به نام خدا

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

پاسخ تمرین کتبی دوم

طراح: آرمان رستمی، arman.rostami.999@gmail.com

---

۱. فرض می‌کنیم نام مجموعه داده‌شده A است و n عضو دارد و اعداد را نیز با شروع از ۰ در نظر می‌گیریم. از آن‌جایی که می‌خواهیم دو بخش شکسته‌شده مجموع یکسانی داشته‌باشند باید مجموع هر کدام از آن‌ها برابر نصف مجموع کل اعضای مجموعه داده‌شده باشد. پس اگر مجموع اعضای مجموعه فرد باشد مسئله جواب ندارد، در غیر این صورت به یافتن اعضا با حاصل‌جمع نصف حاصل‌جمع کل می‌پردازیم.

به این منظور dp را آرایه‌ای از صفر و یک‌ها با اندازه $(n+1) \times (\frac{sum}{2}+1)$ تعریف می‌کنیم که در آن $dp[i][j]$ اگر یک باشد به این معنی است که زیرمجموعه‌ای از اعضای صفرم تا $i-1$ام مجموعه وجود دارد که جمعی برابر j داشته‌باشد و در غیر این صورت وجود ندارد. همچنین sum حاصل‌جمع کل اعضای مجموعه است.

مشخص‌کننده جواب نهایی مسئله $dp[n][\frac{sum}{2}]$ است که اگر یک شود به معنای وجود داشتن جواب برای مسئله است.

برای آپدیت‌کردن dp در خانه iام سعی می‌کنیم با قراردادن یا قرارندادن عضو iام به حاصل جمع مربوطه برسیم. پس داریم:

$$dp[i][j] = dp[i-1][j] \vee dp[i-1][j-A[i-1]]$$

که این معادله برای $i > 0$ و $j > 0$ درست است و همچنین $j - A[i-1]$ باید بزرگتر مساوی صفر باشد تا بتوان آن را در نظر گرفت.

برای حالت‌های پایه دو حالت داریم. وقتی هیچ عضوی انتخاب نشود و به دنبال حاصل جمعی به‌جز صفر باشیم جواب وجود ندارد یعنی داریم:

for $j > 0$ and $j \leq \frac{sum}{2}$:

$$dp[0][j] = 0$$

وقتی به به دنبال حاصل جمع صفر هستیم می‌توان عضوی را انتخاب نکرد و در این حالت جواب وجود دارد:

for $i \geq 0$ and $i \leq n$:

$$dp[i][0] = 1$$

حال برای جمع‌بندی بخش‌های قبل برای پرکردن dp داریم:

```
for j = 1 to sum / 2:
    dp[0][j] = 0

for i = 0 to n:
    dp[i][0] = 1

for i = 1 to n:
    for j = 1 to sum / 2:
        dp[i][j] = dp[i - 1][j]
        if j - A[i - 1] >= 0:
            dp[i][j] = dp[i][j] || dp[i - 1][j - A[i - 1]]
```

حال در صورتی که $dp[n][\frac{sum}{2}]$ یک باشد مسئله جواب دارد و برای یافتن دو بخش مربوطه، دو مجموعه $S1$ و $S2$ را در نظر می‌گیریم. مجموعه $S1$ شامل عناصری از مجموعه است که در ایجاد حاصل جمع مربوطه اثر داشته‌اند و مجموعه $S2$ شامل عناصری است که در ایجاد حاصل جمع مربوطه نقش نداشته‌اند. در هر مرحله $dp[i-1][k]$ را چک می‌کنیم که k حاصل جمعی است که به دنبال ساخت آن هستیم. اگر این مقدار برابر یک بود، عنصر فعلی در ساخت جمع k اثر نداشته و آن را به مجموعه $S2$ اضافه می‌کنیم. در غیر این صورت آن را به مجموعه $S1$ اضافه و در ادامه به دنبال جمع $k - A[i-1]$ می‌گردیم. به طور دقیق‌تر داریم:

```
i = n
k = sum / 2

while i > 0 and k >= 0:
        if dp[i - 1][k]:
                i--;
                S2.add(A[i])

        else if dp[i - 1][k - A[i - 1]]:
                i--;
                k -= A[i];
                S1.add(A[i])
```

در نهایت مجموعه‌های $S1$ و $S2$ جواب‌ها برای دو بخش مربوطه هستند. این راه‌حل دارای پیچیدگی زمانی و میزان حافظه مصرفی $O(n \times \frac{sum}{2})$ است که با توجه به عامل sum راه‌حلی چندجمله‌ای نیست.

