هوالعلیم

# طراحی و تحلیل الگوریتم‌ها

نیمسال دوم سال تحصیلی ۱۴۰۱ – ۱۴۰۰

# تمرینات نظری ۴

مریم رضائی

**۱.** فرض کنید $H_0, H_1, H_2, \cdots$ دنباله‌ای نامتناهی **از** ماتریس‌ها باشد. در این دنباله، $H_0 = [1]$ است و برای هر $k > 0$ ، $H_k$ ماتریسی است $2^k \times 2^k$ با این تعریف بازگشتی:

$$H_k = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix}$$

اگر $v$ برداری ستونی با طول $n = 2^k$ باشد، الگوریتمی با کارایی $O(n \log n)$ برای محاسبه حاصل‌ضرب ماتریس در بردار $H_k v$ ارائه کنید. فرض کنید که همه اعداد آنقدر کوچک باشند که عملیات حسابی جمع و ضرب روی آنها در زمان ثابت قابل انجام باشد.

---

## جواب:

**توصیف الگوریتم:** برای یافتن الگوریتم به ساختار ماتریس‌ها توجه می‌کنیم. متوجهیم که ابعاد ماتریس هر بار برای $k$ بزرگتر، دو برابر می‌شود؛ به همین علت، از آنجا که نمیخواهیم تمامی عناصر ماتریس را پیش رویم و هر ماتریس منظم و قرنیه متشکل از ماتریس‌های قبلیست، می‌توانیم به طور بازگشتی طول ماتریس (و همچنین بردار) را نیم کنیم. در این صورت در نظر داریم که در ورودی بردار $v$ و عدد $k$ نماینده ماتریس را داریم که در بازگشت هر دو نصف می‌شوند، و پایان بازگشت (یعنی حالت پایه) جاییست که به ماتریس $H_0$ یعنی $k = 0$ برسیم که در آن پاسخ ضرب همان بردار است. پس الگوریتم $HMulV$ را با ورودی بردار $v$ و عدد $k$ به طور کلی داریم:

۱. **حالت پایه:** برای شرط $k = 0$، ماتریس را به صورت $H_0 = [1]$ داشته و بنابراین حاصل ضرب در بردار را خودکار خود بردار برمی‌گردانیم.

۲. **فراخوانی بازگشت:** گذر از شرط حالت پایه به معنی بزرگتر بودن $k$ از صفر است، پس برای ماتریس و بردار مربوطه، طول را نصف کرده (یعنی طول هر دو از $2^k$ به $2^{k-1}$ تغییر میابد) و برای هر نیمه بازمی‌گردیم. یعنی:

$$first = HMulV\left(v\left[:\frac{k}{2}+1\right], k-1\right)$$
$$second = HMulV\left(v\left[\frac{k}{2}+1:\right], k-1\right)$$

۳. **محاسبه ضرب:** با استفاده از پاسخ بازگشت‌ها، دو نتیجه را ترکیب کرده تا ضرب سطح حال حاضر جز سطح پایه را محاسبه کنیم. برای این کار بردار عمودی جدیدی را با توجه به اینکه در ضرب ماتریس جمع ضرب‌ها را داریم به شکل زیر شکل می‌دهیم که خروجی کلی این بار فراخوانی الگوریتم است (جز حالت پایه):

$$H = \begin{bmatrix} first + second \\ first - second \end{bmatrix}$$

**کارایی زمانی**: برای کارایی زمانی الگوریتم، سه بخش مستقل را بررسی می‌کنیم:

۱. **حالت پایه:** عملیات ساده و به علت برگرداندن خود بردار (که خود 1×1 است) کارایی زمانی O(1) می‌شود.

۲. **فراخوانی بازگشت:** طول ورودی نصف شده و الگوریتم بازگشتی برایشان اجرا می‌شود، پس دو $C\left(\frac{n}{2}\right)$ داریم. توجه داریم که در اینجا $n$ در اصل همان ارتفاع بردار و ماتریس است که به شکل $2^k$ است.

۳. **محاسبه ضرب:** از دوبار ترکیب دو بردار خروجی به صورت جمع و منهای عضو با عضو، دو بار طول بردارها را (که نیم طول $n$ است) پیموده و در اصل یک بار طول کل را می‌پیماییم. پس زمان را $O(n)$ داریم.

