بسم الله الرحمن الرحيم

دانشگاه یزد

دانشکده‌ی ریاضی

گروه علوم کامپیوتر

پایان‌نامه

برای دریافت درجه کارشناسی ارشد
علوم کامپیوتر

روش‌های تطبیقی برای پیدا کردن مستطیل‌های بهینه مسطح

استاد راهنما:

دکتر محمد فرشی

استاد مشاور:

دکتر مهدیه هاشمی نژاد

پژوهش‌گر:

هادی خادم پور

شهریور ۱۳۹۲

کلیه‌ی حقوق مادی و معنوی مترتب بر نتایج مطالعات، ابتکارات و نوآوری‌های ناشی از تحقیق موضوع این پایان‌نامه/رساله متعلق به دانشگاه یزد است و هرگونه استفاده از نتایج علمی و عملی از این پایان‌نامه/رساله برای تولید دانش فنی، ثبت اختراع، ثبت اثر بدیع هنری، همچنین چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس و ارائه مقاله در سمینارها و مجلات علمی از این پایان‌نامه/رساله منوط به موافقت کتبی دانشگاه یزد است.

تقدیم به

پدر و مادر عزیزم

و همه کسانی که درست اندیشیدن را به من آموختند.

سپاس‌گزاری

سپاس خدا را که مرا به راه دانش رهنمون کرد و فرصت شاگردی اساتید گرامی را بر من ارزانی داشت تا چراغی بر تاریکی جهلم باشند.

از استاد بزرگوارم، جناب آقای دکتر محمد فرشی، استاد راهنمای این پژوهش، که افق‌های جدیدی بر پنجره‌ی ذهنم گشودند، سپاس‌گزارم و رهنمودهای ایشان را ارج می‌نهم.

همراهی و راهنمایی‌های سرکار خانم دکتر مهدیه هاشمی نژاد به عنوان استاد مشاور این پژوهش درخور تقدیر و تشکر است.

از مادرم که پشتیبان همیشگی و پدرم که بزرگ‌ترین مشوق من در راه فراگیری دانش بوده‌اند کمال سپاس و امتنان را دارم و توفیق جبران هر قطره از دریای بی‌کران مهر این عزیزان را غنیمت می‌شمارم.

چکیده

فرض کنید مجموعه P شامل n نقطه در صفحه، دو محور مختصات و یک تابع امتیازدهی f که به هر زیرمجموعه از P یک مقدار حقیقی نسبت می‌دهد، داده شده است. مسأله مستطیل بهینه مسطح عبارت است از پیدا کردن یک مستطیل H (هم‌تراز با محورهای مختصات) به طوری که مقدار $f(H \cap P)$ را بیشینه کند. ما در مسأله، f را تابعی یکنوا و تجزیه‌پذیر در نظر گرفته‌ایم. یعنی تابع ترکیب دو متغیره g که در هر دو متغیر یکنوا است وجود دارد به قسمی که برای هر A زیرمجموعه P و هر بخش $\{A_1, A_2\}$ از A داشته باشیم: $f(A) = g(f(A_1), f(A_2))$. در این پایان‌نامه یک راه حل برای مسأله مستطیل بهینه مسطح بررسی می‌کنیم که در بدترین حالت در $O(n^2 \log n)$ ترکیب امتیاز و مقایسه مختصاتی قابل انجام باشد، و در رده‌های دیگر از موارد تعریف شده توسط اقدامات مختلف برای مسأله بسیار کم‌تر است. یک نتیجه جانبی از این پژوهش، ساختمان داده کاملاً پویای درخت *MCS Splay* است که از حذف و درج همراه با خاصیت انگشت پویا پشتیبانی می‌کند، که این نتایج بر اساس بهبود نتایج قبلی است.

فهرست مطالب

۱ مقدمه **۱**

۱.۱ شرح مسأله ۲

۲.۱ تاریخچه ۳

۳.۱ مفاهیم ۴

۱.۳.۱ دسته‌بندی نقاط ۵

۲.۳.۱ الگوریتم تطبیقی ۵

۳.۳.۱ تحلیل سرشکن شده ۷

۴.۳.۱ درخت‌ها ۱۱

۵.۳.۱ درخت اِی.وی.اِل ۱۵

۶.۳.۱ درخت گسترده ۲۳

۲ درخت ام.سی.اس و تعمیم آن **۳۳**

۱.۲ درخت MCS ۳۴

۲.۲ تعمیم درخت MCS برای تابع امتیازدهی f ۳۷

۳.۲ تعمیم درخت MCS برای یک دنباله پویا ۴۰

۳ روش‌های به دست آوردن مستطیل بهینه مسطح **۴۷**

۱.۳ به دست آوردن مستطیل بهینه مسطح با استفاده از درخت MCS ۴۸

۲.۳ بهره‌گیری از نوارهای تک‌رنگ ۵۰

۳.۳ بهره‌مندی از ترازهای دلخواه برای نقطه ۵۳

۴.۳ بهره‌گیری هم‌زمان از نوارها و تنظیمات نقطه ۵۶

۴ بهره‌گیری از قطرهای بلوک‌ها برای حل مسأله ۵۹

۱.۴ قطری‌سازی ۶۰

۲.۴ D- درخت ۶۴

۳.۴ به دست آوردن مستطیل بهینه مسطح ۶۹

۱.۳.۴ برخورد با مسأله‌ی آسیاب‌بادی ۷۲

۵ کاربردها و نتیجه‌گیری ۷۵

۱.۵ کاربردها ۷۶

۲.۵ نتیجه‌گیری ۷۸

۱.۲.۵ مسأله‌های باز ۷۸

واژه‌نامه فارسی به انگلیسی ۸۰

واژه‌نامه انگلیسی به فارسی ۸۴

مراجع ۸۸

لیست تصاویر

۱.۱ چرخش LL. ۱۷

۲.۱ چرخش RR. ۱۸

۳.۱ چرخش RL. ۱۹

۴.۱ چرخش LR. ۲۰

۵.۱ نوع چرخش مورد نیاز در درخت اِی.وی.اِل. ۲۰

۶.۱ حذف گره حاوی مقدار ۸ در درخت اِی.وی.اِل. ۲۲

۷.۱ مرحله zig. ۲۴

۸.۱ مرحله $zig - zag$. ۲۵

۹.۱ مرحله $zig - zig$. ۲۶

۱۰.۱ در درخت گسترده قسمت (الف)، عنصر c مورد دسترسی قرار گرفته است. . . . ۲۷

۱.۲ شکل، درخت MCS را برای دنباله‌ای با ۸ عنصر نشان می‌دهد. ۳۴

۱.۴ قطری‌سازی $\{A_1, A_2\}$ برای مجموعه نقاط $A = A_1 \cup A_2$. ۶۱

۲.۴ آسیاب‌های بادی. ۶۱

۳.۴ در هر شکل ناحیه سایه‌دار خالی از نقاط مجموعه‌ی A است. ۶۳

۴.۴ یک درخت قطری‌سازی (D-درخت) برای مجموعه نقاط $\{p_1, ..., p_{13}\}$. ۶۴

۵.۴ موقعیت نقاط در صفحه ۶۷

فصل ۱

مقدمه

همیشه در علوم و مهندسی بحث بهینه‌سازی از جایگاه ویژه‌ای برخوردار بوده است. از آغاز علم هندسه، مستطیل یکی از اشکال پرکاربرد و قابل توجه به شمار آمده است. از این رو به دست آوردن یک مستطیل بهینه با شرایطی قابل قبول می‌تواند بسیار کارامد باشد. در علم هندسه محاسباتی به دست آوردن یک مستطیل بهینه در بسیاری از مسائل با توجه به شرایط مسأله به عنوان یک شروع خوب برای حل آن مسأله به حساب می‌آید. همچنین در بسیاری از مسائل کاربردی نیز می‌تواند یک مستطیل مطلوب مورد استفاده قرار گیرد. به عنوان مثال می‌توان از کاربرد هندسه محاسباتی در نقشه‌کشی دیجیتال یاد کرد که استفاده از هندسه محاسباتی در نقشه کشی دیجیتال، کوچک‌ترین منطقه‌های مستطیلی می‌باشد.

۱.۱ شرح مسأله

برای شرح مسأله به چند تعریف نیاز داریم.

تعریف ۱.۱.۱. یک مجموعه از نقاط مجزا در صفحه را در موقعیت کلی[۱] گوییم اگر هیچ دو نقطه‌ایی بر روی خطی موازی با محور مختصات قرار نداشته باشد.

تعریف ۲.۱.۱. مجموعه A را در نظر بگیرید، $\{A_1, A_2\}$ را یک بخش از A گوییم اگر داشته باشیم:

$$A = A_1 \cup A_2$$

و

$$A_1 \cap A_2 = \emptyset.$$

تعریف ۳.۱.۱. برای مجموعه P از نقاط، تابع f را که به هر زیرمجموعه از P یک مقدار حقیقی نسبت می‌دهد، تجزیه‌پذیر[۲] [۹،۳] گوییم اگر تابع دو متغیره g موجود باشد به قسمی که برای هر A زیر مجموعه P و هر بخش $\{A_1, A_2\}$ از A داشته باشیم:

$$f(A) = g(f(A_1), f(A_2)),$$

تابع g را تابع ترکیب متناظر با تابع f گوییم.

[۱] General position
[۲] Decomposable

شرح مسأله

مجموعه P متشکل از n نقطه مسطح را که در موقعیت کلی هستند، دو محور x و y که تشکیل یک پایه برای صفحه می‌دهند و تابع تجزیه‌پذیر یکنوای f را در نظر بگیرید. یک تابع تجزیه‌پذیر را یکنوا گوییم اگر تابع ترکیب متناظر با آن نسبت هر دو آرگومان خود یکنوا باشد. بدون از دست رفتن کلیت مسأله می‌توانیم تابع f را به P به صورت $f(p) = f(\{p\})$ گسترش دهیم. با توجه به تابع تجزیه‌پذیر و یکنوای f منظور از مستطیل مسطح بهینه، مستطیلی مانند H با اضلاع موازی محور مختصات است که مقدار $f(H \cap P)$ را بهینه کند. بدون از دست رفتن کلیت مسأله، فرض می‌کنیم که می‌خواهیم f را به حداکثر برسانیم و تابع ترکیب متناظر با آن یعنی g در دو آرگومانش افزایش یابد (با افزایش هر آرگومانش، اندازه تابع افزایش یابد). با توجه به یکنوایی و تجزیه‌پذیری، تابع f، برای مجموعه تهی خوش‌تعریف می‌باشد. یک مثال ساده از چنین توابع تجزیه‌پذیر یکنوا، شمارش تعداد نقاط موجود در مستطیل است. نمونه های دیگر عبارتند از: شمارش تعداد نقاط آبی، بین تعدادی نقاط آبی و قرمز موجود؛ جمع وزن نقاط موجود؛ و در نظر گرفتن حداکثر وزن نقاط موجود است.

۲.۱ تاریخچه

یکی از حالت‌های خاص مسأله مستطیل بهینه مسطح، مسأله زیرآرایه بیشینه[۱] [۱۷] است. مسأله زیرآرایه بیشینه یک مسأله معروف در علوم رایانه است که در آن، هدف پیدا کردن زیرآرایه‌ای در یک آرایه‌ی اعداد است که بزرگ‌ترین حاصل جمع را دارند (این آرایه دست کم باید شامل یک عدد مثبت باشد). به عنوان مثال، در آرایه [۴, −۵, ۱, ۲, −۱, ۴, −۳, ۱, −۲] پاسخ مسأله عبارت است از زیرآرایه [۱, ۲, −۱, ۴] که حاصل جمعی برابر ۶ دارد. این مسأله نخستین بار در سال ۱۹۷۷ توسط الف گرناندر[۲] ریاضی‌دان سوئدی و استاد دانشگاه براون و به عنوان یک مدل ساده برای تخمین درست‌نمایی بیشینه در الگوهای موجود در تصاویر دیجیتال ارائه شد. تنها پس از چند سال، جی کادان[۳] از دانشگاه کارنگی ملون موفق شد یک الگوریتم خطی برای حل این مسأله ارائه دهد. برای مسأله بزرگ‌ترین جعبه‌ی خالی در سال ۱۹۸۴ توسط

[۱] Maximum subarray problem
[۲] Ulf Grenander
[۳] Jay Kadane

آمنون نمد[۱] ، ون لیان سو[۲] و در تاس لی[۳] [۳] الگوریتمی ارائه شد. در مسأله بزرگ‌ترین جعبه‌ی خالی n نقطه درون یک مستطیل محصور شده‌اند و هدف این است که بزرگ‌ترین مستطیل درون آن را پیدا کنیم به طوری که هیچ نقطه‌ای درون آن نباشد. پیچیدگی زمانی الگوریتم ارائه شده در بدترین حالت $O(n^2)$ و زمان مورد انتظار آن $O(n \log^2 n)$ می‌باشد. پس از آن در سال ۱۹۸۶ این مسأله توسط برناد چزل[۴] ، دریس دیول[۵] و در تاس لی [۴] در زمان قطعی $O(n \log^3 n)$ حل شد. در سال ۲۰۰۳ برای مسأله بزرگ‌ترین مستطیل خالی یانگ لیو[۶] و مک هایل ندایک[۷] [۱۳] الگوریتمی ارائه دادند، صورت مسأله بدین شرح است: فرض کنید که n نقطه در فضای $\mathbb{R}^d$ باشد و توسط X^+ و X^- افراز شده باشد. به دنبال جعبه‌ای مانند B هستیم که اندازه$B \cap X^+$ را بیشینه کند به طوری که $(B \cap X^-) = \emptyset$ باشد. الگوریتم ارائه شده دارای پیچیدگی زمانی $O(s^2 \log s)$ می‌باشد که در آن s برابر با اندازه مجموعه X^+ است. یک مسأله مهم که از تعمیم آن نیز به نوعی در حل مسأله مستطیل بهینه مسطح استفاده شده، مسأله تفکیک نقاط دو رنگ به دو مستطیل است. صورت مسأله بدین گونه است: فرض کنید S مجموعه‌ای از n نقطه با رنگ‌های آبی و قرمز باشد. هدف حذف نقاط محدودی است به طوری که باقی مانده نقاط را بتوان با دو مستطیل موازی هم، پوشاند به طوری که یک مستطیل شامل نقاط آبی و دیگری شامل نقاط قرمز، و هر مستطیل دارای تنها نقاط با یک رنگ باشند. برای این مسأله الگوریتمی با پیچیدگی زمانی $O(n^2 \log n)$ در سال ۲۰۰۹ توسط کورتس[۸] و همکارانش ارائه شده است [۷].

۳.۱ مفاهیم

در این بخش به بررسی مفاهیم مورد نیاز برای حل مسأله مستطیل بهینه مسطح می‌پردازیم.

[۱] A. Naamad
[۲] W. L. Hsu
[۳] D. T. Lee
[۴] B. Chazelle
[۵] R. L. Drysdale
[۶] Y. Liu
[۷] M. Nediak
[۸] C. Cortés

۱.۳.۱ دسته‌بندی نقاط

در بخش ۱.۱ گفته شد که تابع f برای مجموعه تهی قابل تعریف است. در این بخش با توجه به تجزیه‌پذیری و یکنوا بودن تابع f، مجموعه نقاط را به دسته‌های مختلفی تقسیم‌بندی خواهیم کرد.

تعریف ۱.۳.۱. نقطه p از مجموعه P را یک نقطه مثبت گوییم اگر $f(p) > f(\emptyset)$ باشد، در غیر این صورت نقطه p را یک نقطه منفی گوییم.

مشاهده ۲.۳.۱. توجه کنید که اگر p یک نقطه مثبت باشد آنگاه با توجه به یکنوایی تابع g که تابع ترکیب متناظر با f می‌باشد، خواهیم داشت:

$$f(A \cup \{p\}) = g(f(A), f(p)) > g(f(A), f(\emptyset)) = f(A).$$

نکته ۳.۳.۱. نقطه p یک نقطه مثبت است اگر و تنها اگر برای هر زیرمجموعه از P مانند A که شامل نقطه p نیست، داشته باشیم:

$$f(A \cup \{p\}) > f(A).$$

می‌توانیم مجموعه P را به منطقه‌های مثبت و منفی تقسیم بندی کنیم، برای این کار ابتدا نوار[۱] را تعریف می‌کنیم.

تعریف ۴.۳.۱. منطقه‌ای که توسط دو خط موازی با یکی از محورهای مختصات محصور شده باشد را یک نوار گوییم.

با استفاد از تعریف بالا مجموعه نقاط P را به نوارهای مثبت و منفی تقسیم‌بندی خواهیم کرد.

تعریف ۵.۳.۱. نوار مثبت (منفی)، نواری است که تنها حاوی نقاط مثبت (منفی) باشد، و یک نوار را، نواری تک‌رنگ[۲] گوییم که در آن تمام نقاط دارای علائم یکسانی (مثبت یا منفی) باشند.

۲.۳.۱ الگوریتم تطبیقی

الگوریتم تطبیقی[۳] مجموعه‌ای از دستورالعمل‌ها برای انجام هدف خاصی است که می‌تواند با توجه به تغییرات محیط و شرایط خود را با آن شرایط جدید وفق دهد. الگوریتم‌های تطبیقی توانایی آن را دارند

[۱] Strip
[۲] Monochromatic stripe
[۳] Adaptive algorithm

که در صورت بروز شرایطی، فعالیت‌های خود را برای رسیدن به بهترین نتیجه به طور هوشمندانه تنظیم نمایند. به عنوان مثال در یک موتور جست‌وجو، موتور می‌تواند در کل اینترنت خزش انجام دهد و نتایج را از مکان‌های متنوعی برگرداند ولی ممکن است که در ابتدا، پارامترهای مشخصی از کاربر را در نظر بگیرد. اگر آدرس شناسایی کاربر از ایران شروع شده باشد، الگوریتم تطبیقی نتایج فارسی را خواهد برگرداند یا موتور جست‌وجو کاربر را به سایت فارسی هدایت خواهد کرد. به مثال زیر توجه کنید.

مثال ۶.۳.۱. الگوریتم مرتب‌سازی درجی زیر را در نظر بگیرید.

Algorithm 1.3.1: InsertionSort(*arrayA*)

```
   Input: A آرایه .
   Output: A آرایه مرتب شده .
1  for i := 1 to length[A] − 1 do
2  |  value := A[i]؛
3  |  j := i − 1؛
4  |  while j >= 0 and A[j] > value do
5  |  |  A[j + 1] = A[j]؛
6  |  |  j := j − 1؛
7  |  end
8  |  A[j + 1] := value؛
9  end
```

در الگوریتم بالا آرایه A از صفر شروع می‌شود.

دقت کنید الگوریتم بالا کم‌ترین زمان اجرا را هنگامی دارد که کم‌ترین جابه‌جایی را در آرایه A جهت مرتب‌سازی انجام داده باشد.

زوج مرتب (i, j) را یک وارونگی در آرایه A گوییم اگر $i < j$ و $A[i] > A[j]$ باشد. اگر تعداد همه وارونگی‌ها در آرایه A، را برابر d در نظر بگیریم آنگاه زمان اجرای الگوریتم بالا برابر با $O(n + d)$ خواهد شد. توجه کنید که زمان اجرای الگوریتم بالا همچنان در بدترین حالت $O(n^2)$ است و مربوط به زمانی است که تعداد وارونگی‌ها از مرتبه $O(n^2)$ باشد، در حالی که زمان اجرای الگوریتم در بهترین حالت $O(n)$ است. پس زمان اجرای این الگوریتم به پارامتر d وابسته است و به این ترتیب یک الگوریتم تطبیقی می‌باشد.

۳.۳.۱ تحلیل سرشکن شده

در تحلیل سرشکن شده[۱]، برای محاسبه زمان مورد نیاز جهت انجام یک سلسله از اعمال ساختمان داده‌ای روی تمام اعمالی که انجام شده‌اند، میانگین گرفته می‌شود. اگر روی یک توالی از اعمال میانگین گرفته شود، هر چند ممکن است یک عمل داخل آن توالی پر هزینه باشد، اما با استفاده از تحلیل سرشکن شده می‌توان نشان داد که هزینه میانگین عمل کم است. تحلیل سرشکن شده با تحلیل حالت میانگین که در آن احتمال در نظر گرفته نمی‌شود تفاوت دارد. تحلیل سرشکن شده میانگین کارایی هر عمل در بدترین حالت را ضمانت می‌کند.

در این بخش، سه روش تحلیل جمعی، حسابداری و پتانسیل را که در تحلیل سرشکن شده به طور رایج استفاده می‌شوند، توضیح می‌دهیم [۶].

تحلیل جمعی

در تحلیل جمعی[۲] نشان می‌دهیم که به ازای تمام nها، یک توالی از n عمل در مجموع، زمان $T(n)$ را در بدترین حالت صرف می‌کند. بنابراین در بدترین حالت، هزینه میانگین، یا هزینه سرشکن[۳] شده برای هر عمل برابر $\frac{T(n)}{n}$ است. توجه کنید که این هزینه سرشکن شده برای هر عمل به کار می‌رود. حتی وقتی که چندین نوع عمل در توالی وجود داشته باشد.

مثال ۷.۳.۱. مجموعه‌ای از n عمل $push$، pop و $Multipop(k)$ روی پشته s که در ابتدا خالی است انجام داده ایم (عمل $Multipop(k)$، k عنصر از پشته pop می‌کند و اگر کمتر از k عنصر در پشته باشد، همه عناصر را pop می‌کند). درست است که هزینه $Multipop(k)$ ممکن است $O(k)$ باشد ولی n عمل از عملیات فوق روی هم هزینه $O(n)$ دارند پس هزینه هر عمل به صورت سرشکن شده برابر $O(۱)$ است.

روش حسابداری

در روش حسابداری[۴] از تحلیل سرشکن شده، به عمل‌های متفاوت هزینه‌های متفاوتی اختصاص می‌دهیم، به طوری که به تعدادی از اعمال هزینه بیش‌تر و به تعدادی هزینه کم‌تری نسبت به هزینه واقعی آن‌ها

[۱] Amortized analysis
[۲] Aggregate analysis
[۳] Amortized cost
[۴] Accounting method

اختصاص داده می‌شود. مقدار هزینه‌ای که به یک عمل می‌دهیم، هزینه سرشکن شده آن عمل نامیده می‌شود. هنگامی که هزینه سرشکن شده یک عمل از هزینه واقعی آن عمل تجاوز کند، اختلاف هزینه به عنوان موجودی[1] به اشیائی مشخص در ساختمان داده اختصاص می‌یابد. موجودی بعداً می‌تواند جهت کمک به پرداخت برای اعمالی که هزینه سرشکن شده آن‌ها کم‌تر از هزینه واقعی‌شان است، مورد استفاده قرار گیرد. بنابراین می‌توان هزینه سرشکن شده یک عمل را مشاهده کرد که بین هزینه واقعی آن و موجودی که پرداخت شده یا استفاده می‌شود، تقسیم شده است. این روش با تحلیل جمعی که در آن تمام اعمال هزینه‌های سرشکن شده یکسانی دارند، بسیار متفاوت است.

هزینه‌های سرشکن شده اعمال باید به دقت انتخاب شوند. اگر بخواهیم تحلیل را با هزینه‌های سرشکن شده انجام دهیم تا نشان دهیم که بدترین حالت هزینه میانگین هر عمل کم است، کل هزینه سرشکن شده یک توالی از اعمال باید یک حد بالا برای هزینه کل واقعی باشد. علاوه بر این همانند تحلیل جمعی، این رابطه باید برای تمام توالی‌های اعمال برقرار باشد. اگر هزینه واقعی iامین عمل را با c_i و هزینه سرشکن شده iامین عمل را با $\widehat{c_i}$ نشان دهیم، برای تمام توالی‌هایی از n عمل نیاز داریم

$$\sum_{i=1}^{n} \widehat{c_i} \geq \sum_{i=1}^{n} c_i \qquad (1-1)$$

موجودی کل که در ساختمان داده ذخیره شده است برابر تفاوت بین کل هزینه سرشکن شده و کل هزینه واقعی است، یا $\sum_{i=1}^{n} \widehat{c_i} - \sum_{i=1}^{n} c_i$. بنا به نامساوی ۱-۱، کل موجودی مربوط به ساختمان داده باید در همه زمان‌ها نامنفی باشد. اگر کل موجودی می‌توانست منفی باشد (در نتیجه پرداخت ارزش کم به عمل‌های قبلی، با این تعهد که دوباره پس از آن پرداخت صورت گیرد) آنگاه کل هزینه‌های سرشکن شده ایجاد شده در آن هنگام کم‌تر از کل هزینه‌های واقعی ایجاد شده می‌بود و برای توالی عمل‌ها تا آن زمان کل هزینه سرشکن شده یک حد بالا برای کل هزینه واقعی نمی‌بود. بنابرین باید مراقب باشیم تا کل موجودی در ساختمان داده هرگز منفی نشود.

مثال ۸.۳.۱. عملیات پشته را در نظر بگیرید . می‌دانیم هزینه واقعی عملیات $push$، pop و $Multipop(k)$ به ترتیب عبارت است از : $O(1)$، $O(1)$ و $O(\min(k, s))$ که s تعداد عناصر موجود در پشته هنگام فراخوانی $Multipop$ است. ولی ما هزینه سرشکن شده زیر را به عملیات‌ها نسبت می‌دهیم:

[1] Credit

هزینه سرشکن شده *push* برابر ۲؛

هزینه سرشکن شده *pop* برابر ۰؛

هزینه سرشکن شده *Multipop* برابر ۰؛

با این مقادیر، می‌توان هر دنباله از عملیات را پشتیبانی کرد. هر *push* که انجام می‌شود، هزینه واقعی آن ۱ است، پس از شارژ ۲ مقداری آن، ۱ مقدار برای هزینه‌اش صرف می‌شود و ۱ مقدار در عنصر *push* شده به عنوان اعتبار ذخیره می‌شود. پس تمام عناصری که در پشته هستند، اعتبار ۱ مقداری دارند. حال اگر *pop* انجام شود، چون هزینه واقعی *pop*، ۱ است، این ۱ را از اعتبار عنصری که *pop* می‌شود، دریافت می‌کند. پس از آنجایی که هزینه سرشکن شده *push*، *pop* و *Multipop* ثابت است پس هزینه n عمل، $O(n)$ است.

روش پتانسیل

به جای نمایش کار از پیش پرداخت شده به صورت موجودی که با اشیائی مشخص در ساختمان داده ذخیره می‌شود، روش پتانسیل [۱] از تحلیل سرشکن شده، کار از پیش پرداخت شده را به صورت انرژی پتانسیل و یا تنها «پتانسیل» نمایش می‌دهد که می‌تواند جهت پرداخت برای اعمال آینده آزاد شود. پتانسیل به جای آن که به اشیائی خاص در داخل ساختمان داده اختصاص داده شود، به کل ساختمان داده اختصاص داده می‌شود.

