‌ی قابلیت دید باشد، محاسبه و ذخیره‌ی یال‌های شروع و پایان در لیست یال‌های نوع اول یک راس به زمان و حافظه‌ی $O(n^۳ \log n)$ و $O(n^۳)$ نیاز دارد. با دانستن این نکته که در دو ناحیه‌ی قابلیت دید همسایه، لیست یال‌های نوع اول نسبت به هم دارای $O(۱)$ اختلاف هستند، می‌توانیم زمان و حافظه‌ی مورد نیاز برای به دست آوردن و نگهداری آن‌ها را کاهش دهیم.

[1] Sub-list



شکلِ ۴-۳: وقتی p وارد ناحیه‌ی جدید قابلیت دید می‌شود، ساختار ترکیبیاتی $\mathrm{SPT}_L(p)$ را می‌توان در زمان ثابت به‌روز کرد.

لم ۷ یک ناحیه‌ی قابلیت دید $\mathcal{V}$ را در نظر بگیرید و فرض کنید که یال‌های ثانویه‌ی نوع اول یک نقطه‌ی p درون این ناحیه را محاسبه کرده‌ایم. برای یک ناحیه‌ی همسایه که یال مشترکی با $\mathcal{V}$ دارد، این یال‌ها را می‌توان در زمان ثابت به‌روز کرد.

اثبات ۵ وقتی نقطه‌ی p از مرز بین دو ناحیه عبور می‌کند، یک راس چندضلعی نسبت به آن قابل دید و یا غیرقابل دید می‌شود. به عنوان مثال در شکل ۴-۳، وقتی p از مرز مشخص شده توسط u و v عبور می‌کند، یک یال نوع اول از u تبدیل به یک یال اولیه از p می‌شود، و یال‌های نوع دوم که از v شروع می‌شوند تبدیل به یال‌های نوع اول می‌شوند. می‌توان بررسی کرد که هیچ راس دیگری تغییر نخواهد کرد. با توجه به این که این تغییرات شامل موارد زیر است، پردازش آن‌ها را می‌توان در زمان ثابت به انجام رسانید: حذف یک یال ثانویه از u (uv)، اضافه کردن یک یال اولیه (pv)، و انتقال یک اشاره‌گر به آرایه (یال‌های v) از یال‌های نوع دوم uv به یال‌های نوع اول pv. تنها یالی که در این تغییرات شرکت دارد را می‌توان در زمان پیش‌پردازش مشخص کرد. بنابراین زمان صرف شده برای پرس‌وجو $O(1)$ خواهد بود.

با داشتن لم ۷ و استفاده از یک داده‌ساختار ماندگار[۲]، مثلا درخت قرمز-سیاه ماندگار [۴۰]، می‌توانیم هزینه‌ی نگهداری یال‌های نوع اول را کاهش دهیم. یک درخت قرمزسیاه ماندگار، درخت قرمز-سیاهی است که می‌تواند همه‌ی نسخه‌های میانی خود را به یاد بیاورد. اگر مجموعه‌ای از n شیء مرتب شده درون درخت ذخیره شده باشند و m به‌روز رسانی را بر روی درخت اعمال کنیم، هر نسخه‌ی tام که $1 \le t \le m$ است را می‌توانیم در زمان $O(\log n)$ بازیابی کنیم. داده ساختار درخت ماندگار را می‌توانیم در زمان $O((m+n)\log n)$ و حافظه‌ی $O(m+n)$ بسازیم.

[2] Persitent Data Structure

قضیه ۲ یک چندضلعی ساده‌ی $\mathcal{P}$ به اندازه‌ی n و قطر داده شده‌ی e از آن را می‌توان در زمان $O(n^2 \log n)$ و حافظه‌ی $O(n^2)$ پیش‌پردازش کنیم به نحوی که در زمان پرس‌وجو، درخت کوتاه‌ترین مسیر جزئی نقطه‌ی داده شده‌ی p را در زمان $O(\log n + k)$ گزارش کنیم، که k اندازه‌ی بخش گزارش شده‌ی درخت می‌باشد.

اثبات ۶ ابتدا از الگوریتم *Aronov* و دیگران [۲] برای محاسبه‌ی چندضلعی قابل دید جزئی p استفاده می‌کنیم. به این منظور به زمان پیش‌پردازش $O(n^2 \log n)$ و حافظه‌ی $O(n^2)$ احتیاج داریم. برای یال‌های ثانویه، به زمان $O(n^2 \log n)$ و حافظه‌ی $O(n^2)$ برای محاسبه‌ی یال‌ها نیاز داریم. همچنین داده‌ساختار مکان‌یابی بر روی تجزیه‌ی قابل دید جزئی ساخته می‌شود.

در زمان پرس‌وجو، ناحیه‌ی قابل دید جزئی p را در زمان $O(\log n)$ پیدا کرده و لیست راس‌های قابل دید از p را خواهیم داشت. با توجه به این که راس‌های قابل دید p متناظر با یال‌های اولیه در SPT_L می‌باشد، لیست این یال‌ها را هم به دست خواهیم آورد.

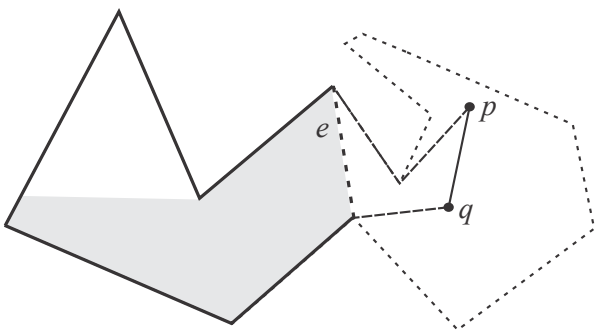
برای به دست آوردن یال‌های نوع اول، یک مسیر برای گردش در همه‌ی سلول‌های تجزیه‌ی قابلیت دید جزئی تشکیل می‌دهیم. با توجه به لم ۷، می‌توانیم از یک سلول دلخواه شروع کنیم و مسیر را برای عبور از همه‌ی سلول‌ها بپیماییم. در طول مسیر، درخت قرمز-سیاه ماندگار را برای لیست یال‌های نوع اول هر سلول می‌سازیم. با توجه به این که $O(n^2)$ سلول وجود دارد و هر سلول $O(n)$ یال نوع اول دارد، این داده‌ساختار را می‌توان در زمان $O(n^2 \log n)$ و حافظه‌ی $O(n^2)$ ایجاد کرد. با داشتن این داده‌ساختار، یال‌های نوع اول سلول شامل نقطه‌ی پرس‌وجوی p را می‌توان در زمان $O(\log n)$ از داده‌ساختار ماندگار استخراج کرد.

در نهایت، در هر راس درخت، لیست یال‌های نوع دوم از آن راس را داریم. بنابراین هزینه‌ی پیمایش درخت SPT_L برابر با هزینه‌ی اولیه‌ی $O(\log n)$ به علاوه‌ی تعداد راس‌های پیمایش شده خواهد بود. به عبارت دیگر هزینه‌ی پرس‌وجو برابر است با $O(\log n + k)$، که k راس‌های پیمایش شده در SPT_L است.

۴-۳ محاسبه‌ی $\mathrm{WVP}_L(pq)$

در این بخش نشان می‌دهیم که چگونه از الگوریتم خطی ارائه شده در بخش ۳-۱ استفاده کنیم و $\mathrm{WVP}_L(pq)$ را به شکل حساس به خروجی به دست آوریم. همان طور که در شکل ۴-۴ دیده می‌شود، این الگوریتم را می‌توانیم به مواردی که pq یک ضلع چندضلعی نیست گسترش دهیم. ایده‌ی کلی به این شکل است که یک سری اطلاعات اضافی در مورد راس‌های چندضلعی به دست آوریم و آن‌ها را ذخیره کنیم، به شکلی که در حین اجرای الگوریتم بخش ۳-۱، دو مرحله‌ی این الگوریتم را با هم ترکیب سازیم.

یک راس $v \in \mathcal{L}$ را بحرانی چپ (به اختصار LC) نسبت به نقطه‌ی $p \in \mathcal{R}$ می‌خوانیم، چنانچه وقتی در $\mathrm{SPT}_L(p)$ و در مسیر $\mathrm{SP}(p, v)$ حرکت می‌کنیم، این مسیر اولین گردش به چپ خود را در v انجام دهد. به عبارت دیگر، هر کوتاه‌ترین مسیر از p به یک راس غیر LC یک زنجیره‌ی محدب است که در راس‌های خود فقط گردش به راست انجام می‌دهد. وضعیت بحرانی یک راس به این گفته می‌شود که آن راس



شکلِ ۴-۴: در محاسبه‌ی $\mathrm{WVP}_L(pq)$ می‌توان فرض کرد که pq یک یال چندضلعی است.

LC هست یا نه. چنانچه وضعیت بحرانی همه‌ی راس‌های چندضلعی را نسبت به یک نقطه‌ی p بدانیم، می‌گوییم که اطلاعات بحرانی نقطه‌ی p را داریم. توجه داشته باشید که با توجه به این که در مرحله‌ی اول الگوریتم گردش به راست را در راس‌ها بررسی می‌کنیم، احتیاجی به ذخیره‌ی وضعیت گردش به راست در زمان پیش‌پردازش نداریم.

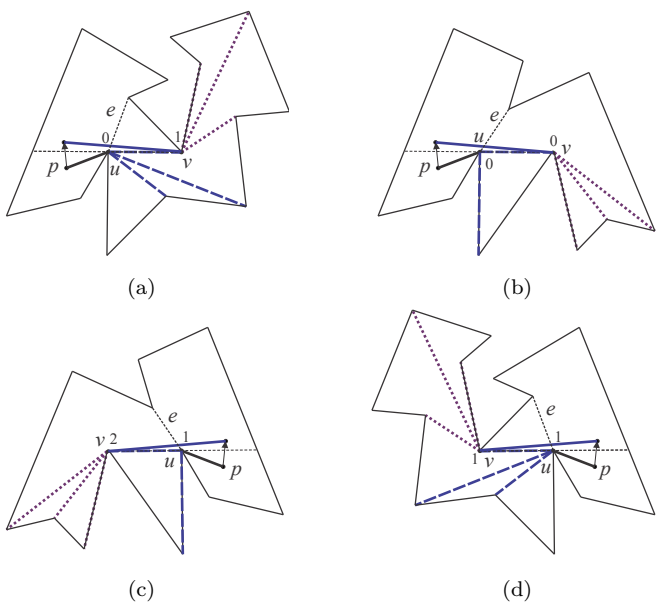
کلیات الگوریتم ما به این شکل است: در مرحله‌ی اول درخت $\mathrm{SPT}_L(p)$ را به صورت عمق اول پیمایش می‌کنیم. در هر راس بررسی می‌کنیم که آیا این راس نسبت به q بحرانی چپ هست یا خیر. اگر این‌چنین باشد، ما مطمئن خواهیم بود که فرزندان این راس نسبت به pq قابل دید ضعیف نخواهند بود. بنابراین پردازش این راس و زیردرخت آن را متوقف می‌کنیم تا در مرحله‌ی دوم دوباره به آن برسیم. در غیراین صورت، پردازش عادی راس را ادامه می‌دهیم و بررسی می‌کنیم که آیا یک گردش به راست صورت گرفته است یا نه (الگوریتم بخش ۳-۱ را ببینید). مرحله‌ی اول را تا بررسی همه‌ی راس‌های درخت ادامه می‌دهیم. در مرحله‌ی دوم، درخت $\mathrm{SPT}_L(q)$ را پیمایش کرده و الگوریتم را به شیوه‌ی قبل دنبال می‌کنیم.

لم ۸ همه‌ی راس‌های پیمایش شده در $\mathrm{SPT}_L(p)$ و $\mathrm{SPT}_L(q)$ راس‌های $\mathrm{WVP}_L(pq)$ هستند.

در مرحله‌ی پیش‌پردازش، اطلاعات بحرانی یک راس از هر ناحیه‌ی قابلیت دید را محاسبه کرده و این اطلاعات را به آن ناحیه نسبت می‌دهیم. در زمان پرس‌وجو و با دریافت پاره‌خط pq، ناحیه‌های قابلیت دید p و q را پیدا می‌کنیم. با استفاده از اطلاعات بحرانی این دو ناحیه، می‌توانیم الگوریتم را اجرا کرده و $\mathrm{WVP}_L(pq)$ را محاسبه کنیم.

با توجه به این که تعداد ناحیه‌ها در تجزیه‌ی قابلیت دید جزئی $O(n^2)$ است، برای نگهداری اطلاعات بحرانی ناحیه‌ها به فضای $O(n^3)$ احتیاج داریم. برای هر ناحیه، درخت کوتاه‌ترین مسیر SPT_L را برای یک نقطه‌ی دلخواه درون ناحیه محاسبه می‌کنیم و با پیمایش درخت، اطلاعات بحرانی هر راس را نسبت به آن ناحیه به دست می‌آوریم. برای هر ناحیه، یک آرایه به اندازه $O(n)$ برای نگهداری این اطلاعات بحرانی راس‌ها در نظر می‌گیریم. همچنین داده‌ساختار توضیح داده شده در بخش ۴-۲ برای محاسبه SPT_L را در زمان $O(n^2 \log n)$ و حافظه‌ی $O(n^2)$ محاسبه می‌کنیم.

در زمان پرس‌وجو، ناحیه‌های قابلیت دید p و q را در زمان $O(\log n)$ مشخص می‌کنیم. با توجه به این که زمان مورد نیاز برای پردازش هر راس برابر $O(1)$ است، و با در نظر گرفتن لم ۸، زمان پرس‌وجو



شکلِ ۴–۵: تغییرات در وضعیت بحرانی راس v نسبت به pq، وقتی p بین دو ناحیه حرکت می‌کند.

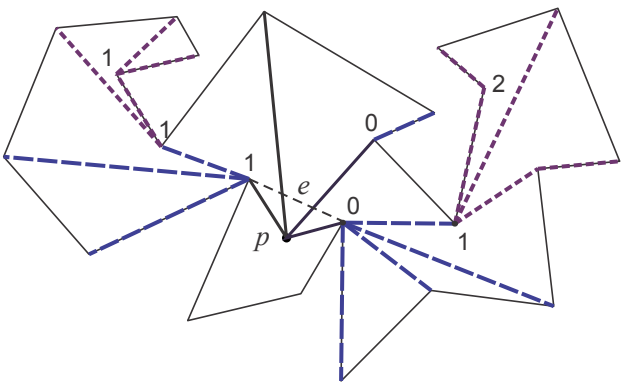
برابر $O(\log n + |\mathrm{WVP}_L(pq)|)$ خواهد بود.

لم ۹ با فرض در اختیار داشتن اطلاعات بحرانی راس‌ها برای یک نقطه در یک ناحیه‌ی قابلیت دید جزئی، این اطلاعات را می‌توان در زمان ثابت برای ناحیه‌ی همسایه‌ی آن به دست آورد.

اثبات ۷ فرض کنید که اطلاعات بحرانی نقطه‌ی p را در اختیار داریم و p از خط بحرانی مشخص شده توسط uv عبور می‌کند، که u و v دو راس $\mathcal{P}$ هستند. در عبور از مرز مشترک دو ناحیه، تنها راس‌هایی که به طور مستقیم تحت تاثیر قرار می‌گیرند u و v هستند. با توجه به وضعیت بحرانی u و v نسبت به p، امکان رخ دادن چهار حالت وجود دارد (شکل ۳–۶ را ببینید). در سه حالت اول، وضعیت بحرانی v تغییر نخواهد کرد. اما در حالت چهارم، وضعیت بحرانی v تغییر می‌کند. قبل از برخورد با خط بحرانی، کوتاه‌ترین مسیر $\mathrm{SP}(p,v)$ یک گردش چپ در u انجام می‌دهد، بنابراین هم u و هم v نسبت به p بحرانی چپ هستند. اما بعد از برخورد، u بر روی $\mathrm{SP}(p,v)$ قرار ندارد و v دیگر بحرانی چپ نیست. به این خاطر وضعیت بحرانی همه‌ی فرزندان v در $\mathrm{SPT}_L(p)$ هم ممکن است تغییر کند.

برای بررسی کردن این حالت‌ها، از یک روش به‌روز رسانی کندرو[3] برای پخش این تغییرات درون زیردرخت استفاده می‌کنیم. به این منظور، روش ذخیره‌ی اطلاعات بحرانی هر راس نسبت به نقطه‌ی p را تغییر می‌دهیم. در هر راس چندضلعی، دو مقدار جدید را ذخیره می‌کنیم: عدد بحرانی که عبارت است از تعداد راس‌های LC که در مسیر $\mathrm{SP}(p,v)$ دیده می‌شوند (شامل خود v)، و عدد بدهی که عدد بحرانی‌ای است که باید در زیر درخت آن راس پخش گردد. اگر یک راس LC باشد به این معنی است که عدد بحرانی

[3] Lazy Updating



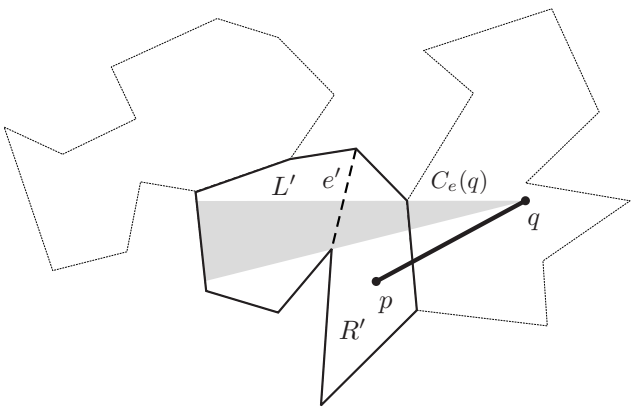
شکلِ ۴–۶: عدد بحرانی برابر است با تعداد راس‌های بحرانی چپ که از p در $\mathrm{SPT}_e(p)$ دیده می‌شود.

آن بزرگتر از صفر است (شکل ۴–۶ را ببینید). همچنین اگر یک راس عدد بدهی غیر صفر داشته باشد، عدد بحرانی فرزندان آن در $\mathrm{SPT}(p)$ باید با این عدد جمع شود. توجه کنید که محاسبه و ذخیره‌ی این اعداد در کنار اطلاعات بحرانی تغییری در زمان و حافظه‌ی پیش‌پردازش نخواهد داشت. همچنین در زمان پرس‌وجو و در حین پیمایش درخت، می‌توانیم عدد بحرانی راس‌ها را در زمان ثابت به‌روز کنیم.

با داشتن این اطلاعات جدید، باید اعداد را در حالت‌های سوم و چهارم شکل ۴–۵ تغییر دهیم. به عنوان مثال حالت چهارم را در نظر بگیرید. وقتی v نسبت به p قابل دید می‌شود، دیگر نسبت به p بحرانی چپ نیست. بنابراین عدد بحرانی v به ۰ تغییر می‌کند. اما به جای کاهش عدد بحرانی فرزندان v، عدد بدهی v را به ۱− کاهش می‌دهیم که نشان دهنده‌ی این است که عدد بحرانی راس‌های زیر درخت v باید یک واحد کاهش پیدا کنند. اعمال این کاهش در زمان پرس‌وجو و در هنگام پیمایش $\mathrm{SPT}_L(p)$ اتفاق می‌افتد. به طور مشابه، وقتی p در مسیر عکس حرکت کند و v نسبت به p غیر قابل دید گردد، با اضافه کردن ۱ به عدد بدهی، تغییرات را به شکل مشابهی در زمان پرس‌وجو اعمال می‌کنیم.

با توجه به لم ۹ می‌توانیم از یک داده‌ساختار ماندگار برای کاهش زمان پیش‌پردازش به $O(n^2 \log n)$ و حافظه‌ی $O(n^2)$ استفاده کنیم. یک مسیر برای پیمایش همه‌ی سلول‌های تجزیه‌ی قابلیت دید جزئی تشکیل می‌دهیم و درخت قرمز-سیاه ماندگار برای اطلاعات بحرانی سلول‌ها تشکیل می‌دهیم. این داده‌ساختار را می‌توان در زمان $O(n^2 \log n)$ و حافظه‌ی $O(n^2)$ ایجاد کرد. همچنین در زمان و حافظه‌ی $O(n^2)$ داده ساختار مکان‌یابی را بر روی تجزیه‌ی قابلیت دید ایجاد می‌کنیم [۳۱]. در نهایت قضیه‌ی زیر را خواهیم داشت.

قضیه ۳ چندضلعی ساده‌ی $\mathcal{P}$ با n راس را که به وسیله‌ی قطر e به دو بخش $\mathcal{L}$ و $\mathcal{R}$ تقسیم شده است می‌توان در زمان $O(n^2 \log n)$ و با مصرف $O(n^2)$ حافظه به‌گونه‌ای پیش پردازش کرد که چندضلعی قابل دید جزئی $\mathrm{WVP}_L(pq)$ برای هر پاره‌خط pq واقع در $\mathcal{R}$ را بتوان در زمان $O(\log n + |\mathrm{WVP}_L(pq)|)$ محاسبه کرد.



شکلِ ۴-۷: با پیش‌پردازش چندضلعی در زمان و حافظه‌ی $O(n)$، می‌توان زیرچندضلعی $\mathcal{P}'$ به اندازه‌ی m را در زمان و حافظه‌ی $O(m^2 \log m)$ و $O(m^2)$ پیش‌پردازش کرد به نحوی که $\mathrm{WVP}'_L(pq)$ را برای یک پاره‌خط گسترش یافته در زمان $O(\log m + k')$ به دست آورد، که k' اندازه‌ی $\mathrm{WVP}'_L(pq)$ درون $\mathcal{P}'$ می‌باشد.

