هوالعلیم

طراحی و تحلیل الگوریتم‌ها

نیمسال دوم سال تحصیلی ۱۴۰۱ – ۱۴۰۰

تمرینات نظری ۵

مریم رضائی

۱. الف) فرض کنید شما کارشناس امنیت رایانه هستید که در شرکتی بزرگ کار می‌کنید و به تازگی، متوجه شده‌اید که بسیاری از رایانه‌های شرکت با بدافزاری آلوده شده‌اند که باید از وبگاه‌های ناپاکی که کاربران دیده‌اند، آمده باشد. برای هر رایانه آلوده، شما یک فایل سابقه دارید که حاوی لیست تمام وبگاه‌هایی است که (از آخرین باری که رایانه برای کشف بدافزارها پویش شده است) از طریق آن رایانه دیده شده‌اند. شما با بررسی فایل‌های سابقه، متوجه می‌شوید که هیچ وبگاهی وجود ندارد که از طریق همه رایانه‌ها دیده شده باشد. بنابراین، نتیجه می‌گیرید که بدافزار را تعدادی از وبگاه‌ها به رایانه‌ها تزریق کرده‌اند و محتمل‌ترین وبگاه‌ها، آنهایی هستند که در کوچک‌ترین مجموعه‌ای از وبگاه‌ها که از طریق همه رایانه‌های آلوده (نه لزوماً هر یک از رایانه‌های آلوده) دیده شده باشند.

آیا مسأله تعیین کمترین تعداد از وبگاه‌هایی که از طریق همه رایانه‌های آلوده دیده شده‌اند، یک مسأله NPC است یا خیر؟

ب) مجموعه $A = \{a_1, a_2, \cdots, a_n\}$ از اعداد صحیح مثبت و اعداد k و b داده شده‌اند و ما می‌خواهیم بدانیم که آیا می‌توان مجموعه A را به k زیرمجموعه دو به دو مجزای $A_1, A_2, \cdots, A_k$ تقسیم کرد به قسمی که «مجموعِ مربعاتِ مجموع اعداد زیرمجموعه‌ها» حداکثر b باشد:

$$\sum_{i=1}^{k}\left(\sum_{a_j \in A_i} a_j\right)^2 \leq b$$

آیا این مسأله، یک مسأله NPC است یا خیر؟

جواب:

الف) بله. برای اثبات NPC بودن آن کافی‌ست مسئله‌ای NPC بیابیم و به این مسئله گسترش دهیم که در آن صورت این مسئله نیز قطعا NPC ست. برای این کار شبکه کامپیوترها را رئوس گرافی در نظر می‌گیریم. آنگاه زمانی که دو کامپیوتر به یک وبسایت یکسان رفته‌اند میان آن دو رأسی در نظر می‌گیریم که برای هر وبسایت رنگ یال خاص است. بدین صورت گرافی داریم که رئوسی از آن که به وبسایتی مشترک رفته‌اند زیرگرافی کامل با رنگی خاص تشکیل می‌دهند. آنگاه کافی‌ست بزرگترین زیرگراف‌های تک رنگی که با هم کل گراف را پوشش می‌دهند بیابیم، که این مسئله خوشه است (بزرگترین زیرگراف کامل) که به مسئله داده شده گسترش داده شده است و بنابراین مسئله NPC است.

ب) بله. به مانند بخش الف به دنبال مسئله‌ای NPC میگردیم که می‌توان آن را به مسئله داده شده گسترش داد تا بنابراین تعریف مسائل NP و NPC، اثبات را انجام دهیم. واضح است مسئله به مسئله افراز (که در آن زیر مجموعه‌هایی با جمع یکسان تشکیل می‌دادیم) شباهت دارد. میبینیم که مسئله افراز در اصل زیر مجموعه‌ای (و نوعی ساده‌تر) از این مسئله است، زیرا در اینجا جمع هر زیرمجموعه می‌تواند مساوی یا نا مساوی با دیگر زیر مجموعه ها باشد. یعنی:

$$k = 2 \; and \sum_{a_j \in A_i} a_j = |A_i|:$$

$$(1)\;\; if\; |A_1| = |A_2| \; then \sum_{i=1}^{k} \left(\sum_{a_j \in A_i} a_j \right)^2 \le b \; \to 2|A_1|^2 \le b$$

$$(2)\;\; if\; |A_1| < |A_2| \; then \sum_{i=1}^{k} \left(\sum_{a_j \in A_i} a_j \right)^2 \le b \; \to |A_1|^2 + |A_2|^2 \le b$$

$$and\; 2|A_1|^2 < |A_1|^2 + |A_2|^2 < 2|A_2|^2$$

$$(3)\;\; if\; |A_1| > |A_2| \; then \sum_{i=1}^{k} \left(\sum_{a_j \in A_i} a_j \right)^2 \le b \; \to |A_1|^2 + |A_2|^2 \le b$$

$$and\; 2|A_1|^2 > |A_1|^2 + |A_2|^2 > 2|A_2|^2$$

که مشاهده می‌کنیم که در همه حالات این مسئله گسترشی از مسئله افراز است. برای مثال در حالت یک (۱) در اصل باید زیرمجموعه‌هایی را بیابیم که مجموع آنها $\sqrt{b/2}$ می‌باشد. بنابراین مسئله داده شده NPC است.

