۱. یک الگوریتم کارا ارائه کنید که با دریافت یک مجموعه $\{x_1, x_2, \ldots, x_n\}$ از نقاط بر روی خط حقیقی، کوچکترین مجموعه از بازه‌های بسته واحد را تعیین می‌کند که شامل تمام نقاط داده شده در مجموعه باشد. اثبات کنید که الگوریتم شما صحیح است.

پاسخ:

بدون تغییر دادن فرض مسئله می‌توانیم فرض کنیم همه نقاط مجموعه به صورت صعودی مرتب شده باشند. حال الگوریتم زیر را ارائه می‌دهیم:

ابتدا i را برابر 1 در نظر بگیرید.

تا زمانی که $i \leq n$:

- نقطه x_i را در نظر می‌گیریم.
- اگر آن نقطه توسط یک بازه بسته پر شده باشد: کاری نمی‌کنیم.
- در غیر اینصورت: یک بازه بسته واحد اضافه می‌کنیم به طوری که نقطه اولیه آن روی نقطه مورد بررسی ما باشد.
- در نهایت i را یکی اضافه می‌کنیم. $i = i + 1$

الگوریتم چرا صحیح است؟ برهان خلف می‌زنیم. فرض کنید الگوریتم اولیه را a بنامیم و از بین بهترین الگوریتم‌ها، آن الگوریتمی که بیشتر به الگوریتم ما شبیه است را $\acute{a}$ بنامیم. بنابر فرض خلف باید جواب نهایی الگوریتم دوم شامل مجموعه های اکیدا کمتری باشد. حال سمت چپ ترین بازه ای که این دو الگوریتم در آن متفاوت‌اند را در نظر بگیرید. یعنی این بازه مشخصا، در الگوریتم دوم از چپ تر از بازه متناظرش در الگوریتم اول شروع شده است. (چرا؟) واضح است که هیچ فایده‌ای ندارد که بازه را بیشتر از نقطه‌ی سمت چپش (x_i) گسترش دهیم، زیرا می‌دانیم که باید آن نقطه را در بر بگیرد و اگر بازه از قبل آن نقطه شروع شده باشد هم بیهوده است. (چرا؟). به سادگی می‌توان به نتیجه رسید که a بهینه است.

۲. مدارات زمان‌بندی یک جزء حیاتی از تراشه‌های $VLSI$ هستند. در اینجا یک مدل ساده از چنین مداری ارائه شده است. یک درخت دودویی متوازن کامل با n برگ در نظر بگیرید، که n یک توان از دو است. هر یال e از درخت دارای طول l_e است که یک عدد مثبت است. فاصله از ریشه تا یک برگ خاص برابر است با مجموع طول تمامی یال‌های موجود در مسیر از ریشه تا آن برگ. ریشه یک سیگنال ساعت تولید می‌کند که در طول یال‌ها به برگ‌ها منتقل می‌شود. فرض می‌کنیم زمانی که طول می‌کشد تا سیگنال به یک برگ خاص برسد، متناسب با فاصله از ریشه تا آن برگ است. حال اگر تمامی برگ‌ها فاصله یکسانی از ریشه نداشته باشند، سیگنال به طور همزمان به برگ‌ها نمی‌رسد، و این یک مشکل بزرگ است. ما می‌خواهیم که برگ‌ها کاملاً همزمان شوند و همگی در یک زمان سیگنال را دریافت کنند. برای رسیدن به این هدف، باید طول برخی از یال‌ها را افزایش دهیم، به طوری که همه مسیرهای از ریشه به برگ‌ها دارای طول یکسان شوند (ما نمی‌توانیم طول یال‌ها را کاهش دهیم). اگر به این هدف برسیم، گفته می‌شود که درخت (با طول‌های جدید یال‌ها) دارای «صفر شیفت» است. هدف ما رسیدن به «صفر شیفت» به گونه‌ای است که مجموع طول تمام یال‌ها تا حد ممکن کوچک باقی بماند. یک الگوریتم ارائه دهید که طول برخی از یال‌ها را افزایش دهد تا درخت حاصل دارای صفر شیفت باشد و مجموع طول یال‌ها تا حد ممکن کوچک باشد.

