

# فصل ۳۳

## هندسه محاسباتی

هندسه محاسباتی، رشته‌ای از علم کامپیوتر است که الگوریتم‌های مربوط به حل مسئله‌های هندسی را مطالعه می‌کند. در مهندسی و ریاضیات مدرن، هندسه محاسباتی کاربردهایی در گرافیک کامپیوتری، روباتیک، طراحی VLSI، طراحی به کمک کامپیوتر، و آمار دارد. ورودی مسئله‌ی هندسه محاسباتی، معمولاً توصیف مجموعه‌ای از اشیای هندسی است، مثل مجموعه‌ای از نقاط، مجموعه‌ای از پاره خط‌ها<sup>۱</sup>، یا رئوس چندضلعی در جهت خلاف عقربه‌های ساعت. خروجی معمولاً پاسخ به درخواستی راجع به اشیا است، مثل این‌که آیا خطوطی متقطع‌اند، یا شیء هندسی جدیدی است، مثل پوش محدب<sup>۲</sup> (کوچکترین چندضلعی محدب در برگیرنده) مجموعه‌ای از نقاط.

در این فصل، چند الگوریتم هندسه محاسباتی را در صفحه (دوبعدی) در نظر می‌گیریم. هر شیء ورودی، به صورت مجموعه‌ای از نقاط  $\{p_1, p_2, p_3, \dots\}$  نمایش داده می‌شود، که هر  $(x_i, y_i) = p_i$  و  $x_i, y_i \in \mathbf{R}$ . به عنوان مثال، چندضلعی  $n$  رأسی  $P$  به صورت دنباله  $\langle p_0, p_1, p_2, \dots, p_{n-1} \rangle$  از رئوس به ترتیب ظاهرشدن در مرز  $P$ ، نمایش داده می‌شود. هندسه محاسباتی می‌تواند در حالت سه‌بعدی و بالاتر نیز انجام شود، اما تصویر این مسئله‌ها و راه حل‌های آن‌ها بسیار دشوار است. حتی در حالت دوبعدی نیز می‌توان نمونه خوبی از تکنیک‌های هندسه محاسباتی را مشاهده کرد.

بخش ۳۳-۱ به پرسش‌های اساسی در زمینه پاره خط‌ها پاسخ می‌دهد: آیا یک پاره خط در جهت عقربه‌های ساعت یا خلاف عقربه‌های ساعت نسبت به دیگری که در یک نقطه مشترک‌اند قرار دارد، هنگام پیمایش دو پاره خط متصل، کدام را ادامه دهیم، و آیا دو پاره خط یکدیگر را قطع می‌کنند یا خیر. بخش ۳۲-۲ تکنیکی به نام "جاروکردن"<sup>۳</sup> را ارائه می‌کند که از آن برای توسعه الگوریتم‌های زمان  $O(n \lg n)$ ، در تعیین متقطع‌بودن مجموعه‌ای از  $n$  پاره خط استفاده می‌شود. بخش ۳۳-۲ دو الگوریتم "جاروکردن دورانی"<sup>۴</sup> را ارائه می‌کند که پوش محدب (کوچکترین چندضلعی محدب در برگیرنده) مجموعه‌ای از  $n$  نقطه را محاسبه می‌کند: پیمایش Graham، که در زمان  $O(n \lg n)$  اجرا می‌شود، و Jarvis's march که در زمان  $O(nh)$  اجرا می‌شود و  $h$  تعداد رئوس پوش محدب است. سرانجام، بخش ۳۳-۴ الگوریتم تقسیم و حل در زمان  $O(n \lg n)$  را برای یافتن نزدیک‌ترین جفت از نقاط در مجموعه‌ای از  $n$  نقطه‌ی موجود در صفحه ارائه می‌کند.

1. line segment

2. convex hull

3. sweeping

4. rotational sweep

## ۳۳-۱ خواص پاره خط

چندین الگوریتم هندسه محاسباتی در این فصل، نیاز به پاسخ به پرسش‌هایی در مورد خواص پاره خط‌ها دارد. نوکیب محدب<sup>۱</sup> دو نقطه‌ی جدا از هم  $(x_1, y_1) = p_1$  و  $(x_2, y_2) = p_2$ ، نقطه‌ای مثل  $p_3 = (x_3, y_3)$  است به طوری که برای یک مقدار  $\alpha$  در بازه‌ی  $0 \leq \alpha \leq 1$ ، داریم  $p_3 = (x_3, y_3) = \alpha x_1 + (1-\alpha)x_2$  و  $x_3 = \alpha x_1 + (1-\alpha)x_2$ .  $y_3 = \alpha y_1 + (1-\alpha)y_2$ . همچنین می‌نویسیم  $p_3 = \alpha p_1 + (1-\alpha)p_2$ . از نظر شهودی،  $p_3$  نقطه‌ای است که روی خط قرار دارد که از  $p_1$  و  $p_2$  عبور می‌کند و روی یا بین  $p_1$  و  $p_2$  روی خط قرار دارد. با توجه به دو نقطه‌ی جدا از هم  $p_1$  و  $p_2$ ، مجموعه‌ای از ترکیب‌های محدب  $p_1$  و  $p_2$  است.  $p_1$  و  $p_2$  را نقاط پایانی<sup>۲</sup> پاره خط  $\overrightarrow{p_1p_2}$  می‌نامیم. گاهی ترتیب  $p_1$  و  $p_2$  مهم است، و راجع به پاره خط جهت دار<sup>۳</sup>  $\overrightarrow{p_1p_2}$  صحبت می‌کنیم. اگر  $p_1$  مبدأ (باشد)، آنگاه با پاره خط جهت دار  $\overrightarrow{p_1p_2}$  مثل بردار  $p_2$  رفتار می‌شود.

در این بخش، پرسش‌های زیر را بررسی خواهیم کرد:

۱. با توجه به دو پاره خط جهت دار  $\overrightarrow{p_0p_1}$  و  $\overrightarrow{p_0p_2}$ ، آیا  $\overrightarrow{p_0p_1}$  از  $\overrightarrow{p_0p_2}$  نسبت به نقطه انتهای مشترک  $p_0$ ، در جهت عقربه‌های ساعت است یا خیر.

۲. با توجه به دو پاره خط  $\overrightarrow{p_0p_1}$  و  $\overrightarrow{p_1p_2}$ ، اگر ابتدا  $\overrightarrow{p_0p_1}$  و سپس  $\overrightarrow{p_1p_2}$  را پیمایش کنیم، آیا چپ گرد<sup>۴</sup> را در نقطه  $p_1$  انجام دادیم؟

۳. آیا پاره خط‌های  $\overrightarrow{p_1p_2}$  و  $\overrightarrow{p_3p_4}$  متقطع‌اند؟

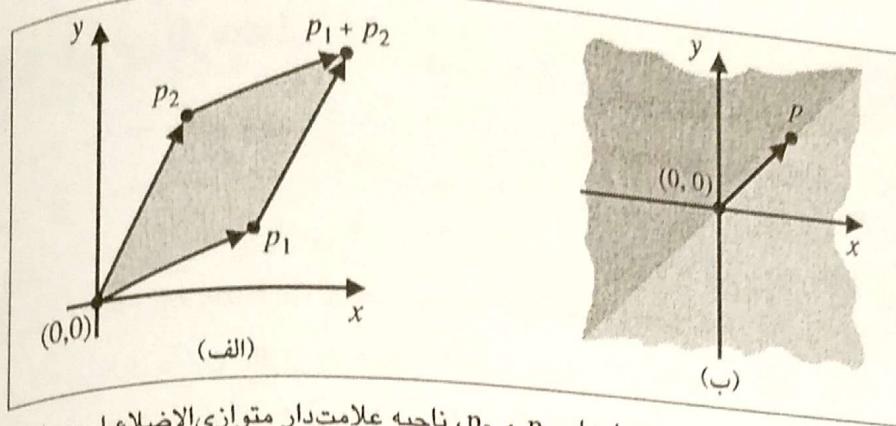
هرچند محدودیتی در نقاط ارائه شده وجود ندارد.

هر پرسش را می‌توان در زمان (۱) O جواب داد، که تعجبی ندارد، زیرا اندازه ورودی هر پرسش، (۱) O است. علاوه‌بر این، روش ما فقط از مجموع، تفریق، ضرب و مقایسه استفاده می‌کند. به توابع مثلثاتی و تقسیم نیاز نداریم، که هر یک از این عملیات گران و مولد خطای گردیدن هستند. به عنوان مثال، روش "ساده‌ی" تعیین متقطع‌بودن دو پاره خط، برای یافتن نقطه تقاطع، از تقسیم استفاده می‌کند. این روش، معادله خط  $y = mx + b$  را برای هر پاره خط تعیین می‌کند ( $m$  شیب و  $b$  عرض از مبدأ نام دارد)، نقطه تقاطع خطوط را پیدا می‌کند، و بررسی می‌کند که آیا این نقطه روی هر دو پاره خط وجود دارد یا خیر. وقتی پاره خط‌ها تقریباً موازی باشند، این روش، نسبت به دقت عملیات تقسیم روش کامپیوترهای حقیقی حساس است. روش استفاده شده در این بخش که از تقسیم اجتناب نمی‌کند، دقیق‌تر است.

### حاصلضرب خارجی

محاسبه حاصلضرب خارجی، در قلب روش پاره خط موردنظر ما وجود ندارد. بردارهای  $p_1$  و  $p_2$  را در نظر بگیرید که در شکل ۳۳-۱ (الف) نشان داده شدند. حاصلضرب خارجی<sup>۵</sup>  $p_1 \times p_2$  را می‌توان ناحیه علامت دار متوازی‌الاضلاعی دانست که توسط نقاط  $(0, 0)$ ،  $p_1$ ،  $p_2$  و  $(x_1 + x_2, y_1 + y_2)$  ایجاد شد. تعریف معادل ولی مفیدتر دیگر، حاصلضرب خارجی را به صورت دترمینان ماتریس<sup>۶</sup> ارائه می‌کند:

convex combination	2. end point	3. directed segment	4. left turn	5. cross product
$ x_1y_2 - x_2y_1 $ است. در این فصل، ثابت می‌شود که بهتر است حاصلضرب خارجی به صورت $x_1y_2 - x_2y_1$ استفاده شود.				



شکل ۱ ۳۳-۱ (الف) حاصلضرب داخلی بردارهای  $p_1$  و  $p_2$ ، ناحیه علامت‌دار متوازی‌الاضلاع است. (ب) ناحیه سایه‌دار کمرنگ شامل بردارهایی است که از  $p$  در جهت عقربه‌های ساعت قرار دارند. ناحیه سایه‌دار پررنگ شامل بردارهایی است که از  $p$  در جهت خلاف عقربه‌های ساعت قرار دارند.

$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -p_2 \times p_1 \end{aligned}$$

اگر  $p_1 \times p_2$  مثبت باشد، آنگاه  $p_1$  و  $p_2$  نسبت به تقاطع مبدأ  $(0, 0)$  در جهت عقربه ساعت هستند؛ اگر این حاصلضرب خارجی منفی باشد، آنگاه  $p_1$  از  $p_2$  در خلاف جهت عقربه‌های ساعت است (تمرین ۱-۳۱-۱ را ببینید). شکل ۱ ۳۳-۱ (ب) ناحیه‌های در جهت عقربه‌های ساعت و خلاف جهت عقربه‌های ساعت را نسبت به بردار  $p$  نشان می‌دهد. اگر حاصلضرب خارجی صفر باشد، شرط مرزی<sup>۱</sup> به وجود می‌آید. در این حالت، بردارها هم‌راستا<sup>۲</sup> هستند، که به یک جهت یا جهت مخالف اشاره دارند.

برای تعیین این که آیا پاره خط جهت دار  $\overrightarrow{P_0P_1}$  از پاره خط جهت دار  $\overrightarrow{P_0P_2}$  نسبت به نقطه پایانی مشترک  $P_0$  در جهت عقربه‌های ساعت قرار دارد یا خیر، با انجام جابه‌جایی، از  $P_0$  به عنوان مبدأ استفاده می‌کنیم. یعنی،  $P_0 - P_1 - P_2$  نشان‌دهنده بردار  $(x'_1, y'_1) = p'_1$  است، که  $x'_1 = x_1 - x_0$  و  $y'_1 = y_1 - y_0$  و  $P_0 - P_2$  را به همین صورت تعریف می‌کنیم. سپس حاصلضرب داخلی را محاسبه می‌کنیم:

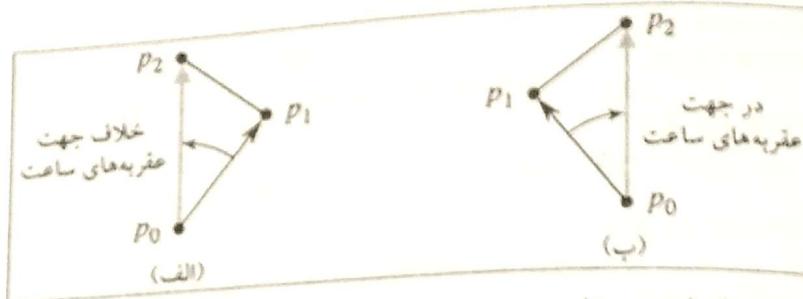
$$(P_0 - P_1) \times (P_0 - P_2) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

اگر این حاصلضرب داخلی مثبت باشد، آنگاه  $\overrightarrow{P_0P_1}$  از  $\overrightarrow{P_0P_2}$  در جهت عقربه‌های ساعت قرار دارد؛ اگر منفی باشد، در خلاف جهت عقربه‌های ساعت واقع است.

### تعیین این که پاره خط‌های متواالی چپ‌گرد یا راست‌گرد هستند

پرسش بعدی این است که آیا دو پاره خط متواالی  $\overrightarrow{P_0P_1}$  و  $\overrightarrow{P_1P_2}$  در نقطه  $P_1$  چپ‌گرد<sup>۳</sup> یا راست‌گرد<sup>۴</sup> هستند. به عبارت دیگر، به روشنی نیاز داریم که تعیین کند زاویه  $\angle P_0P_1P_2$  به کدام طرف می‌چرخد. با استفاده از حاصلضرب‌های داخلی، می‌توان بدون محاسبه زاویه، به این پرسش، پاسخ داد. همان‌طور که در شکل ۳۳-۲ آمده است، بررسی می‌کنیم که آیا پاره خط جهت دار  $\overrightarrow{P_0P_2}$  نسبت به پاره خط جهت دار  $\overrightarrow{P_0P_1}$  در جهت یا خلاف جهت عقربه‌های ساعت قرار دارد. برای این کار، حاصل ضرب خارجی  $(P_1 - P_0) \times (P_2 - P_0)$  را محاسبه می‌کنیم. اگر علامت این

<sup>1</sup>. boundary condition    2. collinear    3. turn left    4. turn right



شکل ۳۳-۲ استفاده از حاصل ضرب خارجی برای تعیین چرخش پاره خط های متوازی  $p_1p_2$  و  $p_0p_1$  در نقطه  $p_1$ . بررسی می کنیم که آیا پاره خط جهت دار  $\overrightarrow{p_0p_2}$  نسبت به پاره خط جهت دار  $\overrightarrow{p_0p_1}$  در جهت با خلاف جهت عقربه های ساعت قرار دارد. (الف) اگر در خلاف جهت عقربه های ساعت باشد، نقاط چپگرد هستند. (ب) اگر در جهت عقربه های ساعت باشد نقاط راستگرد هستند.

حاصل ضرب خارجی منفی باشد، آنگاه  $\overrightarrow{p_0p_2}$  نسبت به  $\overrightarrow{p_0p_1}$  در خلاف جهت عقربه های ساعت قرار دارد، و در نتیجه، در  $p_1$  چپ گرد داریم. حاصل ضرب خارجی مثبت، نشان دهنده جهت عقربه های ساعت و راست گرد است. حاصل ضرب خارجی صفر به معنای این است که نقاط  $p_0$ ,  $p_1$  و  $p_2$  در یک راستا قرار دارند.