بنابراین، با توجه به اینکه بازه $k$ ورودی برای بخش ۱ با بخش ۲ و ۳ متفاوت اســـت، برای الگوریتم بازگشـــتی ارائه شده، رابطه کلی و رابطه پایه (پایان بازگشت) را می‌نویسیم:

$$C(n) = \begin{cases} O(1), & k = 0, n = 1 \\ 2 \times C\left(\frac{n}{2}\right) + O(n), & k > 0, n > 1 \end{cases}$$

با توجه به رابطه‌های بالا، کارایی زمانی را محاسبه می‌کنیم:

$$\begin{aligned} C(n) &= 2C\left(\frac{n}{2}\right) + O(n) = 2\left(2C\left(\frac{n}{4}\right) + O\left(\frac{n}{2}\right)\right) + O(n) = 4C\left(\frac{n}{4}\right) + 2O(n) \\ &= 4\left(2C\left(\frac{n}{8}\right) + O\left(\frac{n}{4}\right)\right) + 2O(n) = 8C\left(\frac{n}{8}\right) + 3O(n) = \cdots \\ &= nC\left(\frac{n}{n}\right) + \log n \, O(n) \xrightarrow{C\left(\frac{n}{n}\right)=c(1)=C(2^0)=O(1)} nO(1) + \log n \, O(n) \\ &= O(n) + O(n \log n) = O(n \log n) \end{aligned}$$

---

**مراجع:**


- https://cglab.ca/~michiel/2804/hadamard.pdf

**۲.** فرض کنید که ساختار شبکه تلفنی به شما داده شده است به شکل گراف $G = \langle V, E \rangle$ که هر رأس آن، نشانگر یک مرکز هادی[1] اســت و هر یال آن، نشــانگر یک خط ارتباطی موجود بین دو مرکز هادی؛ و هر یال گراف با پهنای باند خط ارتباطی متناظر با آن (که حداکثر ســرعتی اســت برحســب بیت بر ثانیه، که داده‌ها را می‌توان در امتداد آن خط ارتباطی انتقال داد) برچســب خورده اســت. پهنای باند یک مسیر در گراف $G$ را پهنای باند یالی از مسیر که کمترین پهنای باند را داشته باشد، تعریف می‌کنیم. الگوریتمی ارائه کنید که ساختار گرافی یک شبکه تلفن و دو مرکز هادی $a$ و $b$ را به عنوان ورودی بگیرد، و حداکثر پهنای باند یک مسیر بین $a$ و $b$ را به عنوان خروجی برگرداند. کارایی زمانی الگوریتم خود را نیز تعیین کنید.

---

## جواب:

**توصیف الگوریتم:** سوال شباهت زیادی به الگوریتم حریصانه دایکسترا (*Dijkstra*) دارد که در آن، به کمک پیمایش سطحی (*BFS*) و یک صف اولویت برای گره‌ها، کوچکترین وزن راه موجود از مبدا به هر گره‌ی دیگر را محاسبه می‌کنیم. بدین صورت که کوچکترین وزن مسیر از مبدا را برای راس‌های مجاور راس حال حاضر هر بار به روز رسانی کرده و راس با کمترین وزن مسیر که بررسی نشده را برای پیشروی انتخاب می‌کنیم تا در نهایت مقدار کمترین وزن مسیر را برای تمام رئوس داشته و همه بررسی شده باشند. در اینجا می‌توانیم با ایجــاد تغییراتی در این الگوریتــم برای حل سوال و همچنیــن انتخاب مناسب برای پیاده‌سازی به قصد بهینه بودن، به نتیجه برسیم. این تغییرات و انتخابات به شکل زیرند:

۱. **تغییر صف اولویت:** در الگوریتم دایکسترا صف کوچکترین وزن مسیر را برای هر گره ذخیره می‌کند و برای انتخاب در شروع کار به راس مبدا وزن مسیر صفر و به بقیه بینهایت می‌دهد. در اینجا قصد ما یافتن بزرگترین وزن مسیر است، پس باید پس از مقایسه بزرگترین وزن مسیر را هر بار برای هر گره در صف ذخیره کنیم و در ابتدا برای شروع الگوریتم راس مبدا را بزرگترین مقدار (یعنی بینهایت) قرار داده و دیگر گره‌ها را صفر بگذاریم.