پاسخ:

برای طراحی یک راه حل بهینه از یک تکنیک به نام Deferred Merge Embedding یا DME استفاده می‌کنیم که یک الگوریتم حریصانه است. فرض کنید v ریشه باشد و v′ و v′′ دو فرزند آن باشند . همچنین d′ را بیشینه فاصله از ریشه تا برگ برای تمام برگ‌هایی که از v′ مشتق می‌شوند و d′′ را بیشینه فاصله از ریشه تا برگ

برای تمام برگ‌هایی که از v'' مشتق می‌شوند در بگیرید. v' و v'' را طوری در نظر بگیرید که $d' \geq d''$ باشد. حالا:

- $d' - d''$ را به طول یال بین v و v'' اضافه می‌کنیم و هیچ مقداری به طول یال بین v و v' اضافه نمی‌کنیم.
- اکنون این رویه را به صورت بازگشتی برای زیردرخت‌های ریشه‌دار در v' و v'' اعمال می‌کنیم.

فرض کنید T درخت دودویی کاملی باشد که مسئله روی آن تعریف شده است. ابتدا دو موضوع را درباره راه حل بهینه بیان و اثبات می‌کنیم و سپس از آن‌ها برای اثبات بهینه بودن DME استفاده می‌کنیم.

۱. فرض کنید w یک گره داخلی در T باشد و e' و e'' دو یال زیر آن باشند. اگر یک راه حل مقداری به طول هر دو یال e' و e'' اضافه کند، آنگاه این راه حل بهینه نیست.

اثبات: فرض کنید $k' > 0$ و $k'' > 0$ به ترتیب به $l_{e'}$ و $l_{e''}$ اضافه شوند و $\delta = \min(k', k'')$ باشد. در این صورت راه حلی که به ترتیب $k' - \delta$ و $k'' - \delta$ به طول این یال‌ها اضافه کند همچنان در تعادل صفر قرار دارد ولی طول کمتری استفاده می‌کند.

۲. فرض کنید w یک گره میانی در T باشد. اگر یک راه حل طول تمام مسیرها از w به برگ‌های زیر w را افزایش دهد آنگاه این راه حل بهینه نیست.

اثبات: فرض کنید $x_1, \dots, x_k$ برگ‌های زیر w باشند. یال‌های e را در زیردرخت زیر w در نظر بگیرید که ویژگی زیر را داشته باشند: راه‌حل طول e را افزایش دهد، اما طول هیچ یالی در مسیر از w به e را افزایش ندهد. مجموعه تمام این یال‌ها را F می‌نامیم. دو ویژگی در F مشاهده می‌کنیم. اول، برای هر برگ x_i، اولین یالی در مسیر w تا x_i که طول آن افزایش یافته است باید متعلق به F باشد(و هیچ یال دیگری در این مسیر نمی‌تواند متعلق به F باشد). بنابراین، دقیقاً یک یال از F در هر مسیر w تا x_i وجود دارد. دوم، $|F| \geq 2$ است.

بگذارید e_w یالی باشد که از والد w وارد w می‌شود. δ را حداقل مقدار طولی که به هر یک از یال‌های F اضافه شده است در نظر بگیرید. اگر δ را از طول اضافه‌شده به هر یال در F کم کنیم و δ را به یال بالای w اضافه کنیم، طول تمام مسیرهای ریشه به برگ بدون تغییر باقی می‌ماند، و بنابراین درخت همچنان تعادل صفر دارد. اما ما مقدار $|F|\delta \geq 2\delta$ را از طول کل درخت کم کرده‌ایم و فقط δ اضافه کرده‌ایم. بنابراین، یک درخت با تعادل صفر و طول کل کمتر داریم.

اثبات نهایی:

هر راه‌حل دیگری را در نظر بگیرید و فرض کنید v گره‌ای در T باشد که راه‌حل در آن به روشی متفاوت از DME طول‌ها را اضافه می‌کند. از نمادگذاری مسئله استفاده کرده و فرض می‌کنیم که $d' \geq d''$.

فرض کنید راه‌حل، δ' را به یال v' و δ'' را به یال v'' اضافه کند.

اگر $(\delta'' - \delta' = d' - d')$، باید $(\delta' > 0)$ باشد، وگرنه راه‌حل دقیقاً همان کاری را انجام می‌دهد که DME انجام می‌دهد؛ در این حالت، طبق (۱) بهینه نیست.

اگر $(\delta'' - \delta' < d' - d')$، آنگاه راه‌حل باید همچنان طول مسیر از v'' به هر یک از برگ‌های آن را افزایش دهد تا درخت بدون انحراف شود؛ بنابراین طبق (۲) بهینه نیست. به طور مشابه،

اگر $(\delta'' - \delta' > d' - d')$، آنگاه راه‌حل همچنان باید طول مسیر v' به هر یک از برگ‌های آن را افزایش دهد تا درخت بدون انحراف شود؛ بنابراین طبق (۲) بهینه نیست.

