ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[زمستان ۹۹]

جلسه ۱۶: الگوریتم‌های هرم — نگارنده: آئیریا محمدی

---

۱ overview

پیش‌تر درباره درخت‌ها صحبت کردم. [۱] ضریب انشعاب [۱] را تعریف کردیم و گفتیم اگر درختی ضریب انشعابش بیشتر از ۲ باشد، مرتبه لیست کردن اعداد در آن و پیدا کردن مینیمم و ماکسیمم در آن چیست. مثلا دیدیم اگر بخواهیم چیزی درج کنیم از مرتبه ارتفاع درخت می‌شود. و مرتب‌سازی هم دیدیم همچنان $O(nlogn)$ است.

چیز دیگری که دیدیم درج و حذف توی binary tree بود که علی‌رغم عدم نیاز آن هم از $O(h)$ می‌شود (چون باید همچنان جایی برای درج کردن پیدا کنیم).

همچنین با داده‌ساختارهایی مانند صف‌اولویت ، هرم و درخت آشنا شدیم. صف اولویت اعداد درج شده را با ترتیب متفاوت بر اساس یک معیار (مثل کوچک‌تری) پس می‌دهد. و یک کاربرد هرم، پیاده‌سازی صف اولویت است.

این جلسه درباره الگوریتم‌هایی که می‌توانیم روی heap اجرا کنیم و مرتبه زمانی آن‌ها صحبت می‌کنیم.

۲ هرم

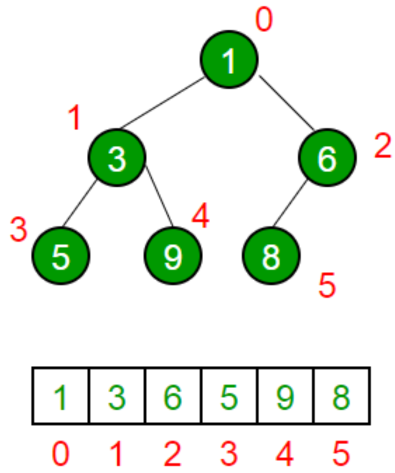
۱.۲ پیاده‌سازی

هرم [۲] را چطوری نگه‌داری کنیم؟ یک گزینه خوب درخت است. هرم یک نوع درخت دودویی [۳] است. هر گره [۴] دو تا رفرنس دارد و یک مقدار [۵] . یکی از رایج‌ترین پیاده‌سازی‌های‌ها هرم دودویی [۶] است که دو نوع بیشینه [۷] و کمینه [۸] دارد . در هرم کمینه مقدار هر گره هم از مقدار دو گره فرزند کمتر است (و در هرم بیشینه بیشتر).

برای این که هرم متوازن را نگه داریم می‌توان از آرایه استفاده کرد. به شکل زیر:

---

[1] Branch factor
[2] Heap
[3] Binary tree
[4] Node
[5] Value
[6] Binary heap
[7] Max heap
[8] Min heap



شکل ۱: نحوه نگهداری هرم با استفاده از آرایه

NODE(A)
left-child(a) = 2a + 1
right-child(a) = 2a + 2
parent(a) = $\lfloor (a-1)/2) \rfloor$

۲.۲ آرایه یا درخت

اگر هرم را توی آرایه نگهداری کنیم به هر عضو، بچه‌هایش و پدرش توی O(۱) دسترسی داریم و نیازی به درخت پیدا نمی‌کنیم. ولی خوبی درخت جایی هست که بخواهیم عضوی را حذف کنیم. آن موقع می‌توانیم راحت زیر درختی را فرزند یک گره دیگر بگذاریم. توی آرایه ولی باید تک‌تک بچه‌ها را جابجا کرد.

۳.۲ ویژگی‌ها

عمق هرم‌های دودویی بهینه است.

توجه توی درخت به ارتفاع $h = [log۲n]$ می‌شود $۲^{h+۱} - ۱$ عضو جا داد.

۴.۲ عملیات‌های هرم (کمینه)

۱.۴.۲ decrease key

مقدار یک گره را عوض می‌کنیم و تا وقتی که از پدرش کوچک‌تر است آن را با پدرش عوض می‌کنیم . پس اردر decrease key این‌جا میشه $O(h)$ که h از $O(logn)$ است.

