



مدرس: دکتر شهرام خزایی

آنالیز الگوریتم‌ها

تمرین سری چهار

شماره دانشجویی: ۴۰۱۱۰۰۰۷۱

نام و نام خانوادگی: کثرال باغستانی

پرسش ۱

برای حل این سوال آرایه دو بعدی $dp[i][j]$ را در نظر میگیریم و به ترتیب طول زیر رشته‌ها این آرایه را پر میکنیم به چه صورت؟ به این صورت که $dp[i][j]$ برابر است با طول بلندترین زیر آرایه پالیندروم با شروع از کاراکتر i ام رشته و پایان در کاراکتر j ام رشته. حالا قرار میدهیم:

$$dp[i][j] = \max(dp[i, j - 1], dp[i + 1][j], dp[i + 1][j - 1] + \delta_{i,j})$$

و داریم:

$$\delta_{i,j} = \begin{cases} 2 & s[i] = s[j] \\ 0 & \text{else} \end{cases}$$

و اگر زیر فضای مساله‌ها را ماتریس دو بعدی در نظر بگیریم که ستون‌های آن نشان دهنده i و سطرهای آن نشان دهنده j است مساله‌های اولیه عناصر روی قطر هستند که همگی یک و در ادامه قطر بعدی را نیز باید به عنوان زیر مساله پر کرد که اگر آن دو حرف با هم برابر بودند مقدار آن عنصر ۲ و در غیر این صورت ۱ است. و در ادامه عناصر هر قطر از روی عناصر قطر قبل ساخته میشود. حالا که طول بلندترین زیر رشته را پیدا کردیم میتوانیم از جدول برگردیم به عقب و خود زیر آرایه را پیدا کنیم. به چه صورت؟

در هر قدم وقتی ماکسیمم میگیریم یکی از آپشن‌ها به عنوان ماکسیمم انتخاب شده حالا اگر یک پوینتر قرار دهیم که از خانه‌ای که آن را پر کرده به خانه‌ای که به عنوان ماکسیمم انتخاب شده اشاره کنیم و اگر خانه پر شده، ماکسیمم خود را وقتی می‌گرفت که $\delta_{i,j}$ برابر با دو بود (یعنی باعث میشد دو حرف به سرو و ته رشته اضافه کنیم) آن خانه را ستاره دار میکنیم. در نتیجه پس از پر کردن جدول از خانه آخر که شامل کل رشته میشود شروع کرده $(dp[1][size(string)])$ و پوینتر هارا دنبال میکنیم در هر مرحله اگر خانه‌ای که به آن رسیدیم ستاره دار بود ما نیز به سرو و ته رشته‌ای که داریم تشکیل میدهیم آن حرف را اضافه میکنیم. و در آخر رشته خروجی ساخته شده همان زیر آرایه مورد نظر است.

تحلیل زمان اجرا:

پر کردن آرایه دوبعدی ما $O(n^2)$ زمان میبرد و همچنین بازگشت از خانه آخر به عقب و ساختن رشته نیز $O(n)$ است. پس کل زمان اجرای الگوریتم برابر است با $O(n^2)$.

پرسش ۲

ایده اصلی حل این سوال استفاده از DP است. برای اینکار کافیسیت ابتدا یک آرایه جدید n عضوی به نام sum معرفی کنیم. سپس روی آرایه داده شده حرکت کرده و هر خانه sum را به صورت مجموع عناصر خانه اول آرایه اصلی تا آن خانه در آرایه اصلی نگه داری کنیم به این صورت که:

$$sum[i] = sum[i - 1] + A[i];$$

که A آرایه ورودی است.

حالا دو لیست جدید به نام $1/3$ و $2/3$ معرفی میکنیم ابتدا روی sum پیمایش کرده و شماره خانه هایی که sum آنها برابر با $\frac{1}{3}$ است را در آرایه $1/3$ میریزیم. دقت کنید که این اعداد مرتب شده هستند. سپس همین کار را دوباره از ابتدای آرایه sum انجام داده و خانه هایی که sum آنها $\frac{2}{3}$ است را در لیست $2/3$ میریزیم. حالا دو لیست مرتب شده داریم و دو پوینتر i و j را در ابتدای لیست های $1/3$ و $2/3$ می گذاریم. و یکی یکی آنها را جلو میبریم به طریقه الگوریتم ۱.

