

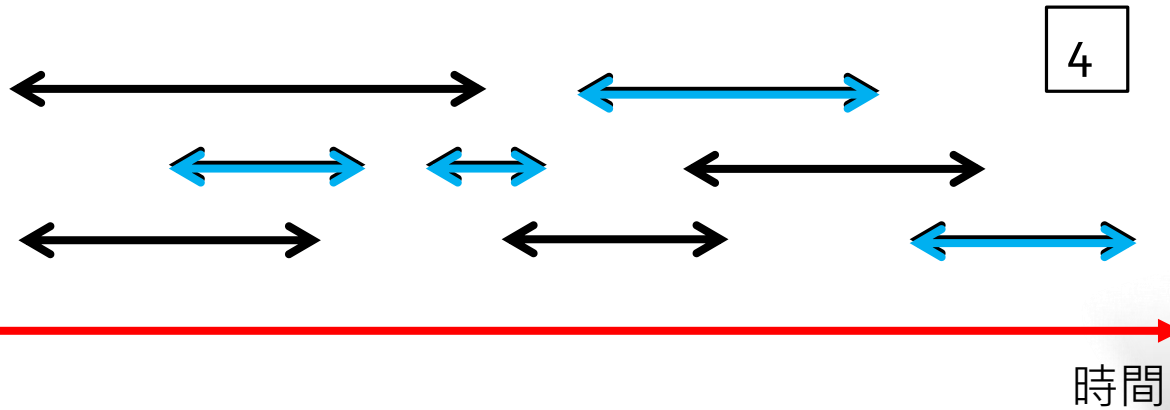


Greedy Algorithms

Michael Tsai

2013/10/17

超級電腦排程問題



- 需要交給超級電腦的工作若干
- 每樣工作有選定的開始與結束時間
- 如何選出一組可以執行的工作(執行時間不重疊), 使這些工作的數量最大?

問題定義

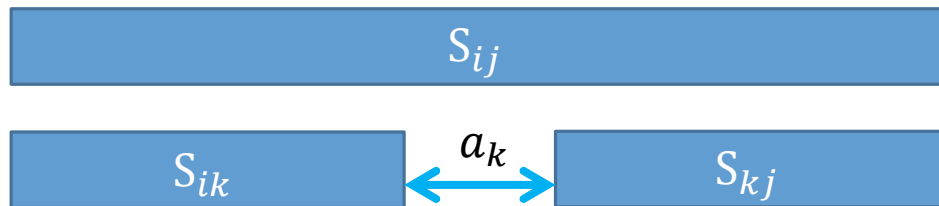


- 集合 $S = \{a_1, a_2, \dots, a_n\}$
- 每一個工作 a_i 有開始時間 s_i 及結束時間 f_i
- $0 \leq s_i < f_i < \infty$
- 每個工作執行時間為 $[s_i, f_i)$
- 可以選兩個工作, 其中一個工作的結束時間和另外一個工作的開始時間一樣
- 當 $s_i \geq f_j$ 或 $s_j \geq f_i$ 時 a_i 和 a_j 可以被同時選入
- 工作已經照結束時間排好了:
- $f_1 \leq f_2 \leq f_3 \leq \dots \leq f_{n-1} \leq f_n$



先請出神聖的dynamic programming 尚方寶劍

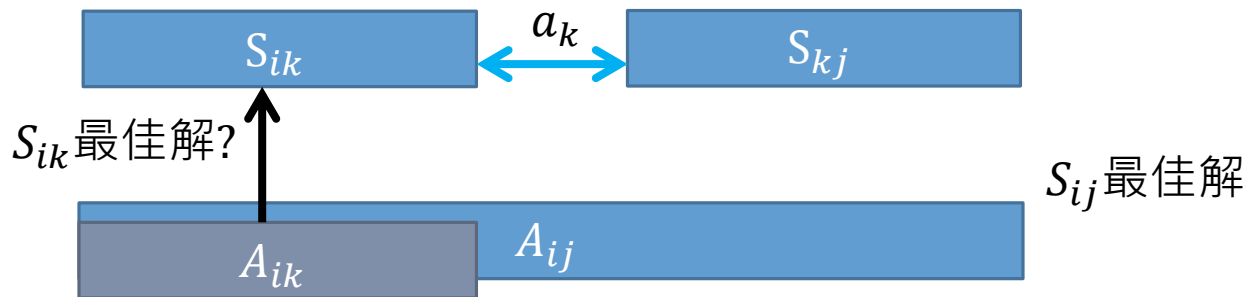
- 定義子問題:
 S_{ij} 為 a_i 完成後, a_j 開始前的所有工作集合
- 要找出 S_{ij} 中的最大可執行集合, 假設其中包含 a_k



