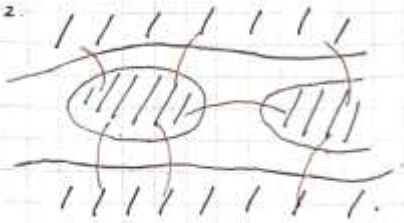
圖形概論 (Graph Basic)

6-1 七橋問題 (起点到終点每个橋都要記过)

1. Leonhard Euler

→ 找出最快記法. (the first paper in the history of graph theory)

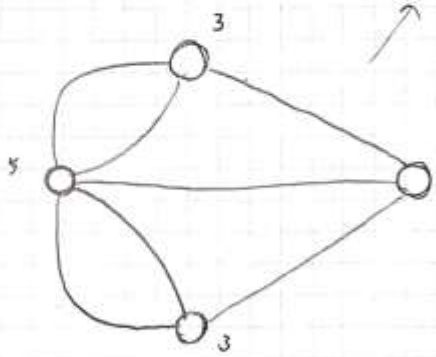
2.



→ 陸地 橋.

2 類:

5+3+3+3 = 7×2



G = { V, E }

V(G): vertex set

E(G): edge set.

O: 陸地 (V)

—: Bridge (E)

6-2 七橋問題的解法

1. degree: 每个点連到边的个数. $\sum \text{degree}(v_i) = |E(G)| \times 2$.

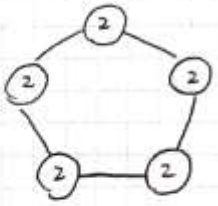
2. vertex types:

(1) odd or even degrees. (有不有充每个迴會走一次的走法)

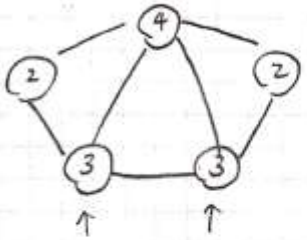
3. Eulerian path (trial) / Euler walk.

(1) visit every edge exactly once. → 好走出有点

(2) 0 or 2 nodes with odd degrees. (O) else (X)



0 node with odd degrees



2 node.

(3) what if 1 node with odd degrees? 起終点不一个

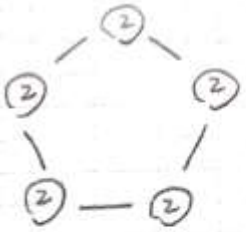
→ 1.3.5.7 奇数个 degree 不会产生.

6-3 應用於一筆畫 (所有邊走一遍)

1. Eulerian circuit / cycle / tour /

(1) begin and end at the same vertex.

(2) 0 node with odd degrees.



6-4 圖形的相關術語

1. Undirected graph (無向圖)

2. Directed graph (digraph) (有向圖)

3. Adjacent vertices (相鄰的)

4. Edge is incident to vertices.

5. Path: a sequence of edges (路徑)

6. small world (小世界): 6 degrees of seperation from the small world experiment

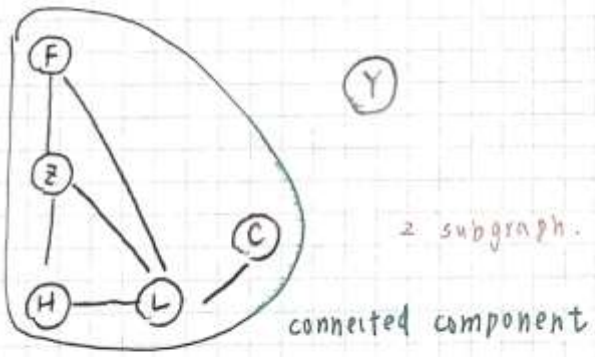
7. simple path: a path that passes through any vertex only 1

8. simple cycle: a cycle that passes through the other vertices only 1

6-5 更多的圖形相關術語.

1. connected graph. ↔ disconnected graph.

(1) There is a path between any 2 vertices.



2 subgraph.

connected component

2. complete graph

(1) There is an edge between any two vertices (任2个頂点間都有關係).

3. strong connected graph

(1) For any two vertices on a digraph, there is a path from one to other. (有方向的前提)

6-11 深度优先 (DFS) stack.

1. recursive DFS (vertex v)

```
Mark v as visited;
for ( each unvisited vertex u adjacent to v )
    recursive DFS (u);
```

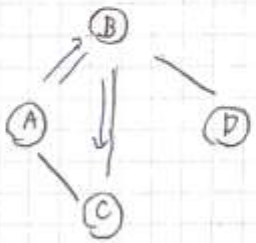
2.

(1) recursive form

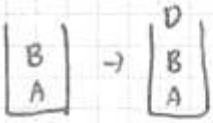
(2) Has an iterative form that uses a stack

3 while + stack

iterative DFS (vertex v)




```
s.createStack ();
s.push (v);
Mark v as visited;
while ( ! s.isEmpty()) {
    u = s.getTop ();
    if ( unvisited vertex w is adjacent to u ) {
        s.push (w);
        Mark w as visited;
    }
    else s.pop(); // 走回 B
}
```



3.

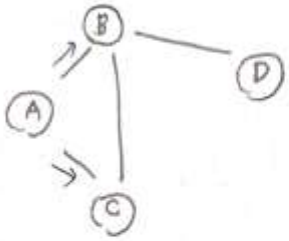
DFS traversal sequence: AB BC BD.

6-12 廣度優先 (BFS) Queue

1. (1) iterative form use a queue

(2) recursive form is possible, but not simple

2. iterative BFS (Vertex v)



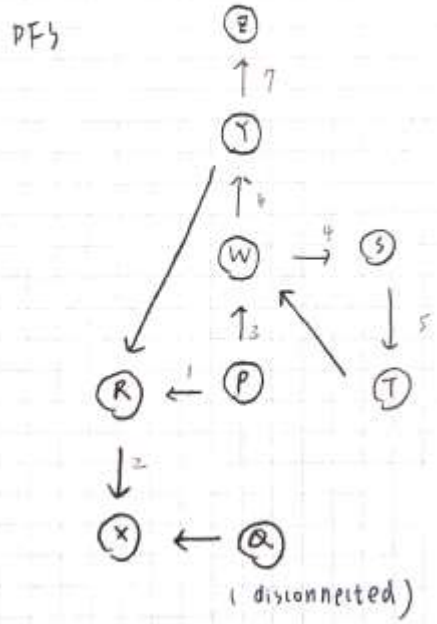

```
q.createQueue();
q.enqueue(v);
Mark v as visited;
While (!q.isEmpty()) {
    q.dequeue(u);
    for (each unvisited vertex w adjacent to u) {
        Mark w as visited;
        q.onqueue(w);
    }
}
```

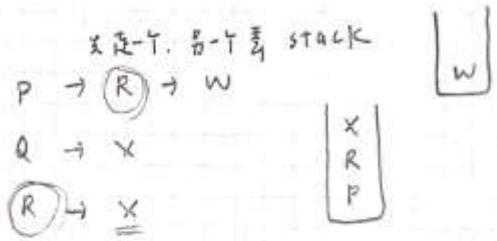
BFS traversal sequence : AB AC BD

3. spanning tree : BFS + DFS.

