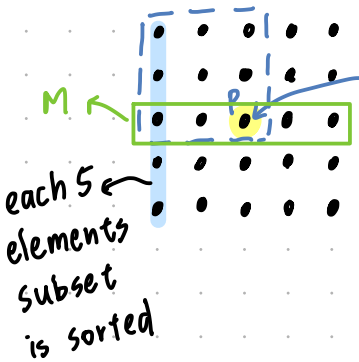
1. The idea of this prune & search approach on selection problem is to find a fraction of elements that does not contain the $k^{th}$ element, and discard the fraction of elements each iteration.

By dividing into groups of 5 elements, we can find the median p of all medians of groups efficiently.




p is the median of M
by choosing p, we can always prune away at least $n/4$ elements each iteration

This is a prune and search algorithm since it prunes away a fraction of input recursively.

② No, this algorithm will not work in linear time

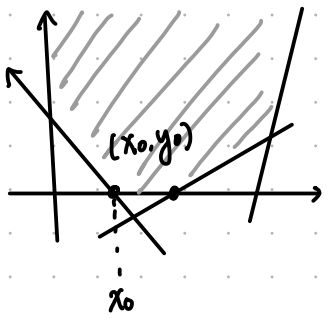
# of elements that are larger than the median of medians $\geq$ (at least) $2\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{7}\right\rceil\right\rceil-6\right)$

# of elements that are smaller than the median of medians $\geq$ (at least) $2\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{7}\right\rceil\right\rceil-6\right)$

It has at most $n-\left(\frac{n}{7}-12\right)$ can be done in $T\left(\frac{6}{7}n+12\right)$

$\therefore T(n) = T\left(\left\lceil\frac{n}{7}\right\rceil\right) + T\left(\frac{6}{7}n+12\right) + O(n) \geq O(n)$ ※

2. (a) True. as the graph shown




$F(x)$: the boundary

$F(x_0) = \min_{-\infty<x<\infty} F(x)$

(b) 觀察 $F(x_m)$ 落在的 constraint 上

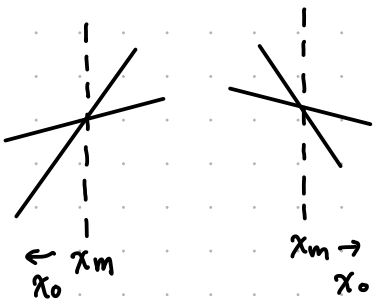
$g_{max}$: $F(x_m)$ 落在斜率 max 的 constraint

$g_{min}$: $F(x_m)$ 落在斜率 min 的 constraint

① $g_{max} > 0, g_{min} < 0 \Rightarrow x_0 = x_m$

② $g_{max} > 0, g_{min} > 0 \Rightarrow x_0 < x_m$

③ $g_{max} < 0, g_{min} < 0 \Rightarrow x_0 > x_m$




(c) Check all constraints' 的交點, $(x_{ij}, y_{ij})$ 為座標

if $x_{ij} > x_m \Rightarrow$ constraint $i, j$ 中必有其中一個是不影響. 與結果無關 (useless)

即可將其刪除.

(d) 將 constraints 之交點按 x 座標排序, 取某 $x_m$

使 $x_{ij} < x_m$ 的 intersections 佔一半, $x_{ij} > x_m$ 的 intersections 佔一半

$\Rightarrow$ 在 prune 時可以確保刪除 $\frac{1}{2} \times \lfloor \frac{n}{2} \rfloor = \lfloor \frac{n}{4} \rfloor$ 個 constraints.

(e) $\lfloor \frac{n}{4} \rfloor$ constraints pruned away

$\because \lfloor \frac{n}{2} \rfloor$ intersections $\therefore \frac{1}{2} \lfloor \frac{n}{2} \rfloor$ Constraints pruned each iteration

$$T(n) = T\left(\frac{3n}{4}\right) + O(n) = O(n)$$

↓

intersections of 2 lines found in $O(n)$

median found in $O(n)$

scanning all constraints: $O(n)$

scanning all intersecting pairs: $O(n)$

3. S: Set of n identical coins, one is fake

(1) $|S| = 1 \Rightarrow$ the coin is fake

(2) $|S| = 2 \Rightarrow$ the heavier one is fake

(3) $|S| \geq 3 \Rightarrow$ Partition S into $S_1, S_2, S_3$ where $S_1$ and $S_2$ has $\lfloor \frac{n}{3} \rfloor$ coins and $S_3$ contains the remaining coins if $|S|$ is odd.

