Time Complexity

classify	algorithm	Time Complexity
Backtrack	BFS	Adjacent list:$O(\|E\|+\|V\|)$,matrix: $O(n^2)$
	DFS	Adjacent list:$O(\|E\|+\|V\|)$,matrix: $O(n^2)$
Dynamic programming	Floyd-Warshall	$O(n^3)$
	Matrix Chain	$O(n^3)$
	OBST	$O(n^3)$
	LCS	O (mn)
	Longest common substring	O (mn)
	Longest Increasing Subsequence	$O(n^2)$
	Huffman	O(nlogn)
	0/1 Knapsack problem	O(nW)
Greedy	Kruskal	adjacency matrix $O(n^2)$ adjacency list O(E lg E)
	Prim's	adjacency matrix $O(n^2)$ binary heap+ adjacency list→O(ElogV) Fibonacci heap+ adjacency list→O(E+VlogV)
	Dijkstra	Linear array→$O(n^2)$ binary heap→$O((\|E\|+\|V\|)\log\|V\|)$ time Fibonacci heap→$O(\|E\|+\|V\|\log\|V\|)$ amortized
	Bellman-Ford	O(VE)
	fractional Knapsack problem	O(nlogn)
Divide-And-Conquer	Strassen's	$O(n^{\log_2 7})$

Matrix Chain Multiplication

Dynamic programming $O(n^3)$

現今已能在 O(NlogN) *時間內解決* Matrix Chain Multiplication

96 中山資工

Matrix Chain $M_{5\times3}M_{3\times7}M_{7\times2}M_{2\times9}M_{9\times4}$

	1	2	3	4	5
1	0	105	72	162	184
2		0	42	96	138
3			0	126	128
4				0	72
5					0

M(1,3)= min{ M(1,2)+5*7*2=175 , M(2,3)+5*3*2=72 }=72

M(2,4)= min{ M(2,3)+3*2*9=96 , M(3,4)+3*7*9=315 }=96

M(3,5)= min{ M(3,4)+7*9*4=378 , M(4,5)+7*2*4=128 }=128

M(1,4)= min{M(2,4)+5*3*9=231,M(1,2)+M(3,4)+5*7*9=546,M(1,3)+5*2*9=162}=162

M(2,5)= min{M(3,5)+3*7*4=212,M(2,3)+M(4,5)+3*2*4=138,M(2,4)+3*9*4=204}=138

M(1,5)= min {M(2,5)+5*3*4=198, M(1,2)+M(3,5)+5*7*4=373,

M(1,3)+M(4,5)+5*2*4=184,M(1,4)+5*9*4=342}=184

$(M_{5\times3}(M_{3\times7}M_{7\times2}))(M_{2\times9}M_{9\times4})$

Optimal Binary Search Tree

Dynamic programming $O(n^3)$

Example 1:

Let n=4, (a1, a2, a3, a4) = (do,for ,void , while)

(p1, p2, p3, p4)=(3,3,1,1) (q0, q1, q2, q3,q4)=(2,3,1,1,1)

w_{00}=2 c_{00}=0 r_{00}=0	w_{11}=3 c_{11}=0 r_{11}=0	w_{22}=1 c_{22}=0 r_{22}=0	w_{33}=1 c_{33}=0 r_{33}=0	w_{44}=1 c_{44}=0 r_{44}=0
w_{01}=8 c_{01}=8 r_{01}=1	w_{12}=7 c_{12}=7 r_{12}=2	w_{23}=3 c_{23}=3 r_{23}=3	w_{34}=3 c_{34}=3 r_{34}=4	
w_{02}=12 c_{02}=19 r_{02}=1	w_{13}=9 c_{13}=12 r_{13}=2	**w_{24}=5** **c_{24}=8** **r_{24}=3 or 4**		
w_{03}=14 c_{03}=25 r_{03}=2	w_{14}=11 c_{14}=19 r_{14}=2			
w_{04}=16 **c_{04}=32** **r_{04}=2**				

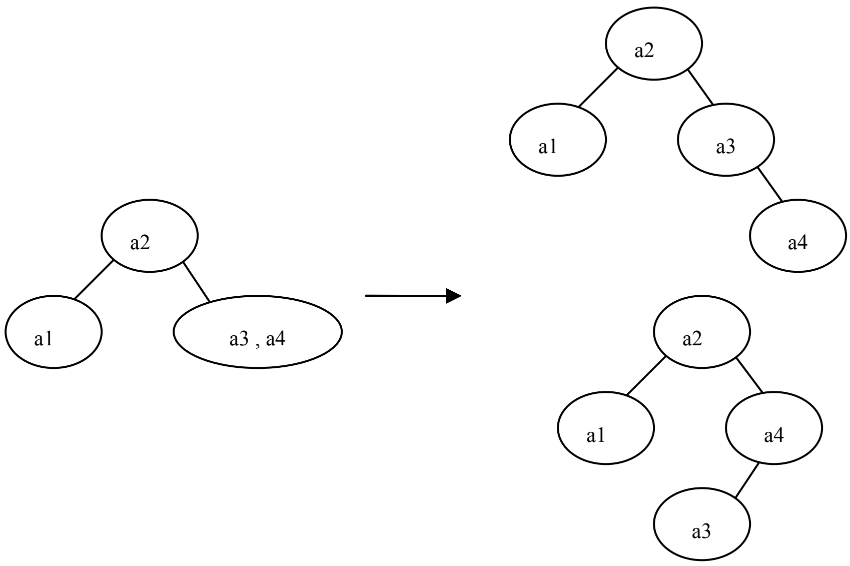
$$\begin{cases} w_{01} = p_1 + q_1 + w_{00} = 8 \\ c_{01} = w_{01} + \min\{c_{00}, c_{11}\} = 8 \\ r_{01} = 1 \end{cases}$$

$$\begin{cases} w_{12} = p_2 + q_2 + w_{11} = 7 \\ c_{12} = w_{12} + \min\{c_{11}, c_{22}\} = 7 \\ r_{12} = 2 \end{cases}$$

$$\begin{cases} w_{23} = p_3 + q_3 + w_{22} = 3 \\ c_{23} = w_{23} + \min\{c_{22}, c_{33}\} = 3 \\ r_{23} = 3 \end{cases}$$

$$\begin{cases} w_{34} = p_4 + q_4 + w_{33} = 3 \\ c_{34} = w_{34} + \min\{c_{33}, c_{44}\} = 3 \\ r_{34} = 4 \end{cases}$$

重建OBST T_{04} 看 r_{04}=2 →root 為 2 右邊 a3,a4 看 T_{24} →root 為 3 or 4



Knapsack problem

0/1 Knapsack problem ➔Dynamic programming

0/1 problem ,opt cost ? Knapsack capacity=5kg

obj	1	2	3
weight	1 kg	2 kg	3 kg
cost	60	100	120

	0	1	2	3	4	5
0	0	0	0	0	0	0
1 {1}	0	60{1}	60{1}	60{1}	60{1}	60{1}
2 {1,2}	0	60{1}	100{2}	160{1,2}	160{1,2}	160{1,2}
3 {1,2,3}	0	60{1}	100{2}	160{1,2}	180{1,3}	220{2,3}

Fractional Knapsack problem➔ Greedy

Fractional Knapsack problem ,opt cost ? Knapsack capacity=15kg

obj	1	2	3	4	5	6	7
weight	4	3	2	4	2	3	4
cost	3	4	5	6	7	8	9

obj	1	2	3	4	5	6	7
cost/weight	3/4	4/3	5/2	6/4	7/2	8/3	9/4
weight	4	3	2	4	2	3	4

Item 5:2 kg Item 6:3 kg Item 3:2 kg Item 7:4 kg Item 4:4 kg

Cost 7+8+5+9+6=35

LCS Example: <ABCBDAB , BDCABA>

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	**1** ↖	1 ←	1 ↖	1 ←	1 ←	1 ↖
D	0	0	1 ↑	1 ←	1 ←	2 ↖	2 ←	2 ←
C	0	0	1 ↑	**2** ↖	2 ←	2 ←	2 ←	2 ←
A	0	1 ↖	1 ←	2 ↑	2 ↑	2 ↑	3 ↖	3 ←
B	0	1 ↑	2 ↖	2 ←	**3** ↖	3 ←	3 ←	4 ↖
A	0	1 ↖	2 ↑	2 ↑	3 ↑	3 ↑	**4** ↖	4 ←

Longest common substring

Example : <baacc ,abaca>

		b	a	a	c	c
	0	0	0	0	0	0
a	0	0	1 ↖	1 ↖	0	0
b	0	1 ↖	0	0	0	0
a	0	0	2 ↖	1 ↖	0	0
c	0	0	0	0	2 ↖	1 ↖
a	0	0	1 ↖	1 ↖	0	0

<u>Longest Increasing Subsequence</u>

LIS for sequence 5,7,1,6,2,4

Let X=<5,7,1,6,2,4>

Y=sort(X)=<1,2,4,5,6,7>

LCS<X,Y>

		1	2	4	5	6	7
	0	0	0	0	0	0	0
5	0	0 ←	0 ←	0 ←	1 ↖	1 ←	1 ←
7	0	0 ←	0 ←	0 ←	1 ↑	1 ←	2 ↖
1	0	1 ↖	1 ←	1 ←	1 ←	1 ←	2 ↑
6	0	1 ↑	1 ↑	1 ↑	1 ↑	2 ↖	2 ←
2	0	1 ↑	2 ↖	2 ←	2 ←	2 ←	2 ←
4	0	1 ↑	2 ↑	3 ↖	3 ←	3 ←	3 ←

LIS=<1,2,4>

<u>Step 1:Make a sorted copy of the sequence A, denoted as B. O(nlogn) time.</u>

<u>Step 2:Use Longest Common Subsequence on with A and B. $O(n^2)$ time.</u>

<u>Total➔ $O(n^2)$</u>

KMP pattern matching

Time complexity : KMP ➔ O(n+m)   Boyer-Moore algorithm 是 O(nm)

```
f[0]=-1;
    for(j=1;j<n;j++)
    {
     i=f[j-1];
    while ((p[j]!=p[i + 1])&&(i>=0))   i = f[i];
    if (p[j] == p[i + 1])              f[j] = i + 1;
    else                               f[j] = -1;
    }
```

Example 1:   ababbababaa

j	0	1	2	3	4	5	6	7	8	9	10
p	a	b	a	b	b	a	b	a	b	a	a
f	-1	-1	0	1	-1	0	1	2	3	2	0

Convex Hull

Greedy O(nlogn)