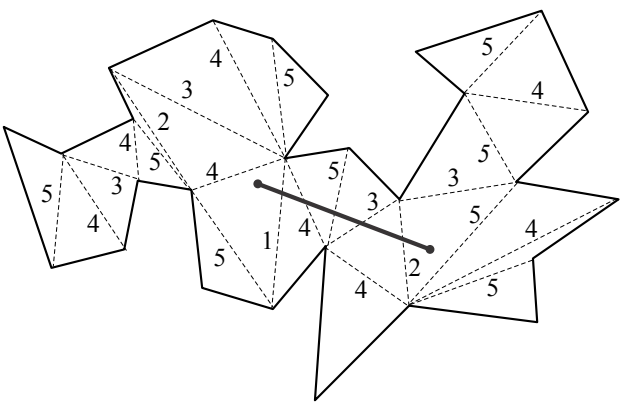
۴-۴ محاسبه‌ی $\mathrm{WVP}_L(pq)$ برای پاره‌خط‌های گسترش یافته

در این بخش مفهوم چندضلعی قابل دید ضعیف را برای پاره‌خط‌های گسترش یافته تعریف می‌کنیم. فرض کنید که $\mathcal{P}$ یک چندضلعی ساده با n راس و $\mathcal{P}'$ یک زیرچندضلعی از $\mathcal{P}$ باشد (شکل ۴-۷ را ببینید). همچنین فرض کنید که اندازه‌ی $\mathcal{P}'$ برابر m است و این چندضلعی توسط قطر e' به دو بخش L' و R' تقسیم شده است. پاره خط pq نیز در سمت راست e' بوده و R' را قطع می‌کند. از آنجایی که pq کاملا درون R' نیست، به آن یک پاره‌خط گسترش یافته می‌گوییم. چندضلعی قابل دید جزئی یک پاره‌خط گسترش یافته pq نسبت به e' را به شکل مشابهی تعریف می‌کنیم. یعنی بخشی از L' که توسط pq و از طریق e' قابل دید می‌باشد.

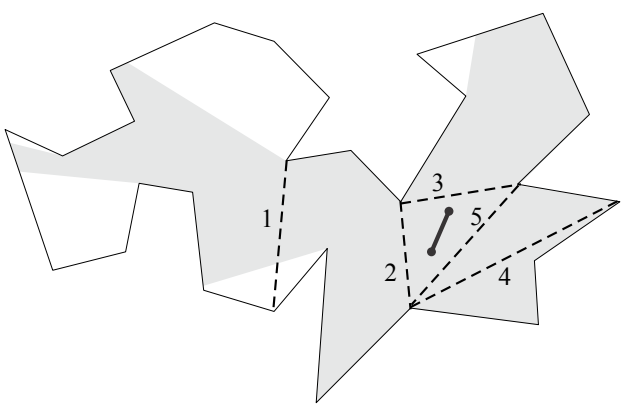
می‌توان نشان داد که با پیش‌پردازش $\mathcal{P}$ در زمان و حافظه‌ی $O(n)$، به ازای هر زیرچندضلعی $\mathcal{P}'$ از $\mathcal{P}$، می‌توانیم $\mathcal{P}'$ را در زمان $O(m^2 \log m)$ و حافظه‌ی $O(m^2)$ پیش‌پردازش کنیم به نحوی که به ازای یک پاره‌خط pq گسترش یافته نسبت به $\mathcal{P}'$، چندضلعی قابل دید جزئی را در زمان $O(\log m + |\mathrm{WVP}_{L'}(pq)|)$ به دست آوریم. در این جا n اندازه‌ی $\mathcal{P}$ و m اندازه‌ی $\mathcal{P}'$ است.

۵-۴ محاسبه‌ی WVP با استفاده از مثلث‌بندی متوازن

به ازای هر چندضلعی ساده‌ی $\mathcal{P}$، یک قطر e وجود دارد که $\mathcal{P}$ را به دو بخش تقسیم می‌کند و اندازه‌ی هر بخش حداکثر $2n/3$ می‌باشد [۷]. با استفاده از این موضوع، می‌توانیم $\mathcal{P}$ را به‌صورت بازگشتی و با افزودن قطرهای داخلی به بخش‌های $\mathcal{L}$ و $\mathcal{R}$ تقسیم کنیم تا در پایان به یک مثلث‌بندی متوازن برسیم. این نوع



شکلِ ۴–۸: مثلث‌بندی متوازن یک چندضلعی ساده. از این مثلث‌بندی برای محاسبه‌ی بازگشتی قابلیت دید پاره‌خط استفاده می‌کنیم.

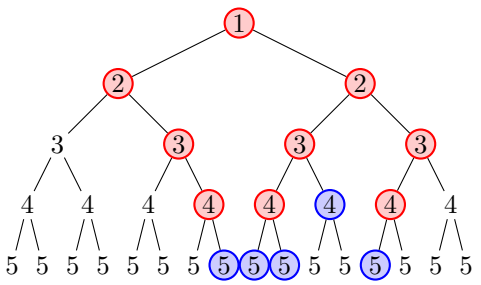


شکلِ ۴–۹: حالت ابتدایی در محاسبه‌ی قابلیت دید ضعیف پاره‌خط.

مثلث‌بندی منجر به یک درخت متوازن می‌گردد که برگ‌های آن مثلث‌های تشکیل شده هستند. نمونه‌ای از این مثلث بندی متوازن در شکل ۴–۸ نشان داده شده است. در این درخت متوازن هر راس میانی i متناظر با یک زیرچندضلعی $\mathcal{P}_i$ است که توسط قطر e_i به دو زیرچندضلعی $\mathcal{L}_i$ و $\mathcal{R}_i$ تقسیم می‌شود.

برای محاسبه‌ی $\mathrm{WVP}(pq)$، ابتدا p و q را در میان مثلث‌های نهایی پیدا می‌کنیم. در ساده‌ترین حالت p و q هر دو متعلق به یک مثلث هستند (شکل ۴–۹). به ازای هر i از ریشه‌ی درخت تا برگ، $\mathrm{PWVP}_i(pq)$ را به دست می‌آوریم. در اینجا $\mathrm{PWVP}_i(pq)$ برابر با چندضلعی قابل دید جزئی pq در $\mathcal{P}_i$ نسبت به e_i است. برای برگ درخت، این مقدار برابر با مثلثی که پاره‌خط درون آن قرار دارد می‌باشد. برای سایر موارد، بر اساس الگوریتم بیان شده برای چندضلعی قابل دید جزئی محاسبات انجام می‌شود. در هر مرحله، اتصال چندضلعی‌های قابل دید به هم در زمان $O(\log n)$ قابل انجام می‌باشد.

با توجه به این که ساخت داده‌ساختار مورد نیاز برای پاسخ به مسئله‌ی چندضلعی قابل دید ضعیف در یک چندضلعی با m راس، به $O(m^2 \log m)$ زمان و $O(m^2)$ حافظه نیاز دارد، در مورد حافظه‌ی مصرفی



شکلِ ۴–۱۰: راس‌های مشخص شده در درخت، متناظر با چندضلعی‌های قابل دید محاسبه شده مربوط به مسئله‌ی شکل ۴–۸ می‌باشد.

و زمان اجرای مربوط به پیش پردازش کل، که به ترتیب با $S(n)$ و $T(n)$ نشان داده می‌شوند، رابطه‌های زیر برقرار است:

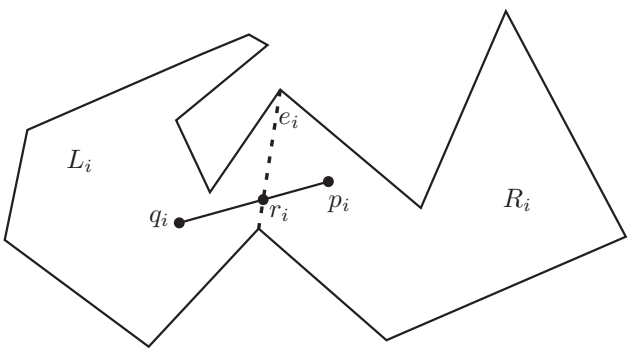
$$S(n) = \max_{n/۳ \le m \le ۲n/۳}(S(m) + S(n-m)) + \Theta(n^۲)$$

$$T(n) = \max_{n/۳ \le m \le ۲n/۳}(T(m) + T(n-m)) + \Theta(n^۲ \log n)$$

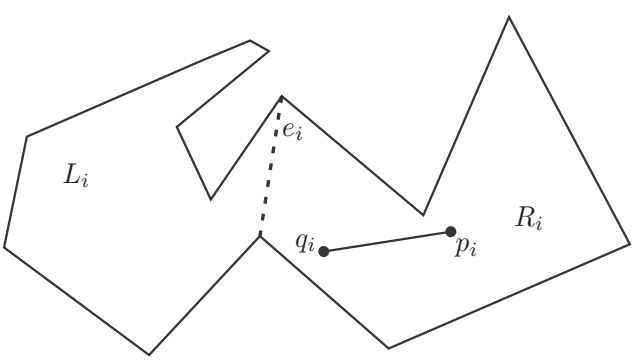
بنابراین کل حافظه مصرفی $O(n^۲)$ و زمان اجرای پیش پردازش $O(n^۲ \log n)$ است. به ازای هر پاره‌خط پرس‌وجو، مثلثی که پاره‌خط در آن واقع شده است را در زمان $O(\log n)$ به دست می‌آوریم. با توجه به متوازن بودن مثلث‌بندی، هر مسیر از برگ به ریشه‌ی درخت دارای اندازه‌ی $O(\log n)$ است. همان‌طور که در قضیه‌ی ۳ نشان دادیم، زمان مورد نیاز برای محاسبه‌ی $\mathrm{PWVP}_i(pq)$ برابر با $O(\log n + \mathrm{PWVP}_i(pq))$ می‌باشد. همچنین ترکیب چندضلعی‌های به دست آمده در هر مرحله را در زمان $O(\log n)$ می‌توان انجام داد. بنابراین زمان کلی پرس‌وجو برابر با $O(\log n + \sum_i(\log n + |\mathrm{PWVP}_i(pq)|))$ و یا $O(\log^۲ n + |\mathrm{WVP}(pq)|)$ خواهد بود.

قسمت دشوار وقتی است که p و q در دو مثلث مجزا باشند. فرض کنید که در مرحله‌ی iام پاره‌خط پرس‌وجوی p_iq_i درون زیرچندضلعی $\mathcal{P}_i$ باشد که توسط قطر e_i به دو قسمت $\mathcal{L}_i$ و $\mathcal{R}_i$ تقسیم شده است. اگر p_iq_i با e_i برخورد نداشته باشد، بدون از بین رفتن کلیت مسئله، فرض کنید که p_iq_i در $\mathcal{R}_i$ قرار گرفته باشد (شکل ۴–۱۱ را ببینید). در این حالت روند عادی الگوریتم را انجام می‌دهیم و $\mathrm{PWVP}_{L_i}(p_iq_i)$ را محاسبه می‌کنیم. سپس به شکل بازگشتی چندضلعی قابل دید ضعیف جزئی را در $\mathcal{R}_i$ محاسبه می‌کنیم. در این حالت، زمان مورد نیاز به شکل $T(n_i, p_iq_i) = T(n_i/۲, p_iq_i) + O(\log n_i) + \mathrm{PWVP}_{L_i}(p_iq_i)$ قابل بیان است.

از طرف دیگر اگر p_iq_i با e_i در نقطه‌ی r_i برخورد داشته باشد، بدون از بین رفتن کلیت مسئله فرض می‌کنیم که p_i در R_i و q_i در L_i قرار داشته باشد (شکل ۴–۱۲ را ببینید). می‌توان $\mathrm{WVP}(p_iq_i)$ را به شکل اجتماع دو زیرمسئله قابلیت دید جزئی برای پاره‌خط‌های گسترش یافته نشان داد: چندضلعی قابل دید از p_iq_i در $\mathcal{R}_i$ و چندضلعی قابل دید از p_iq_i در $\mathcal{L}_i$. به عبارت دیگر برای به دست آوردن جواب باید جواب دو زیرمسئله را به دست بیاوریم. با داشتن جواب این دو زیرمسئله، اجتماع آن‌ها را می‌توان در زمان



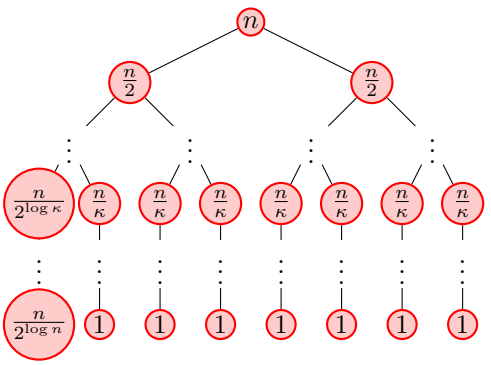
شکلِ ۴-۱۱: حالت اول در محاسبه‌ی بازگشتی قابلیت دید پاره‌خط. این حالت با عبارت $\mathrm{WVP}(p_iq_i) = \mathrm{WVP}_{L_i}(p_iq_i) + \mathrm{WVP}_{R_i}(p_iq_i)$ قابل بیان است.



شکلِ ۴-۱۲: حالت دوم در محاسبه‌ی بازگشتی قابلیت دید پاره‌خط. این حالت با عبارت $\mathrm{WVP}(p_iq_i) = \mathrm{WVP}_{R_i}(p_iq_i) + \mathrm{WVP}_{L_i}(p_iq_i)$ قابل بیان است.

$O(|\mathrm{WVP}_{\mathcal{P}_i}(p_iq_i)|)$ به دست آورد. با توجه به قضیه ۳، زمان پرس‌وجوی صرف شده در این مرحله برابر با $T(n_i, p_iq_i) = ۲T(n_i/۲, p_iq_i) + O(\log n_i)$ خواهد بود.

هزینه‌ی پیش‌پردازش الگوریتم همانند قبل است. در مورد هزینه‌ی زمان پرس‌وجو یک تحلیل خام به رابطه‌ی $T(n) = ۲T(n/۲) + O(\log n)$ و یا $T(n) = O(n)$ منجر می‌شود. در اینجا نشان می‌دهیم که چگونه زمان پرس‌وجو را به تعداد برخوردهای پاره‌خط pq با قطرهای برشی وابسته کنیم. اگر pq در κ مرحله با قطری از مثلث بندی متوازن برخورد کند، κ شاخه در درخت دودویی متناظر مثلث‌بندی خواهیم داشت. با توجه به این که هزینه‌ی زمان اجرا به تعداد شاخه‌های طی شده درون درخت دودویی بستگی دارد، برای یک مقدار ثابت κ، بیش‌ترین هزینه زمانی اتفاق می‌افتد که این شاخه‌ها در نزدیکی ریشه رخ دهند. شکل ۴-۱۳ این حالت را نشان می‌دهد. همان‌طور که دیده می‌شود، این درخت شامل یک درخت متوازن کامل به ارتفاع $\log \kappa$ و κ برگ، و همچنین κ مسیر که طول هرکدام $\log n - \log k = \log n/\kappa$ است می‌باشد. بخش اول درخت متناظر با معادله‌ی $T(m) = ۲T(m/۲) + \log m$ است. همچنین هر کدام از



شکلِ ۴-۱۳: بدترین حالت برای محاسبه‌ی $\mathrm{WVP}(pq)$. این درخت تشکیل شده از یک درخت متوازن به ارتفاع $\log\kappa$، و κ مسیر به اندازه‌ی $\log n - \log\kappa = \log\frac{n}{\kappa}$ می‌باشد.

مسیرهای بخش دوم متناظر با معادله‌ی $T(m) = T(m/۲) + \log m$ و با مقدار اولیه‌ی $m = n/\kappa$ می‌باشد. این معادلات بازگشتی را می توان به شکل زیر گسترش داد:

$$T(n) = \kappa \sum_{i=۱}^{n/\kappa} \log i + \sum_{i=۰}^{\log_۲ \kappa} ۲^i \log(n/۲^i)$$

$$T(n) = \kappa \log^۲ \tfrac{n}{\kappa} + \log n \sum_{i=۰}^{\log_۲ \kappa} ۲^i - \sum_{i=۰}^{\log_۲ \kappa} i ۲^i$$

با استفاده از رابطه‌ی جمع سری هندسی و فرمول زیر

$$\sum_{k=۱}^{n} k a^k = \tfrac{a(۱-a^n)}{(۱-a)^۲} - \tfrac{n a^{n+۱}}{۱-a}$$

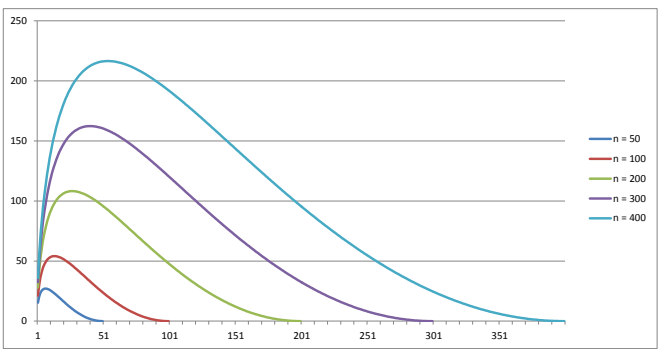
خواهیم داشت:

$$T(n) \le \kappa \log^۲(n/\kappa) + \kappa \log n - \kappa \log\kappa$$

$$T(n) \le \kappa \log^۲(n/\kappa) + \kappa \log(n/\kappa)$$

با توجه به این که هر نقطه‌ی برخورد پاره‌خط پرس‌وجو با یک قطر برشی را می‌توان به یک راس $\mathrm{WVP}(pq)$ نگاشت کرد، $\kappa \le |\mathrm{WVP}(pq)|$. بنابراین در حالت کلی زمان پرس‌وجو برابر با $O(|\mathrm{WVP}(pq)| + \log^۲ n + \kappa \log^۲(n/\kappa))$ خواهد بود. در مجموع خواهیم داشت:

قضیه ۴ یک چندضلعی ساده‌ی $\mathcal{P}$ را می‌توانیم در زمان $O(n^۲ \log n)$ و حافظه‌ی $O(n^۲)$ پیش‌پردازش کنیم به طوری‌که برای هر پاره‌خط پرس‌وجوی pq، $\mathrm{WVP}(pq)$ را در زمان $O(|\mathrm{WVP}(pq)| + \kappa \log^۲ \frac{n}{\kappa})$ گزارش کنیم، که در آن $\kappa \le |\mathrm{WVP}(pq)|$ مقداری وابسته به پاره‌خط pq می‌باشد.



شکلِ ۴-۱۴: نمودار $\kappa \log^{۲}(n/\kappa)$ برای چند مقدار n. محور افقی κ و محور عمودی مقدار $\kappa \log^{۲}(n/\kappa)$ می‌باشد. در هر کدام از این منحنی‌ها مقدار بیشینه در $\kappa = \exp^{-۲} n$ به دست می‌آید، و مقدار بیشینه به ازای این مقدار برابر با $\kappa = \exp^{-۲} n$ است.

۴-۵-۱ تحلیل زمان پرس‌وجو

زمان پاسخ‌گویی به پرس‌وجو در الگوریتم ارائه شده برابر با $O(|\mathrm{WVP}(pq)| + \log^{۲} n + \kappa \log^{۲}(n/\kappa))$ است، که در آن κ تعداد نقاط برخورد پاره‌خط پرس‌وجو با قطرهای مثلث‌بندی متوازن است. هرچند $\kappa = O(|\mathrm{WVP}(pq)|)$، در بیشتر مواقع κ بسیار کمتر از $|\mathrm{WVP}(pq)|$ می‌باشد.

برای مقادیر کوچک κ، می‌توانیم بگوییم $\kappa \log^{۲}(n/\kappa) = O(\log^{۲} n)$، و برای مقادیر بالای آن، $\kappa \log^{۲}(n/\kappa) = O(\kappa) = O(|\mathrm{WVP}(pq)|)$. نمودار نشان داده شده در شکل ۴-۱۴ رفتار پارامتر $\kappa \log^{۲}(n/\kappa)$ را برای مقادیر مختلف n و κ نشان می‌دهد. برای یک مقدار ثابت n، مشتق این عبارت نسبت به κ برابر است با $\log^{۲} \kappa + ۲ \log \kappa \log n + (-۲ \log \kappa - ۲) \log n + \log^{۲} n$. یک محاسبه‌ی ساده نشان می‌دهد که زمان اجرای الگوریتم به ازای $\kappa = \exp^{-۲}(n)$ بیشترین مقدار خود را خواهد داشت، و مقدار بیشینه برابر با ۴κ خواهد بود. برای مقادیر بالاتر κ، مقدار $\kappa \log^{۲}(n/\kappa)$ کاهش پیدا کرده و می‌توان گفت که برای $\kappa \geq \exp^{-۲}(n)$، $\kappa \log^{۲}(n/\kappa) = O(\kappa) = O(|\mathrm{WVP}(pq)|)$. به عبارت دیگر برای مقادیر $\kappa \geq \exp^{-۲}(n)$، زمان پرس‌وجو برابر است با $O(|\mathrm{WVP}(pq)| + \log^{۲} n)$.

۴-۶ خلاصه

در این فصل نتایج فصل ۳ را برای محاسبه‌ی چندضلعی قابل دید ضعیف یک پاره‌خط درون چندضلعی ساده با n راس دنبال کردیم. در بخش اول، چندضلعی قابل دید ضعیف جزئی یک پاره‌خط pq نسبت به قطر e را تعریف کرده و نشان دادیم که چگونه با پیش‌پردازش چندضلعی در زمان $O(n^{۲} \log n)$ و حافظه‌ی $O(n^{۲})$، این چندضلعی را در زمان پرس‌وجوی $O(\log n + |\mathrm{WVP}_e(pq)|)$ به دست آوریم.

در بخش دوم، با توجه به نتیجه‌ی بخش اول، از مثلث‌بندی متوازن استفاده کرده و نشان دادیم با هزینه‌ی پیش‌پردازش گفته شده، چندضلعی قابل دید ضعیف پاره‌خط pq را می‌توان در زمان

$O(\log^۲ n + \kappa \log^۲(n/\kappa) + |\mathrm{WVP}(pq)|)$ به دست آورد. در اینجا κ برابر است با تعداد مثلث‌های درون مثلث‌بندی که با pq برخورد می‌کنند، و $\kappa \leq |\mathrm{WVP}(pq)|$.