۲. فرض می‌کنیم نام رشته داده‌شده S و طول آن n است و اعداد را نیز با شروع از ۰ در نظر می‌گیریم. dp را آرایه‌ای از اعداد صحیح با مقادیر اولیه ۰ و اندازه $n \times n$ تعریف می‌کنیم که در آن $dp[i][j]$ طول بلندترین زیردنباله پالیندروم بین کاراکترهای iام تا jام (با در نظرگرفتن کاراکترهای iام و jام) است.

جواب نهایی مسئله طبق تعریف $dp[0][n-1]$ است.

برای پرکردن dp به طور بازگشتی ۳ حالت داریم:

اگر $A[i]$ و $A[j]$ که اولین و آخرین کاراکتر در زیررشته‌ای است که داریم آن را بررسی می‌کنیم برابر نباشند، زیررشته نمی‌تواند پالیندروم باشد پس باید دو حالت جواب برای حالتی که کاراکتر آخر را در نظر نمی‌گیریم و حالتی که کاراکتر اول را در نظر نمی‌گیریم را بدست آورده و حاصل بیشینه آن‌هاست یعنی داریم:

if S[i] $\neq$ S[j]:

$$dp[i][j] = max(dp[i+1][j], dp[i][j-1])$$

اگر زیررشته‌ای که داریم آن را بررسی می‌کنیم به طول ۲ باشد و دو کاراکتر آن برابر باشند جواب نیز به طول ۲ است و داریم:

if j = i + 1 and S[i] == S[j]:

$$dp[i][j] = 2$$

در نهایت اگر زیررشته طول بیشتر از ۲ داشته باشد و کاراکترهای اول و آخر آن یکسان باشند جواب را برای کارکترهای میانی بدست می‌آوریم و آن را با ۲ جمع می‌کنیم که به منظور برابر بودن کاراکترهای اول و آخر ایجاد شده. پس داریم:

if j - i > 1 and S[i] == S[j]:

$$dp[i][j] = dp[i+1][j-1] + 2$$

برای حالت پایه حالتی را داریم که در آن طول رشته یک باشد که در این صورت بلندترین زیردنباله پالیندروم آن نیز طول یک دارد. پس داریم:

for i = 0 to n - 1:

$$dp[i][i] = 1$$

در نهایت برای پرکردن dp داریم:

```
for i = 0 to n − 1:
        dp[i][i] = 1

for i = n − 1 to 0:
        for j = i + 1 to n − 1:
                if j = i + 1 and S[i] == S[j]:
                        dp[i][j] = 2
                else if S[i] == S[j]:
                        dp[i][j] = dp[i + 1][j − 1] + 2
                else:
                        dp[i][j] = max(dp[i][j − 1], dp[i + 1][j])
```

راه‌حل بالا دارای پیچیدگی زمانی و میزان حافظه مصرفی $O(n^2)$ است. می‌توان حافظه مصرفی آن را بهبود بخشید. به این منظور از آرایه یک‌بعدی dp به اندازه n شامل اعداد صحیح استفاده می‌کنیم.

همانطور که مشاهده می‌شود در راه‌حل قبلی $dp[i][j]$ به ۳ مقدار از dp برای آپدیت‌شدن وابسته است. مقدار $dp[i][j-1]$ حاصل آخرین محاسبه است و می‌توانیم مقدار آن را داشته‌باشیم، مقدار $dp[i+1][j]$ نیز قبلا محاسبه شده و تنها برای آپدیت $dp[i][j]$ به آن نیاز داریم و در ادامه استفاده‌ای از آن نمی‌شود. اما مقدار $dp[i+1][j-1]$ را مستقیما نمی‌توانیم توسط آرایه یک‌بعدی بدست آوریم. بدین منظور آن را در یک متغیر ذخیره می‌کنیم تا از آن استفاده کنیم. پس در این راه‌حل داریم:

```
for i = n − 1 to 0:
        dp[i] = 1
        prev = 0
        for j = i + 1 to n − 1:
                temp = prev
                prev = dp[j]
                if j = i + 1 and S[i] == S[j]:
                        dp[j] = 2
                else if S[i] == S[j]:
                        dp[j] = temp + 2
                else:
                        dp[j] = max(dp[j − 1], dp[j])
```

