به نام خدا

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

تمرین کتبی دوم
موعد تحویل: دوشنبه ۱۵ فروردین ۱۴۰۱، ساعت ۲۳:۵۹
طراح: امیرحسین عباسکوهی amirhossein.abaskohi@gmail.com

---

۱. (۲۰ نمره) با توجه به روی بورس بودن ارز های دیجیتال هادی تصمیم گرفته است تا از این طریق به سود برسد. او قیمت اتریوم در n روز از سال را به صورت آرایه $p_1, p_2, ..., p_n$ دارد. الگوریتمی پویا با مرتبه زمانی $O(n^2)$ ارائه دهید تا بیشترین سودی که می توان با خرید و فروش های متوالی در روز های مختلف به دست آورد را محاسبه کند. دقت کنید که هر روز حداکثر یک اتریوم می توانیم داشته باشیم، در حقیقت زمانی می توان یک اتریوم خرید که در حال حاضر اتریومی نداشته باشیم. این سوال را با استفاده از الگوریتم بازگشتی حافظه دار نیز حل نمایید.

پاسخ: maxprof(i) را به این صورت تعریف میکنیم که تا روز i حداکثر سودی که می توان به دست آورد چقدر می تواند باشد. بدیهی است با هیچ خریدی و فروشی، هیچ سودی حاصل نمی گردد و تا روز اول نیز سودی نخواهیم داشت.

برای آپدیت کردن این تابع باید به ازای خرید در روز های قبلی i و فروش در روز i می توان حداکثر سود را محاسبه نمود و همچنین حالت پایه آن روز اول است که در آن نمیتوان سودی برد.

if $i = 0$ **then return** 0
end if
if $dp_i \neq null$ **then return** dp_i
end if
$dp_i := maxprof(i-1)$
for $j := 1$ to $i-1$ **do**
  $dp_i = max(dp_i, maxprof(j) + a_i - a_j)$
end for
return dp_i

۲. (۱۵ نمره) علی که مدیر تیم استارتاپی خود است، برای افزایش انرژی تیم تصمیم گرفته با تایم هر ماه یکبار به کوه نوردی بروند. متاسفانه اعضای تیم در برنامه ریزی افتضاح هستند که به این ترتیب یک روز کاملا تصادفی را برای کوه نوردی انتخاب می کنند و هیچ خبری هم تا قبل از روز مشخص شده به علی نمی دهند. متاسفانه علی هم باید قبل از رفتن با کوه نوردی، برنامه را با سرمایه گذار پروژه هماهنگ کند تا مرخصی بگیرند. این موضوع باعث می شود که علی هر روز برای روز بعد خود تصمیم بگیرد که آیا باید از مرخصی خود استفاده کند یا خیر.

فرض کنید علی فقط k روز برای مرخصی در m ماه بعدی دارد. بیشترین امید ریاضی تعداد روز هایی که علی می تواند همراه با تیمش به کوه نوردی برود چه مقداری است؟ برای سادگی تمامی ماه ها را ۳۰ روزه در نظر بگیرید(پیچیدگی زمانی راه حل شما باید $O(km)$ باشد). شبهه کد پاسخ خود را بنویسید.

پاسخ: ما زیر مسئله $E(i,j,l)$ را به عنوان به بیشترین امید ریاضی ممکن برای بیرون رفتن برای کوه نوردی در حالی که i ماه باقی مانده باشد(با احتساب ماه کنونی)، j روز از ماه جاری باقی مانده باشد و l تعداد مرخصی های علی باشد.

برای حالت پایه بدیهی است که $E(i,j,0) = 0$، $E(0,1,1) = 1$ و $E(1,1,1) = 1$..

حال باید دقت کنیم امید ریاضی دو بخش مختلف را باید بر اساس وزن آن ها به یک امید ریاضی واحد تبدیل کنیم. از این موضوع در پیاده سازی بخش به روز رسانی جدول برنامه ریزی پویا استفاده می کنیم.

برای تقسیم بندی و ارتباط دادن زیر مسئله ها به هم باید توجه کنیم که بر اساس اینکه علی برای یک روز مرخصی اش را استفاده می کند یا خیر تقسیم بندی انجام می شود.