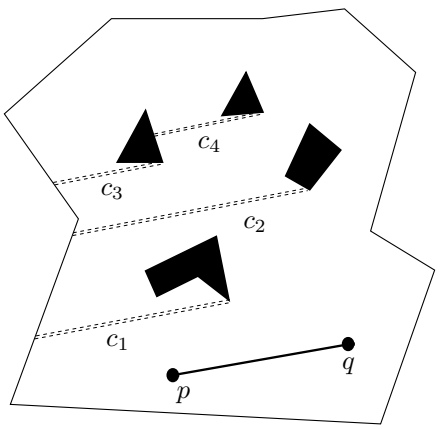
فصلِ ۵

محاسبه‌ی WVP در چندضلعی‌های حفره‌دار

در این فصل به محاسبه‌ی چندضلعی قابل دید ضعیف پاره‌خط در چندضلعی‌های حفره‌دار می‌پردازیم. فرض کنید که $\mathcal{P}$ یک چندضلعی با h حفره و در مجموع n راس باشد. همچنین فرض کنید که pq یک پاره‌خط پرس‌وجو درون این چندضلعی باشد. روشی که ارائه می‌دهیم الهام گرفته از الگوریتم محاسبه چندضلعی قابل دید یک نقطه درون چندضلعی حفره‌دار که در [۴۵] ارائه شده است می‌باشد. در این روش، چندضلعی حفره‌دار $\mathcal{P}$ را با افزودن تعدادی قطر برشی به چندضلعی ساده $\mathcal{P}_s$ تبدیل می‌کنیم. سپس، از الگوریتم ارائه شده برای محاسبه چندضلعی قابل دید ضعیف در چندضلعی‌های ساده استفاده می‌کنیم و صورت اولیه‌ای از $\mathrm{WVP}(pq)$ را به دست می‌آوریم. در نهایت، با انجام محاسباتی اضافه، مقدار نهایی $\mathrm{WVP}(pq)$ را به دست می‌آوریم.

یک حفره $\mathcal{H}$ را می‌توان با اضافه کردن دو قطر برشی که یک راس $\mathcal{H}$ را به مرز $\mathcal{P}$ متصل می‌کنند حذف کرد. با بریدن $\mathcal{P}$ در راستای این قطرهای برشی، به یک چندضلعی جدید می‌رسیم که در آن $\mathcal{H}$ جزئی از مرز چندضلعی است. این کار را برای همه‌ی حفره‌های چندضلعی تکرار می‌کنیم و در نهایت به چندضلعی ساده $\mathcal{P}_s$ می‌رسیم.

فرض کنید که l خط حامل پاره‌خط pq باشد. برای سادگی می‌توانیم فرض کنیم که همه‌ی حفره‌ها بالای خط pq قرار دارند. اگر این چنین نباشد، می‌توان $\mathcal{P}$ را در راستای خط l به دو چندضلعی تقسیم کرد و دو زیر مسئله که این شرط را دارا هستند به دست آورد. برای اضافه کردن قطرهای برشی برای یک حفره، نزدیک‌ترین نقطه‌ی هر حفره به خط l را انتخاب می‌کنیم و اولین نقطه‌ی برخورد $\mathcal{P}$ را با خطی که از این نقطه و در راستای خط l رسم می‌کنیم، پیدا می‌کنیم (شکل ۵-۱ را ببینید). این نقطه می‌تواند روی مرز بیرونی $\mathcal{P}$ و یا مرز یک حفره دیگر باشد. پاره خط متصل کننده حفره به نقطه‌ی روی $\mathcal{P}$ را به عنوان قطر برشی در نظر می‌گیریم. پیدا کردن نزدیک‌ترین نقطه‌ی حفره‌ها در زمان $O(n \log n)$ قابل انجام است. همچنین اجرای الگوریتم پرتوافکنی برای هر حفره در زمان $O(n)$ انجام می‌گیرد. بنابر این اضافه کردن همه‌ی قطرهای برشی به زمان $O(n(h + \log n))$ احتیاج دارد. چندضلعی ساده به دست آمده $O(n + ۲h)$



شکلِ ۵–۱: با اضافه کردن قطرهای برشی چندضلعی ساده $\mathcal{P}_s$ را به دست می‌آوریم.

راس خواهد داشت. از آنجا که h هم $O(n)$ است، تعداد راس‌های $\mathcal{P}_s$ هم $O(n)$ خواهد بود.

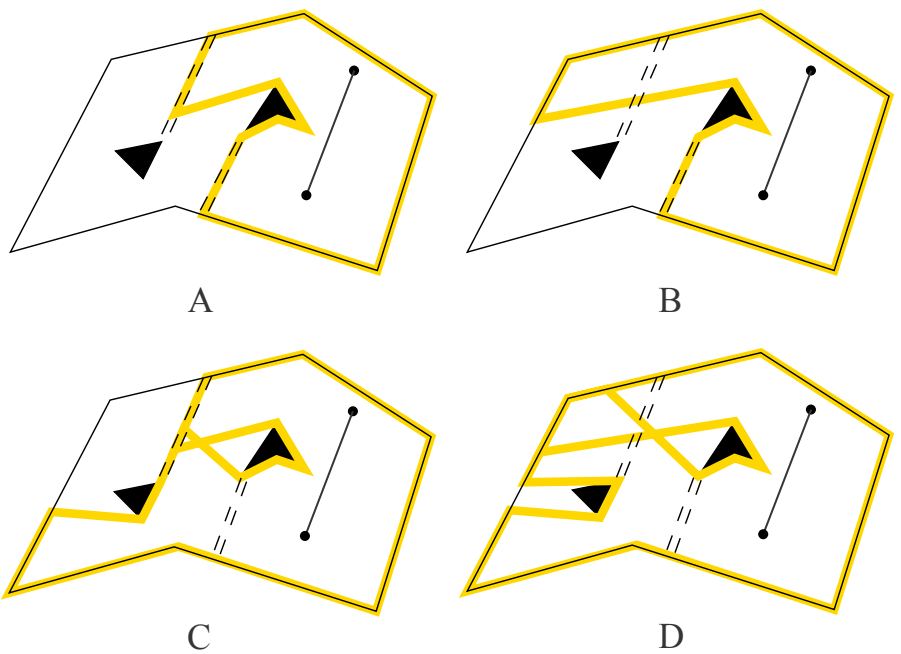
با داشتن چندضلعی ساده $\mathcal{P}_s$، می‌توانیم با استفاده از الگوریتم فصل ۳، $\mathrm{WVP}_s(pq)$ را محاسبه کنیم. سپس به این چندضلعی یال‌هایی که از طریق قطرهای برشی قابل دید هستند را اضافه می‌کنیم. مثالی از الگوریتم ما در شکل ۵–۱ قابل مشاهده است. ابتدا $\mathrm{WVP}_s(pq)$ را در $\mathcal{P}_s$ محاسبه می‌کنیم. سپس به ازای هر قطعه از قطرهای برشی که قابل دیده شدن از pq هستند، قسمت‌هایی از $\mathcal{P}$ که قابل دیده شدن از pq از طریق آن قطر برشی هستند را اضافه می‌کنیم. با در نظر گرفتن همه‌ی قطرهای برشی، به مقدار نهایی $\mathrm{WVP}(pq)$ خواهیم رسید.

۵–۱ محاسبه‌ی قابلیت دید از طریق قطرهای برشی

برای محاسبه‌ی $\mathrm{WVP}(pq)$، لازم است که یال‌هایی که از طریق قطرهای برشی دیده می‌شوند را به $\mathrm{WVP}_s(pq)$ اضافه کنیم. برای این کار، مفهوم چندضلعی قابل دید جزئی را تعریف می‌کنیم. فرض کنید که چندضلعی ساده $\mathcal{P}$ توسط یک قطر e به دو بخش $\mathcal{L}$ و $\mathcal{R}$ تقسیم شده باشد. برای یک پاره خط $pq \in \mathcal{R}$، چندضلعی قابل دید جزئی $\mathrm{WVP}_L(pq)$ را به صورت $\mathrm{WVP} \cap \mathcal{L}$ تعریف می‌کنیم. به عبارت دیگر، $\mathrm{WVP}_L(pq)$ برابر است با بخشی از $\mathcal{P}$ که توسط pq و از طریق e دیده می‌شود. برای محاسبه $\mathrm{WVP}_L(pq)$ می‌توان $\mathrm{WVP}(pq)$ را به دست آورد و سپس بخشی از آن را که در $\mathcal{L}$ قرار دارد به عنوان خروجی گزارش کرد.

لم ۱۰ با داشتن چندضلعی ساده $\mathcal{P}$ و قطر e که $\mathcal{P}$ را به دو بخش $\mathcal{L}$ و $\mathcal{R}$ تقسیم می‌کند، به ازای هر پاره‌خط $pq \in \mathcal{R}$، چندضلعی قابل دید جزئی $\mathrm{WVP}_L(pq)$ را می‌توان در زمان $O(n)$ محاسبه کرد.

لم ۱۰ تنها برای چندضلعی‌های ساده برقرار است، ولی ما از ایده آن در الگوریتم خود استفاده می‌کنیم. فرض کنید که $\mathcal{P}$ تنها یک حفره $\mathcal{H}$ داشته باشد و این حفره توسط قطر برشی u_1u_2 حذف شده



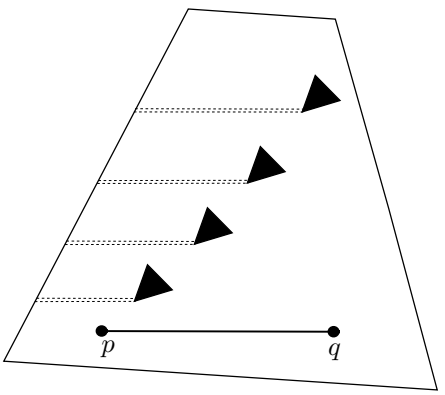
شکلِ ۵-۲: مراحل محاسبه‌ی $\mathrm{WVP}(pq)$ درون یک چندضلعی حفره‌دار.

باشد. همچنین فرض کنید که v_1v_2 قطر دیگری باشد که بر روی خط حامل u_1u_2 و در طرف دیگر $\mathcal{H}$ قرار داشته باشد، به شکلی که v_1 بر روی مرز $\mathcal{H}$ و v_2 بر روی مرز $\mathcal{P}$ باشد. مشخص است که حفره $\mathcal{H}$ را می‌توان توسط v_1v_2 نیز حذف کرد و به چندضلعی ساده دیگر $\mathcal{P}'_s$ رسید. اکنون می‌توانیم از لم ۱۰ استفاده کنیم و با اعمال آن بر روی چندضلعی ساده $\mathcal{P}'_s$، چندضلعی قابل دید جزئی دیده شده از قطر u_1u_2 محاسبه می‌شود. با استفاده از نام‌گزاری زارعی و قدسی [۴۵]، این الگوریتم را با See-Through مشخص می‌کنیم.

با اجرای الگوریتم See-Through برای هر یک از حفره‌های $\mathcal{H}_i$ و با این فرض که $\mathcal{P}$ به چندضلعی ساده تبدیل شده است، می‌توانیم این الگوریتم را به چند حفره گسترش دهیم. با این کار به h داده ساختار به اندازه‌ی $O(n)$ برای نگهداری چندضلعی‌های ساده می‌رسیم که می‌توان لم ۱۰ را برای حفره‌های $\mathcal{H}_i$ بر روی آن‌ها اجرا کرد. با استفاده از این داده ساختارها، می‌توانیم اضلاع $\mathcal{P}$ را که از طریق قطرهای برشی از pq قابل دید هستند محاسبه کنیم.

۵-۲ الگوریتم

ابتدا قطرهای برشی را اضافه می‌کنیم تا به چندضلعی ساده $\mathcal{P}_s$ برسیم. سپس، $\mathrm{WVP}_s(pq)$ را محاسبه می‌کنیم و مجموعه‌ی اظلاعی که در $\mathcal{P}_s$ توسط pq قابل دید هستند را به دست می‌آوریم. اگر قطعه‌ی e بر



شکلِ ۵-۳: یک مثال از حد بالای h'.

روی قطر برشی مربوط به حفره‌ی $\mathcal{H}$ توسط pq قابل دید باشد، از لم ۱۰ استفاده کرده و این قطعه را با چندضلعی قابل دید جزئی دیده شده از طریق e جایگزین می‌کنیم. با توجه به طبیعت مسئله قابلیت دید، این کار خاتمه پذیر است. اگر در طی این الگوریتم، h' قطعه بر روی قطرهای برشی را مورد بررسی قرار داده باشیم، در نهایت به $h'+1$ چندضلعی ساده قابل دید که اندازه هر کدام $O(n)$ است دست می‌یابیم. به راحتی می‌توان نشان داد که اجتماع این چندضلعی‌ها برار با $\mathrm{WVP}(pq)$ خواهد بود.

حال هزینه‌ی اجرای این الگوریتم را بررسی می‌کنیم. قطرهای برشی را می‌توانیم در زمان $O(nh+n\log n)$ اضافه کنیم. محاسبه‌ی قابلیت دید ضعیف در چندضلعی ساده $\mathcal{P}_s$ به زمان $O(n)$ احتیاج دارد. به علاوه، به ازای هر قطعه از قطرهای برشی که در $\mathrm{WVP}_s(pq)$ ظاهر شده است، الگوریتم لم ۱۰ را در زمان $O(n)$ اجرا می‌کنیم. با توجه به این که تعداد این قطعات h' است، نتیجه زیر را خواهیم داشت:

لم ۱۱ زمان مورد نیاز برای محاسبه‌ی $\mathrm{WVP}(pq)$ به شکل $h'+1$ چندضلعی ساده به اندازه‌ی $O(n)$ برابر $O(n(h'+1)+n\log n)$ خواهد بود، که h' تعداد قطعات قطرهای برشی است که در طی الگوریتم بر روی $\mathrm{WVP}_s(pq)$ ظاهر شده‌اند.

لم ۱۲ مقدار حد بالای h' برابر با $O(h^2)$ است.

اثبات ۸ به یاد داشته باشید که قطرهای برشی را به شکلی انتخاب کرده‌ایم که پاره‌خط پرس‌وجو با خط حامل هیچ کدام از این قطرها برخورد نداشته باشد. همچنین قطرهای برشی با یکدیگر برخورد ندارند. بنابر این اگر pq قطر برشی l را از طریق قطر برشی دیگر l' ببیند، pq نمی‌تواند l' را از طریق l ببیند. بنابر این حد بالای h' برابر با $O(h^2)$ خواهد بود. شکل ۵-۲ یک مثال برای حد بالای h' را نشان می‌دهد.

۵–۳ بهبود الگوریتم

در الگوریتم ارائه شده در بخش قبل، به ازای هر قطر برشی، ممکن است الگوریتم SEE-THROUGH($\mathcal{H}$) را تا h بار اجرا کنیم، که به این خاطر به زمان اجرای بالای $O(nh^2)$ می‌رسیم. در این بخش نشان می‌دهیم که چگونه این الگوریتم را تغییر داده و زمان به دست آوردن نتیجه نهایی را بهبود بخشیم.

یک راس v از $\mathcal{P}$ می‌تواند پاره‌خط pq را به شکل مستقیم و یا از درون قطرهای برشی ببیند. به عبارت دیگر، یک راس v می‌تواند h بخش پاره‌خط pq را از طریق قطرهای مختلف ببیند. این قطعات را می‌توانیم با توجه به پاره‌خط‌ها (یا خطوط) بحرانی که مماس بر حفره‌ها هستند و از v می‌گذرند و با pq برخورد می‌کنند دسته‌بندی کنیم. در لم بعدی، حدی برای تعداد این خطوط بحرانی به دست می‌دهیم.

لم ۱۳ تعداد خطوط بحرانی که می‌توانند pq را ببینند برابر $O(nh')$ است، که $h' = 1 + \min(h, |\mathrm{WVP}(pq)|)$ برابر با تعداد حفره‌های قابل دید از pq به اضافه ۱ می‌باشد.

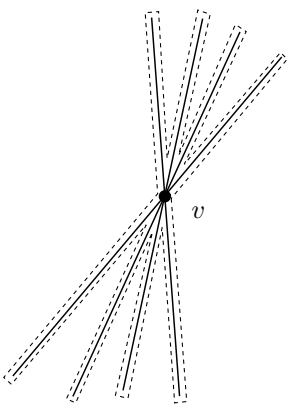
اثبات فرض کنید تعداد راس‌های حفره $\mathcal{H}_i$ برابر با m_i باشد. سه نوع خط بحرانی خواهیم داشت:

- به ازای هر راس v که بر روی مرز $\mathcal{H}_i$ نیست و نسبت به $\mathcal{H}_i$ قابل دید است، حداکثر دو خط بحرانی که با $\mathcal{H}_i$ تماس دارند و با pq برخورد می‌کنند وجود دارد. بنابراین تعداد کل این خطهای بحرانی برابر با $O(h'')$ است، که $h'' = \min(h, |\mathrm{WVP}(pq)|)$.
- تعداد خطوط بحرانی که توسط دو راس از H_i مشخص می‌شوند و با pq برخورد می‌کنند برابر با $O(m_i)$ است. همچنین می‌دانیم که $\sum_i m_i = O(n)$.
- تعداد خطوط بحرانی که با pq برخورد می‌کنند و با هیچ کدام از حفره‌ها برخورد نمی‌کنند برابر با $O(n)$ است [۶].

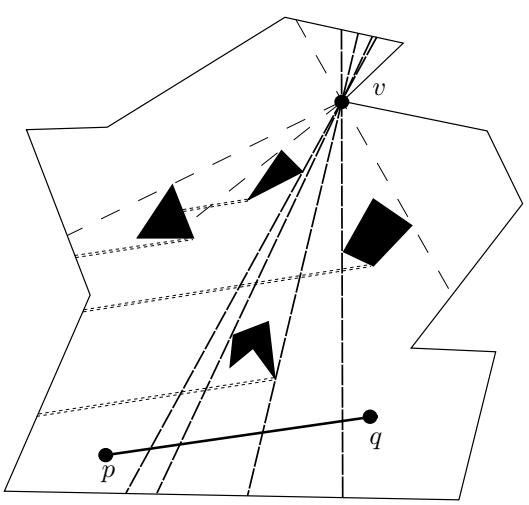
با قرار دادن این مشاهدات در کنار هم، لم اثبات می‌شود.

با پیش‌پردازش چندضلعی $\mathcal{P}$ می‌توانیم خطوط بحرانی که با پاره‌خط pq برخورد می‌کنند را در زمان پرس‌وجو به شکل بهینه پیدا کنیم. تعداد خطوط بحرانی که از هر راس چندضلعی $\mathcal{P}$ عبور می‌کند $O(n)$ است. بنابر این مجموعه‌ی همه‌ی خطوط بحرانی را می‌توان در زمان $O(n^2 \log n)$ و حافظه ی $O(n^2)$ محاسبه کرد. با توجه به این که مجموعه‌ی خطوط بحرانی که از یک راس می‌گذرند را می‌توان به شکل یک چندضلعی ساده تصور کرد (شکل ۵–۴ را ببینید)، می‌توانیم برای هر راس داده ساختار پرتو افکنی را در زمان و حافظه‌ی $O(n)$ تشکیل دهیم، به شکلی که پرس‌وجوهای پرتو افکنی را در زمان $O(\log n)$ پاسخ دهیم. در زمان پرس‌وجو، خطوط بحرانی هر راس که با pq برخورد می‌کنند را در زمان $O(c_v \log c_v)$ به دست می‌آوریم. در اینجا c_v برابر با تعداد خطوط بحرانی است که از v عبور کرده و با pq برخورد می‌کنند. این کار به ازای همه‌ی راس‌ها در زمان $O(nh' \log n)$ قابل انجام است.

با یک پیماش زاویه‌ای بر روی این خطوط، می‌توان بخش‌های قابل دید pq و قطرهای برشی قابل دید از راس‌ها را در زمان $O(nh')$ به دست آورد. اطلاعات مربوط به این بخش‌ها را در هر راس و با توجه به قطرهای قابل دید هر بخش ذخیره می‌کنیم. انجام این عمل برای همه‌ی راس‌های $\mathcal{P}$ که شامل راس‌های



شکلِ ۵–۴: می‌توانیم با پاره‌خط‌هایی که از یک نقطه می‌گذرند مانند یک چندضلعی ساده برخورد کنیم (پاره‌خط‌های نقطه‌چین) و یک داده‌ساختار پرتوافکنی را در زمان $O(n)$ تشکیل دهیم.



شکلِ ۵–۵: از هر راس چندضلعی می‌توان $O(h)$ خط بحرانی در نظر گرفت که با یک قطر برشی و pq برخورد می‌کنند.

روی حفره‌ها نیز می‌باشد، و ذخیره بخش‌های قابل دید pq در هر راس را می‌توان در زمان $O(nh' \log n)$ به اتمام رساند. بنابر این لم زیر را خواهیم داشت.

لم ۱۴ یک چندضلعی ساده $\mathcal{P}$ با h حفره و تعداد راس کلی n را می‌توان در زمان $O(n^2 \log n)$ پیش‌پردازش کرد و یک داده‌ساختار به اندازه‌ی $O(n^2)$ به دست آورد به شکلی که با استفاده از آن، به ازای هر پاره‌خط پرس‌وجوی pq، می‌توان خطوط بحرانی که با pq برخورد می‌کنند را در زمان $O(nh' \log n)$ محاسبه و مرتب‌سازی کرد، که در آن $h' = 1 + \min(h, |\mathrm{WVP}(pq)|)$.

در ادامه این بخش نشان می‌دهیم که این خطوط بحرانی یک آرایش[1] تشکیل می‌دهند که با استفاده از این آرایش، می‌توان $\mathrm{WVP}(pq)$ را محاسبه کرد.

$\mathrm{WVP}_s(pq)$ را به عنوان بخشی از $\mathcal{P}_s$ که به شکل مستقیم توسط pq دیده می‌شود تعریف کردیم. فرض کنید که c_i قطر برشی حفره‌ی H_i باشد. WVP_{c_i} را به عنوان بخشی از $\mathcal{P}$ که توسط pq و از درون c_i دیده می‌شود تعریف می‌کنیم. مشخص است که $\mathrm{WVP}(pq) = \cup_i \mathrm{WVP}_{c_i}(pq) \cup \mathrm{WVP}_s(pq)$.