(深度材才)

6-13 圖形走訪序列





DFS : PR RX PW WS ST WY YZ

→ PRXWSTYZ.

4. weighted graph. (權重)

"the edge have numeric label" (有數字的)

6-6 定義圖型的抽象資料型別 (Graph as ADTs)

1. ADT graph Operations.

(1) int numVertices;

(2) int numEdges;

(3) int getNumVertices();

(4) int getNumEdges();

(5) int getWeight(Edge e);

(6) void add(Edge e);

(7) void remove(Edge e);

(8) bool isEdge(vertex u, vertex v);

(9) int getDegree(vertex v);

(10) bool isConnected(Graph g);

(11) edgeList traverse(Graph g);

2. Graph Representations

(1) Adjacency matrix

(2) Adjacency list

6-7 相鄰矩陣 (Adjacency)

ex:

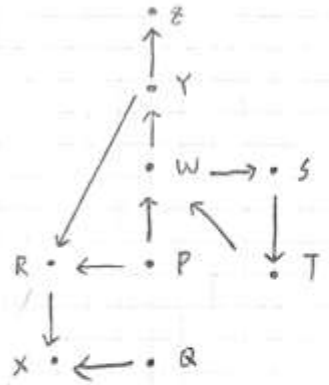
	P	Q	R	S	T	W	X	Y	Z
P	0	0	1	0	0	1	0	0	0
Q	0	0	0	0	0	0	1	0	0
R	0	0	0	0	0	0	1	0	0
S	0	0	0	0	1	0	0	0	0
T	0	0	0	0	0	1	0	0	0
W	0	0	0	1	0	0	0	1	0
X	0	0	0	0	0	0	0	0	0
Y	0	0	1	0	0	0	0	0	1
Z	0	0	0	0	0	0	0	0	0

→ indegree

NO successor (沒有後繼)

no predecessor (沒有前驅指向)

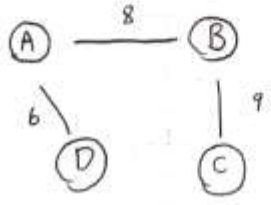
indegree ↔ outdegree



array

3. traverse(g): $O(|V|^2)$

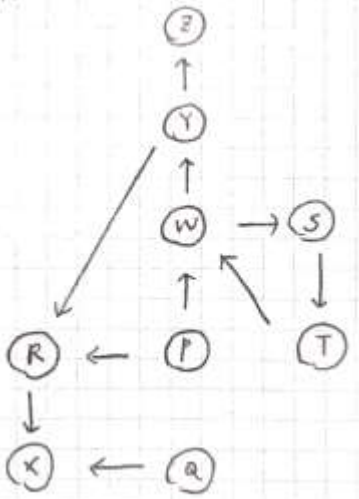
3. 權重.

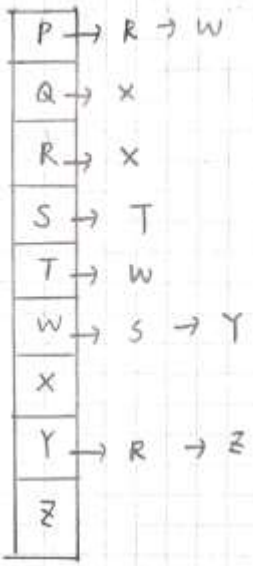


	A	B	C	D
A		8		6
B	8		9	∞
C		9		∞
D	6			

6-8 相鄰串列 (Adjacency List)

1.





traverse(g):

O(|V|+|E|)

* outdegree

沒辦法算 indegree

6-9 循序表示法 (sequence representation)

1. Mapping from vertex labels to array indices

P	Q	R	S	T	W	X	Y	Z
0	1	2	3	4	5	6	7	8

traversal O(|N|+|E|)

2. sequence representation

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]

10 12 13 14 ..... 19 20 20 2 5 6

相鄰代表有幾個邊 · out degree ·

6-10 圖形走訪 (Graph traversal)

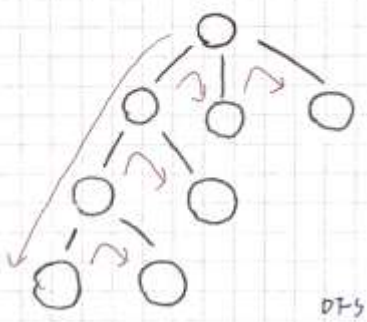
1. Depth-First Search (DFS) Traversal → 深度優先

→ A "last visited, first explored" strategy.

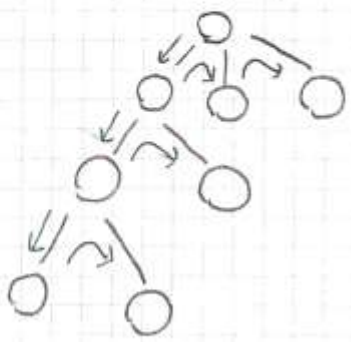
2. Breadth-First Search (BFS) Traversal → 寬度優先

→ A "first visited, first explored" strategy

ex:



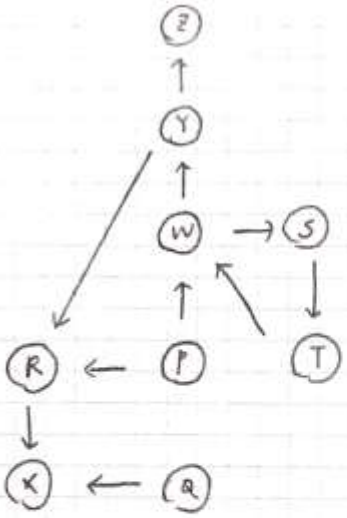
DFS.

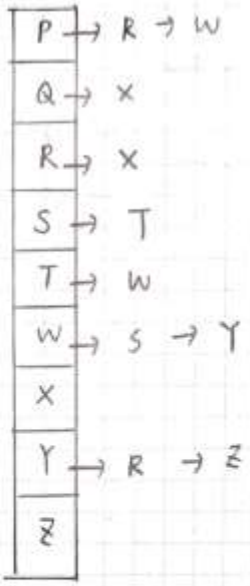


BFS.

6-8 相鄰串列 (Adjacency List)

1.





traverse(g):
O(|V|+|E|)

* outdegree
沒辦法算 indegree

6-9 循序表示法 (sequence representation)

1. Mapping from vertex labels to array indices

P	Q	R	S	T	W	X	Y	Z
0	1	2	3	4	5	6	7	8

traversal: O(N+|E|)

2. sequence representation

[0] [1] [2] [3] [4] [5] [6] [7] [8]. [9] [10] [11] [12]

10 12 13 14 ..... 19 20 20 2 5 6

相減代表有n個邊. outdegree.

6-10 圖形走訪 (Graph traversal)

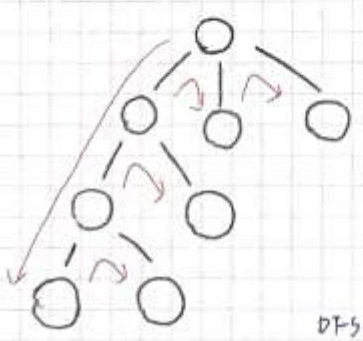
1. Depth-First Search (DFS) Traversal → 深度優先

→ A "last visited, first explored" strategy.