۳. اعداد را با شروع از ۰ در نظر می‌گیریم. ابتدا مسئله را به صورت بازگشتی حافظه دار حل می‌کنیم. $maxProfit(i)$ را بیشترین میزان برق تولیدی به ازای بلوک‌های iام تا $n-1$ام در نظر می‌گیریم.

جواب نهایی مسئله با توجه به این تعریف $maxProfit(0)$ است.

برای بدست‌آوردن جواب به طور بازگشتی در این‌جا دو حالت قراردادن و قرارندادن توربین فعلی را در نظر می‌گیریم و جواب بیشینه برق تولیدی بین آن‌هاست.

برای حالت‌های پایه این رابطه بازگشتی دو حالت داریم. اگر بلوکی وجود نداشته باشد جواب برابر ۰ است. اگر به دنبال جواب در d بلوک آخر باشیم، جواب بیشترین میزان تولیدی برق در d بلوک آخر است.

برای این که زیرمسئله‌های تکراری را دوباره حل نکنیم از آرایه یک‌بعدی dp به اندازه n استفاده می‌کنیم که در ابتدا تمام خانه‌های آن مقدار 1- را دارا هستند که به معنی پرنشدن آن خانه از آرایه توسط راه‌حل است.

در نهایت برای جمع‌بندی حالت‌های قبل داریم:

```
maxProfit(i):
        if i >= n:
                return 0
        else if dp[i] != -1:
                return dp[i]
        else if n - i < d + 2:
                dp[i] = max(p[j]) for j >= i and j < n
                return dp[i]
        else:
                dp[i] = max(p[i] + maxProfit(i + d + 1), maxProfit(i + 1))
                return dp[i]
```

راه‌حل با برنامه‌نویسی پویا از نظر روند مشابه راه‌حل بازگشتی حافظه‌دار است و شبه‌کد آن در ادامه نوشته شده است:

```
dp[n - 1] = p[n - 1]
for i = n - 2 to n - d - 1:
        dp[i] = max(dp[i + 1], p[i])

for i = n - d - 2 to 0:
        dp[i] = max(p[i] + dp[i + d + 1], dp[i + 1])
```

در این‌جا جواب با توجه به تعریف $dp[0]$ است.

۴. در این مسئله dp را آرایه‌ای سه‌بعدی شامل صفر و یک با مقادیر اولیه ۰ و اندازه $N \times M \times (K+1)$ در نظر می‌گیریم که $dp[i][j][k]$ دارای مقدار یک است اگر بتوان با شروع از خانه ابتدایی به خانه $(i+1, j+1)$ رسید به طوری که دقیقا k مین از کار انداخته شده باشد و در غیر این صورت دارای مقدار صفر است.

حواب نهایی مسئله بیشترین kای است که مقدار $dp[N-1][M-1][k]$ برای آن یک باشد.

۳ حالت پایه برای این مسئله وجود دارد.

اگر در خانه اول باشیم:

if i == 0 and j == 0:

$$dp[i][j][k] = (k == M[i+1][j+1])$$

اگر در اولین خانه هر سطر باشیم:

if i == 0:

$$dp[i][j][k] = dp[i][j][k] \vee dp[i][j-1][k - M[i+1][j+1]]$$

اگر در اولین خانه هر ستون باشیم:

if j == 0:

$$dp[i][j][k] = dp[i][j][k] \vee dp[i-1][j][k - M[i+1][j+1]]$$

حال برای حالت‌های بازگشتی تمام حالت‌های حرکت به پایین، راست و پایین‌راست را در نظر می‌گیریم.