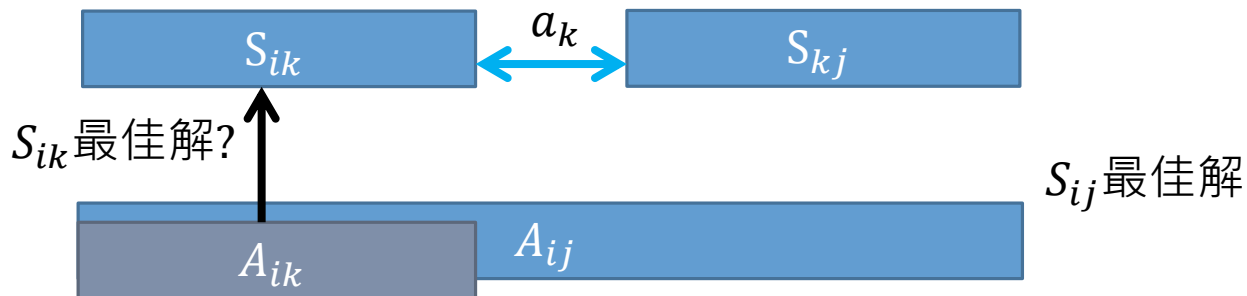
- 所以假設 A_{ij} 為 S_{ij} 的最佳解, 則 $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$

Dynamic programming 尚方寶劍

- Optimal Substructure:
- 證明 A_{ij} 中包含 S_{ik} 與 S_{kj} 的最佳解(即欲證明 A_{ik} 為 S_{ik} 之最佳解, 或 A_{kj} 為 S_{kj} 之最佳解)



Optimal Substructure



- 假設 A_{ik} 不是 S_{ik} 的最佳解, 則表示有一 A'_{ik} 使 $|A'_{ik}| > |A_{ik}|$
- 但如果這樣
- 則此解之工作數目為 $|A'_{ik}| + 1 + |A_{kj}| > |A_{ik}| + 1 + |A_{kj}| = |A_{ij}|$, 表示 A_{ij} 不是最佳解 (矛盾)
- 因此 A_{ik} 為 S_{ik} 的最佳解

有Optimal substructure, 可以用DP解!

接下來怎麼解呢?

- $c[i, j]$ 代表 S_{ij} 最佳解的工作數目
- 則 $c[i, j] = c[i, k] + 1 + c[k, j]$
- k 是我們的選擇 (不知道選哪一個)

如果我們可以證明我們知道呢?

$$c[i, j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i, k] + 1 + c[k, j]\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

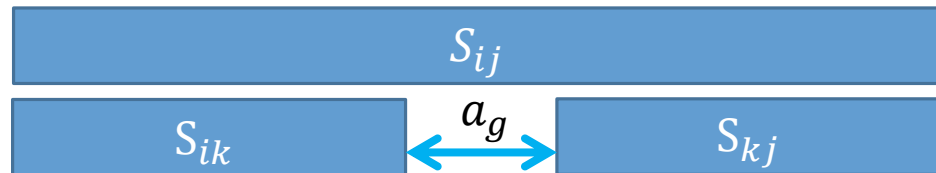
- 接著使用 top-down or bottom-up 方法填表即可



等等, 我們需要用到尚方寶劍嗎?

Greedy Algorithm

- 某些問題, 我們可以知道怎麼做選擇!
- 此稱為greedy choice
- 如果greedy choice為optimal choice, 再加上optimal substructure, 我們就可以非常快速的得到解!

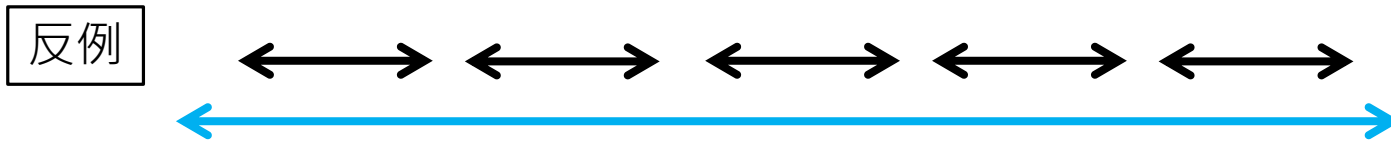


不需要考慮所有 a_k 的選擇!

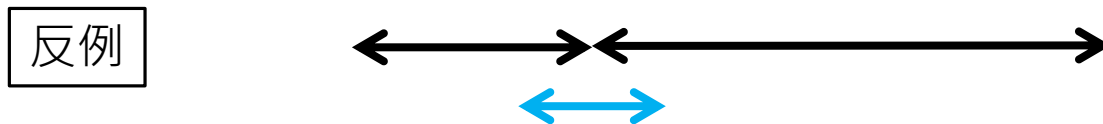
只需要直接解選擇 a_g 之後的subproblem!