مراجع: ـــــــــ

۲. این گونه از مسأله زمانبندی کارها را در نظر بگیرید: مجموعه‌ای از کارها به ما داده شده است که همه آنها را باید به ترتیبی به انجام برسانیم. انجام هر کاری نیازمند یک واحد زمانی است و هر کاری مهلتی دارد و اگر تا قبل از رسیدن مهلتش انجام نشود، جریمه‌ای باید بابت تأخیر در انجام آن کار بپردازیم. پس هر الگوریتم این مسأله، این مجموعه‌ها را به عنوان ورودی می‌گیرد:

- مجموعه $\{t_1, t_2, \cdots, t_n\}$ از کارهایی که مدت زمان انجام هر یک از آنها یک واحد زمانی است؛
- مجموعه $\{d_1, d_2, \cdots, d_n\}$ ؛ d_i مهلت انجام کار t_i است و $1 \le d_i \le n$ است؛ یعنی کار t_i باید تا زمان d_i به انجام برسد؛
- مجموعه $\{p_1, p_2, \cdots, p_n\}$ ؛ p_i جریمه‌ای است که در صورتی که کار t_i تا زمان d_i به انجام نرسد باید بابت تأخیر در انجام آن کار بپردازیم.

هدف از زمانبندی کارها این است که ترتیبی برای انجام آنها مشخص کنیم به گونه‌ای که همه کارها انجام شوند و جریمه کل کارهای تأخیردار حداقل مقدار ممکن باشد.

الف) الگوریتمی برای این مسأله ارائه کنید.

ب) درستی الگوریتم خود را ثابت کنید و کارایی زمانی آن را اندازه بگیرید.

جواب:

الف) مسئله تمرکز بر اولویت دادن به کارها برای انتخاب دارد، پس می‌توانیم آن را با صف اولویت حل کنیم. پیش از توصیف الگوریتم در نظر داریم که ساختار داده صف اولویت از پیش تعیین شده است که از هرم استفاده می‌کند و کار با آن در زمان $O(\log n)$ (ارتفاع درخت) انجام می‌شود.

توصیف الگوریتم: از آنجا که مهلت انجام کار و جریمه اولویت هر کار را تعیین می‌کند، با این دو به کارها در دو مرحله اولویت می‌دهیم. پس در ابتدا کارها را بر اساس مهلت انجام مرتب می‌کنیم، سپس از آخر (دیرترین کار) به عقب آمده و فاصله بین هر دو مهلت را مقایسه می‌کنیم تا ببینیم چند کار در آن فاصله می‌تواند انجام شود. آنگاه برای هر فاصله، از بین کارهای با مهلت مساوی یا دیرتر کارهایی که بیشترین جریمه دارند را انتخاب می‌کنیم. در انتها پس از بررسی همه کارها و بازه‌ها اگر کارهایی باقی‌مانده بودند که تا مهلتشان انجام نشده بودند، آن‌ها را با جریمه انجام می‌دهیم.

شبه کد: با فرض از پیش تعریف شدن ساختار داده صف اولویت حداکثر با استفاده از درخت، شبه کد زیر را داریم که در آن تابع *push* عضوی جدید به ساختار داده (در جای مناسب) اضافه می‌کند و *pop* بزرگترین عضو را از ساختار داده حذف کرده و برمی‌گرداند.

Algorithm: *Schedule*(*T*, *D*, *P*)

```
    // Finds best scheduling with least loss
    // Input: Three sets T, D, P of n elements holding job ID, deadline, and loss
    // Output: Ordered list of jobs to do without loss, and list of jobs to do with loss

    heap ← MaxHeap() and lossless ← [ ] and withloss ← [ ]
    T, D, P ← MergeSort(T, D, P, based_on = D)   // sort the three based on deadline
    for i in n – 1 to 0 do
        if i = 0 then slots ← D[i]
        else then slots ← D[i] – D[i – 1]
        heap.push(priority = P[i], T[i], D[i])
        while slots and heap.notEmpty do
            loss, id, deadline ← heap.pop()
            lossless.append([id, deadline])
            slots ← slots – 1
    lossless ← MergeSort(lossless, based on = lossless[i][1])  // sort based on deadlines
    while heap.notEmpty then
        withloss.append(heap.pop())
    return lossless, withloss
```

ب) دو مورد را جدا بررسی می‌کنیم.