### تعیین متقاطع بودن دو پاره خط

برای این که تعیین کنیم آیا دو پاره خط متقاطع اند یا خیر، بررسی می کنیم که آیا هر خط، خط حاوی پاره خط دیگر را در میان می گیرد یا خیر. پاره خط  $\overrightarrow{p_1p_2}$  در صورتی یک خط را در میان می گیرد که نقطه  $p_1$  در یک طرف خط و نقطه  $p_2$  در طرف دیگر خط قرار داشته باشد. اگر  $p_1$  یا  $p_2$  مستقیماً روی خط باشند، یک حالت مرزی پیش می آید. دو پاره خط متقاطع اند اگر و فقط اگر یکی (یا هر دو) شرط زیر برقرار باشد:

۱. هر پاره خط، خط حاوی دیگری را در میان می گیرد.
  ۲. نقطه پایانی یک پاره خط روی پاره خط دیگر باشد (این شرط از حالت مرزی ناشی می شود).
- رویه زیر، این ایده را پیاده سازی می کند. رویه SEGMENTS-INTERSECT در صورتی TRUE را بر می گرداند که پاره خط های  $\overrightarrow{p_1p_2}$  و  $\overrightarrow{p_3p_4}$  متقاطع باشند، وگرنه FALSE را بر می گرداند. این رویه، زیرروال DIRECTION را فراخوانی می کند که چرخش های نسبی را با استفاده از روش حاصل ضرب خارجی مطرح شده محاسبه می کند، و از زیرروال ON-SEGMENT استفاده می کند که تعیین می کند آیا نقطه ای که هم راستای یک پاره خط

است، روی آن پاره خط قرار می گیرد یا خیر:

```

SEGMENTS-INTERSECT( $p_1, p_2, p_3, p_4$ )
1  $d_1 \leftarrow \text{DIRECTION}(p_3, p_4, p_1)$ 
2  $d_2 \leftarrow \text{DIRECTION}(p_3, p_4, p_2)$ 
3  $d_3 \leftarrow \text{DIRECTION}(p_1, p_2, p_3)$ 
4  $d_4 \leftarrow \text{DIRECTION}(p_1, p_2, p_4)$ 
5 if  $((d_1 > 0 \text{ and } d_2 < 0) \text{ or } (d_1 < 0 \text{ and } d_2 > 0)) \text{ and}$ 
    $((d_3 > 0 \text{ and } d_4 < 0) \text{ or } (d_3 < 0 \text{ and } d_4 > 0))$ 
6   then return TRUE
7 elseif  $d_1 = 0$  and ON-SEGMENT( $p_3, p_4, p_1$ )
8   then return TRUE
9 elseif  $d_2 = 0$  and ON-SEGMENT( $p_3, p_4, p_2$ )
10  then return TRUE
11 elseif  $d_3 = 0$  and ON-SEGMENT( $p_1, p_2, p_3$ )
12  then return TRUE
13 elseif  $d_4 = 0$  and ON-SEGMENT( $p_1, p_2, p_4$ )

```

```

14   then return TRUE
15 else return FALSE

DIRECTION( $p_i, p_j, p_k$ )
1 return  $(p_k - p_i) \times (p_j - p_i)$ 

ON-SEGMENT( $p_i, p_j, p_k$ )
1 if  $\min(x_i, x_j) \leq x_k \leq \max(x_i, x_j)$  and  $\min(y_i, y_j) \leq y_k \leq \max(y_i, y_j)$ 
2 then return TRUE
3 else return FALSE

```

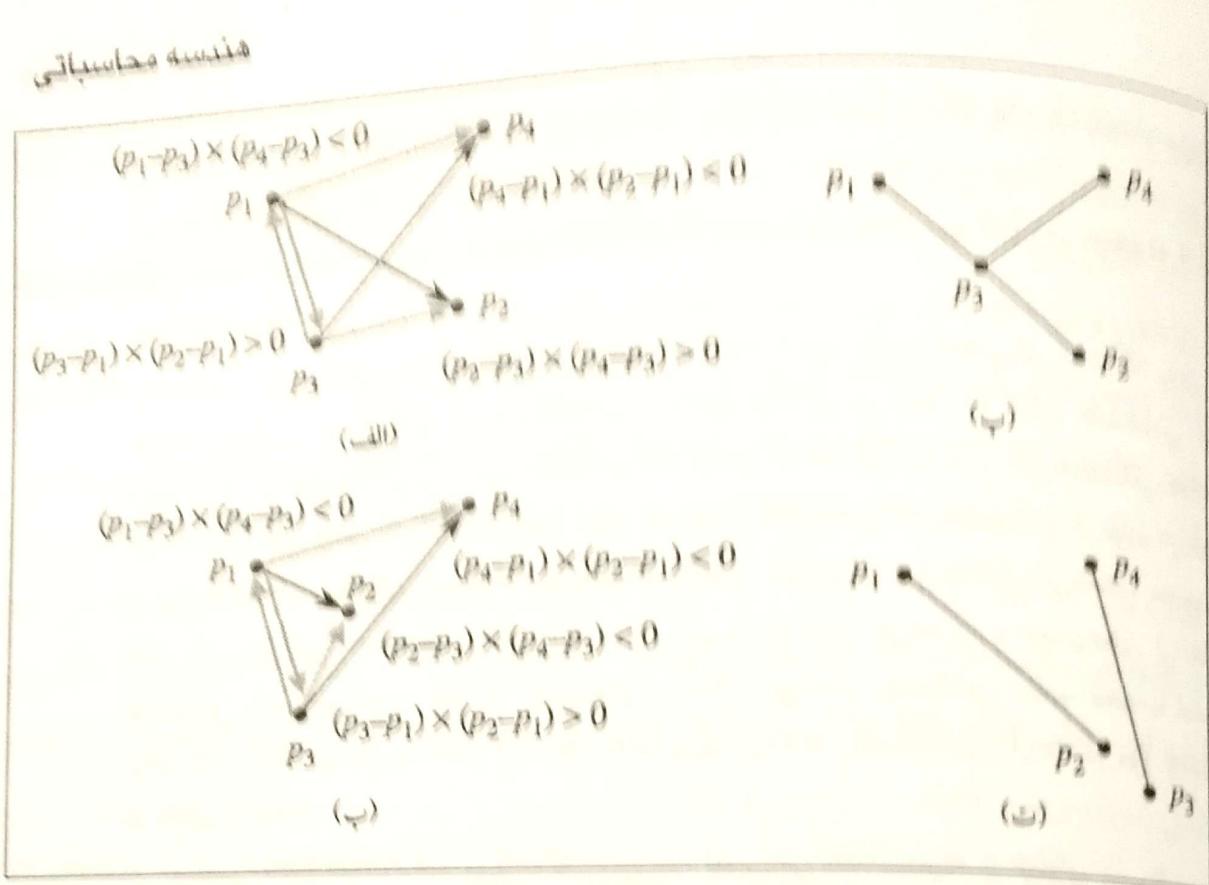
SEGMENTS-INTERSECT به صورت زیر کار می‌کند. خطوط ۱ تا ۴، جهت نسبی  $d_i$  هر نقطه پایانی  $p_i$  را نسبت به پارهخط دیگر محاسبه می‌کند. اگر تمام جهت‌های نسبی غیرصفر باشند، آنگاه به آسانی می‌توان تعیین کرد که آیا پارهخط‌های  $\overline{p_1p_2}$  و  $\overline{p_3p_4}$  متقاطع‌اند یا خیر. پارهخط  $\overline{p_1p_2}$ ، خط حاوی پارهخط  $\overline{p_3p_4}$  را در صورتی درمیان می‌گیرد<sup>۱</sup> که که پارهخط‌های جهت‌دار  $\overrightarrow{p_3p_1}$  و  $\overrightarrow{p_3p_2}$  نسبت به  $\overrightarrow{p_3p_4}$  جهت مخالف داشته باشند. در این حالت، علامت‌های  $d_1$  و  $d_2$  متفاوت است. به طور مشابه،  $\overline{p_3p_4}$  در صورتی خط حاوی  $\overline{p_1p_2}$  را در میان می‌گیرد که علامت‌های  $d_3$  و  $d_4$  متفاوت باشند. اگر تست‌های خط ۵ درست باشد، آنگاه پارهخط‌ها یکدیگر را در میان می‌گیرند، و SEGMENTS-INTERSECT مقدار TRUE را برمی‌گرداند. شکل ۳۳-۳ (الف) این حالت را نشان می‌دهد. و گرنه، پارهخط‌ها خطوط یکدیگر را در میان نمی‌گیرند، گرچه ممکن است حالت مرزی اعمال شود. اگر تمام چرخش‌های نسبی غیرصفر باشند، حالت مرزی اعمال نمی‌شود. در این صورت، تمام تست‌های صفر در خطوط ۷ تا ۱۳ با شکست مواجه می‌شود، و SEGMENTS-INTERSECTS در خط ۵ مقدار FALSE را برمی‌گرداند. شکل ۳۳-۳ (ب) این حالت را نشان می‌دهد.

حالات مرزی وقتی رخ می‌دهد که چرخش نسبی  $d_k$  صفر باشد. در اینجا، می‌دانیم که  $p_k$  با پارهخط دیگر هم راستا است.  $p_k$  روی پارهخط دیگر قرار دارد اگر و فقط اگر بین نقاط پایانی پارهخط دیگر باشد. رویه ON-SEGMENT مشخص می‌کند که آیا  $p_k$  بین نقاط پایانی پارهخط  $\overline{p_ip_j}$  قرار دارد یا خیر، که وقتی در خطوط ۷ تا ۱۳ فراخوانی می‌شود، پارهخط دیگری است. این رویه فرض می‌کند که  $p_k$  هم راستای پارهخط  $\overline{p_ip_j}$  است. شکل‌های ۳۳-۳ (پ) و (ت) حالت‌هایی با نقاط هم راستا را نشان می‌دهند. در شکل ۳۳-۳ (پ)،  $p_3$  روی  $\overline{p_1p_2}$  قرار دارد و در نتیجه SEGMENTS-INTERSECT در خط ۱۲، مقدار TRUE را برمی‌گرداند. در شکل ۳۳-۳ (ت) هیچ نقطه پایانی روی پارهخط‌های دیگر واقع نیست، و در نتیجه رویه SEGMENTS-INTERSECT در خط ۵ مقدار FALSE را برمی‌گرداند.

### کاربردهای دیگر حاصلضرب خارجی

بخش‌های بعدی این فصل، کاربردهای دیگری را برای حاصلضرب‌های خارجی معرفی می‌کند. در بخش ۳۳-۳ نیاز به مرتب‌سازی مجموعه‌ای از نقاط بر اساس زاویه‌های قطبی آن‌ها نسبت به مبدأ معین است. همان‌طور که در تمرین ۳۳-۱ خواهید دید، حاصلضرب‌های خارجی می‌توانند برای اجرای مقایسه‌ها در رویه مرتب‌سازی به کار روند. در بخش ۳۳-۲، از درخت‌های قرمز-سیاه برای نگهداری ترتیب عمودی مجموعه‌ای از پارهخط‌ها

1. straddle      2. red-black tree



شکل ۳۳-۳

حالات مختلف در رویه SEGMENTS-INTERSECT، (الف) پاره خط‌های  $\overline{P_1P_2}$  و  $\overline{P_3P_4}$ ، خطوط بیگنر را در میان می‌گیرند. چون  $\overline{P_3P_4}$  خط حاوی  $\overline{P_1P_2}$  را در میان می‌گیرد، علامت‌های حاصلضرب‌های خارجی  $(p_1 - p_3) \times (p_4 - p_3) < 0$  و  $(p_4 - p_1) \times (p_2 - p_1) \leq 0$  فرق می‌کند. چون  $\overline{P_1P_2}$  خط حاوی  $\overline{P_3P_4}$  را در میان می‌گیرد، علامت‌های حاصلضرب‌های خارجی  $(p_3 - p_1) \times (p_4 - p_3) \geq 0$  و  $(p_3 - p_1) \times (p_2 - p_1) > 0$  متفاوتند. (ب) پاره خط  $\overline{P_3P_4}$  خط حاوی  $\overline{P_1P_2}$  را در میان می‌گیرد، اما  $\overline{P_1P_2}$  خط حاوی  $\overline{P_3P_4}$  را در میان نمی‌گیرد. علامت‌های حاصلضرب‌های خارجی  $(p_4 - p_3) \times (p_2 - p_3) < 0$  و  $(p_1 - p_3) \times (p_4 - p_3) > 0$  یکسان است. (ج) نقطه  $p_3$  هم‌راستای  $\overline{P_1P_2}$  است و بین  $p_1$  و  $p_2$  واقع است. (د) نقطه  $p_3$  هم‌راستای  $\overline{P_1P_2}$  قرار دارد، اما بین  $p_1$  و  $p_2$  قرار ندارد. پاره خط‌ها متقاطع نیستند.

استفاده خواهیم کرد، به جای نگهداری مقادیر کلید صریح، هر مقایسه کلید در کد درخت قرمز - سیاه را با محاسبه حاصلضرب خارجی جایگزین می‌کنیم که تعیین می‌کند کدام دو پاره خطی که یک خط عمودی را قطع می‌کنند، در بالای دیگری قرار دارند.

زاویه‌های قطبی نسبت به نقطه مبدأ  $p_0$  بنویسید. زمان زاویه شما باید  $O(n \lg n)$  باشد و برای مقایسه

زاویه‌ها از حاصل ضرب خارجی استفاده کنید.

تمرین ۳۳-۱-۴: نشان دهید که چگونه می‌توان در زمان  $O(n^2 \lg n)$  تعیین کرد که هر سه نقطه در مجموعه  $n$  نقطه‌ای، هم‌راستا هستند.

تمرین ۳۳-۵: چندضلعی، یک منحنی بسته‌ی تکه‌ای خطی<sup>۱</sup> در صفحه است. یعنی، منحنی‌ای است که روی خودش خاتمه می‌یابد و با دنباله‌ای از پاره‌خط‌های مستقیم تشکیل می‌شود که ضلع چندضلعی نام دارد. نقطه‌ای که دو ضلع متوازی را به هم متصل می‌کند، رأس چندضلعی نام دارد. اگر چندضلعی ساده باشد، که در حالت کلی فرض می‌کنیم این‌طور است، خودش را قطع نمی‌کند. مجموعه‌ای از نقاط در صفحه که توسط چندضلعی ساده‌ای دربرگرفته می‌شوند، داخل<sup>۲</sup> چندضلعی را تشکیل می‌دهند. مجموعه‌ای از نقاط که روی خود چندضلعی قرار دارند، موز آن را تشکیل می‌دهند، و مجموعه‌ای از نقاطی که چندضلعی را دربرمی‌گیرند، خارج<sup>۳</sup> چندضلعی را ایجاد می‌کنند. چندضلعی ساده، محاسبه است اگر با توجه به دو نقطه در مرز یا داخل آن، تمام نقاط روی پاره‌خط رسم شده بین آن‌ها، در مرز چندضلعی یا در داخل آن باشند.

پروفسور Amundsen ادعا می‌کند که روش زیر، تعیین می‌کند آیا دنباله  $n$  نقطه‌ای  $\langle p_0, p_1, \dots, p_{n-1} \rangle$  رئوس متوازی چندضلعی محاسبه را تشکیل می‌دهد یا خیر. اگر مجموعه  $\{i : i = 0, 1, \dots, n-1\}$  که در آن جمع زیرنویس‌ها به پیمانه  $n$  انجام می‌شود، فاقد هر دو چپ‌گرد و راست‌گرد باشد، خروجی این روش "yes" و گرنه "no" است. نشان دهید که گرچه این روش در زمان خطی اجرا می‌شود، همواره پاسخ درست ارائه نمی‌کند. روش این پروفسور را اصلاح کنید به طوری که همواره جواب درستی را در زمان خطی ارائه نماید.

تمرین ۳۳-۱-۶: با توجه به نقطه  $(x_0, y_0) = p_0$ ، نیم خط (شعاع) افقی راست<sup>۴</sup> از  $p_0$ ، مجموعه‌ای از نقاط زیر است:

$$\{p_i = (x_i, y_i) : x_i \geq x_0 \text{ و } y_i = y_0\}$$

یعنی مجموعه‌ای از نقاط در سمت راست  $p_0$  در امتداد خود  $p_0$  است. نشان دهید که چگونه می‌توان در زمان  $O(1)$  تعیین کرد نیم خط افقی راست از  $p_0$ ، پاره‌خط  $\overline{p_1 p_2}$  را قطع می‌کند. برای این کار، این مسئله را به مسئله‌ی متقاطع بودن دو پاره‌خط تبدیل کنید.

تمرین ۳۳-۱-۷: یک روش تعیین این که "آیا نقطه  $p_0$  در داخل چندضلعی ساده ولی نه الزاماً محاسبه P قرار دارد"، این است که هر نیم خط از  $p_0$  بررسی شود تا مشخص گردد که آیا مرز  $p$  را به تعداد دفعات فرد قطع می‌کند یا خیر، ولی خود  $p_0$  در مرز  $p$  نباشد. نشان دهید که چگونه در زمان  $O(n \Theta(n))$  می‌توان تعیین کرد که نقطه  $p_0$  در داخل چندضلعی  $n$  رأسی  $p$  قرار دارد (رهنمایی: از تمرین ۳۳-۱-۶ استفاده کنید). مطمئن شوید که وقتی نیم خط مرز چندضلعی را در رأسی قطع می‌کند و وقتی نیم خط با ضلع چندضلعی همپوشانی دارد، الگوریتم شما درست است).

تمرین ۳۳-۱-۸: نشان دهید که چگونه می‌توان مساحت چندضلعی  $n$  رأسی ساده، ولی نه الزاماً محاسبه را در زمان  $O(n \Theta(n))$  محاسبه کرد (برای تعاریف مربوط به چندضلعی‌ها، تمرین ۱-۵ ۳۳-۱ را ببینید).

## ۳۳-۲ تعیین این که آیا هر جفت پاره‌خط متقاطع‌اند یا خیر

این بخش، الگوریتم را ارائه می‌کند که تعیین می‌کند آیا دو پاره‌خط در مجموعه‌ای از پاره‌خط‌ها متقاطع‌اند یا خیر. این الگوریتم، از تکنیکی به نام "جاروکردن" استفاده می‌کند، که در بین بسیاری از الگوریتم‌های هندسه

1. piecewise – linear      2. interior      3. exterior      4. right horizontal ray

محاسباتی، مشترک است، علاوه بر این، همان طور که تعریف‌های انتهای این فصل نشان می‌دهند، این الگوریتم، یا شکل‌های متفاوتی از آن، می‌تواند برای حل سایر مسئله‌های هنامه محاسباتی به کار رود.

این الگوریتم در زمان  $O(n \lg n)$  اجرا می‌شود، که  $n$  تعداد پاره خط‌ها است. این الگوریتم فقط تعیین می‌کند تقاطعی وجود دارد یا خیر، تمام تقاطع‌ها را چاپ نمی‌کند (با به تعریف ۱-۲-۳، برای یافتن تمام تقاطع‌ها در مجموعه‌ای با  $n$  پاره خط، در بدترین حالت، به زمان  $\Omega(n^2)$  نیاز دارد).

در جارو کردن، یک خط جاروی<sup>۱</sup> عمودی فرضی، از مجموعه‌ای از اشیای هنامی عبور می‌کند (عمولاً از چپ به راست). بعد فضایی که خط جارویی در امتداد آن حرکت می‌کند، در این حالت بعد از  $x$ ، به عنوان بعد زمان در نظر گرفته می‌شود. جارو کردن، روشی را برای مرتب‌سازی اشیای هنامی، عمولاً با فرارادن آن‌ها در ساختمان داده‌ی پویا، فراهم می‌کند، و از امتیاز روابط بین آن‌ها بهره می‌برد. الگوریتم تقاطع پاره خط در این بخش، تمام نقاط پایانی پاره خط را به ترتیب از چپ به راست در نظر می‌گیرد، و هر وقت به یک نقطه پایانی رسید، وجود تقاطع را بررسی می‌کند.

