به نام خدا

دانشگاه تهران، دانشکده مهندسی برق و کامپیوتر

طراحی و تحلیل الگوریتم‌ها، نیمسال دوم، سال تحصیلی ۹۷-۹۸

حل تمرین سری دوم

---

1. برای حل این سوال باید این نکته را درنظر بگیریم که طولانی‌ترین مسیر بین v_1, v_n یک یال طولانی تر از طولانی‌ترین مسیر بین همسایه‌های v_1 و v_n است. پس تابع بازگشتی را به صورت زیر تعریف میکنیم:

$$L(k) = \begin{cases} 0 & if\ k == n \\ \max\limits_{c \in children(k)} \left(1 + L(c)\right) & otherwise \end{cases}$$

الگوریتم باید آرایه L به طول n را به صورت نزولی پرکند. جواب نهایی مساله در L[1] موجود است.

از آنجایی که برای پرکردن آرایه نیاز داریم به ازای هرراس، تمام همسایه‌های آن را بررسی کنیم، پس مرتبه زمانی الگوریتم از $O(n+m)$ است که در آن n تعداد رئوس گراف و m تعداد یال‌های آن است.

2. $d[i][k]$ را کمترین طول پوشاندن i نقطه اول با k بازه می‌نامیم. جواب نهایی مسئله $d[n][k]$ می‌باشد. این داینامیک به صورت زیر به روزرسانی میشود.

$$d[i][k] = \min(d[i-1][k-1] + 0. d[i-2][k-1] + w(i-1, i). \dots . d[0][k-1] + w(1, i))$$

توضیح: بازه آخر را درنظر می‌گیریم(بازه‌ای که پایان آن سمت راست‌ترین باشد.) فرض می‌کنیم که این بازه t نقطه را پوشش می-دهد(t عددی بین 1 تا i می‌باشد.) بنابراین سایر نقاط که i-t نقطه اول می‌باشند را باید با k-1 بازه پوشاند. منظور از w(I, j) فاصله بین نقاط i, j است.

3. فرض کنیم این n عدد که $A_1, A_2, \dots, A_n$ نام دارند، در بازه 0 تا k قرار دارند. $P(i,j)$ را به صورت زیر تعریف می‌کنیم:

$$\text{P(i, j)} = \begin{cases} 1 & if\ some\ subset\ of\ \{A_1, \dots, A_i\}\ has\ a\ sumf\ of\ i \\ 0 & otherwise \end{cases}$$

یعنی اگر با i عدد اول از این n عدد بتوان مقدار j را ساخت، مقدار این تابع برابر با یک و درغیر این صورت مقدار آن صفر است. رابطه‌ی بازگشتی آن را به صورت زیر می‌توان نوشت:

$$\text{P(i, j)} = \begin{cases} 1 & if\ P(i-1,\ j) = 1\ or\ P(i-1, j-A_i) = 1 \\ 0 & otherwise \end{cases}$$

لذا

$$P(i,j) = \max\{P(i-1,j), P(i-1, j-A_i)\}$$

حال به حل مساله اصلی برمی گردیم، فرض کنید S به صورت زیر تعریف شود:

$$S = \left(\sum_{i=1}^{n} A_i \right) / 2$$

اگر تعدادی از $A_1, A_2, \dots, A_n$ پیدا کنیم که مجموع آنها S باشد، به این معنی است که توانسته‌ایم دو مجموعه را طوری تقسیم کنیم که اختلاف جمع آنها صفر شود. این ایده آل‌ترین جواب ممکن است. اما اگر چنین امکانی وجود نداشته باشد، باید این اختلاف را کمینه کنیم. تعریف میکنیم:

$$M = \min_{i \le S} \{ S - i : \ P(n, i) = 1 \}$$

می توان نشان داد که :

$$|S_1 - S_2| = 2S - 2i = 2(S - i) = 2M$$

بنابراین با محاسبه 2M، جواب نهایی مساله بدست می آید.

هزینه الگوریتم نیز $O(n^2k)$ است، زیرا به تعداد $O(n^2k)$ تا $P(i,j)$ وجود دارد. S نیز از مرتبه nk است، محاسبه هر M نیز هزینه O(nk) دارد.