۲. **تعیین راس مقصد:** در الگوریتم اصلی، به طور معمول از راس مبدأ شروع کرده و وزن مسیر به تمامی رئوس دیگر را میابیم و در حلقه پیمایش تا جایی پیش میرویم که تمامی رئوس بررســی شــوند. در اینجا در صــورت برخورد با رأس مقصد در انتخاب کردن رئوس جدید، با شرطی پیمایش را متوقف می‌کنیم.

۳. **اســتفاده از لیســت مجاورت برای گراف و هرم دودویی برای صــف اولویت:** برای داشــتن بهترین کارایی زمانی در بدترین حالت (یعنی زمانی که الگوریتم مجبور شود تمام گراف را بپیماید تا به مقصد برسد) می‌توانیم با ساختار داده‌های مناسب کارایی زمانی پیمایش گراف و انتخاب رأس مجاور بازدید نشده با کمترین وزن مسیر را در کمترین زمان انجام دهیم. لیســت مجاورت برای گراف این اثر را داشــته و هرم دودویی که مرتب اســت پیمایش صف اولویت را به $\log n$ یعنی ارتفاع درخت می‌رساند.

---

[1] switching center

**کارایی زمانی:** کارایی زمانی الگوریتم تفاوتی با الگوریتم اصلی دایکسترا (در صورتی که از همین ساختار داده‌ها استفاده شود) ندارد و برابر با $O(|E| \times \log|V|)$ است. با بررسی زمان هر بخش الگوریتم علت این امر را میبینیم:

۱. ساخت صف اولویت در ابتدا که شامل وزن مسیر پیش فرض برای هر گره است: $O(|V|)$
۲. پیش رفتن برای همه گره‌ها و انتخاب راس با بیشترین وزن مسیر به کمک هرم دودویی: $O(|V| \times \log|V|)$
۳. پیمایش سطحی برای تمام رئوس مجاور گره‌ها (یعنی هر یال) و هر بروز رسانی وزن مسیر: $O(|E| \times \log|V|)$

پس داریم:

$$O(|V|) + O(|V| \times \log|V|) + O(|E| \times \log|V|) = O(|V| \times \log|V|) + O(|E| \times \log|V|)$$
$$= O\big((|V| + |E|) \times \log|V|\big) \xrightarrow{connected\ graph \Rightarrow |E| \geq |V|-1} O(|E| \times \log|V|)$$

---

**مراجع:** ---

**۳.** مدیر یکی از انجمن‌های دانشجویی بزرگِ دانشگاه، با مسأله‌ای به سراغ شما آمده است. او مسئول نظارت بر کار گروهی $n$ نفره از دانشجویان است که هر یک از آنها طبق یک زمانبندی، باید یک نوبت در هفته، در انجمن کار کند. کارهای مربوط به نوبت‌های کاری دانشجویان متفاوت است (مانند حضور در پشت میز، کمک در تحویل بسته‌هایی و غیره) ، اما می‌توانیم هر نوبت را به شکل یک بازه زمانی پیوسته ببینیم. ممکن است چند نوبت کاری در یک زمان باشند.

مدیر می‌خواهد تعدادی از $n$ دانشجوی انجمن را انتخاب کند تا با آنها یک کمیته ناظر تشکیل دهد و با آنها یک جلسه هفتگی داشته باشد. از نظر او، چنین کمیته‌ای وقتی کامل خواهد بود که نوبت کاری هر دانشجویی که در کمیته نیست، با نوبت کاری یکی از دانشجویانی که در کمیته است، تداخل (گرچه جزئی) داشته باشد. بدین طریق، کارایی هر دانشجویی توسط حداقل یکی از افرادی که در کمیته حضور دارند، قابل مشاهده خواهد بود.

الگوریتم کارایی ارائه کنید که زمانبندی $n$ نوبت کاری دانشجویان را بگیرد و یک کمیته ناظرِ کامل تشکیل دهد که شامل کمترین تعداد دانشجو باشد.

مثلاً اگر $n = 3$ باشد و نوبت‌های کاری دانشجویان

- دوشنبه ۴ بعد از ظهر تا ۸ بعد از ظهر
- و دوشنبه ۶ بعد از ظهر تا ۱۰ بعد از ظهر
- و دوشنبه ۹ بعد از ظهر تا ۱۱ بعد از ظهر

باشند، از آنجا که زمان نوبت کاری دوم، با زمان نوبت‌های کاری اول و سوم، تداخل دارد، کوچک‌ترین کمیته ناظرِ کامل، شامل تنها دومین دانشجو خواهد بود.