怎麼做出選擇？（創造力）

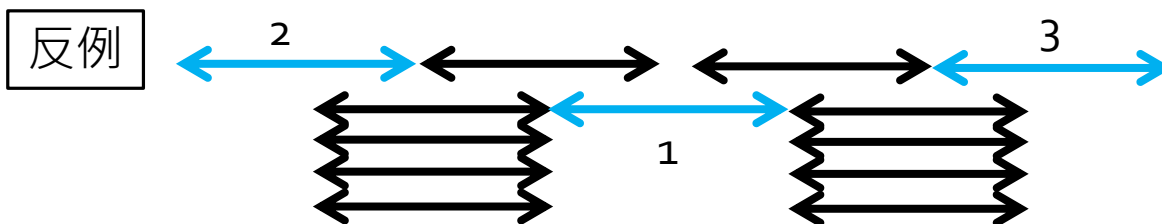
- 每次都選最早開始的工作



- 每次都選花最少時間的工作

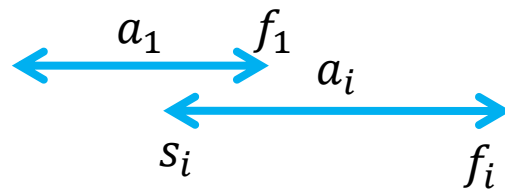


- 每次都選跟其他最少衝突的工作



Greedy choice

- 正確的greedy choice(之一): 選最早結束的.
- 原因: 越早結束, 後面就越多時間讓其他工作執行.
- → 因為原本已經照 f_i 排序好了, 每次我們都選 a_1
- 接下來只需考慮 a_1 結束以後才開始的所有工作
- 因為其他 a_i 的結束時間 $f_i \geq f_1$, s_i 要大於 f_1 才不會重疊.



證明greedy choice是正確的!

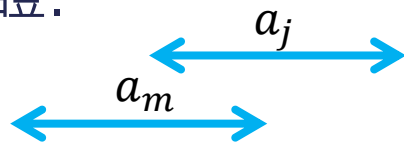
- 定理: 在某subproblem S_k 中 a_m 的完成時間最早, 則 S_k 的某些最佳解(如果有很多個)中一定有 a_m
- 證明: 假設 A_k 是 S_k 的某個最佳解, A_k 裡面完成時間最早的是 a_j .

$$S_k = \{a_i \in S: s_i \geq f_i\}$$

1. 如果 a_j 就是 a_m , 則得證.

2. 如果 a_j 不是 a_m , 則

所以可以把 A_k 中的 a_j 換成 a_m .



其他的

更換後工作數目不變, 和 A_k 一樣多, 因此為另外一個最佳解, 得證.

結論：有時候並不需要DP！

- DP可以用, 但是慢很多. (殺雞不用尚方寶劍)
- Greedy algorithm通常使用top-down的方法:
 1. 根據目前知道的事情選出最好的選擇
 2. 繼續解用了這個選擇之後的subproblem
 3. 重複以上一直到subproblem可以直接解掉
- 重點: 請確定(要證明)greedy choice一定會出現在最佳解裡面!!!

來個pseudo-code

s: 存開始時間的陣列
 f: 存結束時間的陣列
 k: 目前要解的subproblem為 S_k
 n: 總共有幾個task (problem size)

```
Recursive_Activity_Selector(s, f, k, n)
```

```
m=k+1
```

```
while m<=n and s[m]<f[k]
```

```
    m=m+1
```

```
if m<=n
```

```
    return  $\{a_m\} \cup$ 
```

```
Recursive_Activity_Selector(s, f, m, n)
```

```
else
```

```
    return  $\emptyset$ 
```

選完greedy choice以後只剩下
一個recursive call在最後: tail recursive

$\Theta(n)$

來個pseudo-code

```
Greedy_Activity_Selector(s, f)
n=s.length
A={ }
k=1
for m=2 to n
    if s[m]>=f[k]
        A=A ∪ {am}
        k=m
return A
```

 $\Theta(n)$

什麼問題可以使用greedy algorithm來解?