۳. اخیرا بابک به جمع‌آوری کارت‌های فوتبالی علاقه‌مند شده است. هر نوع کارت یک امتیاز مشخص (از اعداد طبیعی) دارد که آن‌ها را از انواع دیگر متمایز می‌کند. او n کارت از فروشگاه خریداری کرد ولی متوجه شد که متاسفانه در بین آن‌ها کارت‌های مشابه وجود دارد (کمترین امتیاز کارت‌های در دست او برابر ۲ است). او علاقه دارد بیش‌ترین تعداد کارت‌های ناهمسان را داشته باشد پس تصمیم گرفت که تعدادی از کارت‌هایش را با کارت‌های دوستانش معاوضه کند. در این معاوضه هیچ محدودیتی وجود ندارد جز اینکه فقط کارت‌هایی می‌توانند معاوضه شوند که اختلاف امتیاز آن‌ها ۱ باشد و همچنین نمی‌تواند کارت‌هایی که در معاوضه به دست آورده را مجددا معاوضه کند. با یک الگوریتم از مرتبه زمانی $O(n \log n)$ به بابک کمک کنید حداکثر تعداد کارت‌های ناهمسانی که می‌تواند داشته باشد را محاسبه کند.

پاسخ:

کارت‌ها را بر اساس امتیازشان مرتب می‌کنیم. یک متغیر به نام $last$ تعریف می‌کنیم که امتیاز آخرین کارتی که بررسی کردیم در آن نگهداری می‌شود و آن را با صفر مقداردهی اولیه می‌کنیم. امتیاز هر کارت را با s نمایش می‌دهیم. از کارت با کمترین امتیاز شروع به بررسی می‌کنیم. اگر $last < s - 1$ باشد کارت را با کارتی با امتیاز $s - 1$ تعویض می‌کنیم. اگر $last = s - 1$ باشد کارت را نزد خود نگه می‌داریم و اگر $last = s$ باشد کارت را با کارتی با امتیاز $s + 1$ تعویض می‌کنیم. در انتهای بررسی هر کارت مقدار $last$ را به امتیاز کارت در دست به‌روز می‌کنیم. بیش‌ترین تعداد کارت‌های ناهمسان برابر است با تعداد دفعاتی که $last \neq s$ بوده است. هزینه این الگوریتم $O(n \log n + n)$ برای مرتب کردن کارت ها به علاوه بررسی آن‌هاست.

۴. فرض کنید مجموعه $S = \{1,2, \dots ,1000000\}$ را داریم و می‌دانیم که یک زیرمجموعه 101 عضوی از آن به نام A وجود دارد. آیا می‌توان 100 عضو متمایز از S مانند $x_1, x_2, \dots , x_{100}$ پیدا کرد به‌طوریکه تمامی مجموعه‌های به فرم $A + x_i$ دوبه‌دو مجزا باشند (هیچ دوتایی اشتراک نداشته باشند)؟ در هر دو صورت برای کسب نمره پاسخ خود را همراه با دلیل منطقی و اثبات‌محور ارائه دهید.

پاسخ:

ابتدا x_1 را برابر ۱ قرار می‌دهیم. قاعده این است که هر بار که می‌خواهیم که مقدار x_i جدید را مشخص کنیم می‌دانیم $i - 1$ مقدار قبلا مشخص شده اند. یک محدودیتی اینجا ایجاد می‌شود.

نباید هیچ $1 \leq j \leq i - 1$ ای وجود داشته باشد به طوری که $A + x_j$ و $A + x_i$ اشتراک داشته باشند.

مقداری دقیق‌تر به آن نگاه کنیم نباید $a, \acute{a} \in A$ وجود داشته باشند که $a + x_i = \acute{a} + x_j$.

$a + x_i = \acute{a} + x_j \quad \rightarrow \quad x_i = \acute{a} - a + x_j$

خب x_j ها که اعدادی ثابت هستند به ازای مقادیر قبلی. انتخاب $a, \acute{a}$ هم به $101 * 100 = 10100$ طریق قابل انجام است.

