طراحی الگوریتم

موسسه آموزش عالی پاسارگاد شیراز

* زمان بندی با مهلت معین: تعدادی کار داریم که همه در یک واحد زمان اجرا تمام می‌شوند که هر یک از این کارها یک مهلت یا یک بهره معین دارد. اگر در بازه‌ای تا قبل از مهلت داده شده به آن اجرا شود، بهره معادل آن حاصل می‌شود.

- هدف از این مسئله این است که کارها به نحوی زمان بندی شوند که بیشترین بهره بدست آید، از طرفی لازم نیست الزاماً همه کارها زمان بندی شوند.

* مثال:

زمان اجرا	بهره	مهلت	کار
1	30	2	1
1	35	1	2
1	25	2	3
1	40	1	4

/ greedy حل مسئله)

اگر بخواهیم greedy باشد همه زمان بندی های ممکن را حساب کنیم، مدت اجرایی فاکتوریل (نمایی) می‌شود.

① کارها را براساس بهره به صورت نزولی مرتب می‌کنیم.

② یک مجموعه جواب درنظر می‌گیریم که این مجموعه امکان پذیر است اگر و فقط اگر مرتب شده کارهای درون آن براساس مهلت های صعودی امکان پذیر باشد.

$S = \{2, 4\}$ ✗ (مهلت 1، 1)　　$S = \{4, 3\}$ ✓ (1، 2 ← مهلت، لحظه)

③ باید در هر لحظه یک کار را انتخاب کرده و آن را به مجموعه S اضافه کنیم و مشخص کنیم که آیا مجموعه S امکان پذیر است یا خیر. درصورتی که S امکان پذیر نشود، باید کار انتخاب شده را از مجموعه S حذف کنیم و به سراغ کار بعدی برویم تا زمانی که به آخرین کار برسیم.

کار	بهره	مهلت
4	40	1
2	35	1
1	30	2
3	25	2

$S = \emptyset$

$S = \{4\}$ ✓ (مهلت 1)　　$S = \{4, 1\}$ ✓ (1، 2)　　$\Longrightarrow S = \{4, 1\}$ ☑

$S = \{4, 2\}$ ✗ (1، 1)　　$S = \{4, 1, 3\}$ ✗ (1، 2، 2)

* مرتبه اجرایی:

$n \log n + n^2 \Longrightarrow O(n^2)$

(مرتب سازی ← $n \log n$؛ انتخاب و امکان پذیری ← n^2)

۵۲

مثال 3: بر اساس بهره مرتب هست. (نزولی)

چون مهلت نهایی کار ۲ الویت دارد

کار	مهلت	بهره
1	3	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \{1\}$ ✓ (3)

$S = \{1, 2\}$ ✓ (3, 1)

$S = \{1, 2, 3\}$ ✗ (3, 1, 1)

$S = \{1, 2, 4\}$ ✓ (3, 1, 3)

$S = \{1, 2, 4, 5\}$ ✗ (3, 1, 3, 1)

$S = \{1, 2, 4, 6\}$ ✗ (3, 1, 3, 3)

$S = \{1, 2, 4, 7\}$ ✗ (3, 1, 3, 2) $\Rightarrow S = \{1, 2, 4\}$ ☑

* مسئله انتخاب فعالیت ها:

n فعالیت با شماره های 1 تا n وجود دارد که از یک منبع استفاده می کنند و در یک زمان معین فقط یک فعالیت می تواند از این یک منبع استفاده کند. فعالیت های نمی توانند هم پوشانی داشته باشند مانند n سخنران که می خواهند از یک سالن اجتماعات استفاده کنند.

* هر کدام از فعالیت ها دارای یک زمان شروع s_i و یک زمان خاتمه f_i است $(s_i < f_i)$

* اگر فعالیت i ام انتخاب شود در بازه زمانی $[s_i, f_i)$ اجرا می شود پس مسئله، انتخاب حداکثر تعداد فعالیت هایی است که با هم، هم پوشانی نداشته باشند مثلاً می خواهیم سخنرانی هایی را انتخاب کنیم که تعداد کل سخنرانی ها حداکثر شود.