الگوریتم ۱

```
 $i \leftarrow 0;$   
 $j \leftarrow 0;$   
 $num \leftarrow 0;$   
While end of  $2/3$  or end of  $1/3$  has not been reached  
If  $1/3[i] < 2/3[j]$   
   $num \leftarrow num + size(2/3) - j;$   
   $i \leftarrow i + 1;$   
Else  
   $j \leftarrow j + 1;$   
End While
```

که در نهایت num تعداد i و j ها با شرایط خواسته شده سوال را میدهد.

اثبات درستی:

اگر مقدار $2/3[j]$ بیشتر از $1/3[i]$ باشد آنگاه این یعنی در آرایه اصلی خانه $1/3[i]$ عقب تر از $2/3[j]$ است و همچنین شرایط جمع های خواسته شده سوال را نیز دارد. و چون آرایه مرتب است تمام خانه های بعدی نیز بزرگ ترند و شرایط خواسته شده را دارند و الگوریتم تعداد این خانه ها را به sum اضافه کرده و سراغ i بعدی میرود. حالا اگر چنین نباشد الگوریتم پوینتر j را جلو میبرد تا جایی که این اتفاق برای آن i بیفتد و همچنین چون آرایه $1/3$ نیز مرتب شده است میدانیم برای i های جلوتر نیز شرایط برقرار نیست پس زیاد کردن j مطمئن است.

تحلیل زمانی:

برای محاسبه آرایه های $1/3$ و $2/3$ هزینه $O(n)$ را اعمال کردیم همچنین الگوریتم ۱ در هر مرحله i یا j را یک واحد زیاد میکند که این یعنی ما کسیم $2n$ بار اجرا میشود و در هر بار عملیات $O(1)$ انجام میدهد پس زمان اجرای الگوریتم برابر است با $O(n)$.

پرسش ۳

ابتدا یک گراف کمکی میسازم که راس های آن راس های علامت دار گراف داده شده هستند. سپس از یک راس علامت دار دلخواه شروع کرده و dfs را اجرا میکنم تا به راس علامت دار دیگری برسیم سپس بین این دو راس علامت دار یک یال به وزن فاصله این دو راس اضافه میکنم و همچنین یال های پیمایش شده را در dfs حذف میکنم حالا یک راس علامت دار دیگر انتخاب کرده و در صورتی که مسیری از آن وجود داشت آن را دنبال کرده تا به راس علامت دار دیگری برسیم یا در dfs به خود آن راس برگردم و تمام یال های مسیر را حذف کنم. در نهایت این کار را انقدر ادامه میدهم تا گراف به گراف بدون یال تبدیل شود. حالا در گراف به دست آمده وزن یال ها را در هم ضرب و به عنوان جواب سوال خروجی میدهم.

اثبات درستی:

چونکه گراف داده شده یک درخت است بین هر دو راس علامت دار دقیقا یک مسیر وجود دارد حالا من در گراف کمکی به صورتی دارم راس های علامت دار را به عنوان مولفه همبندی در نظر گرفته و بین دو راس علامت داری که یک مسیر مستقیم وجود دارد (مسیری که در آن راس علامت دار دیگری نیست) یک یال به وزن طول آن مسیر در نظر میگیرم. و درخت بودن گراف اولیه باعث میشود گراف علامت دار جدید نیز درخت باشد. حالا دو راس علامت دار مستقیم را در نظر بگیرید که بین آنها در گراف کمکی یال هست مثلا وزن این یال r است. این یعنی r یال در گراف اصلی بین این دو راس بوده حالا ما یکی از این r یال را انتخاب و حذف کرده و راس های که متصل به راس علامت دار اول میمانند در مولفه همبندی آن قرار میگیرند و بقیه در مولفه همبندی راس دوم قرار میگیرند. با توجه به این که راس های این مسیر باید حتما در یکی از این دو مولفه همبندی قرار می گرفتند و از یک جایی ما باید برش میزدیم و دقیقا r حالت داریم پس با ضرب تمام این وزن ها در هم تمام حالت ها به پوشش داده میشود. (دقت کنید که گراف کمکی صرفا برای درک بهتر مولفه های همبندی بوده و کمک در اثبات درستی وگرنه میتوانستیم مستقیما اعداد بدست آمده را در هم ضرب کنیم).

تحلیل زمانی:

الگوریتم dfs تعمیم یافته ای که اجرا میکنیم در مجموع به ازای هر یال $O(1)$ عملیات انجام میدهد. به صورت دقیق تر یال های هر مسیر را نگه میدارد تا به راس علامت دار اولیه برسد و یا به راس علامت دار دیگری سپس آن یال های نگه داشته شده را از گراف حذف میکند. که این برابر (طول آن مسیر) O زمان میبرد. و در نهایت مجموع همه مسیر ها میشود $O(m+n)$ که چون گراف داده شده درخت است برابر است با $O(n)$ و در نهایت گراف کمکی در بدترین حالت $1-n$ یال دارد که پیمایش روی آنها نیز $O(n)$ است. پس زمان اجرایی کل الگوریتم برابر است با $O(n)$.