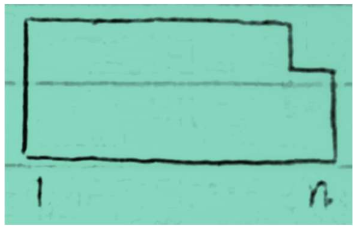
پس می‌توان گفت به ازای هر x_j ثابتی که اضافه می‌کنیم ۱۰۱۰۰ تا از اعداد مجموعه S حذف می‌شوند و نمی‌توان با آن‌ها متغیر x_i جدید و معتبری ساخت. حال $99 * 10100 = 999900$ خواهد بود پس در بدترین حالت ۱۰۰ عدد باقی می‌ماند که همان اعداد $x_1, x_2, \dots , x_{100}$ ای هستند که ما دنبالشان هستیم.

۵. بابک که از کاشی‌کاران خوب کشور است به‌دلیل علاقه شخصی اش به ماهیت علوم مهندسی، به فکر انجام یک تحقیق بین رشته‌ای در حوزه بهینه‌سازی و کاشی‌کاری افتاده است! او که از کاشی‌کاری‌های یکنواخت بازار خسته شده است، می‌خواهد بداند هر زمینی که به فرم $3 * n$ است را به چند روش می‌توان با کاشی‌های به فرم $2 * 1$ یا $1 * 2$ پر کرد؟ به او الگوریتمی ارائه دهید که بتواند این مسئله را حل کند، همچنین الگوریتم خود را تحلیل زمانی کنید.

پاسخ:

فرض کنید جواب سوال (تعداد راه‌های کاشی کاری یک جدول $3 * n$) به ازای n برابر با $f(n)$ باشد.

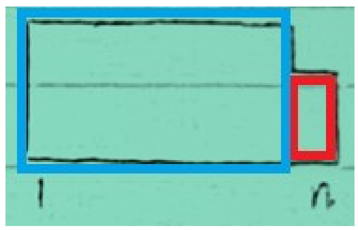
یک تابع دیگر $g(n)$ می‌سازیم که تعداد راه‌های کاشی کاری یک جدول $3 * n$ که یکی از گوشه‌هایش کنده شده باشد را محاسبه کند. پس در این تابع ورودیمان یک جدول $3 * n$ است که یکی از گوشه‌هایش ناقص است.



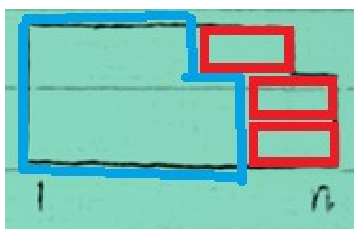
ابتدا بیایید $g(n)$ را محاسبه کنیم.

روی خانه‌ی زیری گوشه بریده شده حالت بندی می‌کنیم:

- اگر به صورت یک کاشی عمودی پر شده باشد به $f(n-1)$ می‌رسیم.



- اگر به صورت افقی پر شده باشد به $g(n-2)$ می‌رسیم.



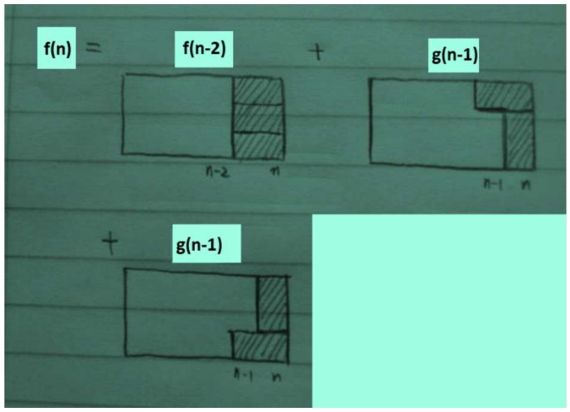
پس $g(n) = f(n-1) + g(n-2) \quad n > 2, \quad g(1) = 1$

حال بیایید $f(n)$ را محاسبه کنیم.

روی خانه گوشه بالاراست آن حالت بندی می‌کنیم:

- اگر به صورت یک کاشی عمودی پر شده باشد به $g(n-1)$ می‌رسیم.
- اگر به صورت افقی پر شده باشد روی خانه زیری آن حالت بندی می‌کنیم:
  - اگر با کاشی عمودی پر شده باشد به $g(n-1)$ می‌رسیم.
  - اگر با کاشی افقی پر شده باشد به $f(n-2)$ می‌رسیم.

پس $f(n) = 2g(n-1) + f(n-2), \; f(0) = 1$



ایده الگوریتم به صورت بازگشتی پیاده می‌شود و می‌توان با یک حلقه for مقادیر f, g را برای ورودی‌های $n, \dots$ و2و1 به راحتی محاسبه کرد. پس پیچیدگی زمانی الگوریتم از $O(n)$ است.