對所有點相對於這個 s 點計算角度儲存起➔O(n)

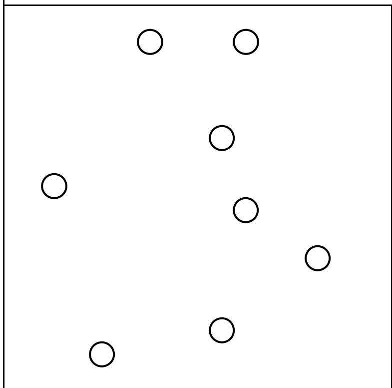
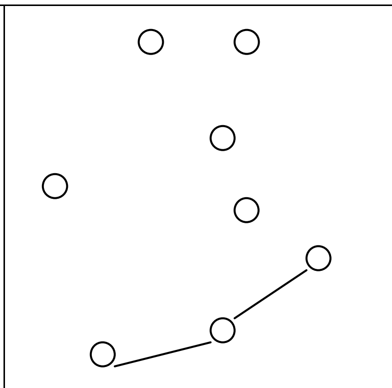
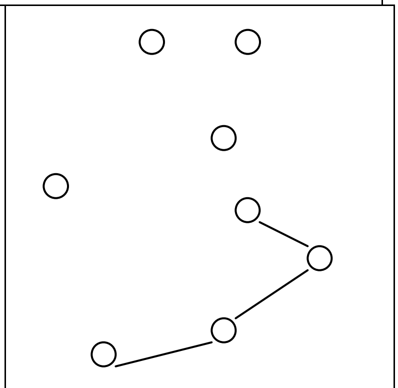
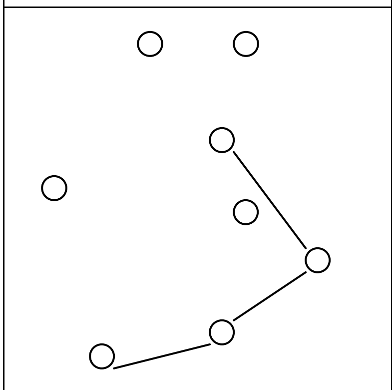
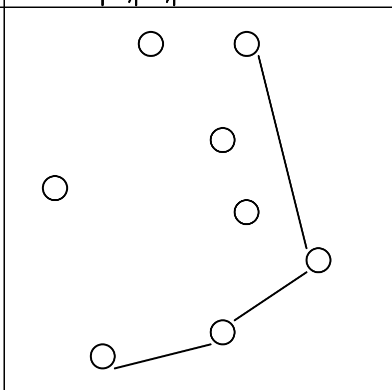
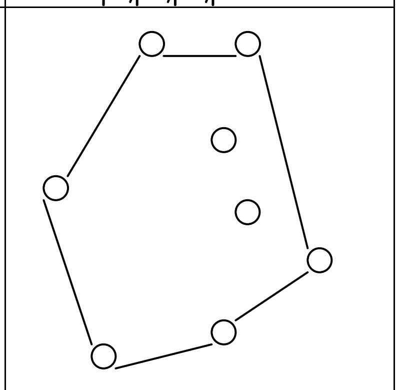
根據角度排序 =>O(nlogn)

連續 3 個點來決定這角度是向外凹 or 內凹，外凹就丟掉➔O(n)

Overall cost 是 O(nlogn)

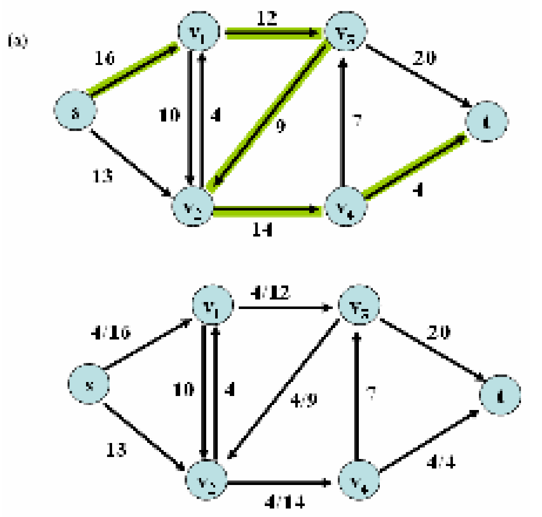
Graham-Scan

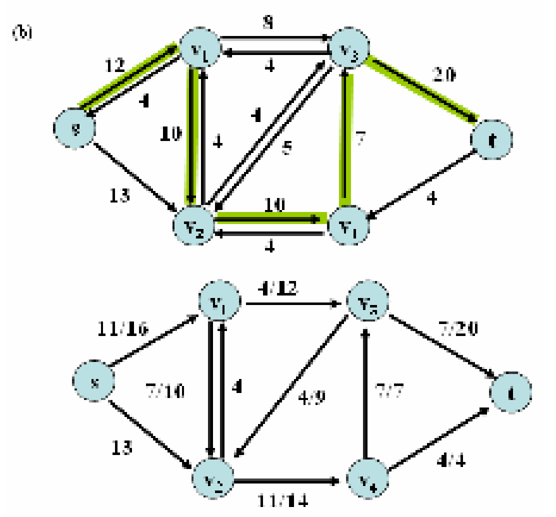
```
Graham-Scan(Q)
令 p0 為 Q 中最低的點， 或最低的點中最左邊的;
令〈p1, p2, … , pm〉為 Q 中剩下的點，並已根據極角依逆時針方向排序好(以 p0 為極點);
Push(p0, S); //S 為 Stack
Push(p1, S);
Push(p2, S);
    for i ← 3 to m do
        while (從 next-to-top(S) 到 top(S) 再到 pi 為右彎)
            do Pop(S) ;
        Push(pi, S);    //即為 6 點鐘方向到 12 點,左半圓
return S;
```

		
	Stack:p1,p2,p3	Stack:p1,p2,p3,p4
		
Stack:p1,p2,p3,p5	Stack:p1,p2,p3,p6	Stack:p1,p2,p3,p6,p7,p8

Flowing Network

Ford-Fulkerson

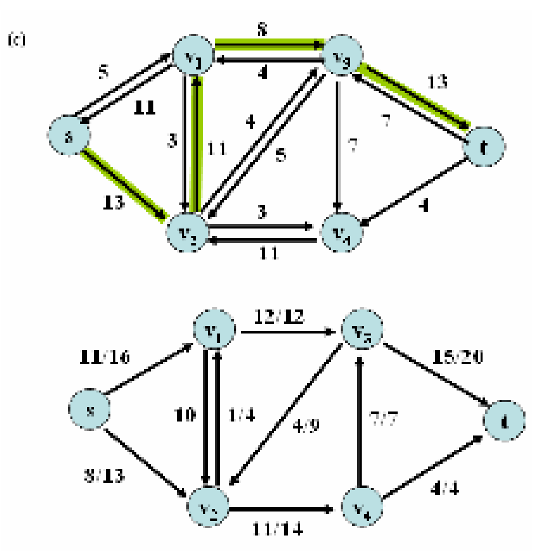


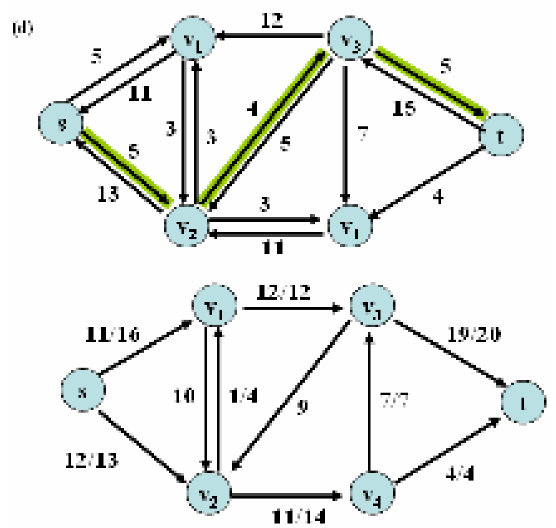


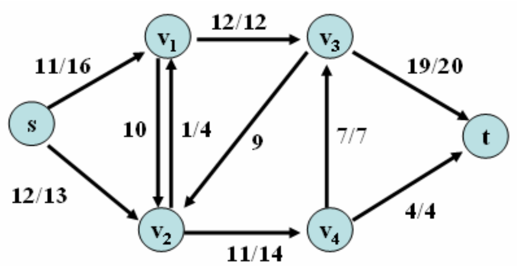
v 1,v2 往下流 7,我們想往上流 8,

等效於往下流 0,往上流 1

全不部爲 14,所以可再往上 11







k^{th} Selection

Prune-and-Search O(n)

Input: A set S of n elements. Output: The kth smallest element of S. Step 1: 將 S 分成 $\lceil n/5 \rceil$ 組資料集合，每一組有 5 個資料，不足 5 個資料以 ∞ 補足。 Step 2: 排序每一組資料 Step 3: 找出所有組中位數的中位數 Step 4: 將 S 區分成三部份 S_1, S_2 and S_3, which contain the elements less than, equal to, and greater than p, respectively. Step 5: 利用三個判斷條件以找出第 k 小的元素: If $\lvert S_1 \rvert \geq k$, then 第 k 小的元素**存在於** S_1，prune away S_2 and S_3。 else, if $\lvert S_1 \rvert + \lvert S_2 \rvert \geq k$, then **p** **即爲第 k 小的元素**。 else,第 k 小的元素**存在於** S_3 **中**，prune away S_1 and S_2。令 $k' = (k - \lvert S_1 \rvert - \lvert S_2 \rvert)$，在 S_3 中找第 k'個元素即爲解答

Time complexity: T(n) = O(n)
step 1: O(n) //掃一輪即可得知
step 2: O(n) //有 $\lceil n/5 \rceil$ 組資料，每組資料排序需固定常數時間 O(1)
step 3: T(n/5) //採遞迴方式找尋，共有 $\lceil n/5 \rceil$ 組
step 4: O(n) //掃一輪即可得知
step 5: T(3n/4) //每次 Prune 掉至少 n/4 資料量後，尚有 3n/4 左右的剩餘資料需遞迴執行
遞迴方程式爲 $T(n) = T(3n/4) + T(n/5) + O(n)$，採遞迴樹法分析，可得知此演算法的時間複雜度爲 O(n)

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);                                   //O(V)
    sort E by edge weight w                           //O(ElogV)
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)                    //O(E)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));            //O(logV)
}
```