۲.۴.۲ delete min

بیاییم آخرین عضوی که درج کردیم را به جای آن‌چه حذف کردیم بگذاریم. اگه از بچه هایش کوچیک تر نیست،ببینیم کدام بچه کوچیک تر است و با آن تعویض کنیم. و این کار را تا جایی که نیاز است ادامه می‌دهیم.

این‌شکلی extract min را داریم که از اردر ارتفاع شد.

۳.۴.۲ find max

حالا می‌خواهیم دنبال max بگردیم. به جای گشتن توی همه عضوها عملا فقط لازم است توی برگ‌ها بگردیم. کی می‌شود فهمید چه چیزی برگ است؟ باید بچه نداشته باشد. کی بفهمیم بچه ندارد؟ اندیس‌های ۲i+۱ و ۲i+۲ رو باید چک کنیم. در بدترین حالت باید نصف اعضا را بگردیم که از O(n) است.

۴.۴.۲ insert

چطوری insert کنیم؟ یک عضو بی‌نهایت بزرگ در انتها می‌گذاریم. بعد آن‌قدر decrease key می‌کنیم تا به مقدار مورد نظر برسد.

پس add/push هم از $O(logn)$ شد.

۵.۲ مرتب‌سازی هرمی

۱.۵.۲ با حافظه کمکی

الان عملا دیگر صف اولویت [9] را داریم. با $O(logn)$ عملیات می‌توانیم درج کنیم و با $O(logn)$ می‌توانیم extract کنیم. یک الگوریتم برای مرتب کردن اعداد به شکل زیر می‌تواند باشد:

```
HeapSort(A, n)
heap : Heap
for i = 0 : n
    heap.add(A[i])
for i = 0 : n
    A[i] = heap.pop-min()
```

مرتبه زمانی آن $O(nlogn)$ می‌شود: $O(nlogn)$ برای اضافه کردن و $O(nlogn)$ برای استخراج متوالی. مصرف حافظه آن $O(n)$ می‌شود چون به آرایه کمکی برای نگهداری heap نیاز داریم.

[9] Priority queue

۲.۵.۲ بدون حافظه کمکی

حال اگه طوری می‌شد خود آرایه را مستقیما تبدیل به heap بکنیم آن وقت دیگر لازم به n بار add کردن نیست. و مرتبه حافظه‌اش $O(۱)$ خواهد بود.

build heap یک ایده این است که از اولین عنصر شروع کنیم و فرض کنیم در آرایه خالی درج می‌شود. برای اعضای بعدی هم کار مشابه می‌کنیم و اعضائی که اضافه می‌شوند در صورت نیاز به بالا می‌آیند و عملا n بار درج انجام شده.

خب پس الان می‌توانیم برای مرتب کردن یک آرایه بر روی آن هرم بسازیم، سپس یکی‌یکی pop کنیم. مرتبه‌زمانی نهایی $O(nlog)$ خواهد بود.

اگر بخواهیم build را بهینه کنیم چی؟ عملیات heapify از $O(n)$ خواهد بود. [۲]

۳ مسئله convex hull

در پایان جلسه ۱۴ام مسئله‌ای مطرح شد که این‌جا پاسخ می‌دهیم.

مسئله فرض کنید n نقطه در صفحه داریم؛ الگوریتمی وجود دارد که چند ضلعی عبوری از تمام این نقاط را میابد. با به کمک sort نشان دهید این کار را در مرتبه زمانی کمتر از $O(nlogn)$ نمی‌توان انجام داد.

سوال چگونه این n نقطه را بیابیم؟

یک پاسخ بدیهی این است که بیابیم نقطه پایین چپ را بگیریم. هر دفعه دنبال نقطه با کمترین زاویه می‌گردیم. اگه مساوی باشند آن که x کمتر دارد را انتخاب می‌کنیم. این البته $O(n^۲)$ خواهد شد.