۶. فرض کنید ما یک دایره از N عدد داشته باشیم. یک الگوریتم از $O(N)$ بنویسید که یک بلاک پیوسته روی دایره پیدا کند که بیشترین جمع ممکن را داشته باشد.

پاسخ:

ابتدا فرض کنید N عددی که داریم روی یک دایره نیستند و به ترتیب روی یک خط قرار گرفته اند. مثلا عدد a_1 سپس عدد a_2 بعد ... در این حالت چطور می‌توان یک بلاک پیوسته روی خط را پیدا کرد که بیشترین جمع ممکن را داراست؟

به ازای عدد i ام خروجی تابع $f(i)$ ماکزیمم جمع ممکن برای یک بلاک پیوسته را نشان می‌دهد که در نقطه i متوقف شده باشد. حالت بندی می‌کنیم روی عدد $i-1$ ام:

- اگر آن عدد در بلاک مذکور آمده باشد به $f(i-1)+a_i$ می‌رسیم.
- اگر هم نیامده باشد تنها جواب ممکن a_i است.

حال ماکزیمم این دو جواب، مقدار $f(i)$ را مشخص می‌کند. بنابراین $f(i) = max(f(i-1)+a_i, a_i)$

همینطور تابع g ای می‌سازیم که به ازای عدد i ام خروجی تابع $g(i)$ مینیمم جمع ممکن برای یک بلاک پیوسته را نشان می‌دهد که در نقطه i متوقف شده باشد. حالت بندی می‌کنیم روی عدد $i-1$ ام:

- اگر آن عدد در بلاک مذکور آمده باشد به $g(i-1)+a_i$ می‌رسیم.
- اگر هم نیامده باشد تنها جواب ممکن a_i است.

حال مینیمم این دو جواب، مقدار $g(i)$ را مشخص می‌کند. بنابراین $g(i) = min(g(i-1)+a_i, a_i)$

حال مجموع تمامی اعداد را هم در یک متغیری به نام sum می‌ریزیم؛ بیشترین مقدار $f(i)$ ها را $\acute{f}$ و کمترین مقدار $g(i)$ هارا $\acute{g}$ می‌نامیم.

در نهایت وقت آن رسیده که اعداد را ببریم به همان فرم دایره ای که داشتیم. برای پیدا کردن بلاک پیوسته ماکزیمم، جواب یا برابر با $\acute{f}$ است یا برابر با sum منهای مقدار $\acute{g}$ است!

چرا؟ حالت اول که بهترین حالاتی که بلاک پیوسته به عدد n ام نمی‌رسد (فرم دایره را در نظر بگیرید) را حساب می‌کند و دومی بهترین حالتی که آن بلاک از یک عددی قبل n شروع شده و به صورت ساعتگرد n را دور می‌زند و در یک عددی بعد از n متوقف می‌شود را حساب می‌کند و بدیهی است که بهترین جواب یکی از این دوتاست.

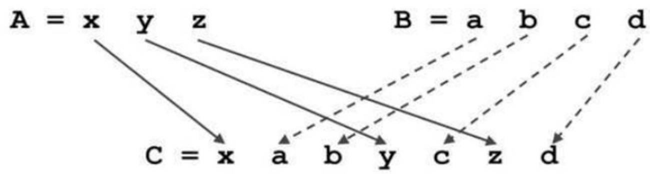
بنابراین جواب نهایی می‌شود $max(\acute{f}, sum-\acute{g})$

۷. گفته می‌شود رشته C درهم‌تنیده‌ی رشته‌های A و B است اگر شامل تمام کاراکترهای A و B باشد و ترتیب نسبی کاراکترهای هر دو رشته در C حفظ شود. به عنوان مثال، اگر مقادیرA، B و C به شرح زیر باشند

$$A = xyz$$

$$B\ =\ abcd$$
$$C\ =\ xabcyzd$$

رشته C درهم‌تنیده‌ی رشته‌های A و B است همان‌طور که در تصویر نشان داده شده است:



با داشتن سه رشته A، B و C، الگوریتمی بنویسید که بررسی کند آیا رشته سوم درهم‌تنیده‌ی دو رشته اول و دوم است یا خیر.

پاسخ:

راه‌حل پویا (Dynamic Solution) شروع به حل مسئله از پایین به بالا می‌کند. در هر مرحله، بررسی می‌کنیم که آیا یک زیررشته از C، درهم‌تنیده زیررشته‌های A و B است یا خیر.