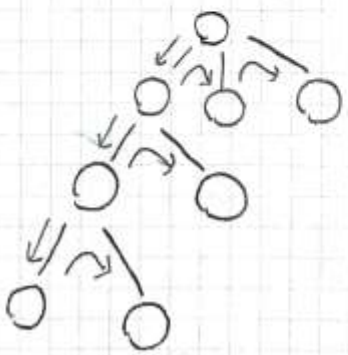
2. Breadth-First Search (BFS) Traversal → 寬度優先

→ A "first visited, first explored" strategy

ex:



DFS.



BFS.

• Wb 補

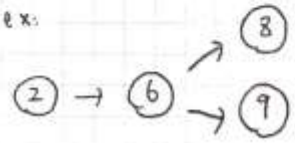
DAG: directed acyclic graph

(1) 在圖論中，如果有一个有向圖從任和頂点出发，無法經过多于條边回到該点，則此圖一定是有向無环圖。

topological (拓撲排序): topological sort

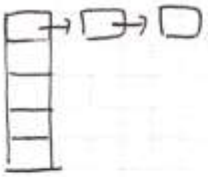
(2) 若 DAG 存在一條 edge (x,y)，那么序列中 vertex (x) 一定要在 vertex (y) 之前出現。

(3) ex:



存在 vertex (2,6), (6,9).

Adjacency list



isomorphism / isomorphic (同構)

→ 2 張圖連接方式一樣為同構

Prufer sequence

(1) 方法：逐次去掉树的頂点，直到剩下 2 个頂点。若原树 T，其頂点為 (1,2…n)，在第 i 步去掉

Eulerian path (尤拉路徑)

一條經过圖中每一條边恰好一次的路径

Eulerian Circuit / Tour

起、終点重合的尤拉路径。

* WJ 補

1. critical path analysis (関鍵方枋法)

(1) 列出完成項目的所有活动.

(2) 每个活动完成需要的 time.

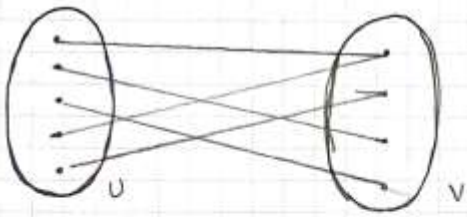
(3) 項目之間的依賴性.

2. four - colorable

(1) 在数学中 四色定理或四色图定理指出，只要将平面分離為連續的区域，就能生成稱為图的图形.

(2) 不需超过4种颜色即可对图的区域進行着色

3. bipartite graph (二分图)



4. bi-connected graph (双連通图).

(1) 在图論中，双連接图是一个連接的"不可分離"的图

(2) 若要刪除任一个項点，則該图仍保持連接

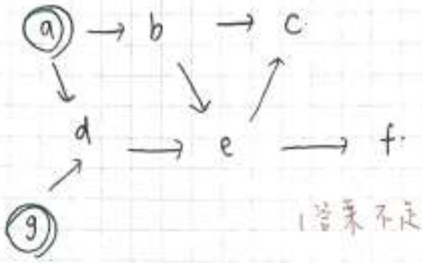
5. planar graph (平面图).

在图論中，平面图是可以畫在平面上且使得不同的边可以不互相交叠的图.

7 圖形應用

拓樸排序 (Topological sort) 7-1.

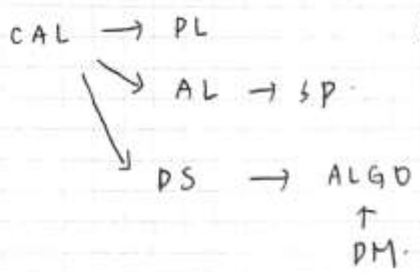
1. directed graph without cycles 沒循環有向圖
2. Acyclic Digraph or Directed Acyclic Graph (DAG)
3. example



a →
g →

(答案不定唯一).

7-2 範例.



(Activity-on-Vertex (AOV) Network)

7-3 拓樸排序的演算法版本

1. topsort 1

(1) Find a vertex that has no successor (out-degree = 0)

(2) Add the vertex to the beginning of a list.

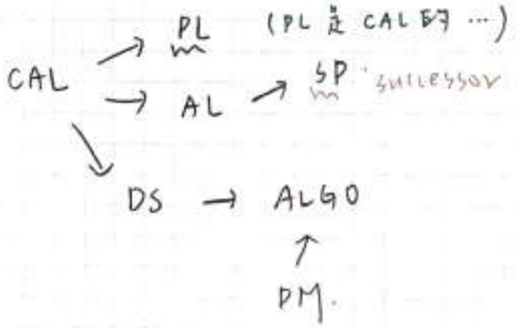
(3) Remove that vertex from the graph, as well as all edges that lead to it.

(4) Repeat, until the graph is empty

2.

immediate successor : 一個邊指向

(PL 是 CAL 的 …)



→ SP

→ ALGO. SP

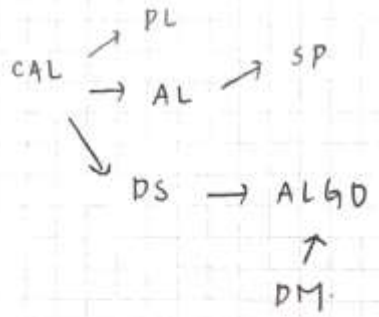
→ DM. ALGO. SP

→ PL, DM ALGO. SP

⋮

→ CAL → AL → DS → DM → ALGO → SP

3.



(out-degree, 紀錄從誰指過來)

=

1 AL → CAL

0 ALGO → DS → DM

3 CAL

1 DM

1 DS → CAL

0 PL → CAL

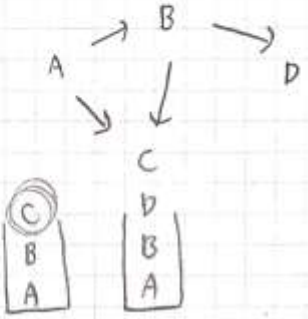
0 SP → AL

9-4 拓撲排序的演算法 2.

1. topsort 2

(1) 利用 DFS (stack)

2. iterative DFS (vertex v)




```
s.createStack ();
s.push (v);
Mark v as visited;
while (! s.isEmpty ()) {
    u = s.getTop ();
    if (unvisited vertex w is adjacent to u) {
        s.push (w);
        Mark w as visited;
    }
    else s.pop ();
}
```

→ C → A B D C

7-5 生成树简介 (Spanning Tree)

1. A tree is an undirected connected graph without cycle (acyclic)
2. 保证满足 connected, acyclic

connected ⟶ spanning tree ⟵ acyclic
(边愈多愈好) (临界点) (边愈少愈好)

7-6 生成树的特性

1. Detecting a cycle in an undirected connected graph.

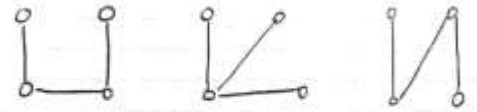
→ DFS, BFS

2. A connected undirected graph that has n vertices must have at least n-1 edges (4个点 3个边)

→ edge > 3 一定有 cycle. → edge < 3 一定不 connected

3. Number of spanning trees.

(1) How many different spanning trees? (4点3边)



→ isomorphic (同构).