اثبات درستی: برای اثبات درستی دو سوال در نظر داریم. (۱) آیا الگوریتم تمام کارها در نظر می‌گیرد و یک بار انجام می‌دهد؟ (۲) آیا الگوریتم برنامه‌ریزی با کمترین جریمه را پیدا می‌کند؟ برای اثبات کامل هر دو سوال را پاسخ می‌دهیم.

۱) **پوشا بودن پاسخ:** الگوریتم در اولین حلقه در تمامی کارها پیش می‌رود و هر کدام را به هرم اولویت اضافه می‌کند، سپس تا جایی که هر بازه اجازه می‌دهد از هرم یکی را حذف و به پاسخ اضافه می‌کند. در نهایت الگوریتم کارهای مانده به هرم را به پاسخ اضافه می‌کند. پس هر کار قطعا یک بار وارد هرم شده و یک بار خارج می‌شود. پس الگوریتم هر کار را قطعا یک بار انجام می‌دهد.

۲) **بهترین بودن پاسخ:** در نظر می‌گیریم پاسخی بهتر با جریمه کمتر وجود دارد، یعنی یکی از کارهایی که بعد از مهلت انجام شده‌اند و جریمه داشته‌اند می‌توانسته‌اند در مهلتشان انجام شود. برای این امر دو تغییر در پاسخ ممکن است: (۱) یا بازه‌ای ممکن برای انجام آن کار در زمانش باز بوده که الگوریتم در نظر نگرفته‌است، (۲) و یا کار به جای یک کار انتخاب شده بدون جریمه می‌توانسته انجام شود که هزینه جریمه کمتری داشته‌است.

۱. از آنجا که ابتدا کارها به ترتیب مهلت مرتب می‌شوند و سپس به همان ترتیب در نظر گرفته می‌شوند (کار با بیشترین مهلت ابتدا وارد هرم می‌شود و در صورت زود خارج نشدن برای مراحل بعد با مهلت کمتر در هرم می‌ماند که اجازه آن را دارد) هر بار کارهایی که می‌توانند در یک بازه زمانی اجرا شوند (مهلتشان بیشتر یا مساوی زمان پایان حال حاضر است) در نظر گرفته می‌شوند و تا جای ممکن (تا جایی که $slot$ موجود باشد) کارها انتخاب می‌شوند. پس اگر جا برای کاری در یک بازه زمانی بود قطعا آن انتخاب می‌شد. پس این حالت ممکن نیست.

۲. هرم اولویت هر بار کارهایی که انتخاب نشده‌اند و بازه زمانی حال حاضر بخشی از مهلتشان است را نگه می‌دارد و در آن بازه از بین کارها و تا جای ممکن کارهایی که جریمه بیشتری را دارند انتخاب می‌کند تا در مهلتشان انجام شوند. بنابراین اگر کاری با جریمه انجام شده جریمه‌ای کمتر از یک کار انتخاب شده برای انجام بی جریمه داشت قطعا در بازه زمانی ممکنش بررسی شده و انتخاب می‌شد. پس این حالت نیز ممکن نیست.

بنابراین فرض خلف باطل است و اثبات کامل است.

کارایی زمانی: برای کارایی زمانی چند بخش الگوریتم را جدا محاسبه کرده و با هم جمع می‌کنیم.

۱) ساخت متغیرهای خالی: $O(1)$
۲) مرتب سازی کارها بر اساس مهلت: $O(n\log n)$
۳) حلقه بررسی یک بار هر کار: $O(n)$

۱. مقدار دهی متغیر تعداد ممکن: $O(1)$
۲. قرار دادن کار در هرم: $O(\log n)$ (در بدترین حالت که تمام کارها در هرم باشند اتفاق می‌افتد، که این نمی‌تواند بیش از یک بار اتفاق بی‌افتد زیرا هر کار یک بار به آن اضافه می‌شود و در هر تکرار حلقه در بدترین حالت که هیچ عنصری از هرم حذف نشود، تعداد عناصر درخت به تعداد $n-i$ خواهد بود.)
۳. حلقه انتخاب کارها تا جای ممکن: $O(1)$ (به طور میانگین است، زیرا تکرار حلقه برابر است با تعداد کار انتخاب شده و می‌دانیم هر کار تنها یک بار انتخاب می‌شود. و از آنجا که در بدترین حالت که همه کارها در حلقه for مخصوص به بررسی کارها انتخاب شوند، به طور میانگین برای هر کدام از تکرارهای حلقه بیرونی که n است، این حلقه یک بار اجرا می‌شود.)

i. حذف از هرم اولویت: $O(\log n)$ (مثل قرار دادن در هرم است.)
ii. اضافه به لیست و کم کردن از متغیر: $O(1)$

۴) مرتب سازی کارهای انتخاب شده بر اساس مهلت: $O(n\log n)$
۵) گذاشتن کارهای انتخاب نشده در آرایه‌ای: $O(n)$ (در اصل تکرار برابر n منهای تعداد انتخاب شده‌هاست.)

