# The Greedy Method (貪進法)

## Outline

- Kruskal’s algorithm Minimum spanning trees (MST)
- Prime’s MST algorithm
- The single-source shortest path problem
- Linear Merge Algorithm
- The minimal cycle basis problem
- The 2-terminal one to any special channel routing problem
- The Minimum Cooperative Guards Problem for 1-Spiral Polygons Solved by the Greedy Method

## 學習目標

- Greedy method的設計概念與限制。
- Minimal Spanning Tree的演算法設計與分析
  - Kruskal's algorithm
  - Prime's algorithm
- 最短路徑問題的演算法設計與分析
- 最佳合併順序問題的演算法設計與分析
- 最佳編碼問題的演算法設計與分析

# 學習目標

- Minimal cycle bases問題的演算法設計與分析
- 2-terminal routing問題的演算法設計與分析
- The Minimum Cooperative Guards Problem for 1-Spiral Polygons 問題的演算法設計與分析
- Knapsack 問題的演算法設計與分析

## The greedy method

- Suppose that a problem can be solved by a sequence of decisions.
- The greedy method has that each decision is locally optimal.  These locally optimal solutions will finally add up to a globally optimal solution.
- Only a few optimization problems can be solved by the greedy method.
- Greedy method at least produces a solution which is usually acceptable.

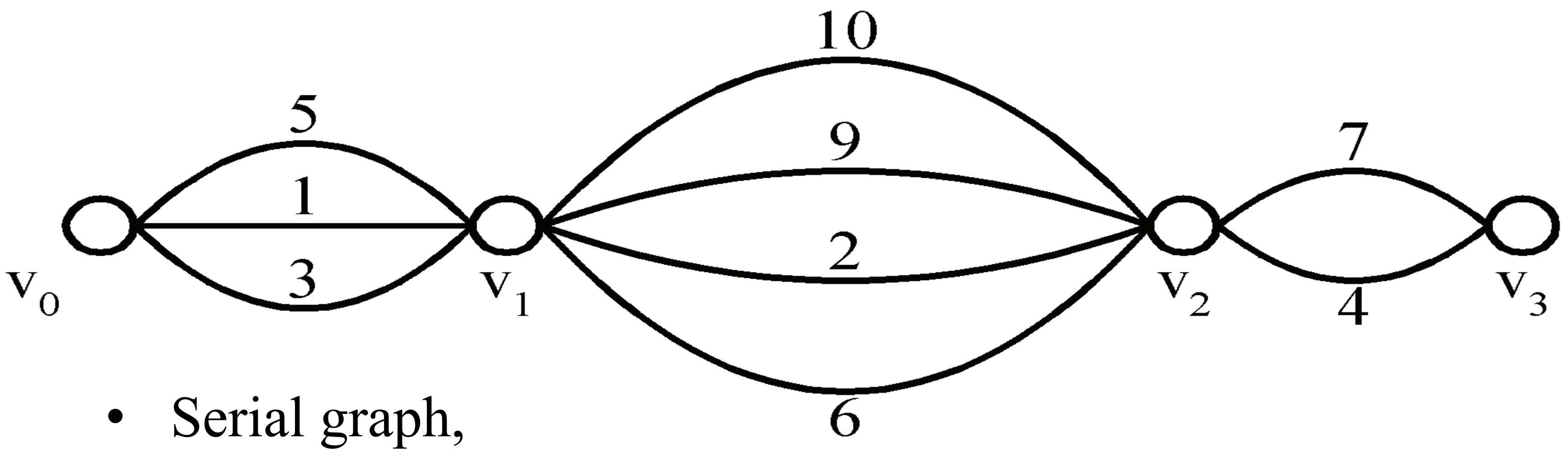
## An simple example

- **Problem:** Pick ***k*** numbers out of n numbers such that the **sum of these k numbers is the largest**.
- Brute-and-force method: $\binom{N}{k}$
- Greedy method:
- Algorithm**:**

```
FOR i = 1 to k
        pick out the largest number and
        delete this number from the input.
ENDFOR
```

## Shortest paths on a special graph

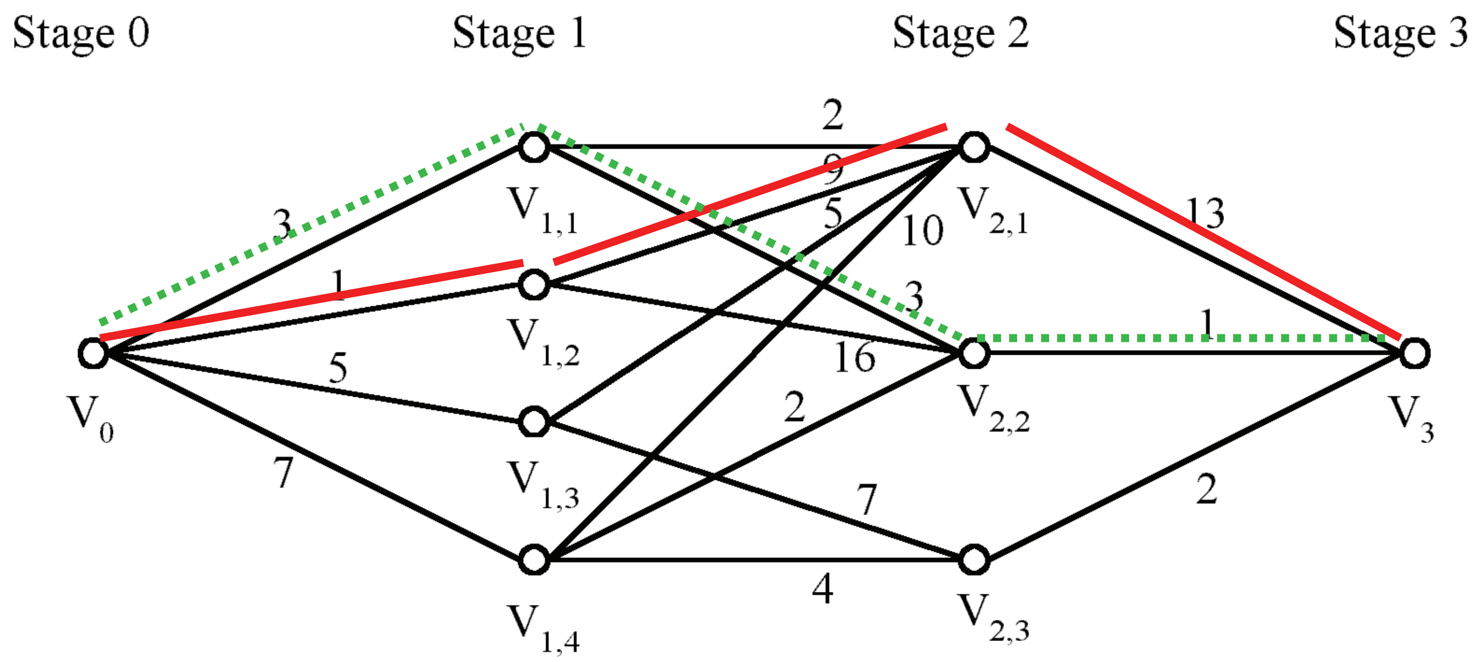
- **Problem**: Find a shortest path from $v_0$ to $v_3$.
- The greedy method can solve this problem.
- The shortest path: $1 + 2 + 4 = 7$.



- Serial graph,
- parallel graph,
- serial-parallel graph

## Shortest paths on a multi-stage graph

- **Problem**: Find a shortest path from $v_0$ to $v_3$ in the multi-stage graph.

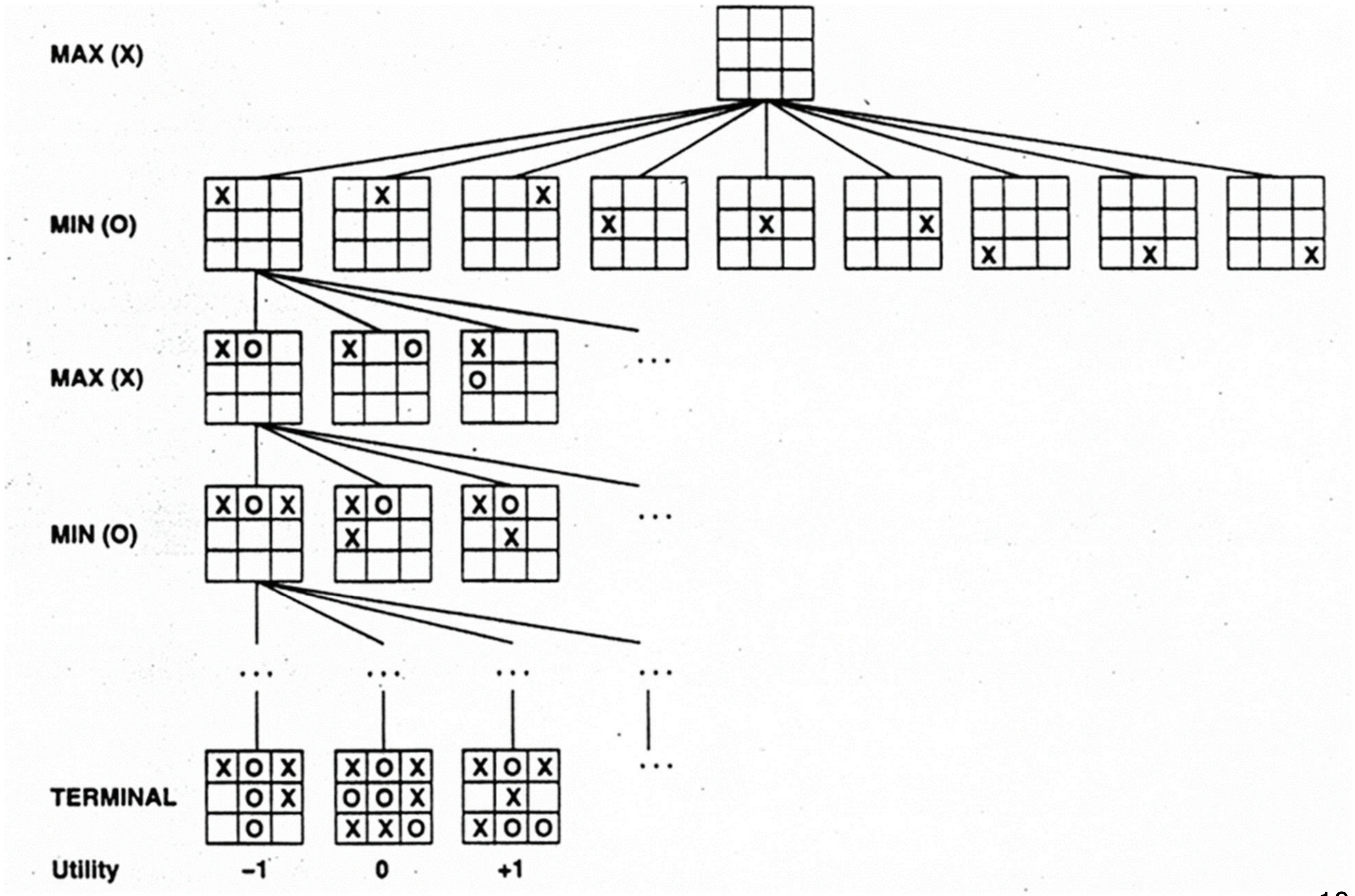


- Greedy method: $v_0 v_{1,2} v_{2,1} v_3 = 23$
- Optimal: $v_0 v_{1,1} v_{2,2} v_3 = 7$
- **The greedy method does guarantee to find an optimal solution.**

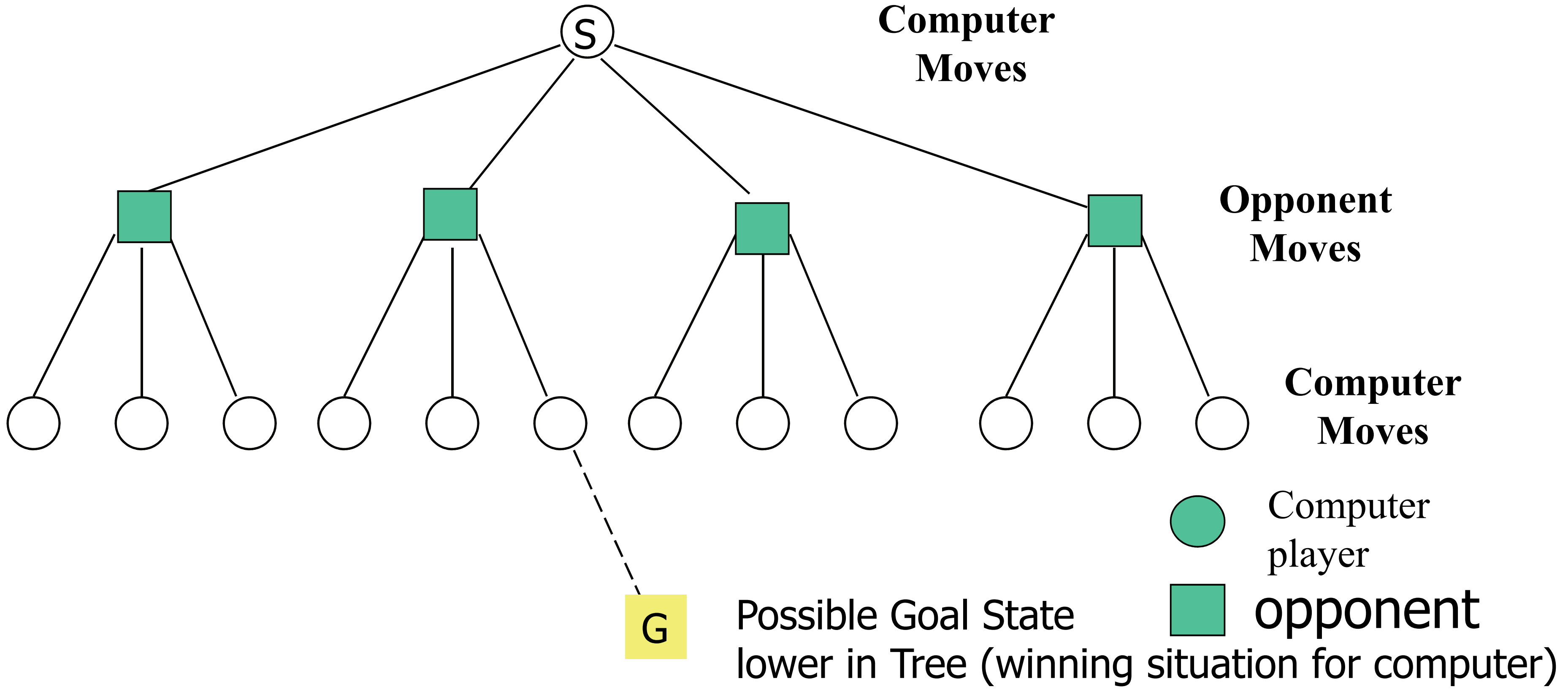
## Game Playing & problem solving

- **Look ahead**.
  - Try to figure out most of the possible moves which he can make and then imagine how his opponent may react.
  - Must understand that his opponent will also look ahead.
  - When we make decisions, we often have to look ahead.
- **The greedy method**
  - **however, never does any work of looking ahead**.
  - **Greedy may fail to produce an optimal solution**.

# Game Trees (Tree Searching method)

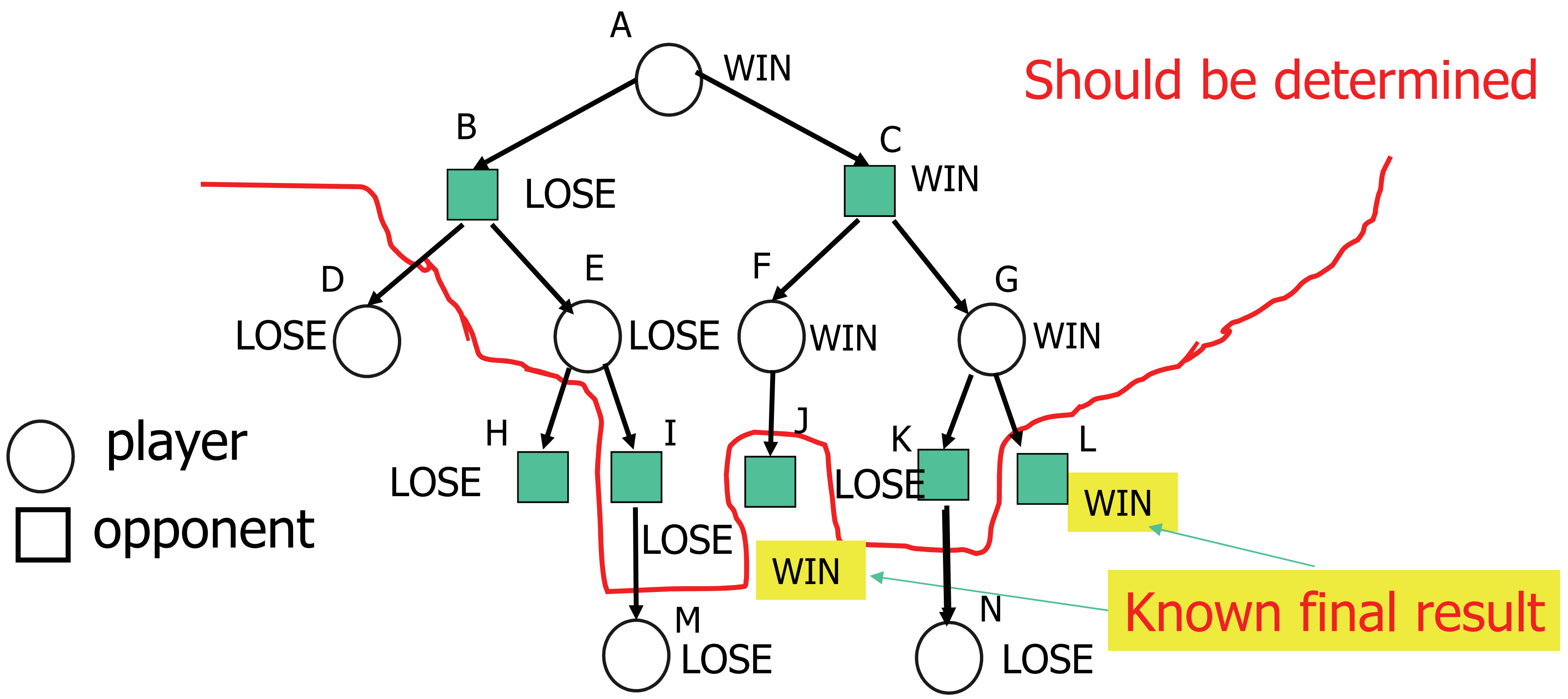


# Game Tree Representation



- New aspect to search problem
  - there's an opponent we cannot control
  - how can we handle this?

## End-Game Tree (partial searching tree)

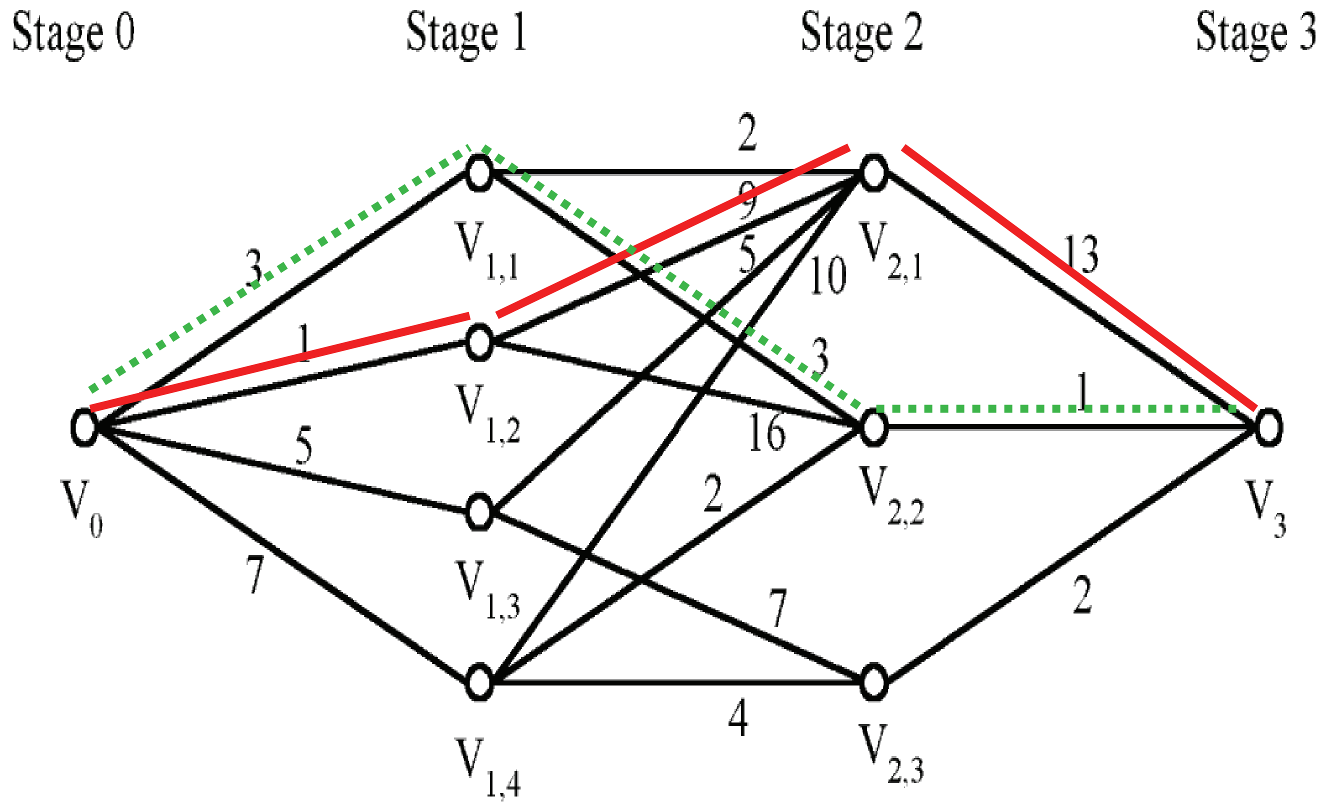


- Since the opponent always wants us to lose. we label I and K as LOSE: if any of these states is reached, we definitely will lose.
- Since we always want to win, we label F and G as WIN. Besides, we must label E as LOSE.
- Again, since the opponent always wants us to lose. we label B and C as LOSE and WIN respectively.
- Since we want to win. we label A as WIN.

# Solution of the above problem

- $d_{min}(i, j)$: minimum distance between i and j.