① {2,2,1,1}

② {3,1,1,1}

一个点有几个边连到 → 2种

n个点 spanning tree 的数量 $= n^{n-2}$

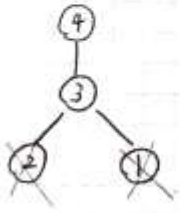
(2) 公式:

(边 ×2) − (点的数量) 允有几种摇奴位置

3×2 − 4 = 2. {1,1,0,0} {2,0,0,0}

7-7 以普吕弗序列记录一棵树 (Pruter sequence)

1. n个点存下来的字母大帝 n-2.
2. conversion algorithms.
(1) leaf with the smallest label
(2) keep the label of its parent



33 (唯一)

3.

degree 知道目前誰是根系.

2 A → B → C

3 B → A → D → E

(1) C → A

1 D → B

1 E → B

↓

(1) A → B → ~~C~~

3 B → ~~A~~ → D → E

(0) C

1 D → B

1 E → B

↓

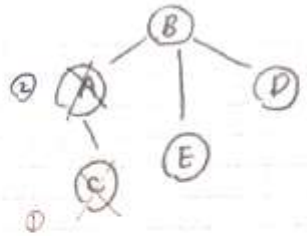
0 A

2 B → D → E

0 C

1 D → B

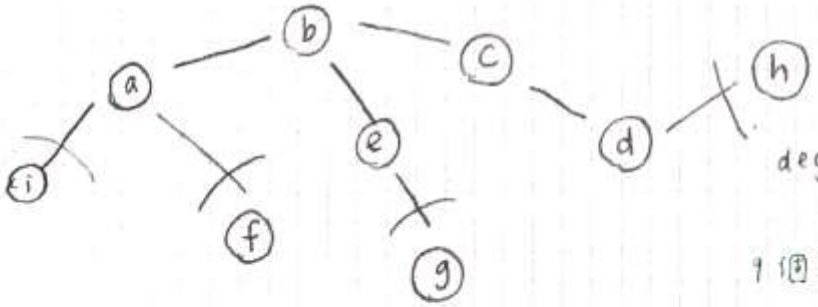
1 E → B



printer sequence : A

printer sequence : A B B

2-8 書名排序的轉枝練習



degree = 1

1個差-2 =7的字的 string

找 min.

1.
(1) 刪 f → a.
(2) 刪 g → ae
(3) 刪 e → aeb.
(4) 刪 h → aebd → last: aebdcba.

出現n次加多少
a+2, b+2, c.d.e+1.

2. 建立 g 的 degree 的 array.

a	b	c	d	e	f	g	h	i
3	3	2	2	2	1	1	1	1

→ 原本

a	b	c	d	e	f	g	h	i
1	1	1	1	1	1	1	1	1

3. 還原.

degree

a	b	c	d	e	f	g	h	i
3	3	2	2	2	1	1	1	1

↓

2	3	2	2	2	0	1	1	1

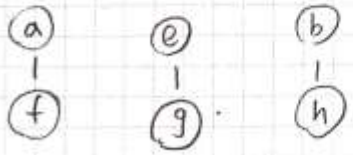
↓

2	3	2	2	1	0	0	1	1

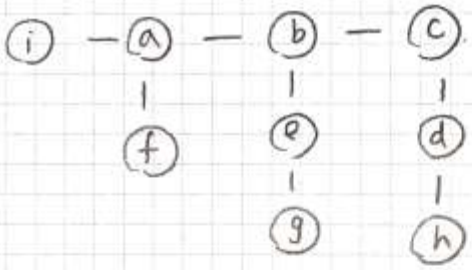
↓

ans: 100000001

先讀到 a.
找最小的: f → a-1. f-1.

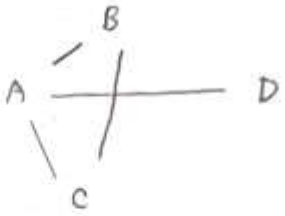


讀到 e. 找最小: g → e-1. g-1



9-9 以深度优先走訪建立生成树 (stack)

1.
```
iterativeDFS (vertex v)
    s.createStack();      count = 0;
    s.push(v);
    Mark v as visit;
    while (!s.isEmpty() && count < |V| - 1) {
        u = s.getTop();
        if (unvisited vertex w is adjacent to u)
            s.push(w);
            count++;
            Mark w as visited;    // (u,w)
        else  s.pop();
    }
```



AB . BC . BD

2. adjacent list. (DFS)

Visit

A → B → C → D F

B → A → C → D F

C → A → B F

D → A → B F

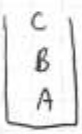
從 A start, A set TRUE

送第1下边. → AB.

到 B 後, A 走过 → 走 C

→ AB , BC

到 C. A.B 走过 → pop C → 走 D



AB, BC AB, BC, BD

9-11 Prim 演算法求最小生成樹

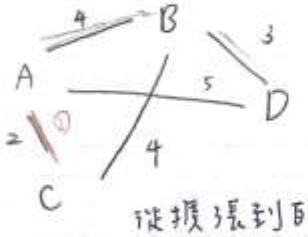
* Find a minimum spanning tree that begins at any given vertex.

(1) Find the least-cost edge (v,u) from a visited vertex v to some unvisited vertex u.

(2) Mark u as visited.

(3) Add the vertex u and the edge (v,u) to the minimum spanning tree.

(4) Repeat



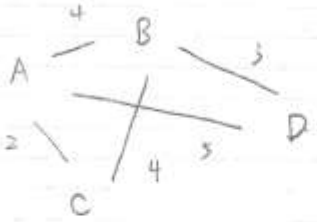
從擴張到的頂的鄰迴找 min

AC, AB, BD

2. PrimAlgorithm (vertex v)

```
Mark v as visited;
count = 0;
while (count < |V| - 1) {
    (v,u) = the least-cost edge from visited to unvisited)
    Mark u as visited;
    Add (v,u) into MST;
    count ++;
}
```

4. adjacent list.



visited

T A → C2 → B4 → D5

F B → D3 → A4 → C4

F C → ~~A2~~ → B4

F D → B3 → A5

AC → AB → 3D.

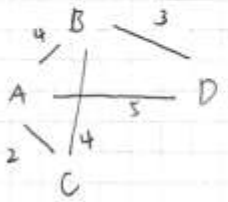
* need priority queue.

3. BFS

```
iterativeBFS (vertex x)
    q.createQueue();        count = 0;
    q.enqueue(v);
    Mark v as visited;
    while (!q.isEmpty() && count < |V|-1) {
        q.dequeue(u);
        for (each unvisited vertex w adjacent to u) {
            Mark w as visited;
            q.enqueue(w);   count++;
        }
    }
```

7-10 最小生成樹 (Minimum Spanning Tree)

1. cost of spanning tree
   → sum of the edge weights on a spanning tree



DFS : 4+4+3 = 11

BFS : 4+2+5 = 11

MST : 4+2+3 = 9.

2. a particular graph could have several minimum spanning trees.