(4) Compare the weight of $S_1$ and $S_2$ (denote $W_i$ as weight)

① If $W_1 > W_2$, the fake coin is in $S_1 \Rightarrow$ prune $S_2, S_3$

② Else If $W_1 < W_2$, the fake coin is in $S_2 \Rightarrow$ prune $S_1, S_3$

③ Else, the fake coin is in $S_3$

(5) If the fake coin is not found, start the next iteration with the remaining set of coins.

With this algorithm, we prune at least $\lfloor \frac{n}{2} \rfloor$ coins away each iteration.
In worst case, we will do the algorithm until $|S| = 1$.
Assume that the number of weighting needed is w, then $2^w = n \Rightarrow w = \log_2 n$
So the number of weighting needed in worst case is $\log_2 n$ *

4. $A = a_1 a_2 \dots \boxed{a_m}$
$B = b_1 b_2 \dots \boxed{b_n}$

compare the last character of both sequence

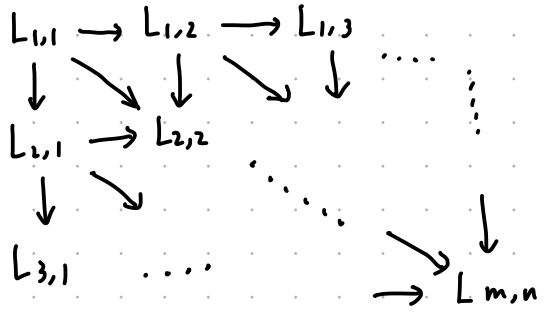
① $a_m = b_n$
$\Rightarrow$ LCS must contain $a_m$ (or $b_n$)
$\Rightarrow$ Continue finding LCS for $a_1 a_2 \dots a_{m-1}$ and $b_1 b_2 \dots b_{n-1}$

② $a_m \neq b_n$
$\Rightarrow$ Continue finding LCS for $a_1 a_2 \dots a_m$ with $b_1 b_2 \dots b_{n-1}$ and $a_1 a_2 \dots a_{m-1}$ with $b_1 b_2 \dots b_n$
$\Rightarrow$ The longer result will be LCS

$L_{ij}$: the lengh of LCS of A and B where $L_{00} = 0 = L_{i0} = L_{0j}$, $1 \leq i \leq m$, $1 \leq j \leq n$

$$L_{ij} = \begin{cases} L_{i-1,j-1} + 1, & \text{if } a_i = b_j \\ \max\{L_{i,j-1}, L_{i-1,j}\}, & \text{if } a_i \neq b_j \end{cases}$$

This recursive formula gives that we may first find $L_{1,1}$ then $L_{1,2}$ and $L_{2,1}$ and then $L_{1,3}$, $L_{2,2}$, $L_{3,1}$ and so on.