کارایی زمانی الگوریتم خود را اندازه بگیرید و درستی آن را نیز ثابت کنید (یعنی ثابت کنید که الگوریتم همیشه جواب بهینه مسأله را برمی‌گرداند).

---

## جواب:

**توصیف الگوریتم:** با استفاده از نوعی از مسئله برنامه‌ریزی می‌توانیم سوال را حل کنیم، بدین صورت که در بازه نوبت‌ها پیش رویم و با مقایسه هر نوبت با دیگر نوبت‌های پوشش داده نشده مانده، بزرگترین بازه که بیشترین تعداد نوبت را می‌پوشاند انتخاب کنیم. برای این کار نیاز به اطمینان از مرتب بودن نوبت‌ها داریم؛ این کار را با مرتب سازی ادغامی بر اساس زمان پایانی بازه هر نوبت انجام می‌دهیم سپس به حل اصلی می‌رویم.

اگر لیست $X$ نوبت‌ها را داشته باشیم، در ابتدا همگی پوشش داده نشده‌اند (کمیته خالیست). اولین نوبت درون $X$ را که اولین نوبتیست ک توسط کمیته پوشش داده نشده است، نوبت پایه‌ای انتخاب می‌کنیم. حال باید آن را با تمامی نوبت‌های بعد مقایسه کنیم تا با در نظر گرفتن همه حالت‌ها بتوانیم مجموعه‌ای از نوبت‌های متداخل با آن یافته و آن‌ها را به لیست موقتی اضافه کنیم. هدف این لیست موقت، نگه داشتن تمامی نوبت‌های متداخل با نوبت پایه به ترتیب زمان پایان است تا در آن طولانی ترین نوبت این دسته را انتخاب کنیم؛ این نوبت بهترین بازه (بلندترین و شامل بیشترین نوبت دیگر) را دارد پس آن را عضوی از کمیته می‌کنیم.

در آخر لازم است $X$ را بروز کنیم تا فقط شامل نوبت‌هایی باشد که توسط کمیته پوشش داده نشده‌اند. آنگاه اولین نوبت آن را نوبت پایه جدید قرار داده و باز پیش می‌رویم تا یا نوبتی به کمیته اضافه کنیم، یا به پایان نوبت ها برسیم. شبه کد به شکل زیر است. ورودی آن بازه‌های نوبت‌ها بوده و در خروجی آرایه کوچکترین کمیته را باز می‌گرداند.

```
Algorithm: CreateCommittee(x1, …, xn)
    // Finds committee with least students that cover all shift intervals
    // Input: Intervals of students' shifts
    // Output: Array of shift intervals of picked students
    X ← MergeSort(x1, …, xn)   // sorts intervals based on end time
    committee ← array()
    while X not empty do
        intersect ← array()   // temp array to hold intervals intersecting with base
        base ← X[0]   // first element in X
        for interval in X do
            if base.end > interval.start then
                intersect.append(interval)
        selected ← intersect.pop()   // last element of intersect with latest end
        committee.append(selected)
        pre ← X   // hold pre delete array of X
        // selected doesn't necessarily end sooner than all intervals so overlap is different
        for interval in pre do
            if max(selected.start, interval.start) < min(selected.end, interval.end) then
                X.remove(interval)
    return committee
```

**کارایی زمانی:** برای محاسبه کارایی زمانی الگوریتم دو بخش اصلی مرتب سازی و پیشروی در نوبت‌ها برای انتخاب را در نظر میگیریم. می‌دانیم مرتب‌سازی ادغامی کارایی زمانی $O(n\log n)$ دارد. برای پیشروی در نوبت‌ها، در نظر داریم که برای هر نوبت، آن را با تمام نوبت‌های ملنده دیگر مقایسه می‌کنیم. از آنجا که در بدترین حالت که هر بار تنها یک نوبت از لیست $X$ حذف می‌شود، حلقه *while* همواره به تعداد $n$ نوبت تکرار می‌شود. با توجه به دو حلقه‌ی موازی درونی آن که تعداد تکرارشان برابر با طول $X$ در آن دور است و با هر بار تکرار حلقه *while* در بدترین حالت یک بار از آن کم می‌شود، برای تکرار عمل مقایسه داریم:

$$\sum_{i=1}^{n}\sum_{j=i}^{n}1=\sum_{i=1}^{n}(n-i+1)=\sum_{i=1}^{n}n-\sum_{i=1}^{n}i+\sum_{i=1}^{n}1=n\times n+\frac{n(n+1)}{2}+n=\frac{3n^2+3n}{2}$$