1. 如果做出一個choice之後, 可以找到剩下要解的單一個subproblem (定義好subproblem)
 - 1.999. 決定要怎麼做greedy choice.
2. 有greedy property. (必須證明最佳解裡面一定有greedy choice)
3. 有optimal substructure (必須證明大問題的最佳解裡面有小問題的最佳解)

0-1 背包問題



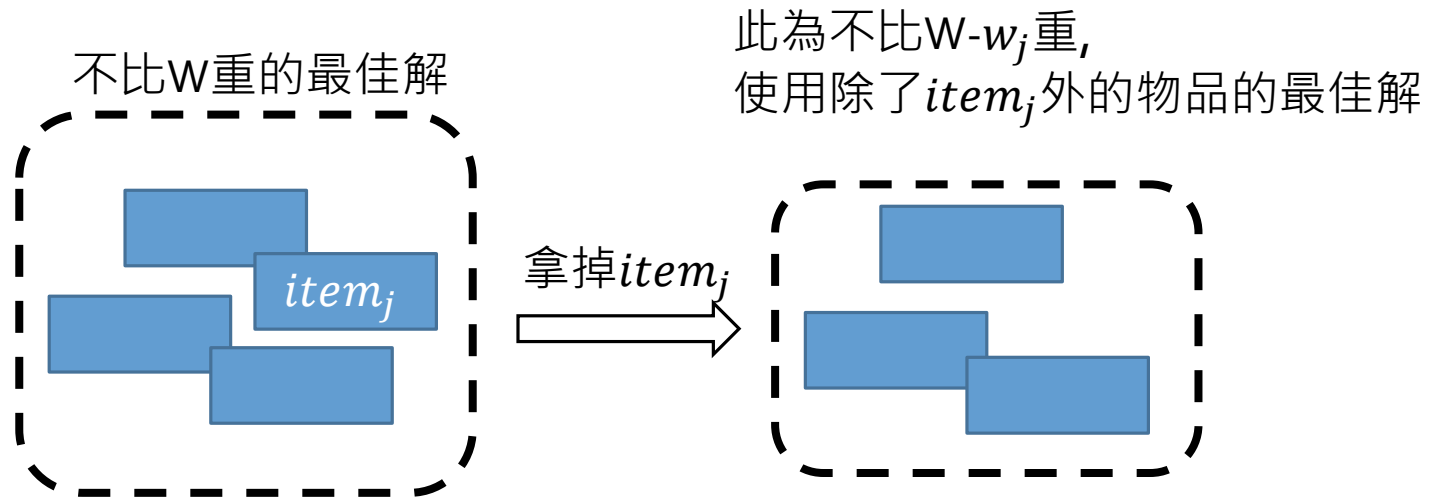
- 某店面有 n 個物品 $\{item_1, item_2, \dots, item_n\}$
- 各價值 $\{v_1, v_2, \dots, v_n\}$, 各重 $\{w_1, w_2, \dots, w_n\}$
- 小偷有一個可以裝 W 這麼重的背包
- 請問要偷走哪些東西可以使小偷拿走的東西總價值最高?

fractional 背包問題

- 某店面有 n 個物品 $\{\text{item}_1, \text{item}_2, \dots, \text{item}_n\}$
- 各價值 $\{v_1, v_2, \dots, v_n\}$, 各重 $\{w_1, w_2, \dots, w_n\}$
- 小偷有一個可以裝 W 這麼重的背包
- **可以拿走“部分物品” (切物品的一部分)**
- 請問要偷走哪些東西可以使小偷拿走的東西總價值最高?

背包問題的optimal substructure

定理:



證明: (適用於0-1 & fractional 背包問題)

1. 假設右邊的不是最佳解的話,
2. 則可以找到一最佳解, 價值更高且不比 $W-w_j$ 重.
3. 此解加上 $item_j$ 以後也還沒有超過 W ,
4. 而且此解加上 v_j 後總價值比左邊的價值還高 (矛盾)

Fractional 背包問題的greedy choice

- 把物品按照單位重量的價值來排序 (也就是 $\frac{v_i}{w_i}$)
 1. 選出單位重量價值最高的(greedy choice).
 2. 如果可用重量(W)比物品重量 w_i 小, 則把背包填滿, 把物品多餘的部分捨棄.
 3. 如果可用重量比物品重量 w_i 大, 則繼續解子問題: 可用重量剩下 $W - w_i$, 把已放入的物品從物品集合中去除.