پس برای کارایی زمانی کل بر اساس بخش‌های مستقل و وابسته الگوریتم داریم:

$$C(n) = O(1) + O(n\log n) + O(n) \times \Big(O(1) + O(\log n) + O(1) \times \big(O(\log n) + O(1)\big)\Big) + O(n\log n)$$
$$+ O(n) = O(n\log n) + O(n) + O(n\log n) + O(n\log n) + O(n\log n) + O(n)$$
$$= O(n\log n)$$

مراجع:

- **https://www.geeksforgeeks.org/job-sequencing-problem**

۳. این بازی راهبردی دو نفره را در نظر بگیرید: با این فرض که n عددی زوج باشد، n سکه با ارزش‌های متفاوت یا یکسان در یک ردیف گذاشته شده‌اند. دو بازیکن به نوبت و هر بار یکی از سکه‌ها را از یکی از دو انتهای (راست یا چپ) ردیف باقیمانده سکه‌ها برمی‌دارند. بازیکنی که در نهایت، سکه‌هایی را با مجموع ارزش بیشتر برداشته باشد برنده بازی خواهد بود.

اگر فرضاً راهبرد هر دو بازیکن در انتخاب سکه، بهینه باشد؛ یعنی در هر حرکت خود، سکه باارزش‌تر را انتخاب کنند، الگوریتمی برای بازیکن اول ارائه کنید تا با آن، مجموع ارزش سکه‌هایی را که در طول بازی برمی‌دارد به حداکثر برساند. کارایی زمانی الگوریتم خود را هم تعیین کنید.

جواب:

توصیف الگوریتم: با استفاده از برنامه‌نویسی پویا می‌توانیم مسئله را به n^2 زیر مسئله شکسته و پاسخ هر کدام را ابتدا بر اساس حالات پایه و سپس بر اساس حالات قبل تشکیل دهیم. در نظر داریم که قصد بیشینه کردن امتیاز کل است که این امر حتما با بیشینه کردن هر انتخاب برای خود اتفاق نمی‌افتد. مثلا فرض کنیم سکه‌ها به ترتیب ۷، ۳، ۱۵، ۸ باشند. اینجا به جای انتخاب ۸ در ابتدا، باید ۷ انتخاب شود تا حریف کمترین امتیاز ممکن را از انتخاب این سطح داشته باشد (نتواند به ۱۵ دسترسی داشته باشد). پس برای بیشینه کردن امتیاز کل باید در سطح انتخابی کنیم که کمترین امتیازها را در انتخاب حریف بگذارد و حریف نیز به همین منوال کاری می‌کند که در سطح بعد انتخاب ما بین کمترین عناصر ممکن باشد. بنابراین می‌توانیم انتخاب ممکنمان در سطح را به دو حالت تقسیم کنیم:

۱) اگر از چپ (عنصر i اُم) انتخاب کنیم آنگاه برای حریف عناصر $i+1$ و j دو سر صف برای انتخاب می‌مانند و می‌دانیم که او از بین این دو آن عنصری را انتخاب می‌کند که انتخاب بعد ما کمترین مقدار ممکن باشد. یعنی اگر با انتخاب $i+1$ برای ما عناصر $i+2$ و j کمترین انتخابند، پس آن را از چپ انتخاب می‌کند. بنابراین رابطه بازگشتی انتخاب ما در هر سطح کلی با در نظر گرفتن اجبار حاصل از حرکت حریف برابر است با:

$$F(i,j)_{left} = C_i + \min\big(F(i+2,j), F(i+1,j-1)\big)$$

۲) اگر از راست (عنصر j اُم) انتخاب کنیم به همین منوال انتخاب سطح بعدمان بین دو جفت عنصر $i+1$ و $j-1$ یا i و $j-2$ به طوری باشد که کمترین امتیاز را بگیریم. پس رابطه سطح را در صورت انتخاب از راست داریم:

$$F(i,j)_{right} = C_j + \min\big(F(i+1,j-1), F(i,j-2)\big)$$

بنابراین بهترین انتخاب در هر سطح (یعنی بین راست و چپ) آن است که بیشترین امتیاز را برای ما داشته باشد، یعنی:

$$F(i,j) = \max\big(F(i,j)_{left}, F(i,j)_{right}\big)$$

در نظر داریم که رابطه بازگشت حالات پایه‌ای دارد که به علت شکست مسئله به تمام زیر مسئله‌های ممکن (از هر عنصر تا هر عنصر دیگر برای انتخاب موجود باشد)، پس امکان دارد یک تنها یک عنصر یا دو عنصر داشته باشیم. یعنی:

$$\xrightarrow{base\ case}\begin{cases} F(i,j) = C_i & , \quad i = j \\ F(i,j) = \max(C_i, C_j), & j = i+1 \end{cases}$$