بنابراین توضیحات الگوریتم به صورت زیر خواهد بود:

for $i := 0$ to m **do**
  for $j := 0$ to 30 **do**
    for $l := 0$ to k **do**
      $E_1(i,j,l) = \frac{1}{j}(1 + E(i-1,30,l-1)) + \frac{j-1}{j}E(i,j-1,l-1)$
      $E_2(i,j,l) = \frac{1}{j}E(i-1,30,l) + \frac{j-1}{j}E(i,j-1,l)$
      $E(i,j,l) = max(E_1(i,j,l), E2(i,j,l))$
    end for
  end for
end for
return $E(m,30,k)$

در اینجا E_1 نشان دهنده بیشترین امید ریاضی کوه نوردی رفتن است اگر علی از مرخصی خود استفاده بکند. E_2 نیز نشان دهنده بیشترین امید ریاضی کوه نوردی رفتن است اگر علی از مرخصی خود استفاده نکند.به طبع در هر روز تصمیمی گرفته می شود که بیشترین امید ریاضی کوه نوردی رفتن را داشته باشیم، بنابراین رابطه $E(i,j,l) = max(E_1(i,j,l), E2(i,j,l))$ رابطه کاملا درستی می باشد.

۳. (۲۰ نمره) فرض کنید به شما یک درخت با n راس به همراه m یال(جدای از یال های درخت) که هزینه هر کدام از آن ها مشخص است، به شما داده شده است. میخواهیم بیشترین هزینه این یال هایی که می توانند به درخت یاد شده اضافه بشوند به طوری که در گراف نهایی دور به طول زوج تشکیل نشود را پیدا کنیم. راه حل خود را بر اساس روش برنامه ریزی پویا با محدودیت زمانی $O(n^2m)$ ارائه دهید.

پاسخ: برای حل سوال درخت را از برگ ها به سمت ریشه مورد بررسی قرار می دهیم و از زیر درخت ها برای به روز رسانی اطلاعات برای راس پدر استفاده می کنیم به این ترتیب نقدار به دست آمده برای ریشه جواب خواهد بود. $dp[i]$ را به صورتی تعریف می کنیم که این خانه در واقع پاسخ مسئله برای زیر درخت با ریشه راس i ام است.

برای داشتن دور های فرد باید برای هر دو دور موجود اشتراک یال نداشته باشیم زیرا در غیر اینصورت دو زوج در اثر آن اشتراک تشکیل خواهد شد.

حال برای به روز رسانی جدول باید دقت کنید که جواب برای هر راس برابر است با جمع جواب برای زیر درخت های موجود به طوری که فرزندان آن راس، ریشه آن زیر درخت ها باشد، و تعداد یال هایی که بین این زیر درخت ها میتوان اضافه به طوری که شرط یال مشترک ایجاد نشود.دور های جدید باید شامل راس i که داریم جدول را برای آن به روز رسانی می کنیم باشد چون در غیر اینصورت شرط یاد شده برقرار نخواهد بود.

این پاسخ تمامی حالات به جز حالتی که بین دور ها راس مشترک داشته باشیم را حساب می کند. برای محاسبه حالت راس مشترک یک $dp2[i][j]$ تعریف می کنیم که هر خانه در واقع برابر است یا پاسخ برای راس i بدون در نظر گرفتن فرزند j ام.

با توجه به اینکه برای هر یال در نهایت این بررسی برای آرایه دو بعدی انجام خواهد شد بنابراین این پاسخ از مرتبه زمانی $O(mn^2)$ خواهد بود.

۴. (۱۵ نمره) به ما یک عدد n و یک آرایه به طول m از اعداد بین ۱ و $2n$ داده شده است. میخواهیم تعداد تمامی حالاتی که که میتوان تعداد $2n$ پرانتز های باز و بسته را به صورت متعادل کنار هم(هر پرانتز باز بسته شده باشد) قرار داد به طوری که در خانه های آرایه داده شده فقط پرانتز های باز قرار بگیرید را پیدا کنیم.