證明fractional背包問題有greedy property

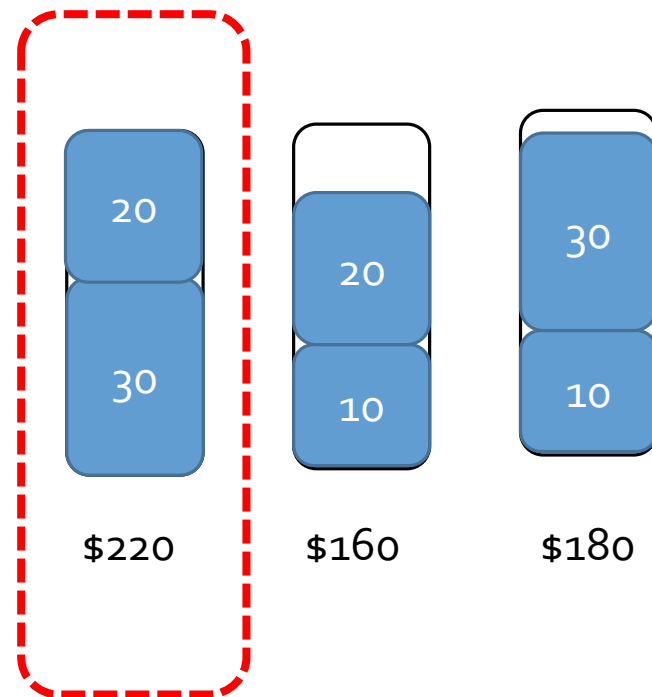
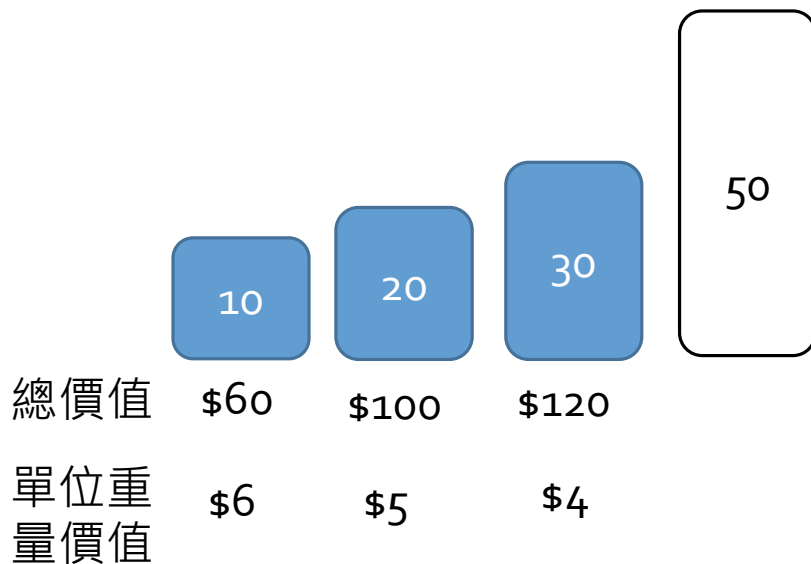
- 其實就是要證明, (某些)最佳解裡面有greedy choice
- 通常使用的方法:
 1. 假設可以拿到最佳解.
 2. 最佳解裡面如果已經有greedy choice的話, 則得證.
 3. 最佳解裡面如果沒有greedy choice的話, 則想辦法把最佳解裡面的一些東西和greedy choice互換. 結果發現這個新解跟greedy choice一樣好 (也是一個最佳解) 或者發現這個新解更好 (矛盾, 所以最佳解裡面不可能沒有greedy choice)

證明fractional背包問題有greedy property

1. 假設可以拿到可用重量為 W 最佳解. 假設 $item_j$ 為單位重量價值最高者(或其中之一).
2. 如果 $W \leq w_j$, 則將最佳解之所有東西都換成 $item_j$.
3. 如果 $W > w_j$, 則將最佳解中的物品按照單位重量價值排序, 取其中重 w_j 之最高價值物品(可能包含部分 $item_j$)與剩餘之 $item_j$ 交換.
4. 因為 $item_j$ 為單位重量價值最高者, 因此作了以上交換以後得到之解, 總價值只可能上升或相等. 但原來已經為最佳解, 因此總價值只可能相等
5.
 - 交換過後的解也是最佳解
 - 因此至少很多種最佳解中其中幾種裡面有greedy choice.

此一greedy choice不適用於0-1背包問題!

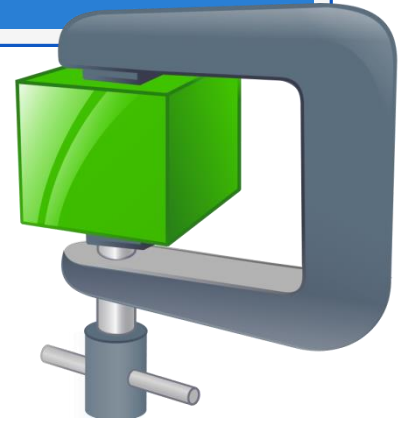
○ 反例:



注意: 但是這不代表這個問題一定沒有greedy property! 只是表示這個greedy choice不是正確的!

最佳解, 但是沒有greedy choice!

因為如果有空間沒有用到, 就會使單位重量的價值下降!!