```
MST-Prim(G, w, r)
{
     Q = V[G];                                            //O(V)
     for each u ∈ Q
          key[u] = ∞;
     key[r] = 0;
     p[r] = NULL;          //p[]: parent of this node
     while (Q not empty)
          u = ExtractMin(Q);                              //O(VlogV)
          for each v ∈ Adj[u]
               if (v ∈ Q and w(u,v) < key[v])
                    p[v] = u;
                    key[v] = w(u,v);                      //O(ElogV)
}
```

```
DFS(G)
{
    for each vertex u ∈ G->V
    {
        color [u] = WHITE;
    }
    time = 0;
    for each vertex u ∈ V [G]
    {
        if (color [u]== WHITE)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    color [u] = GREY;
    d[u] = ++time;
    for each v ∈Adj[u]
    {
        if (color [u]== WHITE)
            DFS_Visit(v);
    }
    color [u]= BLACK;
    f[u]   = ++time;
}
```

```
BellmanFord()
    for each v ∈ V              d[v] = ∞;           //O(v)
    d[s] = 0;
    for i=1 to |V|-1
            for each edge (u,v) ∈ E
                  if (d[v] > d[u]+w(u,v)) then d[v]=d[u]+ (u,v)      //O(VE)
    for each edge (u,v) ∈ E                                          //O(E)
        if (d[v] > d[u] + w(u,v))
              return "FALSE";
```

```
Dijkstra(G)
    for each v ∈ V        d[v] = ∞;
    d[s] = 0;
    S = ∅; Q = V;
    while (Q ≠ ∅)
        u = ExtractMin(Q);
        S = S U {u};
        for each v ∈ Adj[u]
            if (d[v] > d[u]+w(u,v)) then d[v] = d[u]+w(u,v);
linear array→O(V^2)
binary heap for Q →O(E log V)
Fibonacci heaps for Q → O(V log V + E)
```

```
Floyd-Warshall(G,W)
{      n←|V|;D^(0)←W;
       for k = 1 to n do
          for i = 1 to n do
             for j = 1 to n do
                  if D^(k-1)[i,j]>D^(k-1)[i,k]+D^(k-1)[k,j] then
                          D^(k)[i,j]←D^(k-1)[i,k]+D^(k-1)[k,j] ;   π[i,j] ←π[k,j];
                  else D^(k)[i,j]←D^(k-1)[i,j]
       return   D^(n)
}

O(n^3)
```

```
LCS-Length(X,Y)
{
        m ← length[X];
        n ← length[Y];
        for i= 1 to m do c[i,0] ← 0;
       for j= 1 to n do c[0,j] ← 0;
                  for i= 1 to m do
                   for j= 1 to n do
                     {
                             If xi = yj then
                                    c[i,j] = c[i-1,j-1]+1;   b[i,j] = "↖";
                            else if c[i-1,j] ≥ c[i,j-1]
                                       then { c[i,j] = c[i-1,j]; b[i,j] = "↑"; }
                            else { c[i,j] = c[i,j-1]; b[i,j] = "←"; }
                  }
        return b, c
}
```

```
Print-LCS(b, X, i, j)
{
              If i=0 or j=0 then return;
              If b[i,j] = "↖ ";
                     then
                     {
                            Print-LCS(b, X, i-1, j-1);
                            print xi;
                     }
              else if b[i,j] = "↑"
                     then Print-LCS(b, X, i-1, j);
              else Print-LCS(b, X, i, j-1);
}
```

```
Topological Sort fail when graph contains cycle
Topological-Sort()
{
    Run DFS
    When a vertex is finished, output it
    Vertices are output in reverse topological order
}
```

adjacency matrix: $O(V^2)$, adjacency lists :O(V+E)

SCC
{

1. 呼叫 DFS(G)對所有點 u，計算出 f[u]，即 finishing time。
2. 計算出 GT，即點集合與 G 相同，而邊連接方向相反的圖。
3. 呼叫 DFS(GT)，但在 DFS 主迴圈中，選擇點的順序是先挑取 f[u]值較大的點 u。
4. 在 DFS(GT)的 Depth-first forest 中，每一個樹均是一個 Strongly connected component。

}

96 清大資工

Determine the largest number and smallest number in n number,which operation are (3n/2) - 2
Algorithm LARGESMALL Step 1:把 n 個 element 分成左右 2 半, $(a_1, a_{\frac{n}{2}+1}), (a_2, a_{\frac{n}{2}+2}), ..., (a_{\frac{n}{2}}, a_n)$ 比較 小的放左邊,大的放右邊 Step 2:左半找最小 $for(i = 2, smallest = key[1], i <= \frac{n}{2}, i++)$ If(key[i] < smallest) smallest= key[i] Step 2:右半找最大 **Time complexity**: (3n/2) - 2, Step 1: n/2 comparisons Step 2: (n/2) - 1 comparisons Step 3:(n/2)-1 comparisons.

95 清大資工

We have a directed graph G=(V,E),represented using adjacent list .the edge costs are integers in range {1,2,3,4,5},assume that G has no self-loops or multiple edge .Design a algorithm that solve the single-source shortest path problem in O(|V|+|E|)

DAG-Shortest-Path(G,w,s)
{ Topologically sort V[G]
for each v ∈ V
d[v] = ∞;
d[s] = 0;
for each u taken in topological order
do for each v∈adj[u]
if (d[v] > d[u]+w(u,v)) then d[v] = d[u]+w(u,v);
}
以上演算法僅需 O(|V|+|E|)的時間

92 nthu 93 師大資教

Let a graph be dented as $G = (V, E)$. Discuss how to test if the graph is connected in $O(|V| + |E|)$

DFS connected component algorithm
DFS(G)
for each vertex u∈V[G] { do color[u]←WHITE ; π[u] ←NIL }
time=0
Tree_count=0;
for each vertex u∈V[G]
if (color[u] ==WHITE) **{** **Tree_count++ ;** DFS-Visit(u); **}**
If (Tree_count==1) return true
Else return False
DFS-VISIT(u)
color[u]=GRAY //u has just been discovered
d[u]=++time
cc[u]=Tree_count; //cc[u]為 node u 所屬的 connected component number
for each v∈Adj[u]
{
if color[v]=WHITE {π[v] ←u ; DFS-Visit(v) ; }
}
color[u]←black
f[u]=++time
connected component algorithm time complexity:
adjacency matrix :$O(n^2)$ adjacency lists : $O(n+e)$

92 ncku

Show that the second smallest of n elements can be found with $n+\lceil \log n \rceil-2$ comparisions in the worst case
n 個 element,比較 n-1 次就能找到最小的 element 第 2 小的 element,一定是剛才與最小的 element 所比的輸家其中一位 而最小的 element 只會比 $\lceil \log n \rceil$ 次(樹高),因爲 n 個 element 樹高不會超過$\lceil \log n \rceil$ 則 $\lceil \log n \rceil$個 element 再找出最小的需比$\lceil \log n \rceil$-1 次 所以總共爲$n-1+\lceil \log n \rceil-1=n+\lceil \log n \rceil-2$

94 ntu,91 ncku

describe how to use depth first search to determine whether input direct graph G=(V,E) is acyclic
a directed graph G is acyclic iff a DFS of G yields no back edges

```
DFS-Acyclic(G)
{
    for each vertex u∈V[G] { do color[u]←WHITE  ;  π[u] ←NIL  }
    time=0
    for each vertex u∈V[G]
    {
        if (color[u] ==WHITE) DFS-Visit(u);
    }
Return false;
}
```

```
DFS-VISIT(u)
color[u]=GRAY //u has just been discovered
d[u]=++time
for each v∈Adj[u]
{
  if color[v]=WHITE {π[v] ←u ; DFS-Visit(v) ; }
  elseif   color[v] =Gray  Halt and Return True
}
color[u]←black
f[u]=++time
```

93 師大資教

Design an algorithm to test whether a given graph is bipartite on not

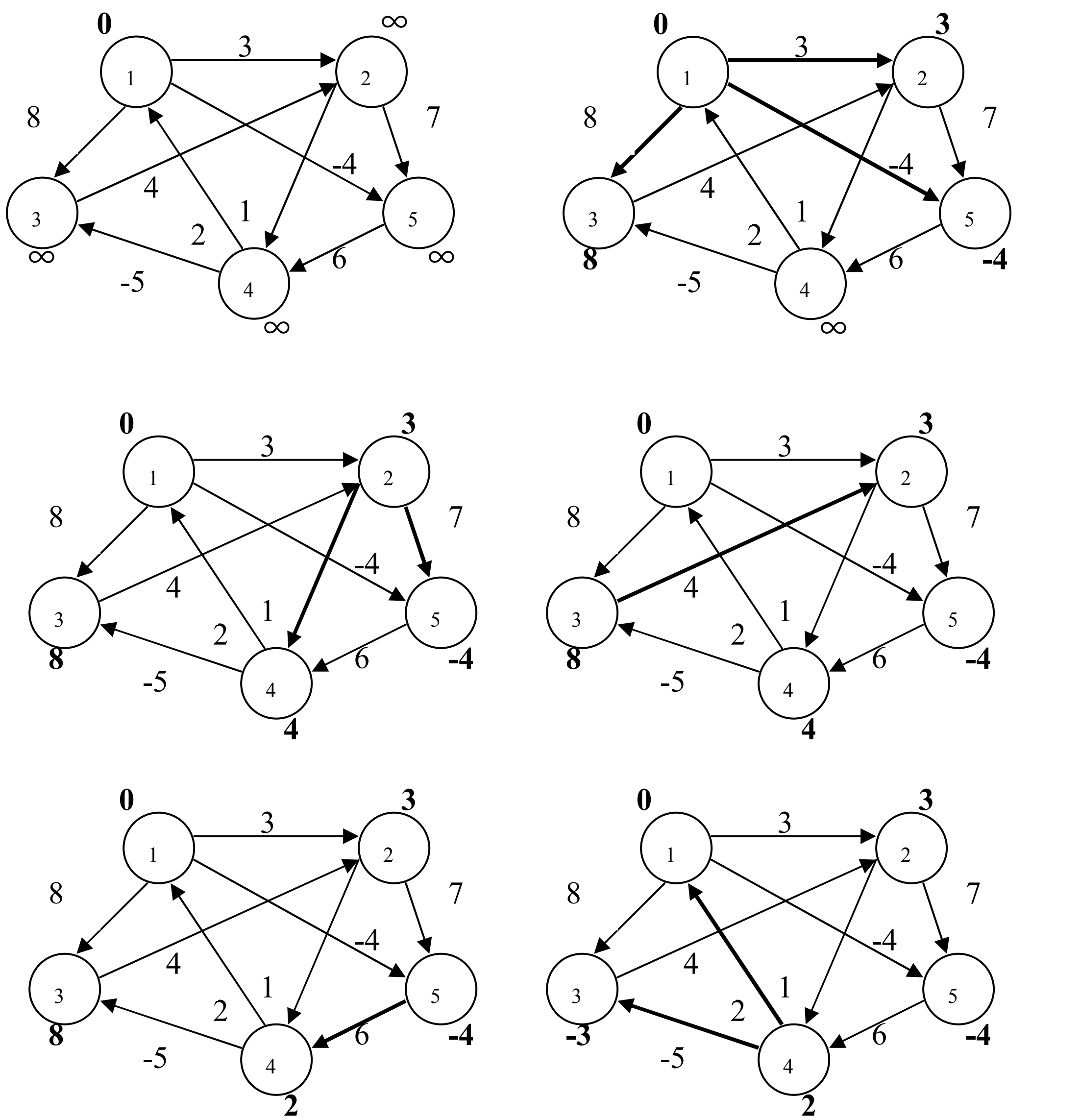
boolean visit:false →未尋訪 true →尋訪過

int mark: 0→沒集合 ,1→位於 set1 ,2→位於 set2