بنابراین کارایی زمانی حلقه $O(n^2)$ بوده و کارایی زمانی الگوریتم را داریم: $O(n\log n)+O(n^2)=O(n^2)$

**اثبات درستی:** در اثبات دو مورد را در نظر داریم: (۱) آیا الگوریتم تمامی نوبت‌های دانشجویان را پوشش می‌دهد؟ (۲) آیا الگوریتم کمترین تعداد دانشجو را باز میگرداند؟ با فرض خلف به این دو مورد جداگانه می‌پردازیم.

۱. فرض می‌کنیم الگوریتم تمامی دانشجویان را در نظر نمی‌گیرد، یعنی نوبتی را داریم به نام $x_k$ که توسط کمیته انتخابی پوشش داده نشده و نسبت به آخرین عضو کمیته به نام $c_m$ داریم $c_m.end \le x_k.start$. در عین حال در الگوریتم در اولین حلقه درونی برای هر نوبت مقایسه $c_m.end > x_j.start$ را نسبت به تمامی دیگر اعضا داریم تا پوشش با همه دیگر اعضا بررسی شود و بنا بر حلقه بیرونی عمل برای همه نوبت‌های پوشش داده نشده تکرار شود (از آنجا که حلقه درونی دوم تنها اعضای پوشش داده شده را حذف می‌کند). پس $x_k$ در صورت وجود بررسی شده و فرض باطل است.

۲. فرض می‌کنیم کمیته خروجی دارای $m$ عضو بهینه نبوده و کمیته‌ای با $k$ عضو موجود است که $k < m$. در این صورت انتخابی نامناسب بوده و قبل‌تر یا بعدتر از آن انتخاب دیگری موجود بوده است که با آن در تعداد کمتر بازه بیشتری پوشیده می‌شده؛ یعنی انتخاب $s$ (۱) با موقعیت بهتر و یا (۲) با طول بیشتر موجود بوده که بررسی نشده است. در عین حال در الگوریتم در نظر داریم که برای هر نوبت تا انتها تمامی نوبت‌های با اشتراک در نظر گرفته شوند و از میان آنها نوبت با بیشترین طول انتخاب شود. بنابراین اگر $s$ موجود بوده الگوریتم یا (۱) در بررسی تمام حالات ($n \times n$ حالت) آن را بررسی کرده است و یا (۲) در انتخاب طولانی‌ترین نوبت اشتراکی انتخاب کرده است. پس $s$ انتخاب شده است و $k = m$ و فرض نقض می‌شود.

بنابراین اثبات درستی الگوریتم کامل است.

---

## مراجع:


- http://www.cs.cornell.edu/courses/cs4820/2013sp/Handouts/samplesolutions.pdf

**۴.** به شما پارچه‌ای مستطیلی شکل با ابعاد $X \times Y$ ($X$ و $Y$ اعداد صحیح مثبت هستند) و لیستی از $n$ محصول که می‌توان با این پارچه تولید کرد، داده شده است. شما می‌دانید که برای تولید محصول $i$ اُم، $1 \leq i \leq n$ ، یک تکه پارچه مستطیلی با ابعاد $a_i \times b_i$ لازم است و اینکه قیمت فروش نهایی محصول $c_i$ خواهد بود (همه $a_i$ ها، $b_i$ ها و $c_i$ ها را اعداد صحیح مثبت بگیرید). شما ماشینی دارید که با آن می‌توانید هر تکه پارچه مستطیلی شکل را به طور افقی یا عمودی برش دهید تا دو تکه کوچک‌تر به دست آورید. الگوریتمی طراحی کنید که بیشترین سود حاصل از برش تکه پارچه $X \times Y$ را تعیین کند؛ یعنی راهبردی برای برش پارچه ارائه کند که طبق آن، محصولات تولید شده از تکه پارچه‌ها، در مجموع، بیشترین قیمت فروش را داشته باشند. شما آزادید هر تعدادی از یک محصول را که می‌خواهید تولید کنید یا آنکه نمونه‌ای از یک محصول را اصلاً تولید نکنید.