Huffman codes

	a	b	c	d	e	f
出現次數	45k	13k	12k	16k	9k	5k
Fixed-length codeword	000	001	010	011	100	101
總共需要300 Kb						
Variable-length codeword	0	101	100	111	1101	1100
總共需要224 Kb						

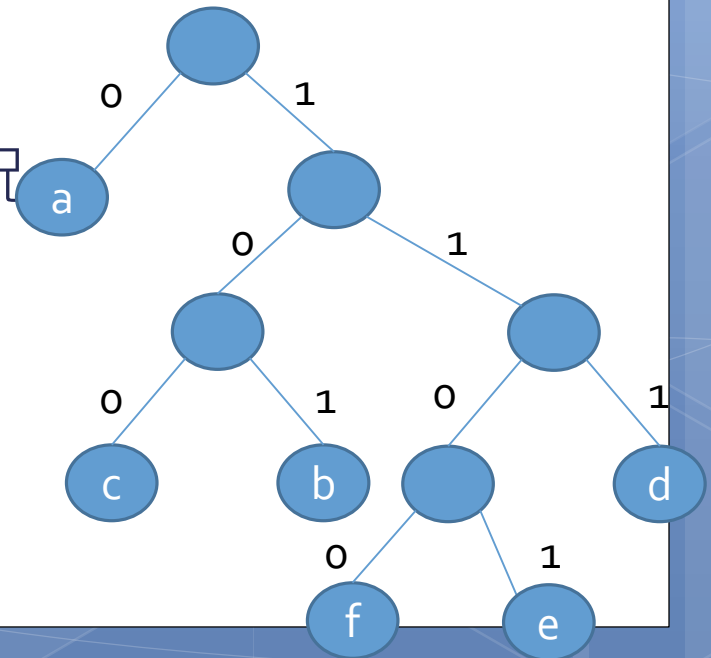
越不常出現的字, codeword應該越長!



再省25%

Prefix code

- 定義: 某個code中, 沒有任何一個codeword是另外一個codeword的prefix, 則稱為prefix code.
- Encode: "abc" → "0 101 100"
- Decode: "0 101 100" → "abc"
- 使用右邊的decoding tree, 走到leaf就解出一個字母
- Prefix code就不會有混淆的狀況產生.
- 例: a=001 b=00 c=1
那看到001是"a" or "bc"?
- 注意: 此非binary search tree!

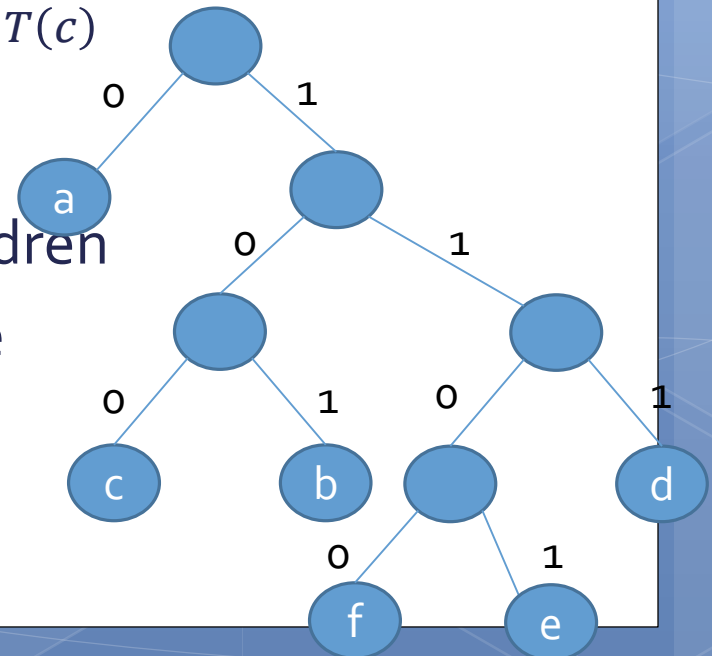


如何產生最佳的code呢?

- 想要產生一棵decode tree, 使得檔案大小能最小.
- 檔案大小(cost):

