# طراحی الگوریتم

## تمرین دوم - برنامه ریزی پویا

آرین باستانی و علی حمزه پور

### ۱ . ربات احمق — ۱۵ نمره

محمد یک ربات برای تمیز کردن خانه‌ی خود ساخته است. خانه به‌صورت یک جدول $n \times n$ مدل شده و میزان کثیفی بلوک سطر $i$ام و ستون $j$ام به‌صورت $a_{i,j}$ نشان داده می‌شود. مشکل این است که ربات فقط می‌تواند به سمت بالا، بالاچپ، و بالاراست حرکت کند. ربات حرکتش را از یکی از بلوک‌های پایینی جدول آغاز می‌کند.

**الف)** با استفاده از برنامه‌ریزی پویا با محدودیت زمانی و حافظه‌ی $O(n^2)$، الگوریتمی طراحی کنید که ربات بتواند بیشترین میزان کثیفی را از بین ببرد.

**ب)** آیا می‌توان این مسئله را با حافظه‌ی کمتری حل کرد؟ اگر پاسخ منفی است، دلیل بیاورید و اگر پاسخ مثبت است، روش خود را شرح دهید.

**پاسخ :**

**الف)**

برای حل این مسئله، یک جدول $DP$ تعریف می‌کنیم. $DP[i][j]$ نشان‌دهنده‌ی بیشترین میزان تمیزکاری‌ای است که با شروع از یکی از خانه‌های پایینی و رسیدن به خانه‌ی $(i,j)$ انجام می‌شود.

- **پرکردن $DP$:**

ربات می‌تواند از خانه‌های $(i-1,j)$، $(i-1,j-1)$ یا $(i-1,j+1)$ به خانه‌ی $(i,j)$ برسد. بنابراین رابطه‌ی برنامه‌ریزی پویا به صورت زیر است:

$$DP[i][j] = \begin{cases} a_{i,j} + \max\Big(DP[i-1][j], DP[i-1][j+1]\Big) & \text{if } j = 0 \\ a_{i,j} + \max\Big(DP[i-1][j], DP[i-1][j-1]\Big) & \text{if } j = n-1 \\ a_{i,j} + \max\Big(DP[i-1][j], DP[i-1][j-1], DP[i-1][j+1]\Big) & \text{otherwise} \end{cases}$$

- **شرایط مرزی:**

برای خانه‌های سطر اول، چون خانه‌ای بالای آن‌ها نیست، داریم:

$$DP[0][j] = a_{0,j} \qquad \text{برای هر } j$$

- **پاسخ نهایی:**

بیشترین مقدار $DP$ در سطر آخر (یعنی سطر $n-1$) برابر با پاسخ مسئله است:

$$\text{answer} = \max\Big(DP[n-1][0], DP[n-1][1], \ldots, DP[n-1][n-1]\Big)$$

**پیچیدگی زمانی:** این الگوریتم برای هر خانه یک بار محاسبه انجام می‌دهد. بنابراین پیچیدگی زمانی آن $O(n^2)$ است.

**ب)** برای بهینه‌سازی حافظه، به‌جای ذخیره‌سازی کل جدول $DP$، می‌توان تنها دو ردیف از جدول را نگه داشت: ردیف فعلی و ردیف قبلی(چرا که برای پر کردن $DP$ فقط به جداول ردیف پایین نیاز داریم). با این روش، حافظه به $O(n)$ کاهش می‌یابد. هر بار که ردیف جدیدی محاسبه می‌شود، ردیف قبلی جایگزین می‌گردد.