کارایی زمانی و کارایی فضایی الگوریتم خود را اندازه بگیرید.

---

## جواب:

**توصیف الگوریتم:** می‌توانیم با استفاده از برنامه‌نویسی پویا به طور بازگشتی مسئله را کوچک کرده و بهترین پاسخ برای هر سطح کوچک را بیابیم تا در نهایت به بیشترین سود کلی رسیم. برای تبدیل مسئله به برنامه‌نویسی پویا، نکته مهم تشخیص دو امر است: تابع حالات که حالات جدید را از حالات قبل میسازد، و ماتریس مورد نیاز برای ذخیره انتخابات که از روی آنها بهترین انتخابات سطح جدید را یافته و در سطر نهایی آن، بهترین انتخابات کلی را بیابیم.

برای تابع حالت به طور بازگشتی داریم که در هر بار، قصد ما برش پارچه به گونه‌ای است که بیشترین سود را بتوان از آن تولید کرد. یعنی ما می‌توانیم با توجه به محصولات ممکن، ضلع $X$ یا $Y$ (و یا هیچکدام) را برش دهیم و برای هر کدام سود پارچه‌های حاصل را به دست آوریم؛ در این صورت بهترین انتخاب برای آن سطح انتخابیست که بیشترین قیمت فروش را داشته باشد. بنابراین برای بیشترین سود هر سطح می‌نویسیم:

$$C(width, height) = \max \left\{ \begin{array}{c} \max\limits_{1 \leq w' < width} \{C(w', height) + C(width - w', height)\}, \\ \max\limits_{1 \leq h' < height} \{C(width, h') + C(width, height - h')\}, \\ \text{cost}(width, height) \end{array} \right\}$$

که $width$ و $height$ عرض و ارتفاع حال حاضر پارچه هستند به طوری که $1 \leq width \leq X$ و $1 \leq height \leq Y$، و $w'$ و $h'$ به ترتیب $width$ و $height$ جدید پس از برش پارچه از عرض یا ارتفاع هستند. همچنین برای تابع $cost$ ابعاد پارچه حال را با تمامی محصولات ممکن مقایسه می‌کنیم تا بیشترین سود را بیابیم، پس:

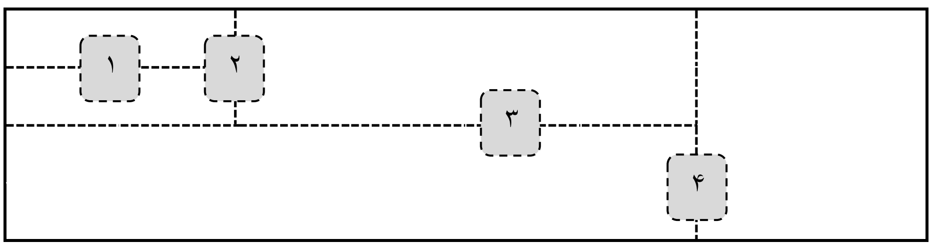
$$\text{cost}(width, height) = \begin{cases} \max\limits_{1 \leq i \leq n} \{c_i\}, & \text{for all product } i \text{ where } a_i = width \text{ and } b_i = height \\ 0, & \text{if no product } i \text{ exists where } a_i = width \text{ and } b_i = height \end{cases}$$

اتمام بازگشت باید در جایی باشد که طول عرض یا ارتفاع به ۱ رسیده و دیگر برشی ممکن نباشد. در آن صورت یا محصولی از ابعاد حال حاضر تشکیل می‌شود، و یا محصولی نمی‌تواند تشکیل شود و سود سطح صفر است. پس:

