طراحی و تحلیل الگوریتم‌ها

مبحث ششم

روش‌های طراحی الگوریتم

روش حریصانه

Methods of Algorithm Design: Greedy

کاظم فولادی
دانشکده مهندسی برق و کامپیوتر
دانشگاه تهران

http://courses.fouladi.ir/algorithm

برای حل یک مسئله با روش حریصانه:

- راه‌حل مسئله، به صورت **گام به گام** تولید می‌شود.
- در هر گام عنصری به راه‌حل اضافه می‌شود که **به نظر می‌رسد** بهتر است.

روش حریصانه:

- **برای حل مسائل بهینه‌سازی به کار می‌رود.**

- **نمونه‌ی مسئله را تقسیم نمی‌کند، بلکه با انجام یک دنباله انتخاب که هر یک در یک لحظه‌ی خاص بهترین به نظر می‌رسد، عمل می‌کند.**

- **امید دارد که راه حل بهینه‌ی سراسری یافت شود، ولی همواره چنین نمی‌شود:**
- **الگوریتم حریصانه، حل بهینه را تضمین نمی‌کند.**
ولی می‌تواند راه‌حلی که نسبتاً مناسب است را به دست آورد.

- **در عمل این روش برای حل مسائل واقعی بسیار قابل استفاده است.**
مسائلی که ارزش تئوری زیادی ندارند ولی در عمل مورد نیاز هستند.

- **می‌توان حداکثر فاصله‌ی روش حریصانه را با راه‌حل بهینه محاسبه کرد.**

برای حل یک مسئله با روش حریصانه:

- کار را با یک مجموعه‌ی تهی آغاز می‌کنیم.
- به ترتیب، عناصری را به این مجموعه اضافه می‌کنیم؛
تا وقتی که این مجموعه راه حلی برای آن نمونه از مسئله را نشان دهد.

هر دور تکرار شامل موارد زیر است:

(۱) انتخاب *Selection*	عنصر بعدی که باید به مجموعه اضافه شود، انتخاب می‌شود. عنصری انتخاب می‌شود که به نظر می‌رسد در همان لحظه بهینه است (حریصانه)
(۲) بررسی امکان‌پذیری *Feasibility Check*	تعیین می‌کند که آیا مجموعه‌ی جدید ممکن است به راه‌حل برسد؟
(۳) بررسی راه‌حل *Solution Check*	تعیین می‌کند که آیا مجموعه‌ی جدید راه حل نمونه است یا خیر؟

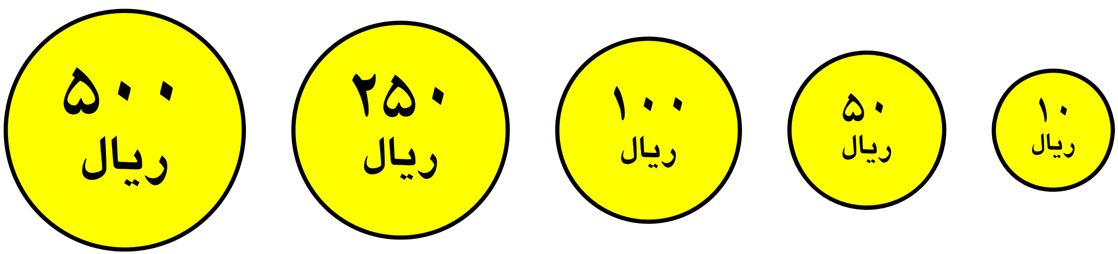
طراحی الگوریتم با روش حریصانه

$$
\begin{array}{l}
\textsc{Greedy}(A) \\
\quad S \leftarrow \{\} \\
\quad \textbf{while } \neg\text{solution}(S) \textbf{ do} \\
\quad\quad x \leftarrow \text{select}(A) \\
\quad\quad \textbf{if } \text{feasible}(S,x) \textbf{ then} \\
\quad\quad\quad S \leftarrow S \cup \{x\} \\
\quad\quad\quad A \leftarrow A - \{x\} \\
\quad \textbf{return } S
\end{array}
$$

(۱) انتخاب $x \leftarrow \text{select}(A)$	عنصر بعدی که باید به مجموعه اضافه شود، انتخاب می‌شود. عنصری انتخاب می‌شود که به نظر می‌رسد در همان لحظه بهینه است (حریصانه).
(۲) بررسی امکان‌پذیری $\text{feasible}(S,x)$	تعیین می‌کند که آیا مجموعه‌ی جدید ممکن است به راه‌حل برسد؟
(۳) بررسی راه‌حل $\text{solution}(S)$	تعیین می‌کند که آیا مجموعه‌ی جدید راه حل نمونه است یا خیر؟

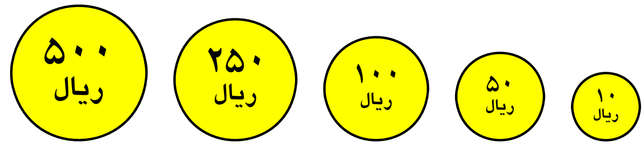
مسئله‌ی خرد کردن پول

می‌خواهیم N واحد پول را با سکه‌های $\{d_1,d_2, \dots ,d_k\}$ واحدی خرد کنیم.
از هر نوع سکه چند مورد برداریم که مجموع تعداد سکه‌ها حداقل (می‌نیمم) شود؟



طراحی الگوریتم با روش حریصانه

$\textsc{Greedy}(A)$
    $S \leftarrow \{\}$
    while $\neg\text{solution}(S)$ **do**
        $x \leftarrow \text{select}(A)$
        if $\text{feasible}(S, x)$ **then**
            $S \leftarrow S \cup \{x\}$
        $A \leftarrow A - \{x\}$
    return S

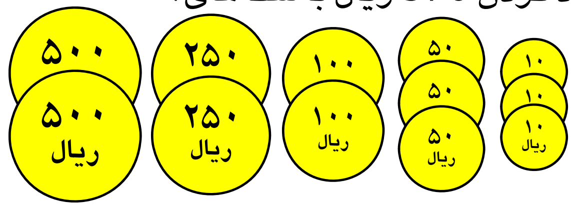


$$\begin{aligned} A = \{&(\mathbf{500},1),\ (\mathbf{500},2),\\ &(\mathbf{250},1),\ (\mathbf{250},2),\\ &(\mathbf{100},1),\ (\mathbf{100},2),\\ &(\mathbf{50},1),\ (\mathbf{50},2),\ (\mathbf{50},3),\\ &(\mathbf{10},1),\ (\mathbf{10},2),\ (\mathbf{10},3),\} \end{aligned}$$

(۱) انتخاب $x \leftarrow \text{select}(A)$	سکه‌ی بعدی که باید به مجموعه اضافه شود را انتخاب می‌کنیم. (سکه‌ای که بیشترین ارزش را در میان سکه‌های باقیمانده دارد (حریصانه))
(۲) بررسی امکان‌پذیری $\text{feasible}(S,x)$	آیا با انتخاب این سکه می‌توانیم پول را خرد کنیم؟ (آیا مجموع ارزش سکه‌ها از کل پول بیشتر نمی‌شود؟)
(۳) بررسی راه‌حل $\text{solution}(S)$	تعیین می‌کند که آیا مجموعه‌ی جدید راه حل نمونه است یا خیر؟ (آیا پول به طور کامل خرد شده است؟)

مسئله‌ی خرد کردن پول: یک نمونه (بهینه)

هدف: خرد کردن 570 ریال با سکه‌های:



