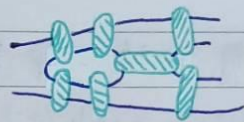


6-1 七橋問題 Seven Bridges of Königsberg

- 應用：某領土因河流被切四塊，七座橋進行路跑，路線必經每一橋。
求一種路線使每橋恰經一次。

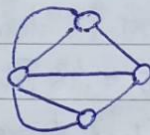
- 屬於圖形領域
抽象化 ^{abstraction} 的目的
在於擴大應用



- 封裝 = Bridge (七座橋), land (四塊陸地)
^{線 (E)} ^{點 (V)}

$E(G)$ = edge set

$V(G)$ = vertex set

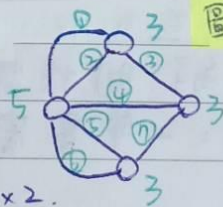


$G = \{V, E\}$

6-2 七橋問題的解法

- 如何判斷無解

degree = 每個點所連到
的邊個數



圖形概論

$$\sum \text{degree}(V_i) = |E(G)| \times 2$$

$$(5+3+3) = 11 \times 2 \quad (\because \text{接出去 \& 接回來})$$

可分度 degree 為奇 or 為偶數

ex = 有 4 個 degree 為奇的點

• 是否有路徑解?

• 有 0 or 2 個奇數 degree 的點.

• 不可能有奇數個奇數 degree 點

∴ degree 的加總為路徑的 2 倍.

6-3 應用於一筆畫

• Eulerian path (trial) / Euler walk (有解)

• 有 0 or 2 個奇數 degree 的點.

• Eulerian circuit (cycle) / Euler tour

0 個奇數 degree 點.

起點 = 終點 \Rightarrow 迴圈 (一進一出).

若 2 個奇數 degree 點, 則起點終點必為奇數 degree 點, 否則可能只進不出.

tour \subseteq walk, 故 6-1 則無解.

• 若有解, 則找出走法, 非唯一.

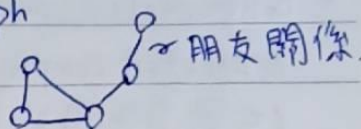
• 應用: 一筆畫圖形.

6-4 圖形的相關術語

- 無向圖 Undirected graph

ex = fb 用戶

雙向關係



- 有向圖 digraph

單向關係



- Adjacent vertices 相鄰

incident = 形容邊 & 點的關係

- small world: 朋友清單的所有路徑都走過
並每條走 6 層, 則能涵蓋所有資料

基於理論

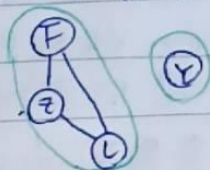
- Cycle = 當路徑經過自己

- simple path = 在一條路徑上, 其每個點只被走一次

simple cycle = 在一 cycle 路徑上,
中間沒有重覆的點

6-5 更多的圖形相關術語

- disconnected graph 某點 & 其他點完全無關連
connected components 相連的部分
(右圖有2個) & subgraphs



- complete graph: 任2點間必有連結.
- 有向圖亦有連通觀念, 較困難.
但若利用單向連結使任2點能連通
則更厲害 = strong connect graph
- weighted graph: 加權圖, 在線上, 可以有數字描述二者間的緊密程度.

6-6 定義圖形的抽象資料型別

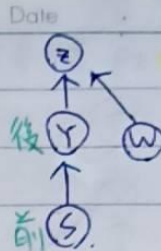
- 如何表示抽象圖形, 且基本運作
ex = 單向 or 雙向, 是否權重, numVertices
add(), remove(), isEdge(), ...
- 儲存方式表示法 Representations
Adjacency matrix 陣列.
Adjacency list 鏈結串列.

6-7 相鄰矩陣

- 有方向性 = successor, predecessor

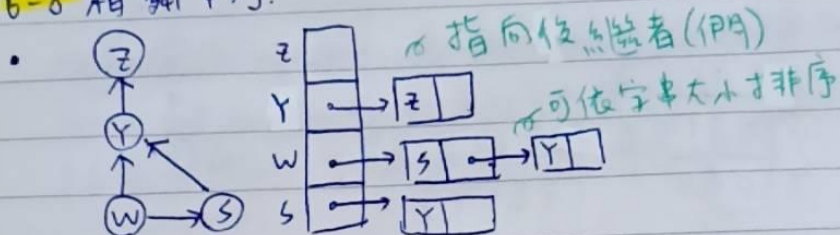
	X	Y
X	0	0
Y	1	0

有指出的箭頭
(X到Y)



- 若某列全0, 即該點無後繼者 (沒指向人)
某行全0, 即無前者 (沒人指)
- 有些問題需要找到其起點 & 終點, 即利用行 or 列的全0判斷
- 亦可利用陣列來制定路徑 = 從前一個後繼者做為下次走路的起點
- degree 分為 inDegree & outDegree
∴ 有向圖
- traverse = $O(|V|^2)$ $V =$ 所有點所形成的集合
- 有時 matrix 非 0 or 1. ∴ 可記權重

6-8 相鄰串列.