## ۲. دنیای تک‌بعدی علی — ۱۵ نمره

علی پسری تک‌بعدی‌ست. دنیای او تنها یک بعد دارد و می‌توان مکان او را با یک عدد نشان داد. او هر روز بیدار می‌شود و برای هدف تک‌بعدی‌اش تلاش می‌کند. او در ابتدای روز اول در مکان $x = 0$ قرار دارد. همچنین او از قبل می‌داند که در روز $i$ام می‌تواند به اندازه‌ی $m_i$ در دنیایش به جلو یا عقب برود. هدف علی در مکان $x = t$ قرار دارد. از آنجا که علی دچار اختلال *نشخوار فکری (overthinking)* است، دائما به این فکر می‌کند که با چه حالاتی می‌تواند به هدفش برسد. به علی کمک کنید و الگوریتمی را پیشنهاد دهید که با دانستن مقادیر $t$ و $m_i$ به او بگوید که به چند حالت مختلف می‌تواند در پایان روز $n$ام به هدفش برسد! در صورتی که $\sum_{i=1}^{n} m_i = M$، پیچیدگی زمانی الگوریتم شما باید $\mathcal{O}(Mn)$ باشد.

**پاسخ :**

- **تعریف $DP$:** $DP[i][j]$ را معادل تعداد حالاتی تعریف می‌کنیم که علی در پایان روز $i$ام در مکان $x = j$ باشد. مقادیر $i$ بین $0$ تا $n$ می‌توانند باشند. مقادیر $j$ نیز از $-M$ تا $M$ نیز می‌توانند متغیر باشند، زیرا علی در صورتی که هر روز جلو برود، به مکان $x = M$ می‌رسد و در صورتی که هر روز به عقب برود، به مکان $x = -M$ می‌رسد.
- **پایه $DP$:** علی در روز $0$ام تنها در خانه $x = 0$ می‌تواند باشد پس:

$$DP[0][j] = \begin{cases} 0 & j \neq 0, \\ 1 & j = 0 \end{cases}$$

- **رابطه** $DP$: علی هر روز می‌تواند یا $m_i$ خانه به جلو برود و یا $m_i$ خانه به عقب برود. پس در صورتی که در پایان روز $i$ام در خانه $x = j$ باشد، در پایان روز $i-1$ام یا در $x = j - m_i$ و یا در $x = j + m_i$ بوده است. پس می‌توان نوشت:

$$DP[i][j] = DP[i-1][j-m_i] + DP[i-1][j+m_i] \quad i \geq 1$$

بدیهی‌ست که مقدار $DP[i][j]$ در صورتی که $j > |M|$ باشد، صفر در نظر گرفته می‌شود زیرا علی به آن مکان‌ها هیچ‌وقت نمی‌تواند برسد.

- **پاسخ** $DP$: علی می‌خواهد بداند که به چند حالت در پایان روز $n$ام در خانه‌ی $x = t$ قرار دارد، پس پاسخ مسئله $DP[n][t]$ خواهد بود.

- **پیچیدگی الگوریتم:** همانطور که توضیح داده شد، مقادیر $i$ از 0 تا $n$ متغیر هستند و مقادیر $j$ نیز از $-M$ تا $M$. از آنجا که هر خانه از جدول دیپی را می‌توان در $O(1)$ پر کرد، پیچیدگی زمانی کلی الگوریتم $O(Mn)$ می‌شود.

## ۳. دبیر ACM ۲۰ نمره

بعد از انتخابات اعضای انجمن ACM، دبیر این انجمن باید مشخص شود. اعضای انجمن به این توافق رسیدند که محبوب‌ترین فرد باید به عنوان دبیر انتخاب شود. هر کس در دانشکده طرفدار یکی از اعضای انجمن است. اعضای انجمن معتقدند که محبوب‌ترین فرد کسی نیست که بیشترین طرفدار را داشته باشد، بلکه کسی هست که از تمام گروه‌ها و قشرها طرفدار داشته باشد! آن‌ها به ترتیب به $n$ نفر از افراد دانشکده در گروه‌ها و ورودی‌های مختلف مراجعه می‌کنند و از آنها می‌پرسند که طرفدار چه نماینده‌ای هستند. این اطلاعات را به صورت دنباله‌ای از اعداد می‌نویسند به طوری که اگر $A_i = j$، یعنی نفر $i$ام در این نمونه‌گیری طرفدار عضو $j$ام انجمن است. آنها سپس طبق این دنباله برای هر عضو یک درجه‌ی گمنامی تعریف میکنند:

کمترین مقدار $k$ که در هر $k$ نفر متوالی در دنباله حداقل یک نفر طرفدار عضو $j$ام باشد، درجه‌ی گمنامی عضو $j$ام در انجمن است (اعضای انجمن از 1 تا $m$ شماره‌گذاری شده‌اند).