3. other variations

(1) (minimum) Steiner tree (指定n個點)

(2) K-minimum spanning tree. (有k=3, 用3個點有序)

9-12 以 Kruskal 演算法找最小生成樹

1. Find a minimum spanning tree that begins at any given vertex
   (1) Create a forest, where each vertex is a tree.
   (2) Find the least cost edge (v, u) where vertex v and vertex u are from two different trees
   (3) Merge the tree of vertex v and vertex u, add (v.u) to tree
   (4) Repeat until |V|-1

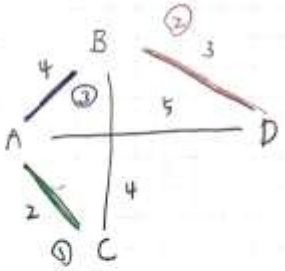
2. Kruskal Algorithm ()

```
Assign a unique label to each vertex ;
count = 0 ;
While (count < |V|-1) {
    (v.u) = the least-cost edge of 2 vertices with dif
    different labels
    Assign the label min(v, u) to all vertices with these
    2 labels ;
    Add (v, u) into MST ;
    count ++ ;
}
```

二維 tree

helpful datastructure

→ min heap



1. 找每個編号权重 min
   priority queue : 2, 3
   → A, C 編号 = 1
2. B, D 編号 = 2

3. Adjacent list
   存 assigned labels.

7-13 以 Sollin 演算法求最小生成樹 (2)

```
Sollin Algorithm ()
    Assign a unique label to each vertex ; size = |V| ;
    while ( size > 1 )
        Initialize Edge [1... size] as empty sets ;
        for each vertex v
            L = v.label ;
            (v.u) = the least-edge from v to u for any vertex
            with a different label ;
            if ( Edge [L] = (v.u) ),
        for each edge (v.u) in Edge but not in MST
            Assign min (v.label , u.label) to vertices in the
            sets of v and u ;
            Add (v.u) to MST ;
            size -- ;
```

7-14 初探最短路徑 (short path)

1. shortest path between 2 vertex in a weighted graph is the path that has the smallest sum of its edge weights.

2. Dijkstra's Algorithm (迪傑斯特拉)

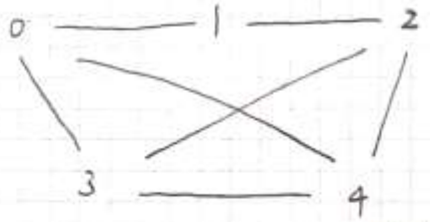
(1) Find the shortest paths between a given origin and all other vertices

7-15 以 Dijkstra 演算法求最短路徑

1. ABCD E

若為最短路徑，則 A→B、A→C → B−C 都是最短。

2.



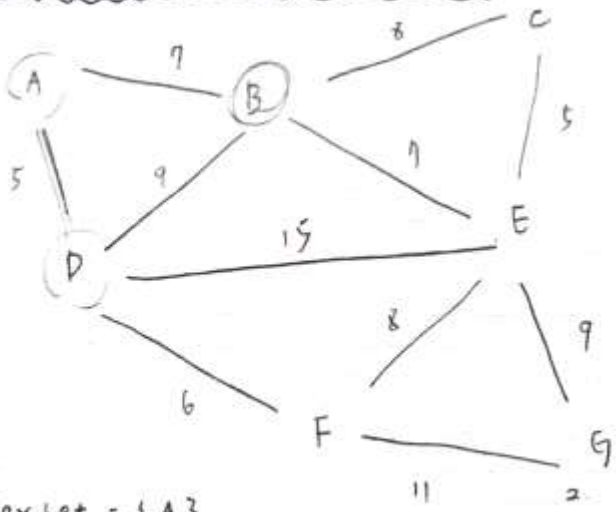
weight

step	v	vertexset	(轉折點)	[0]	[1]	[2]	[3]	[4]
1	–	0 (直達)	φ	0	8	∞	9	4 min
2	4	0, 4	1	0	8	5 min (0→4→2)	9	4
3	2	0, 4, 2		0	7 (0→4→2→1)	5	8	4

7-16 最短路徑樹

```
DijkstraAlgorithm (vertex V0)
    weight[0 … n] = { 0, ∞ …. ∞ };
    vertexset = φ ;        V = V0 ;
    do {
        Add v into vertexset ;
        for edge (v, u) where u is not in vertexset
            weight[u] = min { weight[u], weight[v]
                              + edgeWeight[v, u] ;
        cheapest = ∞ ;
        for vertex u not in vertexset
            if ( weight[u] < cheapest ) {
                v = u ;
                cheapest = weight[u] ;
            }
        }
    } while ( cheapest < ∞ ) ;
```

7-17 Dijkstra 演算法的範例



找到每个点程木的距离.

1. vertexset = {A}

weight: A B C D E F G

0 7 ∞ 5 ∞ ∞ ∞

2. vertexset = {A D B}

A B C D E F G

0 7 15 5 14 11 ∞

→

4 {A D B F E C}

A B C D E F G

0 7 15 5 14 11 22

7-19 Dijkstra 演算法的应用

→ 地图

7-20 任2点之間的最短路径 (All pairs shorest Path) : 全部列出

1. Floyd - Warshall algorithm

(1) Initialize distance matrix D^{-1} = adjaceny matrix.

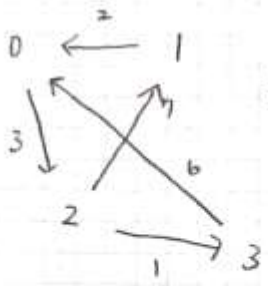
(2) For k=0 to |V| - 1

$D^k \leftarrow D^{k-1}$ // Add vertex k into vertexset (转补次数)

For i=0 to |V| - 1

For j=0 to |V| - 1

$D^k[i,j] = \min\{D^{k-1}[i,j], D^{k-1}[i,k] + D^{k-1}[k,j]\}$;



D^{-1}	0	1	2	3
0	0	∞	3	∞
1	2	0	∞	∞
2	∞	7	0	1
3	6	∞	9	0

(起点)

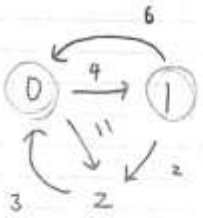
(终点)

D^0	0	1	2	3
0	0	∞	3	∞
1	2	0	5	∞
2	∞	7	0	1
3	6	∞	9	0

(允许经过0)

7-2 Floyd 最短路的算例.

1.



D^{-1}	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

起点

D^0	0	1	2
0	0	4	11
1	6	0	2
2	3	7	0

$D^0 \to D^1$

把1当跳板

已经改.

D_1	0	1	2
0	0	4	6
1	6	0	2
2	3	7	0

D_2	0	1	2
0	0	4	6
1	5	0	2
2	3	7	0

2. 无向图：对称.

D^{-1}	A	B	C	D
A	0	4	2	8
B	4	0	1	3
C	2	1	0	∞
D	8	3	∞	0

3. Summary.

(1) Topological sorting produces a linear order of the vertices in a directed graph without cycles.

(2) Tree are connected undirected graphs without cycles.

(3) Spanning tree of a connected undirected graph is a subgraph that contains all the graph's vertices and enough of its edges to form a tree.