```
bipartite(v,type)
    visit[v]=true
    foreach u∈adj[v]
        if  (mark[u]==0)mark[u]=3-type
        else if (mark[u]==type)  return false
        if  ( ! visit[u])
            if ( ! bipartite (u,3-type) return false
    return true
```

Bellman Ford algorithm example

$\begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$	**Bellman Ford** find a shortest path from vertex 1 to vertex 3.



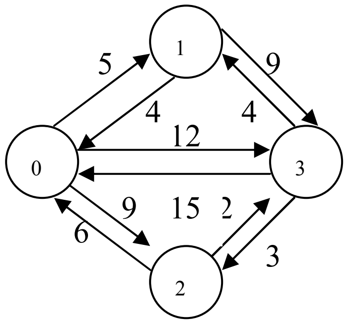
(1) 從 **node 1** 開始

(2) node 2, node 3, node 5

(3) node 4

Floyd-Warshall　example 96 彰師數位

Find the all pair shortest path of following graph?



$$D=\begin{bmatrix}0&5&9&12\\4&0&\infty&9\\6&\infty&0&2\\15&4&3&0\end{bmatrix},D^0=\begin{bmatrix}0&5&9&12\\4&0&13&9\\6&11&0&2\\15&4&3&0\end{bmatrix},D^1=\begin{bmatrix}0&5&9&12\\4&0&13&9\\6&11&0&2\\8&4&3&0\end{bmatrix}$$

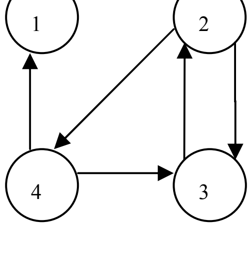
$$D^2=\begin{bmatrix}0&5&9&11\\4&0&13&9\\6&11&0&2\\8&4&3&0\end{bmatrix},D^3=\begin{bmatrix}0&5&9&11\\4&0&12&9\\6&6&0&2\\8&4&3&0\end{bmatrix}$$

$$D^0\Rightarrow\begin{cases}(1,2)=\min\{(1,2),(1,0)+(0,2)\}=13\\(2,1)=\min\{(2,1),(2,0)+(0,1)\}=11\end{cases},D^1\Rightarrow(3,0)=\min\{(3,0),(3,1)+(1,0)\}=8$$

$$D^2\Rightarrow(0,3)=\min\{(0,3),(0,2)+(2,3)\}=11,D^3\Rightarrow\begin{cases}(1,2)=\min\{(1,2),(1,3)+(3,2)\}=12\\(2,1)=\min\{(2,1),(2,3)+(3,1)\}=6\end{cases}$$

如果點爲 0~3 則 $D^0\sim D^3$,如果點爲 1~4 則 $D^1\sim D^4$

Find trsnsitive closure use Floyd-Warshall?



$$A_0=\begin{bmatrix}1&0&0&0\\0&1&1&1\\0&1&1&0\\1&0&1&1\end{bmatrix},A_1=\begin{bmatrix}1&0&0&0\\0&1&1&1\\0&1&1&0\\1&0&1&1\end{bmatrix},A_2=\begin{bmatrix}1&0&0&0\\0&1&1&1\\0&1&1&1\\1&0&1&1\end{bmatrix}$$

$$,A_3=\begin{bmatrix}1&0&0&0\\0&1&1&1\\0&1&1&1\\1&1&1&1\end{bmatrix},A_4=\begin{bmatrix}1&0&0&0\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix}$$

$(1)T(n)=4T(\frac{n}{4})+n\log n \quad (2)T(n)=T(n-1)+\frac{1}{n}$ $(3)T(n)=T(\frac{n}{2})+1 \quad (4)T(n)=T(n-1)+\lg n$ $(5)T(n)=2T(\sqrt{n})+\log n \quad (6)T(n)=4T(\frac{n}{4})+\frac{n}{\log n}$ $(7)T(n)=4T(\frac{n}{4})+\frac{n}{\log^2 n} \quad (8)T(n)=4T(\frac{n}{4})+\frac{n}{\log^3 n}$ $(9)T(n)=\sqrt{n}T(\sqrt{n})+n \quad (10)T(n)=\sqrt{n}T(\sqrt{n})+\sqrt{n}$
$(1)O(n\log^2 n) \quad (2)O(n\log n) \quad (3)O(\log n)$ $(4)T(n)=T(n-1)+\lg n=T(n-2)+\lg(n-1)+\lg n$ $=T(1)+...+\lg(n-1)+\lg n\le \lg n+\lg n+...+\lg n=O(n\lg n)$ $(5)let \quad n=2^{2^k},F(k)=2F(k-1)+2^k=2^2F(k-1)+(2^k+2^k)$ $=2^kF(1)+k(2^k)=\lg n+(\lg\lg n)\log n=O(\lg n\lg\lg n)$ $(6)let \quad n=4^k,F(k)=4F(k-1)+\frac{4^k}{k}=4^2F(k-2)+\frac{4^k}{k-1}+\frac{4^k}{k}$ $=4^kF(0)+4^k(1+\frac{1}{2}+....+\frac{1}{k})=n+n\log\log n=O(n\log\log n)$ $(7)let \quad n=4^k,F(k)=4F(k-1)+\frac{4^k}{k^2}=4^2F(k-2)+\frac{4^k}{(k-1)^2}+\frac{4^k}{k^2}$ $=4^kF(0)+4^k(1^2+\frac{1}{2^2}+....+\frac{1}{k^2})=O(n)$ $(8)let \quad n=4^k,F(k)=4F(k-1)+\frac{4^k}{k^3}=4^2F(k-2)+\frac{4^k}{(k-1)^2}+\frac{4^k}{k^3}$ $=4^kF(0)+4^k(1^3+\frac{1}{2^3}+....+\frac{1}{k^3})=O(n)$ $(9)T(n)=\sqrt{n}T(\sqrt{n})+n$ $T(n)=n^{\frac{1}{2}}T(n^{\frac{1}{2}})+n=n^{\frac{1}{2}}(n^{\frac{1}{4}}T(n^{\frac{1}{4}})+n^{\frac{1}{2}})+n=n^{\frac{1}{2}+\frac{1}{4}}T(n^{\frac{1}{4}})+n+n$ $=........=n^{1-\frac{1}{2^k}}T(n^{\frac{1}{2^k}})+kn\Rightarrow k=\theta(\lg\lg n)\Rightarrow T(n)=\theta(n\lg\lg n)$ $(10)T(n)=\sqrt{n}T(\sqrt{n})+\sqrt{n}$ $T(n)=n^{\frac{1}{2}}T(n^{\frac{1}{2}})+\sqrt{n}=n^{\frac{1}{2}}(n^{\frac{1}{4}}T(n^{\frac{1}{4}})+n^{\frac{1}{2}})+\sqrt{n}=n^{\frac{1}{2}+\frac{1}{4}}T(n^{\frac{1}{4}})+\sqrt{n}+\sqrt{n}$ $=........=n^{1-\frac{1}{2^k}}T(n^{\frac{1}{2^k}})+k\sqrt{n}\Rightarrow k=\theta(\lg\lg n)\Rightarrow T(n)=\theta(\sqrt{n}\lg\lg n)$

$O(g(n))$ {f(n):there exist positive constants c, n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$}
$\Omega(g(n))$ {f(n):there exist positive constants c, n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0$}
$\theta(g(n))$ {f(n):there exist positive constants c_1, c_2, n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$}
$o(g(n))$ {*f*(*n*) : for any positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 < f(n) < cg(n)$ for all $n \ge n_0$}.
$\omega(g(n))$ {*f*(*n*) : for any positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 < cg(n) < f(n)$ for all $n \ge n_0$}.

Prove $(1) f(n) = 3n + 8 \Rightarrow f(n) = O(n)$ $(2) f(n) = 2n^2 + 4n + 3 \Rightarrow f(n) = O(n^2)$ $(3) f(n) = 3n + 2 \Rightarrow f(n) = O(1)?$ $(4) f(n) = 2n^2 + 3n - 9 \Rightarrow f(n) = \Omega(n^2)$ $(5) f(n) = 3n^2 + 6n - 12 \Rightarrow f(n) = \theta(n^2)$
(1) 我們可以找到 $c = 4, n_0 = 8$ 使得 $3n + 8 \le cn, \forall n \ge n_0$ (2) 我們可以找到 $c = 3, n_0 = 5$ 使得 $2n^2 + 4n + 3 \le cn^2, \forall n \ge n_0$ $ps: 2n^2 + 4n + 3 \le 3n^2 \Rightarrow n^2 - 4n - 3 \ge 0 \Rightarrow n_0 = 5$ (3) $3n + 2 \le c * 1$, 我們找不到 c 是常數, $\forall n \ge n_0 \Rightarrow f(n) = O(1)$ wrong (4) 我們可以找到 $c = 2, n_0 = 3$ 使得 $2n^2 + 3n - 9 \ge cn^2, \forall n \ge n_0$ (5) 我們可以找到 $c_1 = 3, c_2 = 4, n_0 = 2$ 使得 $c_1 n^2 \le 3n^2 + 6n - 12 \le c_2 n^2, \forall n \ge n_0$

