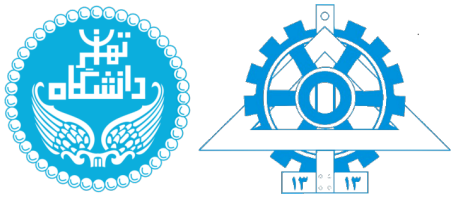
به نام خدا



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

راه حل تمرین کتبی دوم

---

۱. برای حل سوال آن را به طور غیرمستقیم حل می‌کنیم به این صورت که در ابتدا طول بلندترین زیر دنباله $palindrome$ رشته داده شده را پیدا می‌کنیم و این طول را از طول رشته ورودی کم می‌کنیم به این ترتیب، کمترین تعداد حذف حرف برای تبدیل شدن رشته به $palindrome$ را پیدا کرده ایم

حال به حل مساله کمکی گفت شده می‌پردازیم :

$dp[s][f]$ را تعریف می‌کنیم بلندترین زیر دنباله palindrome که تشکیل شده از عناصر s تا f است همینطور $dp[i][i]$ نیز با توجه به اینکه تنها از یک حرف تشکیل شده حتما palindrom است و مقدار آن ۱ است.

به ترتیب زیاد شدن طول رشته در هر مرحله به این شکل عمل می‌کنیم که اگر طول رشته ۲ است و حرف ابتدا و انتها آن برابر است dp آن را برابر ۲ قرار میدهیم و همینطور اگر طول آن بیشتر از ۲ است و حروف ابتدا و انتها آن با هم برابر هستند در این صورت dp آن برابر با جمع ۲ و dp رشته داخلی یعنی $dp[s+1][f-1]$ است. در غیر این صورت نیز dp آن رشته برابر با بیشترین طولی است که می‌توان با palindrome بودن رشته در حالتی که حرف ابتدایی حضور نداشته باشد و در حالتی که حرف انتهایی حضور نداشته باشد ، بدست آورد. (به بیان بهتر $max(dp[s+1][f], dp[s][f-1])$)

با این راه حل در ابتدا جواب مساله برای رشته‌های کوچک تر در رشته اصلی به دست خواهد آمد و به ترتیب جواب مساله با طول‌های بیشتر به دست خواهند آمد و در انتها $dp[0][n-1]$ که جواب مساله اصلی ما است به دست خواهد آمد.

۲. برای حل سوال از یک آرایه استفاده می‌کنیم که خانه i ام آن نشان دهنده این است که $minimum$ تعداد حرکت برای رسیدن به خانه i ام با شروع از خانه اول چقدر است. به این صورت جواب اصلی مساله $dp[n-1]$ است که $minimum$ تعداد پرش برای رسیدن به خانه $n-1$ است با شروع از خانه اول.

به ترتیب صعودی از خانه‌های ابتدایی شروع می‌کنیم و در هر مرحله با توجه به اینکه می‌توان از راه‌های مختلفی به خانه $n-1$ رسید بین این راه‌ها $minimum$ میگیریم . راه‌های مختلف برای رسیدن به هر خانه به این صورت هستند که میتوانیم در هر مرحله تا خانه i آمده باشیم و از آن خانه پرشی به اندازه‌ای که مجاز هستیم انجام دهیم و به خانه مورد نظر برسیم. با $minimum$ گرفتن همه این راه‌ها میتوانیم کمترین تعداد حرکت برای رسیدن به خانه مورد نظر را پیدا کنیم. آرایه گفت شده نیز از خانه‌های ابتدای آن تا انتهاای آن به ترتیب تا رسیدن به جواب اصلی مساله پر خواهد شد.

۳.
- برای حل مساله آرایه dp را در نظر میگیریم و عنصر i ام این آرایه نشان دهند جواب مساله با شرط حضور عنصر i آرایه در جواب است. با در نظر گرفتن این راه حل جواب مساله اصلی $maximum$ کل آرایه خواهد بود به دلیل اینکه عنصر آخری که در جواب حاضر است میتواند هر کدام از عناصر آرایه باشد . برای پر کردن آرایه گفته‌شده به این صورت عمل می‌کنیم که از ابتدای آرایه شروع می‌کنیم و به ترتیب صعودی آرایه را پر می‌کنیم به این صورت که برای بدست اوردن مقدار آرایه در عنصر i ام حالت‌های مختلف جواب را در نظر میگیرم یعنی اینکه عنصر قبلی آن در جواب کدام یک از عناصر قبلی بود و $dp[i]$ آن را نشان میدهد؛حال بین همه این حالت ها $maximum$ میگیریم و جواب آن خانه از آرایه را بدست می‌آوریم.

همینطور $order$ زمانی آن به این دلیل $O(n^2)$ است که روی یک آرای n تایی حرکت می‌کنیم و در هر مرحله مقدار آرایه با توجه به مقدار عناصر قبلی آن در آرایه معین میشود.

