به نام خدا

دانشگاه تهران، دانشکده مهندسی برق و کامپیوتر
تحلیل و طراحی الگوریتم‌ها

تمرین کتبی دوم (برنامه‌نویسی پویا)
موعد تحویل: دوشنبه ۱۶ فروردین ۱۴۰۰، ساعت ۲۳:۵۹
طراح: دانشور امراللهی (amrollahi.daneshvar@gmail.com)

لطفا در تمامی سوالات تعریف آرایه/ماتریس DP خود، مقداردهی اولیه آن، نحوه آپدیت شدن آن از دیگر مقادیر آرایه/ماتریس و نحوه محاسبه جواب نهایی مسئله از روی آن را بنویسید

۱. طولانی‌ترین زیر دنباله مشترک رشته‌های ABACAAC و BACBAAC را با استفاده از روش برنامه‌نویسی پویا به دست آورید. جدول d که $d_{i,j}$ نشان دهنده طول جواب به ازای i حرف اول رشته اول و j حرف اول رشته دوم است و همچنین جدول par که $par_{i,j}$ نشان می‌دهد $d_{i,j}$ از روی چه خانه‌ای آپدیت شده را نمایش دهید و به کمک par خود رشته جواب را نیز بنویسید. (۱۰ نمره)

راه حل) ماتریس d به شکل زیر است:

0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1
0	1	1	1	2	2	2	2
0	1	2	2	2	3	3	3
0	1	2	3	3	3	3	4
0	1	2	3	3	4	4	4
0	1	2	3	3	4	5	5
0	1	2	3	3	4	5	6

برای نمایش درایه (i, j) ماتریس par از مقادیر قراردادی زیر استفاده می‌کنیم:

- D: از روی $(i-1, j-1)$ آپدیت شده
- U: از روی $(i-1, j)$ آپدیت شده
- L: از روی $(i, j-1)$ آپدیت شده
- A: هرکدام از حالات ۲ و ۳ می‌توانسته رخ دهد

-	-	-	-	-	-	-	-
-	A	D	L	L	D	D	L
-	D	A	A	D	L	L	L
-	U	D	L	A	D	D	L
-	U	U	D	L	A	A	D
-	U	D	U	A	D	D	A
-	U	D	U	A	D	D	L
-	U	U	D	A	U	U	D

در صورتی برای نمایش par از قرارداد دیگری (مثل فلش) استفاده کرده باشید یا به جای حالت A هرکدام از حالات U یا L را نیز نوشته باشید نمره تعلق می‌گیرید.

برای یافتن خود رشته جواب از روی par کافی است خانه پایین راست شروع به حرکت کنیم و متناسب با حرف نوشته شده در جدول حرکت کنیم.

- D: قطری به بالا چپ برو و کاراکتر آخر رشته‌ها را مچ کن
- U: به خانه بالا برو و کاراکتر آخر رشته اول را خط بزن
- L: به خانه چپ برو و کاراکتر آخر رشته دوم را خط بزن
- A: هر کدام از حرکات L یا U را به دلخواه انجام بده.

با این روش به جواب BACAAC می‌رسیم.

۲. به یک رشته متقارن می‌گوییم اگر با برعکس خودش برابر باشد. برای مثال رشته $madam$ یک رشته متقارن است. یک رشته به طول n به شما داده می‌شود. شما بایستی طول طولانی‌ترین زیررشته (تعدادی حرف متوالی در رشته) متقارن رشته ورودی را با پیچیدگی زمانی $O(n^2)$ به دست آورید. (۱۰ نمره)
راه حل) رشته ورودی را s می‌نامیم. ماتریسی با مقادیر boolean (صفر و یک) به این شکل تعریف می‌کنیم:

$$d_{i,j} = \begin{cases} 1 & s[i..j] \;\; is \;\; palindrome \\ 0 & elsewhere \end{cases}$$

می‌دانیم رشته $s[i..j]$ متقارن است اگر و تنها اگر $s[i] = s[j]$ و همچنین $s[i+1..j-1]$ متقارن باشد.