Prove $\log(n!) = \theta(n \log n)$

$(O)\ \log(n!) = \log n + \log(n-1) + ... + 2 + 1 < \log n + \log n + ... + \log n + \log n = O(n \log n)$

$(\Omega)\ \log(n!) = \log n + \log(n-1) + ... \log\frac{n+1}{2} + \log\frac{n}{2} + ... + 2 + 1$

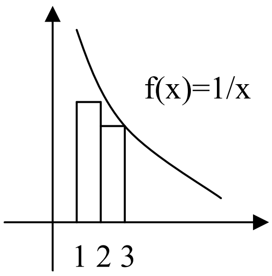
$\geq \log n + \log(n-1) + ... \log\frac{n+1}{2} \geq \log\frac{n}{2} + \log\frac{n}{2} + ... + \log\frac{n}{2} \geq \frac{n}{2}\log\frac{n}{2} = \Omega(n \log n)$

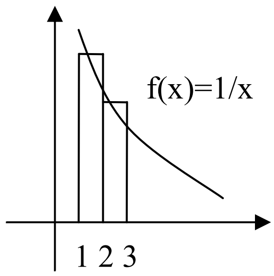
$\therefore \log(n!) = \theta(n \log n)$

$T(n) = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$ prove $T(n) = \theta(\log n)$

$block\ \ area\ \ A_1 = \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} = T(n) - 1$

$A_1 < \int_1^n \frac{1}{x} dx = \ln x - \ln 1 = \ln x \Rightarrow T(n) - 1 < \ln x \equiv T(n) < \ln x + 1 = O(\ln n)$





$block\ \ area\ \ A_2 = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} = T(n)$

$A_2 > \int_1^n \frac{1}{x} dx = \ln x - \ln 1 = \ln x \Rightarrow T(n) > \ln x = \Omega(\ln n)$

$\because T(n) = \Omega(\ln n), T(n) = O(\ln n) \Rightarrow T(n) = \theta(\ln n)$

(4)rank time complexity

$$(\lg n)! < \frac{n^2}{\log n} < n^2 = 4^{\lg n} < \lg(n!) < \left(\lg n\right)^{\lg n} = n^{\lg n \lg n} < n^{\log n} < 2^n < n! < n^{0.0001n}$$

(5)Master Theorem

$$T(n) = aT(\frac{n}{b}) + f(n)$$

$$(a) if \quad f(n) = O(n^{\log_b a - \varepsilon}), for \quad some \quad \varepsilon > 0 \Rightarrow T(n) = \theta(n^{\log_b a})$$

$$(b) if \quad f(n) = n^{\log_b a} \Rightarrow T(n) = \theta(n^{\log_b a} \log n)$$

$$(c) if \quad f(n) = \Omega(n^{\log_b a + \varepsilon}), for \quad some \quad \varepsilon > 0, af(\frac{n}{b}) \le cf(n), c < 1$$

$$\Rightarrow T(n) = \theta(f(n))$$

Master theorem solve

(a) $T(n) = 7T(\frac{n}{2}) + n^2$ (b) $T(n) = 3T(\frac{n}{2}) + n^2$ (c) $T(n) = 4T(\frac{n}{2}) + n^2$

$by \quad master \quad theorem$

(a) $a = 7, b = 2 \Rightarrow n^{\log_b a} = n^{\lg 7}, f(n) = n^2$

$pick \quad f(n) = n^2 = O(n^{\lg 7 - \varepsilon}) \Rightarrow T(n) = \theta(n^{\lg 7})$

$a = 3, b = 2 \Rightarrow n^{\log_b a} = n^{\lg 3}, f(n) = n^2$

(b) $pick \quad \varepsilon = 2 - \lg 3 \quad \Rightarrow f(n) = n^2 = \Omega(n^{\lg 3 + \varepsilon})$

$\Rightarrow T(n) = \theta(n^2)$

$by \quad master \quad theorem$

(c) $a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2, f(n) = n^2$

$and \quad f(n) = \theta(n^2) \Rightarrow T(n) = \theta(n^2 \log n)$

exercises Chapter 4.1

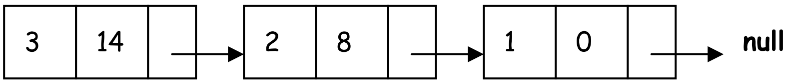
$(a)T(n)=T(\sqrt{n})+1\quad (b)T(n)=5T(\frac{n}{5})+\frac{n}{\lg n}\quad (c)T(n)=2T(\frac{n}{2})+\frac{n}{\lg n}$ $(d)T(n)=T(n-1)+\frac{1}{n}\quad (e)T(n)=T(n-1)+\lg n\quad (f)T(n)=\sqrt{n}T(\sqrt{n})+n$
$(a)T(n)=T(\sqrt{n})+1\ ,\ let\ \ m=\lg n\Rightarrow S(m)=T(2^m)$ $T(2^m)=T(2^{m/2})+1\Rightarrow S(m)=S(m/2)+1=\theta(\lg m)$ $\Rightarrow T(n)=\theta(\lg\lg n)$ $(b)let\ \ n=5^k\Rightarrow T(k)=5T(k-1)+\frac{5^k}{k}=5^2T(k-2)+\frac{5^k}{k-1}+\frac{5^k}{k}$ $=......=5^k(1+\frac{1}{2}+...+\frac{1}{k})=n(1+\frac{1}{2}+...+\frac{1}{\log n})=O(n\log\log n)$ $(c)let\ \ n=2^k\Rightarrow T(k)=2T(k-1)+\frac{2^k}{k}=2^2T(k-2)+\frac{2^k}{k-1}+2\frac{5^k}{k}$ $=......=2^k(1+\frac{1}{2}+...+\frac{1}{k})=n(1+\frac{1}{2}+...+\frac{1}{\log n})=O(n\log\log n)$ $(d)T(n)=T(n-1)+\frac{1}{n}=\frac{1}{1}+\frac{1}{2}+...+\frac{1}{n}=O(\lg n)$ $(e)T(n)=T(n-1)+\lg n=T(n-2)+\lg(n-1)+\lg n=.....$ $=\lg 2+\lg 3+....+\lg n\le\lg n+\lg n+.....+\lg n=O(n\lg n)$ $(f)T(n)=\sqrt{n}T(\sqrt{n})+n$ $T(n)=n^{\frac{1}{2}}T(n^{\frac{1}{2}})+n=n^{\frac{1}{2}}(n^{\frac{1}{4}}T(n^{\frac{1}{4}})+n^{\frac{1}{2}})+n=n^{\frac{1}{2}+\frac{1}{4}}T(n^{\frac{1}{4}})+n+n$ $=........=n^{1-\frac{1}{2^k}}T(n^{\frac{1}{2^k}})+kn$ $\because k=\theta(\lg\lg n)\Rightarrow T(n)=\theta(n\lg\lg n)$

(1) $T(n)=4T(\frac{n}{2})+\frac{n^2}{\log n}=\theta(n^2\lg\lg n)$

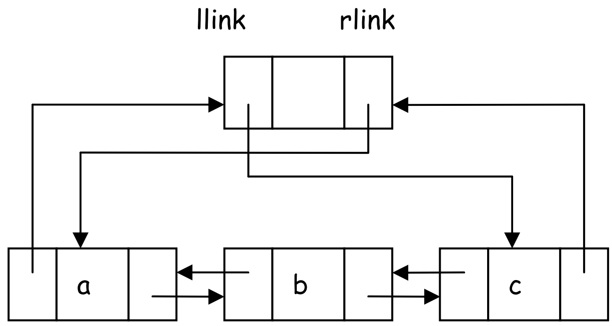
(2) $T(n)=4T(\frac{n}{2})+\frac{n^2}{\log^2 n}=\theta(n^2)$

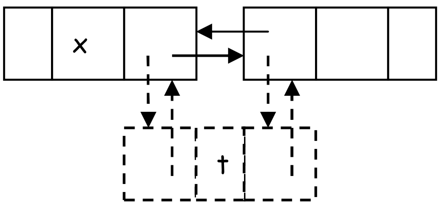
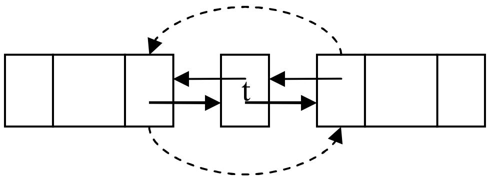
(3) $T(n)=4T(\frac{n}{2})+\frac{n^2}{\log^3 n}=\theta(n^2)$

(6) link list for polynormal

$3x^{14}+2x^{8}+1$


(7) double link list

An example for three element double link list


Insert node t after node x	**Delete node t**
New x t→llink =(x→rlink)→llink t→rlink =x→rlink (x→rlink)→llink=t x→rlink=t  Single link list insert 只需改 2 個 pointer double link list insert 需改 4 個 pointer	(t→rlink)→llink= t→llink (t→llink)→ rlink = t→rlink Free t 

Chapter 3 Stack

(1)Stack operation

(a) Array   Array[1~n]

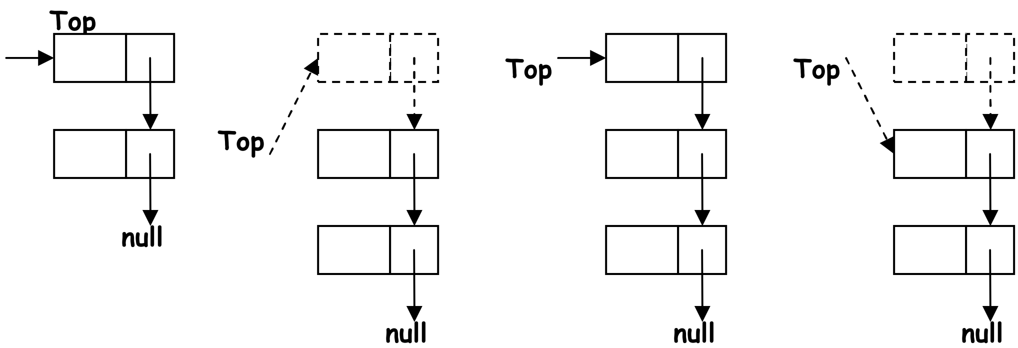
Boolean IsFull(S) if (top>=MAX_STACK_SIZE-1) return true; else return false;	Boolean IsEmpty(S) if (top<0) return true; else return false;

void push (S, element item)   ***O(1)***
if (top ==n)   stack_full();
else stack[++top]=item;
	element pop (S) ***O(1)***
	if (top == 0) return stack_empty();