پرسش ۴

هر جا سخن از گراف جهت دار بدون دور است ، نام ترتیب توپولوژیکال است که میدرخشد. مساله را با تکنیک برنامه ریزی پویا حل میکنیم به این صورت که پس از بدست آوردن sort topological به ترتیب توپولوژیکال روی راس ها حرکت کرده و طول بلند ترین مسیر به i را قرار میدهیم:

$$longest[i] = \max(longest[j] + 1) \quad \text{for all } j \text{ that we have } (j,i) \text{ in } E$$

اثبات درستی:

چون در ترتیب توپولوژیکال اگر (j,i) عضو E باشد آنگاه حتما j قبل از i در این ترتیب آمده پس با حرکت کردن روی راس ها به ترتیب توپولوژیکال و داشتن $longest$ برای تمام j های کوچک تر از i حتما برای تمام j ها که (j,i) عضو E است مقدار $longest$ را داریم. در نتیجه این مساله حل میشود. و البته اولین زیر مساله برای راس اول هست $longest[u] = 0$ در نهایت روی آرایه $longest$ حرکت کرده و \max میگیریم.

تحلیل زمان اجرا:

اردر توپولوژیکال سورت همان اردر dfs است. $O(n + m)$ و همچنین پیمایش برای پر کردن آرایه $longest$ نیز در مجموع روی هر راس پیمایش کرده و هر یال یک بار در نظر گرفته میشود که در مجموع میشود $O(m + n)$ و همچنین ماکسیمم گرفتن نیز $O(n)$ است. پس در کل زمان اجرای کل الگوریتم برابر است با $O(n + m)$.

پرسش ۵

این سوال را خیلی سال پیش یک انسان به اسم جانسون حل کرده و من صرفا راه حل هوشمندانه او را بازگو میکنم: ابتدا یک راس دلخواه انتخاب کرده و الگوریتم بلمن فورد را از آن راس اجرا میکنیم. این راس را s مینامم. حالا به هر راس i یک عدد p_i نسبت میدهیم که طول کوتاه ترین مسیر از s به آن راس است. گراف جدیدی معرفی میکنیم که همه یال های آن وزن مثبت دارند به صورتی که $l'_{uv} = l_{uv} + p_u - p_v$ می باشد حالا در گراف جدید n بار الگوریتم دایکسترا را اجرا میکنیم تا اینکه کوتاه ترین طول ها را برای تمام جفت راس ها محاسبه کنیم. و در نهایت طول بدست آمده برای هر مسیر را به مقدار واقعی آن تبدیل میکنیم.

اثبات درستی:

ادعا یک:

طول تمام یال ها در گراف جدید مثبت است.

اثبات:

فرض کنیم یال (u,v) را داریم. میدانیم $p_v \leq p_u + l_{uv}$ است در نتیجه داریم:

$$0 \leq l_{uv} + p_u - p_v$$

ادعا دو:

طول کوتاه ترین مسیر از a_1 به a_r تحت این تغییر وزن تغییر نمیکند:

اثبات:

فرض کنیم مسیر a_1, a_2, \dots, a_r را داریم طول جدید این مسیر برابر میشود با:

$$\sum_{i=1}^{r-1} l_{a_i a_{i+1}} + p_{a_i} - p_{a_{i+1}}$$

تمرین سری چهار-۴

که برابر است با طول قبلی این مسیر به اضافه $p_{a_l} - p_{a_r}$ که این یعنی مستقل از مسیر طول تمامی مسیر ها به یک اندازه افزایش یافته و در نتیجه کوتاه ترین مسیر همان قبلی میماند. پس الگوریتم درست کار میکند و کافیست پس از n بار اجرای دایکسترا روی تمام راس ها از طول کوتاه ترین مسیر به دست آمده بین راس های u و v عبارت $p_u - p_v$ را کم کنیم. تحلیل زمانی: زمان اجرای بلمن فورد هست $O(mn)$ و زمان اجرای n بار دایکسترا هست $O(nm \log n)$ و در نهایت محاسبه طول واقعی از طول های بدست آمده برابر است با $O(n^2)$ که زمان الگوریتم میشود $O(nm \log n)$.