روش پتانسیل به صورت زیر کار می‌کند. با یک ساختمان داده اولیه D_0 شروع می‌کنیم که روی آن n عمل انجام می‌شود. برای هر $i = 1, 2, ..., n$، c_i را هزینه واقعی iامین عمل قرار می‌دهیم و D_i را ساختمان داده‌ای که پس از بارگیری iامین عمل روی ساختمان داده D_{i-1} حاصل می‌شود، قرار می‌دهیم. تابع پتانسیل [۲] Φ هر ساختمان داده D_i را به یک عدد حقیقی $\Phi(D_i)$ نگاشت می‌کند. این عدد پتانسیل[۳] مربوط به ساختمان داده D_i است. هزینه سرشکن شده $\widehat{c_i}$ مربوط به iامین عمل با توجه به تابع پتانسیل Φ

[۱] Potential method
[۲] Potential function
[۳] Potential

به صورت زیر تعریف می‌شود:

$$\widehat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}) \qquad (۲-۱)$$

بنابراین هزینه سرشکن شده هر عمل برابر هزینه واقعی آن بعلاوه افزایش پتانسیل حاصل از عمل می‌باشد. بنا به معادله ۱-۲، هزینه سرشکن شده کل n عمل برابر است با

$$\sum_{i=1}^{n} \widehat{c_i} = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \qquad (۳-۱)$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0). \qquad (۴-۱)$$

اگر بتوانیم یک تابع پتانسیل تعریف کنیم به طوری که $\Phi(D_n) \geq \Phi(D_0)$، آنگاه هزینه سرشکن شده کل $\sum_{i=1}^{n} \widehat{c_i}$، یک حد بالا برای هزینه واقعی کل $\sum_{i=1}^{n} c_i$ می‌باشد.

در عمل همواره نمی‌دانیم چه تعداد عمل ممکن است انجام شود. بنابراین اگر برای تمام iها تأکید کنیم که $\Phi(D_n) \geq \Phi(D_0)$, آنگاه همانند روش حسابداری تضمین می‌کنیم که از پیش هزینه را پرداخت کرده‌ایم. اغلب مناسب است که $\Phi(D_0)$ را صفر تعریف کنیم و آنگاه نشان دهیم برای تمام iها، $\Phi(D_i) \geq \Phi(D_0)$.

به طور شهودی اگر اختلاف پتانسیل $\Phi(D_i) - \Phi(D_{i-1})$ از iامین عمل مثبت باشد آنگاه هزینه سرشکن شده $\widehat{c_i}$ یک پرداخت اضافه به iامین عمل را نشان می‌دهد و پتانسیل ساختمان داده افزایش می‌یابد. اگر اختلاف پتانسیل منفی باشد، آنگاه هزینه سرشکن شده، یک پرداخت کم به iامین عمل را نشان می‌دهد و هزینه واقعی عمل با کاهش پتانسیل پرداخت می‌شود.

هزینه‌های سرشکن شده تعریف شده به وسیله معادله‌های ۱-۲ و ۱-۳ به انتخاب پتانسیل Φ بستگی دارند. توابع پتانسل متفاوت ممکن است به هزینه‌های سرشکن شده متفاوت که همچنان حدود بالا برای هزینه‌های واقعی‌اند منجر گردد. اغلب در انتخاب یک تابع پتانسیل مبادلاتی می‌توانند انجام گیرند؛ بهترین تابع پتانسیل جهت استفاده بستگی به حدود زمانی مطلوب دارد.

مثال ۹.۳.۱. فرض کنید در عملیات پشته $push$، pop و $Multipop(k)$، تابع پتانسیل Φ را برابر عناصر موجود در پشته فرض کنیم. در حالت پشته خالی D_0، که حالت شروع است، داریم $\Phi(D_0) = 0$. از آنجایی که تعداد عناصر موجود در پشته عددی نامنفی است پس همیشه $\Phi(D_i) \geq 0 = \Phi(D_0)$ است.

پس هزینه کل سرشکن شده برای n عمل با توجه به تابع Φ، یک کران بالا برای هزینه واقعی است. حال هزینه سرشکن شده را برای $push$، pop و $Multipop(k)$، به دست می‌آوریم.

اگر عمل iام روی پشته $push$ باشد و در حال حاضر تعداد s عنصر در پشته باشد آنگاه:

$$\Phi(D_i) - \Phi(D_{i-1}) = (s+1) - s = 1$$

بنابراین هزینه سرشکن شده $push$ برابر است با:

$$\widehat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 1 = 2$$

به همین ترتیب نشان داده می‌شود که هزینه سرشکن شده pop و $Multipop(k)$ برابر صفر است. پس هزینه سرشکن شده هر عمل $O(1)$ است. بنابراین از آنجایی که هزینه سرشکن شده کران بالای هزینه واقعی است پس هزینه واقعی n عمل برابر $O(n)$ است.

۴.۳.۱ درخت‌ها

در علوم کامپیوتر، درخت، ساختمان داده‌ای پر استفاده است که شبیه به یک ساختار درختی با مجموعه‌ای از گره‌های متصل به هم است. در نظریه گراف، درخت گرافی همبند، و بدون دور است. تاریخچه درخت به تاریخچه گراف برمی‌گردد، برخلاف شاخه‌های دیگر ریاضیات، سیر نظریه گراف آغاز معینی در زمان و مکان دارد و آن مسأله هفت پل کونیگسبرگ[۱] است که در سال ۱۷۳۶ توسط لئونارد اویلر[۲] حل شد. در سال ۱۷۵۲ قضیه اویلر برای گراف‌های مسطح ارائه می‌شود. اما پس از آن به مدت تقریباً یک قرن فعالیت اندکی در این زمینه صورت گرفت. در سال ۱۸۴۷، گوستاو کیرشهف[۳] نوع خاصی از گراف‌ها به نام درخت را مورد بررسی قرار داد. کیرشهف این مفهوم را هنگام تعمیم قوانین اُهم برای جریان الکتریکی در کاربردهایی که حاوی شبکه‌های الکتریکی بودند به‌کار گرفت.

تعریف ۱۰.۳.۱. درخت[۴] مجموعه متناهی از یک یا چند گره به صورت زیر می‌باشد:

۱. دارای گره خاصی به نام ریشه است.

[۱] Seven Bridges of Königsberg
[۲] Leonhard Euler
[۳] Gustav Kirchhoff
[۴] Tree

۲. بقیه گره‌ها به $n \geq ۰$ مجموعه مجزا $T_۱, ..., T_n$ تقسیم شده که هر یک از این مجموعه‌ها خود یک درخت هستند و $T_۱, ..., T_n$ زیردرختان ریشه نامیده می‌شود.

توجه کنید این شرط که $T_۱, ..., T_n$ از هم مجزا می‌باشند از هر گونه اتصال زیردرخت‌ها به هم جلوگیری می‌کند.

درجه یک گره: تعداد زیردرخت‌های یک گره درجه[۱] آن نام دارد.

برگ: گره‌هایی که درجه صفر دارند، برگ[۲] یا گره پایانی نامیده می‌شود.

درجه یک درخت: حداکثر درجه گره‌های یک درخت، درجه آن درخت نامیده می‌شود.

گره فرزند و پدر: زیردرختان با گره ریشه‌ای مانند x، فرزندهای x نامیده می‌شود و گره x پدر بچه‌هایش است.

مسیر: هرگاه برای هر $i = ۲, ..., k$، v_i فرزند $v_{i-۱}$ باشد. دنباله $v_۱, ..., v_k$ از گره‌ها، یک مسیر از $v_۱$ به v_k است.

اجداد یک گره: اجداد یک گره، گره‌هایی هستند که در مسیر طی شده از ریشه تا آن گره وجود دارند.

سطح یک گره: سطح یک گره به این صورت تعریف می‌شود که ریشه در سطح یک قرار دارد و اگر یک گره در سطح l باشد آنگاه فرزندان آن در در سطح $l + ۱$ هستند.

ارتفاع یا عمق یک درخت: ارتفاع یا عمق یک درخت، بزرگ‌ترین سطح گره‌های آن درخت است.

درخت مرتب: درخت مرتب، درختی است که ترتیب زیردرخت‌ها در آن مشخص شده است.

درخت kتایی: درختی که تعداد فرزندان هر گره در آن حداکثر k باشد.

درخت دودویی

تعریف ۱۱.۳.۱. یک درخت دودویی[۳] یا تهی است یا مجموعه‌ای محدود از گره‌ها شامل یک ریشه و دو زیردرخت دودویی است. این زیردرخت‌ها، زیردرخت‌های چپ و راست نامیده می‌شوند.

توجه کنید که در درخت دودویی ترتیب زیردرختان مهم می‌باشد.

درخت مورب: درخت مورب چپ، درختی دودویی است که هر گره به جز ریشه، فرزند چپ پدر خود

[۱] Degree
[۲] Leaf
[۳] Binary tree

است. درخت مورب راست به صورت مشابه تعریف می‌شود. درخت مورب، درختی است که درخت مورب چپ یا درخت مورب راست باشد.

پیمایش درخت

منظور از پیمایش درخت، ملاقات کردن همه گره‌هاست به طوری که گره‌ای را بیش از یک بار ملاقات نکنیم. به هنگام پیمایش یک درخت دودویی، می‌خواهیم هر گره و زیردرختانش را به طرز مشابهی پیمایش کنیم.

پیمایش پیش‌ترتیب: در روش پیمایش پیش‌ترتیب[۱] یک درخت دودویی غیرخالی به صورت زیر پیمایش می‌شود:

۱- ریشه را ملاقات کنید.

۲- زیردرخت چپ را به روش پیش‌ترتیب پیمایش کنید.

۳- زیردرخت راست را به روش پیش‌ترتیب پیمایش کنید.

پیمایش پس‌ترتیب: در روش پیمایش پس‌ترتیب[۲] یک درخت دودویی غیرخالی به صورت زیر پیمایش می‌شود:

۱- زیردرخت چپ را به روش پس‌ترتیب پیمایش کنید.

۲- زیردرخت راست را به روش پس‌ترتیب پیمایش کنید.

۳- ریشه را ملاقات کنید.

پیمایش میان‌ترتیب: در روش پیمایش میان‌ترتیب[۳] یک درخت دودویی غیرخالی به صورت زیر پیمایش می‌شود:

۱- زیردرخت چپ را به روش میان‌ترتیب پیمایش کنید.

---

[۱] Preorder
[۲] Postorder
[۳] Inorder

۲- ریشه را ملاقات کنید.

۳- زیردرخت راست را به روش میان‌ترتیب پیمایش کنید.

درخت جست‌وجوی دودویی

درخت جست‌وجوی دودویی[1]، نوعی درخت دودویی است که ممکن است تهی باشد، اگر تهی نباشد، دارای خصوصیات زیر است:

- از تعدادی گره تشکیل شده که هر گره دارای یک کلید است. کلیدها منحصر به فرد هستند و در درخت کلید تکراری وجود ندارد.
- تمام کلیدهایی که در زیردرخت سمت چپ واقع شده‌اند، کوچکتر از کلید گره ریشه هستند.
- تمام کلیدهایی که در زیردرخت سمت راست واقع شده‌اند، بزرگ‌تر از کلید گره ریشه هستند.
- زیردرخت سمت راست و زیردرخت سمت چپ خود درختان جست‌وجوی دودویی هستند.

درخت جست‌وجوی دودویی این ویژگی را تضمین می‌کند که پیمایش میان‌ترتیب یک درخت جست‌وجوی دودویی، کلیدها را به ترتیب صعودی نمایش می‌دهد. توجه کنید که زیردرخت‌های درخت جست‌وجوی دودویی نیز درخت جست‌وجوی دودویی است.

جست‌وجو یک عنصر در درخت جست‌وجوی دودویی

فرض کنید می‌خواهیم عنصری با کلید x را در درخت جست‌وجوی دودویی بیابیم. ابتدا از ریشه درخت شروع می‌کنیم، اگر ریشه درخت تهی باشد، درخت فاقد هر عنصری بوده و جست‌وجو ناموفق خواهد بود. در غیر این صورت، x را با مقدار کلید ریشه مقایسه می‌کنیم. اگر x کوچک‌تر از مقدار کلید ریشه باشد، زیردرخت چپ را به صورت بازگشتی جست‌وجو می‌کنیم. اگر x بزرگ‌تر از مقدار کلید ریشه باشد آنگاه زیردرخت راست را به صورت بازگشتی جست‌وجو می‌کنیم.

[1] Binary search tree

نکته ۱۲.۳.۱. اگر ارتفاع یا عمق یک درخت جست‌وجوی دودویی برابر با h باشد، آنگاه می‌توان عمل جست‌وجو برای یک عنصر را در $O(h)$ انجام داد.

اضافه کردن یک عنصر به درخت جست‌وجوی دودویی

برای درج یک عنصر جدید x، باید ابتدا مشخص نمود که آیا این عنصر با عناصر موجود متفاوت می‌باشد یا خیر. برای این کار باید درخت را برای عنصر x جست‌وجو کرد. اگر جست‌وجو ناموفق باشد آنگاه ما عنصر را در محلی که جست‌وجو خاتمه پیدا کرده است درج می‌کنیم.

نکته ۱۳.۳.۱. اگر ارتفاع یا عمق یک درخت جست‌وجوی دودویی برابر با h باشد، آنگاه می‌توان عمل درج یک عنصر در این درخت را در $O(h)$ انجام داد.

حذف یک عنصر از درخت جست‌وجوی دودویی

جهت حذف از درخت جست‌وجوی دودویی بسته به نوع گره‌ای (برگ، گره دارای یک فرزند، گره دارای دو فرزند) که می‌خواهیم حذف کنیم، سه حالت داریم:

۱) **حذف گره برگ:** فقط گره را حذف می‌کنیم.

۲) **حذف گره میانی که یک فرزند دارد:** گره را حذف کرده و تنها فرزند آن را به جای آن قرار می‌دهیم.

۳) **حذف گره میانی که دو فرزند دارد:** گره را حذف کرده و بزرگ‌ترین فرزند در زیردرخت سمت چپ و یا کوچک‌ترین فرزند زیردرخت سمت راست آن را به جای آن قرار می‌دهیم.

نکته ۱۴.۳.۱. عمل حذف یک عنصر از درخت جست‌وجوی دودویی می‌تواند در $O(h)$ انجام گیرد که در آن h ارتفاع یا عمق درخت جست‌وجوی دودویی است.

۵.۳.۱ درخت اِی.وی.اِل

عنوان درخت اِی.وی.اِل[۱] از اول نام‌های دو مخترع آن به نام‌های جرجی ماکسیموویچ ادلسون-ولسکی[۲] و اوگنی میخائیلوویچ لندیس[۳]، که مقاله خود را در سال ۱۹۶۲ با موضوع «یک الگوریتم برای سازماندهی اطلاعات» منتشر کردند، گرفته شده است.

[۱] AVL Tree
[۲] Georgy Maximovich Adelson-Velsky
[۳] Evgenii Mikhailovich Landis

تعریف ۱.۳.۱۵. یک درخت دودویی تهی از نظر ارتفاع متوازن است. اگر T یک درخت دودویی غیرتهی با زیردرختان چپ و راست T_L و T_R باشد، آنگاه T یک درخت متوازن از نظر ارتفاع است اگر و فقط اگر:

۱. داشته باشیم $|h_L - h_R| \leq ۱$، که در آن h_L و h_R به ترتیب ارتفاع T_L و T_R هستند.

۲. T_L و T_R از نظر ارتفاع متوازن باشند.

به بیان ساده‌تر درخت T از نظر ارتفاع متوازن است اگر و فقط اگر تفاضل ارتفاع زیردرخت چپ و راست آن کوچکتر یا مساوی ۱ باشد و همچنین دو زیردرخت چپ و راست آن متوازن باشند.

تعریف ۱.۳.۱۶. درخت دودویی T را یک درخت خودمتوازن[۱] گوییم اگر به طور خودکار بعد از هر عمل حذف یا اضافه در درخت، درخت از نظر ارتفاع متوازن باقی بماند.

تعریف ۱.۳.۱۷. درخت اِی.وی.اِل یک درخت جست‌وجوی دودویی است که خودمتوازن باشد.

تعریف ۱.۳.۱۸. فاکتور توازن هر گره به صورت تفاوت بین ارتفاع زیردرخت چپ گره از ارتفاع زیردرخت راست آن گره تعریف می‌شود و هر گره با فاکتور توازن ۱، ۰ و ۱- به صورت گره متوازن‌شده در نظر گرفته می‌شود.

یک گره با هر فاکتور توازن دیگری غیر از ۱، ۰ و ۱-، یک گره نامتوازن تلقی می‌شود و در نتیجه نیاز به متوازن کردن مجدد درخت می‌باشد. فاکتور توازن یا به صورت مستقیم در گره ذخیره شده و یا از طریق ارتفاع زیردرختان آن محاسبه می‌شود.

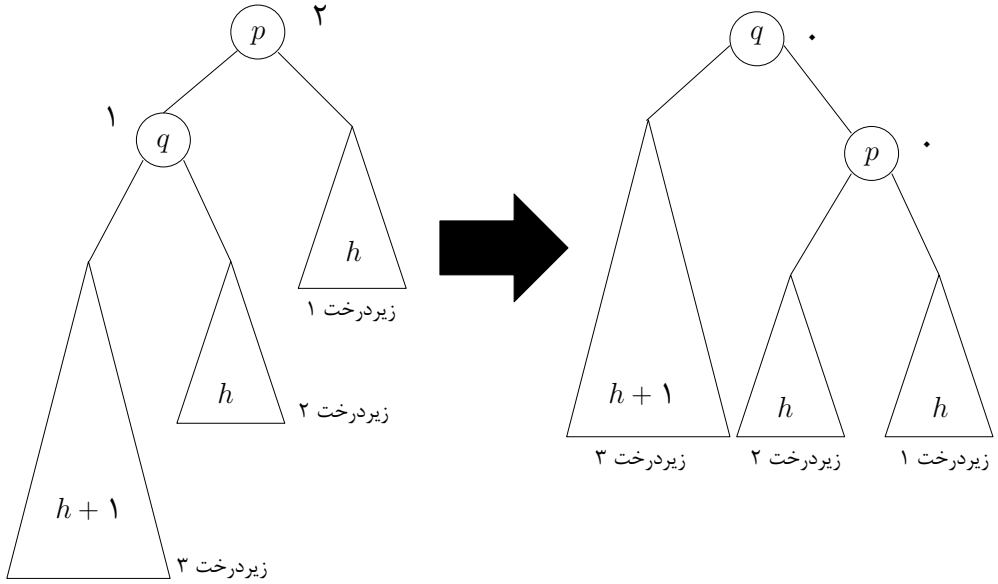
نکته ۱.۳.۱۹. فاکتور توازن هر گره در درخت اِی.وی.اِل برابر با ۱، ۰ یا ۱- است.

چرخش‌های درخت اِی.وی.اِل

در درخت اِی.وی.اِل مواقعی که توازن درخت به هم می‌خورد (مثلاً عنصری از درخت حذف یا در درخت درج شود) و درخت نامتوازن می‌شود نیاز به چرخش در درخت داریم تا درخت متوازن شود.

در درخت اِی.وی.اِل چهار نوع چرخش داریم که در زیر به آن‌ها خواهیم پرداخت:

[۱] Self-balancing

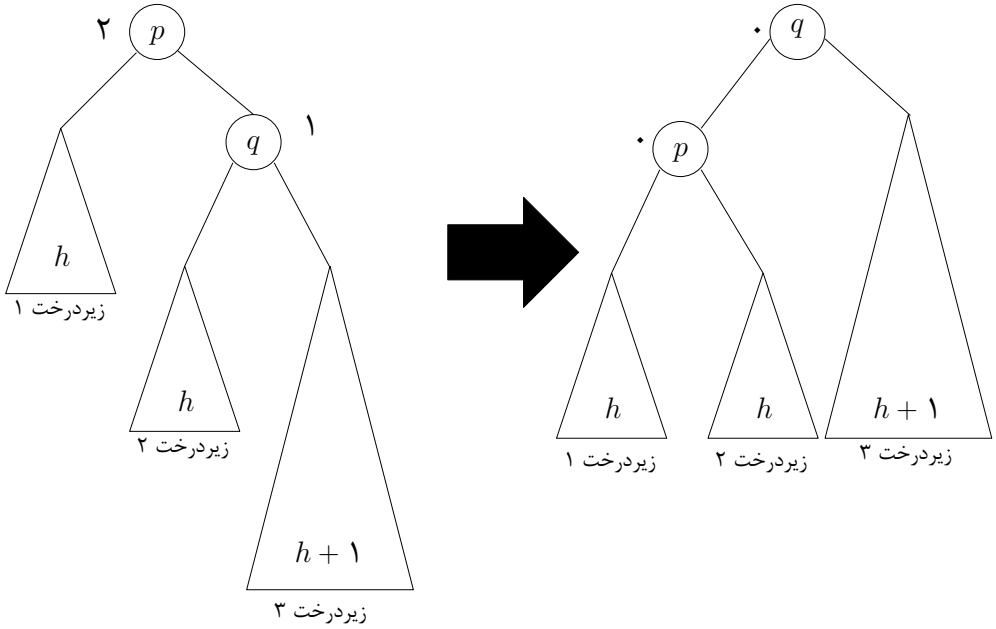


شکل ۱.۱: چرخش *LL*.

۱. **چرخش *LL*:** به عنوان مثال حالتی را در نظر بگیرید که با اضافه کردن گره‌ای به زیردرخت چپ، فرزند چپ یک گره، آن گره نامتوازن شود در این حالت از این چرخش استفاده می‌کنیم. همان گونه که در شکل ۱.۱ دیده می‌شود ارتفاع زیردرخت چپ گره p از ارتفاع زیردرخت راست آن ۲ واحد بیشتر است، پس این درخت در گره p توازن خود را از دست داده است و با چرخش *LL* مطابق شکل ۱.۱ دوباره متوازن می‌گردد (اعداد روی گره‌های p و q نشان دهنده فاکتور توازن آن‌ها می‌باشد).

در واقع برای چرخش *LL* با توجه به شکل ۱.۱ به صورت زیر عمل می‌کنیم:
زیردرخت چپ گره p را برابر زیردرخت راست گره q قرار داده و سپس زیردرخت راست گره q را برابر زیردرخت با ریشه p قرار می‌دهیم.

۲. **چرخش *RR*:** به عنوان مثال حالتی را در نظر بگیرید که با اضافه کردن گره‌ای به زیردرخت راست، فرزند راست یک گره، آن گره نامتوازن شود در این حالت از این چرخش استفاده می‌کنیم. با توجه به شکل ۲.۱ ارتفاع زیردرخت راست گره p از ارتفاع زیردرخت چپ آن ۲ واحد بیشتر است، پس این درخت در گره p توازن خود را از دست داده و باید حول گره p چرخش *RR* انجام دهیم.



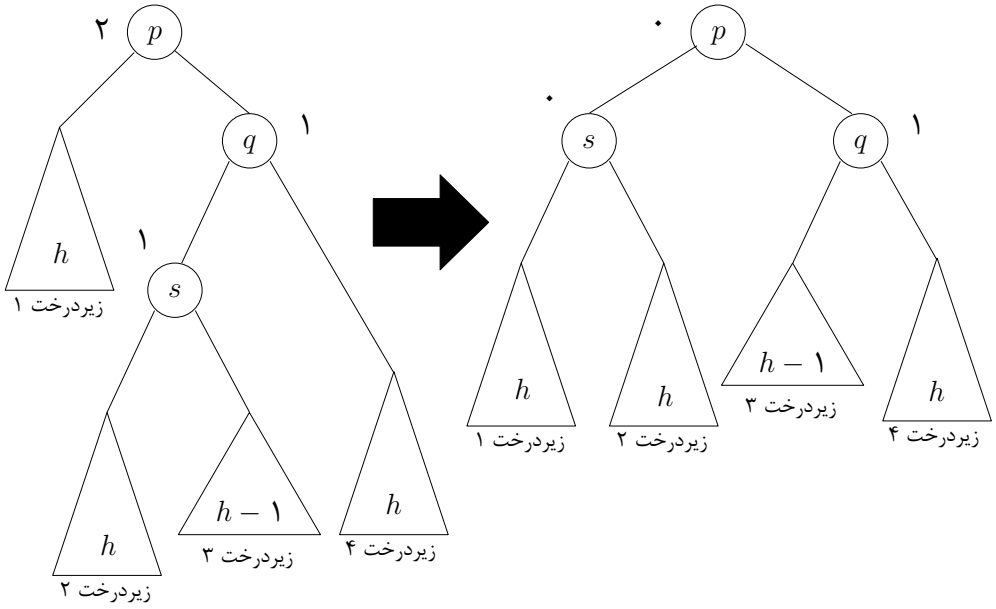
شکل ۲.۱: چرخش *RR*.

در واقع برای چرخش *RR* با توجه به شکل ۲.۱ به صورت زیر عمل می‌کنیم:
زیردرخت راست گره p را برابر زیردرخت چپ گره q قرار داده و سپس زیردرخت چپ گره q را برابر زیردرخت با ریشه p قرار می‌دهیم.

۳. **چرخش *RL*:** همانند شکل ۳.۱ حالتی را در نظر بگیرید که با اضافه کردن گره‌ای به زیردرخت چپ، فرزند چپ، فرزند راست یک گره، آن گره نامتوازن شود در این حالت از این چرخش استفاده می‌کنیم. ارتفاع زیردرخت راست گره p در شکل ۳.۱ از ارتفاع زیردرخت چپ آن ۲ واحد بیشتر است، این حالت با چرخش *RR* تفاوت دارد. در این حالت درخت با چرخش *RL* حول گره p دوباره متوازن می‌گردد.

در واقع برای چرخش *RL* با توجه به شکل ۳.۱ به صورت زیر عمل می‌کنیم:
زیردرخت راست گره p را برابر زیردرخت چپ گره s، زیردرخت چپ گره q را برابر زیردرخت راست گره s، زیردرخت چپ گره s را برابر زیردرخت با ریشه گره p و سپس زیردرخت راست گره s را برابر زیردرخت با ریشه گره q قرار می‌دهیم.

۴. **چرخش *LR*:** همانند شکل ۴.۱ حالتی را در نظر بگیرید که با اضافه کردن گره‌ای به زیردرخت



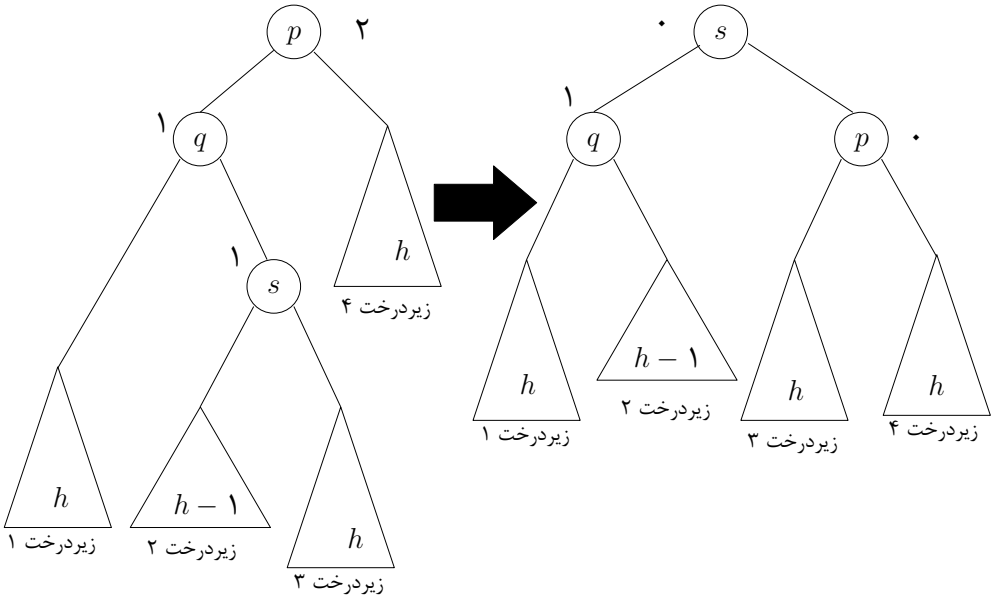
شکل ۳.۱: چرخش *RL*.