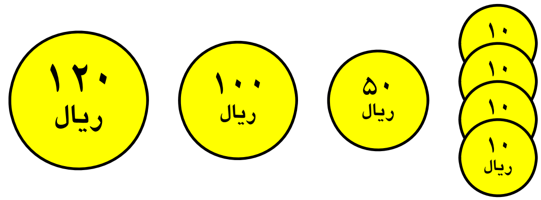
$$A = \{(\mathbf{500},1),\ (\mathbf{500},2),\ (\mathbf{250},1),\ (\mathbf{250},2),\ (\mathbf{100},1),\ (\mathbf{100},2),\ (\mathbf{50},1),\ (\mathbf{50},2),\ (\mathbf{50},3),\ (\mathbf{10},1),\ (\mathbf{10},2),\ (\mathbf{10},3),\}$$

S	$x \leftarrow \text{select}(A)$	feasible(S,x)	$S \leftarrow S \cup \{x\}$	solution(S)
$\{\}$	$(\mathbf{500},1)$	Yes	$\{(\mathbf{500},1)\}$	No
$\{(\mathbf{500},1)\}$	$(\mathbf{500},2)$	No		No
$\{(\mathbf{500},1)\}$	$(\mathbf{250},1)$	No		No
$\{(\mathbf{500},1)\}$	$(\mathbf{250},2)$	No		No
$\{(\mathbf{500},1)\}$	$(\mathbf{100},1)$	No		No
$\{(\mathbf{500},1)\}$	$(\mathbf{100},2)$	No		No
$\{(\mathbf{500},1)\}$	$(\mathbf{50},1)$	Yes	$\{(\mathbf{500},1),\ (\mathbf{50},1)\}$	No
$\{(\mathbf{500},1),\ (50,1)\}$	$(\mathbf{50},2)$	No		No
$\{(\mathbf{500},1),\ (50,1)\}$	$(\mathbf{50},3)$	No		No
$\{(\mathbf{500},1),\ (50,1)\}$	$(\mathbf{10},1)$	Yes	$\{(\mathbf{500},1),\ (\mathbf{50},1),\ (\mathbf{10},1)\}$	No
$\{(\mathbf{500},1),\ (50,1),\ (10,1)\}$	$(\mathbf{10},2)$	Yes	$\{(\mathbf{500},1),\ (\mathbf{50},1),\ (\mathbf{10},1),\ (\mathbf{10},2)\}$	Yes

مسئله‌ی خرد کردن پول: یک نمونه (غیربهینه)

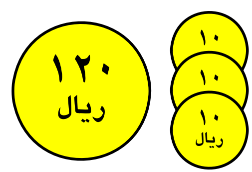
$$A = \{(\mathbf{120},1),\ (\mathbf{100},1),\ (\mathbf{50},1),\ (\mathbf{10},1),\ (\mathbf{10},2),\ (\mathbf{10},3),\ (\mathbf{10},4)\}$$

هدف: خرد کردن 150 ریال با سکه‌های:

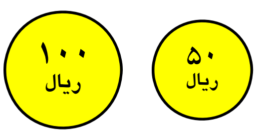


S	$x \leftarrow \text{select}(A)$	feasible(S,x)	$S \leftarrow S \cup \{x\}$	solution(S)
$\{\}$	$(\mathbf{120},1)$	Yes	$\{(\mathbf{120},1)\}$	No
$\{(\mathbf{120},1)\}$	$(\mathbf{100},1)$	No		No
$\{(\mathbf{120},1)\}$	$(\mathbf{50},1)$	No		No
$\{(\mathbf{120},1)\}$	$(\mathbf{10},1)$	Yes	$\{(\mathbf{120},1),\ (\mathbf{10},1)\}$	No
$\{(\mathbf{120},1),\ (\mathbf{10},1)\}$	$(\mathbf{10},2)$	Yes	$\{(\mathbf{120},1),\ (\mathbf{10},1),\ (\mathbf{10},2)\}$	No
$\{(\mathbf{120},1),\ (\mathbf{10},1),\ (\mathbf{10},2)\}$	$(\mathbf{10},3)$	Yes	$\{(\mathbf{120},1),\ (\mathbf{10},1),\ (\mathbf{10},2),(\mathbf{10},3)\}$	Yes

بهینه نیست!



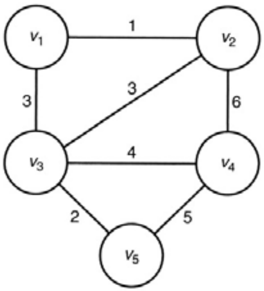
راه حل بهینه:



مسئله‌ی درخت پوشای می‌نیمال

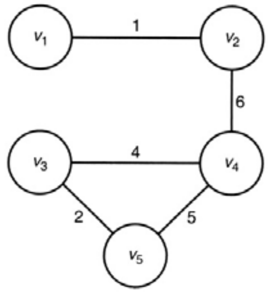
Minimal Spanning Tree (MST)

(a) A connected, weighted, undirected graph G.



گراف همبند بدون جهت

(b) If (v_4, v_5) were removed from this subgraph, the graph would remain connected.



زیرگراف

درخت پوشا:

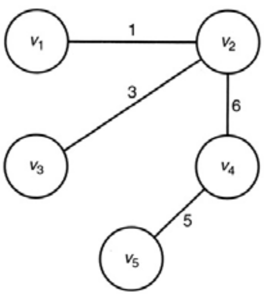
یک زیرگراف درخت از گراف همبند بدون جهت G که شامل همه‌ی رأس‌های آن باشد.

درخت پوشای می‌نیمال:

یک درخت پوشا که با کمترین مجموع وزن یال‌ها

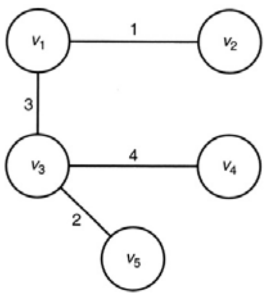
نکته: هر درخت با n رأس، $n-1$ یال دارد.

(c) A spanning tree for G.



یک درخت پوشا

(d) A minimum spanning tree for G.



درخت پوشای می‌نیمال

زیرگراف با می‌نیمم وزن، حتماً یک درخت است:
زیرا اگر زیرگرافی درخت نباشد، دارای دور است و می‌توان با حذف یک یال از دور، گراف همبندی با وزن کمتر پیدا کرد.

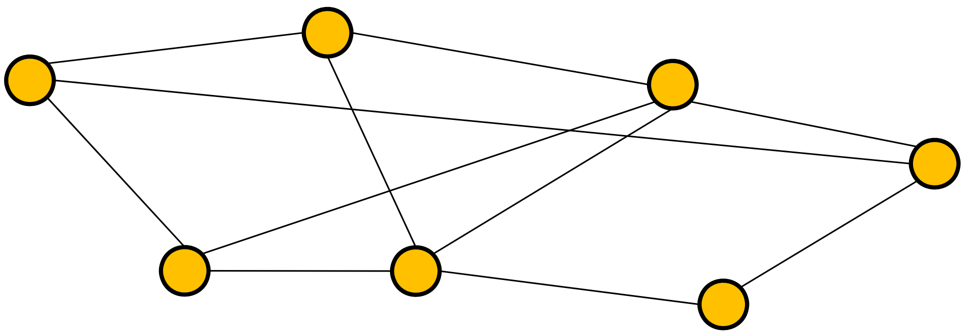
مسئله‌ی درخت پوشای می‌نیمال

کاربردها

جاده‌کشی بین n شهر با حداقل طول مسیر

لوله‌کشی بین n نقطه با حداقل طول لوله

سیم‌کشی بین n نقطه با حداقل سیم



مسئله‌ی درخت پوشای می‌نیمال

الگوریتم ناشیانه

الگوریتم یافتن درخت پوشای می‌نیمال با در نظر گرفتن همه‌ی درخت‌های پوشا، در بدترین حالت از نظر زمانی، بدتر از نمایی است.

نکته: هر گراف ساده با n رأس، n^{n-2} درخت پوشا دارد.