F زمان انتهای کار

S زمان ابتدای کار

• الگوریتم حریصانه برای حل این مسئله به صورت زیر است:

① ابتدا باید فعالیت ها بر اساس زمان خاتمه به صورت صعودی مرتب شود $f_1 \le f_2 \le f_3 \le \dots \le f_n$

```
void activity_selection (int S[ ], int F[ ])
{   A{1};   j=1;
    for (2 ...... n)
    {   if (S[i] ≥ F[j])
        {   A=A∪{i};
            j=i;
        }
    }
```

خروجی A مجموعه

ادامه ←

شماره جلسه i	1	2	3	4	5	6	7	8
S[i]	1	3	0	5	3	5	6	8
مرتب شده F[i]	4	5	6	7	8	9	10	11

i	F[j] ≤ S[i]	A	j
2	X	{1}	1
3	X		
4	✓	{1,4}	4
5	X		
6	X		
7	X		
8	✓	{1,4,8}	8
end			

* جلسه بیت و دوم: کوله پشتی: دزدی با کوله پشتی خود وارد جواهر فروشی می‌شود. کوله پشتی او تحمل حداکثر وزن W را دارد و هر قطعه جواهر دارای وزن w_i و ارزش p_i است. مسئله هم انتخاب قطعاتی است که حداکثر ارزش را داشته باشد و از طرفی مجموع وزن آن‌ها از W تجاوز نشود.

* اگر این مسئله غیر حریصانه حل شود، همه زیر مجموعه های ممکن از n قطعه باید بررسی شود که در نهایت 2^n حالت می‌شود. پس این الگوریتم مرتبه نمایی است.

* اگر بخواهیم حریصانه (greedy) این مسئله را حل کنیم، یک روش ساده این است که جواهر با ارزش بیشتر را زودتر انتخاب کنیم. در واقع جواهرات را بر اساس ارزش آن‌ها از بزرگ به کوچک مرتب می‌کنیم و سپس در مرحله انتخاب، جواهر با ارزش‌تر را انتخاب و در مرحله تحقیق هم وزن کل را چک می‌کنیم که کوله پشتی گنجایش اضافه کردن این جواهر را دارد یا نه.

مثال ۲ $W = 30$

	قطعه اول	قطعه دوم	قطعه سوم
P	۱۰	۱۲	۲۰
w	۸	۷	۲۵

* روش یک :

حریصانه : قطعه سوم انتخاب شود با وزن ۲۵ و ارزش ۲۰

بهترین : قطعه اول و دوم انتخاب شود با وزن ۱۵ و ارزش ۲۵ پس روش ۱ بهینه نمی‌باشد

* روش دوم : قطعات را به ترتیب افزایش ارزش آن‌ها به ازای واحد وزن مرتب کنیم و سپس آن‌ها را به ترتیب انتخاب کنیم

مثال ۲ $W = 30$

	قطعه اول	قطعه دوم	قطعه سوم
w_i	۵	۱۰	۲۰
p_i	۵۰	۶۰	۱۴۰
$\frac{p_i}{w_i}$	$\frac{50}{5} = 10$	$\frac{60}{10} = 6$	$\frac{140}{20} = 7$

* حل با روش دوم حریصانه : اول قطعه ۱ انتخاب می‌شه حالش هم می‌رسه بعد قطعه سوم به می‌شه ... ولی دیگه جای قطعه دوم نمی‌شه ، مجموع وزن = ۲۵ ، مجموع ارزش = ۱۹۰

* بهترین : انتخاب قطعه دوم و سوم ، مجموع وزن = ۳۰ ، مجموع ارزش = ۲۰۰

* اگر کوله‌پشتی کسری باشد یعنی مثلاً جواهرات به صورت کیسه‌های خاک طلا باشد آن گاه دزد می‌تواند بخشی از کیسه‌ها را بردارد ، در این صورت به این مسئله کوله‌پشتی کسری گفته می‌شود و با روش حریصانه فوق (یعنی انتخاب قطعه جواهر بر اساس ارزش واحد) مسئله به راحتی حل می‌شود.