علت وقوع این دو حالت پایه را با تعیین زیر مسئله‌ها به کمک مثالی بهتر می‌بینیم. اگر ماتریس جواب‌های این مسئله‌ی برنامه‌نویسی پویا را $n \times n$ در نظر گیریم که هر بعد وجود عنصری تا عنصری دیگر را تعیین می‌کند (یعنی جایگاه ۱ در ۳ بهترین جواب مسئله در حالتی که عناصر ۱ تا ۳ در بازی باشند را نگه می‌دارد) آنگاه با توجه به اینکه ۱ تا ۳ با ۳ تا ۱ تفاوتی ندارد تنها کافی مثلث بالایی ماتریس را پر کنیم. برای این ماتریس، قطر امتیازهای هر سکه است زیرا جایگاه ۲ در ۲ فقط جواب برای وجود تک عنصر ۲ را نگه می‌دارد. بر این اساس ماتریس مثال داده شده به صورت زیر است:

$$\begin{bmatrix} 8 & 15 & 11 & 22 \\ & 15 & 15 & 18 \\ & & 3 & 7 \\ & & & 7 \end{bmatrix}$$

برای پر کردن این ماتریس در نظر داریم که مانند دیگر مسائل برنامه‌نویسی پویا از حالت پایه شروع کرده و به کامل‌ترین جواب پیش می‌رویم. پس اول با حالت پایه تک عضوی بودن قطر را پر کرده سپس با حالت دو عضوی بودن قطر بالایی کوچک‌تر (بعد ماتریس یکی کمتر) را پر می‌کنیم و در نهایت سطوح بعد را از آن‌ها تشکیل می‌دهیم. در این صورت آخرین عنصر سطر اول جواب بهینه را دارد زیرا از $i = 1$ تا $j = n$ در صف در نظر گرفته می‌شوند.

اگر بخواهیم انتخاب‌های هر سطح (یعنی چه عنصری، عنصر راست یا چپ) را نیز ذخیره کنیم کافی‌ست بُعد دیگری به ماتریس اضافه کنیم که هر انتخابمان را نگه می‌دارد. طول این بعد در بیشترین حالت $n/_2$ اضافه تر است زیرا حداکثر این تعداد انتخاب می‌توانیم داشته باشیم. آنگاه در حالات پایه عنصر انتخاب شده را در بعد سوم ذخیره می‌کنیم و برای ساختن سطوح بعد در حین تشکیل امتیاز از سطوح قبلی، انتخابات آن‌ها را نیز کپی می‌کنیم.

شبه کد الگوریتم در صفحه بعد قابل مشاهده است.

کارایی زمانی: تعداد زیر مسئله‌های بررسی شده به تعداد خانه‌هایی هستند که از ماتریس $n \times n$ پر می‌کنیم، و آن برابر تعداد خانه‌های بالا مثلثی ماتریس است. پس برای محاسبه کارایی زمانی با این اساس که حل هر زیرمسئله زمان ثابت می‌برد، کافی‌ست تعداد خانه‌های ماتریس بالا مثلثی (با در نظر گرفتن قطر) را به دست آوریم. مشاهده می‌کنیم که این تعداد در شبه کد در دو حلقه تو در تو دیده می‌شود که حلقه درونی به بیرونی وابسته است. برای محاسبه هر دو مقدار رابطه یکسانی وجود دارد که به شکل زیر است:

$$\sum_{i=1}^{n}\sum_{j=i}^{n} 1 = \frac{n(n+1)}{2} \in O(n^2)$$

Algorithm: ***CoinGameStrategy*(C[1 … n])**

```
// Finds optimal strategy to get the highest value in the coin game
// Input: Array C of n coin values
// Output: A list with the highest possible value as first element, and choices as the rest

matrix ← Matrix(2, n, n, [0])   // initialises square matrix n×n with elements as a list [0]
for gap in 0 to n – 1 do
    for j in gap to n – 1 do
        i ← j – gap
        f1 ← 0 and f1choices ← [ ]   // max value and choices of F(i + 2, j)
        if i + 2 ≤ j then
            f1 ← matrix[i + 2][j][0] and f1choices ← matrix[i + 2][j][1:]
        f2 ← 0 and f2choices ← [ ]   // max value and choices of F(i + 1, j − 1)
        if i + 1 ≤ j – 1 then
            f2 ← matrix[i + 1][j – 1][0] and f2choices ← matrix[i + 1][j – 1][1:]
        f3 ← 0 and f3choices ← [ ]   // max value and choices of F(i, j − 2)
        if i ≤ j – 2 then
            f3 ← matrix[i][j – 2][0] and f3choices ← matrix[i][j – 2][1:]
        left ← C[i] + min(f1, f2) and right ← C[j] + min(f2, f3)
        if left ≥ right then
            matrix[i][j] ← [left, C[i]]
            if f1 ≤ f2 then matrix[i][j] ← matrix[i][j] + f1choices
            else then matrix[i][j] ← matrix[i][j] + f2choices
        else then
            matrix[i][j] ← [right, C[j]]
            if f2 ≤ f3 then matrix[i][j] ← matrix[i][j] + f2choices
            else then matrix[i][j] ← matrix[i][j] + f3choices
return matrix[0][n – 1]
```