$$C(1, height) = \mathrm{cost}(1, height)$$

$$C(width, 1) = \mathrm{cost}(width, 1)$$

بر اساس این توابع در اصل مسئله به $XY$ مسئله کوچکتر تقسیم می‌شود. پس ماتریس *maxcosts* ذخیره پاسخ‌های بازگشت‌ها ابعاد $XY$ داشته و برای هر ترکیبشان بیشترین قیمت ممکن از آن ترکیب را ذخیره می‌کند. در عین حال انتخاب برش برای هر قیمت نهایی را نیز باید ذخیره کنیم تا در آخر شیوه برش مربوطه را بیابیم؛ پس ماتریس *choices* با همین ابعاد را نیاز داریم تا در خانه‌های معادل *maxcosts* برای هر قیمت بیشینه کل، انتخاب آن آخرین سطح (یعنی محصول بریده شده) را نیز ذخیره کند. آنگاه در خروجی الگوریتم، دو ماتریس داریم که در آنها الگوی مورد نظر ثبت شده است. برای مثال پارچه با ابعاد ۸ در ۴ را داریم که الگوی برش زیر برای آن بهترین است:




در این صورت در جایگاه (8, 4) ماتریس *choices* (که در ماتریس *maxcosts* بیشترین قیمت نهایی را برای کل پارچه ذخیره کرده است) مثلا محصول $p_2$ با ابعاد (2, 4) را داریم که یعنی این محصول را با برش جدا کرده و به عقب به خانه (6, 4) ماتریس *choices* می‌پریم (یعنی ابعاد جدید ۶ در ۴ هستند) تا انتخاب بعد را بیابیم. سپس به (6, 2)، (2, 2)، و در آخر (2, 1) می‌رویم. شبه کد الگوریتم در صفحه بعد قابل مشاهده است.

**کارایی زمانی:** الگوریتم به طور بازگشتی به طور کل $XY$ مسئله کوچکتر را بررسی می‌کند و هر خانه ماتریس را یک بار پر می‌کند. در عین حال هر سطح بازگشت دارای ۳ بخش است: (۱) یافتن قیمت این سطح در زمان $O(n)$؛ (۲) مقایسه با قیمت تمامی $X$ها تا به الان برای یافتن بیشترین قیمت در زمان $O(X)$؛ (۳) مقایسه با قیمت تمامی $Y$ها تا به الان برای یافتن بیشترین قیمت در زمان $O(Y)$. پس کارایی زمانی کل برابر است با $O\big(XY(n + X + Y)\big)$.

**کارایی فضایی:** از آنجا که دو ماتریس $XY$ برای ذخیره مقادیر استفاده کردیم (ماتریس *maxcosts* برای پیش بردن الگوریتم و تصمیم گیری تا یافتن بهترین قیمت در آخرین خانه، و ماتریس *choices* برای تشخیص الگوی ایده‌آل در خروجی)، کارایی فضایی $O(XY)$ می‌باشد.

---

## مراجع:


- https://piazza.com/class_profile/get_resource/honxzoabz711ew/htmhy41zn9g7a5

**Algorithm:** ***CuttingCloth*(*X*, *Y*, *P*[1… *n*])**

```
// recursively finds the pattern for cutting cloth with the most selling price
// Input: Cloth dimensions X and Y, list P of n a×b products with cost c
// Output: Edits two matrices maxcosts and choices

// global matrices maxcosts and choices of XY dimensions are predefined
cost, choice ← CurrentCost(X, Y, P)
if X = 1 or Y = 1 then
    maxcosts[X, Y] ← cost and choices[X, Y] ← choice
    return
CuttingCloth(X – 1, Y, P)
CuttingCloth(X, Y – 1, P)
for width = 1 to X – 1 do
    if (maxcosts[width, Y] + maxcosts[X – width, Y]) > cost then
        cost ← maxcosts[width, Y] + maxcosts[X – width, Y]
        choice ← choices[X – width, Y]
for height = 1 to Y – 1 do
    if (maxcosts[X, height] + maxcosts[X, Y – height]) > cost then
        cost ← maxcosts[X, height] + maxcosts[X, Y – height]
        choice ← choices[X, Y – height]
maxcosts[X, Y] ← cost and choices[X, Y] ← choice
return
```

---

**Algorithm:** ***CurrentCost*(*X*, *Y*, *P*[1… *n*])**

```
// finds product matching current cloth size
cost ← 0 and choice ← None
for product in P do
    if product.a = X and product.b = Y and product.c > cost then
        cost ← product.c
        choice ← product
return cost, choice
```

**۵.** مجموعه $S$ از $n$ عدد صحیح، و عدد صحیح $T$ داده شده است. الگوریتمی را توصیف کنید که بتواند در زمان $O(n^{k-1}\log n)$ ، این را بررسی کند که آیا $k$ عدد صحیح در $S$ وجود دارند که مجموع آنها برابر با $T$ شود یا خیر.