* حل به صورت کسری : اول قطعه اول با ارزش ۵۰ ، سپس قطعه سوم با ارزش ۱۴۰ و سپس ۵ کیلو از قطعه دوم انتخاب و ارزش کل ۲۲۰ می‌شود و وزن کل ۳۰ می‌شود.

* مرتبه اجرای کوله‌پشتی کسری به روش حریصانه $n + n\log n$ است که می‌شود $\Theta(n\log n)$

($n\log n$ → مرتب‌سازی ، n → انتخاب)

حل مسئله کوله پشتی ۰ و ۱ به روش پویا : در این روش اگر A یک زیر مجموعه بهینه از n قطعه باشد، ۲ حالت امکان پذیر است : یا A حاوی قطعه n ام هست یا نیست

اگر A حاوی قطعه n ام نباشد ، A برابر زیر مجموعه‌ای بهینه از n-1 قطعه اول سوال است

و اگر A حاوی قطعه n ام باشد ، منفعت کل برابر با P_n یعنی ارزش قطعه n ام به علاوه منفعت بهینه‌ای است هنگامی که قطعات را از n-1 قطعه اول انتخاب می‌کنیم.

$$P[i][w] = \begin{cases} \max\left(P[i-1][w],\ P_i + P[i-1][w-w_i]\right) & w_i \le w \\ P[i-1][w] & w_i > w \end{cases}$$

$P[1][5] = \max(\underbrace{P[0][5]}_{0}, \underbrace{P_1}_{50} + \underbrace{P[0][\overset{25}{30-5}]}_{0}) = 50$

سطر ماتریس $P \leftarrow n+1$

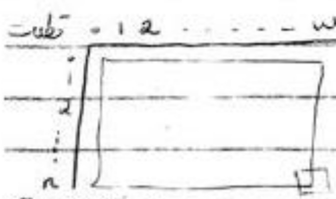
ستون ماتریس $P \leftarrow w+1$

← ادامه حل سوال قبل به روش پویا :

n \ w	0	1	2	3	4	5	6	7	8	9	10-	-15	30
0	0	0	0	0	0	0	0	0	0	—	—	—	0
1	0	0	0	0	0	50	50	50	50	—	—	—	50
2	0	0	0	0	0	50	50	50	50	50	60	...110...	110
3	0												☐

* اعداد درون جدول با استفاده از جدول $P[i][w]$ محاسبه می‌شود

کوله پشتی صفر و یک به روش پویا



* مرتبه اجرای این الگوریتم $O(nw)$ است.

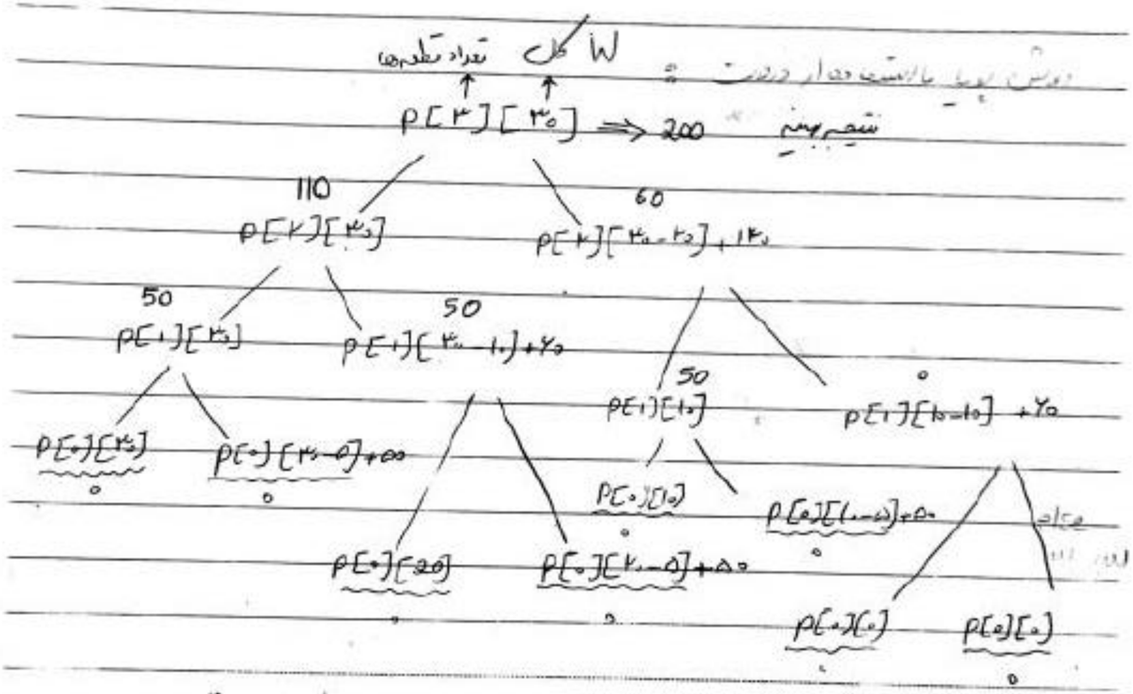
* اگر تمام حالت ها را بدست بیاوریم (بدست آوردن تمام زیرمجموعه‌های ممکن برای n قطعه) $O(2^n)$