	return stack[top--];

(b) Link list

Boolean IsFull(S) ➔no Need	Boolean IsEmpty(S) if (top==null) return true; else return false;

void push (S, element item)
New t ;
t -> data=item ;
t -> link=top ;
top=t ;
	element pop (S)
	if (IsEmpty(S)) return "S is empty"
	else
	t= top;
	item=top-> data;
	top=top -> link ;
	free (t) ;



(2)Queue operation

Insert rear

Delete front

(a)

Method 1: Array　　Array[1~n]

缺點:front!=0 並不代表 Queue 是滿的

解決: 把 front+1 到 rear 左移 front 格 ,rear=rear-front ,front=0

Boolean IsFull if (rear == n)　return true; else return false;	Boolean IsEmpty if (front == rear)　return true; else return false;

void addq (Q, element item)　**O(n)**
if (rear == n)　queue_full();
else　queue[++rear] = item;
	element deleteq (Q)　**O(1)**
	if (front == rear) return queue_empty();
	else return queue[++front];

Method 2:circular array　改善 linear array 碰到 array 底需全部左移

只利用 n-1 個空間,因如果用 front 放 data

當 front == rear 無法判斷 queue 爲空或爲滿

queue 爲空或 queue 爲滿條件式皆爲　rear == front

void addq (Q, element item)　**O(1)**
rear=(rear+1)%n
if (rear == front) queue_full();
else　queue[rear] = item;
	element deleteq (Q)　**O(1)**
	if (front == rear) return queue_empty();
	else
	front=(front +1)%n
	return queue[front];

Method 3:circular array

可充份利用 n 個空間

若 tag=true 表示 queue 是滿

因 add queue 或 delete queue 都多了判斷 tag,所以比 Method 2 耗時

void addq (Q, element item)　**O(1)**
if (rear == front&& tag==true)
queue_full();
else
rear=(rear+1)%n
if(rear==front) tag=true
queue[rear] = item;
	element deleteq (Q)　**O(1)**
	if (front == rear&& tag==false)
	queue_empty();
	else
	front=(front +1)%n
	if(rear==front) tag=true
	return queue[front];

(b) Link list

Method 1 :single link list

Boolean IsFull(S) ➔no Need	Boolean IsEmpty(S)
	  if (top==null) return true;
	  else return false;

void addq (Q, element item)
  New t ;
  t -> data=item ;
  t -> link=null ;
  if(rear==null) then front =t;
  else rear -> link=t ;
  rear =t;
	element deleteq (Q)
	  if (front==null) return "Q is empty"
	  else
	    t= front;
	    item= front -> data;
	    front = front -> link ;
	    free (t) ;

Method 2 :circular link list

void addq (Q, element item)
  New t ;
  t -> data=item ;
  t -> link=null ;
  if(rear==null) then **t->link=t;**
  else
        t -> link= rear -> link ;
        rear -> link =t;
  rear=t ;
	element deleteq (Q)
	  if (front==null) return "Q is empty"
	  else
	    t=rear->link;
	    item= (rear->link)->data;
	    rear->link = (rear->link) -> link ;
	    free (t)
	   free (t) ;

Postfix evaluation:由左而右掃描 **stack** 先拉出來的 **operand** 擺後面
Prefix evaluation:由右而左掃描 **stack** 先拉出來的 **operand** 擺前面

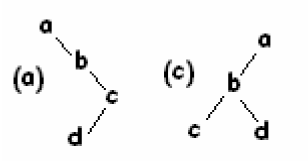
(1)Prefix evaluation of +-/623*42 (2)postfix of 62/3-42*+

		6						
		2		3				
4		3		3		0		
2	*	8	/	8	-	8	+	8

				2				
6		3		4		8		
2	/	3	-	0	*	0	+	8

判斷合法 stack permutation 方法

前序爲 sort abcdef… 中序爲答案選項 ,畫樹,檢查前序是否與樹相同

<table>
<tr><td>Which of following are not stack permutation ?
(a)abdc (b)dcab (c) cbda (d)dacb (e)cadb</td></tr>
<tr><td>(b)無法造出前序爲 abcd,中序爲 dcab 之二元樹(造出來樹之前序不爲 abcd)
(d)無法造出前序爲 abcd,中序爲 dacb 之二元樹(造出來樹之前序不爲 abcd)
(e) 無法造出前序爲 abcd,中序爲 cadb 之二元樹(造出來樹之前序不爲 abcd)

</td></tr>
</table>

(a)tree:樹不可爲空

4 條件等價: (1)G 爲非退化樹 (2)G 中任兩點有唯一路徑

(3)去掉任 1 邊爲不連通 (4)G 加入一邊存在唯一 cycle

3 條件等價: (1)G 爲樹 (2)G 不含迴路且|E|=|V|-1 (3)G 爲連通且|E|=|V|-1

(b)forest :n 棵互斥樹之集合 $n \geq 0 \Rightarrow$ 森林可爲空

(c)compare of tree and binary tree

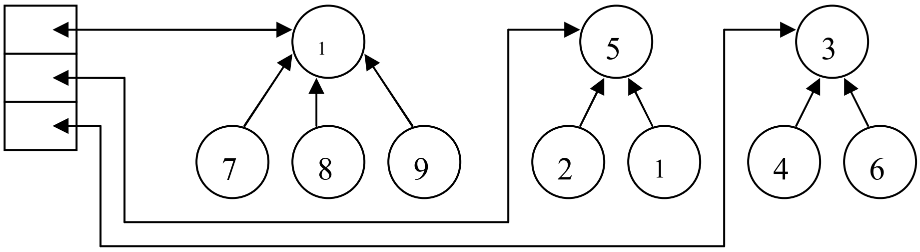
樹不可爲空 child 間無順序 $degree \geq 0$

二元樹可爲空 child 間有順序 $0 \leq degree \leq 2$

(16)Disjoint Set

S1={0, 6, 7, 8}, S2={1, 4, 9}, S3={2, 3, 5}

表示法 1:Link list represent



表示法 2:Array represent

data	1	2	3	4	5	6	7	8	9	10
parent	0	5	0	3	0	3	1	1	1	5

Operation:

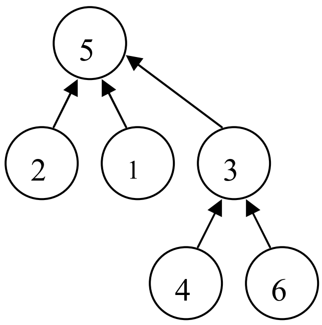
(1) Union(i,j):聯集 Seti 與 Setj

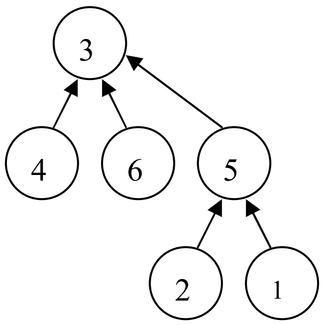
(2) Find(x):找出 x 位於那個 Set

(a)Union and simple find

Union:O(1)　作法:Parent[i]=j 或 Parent[j]=i

Union(S2,S3): 2 種方法





Find: O(k),k 為樹高　作法:延 x 往上走,直到 root

Time complexity:

Union:O(1), Find: O(n)

M 個 Union/ Find ➔O(mn)

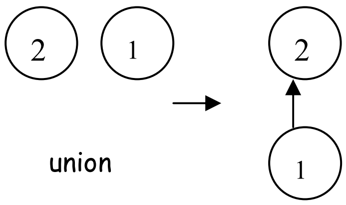
(b)Union-by-height and simple find

Union-by-height:樹高較高的 set 當樹根

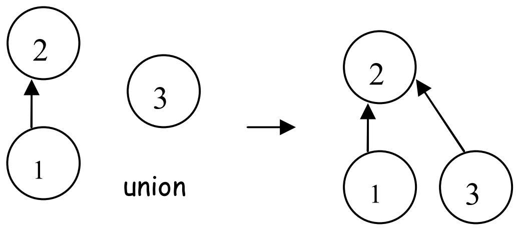
作法:樹高相同作 union,樹高才加 1,

樹高不同 union,樹高不變

例(1):



例(2):



例(3):binomial tree

Time complexity:

Union:O(1), Find: O(logn)➔因爲 binomial tree

M 個 Union/ Find ➔O(mlogn)

(c) Union-by-height and find-with-path compression

find-with-path compression:在 find 過程中,把所有路徑上 node 指到 root

Time complexity:

Union:趨近 O(1), Find: 趨近 O(1)

M 個 Union/ Find ➔ $O(m*\alpha(m,n)) \cong O(m*\log^* n)$

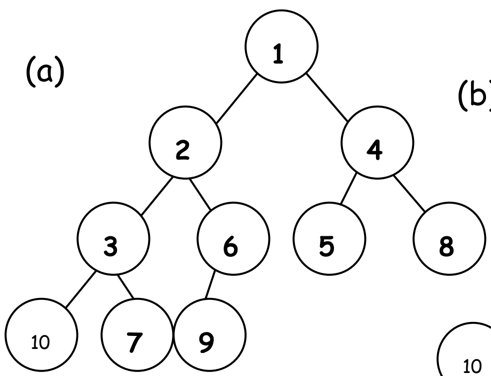
Create a min heap by 10,9,8,7,6,5,4,3,2,1

(a)bottom-up (b)Top-down

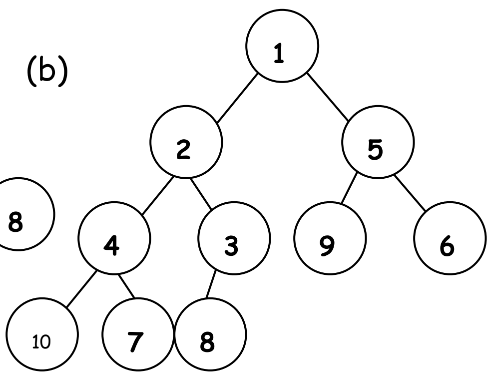
(a) bottom-up ➔先把 10,9,8,7,6,5,4,3,2,1 擺成 complete B.T ,做 Min-Heapfy

(b) Top-down➔一邊 insert ,一邊調整

(a)



(b)



operation	**Link list**	**Binary Heap**	**Binomial Heap**	**Fibonacci Heap**