بنابراین برای تشخیص متقارن بودن $s[i..j]$ می‌توانیم از متقارن بودن/نبودن رشته‌های با طول کوتاه‌تر استفاده کنیم. بنابراین می‌توان روی طول رشته‌ها از ۳ تا n که n طول رشته است حلقه زد و متقارن بودن تمام رشته‌های به طول مشخصی را تشخیص داد.

```
for (int k = 3 ; k <= n ; k++)
{
        int i = 1, j = k;
        while (j <= n)
        {
                if (s[i] == s[j] && d[i + 1][j - 1] == 1)
                        d[i][j] = 1;
                else
                        d[i][j] = 0;
                i++, j++;
        }
}
```

برای حالت پایه نیز کافی است رشته‌های به طول ۱ و ۲ را مقداردهی اولیه کنیم:

$$d_{i,i} = 1$$

$$d_{i,i+1} = \begin{cases} 1 & s[i] = s[i+1] \\ 0 & elsewhere \end{cases}$$

اگر s_i و s_j برابر باشند، $d_{i,j}$ تنها در صورتی ۱ می‌شود که $d_{i+1,j-1}$ نیز ۱ باشد. در غیر اینصورت $d_{i,j}$ برابر صفر است.

بین تمامی $d_{i,j}$ هایی که ۱ هستند کافی است $j-i+1$ را با جواب ماکسیمم بگیریم تا طول طولانی‌ترین زیررشته متقارن به دست آید.

۳. یک جدول $n \times 2$ از اعداد حقیقی به شما داده شده است. می‌خواهیم تعدادی خانه دو به دو نامجا ور (ضلع مشترک نداشته باشند) انتخاب کنیم به طوری که مجموع اعداد آن‌ها بیشینه شود. راه حلی با پیچیدگی زمانی $O(n)$ ارائه دهید تا این بیشترین حاصل جمع دست‌یافتنی را خروجی دهد. (شبه‌کد را با روش بازگشتی حافظه‌دار (memoization) بنویسید) (۱۰ نمره)

راه حل) فرض می‌کنیم ورودی در قالب آرایه‌ای دوبعدی به اسم x داده شود که $x_{i,j}$ نشان دهنده عدد واقع در سطر $0 \leq i \leq n-1$ و ستون $0 \leq j \leq 1$ باشد.

سه تا آرایه a, b, c به شکل زیر تعریف می‌کنیم:

- a_i: بیشترین حاصل جمع دست‌یافتنی با استفاده از i ستون اول x به طوری که از ستون i عدد بالایی انتخاب شود.

- b_i: بیشترین حاصل جمع دست‌یافتنی با استفاده از i ستون اول x به طوری که از ستون i عدد پایینی انتخاب شود.
- c_i: بیشترین حاصل جمع دست‌یافتنی با استفاده از i ستون اول x به طوری که از ستون i هیچ عددی انتخاب نشود.

مقادیر آرایه‌های بالا به ازای ستون i از روی ستون‌های $i-1$ به شکل زیر آپدیت می‌شوند:

$$a_i = max(b_{i-1}, c_{i-1}) + x_{0,i}$$
$$b_i = max(a_{i-1}, c_{i-1}) + x_{1,i}$$
$$c_i = max(c_{i-1}, a_{i-1}, b_{i-1})$$

در توضیح حالت اول داریم: اگر از ستون i بخواهیم عدد بالایی را انتخاب کنیم، از ستون قبلی یا عدد پایینی را انتخاب کرده‌ایم یا هیچ عددی انتخاب نکرده‌ایم.

حالات دوم و سوم نیز با استدلال‌های مشابه به دست آمده‌اند.

```
int solve(int i, int p)
{
        if (p == 1) //Calculate a[i]
        {
                if (a[i] != NULL) //Already Calculated
                        return a[i];
                if (i == 0)
                {
                        a[0] = x[0][0];
                        return a[0];
                }
                a[i] = max(solve(i - 1, 2), solve(i - 1, 3)) + x[0][i];
                return a[i];
        }

        if (p == 2) //Calculate b[i]
        {
                if (b[i] != NULL) //Already Calculated
                        return b[i];
                if (i == 0)
                {
                        b[0] = x[1][0];
                        return b[0];
                }
                b[i] = max(solve(i - 1, 1), solve(i - 1, 3)) + x[1][i];
                return b[i];
        }

        if (p == 3) //Calculate c[i]
        {
                if (c[i] != NULL) //Already Calculated
                        return c[i];
                if (i == 0)
                {
                        c[0] = 0;
                        return c[0];
                }
                c[i] = max(solve(i - 1, 3), max(solve(i - 1, 1), solve(i - 1, 2)));
                return c[i];
        }
}
```

جواب برابر $max(a_n, b_n, c_n)$ است.

برای حالت پایه نیز کافی است جدولی با ۱ ستون در نظر بگیریم که در if های بالا واضح است به چه صورت مقداردهی می‌شوند.