برای مثال در دنباله‌ی $\{1, 3, 2, 3, 1, 3, 2, 3, 2\}$ درجه‌ی گمنامی نماینده‌های 1، 2 و 3 به ترتیب 5، 4 و 2 است. دبیر انجمن فردی‌ست که کمترین درجه‌ی گمنامی را داشته باشد. در صورتی‌که چند عضو با هم کمترین درجه‌ی گمنامی را داشته باشند، دبیر انجمن فردی‌ست که شماره‌ی او از همه کمتر است. در صورتی که بدانیم تعداد اعضای انجمن $m$ است و $m < n$، شبه کد الگوریتم با پیچیدگی زمانی $\mathcal{O}(n)$ را بنویسید که با دریافت دنباله‌ی $A$ به طول $n$، دبیر انجمن را پیدا کند. الگوریتم خود را توضیح بدهید و صرفا به شبه کد بسنده نکنید.

**پاسخ :**

به سادگی اثبات می‌شود که درجه‌ی گمنامی عضو $j$ام انجمن برابر بیشترین فاصله بین دو طرفدار این عضو در دنباله است.(فاصله‌ی اولین طرفدار این عضو با ابتدای دنباله و فاصله‌ی آخرین طرفدار با انتهای دنباله نیز باید لحاظ شود.) برای مثال در دنباله $\{1, 3, 2, 3, 1, 3, 2, 3, 2\}$ بیشترین فاصله‌ی اعداد 1 از هم و از ابتدا و انتهای دنباله 5 است

و درجه گمنامی نیز همان 5 است. پس کافی‌ست به ازای هر عدد $1 \leq j \leq m$، بیشترین فاصله اعداد $j$ در دنباله را از هم و از ابتدا و انتهای دنباله پیدا کنیم. برای این کار دو متغیر تعریف می‌کنیم. $lastseen[j]$ را برابر اندیس آخرین عضوی از دنباله که تا به اینجا بررسی کردیم و مقدار آن $j$ بوده تعریف می‌کنیم. همچنین $max\_distance[j]$ را بیشترین فاصله اعداد $j$ در دنباله از هم و از ابتدا و انتهای دنباله تعریف می‌کنیم. قبل از بررسی اعضای دنباله مقادیر $lastseen[j]$ و $max\_distance[j]$ را به ازای تمام مقادیر $j$ با صفر پر می‌کنیم. اگر در خانه $i$ام دنباله عدد $l$ باشد، مقادیر متغیرها را به این شکل آپدیت می‌کنیم:

$$max\_distance[l] = max(max\_distance[l], i - lastseen[l])$$

$$lastsenn[l] = i$$

این کار را به ترتیب برای تمام خانه‌های دنباله انجام می‌دهیم و در انتها نیز برای تمام مقادیر $j$ متغیرها را آپدیت می‌کنیم:

$$max\_distance[j] = max(max\_distance[j], n + 1 - lastseen][j])$$

حال برای پیدا کردن دبیر انجمن کافی‌ست کوچک‌ترین $j$ را پیدا کنیم که در آن $max\_distance[j]$ مینیمم است. سودوکد این الگوریتم به شکل زیر است:

**Algorithm ۱** Find the Chairman of ACM

**Input:** Integer $n$، Array $A[1 \dots n]$
**Initialize:** Array $last\_seen[1 \dots N]$ with 0، Array $max\_distance[1 \dots N]$ with 0
**for** $i = 1$ to $n$ **do**
  $max\_distance[A[i]] \leftarrow \max(i - last\_seen[A[i]], max\_distance[A[i]])$
  $last\_seen[A[i]] \leftarrow i$
**end for**
**for** $i = 1$ to $n$ **do**
  $max\_distance[i] \leftarrow \max(n + 1 - last\_seen[i], max\_distance[i])$
**end for**
**Initialize:** $min\_distance \leftarrow max\_distance[1]$، $chairman \leftarrow 1$
**for** $i = 2$ to $n$ **do**
  **if** $max\_distance[i] > min\_distance$ **then**
    $min\_distance \leftarrow max\_distance[i]$
    $chairman \leftarrow i$
  **end if**
**end for**
**Output:** $chairman$

پیچیدگی زمانی این الگوریتم $O(n)$ است زیرا با چند پیمایش در دنباله توانستیم جواب را پیدا کنیم.