14) minimum spanning tree

15) The shorest path

2-22 A* 演算法求最短路徑

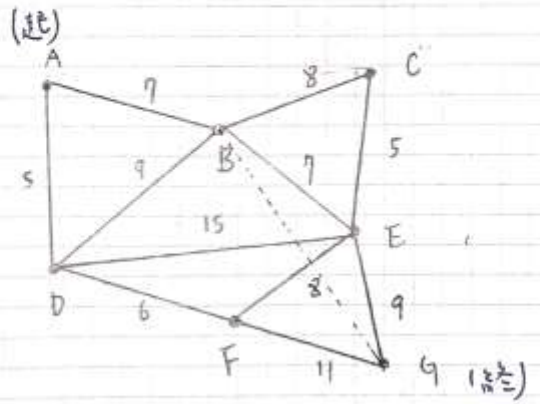
1. Best-first search by keeping a priority queue and traversing a path of the lowest expected total cost
2. combines 2 pieces of information.
   (1) Dijkstra's algorithm : favor vertices close to the origin
   (2) Greedy best-first search : favor vertices close to the goal

3 Expected total cost $f(v) = g(v) + h(v)$.

$g(v)$: expected cost of the path from the origin to vertex v.

$h(v)$: heuristic estimated cost from vertex v to the goal.

4. ex:



$h(B) = 14 < 7+9$
(跟終點的距離)
$h(C) = 12 < 5+9$
$h(D) = 17$
$h(E) = 9$
$h(F) = 11$
(忘掉自己)

→ $f(B) = g(B) + h(B) = 7 + 14 = 21$

$f(D) = g(D) + h(D) = 5 + 17 = 22$

· $f(C) = g(C) + h(C) = 7 + 8 + 12 = 27$

$f(E) = g(E) + h(E) = 7 + 7 + 9 = 23$

$f(G) = g(G) = 11 + 11 = 22$

single-source
single-destination

* L8 圖形問題

8-1 初探關鍵路徑方法

' Activity-on-Vertex (AOV) Network.

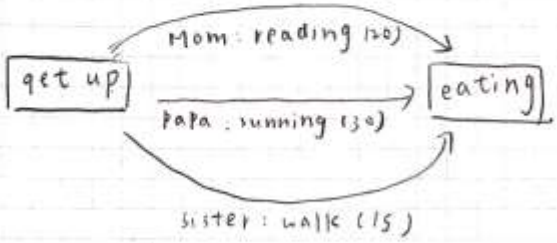
→ 活動在点上

= Activity-on-Edge (AOE) network

→ 活动在边上.

(1) Directed edge : activity (task) to be performed.

(2) vertex : event to signal the completion of certain activities



go to school

(3) 不能有 cycle.

(4) path length : the total time from start to end.

(5) Critical path : a path with the longest length.

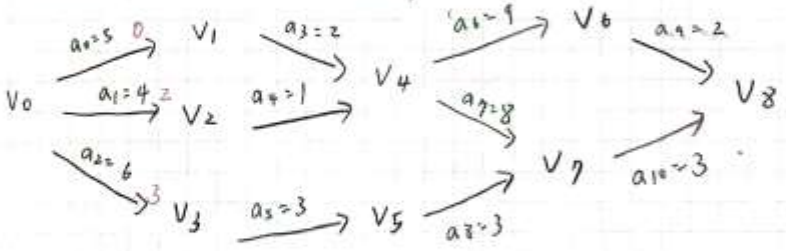
→ the minimum time required to complete the project

8-2 關鍵路徑方法法

' lastest time of an activity : la [0…10].

→ lastest time of event : le [0…8].

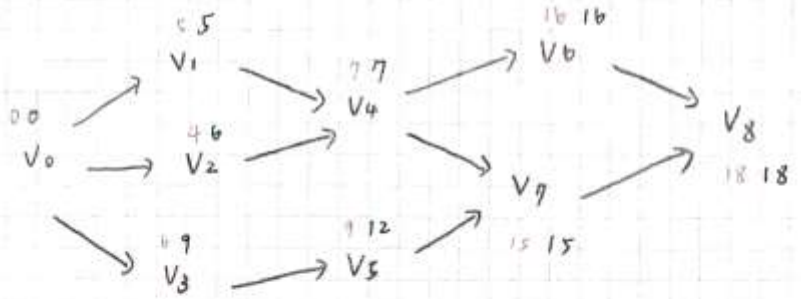
a1 a2 可 delay (媽的可賴床)



(1) la[X] = le[j] - duration of $\langle v_i, v_j \rangle$, where a_x is on $\langle v_i, v_j \rangle$

(2) le[x] = min { le[j] - duration of $\langle v_i, v_j \rangle$ } for every v_j that is an immediate sucleson of v_i

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
la:	0	2	3	5	6	9	7	7	12	16	15



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
→ ea	0	0	0	5	4	6	7	7	9	16	15
← la	0	2	3	5	6	9	7	7	12	16	15
la-ea	0	2	3	0	2	3	0	0	3	0	0

(3) la-ea is called (total) float or stack. (一定能接出一條線，可能不只一條)

→ amount of time that a task can be delayed without causing a delay to project completion time.

le-ea == 0 means a critical activity.

(4) determine critical paths (不急趕該到整个專案)

8-3 關鍵路徑分析的向前階段 送大的

(Critical Path Method : Forward Phase) →

' like topsort (top)

(1) Find vertex v that has no successor (out-degree = 0)

(2) Add v to the beginning of a list

(3) Remove

2.

(1) Find vertex v that has no predecessor (in-degree = 0)

(2) For each immediate successor u, do the following :

→ Set ea[x] = ee[v], where x is the activity on <v, u>

→ Set ee[u] = max { ee[u], ee[v] + duration of <v, u> }

→ Decrease the in-degree of u.

3. in-degree activity duration

V_0	0	→	a_0	5	→	a_1	4	→ a_2 6
V_1	1	→	a_3	2				
V_2	1	→	a_4	1				
V_3	1	→	a_5	3				
V_4	2	→	a_6	9	→	a_7	8	
V_5	1	→	a_8	3				
V_6	1	→	a_9	2				
V_7	2	→	a_{10}	3				
V_8	2	Null						

ee: 事件開始 time

ea: 事件結束 time

* indegree = 0 的先做.