۴. به دنباله $a_1, a_2, ..., a_l$ زیبا می‌گوییم اگر:

- $1 \leq a_1 \leq a_2 \leq ... \leq a_l \leq n$

- به ازای هر i که $1 \leq i \leq n-1$ داشته باشیم: $a_i \mid a_{i+1}$

به عبارتی هر دنباله غیرنزولی از اعداد 1 تا n به طوری که هر عدد بر عدد قبلی خود بخشپذیر باشد را زیبا می‌نامیم.
تعداد دنباله‌های زیبا به طول k با استفاده از اعداد 1 تا n را با پیچیدگی زمانی‌های

(آ) $O(nk\sqrt{n})$ (۵ نمره)
(ب) $O(nk\log(n))$ (۱۰ نمره)

به دست آورید (برای بخش ب شبه‌کد بنویسید)

راه حل) $d_{i,j}$ را تعریف می‌کنیم: تعداد دنباله‌های زیبا به طول i مختوم به عدد j.

یک دنباله زیبا به طول i و مختوم به عدد j با افزودن عدد j به انتهای یک دنباله زیبا به طول $i-1$ به دست می‌آید که عدد آخر آن دنباله کوتاه‌تر، یکی از مقسوم‌علیه‌های j باشد.

بنابراین برای بخش آ کافی است با یک حلقه i را فیکس کنیم، با یک حلقه j را فیکس کنیم و نهایتا روی مقسوم‌های j حلقه بیندازیم که کافی است از ۱ تا $\sqrt{j}$ پیش رویم تا تمام مقسوم‌علیه‌های j را به دست آوریم.

$$d_{i,j} = \sum_k d_{i-1,k} \text{ where } k \text{ is a divisor of } j$$

برای بخش ب داریم:

```
for (int i = 2 ; i <= k ; i++)
        for (int j = 1 ; j <= n ; j++)
                for (int k = j ; k <= n ; k += j)
                        d[i][k] += d[i - 1][j];
```

دو حلقه درونی پیچیدگی زمانی $O(n.log(n))$ دارند. بنابراین پیچیدگی زمانی اجرای کل آن برابر $O(nklog(n))$ است.
برای حالات پایه کافی است $d_{1,j} = 1$ را به ازای هر j مقداردهی کنیم.

۵. به شما رشته‌ای از حروف به طول n داده می‌شود. حداقل تعداد کاراکترهایی که باید به این رشته اضافه کنید (اضافه کردن حرف جدید به هرجایی از رشته مجاز است) تا این رشته متقارن (باهمان تعریف سوال ۲) شود را با پیچیدگی زمانی $O(n^2)$ به دست آورید. (در این سوال پیچیدگی حافظه شما باید از $O(n)$ باشد) (۱۵ نمره)

راه حل) ابتدا سوال را با پیچیدگی حافظه $O(n^2)$ حل می‌کنیم.

رشته ورودی را s بنامید. فرض کنید $f(i,j)$ تعداد کمینه حرکت‌های ممکن برای متقارن‌سازی $s[i..j]$ باشد. فرض کنید میخوایم رشته $s[i..j]$ را با افزودن حداقل تعداد کاراکتر متقارن کنیم. اگر $s[i] = s[j]$ کافی است $f(i+1, j-1)$ عملیات انجام دهیم. در غیر اینصورت حتما نیاز به افزودن کاراکتری جدید داریم. دو حالت داریم:

- کاراکتری مشابه s_i اضافه کنیم: تعداد عملیات برابر است با:

$$f(i, j-1) + 1$$

- کاراکتری مشابه s_j اضافه کنیم: تعداد عملیات برابر است با

$$f(i+1, j) + 1$$

این تابع بازگشتی را می‌توان با استفاده از روش بازگشتی حافظه‌دار پیاده‌سازی کرد به طوری که ماتریس DP همان $f(i,j)$ باشد. پیچیدگی زمانی و حافظه این راه حل از $O(n^2)$ است.

به این نکته توجه کنید که برای متقارن‌سازی رشته‌های به طول k تنها باید حداقل تعداد عملیات متقارن سازی $k-1$ و $k-2$ را بدانیم. تعریف ماتریس DP خود را به این شکل ارائه می‌دهیم:

$$d_{len,i} = minimum\ \ number\ \ of\ \ insertions\ \ to\ \ make\ \ s[i..i+len-1]\ \ a\ \ palindrome$$

با تعریف ماتریس DP به این شکل، به پیاده‌سازی زیر می‌رسیم:

```
for (int len = 2 ; ; len <= n ; len++)
    for (int i = 0 ; i < n ; len++)
        d[len][i] = INF
    for (int i = 0 ; i + len - 1 < n ; i++)
        if (s[i] == s[i + len - 1])
            d[len][i] = d[len - 2][i + 1];
        else
            d[len][i] = min(d[len - 1][i], d[len - 1][i + 1]) + 1
```