case	**Worse case**	**Worse case**	**Worse case**	**amortized**
Meake-Heap	$\theta(1)$	$\theta(1)$	$\theta(1)$	$\theta(1)$
insert	$\theta(1)$	$\theta(\log n)$	$O(\log n)$	$\theta(1)$
delete	$\theta(n)$	$\theta(\log n)$	$\theta(\log n)$	$O(\log n)$
Find-Min	$\theta(n)$	$\theta(1)$	$O(\log n)$	$\theta(1)$
Delete-Min	$\theta(n)$	$\theta(\log n)$	$\theta(\log n)$	$O(\log n)$
Union	$\theta(1)$	$\theta(n)$	$O(\log n)$	$\theta(1)$
Decrease-Key	$\theta(1)$	$\theta(\log n)$	$\theta(\log n)$	$\theta(1)$

operation	Array	Link list	AVl Tree
Search for x	$O(\log n)$	$O(n)$	$O(\log n)$
insert	$O(n)$	$O(1)$	$O(\log n)$
Delete x	$O(n)$	$O(1)$	$O(\log n)$
Search k`th item	$O(1)$	$O(k)$	$O(\log n)$
Delete k`th item	$O(n-k)$	$O(k)$	$O(\log n)$
Output in order	$O(n)$	$O(n)$	$O(\log n)$

	DFS	BFS
adjacency list (both $O(e)$)	DFS examine each node one times	$\sum_{i=1}^{n} degree(i) = O(n)$
adjacency matrix (both $O(n^2)$)	(1)determine the adjacent vertex of v→ O(n) (2) at most travse n vertex	each vertex enter queue one times→while loop O(n) at most travse n vertex

	Single Source/All Destinations	Single Source/All Destinations	All Pairs Shortest Path
algorithm	**Dijkstra**	**Bellman Ford**	**Floyd Warshall**
Time complexity	linear array:$O(V^2)$ binaryheap O((|E|+|V|)log| V |) Fibonacci heap O(| E |+|V|log|V|)	*adjacency matrix:*$O(V^3)$ *adjacency lists*→**O(VE)**	$O(n^3)$
Negtive edge	**X**	**O**	**O**
Negtive cycle	**X**	**X**	**X**
method	Greedy	Dynamic Programming	Dynamic Programming

名詞定義

BST

(1)每一個 node 有一個 key

(2) $key(left\_child) < my\_key < key(right\_child)$

(3)左右子樹仍爲 BST

AVL tree ($\log(n+1) \le height \le 1.44\log(n+2)$)

空樹爲 AVL tree,T 不是空的二元樹, T_L, T_R 爲左右子樹

(1) T_L, T_R 均爲高度平衡樹

(2) $|h_L - h_R| \le 1$,其中 h_L，h_R 是 T_L，T_R 的高度

Red-Black Trees ($\log(n+1) \le height \le 2\log(n+1)$)

爲一BST

(1) node顏色爲黑或紅

(2) root 爲黑節點

(3) external nodes 爲黑節點

(4)若爲紅色node,其child必爲黑node (防止連續紅色node存在)

(5)root至leaf(external) 路徑上,具相同數目黑節點

B-Tree

一棵 order 為 m 的 B-tree 是一 m-way 搜尋樹。可為空樹，假若高度 >1

(1)樹根至少有二個子節點（children），亦即節點內至少有一鍵值（key value）。

(2) 所有 nodes 除了根與葉至少 $\left\lceil \frac{m}{2} \right\rceil$ children

(3)所有的樹葉節點皆在同一階層。

2-3 Trees

A 2-3 tree 為一搜尋樹,可為非空,如為非空,滿足

(a) 每一內部節點 為 2-node (1-element) 或 a 3-node(2-element)

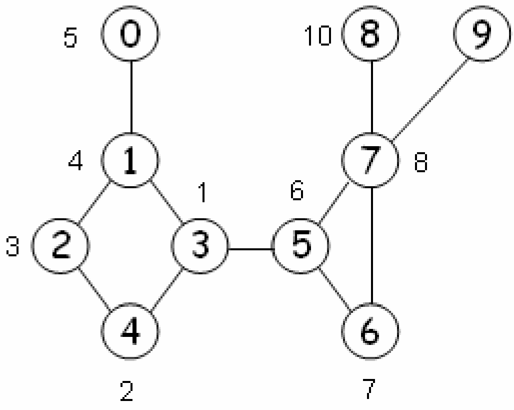
(b) 如為2-node (1-element),則 **LeftChild < LData.key < MiddleChild**

(c) 如為3-node(2-element),則

LeftChild < LData.key < MiddleChild< RData.key < RightChild

(d) 全部外部節點 (external node) 為於同一 level

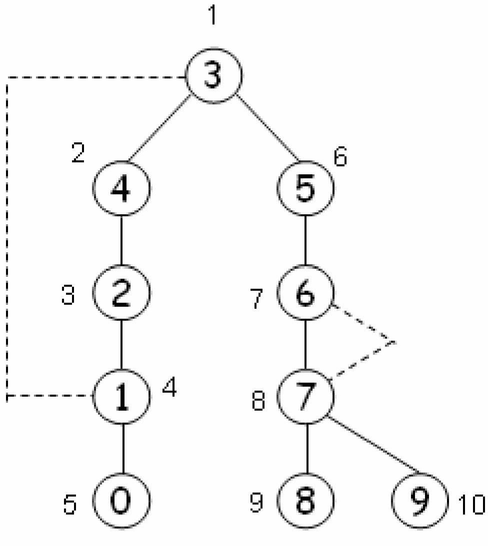
DFS spanning tree

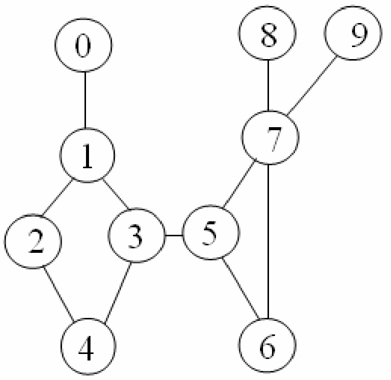


(a) draw the DFS spanning tree

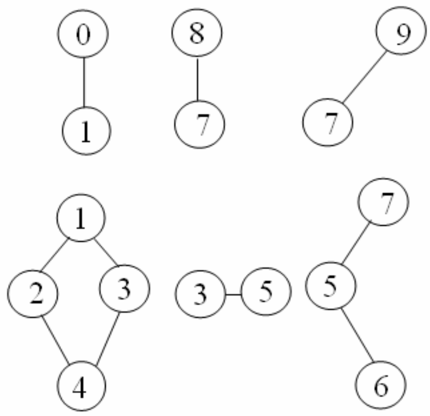
(b) point the articulate point

(c) draw the biconnected component





1,3,5,7 be articulation point



biconnected component

沒有 articulation point 的 connected component

$\because 3: root \quad \& \quad has \quad 2 \quad child$

$\Rightarrow 3: articulation \quad point$

$\because 6 = child[5], lower[6] \geq dfn[5]$

$\Rightarrow 5: articulation \quad point$

$\because 0 = child[1], lower[0] \geq dfn[1]$

$\Rightarrow 1: articulation \quad point$

$\because 9 = child[7], lower[9] \geq dfn[7]$

$\Rightarrow 7: articulation \quad point$

兒子的 low 大於父親的 DFN

則父親爲 articulation point

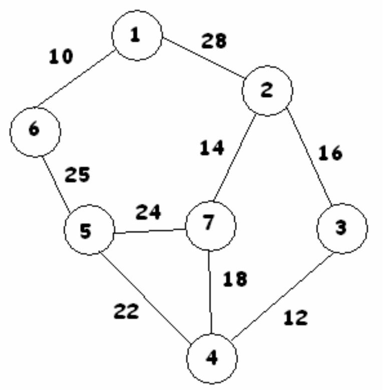
v	0	1	2	3	4	5	6	7	8	9
dfn	5	4	3	1	2	6	7	8	9	10
low	5	1	1	1	1	6	6	6	9	10

Sollins algorithm(Greedy)　(96 台大工科)

(1)把每個 node 視爲一棵 component ,則原題目爲 forest

(2) 每 component 挑一條最小邊,並把連到的 component 加到自己

Sollins algorithm



Sollins algorithm

Orign has 7 forest , forest1~ forest 7 ,each contains a certex

Run 1:

Vertex 1: min{10,28}➔10　**forest 6 ,edge(1,6) add into forest 1**

Vertex 2:min{14,16,28}➔14　**forest 7 ,edge(2,7) add into forest 2**

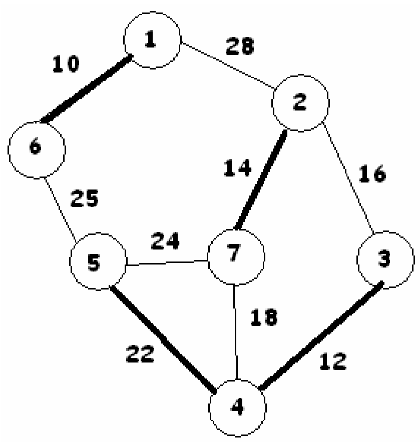
Vertex 3:min{16,12}➔12　**forest 4 ,edge(3,4) add into forest 3**

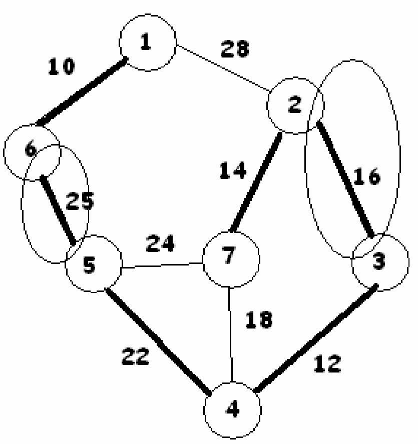
Vertex 5:min{22,24,25}➔22　**forest 5 ,edge(4,5) add into forest 3**

Run 2:

Forest {1,6}: min{25,28}➔25　**forest 1, forest 3** 做合併成新的 **forest 1**

Forest {2,7}: min{28,16,18,24}➔16 **forest 1, forest 2** 再做合併成新的 **forest 1**



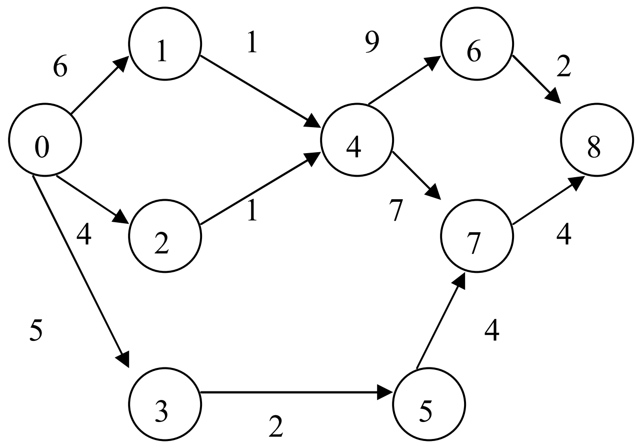


AOE network

Example 1:Find the critical path

earliest time: 由 start node 的後面 node 開始 ,前面路點加路徑,挑最大

latest time: 由 end node 的前面 node 往 start node ,後面路點減路徑,挑最小



Earliest(1)=Max{(0+6)}=6 , Earliest(2)=Max{(0+4)}=4 , Earliest(3)=Max{(0+5)}=5

Earliest(4)=Max{(6+1),(4+1)}=7 , Earliest(5)=Max{(5+2)}=7 , Earliest(6)=Max{(7+9)}=16

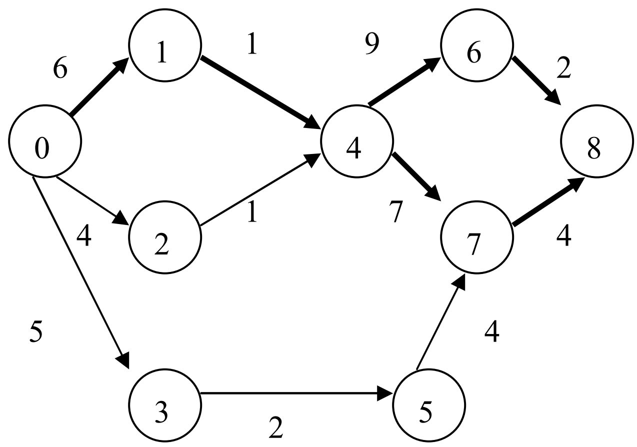
Earliest(7) =Max{(7+7),(7+4)}=14 , Earliest(8)= Max{(16+2),(14+4)}=18

Latest(6)=Min{(18-2)}=16 , Latest(7)= Min {(18-4)}=14 , Latest(4)= Min {(16-9),(14-7)}=7

Latest(5)=Min{(14-4)}=10 , Latest(3)= Min {(10-2)}=8 , Latest(2)= Min {(7-1)}=6

Latest(1)= Min {(7-1)}=6 , Latest(0)= Min {(6-6),(6-6),(8-5)}=0

late(ai)=early(ai),都算是 critical path