راست، فرزند چپ یک گره، آن گره نامتوازن شود در این حالت از این چرخش استفاده می‌کنیم. در شکل ۴.۱ ارتفاع زیردرخت چپ گره P از ارتفاع زیردرخت راست آن ۲ واحد بیشتر است، این حالت با چرخش *LL* تفاوت دارد. در این حالت درخت با چرخش *LR* دوباره متوازن می‌گردد.

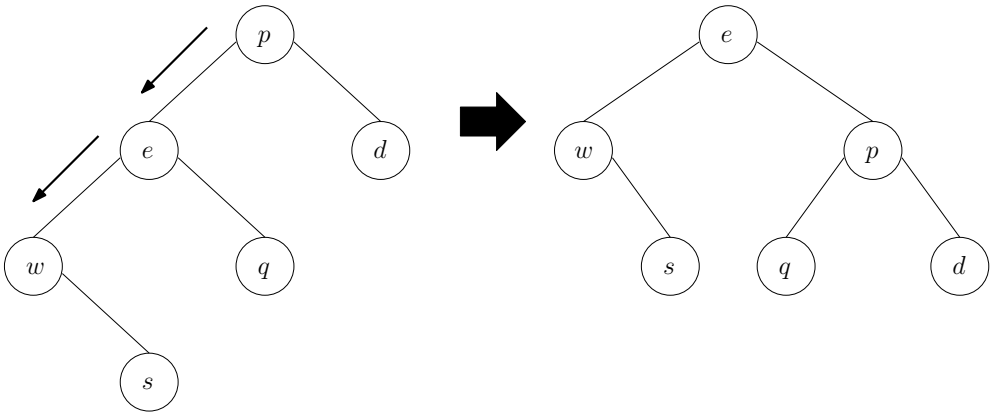
در واقع برای چرخش *LR* با توجه به شکل ۴.۱ به صورت زیر عمل می‌کنیم:
زیردرخت چپ گره p را برابر زیردرخت راست گره s، زیردرخت راست گره q را برابر زیردرخت چپ گره s، زیردرخت راست گره s را برابر زیردرخت با ریشه گره p و سپس زیردرخت چپ گره s را برابر زیردرخت با ریشه گره q قرار می‌دهیم.

چگونگی تشخیص نوع چرخش مورد نیاز

برای این که تشخیص دهیم چه نوع چرخشی باید اعمال کنیم تا درخت نامتوازنی که در اختیار داریم را به درخت متوازن تبدیل کنیم، باید از گره‌ای که توازن آن به هم خورده به سمت عامل بر هم زدن توازن (مثلاً گره جدیدی درج کرده‌ایم و توازن به هم خورده است) حرکت کنیم. دو حرکت اول نشان دهنده نوع چرخش است. برای فهم بهتر به مثال زیر توجه کنید.



شکل ۴.۱: چرخش *LR*.



شکل ۵.۱: نوع چرخش مورد نیاز در درخت اِی.وی.اِل.

مثال ۲۰.۳.۱. شکل ۵.۱ را در نظر بگیرید، با درج گره s در درخت توازن گره p به هم می‌خورد. همان طور که در شکل می‌بینید، با توجه به پیکان‌ها حرکت می‌کنیم (از سمت گره‌ای که توازن ندارد به سمت گره‌ای که عامل نامتوازنی است). برای رفتن از گره p به سمت گره s دو حرکت اول به سمت چپ است، بنابرین چرخش از نوع LL می‌باشد.

عملیات در درخت اِی.وی.اِل

به طور کلی عملیات اساسی در درخت‌های متوازن شامل همان عملیاتی است که در درخت‌های جست‌وجوی دودویی انجام می‌شود، اما اصلاحات و تغییرات به صورت فراخوانی یک یا چند بار چرخش درخت خواهد بود.

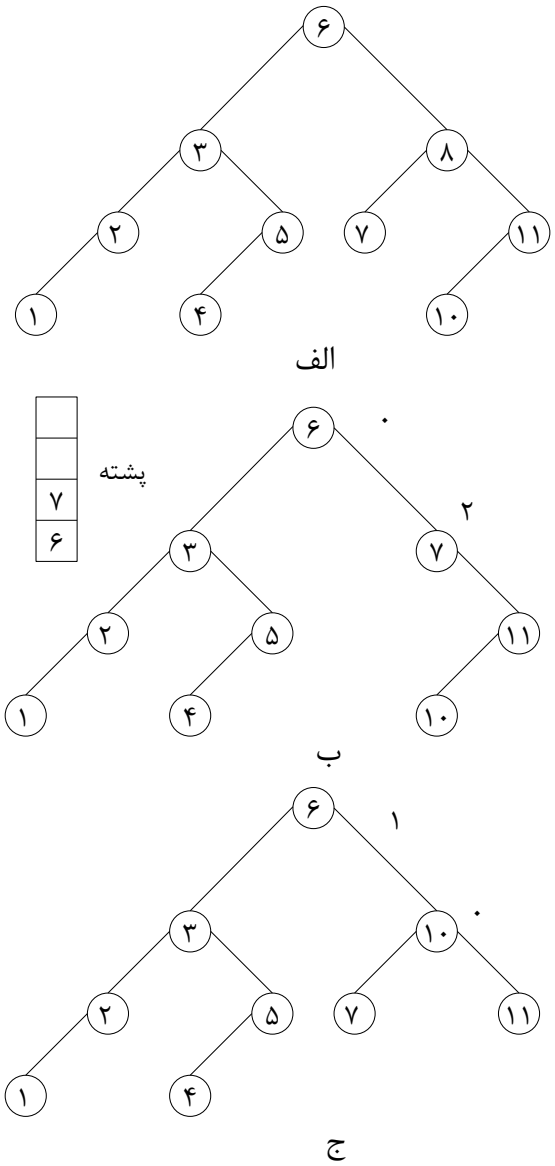
درج و حذف از درخت اِی.وی.اِل

چون درخت اِی.وی.اِل همواره به صورت متوازن است پس باید درج و حذف از آن به گونه‌ای باشد که توازن درخت را بر هم نزند و درخت متوازن باقی بماند.

برای درج در درخت اِی.وی.اِل ابتدا باید مکان گره‌ای را که می‌خواهیم درج کنیم را همانند درج در درخت جست‌وجوی دودویی پیدا کنیم، سپس آن را در درخت درج و مسیر گره پدر گره جدید را تا ریشه بررسی کرده و هر گره‌ای که توازنش به هم خورده بود، روی آن چرخش لازم را انجام می‌دهیم. پس از چرخش ادامه مسیر را برای بررسی توازن گره‌ها طی می‌کنیم. هزینه لازم برای درج از مرتبه $O(\log n)$ می‌باشد. زیرا درخت متوازن است و هر چرخش در زمان ثابت انجام می‌شود.

برای حذف از درخت اِی.وی.اِل ابتدا باید گره‌ای را که می‌خواهیم حذف کنیم را پیدا کنیم. سپس تمام گره‌هایی که در مسیر ریشه تا گره‌ای که باید حذف شود را به ترتیب از ریشه تا گره مورد نظر (از بالا به پایین) را در پشته‌ای ذخیره می‌کنیم و اگر گره‌ای که می‌خواهیم حذف کنیم دارای دو فرزند باشد آنگاه گره‌های در مسیر گره حذف شونده تا گره جایگزین را نیز در پشته ذخیره می‌کنیم. پس از آن همانند حذف از درخت جست‌وجوی دودویی عمل می‌کنیم و گره مورد نظر را حذف می‌کنیم، در مرحله آخر گره‌ها را از پشته pop می‌کنیم و فاکتور توازن آنها را بررسی می‌کنیم و در صورت نیاز، چرخش مورد نظر را انجام می‌دهیم. هزینه صرف شده برای حذف کردن، برابر جمع هزینه‌های لازم برای جست‌وجو و بازیابی مسیر است که هر دو به اندازه $O(\log n)$ می‌باشد. در نتیجه هزینه لازم برای حذف نیز از مرتبه $O(\log n)$ می‌باشد.

مثال ۲۱.۳.۱. گره حاوی مقدار ۸ را می‌خواهیم در درخت اِی.وی.اِل شکل ۶.۱ قسمت (الف) حذف کنیم. گره مورد نظر دارای دو فرزند است که پس از حذف آن باید گره حاوی مقدار ۷ یا ۱۰ را جایگزین



شکل ۶.۱: حذف گره حاوی مقدار ۸ در درخت اِی.وی.اِل.

آن کنیم که ما در این مورد به دلخواه گره حاوی مقدار ۷ را جایگزین می‌کنیم. پس از حذف، درخت به شکل ۶.۱ قسمت (ب) تبدیل می‌شود. با توجه به پشته حاصل و شکل، گره حاوی مقدار ۷ توازن ندارد و نیاز به چرخش RL برای متوازن شدن دارد (ر.ک. ۵.۳.۱). پس از چرخش، درخت به شکل ۶.۱ قسمت (پ) تبدیل می‌شود و در آخر باید توازن گره حاوی مقدار ۶ با توجه به پشته را بررسی کنیم که این گره متوازن می‌باشد و نیازی به چرخش ندارد.

جست‌وجو

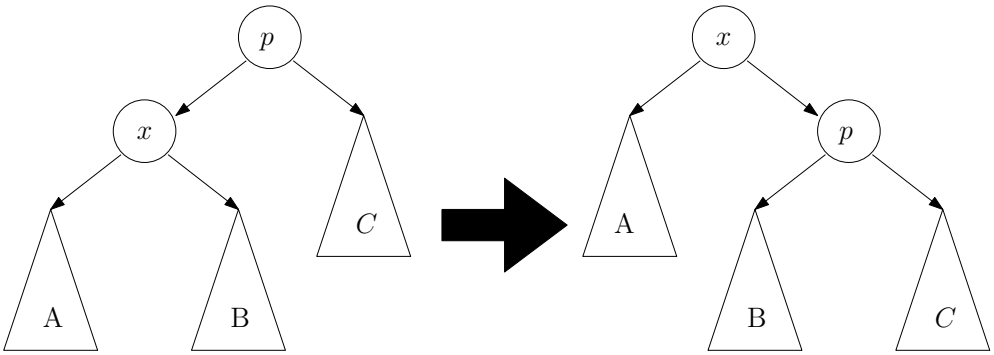
جست‌وجو در درخت متوازن همانند جست‌وجو در درخت دودویی است. به دلیل توازن ارتفاع درخت هزینه جست‌وجو برابر $O(\log n)$ است. هیچ شرط خاصی برای رعایت کردن وجود ندارد و در طول جست‌وجو ساختار درخت تغییری نمی‌کند.

۶.۳.۱ درخت گسترده

درخت گسترده[۱] یک درخت جست‌وجوی دودویی خود-بهبوددهنده[۲] است بدین معنی که هر عنصری که مورد دسترسی[۳] قرار می‌گیرد را در ریشه قرار می‌دهد. منظور از دسترسی هر یک از عملیات‌های حذف، درج یا جست‌وجو است. درخت گسترده از قابلیت‌های جدیدی برخوردار می‌باشد و دسترسی به اطلاعاتی که اخیراً مورد دسترسی قرار گرفته‌اند را سهولت می‌بخشد. در درخت گسترده عناصری که اخیراً دسترسی یافته‌اند سریع‌تر مورد دسترسی قرار می‌گیرند. این درخت اعمال اساسی مانند درج، جست‌وجو و حذف را در $O(\log n)$ زمان سرشکنی انجام می‌دهد. برای خیلی از دنباله‌های غیریکنواخت، این درخت بهتر از سایر درخت‌های جست‌وجو عمل می‌کند. درخت گسترده توسط «دانیل اسلیتور»[۴] و «رابرت تارجان»[۵] در سال ۱۹۸۵ اختراع شد.

در درخت گسترده، همه عملیات معمول در درخت جست‌وجوی دودویی با یک عمل پایه به نام

[۱] Splay tree
[۲] self-adjusting
[۳] Access
[۴] Daniel Dominic Sleator
[۵] Robert Endre Tarjan



شکل ۷.۱: مرحله *zig*.

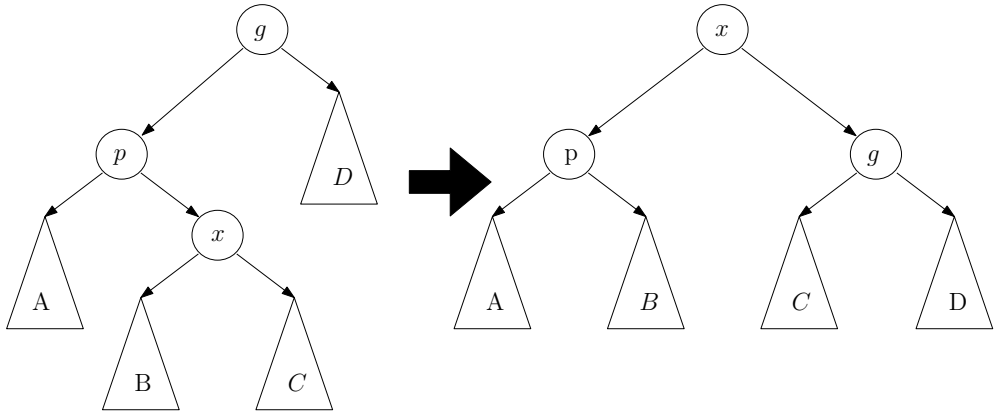
splaying ترکیب می‌شوند. به این معنی که برای یک عنصر خاص، درخت را بازآرایی می‌کند تا عنصر در بالای درخت قرار بگیرد. یک راه انجام این کار این است که نخست یک جست‌وجو در ساختار درختی برای اطلاعات خواسته شده صورت پذیرد و شکلی از چرخش در درخت (عملیات *Splay*) انجام گیرد تا آن عنصر به بالای ساختار آورده شود و این چرخش‌ها ادامه پیدا کند تا عنصر مورد نظر در ریشه درخت قرار گیرد. یک الگوریتم از پایین به بالا می‌تواند با جست‌وجو تلفیق شده و ساختار درختی را مجدداً سازمان‌دهی کند.

عملیات Splay

هنگامی که یک گره x مورد دسترسی قرار می‌گیرد، ساختار درختی گسترده عملیاتی انجام می‌دهد تا آن گره را در ریشه قرار دهد. برای این مقصود یک توالی از *Splay Steps* به انجام می‌رسد که هر مرحله این حرکت را بیش‌تر پیش می‌برد و x را به ریشه نزدیک‌تر می‌کند. با انجام عملیات *splay* در گره مورد نظر بعد از هر دسترسی، گره‌ای که اخیراً مورد دسترسی قرار گرفته در نزدیکی ریشه درخت نگه داشته می‌شود.

فرض کنید گره x مورد دسترسی قرار گرفته است. اگر x گره ریشه نیست پدر آن را p و اگر پدر x هم ریشه نیست، پدر p را g بنامید. اگر گره x ریشه باشد بعد از دسترسی به این گره، دیگر نیازی به انجام عملیات روی این گره نیست تا آن را در ریشه قرار دهیم. فرض کنید که x گره ریشه نباشد.

هر مرحله از عملیات به سه عامل بستگی دارد:



شکل ۸.۱: مرحله $zig - zag$.

- آیا x فرزند چپ یا راست گره پدر خود یعنی p است،
- آیا گره p ریشه است یا نه، و اگر نه
- آیا گره p فرزند چپ یا راست پدر خود یعنی g است (g پدربزرگ x می‌شود).

مراحل *splay*

مرحله zig:

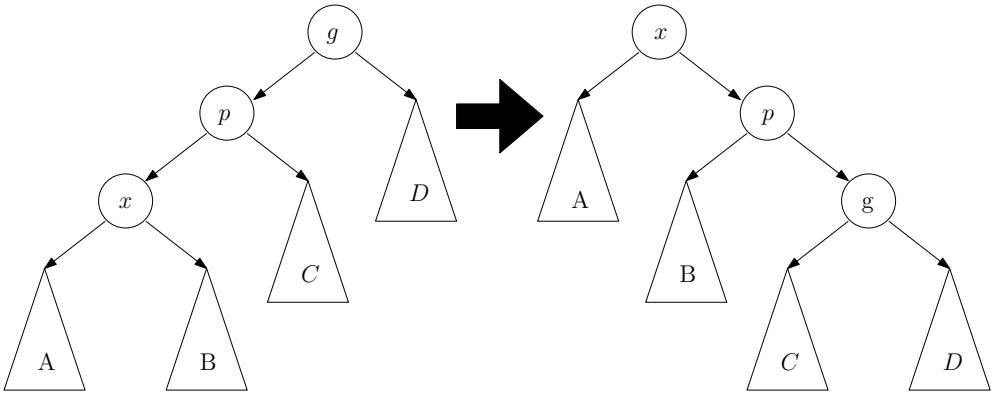
این مرحله هنگامی انجام می‌گیرد که p ریشه باشد. درخت بر روی لبه بین x و p می‌چرخد. تصویر ۷.۱ را ببینید.

مرحله zig-zag:

این مرحله هنگامی اجرا می‌شود که p ریشه نباشد و x فرزند راست پدر خود و p فرزند چپ پدر خود باشد (یا بر عکس). درخت ابتدا روی لبه بین x و p می‌چرخد و سپس روی لبه بین x و پدر جدیدش g چرخش انجام می‌دهد. تصویر ۸.۱ را ببینید.

مرحله Zig-zig:

این مرحله وقتی اجرا می‌شود که p ریشه نباشد و x و p هر دو فرزندان راست یا چپ باشند. تصویر ۹.۱ حالتی را نشان می‌دهد که x و p هر دو فرزندان چپ هستند. درخت ابتدا روی لبه اتصال p با پدرش g، و سپس روی لبه اتصال x با p چرخش انجام می‌دهد.



شکل ۹.۱: مرحله $zig - zig$.

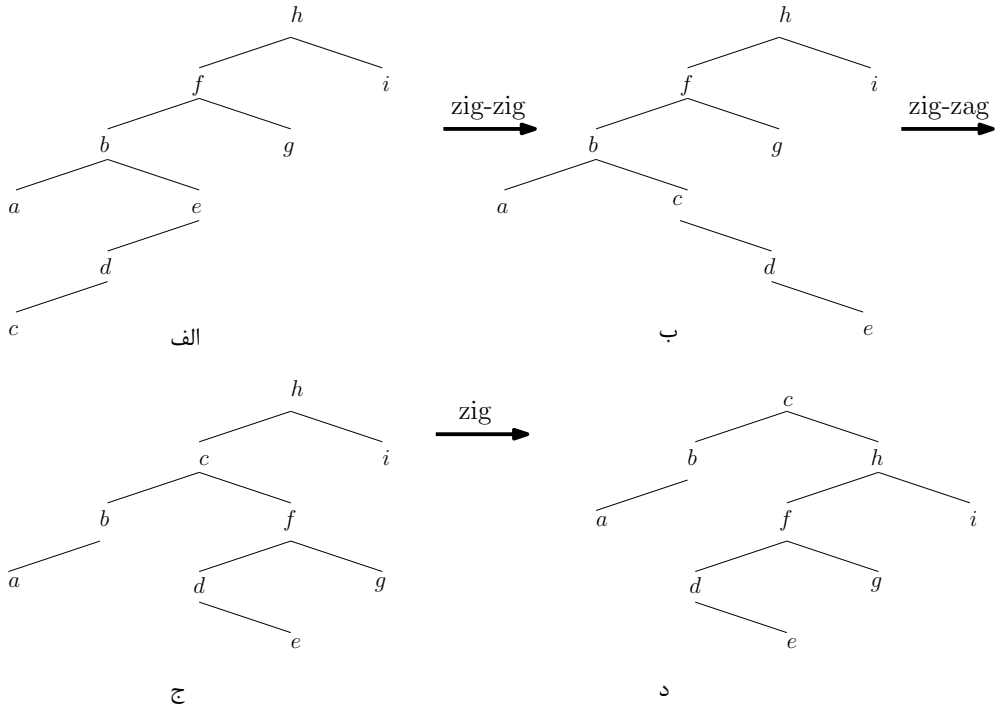
درج در درخت گسترده

برای درج گره x به درخت گسترده، ما ابتدا گره جدید را مانند درخت جست‌وجوی دودویی درج می‌کنیم. سپس با الگوریتم *splay* گره جدید را به بالای درخت می‌آوریم.

مثال ۲۲.۳.۱. در درخت گسترده شکل ۱۰.۱ قسمت (الف) عنصر c مورد دسترسی قرار گرفته است، باید این عنصر در ریشه قرار گیرد. چون که d ریشه نیست و c و d هر دو فرزندان چپ هستند، باید از عملیات $zig - zig$ استفاده کنیم. پس از اجرای عملیات $zig - zig$ بر روی درخت شکل ۱۰.۱ قسمت (الف)، به شکل ۱۰.۱ قسمت (ب) تبدیل می‌شود. در شکل ۱۰.۱ قسمت (ب)، به دلیل این که b ریشه نیست و c فرزند راست b و b فرزند چپ f است، از عملیات $zig - zag$ استفاده می‌کنیم. پس از اجرای عملیات $zig - zag$ بر روی درخت شکل ۱۰.۱ قسمت (ب)، به شکل ۱۰.۱ قسمت (ج) تبدیل می‌شود. به همین ترتیب، چون که پدر c یعنی h، ریشه است، از عملیات zig استفاده می‌کنیم تا به شکل ۱۰.۱ قسمت (د) تبدیل شود. چون عنصری که مورد دسترسی قرار گرفته است، در ریشه قرار دارد، نیازی به انجام عملیات بر روی درخت نیست.

حذف از درخت گسترده

برای حذف گره x، از همان روش حذف در درخت جست‌وجوی دودویی استفاده می‌کنیم. ابتدا گره x را با استفاده از عملیات *splay* در ریشه قرار می‌دهیم و سپس آن گره را مانند حذف از درخت جست‌وجوی



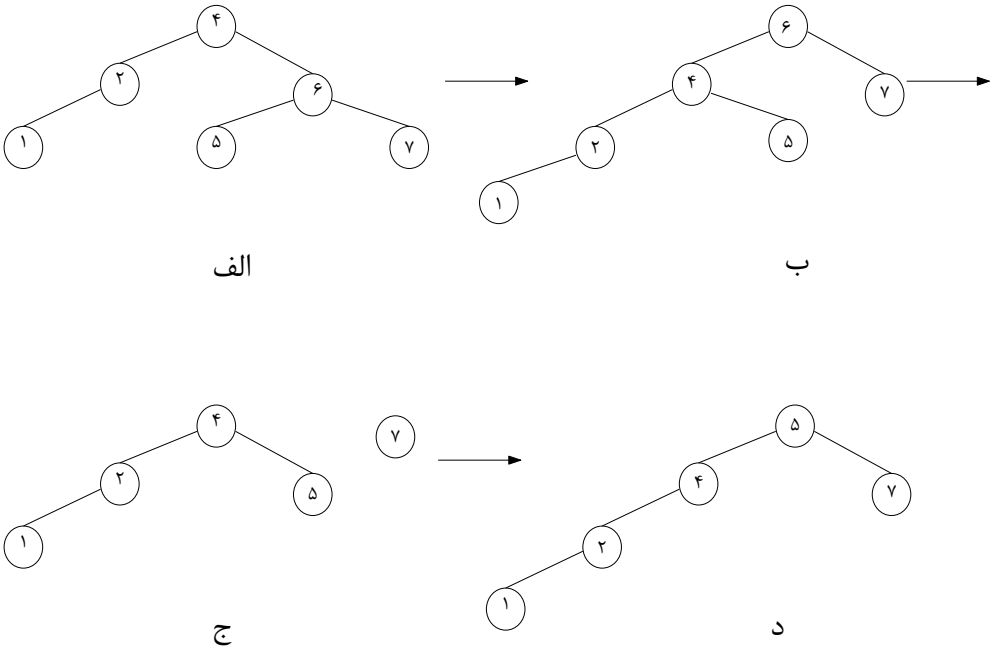
شکل ۱۰.۱: در درخت گسترده قسمت (الف)، عنصر c مورد دسترسی قرار گرفته است.

دودویی، حذف می‌کنیم.

مثال ۲۳.۳.۱. در درخت گسترده شکل ۶.۳.۱ قسمت (الف)، می‌خواهیم گره با مقدار ۶ را از درخت حذف کنیم. برای این کار ابتدا این گره را به وسیله عملیات *splay* در ریشه قرار می‌دهیم. چون که پدر گره‌ای که می‌خواهد حذف شود ریشه است پس از عملیات *zig* استفاده می‌کنیم و درخت به شکل ۶.۳.۱ قسمت (ب)، تبدیل می‌شود. در آخر گره ریشه را با استفاده از روش حذف در درخت جست‌وجوی دودویی حذف می‌کنیم.

تحلیل

یک تحلیل سرشکنی ساده از درخت‌های گسترده، می‌تواند با استفاده از روش پتانسیل بدست آید. فرض کنید $size(r)$ تعداد گره‌های زیردرختی باشد که r ریشه آن است و رتبه r که با $rank(r)$ نمایش داده می‌شود، برابر با $\log_2(size(r))$ باشد. بنابراین تابع پتانسیل $P(t)$ برای یک درخت گسترده t، مجموع رتبه‌های همه گره‌های موجود در درخت است. این باعث خواهد شد که برای درخت‌های با توازن کم‌تر تابع پتانسیل مقداری بزرگ و برای درخت با توازن بهتر تابع پتانسیل مقداری کوچک‌تر باشد. ما می‌توانیم



شکل ۱۱.۱: حذف گره با مقدار ۶ در درخت گسترده قسمت (الف).

هزینه سرشکنی هر یک از عملیات $zig - zig$ یا $zig - zag$ را به صورت زیر محدود به کران بالا کنیم (ر.ک. ۳.۳.۱):

$$\text{هزینه سرشکنی} = cost + P(t_f) - P(t_i) \leq ۳(rank_f(x) - rank_i(x)).$$

که در آن x گره‌ای است که به ریشه برده شده است، اندیس‌های f و i به ترتیب عملیات بعدی و قبلی و $cost$ هزینه واقعی عمل را نشان می‌دهد [۱۶]. هنگامی که بر روی تمام عملیات $splay$ جمع می‌بندیم، بنا به سری تلسکوپی، $۳(rank(r))$ که r ریشه است، حاصل می‌شود که این نیز همان $O(\log n)$ است.

ساختمان داده درخت گسترده، در بدترین حالت، برای نگهداری عناصر در فضای خطی و درج، حذف و جست‌وجو برای یک عنصر در $O(\log n)$ زمان سرشکنی، و در حالت میانگین نیز برای نگهداری عناصر در فضای خطی و درج، حذف و جست‌وجو برای یک عنصر در زمان $O(\log n)$ پشتیبانی می‌کند.