$d_{min}(v_0, v_3)=$
$\min\{$
$3+d_{min}(v_{11}, v_3),$
$1+d_{min}(v_{12},v_3),$
$5+d_{min}(v_{13}, v_3),$
$7+d_{min}(v_{14}, v_3)\}$



- This problem can be solved by the *dynamic programming method*.

## Kruskal’s algorithm for Minimum Spanning Trees (MST)

## 學習目標

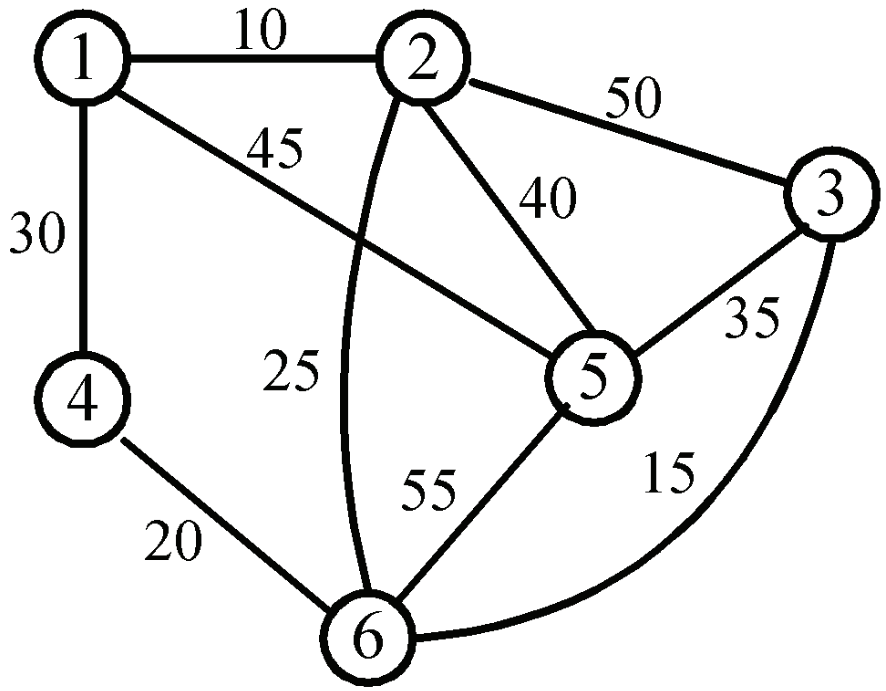
- Minimal Spanning Tree (MST) 問題定義
- Minimal Spanning Tree的演算法設計
- Kruskal’s algorithm
- Kruskal’s algorithm 時間複雜度分析
- Kruskal’s algorithm 實作方法

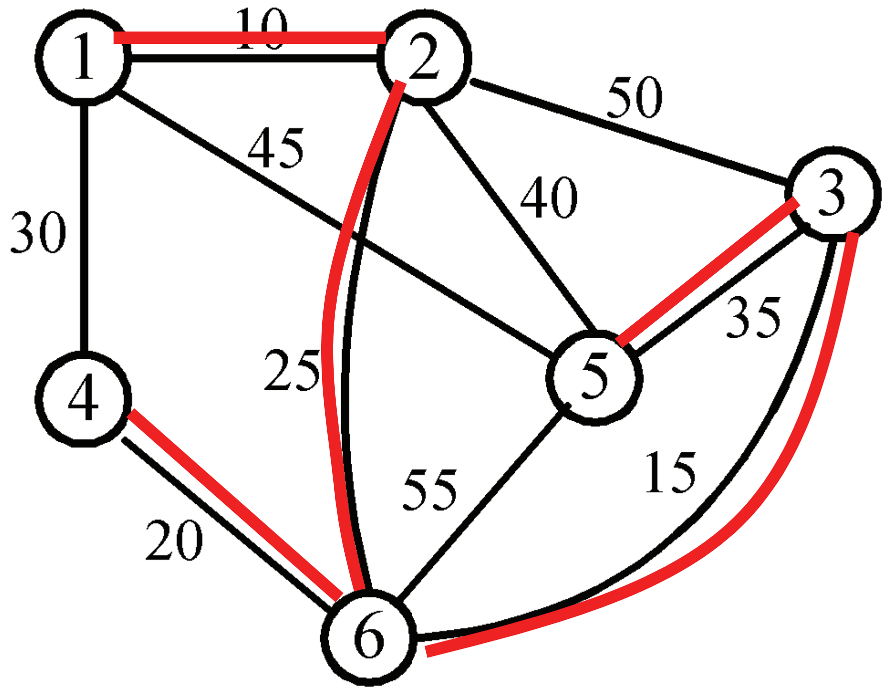
## Minimum spanning trees (MST)

- It may be defined on Euclidean space points or on a graph.
- $G = (V, E)$: weighted connected undirected graph
- Spanning tree : $S = (V, T)$, $T \subseteq E$, undirected tree
- Minimum spanning tree (MST) : a spanning tree with the smallest total weight.

# An example of MST

- A graph and one of its minimum costs spanning tree





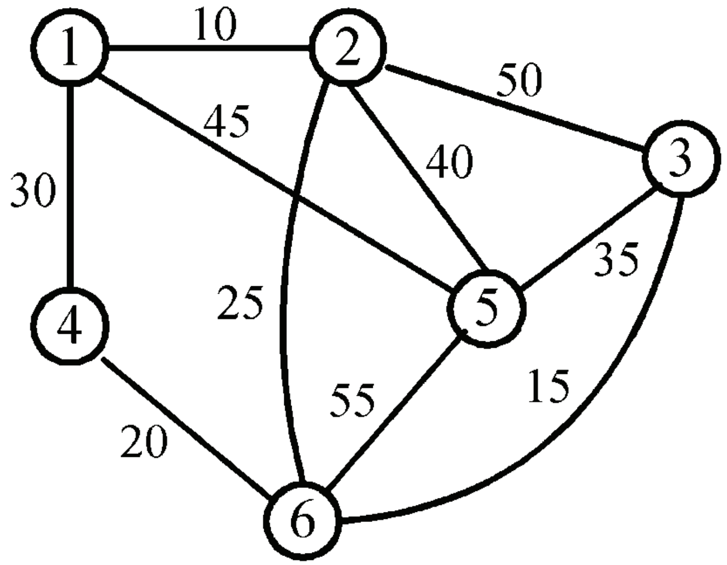
# Kruskal's algorithm for finding MST

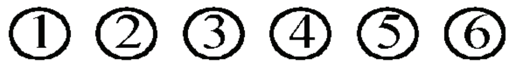
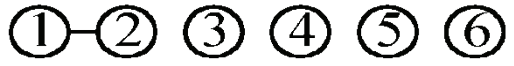
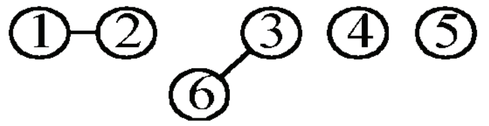
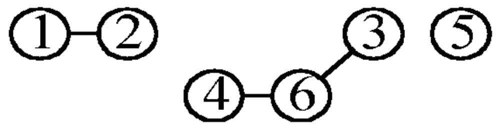
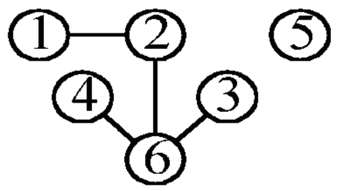
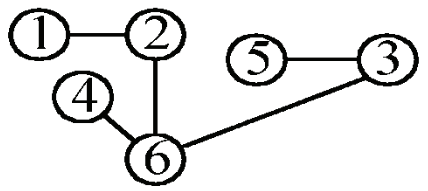
Step 1: Select the edge with the smallest weight  edge from the set of edges. (may or may not sort all edges into non-decreasing order.)

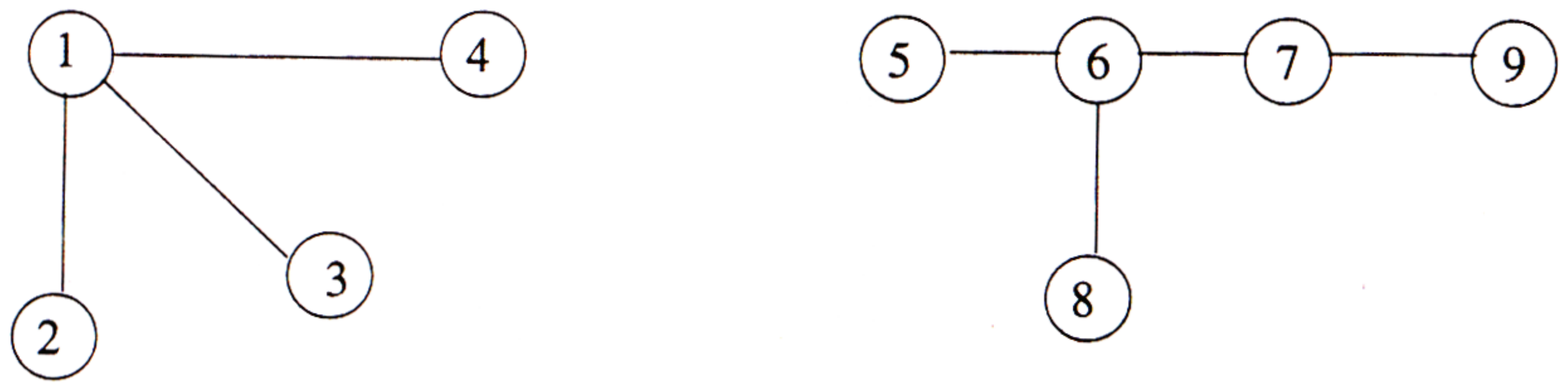
Step 2: Add the next smallest weight edge to the forest if it will not cause a cycle. Discard the selected edge if otherwise.

Step 3: Stop if n-1 edges. Otherwise, go to Step 2.

# An example of Kruskal’s algorithm



| Edge | Cost | Spanning Forest |
|---|---|---|
| | |  |
| (1,2) | 10 |  |
| (3,6) | 15 |  |
| (4,6) | 20 |  |
| (2,6) | 25 |  |
| (1,4) | 30 | (reject) |
| (3,5) | 35 |  |



A spanning forest

- If (3,4) is added, since 3 and 4 belong to the same set, a cycle will be formed.
- If (4,5) is added, since 4 and 5 belong to different sets, a new forest (1,2,3,4,5,6,7,8,9) is created.
- Thus we are always performing the union of two sets.

## The details for constructing MST

- Use heap to maintain the smallest weight edge. (no need sorting)
- How do we check if a cycle is formed when a new edge is added?
  - By the SET and UNION method.
  - A tree in the forest is used to represent a SET.

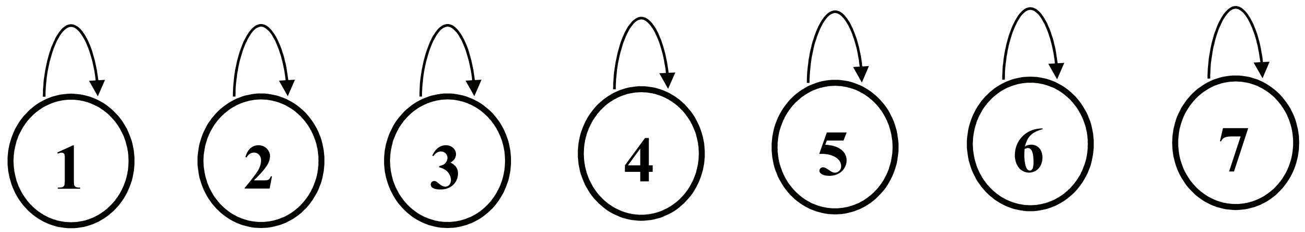
- Tree implementation for quick union
  - Each set a tree:  Root serves as SetName
  - To **Find**, follow parent pointers to the root
  - Initially parent pointers set to self
  - To **union(u,v)**, make v's parent point to u

## Union and Find Operations

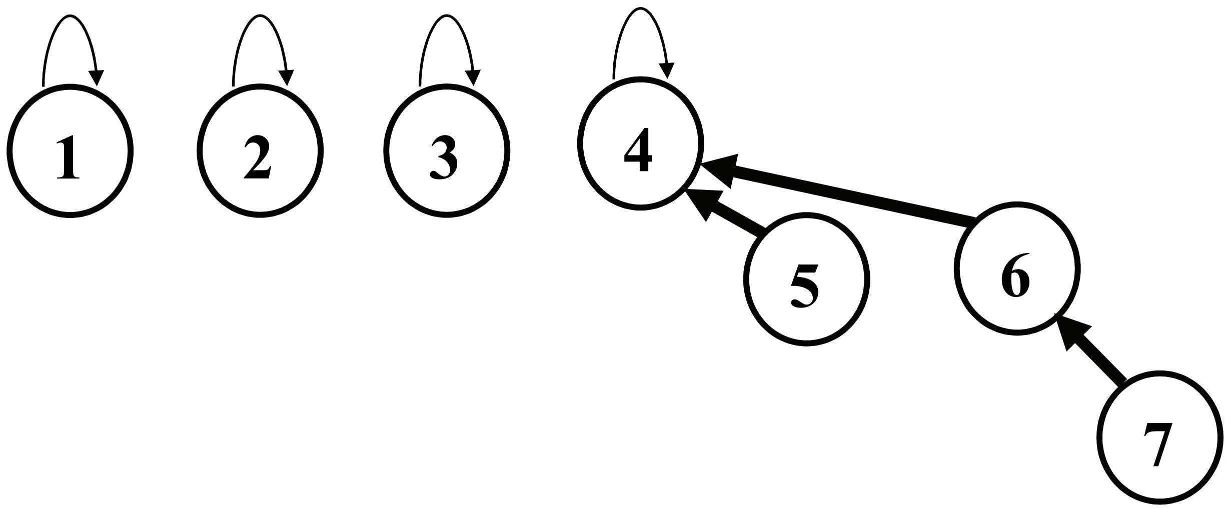
- Union operation:
  - If $(u, v) \in E$ and u, v are in the same set, then the addition of (u, v) will form a cycle.
  - If $(u, v) \in E$ and $u \in S_1$ , $v \in S_2$ , then perform **UNION** of $S_1$ and $S_2$.
  - replaces both sets with a new set
  - the name of the new set is not specified

- Find operation:
  - returns the name of the unique set that contains the given element
  - To ***find*** an element in a set.
  - Chapter 10 (amortized analysis) will show that both **union and find** operations will take **O(m)** steps.

# Union Example



After union(4,5), union(6,7), union(4,6)



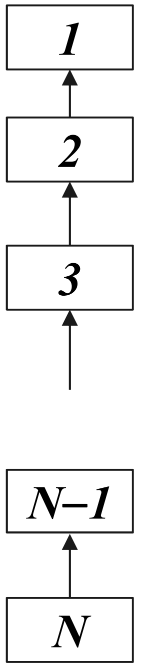
## Analysis of Quick Union

```
Initialize(int N)
  parent = new int [N+1];
  for (int e=1; e<=N; e++)
    parent[e] = 0;

int Find(int e)
  while (parent[e] != 0)
    e = parent[e];
  return e;

Union(int i, int j)
  parent[j] = i;
```

```
Union(N-1, N);
Union(N-2, N-1);
Union(N-3, N-2);
  …
Union(1, 2);
Find(1);
Find(2);
…
Find(N);
```



- Complexity in the worst case:
  - **Union is $O(1)$ but Find is $O(n)$**
  - $u$ Union, $f$ Find : $O(u + f\,n)$
  - $N$-$1$ Unions and $O(N)$ Finds: still $O(N^2)$ total time

## Time complexity Kruskal's algorithm

- Time complexity: O(|E| log|E|)
  - Step 1: O(|E| log|E|)
  - Step 2 & Step 3: $O(|E|\alpha(|E|,|V|))$

    Where α is the inverse of Ackermann's function.

## Question:

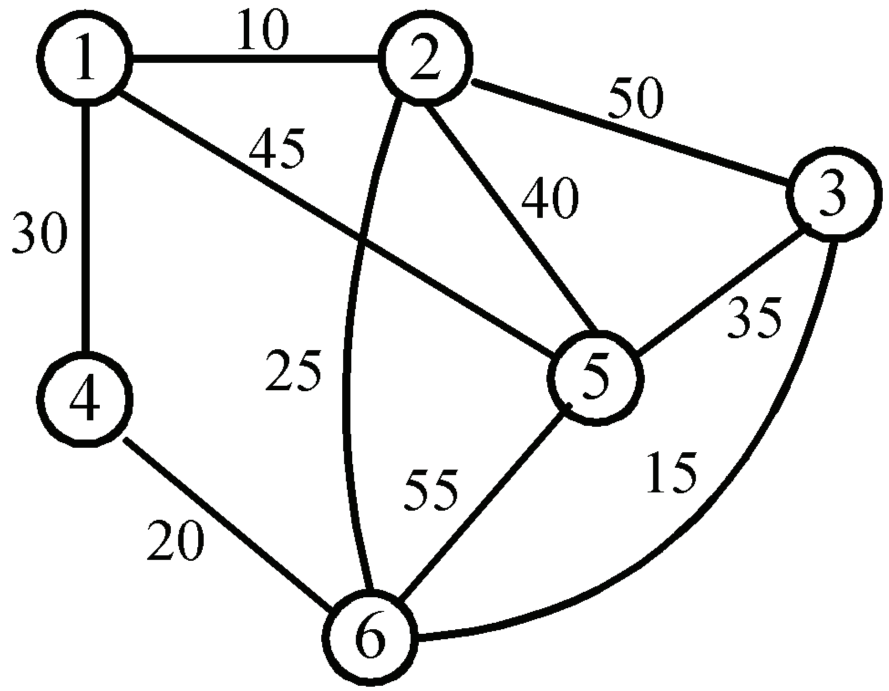
- Which is the total weight of the MST of the graph?

(1) 100

(2) 105

(3) 110

(4) 115



Ans. 2

## Disjoint set 資料結構

一個維護所有 **disjoint dynamic sets** 組成的大集合 $S=\{S_1, S_2, \ldots, S_k\}$ 的資料結構。

1. 每個集合都被一個 **representative** 所代表，而 **representative** 是該集合中的某一個元素。

3. 此資料結構支援以下的指令：
   - **Make-Set(*x*)**: 創造一個新的集合 $\{x\}$
   - **Union(*x*, *y*)**: 把兩個分別包含 $x, y$ 的集合聯集起來
   - **Find-Set(*x*)**: 傳回一個指標指向包含 $x$ 的集合的 representative。

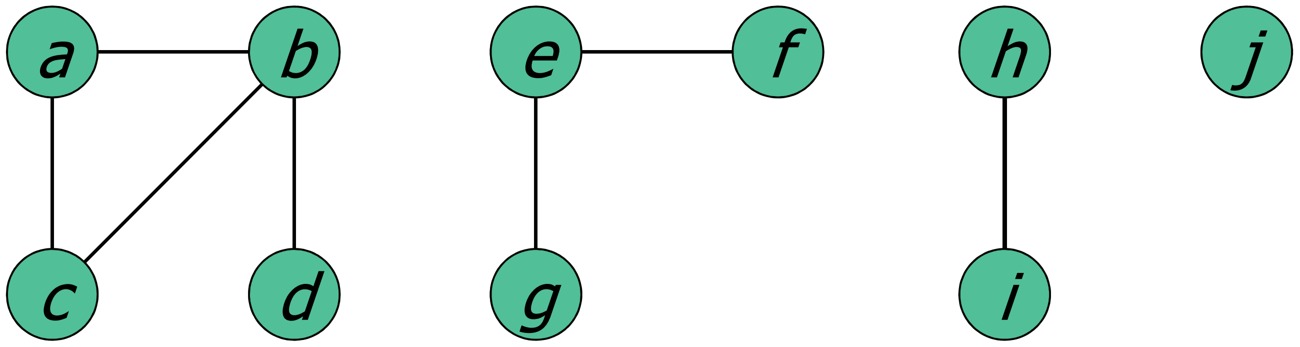
**符號：**

*n*: 所有 Make-Set 指令的總數。

*m*: 所有 Make-Set、Union 及 Find-Set 指令的總數

**註：** $m \geq n$ 且所有 Union 指令的總數最多不會超過 $n$-1。

# disjoint-set 資料結構例子



**(a)**

Union

| Edge processed | Collection of disjoint sets | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| initial sets | $\{a\}$ | $\{b\}$ | $\{c\}$ | $\{d\}$ | $\{e\}$ | $\{f\}$ | $\{g\}$ | $\{h\}$ | $\{i\}$ | $\{j\}$ |
| $(b,d)$ | $\{a\}$ | $\{b,d\}$ | $\{c\}$ | | $\{e\}$ | $\{f\}$ | $\{g\}$ | $\{h\}$ | $\{i\}$ | $\{j\}$ |
| $(e,g)$ | $\{a\}$ | $\{b,d\}$ | $\{c\}$ | | $\{e,g\}$ | $\{f\}$ | | $\{h\}$ | $\{i\}$ | $\{j\}$ |
| $(a,c)$ | $\{a,c\}$ | $\{b,d\}$ | | | $\{e,g\}$ | $\{f\}$ | | $\{h\}$ | $\{i\}$ | $\{j\}$ |
| $(h,i)$ | $\{a,c\}$ | $\{b,d\}$ | | | $\{e,g\}$ | $\{f\}$ | | $\{h,i\}$ | | $\{j\}$ |
| $(a,b)$ | $\{a,b,c,d\}$ | | | | $\{e,g\}$ | $\{f\}$ | | $\{h,i\}$ | | $\{j\}$ |
| $(e,f)$ | $\{a,b,c,d\}$ | | | | $\{e,f,g\}$ | | | $\{h,i\}$ | | $\{j\}$ |
| $(b,c)$ | $\{a,b,c,d\}$ | | | | $\{e,f,g\}$ | | | $\{h,i\}$ | | $\{j\}$ |