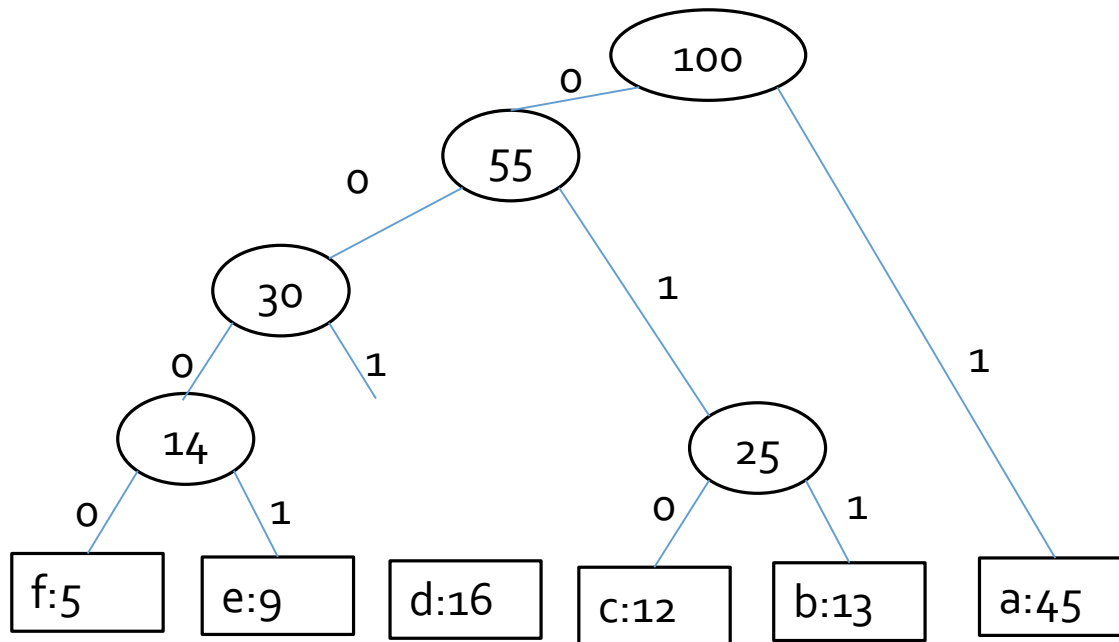
$$B(T) = \sum_{c \in C} c.\text{freq} \cdot d_{T(c)}$$

- Full binary tree
每個non-leaf node都有兩個children
- 一個optimal code畫成的decode tree一定是full binary tree.



Huffman code

- Huffman發明了一種使用greedy algorithm產生 optimal prefix code的方法, 稱為Huffman Code



Pseudo-code

Huffman (C)