حرکت به راست:

$$dp[i][j][k] = dp[i][j][k] \vee dp[i-1][j][k - M[i+1][j+1]]$$

حرکت به پایین:

$$dp[i][j][k] = dp[i][j][k] \vee dp[i][j-1][k - M[i+1][j+1]]$$

حرکت به پایین‌راست:

$$dp[i][j][k] = dp[i][j][k] \vee dp[i-1][j-1][k - M[i+1][j+1]]$$

در نهایت برای جمع‌بندی حالات بالا داریم:

```
for i = 0 to N - 1:
    for j = 0 to M - 1:
        for k = M[i + 1][j + 1] to K:
            if i == 0 and j == 0:
                dp[i][j][k] = (k == M[i + 1][j + 1])

            else if i == 0:
                dp[i][j][k] = dp[i][j][k] ||
                    dp[i][j - 1][k - M[i + 1][j + 1]]

            else if j == 0:
                dp[i][j][k] = dp[i][j][k] ||
                    dp[i - 1][j][k - M[i + 1][j + 1]]

            else:
                dp[i][j][k] = dp[i][j][k] ||
                    dp[i - 1][j][k - M[i + 1][j + 1]]

                dp[i][j][k] = dp[i][j][k] ||
                    dp[i][j - 1][k - M[i + 1][j + 1]]

                dp[i][j][k] = dp[i][j][k] ||
                    dp[i - 1][j - 1][k - M[i + 1][j + 1]]
```

راه‌حل بالا دارای پیچیدگی زمانی و میزان حافظه مصرفی $O(N \times M \times K)$ است.

۵. در این مسئله dp را آرایه‌ای دوبعدی از اعداد صحیح با مقادیر اولیه ۰ و اندازه $(n+1) \times (m+1)$ در نظر می‌گیریم که $dp[i][j]$ در آن به معنای تعداد دفعات تکرار زیررشته شامل j حرف اول S_2 در زیررشته شامل i حرف اول S_1 است.
جواب نهایی مسئله طبق تعریف $dp[n][m]$ است.
برای پرکردن بازگشتی این dp برابر بودن کاراکترهای آخر دو زیررشته در حال بررسی را در نظر می‌گیریم. اگر برابر بودند جمع حالاتی که این برابری را در نظر گرفته و در نظر نگرفته را به عنوان جواب حساب می‌کنیم. اگر برابر نبودند کاراکتر آخر زیررشته S_1 را در نظر نمی‌گیریم و جواب را برای آن حالت در نظر می‌گیریم. یعنی داریم:

```
dp[i][j] = (S1[i] == S2[j] ? dp[i - 1][j] + dp[i - 1][j - 1] : dp[i - 1][j])
```

برای حالت‌های پایه دو حالت داریم. اگر طول زیررشته S_1 برابر صفر باشد و طول زیررشته S_2 صفر نباشد جواب نداریم. اگر طول زیررشته S_2 برابر ۰ باشد، این زیررشته یک بار در زیررشته S_1 تکرار شده در نتیجه جواب برابر ۱ است.

در نهایت برای جمع‌بندی حالات بالا داریم:

```
for j = 1 to m:
        dp[0][j] = 0

for i = 0 to n:
        dp[i][0] = 1

for i = 1 to n:
        for j = 1 to m:
                dp[i][j] = dp[i - 1][j]

                if S1[i] == S2[j]:
                        dp[i][j] += dp[i - 1][j - 1]
```

راه‌حل بالا دارای پیچیدگی زمانی و میزان حافظه مصرفی $O(m \times n)$ است. همانطور که در راه‌حل مشاهده می‌شود برای پرکردن مقادیر در هر سطر فقط نیاز به مقادیر سطر قبل است پس می‌توان فقط مقادیر dp را برای سطر قبل نگه‌داشت. به طور مثال می‌توان dp را با اندازه $m \times 2$ برای هر سطر در هنگام محاسبه اختیار کرد که در این صورت میزان حافظه مصرفی بهینه خواهد شد.