این مسئله را با استفاده از روش برنامه ریزی پویا با محدودیت زمانی $O(n^2)$ و محدودیت حافظه $O(n^2)$ حل کنید. سپس سعی کنید میزان حافظه مصرفی برنامه خود را به $O(n)$ کاهش دهید.

پاسخ: فرض کنید $dp[i][j]$ تعداد حالت های معتبر برای پر کردن i مکان اول باشد به طوری که تعداد پرانتز های باز j تا بیشتر از پرانتزهای بسته باشد. روش معتبر در اینجا به این معنی است که این عبارت پیشوندی از یک عبارت پرانتزی معتبر باشد و شرط قرار نگرفتن پرانتز بسته در آرایه ذکر شده را نقض نکند. با این توضیحات پاسخ کلی مسئله $dp[2n][0]$ پاسخ خواهد بود.

برای حالت پایه هم داریم: $dp[0][0] = 1$ زیرا که باید اولین خانه را با پرانتز باز شروع کنیم که این تنها حالت مجاز است.

الگوریتم زیر نحوه به روز رسانی جدول را به ما نشان می دهد. دقت کنید که pos همان آرایه داده شده توسط سوال است.

```
dp[0][0] = 0
for i := 0 to k do
    h[pos[i]] = 1
end for
for i := 1 to 2n do
    for j := 0 to 2n do
        if h[i] then
            if j ≠ 0 then
                dp[i][j] = dp[i − 1][j − 1]
            else
                dp[i][j] = 0
            end if
        else
            if j ≠ 0 then
                dp[i][j] = dp[i − 1][j − 1] + dp[i − 1][j + 1]
            else
```

$dp[i][j] = dp[i-1][j+1]$
end if
end if
end for
end for
return $dp[2n][0]$

برای کاهش مرتبه زمانی دقت کنیم که ما برای به روز رسانی جدول از مقادیر $dp[i-1][j-1]$ و $dp[i-1][j+1]$ استفاده می شود. پس چون فقط به سطر های قبلی برای به روز رسانی جدول نیاز داریم، می توان با استفاده از متغیر های ثابت مرتبه حافظه این مسئله را به $O(n)$ کاهش داد.

۵. (۱۵ نمره) فروشنده ای یک مغازه پنیر فروشی دارد. در فروش پنیر مسئله ای که وجود دارد این است که هر چه قدر که از زمان آماده شدن پنیر بگذرد، پنیر کیفیت و ارزش بالاتری را پیدا می کند. فروشنده در مغازه خود n پنیر دارد. این فروشنده هر سال یکی از پنیر های خود از اول یا آخر(چپ ترین و یا راست ترین) را می فروشد. قیمت پنیر i ام برابر c_i است و قیمت آن پس از x سال x برابر می شود. با استفاده از روشی پویا در مرتبه زمانی $O(n^2)$ و میزان حافظه مصرفی $O(n^2)$ راه حلی برای این مسئله بیابید که بیشترین سود فروشنده و پنیر هایی که باید در هر سال بفروشد را مشخص کند. در ادامه مسئله را با استفاده از روش بازگشتی حافظه دار نیز حل کنید و شبهه کد پاسخ خود را بنویسید.

پاسخ: برای حل این مسئله از راه حل بازگشتی حافظه دار استفاده می کنیم. dp را آرایه ای شامل اعداد صحیح به اندازه $N \times N$ تعریف می کنیم که $dp[i][j]$ در آن مشخص کننده بیشترین فروش ممکن برای پنیر های $i+1$ام تا $j+1$ است. این آرایه در ابتدا دارای مقادیر -1 است که مشخص می کند جواب برای زیر مسئله پیدا نشده است.
برای حل این مسئله رابطه $maxProfit(i,j)$ را در نظر می گیریم که بیشترین فروش ممکن برای پنیر های $i+1$ ام تا $j+1$ ام باز می گرداند. طبق این تعریف جواب نهایی مسئله برای بیشترین میزان فروش $maxProfit(0, n-1)$ است.
حالت بازکشتی این رابطه برای دو حالت فروش پنیر سمت چپ و فروش پنیر سمت راست و جوتب برای حالات دیگر به دست می آید که جواب نهایی بیشینه جوای این دو حالت است.
حالت پایه این رابطه وقتی رخ می دهد که فقط یک پنیر داشته باشیم که در این صورت میزان فروش پنیر در نظر گرفته می شود.
برای نمایش فروش کدام پنیر در هر سال نیز از آرایه کمکی $sell$ با اندازه NN شامل مقادیر ۰ و ۱ استفاده می شود که در آن $sell[i][j]$ اگر ۰ باشد به معنای فروش عتیقه سمت چپ و اگر ۱ باشد به معنای فروش عتیقه سمت راست است.