اکنون نشان می‌دهیم که چگونه می‌توان WVP_{c_i} را محاسبه کرد. دقت کنید که WVP_{c_i} در نیم‌صفحه بالای c_i قرار دارد. فرض کنید که $\mathcal{P}_{c_i}$ برابر با بخشی از $\mathcal{P}_s$ که بالای c_i قرار دارد باشد. با توجه به این که pq می‌تواند $\mathcal{P}_{c_i}$ را از طریق پاره‌های مختلفی از c_i ببیند، ممکن است WVP_{c_i} چندضلعی ساده نباشد.

فرض کنید D_i برابر با مجموعه‌ی خطوط بحرانی شروع شده از راس‌های $\mathcal{P}_{c_i}$ که با pq برخورد کرده و مستقیما از c_i عبور می‌کنند، به علاوه خطوط بحرانی که با pq برخورد می‌کنند و c_i را قطع کرده و بلافاصله بعد از آن با مرز $\mathcal{P}_{c_i}$ برخورد می‌کنند باشد. هر خط بحرانی با یک یا دو راس رفلکس مشخص می‌شود. به این راس‌ها، لنگر[2] خط بحرانی می‌گوییم. علاوه بر این، هر خط بحرانی مرز $\mathcal{P}_{c_i}$ را حداکثر دو بار قطع می‌کند. فرض کنید مجموعه‌ی قطعات روی مرز $\mathcal{P}_{c_i}$ که از این برخوردهای خطوط بحرانی ایجاد می‌شود برابر S_i باشد. مشخص است که $|S_i| = O(n + 2h) = O(n)$.

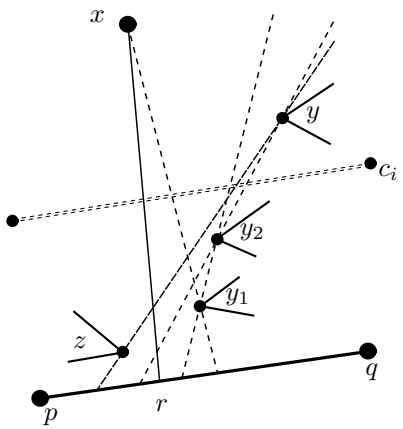
فرض کنید که $L_i = \cup_{k=1 \ldots i}(S_k \cup D_k)$ و $\mathcal{A}_i$ برابر با آرایش به دست آمده از قطعات L_i باشد. نشان می‌دهیم که $\mathcal{A}_i$ چندضلعی $\mathcal{P}_{c_i}$ را به بخش‌های قابل دید و غیرقابل دید تقسیم می‌کند.

لم ۱۵ به ازای هر نقطه‌ی $x \in \mathcal{P}_{c_i}$ که توسط pq قابل دید است، یک پاره‌خط e از L_i وجود دارد که می‌توان آن را حول لنگر آن چرخاند تا زمانی که با x برخورد کند، به شکلی در حین این گردش نسبت به pq قابل دید باقی بماند.

اثبات ۱۰ از آنجایی که x نسبت به pq قابل دید است، x باید یک نقطه‌ی r را در پاره‌خط pq ببینند، به شکلی که xr و c_i با هم برخورد کنند (شکل ۵-۶ را ببینید). پاره‌خط xr را حول x به شکل پادساعتگرد می‌گردانیم تا زمانی که با یک راس $y_1 \in \mathcal{P}$ برخورد کند. توجه کنید که حالت $y_1 = q$ هم قابل رخ دادن است و احتیاج به بررسی جداگانه ندارد. سپس پاره‌خط را به شکل ساعتگرد حول y_1 می‌گردانیم تا زمانی که به راس دیگر $y_2 \in \mathcal{P}$ برخورد کند. این روند را ادامه می‌دهیم تا زمانی که پاره‌خط به یکی از نقاط

[1] Arrangement

[2] Anchor



شکلِ ۵–۶: به ازای هر نقطه‌ی قابل دید $x \in \mathcal{P}_{c_i}$، یک خط بحرانی yz وجود دارد که می‌توان آن را حول y چرخاند تا زمانی که با x برخورد کند.

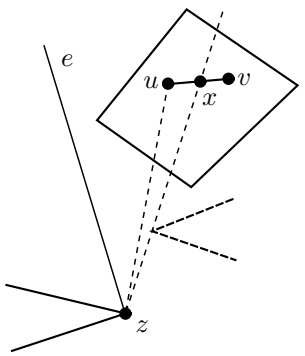
انتهایی c_i برسد، و یا بخش پایینی پاره‌خط با یک نقطه‌ی z از چندضلعی برخورد کند، و یا پاره‌خط به نقطه‌ی انتهایی p برسد. فرض کنید که y آخرین نقطه‌ای باشد که پاره‌خط در بخش بالایی e با آن برخورد می‌کند. از آنجایی که ما پاره‌خط را تنها به شکل ساعتگرد چرخانده‌ایم، این روند حتما پایان‌پذیر خواهد بود. مشخص است که yz یک خط بحرانی در L_i خواهد بود، و ما می‌توانیم با گردش yz حول z به شکل پادساعتگرد به نقطه‌ی x برسیم.

لم ۱۶ همه‌ی نقاط درون یک سلول c از $\mathcal{A}_i$ دارای وضعیت قابلیت دید یکسانی نسبت به pq هستند.

اثبات ۱۱ فرض کنید که نقاط u و v درون سلول c هستند، و u نسبت به pq قابل دید است و v نسبت به pq غیر قابل دید. فرض کنید uv پاره‌خط متصل کننده u و v باشد و x نزدیک‌ترین نقطه روی uv به u باشد که نسبت به pq غیر قابل دید است (شکل ۵–۷را ببینید). بر اساس لم ۱۵، یک پاره‌خط $e \in L_i$ با لنگر z وجود دارد، به شکلی که اگر e را حول z بچرخانیم با u برخورد خواهد کرد. بعد از برخورد با u، به چرخش e ادامه می‌دهیم تا با نقطه‌ی x برخورد کند. با توجه به این که x نسبت به pq غیر قابل دید است، zx باید یک خط بحرانی باشد. این به این معنی است که ما یک خط بحرانی دیگر به مرکز یک راس $z \in \mathcal{P}$ داریم که با pq برخورد دارد و از درون سلول c می‌گذرد. بنابر این فرض این که c یک سلول درون $\mathcal{A}_i$ است نقض می‌گردد.

برای محاسبه‌ی نهایی $\mathrm{WVP}(pq)$ باید $\cup_i \mathrm{WVP}_{c_i}(pq) \cup \mathrm{WVP}_s(pq)$ را محاسبه کنیم. $\mathrm{WVP}_s(pq)$ یک چندضلعی ساده با اندازه $O(n)$ است که می‌توان آن را با $O(n)$ پاره‌خط نشان داد. همچنین $\mathrm{WVP}_{c_i}(pq)$ را می‌توان با آرایشی از $O(n + 2h + d_i)$ پاره‌خط نمایش داد، که $d_i = |D_i|$. مشخص است که $\sum_i d_i = O(nh')$. بنابر این، $\mathrm{WVP}(pq)$ را می‌توان با آرایشی از $O(\sum_{i=1\ldots h'} n + \sum_{i=1\ldots h} d_i) = O(nh')$ پاره‌خط نمایش داد.

در بخش بعد به مسئله‌ی محاسبه‌ی مرز $\mathrm{WVP}(pq)$ می‌پردازیم.



شکلِ ۵–۷: همه‌ی نقاط داخل یک سلول وضعیت قابلیت دید یکسانی نسبت به pq دارند.

۵–۴ محاسبه‌ی مرز $\mathrm{WVP}(pq)$

نشان دادیم که $\mathrm{WVP}(pq)$ را به صورت آرایشی از $O(nh')$ پاره‌خط به دست آوریم. در این بخش نشان می‌دهیم که $\mathrm{WVP}(pq)$ را می‌توان در زمان $O(nh' \log n + |\mathrm{WVP}(pq)|)$ به صورت یک چندضلعی محاسبه و گزارش کرد.

Balaban [۴] نشان داده است که با استفاده از یک داده‌ساختار به اندازه‌ی $O(m)$، می‌توان نقاط برخورد m پاره‌خط را در زمان $O(m \log m + k)$ محاسبه و گزارش داد، که k تعداد این برخوردها می‌باشد. با توجه به این که حداقل به زمان $\Omega(k)$ برای گزارش نقاط نیاز است، این الگوریتم بهینه می‌باشد. در مسئله‌ی ما، تعداد $O(nh')$ پاره خط داریم و گزارش همه‌ی نقاط برخورد به زمان $O(nh' \log n + k)$ و حافظه‌ی $O(nh')$ نیاز دارد. با استفاده از روش Margalit و Knott [۳۴]، می‌توانیم با صرف همین هزینه‌ی زمانی قطعات به دست آمده را در حین گزارش آن‌ها دسته‌بندی کنیم. در نتیجه به قضیه نهایی زیر می‌رسیم.

قضیه ۵ با پیش‌پردازش چندضلعی حفره‌دار $\mathcal{P}$ با h حفره جدا از هم و n رأس در زمان $O(n^2 \log n)$ و ایجاد ساختاری به اندازه‌ی $O(n^2)$، می‌توانیم چندضلعی قابل دید ضعیف $\mathrm{WVP}(pq)$ را در زمان $O(nh' \log n + k)$ و حافظه‌ی $O(nh')$ گزارش کنیم، که در آن $h' = 1 + \min(h, k)$ و k اندازه‌ی خروجی و برابر با $O(n^2 h^2)$ می‌باشد.

۵–۵ خلاصه

در این فصل به بررسی مسئله محاسبه‌ی چندضلعی قابل دید ضعیف یک پاره‌خط درون چندضلعی‌های حفره‌دار پرداختیم. نشان دادیم که چگونه می‌توان $\mathrm{WVP}(pq)$ را در چندضلعی با n راس و h حفره به دست آورد. الگوریتم ارائه شده به پیش‌پردازش چندضلعی در زمان $O(n^2 \log n)$ و ایجاد داده‌ساختاری به اندازه‌ی $O(n^2)$ می‌پردازد. با دریافت پاره‌خط پرس‌وجوی pq می‌توان در زمان $O(nh' \log n + k)$ جواب را محاسبه و گزارش کرد. در اینجا h' یک پارامتر حساس به خروجی و دارای حداکثر مقدار

$1 + \min(h, k)$، و $k = O(n^۲h^۲)$ برابر با اندازه‌ی خروجی است.

فصلِ ۶

اندازه‌ی چندضلعی قابل دید ضعیف

در این فصل به بیان نتایج به دست آمده در مسئله‌ی شمارش قابلیت دید ضعیف یک پاره‌خط در چندضلعی‌های ساده می‌پردازیم. شمارش قابلیت دید ضعیف یک پاره‌خط (به اختصار WVC) را به عنوان اندازه‌ی چندضلعی قابل دید ضعیف پاره‌خط تعریف می‌کنیم.

ما به دو شکل شمارش دقیق و شمارش تقریبی به حل این مسئله می‌پردازیم. در بخش اول، دو الگوریتم شمارش دقیق را که پس از پیش‌پردازش چندضلعی داده شده، جواب را در زمان زیرخطی به دست می‌دهند، ارائه می‌کنیم. در بخش دوم، نشان می‌دهیم که چگونه با تقریب زدن جواب نهایی، هزینه‌ی پیش‌پردازش را کاهش دهیم. خلاصه‌ای از نتایج به دست آمده در جدول ۶–۱ دیده می‌شود. این پژوهش با مشارکت آقای شروین دانش‌پژوه و خانم شراره علیپور انجام شده است. الگوریتم‌های ارائه شده در قالب یک مقاله جمع‌بندی شده و برای ژورنال ارسال شده است.

M. Nouri Bygi, S. Daneshpajouh, S Alipour, M. Ghodsi. Weak Visibility Counting in Simple Polygons

۶–۱ مقدمه

در فصل‌های قبل به بررسی قابلیت دید ضعیف پاره‌خط در چندضلعی‌های ساده و حفره‌دار پرداختیم. در بسیاری از مواقع، داشتن اطلاعات مختصری از این قابلیت دید کمک زیادی در پیدا کردن راه درست برخورد با مسئله می‌کند. مسئله‌ی شمارش قابلیت دید (VCP) در یک چندضلعی عبارت است از پیدا کردن تعداد اشیاء (ضلع‌ها، راس‌ها و غیره) که از یک نقطه‌ی p قابل دید هستند. یک راه حل ناکارامد این است که چندضلعی قابل دید نقطه را محاسبه‌ی کرده و اندازه‌ی خروجی را به دست آوریم.

برای یک چندضلعی ساده، Bose و دیگران نشان دادند که با پیش‌پردازش چندضلعی در زمان $O(n^۳ \log n)$ و ایجاد داده‌ساختاری به اندازه‌ی $O(n^۳)$، می‌توان برای یک نقطه‌ی داده شده، VCP را در زمان $O(\log n)$ محاسبه کرد [۶]. برای یک مجموعه از n پاره‌خط جدا از هم، Suri و O'Rourke [۴۲] با نمایش چندضلعی قابل دید پاره‌خط‌ها به شکل مجموعه‌ای از مثلث‌ها، اولین الگوریتم ۳-تقریب را برای

	زمان پیش‌پردازش	حافظه	زمان پرس‌وجو
ابتدایی	–	$O(n)$	$O(n)$
دقیق	$O(n^{۷})$	$O(n^{۶})$	$O(\log n)$
دقیق	$O(n^{۴+\epsilon})$	$O(n^{۴+\epsilon})$	$O(\log^{۵} n)$
دقیق	$O(n^{۳+\epsilon})$	$O(n^{۳})$	$O(n^{۱/۲+\epsilon})$
تقریبی	$O(n^{۲})$	$O(n^{۲})$	$O(n^{۱/۲+\epsilon})$

جدولِ ۶-۱: خلاصه‌ای از نتایج به دست آمده در مسئله‌ی شمارش قابلیت دید ضعیف. بخش اول الگوریتم‌های دقیق به دست آمده را نشان می‌دهد و بخش دوم نتیجه‌ی الگوریتم تقریبی را بیان می‌کند.

VCP ارائه کردند. Gudmundsson و Morin [۲۵] این نتیجه را با استفاده از یک روش پوشش بهبود یافته، به ۲-تقریب بهبود دادند. نتیجه‌ی مشابهی توسط علی‌پور و زارعی [۱] و همچنین نوری و قدسی [۳۵] به دست آمده است. علی‌پور و زارعی نشان دادند که تعداد نقاط انتهایی پاره‌خط‌های قابل دید یک ۲-تقریب از VCP می‌باشد.

مسئله‌ی شمارش قابلیت دید را می‌توان برای انواع ناظرهای دیگر هم گسترش داد. در این فصل ما مسئله‌ی شمارش قابلیت دید یک پاره‌خط را درون چندضلعی ساده بررسی می‌کنیم.

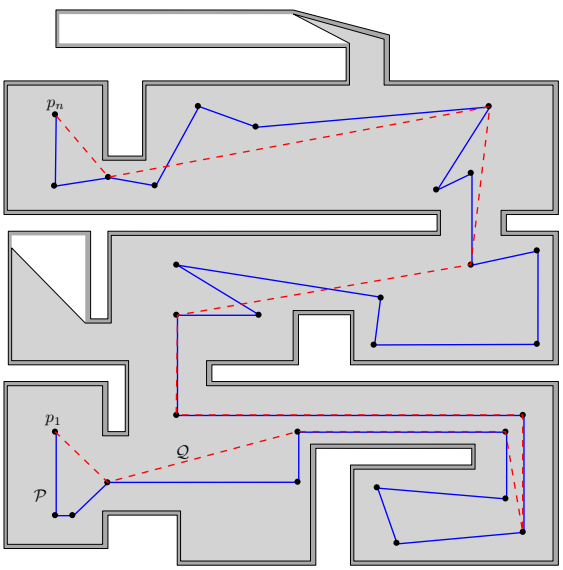
۶-۲ کاربردها و انگیزه‌ها

در بسیاری اوقات، احتیاجی به محاسبه‌ی همه‌ی چندضلعی قابل دید پاره‌خط نیست، بلکه دانستن اندازه‌ی چندضلعی قابل دید به ما در پیشبرد مسئله کمک می‌کند. به عنوان مثال در الگوریتم‌های حذف سطح پنهان[1]، به دست آوردن معیاری برای حذف بخش‌های نامهم صحنه از اهمیت زیادی برخوردار است [۳۳]. با محاسبه‌ی اندازه‌ی چندضلعی قابل دید، می‌توانیم بدون محاسبه‌ی همه‌ی ناحیه‌ی قابل دید، اطلاعات مهمی از نحوه‌ی برخورد با بخش‌های مختلف صحنه به دست آوریم.

یکی دیگر از کاربردهای جالب توجه برای مسئله‌ی اندازه‌ی چندضلعی قابل دید، مسئله‌ی ساده‌سازی مسیر تحت معیار قابلیت دید است. نقشه‌ی یک ساختمان و یک ربات نگهبان را در درون ساختمان در نظر بگیرید. به این ربات یک مسیر چندضلعی داده شده است و ربات وظیفه دارد بر روی این مسیر حرکت کند و در حین حرکت وجود هر گونه شیء ناشناس را گزارش کند. قابلیت دید ربات را ۳۶۰ درجه در نظر می‌گیریم. در این مسئله‌ی ساده‌سازی ما می‌خواهیم مسیر کوتاه‌تر و کمینه‌ای را پیدا کنیم بطوری که قابلیت دید ربات تغییر نکند، و یا تغییر آن تحت مقدار تعیین شده‌ی ϵ باشد. به عبارت دیگر، می‌خواهیم که در زمان عبور از مسیر ساده شده، بخش زیادی از قابلیت دید ربات را از دست ندهیم.

اکنون مسئله را به شکل رسمی‌تر بیان می‌کنیم. فرض کنید مسیر $P = \{p_۱, p_۲, \ldots, p_m\}$ در چندضلعی $\mathcal{R}$ به اندازه‌ی n داده شده است. یک ساده‌سازی Q از P عبارت است از مسیر

[1] Hidden Surface Removal



شکلِ ۶-۱: مسیر چندضلعی P (خطوط آبی رنگ) درون یک چندضلعی ساده، و یک ساده‌سازی Q از آن (خطوط خط‌چین). نواحی خاکستری، نواحی قابل دید از P و Q هستند. در این مثال خطای ساده‌سازی برابر صفر است.

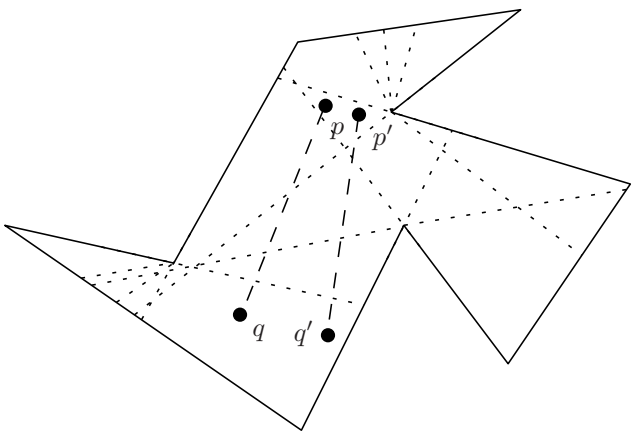
$\{q_1 = p_1, q_2, \ldots, q_k = p_m\}$ که در آن نقاط Q زیرمجموعه‌ای از نقاط P هستند. فرض کنید $\mathrm{Er}(p_ip_j)$ برابر با خطای میانبر p_ip_j که دو نقطه از P را به هم متصل می‌کند باشد. همچنین فرض کنید $P(i,j)$ زیرمسیر از p_i به p_j باشد. برای یک ساده‌سازی Q از P شکل ۶-۱ را ببینید.

خطای ساده‌سازی را به صورت تفاوت اندازه‌های چندضلعی‌های قابل دید ضعیف p_ip_j و $P(i,j)$ تعریف می‌کنیم. در اینجا اندازه‌ی چندضلعی قابل دید به معنای تعداد اضلاع آن می‌باشد. همچنین دقت کنید که یک راس/ضلع توسط یک مسیر دیده می‌شود اگر توسط یک نقطه روی آن دیده شود.

برای یک چندضلعی داده شده‌ی $\mathcal{R}$ و یک مسیر P درون $\mathcal{R}$، کوتاه‌ترین اندازه‌ی ساده‌سازی را می‌توان با در نظر گرفتن همه‌ی میانبرها به دست آورد. با توجه به این که تعداد میانبرها $O(m^2)$ است، با استفاده از الگوریتم بدیهی محاسبه‌ی اندازه‌ی چندضلعی قابل دید ضعیف در زمان $O(n)$، به زمان $O(nm^2)$ برای محاسبه‌ی اندازه‌ی چندضلعی قابل دید میانبرها نیاز داریم. اما با استفاده از الگوریتم‌های ارائه شده در این فصل، می‌توان زمان اجرای این الگوریتم را به شکل خوبی بهبود داد.

۶-۳ اندازه‌ی دقیق قابلیت دید ضعیف

راه حل ابتدایی برای مسئله‌ی شمارش قابلیت دید ضعیف به این شکل خواهد بود که چندضلعی قابل دید ضعیف پاره‌خط داده شده را محاسبه کنیم و سپس اندازه‌ی آن را حساب کرده و گزارش کنیم. با توجه به



شکلِ ۶-۲: قابلیت دید ضعیف یک پاره‌خط با ناحیه‌های قابلیت دید دو سر آن مشخص می‌گردد.

این که محاسبه‌ی چندضلعی قابل دید ضعیف به زمان $O(n)$ احتیاج دارد، این روش جواب مسئله را در زمان $O(n)$ به دست می‌دهد.

در این بخش نشان می‌دهیم که چگونه چندضلعی ساده $\mathcal{P}$ را پیش‌پردازش کنیم به شکلی که با دادن یک پاره‌خط pq در زمان پرس‌وجو، بتوانیم اندازه‌ی چندضلعی قابل دید ضعیف آن را در زمان زیرخطی به دست آوریم. این روش‌ها بر اساس تجزیه‌ی قابلیت دید چندضلعی هستند. برای آشنایی با تجزیه‌ی قابلیت دید به بخش ۲-۲ مراجعه کنید.