Generate cycle

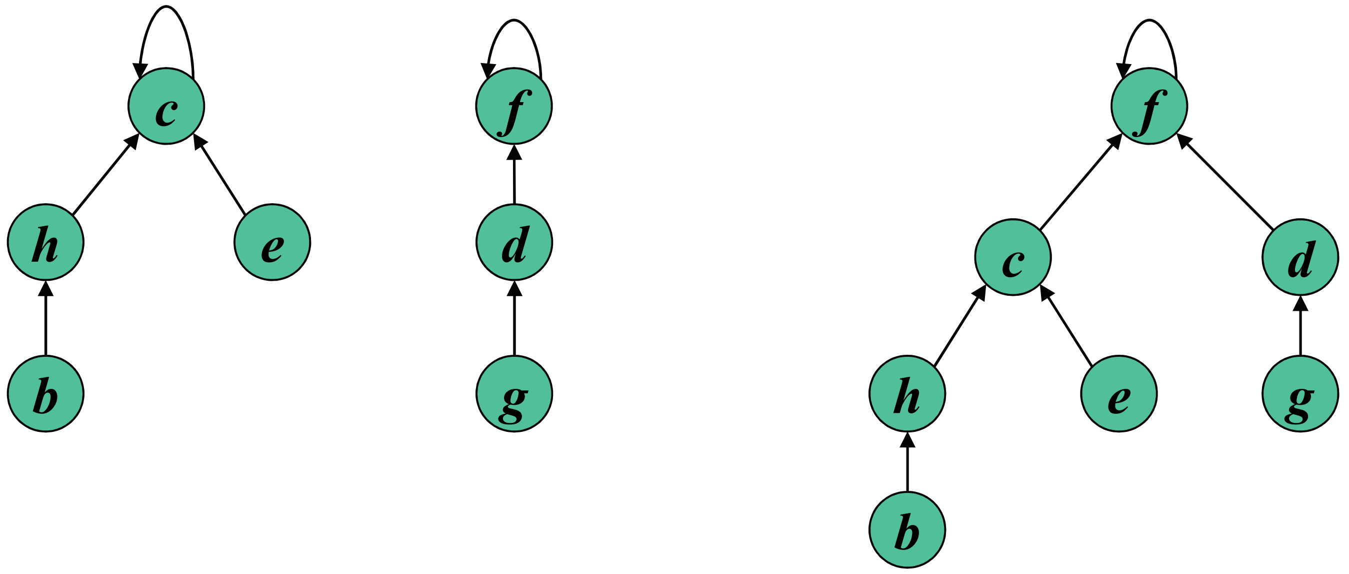
(b)

- $m = 2n\text{-}1$ 個指令耗時 $O(n^2)$

| **Operation** | **Number of objects updated** | |
|---|---|---|
| **MAKE-SET($x_1$)** | **1** | |
| **MAKE-SET($x_2$)** | **1** | |
| ⋮ | ⋮ | $O(n)$ |
| **MAKE-SET($x_n$)** | **1** | |
| **UNION($x_1$, $x_2$)** | **1** | |
| **UNION($x_2$, $x_3$)** | **2** | |
| **UNION($x_3$, $x_4$)** | **3** | $O(1+2+\ldots+n)$ |
| ⋮ | ⋮ | $=O(n^2)$ |
| **UNION($x_{n-1}$, $x_n$)** | **$n$-1** | |

- 每個指令的 amortized time 為 $O(n)$。

## Disjoint-set forests



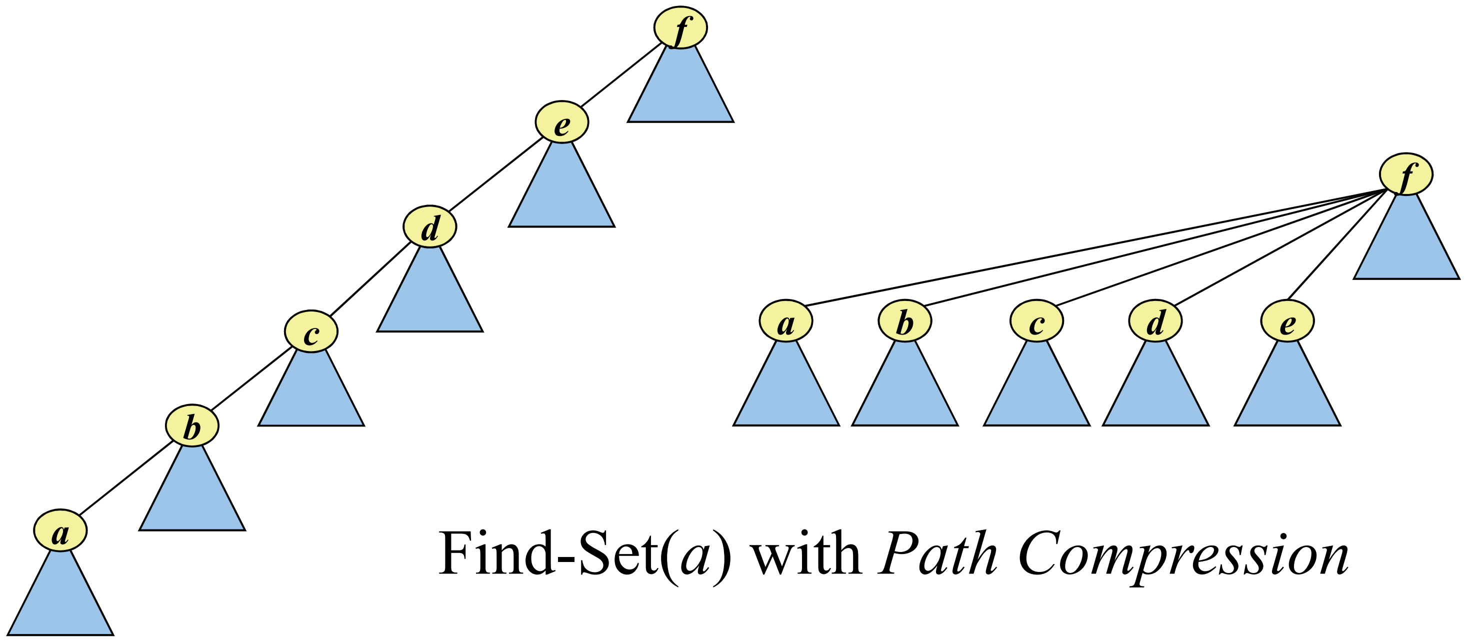
- Tree 的 root 為 representative。
- Make-Set($x$): 耗時 $O(1)$
- Find-Set($x$): 耗時 $O(h)$，$h$ 為包含 $x$ 的 tree 的高度。
  (找 $x \rightarrow$ root 的路徑)
- Union($x, y$): $x$ 的 root 指向 $y$ 的 root。
  $\rightarrow$ 耗時 $O(h)$。

## 一些增進執行速度的技巧

1. ***Union by rank (or size)***: 由較小的 tree 的 root 指向較大的 tree 的 root (依照 tree 的高度區分)。

   ***rank***[*x*]: *x* 的高度(由 *x* 到一 descendant leaf 的最長路徑的邊的數量)

2. ***Path compression***: 在執行 Find-Set(*x*) 對過程中，把所有 find path 上的點指向 root。(不改變任何 rank 的值)



Find-Set(*a*) with *Path Compression*

## Effect of the heuristics on the running time

1. 若只使用了 Union-by-rank， 簡單便可證明總共耗時 $O(m\lg n)$。
2. 若只使用了 Path-compression 可以證明 (在此不證) 總耗時為

   $$\Theta(n + f\cdot(1+\log_{2+f/n} n))，$$

   其中 $n$ 是 Make-Set 指令的數量，且 $f$ 是 Find-Set 指令的數量。
3. 當兩者皆用上時，在最壞的情況下共需耗時 $O(m\alpha(m,n))$。

**註：** $\alpha(m,n)$ 是 Ackermann’s function 的反函數，其成長速率非常慢。在所有可能的應用中，$\alpha(m,n)\le 4$，因此在所有實際的應用上，我們可以把執行時間視為和 $m$ 成正比。

## Ackermann's function and its inverse

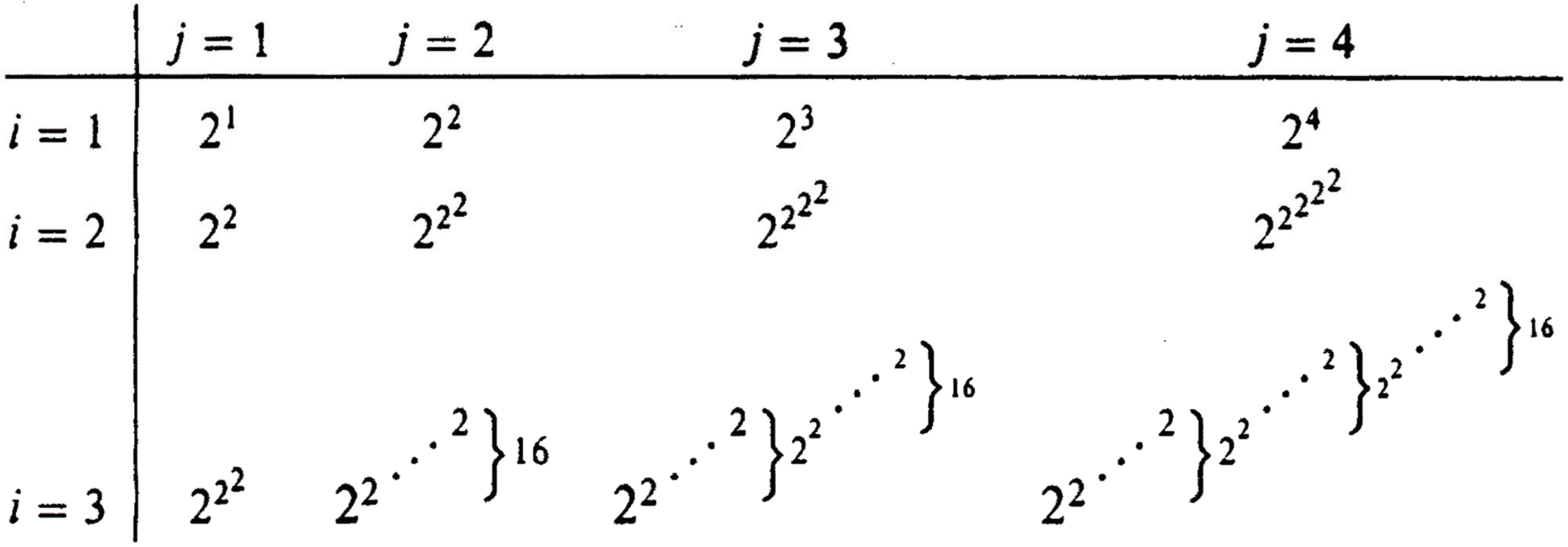
- 令 $g(i) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\Big\}\, i$ 為 repeated exponentiation。
  (即 $g(4) = 2^{2^{2^{2}}}$)

- 函數 **lg* *n*** = min{$i \geq 0$: $\lg^{(i)} n \leq 1$} 本質上為 $g(i)$ 的反函數 (即 $\lg^* 2^{2^{2^{2^{2}}}} = 5$)

註： $\lg^{*} g(i) = i+1$ 。

- The Ackermann's function: 對整數 $i, j \geq 1$，

$$
\begin{array}{ll}
A(1, j) = 2^j & \text{for } j \geq 1 \\
A(i, 1) = A(i\text{-}1, 2) & \text{for } i \geq 2 \\
A(i, j) = A(i\text{-}1, A(i, j\text{-}1)) & \text{for } i, j \geq 2
\end{array}
$$



| | $j = 1$ | $j = 2$ | $j = 3$ | $j = 4$ |
|---|---|---|---|---|
| $i = 1$ | $2^1$ | $2^2$ | $2^3$ | $2^4$ |
| $i = 2$ | $2^2$ | $2^{2^2}$ | $2^{2^{2^2}}$ | $2^{2^{2^{2^2}}}$ |
| $i = 3$ | $2^{2^2}$ | $\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}16$ | $\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}16$ | $\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}16$ |

- 因對所有的 $j \geq 1$，$A(2, j) = 2^{2^{\cdot^{\cdot^{2}}}}\Big\}j = g(j)$ 所以當 $i \geq 2$，$A(i, j) \geq g(j)$。

- Ackermann's function 的反函數:

  $\alpha(m, n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) > \lg n\}$。

- $A(4, 1) = A(3, 2) = g(16) \approx 10^{80}$

- 由於 lg $n < 10^{80}$，且 $A(4, \lfloor m/n \rfloor) \geq A(4,\ 1) \approx 10^{80}$，我們得知在實際應用上，$\alpha(m,\ n) \leq 4$。

- 在實際應用上，$\lg^* n \leq 5$ ($n \leq 2^{65536}$)。

- 由於當 $i \geq 2$，$A(i,\ j) \geq g(j)$，因此 $\alpha(m,\ n) = O(\lg^* n)$。

註：

$$\xrightarrow{\text{increasing}}$$

$1,\quad \alpha,\quad \lg^*,\quad \lg\lg,\quad \lg,\quad n^{1/2},\quad n,\quad n^k,\quad 2^n,\quad g,\quad A$

## Ackermann's function

$$A(p, q) = \begin{cases} 2q, & p = 0 \\ 0,\ q = 0, & p \geq 1 \\ 2,\ p \geq 1, & q = 1 \\ A(p-1, A(p, q-1)), & p \geq 1,\ q \geq 2 \end{cases}$$

$\Rightarrow$ **$A(p, q+1) > A(p, q)$, $A(p+1, q) > A(p, q)$**

$$A(3,4) = \left. 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}} \right\} \text{65536 two's}$$

# Expansion

$$\begin{aligned}
A(1,2) &= A(0, A(1,1)) \\
&= A(0, A(0, A(1,0))) \\
&= A(0, A(0, A(0,1))) \\
&= A(0, A(0,2)) \\
&= A(0,3) \\
&= 4
\end{aligned}$$

To demonstrate how $A(4,3)$'s computation results many steps and in a large number:

$$\begin{aligned}
A(4,3) &= A(3, A(4,2)) \\
&= A(3, A(3, A(4,1))) \\
&= A(3, A(3, A(3, A(4,0)))) \\
&= A(3, A(3, A(3, A(3,1)))) \\
&= A(3, A(3, A(3, A(2, A(3,0))))) \\
&= A(3, A(3, A(3, A(2, A(2,1))))) \\
&= A(3, A(3, A(3, A(2, A(1, A(2,0)))))) \\
&= A(3, A(3, A(3, A(2, A(1, A(1,1)))))) \\
&= A(3, A(3, A(3, A(2, A(1, A(0, A(1,0))))))) \\
&= A(3, A(3, A(3, A(2, A(1, A(0, A(0,1)))))))
\end{aligned}$$

# Ackermann function

Values of $A(m, n)$

| $m \backslash n$ | 0 | 1 | 2 | 3 | 4 | $n$ |
|---|---|---|---|---|---|---|
| **0** | 1 | 2 | 3 | 4 | 5 | $n+1$ |
| **1** | 2 | 3 | 4 | 5 | 6 | $n+2=2+(n+3)-3$ |
| **2** | 3 | 5 | 7 | 9 | 11 | $2n+3=2\cdot(n+3)-3$ |
| **3** | 5 | 13 | 29 | 61 | 125 | $2^{(n+3)}-3$ |
| **4** | 13 $=2^{2^{2}}-3$ | 65533 $=2^{2^{2^{2}}}-3$ | $2^{65536}-3$ $=2^{2^{2^{2^{2}}}}-3$ | $2^{2^{65536}}-3$ $=2^{2^{2^{2^{2^{2}}}}}-3$ | $2^{2^{2^{65536}}}-3$ $=2^{2^{2^{2^{2^{2^{2}}}}}}-3$ | $\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3}-3$ |
| **5** | 65533 $=2\uparrow\uparrow\uparrow 3-3$ | $2\uparrow\uparrow\uparrow 4-3$ | $2\uparrow\uparrow\uparrow 5-3$ | $2\uparrow\uparrow\uparrow 6-3$ | $2\uparrow\uparrow\uparrow 7-3$ | $2\uparrow\uparrow\uparrow(n+3)-3$ |
| **6** | $2\uparrow\uparrow\uparrow\uparrow 3-3$ | $2\uparrow\uparrow\uparrow\uparrow 4-3$ | $2\uparrow\uparrow\uparrow\uparrow 5-3$ | $2\uparrow\uparrow\uparrow\uparrow 6-3$ | $2\uparrow\uparrow\uparrow\uparrow 7-3$ | $2\uparrow\uparrow\uparrow\uparrow(n+3)-3$ |

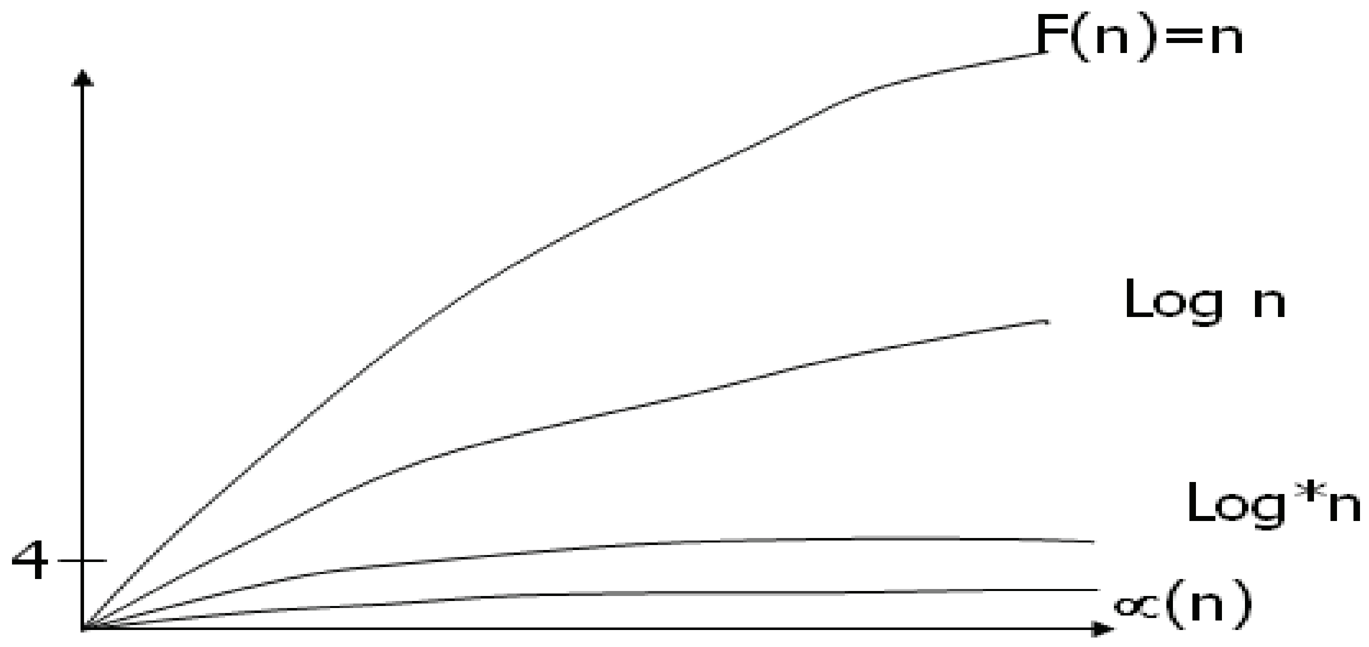
## Inverse of Ackermann's function

- $\alpha(m, n) = \min\{Z \geq 1 \mid A(Z, 4\lceil m/n \rceil) > \log_2 n\}$

Practically, $A(3,4) > \log_2 n$

$\Rightarrow \alpha(m, n) \leq 3$

$\Rightarrow \alpha(m, n)$ is almost a constant.



## Question:

- Which is the value of the function A(1,2)?

$$A(p, q) = \begin{cases} 2q, & p = 0 \\ 0,\ q = 0, & p \geq 1 \\ 2,\ p \geq 1, & q = 1 \\ A(p\text{-}1, A(p, q\text{-}1)), & p \geq 1,\ q \geq 2 \end{cases}$$

(1) 2

(2) 4

(3) 16

(4) 1.

Ans. 2

## Prime’s MST algorithm

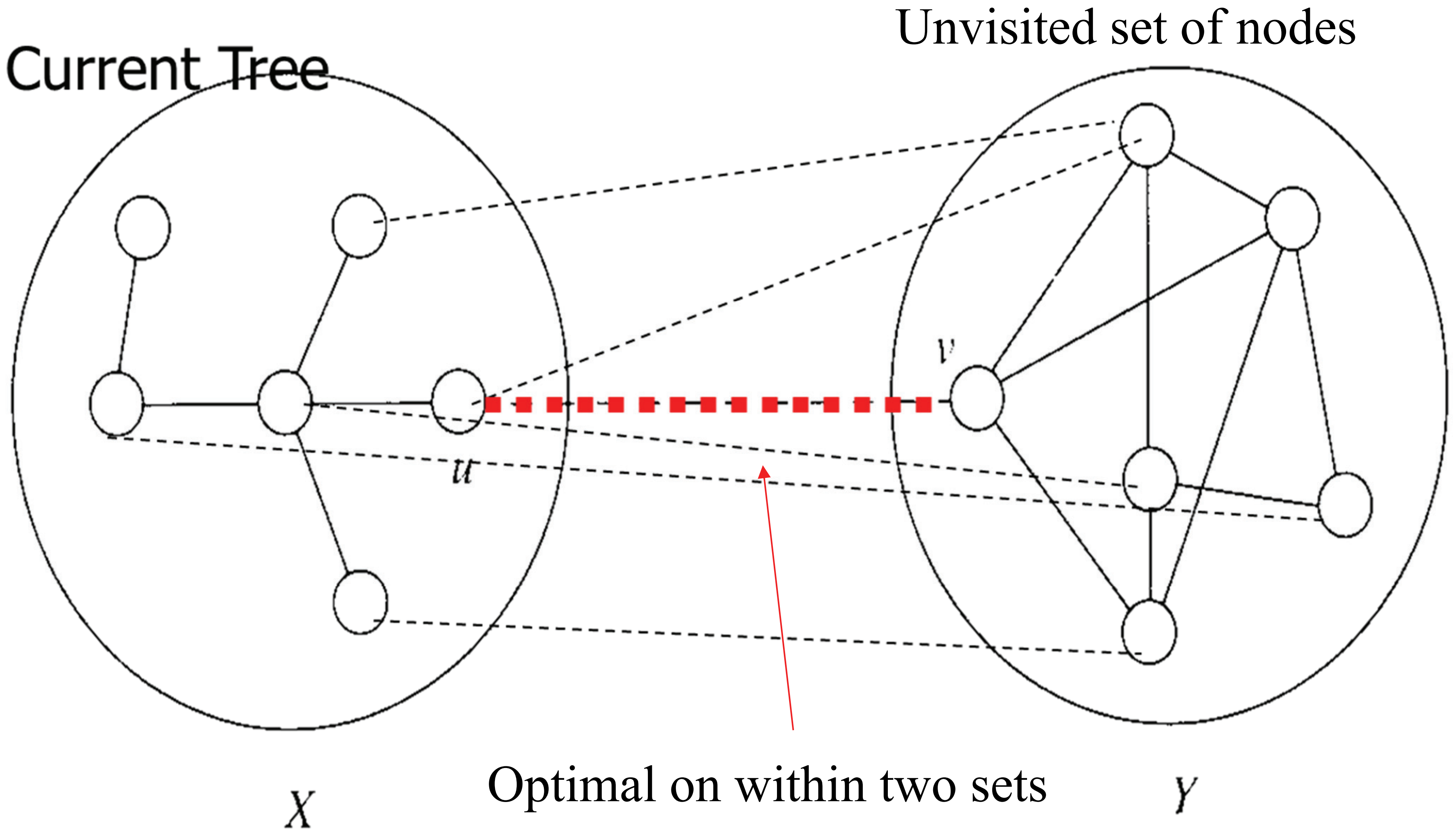
## 學習目標

- Minimal Spanning Tree 問題定義
- Minimal Spanning Tree的演算法設計
- Prime’s algorithm
- Prime’s algorithm 時間複雜度分析

## Prime's MST algorithm

- Let **X** denote the set of vertices contained in the partially constructed minimal spanning tree.
- Let **Y=V-X**.
- The next edge (u, v) to be added is an edge between X and Y (u∈X and v∈Y) with the **smallest weight**.
- V is added to X.
- **Can start with any vertex**.