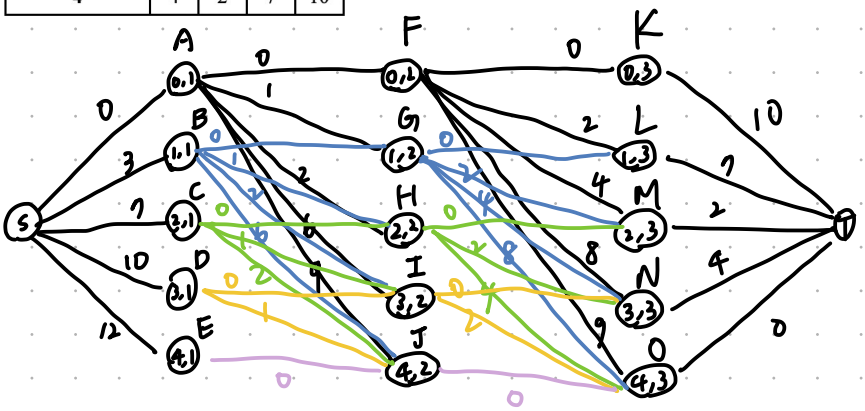

Time complexity: $O(mn)$

Running the algorithm. we calculate $L_{1,1}, L_{1,2}, L_{1,3} \dots L_{1,n}, L_{2,1}, \dots L_{2,n}, L_{3,1} \dots L_{3,n} \dots L_{m,n}$. Calculating each $L_{i,j}$ cost constant time so the time complexity is $O(m \times n)$

5.

6.

| Project \ Resource | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 3 | 7 | 10 | 12 |
| 2 | 1 | 2 | 6 | 9 |
| 3 | 2 | 4 | 8 | 9 |
| 4 | 4 | 2 | 7 | 10 |

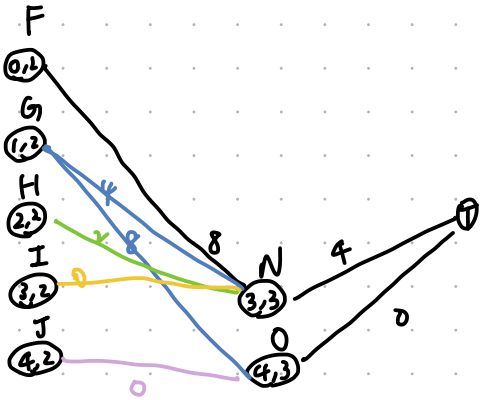



$d(K,T) = 10$
$d(L,T) = 7$
$d(M,T) = 2$
$d(N,T) = 4$
$d(O,T) = 0$

Find the longest path from S → T (backward reasoning)

$$d(F,T) = \max \begin{cases} d(F,K)+d(K,T) \\ d(F,L)+d(L,T) \\ d(F,M)+d(M,T) \\ d(F,N)+d(N,T) \\ d(F,O)+d(O,T) \end{cases} = \max\{10,9,6,12,9\} = 12$$

$d(J,T) = 0$




$$d(G,T) = \max \begin{cases} d(G,L)+d(L,T) \\ d(G,M)+d(M,T) \\ d(G,N)+d(N,T) \\ d(G,O)+d(O,T) \end{cases} = \max\{7,4,8,8\} = 8$$

$$d(H,T) = \max \begin{cases} d(H,M)+d(M,T) \\ d(H,N)+d(N,T) \\ d(H,O)+d(O,T) \end{cases} = \max\{2,6,4\} = 6$$

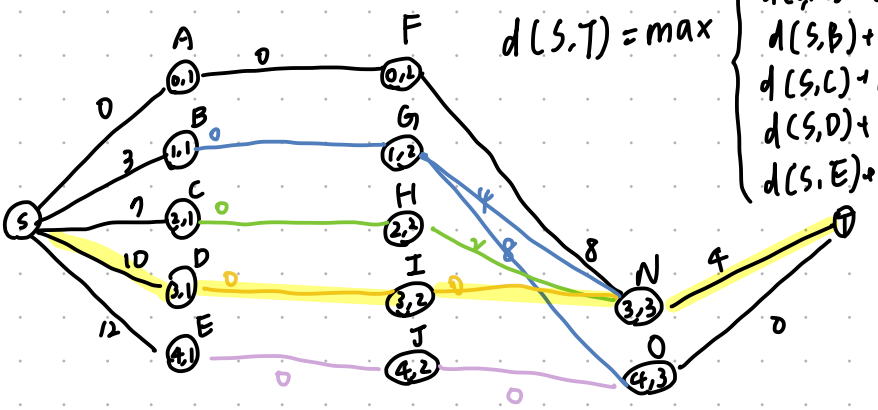
$$d(I,T) = \max \begin{cases} d(I,N)+d(N,T) \\ d(I,O)+d(O,T) \end{cases} = \max\{4,2\} = 4$$