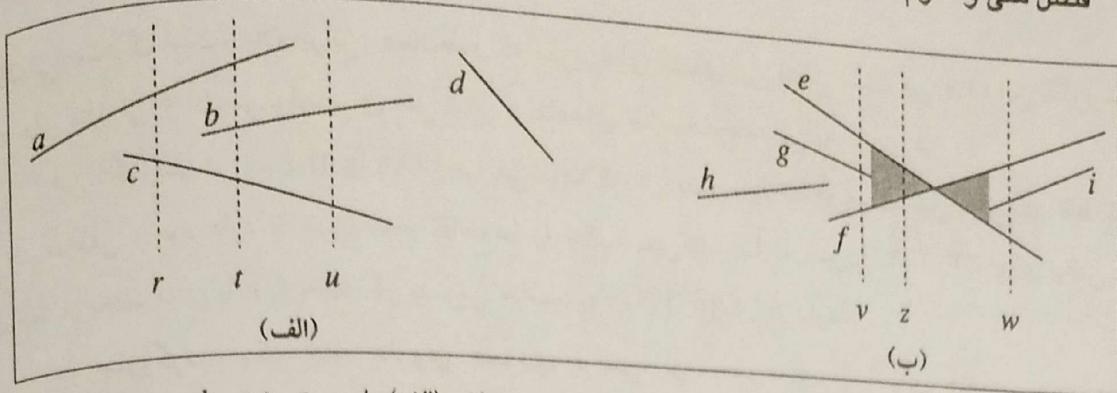
برای توصیف و اثبات الگوریتم در تعیین متقاطع‌بودن هر دو پاره خط از  $n$  پاره خط، دو فرض ساده‌سازی را انجام می‌دهیم. اولًا، فرض می‌کنیم هیچ پاره خط ورودی، عمودی نیست. ثانیًا، فرض می‌کنیم هیچ سه پاره خطی در یک نقطه یکدیگر را قطع نمی‌کنند. تعریف‌های ۲-۸ و ۲-۹ از شما می‌خواهند که نشان دهنده الگوریتم به اندازه کافی قوی است و برای این‌که بدون این فرض‌ها نیز به درستی عمل کند، نیاز به تغییر اندکی است. در واقع، حذف این فرض‌های ساده‌سازی و اداره کردن شرط‌های مرزی، مشکل‌ترین بخش برنامه‌نویسی الگوریتم‌های محاسباتی و اثبات صحت آن‌ها است.

## مرتب‌سازی پاره خط‌ها

چون فرض می‌کنیم پاره خط‌های عمودی وجود ندارند، هر پاره خط ورودی که خط جاروی عمودی را قطع می‌کند، آن را در یک نقطه قطع می‌کند. بنابراین، پاره خط‌هایی که خط جاروی عمودی را قطع می‌کنند، می‌توانند بر حسب مختصات  $y$  نقاط تقاطع مرتب شوند.

به عبارت دیگر، دو پاره خط  $s_1$  و  $s_2$  را در نظر بگیرید. می‌گوییم این پاره خط‌ها در  $x$  قابل مقایسه‌اند، اگر خط جاروی عمودی با مختصات  $x$ ، هر دو را قطع کنند. می‌گوییم  $s_1$  در بالای  $s_2$  در  $x$  قرار دارد، و به صورت  $x > x_1$  می‌نویسیم اگر  $s_1$  و  $s_2$  در  $x$  قابل مقایسه باشند و تقاطع  $s_1$  با خط جارو در  $x$ ، بالاتر از تقاطع  $s_2$  با همان خط جارو باشد. به عنوان مثال، در شکل ۴-۳۳ (الف)، رابطه‌های  $a >_x c$ ،  $b >_x c$ ،  $a >_x b$ ،  $a >_x c$ ،  $b >_x c$  (الف)، رابطه‌های  $a >_x b$ ،  $c >_x b$ ،  $c >_x a$  (ب)، رابطه‌های  $a >_x d$ ،  $b >_x d$ ،  $c >_x d$  (ج) می‌نماییم.

و  $c >_x b$  را داریم. پاره خط  $d$ ، با هیچ پاره خط دیگری قابل مقایسه نیست. برای هر  $x$ ، رابطه " $x >$ ", ترتیب کلی (پیوست (ب)) روی پاره خط‌هایی است که خط جارو را در  $x$  قطع می‌کند. این ترتیب ممکن است برای مقادیر مختلف  $x$ ، فرق کند، به طوری که پاره خط‌هایی وارد ترتیب یا از آن خارج می‌شوند. پاره خط وقتی وارد ترتیب می‌شود که نقطه پایانی چپ آن، توسط جارو کردن دیده شود، و وقتی ترتیب را ترک می‌کند که نقطه پایانی راست آن مشاهده می‌گردد.



شکل ۳۳-۴ مرتب‌سازی بین پاره‌خط‌ها در خطوط جاروی عمودی مختلف. (الف) داریم  $a >_r c, b >_r c, a >_r b, a >_r c$  و  $c >_r b$ . پاره‌خط  $d$  با هیچ پاره‌خط دیگر قابل مقایسه نیست. (ب) وقتی پاره‌خط‌های  $e$  و  $f$  یکدیگر را قطع می‌کنند، ترتیب آن‌ها معکوس می‌شود: داریم  $f >_v e >_w e$ . هر خط جارو (مثل  $v$ ) که از ناحیه سایه‌دار عبور می‌کند، در ترتیب کلی<sup>۱</sup> خود،  $e$  و  $f$  را متواالی می‌بیند.

وقتی خط جارو، از تقاطع دو پاره‌خط عبور می‌کند چه اتفاقی می‌افتد؟ همان‌طور که شکل ۳۳-۴ (ب) نشان می‌دهد، مکان‌های آن‌ها در ترتیب کلی، معکوس می‌شود. خطوط جاروی  $v$  و  $w$  به ترتیب در چپ و راست نقطه تقاطع پاره‌خط‌های  $e$  و  $f$  وجود دارند، و داریم  $f >_v e >_w e$ . توجه کنید که چون فرض می‌کنیم هیچ سه پاره‌خطی یکدیگر را در یک نقطه قطع نمی‌کنند، باید خط جاروی عمودی خاصی وجود داشته باشد که برای آن، پاره‌خط‌های قطع‌کننده  $e$  و  $f$ ، در ترتیب کلی  $x$ ، متواالی‌اند. هر خط جارو که از ناحیه سایه‌دار شکل ۳۳-۴ (ب) عبور می‌کند، مثل  $z$ ، پاره‌خط‌های  $e$  و  $f$  در ترتیب کلی آن، متواالی‌اند.

### انتقال خط جارو

الگوریتم‌های جارو کردن معمولاً، دو مجموعه از داده‌ها را مدیریت می‌کند:

۱. وضعیت خط جارو<sup>۲</sup>، روابط بین اشیای متقاطع با خط جارو را مشخص می‌کند.
۲. زمان‌بندی نقطه رویداد<sup>۳</sup>، دنباله‌ای از مختصات  $x$  است که از چپ به راست مرتب است، و موقعیت‌های توقف خط جارو را مشخص می‌کند. هر موقعیت توقف را نقطه رویداد می‌گوییم. تغییر در وضعیت خط جارو، فقط در نقاط رویداد رخ می‌دهد.

برای بعضی از الگوریتم‌ها (به عنوان مثال، الگوریتم خواسته‌شده در تمرین ۷-۲-۷)، زمان‌بندی نقطه رویداد به طور پویا در زمان اجرای الگوریتم تعیین می‌شود. اما، الگوریتم موردنظر، نقاط رویداد را به طور ایستا تعیین می‌کند، که مبنی بر خواص ساده‌های ورودی است. مخصوصاً، هر نقطه پایانی پاره‌خط، یک نقطه رویداد است. نقاط پایانی پاره‌خط را با افزودن مختصات  $x$  و پیش‌روی از چپ به راست، مرتب می‌کنیم (اگر دو یا چند نقطه پایانی، پوششی باشند)، یعنی مختصات  $x$  آن‌ها یکسان باشد، گره را با قراردادن تمام نقاط پایانی چپ پوششی، قبل از نقاط پایانی راست پوششی، می‌شکنیم. در داخل مجموعه‌ای از نقاط پایانی چپ پوششی، ابتدا نقاطی با مختصات  $y$  کمتر را قرار می‌دهیم، و همین کار را در داخل مجموعه‌ای از نقاط پایانی راست پوششی انجام می‌دهیم. پاره‌خط را در صورتی در وضعیت خط جارو قرار می‌دهیم که نقطه پایانی چپ آن مشاهده شود، و وقتی آن را از وضعیت خط جارو حذف می‌کنیم که نقطه پایانی راست آن مشاهده شود. هر وقت دو پاره‌خط، ابتدا در ترتیب کلی به طور متواالی قرار می‌گیرند، بررسی می‌کنیم که آیا متقاطع‌اند یا خیر.

وضعیت خط جارو، ترتیب کلی  $T$  است، که برای آن به عملیات‌های زیر نیاز داریم:

$\text{INSERT}(T, s)$  : قراردادن پاره خط  $s$  در  $T$ .

$\text{DELETE}(T, s)$  : حذف پاره خط  $s$  از  $T$ .

$\text{ABOVE}(T, s)$  : پاره خط موجود در بالای پاره خط  $s$  را در  $T$  برمی‌گرداند.

$\text{BELOW}(T, s)$  : پاره خط موجود در زیر پاره خط  $s$  را در  $T$  برمی‌گرداند.

اگر  $n$  پاره خط در ورودی وجود داشته باشد، می‌توان هر یک از این عملیات‌ها را با استفاده از درخت قرمز – سیاه در زمان  $O(\lg n)$  انجام داد. به یاد داشته باشید که عملیات‌های درخت قرمز – سیاه شامل مقایسه کلیدها است. مقایسه‌های کلید را می‌توان با مقایسه‌هایی جایگزین کرد که برای تعیین ترتیب نسبی دو پاره خط، از حاصل ضرب خارجی استفاده می‌کند (تمرین ۲-۲-۳۲ را ببینید).

## شبکه‌گرد تقاطع پاره خط

الگوریتم زیر، مجموعه  $S$  با  $n$  پاره خط را به عنوان ورودی می‌گیرد، و اگر جفتی از پاره خط‌ها در  $S$  متقاطع باشند، مقدار  $\text{TRUE}$ ، وگرنه مقدار  $\text{FALSE}$  را برمی‌گرداند. ترتیب کلی  $T$  توسط درخت قرمز – سیاه پیاده‌سازی می‌شود:

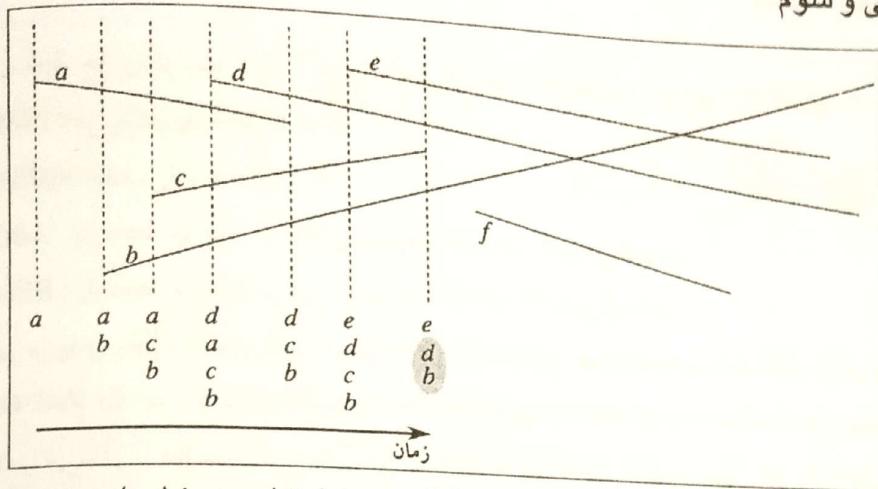
### ANY-SEGMENTS-INTERSECT( $S$ )

```

1    $T \leftarrow \emptyset$ 
2   sort the endpoints of the segments in  $S$  from left to right,
      breaking ties by putting left endpoints before right endpoints
      and breaking further ties by putting points with lower
      y-coordinates first
3   for each point  $p$  in the sorted list of endpoints
4     do if  $p$  is the left endpoint of a segment  $s$ 
5       then  $\text{INSERT}(T, s)$ 
6         if ( $\text{ABOVE}(T, s)$  exists and intersects  $s$ )
7           or ( $\text{BELOW}(T, s)$  exists and intersects  $s$ )
8             then return  $\text{TRUE}$ 
9         then if both  $\text{ABOVE}(T, s)$  and  $\text{BELOW}(T, s)$  exist
          and  $\text{ABOVE}(T, s)$  intersects  $\text{BELOW}(T, s)$ 
10        then return  $\text{TRUE}$ 
11         $\text{DELETE}(T, s)$ 
12    return  $\text{FALSE}$ 
```

شکل ۵-۳۳-۵ اجرای این الگوریتم را تشریح می‌کند. خط ۱، مقدار اولیه ترتیب کلی را تهی درنظرمی‌گیرد. خط ۲ زمانبندی نقطه رویداد را با مرتب‌سازی  $2n$  نقطه پایانی پاره خط، از چپ به راست، تعیین می‌کند، و مانند آنچه که شرح داده شد، گره‌ها شکسته می‌شوند. توجه کنید که خط ۲ می‌تواند با مرتب‌سازی واژگانی نقاط پایانی روی  $(y, x, e)$  انجام شود، که  $x$  و  $y$  مختصات معمولی‌اند، و برای نقطه پایانی چپ داریم  $e = 0$  برای نقطه پایانی راست داریم  $e = 1$ .

هر تکرار حلقه **for** در خطوط ۳ تا ۱۱، یک نقطه رویداد  $p$  را پردازش می‌کند. اگر  $p$  نقطه پایانی چپ پاره خط  $s$  باشد، خط ۵،  $s$  را به ترتیب کلی می‌افزاید، و خطوط ۶ و ۷ در صورتی  $\text{TRUE}$  را برمی‌گرداند که  $s$  با یکی از پاره خط‌های مجاور خود در ترتیب کلی تعریف شده توسط خط جارویی که از  $p$  عبور می‌کند، متقاطع باشد.



شکل ۵-۳۳

اجرای ANY-SEGMENTS-INTERSECT . هر خط نقطه‌چین، خط جارویی در نقطه زوج است، و ترتیب اسامی پاره‌خطها در زیر هر خط جارو، ترتیب کلی  $T$  در انتهای حلقه for است که در آن نقطه زوج متناظر آن پردازش می‌شود. تقاطع قطعه‌خطهای  $d$  و  $b$  وقتی پیدا می‌شود که قطعه  $c$  حذف شود.

(اگر  $p$  در پاره‌خط دیگر  $s'$  قرار داشته باشد، شرط مرزی رخ می‌دهد. در این حالت، فقط لازم است  $s$  و  $s'$  به طور متواالی در  $T$  قرار گیرند). اگر  $p$  نقطه پایانی راست پاره‌خط  $s$  باشد، آنگاه  $s$  باید از ترتیب کلی حذف شود. اگر در ترتیب کلی تعریف شده توسط خط جاروی عبور کننده از  $p$  ، تقاطعی بین پاره‌خطهای دربرگیرنده  $s$  وجود داشته باشد، خطوط ۹ و ۱۰ مقدار TRUE را بر می‌گرداند. وقتی  $s$  حذف می‌شود، این پاره‌خطها در ترتیب کلی، متواالی خواهند بود. اگر این پاره‌خطها متقاطع نباشند، خط ۱۱ پاره‌خط  $s$  را از ترتیب کلی حذف می‌کند. سرانجام، اگر هیچ تقاطعی در پردازش تمام  $n$  نقطه رویداد نباشد، خط ۱۲ مقدار FALSE را بر می‌گرداند.

### درستی (صحت) الگوریتم

برای این که نشان دهیم ANY-SEGMENTS-INTERSECT درست است، ثابت خواهیم کرد که فراخوانی ANY-SEGMENTS-INTERSECT(S) مقدار TRUE را بر می‌گرداند اگر و فقط اگر تقاطعی بین پاره‌خطها در وجود داشته باشد.

به آسانی می‌توان دید که ANY-SEGMENTS-INTERSECT (در خطوط ۷ و ۱۰) مقدار TRUE را بر می‌گرداند فقط اگر تقاطعی را بین دو پاره‌خط ورودی پیدا کند. بنابراین، اگر TRUE را برگرداند، تقاطعی وجود دارد. لازم است عکس آن را نیز نشان دهیم: اگر تقاطعی وجود داشته باشد، آنگاه ANY-SEGMENTS-INTERSECT مقدار TRUE را بر می‌گرداند. فرض می‌کنیم حداقل یک تقاطع وجود دارد. فرض کنید  $p$  چهارمین نقطه تقاطع باشد، که گره‌ها با انتخاب نقطه تقاطعی با کمترین مقدار مختصات  $y$  شکسته می‌شود، و  $a$  و  $b$  پاره‌خطهایی باشند که در  $p$  یکدیگر را قطع می‌کنند. چون هیچ تقاطعی در چهارمین نقطه  $p$  رخ نمی‌دهد، ترتیب مشخص شده توسط  $T$  ، در تمام نقاط سمت چهارمین نقطه  $p$  درست است. چون هیچ سه پاره‌خطی در یک نقطه یکدیگر را قطع نمی‌کنند، یک خط جاروی  $z$  وجود دارد که در آن،  $a$  و  $b$  در ترتیب کلی، متواالی خواهند بود.