- فرض کنید می خواهیم مساله LIS را برای آرایه A_i حل کنیم. آرایه کمکی $dp[i]$ را تعریف می کنیم در بین همه دنباله های صعودی به طول i کمترین عدد در بین عناصر آخر چند است. آرایه A_i را از اول تا آخر پیمایش می کنیم و در این بین آرایه $dp[i]$ را پر می کنیم ابتدا همه عناصر dp صفر هستند. فرض کنیم الان روی عنصر i ام هستیم فرض کنید طول LIS تا الان x باشد. واضح است که از خانه ۱ تا x آرایه dp پر شده است و بقیه خانه ها صفر مانده اند (فرض کنید عناصر آرایه اصلی مثبت اند) این x عدد آرایه dp حتما صعودی هستند (چرا؟!) پس می توانیم روی این آرایه search binary بزنیم عدد A_i را روی dp باینری سرچ می زنیم و اولین عدد بزرگتر از آن را پیدا می کنیم (فرض کنید اندیس j) با اضافه شدن A_i تنها مقداری که در آرایه dp تغییر می کند $dp[j]$ است که مقدارش A_i خواهد شد و بقیه عناصر حتما ثابت می مانند (چرا؟!) این پیمایش را تا آخرین A_i ادامه می دهیم. فرض کنید در آخر کار آخرین عنصر از آرایه dp که مقدار دارد و صفر نیست $dp[k]$ باشد. LIS جواب برابر k خواهد بود

۴. آرایه‌ای را در نظر میگیریم که عنصر i ام آن جواب مساله را برای i کاگر مشخص می‌کند به عنوان مثال عنصر ۳ آرایه جواب مساله را برای حالتی که ۳ کارگر داریم معین می‌کند. برای پر کردن آرایه نیز از مانند سوال قبل از ابتدا شروع می‌کنیم و تا پر شدن کامل آن ادامه میدهیم

در هر مرحله میتوانیم دو گروه با تعداد عضو دلخواه بسازیم پس برای به دست آوردن مقدار عنصر عنصر i ام آرایه که نشان دهنده i کارگر است تمام گروه بندی‌های مختلف را در نظر میگیریم و مقدار بازدهی آن‌ها را بدست آورده و جمع می‌کنیم (این مقادیر در مراحل قبلی ساخت آرایه به دست آماده است) جواب انتهایی مساله نیز در عنصر آخر آرایه به دست می آید.

$order$ زمانی راه حل هم با توجه به اینکه یک بار آرایه را طی می‌کنیم و در هر مرحله نیز برای بدست آوردن مقدار یک عنصر آرایه مقادیر قبلی آرایه را مورد بررسی قرار میدهیم پس $O(n^2)$ خواهد شد .

۵. $dp[i][j]$ را به این صورت تعریف می‌کنیم که خانه i و j آن نشان دهند جواب مساله با تعداد i خرید و فروش در تا روز j ام است. بدیهی است که با 0 خرید و فروش سودی که میتوانیم به دست بیاوریم 0 است همینطور اگر تعداد روزها 0 باشد .

حال از تعداد 0 خرید و فروش شروع می‌کنیم و در هر کدام از این مراحل برای تعداد روزهای مختلف که از 0 شروع میشود جواب مساله را بدست می‌آوریم .

برای بدست آوردن جواب مساله با تعداد i خرید و فروش و تا روز j ام نیز از رابطه زیر استفاده می‌کنیم که به این معنا است که یا خرید و فروش انجام میدهیم و یا نمیدهیم که در آن صورت با استفاده از نگه داشتن بیشترین مقدار تفاوتی که بین قیمت سهام در روزهای مختلف بوده، سود $maximum$ خود را حساب می‌کنیم.

```
def max_profit(prices, K):
    if K == 0 or prices == []:
        return 0

    days = len(prices)
    # 0th transaction up to and including kth transaction is considered.
    num_transactions = K + 1

    T = [[0 for _ in range(days)] for _ in range(num_transactions)]

    for transaction in range(1, num_transactions):
        max_diff = -prices[0]
        for day in range(1, days):
            T[transaction][day] = max(T[transaction][day - 1],
                                      # No transaction
                                      prices[day] + max_diff)
            # price on that day with max diff
            max_diff = max(max_diff,
```

```
                    T[transaction - 1][day] - prices[day])
            # update max_diff

    print_actual_solution(T, prices)

    return T[-1][-1]
```

۶. $dp[i][j]$ را برابر جواب مساله تا رسیدن به سطر i و ستون j می نامیم. جواب نهایی مساله بیشترین مقدار این آرایه در همه خانه‌های جدول است.

- برای حل قسمت اول آن، این نکته را در نظر میگیریم که برای رسیدن به هر نقطه ۳ راه داریم پس بین جوابی که از عبور از خانه‌ها به دست می آید باید $maximum$ بگیریم که شبه کد آن در ادامه آماده است

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

- برای این حالت این نکته را در نظر میگیریم که بد از ورود به یک سطر میتونیم بین ستون‌های آن حرکت کنیم و امتیاز کسب کنیم پس اینکه از کدام ستون وارد آن سطر شدیم مهم است و روی آن حالت بندی می‌کنیم.
  آرایه کمکی که تعریف کرده ایم نشان دهند ماکزیمم امتیاز کسب شده در حرکت عرضی بین ستون k تا j در سطر i است.

$dp[i][j] = max(dp[i-1][k] + t[i][k][j]) \quad for \quad all \quad Ks$