# Idea of MST


Unvisited set of nodes
Current Tree
v
u
Optimal on within two sets
X
Y

## Prim’s algorithm for finding MST

Step 1: $x \in V$, Let $A = \{x\}$, $B = V - \{x\}$.

Step 2: Select $(u, v) \in E$, $u \in A$, $v \in B$ such that $(u, v)$ has the smallest weight between A and B.

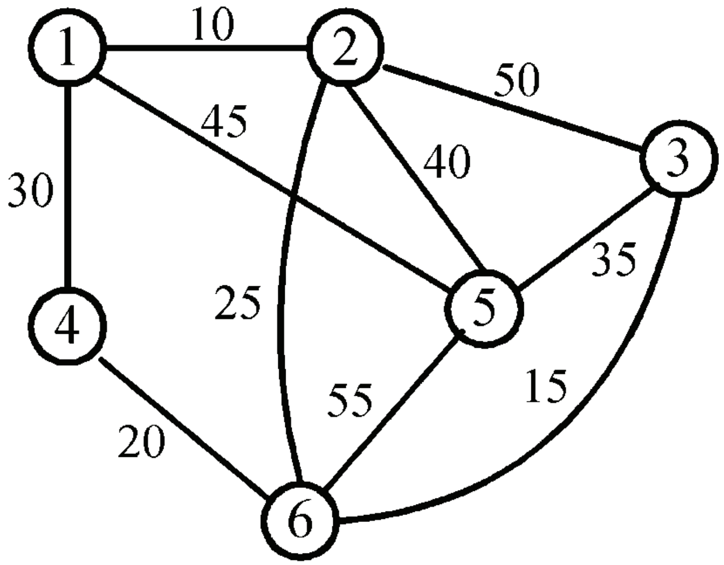
Step 3: Put $(u, v)$ in the tree. $A = A \cup \{v\}$, $B = B - \{v\}$

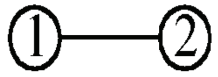
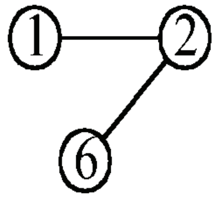
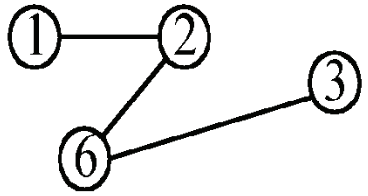
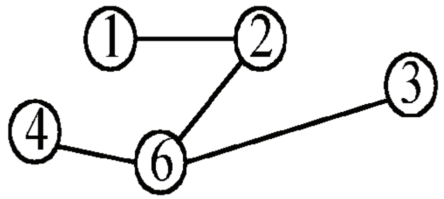
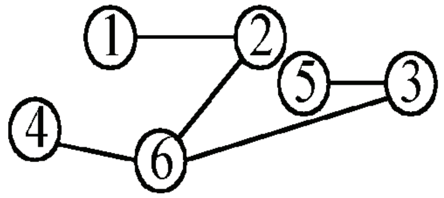
Step 4: If $B = \varnothing$, stop; otherwise, go to Step 2.

- Time complexity : $O(n^2)$, $n = |V|$.

No test the cycle

# An example for Prim's algorithm



| Edge | Cost | Spanning tree |
|---|---|---|
| (1,2) | 10 |  |
| (2,6) | 25 |  |
| (3,6) | 15 |  |
| (6,4) | 20 |  |
| (3,5) | 35 |  |

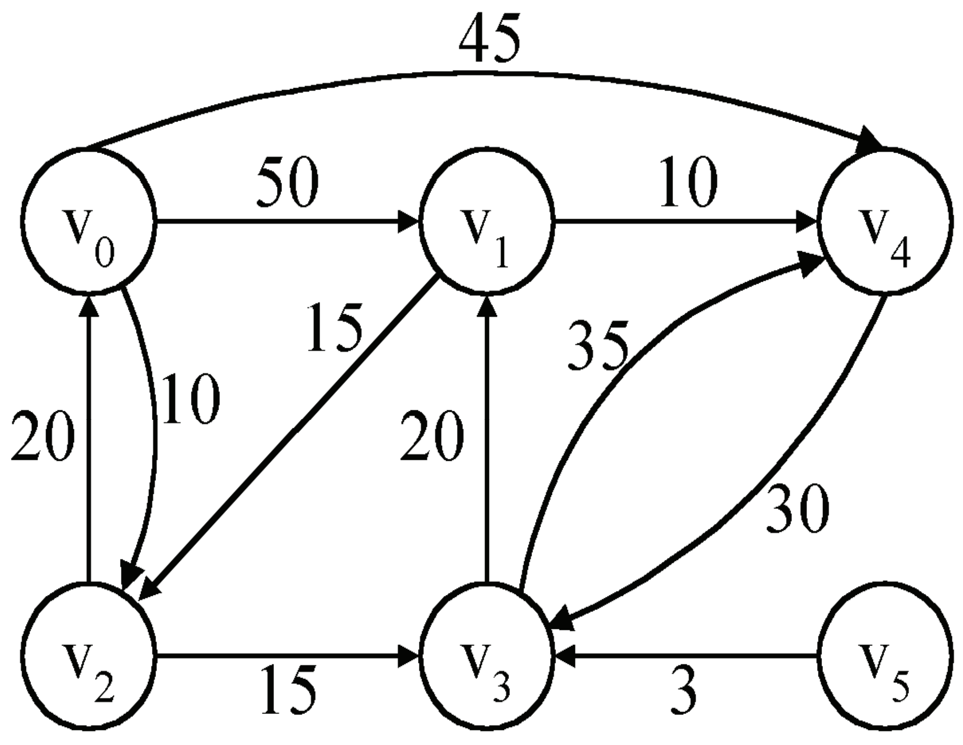
## The single-source shortest path problem

## 學習目標

- Single-shortest path 問題定義
- Single-shortest path的演算法設計
- Single-shortest path時間複雜度分析

# The single-source shortest path problem

- shortest paths from ***$v_0$*** to all destinations

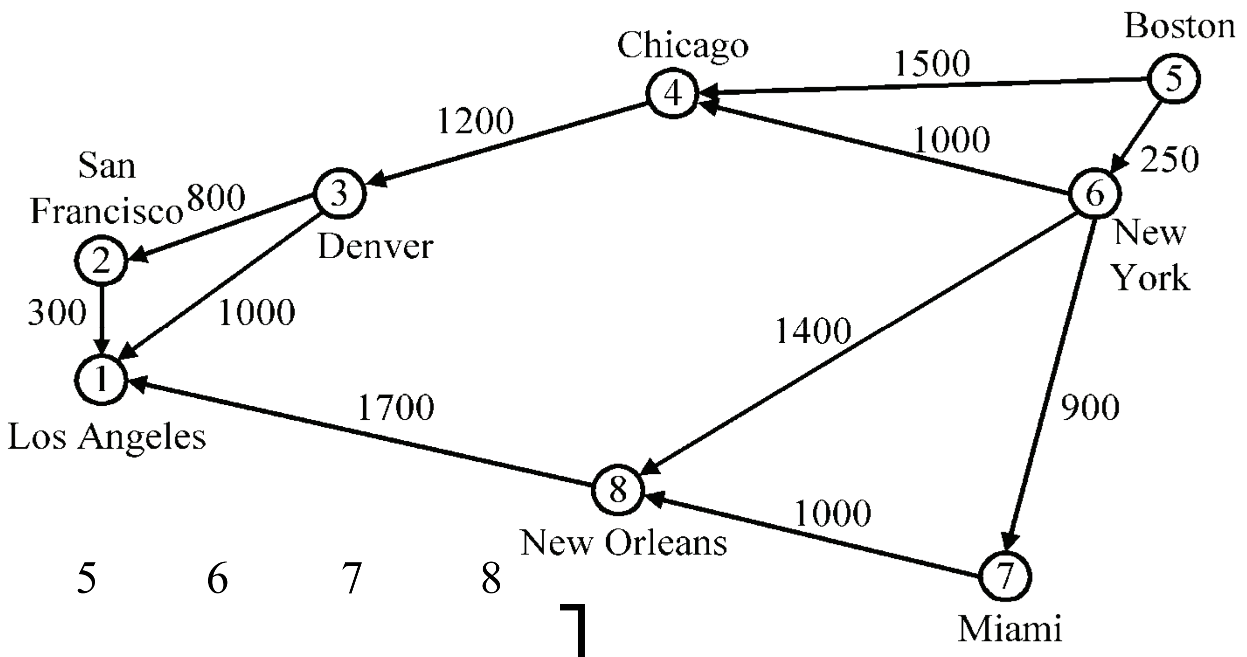


(a)

|  | Path | Length |
|---|---|---|
| 1) | $v_0v_2$ | 10 |
| 2) | $v_0v_2v_3$ | 25 |
| 3) | $v_0v_2v_3v_1$ | 45 |
| 4) | $v_0v_4$ | 45 |

(b)

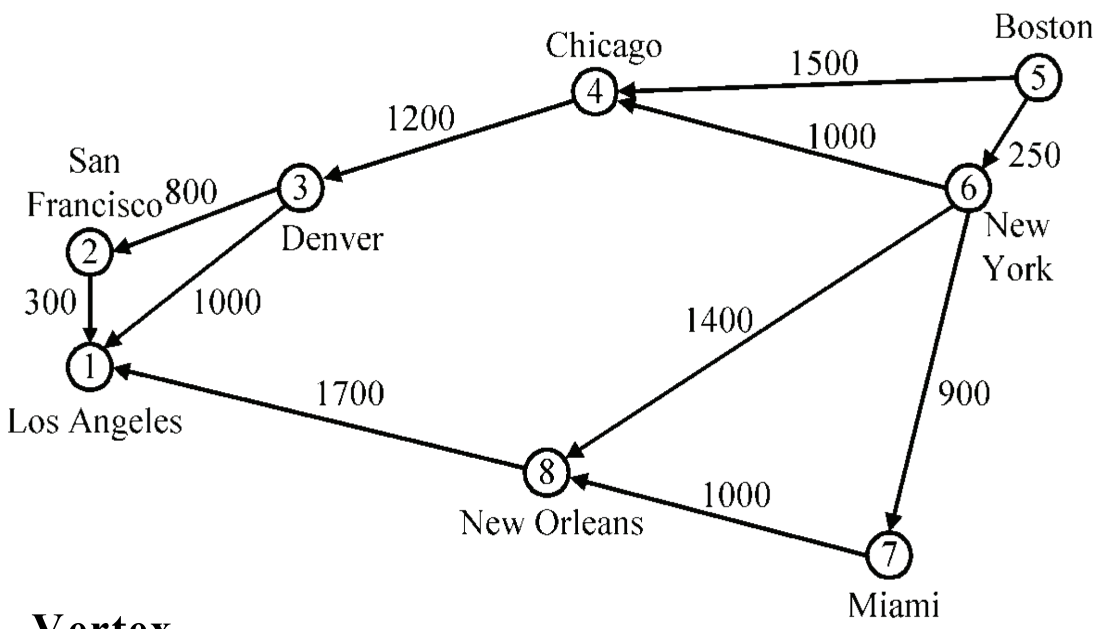
# Dijkstra's algorithm



Cost adjacency matrix. All entries not shown are $+\infty$.

$$\begin{array}{c|cccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 0 & & & & & & & \\
2 & 300 & 0 & & & & & & \\
3 & 1000 & 800 & 0 & & & & & \\
4 & & & 1200 & 0 & & & & \\
5 & & & & 1500 & 0 & 250 & & \\
6 & & & & 1000 & & 0 & 900 & 1400 \\
7 & & & & & & & 0 & 1000 \\
8 & 1700 & & & & & & & 0
\end{array}$$

```
Procedure SHORTEST_PATH (v, COST[1..n,1..n], DIST[1..n], n)
Declare S (1 : n ) //=1 visited, =0 unvisited
for j ← 1 to n
   S( j )← 0 ; DIST( j )←COST( v , j )
End
S( v )←1 ; DIST( v )←0 ; num←2
While (num ≤ n)  do
      Choose u : DIST(u) = min{DIST(w)}
      S(u)←1 ; num←num+1
      For all w with S(w) == 0  do
         DIST(w) ←min {DIST(w), DIST(u) + COST(u , w ) }
      end
end
end SHORTEST_PATH
```



| Iteration | S | Vertex Selected | (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) |
|---|---|---|---|---|---|---|---|---|---|---|
| Initial |  | ---- |  |  |  |  |  |  |  |  |
| 1 | 5 | 6 | +∞ | +∞ | +∞ | 1500 | 0 | 250 | +∞ | +∞ |
| 2 | 5,6 | 7 | +∞ | +∞ | +∞ | 1250 | 0 | 250 | 1150 | 1650 |
| 3 | 5,6,7 | 4 | +∞ | +∞ | +∞ | 1250 | 0 | 250 | 1150 | 1650 |
| 4 | 5,6,7,4 | 8 | +∞ | +∞ | 2450 | 1250 | 0 | 250 | 1150 | 1650 |
| 5 | 5,6,7,4,8 | 3 | 3350 | +∞ | 2450 | 1250 | 0 | 250 | 1150 | 1650 |
| 6 | 5,6,7,4,8,3 | 2 | 3350 | 3250 | 2450 | 1250 | 0 | 250 | 1150 | 1650 |
|  | 5,6,7,4,8,3,2 |  | 3350 | 3250 | 2450 | 1250 | 0 | 250 | 1150 | 1650 |

- Time complexity : $O(n^2)$

## The longest path problem*

- Can we use Dijkstra's algorithm to find the longest path from a starting vertex to an ending vertex in an acyclic directed graph?
- There are 3 possible ways to apply Dijkstra's algorithm:
  - Directly use "max" operations instead of "min" operations.
  - Convert all positive weights to be negative. Then find the shortest path.
  - Give a very large positive number M. If the weight of an edge is w, now M-w is used to replace w. Then find the shortest path.
- All these 3 possible ways would not work!
- **NP-complete problem**

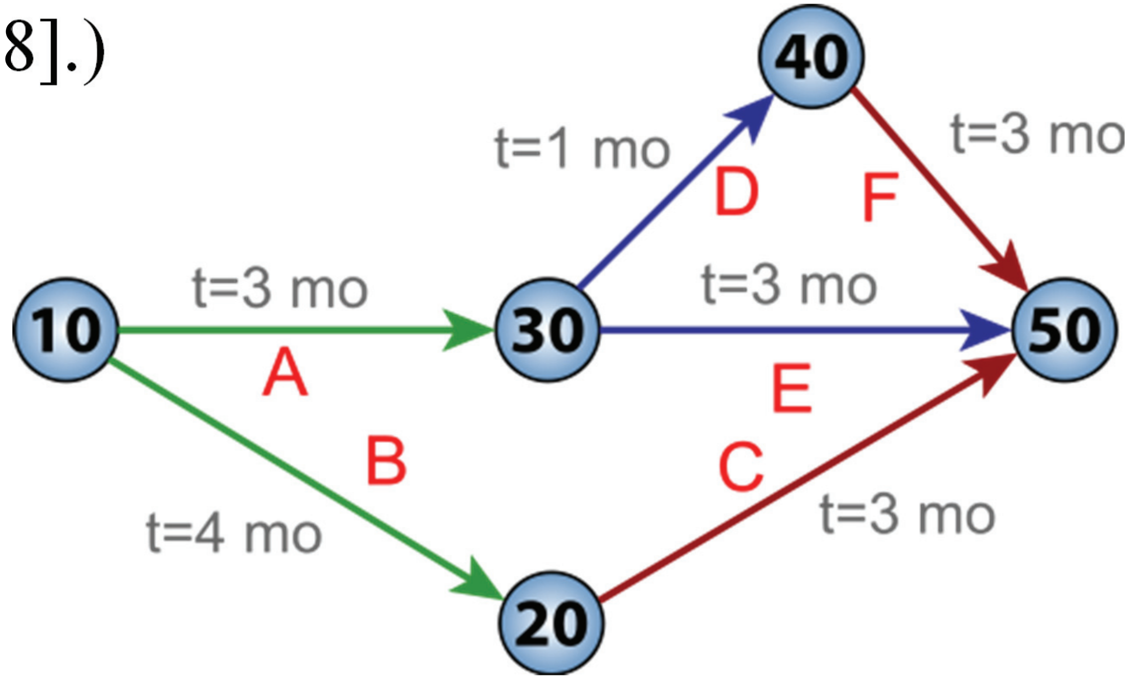
## CPM for the longest path problem

- The longest path(critical path) problem can be solved by the critical path method (CPM) :

Step 1:Find a topological ordering.

Step 2: Find the critical path.

(see [Horiwitz 1998].)



## Question:

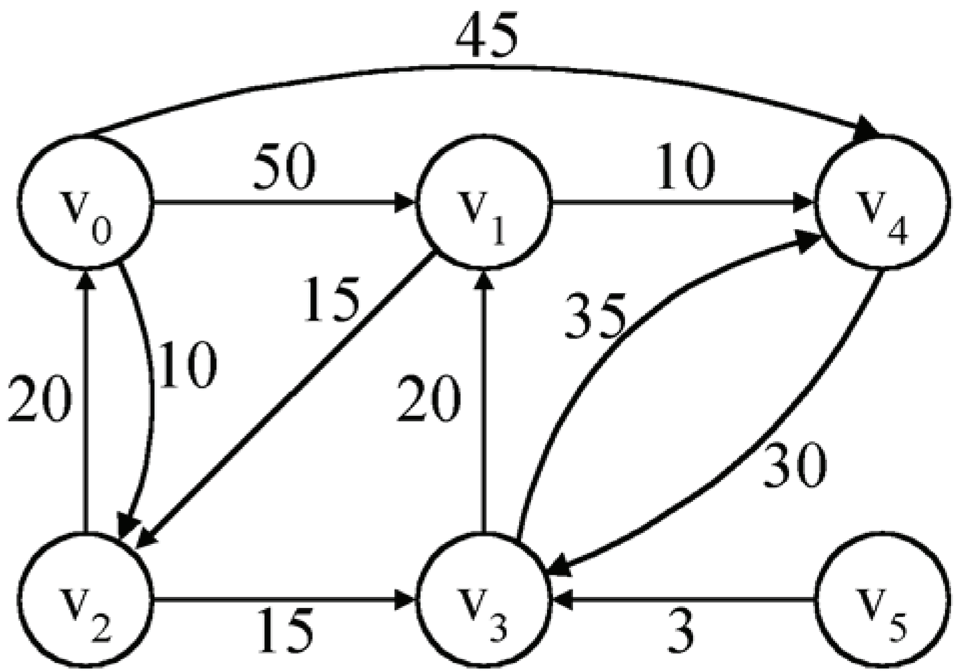
- Which is the shortest distance from node $v_0$ to $v_1$ on the given graph?

(1) 10

(2) 25

(3) 45

(4) 35.