	best	Worse	average	storage	**Stable**	**Compar based**
insert	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	O	O
bubble	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	O	O
selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	X	O
quick	$O(nlogn)$	$O(n^2)$	$O(nlogn)$	$O(nlogn)$	X	O
merge	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(n)$	O	O
heap	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(1)$	O	O
shell	-	$O(n(\log n)^2$	-	$O(1)$	X	O
radix	$O(d(n+k))$	$O(d(n+k))$	$O(d(n+k))$	O(n*r)	Msd :X Lsd : O	X

QuickSort prove

worse case: partition N item into two part 1 N-1

$$T(n) = T(n-1) + cn$$
$$= \left[T(n-2) + c(n-1)\right] + cn$$
$$= ....$$
$$= T(1) + c(2 + ... + n) = T(1) + c\left(\frac{(n+2)(n-1)}{2}\right) = O(n^2)$$

best case: partition N item into two part N/2 N/2 (same as merge sort)

$$T(n) = 2T(\frac{n}{2}) + cn$$
$$= 2\left[2T(\frac{n}{4}) + c\frac{n}{2}\right] + cn$$
$$= ....$$
$$= 2^k T(\frac{n}{n}) + ckn \qquad (2^k = n \Rightarrow k = \log n)$$
$$= nT(1) + cn\log n = O(n\log n)$$

Prove Compare-base sorting algorithm $\Omega(n\log n)$
n data decision tree has n! leaf node $n! \le 2^{k-1} \Rightarrow \log n! + 1 \le k$ decision tree height at least $\log n! + 1$ $\because \log n! \ge \frac{n}{2}\log\left(\frac{n}{2}\right) \Rightarrow \log n! + 1 = \Omega(n\log n)$

(1)definition

(a)Hashing definition

一總資料儲存機制,欲存取資料 x 時,先經由 hashing function H(k)算出 hashing address 再到 hashing table 之 bucket 中存取

(b)Load density:

$\alpha = \dfrac{n}{b*s}$,$b*s$:hashing table size , n:使用過的 identifier 總數

α↑➔hash table 使用度↑➔collision 機會↑

(c) identifier density : $\dfrac{n}{T}$,T :total identifier, n:使用過的 identifier 總數

(d)bucket:一個 hashing table 分爲 b 個 bucket

(e)slot: 一個 bucket 分爲 s 個 slot ,每個 slot 可存一個 key

(f) collision:不同的key (如x & y)經過hashing function，得到相同的address

(g)overflow :新的key hash到滿的的bucket中

(h)perfect hashing: hashing沒有collision與overflow

(2)property

(a)使用雜湊法搜尋，檔案不須事先**sorting**。

(b)沒有**collision**及**overflow**，只需一次讀取即可，且搜尋的速度與資料量的多寡無關。

(c)保密性高，若不知雜湊函數，無法擷取到資料。

(d)可做**data compression**，適當的散置函數，將資料壓縮到一個較小的範圍內，節省空間。

(3)Hashing function

(a) Mid-Square : key 平方後,取中間三位數 ,e.g. $8128^2 = 66015625$ ➔pick 015 or 156

(b) Division: mod 取餘數,如 buck size=13 , h(57)=5, h(62)=10, h(26)=0

(c) Folding: e.g. 38123159639

(1)shift folding: 381 231 596 39 , 381+231+596+39=1247 , 1248 % b

(2)boundary shift folding: 381 231 596 39 ➔381 **132** 596 **93**=1202, 1248 % b

(4)collsion handle

Method 1:open addressing ➔Linear probing , Quadratic probing , Rehashing

Method 2:separate chaining

Given a hash table of size 10 (assuming that the hash table starts with index 0), show how the following data (in the given order) would be stored in the table using

(a) linear probing

(b) Quadratic probing

(c) double hashing: h1(x) = x % 10 and h2(x) = 2 + (x % 7)

Data: 99, 15, 75, 36, 20, 25, 89, 0, 47, 42

h(99)=9 ,h(15)=5 ,h(75)=5 ,h(36)=6 ,h(20)=0 ,h(25)=5 ,h(89)=9 ,h(0)=0,h(47)=7 ,h(42)=2

(a)

bucket	0	1	2	3	4	5	6	7	8	9
key	20	89	0	47	42	15	75	36	25	99
collision	0	2	2	6	2	0	1	1	3	0

(b)

bucket	0	1	2	3	4	5	6	7	8	9
key	20	0	42	47	25	15	75	36	89	99
collision	0	1	0	3	2	0	1	1	2	0

(c)

bucket	0	1	2	3	4	5	6	7	8	9
key	20	25	75	89	0	15	36	47	42	99
collision	0	1	1	2	2	0	0	7	3	0

$$h(75,1) = (h_1(75) + h_2(75) * 1)\%10 = 2$$

$$h(25,1) = (h_1(25) + h_2(25) * 1)\%10 = 1$$

$$h(89,1) = (h_1(89) + h_2(89) * 1)\%10 = 6$$
$$h(89,2) = (h_1(89) + h_2(89) * 2)\%10 = 3$$

$$h(0,1) = (h_1(0) + h_2(0) * 1)\%10 = 2$$
$$h(0,2) = (h_1(0) + h_2(0) * 2)\%10 = 4$$

$$h(42,1) = (h_1(42) + h_2(42) * 1)\%10 = 4$$
$$h(42,2) = (h_1(42) + h_2(42) * 2)\%10 = 6$$
$$h(42,3) = (h_1(42) + h_2(42) * 3)\%10 = 8$$

Hash (a)insert worse case (b)delete worse case (c) search worse case (d)search best case
(a)O(n)   (b)O(n)   **(c)O(n)   (d) O(1)**

(5) Linear probing

如 collision ,往後一個 bucket ,到最佳一個 bucket 時再繞回第一個 bucket

易形成**資料群聚 (Clustering)**現象，增加 Searching Time

(6)Quadratic probing

爲改善 Clustering 現象而提出。當 h(x)發生 overflow 時

h(x) overflow ➔ $(h(x) \pm i^2)\%b$ ， $1 \le i \le \frac{b-1}{2}$

$$(h(x)+1)\%b, (h(x)-1)\%b$$
$$(h(x)+4)\%b, (h(x)-4)\%b$$
$$(h(x)+9)\%b, (h(x)-9)\%b$$

(7)double hashing

使用 h1 hashing function,如果collision 則使用 $(h_1(key)+h_2(key)*i)\%bucket, i=1,2,3...$

double hashing: h1(x) = x % 10 and h2(x) = 7- (x % 7) ,bucket=10

5,10,15,20,6,35,11,40

bucket	0	1	2	3	4	5	6	7	8	9
key	10	15	20		11	5	6		40	35
collision	0	1	2		1	0	0		4	2

h1(15)=5 ,collision ➔ [h1(15)+ h2(15)]%10=1

h1(20)=0,collision ➔ [h1(20)+ h2(20)]%10=1,collision,[h1(20)+2*h2(20)]%10=2

h1(35)=5,collision ➔ [h1(35)+ h2(35)]%10=2,collision,[h1(35)+2*h2(35)]%10=9

h1(11)=1,collision ➔ [h1(11)+ h2(35)]%10=4

h1(40)=0,collision ➔ [h1(40)+ h2(40)]%10=2,collision,[h1(40)+2*h2(40)]%10=4,collision

➔ [h1(40)+ 3*h2(40)]%10=6,collision,[h1(40)+4*h2(40)]%10=8

(1) rehashing

準備多組雜湊函數 $h_1(x), h_2(x), h_3(x), \cdots, h_m(x)$，用 $h_1(x)$發生溢位，則改用 $h_2(x)$，若再發生溢位則改用 $h_3(x)$，…