۶. در این مسئله dp را آرایه‌ای یک‌بعدی شامل اعداد صحیح با اندازه n تعریف می‌کنیم که در آن $dp[i]$ به معنای کمترین هزینه کل برای دفاتر $i+1$ام تا nام است به شرطی که در دفتر $i+1$ام بخش خدمات پس از فروش ایجاد شده باشد.

در این مسئله به سادگی با ایجاد بخش پس از فروش در دفتر $i+1$ام و محاسبه کمترین هزینه کل به ازای دفاتر بعد از آن، $dp[i]$ را آپدیت می‌کنیم. نکته حائز اهمیت در این‌جا این است که برای کمینه‌شدن هزینه کل برای تعدادی دفتر باید جمع تمام هزینه‌های دسترسی برای آن دفترها را در نظر گرفت.

تنها حالت پایه در این مسئله برای حالتی است که فقط یک دفتر نهایی داریم که جواب برای این حالت، هزینه ایجاد بخش در آن دفتر است.

همچنین جواب نهایی این مسئله به کمک آرایه dp و در نظر گرفتن قرار دادن یا ندادن بخش پس از فروش در دفتر اول بدست می‌آید.

در نهایت برای جمع‌بندی این حالات داریم:

```
getMinPrice(i):
        minPrice = ∞

        for k = i + 1 to n - 1:
                minPrice = min(minPrice, (k-i)×(k-i-1)/2 + dp[k])

        return minPrice


dp[n - 1] = c[n]

for i = n - 2 to 0:
        dp[i] = c[i + 1] + getMinPrice(i)

ans = dp[0]
for i = 1 to n:
        ans = min(ans, i×(i+1)/2 + dp[i])
```

۷. برای حل این مسئله از راه‌حل بازگشتی حافظه‌دار استفاده می‌کنیم. dp را آرایه‌ای شامل اعداد صحیح با اندازه $N \times N$ تعریف می‌کنیم که $dp[i][j]$ در آن مشخص‌کننده بیشترین فروش ممکن برای عتیقه‌های $i + 1$ام تا $j + 1$ام است. این آرایه در ابتدا دارای مقادیر 1- است که مشخص می‌کند جواب برای زیرمسئله پیدا نشده است.

برای حل این مسئله رابطه $maxProfit(i, j)$ را در نظر می‌گیریم که بیشترین فروش ممکن را برای عتیقه‌های $i + 1$ام تا $j + 1$ام باز می‌گرداند. طبق این تعریف جواب نهایی مسئله برای بیشترین میزان فروش $maxProfit(0, n - 1)$ است.

حالت بازگشتی این رابطه برای دو حالت فروش عتیقه سمت چپ و فروش عتیقه سمت راست و جواب برای حالات دیگر بدست می‌آید که جواب نهایی بیشینه جواب این دو حالت است.

حالت پایه این رابطه وقتی رخ می‌دهد که فقط یک عتیقه داشته که در این صورت میزان فروش عتیقه در نظر گرفته می‌شود.

برای نمایش فروش کدام عتیقه در هر سال نیز از آرایه کمکی $sell$ با اندازه $N \times N$ شامل مقادیر ۰ و ۱ بهره می‌گیریم که در آن $sell[i][j]$ اگر ۰ باشد به معنای فروش عتیقه سمت چپ و اگر ۱ باشد به معنای فروش عتیقه راست است.

در نهایت برای جمع‌بندی این حالات داریم:

```
maxProfit(i, j):
    if dp[i][j] != -1:
        return dp[i][j]

    year = n - (j - i)

    if(j == i)
        return year * p[i + 1]

    maxProfitLeft = year * p[i + 1] + maxProfit(i + 1, j)
    maxProfitRight = year * p[j + 1] + maxProfit(i, j - 1)

    ans = max(maxProfitLeft, maxProfitRight)
    dp[i][j] = ans

    if maxProfitLeft >= maxProfitRight:
        sell[i][j] = 0
    else:
        sell[i][j] = 1

    return ans

ans = maxProfit(0, n - 1)

i = 0, j = n - 1
while i <= j:
    if sell[i][j] == 0:
        print 'left '
        i++
    else:
        print 'right '
        j--
```