Ans. 3

## Optimal 2-way Merge Problem

## 學習目標

- 2-Way Merge problem 問題定義
- 2-Way Merge problem的演算法設計
- 2-Way Merge problem 時間複雜度分析

# Linear Merge Algorithm

- We are given two sorted lists $L_1$ and $L_2$, $L_1 = (a_1, a_2, \dots, a_m)$ and $L_2 = (b_1, b_2, \dots, b_n)$. $L_1$ and $L_2$ can be merged into one sorted list by applying the linear merge

**Input**: Two sorted lists, $L_1 = (a_1, a_2, \dots, a_m)$ and $L_2 = (b_1, b_2, \dots, b_n)$.

**Output**: A sorted list consisting of elements in $L_1$ and $L_2$ .

```
Begin
   $i := 1$     $j := 1$
  do
    compare $a_i$ and $b_j$
    if $a_i > b_j$ then output $b_j$ and $j := j + 1$
              else output $a_i$ and $i := i + 1$
  while ($i \le m$ and $j \le n$)
  if $i > m$ then output $b_j, b_{j+1}, \dots, b_n$ ,
            else output $a_i, a_{i+1}, \dots, a_m$ .
End.
```

The worst case of merge sort

$L_1$: 1 3 5

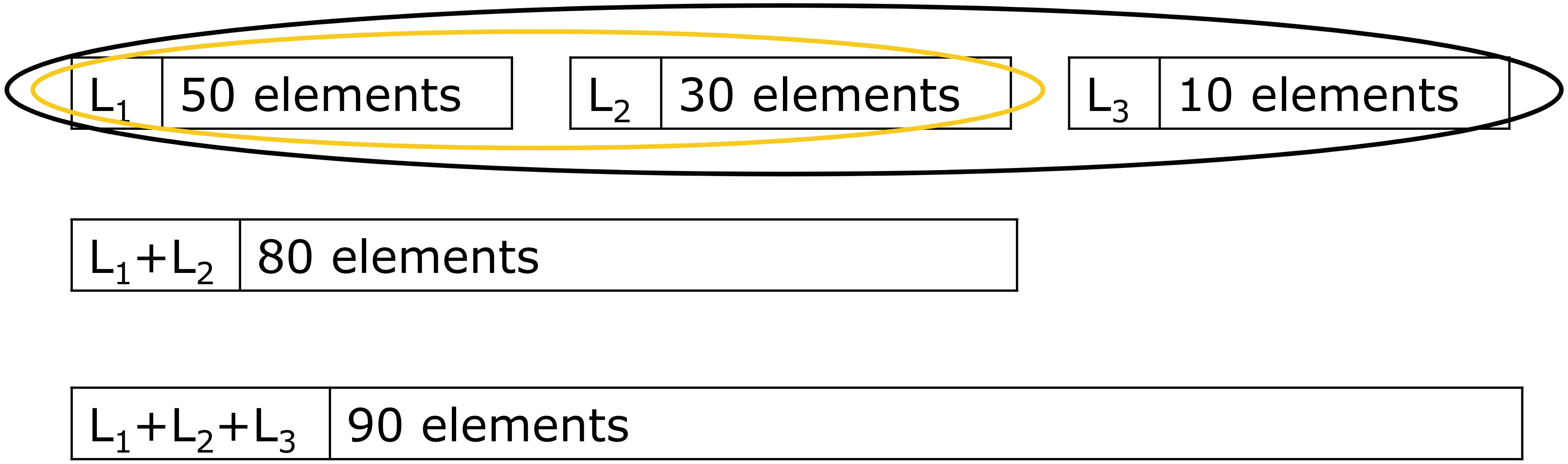
$L_2$: 2 4 6

The number of comparison required is m+n-1

## The 2-way merging problem

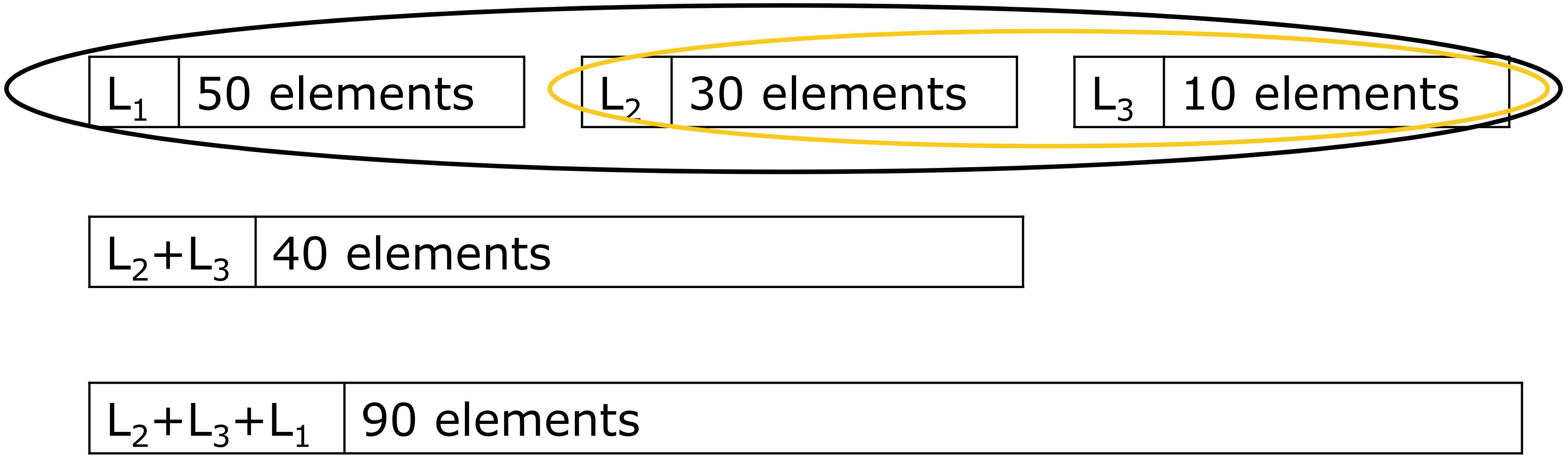
- # of comparisons required for the linear 2-way merge algorithm is ***m+ n -1*** where *m* and *n* are the lengths of the two sorted lists respectively.
- The problem: There are **m sorted lists**, each of length $n_i$. What is the optimal sequence of merging process to merge these n lists into one sorted list ?
- 50, 30, 10
  - 50+30-1=79, 80+10-1=89, total=79+89=168
  - 30+10-1=39, 40+50-1=89, total=39+89=128

## 2-WAY MERGE



$$\left.\begin{array}{l} 50+30-1 = 79 \\ 80+10-1 = 89 \end{array}\right\}$$ the number of comparisons required in this merging sequence is 168

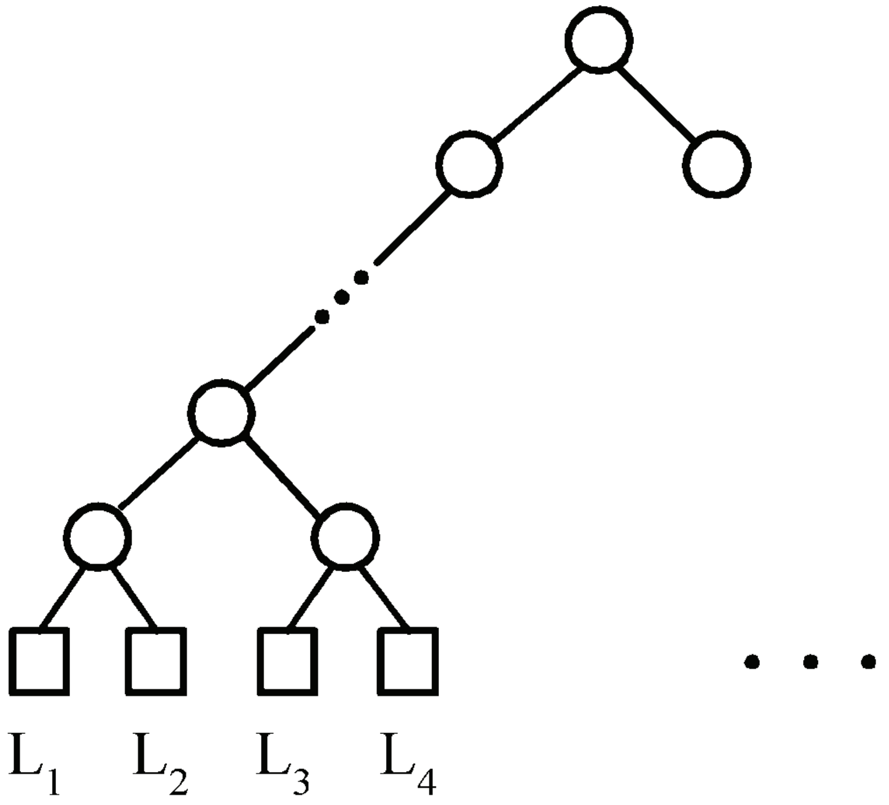
## 2-WAY MERGE



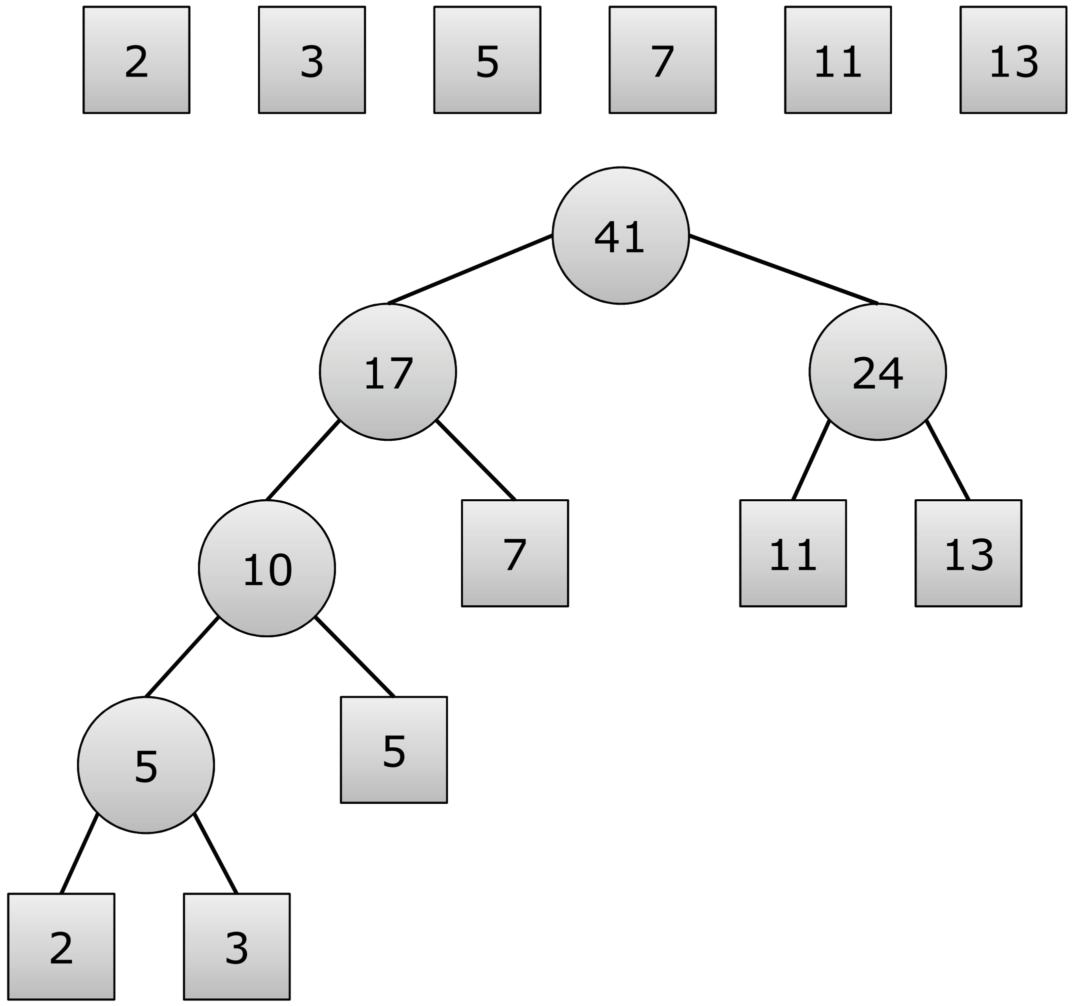
$$\left.\begin{array}{l} 30+10-1 = 39 \\ 40+50-1 = 89 \end{array}\right\}$$ the number of comparisons required is only 128

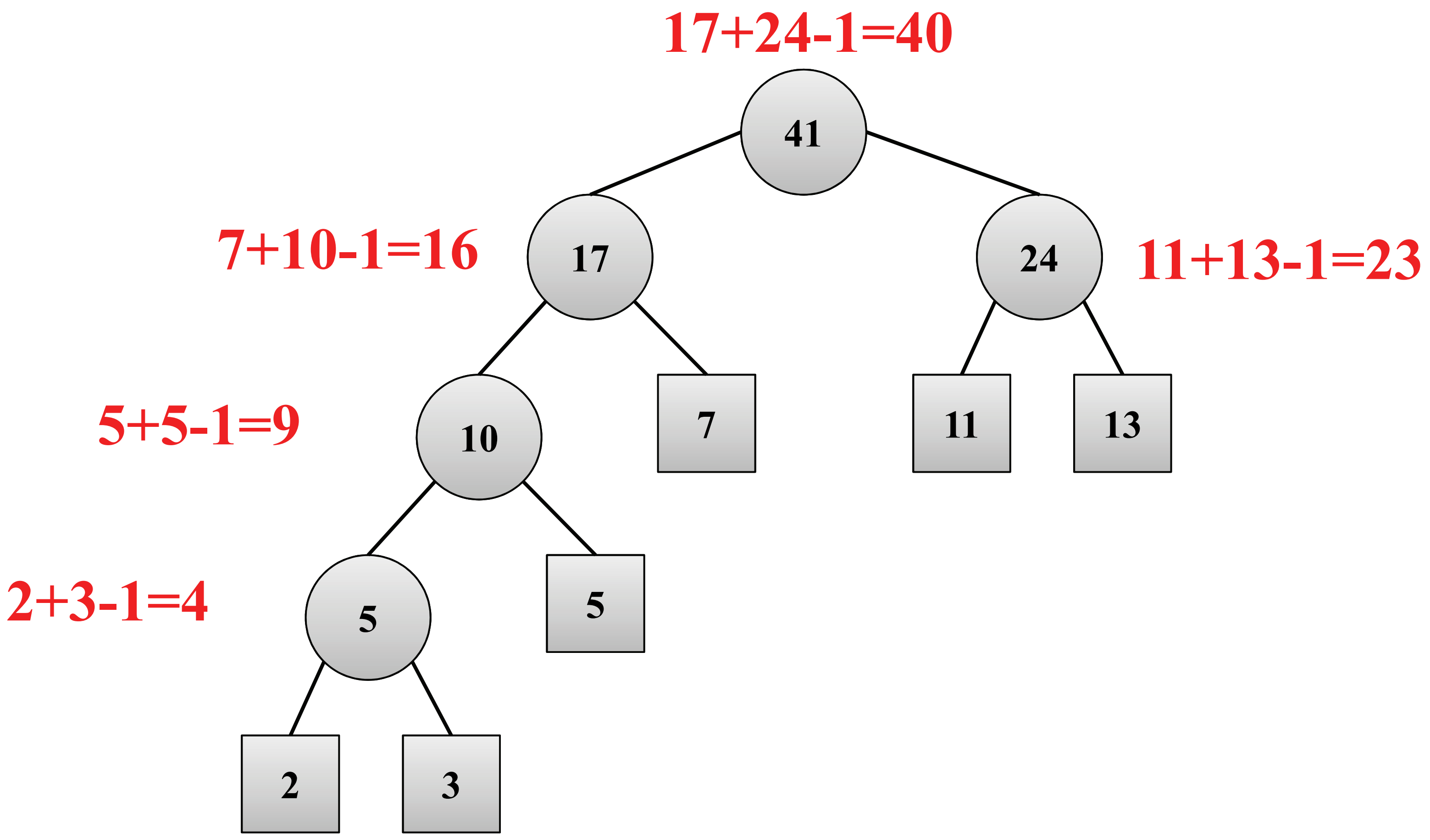
# Extended binary trees

- Extended Binary Tree Representing a 2-way Merge



# An Example Of Optimal 2-way Merge Problem





Use n+m instead of n+m-1.

# Optimal 2-way merge tree

**Input:** $m$ sorted lists, $L_i$, $i = 1, 2, \dots, m$, each $L_i$ consisting of $n_i$ elements.

**Output:** An optimal 2-way merge tree.

**Step 1:** Generate $m$ trees, where each tree has exactly one node (external node) with weight $n_i$.

**Step 2:** Choose two trees $T_1$ and $T_2$ with minimal weights.

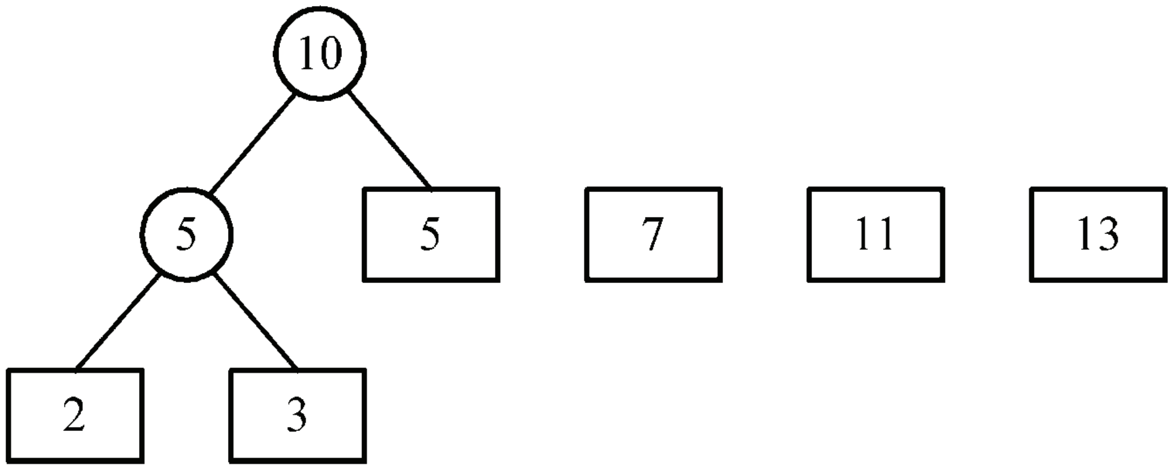
**Step 3:** Create a new tree $T$ whose root has $T_1$ and $T_2$ as its subtrees and weight is equal to the sum of weights $T_1$ and $T_2$.

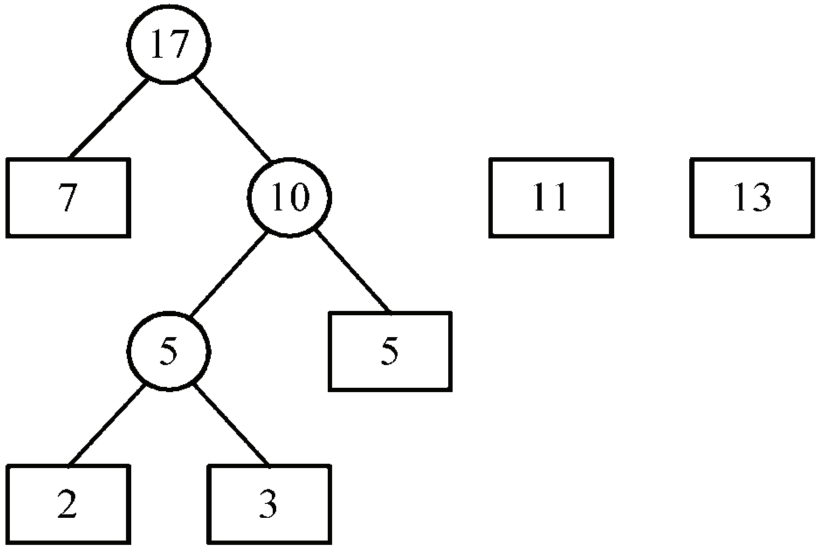
**Step 4:** Replace $T_1$ and $T_2$ by $T$.

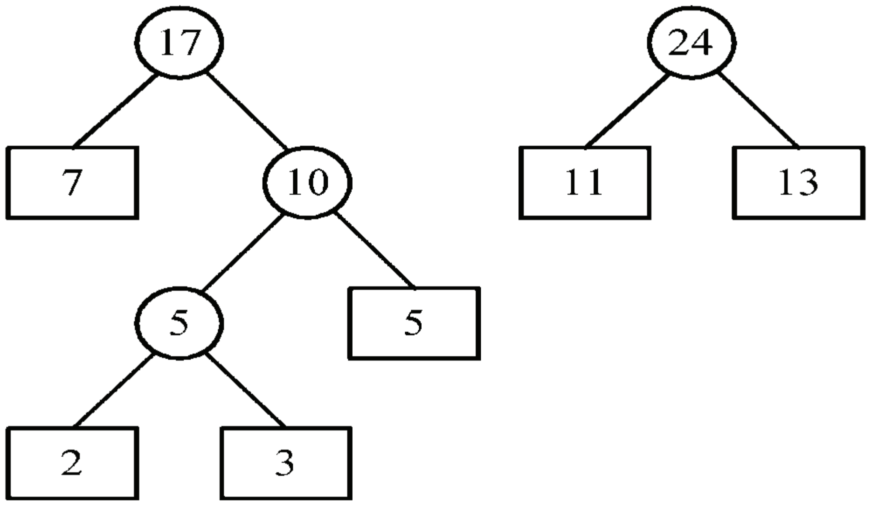
**Step 5:** If there is only one tree left, stop and return; otherwise, go to Step 2.

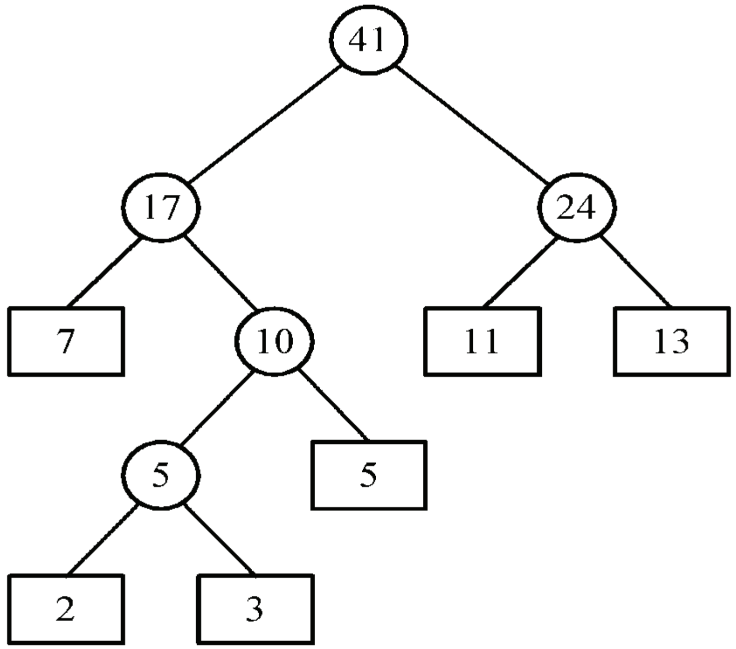
## An example of 2-way merging

- 6 sorted lists with lengths 2, 3, 5, 7, 11 and 13.









- Corrected proof by induction.
- Time complexity for generating an optimal extended binary tree:

**O(n log n)** using min-heap,

## Question:

- Use two-way merging algorithm to merge three sorted lists with lengths 50, 30, 10, what is the minimal number of comparisons?

(1) 168

(2) 128

(3) 130

(4) 170.