用一个 array 存 ee, ea

8-4 關鍵路徑分析的向後階段. (Backward Phase) 逆小的 ←

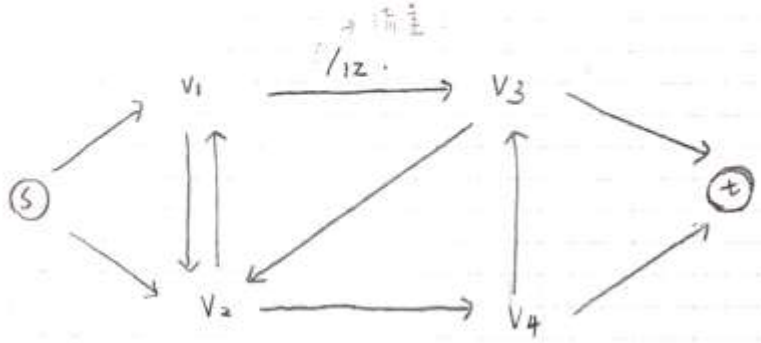
1. Find vertex u that has no successor (out-degree = 0)
2. For each immediate predecessor v, do the following :
   - → Set la[x] = le[u] - duration of <v, u> }, where x is the activity on <v, u>
   - → Set le[v] = min { le[v], le[u] - duration of <v, u> }.
   - → Decrease the out-degree of v

la: 對件最晚發生的 time

le: 這最晚開始的時間

8-5 最大流量的問題 (maximum flow problem)

1. We are given a flow network G with source s and sink t, and we wish to find a flow of maximum value from s to t

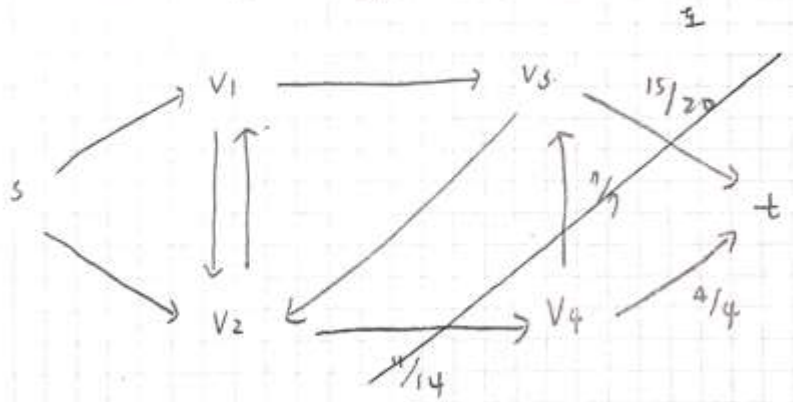


s. 水龍頭 (source)
t. 水桶 (sink)
找手水管 0 注

flow f(u,v) / capacity c(u,v).

(1) single source single sink maximum flow problem.

(2) maximum-flow min-cut theorem.



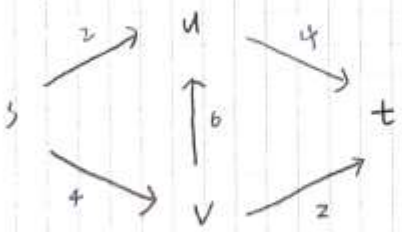
8-7 剩餘圖 (Residual graph)

1. residual capacity: $c(u,v) = c(u,v) - f(u,v)$

2. Edmonds-Karp algorithm.

→ Heuristic to find augmenting path.

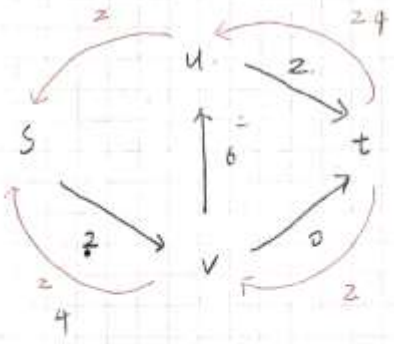
(放大的)



$s \to u \to t$ $c(s,u) = 2$, $c(u,t) = 4$

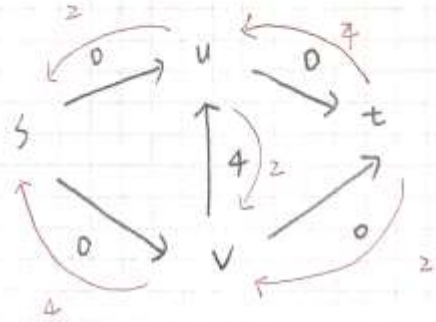
→ flow (u,v) = 2 (全部減 2)

$s \to v \to t$ flow (u,v) = 2

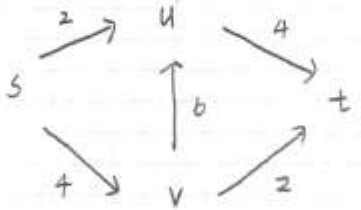


$s \to v \to t$ flow(u,v) = 2

$s \to v \to u \to t$ flow(u,v) = 2



8-8 以 FordFulkerson 演算法



結

c_f	s	u	v	t
s	0	2 (-2)	(4)	0
u	0 (+2)	0	0	4 (-2)
v	0	6	0	2
t	0	0 (+2)	0	0

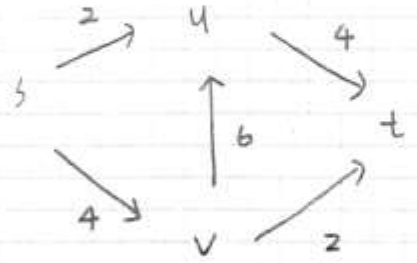
p: $s \to u \to t$ $c_f(p) = 2$ 正(-2) 反向(+2)

p: $s \to v \to u \to t$ $c_f(p) = 2$

8-9 以 EdmondsKarp 演算法求最大流量

1. Initialize $c_f(u,v)$ for every edge.
2. Find a path P from s to t by a heuristic ; (直覺)

→ Heuristic : max-capacity first.



c_f	s	u	v	t
s	0	2	4 (-4)	0
u	0	0	0 (+4)	4 (-4)
v	0 (+4)	6 (-4)	0	2
t	0	0 (+4)	0	0

end.

→

	s	u	v	t
s	0	0	0	0
u	2	0	2	0
v	4	4	0	0
t	0	4	2	0

maximum flow.

$s \to v \to u \to t$. $c_f(P) = 4$

$s \to u \to v \to t$ $c_f(P) = 2$

→ Heuristic : breadth first (廣度優先).

$s \to u \to t$ / $s \to v \to t$ / $s \to v \to u \to t$

8-11 歐拉迴路

1. Eulerian circuit (Euler tour)

→ Find a tour that would pass each edge exactly once and finally return to the starting vertex.

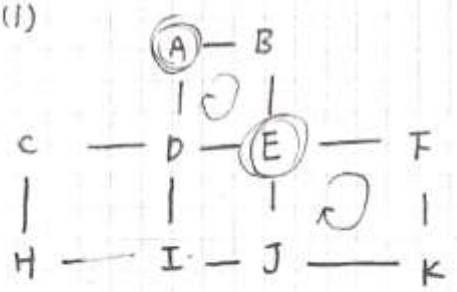
2. Hamilton circuit

→ Find a tour that would visit each vertex exactly once and finally return to the starting vertex

3. DFS-based Algorithm

2. DFS-based Algorithm

(1)

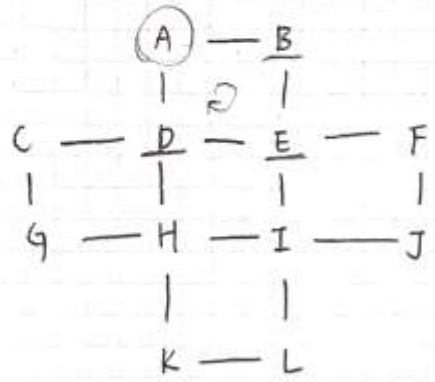


① A → B → E → D → A
(F,J) (C,I)

② A → B → E → F → K → J → E → D → A
(I) (C,I)
(走不回，break)