$$d(A,T) = \max \begin{cases} d(A,F)+d(F,T) \\ d(A,G)+d(G,T) \\ d(A,H)+d(H,T) \\ d(A,I)+d(I,T) \\ d(A,J)+d(J,T) \end{cases} = \max\{12,9,8,10,9\} = 12$$

$$d(B,T) = \max \begin{cases} d(B,G)+d(G,T) \\ d(B,H)+d(H,T) \\ d(B,I)+d(I,T) \\ d(B,J)+d(J,T) \end{cases} = \max\{8,7,6,6\} = 8$$

$$d(C,T) = \max\begin{Bmatrix} d(C,H)+d(H,T) \\ d(C,I)+d(I,T) \\ d(C,J)+d(J,T) \end{Bmatrix} = \max\{6,5,2\} = 6 \qquad d(E,T) = 0$$

$$d(D,T) = \max\begin{Bmatrix} d(D,I)+d(I,T) \\ d(D,J)+d(J,T) \end{Bmatrix} = \max\{4,1\} = 4$$




$$d(S,T) = \max\begin{Bmatrix} d(S,A)+d(A,T) \\ d(S,B)+d(B,T) \\ d(S,C)+d(C,T) \\ d(S,D)+d(D,T) \\ d(S,E)+d(E,T) \end{Bmatrix} = \max\{12,11,13,14,12\} = 14$$

The longest path is $S \to D \to I \to N \to T$

3 resources allocated to project 1
0 resources allocated to project 2
0 resources allocated to project 3
1 resource allocated to project 4

Time complexity:

# of calculation: $(m+1) + (m+1+m+\cdots+1) \times (n-2) + (m+1)$

$\Rightarrow O(mn)$

7.

$-x_1 \vee -x_2 \vee x_3$ (1)
& $-x_1$ (2)
& $x_2 \vee x_3$ (3)
& $-x_3$ (4)

(3)+(4) → $x_2$ (5)
(1)+(5) → $-x_1 \vee x_3$ (6)
(4)+(6) → $-x_1$ (7)

⇒ no any new clause
⇒ C is satisfiable *

8. Prove halting problem an NP-hard problem.

Halting problem: undecidable problem ⇒ it is not in NP, and not in NP-complete

$$\text{Halt}(A,I) = \begin{cases} 1 & \text{if } A(I) \text{ halts} \\ 0, & \text{o/w} \end{cases}$$

T(A): if Halt(A,A)=1 then infinite loop
else halt

A problem A is NP-hard if every NP problem $\propto$ A

To prove this, we need to show that SAT $\propto$ halting problem.

Let algorithm A be an algorithm whose input is a prepositional formula X. where X has n variables.

This algorithm A will run all possible truth assignments and check if X is satisfiable.
If X is satisfiable than A halt
else, A enters infinite loop
⇒ A halts iff X is satisfiable
If we had a polynomial time algorithm for halting problem, we could solve the SAT problem in polynomial time by using A and X as input for halting problem.
⇒ Halting problem is an NP-hard problem, it is not in NP

Q. We shall prove that SAT $\propto$ 6SAT, $F_i$ denotes boolean formula

$\forall$ $F_1$ in SAT, we create another $F_2$ that there are 6 literals in each clause

$\Rightarrow$ $F_1$ is satisfiable iff $F_2$ is satisfiable

10. TSP is a NP problem since the process of checking the tour, summing cost and check minimum cost can be finished in polynomial time.

Reduce the x-y path problem to TSP:

Given $G(V, E)$, $|V| = n$

Let $c(e) = 1 \ \forall e \in E$

We add edges $E'$ to $G$ and make $G$ complete graph and let $c(e) = 2 \ \forall e \in E'$.

↳ polynomial time

Given this cost, if the result to TSP is "yes" then there is a cycle that visits all nodes exactly once. The edges it uses are from $E$, therefore there is for sure a path from $x$ to $y$ that visits every vertex exactly once. $(x \in V)$ $(y \in V)$

⇒ The x-y path problem reduce to TSP problem.

11. (a) True, $O(n^k)$, this is true only when $k$ is a positive constant.

(b) True

(c) True

(d) False. The description here is about traveling salesperson "optimization" problem, which is an NP-hard problem.

(e) True