در ابتدا نشان می‌دهیم که کافی است که ناحیه‌های قابلیت دید دو سر انتهایی پاره‌خط را در نظر بگیریم. به عبارت دیگر، با دانستن این دو ناحیه، ساختار چندضلعی قابل دید ضعیف و اندازه‌ی آن مشخص می‌گردد.

لم ۱۷ فرض کنید که نقاط p و p' در ناحیه‌ی قابلیت دید R، و نقاط q و q' در ناحیه‌ی قابلیت دید R' قرار داشته باشند. خواهیم داشت: $\mathrm{WVC}(pq) = \mathrm{WVC}(p'q')$.

اثبات ۱۲ از آنجایی که p و p' در ناحیه‌ی قابلیت دید یکسانی هستند، دنباله‌ی یکسانی از راس‌ها و یال‌های $\mathcal{P}$ را مشاهده می‌کنند. بنابر این می‌دانیم که $\mathrm{SPT}(p) = \mathrm{SPT}(p')$ (شکل ۶-۲ را ببینید). در اینجا $\mathrm{SPT}(p)$ درخت کوتاه‌ترین مسیر در $\mathcal{P}$ و به ریشه p می‌باشد. وضعیت مشابهی نیز برای q و q' برقرار می‌باشد. می‌دانیم که الگوریتم *Guibas* و دیگران [۲۷] برای محاسبه‌ی چندضلعی قابل دید ضعیف پاره خط pq درون چندضلعی $\mathcal{P}$ بر اساس $\mathrm{SPT}(p)$ و $\mathrm{SPT}(q)$ می‌باشد (بخش ۳-۱ را ببینید). بنابر این نتیجه می‌گیریم که این الگوریتم برای دو پاره‌خط pq و $p'q'$ به شکل یکسانی اجرا شده و نتیجه‌ی یکسانی به عنوان $\mathrm{WVP}(pq)$ و $\mathrm{WVP}(p'q')$ به دست می‌دهد.

بر اساس لم ۱۷، تنها بر روی ناحیه‌ی قابلیت دید نقاط انتهایی پاره خط تمرکز می‌کنیم. در ادامه این بخش دو الگوریتم برای محاسبه‌ی دقیق اندازه‌ی WVC ارائه می‌دهیم که در هزینه‌های پیش‌پردازش و پرس‌وجو با هم متفاوت هستند.

۶-۳-۱ الگوریتم با بهترین زمان پرس‌وجو

تجزیه‌ی قابلیت دید $\mathcal{P}$ و دو ناحیه‌ی S_i و S_j درون این تجزیه را در نظر بگیرید. می‌گوییم که این دو ناحیه نسبت به هم قابل دید هستند، اگر نقطه‌های $s_i \in S_i$ و $s_j \in S_j$ وجود داشته باشند به طوری که s_i و s_j نسبت به هم قابل دید باشند. برای دو ناحیه‌ی قابلیت دید داده شده، در زمان $O(n)$ می‌توان بررسی کرد که آیا نسبت به هم قابل دید هستند، و در صورت قابل دید بودن، یک زوج (s_i, s_j) را پیدا کرد.

با توجه به این که چندضلعی قابل دید ضعیف توسط ناحیه‌های قابلیت دید دو سر پاره خط پرس‌وجو مشخص می‌گردد، ایده‌ی این الگوریتم به این شکل است که $\mathrm{WVP}(s_i, s_j)$ را به ازای هر زوج ناحیه‌های (S_i, S_j) در زمان $O(n)$ محاسبه می‌کنیم و اندازه‌ی چندضلعی به دست آمده را ذخیره می‌کنیم. همچنین برای پیدا کردن ناحیه‌های قابلیت دید دو سر پاره‌خط، یک داده ساختار نقطه‌یابی بر روی تجزیه‌ی قابلیت دید می‌سازیم.

با توجه به این که $O(n^۶)$ زوج ناحیه وجود دارد، به زمان $O(n^۷)$ و حافظه‌ی $O(n^۶)$ برای محاسبه و نگهداری همه‌ی مقادیر ممکن نیاز داریم. در زمان پرس‌وجو و با دریافت پاره‌خط pq، ناحیه‌های شامل p و q را پیدا می‌کنیم و اندازه‌ی چندضلعی قابلیت دید متناظر با آن‌ها را در زمان $O(\log n)$ گزارش می‌کنیم.

قضیه ۶ با پیش‌پردازش یک چندضلعی ساده در زمان $O(n^۷)$ و با استفاده از حافظه‌ی $O(n^۶)$، می‌توانیم اندازه‌ی چندضلعی قابلیت دید یک پاره‌خط پرس‌وجو را در زمان $O(\log n)$ به دست آوریم.

۶-۳-۲ کاهش هزینه‌های پیش‌پردازش

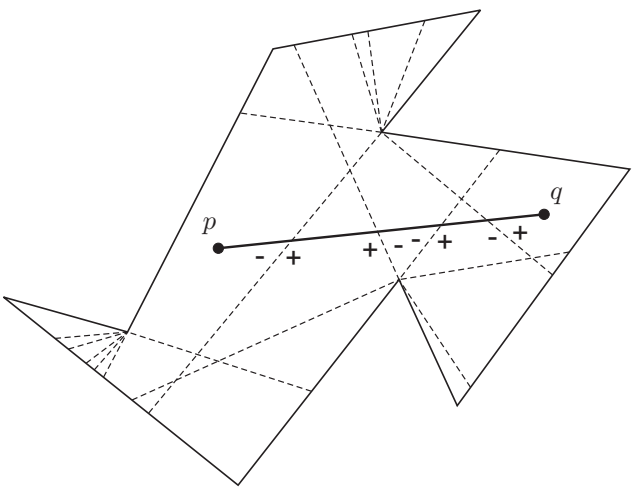
هرچند الگوریتم قضیه ۶ خروجی مورد نظر را به شکل بهینه به دست می‌دهد، هزینه‌ی بالای پیش‌پردازش آن سبب می‌شود که به دنبال الگوریتم‌های بهتری برای این مسئله باشیم. در این بخش نشان می‌دهیم که چگونه از داده‌ساختار جستجوی بازه‌ای چند لایه استفاده کنیم و هزینه‌های پیش‌پردازش را، در ازای افزایش زمان پرس‌وجو، کاهش دهیم.

برای محاسبه‌ی اندازه‌ی چندضلعی قابل دید ضعیف یک پاره‌خط pq، از نقطه‌ی انتهایی p شروع کرده و اندازه‌ی قابلیت دید p را محاسبه می‌کنیم. سپس به سمت q حرکت می‌کنیم و در حین حرکت، مقدار اندازه‌ی قابلیت دید پاره‌خط را تا نقطه‌ای که پیش آمده‌ایم به‌روز می‌کنیم. وقتی به نقطه‌ی انتهایی q می‌رسیم، مقدار نهایی اندازه را به عنوان جواب مسئله گزارش می‌کنیم.

اولین مسئله‌ای که باید به آن بپردازیم، محاسبه‌ی اندازه‌ی اولیه‌ی قابلیت دید نقطه‌ی p است. برای این منظور از لم زیر استفاده می‌کنیم.

لم ۱۸ [۶] یک چندضلعی ساده $\mathcal{P}$ را می‌توان در زمان $O(n^۳ \log n)$ و حافظه‌ی $O(n^۳)$ پیش‌پردازش کرد، به نحوی که با دادن یک نقطه‌ی دلخواه درون چندضلعی، اندازه‌ی قابلیت دید آن نقطه را در زمان $O(\log n)$ به دست آورد.

وقتی که از p به سمت q در حال حرکت هستیم، ممکن است با تعدادی از خطوط بحرانی چندضلعی برخورد کنیم. عبور از یک خط بحرانی به معنای اضافه شدن یک راس به راس‌های قابل دید در



شکلِ ۶–۳: مسئله‌ی اندازه‌ی چندضلعی قابل دید ضعیف را می‌توان با جمع قابلیت دید نقطه‌ی شروع p و تعداد برخوردهای مثبت از p به q به دست آورد.

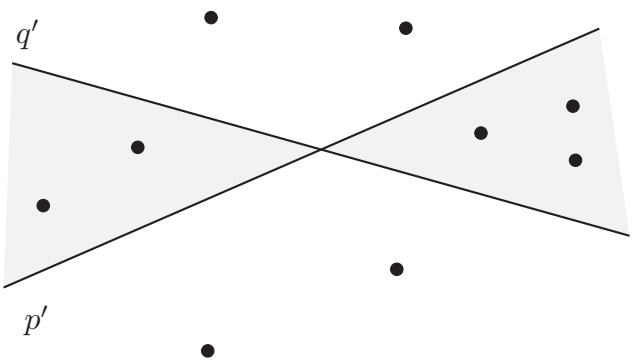
یک جهت و کاهش این راس در جهت عکس می‌باشد. با توجه به این افزایش و یا کاهش قابلیت دید، دو طرف یک خط بحرانی را با نام‌های + و - علامت‌گذاری می‌کنیم (شکل ۶–۳ را ببینید). اگر با یک خط بحرانی از سمت + آن برخورد کنیم، به آن یک برخورد مثبت می‌گوییم و در غیر این صورت یک برخورد منفی. با توجه به این که غیر قابل دید شدن یک راس که قبلا دیده می‌شده است تاثیری در اندازه‌ی قابلیت دید ضعیف ندارد، تنها کافی است که برخورهای مثبت را در نظر بگیریم. به عبارت دقیق‌تر، هر برخورد مثبت معادل یک مقدار اضافه در اندازه‌ی قابلیت دید pq است.

لم ۱۹ [۶] پاره‌خط ab درون چندضلعی ساده $\mathcal{P}$ را در نظر بگیرید. اگر یک نقطه‌ی $z \in \mathcal{P}$ دو سر انتهایی a و b را ببیند، آنگاه z همه‌ی نقاط روی پاره‌خط ab را خواهد دید.

لم ۲۰ فرض کنید که پاره‌خط pq بتواند یک راس v از چندضلعی را ببیند. همچنین فرض کنید X مجموعه‌ی نقاط روی pq باشد که v را می‌بینند. در این صورت X یک زیرپاره‌خط ab از pq خواهد بود. علاوه بر این a یا برابر با p است و یا نقطه‌ی برخورد pq با یک خط بحرانی در یک برخورد مثبت (در جهت p به q) است. همچنین b برابر با q و یا نقطه‌ی برخورد pq با یک خط بحرانی در یک برخورد منفی (در جهت p به q) است.

اثبات ۱۳ این مسئله را می‌توان با استفاده از لم ۱۹ و این واقعیت که هر خط بحرانی معادل با یک تغییر در قابلیت دید دو طرف آن است ثابت کرد.

بر اساس لم ۲۰، برای محاسبه‌ی اندازه‌ی قابلیت ضعیف پاره‌خط pq، می‌توانیم مقدار اولیه‌ی اندازه‌ی قابلیت دید p را به تعداد برخوردهای مثبت اضافه کنیم. مقدار اولیه‌ی قابلیت دید p را می‌توانیم توسط لم ۱۸ به دست آوریم. بنابر این کافی است که تعداد خطوط بحرانی که با pq برخورد می‌کنند و p در سمت



شکلِ ۶-۴: در فضای دوگان خطوطی که با pq برخورد می‌کنند به نقاطی داخل گوه‌ی دوگانه‌ی $p'q'$ تبدیل می‌شوند. این‌ها نقاطی هستند که زیر p' و بالای q'، و یا بالای p' و زیر q' قرار گرفته‌اند.

مثبت آن‌ها قرار دارد را حساب کنیم. در ادامه نشان می‌دهیم که این مسئله را می‌توان با استفاده از زنجیره‌ای از جستجوی بازه‌ای نیم‌صفحه‌ای انجام داد.

فرض کنید که هر پاره‌خط بحرانی توسط دو انتهای آن s_{right} و s_{left} مشخص شود. همچنین فرض کنید که l خط حامل پاره‌خط پرس‌وجوی pq باشد. ابتدا پاره‌خط‌های بحرانی که با l برخورد می‌کنند را پیدا می‌کنیم. برای این کار از جستجوی بازه‌ای نیم‌صفحه‌ای استفاده می‌کنیم تا پاره‌خط‌های بحرانی که انتهای سمت چپشان بالای l قرار دارد را پیدا کنیم. سپس در مجموعه‌ی جواب به دست آمده، پاره‌خط‌هایی را انتخاب می‌کنیم که انتهای سمت راستشان پایین l قرار داشته باشد. این کار را با اجرای دوباره جستجوی بازه‌ای نیم‌صفحه‌ای بر روی نتایج جستجوی اول امکان پذیر می‌باشد. در مرحله‌ی سوم، مجموعه‌ی جواب را با پاره‌خط‌هایی که p در سمت + آن‌ها قرار دارد محدود می‌کنیم.

در ادامه باید در مجموعه‌ی به دست آمده، پاره‌خط‌هایی را پیدا کنیم که خط حامل آن‌ها با pq برخورد می‌کنند. این کار با دو جستجوی بازه‌ای نیم‌صفحه‌ای در فضای دوگان قابل انجام است. به خاطر بیاورید که در فضای دوگان، خطوط حامل پاره‌خط‌های بحرانی به نقطه، و پاره‌خط pq به یک گوه دوگانه تبدیل می‌شوند (شکل ۶-۴ و بخش ۲-۵ را ببینید). بنابر این باید به دو جستجوی بازه‌ای در فضای دوگان پاسخ دهیم. به عبارت دقیق‌تر، ما به دنبال نقاطی در فضای دوگان هستیم که بین دو خط متناظر p و q قرار دارند.

توجه کنید که باید کار مشابهی را برای یافتن پاره‌خط‌های بحرانی که انتهای سمت راستشان بالای l و انتهای سمت چپشان پایین l قرار دارد تکرار کنیم.

همانطور که در بخش ۲-۴ اشاره شده است، این جستجوی بازه‌ای ۵ مرحله‌ای را می‌توان با پیش‌پردازش در زمان و حافظه‌ی $O(k^{۲+\epsilon})$، در زمان پرس‌وجوی $O(\log^۵ k)$ پاسخ داد، که k تعداد پاره‌خط‌های بحرانی می‌باشد. از آنجایی که تعداد پاره‌خط‌های بحرانی در یک چندضلعی ساده با n راس برابر با $O(n^۲)$ است، زمان و حافظه‌ی پیش‌پردازش برای ایجاد داده‌ساختار جستجوی بازه‌ای چندمرحله‌ای برابر با $O(n^{۴+\epsilon})$ خواهد بود. همچنین به الگوریتم لم ۱۹ برای محاسبه‌ی مقدار اندازه‌ی اولیه‌ی قابلیت دید p احتیاج داریم. زمان و حافظه‌ی مورد نیاز برای این الگوریتم به ترتیب $O(n^۳ \log n)$ و $O(n^۳)$ می‌باشد.

در زمان پرس‌وجو، به زمان $O(\log n)$ برای به دست آوردن اندازه‌ی قابلیت دید p، و به زمان $O(\log^5 n)$ برای به دست آوردن تعداد برخوردهای مثبت pq با پاره‌خط‌های بحرانی نیاز داریم. در نهایت قضیه زیر را خواهیم داشت.

قضیه ۷ با پیش‌پردازش یک چندضلعی ساده در زمان و حافظه‌ی $O(n^{4+\epsilon})$، می‌توانیم اندازه‌ی چندضلعی قابل دید یک پاره‌خط پرس‌وجو را در زمان $O(\log^5 n)$ به دست آوریم.

۶-۳-۳ برقراری توازن بین هزینه‌ی پیش‌پردازش و زمان پرس‌وجو

با استفاده از توازنی که در داده‌ساختار جستجوی بازه‌ای وجود دارد (بخش ۲-۴ را ببینید)، می‌توانیم در مسئله‌ی خود به توازنی بین هزینه‌ی پیش‌پردازش و هزینه‌ی زمان پرس‌وجو دست یابیم. اگر k نقطه در صفحه وجود داشته باشد، می‌توان داده ساختار جستجوی بازه‌ای به اندازه‌ی $O(m)$ که $k \leq m \leq k^2$، ایجاد کرد که با آن می‌توان به پرس‌وجوهای جستجو در زمان $O(\frac{k^{1+\epsilon}}{\sqrt{m}})$ پاسخ داد. همچنین جستجوهای چند مرحله‌ای را در همین زمان و حافظه می‌توان انجام داد [۱۱].

در مسئله‌ی مورد بررسی ما، k تعداد پاره‌خط‌های بحرانی است و برابر با $O(n^2)$ است. همچنین از الگوریتم لم ۱۹ استفاده می‌کنیم. در نتیجه خواهیم داشت:

قضیه ۸ با پیش‌پردازش یک چندضلعی ساده $\mathcal{P}$ در زمان $O(n^3 \log n + m^{1+\epsilon})$ و ساختن داده ساختاری به اندازه‌ی $O(n^3 + m)$، می‌توانیم اندازه‌ی چندضلعی قابل دید یک پاره‌خط پرس‌وجو را در زمان $O(\frac{n^{2+\epsilon}}{\sqrt{m}})$ گزارش کنیم که $n^2 \leq m \leq n^4$.

به عنوان یک نتیجه از قضیه‌ی ۸، کم‌ترین مقدار پیش‌پردازش زمانی خواهد بود که $m = n^3$. در این حالت زمان و حافظه‌ی مورد نیاز پیش‌پردازش به ترتیب $O(n^{3+\epsilon})$ و $O(n^3)$ است و زمان پاسخ‌دهی به پرس‌وجو $O(n^{1/2+\epsilon})$ می‌باشد.

۶-۴ اندازه‌ی تقریبی قابلیت دید ضعیف

در بخش قبل توانستیم تعادلی بین هزینه‌ی پیش‌پردازش و زمان اجرای پرس‌وجو به دست آوریم. اما با این حال هزینه‌ی پیش‌پردازش باز هم زیاد است. در این بخش الگوریتم تقریبی برای پیدا کردن اندازه‌ی چندضلعی قابل دید ضعیف ارائه می‌کنیم که دارای هزینه‌ی پیش‌پردازش کمتری است.

برای تقریب زدن اندازه‌ی چندضلعی قابلیت دید ضعیف از روش نمونه‌برداری تصادفی استفاده می‌کنیم. برای این که این روش قابل استفاده باشد، باید اطمینان داشته باشیم که چندضلعی قابل دید ضعیف به اندازه‌ی کافی بزرگ باشد. برای این منظور، احتیاج داریم که WVP را به شکل حساس به خروجی محاسبه کنیم تا هر زمان که از بزرگی آن اطمینان حاصل کردیم، الگوریتم را متوقف ساخته و نیازی به محاسبه‌ی کل چندضلعی قابل دید ضعیف نداشته باشیم.

لم ۲۱ [۱۳] با پیش‌پردازش یک چندضلعی ساده $\mathcal{P}$ در زمان $O(n)$ و ایجاد داده‌ساختاری به اندازه‌ی $O(n)$، می‌توانیم با دریافت پاره‌خط پرس‌وجوی pq، $\mathrm{WVP}(pq)$ را در زمان $O(k \log n)$ گزارش کنیم، که k اندازه‌ی خروجی گزارش شده می‌باشد.

با استفاده از لم ۲۱، می‌توانیم در زمان $O(\sqrt{n} \log n)$ بررسی کنیم که آیا اندازه‌ی $\mathrm{WVP}(pq)$ از $n^{1/2+\epsilon}$ بزرگ‌تر است یا خیر. از این نتیجه در بخش ۶-۴-۲ که یک نمونه‌ی تصادفی از اضلاع چندضلعی می‌سازیم استفاده خواهیم کرد.

۶-۴-۱ امتحان قابلیت دید ضعیف

برای تقریب زدن اندازه‌ی چندضلعی قابل دید ضعیف، باید بتوانیم در زمان سریع تشخیص بدهیم که آیا پاره‌خط پرس‌وجوی pq یکی از اضلاع چندضلعی را می‌تواند ببیند یا نه. لم زیر چگونگی انجام این کار را نشان می‌دهد.

لم ۲۲ چندضلعی ساده $\mathcal{P}$ را می‌توان در زمان و حافظه‌ی $O(n^2)$ پیش‌پردازش کرد، به نحوی که به ازای یک پاره‌خط پرس‌وجوی pq و یک ضلع دلخواه e از $\mathcal{P}$، در زمان $O(\log n)$ می‌توانیم بررسی کنیم که آیا pq و e نسبت به هم قابل دید هستند یا نه.

اثبات ۱۴ چندضلعی قابل دید ضعیف $\mathrm{WVP}(e)$ که عبارت است از نقاطی از $\mathcal{P}$ که از e قابل دید ضعیف هستند در نظر بگیرید. این چندضلعی دارای اندازه‌ی $O(n)$ است و آن را می‌توان در زمان $O(n)$ محاسبه کرد. در زمان پیش پردازش به ازای همه‌ی اضلاع $\mathcal{P}$، $\mathrm{WVP}(e)$ را محاسبه می‌کنیم. به ازای هر یک از این چندضلعی‌های قابل دید ضعیف، داده‌ساختار پرتو افکنی را در زمان و حافظه‌ی $O(n)$ می‌سازیم، به نحوی که در زمان $O(\log n)$ به پرسش‌های پرتوافکنی پاسخ دهیم [۹]. در زمان پرس‌وجو کافی است که دو پرسش پرتوافکنی از p و q انجام دهیم تا تشخیص دهیم که آیا pq با $\mathrm{WVP}(e)$ برخورد می‌کند یا خیر.

محاسبه‌ی $\mathrm{WVP}(e)$ به ازای همه‌ی اضلاع $\mathcal{P}$ و آماده کردن داده‌ساختار پرتو افکنی برای هر یک از آن‌ها را می‌توان در زمان و حافظه‌ی $O(n^2)$ انجام داد. همچنین زمان اجرای پرس‌وجوی این الگوریتم $O(\log n)$ می‌باشد.