خواص مربوط به بازده

حدس‌ها و قضایای در مورد بدترین حالت زمان اجرا برای به دست آوردن دنباله S از m دسترسی در یک درخت گسترده با n عضو وجود دارد که ما در اینجا برخی از این قضیه‌ها را می‌آوریم و برای اثبات آن‌ها

می‌توانید به [۵، ۱۶] مراجعه کنید.

خاصیت ۱.۳.۲۴ (تعادل). هزینه به دست آوردن دنباله S برابر است با:

$$O(m(1 + \log n) + n \log n).$$

به بیان دیگر می‌توان گفت، درختان گسترده به خوبی درختان جست‌وجوی دودویی متوازن روی دنباله‌ای از حداقل n دسترسی، عمل می‌کنند.

خاصیت ۱.۳.۲۵ (بهبودی ایستا). فرض کنید q_i تعداد دفعاتی باشد که i در S مورد دسترسی قرار گرفته است. در این صورت هزینه اجرا کردن S از مرتبه

$$O\left(m + \sum_{i=1}^{n} q_i \log m/q_i\right)$$

خواهد بود.

خاصیت ۱.۳.۲۶ (انگشت ایستا). فرض کنید i_j عنصری است که در jامین دسترسی از S به دست آمده است و فرض کنید f هر عنصر ثابتی باشد. در این صورت هزینه اجرای S از مرتبه

$$O\left(m + n + \sum_{j=1}^{m} q_i \log(|i_j - f| + 1)\right)$$

است.

خاصیت ۱.۳.۲۷ (مجموعه کاری). فرض کنید t_j تعداد عناصر مجزای مورد دسترسی قرار گرفته بین دسترسی j و عنصر مورد دسترسی قرار گرفته قبل از i_j باشد. در این صورت هزینه اجرای S برابر است با:

$$O\left(m + n + \sum_{j=1}^{m} q_i \log(t_j + 1)\right).$$

خاصیت ۱.۳.۲۸ (انگشت پویا). هزینه اجرای S برابر است با:

$$O\left(m + n + \sum_{j=1}^{m} q_i \log(|i_{j+1} - i_j| + 1)\right).$$

خاصیت ۱.۳.۲۹ (دسترسی ترتیبی). بدون در نظر گرفتن ساختار اولیه درخت گسترده، دسترسی n عنصر از درخت گسترده در زمان $O(n)$ انجام می‌شود.

به طور دقیق‌تر، ثابت شده است که n دسترسی بدون در نظر گرفتن ساختار اولیه درخت گسترده در زمان $۴/۵n$ قابل انجام است.

مزایا و معایب

درخت گسترده دارای مزایا و معایبی می‌باشد که در زیر به صورت مختصر به آنها خواهیم پرداخت.

مزایا

۱. در یک درخت گسترده عناصری که بارها دسترسی یافته‌اند به ریشه نزدیک‌تر شوند تا سریع‌تر مورد دسترسی قرار بگیرند. این ساختار هنگامی کارامدتر خواهد بود که دسترسی به صورت یک‌پارچه نباشد.

۲. ساختارهای درختی گسترده هم‌چنین از مزیت ساختار ساده‌تر نسبت به ساختارهای جستجوی دودویی خود متوازن‌کننده‌ی دیگر نظیر: درخت AVL برخوردارند و کارامدتر هستند.

۳. یکی دیگر از مزایای درخت گسترده امکان ایجاد یک نسخه پایدار است به این معنی که اجازه دسترسی به هر دو نسخه قبلی و جدید را پس از به روز رسانی می‌دهد. این می‌تواند در پیاده‌سازی تابعی مفید باشد و به $O(\log n)$ فضا در هر به روز رسانی نیاز دارد.

معایب

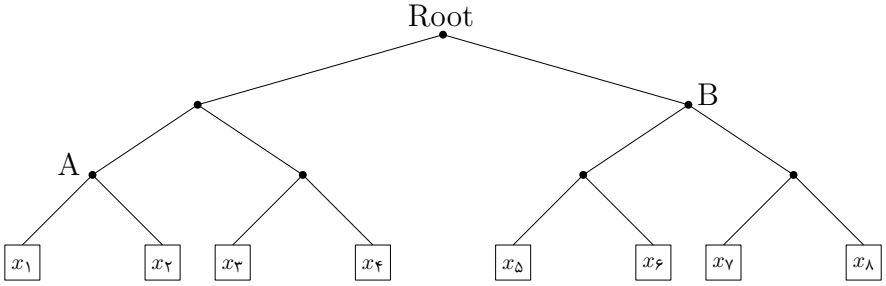
۱. عملکرد درخت گسترده بطور قابل توجهی بدتر از درخت جست‌وجوی دودویی متوازن برای دسترسی یکنواخت است.

۲. یکی از معایب درخت گسترده، این است که احتمال دارد ارتفاع درخت خطی باشد. برای مثال، فرض کنید می‌خواهیم به n عنصر درخت به ترتیب غیرنزولی دسترسی داشته باشیم. چون ارتفاع

درخت متناظر با بدترین زمان دسترسی است، دسترسی دوباره به عنصر اول، باعث می‌شود که عملیاتی که از $O(n)$ است اجرا شود تا درخت دوباره این عنصر را در ریشه قرار دهد. این تأخیر قابل توجهی است.

فصل ۲

درخت ام.سی.اس و تعمیم آن



شکل ۱.۲: شکل، درخت MCS را برای دنباله‌ای با ۸ عنصر نشان می‌دهد.

در این فصل ما ساختمان داده درخت MCS[۱] را معرفی خواهیم کرد. این ساختمان داده درختی، معیاری برای ارزیابی یک دنباله به شمار می‌آید. ساختمان داده درخت MCS برای نخستین بار در مقاله کورتس با همکارانش تحت عنوان «تفکیک دورنگی‌ها با دو جعبه» در سال ۲۰۰۹ مورد استفاده قرار گرفت. تعمیم ساختمان داده درخت MCS را در دو زمینه بررسی خواهیم کرد: اول تعمیم ساختمان داده درخت MCS را برای تابع امتیازدهی تجزیه‌پذیر و یکنوا بررسی خواهیم کرد، سپس تعمیم این ساختمان داده را برای یک دنباله پویا که از حذف و درج نقاط پشتیبانی کند، شرح خواهیم داد.

۱.۲ درخت MCS

درخت MCS به عنوان یک شاخص برای یک دنباله ثابت $S = (x_i)_{i\in[۱..n]}$ برای n عنصر، تعریف شده است [۷] . هر عنصر x_k از دنباله دارای وزن $w(x_k)$ می‌باشد، به طوری که هر زمان که وزن $w(x_k)$ بروز می‌شود، زیردنباله متوالی $(x_i)_{i\in[l..r]}$ از S که $\sum_{i=l}^{r} w(x_i)$ را به بیشترین مقدار می‌رساند، در زمان $O(\log n)$ بدست می‌آید. منظور از زیردنباله متوالی، زیردنباله‌ای از دنباله اصلی است که اندیس‌ها در آن متوالی هستند.

تعریف ۱.۱.۲. فرض کنید $S = (x_i)_{i\in[۱..n]}$ یک دنباله باشد. درخت MCS یک درخت متوازن (ر.ک. ۱۵.۳.۱) روی n عنصر $x_۱, x_۲, x_۳, ...x_n$ است به طوری که:

۱. دارای n برگ که نشان دهنده دنباله‌ی $x_۱, x_۲, x_۳, ...x_n$ است و برگ kام از چپ به راست نشان دهنده x_k باشد.

۲. هر گره داخلی v نشان دهنده فاصله زیردنباله متناظر با برگ‌های زیردرختی است که v ریشه آن است.

[۱] Maximum Consecutive Subsequence

شکل ۱.۲، درخت MCS را برای دنباله‌ای با ۸ عنصر نشان می‌دهد. گره A، B و ریشه به ترتیب نشان دهنده فواصل $[x_1, x_2]$، $[x_5, x_6, x_7, x_8]$ و $[x_1, x_2, ..., x_8]$ است.

اصطلاح فاصله[1] به عنوان اندیس‌های اولین و آخرین عناصر یک زیردنباله متوالی از S که با دنباله اصلی مشترک است، استفاده می‌شود.

تعریف ۲.۱.۲. برای زیردرنباله متوالی $(x_i)_{i\in[l..r]}$، فاصله این زیردنباله را با $[x_l, x_r]$ نشان می‌دهیم و $[x_{l'}, x_{r'}]$ به طوری که $l \le l' \le r' \le r$ را یک عامل[2] از فاصله $[x_l, x_r]$ گوییم. عاملی را که شامل اولین اندیس از زیردنباله $(x_i)_{i\in[l..r]}$ باشد، پیشوند[3] و عاملی که شامل آخرین اندیس از زیردنباله $(x_i)_{i\in[l..r]}$ باشد را یک پسوند[4] از فاصله $[x_l, x_r]$ گویند.

برای هر گره N فواصل زیر ذخیره می‌شود:

$I(N)$: نشان دهنده فاصله گره N (اگر N گره برگ باشد که در آن x_k ذخیره شده است آنگاه $I(N) = [x_k]$ است)؛

$L(N)$: نشان دهنده یک پیشوند غیر تهی از فاصله $I(N)$ که مجموع وزن‌ها را بیشینه می‌کند؛

$R(N)$: نشان دهنده یک پسوند غیر تهی از $I(N)$ که مجموع وزن‌ها بیشینه می‌کند؛

$M(N)$: نشان دهنده یک عامل غیر تهی از $I(N)$ که مجموع وزن‌ها را بیشینه می‌کند.

واضح است، اگر v گره ریشه باشد آنگاه $I(v)$، $L(v)$، $R(v)$ و $M(v)$ به ترتیب به شکل فواصل $[x_1, x_2, ..., x_n]$، $[x_1, x_2, ..., x_j]$، $[x_i, x_2, ..., x_n]$ و $[x_i, x_2, ..., x_j]$ خواهد بود که مجموع وزن‌ها را بیشینه می‌کنند.

این درخت دارای پیچیدگی خطی است و عملگرهای زیر برای محاسبه درخت MCS استفاده می‌شود [۷]:

---
[1] Interval
[2] Factor
[3] Prefix
[4] Suffix

- + : دو فاصله هم‌جوار را متصل کرده و برمی‌گرداند. این عمل در زمان $O(1)$ انجام می‌شود؛
- $MaxSum\{I_1, I_2, ..., I_m\}$: فاصله‌ای را که دارای حداکثر مجموع وزن در بین فاصله‌های $I_1, I_2, ..., I_m$ است، برمی‌گرداند و در زمان $O(1)$ انجام می‌شود؛
- $CreateTree(X)$: درخت MCS با گره‌هایی خالی از اطلاعات را با توجه به $X = \{x_1, x_2, ..., x_n\}$ در زمان $O(1)$ می‌سازد؛
- $UpdateNode(N)$: اگر گره N یک گره برگ باشد اطلاعات آن با استفاده از اطلاعات داخلی گره و اگر گره N، یک گره داخلی باشد با استفاده از اطلاعات فرزندان آن، اطلاعاتش را به روز رسانی می‌کند و در زمان $O(1)$ انجام می‌شود؛
- $UpdateTree$: به روز رسانی کل درخت را در زمان $O(n)$ انجام می‌دهد؛
- $UpdateLeaf(x_k)$: وقتی وزن عنصر x_k تغییر پیدا می‌کند، اِعمال این عملگر بر روی آن باعث می‌شود که عملگر $UpdateNode(.)$ برای تمامی گره‌ها در مسیر این گره تا ریشه در زمان $O(\log n)$ اجرا شود؛
- $MaxSumInterval$: فاصله $M(v)$ را که v گره ریشه است در زمان $O(1)$ برمی‌گرداند؛
- $MaxSumLeftInterval$: فاصله $L(v)$ را که v گره ریشه است در زمان $O(1)$ برمی‌گرداند؛
- $MaxSumRightInterval$: فاصله $R(v)$ را که v گره ریشه است در زمان $O(1)$ برمی‌گرداند؛
- $MaxSumIntervalContains(x_k)$: فاصله‌ای را با حداکثر مجموع وزن‌ها که شامل x_k است در زمان $O(\log n)$ محاسبه می‌کند؛

با توجه به ساختار درخت MCS، مطالب گفته شده و عملگرهای موجود تعریف شده در بالا، نتیجه گرفته می‌شود که می‌توان این درخت را برای یک دنباله وزن‌دار، از پایین به بالا ساخت و هر وقت که وزن عنصری تغییر پیدا کند، درخت را در زمان $O(\log n)$ به روز رسانی کرد و زیردنباله متوالی بهینه را به دست آورد. با توجه به مطالب گفته شده نتایج را در زیر به عنوان یک گزاره می‌آوریم [۷] .

گزاره ۳.۱.۲. درخت MCS بر روی دنباله $X = \{x_1, x_2, ..., x_n\}$ را می‌توان در زمان $O(n)$ ساخت و هر زمان که وزن عنصری تغییر پیدا کند فاصله مربوط به زیردنباله متوالی با حداکثر مجموع وزن از این عنصر در زمان $O(\log n)$ بروز می‌شود. همچنین این ساختمان داده می‌تواند زیردنباله متوالی با حداکثر مجموع وزن که شامل عنصر خاص x_k از دنباله X است در زمان $O(\log n)$ محاسبه کند.

۲.۲ تعمیم درخت MCS برای تابع امتیازدهی f

همان طور که در بخش قبل گفته شد، درخت MCS به عنوان یک شاخص برای یک دنباله وزن‌دار ثابت در نظر گرفته شده است ولی مسأله مستطیل بهینه مسطح به گونه‌ای است که هر زیرمجموعه‌ای از نقاط داده شده با ارزشی که تابع امتیازدهی f به آن نسبت می‌دهد، همراه است. در این بخش می‌خواهیم درخت MCS بر روی یک دنباله ثابت که هر زیردنباله آن با ارزشی که تابع امتیازدهی f به آن نسبت می‌دهد، را به گونه‌ایی تعمیم دهیم، که همان خاصیت‌های درخت MCS را دارا باشد [۲].

تابع تجزیه‌پذیر و یکنوا f و تابع g را که تابع ترکیب متناظر با f می‌باشد در نظر بگیرید. با در نظر گرفتن دنباله ثابت $S = (x_i)_{i\in[1..n]}$، هر عنصر x_i از S با ارزش $f(x_i)$ همراه است. همان طور که می‌دانید درخت MCS ساختمان داده‌ای متشکل از یک درخت دودویی متوازن با n برگ است، که در آن برگ‌های درخت از چپ به راست عناصر $x_1, x_2, x_3, ..., x_n$ از S را در خود ذخیره می‌کنند.

اصطلاح فاصله را همان فاصله تعریف شده در درخت MCS در نظر بگیرید. فاصله مربوط به زیردنباله $(x_i)_{i\in[l..r]}$ را با $[x_l, x_r]$ و $[x_k, x_k]$ را با $[x_k]$ نشان می‌دهیم. فرض کنید که $\emptyset$ فاصله با ارزش صفر تعریف شود، و با توجه به فواصل $I_1, I_2, ..., I_t$ داده شده فرض کنید که $\max\{I_1, I_2, ..., I_t\}$ فاصله‌ای با حداکثر امتیاز در بین این فاصله‌ها باشد. منظور از امتیاز فاصله، امتیازی است که تابع امتیازدهی به زیردنباله مربوط به آن فاصله نسبت می‌دهد. هر گره v از درخت T چهار فاصله با حداکثر امتیاز را به شرح زیر در خود ذخیره می‌کند. اگر $x_l, x_{l+1}, ..., x_r$ از چپ به راست برگ‌های درخت T و نوه‌های گره v باشد:

- $I(v) = [x_l, x_r]$، اطلاعات کامل در مورد دنباله؛
- $L(v) = \max\{[x_l, x_l], [x_l, x_{l+1}], ..., [x_l, x_r], \emptyset\}$، اطلاعات در مورد بهترین پیشوند از دنباله؛

- $R(v) = \max\{[x_l, x_r], [x_{l+1}, x_r], ..., [x_r, x_r], \emptyset\}$، اطلاعات در مورد بهترین پسوند از دنباله؛
- $M(v) = \max\{\max\{[x_{l'}, x_{r'}] : l \le l' \le r' \le r\}, \emptyset\}$، اطلاعات در مورد بهترین عامل از دنباله؛

که اصطلاحات عامل، پیشوند و پسوند همان اصطلاحات تعریف شده در ۲.۱.۲ می‌باشد. پس زیردنباله متوالی با حداکثر امتیاز از دنباله‌ی S توسط فاصله $M(r)$ که r ریشه درخت T است، به دست می‌آید. برای هر دو فاصله هم‌جوار $[x_l, x_k]$ و $[x_{k+1}, x_r]$ فرض کنید که $[x_l, x_k] + [x_{k+1}, x_r]$ تعریف فاصله $[x_l, x_r]$ باشد که امتیاز آن با توجه به تجزیه‌پذیر بودن f به صورت $f([x_l, x_r]) = g(f([x_l, x_k]), f([x_{k+1}, x_r]))$ از طریق ترکیبی از امتیازات $[x_l, x_k]$ و $[x_{k+1}, x_r]$ به دست می‌آید.

بدیهی است برای I، $I + \emptyset = \emptyset + I = I$ است. نکته قابل توجه در درخت MCS این است که برای هر گره v که v_1 و v_2 به ترتیب فرزندان چپ و راست آن است، اگر ما $I(v_1)$، $L(v_1)$، $R(v_1)$، $M(v_1)$، $I(v_2)$، $L(v_2)$، $R(v_2)$ و $M(v_2)$ را محاسبه کرده باشیم آنگاه می‌توانیم $I(v)$، $L(v)$، $R(v)$ و $M(v)$ را برای v با تعداد ثابتی عملیات به شرح زیر محاسبه کنیم:

- $I(v) = I(v_1) + I(v_2)$
- $L(v) = \max\{L(v_1), g(I(v_1), L(v_2))\}$
- $R(v) = \max\{g(R(v_1), I(v_2)), R(v_2)\}$
- $M(v) = \max\{M(v_1), M(v_2), g(R(v_1), L(v_2))\}$

توجه کنید که هر پیشوند از $I(v_1) + I(v_2)$ یا به صورت یک پیشوند از $I(v_1)$ یا به صورت $I(v_1)$ همراه با پیشوندی از $I(v_2)$ است. هر پسوندی از $I(v_2) + I(v_1)$ یا به صورت پسوندی از $I(v_2)$ یا به صورت پسوندی از $I(v_1)$ همراه با $I(v_2)$ است و هر عامل از $I(v_2) + I(v_1)$ یا به صورت عاملی از $I(v_1)$ یا به صورت عاملی از $I(v_2)$ یا به صورت یک پسوند از $I(v_1)$ همراه با یک پیشوند از $I(v_2)$ است. این مشاهدات می‌تواند هر زمان که امتیاز یک عنصر مانند x_i از S تغییر پیدا کند، مورد استفاده قرار گیرد. به روز رسانی

از طریق تعداد ثابتی عملیات برای هر گره که در مسیر برگ درخت که عنصر x_i در آن ذخیره شده است تا ریشه درخت انجام می‌شود. چون که درخت T متوازن است این تعداد متناظر با $O(\log n)$ عملیات است. ساختمان داده درخت MCS همچنین از پرس‌وجو برای پیدا کردن زیرمحدوده f-بهینه از R که R خود زیرمحدوده‌ایی از S است، در $O(\log n)$ عملیات پشتیبانی می‌کند. کافی است گره‌های v_l، v_r و $v_{l,r}$ را که به ترتیب گره‌هایی از درخت هستند که در آن اولین عنصر R، آخرین عنصر R و پایین‌ترین گره جد مشترک v_l و v_r هستند را در زمان $O(\log n)$ پیدا کنیم. زیرمحدوده f-بهینه R همان فاصله‌ای است که توسط $M(v_{l,r})$ ذخیره شده است.

در این ساختمان داده برای اینکه عنصری را حذف کنیم، می‌توانیم امتیاز این عنصر را با $f(\emptyset)$ در درخت به روز رسانی کنیم تا امتیاز این عنصر تأثیری در زیردنباله متوالی با حداکثر امتیاز نداشته باشد. اصطلاحاً این کار را غیرفعال‌سازی [۱] عنصر گویند و اگر عنصری امتیازش را با مقداری که تابع امتیازدهی f به آن نسبت می‌دهد، در درخت به روز رسانی کنیم آن را فعال‌سازی [۲] آن عنصر گویند. یکی از معایب این روش این است که با حذف عنصری، تغییر فیزیکی در درخت صورت نمی‌گیرد و زمان اجرا برای به روز رسانی کم‌تر نخواهد شد.

ساختمان داده موجود یک ساختمان داده نیمه-پویا[۳] می‌باشد که در اینجا رفتار نیمه-پویا به این معنی است که اجازه می‌دهد وزن عنصر تغییر پیدا کند، و در عین حال نه عناصر موجود حذف و نه عناصر جدید درج شوند. این مشاهدات را در زیر به عنوان یک لم می‌آوریم.

لم ۱.۲.۲. فرض کنید S دنباله‌ای ثابت با n عنصر باشد و همچنین f تابع امتیازدهی تجزیه‌پذیر و یکنوایی باشد که به عنوان آرگومان ورودی زیردنباله‌ای از S را دریافت می‌کند. در این صورت ساختمان داده نیمه-پویایی برای نگهداری S با استفاده از فضای خطی وجود دارد که از جست‌وجو برای یک عنصر در $O(\log n)$ مقایسه، به روز رسانی امتیاز یک عنصر در $O(\log n)$ ترکیب امتیاز و یافتن زیرمحدوده f-بهینه در $O(\log n)$ مقایسه و ترکیب امتیاز پشتیبانی می‌کند.

یکی از مشکلات این ساختمان داده موجود این است که نمی‌توان آن را برای یک دنباله پویا استفاده

[۱] Deactivation
[۲] Activation
[۳] Semi-dynamic

کرد، ما در بخش بعد به بررسی تعمیمی از این ساختمان داده می‌پردازیم که برای یک دنباله پویا نیز قابل استفاده باشد.

۳.۲ تعمیم درخت MCS برای یک دنباله پویا

ساختمان داده کاملاً پویا[1] دارای این خاصیت است که در آن علاوه بر تغییر پیدا کردن وزن عناصر، اجازه می‌دهد که عناصر موجود حذف یا عناصر جدید درج شوند. ساختمان داده درخت MCS یک ساختمان داده نیمه-پویا است، این ساختمان داده نیمه-پویای موجود را نمی‌توان به سادگی با یک ساختمان داده کاملاً پویا با ویژگی همان درخت MCS سازگار کرد.

می‌خواهیم ساختمان داده درخت MCS را به یک ساختمان داده کاملاً پویا تبدیل کنیم، که از حذف و درج عناصر پشتیبانی کند. این ساختمان داده می‌تواند برای اندیس یک دنباله پویا $S = (x_i)_{i\in[1..n]}$ از n عنصر مورد استفاده قرار گیرد به طوری که هر زمان یک عنصر حذف و یا درج شود، زیردنباله متوالی $S' = (x_i)_{i\in[l..r]}$ از S که $f(S')$ را بهینه می‌کند، می‌تواند در $O(\log n)$ ترکیب امتیاز و مقایسه محاسبه گردد. در زیر ویژگی این ساختمان داده، که آن را درخت $MCS\ AVL$ می‌نامیم، نشان می‌دهیم.

فرض کنید $S = (x_i)_{i\in[1..n]}$ باشد، f تابع امتیازدهی تجزیه‌پذیر و یکنوا و تابع g تابع ترکیب متناظر با f باشد. فرض کنید T یک درخت AVL از n گره باشد به طوری که برای هر $i \in [1..n]$، iامین گره در پیمایش میان‌ترتیب از درخت T، عنصر x_i از S در آن ذخیره شده باشد. این درخت وجود دارد زیرا درخت متوازن (AVL)، یک نوع درخت جست‌وجوی دودویی خودمتوازن‌کننده (ر.ک. ۱۶.۳.۱) است و چون که ما این درخت جست‌وجوی دودویی را بر اساس اندیس دنباله S می‌سازیم وقتی که آن را به صورت میان‌ترتیب پیمایش می‌کنیم دنباله مرتب اولیه حاصل می‌شود. عملیات درج و حذف در یک درخت متوازن در بدترین حالت به اندازه $O(\log n)$ خواهد بود به طوری که n تعداد گره‌های درخت می‌باشد. ساختمان داده درخت موجود را به یک ساختمان داده درخت MCS تعمیم می‌دهیم. برای هر گره v از درخت T، که در پیمایش میان‌ترتیب از زیردرخت با ریشه v عناصر $x_l, x_{l+1}, .., x_r$ از S گزارش شده باشد، همان چهار فاصله $I(v)$، $L(v)$، $R(v)$ و $M(v)$ در ساختمان داده درخت MCS را استفاده می‌کنیم. یکی از وجه

[1] Fully Dynamic

اشتراک‌هایی که این ساختمان داده با ساختمان داده نیمه-پویای قبلی دارد همان چهار حداکثر فاصله‌ایی است که در هر گره ذخیره می‌شود، و یکی از تفاوت‌های آن این است که در ساختمان داده نیمه-پویای قبلی تمام عناصر در برگ‌ها ذخیره می‌شد ولی همان طور که می‌بینید این ساختمان داده به این صورت نمی‌باشد. اگر در گره v عنصر x_k ذخیره شده باشد و v_1 و v_2 به ترتیب فرزند چپ و راست v باشد و برای آنها $I(v_1)$، $L(v_1)$، $R(v_1)$، $M(v_1)$، $I(v_2)$، $L(v_2)$، $R(v_2)$ و $M(v_2)$ محاسبه شده باشد آنگاه می‌توان $I(v)$، $L(v)$، $R(v)$ و $M(v)$ را برای v با تعداد ثابتی ترکیب امتیاز به شرح زیر محاسبه کنیم:

- $I(v) = I(v_1) + [x_k] + I(v_2)$
- $L(v) = \max\{L(v_1), I(v_1) + [x_k] + L(v_2)\}$
- $R(v) = \max\{R(v_1) + [x_k] + I(v_2), R(v_2)\}$
- $M(v) = \max\{M(v_1), M(v_2), R(v_1) + [x_k] + L(v_2)\}$

در این محاسبه ارزش f برای هر فاصله به فرم $I_1 + [x_k] + I_2$ برابر است با:

$$g(g(f(s_1), f(x_k)), f(s_2))$$

به طوری که s_1 و s_2 زیردنباله‌های مربوط به فاصله‌های I_1 و I_2 باشد. برای گره تهی v در نظر می‌گیریم:

$$I(v) = L(v) = R(v) = M(v) = \emptyset$$

این نشان می‌دهد که این فاصله‌ها چگونه برای برگ‌ها و گره‌ها با یک فرزند محاسبه می‌شود. نشان می‌دهیم محاسبه $L(v)$ با در نظر گرفتن دو حالت که $L(v)$ شامل x_k نیست و در حالتی که $L(v)$ شامل x_k است، درست است.

حالت اول) اگر x_k در $L(v)$ نباشد آنگاه $L(v)$ فقط باید عناصر $x_l, x_{l+1}, .., x_{k-1}$ را در نظر بگیرد که در این صورت باید برابر با $L(v_1)$ می‌باشد.