## ۴. سازمان رنجش ۲۰ نمره

مدتی است که سازمان رنجش تصمیم به اضافه کردن یک گزینه‌ی جایگزین برای کسانی که نتیجه‌ی کنکور مورد انتظار را نگرفته‌اند، گرفته است. این تصمیم بدین صورت است که هر داوطلب از آزمون‌های آزمایشی که شرکت کرده می‌تواند تعدادی را انتخاب کند (می‌توان از هر آزمون بیش از یک‌بار استفاده کرد) و با وجود قابلیت تعویض ترتیب آزمون‌ها و تعویض ترتیب ترازهای دروس به ازای هر آزمون (در اینجا فقط ترازهای ریاضی، فیزیک و شیمی مطرح است و می‌توان ترازهای آنها را بین هم جابجا کرد)، آزمون‌ها را به چینش انتخابی خود به سازمان رنجش بدهد. این آزمون‌ها باید به طوری چیده شوند که در هر آزمون، نسبت به آزمون قبلی تراز فیزیک و تراز شیمی هر دو پیشرفت داشته باشند و اگر برای آزمون‌های دریافت شده از داوطلب این شرایط برقرار بود، مجموع ترازهای ریاضی در این آزمون‌ها محاسبه شده و با توجه به آن، قبولی داوطلب در دانشگاه‌ها و رشته‌های مورد نظر مشخص می‌شود. برای آزمون $i$ام، ترازهای ریاضی، فیزیک و شیمی به ترتیب برابر $m_i$ و $p_i$ و $c_i$ هستند و $N$ آزمون وجود دارد.

الگوریتمی برای پیدا کردن ترتیب با بیشینه مجموع ترازهای ریاضی با شرایط گفته شده ارائه دهید (محدودیت پیچیدگی زمانی برابر $O(N^2)$ می باشد).

**پاسخ :**

برای حل این مسئله، می‌توان آن را برابر مسئله‌ی Problem Stacking Box در نظر گرفت که راه‌حل آن به شرح زیر است:

۱. ابتدا تمام جایگشت های ممکن برای تراز فیزیک و شیمی و ریاضی به ازای هر آزمون را تولید می‌کنیم. پس برای هر آزمون، شش ترکیب ممکن وجود دارد.

۲. سپس این ترکیب‌ها را بر اساس ترازهای فیزیک و شیمی به ترتیب نزولی مرتب می‌کنیم.

۳. حال باید مشابه الگوریتم «بیشینه‌ی دنباله‌ی افزایشی»(LIS)، برای هر آزمون بهترین انتخاب را پیدا کنیم. تعریف زیر را داریم:

$$\text{DP}(i) = \text{بیشینه مجموع تراز ریاضی با آزمون } i \text{ در انتهای دسته}$$
$$\text{DP}(i) = \max\left(\text{DP}(j)\right) + m_i \quad \text{for every } j < i \quad \text{و} \quad p_j < p_i \quad \text{و} \quad c_j < c_i$$

اگر هیچ $j$ای برای آزمون $i$ وجود نداشت که شرایط بالا را ارضا کند، آنگاه $\text{DP}(i) = m_i$ خواهد بود.

۴. در نهایت، بیشترین مقدار $\text{DP}(i)$ را که برای تمام آزمون‌ها محاسبه شده است، برمی‌گردانیم.