مسئله‌ی درخت پوشای می‌نیمال: الگوریتم حریصانه

```
GREEDY-MINIMAL-SPANNING-TREE(G)
    F ← {}
    while ¬solution(F) do
        x ← select(E)
        if feasible(F,x) then
            F ← F ∪ {x}
        E ← E − {x}
    return F
```

گراف

$G = (V, E)$

V : مجموعه‌ی رأس‌ها

E : مجموعه‌ی یال‌ها

(۱) انتخاب $x \leftarrow \text{select}(E)$	یال بعدی که باید به مجموعه اضافه شود، انتخاب می‌شود. (یالی انتخاب می‌شود که به نظر می‌رسد بهینه است (حریصانه))
(۲) بررسی امکان‌پذیری $\text{feasible}(F,x)$	تعیین می‌کند که آیا مجموعه‌ی یال‌ها یک درخت ایجاد می‌کند یا خیر؟ (آیا اضافه کردن این یال، دور ایجاد می‌کند یا خیر؟)
(۳) بررسی راه‌حل $\text{solution}(F)$	تعیین می‌کند که آیا مجموعه‌ی یال‌های جدید یک درخت پوشا است یا خیر؟ (آیا $T = (V, F)$ یک درخت پوشا است؟)

مسئله‌ی درخت پوشای می‌نیمال: الگوریتم پریم

الگوریتم **پریم** با دو زیرمجموعه‌ی تهی شروع می‌شود:

- F زیرمجموعه‌ی تهی از یال‌ها (E)
- Y زیرمجموعه‌ی تهی از رأس‌ها (V)

در ابتدا Y حاوی یک رأس دلخواه می‌شود (مثلاً $Y = \{v_1\}$)

نزدیک‌ترین رأس به Y،

رأسی در $V - Y$ است که توسط یالی با کمترین وزن به یک رأس در Y متصل می‌شود.

یال مربوطه به F اضافه می‌شود.

...

این فرآیند آن قدر تکرار می‌شود تا $Y = V$ شود.

انتخاب رأس جدید از $V - Y$ تضمین می‌کند که دور ایجاد نشود.

مسئله‌ی درخت پوشای می‌نیمال: الگوریتم پریم

$$
\begin{array}{l}
\textsc{Prim-Minimal-Spanning-Tree}(G)\\
\quad F \leftarrow \{\}\\
\quad Y \leftarrow \{v_1\}\\
\quad \textbf{while } \neg\text{solution}(F) \textbf{ do}\\
\quad\quad (x,y) \leftarrow \text{select}(V-Y, Y)\\
\quad\quad Y \leftarrow Y \cup \{x\}\\
\quad\quad F \leftarrow F \cup \{(x,y)\}\\
\quad\quad E \leftarrow E - \{(x,y)\}\\
\quad \textbf{return } F
\end{array}
$$

گراف

$G = (V,\ E)$

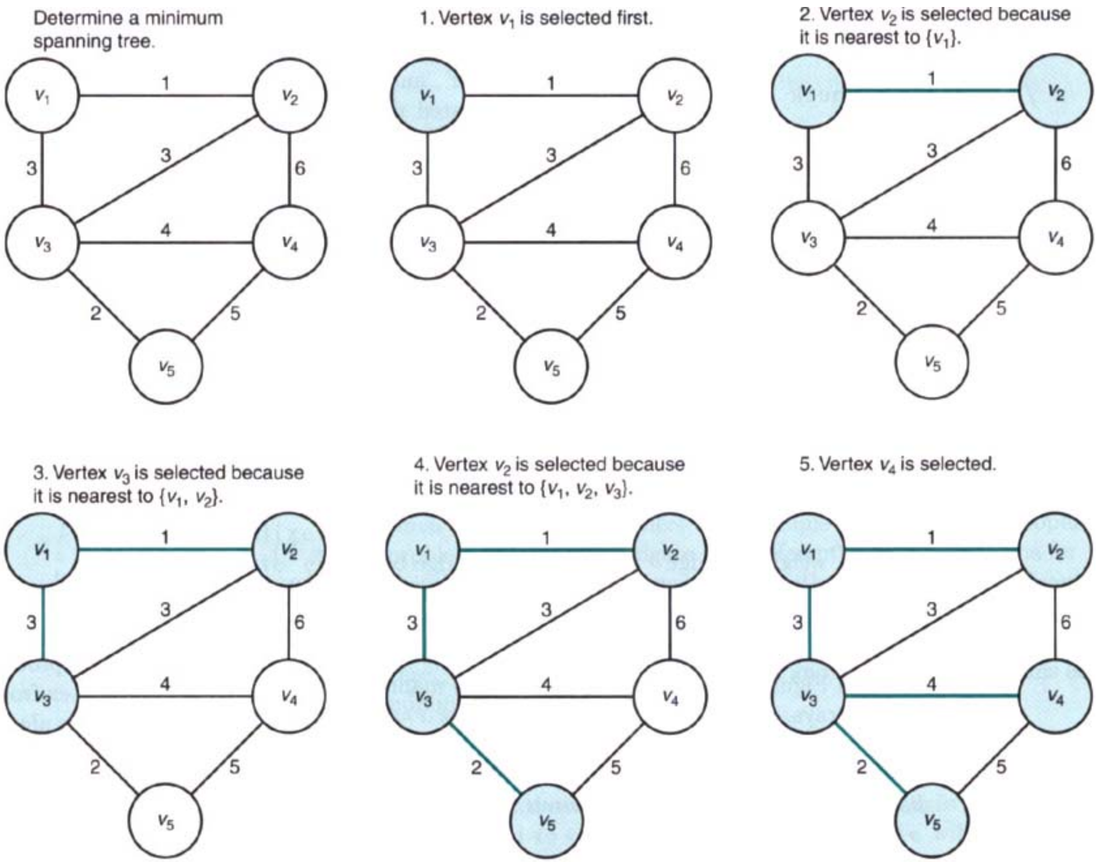
V : مجموعه‌ی رأس‌ها

E : مجموعه‌ی یال‌ها

(۱) انتخاب	
$x \leftarrow \text{select}(V-Y, Y)$	یال بعدی که باید به مجموعه اضافه شود، انتخاب می‌شود. (از $V-Y$ نزدیک‌ترین رأس به Y انتخاب می‌شود (حریصانه))
(۲) بررسی امکان‌پذیری	
$\text{feasible}(F,x)$	تعیین می‌کند که آیا مجموعه‌ی یال‌ها یک درخت ایجاد می‌کند یا خیر؟ (آیا اضافه کردن این یال، دور ایجاد می‌کند یا خیر؟) **لازم نیست!** : انتخاب رأس جدید از V - Y تضمین می‌کند که دور ایجاد نشود.
(۳) بررسی راه‌حل	
$\text{solution}(F)$	تعیین می‌کند که آیا مجموعه‌ی یال‌های جدید یک درخت پوشا است یا خیر؟ (آیا $Y = V$ شده است؟ یعنی: آیا همه‌ی رئوس گراف انتخاب شده است؟)

مسئله‌ی درخت پوشای می‌نیمال: الگوریتم پریم

مثال



مسئله‌ی درخت پوشای می‌نیمال: الگوریتم کراسکال

الگوریتم **کراسکال** به صورت زیر عمل می‌کند:

- ابتدا مجموعه‌ی یال‌ها به صورت غیرنزولی مرتب می‌شود.
- در هر تکرار یک یال انتخاب می‌شود:
  - اگر اضافه کردن آن یال، در درخت دور ایجاد نمی‌کند، آن را اضافه می‌کنیم.

❋ از نظر پیاده‌سازی

الگوریتم **کراسکال** با ایجاد زیرمجموعه‌های مجزای V شروع می‌شود.

- برای هر رأس یک زیرمجموعه شامل همان رأس ایجاد می‌کنیم.