حالت دوم) در غیر این صورت اگر x_k متعلق به $L(v)$ باشد آنگاه $L(v)$ باید به شکل $I(v_1) + [x_k] + I_2$ باشد که I_2 به صورت پیشوندی از $[x_{k+1}, x_r]$ است. مشاهده می‌شود که $L(v_2)$ فاصله‌ایی به همان شکل

I_2 است و f را حداکثر می‌کند. از آنجایی که g یکنوا (فرض کنید صعودی باشد) در مولفه اول و دوم است، داریم:

$$\begin{aligned} f(I(v_1) + [x_k] + I_2) &= g(g(f(I(v_1)), f(x_k)), f(I_2)) \\ &\leq g(g(f(I(v_1)), f(x_k)), f(L(v_2))) \\ &= f(I(v_1) + [x_k] + L(v_2)) \end{aligned}$$

بنابراین اگر x_k متعلق به $L(v)$ باشد آنگاه $I(v_1) + [x_k] + L(v_2)$ یک انتخاب صحیح برای $L(v)$ می‌باشد.

مشابه این استدلال را می‌توان برای $R(v)$ و $M(v)$ آورد. اگر عناصر S در یک کلید مرتب باشند آنگاه این درخت یک درخت جست‌وجوی دودویی در آن کلید می‌باشد و حذف و اضافه با استفاده از معیار جست‌وجوی دودویی انجام می‌شود. در غیر این صورت اگر عناصر S نامرتب باشند برای هر گره v، در زیردرختی که این گره ریشه آن است و دارای چند گره می‌باشد به منظور انجام اعمال حذف و درج گره‌ها، از رتبه[1] آن گره‌ها استفاده می‌کنیم [۶]. رتبه یک گره، تعداد گره‌های قبل از این گره در پیمایش میان‌ترتیب می‌باشد. اندازه[2] یک گره، تعداد گره‌های زیردرخت آن گره به اضافه یک است. الگوریتم پیدا کردن یک گره با استفاده از رتبه داده شد به این صورت می‌باشد:

ورودی الگوریتم ۳.۲، x است که ریشه درخت v و رتبه داده شده i می‌باشد و خروجی الگوریتم گره‌ای با رتبه داده شده است.

درج کردن یک گره در درخت شامل دو مرحله می‌باشد:

- مرحله اول، همانند درج یک گره جدید در درخت جست‌وجوی دودویی عمل می‌کنیم و در محل مناسب به عنوان فرزند یکی از گره‌های موجود گره جدید را درج می‌کنیم، و
- مرحله دوم، از همان گره به سمت ریشه درخت حرکت می‌کنیم برای حفظ خاصیت متوازن بودن درخت همانند متوازن کردن یک درخت نامتوازن عمل می‌کنیم و اگر لازم شد چرخش انجام می‌دهیم

[1] Rank
[2] Size

Algorithm 2.3.1: SELECT(x, i)

```
Input: x, i.
Output: Node with Rank i.
r = size[left[x]] + 1;
if i == r then
    return x;
else
    if i < r then
        return SELECT(left[x], i);
    else
        return SELECT(right[x], i − r);
    end
end
```

و در هر مرحله از چرخش، اطلاعات گره‌هایی که موقعیت آنها تغییر پیدا می‌کند را با استفاده از اطلاعات فرزندانشان، به روز رسانی می‌کنیم.

حذف کردن گره از درخت نیز شامل دو مرحله می‌باشد:

- مرحله اول، با استفاده از رتبه عنصری که باید حذف شود آن عنصر را در درخت می‌یابیم سپس آن عنصر را همانند حذف از درخت جست‌وجوی دودویی حذف کرده، و
- مرحله دوم، درخت را به وسیله چرخش‌ها متوازن می‌سازیم که مشابه درج کردن در هر مرحله از چرخش اطلاعات گره‌هایی که موقعیت آنها تغییر پیدا کرده است با استفاده از اطلاعات فرزندانشان، به روز رسانی می‌شود.

در هر صورت، هرگاه یک عنصر را وارد یا خارج می‌کنیم ($O(\log n)$ مقایسه انجام می‌دهیم. علاوه بر این هنگامی که یک گره برگ وارد می‌شود، یا یک گره برگ یا گره با یک فرزند حذف می‌شود ما از پایین به بالا برای تمام گره‌ها در مسیر از آن گره تا ریشه یک به روز رسانی انجام می‌دهیم. اگر به روز رسانی یک گره نیازی به چرخش نداشته باشد، آنگاه اطلاعات آن گره در $O(۱)$ ترکیب امتیاز از اطلاعات مشابه از دو گره فرزندانش که در دست می‌باشد، تکمیل می‌شود. در غیر این صورت، اگر به روز رسانی برای یک گره نیاز به چرخش داشته باشد، چرخش را انجام داده، سپس یک به روز رسانی برای اطلاعات تکمیل شده از پایین به بالا برای گره‌های تغییر پیدا کرده در چرخش که این گره‌ها عبارتند از گره مورد نظر و گره‌های گرفته شده از موقعیت آن می‌باشد، با استفاده از اطلاعات فرزندان جدید خود انجام می‌دهیم، که

این هم مجدداً در $O(1)$ ترکیب امتیاز انجام می‌شود چون چرخش حداکثر در سه گره تاثیر می‌گذارد و در هر چرخش اطلاعات با استفاده از اطلاعات فرزندان در حال حاضر خود تکمیل می‌شوند. به روز رسانی برای هر گره همیشه نیاز به $O(1)$ ترکیب امتیاز دارد، و آنگاه برای هر حذف یا درج در مجموع $O(\log n)$ ترکیب امتیاز استفاده می‌شود. زیردنباله S' مربوط به فاصله $M(v)$ که v ریشه درخت است، زیردنباله‌ای از S است که تابع امتیازدهی f را حداکثر می‌کند و در هر حذف و درج بروز شده است.

پرس‌وجو برای به دست آوردن زیرمحدوده بهینه در درخت را می‌توان به شرح زیر انجام داد. فرض کنید x_l و x_r اولین و آخرین عناصر محدوه ورودی R باشد. با استفاده از $O(\log n)$ مقایسه پیدا می‌کنیم: گره v_l که در آن x_l ذخیره شده، گره v_r که در آن x_r ذخیره شده و پایین‌ترین گره جد مشترک $v_{l,r}$ از v_l و v_r. سپس چهار متغیر کمکی R_l، M_l، L_r و M_r از فاصله‌ها را در نظر می‌گیریم و آن‌ها را به صورت زیر محاسبه خواهیم کرد:

در ابتدا $R_l = M_l = \emptyset$ را مقداردهی اولیه می‌کنیم و پس از آن برای هر گره v گرفته شده از مسیر پایین به بالای v_l به سمت گره فرزند چپ $v_{l,r}$، به طوری که عنصر x_k در v متعلق به R ذخیره شده است، به روز رسانی را به صورت زیر انجام می‌دهیم:

$$R_l = \max\{R_l + [x_k] + I(v_r), R(v_r)\}$$

و

$$M_l = \max\{M_l, M(v_r), R_l + [x_k] + L(v_r)\}$$

به این ترتیب که در آن v_r فرزند راست v است. به طور مشابه، $R_r = M_r = \emptyset$ را مقداردهی اولیه می‌کنیم و برای هر گره v گرفته شده از مسیر پایین به بالای v_r به سمت گره فرزند راست $v_{l,r}$، به طوری که عنصر x_k در v متعلق به R ذخیره شده باشد، قرار می‌دهیم:

$$L_r = \max\{L(v_1), I(v_1) + [x_k] + L_r\}$$

و

$$M_r = \max\{M(v_1), M_r, R(v_1) + [x_k] + L_r\}$$

به طوری که در آن v_1 فرزند چپ v است. در نهایت، زیرمحدوده بهینه از R با $\max\{M_l, M_r, R_l + [x_{l,r}] + L_r\}$ برابر است، به طوری که $x_{l,r}$ عنصری از S باشد و در $v_{l,r}$ ذخیره شده باشد. تمام عملیات مستلزم $O(\log n)$ ترکیب امتیاز می‌باشد. نتیجه این مشاهدات را در لم زیر جمع‌بندی می‌کنیم.

لم ۱.۳.۲. فرض کنید S دنباله‌ای پویا با n عنصر باشد و همچنین f تابع امتیازدهی تجزیه‌پذیر و یکنوایی باشد که به عنوان آرگومان ورودی زیردنباله‌ای از S را دریافت می‌کند. در این صورت ساختمان داده کاملاً پویایی برای نگهداری S با استفاده از فضای خطی وجود دارد که از جست‌وجو برای یک عنصر در $O(\log n)$ مقایسه، به روز رسانی وزن یک عنصر در $O(\log n)$ ترکیب امتیاز، درج و حذف یک عنصر در $O(\log n)$ مقایسه و ترکیب امتیاز، و یافتن زیرمحدوده f-بهینه در $O(\log n)$ مقایسه و ترکیب امتیاز پشتیبانی می‌کند.

در میان خاصیت‌های درخت گسترده می‌خواهیم از خاصیت انگشت پویا[۱] استفاده کنیم. خاصیت انگشت پویا توسط «دانیل اسلیتور»[۲] و «رابرت تارجان»[۳] [۱۶] حدس زده شده و توسط کول[۴] و همکارانش [۵] اثبات شده است. خاصیت انگشت پویا به این صورت بود که هر دنباله از m دسترسی[۵] در درخت گسترده دلخواه با n گره دارای هزینه $O\left(m + n + \sum_{j=1}^{n} \log(d_j + 1)\right)$ چرخش است، که در آن d_j اختلاف بین رتبه‌های دسترسی jام و $(j-1)$ام می‌باشد. برای $j = 0$، عنصر jام، عنصر ذخیره شده در ریشه است. اگر ما بخواهیم n دسترسی انجام دهیم که اختلاف بین رتبه‌های دسترسی‌های متوالی ۱ باشد، با استفاده از خاصیت انگشت پویا و درخت گسترده در $O(n)$ قابل انجام است، در حالی که اگر بخواهیم این کار را با استفاده از یک درخت AVL انجام دهیم در زمان $O(n \log n)$ انجام می‌شود. مشاهده می‌کنیم که در درخت $MCS\ AVL$ می‌توان به جای درخت AVL از درخت $Splay$ استفاده نمود و لم بعدی را برای ساختمان داده درخت $MCS\ Splay$ به دست آورد.

لم ۲.۳.۲. فرض کنید S دنباله‌ای پویا از n عنصر باشد و همچنین f تابع امتیازدهی تجزیه‌پذیر و یکنوایی

[۱] Dynamic Finger Property
[۲] Daniel dominic sleator
[۳] Robert endre tarjan
[۴] Cole
[۵] Accesses

باشد که به عنوان آرگومان ورودی زیردنباله‌ای از S را دریافت می‌کند. در این صورت ساختمان داده کاملاً پویایی برای نگهداری S با استفاده از فضای خطی وجود دارد که از جست‌وجو برای یک عنصر در $O(\log n)$ مقایسه سرشکنی[1]، به روز رسانی وزن یک عنصر در $O(\log n)$ ترکیب امتیاز سرشکنی، درج و حذف یک عنصر در $O(\log n)$ مقایسه و ترکیب امتیاز سرشکنی پشتیبانی می‌کند. هر بار که عنصری حذف یا درج می‌شود، زیردنباله متوالی S' از S که $f(S')$ را بیشینه می‌کند، دوباره محاسبه می‌شود.

برهان. پیچیدگی‌های دسترسی، و نیز خاصیت انگشت پویا، به دنبال درخت گسترده این خاصیت را می‌دهد که اطلاعات تکمیل شده در هر گره را می‌توان از تعداد ثابتی مقایسه و ترکیب امتیاز از اطلاعات ذخیره شده مربوطه در گره‌های فرزندان آن محاسبه کرد. از آنجا که بعد از هر چرخش، اطلاعات تکمیل شده از گره‌هایی که تحت تأثیر قرار گرفته‌اند بروز می‌شوند، زیردنباله متوالی S' از S که در ریشه ذخیره شده است، و $f(S')$ را بیشینه می‌کند، دوباره محاسبه می‌شود. همان طور که در مورد درخت $MCS\ AVL$ دیدیم، به روز رسانی وزن یک عنصر را می‌توان به حذف و سپس درج کردن آن عنصر با وزن جدید کاهش داد، به طوری که از $O(\log n)$ مقایسه سرشکنی و ترکیب امتیاز پشتیبانی می‌کند که یک نتیجه ساده از پشتیبانی از حذف و درج است. بدیهی است که عمل به روز رسانی، می‌تواند در عمل بسیار ساده‌تر اجرا شود ولی لازمه این کار متوازن‌سازی مجدد درخت بر عنصری است که در دسترس قرار گرفته است.

□

[1] Amortized Comparisons

فصل ۳

روش‌های به دست آوردن مستطیل بهینه مسطح

در این فصل ما الگوریتم کورتس با همکارانش [۷] برای حل مسأله مستطیل بهینه مسطح را شرح خواهیم داد. سپس الگوریتم مرجع [۲] که در آن، درخت MCS با یک درخت $MCS\ Splay$ جایگزین و با استفاده از خواص درخت $MCS\ Splay$ و روش‌هایی از جمله بهره‌گیری از نوارهای تک‌رنگ، بهره‌گیری از تنظیمات نقطه و بهره‌گیری هم‌زمان از نوارهای تک‌رنگ و تنظیمات نقطه، الگوریتم موجود [۷] به یک الگوریتم تطبیقی تبدیل می‌شود را بیان خواهیم کرد.

یادآوری ۳.۰.۳. داده‌های مسأله مستطیل بهینه مسطح به صورت مجموعه P از n نقطه در صفحه، که در موقعیت کلی هستند و تابع f یک تابع امتیازدهی یکنوا (صعودی یا نزولی) و تجزیه‌پذیر می‌باشد. مسأله‌ی مستطیل بهینه مسطح عبارت است از پیدا کردن یک مستطیل H با ضلع‌های موازی محور مختصات که مقدار $f(H \cap P)$ را بیشینه کند.

۱.۳ به دست آوردن مستطیل بهینه مسطح با استفاده از درخت MCS

در این بخش الگوریتمی را برای به دست آوردن مستطیل بهینه مسطح با استفاده از ساختمان داده درخت MCS که یک ساختمان داده نیمه-پویا می‌باشد، بررسی خواهیم کرد.

ایده الگوریتم به این صورت است: مجموعه‌ای از تمام جفت خطوط افقی را در نظر بگیرید که هر خط بر روی یک نقطه از نقاط داده شده در مسأله واقع است، همان طور که در فصل اول اشاره شد، هر یک از جفت خطوط تشکیل یک نوار می‌دهد (ر.ک. ۵.۳.۱) که بین جفت این خطوط محصور شده است. حال هر نوار را به عنوان مختصات خطوط افقی مستطیل فرض کنید، پس با این ترتیب تعداد $O(n^2)$ نوار افقی خواهیم داشت. برای هر نوار، مکان دو خط عمودی مستطیلی که ضلع‌های افقی آن همان نوار می‌باشد را به گونه‌ای به دست آوریم که این مستطیل دارای بیش‌ترین امتیاز در بین مستطیل‌هایی که خطوط افقی مشترک با این نوار را دارند، باشد. در مرحله آخر بین امتیاز تمام این مستطیل‌ها بیش‌ترین را انتخاب کنیم، و به این ترتیب مستطیل با بیش‌ترین امتیاز را به دست آورده‌ایم.

ملاحظه ۱.۱.۳. در مسأله مستطیل بهینه مسطح اگر مستطیلی را در نظر بگیریم که در آن ضلعی وجود داشته باشد که بر روی آن هیچ نقطه‌ای قرار نگرفته باشد، می‌توانیم مساحت این مستطیل را به گونه‌ای کاهش دهیم که تمام اضلاع از نقطه‌ای گذشته باشند و امتیاز این مستطیل با امتیاز مستطیل اولیه برابر

باشد.

پس ما می‌توانیم فقط مستطیل‌هایی را در نظر بگیریم به طوری که حداقل یک نقطه روی هر ضلع از آن‌ها واقع شود. به این ترتیب مستطیل‌هایی را که در بالا در نظر گرفته‌ایم، می‌تواند تمام مستطیل‌هایی را که ممکن است مستطیل بهینه با بیش‌ترین امتیاز باشد پوشش می‌دهند.

الگوریتم ۲.۱.۳. برای به دست آوردن مستطیل مسطح f-بهینه با استفاده از الگوریتم کورتس و همکارانش [۷] می‌توان به صورت زیر عمل کرد:

ابتدا نقاط را بر حسب مختصات y آن‌ها در زمان $O(n \log n)$ مرتب می‌کنیم و سپس آن دنباله نقاط مرتب شده $p_1, ..., p_n$ را به شرح زیر پیمایش می‌کنیم:

برای هر نقطه p_i، $i \in [1..n]$، یک درخت MCS با مجموعه نقاط $p_i, ..., p_n$ می‌سازیم که کلید آن بر حسب مختصات x آن‌ها می‌باشد، و در ابتدا ارزش همه نقاط را برابر $f(\emptyset)$ قرار می‌دهیم. سپس برای نقاط p_j برای $j \in [i..n]$ به صورت متوالی وزن آن‌ها با مقدار $f(p_j)$ به روز رسانی می‌کنیم، وقتی که درخت MCS موجود را با این وزن‌ها به روز رسانی کنیم یک مستطیل بهینه بین مختصات y نقاط $p_i, ..., p_j$ می‌توانیم در دست داشته باشیم، زیرا در این صورت نواری را که به وسیله نقاط p_i و p_j تولید شده است، در نظر گرفته‌ایم و بر اساس لم ۱.۲.۲ هر زمانی که وزن عنصری تغییر پیدا کند زیردنباله متوالی که مقدار f را بیشینه می‌کند، دوباره محاسبه می‌شود. دقت کنید که با استفاده از درخت MCS موجود می‌توانیم مختصات ضلع‌های عمودی مستطیل را به دست آوریم (همان فاصله به دست آمده توسط $M(r)$ که r ریشه درخت MCS موجود است) و مختصات ضلع‌های افقی مستطیل نیز از مختصات y نقطه p_i تا مختصات y نقطه p_j است. در پایان از بین مستطیل‌های بهینه موجود برای هر نوار، مستطیلی را در نظر می‌گیریم که دارای بیش‌ترین امتیاز در بین این مستطیل‌ها باشد، و به این ترتیب مستطیل بهینه با بیش‌ترین امتیاز به دست می‌آید. الگوریتم را به صورت ۳.۱.۱ جمع‌بندی می‌کنیم.

الگوریتم بالا را در نظر بگیرید، در ابتدا نقاط را بر حسب مختصات y آن‌ها در زمان $O(n \log n)$ مرتب می‌کنیم سپس برای هر نقطه p_i ما یک درخت MCS با مجموعه نقاط $p_i, ..., p_n$ می‌سازیم. زمان ساخت این درخت برابر $O(n - i)$ است. پس از آن برای نقاط p_j برای $j \in [i..n]$ به صورت متوالی وزن آن‌ها با مقدار $f(p_j)$ به روز رسانی می‌کنیم. برای این درخت ما تعداد $n - i$ به روز رسانی انجام می‌دهیم که هر

Algorithm 3.1.1:

```
1  Sort set P by y-coordinate form p1، p2، ،... pn and m=-∞؛
2  for i := 1 to n do
3  |  Create MCS tree by using (pi to pn )'s x-coordinate and set score of all
   |  points f(∅)؛
4  |  for j := i to n do
5  |  |  Update weight pj in MCS tree by f(∅)؛
6  |  |  if m < f(M(r) then
7  |  |  |  m=f(M(r) and save the informations of Rectangle؛
8  |  |  end
9  |  end
10 end
```

یک از اعمال به روزرسانی طبق لم ۱.۲.۲ در زمان $O(\log(n-i))$ انجام می‌شود.

وقتی که الگوریتم کلیه نقاط را پیمایش می‌کند زمان اجرای آن برابر است با:

$$O(n\log n + \sum_{i=1}^{n}(n-i) + \sum_{i=1}^{n}(n-i)\log(n-i)) \subset O(n^2\log n).$$

۲.۳ بهره‌گیری از نوارهای تک‌رنگ

در این بخش می‌خواهیم بررسی کنیم که چگونه الگوریتم ۲.۱.۳ به یک الگوریتم تطبیقی به وسیله دسته‌بندی نقاط به نوارهای تک‌رنگ، تبدیل می‌شود [۲]. یادآور می‌شویم که نواری، نوار تک‌رنگ است که همه نقاط آن مثبت ($f(p) > f(\emptyset)$) یا همه نقاط آن منفی ($f(p) \leq f(\emptyset)$) باشد و نوار تک‌رنگی مثبت (منفی) است که نقاط آن مثبت (منفی) باشد.

ملاحظه ۱.۲.۳. هر یک از اضلاع یک مستطیل بهینه مسطح در هیچ حالتی بین یک نوار تک‌رنگ قرار نمی‌گیرد و همواره بر روی مرز یک نوار تک‌رنگ قرار دارد، زیرا اگر ضلع مستطیلی در بین یک نوار تک‌رنگ قرار گیرد می‌توان امتیاز آن مستطیل را به وسیله کوچک‌تر کردن (اگر نوار منفی باشد) یا بزرگ‌تر کردن (اگر نوار مثبت باشد) بهبود داد.

یک نمونه از نقاط مثبت و منفی را که به δ نوار مثبت و منفی به صورت مجموعه $n_1, ..., n_\delta$ در امتداد یک محور دسته‌بندی کرده‌ایم، در نظر بگیرید. با توجه به ملاحظه بالا، در چنین مواردی نیازی به در نظر

گرفتن مستطیل‌هایی که مرزهای آن در وسط این نوارها قرار می‌گیرد نیست: تمام مستطیل‌های بهینه در لبه نوارها شروع خواهد شد، به خصوص این که، بالای (و متناظراً پایین) مستطیل بهینه هم‌تراز با یک نقطه مثبت در بالای (و متناظراً پایین) نوار مثبت است.

این مشاهدات بسیار ساده، نه تنها تعداد مستطیل‌هایی را که باید امتیاز آن‌ها را محاسبه کنیم به $O(\delta^2)$ محدود می‌سازد، بلکه باعث می‌شود که محاسبه امتیاز هر مستطیل راحت‌تر شود: اضافه کردن نقاط n_i از iامین نوار، به صورت ترتیب افزایشی مختصات دوم آن‌ها در درخت *MCS Splay* که اندازه آن از n بیشتر نخواهد شد، به

$$O(n + \sum_{i=1}^{\delta} n_i \log(n/n_i))$$

مقایسه مختصات و ترکیب امتیاز سرشکن می‌شود.

هزینه $O(n + \sum_{j=1}^{n} \log(d_j + 1))$ به وسیله فرمول ذیل به کران بالا محدود می‌شود:

$$O\left(n + \sum_{j=1}^{n} \log(d_j + 1)\right) = O\left(n + \sum_{i=1}^{\delta} \left(\sum_{j=1}^{n_i} \log(d_{ij} + 1)\right)\right) \quad (۱-۳)$$

$$\subseteq O\left(n + \sum_{i=1}^{\delta} \left(\log\left(\sum_{j=1}^{n_i} (d_{ij} + 1)\right)\right)\right) \quad (۲-۳)$$

$$\subseteq O\left(n + \sum_{i=1}^{\delta} (\log(n + n_i))\right) \quad (۳-۳)$$

$$= O\left(n + \sum_{i=1}^{\delta} (\log(n_i(1 + n/n_i)))\right) \quad (۴-۳)$$

$$\subset O\left(n + \sum_{i=1}^{\delta} (n_i \log(1 + n/n_i))\right) \quad (۵-۳)$$

$$\subset O\left(n + \sum_{i=1}^{\delta} n_i + n_i \log(n/n_i)\right) \quad (۶-۳)$$

$$= O\left(n + n + \sum_{i=1}^{\delta} n_i \log(n/n_i)\right) \quad (۷-۳)$$

$$= O\left(n\left(1 + H(n_1, ..., n_\delta)\right)\right) \quad (۸-۳)$$

$$\subset O\left(n \log(\delta + 1)\right), \quad (۹-۳)$$

که در آن $H(n_1, ..., n_\delta) = \sum_{i=1}^{\delta} (n_i/n) \log(n/n_i)$ تابع آنتروپی[1] است.

[1] entropy function

در روابط بالا، برای تبدیل رابطه ۳-۱ به رابطه ۳-۲ از تحدب تابع لگاریتم و برای رابطه ۳-۲ از سری تلسکوپی به صورت زیر استفاده شده است:

$$\sum_{j=۱}^{n_i}(d_{ij}+۱) =(m_{i۱}-m_{i۰}+۱)+(m_{i۲}-m_{i۱}+۱)+(m_{i۳}-m_{i۲}+۱)+\ldots$$

$$+(m_{in_i}-m_{in_i-۱})=m_{in_i}-m_{i۰}+n_i\le n+n_i,$$

که m_{ij} نشان دهنده رتبه دسترسی jام در نوار iام است. از جایگذاری رابطه

$$\sum_{j=۱}^{n_i}(d_{ij}+۱)\le n+n_i$$

در رابطه ۳-۲ به رابطه ۳-۳ می‌رسیم. برای تبدیل رابطه ۳-۴ به رابطه ۳-۵ به طور مشابه از خاصیت تحدب تابع لگاریتم استفاده می‌کنیم و برای تبدیل رابطه ۳-۵ به رابطه ۳-۶ نیز از اینکه $\log(x+۱)\le \log(x)+۱$ برای هر $x\in(۱/(e-۱),\infty)$، و همچنین $n/n_i\ge ۱$ برقرار است، کمک می‌گیریم.

تلفیق این موضوع با این واقعیت که نقطه بالایی یک مستطیل بهینه با نقطه مثبتی در بالای یک نوار مثبت تراز می‌شود منجر به نتیجه ذیل می‌شود.

قضیه ۳.۲.۲. برای هر $\delta\in[۱..n]$ و $n_۱,\ldots,n_\delta$ که مجموع آن‌ها n است، در بدترین حالت نسبت به نمونه‌های متشکل از δ نوار از علائم متناوب روی یک محور، به طوری که i امین نوار حاوی n_i نقطه است، الگوریتمی وجود دارد که مستطیل f-بهینه مسطح را در

$$O(\delta n(۱+H(n_۱,\ldots,n_\delta)))\subset O(\delta n\log(\delta+۱))$$

ترکیب امتیاز و

$$O(\delta n(۱+H(n_۱,\ldots,n_\delta))+n\log n)\subset O(\delta n\log(\delta+۱)+n\log n)$$

مقایسه مختصات می‌یابد.

توجه کنید که علت اضافه شدن $O(n \log n)$ مقایسه مختصاتی در قضیه بالا، هزینه‌ایی است که الگوریتم ۲.۱.۳ در ابتدا برای مرتب کردن نقاط بر حسب مختصات y مصرف می‌کند.

۳.۳ بهره‌مندی از ترازهای دلخواه برای نقطه

اجرای الگوریتم مشخص شده در بخش قبل درخصوص استفاده از درخت *MCS Splay* به جای یک درخت *MCS* پیامدهای بیشتری دارد. ما در این بخش بررسی می‌کنیم که چگونه درخت *MCS Splay*، الگوریتم را با هم‌ترازسازی نقاط محلی منطبق می‌نماید.

