به نام خدا

دانشگاه تهران، دانشکده مهندسی برق و کامپیوتر

طراحی و تحلیل الگوریتم‌ها، نیمسال دوم، سال تحصیلی ۹۶-۹۵

پاسخ تمرین سری اول

طراح : سینا کچویی sina95kachoei@gmail.com

---

1. مسئله‌ای کلی‌تر را حل میکنیم: در آرایه A اندیس‌های i,j به گونه‌ای که $i < j$ و $A[j] - A[i]$ بیشینه باشد را بعلاوه اندیس عنصرهای بیشینه و کمینه A را میابیم. حال با استفاده از یک الگوریتم تقسیم و حل این مسئله را حل میکنیم:
میدانیم که برای i,j سه حالت داریم: یا هر دو کوچکتر از $n/2$ هستند، یا هر دو بزرگتر از $n/2$ هستند یا اینکه i کوچکتر از $n/2$ و j بزرگتر از $n/2$ است. برای پوشش دادن هر سه حالت اینگونه عمل میکنیم: ابتدا مسئله را برای $A[1:n/2]$ و $A[n/2:n]$ به صورت بازگشتی حل میکنیم، میدانیم که اگر جواب بهینه حالت اول را داشته باشد همان پاسخ مسئله برای $A[1:n/2]$ است، اگر حالت دوم را داشته باشد همان پاسخ مسئله برای $A[n/2:n]$ است و اگر پاسخ بهینه حالت سوم را داشته باشد میدانیم که i باید اندیس عنصر کمینه در $A[1:n/2]$ و j باید اندیس عنصر بیشینه در $A[n/2:n]$ باشد و این دو مقدار را هم در حل بازگشتی زیرمسئله‌ها بدست آورده‌ایم و نهایتا مسئله با مقایسه این سه مقدار برای سه حالت و انتخاب بیشینه آن‌ها حل میشود. تنها چیزی که باقی میماند بدست آوردن عناصر بیشینه و کمینه A است:
عنصر بیشینه برابر است با $max(M_1, M_2)$ که در آن M_1 عنصر بیشینه $A[1:n/2]$ و M_2 عنصر بیشینه $A[n/2:n]$ است. عنصر کمینه برابر است با $min(m_1, m_2)$ که در آن m_1 عنصر کمینه $A[1:n/2]$ و m_2 عنصر کمینه $A[n/2:n]$ است.
زمان اجرای این الگوریتم از رابطه بازگشتی $T(n) = 2T(n/2) + O(1)$ پیروی میکند. با استفاده از قضیه اصلی زمان اجرای این الگوریتم از $O(n)$است.

2. براي حل اين سوال می توان از ايده اي مانند الگوريتم Quicksort استفاده می کنيم.

1. نقطه ای با کمترین مقدار x و نقطه با بیشترین مقدار x را پیدا میکنیم

2. این دو نقطه را با یک خط (L) به یکدیگر وصل می کنیم.در نتیجه مجموعه نقاط به دو قسمت بالاي خط L و پایین خط L تقسیم می شوند. هر کدام از بخش ها را جداگانه با قدم های زیر حل میکنیم.

3. در نقاط یک بخش، نقطه ای را پیدا می کنیم که بیشترین فاصله با خط L را داشته باشد. (نقطه P). می دانیم که نقطه ی P جزو نقاطی است که در جواب نهایی وجود دارند. این نقطه را به دو سر پاره خط L وصل می کنیم، یک مثلث تشکیل می شود، واضح است که نقاطی که درون این مثلث هستند نمی توانند در جواب نهایی ما باشند.

3. ایده ی حل این سوال divide and conquer می باشد. به گونه ای که ما در این مستطیل ها، ارتفاع مینیموم را پیدا کرده و سپس مساحت بزرگترین مستطیل، maximum یکی از مقادیر زیر می باشد:

1. مساحت maximum در سمت راست minimum value
2. مساحت maximum در سمت چپ minimum value
3. تعداد کل میله ها ضرب در مقدار minimum value

که مساحت بزرگترین مستطیل مناطق سمت چپ و راست می توان به صورت بازگشتی استفاده کرد، پس بدترین حالت زمان پیچیدگی این الگوریتم O (n^2) می شود. در بدترین حالت، ما همیشه یک عنصر (n−1) در یک طرف و 0 عنصر در طرف دیگر داریم. هزینه O(n^2) می باشد.