سپس یال‌ها به ترتیب غیرنزولی وزن آنها بررسی می‌شوند:

- اگر یالی دو رأس را در مجموعه‌های مجزا به هم وصل می‌کند،
  - آن یال را به مجموعه اضافه می‌کنیم.
  - دو زیرمجموعه را در هم ادغام می‌کنیم.
- فرآیند فوق آن قدر تکرار می‌شود تا همه‌ی زیرمجموعه‌ها با هم ادغام شوند.

مسئله‌ی درخت پوشای می‌نیمال: الگوریتم کراسکال

```
KRUSKAL-MINIMAL-SPANNING-TREE(G)
    F ← {}
    while ¬solution(F) do
        e ← select(E)
        if feasible(F,e) then
            F ← F ∪ {e}
        E ← E − {e}
    return F
```

گراف

$G = (V, E)$

V : مجموعه‌ی رأس‌ها

E : مجموعه‌ی یال‌ها

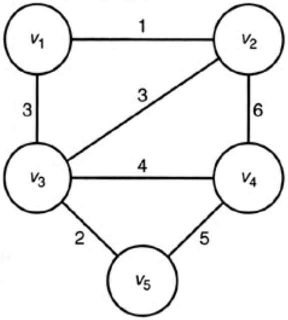
w : وزن یال‌ها

(۱) انتخاب	یال بعدی که باید به مجموعه اضافه شود، انتخاب می‌شود.
$e \leftarrow \mathrm{select}(E)$	(کوتاه‌ترین یال بعدی انتخاب می‌شود (حریصانه))
(۲) بررسی امکان‌پذیری	تعیین می‌کند که آیا مجموعه‌ی یال‌ها یک درخت ایجاد می‌کند یا خیر؟
$\mathrm{feasible}(F,e)$	(آیا اضافه کردن این یال، دور ایجاد می‌کند یا خیر؟)
(۳) بررسی راه‌حل	تعیین می‌کند که آیا مجموعه‌ی یال‌های جدید یک درخت پوشا است یا خیر؟
$\mathrm{solution}(F)$	(آیا *تعداد یال‌ها یکی کمتر از تعداد رأس‌ها* شده است؟)

مسئله‌ی درخت پوشای می‌نیمال: الگوریتم کراسکال

مثال

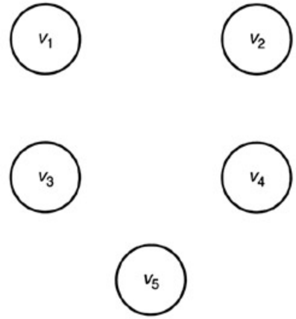
Determine a minimum spanning tree.



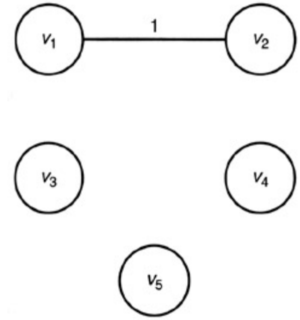
1. Edges are sorted by weight.

(v_1, v_2)	1
(v_3, v_5)	2
(v_1, v_3)	3
(v_2, v_3)	3
(v_3, v_4)	4
(v_4, v_5)	5
(v_2, v_4)	6

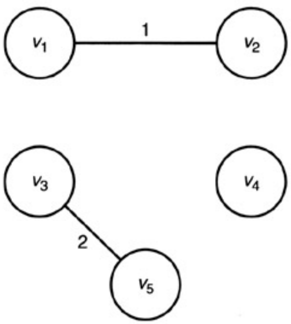
2. Disjoint set are created.



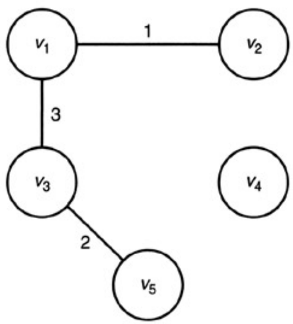
3. Edge (v_1, v_2) is selected.



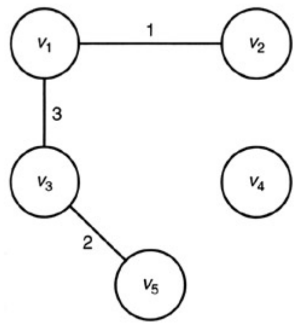
4. Edge (v_3, v_5) is selected.



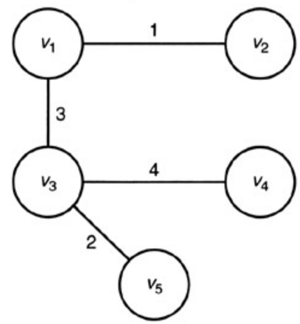
5. Edge (v_1, v_3) is selected.



6. Edge (v_2, v_3) is selected.



7. Edge (v_3, v_4) is selected.



الگوریتم درخت پوشای می‌نیمال

زمان اجرا

گراف $G = (V, E)$

V : مجموعه‌ی رأس‌ها، n : تعداد رأس‌ها

E : مجموعه‌ی یال‌ها، m : تعداد یال‌ها

الگوریتم پریم

n رأس داریم و هر رأس حداکثر با n رأس دیگر باید بررسی شود، پس

$$T(n) \in \Theta(n^2)$$

الگوریتم کراسکال

m یال داریم که باید به ترتیب طول مرتب شوند و یک به یک انتخاب و بررسی شوند، پس

$$T(m) \in \Theta(m \log m)$$

در گراف همبند $\quad n - 1 \leq m \leq \frac{1}{2} n(n-1) \quad$ در گراف کامل

برای گراف‌های پر، الگوریتم پریم و برای گراف‌های خلوت الگوریتم کراسکال بهتر است.

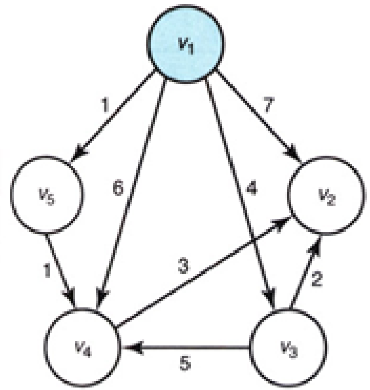
مسئله‌ی کوتاه‌ترین مسیر تک‌منبع

SINGLE SOURCE-SHORTEST PATH (SSSP)

گراف جهت‌دار $G = (V, E)$
V : مجموعه‌ی رأس‌ها، n : تعداد رأس‌ها
E : مجموعه‌ی یال‌ها، m : تعداد یال‌ها

هدف:

یافتن طول کوتاه‌ترین مسیر از **یک رأس مشخص** به سایر رئوس یک گراف جهت‌دار



v	v_1	v_2	v_3	v_4	v_5
$d(v_1, v)$	0	?	?	?	?

مسئله‌ی کوتاه‌ترین مسیر تک‌منبع: الگوریتم دایکسترا

DIJKSTRA ALGORITHM

الگوریتم **دایکسترا** با دو زیرمجموعه‌ی تهی شروع می‌شود:
- F زیرمجموعه‌ی تهی از یال‌ها (E)
- Y زیرمجموعه‌ی تهی از رأس‌ها (V)

در ابتدا Y حاوی یک رأس مبدأ مورد نظر می‌شود (مثلاً $Y = \{v_1\}$)
سپس رأس v با کمترین فاصله به رأس مبدأ را انتخاب، آن را به Y و یالش را به F اضافه می‌کنیم.

سپس، مسیرهایی از v_1 به رئوس موجود در $V - Y$ را بررسی می‌کنیم که
فقط از رئوس Y به عنوان رأس واسط استفاده می‌کند.
کوتاه‌ترین این مسیرها مشخص می‌شود:
- رأس واقع در انتهای این مسیر به Y اضافه می‌شود.
- یال شامل این رأس به F اضافه می‌شود.

...
این فرآیند آن قدر تکرار می‌شود تا $Y = V$ شود.