4. میزان باحال بودن شخص v را $coolness(v)$ تعریف میکنیم. برای هر شخص v، دو پارامتر نیز تعریف میکنیم:

$$T[v] = maximum\ coolness\ of\ subtree\ that\ v\ is\ it's\ root, where\ the\ person\ v\ is\ attending$$

$$T[v] = maximum\ coolness\ of\ subtree\ that\ v\ is\ it's\ root, where\ the\ person\ v\ is\ not\ attending$$

جواب نهایی مساله نیز $\max\{F[boss], T[boss]\}$ است که $boss$ رئیس کل است. برای پر کردن پارامترهای T, F برای هر شخص v از روابط زیر استفاده میکنیم:

$$T[v] = coolness(v) + \sum_{i=1}^{k} F[A_k] \qquad where\ \ A_1, \dots, A_k\ are\ the\ direct\ childs\ of\ v\ in\ graph$$

$$F[v] = \sum_{i=1}^{k} \max\{T[A_k], F[A_k]\} \qquad where\ \ A_1, \dots, A_k\ are\ the\ direct\ childs\ of\ v\ in\ graph$$

که این مقدار برای هر گره از گراف یک بار محاسبه می شود، هزینه زمانی الگوریتم و پیچیدگی حافظه O(n) است. توجه داشته باشید که برای محاسبه مقادیر V, T برای یک راس، ابتدا باید مقادیر V, T فرزندان آن راس حساب شده باشد. برای انجام این کار، می‌توان از پیمایش DFS بر روی درخت استفاده کرد. شبه کد زیر این کار را انجام میدهد:

```
DFS(node v){
        if (v is leaf) {
                F[v] = 0;
                T[v] = coolness(v);
        }
        else {
                for each (u child of v) { DFS(u); }
                T[v] = coolness(v) + sum of (F[u] for each u child of v);
                F[v] = sum of (max{F[u], T[u]} for each u child of v);
        }
}
```

5. تعریف میکنیم d[n][l][h] تعداد درخت‌های متوازن با ارتفاع h است که دقیقا n گره و l برگ دارند. برای یافتن این مقدار، باید تعداد حالت‌هایی که برای دو زیردرخت متصور هستیم را درهم ضرب کنیم.

$$d[n][l][h] = sum\begin{cases}\sum_{p=0}^{n}\sum_{t=0}^{n} d[t][p][h-1] * d[n-t][l-p][h-1] \\ \sum_{p=0}^{n}\sum_{t=0}^{n} d[t][p][h-2] * d[n-t][l-p][h-1] \\ \sum_{p=0}^{n}\sum_{t=0}^{n} d[t][p][h-1] * d[n-t][l-p][h-2]\end{cases}$$

جواب نهایی مسئله نیز به ازای n گره و l برگ برابراست با$\sum_h d[n][l][h]$. حالت اولیه نیز $d[1][1][1] = d[0][0][0] = 1$ است.

6. زیرمسئله را به این صورت تعریف میکنیم:

A[n][i][j]:

اگر تعداد n موشک‌انداز داشته باشیم وازسمت شرق(راست) i عدد از آن‌ها دیده‌شود و از غرب(چپ) j عدد دیده شود، چند جایگشت برای موشک‌اندازها می‌توان متصور شد. از آن‌جایی که در فرض مسئله گفته شده که ازهر ارتفاع موشک‌انداز تنها یک عدد در پایگاه موجود است، درمی‌یابیم که ارتفاع موشک‌اندازها متفاوت است. برای حل مسئله برای یک جایگشت، کوتاه‌ترین موشک‌انداز را درنظر میگیریم. این موشک‌انداز یا شرقی‌ترین موشک‌انداز است یا غربی‌ترین موشک‌انداز یا اینکه درمیان صف قرار گرفته‌است. بنابراین داریم:

A[n][i][j] = sum$\begin{cases}\text{کوتاهترین در شرق}: A[n-1][i-1][j] \\ \text{کوتاهترین در غرب}: A[n-1][i][j-1] \\ \text{کوتاهترین در میان}: A[n-1][i][j]\end{cases}$

زیرا وقتی شرقی‌ترین است، جایگشتی مشابه با حالتی دارد که $n-1$ موشک‌انداز داریم و $j-1$ موشک‌انداز از مشرق دیده می شود و j تا از غرب که تنها یک موشک‌انداز کوتاه به شرق آن اضافه شدهاست. در غربی‌ترین حالت نیز به همین‌صورت می‌توان استدلال نمود. در شرایطی که این موشک‌انداز در میانه صف باشد، حالتی اس که در یکی از $n-2$ جای خالی میان $n-1$ موشک‌انداز قرارگرفته است که i تای آنها از شرق و j تای آنها از غرب دیده میشود. حالت اولیه ی مسئله را ,A[n][1][n]=1 A[n][n][1]=1 درنظر میگیریم. هزینه زمانی این الگوریتم و هزینه حافظه آن O(n^3) است.