تصور کنید که، وقتی نقاط $p_1, p_2, ..., p_n$ توسط مختصات y مرتب شده‌اند، توسط مختصات x نیز مرتب شوند. سپس، هنگامی که نقطه‌ها را در درخت *MCS Splay* درج می‌کنیم، فاصله از هر مورد درج شده به وضعیت یکی قبل برابر با ۱ است، و در نتیجه هزینه کلی الگوریتم، $O(n^2)$ ترکیب امتیاز است (به شرح الگوریتم ۲.۱.۳ رجوع شود).

بنا به فرض مسأله، هیچ جفت نقطه‌ای بر روی یک محور مختصات قرار ندارد (نقاط در موقعیت کلی هستند)، بنابرین ترتیب بر حسب مختصات y جایگشتی از ترتیب بر حسب مختصات x است. Π را جایگشتی در نظر بگیرید که نقاط $p_1, p_2, ..., p_n$ را از یک ترتیب مرتب شده توسط مختصات x به یک وضعیت مرتب شده توسط مختصات y، بازآرایی می‌کند. توجه داشته باشید که جایگشت Π بر روی نقاط $p_1, p_2, ..., p_n$ که بر حسب مختصات x مرتب شده‌اند به صورت زیر عمل می‌کند (تعریف جایگشت برای یک دنباله):

$$\Pi.(p_1, p_2, ..., p_n) = (p_{\Pi^{-1}(1)}, p_{\Pi^{-1}(2)}, ..., p_{\Pi^{-1}(n)}),$$

که در آن نقاط $p_{\Pi^{-1}(1)}, p_{\Pi^{-1}(2)}, ..., p_{\Pi^{-1}(n)}$ بر حسب مختصات y مرتب شده‌اند.

فرض کنید در الگوریتم ۲.۱.۳ می‌خواهیم نقاط $p_1, p_2, ..., p_n$ را که بر حسب مختصات y مرتب شده‌اند در درخت *MCS Splay* با شروع از نقطه p_1 درج کنیم. برای هر j در نظر بگیرید که مختصات x_j از نقطه p_j، در موقعیت r_j در دنباله $(x_1, x_2, ..., x_{j-1})$، مرتب شده براساس مختصات پیشین (x)، قرار می‌گیرد، در این صورت با توجه به d_j تعریف شده در لم ۲.۳.۲، مربوط به خاصیت انگشت پویا،

$d_1 = 0$ و برای $j > 1$، $d_j = |r_j - r_{j-1}|$ خواهد بود. دقت کنید که r_j همان رتبه دسترسی jام است. حال $\lambda = \sum_{j=2}^{n} |r_j - r_{j-1}|$ را به عنوان «پیچیدگی درج محلی»[1] توالی تعریف نماییم [۱۴]. توجه داشته باشید که کران بالای ساده $\lambda \leq \sum_{j=2}^{n} |\Pi_j - \Pi_{j-1}|$ است که Π به موقعیت قطعی عناصر اشاره دارد، در حالی که λ به حرکات مورد نیاز در پیشوند جایگشت اشاره می‌کند. دلیل آن این است که برای هر $j \in [2..n]$ داریم:

$$|r_j - r_{j-1}| \leq |\Pi_j - \Pi_{j-1}|$$

که این رابطه نشان دهنده این است که فاصله رتبه‌های دو دسترسی در درج محلی حداکثر به میزان فاصله رتبه‌های آن‌ها در موقعیت قطعی آن‌ها می‌باشد.

هزینه کل به واسطه تحدب لگاریتم و به دلیل این که $\sum_{j=1}^{n}(d_j + 1) \leq \lambda + n$ است، با تعداد ترکیب امتیاز زیر برابر می‌شود:

$$\begin{aligned} O\left(n + \sum_{j=1}^{n} \log(d_j + 1)\right) &\subseteq O\left(n + \log\left(\sum_{j=1}^{n}(d_j + 1)\right)\right) \\ &\subseteq O\left(n + \log(\lambda + n)\right) \\ &= O\left(n + \log(n(1 + \lambda/n))\right) \\ &\subseteq O\left(n + n\log(1 + \lambda/n)\right). \end{aligned}$$

کران بالای ساده برای پیمایش کلیه nها از الگوریتم ۲.۱.۳ به شرح ذیل به دست می‌آید.

لم ۱.۳.۳. اگر P مجموعه‌ای از n نقطه در صفحه و f تابع امتیازدهی تجزیه‌پذیر و یکنوا باشد که به عنوان آرگومان ورودی هر زیرمجموعه‌ای از P را دریافت کند، در اینجا الگوریتمی وجود خواهد داشت که یک مستطیل f-بهینه را در

$$O(n^2(1 + \log(1 + \lambda/n)))$$

ترکیب امتیاز و

$$O(n^2(1 + \log(1 + \lambda/n)) + n\log n)$$

[1] Local insertion complexity

مقایسه مختصات می‌یابد که در آن $\lambda \leq n^۲$ پیچیدگی استقرار محلی توالی مختصات x مربوط به نقاط مرتب شده بر اساس مختصات y است.

بدترین حالت الگوریتم مربوط به $\lambda = n^۲$ است که هزینه آن منطبق با $O(n^۲ \log n)$ می‌باشد. در حالی که بهترین حالت الگوریتم مربوط به $\lambda = ۰$ و هزینه با $O(n^۲)$ منطبق است.

می‌توانیم با استفاده از دیگر اقدامات از بی‌نظمی در جایگشت‌ها λ را محدود به کران بالا نماییم. به عنوان مثال، اجازه دهید Inv[۱] تعداد وارونگی‌های صورت پذیرفته در جایگشت Π، یا به بیان دیگر، تعداد جفت‌های نامناسب در توالی در نظر بگیریم [۱۵]. زوج مرتب (i, j) را یک وارونگی برای جایگشت Π گوییم اگر $i < j$ و $\Pi(i) > \Pi(j)$ باشد. به مجموع همه وارونگی‌ها در یک جایگشت، تعداد وارونگی‌ها در جایگشت گوییم. جایگزین کردن λ با Inv در لم قبل باعث ایجاد یک کران بالای معتبر براساس پیچیدگی O- بزرگ می شود.

لم ۲.۳.۳. برای هر جایگشت Π از λ مربوط به پیچیدگی استقرار محلی و Inv نشان‌دهنده وارونگی‌ها، خواهیم داشت:

$$Inv \geq \lambda/۲ - n.$$

برهان. برای هر x_j، λ به میزان $|\Pi_j - \Pi_{j-۱}|$ افزایش می‌یابد، در حالی که Inv به میزان $j - \Pi_j$ افزایش می‌یابد. از آنجایی که $\Pi_j \leq j$ است، اگر $\Pi_j > \Pi_{j-۱}$ بود می‌توانیم از $\Pi_j - \Pi_{j-۱} + ۱ \geq \Pi_j - \Pi_{j-۱}$ $(j-۱)$ یا در غیر این صورت از $j - \Pi_j \geq \Pi_{j-۱} - \Pi_j$ استفاده نماییم. بنابراین استفاده از $x_{j-۱}$ به علاوه ۱ و یا x_j در Inv، λ را به وسیله x_j به کران بالا محدود می‌کند. با جمع‌بندی هر دو (یعنی $x_{j-۱}$ به علاوه ۱ و x_j) برای هر j، خواهیم داشت:

$$\begin{aligned}\lambda \leq \sum_{j=۲}^{n} |\Pi_j - \Pi_{j-۱}| &= \sum_{j=۲}^{n} |(j-۱) - \Pi_{j-۱} + ۱ - (j - \Pi_j)| \\ &\leq \sum_{j=۲}^{n} |(j-۱) - \Pi_{j-۱} + ۱| + \sum_{j=۲}^{n} |j - \Pi_j| \\ &\leq ۲.(Inv + n)\end{aligned}$$

بنابراین حکم ثابت می‌شود. □

[۱] Inversions

یک معیار مشهور دیگر برای پیچیدگی جایگشت، تعداد ردیف‌های صعودی [۱] ρ یا همان حداقل تعداد زیردنباله‌های یکنوای صعودی است که Π را پوشش می‌دهند [۱۲]. فرض کنید $r_1, ..., r_\rho$ طول این ردیف‌ها باشند که با $O(n)$ مقایسه محاسبه شده‌اند. در این صورت مجموع مقادیر $|\Pi_j - \Pi_{j-1}|$ در iامین ردیف و نیز مجموع مقادیر d_j حداکثر به n می‌رسد. بنابراین با توجه به تحدب تابع لگاریتم داریم که:

$$\sum_{j=1}^{n} \log(d_j + 1) \leq \sum_{i=1}^{\rho} r_i \log(1 + n/r_i) \leq n + \sum_{i=1}^{\rho} r_i \log(n/r_i).$$

این امر کران بالای جایگزین زیر را نتیجه می‌دهد.

لم ۳.۳.۳. فرض کنید P مجموعه‌ای از n نقطه در صفحه و همچنین f یک تابع یکنوا و تجزیه‌پذیر باشد که هر زیرمجموعه از P را به عنوان آرگومان می‌پذیرد. الگوریتمی وجود دارد که یک مستطیل f-بهینه را با $O(n \log n)$ مقایسه‌ی مختصات و $O(n^2(1 + H(r_1, ..., r_\rho)) \subset O(n^2 \log(\rho + 1))$ ترکیب امتیاز به دست می‌آورد. که در آن $r_1, ..., r_\rho$ طول‌های ρ زیردنباله یکنوای صعودی هستند که جایگشت دنباله‌ای از نقاط مرتب شده را بر اساس مختصات x، به نقاط مرتب‌شده بر اساس مختصات y هماهنگ می‌کند، پوشش می‌دهد.

۴.۳ بهره‌گیری هم‌زمان از نوارها و تنظیمات نقطه

در بخش ۲.۳ بررسی شد که چگونه از نوارهای تک‌رنگ برای تبدیل الگوریتم ۲.۱.۳ به یک الگوریتم تطبیقی استفاده شده است، بدین صورت که تعداد مستطیل‌هایی که می‌بایست امتیاز آن‌ها را محاسبه کنیم به $O(\delta^2)$ کاهش داده شد و دیدیم که محاسبه امتیاز هر مستطیل نیز راحت‌تر می‌شود. در بخش ۳.۳ نشان داده شد که استفاده از درخت *MCS Splay* به جای درخت *MCS* چگونه الگوریتم ۲.۱.۳ را به واسطه هر کدام از عامل‌های هم‌ترازی نقاط محلی، تعداد وارونگی‌ها و حداقل تعداد زیردنباله‌های یکنوای صعودی منطبق می‌سازد. ترکیب روش‌های بخش‌های ۲.۳ و ۳.۳ را می‌توان به خوبی تحلیل نمود. یک نتیجه‌ی ساده این خواهد بود که تنها لازم است از هر نوار از δ نوار، از مقدار p_i متفاوت شروع کنیم، و

[۱] Increasing runs

بنابراین کران بالای پیچیدگی زمانی الگوریتم ۲.۱.۳:

$$O(n\delta((\text{۱} + \log(\text{۱} + \lambda/n)))$$

خواهد بود، که در آن λ همان پیچیدگی درج محلی است. در واقع با مرتب کردن صعودی نقاط برحسب مختصات x در هر نوار تک‌رنگ می‌توانیم عملکرد خود را اندکی بهبود دهیم، به این صورت که در هر نوار تک‌رنگ به جای این که نقاط به ترتیب افزایشی مختصات y آن‌ها در درخت به روز رسانی کنیم، به ترتیب مختصات x آن‌ها در درخت موجود به روز رسانی می‌کنیم [۲]. در حالی که اندازه‌ی λ' ناشی از مرتب‌سازی مجدد ممکن است بزرگ‌تر از λ باشد (یعنی λ پیچیدگی درج محلی برای وقتی که نقاط را بر حسب ترتیب افزایشی y آن‌ها در درخت اضافه کنیم باشد و λ' پیچیدگی درج محلی حاصل از به روز رسانی نقاط در درخت موجود به ترتیب افزایشی مختصات x آن‌ها در هر نوار تک‌رنگ باشد)، کران‌های بالای مربوط به Inv و ρ، یعنی Inv' و ρ'، و همچنین $H(n'_{\text{۱}}, ..., n'_{\rho'})$ ناشی از δهای متفاوت افزایش نمی‌یابند. به ویژه به سادگی می‌توان دید که با توجه به این که $\rho' \leq \delta$ و $H(r'_{\text{۱}}, ..., r'_{\rho'}) \leq H(n_{\text{۱}}, ..., n_{\delta})$ کران بالای قضیه ۲.۲.۳ از طریق ترکیب کاهش می‌یابد (زیرا وقتی که یک نوار تک‌رنگ مرتب شده باشد، در هیچ ردیفی برش نمی‌خورد).

قضیه ۱.۴.۳. فرض کنید P مجموعه‌ای از n نقطه در صفحه و f تابعی یکنوا و تجزیه‌پذیر باشد که آرگومان آن زیرمجموعه‌ای از P باشد. الگوریتمی وجود دارد که مستطیل f-بهینه را با استفاده از $O(n \log n)$ مقایسه‌ی مختصات و $O(n\delta(\text{۱} + min(\log(\text{۱} + Inv/n), H(r_{\text{۱}}, ..., r_{\rho})))) \subset O(n\delta \log(\rho + \text{۱}))$ ترکیب امتیاز می‌یابد. در این روابط δ حداقل تعداد نوارهای تک‌رنگ است که نقاط مرتب شده برحسب مختصات y را می‌توان به آن‌ها افراز نمود؛ X دنباله مختصات x نقاط پس از مرتب‌سازی (مجدد) نقاط درون هر نوار تک‌رنگ برحسب مختصات x است؛ $Inv \leq n^{\text{۲}}$ تعداد جفت‌های نامرتب در X است و $r_{\text{۱}}, ..., r_{\rho}$ طول‌های مربوط به حداقل تعداد $\rho \leq \delta$ ردیف یکنوا و صعودی است، که X را پوشش می‌دهد. با جابه‌جا کردن محورهای x و y نتیجه‌ی مشابه‌ایی برقرار خواهد بود.

فصل ۴

بهره‌گیری از قطرهای بلوک‌ها برای حل مسأله

در این فصل برای به دست آوردن یک الگوریتم تطبیقی (وفقی) متفاوت، از موقعیت نسبی نقاط استفاده می‌کنیم [۲] . افراز مجموعه نقاط به دو زیرمجموعه را در نظر می‌گیریم. این افرازها با استفاده از دو خط موازی با محورهای مختصات و عمود بر یکدیگر ایجاد شده‌اند. ترکیب مستطیل‌های بهینه‌ی هر زیرمجموعه منجر به دست‌یابی به مستطیل بهینه‌ی کل مجموعه نقاط می‌گردد.

۱.۴ قطری‌سازی

لازم به یادآوری است که داده‌های مسأله مستطیل بهینه مسطح به صورت مجموعه P از n نقطه در صفحه، که در موقعیت کلی هستند و تابع f یک تابع امتیازدهی یکنوا (صعودی یا نزولی) و تجزیه‌پذیر می‌باشد، مسأله مستطیل بهینه مسطح عبارت است از پیدا کردن یک مستطیل H با ضلع‌های موازی محور مختصات که مقدار $f(H \cap P)$ را بیشینه کند.

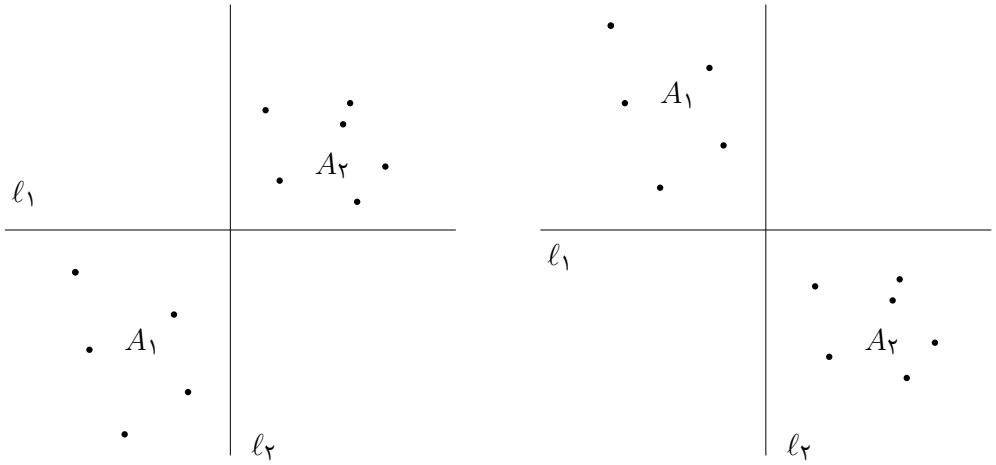
تعریف ۱.۱.۴. برای هر زیرمجموعه‌ی $A \subseteq P$، یک قطری‌سازی[۱] A افرازی از A به صورت $\{A_1, A_2\}$ با استفاده از دو خط ℓ_1 و ℓ_2، به ترتیب به موازات محورهای x و y است؛ به گونه‌ای که عناصر A_1 و A_2 متعلق به ربع‌های مقابل هم (نسبت به نقطه‌ی $\ell_2 \cap \ell_1$) باشند.

شکل ۱.۴ مثال‌هایی از قطری‌سازی را نشان می‌دهد. به ویژه، با فرض این‌که نقاط A_1 سمت چپ نقاط A_2 هستند، اگر عناصر A_1 زیر عناصر A_2 باشند قطری‌سازی را پایین به بالا شکل ۱.۴ بخش (الف) و در غیر این صورت بالا به پایین شکل ۱.۴ بخش (ب) می‌نامیم. توجه کنید اگر عناصر A که برحسب مختصات x به صورت $p_1, p_2, ..., p_m$ مرتب شده باشند، آنگاه هر قطری‌سازی A به صورت $\{\{p_1, ..., p_k\}, \{p_{k+1}, ..., p_m\}\}$ که در آن $k \in [1...m-1]$ خواهد بود.

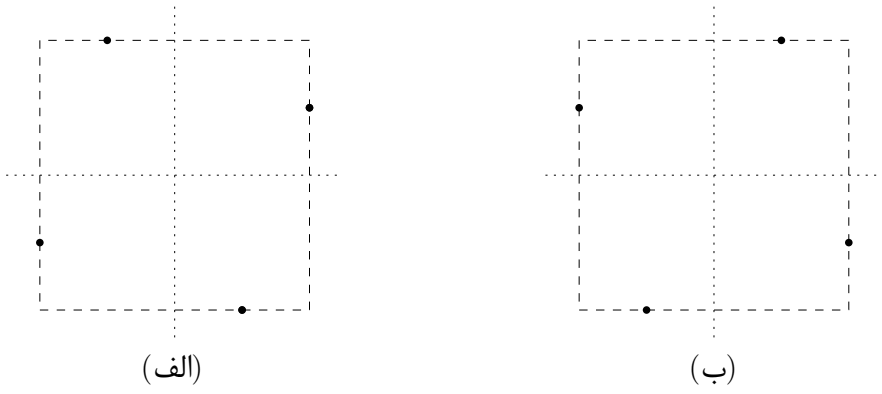
با داشتن هر مجموعه‌ی کردان‌دار $S \subset \mathbb{R}^2$، فرض کنید $Box(S)$ نشان‌دهنده‌ی کوچک‌ترین مستطیل شامل نقاط S و نقاط کرانی[۲] A، نقاط متعلق به مرز $Box(A)$ باشند.

هر مجموعه از نقاط را نمی‌توان به صورت قطری تجزیه کرد، ساده‌ترین حالت مجموعه‌ای از چهار نقطه واقع در رئوس یک مربع است که اضلاع آن اندکی نسبت به محورهای x و y دوران نموده‌اند. چنین

[۱] Diagonalization
[۲] Extreme



شکل ۱.۴: قطری‌سازی $\{A_1, A_2\}$ برای مجموعه نقاط $A = A_1 \cup A_2$.



شکل ۲.۴: آسیاب‌های بادی.

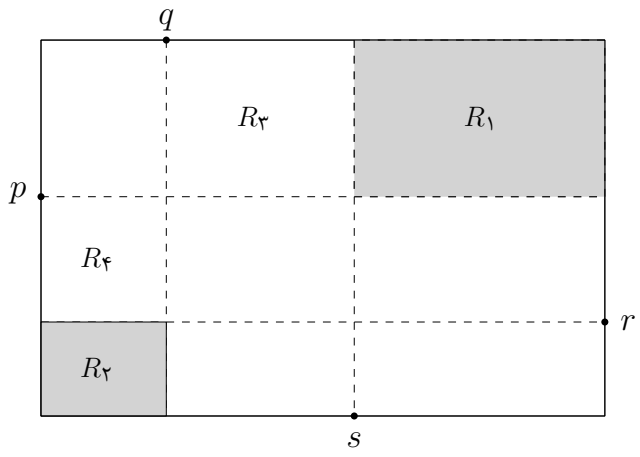
مجموعه‌هایی از چهار نقطه را که به دلیل موقعیت شاخص نقاط کرانی آن‌ها، نمی‌توان به صورت قطری تجزیه نمود، آسیاب بادی[1] می‌نامیم. برای مثال شکل ۲.۴ را ببینید.

لم ۲.۱.۴. فرض کنید A مجموعه‌ای از نقاط باشد که قطری‌سازی آن ممکن نیست. در این صورت A دقیقاً چهار نقطه‌ی کرانی خواهد داشت. به علاوه، A دارای یک آسیاب بادی شامل حداقل یک نقطه‌ی کرانی A است.

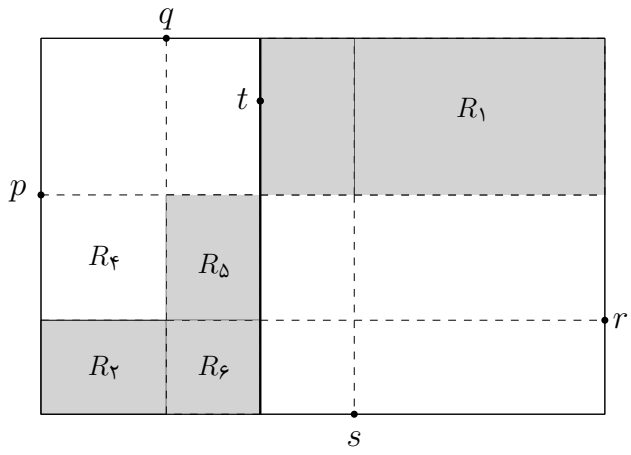
برهان. نشان دادن بخش اول حکم ساده است، فرض کنید که نقاط کرانی مجموعه A کم‌تر از چهار نقطه باشد، در این صورت می‌توانیم یکی از نقاط کرانی را در نظر بگیریم و موقعیت خطوط ℓ_1 و ℓ_2 را به گونه‌ای در نزدیکی این نقطه در نظر بگیریم که بقیه نقاط موجود در ربع مقابل نسبت به نقطه $\ell_1 \cap \ell_2$ قرار گیرند، و

[1] Windmill

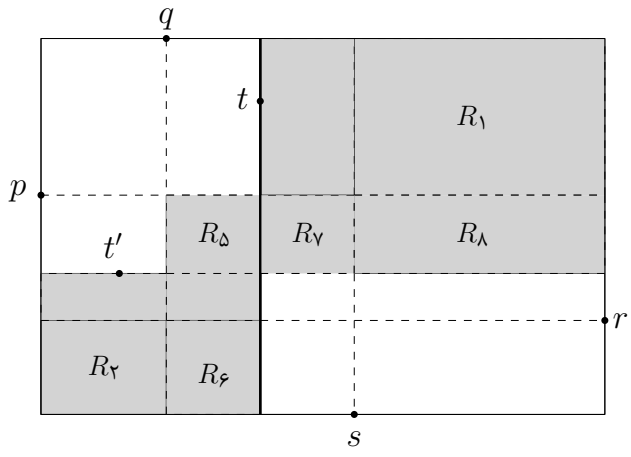
به این صورت مجموعه A را به صورت قطری تجزیه نمودیم که این خلاف فرض است. پس به این ترتیب نقاط کرانی A نمی‌تواند کم‌تر از چهار نقطه باشد و از طرفی هم این نقاط کرانی نمی‌تواند بیش‌تر از چهار نقطه باشد (دقت کنید که نقاط در موقعیت کلی هستند). بخش دوم حکم را اثبات می‌کنیم. فرض کنید $\{p, q, r, s\}$ نقاط کرانی A باشند. اگر این نقاط یک آسیاب بادی تشکیل دهند، حکم ثابت شده است. در غیر این صورت، بدون از دست رفتن کلیت مسأله، فرض کنید که موقعیت نسبی آن‌ها مطابق شکل ۳.۴ بخش (الف) است، زیرا می‌توان این چهار نقطه را به صورت قطری تجزیه کرد به این ترتیب که مطابق شکل از نقطه q یک خط عمودی به سمت پایین، از نقطه s یک خط عمودی به سمت بالا، از نقطه p یک خط افقی به سمت چپ و از نقطه r یک خط افقی به سمت راست رسم می‌کنیم، سپس خط افقی ℓ_1 را بین خطوط افقی رسم شده از نقطه p و r، و به همین صورت خط عمودی ℓ_2 را بین خطوط عمودی رسم شده از نقطه q و s رسم می‌کنیم. ناحیه‌های R_1، R_2، R_3 و R_4 را مطابق شکل ۳.۴ بخش (الف) نام‌گذاری می‌کنیم. اگر عنصر q' از مجموعه‌ی A در ناحیه‌ی R_1 باشد آن‌گاه $\{p, q', r, s\}$ یک آسیاب بادی تشکیل داده و کار اثبات تمام است. به طور مشابه، اگر نقطه‌ی s' از مجموعه‌ی A در ناحیه‌ی R_2 باشد آن‌گاه $\{p, q, r, s'\}$ یک آسیاب بادی تشکیل می‌دهد. حال فرض کنید که $R_1 \cup R_2$ شامل هیچ عنصری از A نباشد. بدون از دست رفتن کلیت فرض کنید که R_3 شامل یک نقطه باشد. فرض کنید t سمت راست‌ترین نقطه‌ی R_3 بوده و نواحی R_5 و R_6 را مطابق شکل ۳.۴ بخش (ب) ایجاد نماید. اگر یک نقطه‌ی u در ناحیه‌ی R_5 وجود داشته باشد آن‌گاه مجموعه‌ی $\{p, q, t, u\}$ یک آسیاب بادی تشکیل می‌دهد. به طور مشابه، اگر یک نقطه‌ی u در ناحیه‌ی R_6 وجود داشته باشد آن‌گاه مجموعه‌ی $\{p, t, r, u\}$ یک آسیاب بادی را تشکیل داده و حکم اثبات می‌شود. حال فرض کنید که $R_5 \cup R_6$ شامل هیچ عنصری از A نباشد. از آنجا که نمی‌توان A را قطری نمود، ناحیه‌ی R_4 بایستی شامل یک نقطه از A باشد. پس فرض کنید که t' پایین‌ترین نقطه در ناحیه‌ی R_4 بوده و مطابق شکل ۳.۴ بخش (ج) نواحی R_7 و R_8 را ایجاد نماید. اکنون ما با نقطه‌ی t' همان‌گونه برخورد می‌کنیم که با نقطه‌ی t برخورد نموده بودیم. اگر ناحیه‌ی R_7 شامل یک نقطه‌ی u از مجموعه‌ی A باشد آنگاه $\{p, t, u, t'\}$ یک آسیاب بادی است. در غیر این صورت ناحیه‌ی R_8 بایستی شامل یک نقطه‌ی u' از A باشد (زیرا در غیر این صورت A یک قطری‌سازی خواهد داشت) و آنگاه $\{t, u', s, t'\}$ یک آسیاب بادی است. بنابرین حکم اثبات می‌شود. □



(الف)

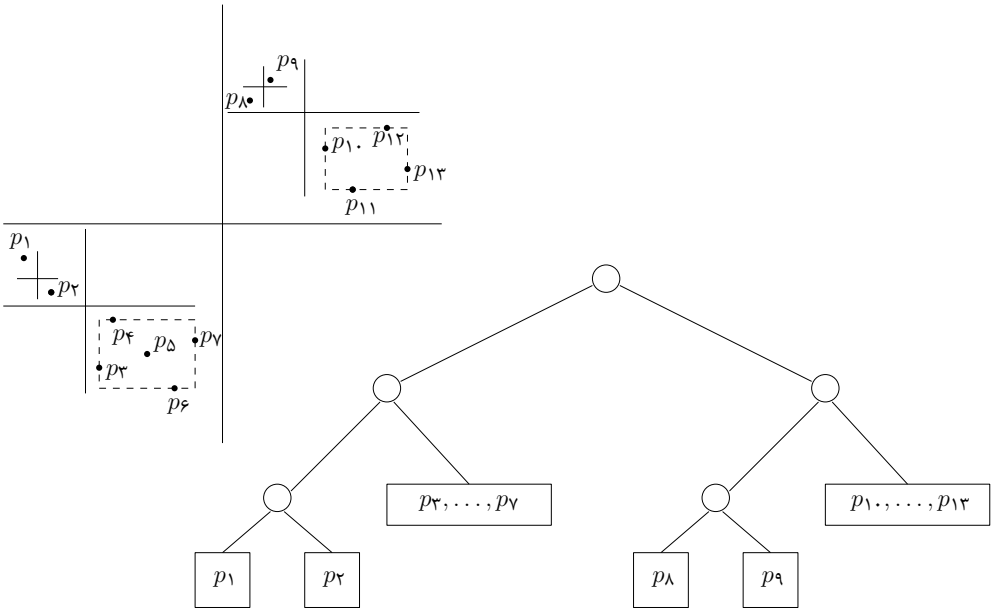


(ب)



(ج)

شکل ۳.۴: در هر شکل ناحیه سایه‌دار خالی از نقاط مجموعه‌ی A است.