مثال زیر که راه حل سوال به کمک آن بیان می‌شود را در نظر بگیرید:

A = bcc

B = bbca

C = bbcbcac

از یک ماتریس برای حل این مسئله استفاده می‌کنیم به نحوی که یک رشته روی محور افقی و رشته دیگر روی محور عمودی قرار گیرند. تصویر ماتریس مربوط به مثال در ادامه نشان داده شده.

		b	b	c	a
b					
c					
c					

مقدار در خانه (i,j) برابر **True** است اگر **اولین i** کاراکتر از رشته A و **اولین j** کاراکتر از رشته B، درهم‌تنیده(interleave) شوند تا **اولین i+ j** کاراکتر از رشته C را بسازند. هنگام پر کردن ماتریس، اگر در خانه (i,j) باشیم، کاراکتر i+j−1 رشته C را بررسی می‌کنیم.

برای مثال، خانه (1,2) بررسی می‌کند که آیا b (اولین کاراکتر bcc) و bb (دو کاراکتر اول bbca) در هم تنیده می‌شوند تا bbc (سه کاراکتر اول bbcbcac) را بسازند یا خیر که در این حالت پاسخ False است.

خانه (0,0) برابر True است. سطر اول(جز (0,0)) تنها، مشابهت بین زیر رشته B و زیر رشته C را بررسی می‌کند و ستون اول(جز (0,0)) نیز تنها، مشابهت بین زیررشته A و زیر رشته C را بررسی می‌کند.

```
If (B[i - 1] != C[i – 1])
   Matrix[0][i] = False
```

```
Else
  Matrix[0][i] = Matrix[0][i - 1]
If (A[i - 1] != C[i – 1])
  Matrix[i][0] = False
Else
  Matrix[i][o] = Matrix[i - 1][0]
```

		b	b	c	a
	T	T	T	T	F
b	T				
c	F				
c	F				

حالا از ستون دوم و از چپ به راست شروع به پر کردن خانه‌ها می‌کنیم. کاراکتر فعلی A و B را با کاراکتر فعلی C مقایسه می‌کنیم. اگر در حال پر کردن خانه (i,j) هستیم کاراکتر فعلی A، B و C به ترتیب حرف $i - 1$، $j - 1$ و $i + j - 1$ آن‌ها هستند. یکی از ۴ حالت زیر پیش می‌آید:

۱. اگر کاراکتر فعلی C با کارکتر فعلی A یا B برابر نباشد مقدار خانه برابر False می‌شود.
۲. اگر کاراکتر فعلی C برابر کاراکتر فعلی A و متفاوت با کاراکتر فعلی B باشد، مقدار خانه مشابه خانه بالایی اش خواهد بود.
۳. اگر کاراکتر فعلی C برابر کاراکتر فعلی B و متفاوت با کاراکتر فعلی A باشد، مقدار خانه مشابه خانه سمت چپ اش خواهد بود.
۴. اگر کاراکتر فعلی C هم با کاراکتر فعلی A و هم با کاراکتر فعلی B برابر باشد؛ مقدار خانه در صورتی برابر True خواهد بود که هم خانه بالایی و هم خانه سمت چپ آن برابر True باشند. در غیر این صورت False می‌شود.

جدول تکمیل شده مثال را در ادامه مشاهده می‌کنید که گوشه سمت راست پایین آن جواب نهایی مسئله است.

		b	b	c	a
	T	T	T	T	F
b	T	T	F	T	F
c	F	T	T	T	T
c	F	F	T	F	T

۸. قصد داریم از سرزمینی مستطیلی شکل با m ایالت در هر سطر و n ایالت در هر ستون گذر کنیم. برای عبور از هر خانه باید مقدار مشخصی عوارض بپردازیم. قصد داریم از گوشه بالای سمت راست سرزمین وارد و از گوشه پایین

سمت چپ سرزمین خارج شویم و در این حرکت فقط می‌توانیم به چپ، راست و پایین حرکت کنیم و مجاز به حرکت به سمت بالا نیستیم. شبه کدی ارائه کنید که با الگوریتمی بهینه، حداقل پولی که باید برای گذر از این سرزمین همراه داشته باشیم را محاسبه کند.

پاسخ:

به کمک برنامه ریزی پویا و از پایین به بالا مسئله را حل می‌کنیم. کد پاسخ این سوال به همراه این پاسخ‌نامه در اختیار شما قرار گرفته است. حل این مسئله از مرتبه زمانی $O(n^2)$ است.