همان‌طور که مشاهده می‌کنید $d_{len,i}$ تنها از روی $d_{len-1,j}$ و $d_{len-2,j}$ آپدیت می‌شود.
بنابراین می‌تواند ابعاد درایه اول ماتریس DP را به ۳ کاهش داد و کد بالا را دوباره پیاده‌سازی کرد:

```
for (int len = 2 ; ; len <= n ; len++)
    for (int i = 0 ; i < n ; len++)
        d[2][i] = INF
    for (int i = 0 ; i + len - 1 < n ; i++)
        if (s[i] == s[i + len - 1])
            d[2][i] = d[0][i + 1];
        else
            d[2][i] = min(d[1][i], d[1][i + 1]) + 1

    swap(d[1], d[0]);
    swap(d[1], d[2]);
```

با swap های بالا مقادیر $d_{0,j}$ و $d_{1,j}$ در گام بعدی حلقه بیرونی، برابر $d_{1,j}$ و $d_{2,j}$ در گام فعلی حلقه خواهند بود که همین مورد نظر است.
برای مقداردهی اولیه ماتریس‌های DP رشته‌های به طول صفر و یک خود به خود متقارن هستند پس درایه‌های متناظر آن‌ها باید صفر باشد.

$$d_{0,j} = d_{1,j} = 0$$

۶. تعداد n توپ در یک ردیف کنار هم چیده‌شده‌اند. توپ iم رنگ c_i دارد که $0 \leq c_i \leq m$. رنگ‌ها از 1 تا m شماره‌گذاری می‌شوند و $c_i = 0$ به معنای رنگ‌نشده بودن توپ i است. برای این که توپ رنگ نشده i را به رنگ j دربیاوریم باید $p_{i,j}$ واحد هزینه کنیم.
زیبایی یک دنباله از توپ‌ها را تعریف می‌کنیم: حداقل تعداد گروه‌های متوالی از توپ‌ها که بتوان همه n توپ را به آن افراز کرد به طوری که اعضای هر گروه همرنگ باشد. برای مثال اگر دنباله رنگ‌ها به شکل $1, 2, 2, 7, 1, 1, 1$ باشد زیبایی این دنباله برابر با 4 است زیرا حداقل تعداد گروه‌ها به شکل رو به رو است: $[1], [2, 2], [7], [1, 1, 1]$.
اگر بخواهیم توپ‌های رنگ نشده را رنگ کنیم به طوری که زیبایی دنباله نهایی دقیقا برابر k باشد، راه حلی با پیچیدگی زمانی $O(nkm^2)$ ارائه دهید تا کمترین هزینه را پیدا کند. (۲۰ نمره)

راه حل) مقدار $d_{i,j,a}$ را تعریف می‌کنیم: کمترین هزینه برای رنگ کردن i توپ اول که زیبایی آن‌ها j شود و همچنین رنگ توپ آخر یعنی iم برابر a باشد.

فرض کنید می‌خواهیم مقدار $d_{i,j,a}$ را آپدیت کنیم. روی رنگ نشده بودن/نبودن توپ آخر و همچنین رنگ توپ ما قبل آن حالت‌بندی می‌کنیم:

- توپ i رنگ نشده‌باشد: در این صورت دو حالت داریم:
  - $i-1$ توپ اول را با زیبایی j رنگ کنیم به طوری که رنگ توپ آخر یعنی $i-1$م نیز a باشد. سپس رنگ توپ i را a کنیم. در این حالت خرج برابر است با:

$$d_{i-1,j,a} + p_{i,a}$$

  - $i-1$ توپ اول را با زیبایی $j-1$ رنگ کنیم به طوری که رنگ توپ آخر یعنی $i-1$م برابر رنگی به غیر از a مانند b باشد. سپس رنگ توپ i را a کنیم. در این حالت خرج برابر است با:

$$d_{i-1,j-1,b} + p_{i,a}$$

- توپ i رنگ شده باشد: در این صورت دو حالت داریم:
  - $i-1$ توپ اول را با زیبایی j رنگ کنیم به طوری که رنگ توپ آخر یعنی $i-1$م برابر c_i باشد.
در این حالت خرج برابر است با:

$$d_{i-1,j,c_i}$$

  - $i-1$ توپ اول را با زیبایی j رنگ کنیم به طوری که رنگ توپ آخر یعنی $i-1$م رنگی به غیر از a مانند b باشد. در این حالت خرج برابر است با:

$$d_{i-1,j-1,b}$$

بسته به این که توپ iم رنگ شده باشد یا نشده باشد، بین ۲ زیر حالتی که به وجود می‌آید مینیمم می‌گیریم.