مسئله‌ی کوتاه‌ترین مسیر تک‌منبع: الگوریتم دایکسترا: شبه‌کد

$\textsc{Dijkstra-Single-Source-Shortest-Path}(G,\ v_1)$

$F \leftarrow \{\}$

$Y \leftarrow \{v_1\}$

while $\neg\text{solution}(F)$ **do**

$\quad v \leftarrow \text{select}(v_1, Y, V - Y)$

$\quad e \leftarrow$ the edge on shortest path that touches v to F

$\quad Y \leftarrow Y \cup \{v\}$

$\quad F \leftarrow F \cup \{e\}$

return F

گراف

$G = (V,\ E)$

V : مجموعه‌ی رأس‌ها

E : مجموعه‌ی یال‌ها

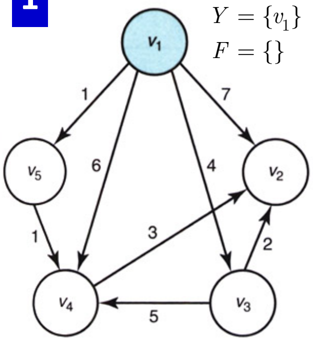
(۱) انتخاب	
$v \leftarrow \text{select}(v_1, Y, V - Y)$	رأس بعدی که باید به مجموعه اضافه شود، انتخاب می‌شود. (رأس v از $V - Y$ انتخاب می‌شود که فقط از رئوس میانی واقع در Y استفاده می‌کند و دارای کوتاه‌ترین مسیر از v_1 می‌باشد. (حریصانه))
(۲) بررسی امکان‌پذیری	
$\text{feasible}(F, x)$	**لازم نیست!** : در همان تابع انتخاب لحاظ شده است.
(۳) بررسی راه‌حل	
$\text{solution}(F)$	تعیین می‌کند که آیا می‌نیمم فاصله تا همه‌ی رئوس حساب شده است؟ (آیا $Y = V$ شده است؟ یعنی: آیا همه‌ی رئوس گراف انتخاب شده است؟)

مسئله‌ی کوتاه‌ترین مسیر تک‌منبع: الگوریتم دایکسترا

1

Compute shortest paths from v_1.

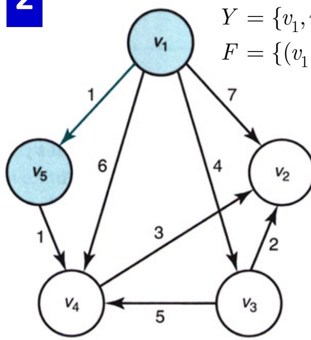
$Y = \{v_1\}$
$F = \{\}$



2

1. Vertex v_5 is selected because it is nearest to v_1.

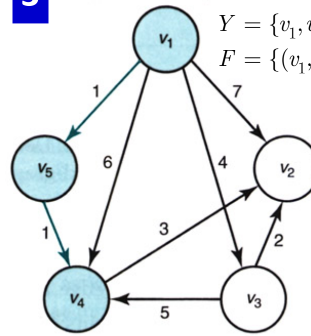
$Y = \{v_1, v_5\}$
$F = \{(v_1, v_5)\}$



3

2. Vertex v_4 is selected because it has the shortest path from v_1 using only vertices in $\{v_5\}$ as intermediates.

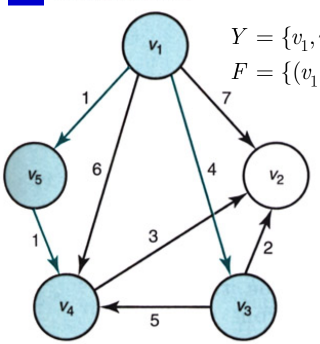
$Y = \{v_1, v_5, v_4\}$
$F = \{(v_1, v_5), (v_5, v_4)\}$



4

3. Vertex v_3 is selected because it has the shortest path from v_1 using only vertices in $\{v_4, v_5\}$ as intermediates.

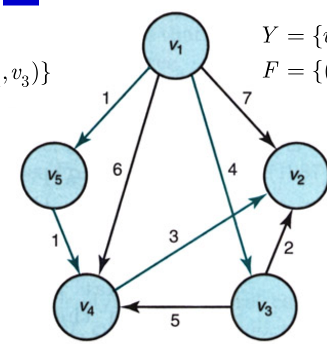
$Y = \{v_1, v_5, v_4, v_3\}$
$F = \{(v_1, v_5), (v_5, v_4), (v_1, v_3)\}$



5

4. The shortest path from v_1 to v_2 is $[v_1, v_5, v_4, v_2]$.

$Y = \{v_1, v_5, v_4, v_3, v_2\}$
$F = \{(v_1, v_5), (v_5, v_4), (v_1, v_3), (v_4, v_2)\}$



v	v_1	v_2	v_3	v_4	v_5
$d(v_1, v)$	0	5	4	2	1

الگوریتم کوتاه‌ترین مسیر تک‌منبع : زمان اجرا

گراف $G = (V, E)$

V : مجموعه‌ی رأس‌ها، n : تعداد رأس‌ها

E : مجموعه‌ی یال‌ها، m : تعداد یال‌ها

الگوریتم دایکسترا

n رأس داریم و هر رأس حداکثر با n رأس دیگر باید بررسی شود، پس

$$T(n) \in O(n^2)$$

HUFFMAN CODE

می‌خواهیم کاراکترها را با یک کد دودویی با طول متغیر کدگذاری کنیم، به طوری که:

- کاراکترهایی که **تعداد تکرار بیشتر** دارند با رشته‌ی بیتی **کوتاه‌تر** و
- کاراکترهایی که **تعداد تکرار کمتر** دارند با رشته‌ی بیتی **بلندتر** کدگذاری شوند.

⇐ متن با کوتاه‌ترین طول ممکن کدگذاری شود.

راه حل: **کد هافمن**

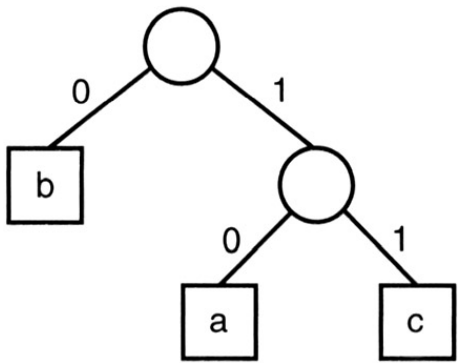
مثال:

a	b	c	d	e	f
16	**5**	**12**	**17**	**10**	**25**

کد پیشوندی

PREFIX CODE

در **کد پیشوندی**، هیچ کلمه‌ی کدی، پیشوند هیچ کلمه‌ی کد دیگری نیست.