ب) میتواینم از Segment tree استفاده کنیم و تعریف آن به این صورت است که:

Representation of Segment trees

1. Leaf Nodes are the elements of the input array.

2. Each internal node represents minimum of all leaves under it.

هزینه ی کل = هزینه ی ساخت segment tree + هزینه ی بازگشتی پیدا کردن بزرگترین مستطیل

که در این صورت T(n) = O(Logn) + T(n−1) که در نتیجه هزینه ی کل O(nLogn) محاسبه می شود.

4. از ایده ی Binary Search استفاده می کنیم، و در آن 1− (n+1)*5 عددی است که به عنوان سقف در نظر میگیریم. حال روی اعداد 1 تا 1− (n+1)*5 این جستجو را اعمال می کنیم. و کافی است 4 عدد بعد از آن را هم مورد بررسی قرار دهیم.

5. الف) رئوس گراف را به دو مجموعه هم‌اندازه V_1 و V_2 افراز میکنیم. سپس به صورت بازگشتی یک مسیر همیلتونی در گراف القا شده توسط V_1 و یک مسیر همیلتونی در گراف القا شده توسط V_2 میابیم. این دو مسیر را $P_1 = < x_1, x_2, \ldots, x_{n/2} >$ و $P_2 = < y_1, y_2, \ldots, y_{n/2} >$ مینامیم. حال باید این دو مسیر را ادغام کنیم تا یک مسیر همیلتونی برای گراف اصلی بدست آوریم. بدون لطمه وارد کردن به کلیت مسئله میتوان فرض کرد که $x_1 \to y_1$ حال اگر داشته باشیم $x_{n/2} \to y_1$ آنگاه مسئله حل شده‌است: مسیر $< x_1, \ldots, x_{n/2}, y_1, \ldots, y_{n/2} >$ پاسخ مسئله است پس حالتی را در نظر میگیریم که $y_1 \to x_{n/2}$: در مسیر P_1 جلو میرویم تا به x_i برسیم به گونه‌ای که $x_i \to y_1$ و $y_1 \to x_{i+1}$ (به آسانی میتوان نشان داد که چنین iای وجود دارد) و سپس یال $x_i \to y_1$ را طی میکنیم. حال داریم $y_1 \to x_{i+1}$. اگر داشته باشیم $y_{n/2} \to x_{i+1}$ با اضافه کردن باقیمانده P_2 به مسیر و رفتن به x_{i+1} و سپس طی کردن باقیمانده P_1 مسئله حل میشود. حالتی را در نظر میگیریم که $x_{i+1} \to y_{n/2}$. حال در وضعیتی مشابه قبل

هستیم: در مسیر P_2 جلو میرویم تا به y_j برسیم که $y_j \to x_{i+1}$ و $x_{i+1} \to y_{j+1}$ و یال $y_j \to x_{i+1}$ را به مسیر اضافه میکنیم. این عمل را تا جاییکه دو مسیر کاملا ادغام شوند ادامه میدهیم. مانند شکل زیر:

زمان اجرای این الگوریتم از رابطه بازگشتی $T(n) = 2T(n/2) + O(n)$ پیروی میکند که با استفاده از قضیه اصلی بدست میاید $T(n) \in O(nlgn)$.