شکل ۴.۴: یک درخت قطری‌سازی (D-درخت) برای مجموعه نقاط $\{p_۱, ..., p_{۱۳}\}$.

۲.۴ D- درخت

در این فصل ساختمان داده D-درخت و همچنین نحوه ساخت آن را با استفاده از قطری‌سازی شرح خواهیم داد.

تعریف ۱.۲.۴. یک درخت قطری‌سازی مجموعه‌ی P، D-درخت، یک درخت دودویی است که:

۱. هر برگ u آن شامل یک زیرمجموعه‌ی $S(u) \subseteq P$ است که امکان قطری‌سازی ندارد؛

۲. مجموعه‌ی تمام $S(u)$ها که u یک برگ است، یک افراز از P است؛ و

۳. هر گره داخلی v دقیقا دو فرزند $v_۱$ (فرزند چپ) و $v_۲$ (فرزند راست) دارد و $\{A(v_۱), A(v_۲)\}$ یک قطری‌سازی از $A(v)$ است. به طوری که برای هر گره v، $A(v)$ نشان دهنده‌ی اجتماع مجموعه‌های $S(u)$ برای همه‌ی برگ‌های u، که نوادگان گره v هستند، می‌باشد (شکل ۴.۴ را ببینید).

نکته ۲.۲.۴. درخت قطری‌سازی برای یک مجموعه مانند P متشکل از n نقطه لزوماً منحصر به فرد نیست.

لم ۳.۲.۴. فرض کنید P مجموعه‌ای از n نقطه در صفحه باشد. تعداد برگ‌های همه‌ی درختان قطری‌سازی مجموعه‌ی P، برابر خواهد بود. به علاوه، iامین برگ (از چپ به راست) هر دو D-درخت دلخواه مجموعه‌ی P، شامل زیرمجموعه‌ی $S(.)$ یکسانی از P خواهد بود.

برهان. از استقرا بر روی تعداد اعضای P استفاده می‌کنیم. اگر P تنها شامل یک عنصر باشد، آنگاه حکم اثبات شده است. زیرا D-درخت آن تنها یک گره‌ای می‌باشد. توجه داشته باشید اگر P امکان قطری‌سازی نداشته باشد، D-درخت آن نیز تک گره‌ای می‌باشد. فرض کنید که حکم برای همه مجموعه‌هایی که تعداد اعضای آن‌ها کم‌تر از P است، برقرار باشد. حال برای مجموعه P حکم را ثابت می‌کنیم، فرض کنید که P دقیقاً $k \geq ۱$ قطری‌سازی متفاوت داشته باشد که همگی باید از یک نوع (پایین به بالا یا بالا به پایین) باشند، زیرا یک مجموعه را نمی‌توان به دو نوع مختلف (پایین به بالا و بالا به پایین) قطری‌سازی کرد. بدون از دست دادن کلیت مسأله فرض کنید که همه این k قطری‌سازی‌ها بالا به پایین باشد. آنگاه P از طریق این k قطری‌سازی متفاوت به $k + ۱$ مجموعه‌ی ناتهی افراز شده است که به ترتیب از چپ به راست با $P_۱, P_۲, ..., P_{k+۱}$ نشان داده شده‌اند و k قطری‌سازی به صورت $\{\bigcup_{i=۱}^{j} P_i, \bigcup_{i=j+۱}^{k+۱} P_i\}$ برای $j = ۱, ..., k$ هستند. توجه کنید که این مجموعه‌ها را به این صورت می‌توان به دست آورد: فرض کنید که این k قطری‌سازی را از بالا به پایین از ۱ تا k شماره‌گذاری کرده‌ایم، مجموعه $P_۱$ را همان مجموعه بالای قطری‌سازی اول در نظر می‌گیریم همچنین برای $i \in [۲..k]$ هر P_i را مجموعه نقاط محصور شده بین خطوط قطری‌سازی $(i - ۱)$ام و خطوط قطری‌سازی iام در نظر می‌گیریم و در آخر $P_{k+۱}$ را مجموعه نقاط پایین قطری‌سازی kام در نظر می‌گیریم. هیچ مجموعه‌ی P_i امکان قطری‌سازی به صورت بالا به پایین را ندارد زیرا فرض کرده‌ایم که مجموعه P دقیقاً k قطری‌سازی داشته باشد و اگر یک مجموعه P_i باشد که امکان قطری‌سازی بالا به پایین را داشته باشد، آنگاه مجموعه P می‌تواند $k + ۱$ قطری‌سازی بالا به پایین داشته باشد. هر D-درخت مجموعه‌ی P می‌تواند با ایجاد یک درخت دودویی که گره‌های داخلی آن دقیقاً دو فرزند دارند و برگ‌های آن از چپ به راست به ترتیب $P_۱, P_۲, ..., P_{k+۱}$ هستند و سپس جایگزین کردن هر برگ P_i با یک D-درخت از P_i نتیجه شده از اعمال فرض استقرا به مجموعه‌های $P_۱, P_۲, ..., P_{k+۱}$ به دست آید. و به این ترتیب نتیجه مورد نظر حاصل می‌شود. □

با توجه به لم ۳.۲.۴ می‌توان نتیجه گرفت که هر درخت قطری‌سازی T مجموعه‌ی P، این مجموعه را به شکل یکسانی افراز می‌نماید، اگر $u_۱, ..., u_\beta$ گره‌های برگ T باشند، این افراز برابر با $\{S(u_۱), ..., S(u_\beta)\}$ است.

لم ۴.۲.۴. فرض کنید P مجموعه‌ای شامل n نقطه در صفحه باشد. یک D-درخت مجموعه‌ی P به

فضای $O(n)$ نیاز داشته و می‌تواند با $O(n \log n)$ مقایسه ساخته شود.

برهان. فرض کنید $p_1, p_2, ..., p_n$ عناصر P مرتب شده بر اساس مختصات x و $p_{\Pi_1}, p_{\Pi_2}, ..., p_{\Pi_n}$ عناصر P مرتب شده بر اساس مختصات y باشند. هر دو مرتب‌سازی می‌تواند با $O(n \log n)$ مقایسه‌ی مختصات انجام شود. با در نظرگرفتن محاسبه‌ی جایگشت Π به عنوان پیش‌پردازش، نشان می‌دهیم که: اگر P امکان قطری‌سازی به صورت $\{\{p_1, ..., p_k\}, \{p_{k+1}, ..., p_n\}\}$ را داشته باشد آنگاه این قطری‌سازی می‌تواند با $O(\min(k, n-k))$ مقایسه انجام شود. در غیر این صورت، اگر P امکان قطری‌سازی نداشته باشد این امر با $O(n)$ مقایسه قابل تشخیص است.

برای هر نمایه $i \in [1..n]$، فرض کنید:

$$M_L(i) = \max_{j \in [1..i]} \Pi_j$$

$$m_L(i) = \min_{j \in [1..i]} \Pi_j$$

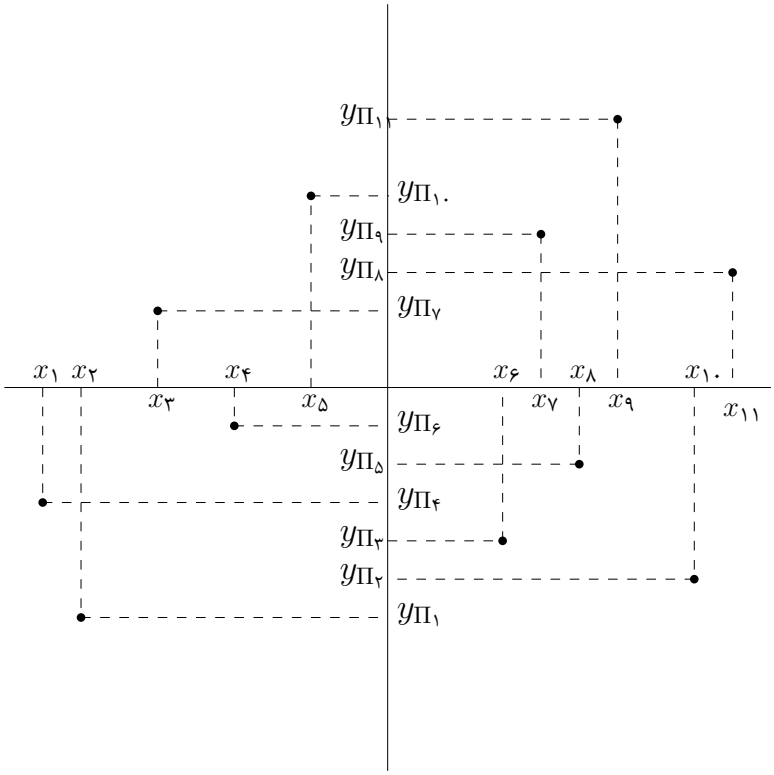
$$M_R(i) = \max_{j \in [i..n]} \Pi_j$$

$$m_R(i) = \min_{j \in [i..n]} \Pi_j$$

به عنوان مثال در شکل ۵.۴، $M_L(5) = \max_{j \in [1..i]} \Pi_j = 10$، $m_L(5) = \min_{j \in [1..i]} \Pi_j = 1$، $M_R(5) = \max_{j \in [i..n]} \Pi_j = 11$ و $m_R(5) = \min_{j \in [i..n]} \Pi_j = 3$ می‌باشد، دقت کنید که برای این شکل جایگشت آن به صورت $\Pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 2 & 10 & 6 & 1 & 8 & 4 & 3 & 11 & 7 & 5 & 9 \end{pmatrix}$ می‌باشد.

مشاهده می‌کنید که اگر $\{\{p_1, ..., p_k\}, \{p_{k+1}, ..., p_n\}\}$ یک قطری‌سازی از P باشد آنگاه $k \in [1..n-1]$ در رابطه‌ی $M_L(k) = k$ (قطری‌سازی پایین به بالا) یا $m_L(k) = n-k+1$ (قطری‌سازی بالا به پایین) صدق می‌کند. به علاوه، $M_L(k) = k$ و $m_L(k) = n-k+1$ به ترتیب معادل $m_R(k+1) = k+1$ و $M_R(k+1) = n-k$ هستند.

نشان می‌دهیم که اگر $M_L(k) = k$ آنگاه $\{\{p_1, ..., p_k\}, \{p_{k+1}, ..., p_n\}\}$ یک قطری‌سازی پایین به بالا می‌باشد. خط افقی ℓ_1 را بین نقاط P_{Π_k} و $P_{\Pi_{k+1}}$، و خط عمودی ℓ_2 را بین نقاط P_k و P_{k+1} رسم می‌کنیم، اول این که واضح است ربع‌های اول و سوم (نسبت به خطوط ℓ_1 و ℓ_2) خالی از نقطه نیست و دوم نشان



شکل ۵.۴: موقعیت نقاط در صفحه

می‌دهیم که ربع‌های دوم و چهارم خالی از نقطه است. فرض خلف، فرض کنید که نقطه P_z در ربع چهارم باشد، چون مختصات دوم نقطه P_z یعنی y_z کم‌ترمساوی $\max\{y_{\Pi_1}, y_{\Pi_2}, ..., y_{\Pi_k}\}$ است (زیرا در ابتدا خط افقی ℓ_1 را بین نقاط P_{Π_k} و $P_{\Pi_{k+1}}$ رسم کردیم) پس $j \in [1..k]$ وجود دارد به طوری که $\Pi_j = z$ باشد و از آنجایی که $z > k$ پس باید داشته باشیم که $M_L(k) \geq z$ که این با فرض $M_L(k) = k$ در تناقض است و از این رو فرض خلف باطل است یعنی هیچ نقطه‌ای در ربع چهارم وجود ندارد. حال فرض کنید که نقطه‌ای مانند P_t در ربع دوم باشد در این صورت چون که $t < k$ است لذا $M_L(k) = k + 1$ (اگر فقط یک نقطه در ربع دوم باشد) که این نیز با فرض $M_L(k) = k$ در تناقض است. پس بنابرین ربع‌های دوم و جهارم خالی از نقطه می‌باشند و حکم ثابت می‌شود.

برای مورد دیگر (اگر $m_L(k) = n - k + 1$ باشد آنگاه $\{\{p_1, ..., p_k\}, \{p_{k+1}, ..., p_n\}\}$ یک قطری‌سازی بالا به پایین می‌باشد) می‌توان به صورت مشابه اثبات کرد.

بنابرین در صورتی که قطری‌سازی برای P وجود داشته باشد، ما آن را به صورت زیر تعیین می‌کنیم: برای $j = 1..\lfloor n/2 \rfloor$ بررسی می‌کنیم که آیا $\{\{p_1, ..., p_j\}, \{p_{j+1}, ..., p_n\}\}$ یک قطری‌سازی است (یعنی

$\{\{p_1, ..., p_{n-j}\}, \{p_{n-j+1}, ..., p_n\}\}$ یا $(m_L(j) = n - j + 1$ یا $M_L(j) = j$ یک قطری‌سازی است (یعنی $m_R(n - j + 1) = n - j + 1$ یا $M_R(n - j + 1) = j$). توجه کنید که اگر $j > 1$ آنگاه $M_L(j)$، $m_L(j)$، $M_R(n - j + 1)$ و $m_R(n - j + 1)$ همگی می‌توانند با $O(1)$ مقایسه از روی $M_L(j - 1)$، $m_L(j - 1)$، $M_R(n - j + 2)$، $m_R(n - j + 2)$ و Π_{n-j+1} محاسبه شوند، به عنوان مثال $M_R(n-j+1)$ را می‌تواند در $O(1)$ از روی $M_R(n-j+2)$ و Π_{n-j+1} به صورت $M_R(n - j + 1) = \max\{M_R(n - j + 2), \Pi_{n-j+1}\}$ ساخته شود. بنابرین، در صورتی که قطری‌سازی $\{\{p_1, ..., p_k\}, \{p_{k+1}, ..., p_n\}\}$ برای P موجود باشد، این امر به ازای $t = \min\{k, n-k\} \leq \lfloor n/2 \rfloor$ تشخیص داده شده و بنابرین با $O(t)$ مقایسه تعیین می‌شود. اگر هیچ قطری‌سازی به ازای مقادیر $j \in [1..\lfloor n/2 \rfloor]$ یافت نشود، آنگاه الگوریتم در کل $O(n)$ مقایسه خواهد داشت.

سپس ما می‌توانیم D-درخت مجموعه‌ی P را به روش زیر به صورت بازگشتی بسازیم. الگوریتم بالا را برای P اجرا می‌کنیم. اگر قطری‌سازی $\{\{p_1, ..., p_k\}, \{p_{k+1}, ..., p_n\}\}$ برای P وجود داشته باشد (این امر با $O(t)$ مقایسه تعیین می‌شود که $t = \min\{k, n-k\}$) یک گره ریشه ایجاد نموده و یک D-درخت از $\{p_1, ..., p_k\}$ را به عنوان فرزند چپ و یک D-درخت از $\{p_{k+1}, ..., p_n\}$ را به عنوان فرزند راست آن قرار می‌دهیم. در غیر این صورت، اگر P دارای هیچ قطری‌سازی نباشد (این امر با $O(n)$ مقایسه تشخیص داده می‌شود)، یک گره برگ با مجموعه‌ی $S(.)$ برابر P ایجاد می‌نماییم. این روش رابطه‌ی بازگشتی زیر را برای تعداد کل مقایسه‌ها $T(n)$ به دست می‌آورد، که در آن $1 \leq t \leq \lfloor n/2 \rfloor$:

$$T(n) = \begin{cases} O(t) + T(t) + T(n-t) & n > 1, \text{ اگر امکان قطری‌سازی داشته باشد} \\ O(n) & \text{در غیر این‌صورت} \end{cases}$$

بدون از دست رفتن کلیت فرض کنید که ثابت‌ها در $O(t)$ و $O(n)$ در رابطه‌ی بازگشتی برابر با یک هستند. سپس با استقرا ثابت می‌کنیم که $T(n) \leq n + n \log n$ است. برای $n = 1$ حکم استقرا صحیح است زیرا داریم $1 \leq 1 + \log 1 = 1$ حالت پایه‌ی استقرا خط دوم رابطه‌ی بازگشتی است، یعنی اگر

مجموعه‌ای با n عضو امکان قطری‌سازی نداشته باشد، در آن رابطه‌ی $n \leq n + n\log n$ همواره برقرار است. فرض کنید که فرض استقرا برای $k \leq n$ برقرار باشد. در حالت استقرایی داریم:

$$\begin{aligned}
T(n) &= t + T(t) + T(n-t) \\
&\leq t + t + t\log t + (n-t) + (n-t)\log(n-t) \\
&= n + t + t\log t + (n-t)\log(n-t) \\
&= n + t + t\log t + (n-t)\log(n-t) + (n\log n - t\log n - (n-t)\log n) \\
&= n + n\log n + t - t\log(n/t) - (n-t)\log(n/(n-t)) \\
&= n + n\log n + n(t/n - H(t/n)) \\
&\leq n + n\log n,
\end{aligned}$$

در خط دوم از فرض استقرا استفاده می‌کند. در خط چهارم $n\log n$ را که به دو صورت متفاوت نوشته شده است اضافه و کم می‌کنیم. در خط پنجم جملات را دسته‌بندی نموده و در خط ششم تابع انتروپی دودویی $H(x) = x\log(۱/x) + (۱-x)\log(۱/(۱-x))$ را وارد می‌کنیم؛ که در آن $x = t/n$ است. در نهایت، در خط آخر نامساوی تحلیلی $x \leq H(x)$ را که حداقل به ازای $x \leq ۱/۲$ صادق است اعمال می‌کنیم. بنابرین، $T(n) \leq n + n\log n$ و از مرتبه‌ی $O(n\log n)$ است. می‌توان مشاهده نمود که این حل با توجه فرض $t = n/۲$ محدود است (به دلیل محدودیت برای تابع انتروپی).

به سادگی می‌توان دید که هر D-درخت مجموعه‌ی P به $O(n)$ فضا نیاز دارد. □

۳.۴ به دست آوردن مستطیل بهینه مسطح

در این بخش نشان خواهیم داد که چگونه با استفاده از D-درخت‌ها می‌توان مستطیل f-بهینه را محاسبه نمود. برای این کار ابتدا مجموعه‌ی «ده» مستطیل f-بهینه را برای یک زیرمجموعه‌ی غیرتهی $A \subseteq P$ که P مجموعه نقاط داده شده مسأله است، تعریف می‌کنیم.

تعریف ۱.۳.۴. برای هر زیرمجموعه‌ی غیرتهی $A \subseteq P$، مجموعه‌ی «ده» مستطیل f-بهینه مجموعه‌ی A، که با $Ten(A)$ نشان داده می‌شود، از مستطیل‌های f-بهینه‌ی زیر تشکیل شده است، که همگی در $Box(A)$

محاط هستند.

۱- $Box(A)$.

۲- $B_{opt}(A)$، یک مستطیل f-بهینه.

۳- $B_1(A)$، یک مستطیل f-بهینه شامل رأس چپ پایین $Box(A)$.

۴- $B_2(A)$، یک مستطیل f-بهینه شامل رأس راست پایین $Box(A)$.

۵- $B_3(A)$، یک مستطیل f-بهینه شامل رأس راست بالای $Box(A)$.

۶- $B_4(A)$، یک مستطیل f-بهینه شامل رأس چپ بالای $Box(A)$.

۷- $B_{1,2}(A)$، یک مستطیل f-بهینه شامل رئوس پایین $Box(A)$.

۸- $B_{2,3}(A)$، یک مستطیل f-بهینه شامل رئوس سمت راست $Box(A)$.

۹- $B_{3,4}(A)$، یک مستطیل f-بهینه شامل رئوس بالای $Box(A)$.

۱۰- $B_{4,1}(A)$، یک مستطیل f-بهینه شامل رئوس سمت چپ $Box(A)$.

لم ۲.۳.۴. برای هر زیرمجموعه‌ی غیرتهی $A \subseteq P$ و هر قطری‌سازی $\{A_1, A_2\}$ از مجموعه‌ی A، $Ten(A)$ می‌تواند با $O(1)$ ترکیب امتیاز از روی $Ten(A_1)$ و $Ten(A_2)$ محاسبه شود.

برهان. بدون از دست رفتن کلیت مسأله فرض کنید که $\{A_1, A_2\}$ یک قطری‌سازی پایین به بالا و u_2 و u_4 به ترتیب رئوس پایین راست و بالای چپ $Box(A)$ باشند. فرض کنید که $\max\{H_1, ..., H_t\}$ نشان دهنده‌ی مستطیل با بیش‌ترین امتیاز برای مجموعه‌ی A در میان مستطیل‌های $H_1, ..., H_t$ باشد. مشاهده می‌شود که:

- $Box(A) = Box(A_1 \cup A_2) = Box(Box(A_1) \cup Box(A_2))$.
- $B_{opt}(A) = \max\{B_{opt}(A_1), B_{opt}(A_2), Box(B_3(A_1) \cup B_1(A_2))\}$.
- $B_1(A) = \max\{B_1(A_1), Box(Box(A_1) \cup B_1(A_2))\}$.
- $B_2(A) = Box(\max\{B_2(A_1), B_2(A_2), Box(B_{2,3}(A_1) \cup B_{1,2}(A_2))\} \cup \{u_2\})$.

- $B_{۳}(A) = \max\{Box(B_{۳}(A_{۱}) \cup Box(A_{۲})), B_{۳}(A_{۲})\}$.
- $B_{۴}(A) = Box(\max\{B_{۴}(A_{۱}), B_{۴}(A_{۲}), Box(B_{۳,۴}(A_{۱}) \cup B_{۴,۱}(A_{۲}))\} \cup \{u_{۴}\})$.
- $B_{۱,۲}(A) = \max\{Box(B_{۱,۲}(A_{۱}) \cup \{u_{۲}\}), Box(Box(A_{۱}) \cup B_{۱,۲}(A_{۲}))\}$
- $B_{۲,۳}(A) = \max\{Box(B_{۲,۳}(A_{۲}) \cup \{u_{۲}\}), Box(Box(A_{۲}) \cup B_{۲,۳}(A_{۱}))\}$
- $B_{۳,۴}(A) = \max\{Box(B_{۳,۴}(A_{۲}) \cup \{u_{۴}\}), Box(Box(A_{۲}) \cup B_{۳,۴}(A_{۱}))\}$
- $B_{۴,۱}(A) = \max\{Box(B_{۴,۱}(A_{۱}) \cup \{u_{۴}\}), Box(Box(A_{۱}) \cup B_{۴,۱}(A_{۲}))\}$

از آنجا که هر یک از عناصر $Ten(A)$ می‌تواند با استفاده از تعداد ثابتی مستطیل‌های دو به دو مجزای $Ten(A_{۱})$ و $Ten(A_{۲})$ به دست آید، و برای هر دو مستطیل مجزای H و H' داریم:

$$f((H \cup H') \cap A) = f((H \cap A) \cup (H' \cap A)) = g(f(H \cap A), f(H' \cap A)),$$

و به این ترتیب نتیجه مورد نظر به دست خواهد آمد. □

قضیه ۳.۳.۴. فرض کنید P مجموعه‌ای شامل n نقطه در صفحه و f تابعی یکنوا و تجزیه‌پذیر بوده که زیرمجموعه‌ای از P را به عنوان آرگومان بپذیرد. الگوریتمی وجود دارد که می‌تواند مستطیل f-بهینه مجموعه‌ی P را با $O\left(n \log n + \sum_{i=۱}^{\beta} h_c(n_i)\right)$ مقایسه (بر روی مختصات و اندیس‌ها) و $O\left(\sum_{i=۱}^{\beta} h_s(n_i) + \beta\right)$ ترکیب امتیاز بیابد، که در آن $\{P_{۱}, ..., P_{\beta}\}$ یک افراز مجموعه‌ی P است که توسط یک D-درخت دلخواه مجموعه‌ی P ایجاد شده است. β اندازه‌ی افراز، n_i عدد اصلی مجموعه‌ی P_i، و $h_c(n_i)$ و $h_s(n_i)$ به ترتیب تعداد مقایسه‌های مختصات نقاط و ترکیب امتیازهای به کار رفته برای محاسبه‌ی «ده» مستطیل بهینه‌ی P_i هستند.