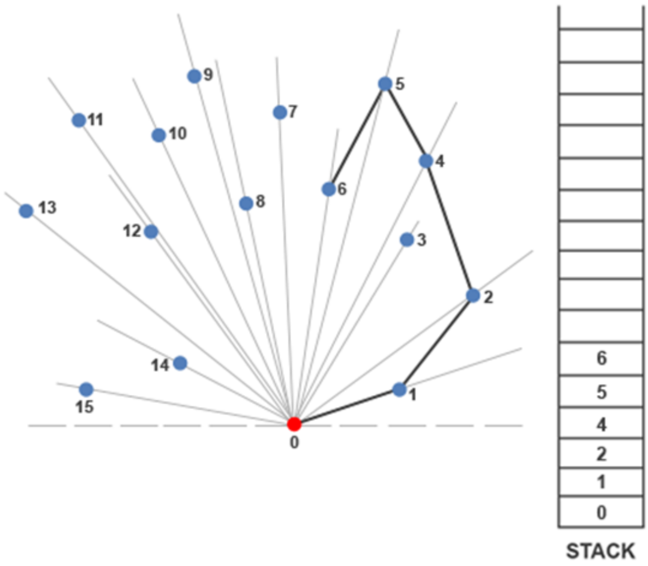
یک راه بهتر این است که نقطه پایین چپ را پیدا کنیم، زاویه همه نقاط نسبت به آن یک نقطه را اندازه بگیریم. و نقاط را بر حسب اندازه زاویه مرتب کنیم. و حالا روی آن گراهام اسکن [۱۰] انجام بدهیم. به شکل سرشکن [۱۱] $O(nlogn)$ خواهد شد.

حل حالا نشون می‌دهیم پوش محدب [۱۲] را نمی‌شود در کمتر از $O(nlogn)$ پیدا کرد.

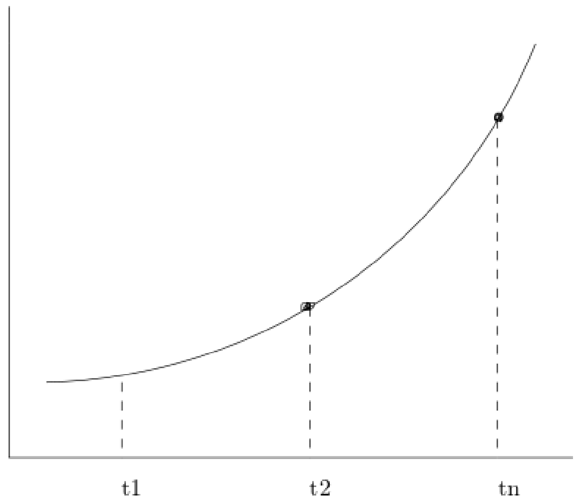
بیاییم x ها را روی نقاط سهمی تصویر [۱۳] کنیم و برای آن‌ها پوش محدب پیدا کنیم (سهمی اصلا خودش یک تابع محدب [۱۴] هست). در اینجا می‌خواهیم راس‌ها و ترتیبشان را به دست بیاوریم. که از چند ضلعی محدب روی سهمی به دست می‌اید. و ترتیب راس‌ها، همان مرتب‌شده اعضایمان است.

به عبارتی پیداکردن پوش محدب در عمل هم‌ارز مرتب کردن نقاط است(چرا؟). یعنی اگر بتوان پوش محدب را در زمانی کمتر از $O(nlogn)$ پیدا کرد، می‌توان نقاط را در همان زمان مرتب کرد. در حالی که می‌دانیم همچین چیزی نمی‌تواند امکان داشته باشد.

[10] Graham's scan
[11] Amortized analysis
[12] Convex hull
[13] Project
[14] Convex function



شکل ۲: اجرای الگوریتم گراهام‌اسکن برای پیدا کردن پوش محدب



شکل ۳: تصویرکردن اعداد روی سهمی

حالا اگر الگوریتم ترتیب را ندهد و فقط نقاط را داشته باشیم؟ آن وقت به این شکل نمی‌شود ثابت کرد که هیچ وقت از $O(nlogn)$ کمتر نمی‌شود ولی به روش دیگه‌ای می‌شود.

۴ بیشتر بخوانید

https://en.wikipedia.org/wiki/Graham_scan

https://en.wikipedia.org/wiki/Amortized_analysis

مراجع

[۱] جزوه جلسه ۱۵

[۲] علما، محمدعلی. *ساخت هرم در زمان خطی*