۱. اگر اجزه دهیم سه پاره‌خط در یک نقطه یکدیگر را قطع کنند، ممکن است یک پاره‌خط میانی  $c$  وجود داشته باشد که  $a$  و  $b$  را در نقطه  $p$  قطع کند. یعنی، ممکن است برای تمام خطوط جاروی  $w$  در سمت چهارمین نقطه  $p$  که برای آنها  $b <_w a$  است، داشته باشیم  $c <_w b$  و  $a <_w c$ . تمرین ۳۳-۲-۱ می‌خواهد که نشان دهید رویه ANY-SEGMENTS-INTERSECT درست است حتی اگر سه پاره‌خط در یک نقطه یکدیگر را قطع کنند.

علاوه براین،  $p$  در سمت چپ قرار دارد یا از  $p$  عبور می‌کند. یک نقطه پایانی پاره خط به نام  $q$  روی خط جاروی  $z$  وجود دارد که نقطه رویدادی است که در آن،  $a$  و  $b$  در ترتیب کلی، متواالی خواهد بود. اگر  $p$  روی خط جاروی  $z$  باشد، آنگاه  $p = q$ . اگر  $p$  روی خط جاروی  $z$  نباشد، آنگاه  $q$  سمت چپ  $p$  است. در هر حال، ترتیب ارائه شده توسط  $T$ ، درست قبل از دیدن  $q$  درست است (اینجا، جایی است که از ترتیب واژگانی استفاده می‌کنیم که در آن، الگوریتم، نقاط رویداد را پردازش می‌کند). چون  $p$  پایین‌ترین نقطه تقاطع چپ است، حتی اگر  $p$  روی خط جاروی  $z$  باشد و نقطه تقاطع دیگر  $p'$  روی  $z$  باشد، قبل از این که نقطه تقاطع دیگر  $p'$  بتواند با ترتیب کلی  $T$  تداخل ایجاد کند، نقطه رویداد  $= p$  پردازش می‌شود. علاوه براین، حتی اگر  $p$  نقطه پایانی چپ پاره خطی مثل  $a$  و نقطه پایانی راست، پاره خط دیگری مثل  $b$  باشد، چون رویدادهای نقطه پایانی چپ، قبل از رویدادهای نقطه پایانی راست رخ می‌دهند، پاره خط  $b$  وقتی در  $T$  هست که ابتدا پاره خط  $a$  دیده شود) چه، نقطه  $q$  توسط ANY-SEGMENTS-INTERSECT پردازش شود و چه نشود.

اگر  $q$  توسط ANY-SEGMENTS-INTERSECT پردازش شود، فقط دو عمل ممکن می‌تواند انجام

شود:

۱.  $a$  یا  $b$  در  $T$  قرار می‌گیرند، و پاره خط دیگر در بالا یا زیر آن در ترتیب کلی قرار دارد. خطوط ۴ تا ۷ این حالت را تشخیص می‌دهند.

۲. پاره خطهای  $a$  و  $b$  در  $T$  قرار دارند، و پاره خط بین آنها در ترتیب کلی حذف شده است، و در نتیجه  $a$  و  $b$  متواالی می‌شوند. خطوط ۸ تا ۱۱ این حالت را تشخیص می‌دهد.

در هر حالت، تقاطع  $p$  پیدا می‌شود و ANY-SEGMENTS-INTERSECT مقدار TRUE را بر می‌گرداند. اگر نقطه رویداد  $q$  توسط ANY-SEGMENTS-INTERSECT پردازش نشود، رویه، قبل از پردازش ANY-SEGMENTS-INTERSECT تقاطعی را پیدا کرده و TRUE را برگرداند. این وضعیت در صورتی می‌توانست رخ دهد که نام نقاط رویداد، می‌بایست خاتمه یابد. این وضعیت در صورتی می‌توانست رخ دهد که ANY-SEGMENTS-INTERSECT تقاطعی را پیدا کرده و TRUE را برگرداند باشد. بنابراین، اگر تقاطعی وجود داشته باشد، ANY-SEGMENTS-INTERSECT مقدار TRUE را بر می‌گرداند. همان‌طور که دیدیم، اگر رویه ANY-SEGMENTS-INTERSECT مقدار TRUE را برگرداند، تقاطعی وجود دارد. بنابراین، ANY-SEGMENTS-INTERSECT همیشه پاسخ درستی را بر می‌گرداند.

## زمان اجرا

اگر پاره خطهایی در مجموعه  $S$  وجود داشته باشند، آنگاه رویه ANY-SEGMENTS-INTERSECT  $O(n \lg n)$  اجرا می‌شود. خط ۱ به زمان  $(1)$  نیاز دارد. خط ۲، با استفاده از مرتب‌سازی ادغام یا مرتب‌سازی ۲ $n$  هیپ، نیاز به زمان  $O(n \lg n)$  دارد. چون  $2n$  نقطه رویداد وجود دارد، حلقه for در خطوط ۳ تا ۱۱ حداقل  $O(n \lg n)$  بار اجرا می‌شود. هر تکرار در زمان  $O(\lg n)$  اجرا می‌شود، زیرا هر عملیات درخت قرمز - سیاه به زمان  $O(\lg n)$  دارد، و با استفاده از روش بخش ۱-۳۳، هر تست تقاطع به زمان  $(1)$  نیاز دارد. بنابراین، کل زمان،  $O(n \lg n)$  است.

### تمرین‌های بخش ۲-۳

تمرين ۱-۲-۳: نشان دهيد که ممکن است  $(n^2)$  تقاطع در مجموعه‌ای از  $n$  پاره خط وجود داشته باشد.

تمرين ۲-۲-۳: با توجه به پاره خط‌های  $a$  و  $b$  که در  $x$  قابل مقایسه‌اند، نشان دهيد که چگونه می‌توان در زمان  $(1)$  تعیین کرد  $b >_x a$  یا  $a >_x b$  برقرار است. فرض کنید هیچ‌کدام از پاره خط‌ها عمودی نیستند (راهنمایی: اگر  $a$  و  $b$  متقطع نباشند، می‌توانید از حاصلضرب‌های خارجی استفاده کنید. اگر  $a$  و  $b$  متقطع باشند – که البته از طریق حاصلضرب خارجی تعیین می‌شود – هنوز می‌توانید فقط از جمع، تفریق و ضرب استفاده کنید و از تقسیم اجتناب نمایید. البته، در کاربرد رابطه  $>$  که در اینجا استفاده شد، اگر  $a$  و  $b$  متقطع باشند، می‌توانیم متوقف شویم و اعلان کنیم که تقاطعی را پیدا کردیم).

تمرين ۳-۲-۳: پروفسور Maginot پیشنهاد می‌کند که رویه ANY-SEGMENTS-INTERSECT اصلاح شود، به طوری که به جای خاتمه‌یافتن به محض یافتن تقاطع، پاره خط‌های متقطع را چاپ کند، و به تکرار بعدی PRINT-INTERSEGMENTING-SEGMENTS حلقه for ادامه دهد. پروفسور، رویه حاصل را PRINT-INTERSEGMENTING-SEGMENTS می‌نامند و ادعا می‌کند که تمام تقاطع‌ها را از چپ به راست، به صورتی که در مجموعه‌ای از پاره خط‌ها ظاهر می‌شوند، چاپ می‌کند. با ارائه مجموعه‌ای از پاره خط‌هایی که برای آن‌ها، اولین تقاطعی که توسط PRINT-INTERSEGMENTING-SEGMENTS پاره خط‌هایی که برای آن‌ها، PRINT-INTERSEGMENTING-SEGMENTS در یافتن تمام تقاطع‌ها با شکست مواجه می‌شود، نشان دهید در دو شمارش دچار اشتباه می‌شود.

تمرين ۴-۲-۳: یک الگوریتم زمان  $O(n \lg n)$  ارائه دهید که تعیین کند آیا چندضلعی  $n$  رأسی ساده است یا خیر.

تمرين ۵-۲-۳: یک الگوریتم زمان  $O(n \lg n)$  ارائه دهید که تعیین کند آیا دو چندضلعی ساده با تعداد کل رأس‌های  $n$ ، متقطع‌اند یا خیر.

تمرين ۶-۲-۳: دیسک، شامل یک دایره به اضفه داخل آن است و توسط نقطه مرکزی و شعاع نمایش داده می‌شود. دو دیسک در صورتی متقطع‌اند که نقطه مشترکی داشته باشند. یک الگوریتم زمان  $O(n \lg n)$  ارائه دهید که تعیین کند آیا دو دیسک در مجموعه‌ای از  $n$  دیسک، متقطع‌اند یا خیر.

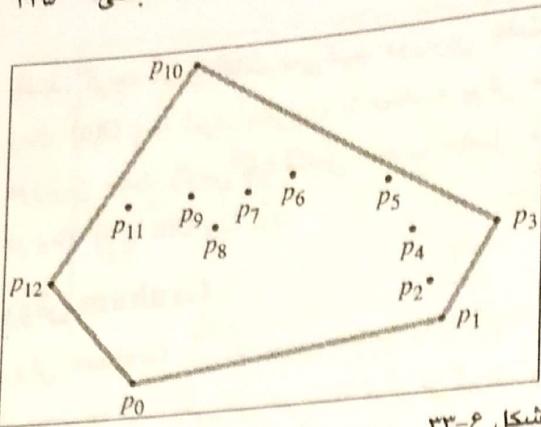
تمرين ۷-۲-۳: با توجه به مجموعه‌ای از  $n$  پاره خط که شامل  $k$  تقاطع است، نشان دهید که چگونه می‌توان تمام  $k$  تقاطع را در زمان  $O((n+k) \lg n)$  چاپ کرد.

تمرين ۸-۲-۳: ثابت کنید که حتی اگر سه یا چند پاره خط در یک نقطه یکدیگر را قطع کنند، رویه ANY-SEGMENTS-INTERSECT به درستی کار می‌کند.

تمرين ۹-۲-۳: اگر نقطه پایانی پاره خط عمودی طوری پردازش شود که گویی نقطه پایانی چپ است، و نقطه پایانی بالایی طوری پردازش شود که گویی نقطه پایانی راست است، رویه ANY-SEGMENTS-INTERSECT با وجود پاره خط‌های عمودی به درستی کار می‌کند. اگر پاره خط‌های عمودی مجاز باشند، پاسخ شما به تمرين ۲-۲-۳ چه تغییری خواهد کرد.

### ۳۳-۳ یافتن پوش محدب

پوش محدب مجموعه  $Q$  از نقاط، کوچک‌ترین چندضلعی محدب  $P$  است که برای آن، هر نقطه در  $Q$ ، در مرز  $P$  یا در داخل آن قرار دارد (تمرين ۱-۵ ۳۳-۱ را ببینید تا تعریف دقیق چندضلعی محدب را مشاهده کنید). پوش محدب  $Q$  را با  $CH(Q)$  نشان می‌دهیم. از نظر شهودی، می‌توان هر نقطه در  $Q$  را مثل میخی دانست که از صفحه‌ای خارج می‌شود. سپس پوش محدب، شکلی است که توسط نوار پلاستیکی ایجاد شد که تمام میخ‌ها را احاطه می‌کند. شکل ۳۳-۶ مجموعه‌ای از نقاط و پوش محدب آن‌ها را نشان می‌دهد.



شکل ۳۳-۶ مجموعه‌ای از نقاط  $Q = \{p_0, p_1, \dots, p_{12}\}$  که پوش محدب  $\text{CH}(Q)$  با رنگ خاکستری نشان داده شد.

خاصیت را دارند که، تصمیم می‌گیرند کدام رئوس در  $Q$  به عنوان پوش محدب نگهداری شوند و کدام رئوس از  $Q$  حذف گردند.

در واقع، روش‌های مختلفی برای محاسبه پوش محدب در زمان  $O(n \lg n)$  وجود دارد. هر دو روش Graham و Jarvis، از تکنیکی به نام "جاروی دورانی" استفاده می‌کند، که در آن، رئوس به ترتیب زاویه‌های نقطی که تشکیل می‌دهند پردازش می‌شوند و یک رأس به عنوان رأس مرجع است. سایر روش‌ها عبارتنداز: در روش افزایشی<sup>۱</sup>، نقاط از چپ به راست ذخیره می‌شوند، تا دنباله  $\langle p_1, p_2, \dots, p_n, p_1 \rangle$  به وجود آید. در مرحله  $i$ ام، پوش محدب  $1-i$  نقطه‌ی چپ، یعنی  $\text{CH}(\{p_1, p_2, \dots, p_{i-1}\})$ ، بر اساس نقطه  $i$ ام از چپ، به‌هنگام می‌شود، و در نتیجه  $\text{CH}(\{p_1, p_2, \dots, p_i\})$  به وجود می‌آید. چنانچه در تمرین ۳۳-۶ خواهد دید، این روش می‌تواند طوری پیاده‌سازی شود که در زمان  $O(n \lg n)$  اجرا گردد.

در روش تقسیم و حل<sup>۲</sup>، مجموعه  $n$  نقطه‌ای در زمان  $\Theta(n)$  به دو زیرمجموعه تقسیم می‌شود، که یکی از آنها حاوی  $\lceil n/2 \rceil$  نقطه‌ی سمت چپ و دیگری شامل  $\lceil n/2 \rceil$  نقطه‌ی سمت راست است؛ پوش‌های محدب زیرمجموعه‌ها به طور بازگشتی محاسبه می‌شوند، و سپس یک روش هوشمند، پوش‌های محدب را در زمان  $O(n)$  ترکیب می‌کند. زمان اجرا با عبارت بازگشتی معروف  $T(n) = 2T(n/2) + O(n)$  توصیف می‌شود، و در نتیجه روش تقسیم و حل در زمان  $O(n \lg n)$  اجرا گردد.

روش هرس و جست‌وجو<sup>۳</sup>، شبیه الگوریتم میانه زمان خطی در بدترین حالت است (بخش ۹-۳). این روش، بخش بالایی (یا "زنجیره‌ی بالایی") پوش محدب را محاسبه می‌کند. برای این کار، کسر ثابتی از نقاط باقیمانده را حذف می‌کند تا فقط زنجیره‌ی بالایی پوش محدب باقی بماند. سپس این کار را برای زنجیره‌ی پایینی انجام می‌دهد. این روش، از نظر مجانبی سریع‌تر است. اگر پوش محدب شامل ۱۱ رأس باشد، فقط در زمان  $O(n \lg n)$  اجرا می‌شود.

محاسبه پوش محدب مجموعه‌ای از نقاط، مسئله جالبی است. علاوه‌براین، الگوریتم‌های مربوط به سایر مسئله‌های هندسه محاسباتی، با محاسبه پوش محدب شروع می‌شوند. به عنوان مثال، مسئله دورترین جفت<sup>۴</sup> را درنظر بگیرید: مجموعه‌ای از  $n$  نقطه در صفحه وجود دارد، و می‌خواهیم دو نقطه‌ای را پیدا کنیم که فاصل آنها از یکدیگر، ماکزیمم است. همان‌طور که در تمرین ۳۳-۳ خواهد دید، این دو نقطه باید رئوس پوش محدب

در این بخش، دو الگوریتم ارائه می‌کنیم که پوش محدب مجموعه‌ای از  $n$  نقطه را محاسبه می‌کند. هر دو الگوریتم، رئوس پوش محدب را در جهت عقربه‌های ساعت چاپ می‌کنند. اولی، که خلاف پیمایش Graham نام دارد، در زمان  $O(n \lg n)$  اجرا می‌شود. دومی که تعداد رئوس پوش محدب  $O(nh)$  اجرا می‌شود، که در شکل ۳۳-۶ دیده می‌شود، هر رأس  $\text{CH}(Q)$  نقطه‌ای در  $Q$  است. هر دو الگوریتم این است. همان‌طور که در شکل ۳۳-۶ دیده می‌شود، هر  $Q$  حذف گردند.

در واقع، روش‌های مختلفی برای محاسبه پوش محدب در زمان  $O(n \lg n)$  وجود دارد. هر دو روش Graham و Jarvis، از تکنیکی به نام "جاروی دورانی" استفاده می‌کند، که در آن، رئوس به ترتیب زاویه‌های نقطی که تشکیل می‌دهند پردازش می‌شوند و یک رأس به عنوان رأس مرجع است. سایر روش‌ها عبارتنداز: در روش افزایشی<sup>۱</sup>، نقاط از چپ به راست ذخیره می‌شوند، تا دنباله  $\langle p_1, p_2, \dots, p_n, p_1 \rangle$  به وجود آید. در مرحله  $i$ ام، پوش محدب  $1-i$  نقطه‌ی چپ، یعنی  $\text{CH}(\{p_1, p_2, \dots, p_{i-1}\})$ ، بر اساس نقطه  $i$ ام از چپ، به‌هنگام می‌شود، و در نتیجه  $\text{CH}(\{p_1, p_2, \dots, p_i\})$  به وجود می‌آید. چنانچه در تمرین ۳۳-۶ خواهد دید، این روش می‌تواند طوری پیاده‌سازی شود که در زمان  $O(n \lg n)$  اجرا گردد.

در روش تقسیم و حل<sup>۲</sup>، مجموعه  $n$  نقطه‌ای در زمان  $\Theta(n)$  به دو زیرمجموعه تقسیم می‌شود، که یکی از آنها حاوی  $\lceil n/2 \rceil$  نقطه‌ی سمت چپ و دیگری شامل  $\lceil n/2 \rceil$  نقطه‌ی سمت راست است؛ پوش‌های محدب زیرمجموعه‌ها به طور بازگشتی محاسبه می‌شوند، و سپس یک روش هوشمند، پوش‌های محدب را در زمان  $O(n)$  ترکیب می‌کند. زمان اجرا با عبارت بازگشتی معروف  $T(n) = 2T(n/2) + O(n)$  توصیف می‌شود، و در نتیجه روش تقسیم و حل در زمان  $O(n \lg n)$  اجرا گردد.