Ans. 2

## Huffman codes

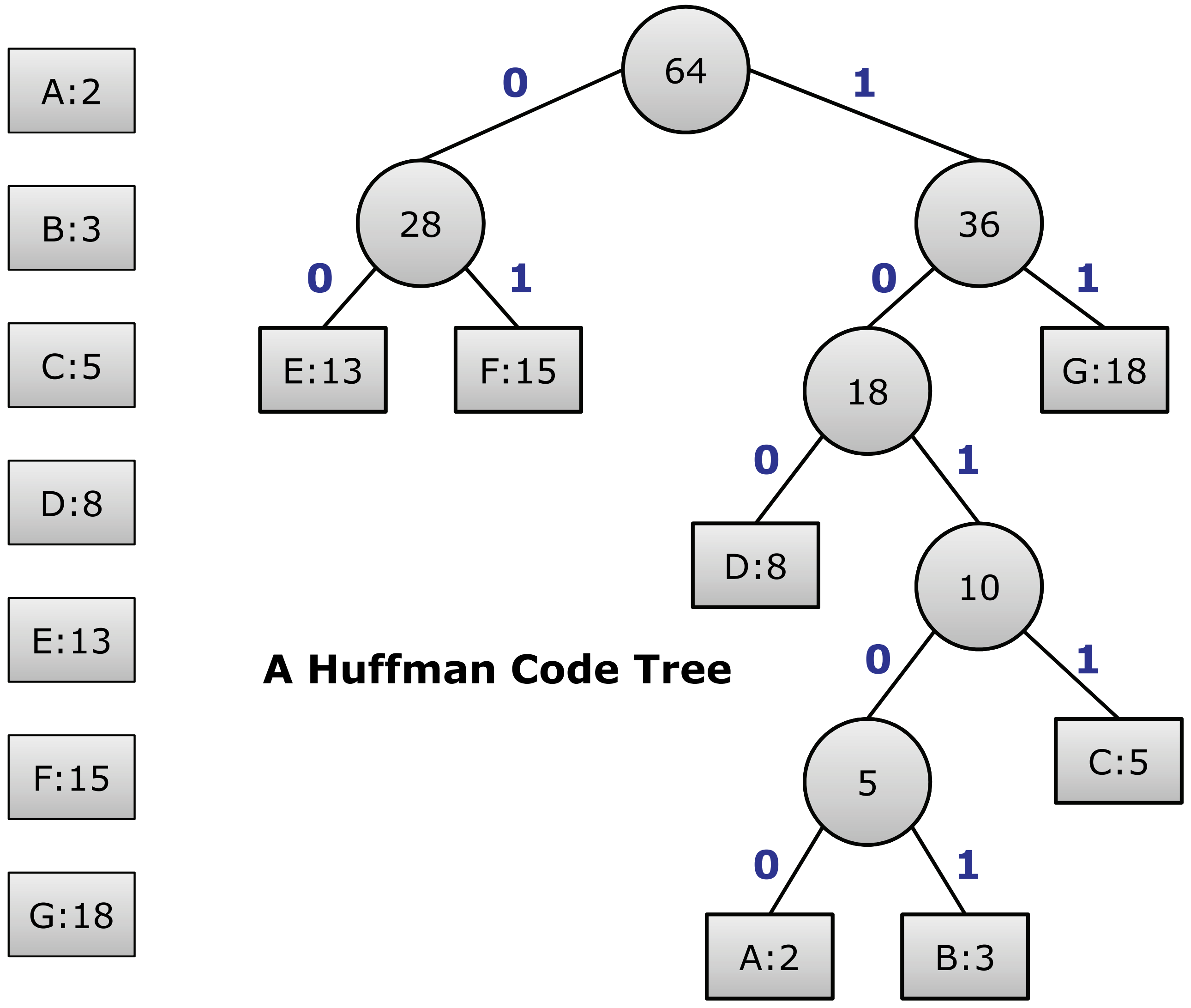
## 學習目標

- **Hoffman codes problem 問題定義**
- **Hoffman codes problem的演算法設計**
- **Hoffman codes problem時間複雜度分析**

## Example Huffman codes

- In telecommunication, how do we represent a set of messages, each with an access frequency, by a sequence of 0's and 1's?
- To minimize the transmission and decoding costs, we may use short strings to represent more frequently used messages.
- This problem can by solved by using an extended binary tree which is used in the 2-way merging problem.

# AN EXAMPLE OF HUFFMAN ALGORITHM



A Huffman Code Tree

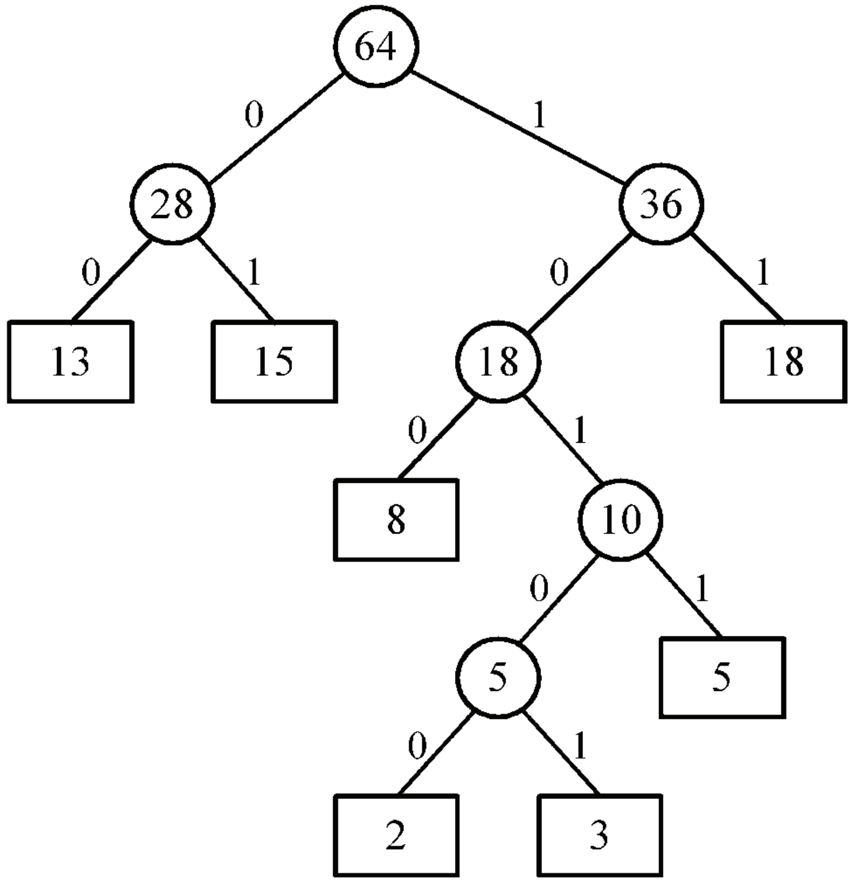
| Huffman Codes | |
|---|---|
| A | 10100 |
| B | 10101 |
| C | 1011 |
| D | 100 |
| E | 00 |
| F | 01 |
| G | 11 |

| Symbols | frequencies | Huffman | ASCII | Reduced Bits |
|---|---|---|---|---|
| A | 2 | 10100 | 1000001 | 4 |
| B | 3 | 10101 | 1000010 | 6 |
| C | 5 | 1011 | 1000011 | 15 |
| D | 8 | 100 | 1000100 | 32 |
| E | 13 | 00 | 1000101 | 65 |
| F | 15 | 01 | 1000110 | 75 |
| G | 18 | 11 | 1000111 | 90 |

## An example of Huffman algorithm

- Symbols: A, B, C, D, E, F, G
  freq. : 2, 3, 5, 8, 13, 15, 18

- Huffman codes:
  A: 10100 B: 10101 C: 1011
  D: 100 E: 00 F: 01
  G: 11
- Decode:
  Given sequence
  1100100101100101…



A Huffman code Tree

## The minimal cycle basis problem

## 學習目標

- Minimal cycle basis problem 問題定義
- Minimal cycle basis problem的演算法設計
- Minimal cycle basis problem 時間複雜度分析

## The minimal cycle basis problem

- 3 cycles:

  $A_1 = \{ab, bc, ca\}$

  $A_2 = \{ac, cd, da\}$

  $A_3 = \{ab, bc, cd, da\}$

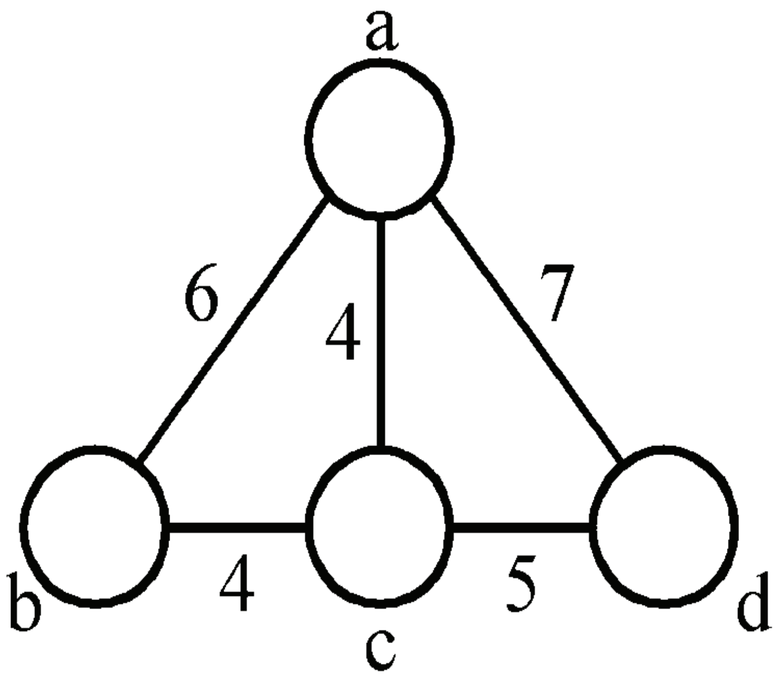
  where $A_3 = A_1 \oplus A_2$

  $(A \oplus B = (A \cup B)-(A \cap B))$

  $A_2 = A_1 \oplus A_3$

  $A_1 = A_2 \oplus A_3$

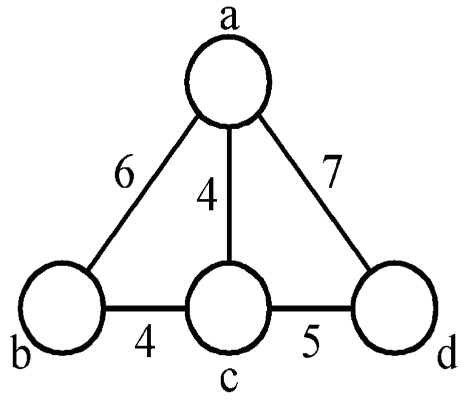
  Cycle basis : $\{A_1, A_2\}$ or $\{A_1, A_3\}$ or $\{A_2, A_3\}$



# The minimal cycle basis problem

- **Def** : A cycle basis of a graph is a set of cycles such that every cycle in the graph can be generated by applying ring sum operator ⊕ on some cycles of this basis.
- Assume weighted graph.

- Weight of cycle is the total weight of all edges in this cycle.
- The weight of a cycle basis is the total weight of all cycle in the cycle basis.
- Minimal cycle basis : smallest total weight of all edges in this cycle.
- e.g. $\{A_1, A_2\}$



## minimal cycle basis algorithm

- Algorithm for finding a minimal cycle basis:

  Step 1: Determine the size of the minimal cycle basis, demoted as k.

  Step 2: Find all of the cycles. Sort all cycles( by weight).

  Step 3: Add cycles to the cycle basis one by one. Check if the added cycle is already combination of some cycles already existing in the basis. If it is, delete this cycle.

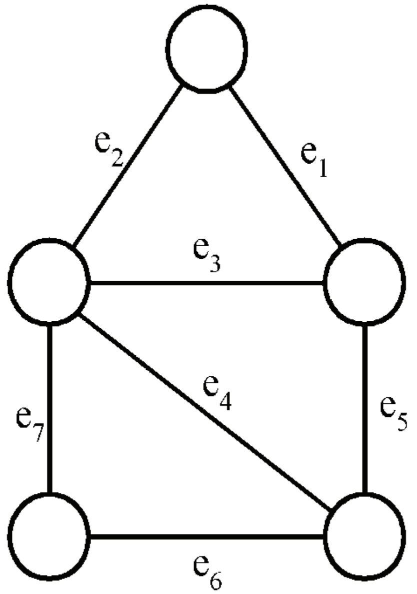
  Step 4: Stop if the cycle basis has k cycles.

# The minimal cycle basis problem – detail description

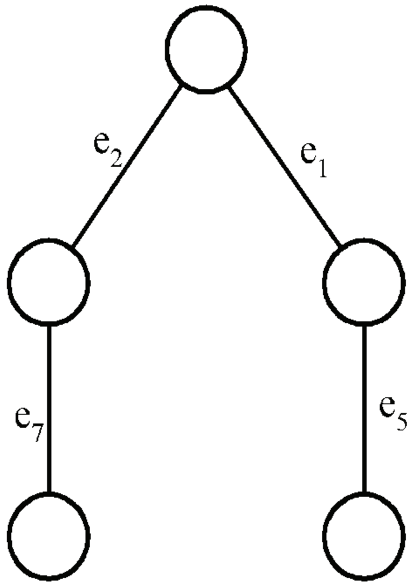
- Step 1 : (Determine the size of the minimal cycle basis, demoted as k)

A cycle basis corresponds to the fundamental set of cycles with respect to a spanning tree.

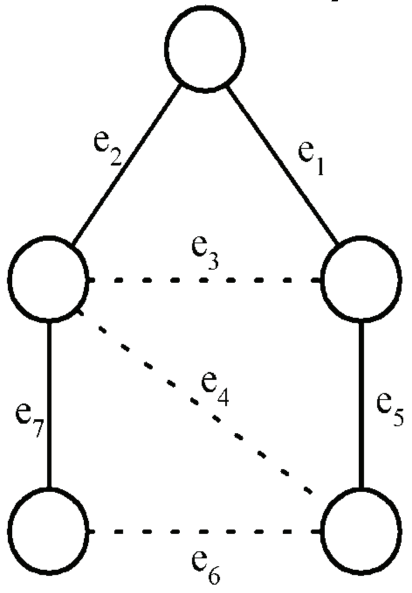
a graph



a spanning tree



a fundamental set of cycles



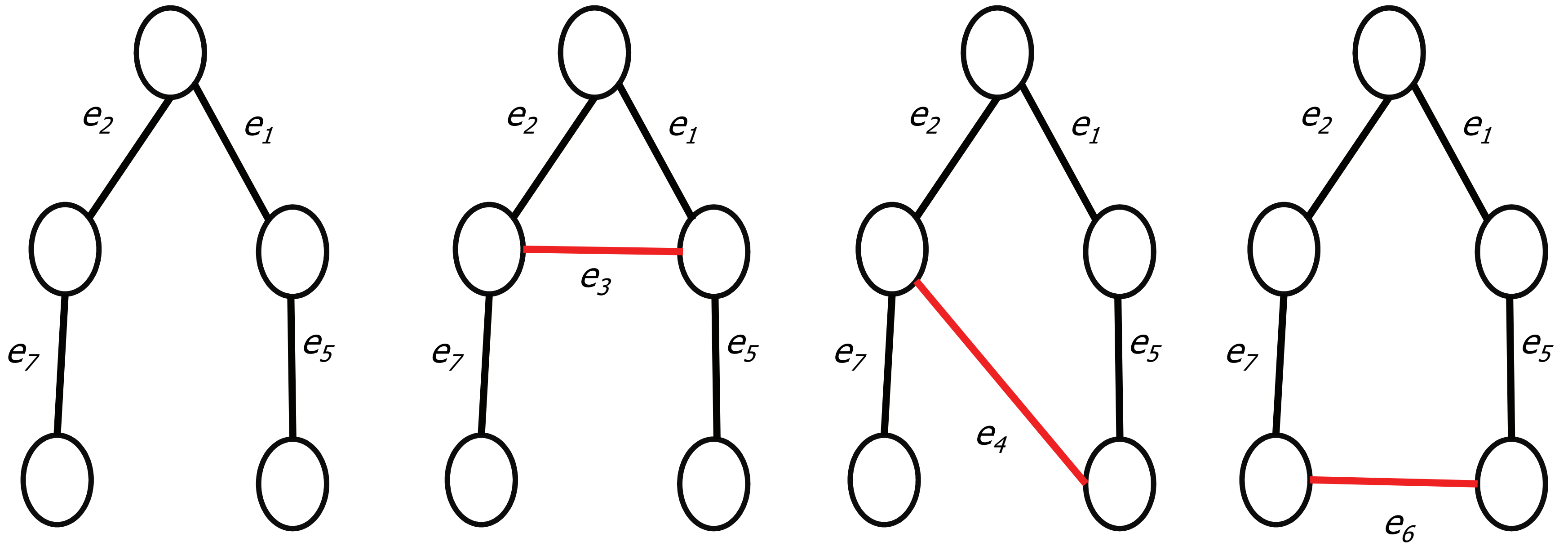
# of cycles in a cycle basis :

= k

= |E| - (|V|- 1)

= |E| - |V| + 1

# Relationship of SPT and cycles



- The number of independent cycles is equal to the number of edges which can be added to the spanning tree.
- Which is the size of the cycle basis. **|E|-|V|+1**

- Step 2:
  **How to find all cycles in a graph?**
  [Reingold, Nievergelt and Deo 1977]
  **How many cycles in a graph in the worst case?**

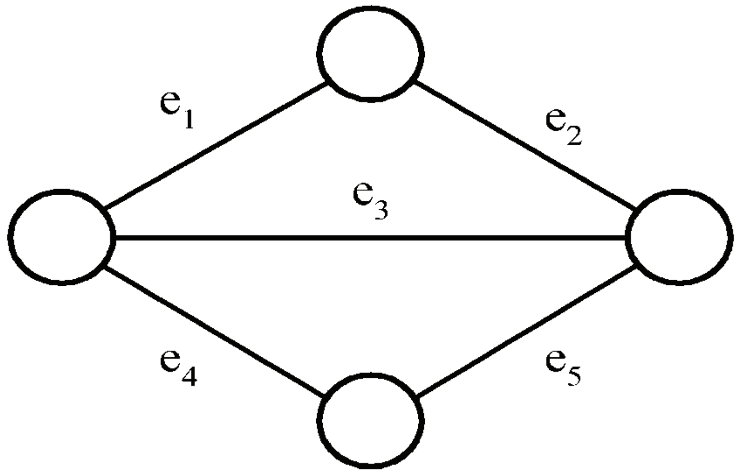
  In a complete digraph of n vertices and n(n-1) edges:

$$\sum_{i=2}^{n} C_i^n (i-1)! > (\mathrm{n}-1)!$$

i: cycle length

- Step 3:
  How to check if a cycle is a linear combination of some cycles?
  Using Gaussian elimination.

# Gaussian elimination



- 2 cycles $C_1$ and $C_2$ are represented by a 0/1 matrix

$$\begin{array}{c|ccccc} & e_1 & e_2 & e_3 & e_4 & e_5 \\ \hline C_1 & 1 & 1 & 1 & & \\ C_2 & & & 1 & 1 & 1 \end{array}$$

- Add $C_3$

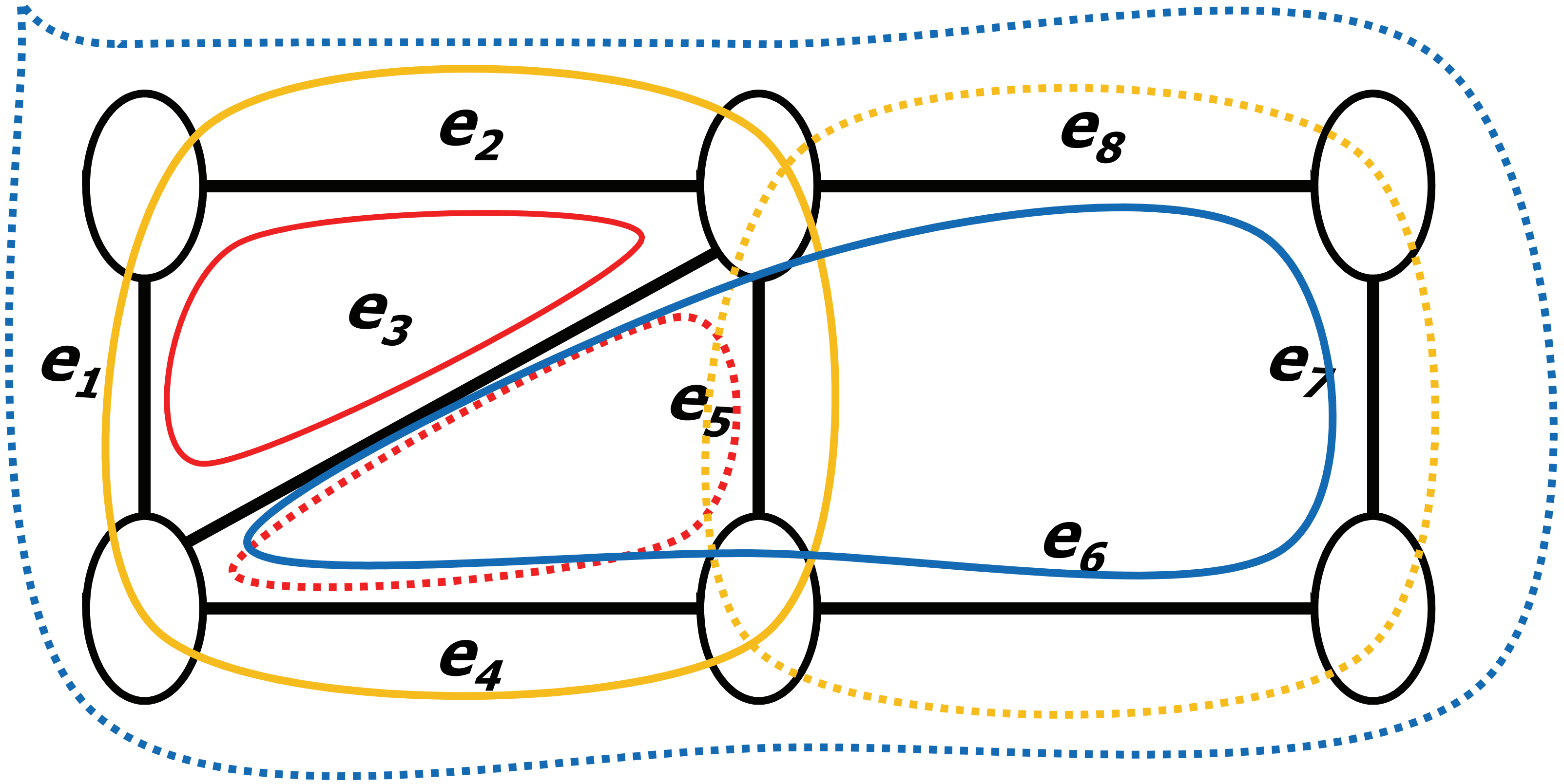
$$\begin{array}{c|ccccc} & e_1 & e_2 & e_3 & e_4 & e_5 \\ \hline C_1 & 1 & 1 & 1 & & \\ C_2 & & & 1 & 1 & 1 \\ C_3 & 1 & 1 & & 1 & 1 \end{array}$$

- $\oplus$ on rows 1 and 3

$$\begin{array}{c|ccccc} & e_1 & e_2 & e_3 & e_4 & e_5 \\ \hline C_1 & 1 & 1 & 1 & & \\ C_2 & & & 1 & 1 & 1 \\ C_3 & & & 1 & 1 & 1 \end{array}$$