---

## جواب:

**توصیف الگوریتم:** برای به دست آوردن این کارایی می‌توان با شکستن تعداد $k$ پیش رفت. یعنی زیرمجموعه‌های $k-1$ عضوی مجموعه $n$ را تشکیل داده و جمع هر کدام را از عدد $T$ کم کرد تا عدد $k$اُم لازم را بیابیم. سپس باید بررسی کنیم آیا این عدد عضوی از $S$ می باشد که $k$ عضو آن مجموع مورد نظر را داشته باشند یا خیر. برای این کار (جستجو در مجموعه) بهترین کار استفاده از جستجوی دودویی است تا برای هر زیرمجموعه با کارایی زمانی $O(n)$ جستجو نکنیم. پس در ابتدای الگوریتم مجموعه را مرتب می‌کنیم تا بتوانیم پس از یافتن عدد $k$اُم صحیح برای هر زیرمجموعه، با زمان $O(\log n)$ در مجموعه به دنبال آن بگردیم. در نهایت الگوریتم را به شکل زیر داریم:

**Algorithm:** ***KSubsetSum*(*S*, *T*, *k*)**

```
    // finds if k elements of set S sum up to given int T
    // Input: Subset S, integer T, count k
    // Output: True if possible, False if not

    S ← MergeSort(S)
    subsets ← KSubset(S, k – 1)   // returns an array of all the S subsets of given length
    for subset in subsets do
        kth ← T – sum(subset)
        exists ← BinarySearch(S, kth)   // returns Boolean True if search is successful
        if exists then return exists
    return False
```

در این شبه کد از سه تابع از پیش تعریف شده مرتب‌سازی ادغامی، جستجوی دودویی، و یافتن زیرمجموعه‌های $k$ عضوی استفاده می‌کنیم. دو تابع اول توابعی‌اند که در درس مطالعه کرده‌ایم. تابع یافتن زیرمجموعه‌ها نیز الگوریتم‌های معروف با کارایی‌های متفاوت دارد. ساده‌ترین و پراستفاده‌ترین نوع این الگوریتم که به طور بازگشتی هر بار مجموعه را نصف کرده و زیرمجموعه‌ها را می‌یابد کارایی زمانی $O(2^n)$ را دارد؛ این الگوریتم بهینه نبوده و از تعداد اصلی زیرمجموعه‌های این مسئله که کارایی آن بسته است (یعنی $\binom{n}{k}$) بسیار بیشتر است. برای کارایی بهتر، نوع‌های بهینه شده الگوریتم که با شروط از حالات تکراری اجتناب می‌کنند و یا همچنین الگوریتم‌های معروف دیگری مانند باکلز موجودند که همگی کارایی $O(n^k)$ دارند و حد بسته برای مسئله را می‌دهند. این کف برای زمان مسئله موجود است زیرا بنابر تعداد زیرمجموعه‌های انتخابی $k$ عضوی از مجموعه $n$ عضوی داریم:

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!} = \frac{1}{k!}n(n-1)(n-2)\dots(n-k+1) \xRightarrow{\frac{1}{k!}=\theta(1)} n(n-1)(n-2)\dots(n-k+1)$$
$$= n^k\left(1\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\dots\left(1-\frac{k}{n}\right)\right) \xRightarrow{n\to\infty,\ \ k\ constant} n^k \times 1 = \theta(n^k)$$

**کارایی زمانی:** برای کارایی زمانی کل الگوریتم ارائه شـــده بخش‌های آن را در نظر می‌گیریم. می‌دانیم که مرتب‌ســـازی ادغامی $O(n\log n)$، یافتن و همانطور که نشـــان دادیم تعداد زیرمجموعه‌های $k-1$ عضـــوی $O(n^{k-1})$، و جســـتجوی دودویی در مجموعه $O(\log n)$ زمان می‌برد؛ پس:

$$O(n\log n) + O(n^{k-1}) + n^{k-1} \times O(\log n) = O(n^{k-1}\log n)$$

---

## مراجع:


- **https://stackoverflow.com/questions/9041353/design-an-algorithm-to-check-whether-k-integers-add-up-to-x-in-the-given-set**