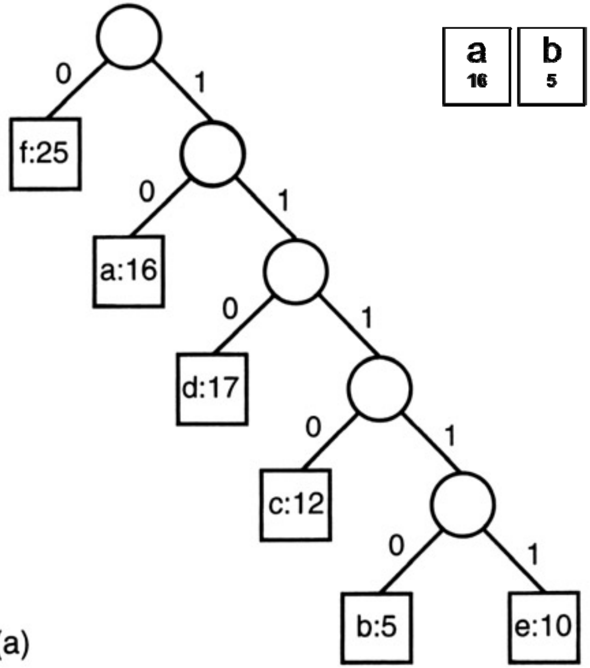


a	10
b	0
c	11

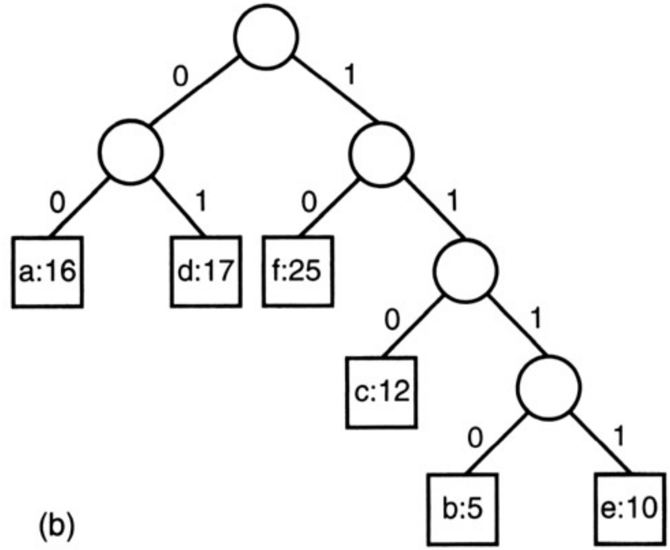
کد هافمن، حالت خاصی از **کد پیشوندی** است.

کدگذاری: روش پیشوندی و هافمن

a	b	c	d	e	f
16	5	12	17	10	25



(a)

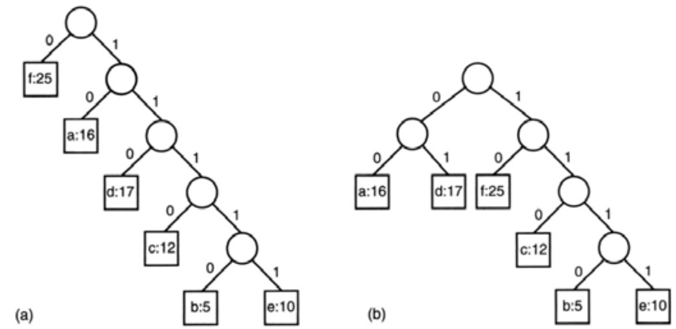


(b)

کد پیشوندی:
برای ایجاد کد پیشوندی برای هر کاراکتر، یک سطح درخت را در نظر می‌گیریم. اولین سطح را به پرتکرارترین کاراکتر نسبت می‌دهیم.

کد هافمن:
برای ایجاد کد هافمن، درخت کد به گونه‌ای تنظیم می‌شود که کدگذاری بهینه استخراج شود.

کدگذاری دودویی متن: هافمن بهینه است!



Character	Frequency	C1(Fixed-Length)	C2 (Prefix)	C3(Huffman)
a	16	000	10	00
b	5	001	11110	1110
c	12	010	1110	110
d	17	011	110	01
e	10	100	11111	1111
f	25	101	0	10

تعداد بیت لازم برای کد کردن متن با استفاده از کدهای فوق:

$$Bits(C1) = 16(3) + 5(3) + 12(3) + 17(3) + 10(3) + 25(3) = 255$$
$$Bits(C2) = 16(2) + 5(5) + 12(4) + 17(3) + 10(5) + 25(1) = 231$$
$$Bits(C3) = 16(2) + 5(4) + 12(3) + 17(2) + 10(4) + 25(2) = \boxed{212}$$

کد هافمن: الگوریتم

n کاراکتر بر حسب تعداد تکرار آنها به صورت صعودی مرتب می‌شوند.
یک **درخت دودویی** به صورت زیر ساخته می‌شود:
در هر مرحله:

- دو عنصری که کمترین تکرار را دارند با هم ادغام می‌شوند.
- عناصر ادغام شده حذف می‌شوند.
- نتیجه‌ی ادغام به صورت یک درخت در لیست فوق اضافه می‌شود.
(وزن عنصر حاصل، برابر با مجموع وزن عناصر ادغام شده است).

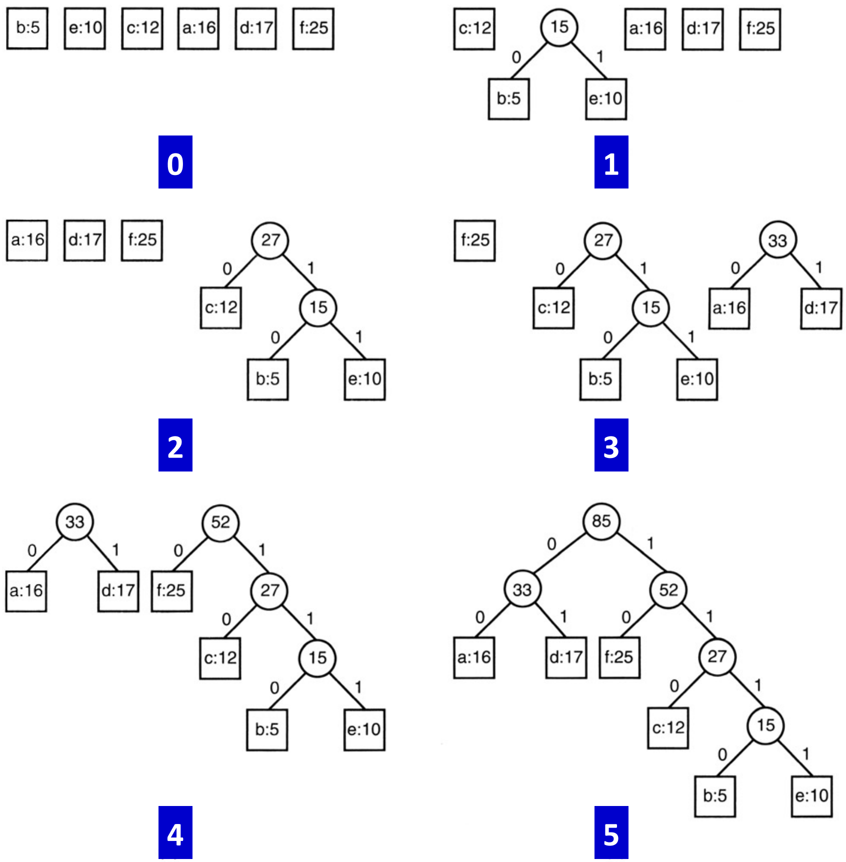
- عملیات فوق آن‌قدر تکرار می‌شود تا همه‌ی عناصر در ادغام استفاده شوند.
- در درخت حاصل:
به یال‌های سمت چپ برچسب 0 و به یال‌های سمت راست برچسب 1 داده می‌شود.
- کد هر کاراکتر می‌شود: دنباله‌ی برچسب‌ها از ریشه تا برگ مربوط به آن کاراکتر

الگوریتم هافمن: زمان اجرا

عملیات اصلی، مرتب‌سازی n کاراکتر بر حسب تعداد تکرار آنهاست، پس: $T(n) \in \Theta(n \log n)$

کد هافمن

مثال



a	00
b	1110
c	110
d	01
e	1111
f	10

Optimal Job Scheduling

n کار با زمان‌های سرویس مختلف داریم.

کارها را به چه ترتیبی انجام بدهیم تا **زمان کل (حضور در سیستم)** می‌نیمم شود.

Job	Service Time
1	5
2	10
3	4

مثال: ۳ کار با زمان‌های ۵، ۱۰ و ۴ واحد داریم.