مراجع:

- https://www.geeksforgeeks.org/optimal-strategy-for-a-game-dp-31

۴. این دو تعریف را در نظر بگیرید:

- اگر p عددی اول باشد، یک **ریشه اولیه** به پیمانه p، عدد صحیح r است در مجموعه $\mathbb{Z}_p$ که هر عنصر ناصفر در $\mathbb{Z}_p$، توانی از r به پیمانه p باشد.
- فرض می‌کنیم p عددی اول باشد و r یک **ریشه اولیه** به پیمانه p باشد، و a عددی صحیح در محدوده ۱ تا $p-1$ باشد. اگر $r^e \textbf{mod}\, p = a$ باشد و $0 \leq e \leq p-1$ باشد، می‌گوییم که e **لگاریتم گسسته‌ی** a در مبنای r و به پیمانه p است و این را به صورت $\mathrm{dlog}_{r,p}(a) = e$ می‌نویسیم.

الف) لگاریتم گسسته ۳ در مبنای ۱۳ و به پیمانه ۱۹ را محاسبه کنید.

ب) الگوریتمی ساده‌اندیشانه برای محاسبه لگاریتم گسسته‌ی $\mathrm{dlog}_{r,p}(a)$ طراحی کنید و کارایی زمانی آن را اندازه بگیرید. آیا الگوریتم‌تان، الگوریتمی چندجمله‌ای است؟

جواب:

الف) $a = 3$، $r = 13$ و $p = 19$ است که یعنی میخواهیم e را بیابیم که: $e = \mathrm{dlog}_{13,19}(3) \rightarrow 13^e \mathrm{mod}\ 19 = 3$

از آنجا که اعداد مسئله کوچکند، برای e از 0 تا $p = 19$ را که مقادیر ممکن برای این توان هستند را در رابطه راست قرار می‌دهیم تا به جوابی صحیح رسیم. میابیم که اگر $e = 17$ آنگاه تساوی برقرار است:

$$13^{17} \mathrm{mod}\ 19 = (13^2)^8 \times 13 \mathrm{\ mod\ } 19 = ((13^2 \mathrm{\ mod\ } 19)^8 \mathrm{\ mod\ } 19) \times (13 \mathrm{\ mod\ } 19) \mathrm{\ mod\ } 19 = 3$$

ب) توصیف الگوریتم: برای الگوریتم ساده اندیشانه کافی‌ست حلقه‌ای قرار دهیم تا اعداد را از صفر تا پیمانه p در رابطه مثل (الف) امتحان کنیم. اگر جایی رابطه برقرار بود و پاسخ برابر با a شد، آن جواب را برای e بر می‌گردانیم. در صورت اتمام حلقه و عدم یافت مقدار، ناموفق بودن را اعلام می‌کنیم، مثلا با عددی منفی یا *None*.

```
Algorithm: DLog(a, r, p)
    // Finds discrete logarithm using brute force algorithm
    // Input: Integers a, r, and p for power, base, and modulo respectively
    // Output: Discrete logarithm solution, or −1 if no answer is found

    for e in 0 to p do
        sol ← r^e mod p
        if sol = a then return e
    return −1
```

کارایی زمانی: الگوریتم به طور کلی از یک حلقه تشکیل شده است که تعداد تکرار آن در بدترین حالت $p + 1$ است. در عین حال در نظر داریم که اعمال ریاضی درون حلقه برای اعداد بزرگ در زمان ثابت انجام نشده و هر کدام زمان بر هستند. پس برای محاسبه کارایی زمانی اعمال را تعیین می‌کنیم:

- **عمل توان:** اگر توان r^e را تعداد e بار ضرب r در خودش بگیریم (یعنی حلقه‌ای از ضرب‌ها)، آنگاه به کارایی زمانی ضرب اعداد بزرگ نیاز داریم. الگوریتم‌های متفاوتی برای محاسبه ضرب با کارایی‌های متنوع وجود دارند. می‌دانیم الگوریتم کاراتسوبا (۱۹۶۰) ضرب را در زمان $O(n^{1.58})$ به دست می‌آورد. همچنین الگوریتم هاروی و وندرو هوون (۲۰۱۹) ضرب را برای اعداد بسیار بسیار بزرگ در زمان $O(n \log n)$ حساب می‌کند (برای اعداد کوچکتر کندتر است و مناسب استفاده کلی نیست). اگر فرض را بر این حد پایین بگذاریم (که البته دقیق نیست)، کارایی زمانی توان دو عدد بسیار بزرگ به اندازه n برابر با $O(n^2 \log n)$ می‌شود. با دیگر الگوریتم‌های ضرب کارایی توان متفاوت می‌شود و به طور کلی تا $O(n^3)$ می‌تواند برود.