روش هرس و جست‌وجو<sup>۳</sup>، شبیه الگوریتم میانه زمان خطی در بدترین حالت است (بخش ۹-۳). این روش، بخش بالایی (یا "زنجیره‌ی بالایی") پوش محدب را محاسبه می‌کند. برای این کار، کسر ثابتی از نقاط باقیمانده را حذف می‌کند تا فقط زنجیره‌ی بالایی پوش محدب باقی بماند. سپس این کار را برای زنجیره‌ی پایینی انجام می‌دهد. این روش، از نظر مجانبی سریع‌تر است. اگر پوش محدب شامل ۱۱ رأس باشد، فقط در زمان  $O(n \lg n)$  اجرا می‌شود.

محاسبه پوش محدب مجموعه‌ای از نقاط، مسئله جالبی است. علاوه‌براین، الگوریتم‌های مربوط به سایر مسئله‌های هندسه محاسباتی، با محاسبه پوش محدب شروع می‌شوند. به عنوان مثال، مسئله دورترین جفت<sup>۴</sup> را درنظر بگیرید: مجموعه‌ای از  $n$  نقطه در صفحه وجود دارد، و می‌خواهیم دو نقطه‌ای را پیدا کنیم که فاصل آنها از یکدیگر، ماکزیمم است. همان‌طور که در تمرین ۳۳-۳ خواهد دید، این دو نقطه باید رئوس پوش محدب

باشدند. گرچه آن را اثبات نمی‌کنیم، دورترین جفت رئوس مربوط به چندضلعی محدب  $n$  رأسی می‌تواند در زمان  $O(n)$  پیدا شود. بنابراین، با محاسبه پوش محدب  $n$  نقطه ورودی در زمان  $O(n \lg n)$  و سپس یافتن دورترین جفت رئوس چندضلعی محدب حاصل، می‌توان دورترین جفت نقاط را در هر مجموعه  $n$  نقطه‌ای، در زمان  $O(n \lg n)$  پیدا کرد.

## روش Graham

روش **Graham**، مسئله پوش محدب را با نگهداری پسته<sup>1</sup>  $S$  از نقاط کاندید، حل می‌کند. هر نقطه‌ی ورودی از مجموعه  $Q$ ، یکی‌یکی در پسته قرار می‌گیرند، و نقاطی که رئوس  $\text{CH}(Q)$  نیستند، سرانجام از پسته حذف می‌شوند. وقتی الگوریتم خاتمه می‌یابد، پسته  $S$  دقیقاً شامل رئوس  $\text{CH}(Q)$  است (به ترتیب عکس جهت عقربه‌های ساعت قرار گرفتن آن‌ها در مرز).

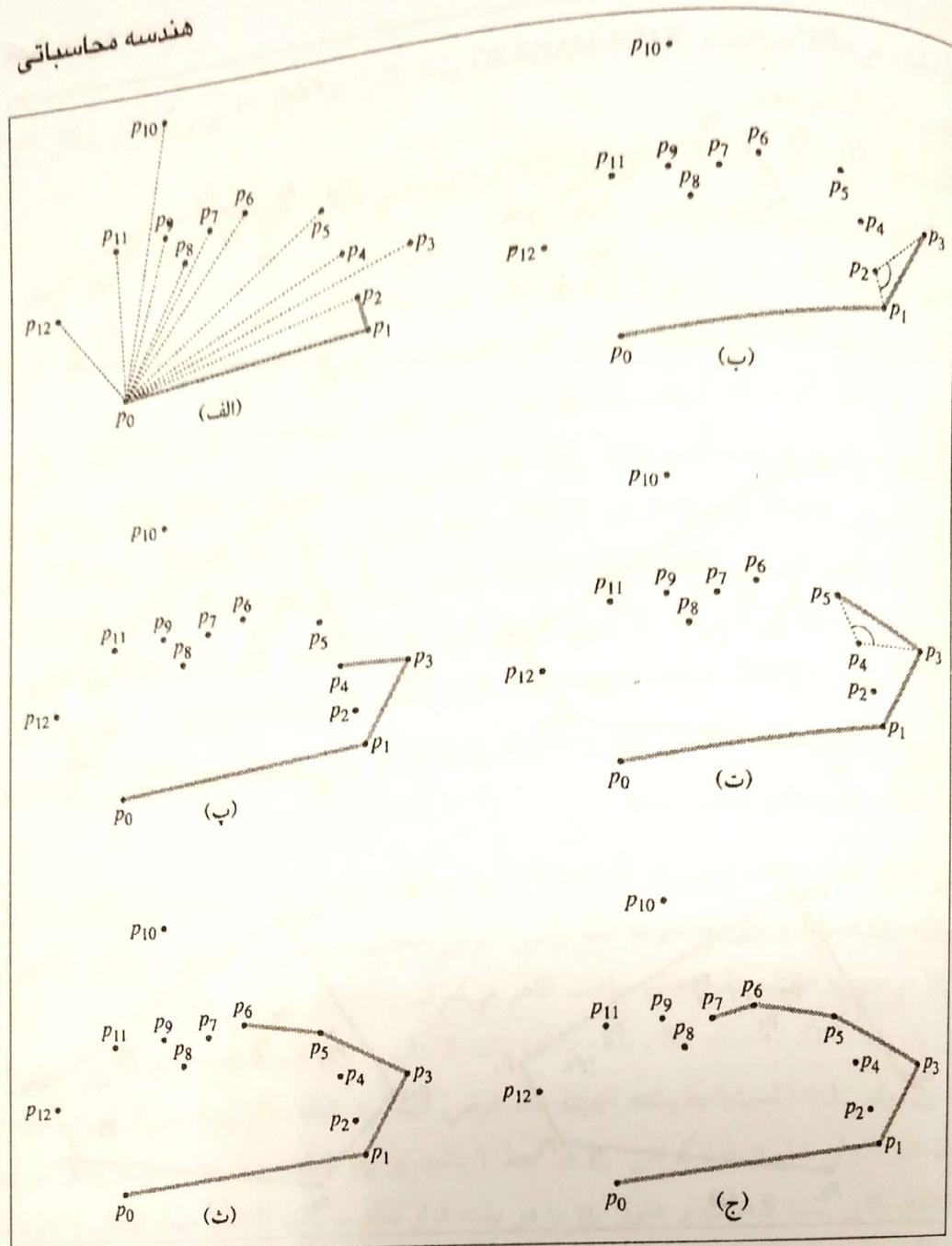
رویه GRAHAM-SCAN، مجموعه نقاط  $Q$  را به عنوان ورودی دریافت می‌کند که  $|Q| \geq 3$ .تابع NEXT-TO-TOP( $S$ ) را فراخوانی می‌کند تا عنصر بالای پسته را بدون تغییر در پسته برگرداند، و تابع ( $S$ ) را فراخوانی می‌کند که نقطه‌ی موجود در زیر عنصر بالای پسته را، بدون تغییر پسته، برگرداند. همان‌طور که در ادامه اثبات می‌کنیم، پسته  $S$  که توسط GRAHAM-SCAN برگردانده می‌شود، از چپ به راست، دقیقاً شامل رئوس  $\text{CH}(Q)$  در جهت عکس عقربه‌های ساعت است:

### GRAHAM-SCAN( $Q$ )

- 1 let  $p_0$  be the point in  $Q$  with the minimum  $y$ -coordinate,  
or the leftmost such point in case of a tie
- 2 let  $(p_1, p_2, \dots, p_m)$  be the remaining points in  $Q$ ,  
sorted by polar angle in counterclockwise order around  $p_0$   
(if more than one point has the same angle, remove all but  
the one that is farthest from  $p_0$ )
- 3 PUSH( $p_0, S$ )
- 4 PUSH( $p_1, S$ )
- 5 PUSH( $p_2, S$ )
- 6 for  $i \leftarrow 3$  to  $m$
- 7     do while the angle formed by points NEXT-TO-TOP( $S$ ), TOP( $S$ ),  
            and  $p_i$  makes a nonleft turn
- 8         do POP( $S$ )
- 9         PUSH( $p_i, S$ )
- 10 return  $S$

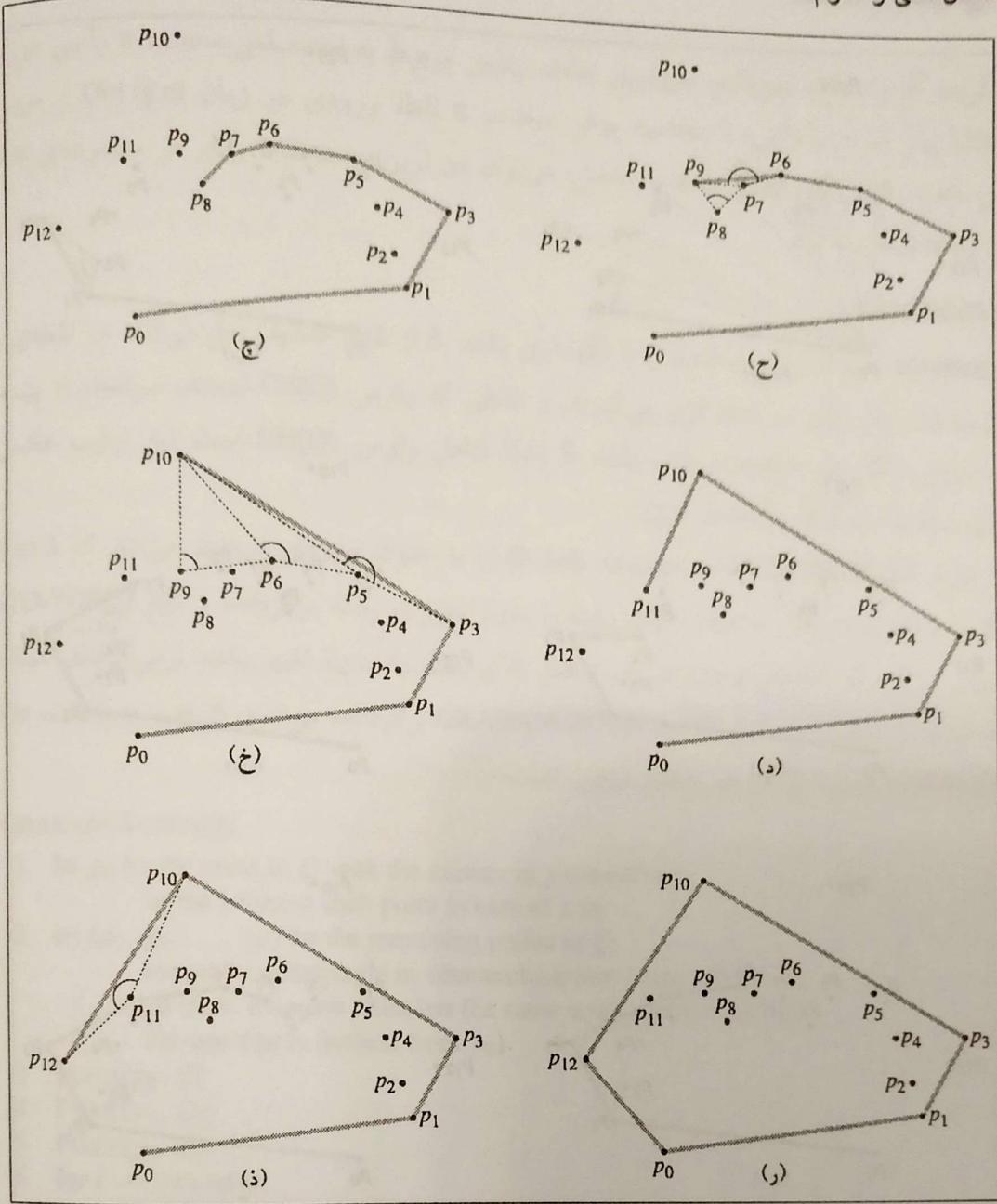
شکل ۳۳-۷ فرآیند GRAHAM-SCAN را نشان می‌دهد. خط ۱، نقطه  $p_0$  را به عنوان نقطه‌ای با کمترین مختص  $y$  انتخاب می‌کند، برای این کار، چپ‌ترین نقطه با این خاصیت را در مورد یک گره انتخاب می‌نماید. چون نقطه‌ای در  $Q$  وجود ندارد که زیر  $p_0$  باشد و نقاط دیگر با مختص  $y$  یکسان، در سمت راست آن قرار دارند،  $p_0$  رأسی از  $\text{CH}(Q)$  است. خط ۲، بقیه نقاط  $Q$  را به وسیله زاویه قطبی نسبت به  $p_0$ ، به روشنی مشابه با تمرین ۳۳-۱ مرتب می‌کند (با مقایسه حاصلضرب‌های خارجی). اگر دو یا چند نقطه، زاویه قطبی یکسانی نسبت به  $p_0$  داشته باشند، تمام این نقاط به جز دورترین نقطه با این خاصیت، ترکیب‌های محدب  $p_0$  و دورترین نقطه هستند، و در نتیجه هیچ کدام از آن‌ها را درنظر نمی‌گیریم.

1. stack



شکل ۳۳-۷ اجرای GRAHAM-SCAN روی مجموعه Q از شکل ۳۳-۶. پوش محدب فعلی که در پشت S قرار دارد، در هر مرحله با خاکستری نشان داده شده است. (الف) دنباله نقاط  $\langle p_0, p_1, p_2, \dots, p_{12} \rangle$  بر حسب افزایش درجه قطبی نسبت به  $p_0$  شماره‌گذاری شدند، و پشته اولیه S شامل  $p_0$ ,  $p_1$  و  $p_2$  است. (ب) تا (د) پشته S پس از هر تکرار حلقه for در خطوط ۶ تا ۹ خطوط نقطه‌چین، چرخش‌های غیرچپ را نشان می‌دهند، که باعث می‌شوند نقاط از پشته حذف گردند. به عنوان مثال، در قسمت (ج) راستگرد در زاویه  $\angle p_7 p_8 p_9$  موجب حذف  $p_8$  می‌شود و سپس راستگرد در زاویه  $\angle p_6 p_7 p_9$  موجب حذف  $p_7$  می‌شود. پوش محدب توسط رویه برگردانده می‌شود، که مثل شکل ۳۳-۶ است (شکل ادامه دارد...).

فرض کنید  $m$  تعداد نقاط به جز  $p_0$  باشد که باقی می‌مانند. زاویه قطبی هر نقطه در Q نسبت به  $p_0$ ، در فاصله‌ی نیمه‌باز  $[0, \pi]$  قرار دارد (زاویه قطبی بر حسب رادیان است). چون نقاط بر اساس زاویه‌های قطبی ذخیره می‌شوند، نسبت به  $p_0$  در خلاف جهت عقربه‌های ساعت ذخیره می‌گردند. این نقاط ذخیره شده را با  $\langle p_1, p_2, \dots, p_m \rangle$  نشان می‌دهیم. توجه کنید که نقاط  $p_1$  و  $p_m$  رئوس  $CH(Q)$  هستند (تمرین ۳۳-۱ را بینید). شکل ۳۳-۷ (الف) نقاط شکل ۳۳-۶ را نشان می‌دهد که بر حسب افزایش زاویه قطبی نسبت به  $p_0$  شماره‌گذاری شدند.



ادامه شکل ۳۳-۷

بقیه رویه از پشته  $S$  استفاده می‌کند. خطوط ۳ تا ۵ محتویات پشته را مقدار اولیه می‌دهد. برای این کار، از پایین به بالا نقاط  $p_0$ ,  $p_1$  و  $p_2$  را در آن قرار می‌دهد. شکل ۳۳-۷ (الف) پشته اولیه را نشان می‌دهد. حلقه for خطوط ۶ تا ۹، برای هر نقطه در دنباله  $\langle p_3, p_4, \dots, p_m \rangle$  یک بار تکرار می‌شود. هدف این است که پس از پردازش نقطه  $p_i$ ، پشته  $S$  از پایین به بالا حاوی رئوس  $\langle p_0, p_1, \dots, p_i \rangle$  در جهت خلاف عقربه‌های ساعت باشد. حلقه while در خطوط ۷ و ۸، نقاطی را از پشته حذف می‌کنند که رئوس پوش محدب نیستند. وقتی پوش محدب را در خلاف جهت عقربه‌های ساعت پیمایش می‌کیم، در هر رأس، چپ‌گرد را انجام می‌دهیم. بنابراین، هر وقت حلقه while رأسی را می‌یابد که در آن چپ‌گرد را انجام می‌دهیم، این رأس از پشته حذف می‌شود (با بررسی چپ‌گرد، به جای فقط راست‌گرد، این تست، مانع از احتمال زاویه مستقیم در رأسی از پوش محدب حاصل می‌شود. زاویه‌های مستقیم را نمی‌خواهیم، زیرا هیچ رأسی از چندضلعی محدب نمی‌تواند ترکیب محدب سایر رئوس چندضلعی باشد). هنگام حرکت به سمت نقطه  $p_i$ ، وقتی تمام نقاط بدون چپ‌گرد را حذف کردیم،  $p_i$  را در پشته قرار می‌دهیم. شکل‌های ۳۳-۷ (ب) تا (ذ) حالت پشته  $S$  را پس از هر تکرار

حلقه for نشان می‌دهد. سرانجام، GRAHAM-SCAN پشته S را در خط ۱۰ برمی‌گرداند، شکل ۷ (۲۳-۷) پوش محدب را نشان می‌دهد.

قضیه زیر، درستی روش GRAHAM-SCAN را اثبات می‌کند.

### (Graham ۳۳-۱) درستی روش

قضیه ۱-۱ GRAHAM-SCAN روی مجموعه نقاط Q اجرا شود، که  $|Q| \geq 3$ ، آنگاه در پایان، پشته S، از چهار به راست، حاوی رئوس  $CH(Q)$  به ترتیب عکس عقربه‌های ساعت است.