برای ترتیب $[1{,}2{,}3]$ زمان کل می‌شود:

$$\underbrace{(\mathbf{5})}_{\text{Job 1}} + \underbrace{(\overset{\text{Wait}}{5} + \mathbf{10})}_{\text{Job 2}} + \underbrace{(\overbrace{5+10}^{\text{Wait}} + \mathbf{4})}_{\text{Job 3}} = 39$$

Job	Time in the System
1	5 (*service time*)
2	5 (*wait for job 1*) + 10 (*service time*)
3	5 (*wait for job 1*) + 10 (*wait for job 2*) + 4 (*service time*)

مسأله‌ی زمان‌بندی بهینه‌ی کارها

مثال: ۳ کار با زمان‌های ۵، ۱۰ و ۴ واحد داریم.

Job	Service Time
1	5
2	10
3	4

Schedule	***Total Time in the System***
[1, 2, 3]	5+(5+10)+(5+10+4) = 39
[1, 3, 2]	5+(5+4)+(5+4+10) = 33
[2, 1, 3]	10+(10+5)+(10+5+4) = 44
[2, 3, 1]	10+(10+4)+(10+4+5) = 43
[3, 1, 2]	4+(4+5)+(4+5+10) = 32 جواب بهینه
[3, 2, 1]	4+(4+10)+(4+10+5) = 37

مسأله‌ی زمان‌بندی بهینه‌ی کارها: الگوریتم حریصانه

```
GREEDY-SCHEDULING(A)
    S ← [ ]
    while ¬solution(S) do
        x ← select(A)
        S ← [S, x]
        A ← A − {x}
    return S
```

زمان‌بندی بهینه، با ترتیب صعودی کارها بر حسب زمان سرویس به دست می‌آید.

(۱) انتخاب $x \leftarrow \text{select}(A)$	عنصر بعدی که باید به مجموعه اضافه شود، انتخاب می‌شود. (کار با کوتاه‌ترین زمان سرویس انتخاب می‌شود (حریصانه)).
(۲) بررسی امکان‌پذیری $\text{feasible}(S,x)$	لازم نیست! (راه حل در هر صورت امکان‌پذیر است).
(۳) بررسی راه‌حل $\text{solution}(S)$	تعیین می‌کند که آیا مجموعه‌ی جدید راه حل نمونه است یا خیر؟ (آیا همه‌ی کارها زمان‌بندی شده‌اند؟)

زمان اجرا عملیات اصلی، مرتب‌سازی n کار بر حسب زمان سرویس آنهاست، پس: $T(n) \in \Theta(n \log n)$

مسأله‌ی زمان‌بندی بهینه‌ی کارها با مهلت معین

OPTIMAL JOB SCHEDULING WITH DEADLINE

n کار با مهلت و منفعت مختلف داریم.
هر یک از کارها در یک واحد زمانی انجام می‌شوند.
کارها را به چه ترتیبی انجام بدهیم تا **منفعت کل** ماکزیمم شود.
(لازم نیست همه‌ی کارها در زمان‌بندی وارد شود، فقط باید زیرمجموعه‌ای از آنها انتخاب شود که منفعت کل را ماکزیمم کند).

مثال: ۴ کار مطابق جدول داریم:

زمان ورود همه‌ی کارها: صفر

Job	Deadline	Profit
1	2	30
2	1	35
3	2	25
4	1	40

زمان‌بندی‌های امکان‌پذیر و منفعت حاصل از آنها:

Schedule	Total Profit
[1, 3]	30+25 = 55
[2, 1]	35+30 = 65
[2, 3]	35+25 = 60
[3, 1]	25+30 = 55
[4, 1]	40+30 = 70
[4, 3]	40+25 = 65

جواب بهینه: [4, 1]

(مثلاً زمان‌بندی $[1,4]$ امکان‌پذیر نیست: زیرا اگر کار ۱ انجام شود، مهلت کار ۴ تمام شده است و دیگر اجرای آن منفعتی ندارد!)

مسأله‌ی زمان‌بندی بهینه‌ی کارها با مهلت معین

تعیین زمان‌بندی امکان‌پذیر

یک مجموعه از کارها امکان‌پذیر است **اگر و فقط اگر** مرتب‌شده‌ی صعودی آن بر حسب مهلت‌ها امکان‌پذیر باشد.

الگوریتم حریصانه‌ی زمان‌بندی بهینه‌ی کارها با مهلت معین

- ابتدا کارها را بر اساس منفعت آنها به صورت نزولی مرتب می‌کنیم.
- لیست S را در ابتدا تهی در نظر می‌گیریم.
- در هر مرحله
  - کار با بیشترین منفعت را انتخاب می‌کنیم.
  - اگر زمان‌بندی با اضافه کردن آن کار امکان‌پذیر بود، آن را به S اضافه می‌کنیم.
    - در غیر این صورت آن کار نادیده گرفته می‌شود.

❋ برای تشخیص امکان‌پذیر بودن، کارهای موجود در S را به ترتیب مهلت‌های صعودی مرتب می‌کنیم و امکان‌پذیری آن را بررسی می‌کنیم.

مسأله‌ی زمان‌بندی بهینه‌ی کارها با مهلت معین: الگوریتم حریصانه

```
GREEDY-SCHEDULING-WITH-DEADLINE(A)
    S ← [ ]
    while ¬solution(S) do
        x ← select(A)
        if feasible(S,x) then
    S ← [S, x]
        A ← A − {x}
    return S
```

زمان اجرا

عملیات اصلی:

- مرتب‌سازی n کار بر حسب منفعت:
  $n \log n$
- تشخیص امکان‌پذیر بودن هر کار: n
  - برای n کار: n^2

$$T(n) \in \Theta(n \log n) + \Theta(n^2) = \Theta(n^2)$$

(۱) انتخاب $x \leftarrow \text{select}(A)$	عنصر بعدی که باید به مجموعه اضافه شود، انتخاب می‌شود. (کار با بیشترین منفعت ممکن انتخاب می‌شود (حریصانه)).
(۲) بررسی امکان‌پذیری $\text{feasible}(S,x)$	آیا اضافه کردن این کار به زمان‌بندی امکان‌پذیر می‌رسد؟ (کارهای موجود در S را به ترتیب مهلت‌های صعودی مرتب می‌کنیم و امکان‌پذیری آن را بررسی می‌کنیم.)
(۳) بررسی راه‌حل $\text{solution}(S)$	تعیین می‌کند که آیا مجموعه‌ی جدید راه حل نمونه است یا خیر؟ (آیا همه‌ی کارها بررسی شده‌اند؟)

مسأله‌ی زمان‌بندی بهینه‌ی کارها با مهلت معین: مثال

Job	*Deadline*	*Profit*
1	3	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

زمان ورود همه‌ی کارها: صفر
زمان سرویس هر کار: یک واحد

1	*S* is set to Ø.
2	*S* is set to {1} because the sequence [1] is feasible.
3	*S* is set to {1,2} because the sequence [2,1] is feasible.
4	{1,2, 3} is rejected because there is no feasible sequence for this set.
5	*S* is set to {1,2,4} because the sequence [2,1,4] is feasible.
6	{1,2,4,5} is rejected because there is no feasible sequence for this set.
7	{1,2,4,6} is rejected because there is no feasible sequence for this set.
8	{1,2,4,7} is rejected because there is no feasible sequence for this set.

The final value of *S* is {1,2,4}, and a feasible sequence for this set is [2,1,4]. Because jobs 1 and 4 both have deadlines of 3, we could use the feasible sequence [2,4,1] instead.

مسئله‌ی کوله‌پشتی کسری

FRACTIONAL KNAPSACK

n قطعه با وزن‌ها و ارزش‌های مختلف داریم.
یک کوله‌پشتی با ظرفیت W موجود است.
می‌خواهیم از هر قطعه کسری را انتخاب کنیم که
- کل کوله‌پشتی پر شود، و
- مجموع ارزش کل ظرف ماکزیمم شود.