if $dp[i][j] \neq -1$ **then return** $dp[i][j]$
end if
$year = n - (j-1)$
if $j = i$ **then return** $year * c[i+1] + maxProfit(i+1, j)$
end if

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

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

if $x \geq y$ **then**
    $sell[i][j] = 0$
else
    $sell[i][j] = 1$
end if
return ans

۶. (۱۵ نمره) حسابداری قصد دارد برای افزایش سرعت محاسبه خود، از این به بعد در هنگام محاسبات دست خود را از روی کیبورد بر ندارد. او برای تسلط به اینکار برای خود تمرینی تعبیه کرده است. او قصد دارد تمامی حالاتی که می تواند رشته هایی به طول n با شروع از هر کدام از کلید ها موجود را تنها با حرکت به چپ، بالا، راست و پایین حرکت کند بسازد. روشی پویا برای این مسئله ارائه دهید تا تعداد رشته هایی که می توان ساخت را بتوان محاسبه کرد. در ابتدا مسئله را با محدودیت حافظه $O(n)$ حل کنید و سپس حافظه را به $O(1)$ کاهش دهید. صفحه کلید را ۳ در ۳ از اعداد ۱ تا ۹ در نظر بگیرید.(مرتبه زمانی راه حل شما باید $O(n)$ باشد.)

پاسخ:برای حل سوال صفحه کلید به صورت شکل ۱ در نظر گرفته شده است.

راه حل بازگشتی این مسئله بسیار ساده است اما پیمایش های مکرر زیادی انجام می شود. برای مثال برای طول مسیر ۴ با نقطه شروع ۴ و ۸ چندین مسیر تکراری با طول ۲ دارد(برای مثال ۴ به ۱).

برای حل مسئله حافظه $coun[i][j]$ را به عنوان حافظه در نظر میگیریم. که i نشان دهنده دکمه شروع مسیر و j نشان دهنده طول مسیر می باشد. پایه ما برای تمامی نقاط شروع برای مسیر طول ۱ برابر ۱ و برای مسیر به طول ۰ برابر ۰ می باشد.
حال از الگوریتم زیر برای آپدیت جدول استفاده می کنیم:

$row[] = 0, 0, -1, 0, 1$
$col[] = 0, -1, 0, 1, 0$
for $k := 2$ to n **do**
    for $i := 0$ to 3 **do**
        for $j := 0$ to 3 **do**
            $num = keypad[i][j] -' 0'$
            $count[num][k] = 0$
            for $move := 0$ to 5 **do**
                $ro = i + row[move]$
                $co = j + col[move]$
                if $ro \geq 0 ro \leq 2 co \geq 0 co \leq 2$ **then**
                    $nextNum = keypad[ro][co] -' 0'$
                    $count[num][k] += count[nextNum][k-1]$
                end if
            end for

```
        end for
    end for
end for

totalCount = 0
for i := 1 to 9 do
    totalCount+ = count[i][n]
end for
return totalCount
```

روش برنامه ریزی پویا ای که در بالا گفته شد به حافظه $O(n)$ نیاز دارد و تنها دلیل موضوع حلقه ای است که در این مرتبه زمانی انجام می شود چراکه بقیه جلقه ها در زمان $O(1)$ اجرا می شوند. از طرفی هم میبینیم که مرحله n ام به مرحله $n-1$ نیاز دارد پس نیازی به نگه داری داده از چندین مرحله قبل نیست. پس با توجه به اینکه فقط داده مرحله قبل را میخواهیم ذخیره کنیم، مرتبه حافظه به $O(1)$ کاهش پیدا میکند.

شکل ۱: صفحه کلید ماشین حساب