ب) میدانیم که هر الگوریتمی که یک آرایه n عضوی از اعداد را مرتب میکند زمان اجرای آن در $\Omega(nlgn)$ است. بدون لطمه خوردن به کلیت مسئله میتوان فرض کرد که الگوریتمی که این مسئله را حل میکند برای یافتن جهت یال بین u, v از یک تابع مانند $d(u, v)$ استفاده میکند که اگر $u \to v$ آنگاه $d(u, v) = 1$ و در غیر این صورت $d(u, v) = 0$. به عبارت دیگر دسترسی به ماتریس مجاورت گراف از طریق این تابع انجام میشود. حال فرض کنید که الگوریتمی داریم که این مسئله را با زمان اجرای $o(nlgn)$ حل میکند (به تفاوت $O(nlgn)$ و $o(nlgn)$ دقت کنید). از این الگوریتم استفاده میکنیم تا یک الگوریتم با زمان اجرای $o(nlgn)$ برای مسئله مرتب سازی ارایه دهیم: رئوس گراف را اعدادی که قرار است مرتب شوند در نظر میگیریم و $d(u, v) = 1$ اگر $u \leq v$. مشخص است که یک مسیر همیلتونی در این گراف متناظر با یک ترتیب صعودی برای عناصر آرایه است. پس وجود چنین الگوریتمی با حد پایین $\Omega(nlgn)$ برای مرتب‌سازی در تناقض است.

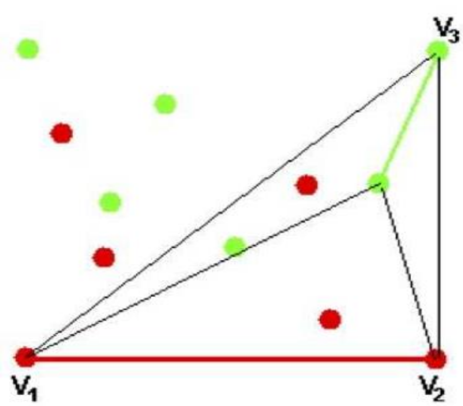
6. ایده استفاده از Binary Search می باشد. عنصر میانی را با همسایه هایش مقایسه می کنیم. اگر عنصر میانی کوچکتر از هر یک از همسایه هایش نیست، پس جواب ما می باشد. درغیراین صورت، اگرعنصرمیانی کوچکترازهمسایه چپ آن باشد، در اینجا همیشه یک Top Element در سمت چپ وجود دارد. اگرعنصرمیانی کوچکترازهمسایه راست آن باشد، دراینجا همیشه یک Top Element درسمت راست وجود دارد

7. یک مثلث متشکل از رئوس v_1،v_2 و v_3 را در نظر بگیرید که v_1 و v_2 همرنگ هستند و رنگ c را دارند و v_3 رنگ c' را دارا میباشد. رئوس v_1 و v_2 را به هم متصل میکنیم و به شکل زیر ادامه میدهیم:

- اگر نقطه دیگری درون این مثلث وجود نداشته باشد کار ما با این مثلث تمام شده‌است.
- اگر تمام نقاط درون این مثلث همرنگ باشند همه این نقاط را به یکی از رئوس مثلث که همرنگ آن‌هاست متصل میکنیم و سپس کار ما با این مثلث تمام شده‌است.
- اگر درون مثلث نقاط با رنگ متفاوت داشته باشیم، یک نقطه که رنگ c' را داراست را انتخاب میکنیم و آن را به v_3 متصل میکنیم. حال مثلث اولیه به سه مثلث کوچکتر تبدیل شده که به صورت بازگشتی به همین طریق قابل حل هستند.

حال برای حل مسئله اصلی کافیست که مربع را روی قطرش به دو مثلث تقسیم کنیم و مسئله را برای دو مثلث بدست آمده حل کنیم:

زمان اجرای این الگوریتم از رابطه $T(n) = T(n-(i+j)) + T(i) + T(j) + O(n)$ پیروی میکند که در آن $n-(i+j)$، i و j تعداد نقاط واقع در هر یک از مثلث‌های جدید بدست آمده هستند. اگر نقطه‌ای که برای تقسیم مثلث انتخاب میکنیم به میانه نزدیک باشد زمان



اجرای الگوریتم از $O(nlgn)$خواهد بود. در حالتی هم که این نقطه را به صورت تصادفی

انتخاب کنیم زمان اجرای الگوریتم در حالت میانگین از $O(nlgn)$ خواهد بود. (مانند زمان اجرای $quicksort$)