## ۵. چالش نمره گرفتن ۲۰ نمره

پس از اعلام نمرات درس طراحی الگوریتم، آرین به دفتر دکتر دوستی مراجعه می‌کند و از او خواهش می‌کند که به دلیل حضور فعال در کلاس نمره‌اش را افزایش دهد. دکتر دوستی قبول می‌کند اما از آنجا که به همین سادگی‌ها به کسی نمره نمی‌دهد، یک چالش برای او مطرح می‌کند. او به آرین می‌گوید: «من به تو مقدار ثابت $(x)$ نمره اضافی می‌دهم و تو باید مقداری دلخواه از آن $(k)$ را خرج یک بازی با من کنی. در صورتی که بازی را ببری، $x - k$ نمره به

تو تعلق می‌گیرد، اما اگر ببازی از نمره خبری نیست!»
بازی به شکل زیر است:

- دکتر دوستی یک عدد دلخواه از 1 تا $n$ انتخاب می‌کند.
- آرین در هر مرحله می‌تواند یک عدد حدس بزند. در صورتی که حدس او غلط باشد، به اندازه‌ی عددی که حدس زده است، از $k$ امتیازی که ابتدا انتخاب کرده بود کم می‌شود. دکتر دوستی هم به او می‌گوید که عدد مدنظرش از عدد آرین بزرگتر است یا کوچک‌تر.
- بازی به همین رویه ادامه پیدا می‌کند تا زمانی که آرین عدد درست را پیدا کند و یا امتیازهایش تمام شوند. در صورتی که امتیازهایش تمام شود، او بازی را باخته و هیچ نمره‌ای نمیگیرد!

به آرین کمک کنید و الگوریتمی طراحی کنید که با دانستن مقدار $n$، کمترین میزان نمره‌ای که نیاز است خرج شود تا آرین در هر صورت بتواند ببرد را پیدا کند (به این معنا که صرف نظر از هر عددی که دکتر دوستی انتخاب می‌کند، آرین می‌تواند با استراتژی خود و آن میزان نمره بازی را ببرد). پیچیدگی زمانی الگوریتم شما باید $\mathcal{O}(n^3)$ باشد.

**پاسخ :**

- **تعریف** $DP$: $DP[i][j]$ کمترین میزان نمره موردنیاز برای بردن بازی‌ست در صورتی که بدانیم عدد بین $i$ و $j$ است.
- **پایه** $DP$: اگر $i = j$ تنها یک آپشن برای حدس زدن داریم که خود جواب درست است پس در این حالت نیاز به خرج کردن نمره نداریم. پس:

$$DP[i][i] = 0$$

برای راحتی در ادامه راه حل فرض می‌کنیم اگر $i > j$ باشد نیز مقدار $DP$ صفر است.

- **رابطه** $DP$: فرض کنید عدد $p$ را حدس زدیم. در این صورت اگر مقدار مدنظر از $p$ کوچک‌تر باشد مسئله به $DP[i][p-1]$ تبدیل می‌شود. در صورتی‌که مقدار مدنظر از $p$ بزرگتر باشد نیز مسئله به $DP[p+1][j]$ تبدیل می‌شود. از آنجا که ما نمی‌دانیم کدام یک از این حالات پیش می‌آید باید بدترین سناریوی ممکن را در نظر بگیریم، یعنی در صورت انتخاب عدد $p$ مینیمم نمره مورد نیاز برای تضمین شدن برد به شکل زیر خواهد بود:

$$minimum\,points\,for\,winning\,when\,choosing\,pivot\,p = p + max(DP[i][p-1], DP[p+1][j])$$

حال برای اینکه $DP[i][j]$ را پیدا کنیم بین تمام مقادیر ممکن $p$ مینیمم می‌گیریم:

$$DP[i][j] = min_{p\in[i,j]}(p\ +\ max(DP[i][p-1], DP[p+1][j]))$$

- **پاسخ** $DP$: پاسخ مسئله در $DP[1][n]$ قرار دارد.
- **پیچیدگی الگوریتم:** به ازای پر کردن مقدار $DP[i][j]$ باید تمام مقادیر بین آن را برای حدس زدن امتحان کنیم، پس پیچیدگی پر کردن هر خانه جدول دیپی $O(n)$ است و در نتیجه پیچیدگی زمانی مسئله $O(n^3)$ می‌شود.