اثبات: پس از خط ۲، دنباله  $\langle p_1, p_2, \dots, p_m \rangle$  را داریم. فرض کنید، برای  $i = 2, 3, \dots, m$ ،  $p_i$  مجموعه زاویه قطبی  $Q_i = \{p_0, p_1, \dots, p_i\}$  را تعریف کنیم. نقاط  $Q - Q_m$  آن‌ها بی هستند که خلاف شوند، زیرا نسبت به نقطه  $p_0$  زاویه قطبی یکسانی داشتند. این نقاط، در  $CH(Q)$  وجود ندارند، و در نتیجه  $CH(Q_m) = CH(Q) = CH(Q_{m-1})$ . بنابراین، کافی است نشان دهیم که وقتی GRAHAM-SCAN خاتمه می‌یابد، پشته S شامل رئوس  $CH(Q_m)$  به ترتیب عکس عقربه‌های ساعت، از پایین به بالا است. توجه کنید که  $p_0, p_1, \dots, p_m$  شامل رئوس  $CH(Q)$  هستند، و نقاط  $p_i$  و  $p_0, p_1, \dots, p_{i-1}$  رئوس  $CH(Q_i)$  هستند. این اثبات، از ثابت حلقه زیر استفاده می‌کند:

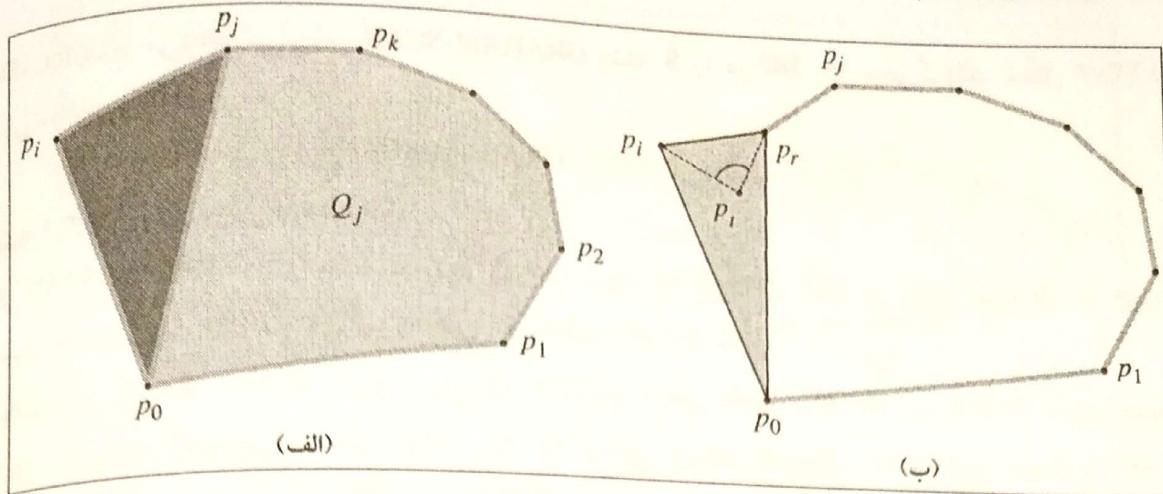
در شروع هر تکرار حلقه for در خطوط ۶ تا ۹، پشته S، از پایین به بالا شامل رئوس  $CH(Q_{i-1})$  در جهت خلاف عقربه‌های ساعت است.

نقداردهی اولیه: ثابت حلقه، اولین بار که خط ۶ اجرا می‌شود، برقرار است، زیرا در آن زمان، پشته S شامل رئوس  $Q_2 = Q_{i-1}$  است، و این مجموعه سه رأسی، پوش محدب خودش را تشکیل می‌دهد.علاوه براین، از پایین به بالا در جهت خلاف عقربه‌های ساعت ظاهر می‌شوند.

نگهداری: با ورود به تکرار حلقه for، نقطه بالای پشته S برابر با  $p_{i-1}$  است، که در انتهای تکرار قبلی (یا قبل از تکرار اول، وقتی  $i=3$  است) به پشته اضافه شد. فرض کنید  $p_j$  نقطه بالای پشته پس از اجرای حلقة while در خطوط ۷ و ۸ ولی قبل از قرارگرفتن  $p_i$  در خط ۹ باشد، و  $p_k$  نقطه زیر  $p_j$  در پشتے باشد. در لحظه‌ای که  $p_j$  نقطه بالای پشتے S است و هنوز  $p_i$  را در پشتے قرار ندادیم، پشتے S دقیقاً همان نقاطی را دارد که پس از تکرار  $j$ ام حلقه for داشته است. بنا به ثابت حلقه، S دقیقاً شامل رئوس  $CH(Q_j)$  در آن لحظه است، و از پایین به بالا به ترتیب عقربه‌های ساعت ظاهر می‌شوند.

اکنون به لحظه‌ی درست قبل از قرارگرفتن  $p_i$  در پشتے متمرکز می‌شویم. با مراجعه به شکل ۷-۸ (الف)، چون زاویه قطبی  $p_i$  نسبت به  $p_0$  بزرگ‌تر از زاویه قطبی  $p_j$  است، و چون زاویه  $p_k p_j p_i$  چپ‌گرد را انجام می‌دهد (در غیر این صورت  $p_j$  از پشتے حذف می‌شد)، می‌بینیم که چون S حاوی رئوس  $CH(Q_i)$  است، وقتی  $p_i$  را در پشتے قرار می‌دهیم، پشتے S دقیقاً شامل رئوس  $(CH(Q_i) \cup \{p_i\})$  خواهد بود، که از پایین به بالا در جهت خلاف عقربه‌های ساعت است.

اکنون نشان می‌دهیم که  $CH(Q_j \cup \{p_i\})$  مجموعه نقاطی مشابه با  $CH(Q_i)$  است. نقطه‌ای مثل  $p_i$  را در نظر بگیرید که در اثنای تکرار  $i$ ام حلقه for از پشتے حذف شد، و فرض کنید  $p_r$  نقطه‌ای زیر  $p_i$  در پشتے S در زمانی باشد که  $p_i$  از پشتے حذف شد ( $p_r$  ممکن است  $p_j$  باشد). زاویه  $p_r p_i p_j$  چرخش غیرچپ را انجام می‌دهد، و زاویه قطبی  $p_t$  نسبت به  $p_0$  بزرگ‌تر از زاویه قطبی  $p_r$  است. همان‌طور که شکل ۷-۸ (ب) نشان می‌دهد،  $p_t$  باید در داخل مثلث حاصل از  $p_0, p_r$  و  $p_i$  باشد، یا روی ضلع مثلث باشد (ولی رأس مثلث نیست).



شکل ۳۳-۸ اثبات درستی GRAHAM-SCAN . (الف) چون زاویه قطبی  $p_i$  نسبت به  $p_0$  بزرگتر از زاویه قطبی  $p_j$  است، و چون زاویه  $\angle p_k p_j p_i$  چپگرد را انجام می‌دهد، با اضافه شدن  $p_i$  به  $CH(Q_j \cup \{p_i\})$ ، رئوس  $CH(Q_j \cup \{p_i\})$  به دست می‌آید. اگر زاویه  $\angle p_r p_i p_j$  چرخش غیرچپ را انجام دهد، آنگاه  $p_i$  در داخل مثلث حاصل از  $p_0, p_r$  و  $p_i$  قرار دارد، یا روی ضلع مثلث واقع است، و نمی‌تواند رأسی از  $CH(Q_i)$  باشد.

روشن است که چون  $p_i$  در داخل مثلثی قرار دارد که با سه نقطه دیگر  $Q_i$  تشکیل شد، نمی‌تواند رأسی از  $CH(Q_i)$  باشد. چون  $p_t$  رأسی از  $CH(Q_i)$  نیست، داریم:

$$CH(Q_i - \{p_t\}) = CH(Q_i) \quad (33-1)$$

فرض کنید  $p_i$  مجموعه‌ای از نقاط باشد که در حین تکرار  $i$  حلقه for از پشته حذف شدند. چون معادله (33-1) برای تمام نقاط  $p_i$  اجرا می‌شود، می‌توان آن را مکرراً اجرا کرد تا نشان داد که  $CH(Q_i - P_i) = CH(Q_i - P_i) = CH(Q_i)$ . اما  $CH(Q_j \cup \{p_i\}) = CH(Q_i - P_i) = CH(Q_i)$  که  $Q_i - P_i = Q_j \cup \{p_i\}$ .  $Q_i - P_i$ ، و بنابراین، نتیجه می‌گیریم که خاتمه: وقتی حلقه خاتمه می‌یابد، داریم  $i = m + 1$ ، و در نتیجه ثابت حلقه دلالت می‌کند که پشته  $S$  شامل دقیقاً رئوس  $CH(Q_m)$  است، که  $CH(Q)$  می‌باشد (از پایین به بالا در خلاف جهت عقربه‌های ساعت). به این ترتیب، اثبات کامل می‌شود. ■

اکنون نشان می‌دهیم که زمان اجرای GRAHAM-SCAN برابر با  $O(n \lg n)$  است، که  $|Q| = n$ . خط ۱ به زمان  $\Theta(n)$  نیاز دارد. خط ۲ به زمان  $O(n \lg n)$  نیاز دارد، که با استفاده از مرتب‌سازی ادغام یا هیب، زاویه‌های قطبی را مرتب می‌کند، و با استفاده از روش حاصل‌ضرب خارجی، زاویه‌ها را مقایسه می‌کند (حذف تمام نقاط به جز دورترین نقطه، با زاویه قطبی یکسان، می‌تواند در زمان  $O(n)$  انجام شود). خطوط ۳ تا ۵، به زمان  $O(1)$  نیاز دارند. چون PUSH به زمان  $O(1)$  نیاز دارد، هر تکرار، صرف‌نظر از زمان مصرف‌شده برای حلقه while در خطوط ۷ تا ۸، به زمان  $O(1)$  نیاز دارد، و در نتیجه کل حلقه for، صرف‌نظر از حلقه while تودرتو، به زمان  $O(n)$  نیاز دارد.

با استفاده از تحلیل جمعی<sup>1</sup> نشان می‌دهیم که حلقه while به زمان  $O(n)$  نیاز دارد. برای  $i = 0, 1, \dots, m$  هر نقطه  $p_i$  دقیقاً یک بار در پشته قرار می‌گیرد. همانند تحلیل رویه MULTIPUSH در بخش ۱۷-۱، مشاهده می‌کنیم که برای هر عمل PUSH، حداقل یک عمل POP وجود دارد. حداقل سه نقطه  $p_0, p_1$  و  $p_m$  از پشته

1. aggregate

حذف نمی شوند، به طوری که حداکثر  $2^m$  عمل POP اجرا می شود. هر تکرار حلقه while، یک POP را ایام می دهد، و در نتیجه حداکثر  $2^m$  تکرار حلقه while وجود دارد. چون تست در خط ۷ در زمان  $O(1)$  انجام می شود، هر فرآخوانی POP به زمان  $O(1)$  نیاز دارد، و چون  $m \leq n-1$ ، کل زمان حلقه while برابر با GRAHAM-SCAN است. بنابراین، زمان اجرای  $O(n \lg n)$  است.

### Jarvis

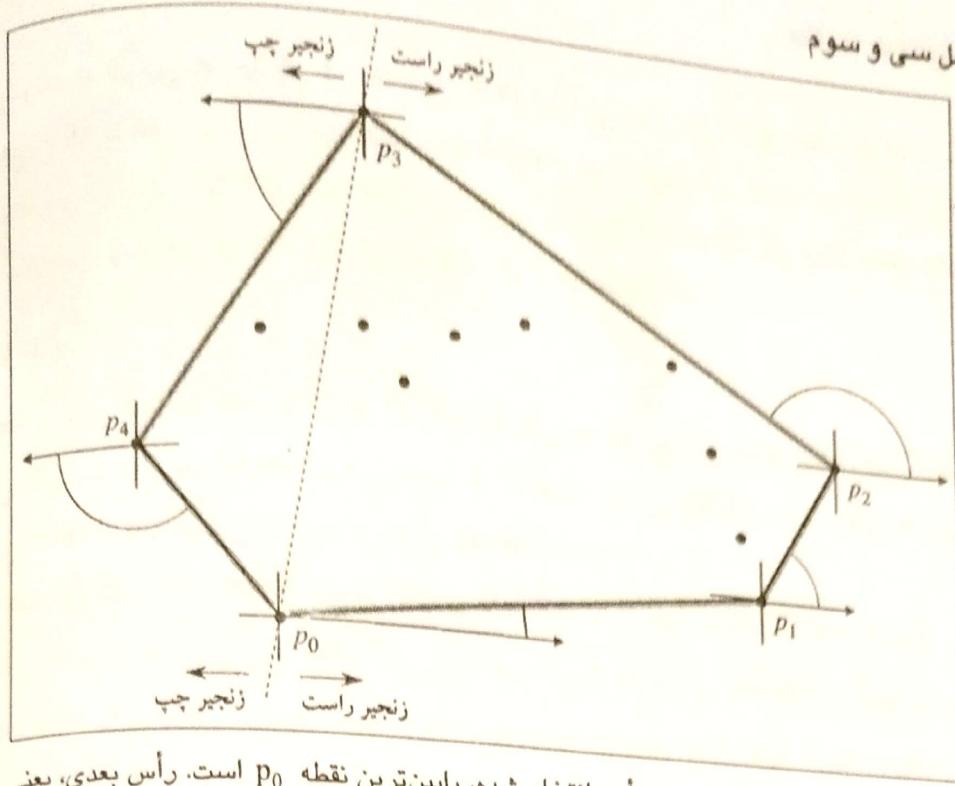
روش Jarvis، پوش محدب مجموعه نقاط Q را با تکنیکی به نام روکش بسته<sup>۱</sup> (یا روکش هدیه) استفاده می کند. این الگوریتم در زمان  $O(nh)$  اجرا می شود، که  $h$  تعداد رئوس  $CH(Q)$  است. وقتی  $h$  برابر با  $O(\lg n)$  است، روش Jarvis از نظر مجانبی سریع تر از روش Graham است. از نظر شهودی، روش Jarvis، روش تکه‌ای کاغذ فشرده را حول مجموعه Q شبیه‌سازی می کند. برای

شروع، انتهای کاغذ را در پایین ترین نقطه، یعنی نقطه‌ای مثل  $p_0$  که روش Graham از آنجا شروع شد، قرار می دهیم. این نقطه، رأس پوش محدب است. کاغذ را به سمت راست می کشیم تا فشرده شود، و سپس آن را بالاتر می کشیم تا با نقطه‌ای برخورد کند. این نقطه نیز باید رأسی از پوش محدب باشد. با فشرده نگهداشتن کاغذ، این کار را حول مجموعه‌ای از رئوس ادامه می دهیم تا به نقطه اولیه  $p_0$  برسیم.

به طور رسمی تر، روش Jarvis، دنباله  $H = \langle p_0, p_1, \dots, p_{n-1} \rangle$  را از رئوس  $CH(Q)$  می سازد. با  $p_0$  شروع می کنیم. همان‌طور که شکل ۳۳-۹ نشان می دهد، رأس بعدی پوش محدب، یعنی  $p_1$ ، دارای کوچکترین زاویه نسبت به  $p_0$  است (در مورد گره‌ها، دورترین نقطه از  $p_0$  را انتخاب می کنیم). به طور مشابه،  $p_2$  کوچکترین زاویه قطبی را نسبت به  $p_1$  دارد، و غیره. وقتی به بالاترین رأس، مثلاً  $p_k$  می رسیم (شکستن گره با انتخاب دورترین نقطه)، همان‌طور که شکل ۳۳-۹ نشان می دهد، زنجیر راست<sup>۲</sup>  $CH(Q)$  را ساختیم. برای ساخت زنجیر چپ، از  $p_k$  شروع می کنیم و  $p_{k+1}$  را به عنوان نقطه‌ای با کوچکترین زاویه قطبی نسبت به  $p_k$  انتخاب می کنیم، اما از جهت منفی محور  $x$ ، با ادامه کار، و با انتخاب زاویه‌های قطبی از جهت منفی محور  $x$ ، به رأس اصلی  $p_0$  می رسیم و زنجیر چپ تشکیل می شود.

روش Jarvis را می توان در یک جاروی ادراکی<sup>۳</sup> حول پوش محدب پیاده‌سازی کرد، یعنی لازم نیست زنجیرهای چپ و راست به طور جداگانه ساخته شوند. این پیاده‌سازی‌ها، معمولاً زاویه آخرین ضلع انتخاب شده‌ی پوش محدب را نگه می دارند، و لازم است دنباله‌ای از زاویه‌های اضلاع پوش، اکیداً صعودی باشند (در بازه  $0$  تا  $2\pi$  رادیان). امتیاز ساخت زنجیرهای جداگانه این است که لازم نیست زاویه‌ها صریحاً محاسبه شوند؛ و تکنیک‌های بخش ۳۳-۱ برای مقایسه زاویه‌ها کافی‌اند.

اگر روش Jarvis به طور مناسب پیاده‌سازی شود، در زمان  $O(nh)$  اجرا می گردد. برای هر  $h$  رأس  $CH(Q)$ ، رأسی با کمترین زاویه قطبی را پیدا می کنیم. هر مقایسه بین زاویه‌های قطبی، با استفاده از تکنیک‌های بخش ۳۳-۱، به زمان  $O(1)$  نیاز دارد، اگر هر مقایسه در زمان  $O(1)$  انجام شود، کمترین مقدار از بین  $n$  مقدار را می توان در زمان  $O(n)$  پیدا کرد. بنابراین، روش Jarvis در زمان  $O(nh)$  انجام می شود.