- **عمل پیمانه‌گیری:** بسته به نوع پیاده سازی (سخت افزاری یا نرم افزاری و با چه الگوریتمی)، کارایی زمانی پیمانه از مقدار ثابت تا $O(n^2)$ می‌تواند برود که در هر صورت از کارایی زمانی توان همواره کمتر است.

بنابراین بزرگترین بخش تکرار حلقه برای محاسبه توان است، و کارایی زمانی الگوریتم ساده اندیشانه در بدترین حالت‌ها با ساده‌ترین الگوریتم‌ها به شکل زیر است که چند جمله‌ای نیست:

$$T(n) = O(n) \times \left(O(n^3) + O(n^2)\right) = O(n^4)$$

مراجع: ـــــــ

۵. مسأله ازدواج پایدار را به این شکل تعمیم می‌دهیم که تشکیل زوج‌های خاصی از مردها- زن‌ها صریحاً ممنوع باشد. (در مورد تطابق کارفرماها و کارجوها، می‌توانیم این گونه فکر کنیم که بعضی از کارجوها فاقد صلاحیت‌ها یا گواهی‌های لازم باشند و بنابراین، با وجود آنکه موجه به نظر می‌رسند، نتوانند در شرکت‌های خاصی استخدام شوند). پس ما یک مجموعه M شامل n مرد داریم و یک مجموعه W شامل n زن. و یک مجموعه $F \subseteq M \times W$ شامل زوج‌هایی که به سادگی، مجاز به ازدواج با یکدیگر نیستند. هر مرد m ، تمام زن‌های w را با شرط $(m, w) \notin F$ رتبه‌بندی می کند و هر زن w' ، تمام مردهای m' را با شرط $(m', w') \notin F$ رتبه‌بندی می کند.

در این قالب کلی‌تر از مسأله ازدواج پایدار، ما می‌گوییم که یک تطابق ازدواج S پایدار است، اگر هیچ یک از این نوع ناپایداری‌ها را نداشته باشد:

- دو زوج (m, w) و (m', w') در S وجود داشته باشند و با شرط $(m, w') \notin F$ ، مرد m ترجیح دهد زن w' را به w ، و زن w' ترجیح دهد مرد m را به m' . (این حالت، همان نوعِ عادی ناپایداری است.)
- زوج $(m, w) \in S$ باشد، اما یک مرد m' وجود داشته باشد که در هیچ زوجی از تطابق قرار نگرفته باشد، و با شرط $(m', w) \notin F$ ، w ترجیح دهد m' را به m . (در این حالت، زنی که با مردی زوج شده است، مردی مجرد را که ازدواج با او ممنوع نیست، به آن مرد ترجیح می‌دهد.)
- زوج $(m, w) \in S$ باشد، اما یک زن w' وجود داشته باشد که در هیچ زوجی از تطابق قرار نگرفته باشد، و با شرط $(m, w') \notin F$ ، m ترجیح دهد w' را به w . (در این حالت، مردی که با زنی زوج شده است، زنی مجرد را که ازدواج با او ممنوع نیست، به آن زن ترجیح می‌دهد.)
- یک مرد m و یک زن w وجود داشته باشند که با شرط $(m, w) \notin F$ ، هیچ یک از آن دو در هیچ زوجی از تطابق قرار نگرفته باشند. (در این حالت، یک مرد مجرد و یک زن مجرد وجود دارند که مانعی برای ازدواج آنها با یکدیگر وجود ندارد.)

با این تعریف از ناپایداری یک تطابق ازدواج، یا الگوریتمی ارائه کنید که همیشه برای هر مجموعه‌ای از لیست‌های ترجیحات مردان و زنان و هر مجموعه‌ای از زوج‌های ممنوع، یک تطابق ازدواج پایدار تولید کند؛ یا مثالی بزنید از مجموعه‌ای از لیست‌های ترجیحات مردان و زنان و مجموعه‌ای از زوج‌های ممنوع که نتوان از روی آنها به یک تطابق ازدواج پایدار رسید.