## ۶. روباه جادویی ۲۰ نمره

شما یک جادوگر هستید که یک روباه جادویی را در جنگل افسون‌شده هدایت می‌کنید تا به انبار طلسم‌ها برسد. روباه از موقعیت 0 شروع می‌کند و مقدار پرش اولیه آن $+1$ است. مسیر در هر دو جهت به طور بی‌نهایت ادامه دارد و روباه می‌تواند به جلو یا عقب بپرد.

روباه به دو نوع طلسم جادویی واکنش نشان می‌دهد:

- **طلسم امید (H):** روباه به اندازه مقدار پرش کنونی به جلو می‌پرد و مقدار پرش دو برابر می‌شود:

```
position += jump

jump *= 2
```

- **طلسم بازگشت (B):** جهت پرش تغییر می‌کند:
  - اگر مقدار پرش مثبت باشد، به $-1$ تغییر می‌یابد.
  - اگر مقدار پرش منفی باشد، به $+1$ تغییر می‌یابد.

برای مثال، با دنباله طلسم «HBH»، روباه به صورت زیر حرکت می‌کند:

۱. شروع در موقعیت 0، مقدار پرش 1

۲. **H**: پرش به موقعیت 1 (مقدار پرش به 2 می‌رسد)

۳. **B**: تغییر جهت (مقدار پرش به $-1$ تغییر می‌کند، موقعیت در 1 باقی می‌ماند)

۴. **H**: پرش به موقعیت 0 (مقدار پرش به $-2$ می‌رسد)

هدف، کمینه‌سازی تعداد طلسم‌های استفاده‌شده برای رساندن روباه به مقصد است. الگوریتمی برای پیدا کردن این دنباله ی کمینه از طلسم ها با پیچیدگی زمانی $\mathcal{O}(T \log T)$ ارائه کنید ($T$ فاصله ی مکان انبار طلسم‌ها از موقعیت اولیه ی روباه است).

**پاسخ :**

DP[i] را برابر کمترین تعداد طلسم برای رسیدن به نقطه با فاصله ی i در نظر میگیریم. ابتدا اجازه دهید حالات کلی را برای یک عدد $i$ با طول بیت $n$ در نظر بگیریم:

۱. حالت اول: $i = 2^n - 1$. در این حالت بهترین راه این است که با استفاده از $n$ پرش به جلو (طلسم امید) به این نقطه برسیم.

۲. حالت دوم: $2^{(n-1)} - 1 < i < 2^n - 1$. در این حالت دو راه ممکن وجود دارد:

- ابتدا با $n$ پرش به جلو به نقطه $2^n - 1$ برسیم و سپس با یک پرش به عقب (طلسم بازگشت (B جهت را تغییر دهیم. در این حالت $n + 1$ عمل لازم است (شامل $n$ طلسم امید و یک طلسم بازگشت). حالا باید مابقی مسیر را از $2^n - 1$ به $i$ طی کنیم که برابر است با $DP[2^n - 1 - i]$.
- ابتدا با $n - 1$ پرش به جلو به نقطه $2^{(n-1)} - 1$ برسیم، سپس با یک طلسم بازگشت جهت را تغییر دهیم و با $m$ پرش به عقب برویم و دوباره جهت را تغییر دهیم. در این حالت $n + m + 1$ عمل لازم است (شامل $n - 1$ پرش امید، دو طلسم بازگشت و $m$ پرش امید). در نهایت موقعیت به $2^{(n-1)} - 2^m$ می‌رسد و مابقی مسیر از این نقطه به $i$ برابر است با $DP[i - 2^{(n-1)} + 2^m]$.

با این استدلال، فرمول بازگشتی برای برنامه‌ریزی پویا به صورت زیر است:

$$DP[i] = \min(n + 1 + DP[2^n - 1 - i], n + m + 1 + DP[i - 2^{(n-1)} + 2^m])$$

و برای حالت پایه، برای رسیدن به نقطه با فاصله ی ۰، به صفر طلسم نیاز داریم:

$$DP[0] = 0$$

و در نهایت، جواب برابر DP[T] خواهد بود.