$\oplus$ on rows 2 and 3 : empty

$\because C_3 = C_1 \oplus C_2$



$C_1=\{e_1, e_2, e_3\}$

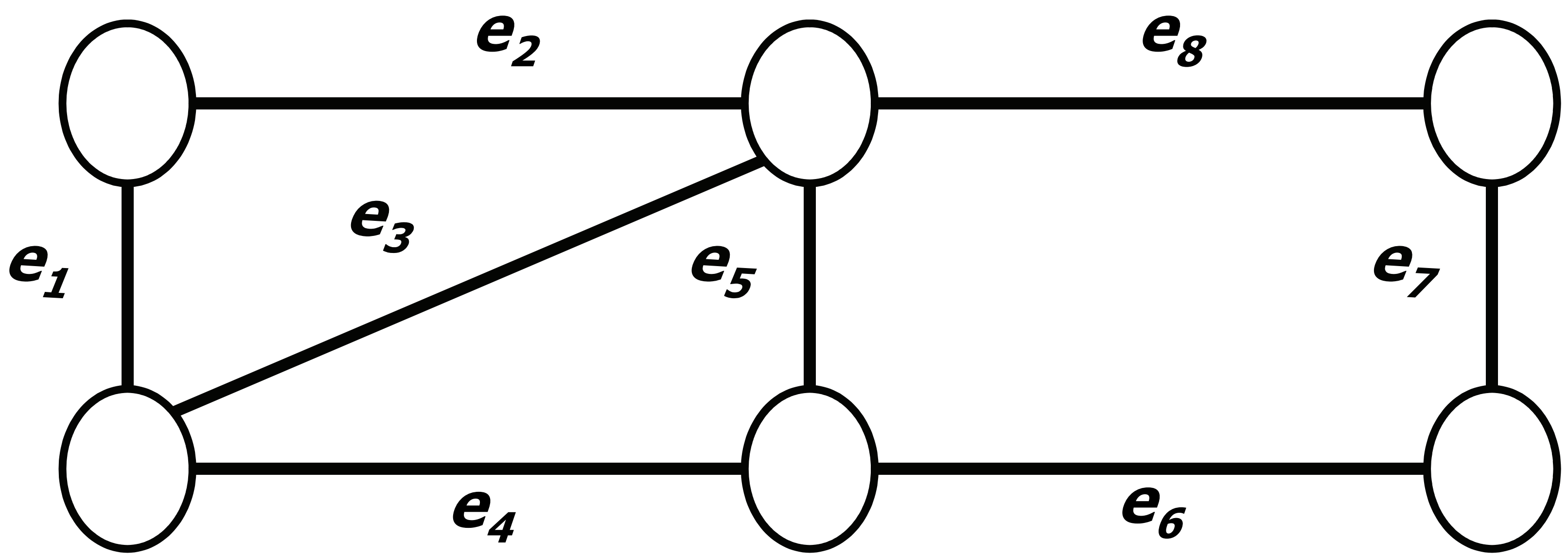
$C_2=\{e_3, e_5, e_4\}$

$C_3=\{e_2, e_5, e_4, e_1\}$

$C_4=\{e_8, e_7, e_6, e_5\}$

$C_5=\{e_3, e_8, e_7, e_6, e_4\}$

$C_6=\{e_2, e_8, e_7, e_6, e_4, e_1\}$



|E|=8, |V|=6, k= 8-6+1

$C_1$ is added

$C_2$ is added

$C_3$ is discarded

$C_4$ is added

Minimal cycle basis is {$C_1$, $C_2$, $C_4$}

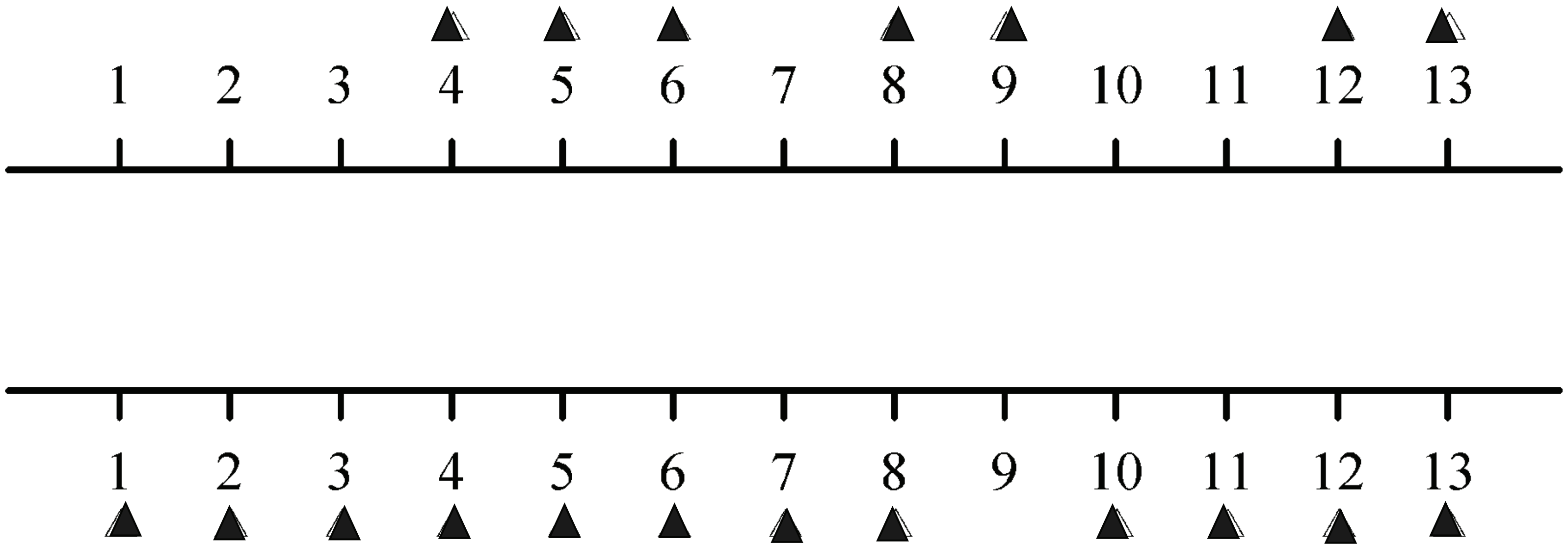
## The 2-terminal one to any special channel routing problem

## 學習目標

- 2-terminal one-to-any special channel routing problem 問題定義
- 2-terminal one-to-any special channel routing problem的演算法設計
- 2-terminal one-to-any special channel routing problem時間複雜度分析

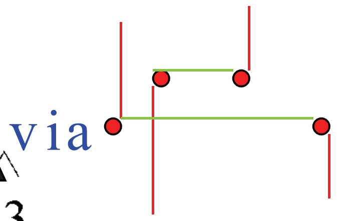
# The 2-terminal one to any special channel routing problem

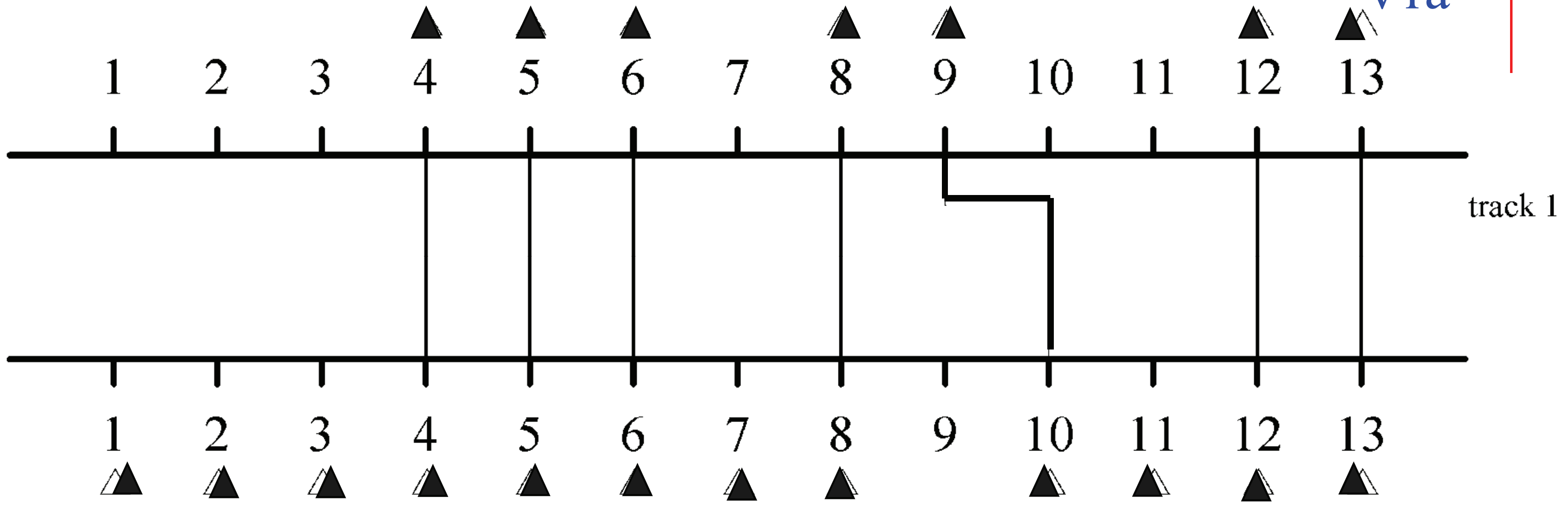
- Def: Given a set of terminals on the upper row and another set of terminals on the lower row, we have to connect each upper terminal to the lower row in a one to one fashion. This connection requires that # of tracks used is minimized.

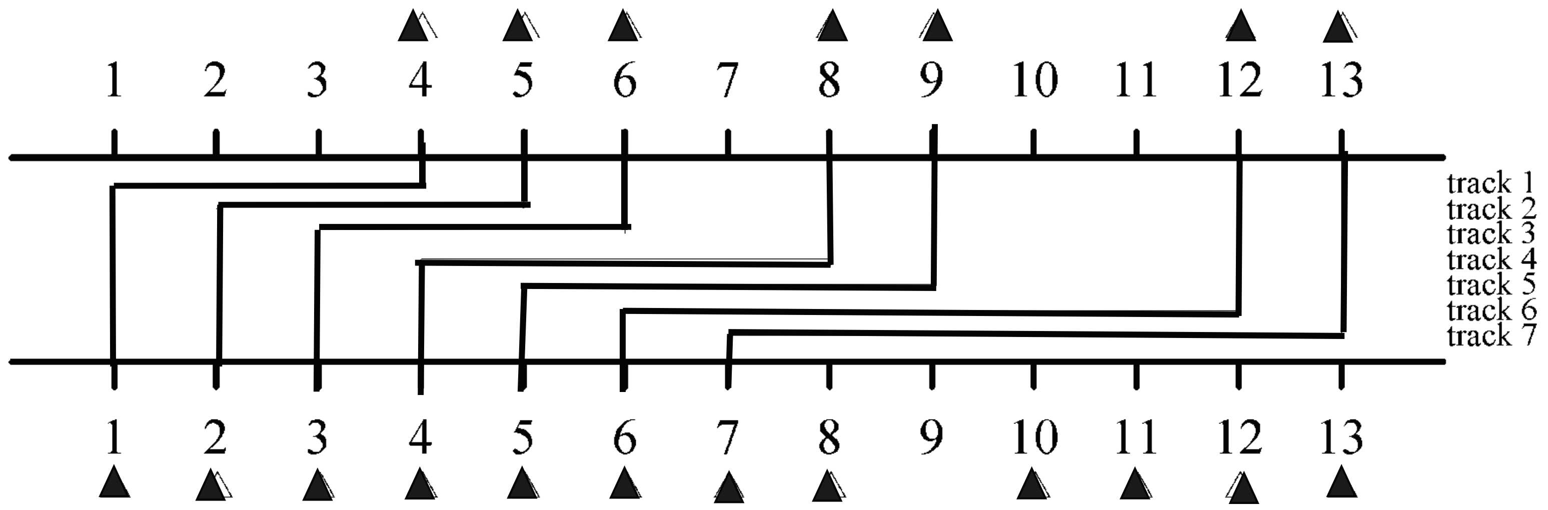


terminal

# 2 feasible solutions

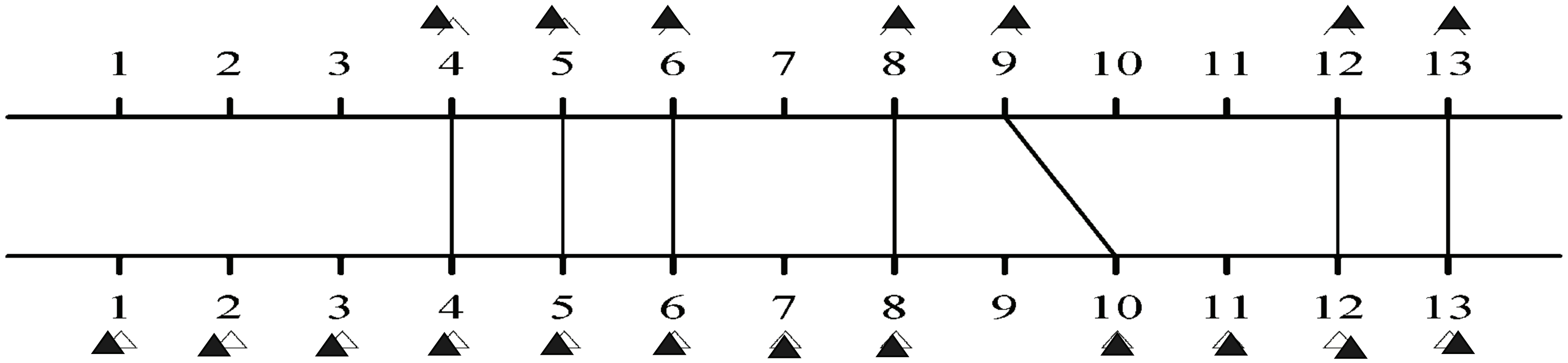




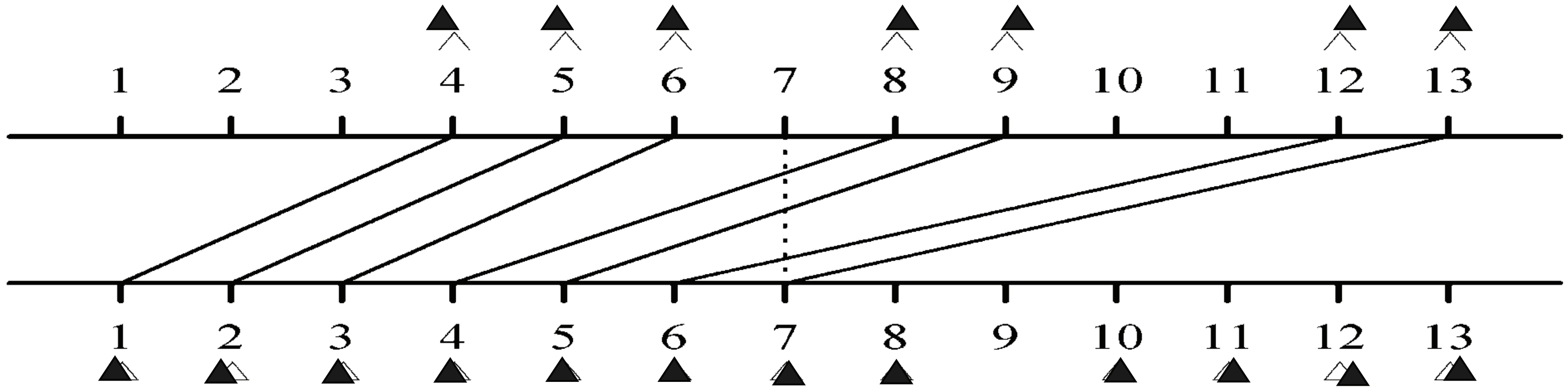


# Redrawing solutions (local density)

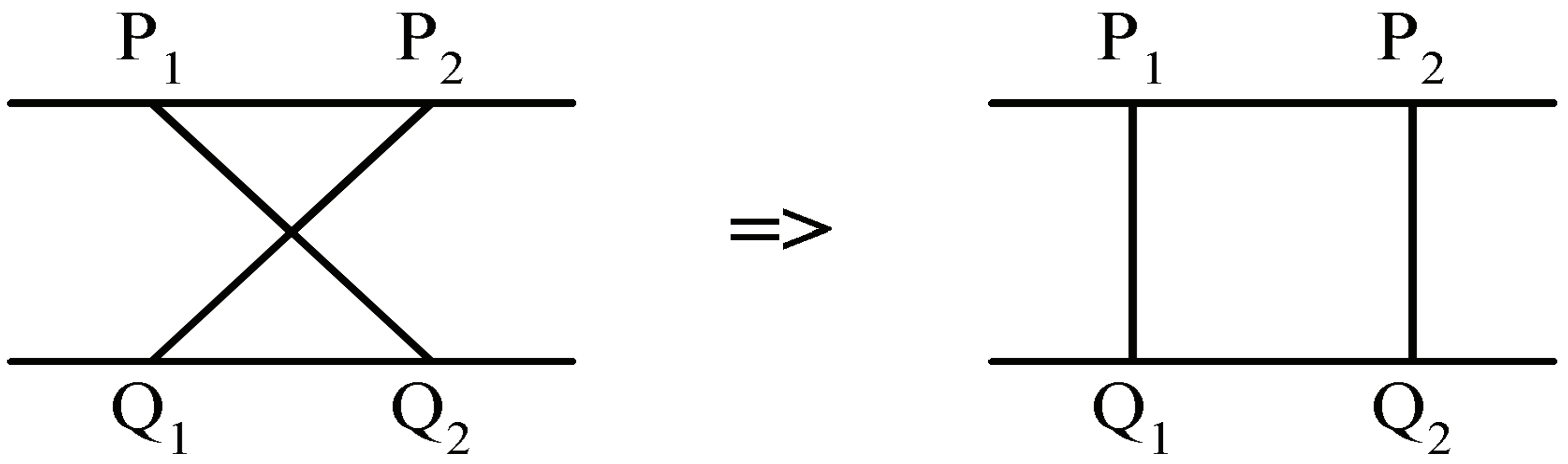
(a) Optimal solution



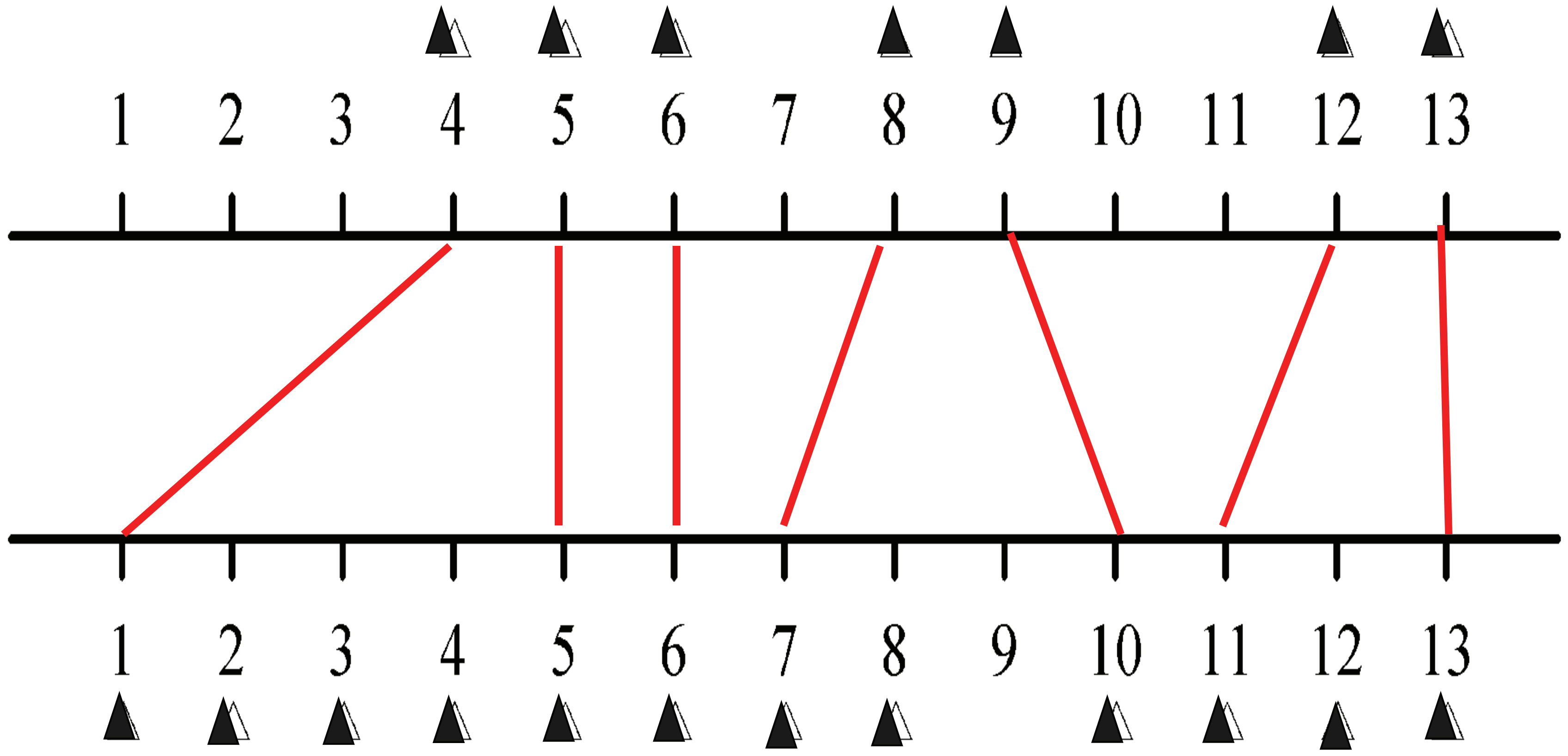
(b) Another solution



- At each point, the local density of the solution is # of lines the vertical line intersects.
- The density of a solution is the maximum local density.
- The problem: to minimize the density.
- **The density is a lower bound of # of tracks.**
- **Assume the minimum density is known.**
- Upper row terminals: $P_1, P_2, \dots, P_n$ from left to right
- Lower row terminals: $Q_1, Q_2, \dots, Q_m$ from left to right $m > n$.
- It would never have a **crossing connection**:



## Assume minimum density is d=1



## Greedy Algorithm

- Suppose that we have a method to determine the minimum density, d, of a problem instance.
- The **greedy algorithm**:

Step 1 : $P_1$ is connected $Q_1$.

Step 2 : After $P_i$ is connected to $Q_j$, we check whether $P_{i+1}$ can be connected to $Q_{j+1}$. If the density is increased to d+1, try to connect $P_{i+1}$ to $Q_{j+2}$.

Step 3 : Repeat Step2 until all $P_i$'s are connected.