برهان. با $O(n \log n)$ مقایسه، یک درخت قطری‌سازی T برای P بسازید (لم ۴.۲.۴). فرض کنید $u_{۱}, ..., u_{\beta}$ برگ‌های T بوده و برای همه‌ی مقادیر $i \in [۱..n]$ در رابطه‌ی $S(u_i) = P_i$ صدق می‌کنند. مجموعه‌ی $Ten(S(u_i)) = Ten(P_i)$ را با $h_c(n_i)$ مقایسه‌ی مختصات و $h_s(n_i)$ ترکیب امتیاز محاسبه نمایید. با استفاده از پیمایش پس‌ترتیب [۱] درخت T برای هر گره داخلی v از آن، $Ten(A(v))$ را از روی

[۱] Postfix

$Ten(A(v_1))$ و $Ten(A(v_2))$ با $O(1)$ ترکیب امتیاز محاسبه نمایید (لم ۲.۳.۴)، که در آن v_1 و v_2 گره‌های فرزند v هستند. مستطیل بهینه‌ی f مجموعه‌ی P همان $B_{opt}(A(r))$ است، که در آن r گره ریشه‌ی T بوده و در رابطه‌ی $A(r) = P$ است. در مجموع، این الگوریتم با

$$O(n \log n) + \sum_{i=1}^{\beta} h_c(n_i) = O\left(n \log n + \sum_{i=1}^{\beta} h_c(n_i)\right)$$

مقایسه‌ی مختصات و

$$\sum_{i=1}^{\beta} h_s(n_i) + \sum_{i=1}^{\beta-1} O(1) = O\left(\sum_{i=1}^{\beta} h_s(n_i) + \beta\right)$$

ترکیب امتیاز اجرا می‌شود. □

بهترین حالت الگوریتم قضیه‌ی بالا زمانی است که برای هر گره برگ u_i در درخت قطری‌سازی مجموعه‌ی P، $|S(u_i)|$ از مرتبه‌ی $O(1)$ باشد. این حالت پیچیدگی $O(n \log n)$ مقایسه‌ی مختصات و $O(n)$ ترکیب امتیاز را در پی خواهد داشت. بدترین حالت، زمانی است که P امکان قطری‌سازی نداشته باشد. حتی در این حالت نیز $O(n \log n)$ مقایسه‌ی مختصات اضافی انجام شده هنگام تلاش برای قطری‌سازی، به پیچیدگی مجانبی کلی اضافه نمی‌کند. بنابرین پیچیدگی الگوریتم قطری‌سازی در بدترین حالت، کم‌تر از حالت‌های بررسی شده در بخش‌های قبل، با پیچیدگی حداکثر $O(n \log n)$ مقایسه مختصات و $O(n^2 \log n)$ ترکیب امتیاز است.

۱.۳.۴ برخورد با مسأله‌ی آسیاب‌بادی

در این بخش از لم ۲.۱.۴ برای دست‌یابی به شکلی از الگوریتم مطرح شده در قضیه‌ی ۳.۳.۴ استفاده می‌نماییم. امکان قطری‌سازی مجموعه‌ی $S(u)$ متناظر با گره برگ u در درخت قطری‌سازی (D-درخت) مجموعه‌ی P وجود نداشته و این مجموعه بنا به لم ۲.۱.۴ دارای یک آسیاب‌بادی شامل یکی از نقاط کرانی $S(u)$ است. ایده‌ی اصلی، حذف نقاط کرانی $S(u)$ و ایجاد یک درخت قطری‌سازی براساس نقاط باقی‌مانده به صورت بازگشتی است. این روش یک قطری‌سازی در عمق مجموعه نقاط را به دنبال خواهد داشت و به صورت بالقوه می‌تواند تعداد اعمال ترکیب امتیاز را کاهش دهد.

تعریف ۴.۳.۴. یک درخت قطری‌سازی تعمیم‌یافته برای مجموعه نقاط P، D^*-درخت، به صورت بازگشتی و مطابق روند زیر تعریف می‌شود:

«هر گره برگ u متعلق به یک D-درخت از مجموعه نقاط P که در شرط $|S(u)| > ۱$ صدق کند، با گره u' که شامل مجموعه‌ی نقاط $X(u)$، مجموعه‌ی نقاط کرانی $S(u)$، است، جایگزین می‌شود؛ اگر مجموعه‌ی $S(u)\backslash X(u)$ غیرتهی باشد، آنگاه تنها فرزند گره u' یک D^*-درخت از نقاط مجموعه‌ی $S(u)\backslash X(u)$ خواهد بود.»

لم ۵.۳.۴. فرض کنید P مجموعه‌ای از n نقطه در صفحه باشد. همه‌ی D^*-درخت‌های مجموعه‌ی P، دارای تعداد یکسان σ گره تک فرزند و $n - ۴\sigma$ گره برگ بوده و هر گره برگ u در یکی از شرط‌های $|S(u)| = ۱$ یا $|S(u)| = ۴$ صدق می‌کند. یک D^*-درخت مجموعه‌ی P به فضایی از مرتبه‌ی $O(n)$ نیاز دارد و می‌تواند با $O(n \log n + \sigma n)$ مقایسه ساخته شود.

برهان. بخش اول این لم با توجه به لم ۳.۲.۴ و تعریف ۴.۳.۴ اثبات می‌شود. با دنبال کردن الگوریتمی مشابه الگوریتم ایجاد D-درخت تا جایی که گره برگی که قابل قطری‌سازی نباشد یافت شود، می‌توان یک D^*-درخت مجموعه‌ی P را با $O(n \log n + \sigma n)$ مقایسه ایجاد نمود. در اینجا برای ادامه‌ی الگوریتم بر روی مجموعه‌ی $S(u)\backslash X(u)$ بر اساس تعریف ۴.۳.۴، $O(n)$ مقایسه انجام می‌شود. از آنجا که الگوریتم در نهایت تعداد σ گره u می‌یابد، تعداد کل مقایسه‌ها از مرتبه‌ی $O(n \log n + \sigma n)$ است. درخت D^*-درخت دارای n گره با درجه‌ی محدود است، بنابرین این درخت می‌تواند در فضایی خطی ساخته شود. □

قضیه ۶.۳.۴. فرض کنید P مجموعه‌ای از n نقطه در صفحه باشد و f تابعی یکنوا و تجزیه‌پذیر باشد که به عنوان آرگومان ورودی خود هر زیرمجموعه‌ای از P را دریافت می‌کند. الگوریتمی وجود دارد که می‌تواند مستطیل بهینه‌ی f مجموعه‌ی P را با $O(n \log n + \sigma n)$ مقایسه مختصات و $O(n + \sigma n \log n)$ ترکیب امتیاز بیابد، که در اینجا σ تعداد گره‌های تک‌فرزند هر D^*-درخت مجموعه‌ی P است.

برهان. با $O(n \log n + \sigma n)$ مقایسه یک درخت قطری تعمیم‌یافته (D^*-درخت) T برای مجموعه‌ی P ایجاد نمایید (لم ۵.۳.۴). برای هر یک از $n - ۴\sigma$ گره برگ u از درخت T با استفاده از تعداد ثابتی ترکیب

امتیاز، $Ten(S(u))$ را محاسبه کنید. سپس، با استفاده از پیمایش پس‌ترتیب درخت T، $Ten(S(u))$ را برای هر گره داخلی u به صورت زیر محاسبه نمایید:

الف) اگر v دارای دو فرزند v_1 (فرزند چپ) و v_2 (فرزند راست) باشد، آنگاه $\{A(v_1), A(v_2)\}$ یک قطری‌سازی $A(v)$ بوده و با استفاده از $Ten(A(v_1))$ و $Ten(A(v_2))$ می‌توان $Ten(A(v))$ را با $O(1)$ ترکیب امتیاز محاسبه نمود (لم ۲.۳.۴).

ب) در غیر این صورت، اگر v یکی از σ گره تک‌فرزند باشد، آنگاه در بدترین حالت می‌توان $Ten(A(v))$ را با $O(n \log n)$ مقایسه و ترکیب امتیاز محاسبه نمود. یعنی، اگر مستطیل $Ten(A(v))$ شامل حداقل یکی از نقاط $X(u)$ بر روی مرز خود باشد، آنگاه می‌توان این مستطیل را با $O(n \log n)$ مقایسه و ترکیب امتیاز یافت.

ج) در غیر این صورت، این مستطیل همان مستطیل $Ten(A(v'))$ است، که در آن v' گره فرزند گره v است.

برای هر یک از تعداد $O(n)$ گره دارای دو فرزند، تعداد $O(1)$ ترکیب امتیاز و برای هر یک از σ گره تک فرزند تعداد $O(n \log n)$ ترکیب امتیاز انجام می‌شود. بنابرین تعداد کل ترکیب امتیازها از مرتبه‌ی $O(n + \sigma n \log n)$ بوده و سایر نتایج به دست می‌آید. □

فصل ۵

کاربردها و نتیجه‌گیری

۱.۵ کاربردها

مسأله مستطیل بهینه مسطح یک مسأله بهینه‌سازی هندسی است که بسته به تعریف تابع f می‌تواند کاربردهای بسیاری داشته باشد. نکته قابل توجه این است که این مسأله می‌تواند بسیاری مسائل ارائه شده هندسی قبل که مربوط به بهینه‌سازی مستطیل روی نقاط است را پوشش دهد. از کاربردهای دیگر مسأله مستطیل بهینه مسطح می‌توان به شناسایی مناطق مستطیل شکل مطلوب در تصاویر ماهواره‌ای، تجزیه و تحلیل عکس‌ها در پزشکی اشاره کرد که در زیر به برخی از آنها می‌پردازیم.

مسأله تفکیک نقاط دو رنگ با دو مستطیل

این مسأله در سال ۲۰۰۹ توسط کورتس و همکاران مطرح گردید. صورت مسأله بدین گونه می‌باشد: فرض کنید S مجموعه‌ای از n نقطه با رنگ‌های آبی و قرمز می‌باشد، هدف یافتن دو مستطیل بیشینه (از نظر تعداد نقاط) موازی هم است که هر کدام تنها شامل نقاط با یک رنگ باشد به‌طوری که یک مستطیل شامل نقاط آبی و دیگری شامل نقاط قرمز، و هر مستطیل دارای تنها نقاط با یک رنگ باشند. برای این مسأله الگوریتمی با پیچیدگی زمانی $O(n^{۲}\log n)$ ارائه شده است [۷]. می‌خواهیم این مسأله را با استفاده از مسأله مستطیل بهینه مسطح حل کنیم، هدف در این مسأله این است که دو مستطیل بیشینه (از نظر تعداد نقاط) موازی هم بیابیم که هر کدام تنها شامل نقاط با یک رنگ باشد. بدون از دست دادن کلیت مسأله برای این کار ابتدا مستطیل بیشینه که فقط شامل نقاط قرمز است را پیدا می‌کنیم و سپس مستطیل بیشینه که تنها شامل نقاط آبی است را می‌یابیم. برای به دست آوردن مستطیل بیشینه تنها شامل نقاط قرمز، تابع f را به صورت زیر تعریف می‌کنیم:

وقتی که تابع f روی زیرمجموعه‌ای از S مانند A اعمال می‌شود اگر مجموعه A تنها شامل نقاط قرمز باشد ارزشی که تابع f به مجموعه A نسبت می‌دهد برابر تعداد نقاط مجموعه A است و در صورتی که این مجموعه شامل نقطه‌ای با رنگ آبی باشد تابع f به این مجموعه ارزش ∞ را نسبت می‌دهد، توجه کنید که تابع f یک تابع تجزیه‌پذیر و یکنوا می‌باشد و تابع تجزیه متناظر با آن می‌تواند به صورت $g(x,y)=x+y$ باشد. با حل مسأله مستطیل بهینه مسطح بر روی مجموعه S با توجه به تابع f بیشینه مستطیل که تنها شامل نقاط با رنگ قرمز است حاصل می‌شود. به همین ترتیب می‌توان بیشینه مستطیل که تنها شامل نقاط

آبی است را به دست آورد. توجه کنید که پیچیدگی زمانی مسأله مستطیل بهینه مسطح در بدترین حالت برابر $O(n^2 \log n)$ است.

مسأله بزرگترین مستطیل خالی

در سال ۲۰۰۳ برای مسأله بزرگترین مستطیل خالی یانگ لیو و مک هایل ندایک [۵] الگوریتمی ارائه دادند، مسأله این است، فرض که n نقطه داشته باشیم و توسط X^+ و X^- افراز شده باشد. هدف به دست آوردن مستطیلی مانند B است که اندازه$B \cap X^+$ را بیشترین مقدار کند به طوری که $(B \cap X^-) = \emptyset$ باشد. الگوریتم ارائه شده دارای پیچیدگی زمانی $O(s^2 \log s)$ می‌باشد که در آن s برابر اندازه مجموعه X^+ است. این مسأله را نیز می‌توان به سادگی مانند «مسأله تفکیک نقاط دو رنگ با دو مستطیل» به مسأله مستطیل بهینه مسطح تبدیل کرد و با حل آن نتیجه مورد نظر را بدست آورد.

شناسایی مناطق مستطیل شکل مطلوب در تصاویر ماهواره‌ای

فرض کنید که می‌خواهیم به منظور تأمین چوب از جنگل تعدادی از درختان را قطع کنیم، برای این کار باید چند مورد را در نظر بگیریم: اول این که تا حد امکان کم‌ترین خسارت را به جنگل وارد کنیم مثلاً درختانی که دارای سن کم‌تر یا ارتفاع کمتر و یا گونه خاصی از درختان هستند تا حد امکان قطع نشوند و دوم این که درختانی که باید قطع کنیم به صورت یک محدوده مثلاً مستطیلی شکل باشد تا در زمان و همچنین هزینه حاصل از حمل‌ونقل آنها صرفه‌جویی شود. به عنوام مثال ما می‌خواهیم منطقه‌ای مستطیلی شکلی را بیابیم که مجموع ارتفاع درختان آن بیشینه باشد. برای این کار از تصاویر ماهواره‌ای می‌توان کمک گرفت به این صورت که هر درخت را به عنوان یک نقطه فرض کرده و امتیازی که تابع f به آن نسبت می‌دهد متناسب با ارتفاع آن باشد، توجه داشته باشید که بنا به تعریف تابع f این تابع یک تابع تجزیه‌پذیر و یکنوا است و تابع g تابع ترکیب متناظر با آن را می‌توان به صورت $g(x, y) = x + y$ در نظر گرفت. به این ترتیب ما می‌توانیم مسأله برش بهینه درختان جنگل را به مسأله مستطیل بهینه مسطح تبدیل و اقدام به حل کردن آن نماییم.

مسأله مستطیل بهینه مسطح در بسیاری از علوم و مهندسی‌ها می‌تواند یک شروع خوب برای حل یک مسأله بزرگ‌تر باشد.

۲.۵ نتیجه‌گیری

کورتس و همکاران [۷] با استفاده از ساختمان داده‌ای به نام درخت MCS که برای نگهداری امتیاز سلسه‌مراتب قسمت‌ها در نظر گرفته شده بود، راه‌حلی برای مسأله جعبه دارای وزن بیشینه[۱] پیشنهاد نمودند. در این پایان‌نامه این راه‌حل که از چندین جنبه تعمیم داده شده [۲]، بررسی شد:

۱. نشان داده شد که چگونه می‌توان عملگر جمع را با یک تابع یکنوا و تجزیه‌پذیر f جایگزین نمود، به نحوی که بتوان از الگوریتم کورتس و همکاران [۷] به منظور بهینه‌سازی توابع امتیازدهی کلی‌تری در مقایسه با صرفاً حاصل‌جمع وزن نقاط استفاده نمود.

۲. ساختمان داده درخت MCS به گونه‌ای تعمیم داده شد که کاملاً پویا شده و بتواند اعمال اضافه کردن و حذف نقاط به ساختمان داده و با توجه به ترتیب نسبی آن‌ها را در زمان لگاریتمی نسبت به تعداد نقاط انجام دهد.

۳. در این پایان‌نامه روش‌هایی تطبیقی به منظور بهره‌گیری از ویژگی‌های خاص نمونه‌ها (مانند خوشه‌بندی نقاط مثبت و منفی یا موقعیت نسبی نقاط) بدون افزایش پیچیدگی زمانی در بدترین حالت بررسی شد.

۱.۲.۵ مسأله‌های باز

مسأله‌های بازی که می‌توان مطرح نمود عبارت است از:

۱. دسته‌بندی‌های متفاوت دیگر از آنچه بررسی نمودیم در نظر گرفته و نشان دهیم که الگوریتم چگونه به واسطه آن تطبیقی می‌شود،

۲. تعمیم مسأله به ابعاد بالاتر، به عنوان مثال برای سه بعدی مسأله به این صورت است که می‌توان نقاط را در فضا در نظر گرفت و مکعب مستطیلی با اضلاع موازی محور مختصات چنان بیابیم که

[۱] Maximum Weight Box

نقاط در درون این مکعب مستطیل دارای بیشترین امتیاز در بین تمام مکعب مستطیل‌های موجود باشد و

۳. تعمیم مسأله برای اشکال غیر از مستطیل مانند چندضلعی‌های محدب است، به عنوان مثال به جای مستطیل پنج‌ضلعی محدبی را پیدا کنیم که دارای بیشترین امتیاز باشد و یکی از اضلاع آن موازی محور مختصات باشد.

واژه‌نامه فارسی به انگلیسی

آخرین ..... Last
آرگومان ..... Argument
آسیاب‌بادی ..... Windmill
اطلاعات ..... Information
امتیاز ..... Score
اندازه ..... Size
اولین ..... First
بازآرایی ..... Rearrange
بالا به پایین ..... Top-down
بخش ..... Partition
بدترین حالت ..... Worst case
برگ ..... Leaf
برگ‌ها ..... Leaves
به روز رسانی ..... Updat
بهترین حالت ..... Best case
بهینه ..... Optimal
بیشینه‌سازی ..... Maximiz
پایین به بالا ..... Bottom-up
پسوند ..... Suffix
پشتیبانی ..... Support
پویا ..... Dynamic
پیچیدگی درج محلی ..... Local insertion complexity
پیچیدگی ..... Complexity
پیداکردن ..... Find
پیشوند ..... Prefix
پیمایش پس‌ترتیب ..... post-order traversal
پیمایش میان‌ترتیب ..... In-order traversal
پیمایش ..... Traverse
تابع انتروپی ..... Entropy function
تابع ..... Function
تجزیه‌پذیر ..... Decomposable

تحدب Convexity

تحلیل جمعی Aggregate analysis

تحلیل Analyze

ترکیب Compositions

تطبیقی Adaptive

تک‌رنگ Monochromatic

ثابت Static

جایگزین کردن Replace

جایگشت Permutation

جد Ancestor

جست‌وجو Query

جمع Sum

چرخش Rotation

چندضلعی Polygon

حذف Deletion

حقیقی‌مقدار Real-valued

خاصیت انگشت پویا Dynamic finger property

خاصیت Property

خودبهبوددهنده Self-adjusting

خودمتوازن Self-balancing

درج Insertions

درخت گسترده Splay tree

درخت Tree

دسترسی Access

دسترسی Accesses

دلخواه Arbitrary

دنباله Sequence

دودویی Binary

رتبه Rank

ریشه Root

زمان سرشکن‌شده Amortized time

زیردنباله متوالی Consecutive subsequence

زیرمحدوده Subrange

ساختمان داده . . . . . Data structure
طول . . . . . Length
عامل . . . . . Factor
عملیات . . . . . Operations
عنصر . . . . . Element
غیرخالی . . . . . Non-empty
غیرفعال‌سازی . . . . . Deactivation
فاصله . . . . . Distance
فاصله . . . . . Interval
فرزندان . . . . . Children
فعال‌سازی . . . . . Activation
قطری‌سازی . . . . . Diagonalization
کران بالا . . . . . Upper bound
کم‌ترین . . . . . Minimum
گره . . . . . Node
متقارن . . . . . Symmetric
متوازن . . . . . Balanced
متوالی . . . . . Successively
مثبت . . . . . Positive
محدود . . . . . Bounded
مختصات . . . . . Coordinate
مرتب کردن . . . . . Sort
مستطیل . . . . . Rectangle
مسطح . . . . . Planar
مسیر . . . . . Path
معتبر . . . . . Valid
مقایسه . . . . . Comparison
منفی . . . . . Negative
موازی با محور . . . . . Axis-aligned
موقعیت کلی . . . . . General position
نقاط کرانی . . . . . Extreme points
نقطه . . . . . Point
نمایه . . . . . Index

نوار تک‌رنگ ..... Monochromatic strip
نوار ..... Stripe
نیمه‌پویا ..... Semi-dynamic
نیمه‌پویا ..... Semi-dynamic
هزینه ..... Cost
هم‌ترازی ..... Alignment
وارونگی ..... Inversion
وزن ..... Weight
یکنوا ..... Monotone

واژه‌نامه انگلیسی به فارسی

دسترسی ..... Access
دسترسی ..... Accesses
فعال‌سازی ..... Activation
تطبیقی ..... Adaptive
تحلیل جمعی ..... Aggregate analysis
هم‌ترازی ..... Alignment
زمان سرشکن‌شده ..... Amortized time
تحلیل ..... Analyze
جد ..... Ancestor
دلخواه ..... Arbitrary
آرگومان ..... Argument
موازی با محور ..... Axis-aligned
متوازن ..... Balanced
بهترین حالت ..... Best case
دودویی ..... Binary
پایین به بالا ..... Bottom-up
محدود ..... Bounded
فرزندان ..... Children
مقایسه ..... Comparison
پیچیدگی ..... Complexity
ترکیب ..... Compositions
زیردنباله متوالی ..... Consecutive subsequence
تحدب ..... Convexity
مختصات ..... Coordinate
هزینه ..... Cost
ساختمان داده ..... Data structure
غیرفعال‌سازی ..... Deactivation
تجزیه‌پذیر ..... Decomposable
حذف ..... Deletion
قطری‌سازی ..... Diagonalization
فاصله ..... Distance

Dynamic finger property ..... خاصیت انگشت پویا
Dynamic ..... پویا
Element ..... عنصر
Entropy function ..... تابع انتروپی
Extreme points ..... نقاط کرانی
Factor ..... عامل
Find ..... پیداکردن
First ..... اولین
Function ..... تابع
General position ..... موقعیت کلی
Index ..... نمایه
Information ..... اطلاعات
In-order traversal ..... پیمایش میان‌ترتیب
Insertions ..... درج
Interval ..... فاصله
Inversion ..... وارونگی
Last ..... آخرین
Leaf ..... برگ
Leaves ..... برگ‌ها
Length ..... طول
Local insertion complexity ..... پیچیدگی درج محلی
Maximiz ..... بیشینه‌سازی
Minimum ..... کم‌ترین
Monochromatic strip ..... نوار تک‌رنگ
Monochromatic ..... تک‌رنگ
Monotone ..... یکنوا
Negative ..... منفی
Node ..... گره
Non-empty ..... غیرخالی
Operations ..... عملیات
Optimal ..... بهینه
Partition ..... بخش
Path ..... مسیر

جایگشت Permutation

مسطح Planar

نقطه Point

چندضلعی Polygon

مثبت Positive

پیمایش پس‌ترتیب post-order traversal

پیشوند Prefix

خاصیت Property

جست‌وجو Query

رتبه Rank

حقیقی‌مقدار Real-valued

بازآرایی Rearrange

مستطیل Rectangle

جایگزین‌کردن Replace

ریشه Root

چرخش Rotation

امتیاز Score

خودبهبوددهنده Self-adjusting

خودمتوازن Self-balancing

نیمه‌پویا Semi-dynamic

نیمه‌پویا Semi-dynamic

دنباله Sequence

اندازه Size

مرتب کردن Sort

درخت گسترده Splay tree

ثابت Static

نوار Stripe

زیرمحدوده Subrange

متوالی Successively

پسوند Suffix

جمع Sum

پشتیبانی Support

متقارن Symmetric

بالا به پایین Top-down

پیمایش Traverse

درخت Tree

به روز رسانی Updat

کران بالا Upper bound

معتبر Valid

وزن Weight

آسیاب‌بادی Windmill

بدترین حالت Worst case

مراجع

[1] G. Adelson-Velskii and E. M. Landis. An information organization algorithm. *Doklady Akademia Nauk SSSR.*, pages 263–266, 1962.

[2] Barbay, Jérémy, G. Navarro, and P. Pérez-Lantero. Adaptive Techniques to Find Optimal Planar Boxes. *arXiv preprint arXiv:1204.2034*, 2012.

[3] C. Bautista-Santiago, J. M. Díaz-Bánez, D. Lara, P. Pérez-Lantero, J. Urrutia, and I. Ventura. Computing optimal islands. *Operations Research Letters*, pages 246–251, 2011.

[4] J. Bentley. Programming pearls: algorithm design techniques. *Communications of the ACM 27*, pages 865–873, 1984.

[5] R. Cole, B. M. J. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. part I: Splay sorting log n-block sequences. *SIAM Journal on Computing 30*, pages 1–43, 2000.

[6] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001.

[7] C. Cortés, J. M. Díaz-Bánez, P. Pérez-Lantero, C. Seara, J. Urrutia, and I. Ventura. Bichromatic separability with two boxes: A general approach. *Journal of Algorithms 64*, pages 79–88, 2009.

[8] D. P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *Journal of computer and system sciences 52*, pages 453–470, 1996.

[9] D. P. Dobkin and S. Suri. Dynamically computing the maxima of decomposable functions, with applications. *In Foundations of Computer Science*, pages 488–493, 1989.

[10] J. Eckstein, P. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its application to data analysis. *Computational Optimization and Applications 23*, pages 285–298, 2002.

[11] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys (CSUR) 24*, pages 441–476, 1992.

[12] D. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1968.

[13] Y. Liu and M. Nediak. Planar case of the maximum box and related problems. *Canadian Conference on Computational Geometry*, pages 14–18, 2003.

[14] H. Mannila. Measures of presortedness and optimal sorting algorithms. *Institute of Electrical and Electronics Engineers Transactions on 100*, pages 318–325, 1985.

[15] A. Moffat and O. Petersson. An overview of adaptive sorting. *Australian Comp. J.*, pages 70–77, 1992.

[16] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, pages 652–686, 1985.

[17] T. Takaoka. Effcient algorithms for the maximum subarray problem by distance matrix multiplication. *Electronic Notes in Theoretical Computer Science 61*, pages 191–200, 2002.

Abstract

Assume a given set P of n planar points, and a scoring function f which assigns a real-value to every subset of P. The problem of *Optimal Planar Box* is to find a rectangle H (aligned with the coordinate axes) which maximizes the value of $f(H \cap P)$. In this problem, we assume f to be a monotone and decomposable function, i.e. there exists a composition function g monotone in its two arguments such that $f(A) = g(f(A_1), f(A_2))$ for every subset $A \subseteq P$ and every partition $\{A_1, A_2\}$ of A. In the present thesis, we examine a solution to the *Optimal Planar Box* problem where, in the worst case, $O(n^2 \log n))$ score compositions and coordinate comparisons can be performed, and much less on other classes of instances defined by various measures of difficulty. A secondary result of this research is the fully dynamic *MCS Splay tree* data structure which supports delete and insert operations as well as the *dynamic finger* property, thus improving previous results.

Yazd University

Faculty of Mathematics

Department of Computer Science

Thesis submitted

for the degree of Master of Science

Title:

Adaptive Techniques to find Optimal Planar Boxes

Supervisor:

Dr. Mohammad Farshi

Advisor:

Dr. Mahdieh Hasheminezhad

By:

Hadi Khadempoor

September 2013