→ X

(2)



A B E (F,I) D

A B E F J I E D A

A B E F J I H D C G H K L I E D A

(呼叫 3 次)

8-12 旅行銷售員問題 (Traveling Salesman Problem: TSP)

1. Brute-fource algorithm (暴力法)
速度慢 答案好

2. Greedy algorithm → 速度快，答案不好

3. Branch-and-bound algorithm 介於中間

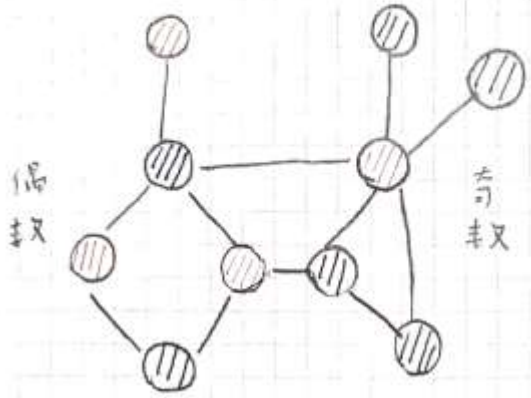
8-13 圖形著色的問題 (Graph coloring problem)

1. vertex coloring (edge coloring).

2. sequential ordering algorithm. (給定出現的順序) → greedy coloring

2. max-degree first
缺點：二方圖，degree 一樣.

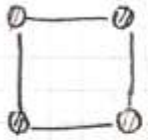
8-14 以貪婪策略為圖形塗色的範例



chromatic number = 3

*平面图 color ≤ 4.

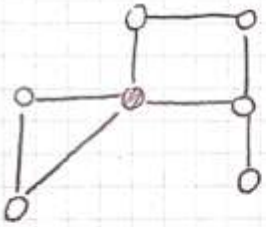
1. cycle graph $G_n = 2 \vee 3$



2. wheel graph $G_n = 3 \vee 4$

8-15 双連通图 (Bi-connected graph)

1. Articulation point (関節)

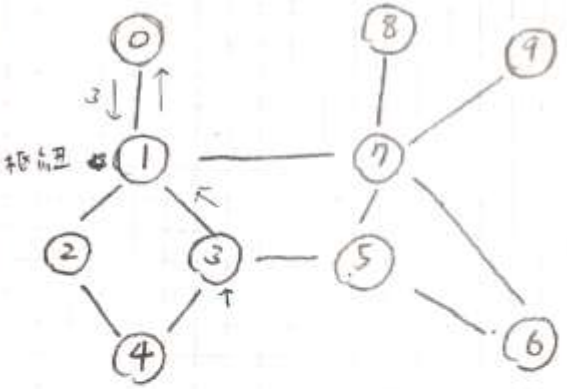


若 ● 所持整个就不気通 → disconnect (羽矢)

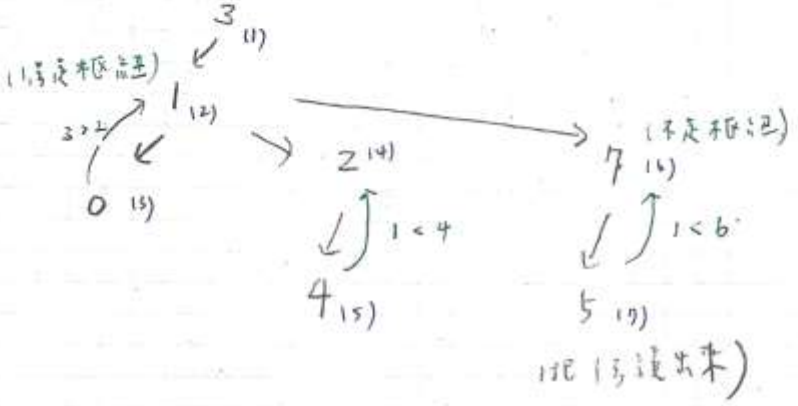
2. A connected graph that has no articulation point

→ Bi-connected graph

8-16 以深度優先走訪找出圖形的關節點



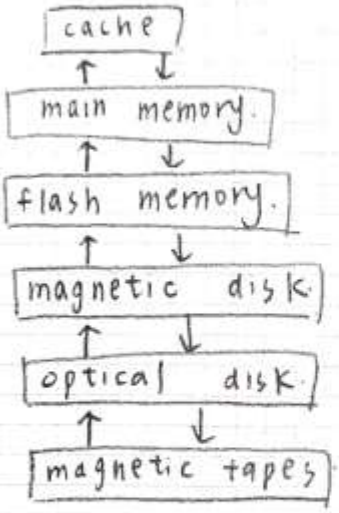
判斷③是否為關節點 (?).



9 次要儲存体

9-1 次要儲存体的基本观念

1. main memory vs secondary storage



→ CPU time vs. I/O time

↑

seek + latency + transfer

讀取的数量.

9-3 外部排序

1. internal sort : main memory.
2. external sort : secondary storage + main memory
3. example. 0400 个 student records? (if each record takes 100 KB)

→ Can we use very less space to do the same job!

(1) The records are divided into 64 blocks.

(2) each block keeps only 100 student records (10 MB).

- step 1 : Internal sort on each block. (bubble sort, 100 笔排序)
- step 2 : (external sort) merge sort

4. step 1 : Internal sort

10MB 10MB.

→ 64 笔的排序. $(R_{1,1}, R_{1,2}, R_{1,3}, \cdots R_{1,100})$ $(R_{2,1}, R_{2,2} \cdots R_{2,100}) \cdots (R_{64,1}\ R_{64,2} \cdots$

step 2: Merge sort (Round 1)

$B_1, B_2, B_3 \cdots B_{64}$

10MB 10MB

$B_{63} + B_{64} = R_{32}$

10MB 10MB. 20MB

→ $R_1\ R_2 \cdots R_{32}$

(20MB)

讀 + 寫

step 2 (Round 2)

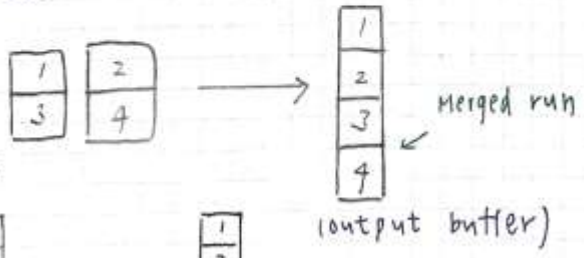
Memory Space : 40 MB.

$R_1 + R_2 = S_1$

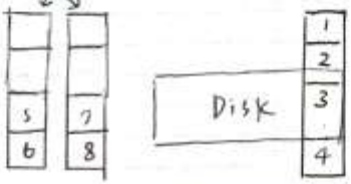
20MB 20MB 40MB.

9-4 多重合適合併. 2+2+4=8

1.



Runs being merge



(input buffer) 各自排序好.

(2-way merge)

2. 2-way merge (每筆資料被讀兩次)

(1) 64 runs → 32, 16, 8, 4, 2, 1 → $\log_2 64 = 6$ passes.

(2) 16 runs → 8, 4, 2, 1 → $\log_2 16 = 4$ passes.

3. Higher-order merge can reduce I/O time