$n = |C|$

$Q = C$

起始Priority queue: $O(n)$

for $i = 1$ to $n - 1$

 allocate a new node z

$z.\text{left} = \text{Extract_Min}(Q)$

$z.\text{right} = \text{Extract_Min}(Q)$

在Priority Queue Q 拿出item:
 $O(\log n)$

$z.\text{freq} = z.\text{left}.\text{freq} + z.\text{right}.\text{freq}$

$\text{Insert}(Q, z)$

在Priority Queue Q 插入 z : $O(\log n)$

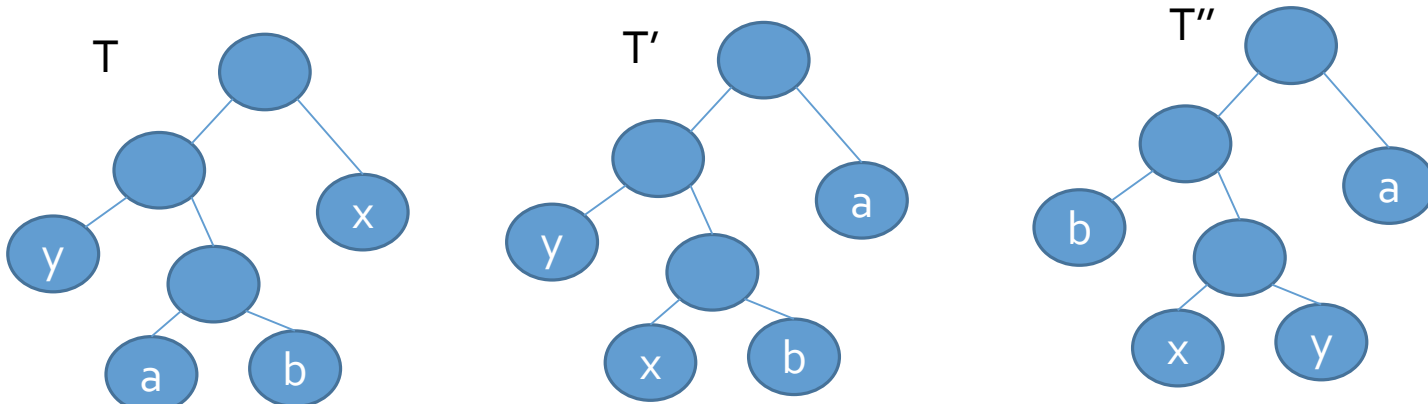
return $\text{Extract_Min}(Q)$

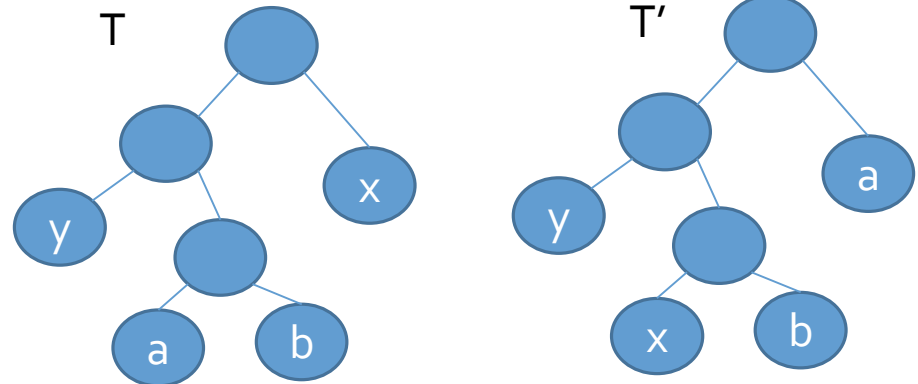
證明最佳解裡面一定有greedy choice

- 定理: 假設所有的字都在集合 C 裡面. 每個字 $c \in C$, 出現的頻率為 $c.freq$. 假設 x 和 y 為 C 中頻率最低的. 則一定有一組最佳的code是使得 x 和 y 的codeword長度一樣且只有最後一個bit不同.
- 證明:
 1. 假設有 T 代表任一的最佳code. 在其中假設 a 和 b 在 T 中為depth最大且為sibling的兩個字元. 我們可以假設 $a.freq \leq b.freq$ 及 $x.freq \leq y.freq$.
 2. 因為" x 和 y 為 C 中頻率最低的", 所以 $x.freq \leq a.freq$ 及 $y.freq \leq b.freq$.

證明最佳解裡面一定有greedy choice

3. 如果 $x.freq = b.freq$, 則 $x.freq = a.freq = y.freq = b.freq$. 如此的話可以直接得證 (可以把 x 和 a 交換, y 和 b 交換, 則得到另一 optimal code, x 和 y 的codeword長度一樣且只有最後一個bit不同)
4. 如果 $x.freq \neq b.freq$, 則先將 a, x 交換(得到 T'), 再將 y, b 交換(得到 T'')





$$\bullet B(T) - B(T')$$

$$= \sum_{c \in C} c.\text{freq} \cdot d_T(c) - \sum_{c \in C} c.\text{freq} \cdot d_{T'}(c)$$

$$= x.\text{freq} \cdot d_T(x) + a.\text{freq} \cdot d_T(a) - x.\text{freq} \cdot d_{T'}(x) - a.\text{freq} \cdot d_{T'}(a)$$

$$= x.\text{freq} \cdot d_T(x) + a.\text{freq} \cdot d_T(a) - x.\text{freq} \cdot d_T(a) - a.\text{freq} \cdot d_T(x)$$

$$= (a.\text{freq} - x.\text{freq})(d_T(a) - d_T(x))$$

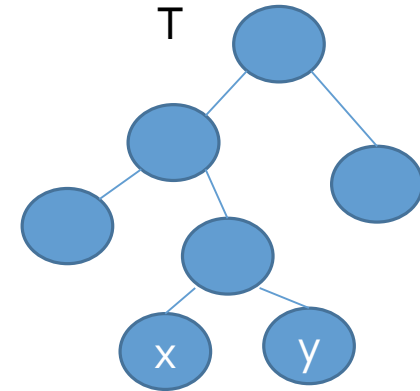
$$\geq 0$$

$B(T'')$. 但因T已為最佳解, 所以只可能 $B(T) = B(T'')$, T''

證明題目有optimal substructure

C

T 為 C 之最佳解

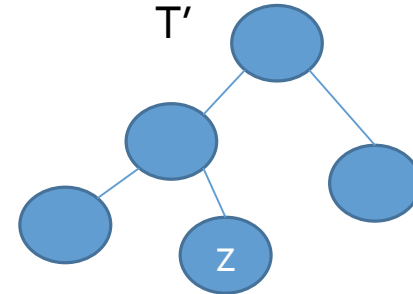


$$C' = C - \{x, y\} \cup \{z\}$$

最佳解為 T'



z 為一新字元, $z.freq = x.freq + y.freq$



$$\begin{aligned} B(T) &= B(T') - z.freq \cdot d_{T'}(z) + x.freq \cdot d_T(x) + y.freq \cdot d_T(y) \\ &= B(T') - (x.freq + y.freq) \cdot d_{T'}(z) + x.freq \cdot (1 + d_{T'}(z)) + y.freq \cdot (1 + d_{T'}(z)) \\ &= B(T') + x.freq + y.freq \end{aligned}$$

證明題目有optimal substructure