جواب:

توصیف الگوریتم: برای این مسئله همان الگوریتم اصلی گیل-شیپلی حل مسئله ازدواج پایدار جواب درست می‌دهد. در الگوریتم اصلی شرط ادامه دادن، وجود مردی‌ست که مجرد باشد و به تمام زنان درخواست نداده باشد. اگر شرط را درخواست به زنان لیستش کنیم، از آنجا که در این مسئله زنانی که ازدواج با آنها برای مرد ممنوعه است در لیستش نیستند آنگاه هیچگاه مردان اقدام به درخواست به آنان نمی‌کنند و رابطه ممنوعه‌ای شکل نمیگیرد (برعکس هم می‌توان گفت به زنان درخواست از مردی ممنوعه داده نمی‌شود زیرا رابطه ممنوعیت دو طرفه است) و هر درخواستی که بتواند داده شود می‌شود. پس با داشتن لیست اولویت مردان و زنان می‌توان تطابق را یافت.

الگوریتم به طور خلاصه به شرح زیر است:

۱) در ابتدا تمام مردان و زنان آزادند؛
۲) تا زمانی که مردی آزاد مانند m وجود دارد که به تمام زنان لیستش درخواست نداده است تکرار کن:
   ۱. فرض کنیم w با اولویت‌ترین زنی در لیست مرد m است که m به او درخواست نداده است؛
   ۲. اگر w آزاد است: w و m جفت می‌شوند؛
   ۳. اگر w آزاد نیست:
      i. اگر w جفتش (m') را به m ترجیح می‌دهد: m آزاد می‌ماند و w را از لیستش خط می‌زند؛
      ii. اگر w جفتش (m') را به m ترجیح نمی‌دهد: m' آزاد می‌شود و m و w جفت می‌شوند.
۳) پس از پایان حلقه جفت‌های یافت شده برگردانده می‌شوند.

اثبات درستی: ثابت می‌کنیم تطابق‌های یافت شده پایدارند. برای این کار، ۴ شرط داده شده را بررسی می‌کنیم:

۱) **ناپایداری عادی:** فرض می‌کنیم دو زوج (m,w) و (m',w') در S وجود دارد که با شرط $(m,w') \notin F$، مرد m ترجیح دهد زن w' را به w ، و زن w' ترجیح دهد مرد m را به m'. از آنجا که در خط ۲ مطمئن می‌شویم مرد m به تمام زنان ممکن درخواست داده، پس باید به w' نیز درخواست داده باشد. و در صورت درخواست، اگر زن w' مردی دیگری را به m ترجیح نمی‌داده ازدواجشان شکل گرفته و برای مرد دیگری شکسته نمی‌شده. حال اما داریم که این دو فرد در تطابق‌های نهایی جفت نیستند، پس باید زن w' یکجا به مرد نه گفته باشد و ترجیح دیگری داشته باشد. پس فرض باطل است.

۲) **ترجیح مرد مجرد:** فرض می‌کنیم زوج $(m,w) \in S$ باشد، اما یک مرد m' وجود داشته باشد که در هیچ زوجی از تطابق قرار نگرفته باشد، و با شرط $(m',w) \notin F$، w ترجیح دهد m' را به m. در صورت ممنوع نبودن پس m' به w باید زمانی درخواست داده باشد. و چون حال زوج نیستند، پس w به m' جواب رد داده بوده است و جفت حال حاضر خود یعنی m را به m' ترجیح داده است، که این خلاف فرض است.

۳) **ترجیح زن مجرد:** فرض می‌کنیم زوج $(m,w) \in S$ موجود باشد و یک زن w' که مجرد است، آنگاه با شرط $(m,w') \notin F$، m ترجیح دهد w' را به w. در این صورت باید w' در لیست m موجود می‌بوده و نسبت به w اولویت بالاتری می‌داشته. این یعنی m باید به او قبل از w درخواست می‌داده، و از آنجا که حال جفت نیستند باید جایی زن w' او را برای مرد دیگری ترک می‌کرده. اما از آنجا که زن w' مجرد است، پس نمی‌تواند رابطه را شکسته باشد و درخواستی از مردی دریافت کرده باشد زیرا جواب به درخواست در صورت آزاد بودن همیشه بله است. پس فرض خلف رد می‌شود.

۴) **جفت مجرد:** در آخر فرض می‌کنیم یک مرد m و یک زن w وجود داشته باشند با شرط $(m,w) \notin F$ اما هیچ یک از آن دو در هیچ زوجی از تطابق قرار نگرفته باشند. بنا بر الگوریتم، مرد m باید به تمام زنان غیر ممنوعش درخواست می‌داده است و از آنجا که w بر او ممنوع نیست به او نیز درخواست داده بوده است. اما زن w مجرد است در حالی که زنان درصورت آزاد بودن به درخواست حتما بله می‌گویند، پس ممکن نیست و باید زن w بر مرد m ممنوع بوده باشد.

بنابراین با تناقض برقراری ۴ شرط برای الگوریتم ثابت شده و الگوریتم تطابق‌های ازدواج پایدار را در مسئله میابد.

مراجع:

- **https://www.unipa.it/dipartimenti/matematicaeinformatica/.content/documenti/2018_Seminario_Erasmus_Lecture_Stable_Marriage_Problem.pdf**