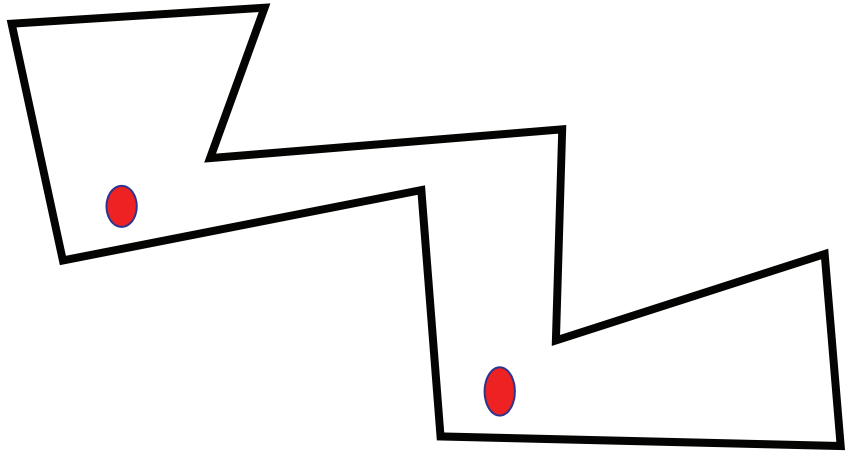
## The Minimum Cooperative Guards Problem for 1-Spiral Polygon

## 學習目標

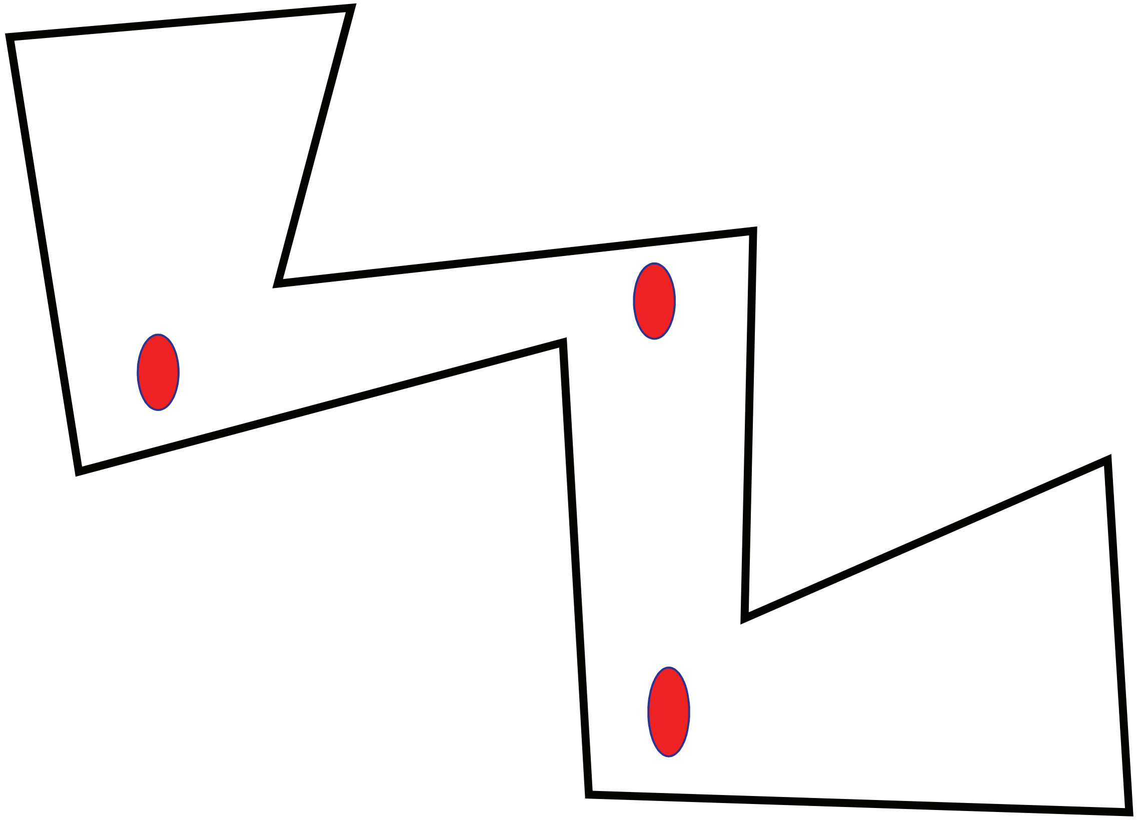
- Minimum Cooperative Guards Problem for 1-spiral problem 問題定義
- Minimum Cooperative Guards Problem for 1-spiral problem的演算法設計
- Minimum Cooperative Guards Problem for 1-spiral problem時間複雜度分析

## The Minimum Cooperative Guards Problem for 1-Spiral Polygons Solved by the Greedy Method

- The minimum cooperative guards problem is a variation of the art gallery problem,
- We are given a polygon, which represents an art gallery, and we are required to place a minimum number of guards in the polygon such that every point of the polygon is visible to at least one guard.
- The art gallery problem is an NP-hard problem.



## Minimum cooperative guards problem



NP-hard problem

## The minimum cooperative guards problem

- We represent the relationship among guards by a visibility graph G of the guards.
- In G, every guard is represented by a vertex and there is an edge between two vertices if and only if the corresponding guards can see each other.
- In addition to finding a minimum set of guards who can see the given polygon, we further require that the visibility graph of these guards is connected.
- In other words, we require that **no guard is isolated and there is a path between every pair of guards.** We call such a problem the ***mininnun cooperative guards problem.***

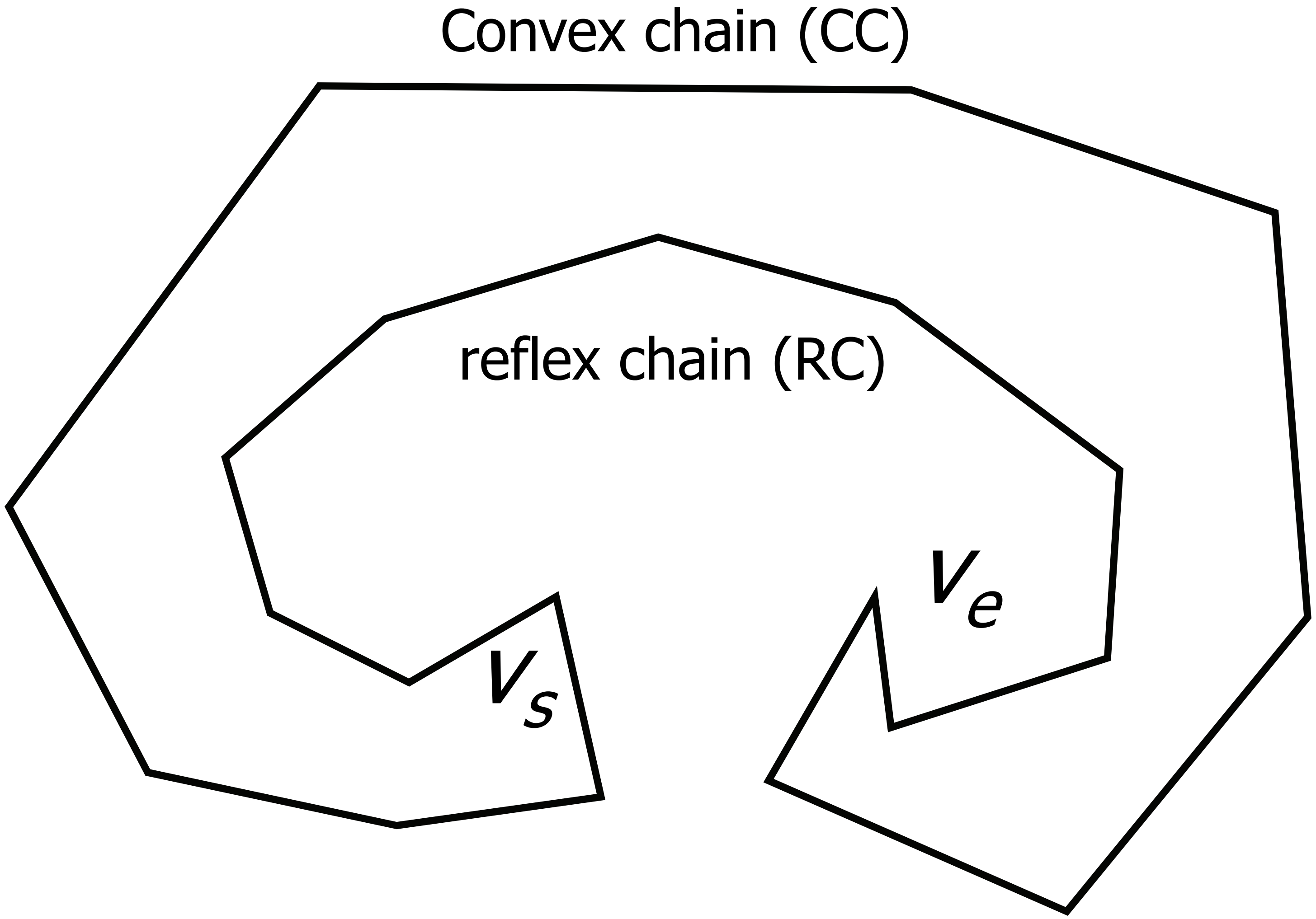
## 1-spiral polygons

- the minimum cooperative guards problem is NP-complete
- There is a greedy algorithm for this problem for 1-spiral (螺旋形的) polygons.

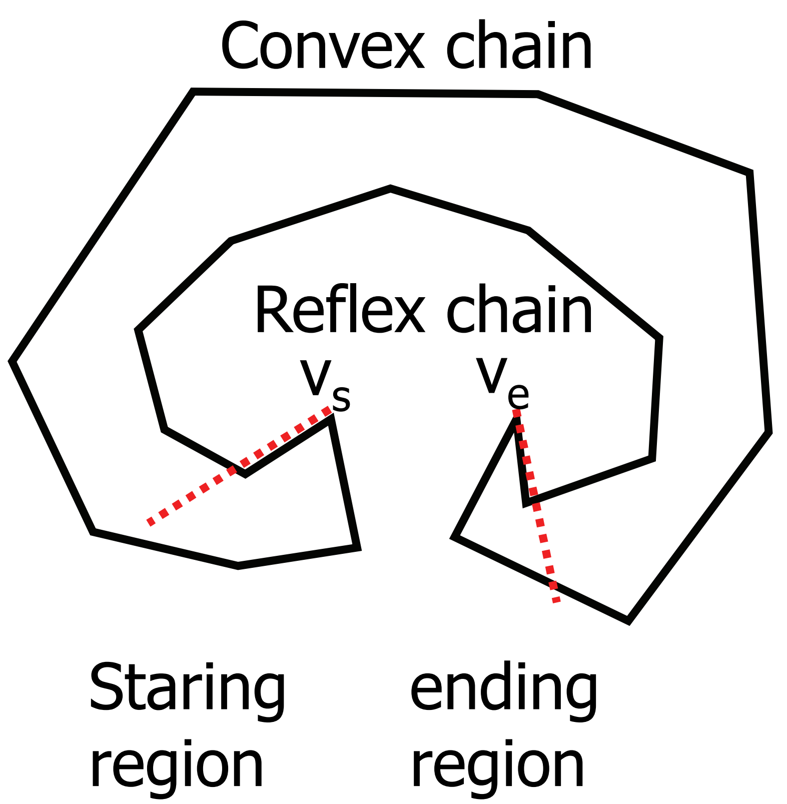
- reflex (convex) chains, denoted as RC (CC), of a simple polygon is a chain of this polygon if all of the vertices on this chain are reflex (convex) with respective to the interior of the polygon.

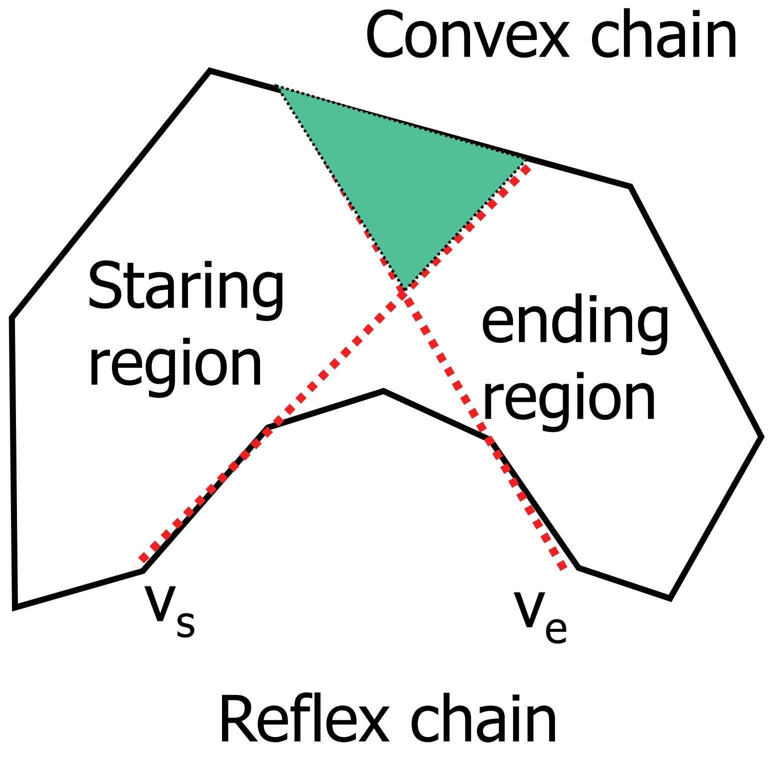
- A 1-spiral polygon P is a simple polygon whose boundary can be partitioned into a reflex chain and a convex chain.

# 1-spiral polygons

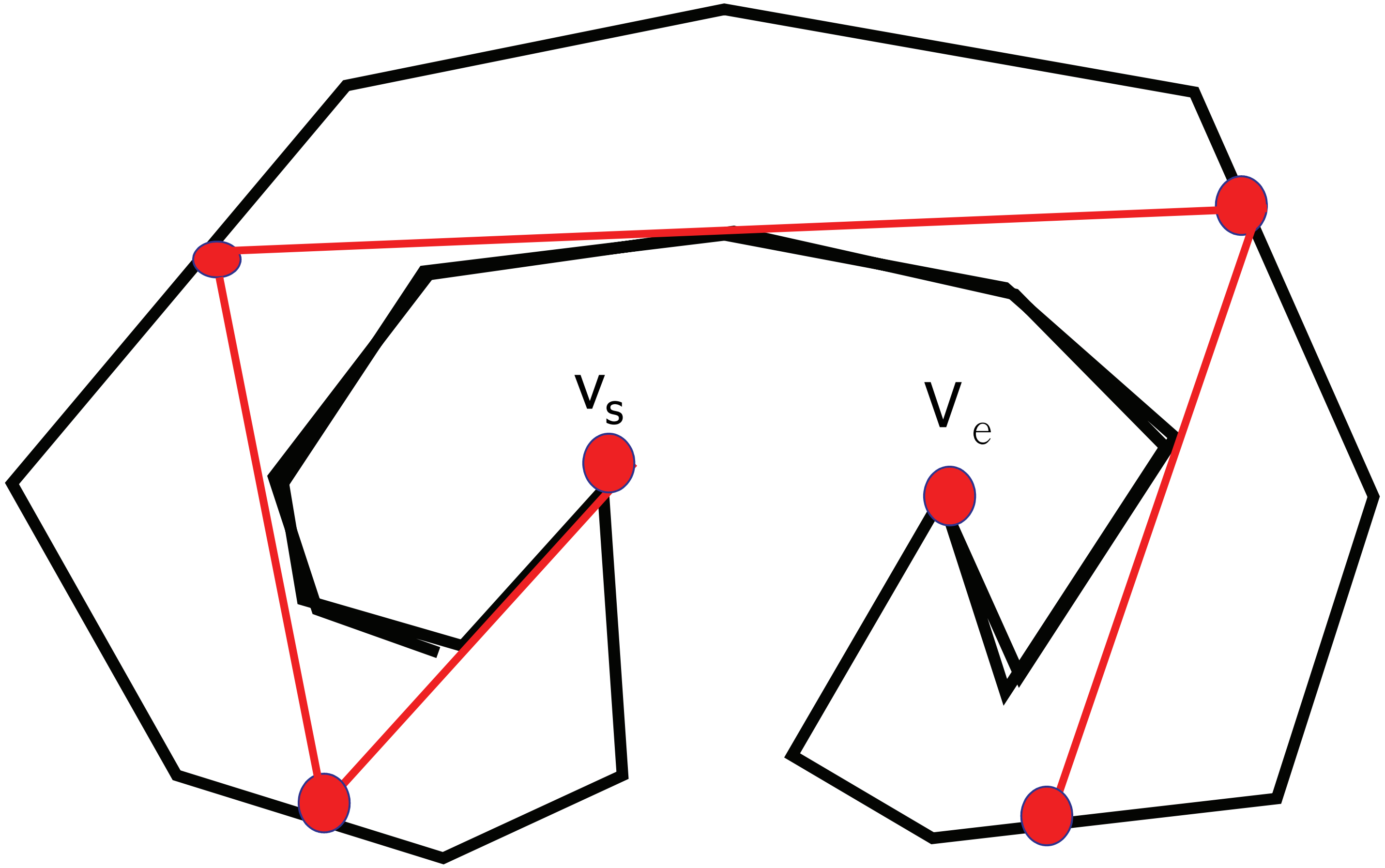


## Staring & ending regions

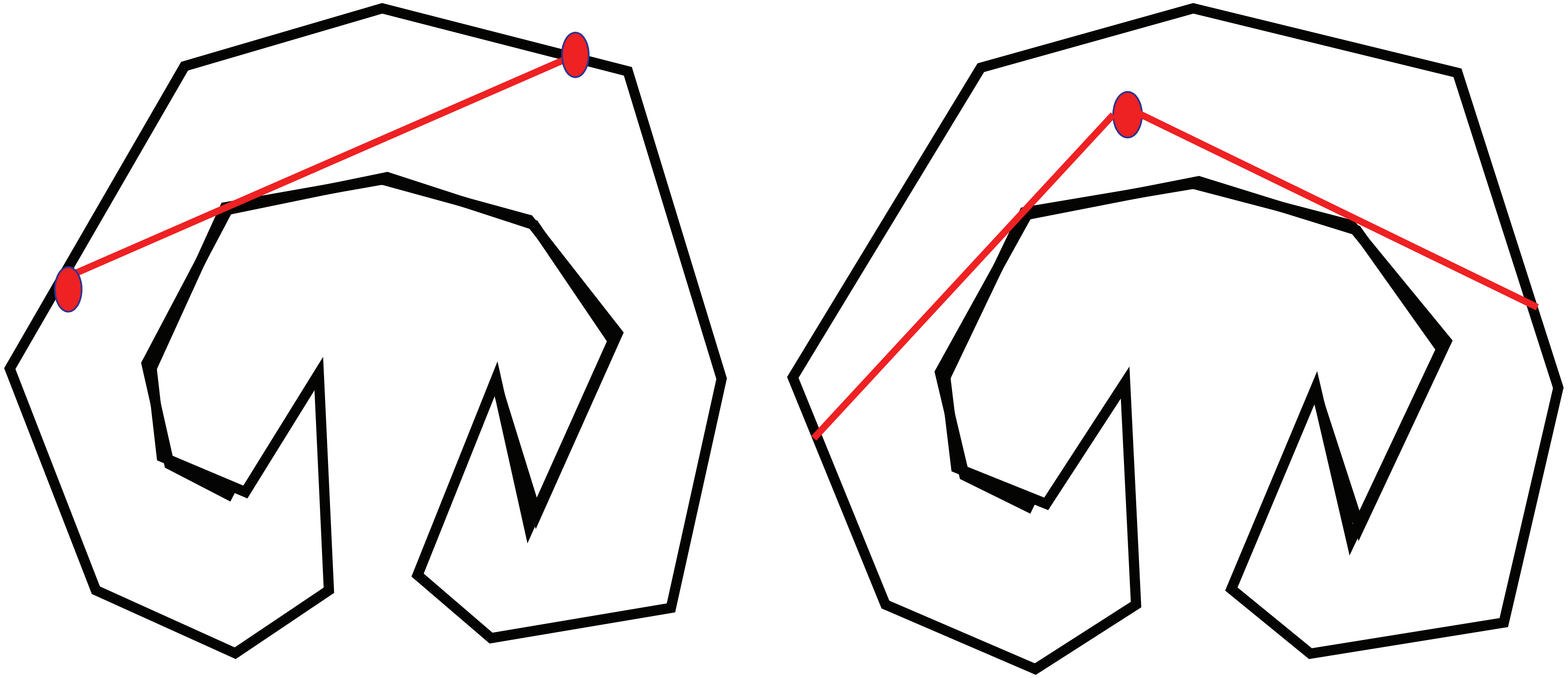




# Example



# The left and right supporting lines w.r.t. a



Supporting line segment

**Algorithm 3–7 □ An algorithm to solve the minimum cooperative guards problem for 1-spiral polygons**

**Input:** A 1-spiral polygon P.

**Output:** A set of points which is the solution of the minimum cooperative guards problem.

```
Step 1. Find the reflex chain $RC$ and convex chain $CC$ of $P$.
Step 2. Find the intersection points $l_1$ and $r_1$ of $CC$ with the directed lines
        starting from $v_s$ and $v_e$ along the first and last edges of $RC$, respectively.
Step 3. Let $k = 1$.
        While $l_k$ is not on the ending region do
            Draw the left tangent of $l_k$ with respect to $RC$ until it hits $CC$ at
            $l_{k+1}$. ($\overline{l_k l_{k+1}}$ is a left supporting line segment with respect to $l_k$.)
            Let $k = k + 1$.
        End While.
Step 4. Report $\{l_1, l_2, \ldots, l_k\}$.
```

## The knapsack problem

## 學習目標

- The Knapsack problem 問題定義
- The Knapsack problem的演算法設計

## Appendix: The knapsack problem

- n objects, each with a weight $w_i > 0$
  a profit $p_i > 0$
  capacity of knapsack: M

Maximize $\sum_{1 \le i \le n} p_i x_i$

Subject to $\sum_{1 \le i \le n} w_i x_i \le M$

$0 \le x_i \le 1,\ 1 \le i \le n$

## The knapsack algorithm

- The greedy algorithm:

  Step 1: Sort $p_i/w_i$ into nonincreasing order.

  Step 2: Put the objects into the knapsack according to the sorted sequence as possible as we can.

- e. g.

  $n = 3$, $M = 20$, $(p_1, p_2, p_3) = (25, 24, 15)$

  $(w_1, w_2, w_3) = (18, 15, 10)$

  Sol: $p_1/w_1 = 25/18 = 1.32$

  $p_2/w_2 = 24/15 = 1.6$

  $p_3/w_3 = 15/10 = 1.5$

  Optimal solution: $x_1 = 0$, $x_2 = 1$, $x_3 = 1/2$

## Question

- Consider the knapsack problem with 3 items and weight limitation 20, and $(p_1, p_2, p_3) = (25, 24, 15)$ and $(w_1, w_2, w_3) = (18, 15, 10)$, what is the maximal profit of the selection?

(1) $x_1 = 0, x_2 = 1, x_3 = 1/2$

(2) $x_1 = 1, x_2 = 0, x_3 = 1/2$

(3) $x_1 = 1, x_2 = 1, x_3 = 0$

(4) $x_1 = 1, x_2 = 0, x_3 = 1$ .

Ans. 1