۶-۴-۲ نمونه‌برداری تصادفی

اکنون با استفاده از الگوریتم بخش ۶-۴-۱، روشی را برای تقریب مقدار WVC ارائه می‌کنیم. فرض کنید که مقدار دقیق WVC برابر با m_p باشد. روشی که ارائه می‌دهیم از دو بخش تشکیل شده است. در بخش اول، الگوریتم لم ۲۱ را اجرا می‌کنیم تا زمانی که $O(\sqrt{n})$ ضلع از اضلاع چندضلعی قابل دید ضعیف را به دست آوریم. چنانچه قبل از به دست آمدن این تعداد ضلع الگوریتم به اتمام برسد، مقدار دقیق m_p را به دست آورده‌ایم و کار ما به اتمام می‌رسد. در غیر این صورت، می‌دانیم که $m_p = k > O(\sqrt{n})$. در مرحله‌ی دوم، مجموعه‌ی تصادفی R_i از اضلاع چندضلعی را به گونه‌ای تشکیل می‌دهیم که هر ضلع

چندضلعی با احتمال $۱/\sqrt{n}$ در این مجموعه شرکت داشته باشد. سپس به ازای هر ضلع $s \in \mathcal{R}_i$، با استفاده از لم ۲۲ بررسی می‌کنیم که آیا این ضلع نسبت به pq قابل دید هست یا نه.

به شیوه‌ای که در بالا گفته شد، t مجموعه‌ی $\mathcal{R}_i$ از اضلاع تصادفی چندضلعی تشکیل می‌دهیم. اگر X_i تعداد اضلاع قابل دید از pq در مجموعه‌ی $\mathcal{R}_i$ باشد، مقدار $m'_p = \frac{\sum_{i=۱}^{t} \sqrt{n} X_i}{t}$ را به عنوان تقریبی از m_p گزارش می‌دهیم.

لم ۲۳ (لم چبیشف) فرض کنید $X_۱, X_۲, \ldots, X_t$ متغیرهای تصادفی باشند و $E(X_i) = \mu$، و فرض کنید $\epsilon > ۰$ هر مقدار حقیقی مثبت باشد. در اینصورت

$$P((|\frac{X_1 + X_2 + \ldots + X_t}{t} - \mu|) > \epsilon) \leq \frac{Var(X)}{t\epsilon^2}$$

با استفاده از لم ۲۴ نتیجه زیر به دست می‌آید.

لم ۲۴ با احتمال نزدیک به ۱ خواهیم داشت:

$$(1-\delta)m_p \leq \sqrt{n}\frac{X_1 + X_2 + \ldots + X_t}{t} \leq (1+\delta)m_p$$

اثبات ۱۵ در لم ۲۴ قرار می‌دهیم $\epsilon = \delta m_p$، که δ یک عدد ثابت است که در زمان پیش‌پردازش به میزان دلخواه کوچک انتخاب می‌شود، و $t = ۱/\delta$. بر این اساس خواهیم داشت $E(\sqrt{n}X_i) = m_p$، و $Var(\sqrt{n}X_i) = nm_p(۱ - \frac{۱}{\sqrt{n}})\frac{۱}{\sqrt{n}}$. بنابر این:

$$P = P(|\sqrt{n}\frac{X_1 + X_2 + \ldots + X_t}{t} - m_p| > \delta m_p) \leq \frac{\sqrt{n}m_p}{\delta^2 m_p^2}$$

از آنجایی که $m_p > O(n^{۱/۲+\epsilon})$، می‌دانیم $P \sim ۰$. به این خاطر با احتمال حداقل $P - ۱$ داریم:

$$(1-\delta)m_p \leq m'_p \leq (1+\delta)m_p$$

قضیه ۹ با پیش‌پردازش چندضلعی ساده در زمان و حافظه‌ی $O(n^۲)$ می‌توان اندازه‌ی چندضلعی قابل دید ضعیف یک پاره‌خط داده شده را در زمان $O(۱/\delta n^{۱/۲+\epsilon})$ تقریب زد. مقدار m'_p که این الگوریتم به عنوان پاسخ ارائه می‌دهد به گونه‌ای است که اگر $m_p > \sqrt{n}$، با احتمال بالایی خواهیم داشت $(۱-\delta)m_p \leq m'_p \leq (۱+\delta)m_p$. اگر $m_p \leq \sqrt{n}$، الگوریتم مقدار دقیق m_p را به دست می‌دهد.

اثبات ۱۶ در مرحله‌ی اول از لم ۲۱ استفاده می‌کنیم که به پیش‌پردازش $O(n)$ احتیاج دارد و در زمان پرس‌وجوی $O(n^{۱/۲+\epsilon})$ اجرا می‌شود. در مرحله‌ی دوم، بر اساس لم ۲۲ برای انجام آزمون قابلیت دید یک ضلع چندضلعی و پاره‌خط pq در زمان $O(\log n)$، به پیش‌پردازش $O(n^۲)$ نیاز خواهیم داشت. از آنجایی که برای هر $\mathcal{R}_i$ تعداد $O(\sqrt{n})$ ضلع تصادفی انتخاب می‌کنیم و $t = ۱/\delta$ یک مقدار ثابت می‌باشد، زمان اجرا برابر است با $O(۱/\delta n^{۱/۲+\epsilon} + \log n\sqrt{n}) = O(۱/\delta n^{۱/۲+\epsilon})$. بنابر این زمان پرس‌وجوی هر دو مرحله $O(n^{۱/۲+\epsilon})$ خواهد بود و زمان پیش‌پردازش مورد نیاز $O(n^{۲+\epsilon})$ است.

۶-۵ خلاصه

در این فصل به مسئله‌ی محاسبه‌ی اندازه‌ی چندضلعی قابل دید ضعیف یک پاره‌خط در یک چندضلعی ساده پرداختیم. دو الگوریتم دقیق که این اندازه را در زمان زیرخطی محاسبه می‌کنند ارائه کردیم. همچنین یک توازن بین هزینه‌ی پیش‌پردازش و زمان پرس‌وجو معرفی شد. درنهایت، با تقریب زدن مقدار مورد نظر توسط نمونه‌برداری تصادفی، به هزینه‌ی پیش‌پردازش کمتر دست یافتیم.

فصلِ ۷

قابلیت دید پاره‌خط متحرک

۱-۷ مقدمه

در این فصل به بررسی قابلیت دید پاره‌خط در میان چند پاره‌خط و یا شیء محدب در صفحه می‌پردازیم و نشان می‌دهیم که چگونه قابلیت دید را در حالت اولیه محاسبه کرده، و سپس برای یک ناظر متحرک که در صفحه حرکت می‌کند، آن را نگهداری کنیم. به عبارت دیگر، هدف ما در این فصل محاسبه و نگهداری ساختار قابلیت دید پاره‌خط است.

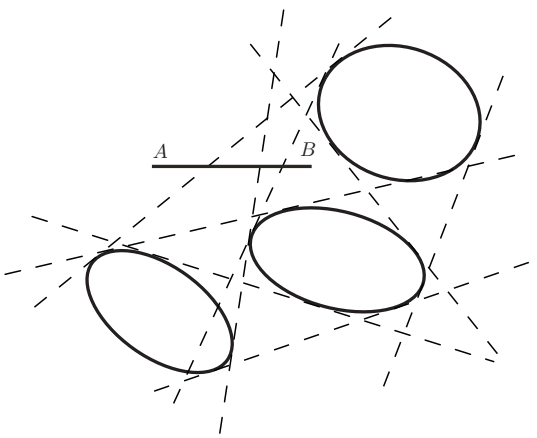
ابتدا حالت ایستای مسئله که ناظر و اشیاء ثابت هستند را بررسی می‌کنیم و سپس به حالت پویای مسئله می‌پردازیم که در آن ناظر می‌تواند در بین اشیاء حرکت کند.

۲-۷ قابلیت دید پاره‌خط ایستا

از این پس فرض می‌کنیم که پاره‌خط AB در صحنه‌ای که شامل n شیء محدب است، قرار گرفته است. در حالت ایستا هیچ یک از اشیاء و ناظر حرکت نمی‌کنند.

ابتدا گراف قابلیت دید اشیاء داده شده را به‌دست‌می‌آوریم. یال‌های قابلیت دید یا همان پاره‌خط‌های بحرانی، تجزیه‌ی قابلیت دید را به وجود می‌آورند که متشکل از نواحی با قابلیت دید ثابت می‌باشد. یک ناظر نقطه‌ای p را در نظر بگیرید که مکان اولیه‌ی آن روی سر پاره‌خط، A باشد. قابلیت دید ناظر p را می‌توانیم به وسیله‌ی روش Riviere [۴۱] به دست بیاوریم که در آن از مجتمع قابلیت دید[1] برای محاسبه و نگهداری قابلیت دید برای ناظر نقطه‌ای متحرک استفاده می‌شود. بسته به روش پیاده‌سازی الگوریتم، قابلیت دید را می‌توان در زمان $O(v \log n)$ و یا $O(n \log n)$ به‌دست‌آورد، که v اندازه‌ی فضای قابل دید ناظر است. منظور از اندازه‌ی فضای قابل دید، پیچیدگی و یا تعداد تغییرات ساختاری در آن می‌باشد. با توجه به این که تعداد این تغییرات حداکثر $O(n)$ است، این اندازه برابر با $O(n)$ خواهد بود. بعد از به دست آوردن قابلیت

[1] Visibility Complex



شکلِ ۷–۱: یک ناظر پاره‌خط بین تعدادی شیء محدب.

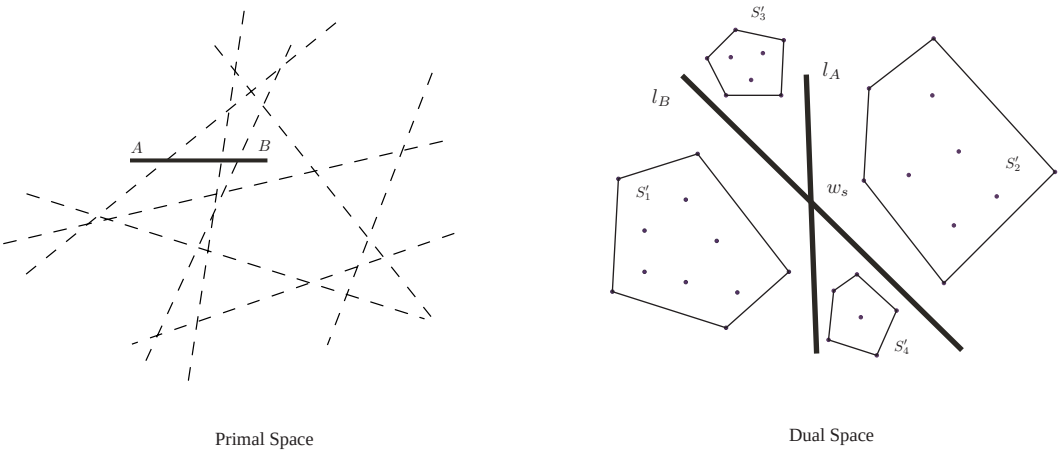
دید اولیه‌ی نقطه‌ی p، ناظر p را از A به سمت B و در راستای پاره‌خط حرکت می‌دهیم. تا هنگامی که ناظر درون یک ناحیه قابلیت دید آن قرار دارد، قابلیت دید تغییر نمی‌کند. هنگامی که ناظر یک پاره‌خط بحرانی را قطع کند، قابلیت دید تغییر کرده و باید آن را به‌روز کنیم. بعد از محاسبه‌ی $v_\circ$، قابلیت دید اولیه‌ی حول ناظر نقطه‌ای p، و پیش‌پردازشی که در زمان $O(v_\circ \log v_\circ) = O(n \log n)$ انجام می‌پذیرد، قابلیت دید حول p را در هر تغییر قابلیت دید می‌توان در زمان $O(\log v) = O(\log n)$ به‌روز کرد [۴۱]، که v اندازه‌ی قابلیت دید در لحظه‌ی رخ دادن تغییر می‌باشد. بنابراین چنانچه تعداد پاره‌خط‌های بحرانی که ناظر قطعه‌ای را قطع می‌کنند برابر k باشد، زمان مورد نیاز برای محاسبه‌ی قابلیت دید ناظر پاره‌خطی برابر $O((n+k)\log n)$ خواهد بود. با لحاظ کردن زمان مورد نیاز برای ساختن مجتمع قابلیت دید، نتیجه زیر را خواهیم داشت:

گزاره ۱ بعد از ساختن مجتمع قابلیت دید در صحنه‌ای شامل n شیء هموار محدب، و پیش‌پردازشی در زمان $O(m + n \log n)$، که m تعداد پاره‌خط‌های بحرانی موجود در صحنه است، قابلیت دید از یک پاره‌خط را می‌توان در زمان $O((n+k)\log n)$ محاسبه کرد، که k تعداد پاره‌خط‌های بحرانی است که با پاره‌خط ناظر برخورد می‌کنند.

۷–۳ قابلیت دید پاره‌خط متحرک

اکنون حالت پویای مسئله‌ی قابلیت دید پاره‌خط را بررسی می‌کنیم: یک ناظر پاره‌خطی s در بین مجموعه‌ای از n شیء محدب حرکت می‌کند. می‌خواهیم قابلیت دید ناظر را در حین حرکت آن نگهداری کنیم. روشی که انجام خواهیم داد، محاسبه‌ی قابلیت دید به روش گفته شده در قسمت ۷–۲ و به‌روزرسانی قابلیت دید همراه با تغییر حرکت ناظر می‌باشد.

برای محاسبه‌ی قابلیت دید، باید اطلاعاتی درباره‌ی مجموعه‌ی سلول‌هایی از تجزیه‌ی قابلیت دید که ناظر با آن‌ها برخورد می‌کند داشته باشیم و، برای هر سلول، پاره‌خط‌های بحرانی که مرز آن را تشکیل



شکلِ ۷-۲: ناظر درون تجزیه‌ی قابلیت دید، در فضای اصلی (سمت چپ) وفضای دوگان (سمت راست).

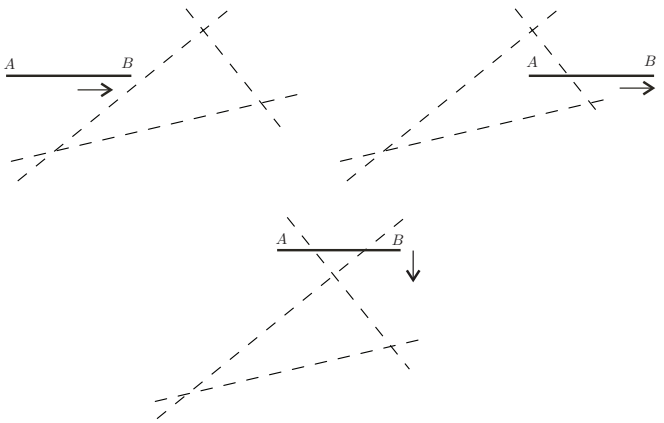
داده‌اند را بدانیم. برای سادگی محاسبات و فهم بهتر، مسئله را به فضای دوگان می‌بریم (برای تعریف فضای دوگان به بخش ۲-۵ مراجعه کنید).

در ابتدا دوگان مجموعه‌ی خطوط بحرانی S و دوگان ناظر قطعه‌ای s را محاسبه می‌کنیم. نام این دوگان‌ها را $S' = \mathcal{D}(s)$ و $w_s = \mathcal{D}(s)$ می‌گذاریم. در صفحه‌ی دوگان، هر خط بحرانی l به یک نقطه p_l، و ناظر s به گوه دوگانه[2] w_s تبدیل می‌شود (شکل ۷-۲). منظور از گوه‌ی دوگانه، دو ناحیه‌ی مقابل هم است که از برخورد دو خط راست با هم تشکیل می‌شوند. توجه کنید که نقاط پایانی ناظر، یعنی A و B، به دو خط l_A و l_B تبدیل می‌شوند که مرزهای w_s را تشکیل می‌دهند. گوه‌ی دوگانه‌ی w_s صفحه و مجموعه‌ی نقاط S' را به چهار مجموعه‌ی S'_1، S'_2، S'_3 و S'_4 تقسیم می‌کند که دو مجموعه‌ی اول بیرون گوه و دو مجموعه‌ی دوم درون گوه هستند. وقتی دوگان یک خط بحرانی بیرون گوه قرار دارد به این معنی است که ناظر پاره‌خطی با خط بحرانی در صفحه اصلی برخورد نمی‌کند، و چنانچه دوگان آن داخل گوه باشد، ناظر با خط بحرانی برخورد کرده است. این ویژگی به ما کمک می‌کند که به راحتی تشخیص بدهیم که آیا خط بحرانی با ناظر برخورد می‌کند یا خیر.

حال ببینیم وقتی که قابلیت دید ناظر تغییر می‌کند چه اتفاقی می‌افتد. هنگامی که ناظر در صفحه شروع به حرکت می‌کند، تعدادی از ناحیه‌های قابلیت دید که توسط خطوط بحرانی ایجاد شده‌اند را قطع می‌کند. تا زمانی که این ناحیه‌های قطع شده و ترتیب برخورد با آن‌ها توسط ناظر تغییر نکند، تغییری در قابلیت دید ناظر ایجاد نمی‌شود. بنابراین برای یافتن شرایطی که باعث تغییر قابلیت دید می‌گردد، باید حالاتی را درنظر بگیریم که این ناحیه‌های برخورد شده تغییر کنند.

یک تغییر در مجموعه‌ی ناحیه‌های قابلیت دید در دو حالت رخ خواهد داد: ۱) ناظر با یک خط بحرانی جدید که قبلاً برخورد نداشته، برخورد کند و در نتیجه وارد یک ناحیه‌ی قابلیت دید جدید شود، و یا برخورد آن با یک خط بحرانی تمام شود و در نتیجه از یک ناحیه‌ی قابلیت دید خارج شود. ۲) ترتیب

[2] Double Wedge



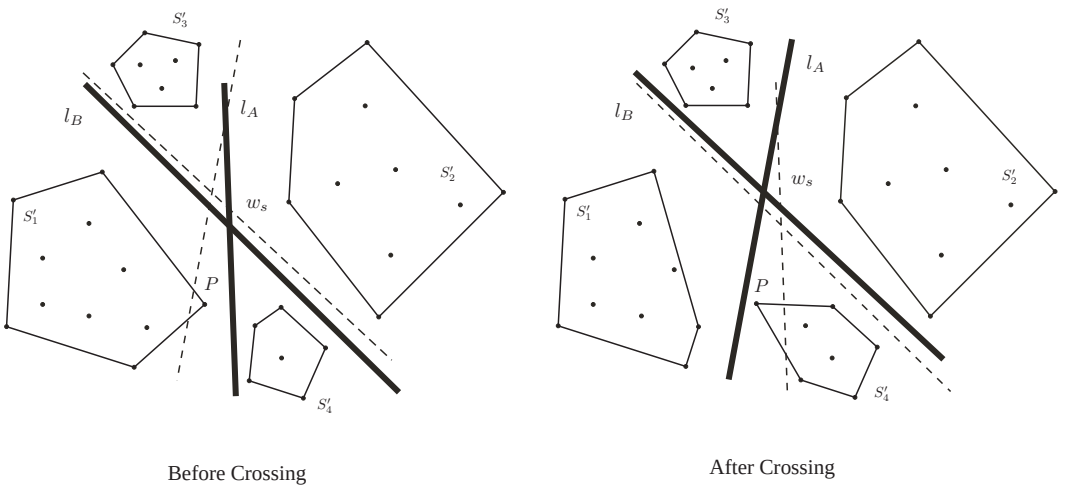
شکلِ ۷–۳: رخداد گذر (بالا) و رخداد تقاطع (پایین).

برخورد ناظر با دو خط بحرانی تغییر کند و در نتیجه ناظر از یک ناحیه خارج و وارد یک ناحیه‌ی دیگر شود (شکل ۷–۳). می‌توان بررسی نمود که این دو حالت تنها حالاتی هستند که ناحیه‌های قطع شده تغییر می‌کنند. دو رخداد ۱ و ۲ را به ترتیب رخداد گذر و رخداد تقاطع می‌گوییم. در ادامه نشان خواهیم داد که هر کدام از این نوع رخدادها چگونه پردازش می‌شوند و تاثیر آن‌ها بر روی قابلیت دید ناظر را بررسی می‌کنیم.

۷–۳–۱ رخداد گذر

چنانچه بیان گردید، یک رخداد گذر در دو حالت اتفاق می‌افتد: ۱) هنگامی که ناظر با یک خط بحرانی جدید برخورد کند و درنتیجه وارد یک ناحیه‌ی قابلیت دید جدید شود، ۲) هنگامی که ناظر دیگر با خط بحرانی که برخورد داشته، برخورد نکند و به عبارت دیگر، از یک ناحیه‌ی قابلیت دید خارج گردد (شکل ۷–۳). در فضای دوگان، این دو رخداد را به این صورت می‌توان بیان کرد: وقتی ناظر شروع به برخورد با یک خط بحرانی می‌کند، دوگان خط بحرانی در فضای دوگان وارد گوه‌ی دوگانه می‌شود. به‌همین شکل، وقتی ناظر دیگر با خط بحرانی برخورد نداشته باشد، دوگان خط بحرانی در فضای دوگان از گوه خارج می‌گردد. زمان دقیق وقوع این رخدادها هنگامی است که دوگان یک خط بحرانی بر روی یال‌های گوه‌ی دوگانه قرار بگیرد.

برای تشخیص رخدادهای گذر، کافی است که دوگان دو سر ناظر، یعنی A و B را درنظر بگیریم. هنگامی که نقطه‌ی پایانی A (به همین شکل B) به خط l در مرز یک ناحیه‌ی قابلیت دید نزدیک می‌شود و در مجاورت آن قرار می‌گیرد، خط دوگان $l_A = \mathcal{D}(A)$ (به همین شکل $l_B = \mathcal{D}(B)$) در مجاورت نقطه‌ی دوگان $P = \mathcal{D}(l)$ قرار می‌گیرد. چنانچه A در صفحه‌ی اصلی خط l را قطع کند، در فضای دوگان نقطه‌ی P خط l_A را قطع خواهد کرد (شکل ۷–۴). در ادامه نشان می‌دهیم که چگونه بعد از این نوع رخداد، داده‌ساختار را به‌روز کنیم.



شکلِ ۷–۴: آرایش فضای دوگان قبل و بعد از رخداد گذر.