جواب برابر است کمینه مقدار $d_{n,k,i}$ به ازای i های مختلف. درواقع حالت‌بندی می‌کنیم که رنگ توپ آخر چه باشد.
برای مقدار اولیه‌های $d_{i,j,a}$ داریم:

- توپ اول رنگ نشده باشد یعنی $c_i = 0$:

$$d_{1,1,a} = p_{1,a}$$

به ازای تمام a های مختلف از ۱ تا m

- توپ اول رنگ شده باشد:

$$d_{1,1,c_1} = 0$$

۷. در یک رستوران n نوع غذا موجود است. اریک می‌خواهد دقیقا m تا از غذاها را بخورد که $m \leq n$ (از هر نوع غذا حداکثر ۱ بار می‌تواند بخورد). اریک می‌داند غذای iم به خوشحالی او اندازه a_i واحد اضافه می‌کند همچنین k قانون به این فرم وجود دارد که:
اگر غذای x_i را دقیقا قبل از y_i بخورد (بین این دو وعده نباید غذای دیگری بخورد)، آنگاه خوشحالی او به اندازه c_i واحد علاوه بر مقدار اشاره شده در بالا، اضافه خواهد شد. راه حلی با پیچیدگی زمانی $O(2^n n^2)$ پیدا کنید تا حداکثر خوشحالی که اریک می‌تواند به دست آورد را محاسبه کند. (راهنمایی: برای حل این سوال می‌توانید از ایده DP Bitmask استفاده کنید. برای آشنایی با این ایده اینجا کلیک کنید) (۲۰ نمره)

راه حل) متغیر $mask$ را یک عدد n بیتی درنظر می‌گیریم که بیت‌های ۱ آن نشان‌دهنده غذاهایی هستند که تا الان خورده‌ایم. مقدار $dmask, i$ را تعریف می‌کنیم: حداکثر خوشحالی که می‌توان با غذاهایی که بیت متناظرشان درون $mask$ یک است به دست آوریم به طوری که غذای آخری که میل کردیم غذای شماره i باشد.

ابتدا برای راحتی کار $p_{i,j}$ را تعریف می‌کنیم مقدار خوشحالی افزوده‌ای که به ازای خوردن غذای j دقیقا بعد از غذای i به دست می‌آید.
به عبارتی:

$$p_{x_i,y_i} = c_i$$

فرض کنید می‌خواهیم $d_{mask,last}$ را آپدیت کنیم. ابتدا باید تضمین کنیم که بیت شماره $last$ از $mask$ برابر ۱ است. سپس باید یک بیت ۱ دیگر به جز $last$ از $mask$ پیدا کنیم. فرض کنید آن بیت، بیت iم باشد.
مقدار $nmask$ را تعریف می‌کنیم:

$$nmask = mask - 2^i$$

حال می‌توانیم $d_{mask,last}$ را از روی $d_{nmask,i}$ آپدیت کنیم.

یعنی فرض می‌کنیم تا الان غذاهای متناظر با $nmask$ را خورده‌ایم به طوری که آخرین آن‌ها i بوده. حالا پس از آن قصد داریم غذای شماره $last$ را بخوریم.

می‌دانیم به خاطر خوردن غذای $last$ مقدار a_{last} واحد به خوشحالی او اضافه می‌شود. همچنین چون غذای $last$ را بعد از غذای i خورده، دوباره خوشحالی او به اندازه $p_{i,last}$ واحد اضافه می‌شود.

آپدیت به این شکل انجام می‌شود:

$$dp_{mask,last} = max(dp_{mask,last},\ \ dp_{nmask,i} + a_{last} + p_{i,last})$$

به شبه‌کد زیر توجه کنید:

```
for (int mask = 1 ; mask < (1 << n) ; mask++)
    for (int last = 0 ; last < n ; last++)
    {
            if (!(mask & (1 << last))) continue;
            for (int i = 0 ; i < n ; i++)
            {
                if ( (!(mask & (1 << i)))  || (i == last) ) continue;
                int nmask = mask - (1 << last);
                dp[mask][last] = max(dp[mask][last], dp[nmask][i] + a[last] + c[i][last]);
            }

    }
```

برای حالات پایه باید $mask$ هایی که دقیقا ۱ بیت ۱ دارند را مقداردهی کنیم.

$$d_{2^i,i} = a_i$$

جواب نهایی برابر است با ماکسیمم مقدار $d_{mask,i}$ هایی که تعداد بیت‌های ۱ از $mask$ دقیقا برابر m باشد.