مثال:

$W = 30$

قطعه	وزن	ارزش	نسبت
i	w_i	p_i	p_i / w_i
1	5	50	50 / 5 = 10
2	10	60	60 / 10 = 6
3	20	140	140 / 20 = 7

سه انتخاب حریصانه:
- به ترتیب صعودی وزن
- به ترتیب نزولی ارزش
- **به ترتیب نزولی نسبت ارزش به وزن (انتخاب بهینه)**

آخرین انتخاب می‌تواند به صورت کسری از قطعه انجام شود.

زمان اجرا عملیات اصلی، مرتب‌سازی n قطعه بر اساس نسبت ارزش به وزن آنهاست، پس: $T(n) \in \Theta(n \log n)$

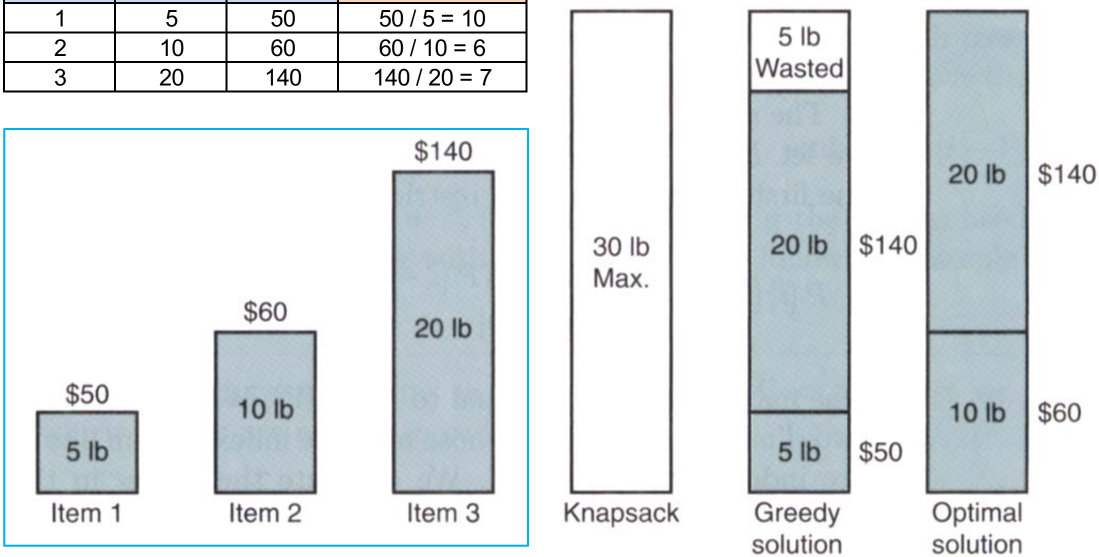
مسئله‌ی کوله‌پشتی 0/1

0/1 Knapsack

در مسئله‌ی کوله‌پشتی $0/1$، یک قطعه باید برداشته شود یا اصلاً برداشته نشود.

مسئله‌ی کوله‌پشتی $0/1$ با روش حریصانه لزوماً به جواب بهینه نمی‌رسد.
(زیرا می‌تواند ظرفیت کوله‌پشتی را هدر بدهد.)

قطعه	وزن	ارزش	نسبت
i	w_i	p_i	p_i / w_i
1	5	50	50 / 5 = 10
2	10	60	60 / 10 = 6
3	20	140	140 / 20 = 7



مسئله‌ی کوله‌پشتی کسری

FRACTIONAL KNAPSACK

$$S = \{item_1, item_2, \ldots, item_n\}$$
$$w_i = weight(item_i)$$
$$p_i = value(item_i)$$
$$W = \text{total weight of knapsack}$$

$$\boxed{\begin{array}{ll} \text{select } A \subseteq S & \\ \text{maximize} & \displaystyle\sum_{item_i \in A} p_i x_i \\ \text{subject to} & \displaystyle\sum_{item_i \in A} w_i x_i \le W \\ \text{where} & 0 \le x_i \le 1, \quad 1 \le i \le n \end{array}}$$

مسئله‌ی کوله‌پشتی 0/1

0/1 KNAPSACK

$$S = \{item_1, item_2, \ldots, item_n\}$$
$$w_i = weight(item_i)$$
$$p_i = value(item_i)$$
$$W = \text{total weight of knapsack}$$

$$\text{select } A \subseteq S$$
$$\text{maximize } \sum_{item_i \in A} p_i x_i$$
$$\text{subject to } \sum_{item_i \in A} w_i x_i \le W$$
$$\text{where } x_i \in \{0,1\}, \quad 1 \le i \le n$$

مسئله‌ی انتخاب فعالیت‌ها

(زمان‌بندی بازه‌ای)

Activity Selection Problem

n فعالیت (با شماره‌های 1 تا n) می‌خواهند از یک منبع استفاده کنند.

- در هر زمان فقط یک فعالیت می‌تواند از این تک منبع استفاده کند.
  (فعالیت‌ها نمی‌توانند همپوشانی زمانی داشته باشند!)
- هر فعالیت i دارای یک زمان شروع s_i و یک زمان پایان f_i است که $s_i \le f_i$.
- فعالیت i در صورت انتخاب شدن در بازه‌ی زمانی $[s_i, f_i)$ اجرا خواهد شد.

هدف: انتخاب حداکثر تعداد فعالیت‌های ناهمپوشان زمانی

مثال: n سخنران که می‌خواهند از یک سالن اجتماعات استفاده کنند.
مثلاً: انتخاب سخنرانی‌هایی که تعداد کل سخنران‌ها با توجه به بازه‌های زمانی ناهمپوشان ماکزیمم شود.

مسئله‌ی انتخاب فعالیت‌ها

الگوریتم حریصانه

Activity Selection Problem

n فعالیت را برحسب زمان‌های پایان آنها به صورت صعودی مرتب می‌کنیم:

$$f_1 \leq f_2 \leq f_3 \leq \ldots \leq f_n$$

حال، فعالیت 1 را انتخاب می‌کنیم.

سپس برای سایر فعالیت‌ها:

اگر زمان شروع آن فعالیت بزرگتر یا مساوی با زمان پایان آخرین فعالیت انتخاب شده بود،

آن فعالیت را انتخاب می‌کنیم.

با فرض مرتب بودن f :

```
ACTIVITY-SELECTION(s, f)
    A ← {1}
    j ← 1
    for i ← 2 to n do
        if s[i] ≥ f[j] then
            A ← A ∪ {i}
            j ← i
    return A
```

مسئله‌ی انتخاب فعالیت‌ها

الگوریتم حریصانه: زمان اجرا

ACTIVITY SELECTION PROBLEM

با فرض مرتب بودن f :

$$
\begin{array}{l}
\textsc{Activity-Selection}(s, f) \\
\quad A \leftarrow \{1\} \\
\quad j \leftarrow 1 \\
\quad \textbf{for } i \leftarrow 2 \textbf{ to } n \textbf{ do} \\
\quad\quad \textbf{if } s[i] \geq f[j] \textbf{ then} \\
\quad\quad\quad A \leftarrow A \cup \{i\} \\
\quad\quad\quad j \leftarrow i \\
\quad \textbf{return } A
\end{array}
$$

$$T(n) \in \underbrace{\Theta(n \log n)}_{\text{مرتب‌سازی}} + \underbrace{\Theta(n)}_{\text{حلقه‌ی انتخاب}} = \Theta(n \log n)$$

مسئله‌ی انتخاب فعالیت‌ها

مثال

ACTIVITY SELECTION PROBLEM

i	1	2	3	4	5	6	7	8
$s[i]$	1	3	0	5	3	5	6	8
$f[i]$	4	5	6	7	8	9	10	11

مرتب بر حسب f:

i	$s[i] \geq f[j]?$	A	j
1		$\{1\}$	1
2	**flase**	–	
3	**flase**	–	
4	**true**	$\{1, 4\}$	4
5	**flase**	–	–
6	**flase**	–	–
7	**flase**	–	–
8	**true**	$\boxed{\{1, 4, 8\}}$	8