با حرکت ناظر در صفحه، گوه‌ی معادل ناظر در صفحه‌ی دوگان شروع به حرکت و احتمالاً دوران خواهد کرد. تا هنگامی که مجموع خطوط بحرانی برخورد شده توسط ناظر تغییر نکند، رخداد گذر به وقوع نمی‌پیوندد. در صفحه‌ی دوگان، این مسئله معادل این‌است که تا هنگامی که گوه هیچ کدام از نقاط بحرانی را قطع نکرده، در صفحه حرکت و دوران کند. وقتی که ناظر با یک خط بحرانی جدید برخورد کند و یا برخورد آن با خط بحرانی پایان یابد، ساختار باید تغییر کند. در صفحه‌ی دوگان، این رخدادها معادل این است که یک نقطه وارد گوه گردد و یا یک نقطه از آن خارج شود. برای تشخیص پدید آمدن یک رخداد گذر، رخداد معادل آن در صفحه‌ی دوگان، یعنی برخورد یک نقطه‌ی بحرانی با مرزهای گوه را تشخیص می‌دهیم، یعنی چهار مجموعه‌ی نقطه‌ای که توسط گوه به‌وجود می‌آیند، یعنی S'_1، S'_2، S'_3 و S'_4 را نگهداری می‌کنیم. برای این کار از روش زیر استفاده می‌کنیم.

پوشش محدب هریک از مجموعه‌های $S_{i(i=1..4)}$ را به‌دست می‌آوریم. به‌راحتی می‌توان نشان داد که یک نقطه قبل از این که با مرزهای گوه برخورد کند باید جزء یکی از پوشش‌های محدب باشد. بنابراین رخداد گذر زمانی ایجاد می‌شود که مرزهای گوه‌ی درحال حرکت با پوشش‌های محدب برخورد کند. در این حالت، یک نقطه از یک پوشش محدب خارج شده و وارد یک پوشش محدب دیگر می‌شود و باید برای حفظ ساختار، پوشش‌های محدب به‌روز گردند. هزینه‌ی به‌روزرسانی یک تغییر در پوشش محدب پویا برابر $O(\log^2 n)$ است [۳۹]. این به‌روزرسانی را می‌توان در زمان سرشکن شده‌ی $O(\log n)$ هم انجام داد [؟].

مجموعه راس‌هایی که در یک پوشش محدب قرار دارند مرزهای سلول‌های تجزیه‌ی قابلیت دید را مشخص می‌کنند. دو پوشش محدب خارج گوه را درنظربگیرید. وقتی یک نقطه به مجموعه‌ی نقاط پوشش محدب اضافه می‌گردد، این نقطه توسط یک مماس که به پوشش محدب سابق ایجاد می‌کند، وارد پوشش می‌گردد. نقاطی از پوشش سابق که در پوشش محدب جدید قرار ندارند، عبارتند از خطوط بحرانی که دیگر جزء مرز سلولی که نقطه‌ی نهایی پاره‌خط در آن قرار دارد، نیستند. به همین شکل، با حذف یکی از

راس‌های یک پوشش محدب، تعداد نامشخصی نقطه ممکن است به پوشش اضافه‌گردند. این نقاط معادل خطوط بحرانی مرزی سلول جدیدی هستند که نقطه‌ی پایانی در آن وارد شده است [۲۲]. چهار پوشش محدبی که در صفحه‌ی دوگان نگهداری می‌کنیم، توصیف دقیقی از خطوط بحرانی که ناظر را قطع‌کرده‌اند می‌دهند. این پوشش‌ها را می‌توان در حافظه‌ی خطی نسبت به تعداد خطوط بحرانی نگهداری کرد.

با حرکت پاره‌خط ناظر در صفحه، گوه‌ی متناظر آن هم در صفحه‌ی دوگان شروع به حرکت می‌کند. با دانستن معادله‌ی حرکت دو نقطه‌ی پایانی A و B، مسیر حرکت خطوط مرزی گوه، l_A و l_B هم مشخص است و می‌توانیم لیستی از زمان‌های برخورد این خطوط با نقاط متناظر اشیاء در صفحه‌ی دوگان به‌صورت مرتب شده به دست بیاوریم. هر کدام از این زمان‌ها یک رخداد گذر هستند و به ترتیب اتفاق افتادن باید پردازش شوند. هرکدام از این رخدادها را می‌توان در زمان سرشکن‌شده‌ی $O(\log n)$ و در بدترین زمان $O(\log^۲ n)$ پردازش کرد و داده‌ساختار نگهداری شده، که در این‌جا چهار پوشش محدب در صفحه هستند را نگهداری کنیم. بنابر آن‌چه گفته شد، نتیجه‌ی زیر را خواهیم داشت:

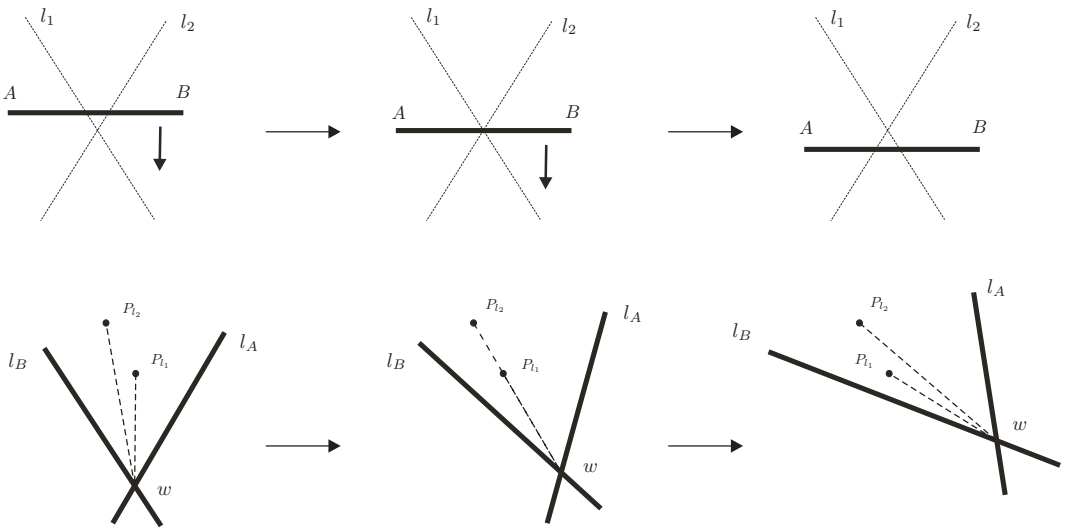
گزاره ۲ بعد از پیش‌پردازش در زمان $O(m + n \log n)$ و محاسبه قابلیت دید اولیه‌ی ناظر پاره‌خطی در زمان $O((n + k) \log n)$، که m تعداد کل خطوط بحرانی و k تعداد خطوط بحرانی‌ای است که در ابتدا با ناظر برخوردکرده‌اند، با هر رخداد گذر، قابلیت دید ناظر را می‌توان در زمان $O(\log n)$ به‌روز کرد.

۷-۳-۲ رخداد تقاطع

نوع دوم تغییری که در ساختار قابلیت دید ناظر پاره‌خط اتفاق می‌افتد زمانی است که ترتیب برخورد ناظر با دو ناحیه‌ی قابلیت دید تغییرکند (شکل ۷-۳). دقت کنید که در این حالت، مجموعه خطوط بحرانی‌ای که ناظر با آن‌ها برخورد می‌کند تغییر نمی‌کند. بنابراین افرازی از نقاط که گوه در صفحه ایجادکرده‌است و مجموعه‌ی پوشش‌های محدب تغییر نمی‌یابند.

اکنون بررسی می‌کنیم که در زمان پدید آمدن یک رخداد تقاطع چه اتفاقی می‌افتد (شکل ۷-۵ را ببینید). ناظر AB را به همراه دو خط انفصال $l_۱$ و $l_۲$ در نظر می‌گیریم. در صفحه‌ی دوگان، ناظر به گوه‌ی با مرکز w و با مرزهای l_A و l_B تبدیل می‌شود و خطوط $l_۱$ و $l_۲$ به نقاط $P_{l_۱}$ و $P_{l_۲}$. با حرکت پاره‌خط، در صفحه‌ی دوگان پاره‌خط‌های $wp_{l_۱}$ و $wp_{l_۲}$ به هم نزدیک‌تر می‌شوند و زاویه‌ی بین آن‌ها کاهش می‌یابد. در زمان رخداد، هنگامی که AB در نقطه‌ی برخورد $l_۱$ و $l_۲$ قرارمی‌گیرد، $wp_{l_۱}$ و $wp_{l_۲}$ بر هم منطبق می‌گردند. بعد از رخداد تقاطع، ترتیب برخورد ناظر با این خطوط بحرانی تغییر می‌کند و $wp_{l_۱}$ و $wp_{l_۲}$ ترتیب زاویه‌ای خود را نسبت به مرکز گوه تغییر می‌دهند. به این ترتیب می‌توان نشان داد که هرگاه دو نقطه‌ی دوگان $p_{l_۱}$ و $p_{l_۲}$ و مرکز گوه‌ی w، در یک راستا قرار بگیرند یک رخداد تقاطع به وجود می‌آید، و برعکس.

اکنون نشان می‌دهیم که چگونه این نوع رخداد را به وسیله‌ی الگوریتمی که در [۴۱] برای محاسبه قابلیت دید یک ناظر نقطه‌ای مطرح شده است، پردازش کنیم. فرض کنید که تعداد نقاط داخل گوه برابر m' است، که $m' = O(m) = O(n^۲)$. این نقاط داخل گوه را به عنوان رئوس چندضلعی‌ها، و مرکز گوه را به عنوان ناظر نقطه‌ای برای مسئله قابلیت دید ناظر متحرک در میان اشیاء چندضلعی در نظر می‌گیریم. وقتی دو نقطه از این رئوس با w هم‌راستا شوند، معادل این است که w از خط بحرانی تعریف شده توسط این دو راس عبور کند.



شکلِ ۷–۵: وضعیت صفحه اصلی (بالا) و صفحه دوگان (پایین)، قبل از رخداد تقاطع (سمت چپ)، در هنگام تقاطع (وسط) و بعد از رخداد تقاطع (سمت راست).

با تعداد m' رأس در صفحه، $m'' = O(m'^2) = O(n^4)$ خط بحرانی خواهیم داشت، هرچند به‌صورت میانگین، m' خیلی کمتر از $O(n^2)$ خواهد بود. با استفاده از الگوریتم [۴۱]، تغییرات قابلیت دید نقطه‌ای در این مسئله، و یا همان رخداد تقاطع در مسئله اصلی، را پردازش می‌کنیم. در این الگوریتم با محاسبه‌ی مجتمع قابلیت دید برای اشیاء صحنه، هر تغییر قابلیت دید را می‌توان در زمان $O(\log n)$ پردازش کرد، که n در این‌جا پیچیدگی صحنه است. مجتمع قابلیت دید صحنه را در زمان $O(m'' + m' \log m')$ و یا $O(n^4 + n^2 \log n)$ (با توجه به این‌که $O(\log n^2) = O(\log n)$) و با حافظه $O(m') = O(n^2)$ به‌دست آوریم. با داشتن مجتمع قابلیت دید، هر رخداد قابلیت دید را در زمان $O(\log m') = O(\log n)$ پردازش می‌کنیم. همان‌طور که گفتیم، هر رخداد قابلیت دید متناظر با هم‌راستا شدن w با دو نقطه‌ی داخل گوه است، و این معادل یک رخداد تقاطع می‌باشد.

نکته‌ای که باقی می‌ماند، بررسی تاثیر یک رخداد گذر بر این داده‌ساختاری که برای پردازش رخداد تقاطع ایجاد کرده‌ایم، می‌باشد. این تاثیر وقتی ایجاد می‌شود که در صفحه‌ی دوگان، نقطه‌ای از گوه خارج و یا نقطه‌ای وارد گوه شود. در این مواقع باید مجتمع قابلیت دید نقاط درون گوه را به‌روز کنیم. وقتی نقطه‌ای وارد گوه می‌شود، بایستی آن را به مجتمع قابلیت دید اضافه کنیم. این کار را می‌توان با محاسبه‌ی قابلیت دید این نقطه به وسیله مجتمع قابلیت دید که در زمان $O(m' \log n)$ قابل محاسبه است انجام داد، که m' تعداد نقاط درون گوه می‌باشد. به شکل مشابه، وقتی نقطه‌ای گوه را ترک می‌کند، باید یال‌های متناظر آن در مجتمع قابلیت دید را حذف کرد، که این کار را می‌توان در زمان $O(m')$ انجام داد [۴۱].

از مباحث بالا به نتیجه‌ی کلی زیر می‌رسیم:

گزاره ۳ بعداز مرحله‌ی پیش‌پردازش در زمان $O(m'^2 + (m' + n) \log n)$ و محاسبه قابلیت دید اولیه یک

ناظر پاره‌خطی در زمان $O((m'+n)\log n)$، که m' تعداد خطوط بحرانی است که با ناظر برخورد کرده‌اند، می‌توانیم یک رخداد گذر را در زمان $O((m'+۱)\log n)$ و یک رخداد تقاطع را در زمان $O(\log n)$ پردازش کنیم.

۷–۴ محاسبات محلی برای پردازش رخدادها

در بخش قبل دیدیم که تغییرات قابلیت دید در خطوط بحرانی تشکیل شده توسط اشیاء رخ می‌دهند. برای یک صحنه شامل n شیء، تعداد خطوط بحرانی $\Omega(n)$ و $O(n^۲)$ می‌باشد [۲۹].

در روشی که در بخش قبل دیدیم، تجزیه‌ی قابلیت دید تشکیل شده توسط خطوط بحرانی را محاسبه کردیم. این تجزیه صفحه را به سلول‌هایی تقسیم می‌کند که قابلیت دید در یک سلول ثابت است. در هنگام حرکت یک ناظر، کافی است که اطلاعات سلول‌هایی که در آن‌ها قرار دارد را داشته باشیم. با تغییر این سلول‌ها، قابلیت دید را با اطلاعاتی که در سلول‌ها ذخیره شده‌اند به‌روز می‌کنیم.

مشکلی که روش مطرح شده وجود دارد، پیچیدگی بالای آرایش خطوط بحرانی است. تعداد سلول‌ها برای m خط بحرانی $O(m^۲)$ است، و خود تعداد خطوط بحرانی برای n شیء در صحنه، $O(n^۲)$ است. به عبارت دیگر، اطلاعات همه‌ی سلول‌های صحنه پردازش می‌شوند، ولی تنها اطلاعات سلول‌های مجاور ناظر مورد استفاده قرار می‌گیرند. مشکل دیگری که این روش داراست (علاوه بر پیش‌پردازش پرهزینه برای محاسبه اطلاعات همه‌ی سلول‌ها)، رخدادهای غیر واقعی هستند که از برخورد ناظر با خطوط بحرانی مرزی سلول‌های دور ایجاد می‌شوند.

در این بخش روشی برای محلی کردن محاسبات برای قابلیت دید ناظر قطعه‌ای ارائه می‌دهیم.

۷–۴–۱ نقشه توپولوژیکی

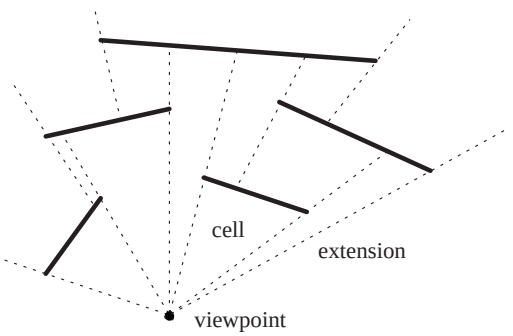
در ابتدا ساختاری به اسم نقشه توپولوژیکی[3] را مطرح می‌کنیم که در [۳۶] معرفی شده است. این ساختار تا حدودی شبیه نقشه ذوزنقه‌ای[4] است که در مسئله مکان‌یابی مورد استفاده قرار می‌گیرد [۱۵].

فرض می‌کنیم که در صفحه، مجموعه S از n پاره‌خط که برخوردی با یکدیگر ندارند، و یک ناظر نقطه‌ای v وجود داشته باشد. نقشه‌ی توپولوژیکی از رسم دو امتداد از هر یک از راس‌های هر پاره‌خط به‌دست می‌آید، که یک امتداد از راس به سمت نقطه‌ی دید v و امتداد دیگر در جهت مخالف می‌باشد. این امتدادها وقتی که با نقطه دید و یا یک پاره‌خط دیگر برخورد کنند متوقف می‌شوند. نقشه‌ی توپولوژیکی S، یک تقسیم‌بندی صفحه است که توسط S، نقطه‌ی ناظر v و امتدادها ایجاد می‌شود (شکل ۷–۶). وجه‌های این تقسیم‌بندی سلول‌های توپولوژیکی یا به طور ساده سلول گفته می‌شود.

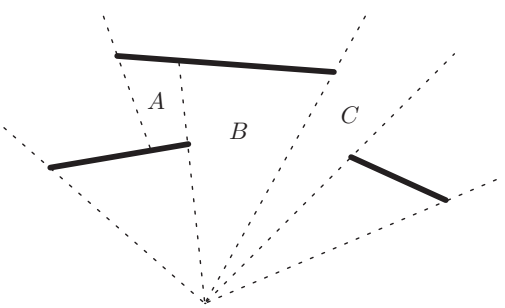
دو سلول $c_۱$ و $c_۲$ همسایه هستند اگر یکی از امتدادها جزء مرز هر دو سلول باشد. هر سلول حداکثر چهار سلول همسایه خواهدداشت (شکل ۷–۷). به این ترتیب برای نگهداری اطلاعات کامل یک

[3] Topological map

[4] Trapezoidal map



شکلِ ۷–۶: نقشه‌ی توپولوژیکی برای مجموعه‌ای از پاره‌خط‌ها.

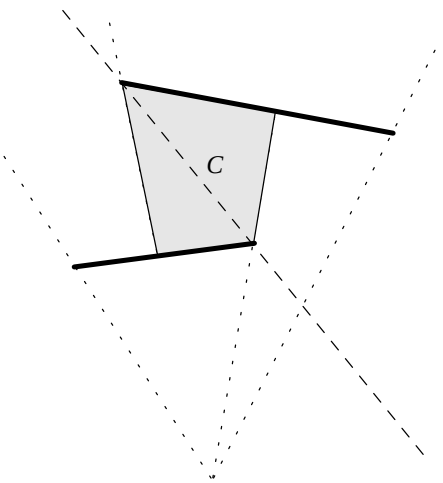


شکلِ ۷–۷: همسایگی‌های مختلف در نقشه‌ی توپولوژیکی. سلول A دو سلول همسایه دارد، سلول B در مجاورت سه سلول دیگر است و سلول C در همسایگی چهار سلول دیگر قرار دارد.

سلول، علاوه بر اشاره‌گرهایی به سلول‌های مجاور، اشاره‌گرهایی به راس سمت راست و چپ مجاور سلول و پاره‌خط‌های بالا و پایین آن نگهداری می‌کنیم.

به طور کلی سه نوع سلول وجود دارد: سلول‌هایی که در نقطه‌ی دید برخورد می‌کنند، سلول‌های درونی داخل صحنه و سلول‌های باز در مرز. چنانچه یک سلول که شامل نقطه‌ی دید است از سمت بالا محدود شده باشد، مرز بالایی سلول شامل قسمت قابل دید یک پاره‌خط است. برای نگهداری پاره‌خط‌های قابل دید، با داشتن نقشه‌ی توپولوژیکی، می‌توانیم یکی از این سلول‌های مجاور نقطه‌ی دید را داشته باشیم و با استفاده از اطلاعات مجاورتی سلول‌ها، سلول‌های دیگر را به‌دست آوریم. طبق فرض‌هایی که صورت گرفت، می‌توان نشان داد که برای n پاره‌خط، دقیقا $3n$ سلول وجود خواهد داشت. با استفاده از یک الگوریتم روبش زاویه‌ای حول نقطه‌ی دید، می‌توان نقشه‌ی توپولوژیکی را در زمان $O(n \log n)$ و فضای $O(n)$ ساخت [۳۶].

در ادامه روش استفاده از نقشه‌ی توپولوژیکی در محاسبه‌ی محلی قابلیت دید یک پاره‌خط و نگهداری آن در طی حرکت پاره‌خط را بیان خواهیم کرد.



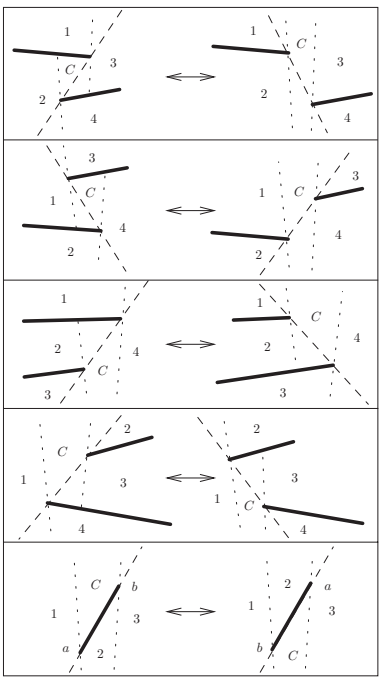
شکلِ ۷–۸: خط بحرانی ایجاد شده توسط سلول C در نقشه‌ی توپولوژیکی.

۷–۴–۲ قابلیت دید ایستا

هر سلول در نقشه‌ی توپولوژیکی دو راس مجاور دارند. اگر این دو راس را با یک خط به هم متصل کنیم، خط بحرانی که توسط سلول به وجود می‌آید را به دست می‌آوریم (شکل ۷–۸). چون تعداد سلول‌ها خطی است، تعداد خطوط بحرانی که به این شکل به دست می‌آید نیز خطی خواهد بود. این خطوط زیرمجموعه‌ای از همه‌ی خطوط بحرانی است که در گراف قابلیت دید به دست می‌آید.