شکل ۳۳-۹: عملکرد روش Jarvis. اولین رأس انتخاب شده، پایین‌ترین نقطه  $p_0$  است. رأس بعدی، یعنی  $p_1$ ، دارای کوچکترین زاویه قطبی نسبت به  $p_0$  است. سپس  $p_2$  دارای کوچکترین زاویه قطبی نسبت به  $p_1$  است. رُنجیر راست تا بالاترین نقطه  $p_3$  بالا می‌رود. سپس، رُنجیر چپ، با یافتن کوچکترین زاویه‌های قطبی نسبت به جهت منفی محور X، ساخته می‌شود.

### تمرین‌های بخش ۳۳-۳

تمرین ۱-۳-۱: ثابت کنید در رویه GRAHAM-SCAN، نقاط  $p_1$  و  $p_m$  باید رئوس از  $\text{CH}(Q)$  باشند.

تمرین ۲-۳-۲: مدل محاسبه‌ای را درنظر بگیرید که از جمع، ضرب، و مقایسه پشتیبانی می‌کند، و برای آن، حد پایین

$\Omega(n \lg n)$  برای مرتب‌سازی n عدد وجود دارد. ثابت کنید، در این مدل، حد پایین محاسبه رئوس پوش

محدب از مجموعه n نقطه‌ای،  $\Omega(n \lg n)$  است.

تمرین ۳-۳-۳: با توجه به مجموعه‌ای از نقاط Q، ثابت کنید جفتی از نقاط با بیشترین فاصله از یکدیگر، باید رئوس از  $\text{CH}(Q)$  باشند.

تمرین ۴-۳-۴: برای چندضلعی P و نقطه q روی مرز آن، سایه<sup>۱</sup> q، مجموعه‌ای از نقاط ۲ است که پاره خط  $\overline{qP}$  کاملاً

روی مرز یا در داخل P است. اگر نقطه‌ای مثل p در داخل چندجمله‌ای P باشد که در سایه هر نقطه

روی مرز P باشد، این چندجمله‌ای را ستاره‌شکل<sup>۲</sup> می‌نامند. مجموعه‌ای از نقاط p، هسته‌ی<sup>۳</sup>

نامیده می‌شود (شکل ۳۳-۱۰). با توجه به چندجمله‌ای n رأسی ستاره‌شکل که با رئوس خود به ترتیب

عکس عقربه‌های ساعت مشخص شد، نشان دهید که  $\text{CH}(P)$  می‌تواند در زمان  $O(n)$  انجام شود.

تمرین ۵-۳-۳: در مسئله پوش محدب online، مجموعه n نقطه‌ای Q، به صورت هر بار یک نقطه، داده شده است.

پس از دریافت یک نقطه، پوش محدب نقطی را که تاکنون دیده شدن، محاسبه می‌کنیم. بدیهی است

که روش Graham را می‌توان برای هر نقطه یک بار اجرا کرد، که کل زمان آن  $O(n^2 \lg n)$  است

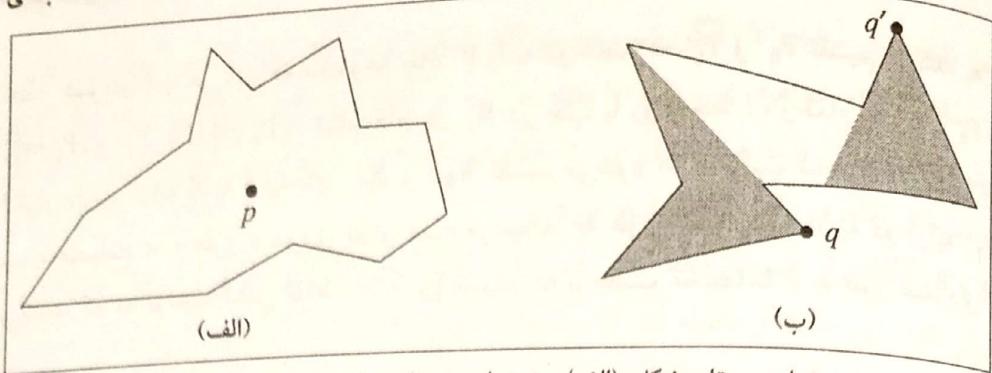
نشان دهید که چگونه می‌توان مسئله پوش محدب online را در زمان  $O(n^2)$  حل کرد.

\* تمرین ۶-۳-۳: نشان دهید که چگونه می‌توان روش افزایشی را برای محاسبه پوش محدب از n نقطه پیاده‌سازی کرد به طوری که در زمان  $O(n \lg n)$  اجرا شود.

1. shadow

2. star-shaped

3. kernel



شکل ۳۳-۱۰ تعریف چندضلعی ستاره‌شکل. (الف) چندضلعی ستاره‌شکل. پاره‌خطی از نقطه  $p$  به هر نقطه  $q$  روی مرز، مرز را فقط در  $q$  قطع می‌کند. (ب) چندضلعی غیرستاره‌شکل. ناحیه سایه‌دار در سمت چپ، سایه  $q$  و ناحیه سایه‌دار در سمت راست، سایه  $q'$  است. چون ادعا نموده شده که این دو ناحیه‌ها جدا از هم هستند، هسته تهی است.

### ۳۳-۴ یافتن نزدیک‌ترین جفت نقاط

اکنون مسئله‌ی یافتن نزدیک‌ترین جفت نقاط را در مجموعه  $Q$  با  $2 \leq n$  نقطه، درنظرمی‌گیریم. منظور از "نزدیک‌ترین"، فاصله اقلیدسی معمولی است: فاصله بین نقاط  $(x_1, y_1) = p_1$  و  $(x_2, y_2) = p_2$  برابر است با  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ . در نقطه در مجموعه  $Q$  ممکن است منطبق<sup>۱</sup> (یکسان) باشند، که در این صورت فاصله آن‌ها صفر است. این مسئله، کاربردهایی در سیستم‌های کنترل ترافیک دارد. در سیستم کنترل ترافیک دریابی یا هوابی، ممکن است لازم باشد بدانند که کدام دو وسیله نقلیه نزدیک‌ترین فاصله را به یکدیگر دارند، تا احتمال برخورد تعیین شود. الگوریتم غیرهوشمند<sup>۲</sup> (جست‌وجوی جامع) برای نزدیک‌ترین جفت، تمام  $\Theta(n^2) = \binom{n}{2}$  جفت از نقاط را نگاه می‌کند. در این بخش، الگوریتم تقسیم و حل را برای این مسئله ارائه می‌کنیم که زمان اجرای آن توسط عبارت بازگشتی معروف  $T(n) = 2T(n/2) + O(n)$  توصیف می‌شود. بنابراین، این الگوریتم در زمان  $O(n \lg n)$  اجرا می‌شود.

### الگوریتم تقسیم و حل

هر فراخوانی بازگشتی الگوریتم، زیرمجموعه  $Q \subseteq P$ ، و آرایه‌های  $X$  و  $Y$  را که هر کدام شامل تمام نقاط زیرمجموعه ورودی  $P$  هستند، به عنوان ورودی دریافت می‌کند. نقاط در آرایه  $X$  طوری مرتب‌اند که مختصات  $x$  آن‌ها صعودی یکنواخت‌اند. به طور مشابه، آرایه  $Y$  طوری مرتب شده است که مختصات  $y$  آن‌ها صعودی یکنواخت‌اند. توجه کنید که برای به‌دست آوردن حد زمانی  $O(n \lg n)$ ، در هر فراخوانی بازگشتی نمی‌توانیم مرتب‌سازی را انجام دهیم؛ اگر انجام دهیم، عبارت بازگشتی مربوط به زمان اجرا به صورت  $T(n) = 2T(n/2) + O(n \lg n)$  می‌شود، که جواب آن  $O(n \lg^2 n) = O(n \lg n)$  است. در ادامه خواهید دید که چگونه می‌توان از "پیش‌مرتب‌سازی" استفاده کرد تا این خاصیت مرتب‌شده باقی بماند، و در هر فراخوانی بازگشتی مرتب‌سازی انجام نشود.

یک فراخوانی بازگشتی خاص با ورودی‌های  $P$ ،  $X$  و  $Y$ ، ابتدا بررسی می‌کند آیا  $|P| \leq 3$  است یا خیر. اگر باشد، این فراخوانی، روش غیرهوشمند توصیف‌شده در بالا را اجرا می‌کند. تمام  $\binom{|P|}{2}$  جفت از نقاط را بررسی می‌کند و نزدیک‌ترین جفت را برمی‌گرداند. اگر  $|P| > 3$ ، فراخوانی بازگشتی، الگوریتم تقسیم و حل را به صورت زیر دنبال می‌کند:

1. coincident

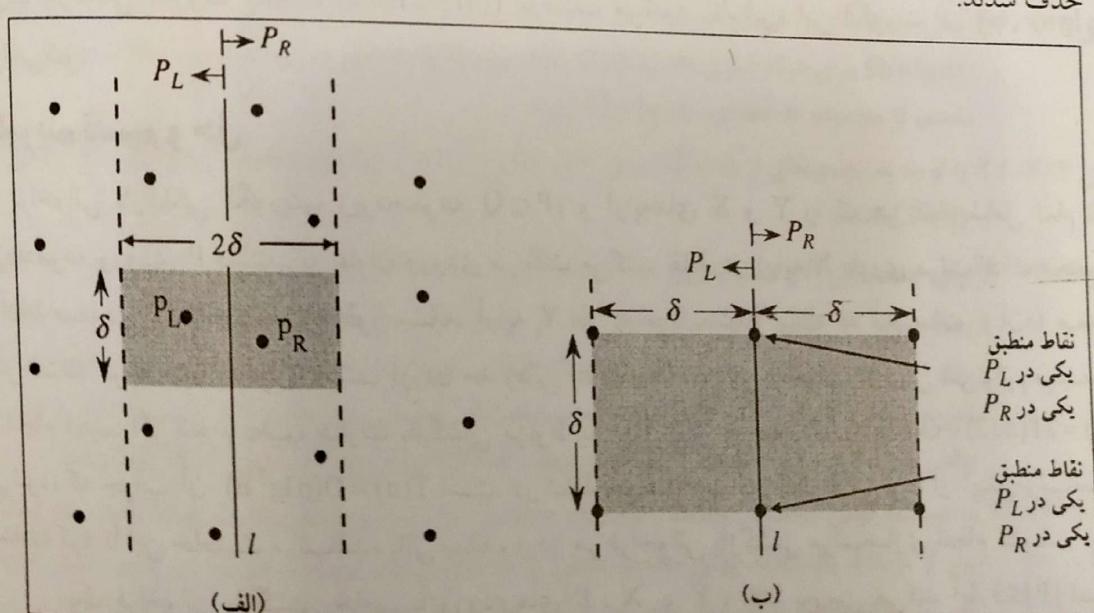
2. brute force

تقسیم: خط عمودی  $/$  را می‌یابد که مجموعه نقاط  $P$  را به دو مجموعه  $P_L$  و  $P_R$  تقسیم می‌کند، به طوری که تمام نقاط در  $P_L$  در چپ یا روی خط  $/$ ، و تمام نقاط در  $P_R$  در راست  $/$  قرار دارند. آرایه  $X$  به آرایه‌های  $X_L$  و  $X_R$  تقسیم می‌شود که به ترتیب شامل نقاط  $P_L$  و  $P_R$  هستند یا روی  $/$  قرار دارند. آرایه  $Y$  به آرایه‌های  $Y_L$  و  $Y_R$  تقسیم می‌شود که به ترتیب شامل نقاط  $P_L$  و  $P_R$  هستند و بر حسب مختصات  $x$  به طور صعودی یکنواخت، مرتب‌اند. به طور مشابه، آرایه  $Y$  به دو آرایه  $Y_L$  و  $Y_R$  تقسیم می‌شود، که به ترتیب شامل نقاط  $P_L$  و  $P_R$  است، که بر حسب مختصات  $y$  به طور صعودی یکنواخت مرتب‌اند.

حل: پس از تقسیم  $P$  به  $P_L$  و  $P_R$ ، دو فراخوانی بازگشتی انجام می‌شود، یکی برای یافتن نزدیک‌ترین جفت نقاط در  $P_L$  و دیگری برای یافتن نزدیک‌ترین جفت نقاط در  $P_R$ . ورودی‌های فراخوانی اول، زیرمجموعه  $P_L$  و آرایه‌های  $X_L$  و  $Y_L$  و ورودی‌های فراخوانی دوم  $P_R$ ،  $X_R$  و  $Y_R$  هستند. فرض کنید فاصله‌های نزدیک‌ترین جفت‌های برگردانده شده برای  $P_L$  و  $P_R$  به ترتیب  $\delta_L$  و  $\delta_R$  باشند، و  $\delta = \min(\delta_L, \delta_R)$ .

ترکیب: نزدیک‌ترین جفت، یا جفتی با فاصله  $\delta$  است که توسط یکی از فراخوانی‌های بازگشتی پیدا شد، یا جفتی از نقاط است که یک نقطه در  $P_L$  و نقطه دیگر در  $P_R$  است. الگوریتم، تعیین می‌کند آیا چنین جفتی وجود دارد که فاصله آن کمتر از  $\delta$  باشد یا خیر. مشاهده می‌کنید که اگر جفتی از نقاط با فاصله کمتر از  $\delta$  وجود داشته باشد، هر دو نقطه‌ی جفت باید در داخل  $\delta$  واحد از خط  $/$  باشند. بنابراین همانند شکل ۳۳-۱۱ (الف)، هر دو باید در نوار عمودی با پهنای  $2\delta$  به مرکز خط  $/$  باشند. برای یافتن چنین جفتی، در صورت

وجود، الگوریتم به صورت زیر عمل می‌کند:  
۱. آرایه  $Y$  را ایجاد می‌کند، که همان آرایه  $Y$  است که نقاطی که در نوار عمودی با پهنای  $2\delta$  نیستند، از آن حذف شدند.



شکل ۳۳-۱۱ مفهوم کلیدی در اثبات این‌که الگوریتم نزدیک‌ترین جفت فقط پس از هر نقطه در آرایه  $Y$  را بررسی کند. (الف) اگر  $p_L \in P_L$  و  $p_R \in P_R$  کمتر از  $\delta$  واحد از هم دور باشند، باید در داخل چهارگوش  $2\delta \times \delta$  به مرکز خط  $/$  باشند. (ب) چگونه ۴ نقطه که دو به دو حداقل در فاصله  $\delta$  واحدی قرار دارند، می‌توانند در مربع  $\delta \times \delta$  باشند در سمت چپ، ۴ نقطه در  $P_L$  وجود دارند، و در سمت راست، ۴ نقطه در  $P_R$  هستند. اگر نقاط نشان‌داده شده روی خط  $/$  در واقع جفت‌هایی از نقاط منطبق (یکسان) باشند که یک نقطه در  $P_L$  و نقطه دیگر در  $P_R$  باشد، می‌تواند ۸ نقطه در چهارگوش  $2\delta \times 2\delta$  وجود داشته باشد.

۱. برای هر نقطه  $p$  در آرایه ' $Y$ '، الگوریتم سعی می‌کند نقاطی را در ' $Y$ ' بیابد که در داخل  $\delta$  واحد از  $p$  قرار دارند. همان‌طور که خواهید دید، فقط ۷ نقطه در ' $Y$ ' که بعد از  $p$  قرار دارند، باید در نظر گرفته شوند. الگوریتم، فاصله را از  $p$  به هر یک از این ۷ نقطه پیدا می‌کند و فاصله نزدیک‌ترین جفت  $\delta$  پیدا شده در نام جفت‌های نقاط در ' $Y$ ' را نگهداری می‌کند.

۲. اگر  $\delta > \delta'$ ، آنگاه نوار عمودی فاقد جفتی است که نسبت به آن که توسط فراخوانی‌های بازگشته پیدا شده، نزدیک‌تر باشد. این جفت و فاصله آن  $\delta'$  برگردانده می‌شوند. و گرنه، نزدیک‌ترین جفت و فاصله آن  $\delta$  که توسط فراخوانی‌های بازگشته پیدا شده، برگردانده می‌شوند.

ابن توصیف، بعضی از جزیيات پیاده‌سازی را که برای دستیابی به زمان اجرای  $O(n \lg n)$  لازم است، حذف کرده است. پس از اثبات درستی الگوریتم، بحث می‌کنیم که چگونه می‌توان الگوریتم را پیاده‌سازی کرد تا حد زمان مطلوب به دست آید.

### درستی الگوریتم

درستی این الگوریتم نزدیک‌ترین جفت، به استثنای دو جنبه آن، روشن است. جنبه اول این است که، با کم کردن بازگشته در حالتی که  $|P| \leq 3$  است، تضمین می‌شود که سعی نمی‌کنیم زیرمسئله‌ی حاوی فقط یک نقطه را حل کنیم. جنبه دوم این است که فقط لازم است ۷ نقطه پس از هر نقطه  $p$  در آرایه ' $Y$ ' بررسی شود؛ این خاصیت را اثبات خواهیم کرد.

فرض کنید در سطحی از بازگشته، نزدیک‌ترین جفت نقاط،  $P_L \in P_L$  و  $P_R \in P_R$  باشند. بنابراین، فاصله  $\delta'$  بین  $P_L$  و  $P_R$  اکیداً کمتر از  $\delta$  است. نقطه  $P_L$  باید در چپ یا روی خط / و کمتر از  $\delta$  واحد از آن باشد. به طور مشابه،  $P_R$  در راست یا روی / و کمتر از  $\delta$  واحد از آن باشد. علاوه براین،  $P_L$  و  $P_R$  در داخل  $\delta$  واحد از یکدیگر به طور عمودی قرار دارند. بنابراین، همان‌طور که شکل ۱۱-۳۳ (الف) نشان می‌دهد،  $P_L$  و  $P_R$  در داخل چهارگوش  $2\delta \times 2\delta$  قرار دارند که مرکز آن / است (ممکن است نقاط دیگری در این چهارگوش باشند).