MICRO

• مبحث رسوم:

• اگر احتیاجی n و w با دیتاست معمولاً $O(n^2)$ بهتر هست تا $O(nw)$

روش پویا با استفاده از درخت:



مرتبه اجرایی: $O(\min(2^n, nw))$

* الگوریتم greedy معرفی:

زمان‌بندی سیستم‌ها (مینیمم کردن زمان کل) چون این مسئله در سیستم عامل نیز کاربرد دارد

منظور از این الگوریتم کم کردن مجموع زمان‌های بین ورود تا خروج پردازش‌ها است. (ورود و خروج هر پردازش را نگه می‌داریم، جمع کنیم و این جمع کمترین باشد)

فرض کنیم کارها همزمان وارد سیستم می‌شوند ولی مدت زمان مربوط به هر کار متفاوت است و کارها انحصاری هستند

← کارها همزمان وارد می‌شوند، مدت زمان اجرای هر کار متفاوت است، می‌خواهیم مجموع زمان بین ورودی و خروجی کارها کمینه شود.

فرض کنید ۳ پردازش (کار) با زمان های $P_1=5$ ، $P_2=10$ و $P_3=4$ هم زمان وارد سیستم شوند. در سیستم عامل هنگامی که پردازنده را به پردازشی می دهد، نمی تواند آن را پس بگیرد تا هنگامی که کار آن تمام شود. ترتیب اجرا این کارها به چه صورت باشد تا مجموع زمان سپری شده (زمان کل در سیستم) حداقل شود؟

- یک الگوریتم ساده این است که تمام حالت های ممکن را بدست آوریم و از بین آن ها مقدار کمینه را انتخاب کنیم (مرتبه اجرایی $n!$)

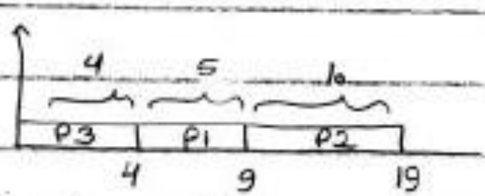
$\langle P_1, P_2, P_3 \rangle$ — $\langle P_3, P_2, P_1 \rangle$

$\langle P_1, P_3, P_2 \rangle$ — $\langle P_3, P_1, P_2 \rangle$

$\langle P_2, P_1, P_3 \rangle$

$\langle P_2, P_3, P_1 \rangle$

الگوریتم حریصانه برای حل این مسئله (shortest job first = SJF): ابتدا کاری را انتخاب کنید که زمان اجرای کمتری دارد.



زمان پاسخ برای $P_1 = 9$

$P_2 = 19$

$P_3 = 4$

32

کارها را براساس مدت اجرا مرتب می کنیم سپس عملیات انتخاب را براساس کمترین زمان اجرا انجام می دهیم و این کار را تا انتخاب تمام کارها ادامه می دهیم

مرتب سازی + انتخاب

$n\log n + n$

$\Rightarrow O(n \log n)$