حال مفهوم نقشه‌ی توپولوژیکی را برای یک ناظر قطعه‌ای ایستا بیان می‌کنیم. در ابتدا یک ناظر نقطه‌ای را روی یکی از رئوس پاره‌خط در نظر می‌گیریم و نقشه‌ی توپولوژیکی صحنه را نسبت به آن به دست می‌آوریم. با حرکت این ناظر نقطه‌ای در طول پاره‌خط و به سمت راس دیگر پاره‌خط، باید تغییرات نقشه‌ی توپولوژیک را مشخص کنیم. خوشبختانه فقط کافی است مواقعی که توپولوژی نقشه تغییر می‌کنند را به دست آوریم. این تغییر وقتی رخ می‌دهد که به تقاطع یک خط بحرانی با پاره‌خط ناظر برسیم. این برخورد با تغییر جهت دو رأس سلول تشکیل دهنده‌ی خط بحرانی مورد نظر نسبت به هم متناظر است. تعداد سلول‌های نقشه بعد از برخورد ثابت می‌ماند. در این رخداد توپولوژی سلول مورد نظر و سلول‌های مجاور آن تغییر می‌کنند و خطوط بحرانی به‌دست آمده از این سلول‌ها هم تغییر می‌کنند. تعداد حالت‌های تغییر یک سلول محدود می‌باشد و در شکل ۷–۹ آمده است.

با حذف شدن تعدادی از خطوط بحرانی نقشه و به وجود آمدن تعداد جدیدی از خطوط، مجموعه‌ی رخدادهای تغییر نقشه‌ی توپولوژیکی در طول ناظر قطعه‌ای هم تغییر خواهد کرد، به این ترتیب که اگر یکی از خطوط بحرانی که حذف شده است، قبل از حذف در نقطه‌ای بعد از نقطه‌ی رخداد فعلی با ناظر برخورد کرده باشد، رخداد متناظر این برخورد هم حذف خواهد شد. همچنین اگر یک خط بحرانی جدید در نقطه‌ای بعد از نقطه‌ی رخداد فعلی با ناظر برخورد کند، یک رخداد جدید متناظر این برخورد در صف رخدادها اضافه می‌شود. بدیهی است که چون تعداد سلول‌های تغییر یافته ثابت می‌باشد، تعداد این حذف و



شکلِ ۷–۹: تغییرات ممکن در نقشه‌ی توپولوژیکی در هنگام عبور از خط بحرانی سلول C. سلول‌های مجاور با عدد مشخص شده‌اند.

اضافه‌ی رخدادها هم ثابت خواهد بود و این حذف و اضافه از صف اولویت رخدادها در زمان $O(\log n)$ قابل پردازش است.

مسئله‌ی مورد نظر ما در اینجا محاسبه‌ی قابلیت دید ناظر پاره‌خط در حالت ایستا است. محاسبه‌ی نقشه‌ی توپولوژیکی و محاسبه‌ی قابلیت دید رأس شروع پاره‌خط در زمان $O(n \log n)$ و حافظه‌ی $O(n)$ قابل انجام است [۳۶]. ساختن لیست رخدادها بر اساس برخورد خطوط بحرانی با پاره‌خط هم به زمان $O(n \log n)$ احتیاج دارد. دقت کنید که n خط بحرانی وجود دارد و بررسی برخورد هر خط بحرانی با پاره‌خط در $O(1)$ قابل انجام است، با توجه به این که $O(n)$ رخداد ممکن را باید بر اساس زمان رخداد و یا ترتیب نقطه‌ای برخورد روی پاره‌خط مرتب کرد، به زمان $O(n \log n)$ می‌رسیم. تا وقتی که ساختار توپولوژیک نقشه تغییر نکند، قابلیت دید در طول پاره‌خط ناظر تغییر ساختاری نخواهد داشت. با رسیدن به یک رخداد، نقشه‌ی توپولوژیک و قابلیت دید ناشی از آن در $O(\log n)$ به‌روز می‌شوند. پس اگر k رخداد در طول پاره‌خط ناظر وجود داشته باشد، با توجه به هزینه‌ی پیش‌پردازش $O(n \log n)$، قابلیت دید ناظر پاره‌خطی را می‌توان در زمان $O((n+k) \log n)$ به دست آورد.

گزاره ۴ در صحنه‌ای شامل n شیء محدب، قابلیت دید از یک پاره‌خط را می‌توان در زمان $O((n+k) \log n)$ محاسبه کرد، که k تعداد خطوط بحرانی است که با پاره‌خط برخورد می‌کنند.

به این ترتیب در مقایسه با گزاره ۳، مرحله‌ی پیش‌پردازش را از دست داده‌ایم و احتیاجی به محاسبه‌ی مجتمع قابلیت دید صحنه نداریم.

۷-۴-۳ قابلیت دید پویا

در حالت پویا هم مانند روشی که قبلاً استفاده کردیم عمل می‌کنیم: مجموعه‌ی خطوط انفصال و ناظر پاره‌خطی را به فضای دوگان می‌بریم. تفاوتی که در این‌جا وجود دارد این است که مجموعه‌ی خطوط بحرانی که مورد بررسی قرار می‌گیرد، همه‌ی $O(n^2)$ خط بحرانی موجود در صحنه نیست، بلکه تنها خطوطی در نظر گرفته می‌شوند که در نقشه‌ی توپولوژیکی ناظر حضور داشته باشند. تعداد این خطوط $O(n+k)$ است که k تعداد برخورد ناظر در حالت اولیه با خطوط بحرانی است. مانند قبل رخدادهای گذر و تقاطع را به کمک وضعیت این خطوط بحرانی در فضای دوگان تشخیص می‌دهیم.

نکته‌ی مهمی که باید در نظر داشت، وضعیت نقشه‌ی توپولوژیکی ناظر قطعه‌ای بعد از به وجود آمدن رخدادهای گذر و تقاطع است. با هر رخداد گذر، یک نقطه به محل تغییر نقشه‌ی توپولوژیکی روی ناظر افزوده می‌شود. در یک رخداد تقاطع، ترتیب دو نقطه‌ی تغییر نقشه‌ی توپولوژیکی بر روی پاره‌خط ناظر تغییر می‌کند.

برای یادآوری مفهوم نقشه‌ی توپولوژیکی ناظر قطعه‌ای به این شکل است که نقشه‌ی توپولوژیکی یکی از رأس‌های پاره‌خط را محاسبه می‌کنیم و ضمن حرکت به سمت رأس دیگر در طول پاره‌خط، هرجا پاره‌خط با خط بحرانی برخورد کرده باشد و سلول‌های متناظر آن خط بحرانی در نقشه‌ی توپولوژیکی تغییر کند، این تغییر را محاسبه و نگهداری می‌کنیم. پس تعداد و ترتیب نقاط برخورد پاره‌خط با خطوط بحرانی برای تعیین نقشه‌ی توپولوژیکی پاره‌خط اهمیت دارد. همانطور که گفتیم، در رخداد گذر، به این تقاطع‌ها اضافه و یا از آن کم می‌شود، درحالی‌که در رخداد تقاطع، ترتیب دو نقطه تقاطع مجاور تغییر می‌کند. بدیهی است که تغییر ترتیب برخورد هم به تغییر توپولوژی نقشه‌ی توپولوژیکی در طول پاره‌خط می‌انجامد.

حال به بررسی پیچیدگی پردازش رخدادهای گذر و تقاطع در این حالت می‌پردازیم. بعد از محاسبه‌ی قابلیت دید پاره‌خط در حالت اولیه، در هر رخداد گذر یا تقاطع تنها کافی است که ساختار نقشه‌ی توپولوژیکی در آن نقطه به‌روز گردد و تغییر قابلیت دید متناظر با آن محاسبه شود. همان‌طور که بیان شد به‌روزرسانی نقشه‌ی توپولوژیکی را می‌توان در زمان $O(\log n)$ انجام داد. همین زمان هم برای به‌روزرسانی قابلیت دید در آن نقطه احتیاج است.

۷-۵ خلاصه

در این فصل به بررسی قابلیت دید یک پاره‌خط در میان تعدادی شیء مجزا در صفحه پرداختیم. نشان دادیم که چگونه با استفاده از الگوریتم‌های محاسبه‌ی قابلیت دید نقطه، این مسئله را در حالت ایستا حل کنیم. سپس پاره‌خط در حال حرکت را در نظر گرفتیم و رخدادهای ایجاد شده در حین حرکت پاره‌خط را بررسی کردیم و نشان دادیم که چگونه در زمان مناسب این رخدادها را پردازش کرده و قابلیت دید پاره‌خط

را نگهداری کنیم.

فصلِ ۸

نتیجه‌گیری

۸-۱ نتایج به دست آمده

در این پژوهش به بررسی مسائل قابلیت دید پاره خط پرداختیم و الگوریتم‌های مختلفی را برای مسائل مطرح شده ارائه کردیم. در فصل‌های ۳ و ۴ مسئله‌ی محاسبه‌ی چندضلعی قابل دید پاره‌خط در چندضلعی‌های ساده را مورد بررسی قرار دادیم و الگوریتم‌هایی برای محاسبه‌ی حساس به خروجی آن ارائه کردیم. در فصل ۵، همین مسئله را برای چندضلعی‌های حفره‌دار در نظر گرفته و الگوریتمی حساس به خروجی برای محاسبه‌ی آن به دست آوردیم.

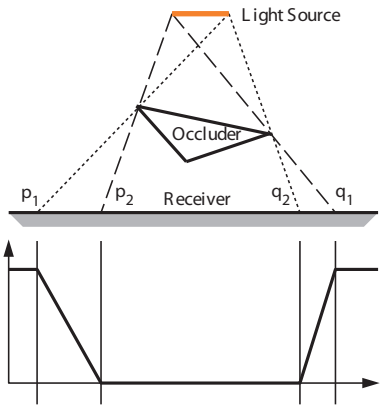
در ادامه کار، مسئله محاسبه‌ی اندازه‌ی چندضلعی قابل دید پاره‌خط را در فصل ۶ مطرح کردیم و الگوریتم‌های متفاوتی را برای محاسبه‌ی آن ارائه دادیم. در نهایت، در فصل ۷ به بررسی قابلیت دید پاره‌خط متحرک پرداختیم و انواع رخدادهای قابلیت دید را بررسی نمودیم.

۸-۲ پژوهش‌های آینده و مسائل باز

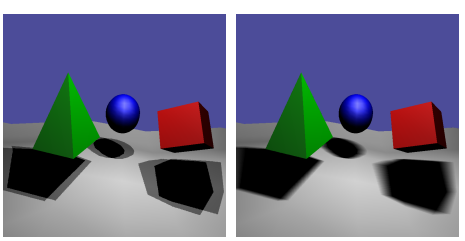
آن‌چه پیش از این در این گزارش ملاحضه شد، خلاصه‌ای از مطالعات انجام شده در ضمن تحقیقات و نیز کارهایی که تا کنون انجام شده است بود. در این فصل به معرفی زمینه‌های مناسب برای ادامه‌ی پژوهش می‌پردازیم.

قابلیت دید آلفا

گام مهم بعدی در ادامه‌ی این پژوهش، یافتن درجه‌ی پدیداری ناظر پاره‌خطی برای هر نقطه‌ی صفحه می‌باشد. همان‌طور که قبلا بیان شد، تعاریف گذشته‌ی پدیداری پاره‌خط دیده‌شدن یا دیده‌نشدن ناظر پاره‌خطی را محاسبه می‌کنند. هدف ما در این قسمت نسبت دادن عددی بین ۰ و ۱ به هر نقطه است که مشخص کننده‌ی میزان پدیداری آن نسبت به ناظر نقطه‌ای می‌باشد.



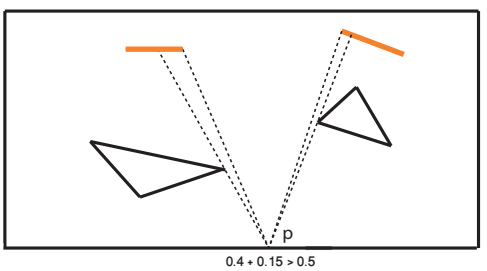
شکلِ ۸-۱: قابلیت دید آلفا: هر نقطه از صفحه بخشی از منبع نور را می‌بیند و بر همین اساس درجه‌ی روشنایی آن از روشنایی کامل ۱ تا تاریکی کامل ۰ تغییر می‌کند.



شکلِ ۸-۲: محاسبه‌ی بهینه‌ی سایه‌های نرم یکی از مسائل مهم در گرافیک کامپیوتری می‌باشد.

محاسبه‌ی سایه و نیم‌سایه

محاسبه‌ی بهینه و پایدار سایه‌ها هنوز یکی از چالش‌های موجود در گرافیک کامپیوتری است. سایه‌ها در ترسیم یک صحنه‌ی سه‌بعدی نقش پر اهمیتی دارند و نقش زیادی در درک موقعیت اشیاء دارند (شکل ۸-۲ را ببینید). سایه‌ی نرم توسط منبع خطی یا ناحیه‌ای ایجاد می‌گردد. نقاط در صحنه می‌توانند همه، بخشی و یا هیچ قسمت از یک چنین منبعی را ببینند، که به ترتیب بخش نور کامل، نیم سایه و یا سایه را تشکیل می‌دهند. اندازه‌ی ناحیه‌ی نیم سایه وابسته به فاصله بین منبع، مسدود کننده و شیء نورانی شده تغییر می‌کند. محاسبه‌ی سایه و نیم‌سایه برای محیط‌های مختلف، مبحث بسیار مهم و کاربردی در گرافیک کامپیوتری است که هر بهبودی در انجام محاسبات مربوطه، تأثیر مستقیمی بر سرعت و کیفیت نتیجه‌ی نهایی دارد.



شکلِ ۸–۳: موزه‌ی هنر با وجود نگهبان‌های پاره‌خطی. یک نقطه توسط نگهبان‌ها کنترل می‌شود چنانچه حداقل بخشی از نگهبان‌ها توسط آن دیده شود.

قابلیت دید در صورت وجود چند ناظر پاره‌خطی

مسئله‌ی پیچیده‌تر وقتی است که فرض کنیم بیش از یک منبع نور پاره‌خطی (ثابت و متحرک) در صفحه موجود است. در این حالت، محاسبه‌ی پدیداری جزئی نقاط صفحه، در محاسبه‌ی نیم‌سایه‌ها کمک می‌کند.

مسئله‌ی موزه‌ی هنر با وجود نگهبان‌های پاره‌خطی

مسئله‌ی موزه‌ی هنر همیشه وابستگی زیادی به مسئله‌ی قابلیت دید داشته است. در صورت کلاسیک این مسئله، یک یا تعدادی نگهبان نقطه‌ای را باید طوری در یک موزه قرار دهیم که همه‌ی نقاط موزه را برای جلوگیری از سرقت پوشش دهند و ببینند. می‌توان مسئله‌ی موزه‌ی نگهبان را در حالتی که نگهبان‌ها (سنسورها)یی به صورت پاره‌خطی وجود دارند را در نظر گرفت، با این فرض که یک نقطه توسط نگهبان‌ها پوشش داده می‌شود اگر حداقل توسط درصدی از سطح نگهبان‌ها دیده شود. این مسئله وابستگی زیادی به مسئله‌ی قابلیت دید آلفا که در بالا مطرح کردیم دارد.

کتاب‌نامه

[1] S. Alipour, A. Zarei. Visibility Testing and Counting. In *Proceedings of FAW-AAIM*, pages 343–351 , 2011.

[2] B. Aronov, L Guibas, M Teichmann, and L. Zhang. Visibility queries and maintenance in simple polygons. *Discrete and Computational Geometry*, 27(4):461–483, 2002.

[3] T. Asano, S. Ghosh, and T. Shermer. Visibility in plane. *In Handbook in Computational Geometry*. Elsevier Science, 1999.

[4] I.J. Balaban. An optimal algorithm for finding segment intersections. In *Proceedings of the 11th Annual ACM Symposium of Computational Geometry*, pages 211–219. 1995.

[5] J. Bittner. Visibility in Computer Graphics. In *Environment and Planning B: Planning and Design*, pages 729-756, 2003.

[6] P. Bose, A. Lubiw and J. I. Munro. Efficient visibility queries in simple polygons. *Computational Geometry: Theory and Applications*, 23(3):313–335, 2002

[7] B. Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 339–349, 1982.

[8] B. Chazelle and L.J. Guibas. Visibility and intersection problems in plane geometry, *Report CS-TR-167-88*, Princeton University, 1988.

[9] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. *Discrete and Computational Geometry*, 4(6):551–581, 1989.

[10] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485–524, 1991.

[11] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, 8(5&6), 407–429, 1992.

[12] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete and Computational Geometry*, 9:145–158, 1993.

[13] D. Z. Chen and H. Wang. Weak visibility queries of line segments in simple polygons. In *Proceedings of the 23rd International Symposium on Algorithms and Computation*, pages 609–618, 2012.

[14] D. Z. Chen and H. Wang. Computing the visibility polygon of an island in a polygonal domain. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming*, pages 218–229, 2012.

[15] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational Geometry: Algorithms and Applications. *Springer-Verlag*, Heidelberg, Germany, 2nd edition, 2000.

[16] D.P. Dobkin and H. Edelsbrunner. Space searching for intersecting objects. *Journal of Algorithms*, 8(3):348–361 1987.

[17] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. In *Computer Graphics (Proceedings of SIGGRAPH*, pages 89-100, 1997.

[18] H. ElGindy and D. Avis. A Linear Algorithm for Computing the Visibility Polygon from a Point, *Journal of Algorithms*, 2:186–197, 1981.

[19] H. ElGindy. Efficient algorithms for computing the weak visibility polygon from an edge. *Technical Report MS-CIS-86-04*, University of Pennsylvania, Philadelphia, USA, 1986.

[20] M. Fischer, M. Hilbig, C. Jahn, F. Meyer auf der Heide, and M. Ziegler. Planar visibility counting. *CoRR, abs/0810.0052*, 2008.

[21] M. Fischer, M. Hilbig, C. Jahn, F. Meyer auf der Heide, and M. Ziegler. Planar visibility counting. In *Proceedings of European Workshop on Computational Geometry*, pages 203–206, 2009.

[22] S. Ghali and A. J. Stewart. Maintenance of the set of segment visible from a moving viewpoint in two dimensions. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages V3-V4, 1996.

[23] S.K. Ghosh and D.M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20:888–910, 1991.

[24] S. K. Ghosh. Visibility Algorithms in the Plane. *Cambridge University Press*, New York, NY, USA, 2007.

[25] J. Gudmundsson and P. Morin. Planar visibility: testing and counting. In *Proceedings of Annual Symposium on Computational Geometry*, pages 77–86, 2010.

[26] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. In *Proc. Second Annual ACM Symp. on Computational Geometry*, pp. 1–13, 1986.

[27] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[28] L. Guibas, R. Motwani, and P. Raghavan, The Robot Localization Problem in two Dimensions, *SIAM J. of Computing*, 26(4):1120–1138, 1997.

[29] P.S. Heckbert. Discontinuity meshing for radiosity. In *Third Eurographics Workshop on Rendering*, pages 203-216, Bristol, UK, 1992.

[30] J. Hershberger and S.Suri. A pedestrian approach to ray shooting: shoot a ray, take a walk. *J. Algorithms*, 18:403-431, 1995.

[31] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[32] D. T. Lee and A. K. Lin. Computing the visibility polygon from an edge. *Computer Vision, Graphics, and Image Processing*, 34:1–19, 1986.

[33] M. McKenna. Worst-case optimal hidden surface removal. In *ACM Transaction on Graphics*, pages 19–28, vol.6, 1987.

[34] A. Margalit and G.D. Knott. An algorithm for computing the union, intersection or difference of two polygons. *Computation and Graphics*, 13:167–183, 1989.

[35] M. Nouri and M. Ghodsi. Space/query-time trade-off for computing the visibility polygon. *Computational Geometry: Theory and Applications*. 46(3), 371–381, 2013.

[36] K. Nechvile, P. Tobola. Local approach to dynamic visibility in the place. In *Proceedings of the 7th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media'99*, pages 202–208, 1999.

[37] M. Nouri Bygi and M. Ghodsi. Weak visibility queries in simple polygons. In *Proceedings of the 23rd Canadian Conference of Computational Geometry,* 2011.

[38] M. Nouri Bygi and M. Ghodsi. Weak visibility queries of line segments in simple polygons and polygonal domains. *CoRR*, abs/1310.7197, 2013.

[39] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. In *J. Comput. Syst. Sci.*, 23:166-204, 1981.

[40] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29:669–679, 1986.

[41] S. Rivière. Dynamic visibility in polygonal scenes with the visibility complex. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 421-423, 1997.

[42] S. Suri and J. O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proceedings of Annual Symposium on Computational Geometry*, pages 14–23, 1984.

[43] G. T. Toussainta. A linear-time algorithm for solving the strong hidden-line problem in a simple polygon. *Pattern Recognition Letters*, 4:449–451, 1986.

[44] A. Zarei, M. Ghodsi. Efficient Computation of Query Point Visbility in Polygons with Holes. In *Proceedings of the Annual Symposium on Computational Geometry*, 2005.

[45] A. Zarei and M. Ghodsi. Query point visibility computation in polygons with holes. *Computational Geometry: Theory and Applications*, 39(2):78–90, 2008.

Weak Visibility of Line Segments in Different Environments

Abstract

Visibility is an important topic in computational geometry, computer graphics, and motion planning. To deal with the increasing complexity of the scenes considered, some research has been performed in visibility processing in order to accelerate the visibility determination.

Two points inside a polygon are visible to each other if their connecting segment remains completely inside the polygon. Visibility polygon of a point in a simple polygon $\mathcal{P}$ is the set of points inside $\mathcal{P}$ that are visible from the point. The visibility problem has also been considered for line segments. A point v is said to be weakly visible to a line segment pq if there exists a point $w \in pq$, such that w and v are visible to each other. The problem of computing the weak visibility polygon (or WVP) of pq inside a polygon $\mathcal{P}$ is to compute all points of $\mathcal{P}$ that are weakly visible from pq.

In this thesis we consider the problem of computing weak visibility of a line segment inside various environments. The main problems we focused on are as follows: computiong the weak visibility polygon of a line segment in simple polygons, computing the weak visibility polygon of a line segment in polygonal domains, computing the size of the weak visibility of a line segment in simple polygons, and maitaining the visibility of a moving line segment in polygonal domains.

Keywords: *Computational Geometry, Visibility problems, Visibility polygon, Weak visibility, Visibility counting.*

Weak Visibility of Line Segments in Different Environments

by

Mojtaba Nouri Bygi

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Ph.D
in
Computer Engineering (Software)

Under supervision of
Prof. Mohammad Ghodsi

January 2014

Computer Engineering Department

Sharif University of Technology

Tehran