سپس نشان می‌دهیم که حداقل  $8$  نقطه از  $P$  می‌توانند در این چهارگوش  $2\delta \times 2\delta$  وجود داشته باشند. مربع  $8 \times 8$  را در نظر بگیرید که نیمه‌ی چپ این چهارگوش را تشکیل می‌دهد. چون تمام نقاط در  $P_L$  حداقل در فاصله  $\delta$  واحد از هم قرار دارند، حداقل  $4$  نقطه می‌توانند در این مربع باشد (شکل ۱۱-۳۳ (ب)). به طور مشابه، حداقل  $4$  نقطه در  $P_R$ ، می‌توانند در مربع  $8 \times 8$  باشند که نیمه‌ی راست چهارگوش را تشکیل می‌دهند. بنابراین، حداقل  $8$  نقطه از  $P$  می‌توانند در چهارگوش  $2\delta \times 2\delta$  باشد (توجه کنید که چون نقاط روی خط / ممکن است  $P_L$  یا  $P_R$  باشند، ممکن است تا  $4$  نقطه روی / باشند). این حد، در صورتی تأمین می‌شود که دو جفت از نقاط منطبق وجود داشته باشند که هر جفت شامل یک نقطه از  $P_L$  و یک نقطه از  $P_R$  است، یک جفت در تقاطع / و بالای چهارگوش قرار دارد، جفت دیگر در جایی واقع است که / پایین چهارگوش را قطع می‌کند).

با توجه به این که نشان داده شد حداقل  $8$  نقطه از  $P$  می‌توانند در چهارگوش باشند، به آسانی می‌توان دید که فقط لازم است ۷ نقطه پس از هر نقطه در آرایه ' $Y$ ' بررسی شود. هنوز، با فرض این که نزدیک‌ترین جفت،  $P_L$  و  $P_R$  است، بدون از دست دادن کلیت، فرض کنید  $p_L$  قبل از  $p_R$  در آرایه ' $Y$ ' قرار دارد. سپس،

حتی اگر  $p_L$  حتی الامکان در ابتدای  $Y'$  باشد، و  $p_R$  حتی الامکان در انتهای باشد،  $p_R$  یکی از ۷ موقعیت ممکن پس از  $p_L$  است. بنابراین، درستی الگوریتم نزدیک‌ترین جفت را نشان دادیم.

### پیاده‌سازی و زمان اجرا

همان‌طور که گفته شد، هدف، داشتن عبارت بازگشتی است تا این‌که زمان اجرا برابر با  $T(n) = 2T(n/2) + O(n)$  باشد، که  $T(n)$  زمان اجرا برای مجموعه  $n$  نقطه‌ای است. مشکل اصلی، تضمین این نکته است که آرایه‌های  $X_L, X_R, Y_L$  و  $Y_R$ ، که به فراخوانی‌های بازگشتی ارسال می‌شوند، بر حسب مختصات مناسبی مرتب باشند، و آرایه  $Y'$  نیز بر حسب مختصات  $Y$  مرتب باشد (توجه کنید که اگر آرایه  $X$  که توسط فراخوانی بازگشتی در زمان خطی انجام می‌شود).

دریافت می‌شود، مرتب باشد، آنگاه تقسیم مجموعه  $P$  به  $P_L$  و  $P_R$  در زمان خطی انجام می‌شود. مشاهده مهم این است که در هر فراخوانی، می‌خواهیم یک زیرمجموعه مرتب از آرایه مرتب را ایجاد کنیم. به عنوان مثال، فراخوانی خاصی که زیرمجموعه  $P$  و آرایه  $Y$  را می‌گیرد، بر حسب مختصات  $y$  مرتب است. با تقسیم  $P$  به  $P_L$  و  $P_R$ ، لازم است آرایه‌های  $Y_L$  و  $Y_R$  تشکیل شوند، که بر حسب مختصات  $y$  مرتب‌اند. علاوه‌براین، این آرایه‌ها باید در زمان خطی ایجاد شوند. رویه زیر این ایده را ارائه می‌دهد:

```

1  $length[Y_L] \leftarrow length[Y_R] \leftarrow 0$ 
2 for  $i \leftarrow 1$  to  $length[Y]$ 
3   do if  $Y[i] \in P_L$ 
4     then  $length[Y_L] \leftarrow length[Y_L] + 1$ 
5      $Y_L[length[Y_L]] \leftarrow Y[i]$ 
6   else  $length[Y_R] \leftarrow length[Y_R] + 1$ 
7      $Y_R[length[Y_R]] \leftarrow Y[i]$ 
```

نقاط موجود در آرایه  $Y$  را به ترتیب بررسی می‌کنیم. اگر نقطه  $Y[i]$  در  $P_L$  باشد، آن را به انتهای آرایه  $Y_L$ ، وگرنه آن را به انتهای آرایه  $Y_R$  اضافه می‌کنیم. شبکه کد مشابهی برای ایجاد آرایه‌های  $X_L, X_R$  و  $Y'$  به کار می‌رود. تنها پرسش باقیمانده، چگونگی دریافت نقاط به صورت مرتب است. این کار را با پیش‌مرتب‌سازی آن‌ها انجام می‌دهیم. یعنی، آن‌ها را یک بار، و قبل از اولین فراخوانی بازگشتی مرتب می‌کنیم. این آرایه‌های مرتب شده، به اولین فراخوانی بازگشتی ارسال می‌شوند، و از آنجا به بعد، در صورت لزوم، از طریق فراخوانی‌های بازگشتی کوچک می‌شوند. پیش‌مرتب‌سازی، زمان  $O(n \lg n)$  دیگری را به زمان اجرا اضافه می‌کند، ولی اکنون هر مرحله از بازگشتی، صرف‌نظر از فراخوانی‌های بازگشتی، در زمان خطی انجام می‌شود. بنابراین، اگر فرض کنیم  $T(n)$  زمان اجرای هر مرحله بازگشتی، و  $T'(n) = T(n) + O(n \lg n)$  زمان اجرای کل الگوریتم باشد، داریم:

$$T(n) = \begin{cases} 2T(n/2) + O(n) & \text{اگر } n > 3 \\ O(1) & \text{اگر } n \leq 3 \end{cases}$$

بنابراین،  $T'(n) = O(n \lg n)$  و  $T(n) = O(n \lg n)$ .

### تمرین‌های بخش ۳۳-۴

تمرین ۱-۴-۳: پروفسور Smothers طرحی ارائه کرد که اجزه می‌دهد الگوریتم نزدیک‌ترین جفت فقط ۵ نقطه را پس از هر نقطه در آرایه  $Y'$  بررسی کند. ایده این است که همیشه نقاط روی خط  $l$  در مجموعه  $P_L$  قرار گیرند. به این ترتیب، جفت‌هایی از نقاط منطبق نمی‌توانند روی  $l$  باشند، به طوری که یک نقطه در  $P_L$  و نقطه دیگر در  $P_R$  باشد. بنابراین، حداقل ۶ نقطه می‌تواند در چهارگوش  $28 \times 8$  باشد. عیب طرح این پروفسور چیست؟

تمرین ۴-۲: بدون افزایش زمان اجرای مجانبی الگوریتم، نشان دهید که چگونه می‌توان تضمین کرد که مجموعه نقاط ارسال شده به اولین فرآخوانی بازگشتی فاقد نقاط منطبق است. ثابت کنید، در این صورت کافی است نقاط موجود در  $\Delta$  موقعیت آرایه پس از هر نقطه در آرایه  $Y'$  بررسی شود.

تمرین ۴-۳: فاصله بین دو نقطه می‌تواند به غیر از روش اقلیدس تعریف شود. در صفحه، فاصله  $L_m$  بین نقاط  $p_1$  و  $p_2$  با عبارت  $(|x_1 - x_2|^m + |y_1 - y_2|^m)^{1/m}$  مشخص می‌شود. بنابراین، فاصله اقلیدسی، فاصله  $L_2$  است. الگوریتم نزدیک‌ترین جفت را تغییر دهید تا از فاصله  $L_1$  استفاده کند، که فاصله مانهاتان<sup>۱</sup> نیز نام دارد.

تمرین ۴-۴: با توجه به دو نقطه  $p_1$  و  $p_2$  در صفحه، فاصله  $L_\infty$  بین آن‌ها با  $\max(|x_1 - x_2|, |y_1 - y_2|)$  مشخص می‌شود. الگوریتم نزدیک‌ترین جفت را تغییر دهید تا از فاصله  $L_\infty$  استفاده کند.

تمرین ۴-۵: تغییری را در الگوریتم نزدیک‌ترین جفت پیشنهاد کنید که از پیش‌مرتب‌سازی آرایه  $Y$  اجتناب می‌کند، ولی زمان اجرا هنوز  $O(n \lg n)$  است (راهنمایی: آرایه‌های مرتب  $Y_L$  و  $Y_R$  را ادغام کنید تا آرایه مرتب  $Y$  به وجود آید).

## ۳۳-۵ مسئله‌ها

مسئله ۱: لایه‌های محدب.

با توجه به مجموعه  $Q$  از نقاط در صفحه، لایه‌های محدب<sup>۲</sup>  $Q$  را با استقرار تعریف می‌کنیم. اولین لایه محدب  $Q$  شامل نقاطی در  $Q$  است که رئوس  $CH(Q)$  هستند. برای  $i > 1$ ،  $Q_i$  شامل نقاطی از  $Q$  است که تمام نقاط موجود در لایه‌های محدب  $1-i, 2, \dots, Q_i \neq \emptyset$  حذف شدند. آنگاه، لایه محدب  $Q$  برابر با  $CH(Q_i)$  است، و گرنه تعریف شده نیست.

الف. یک الگوریتم زمان  $O(n^2)$  برای یافتن لایه‌های محدب مجموعه‌ای از  $n$  نقطه ارائه دهید.

ب. ثابت کنید زمان لازم برای محاسبه لایه‌های محدب مجموعه‌ای از نقاط با هر مدل محاسباتی که برای مرتب‌سازی  $n$  عدد حقیقی به زمان  $\Omega(n \lg n)$  نیاز دارد، برابر با  $\Omega(n \lg n)$  است.

مسئله ۲: لایه‌های ماکسیمال.

فرض کنید  $Q$  مجموعه‌ای از  $n$  نقطه در صفحه باشد. می‌گوییم نقطه  $(x, y)$  بر نقطه  $(x', y')$  غلبه می‌کند اگر  $x' \geq x$  و  $y' \geq y$ . نقاطی در  $Q$  که توسط هیچ نقطه دیگری در  $Q$  غلبه نمی‌شود، ماکسیمال<sup>۳</sup> نام دارد. توجه کنید که  $Q$  ممکن است چندین نقطه ماکسیمال داشته باشد، که می‌تواند به صورت زیر در لایه‌های ماکسیمال سازماندهی شود. برای  $i > 1$ ، لایه ماکزیمم  $i$ ، یعنی  $L_i$ ، مجموعه‌ای از نقاط ماکسیمال در  $Q - \bigcap_{j=1}^{i-1} L_j$ .

فرض کنید  $Q$  دارای  $k$  لایه ماکسیمال غیرتنهی است، و  $y_i$  مختصات  $y$  چپ‌ترین نقطه در  $L_i$  برای  $i = 1, 2, \dots, k$  باشد. اکنون، فرض کنید هیچ دو نقطه‌ای در  $Q$  مختصات  $x$  یا  $y$  یکسان ندارند.

الف. نشان دهید  $y_1 > y_2 > \dots > y_k$ .

نقطه  $(x, y)$  را درنظر بگیرید که در چپ هر نقطه‌ای در  $Q$  است و برای آن،  $y$  جدا از مختصات  $x$  هر نقطه در  $Q$  است. فرض کنید  $\{(x, y) \in Q\} = Q'$ .

ب. فرض کنید  $j$  اندیس مینیممی باشد که  $y < y_j$  به جز  $y_k < y$ ، که در این حالت قرار می‌دهیم  $j = k+1$ . نشان دهید که لایه‌های ماکسیمال  $Q'$  به صورت‌های زیر هستند:

اگر  $k \leq j$ ، آنگاه لایه‌های ماکسیمال  $Q'$  همانند لایه‌های ماکسیمال  $Q$  هستند، به استثنای این که  $z_i$  شامل

(x, y) به عنوان چپ‌ترین نقطه جدید خود است.

اگر  $1 \leq k < j$ ، آنگاه اولین  $k$  لایه ماکسیمال  $Q'$  همانند  $Q$  است، اما، علاوه بر این،  $Q'$  دارای لایه ماکسیمال

غیرتهی  $(k+1)$  است:  $L_{k+1} = \{(x, y)\}$ .

پ. یک الگوریتم  $O(n \lg n)$  برای محاسبه لایه‌های ماکسیمال مجموعه  $n$  نقطه‌ای  $Q$  توصیف کنید (راهنمایی).

یک خط جارو را از چپ به راست حرکت دهید.  
ت. اگر اجازه دهیم که نقطه‌های ورودی مختصات  $x$  یا  $y$  یکسان داشته باشند، آیا مشکلات خاصی پیش می‌آید؟

#### مسئله ۳-۲: Ghostbuster و ارواح.

گروهی از Ghostbuster، با  $n$  روح مبارزه می‌کنند. هر Ghostbuster به بسته‌ای پروتون مسلح است، که جریانی را به روح شوت می‌کند و آن را نابود می‌سازد. این جریان در یک خط مستقیم حرکت می‌کند و با رسیدن به روح متوقف می‌شود. Ghostbuster از روش زیر استفاده می‌کند. آنها با ارواح جفت می‌شوند و  $n$  جفت روح - Ghostbuster را تشکیل می‌دهند، و سپس همزمان هر Ghostbuster یک جریان به روح انتخابی خود می‌فرستد. همان‌طور که می‌دانید، تقاطع جریان‌ها بسیار خطرناک است، و در نتیجه Ghostbuster باید جفتی را انتخاب کند که جریان‌ها یکدیگر را قطع نکنند.

فرض کنید موقعیت هر Ghostbuster و هر روح، نقطه ثابتی در صفحه است و هیچ سه نقطه‌ای هم راستا نیستند.

الف. ثابت کنید خطی وجود دارد که از یک Ghostbuster و یک روح عبور می‌کند، به طوری که تعداد Ghostbuster در یک طرف از خط، برابر با تعداد ارواح در همان طرف است. توضیح دهید که چگونه می‌توان چنین خطی را در زمان  $O(n \lg n)$  پیدا کرد.

ب. یک الگوریتم  $O(n^2 \lg n)$  برای جفت Ghostbuster با ارواح ارائه دهید که هیچ جریانی یکدیگر را قطع نکنند.

#### مسئله ۴-۳: انتخاب جواب‌ها.

پروفسور Charon تعدادی چوب دارد، که با پیکربندی خاصی روی هم قرار دارند. هر چوب با نقاط پایانی اش مشخص می‌شود، و هر نقطه پایانی، یک سه‌تایی مرتب است که با مختصات  $(x, y, z)$  مشخص می‌شود. هیچ چوبی عمودی نیست. او می‌خواهد، تمام چوب‌ها را یکی‌یکی انتخاب کند، به طوری که وقتی چوبی را انتخاب می‌کند که چوب دیگری در بالای آن نباشد.

الف. رویه‌ای ارائه دهید که دو چوب  $a$  و  $b$  را گرفته و گزارش می‌دهد که آیا  $a$  در بالا، پایین یا زیر  $b$  قرار دارد.

ب. الگوریتم کارآمدی ارائه کنید که تعیین کند آیا می‌توان تمام چوب‌ها را انتخاب کرد یا خیر، و اگر بتوان انتخاب کرد، دنباله معتبری از انتخاب چوب‌ها را ارائه کنید.

#### مسئله ۵-۳: توزیع‌های پوش اسپارس.

مسئله‌ی محاسبه پوش محدب مجموعه‌ای از نقاط را در صفحه درنظر بگیرید که بر اساس توزیع تصادفی معینی ترسیم شده است. گاهی، تعداد نقاط، یا اندازه پوش محدب  $n$  نقطه‌ای که با این توزیع رسم شدند، دارای امید

$O(n^{1-\epsilon})$  است که  $\epsilon$  ثابتی با شرط  $0 < \epsilon$  است. این نوع توزیع را پوش اسپارس<sup>۱</sup> می‌گویند. توزیع‌های پوش اسپارس، شامل موارد زیر است:

- نقطه به طور یکنواخت از یک دیسک با شعاع واحد رسم شدند. پوش محدب دارای اندازه مورد انتظار  $\Theta(n^{1/3})$  است.

- نقطه به طور یکنواخت از داخل چندضلعی محدب با  $k$  ضلع، برای هر مقدار ثابت  $k$ ، رسم شده‌اند. پوش محدب دارای  $\Theta(\lg n)$  اندازه مورد انتظار است.

- محدب بر اساس توزیع نرمال دو بعدی رسم شدند. پوش محدب دارای  $\Theta(\sqrt{\lg n})$  اندازه مورد انتظار است.
- نقطه بر اساس توزیع نرمال دو بعدی رسم شدند. پوش محدب دارای  $\Theta(n_1 + n_2)$  اندازه می‌توان پوش اف. با توجه به دو چندضلعی محدب، به ترتیب با  $n_1$  و  $n_2$  رأس، نشان دهید که چگونه می‌توان پوش محدب  $n_1 + n_2$  نقطه را در زمان  $\Theta(n_1 + n_2)$  محاسبه کرد (چندضلعی‌ها ممکن است همپوشانی داشته باشند).

باشند) ب. نشان دهید که پوش محدب مجموعه‌ای از  $n$  نقطه، که بر اساس توزیع اسپارس محدب و به طور مستقل از هم ترسیم شدند، می‌تواند در زمان انتظار  $O(n)$  محاسبه شود (راهنمایی: به طور بازگشتی، پوش‌های محدب اولین  $n/2$  نقطه، و دومین  $n/2$  نقطه را پیدا کنید و سپس نتایج را ادغام نمایید).