只能用於 out-degree

- 若為無向圖，則起 & 終點出現在對方陣列
- $\text{traverse}(g) = O(|V| + |E|)$
- 比較: 若邊少, matrix 會浪費空間

6-9 循序表示法

- 序列式表示法. sequential representation
用陣列存, 但用 list 觀念.

陣列大小 = 邊 + 點的個數, 再將點編號
ex = 9 個點, 用 0~8 編號,

[0] [1] ... [8] [9] [10] [11] [12] ...

10 12 ... 20 20 2 5 6

- 適用於不變的圖

6-10 圖形走訪

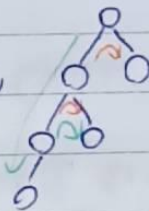
- 因 & 樹不同, 沒有順序, 故要能從任一點完成走訪且解決 subgraph & loop
- 利用 visit array 記錄路徑

① DFS (Depth-First Search) 深度優先 可退回

利用 stack (後進先出) 擴張

② BFS (Breadth-First Search) 寬度優先

利用 queue (先進先出) 逐步擴張

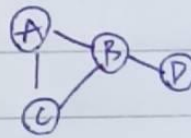


6-11 深度優先走訪

- 可分遞迴 & 非遞迴, 可利用有向 & 無向圖或相鄰的點走訪, 並記錄路徑

```
while (!s.isEmpty()) { u = s.getTop();  
    if (unvisited.) s.push(w), mark visited
```

$\begin{bmatrix} C \\ B \\ A \end{bmatrix}$



```
else s.pop(); // back track.
```

→ 判斷 cycle

6-12 寬度優先走訪

- 因利用 queue 概念, 故難以用遞迴完成
先將鄰近的走完再擴張
- ```
while (!q.isEmpty()) { q.dequeue
 for (each unvisited neighbor)
 mark visited q.enqueue
```
- spanning tree 用最少的邊連起點. (生成樹)

### 6-13 圖形走訪序列

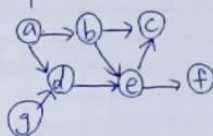
- 對 list 進行走訪
- 記得最後檢查走過的點是否等於  
點的個數. (要處理 subgraph)
- DFS = PRXWSTYZ  
BFS = PRWXSYTZ



## 7. 圖形應用

### 7-1 初探拓撲 Topological

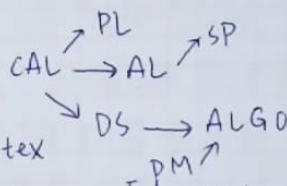
- 常用於網路，描述電路連線關係。  
給定一個圖，並根據圖形結構排序。  
且必須有向圖，才有先後順序可參照。且無 cycle (圖無起點)  
稱為 Acyclic Digraph or Directed Acyclic Graph **DAG**
- Ex = 開關改裝 可從 a 或 g 出發。  
or 工廠製程 (零件產完才有機會組裝)



Ans-1 = a → g → d → b → e → c → f

### 7-2 拓撲排序的範例

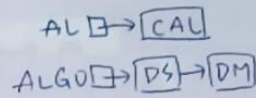
- 修課應用，且有擋修 只要能產生一個即成功
- 以點 (Vertex) 為主 Activity-on-Arrow (AOA) Network. [以邊為主 Activity-on-Edge Network] **AOE**
- AOE 較複雜，邊較多



### 7-3 拓撲排序的演算法版本一

- 無中生有：可從頭 (no predecessor) 或從尾端 (no successor) 開始
- 從尾端開始放入答案區，
- immediate = 立即指向。ex: 7-2 的圖示。PL 是 CAL 的 immediate successor
- 從尾端走 (ex: SP) 則去除 SP 和其 immediate predecessor
- 下一步可選 PL, AL, ALGO。舉凡最尾端都可 or 孤立者 **out degree = 0**
- 其中一解：CAL, AL, DS, PL, DM, ALGO, SP

- 結構：以有那些 predecessor 記錄於主陣列

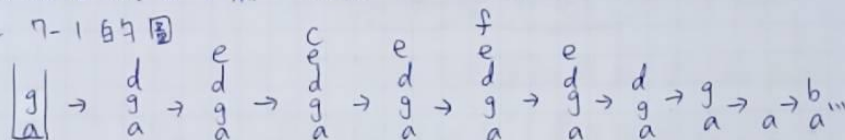


- 亦可從頭 (no predecessor) 做，影響存取結構。

#### 7-4 拓撲排序的演算法版本二

- 利用第六單元建立好的深度優先 (DFS) 走訪，且亦可從頭或尾且同於 7-3 的方法程度，只是走訪方法
- 先將所有 indegree = 0 的點放到 stack，當 stack 放好，即可開始輸出答案，且結果由右側開始建立。

- ex: 7-1 的圖



#### 7-5 生成樹簡介

- 必須看過每個點，點 & 點間至少有一路徑，且無向無 cycles  $\Rightarrow$  acyclic，可透過樹大略看到圖的樣子。
- Spanning 意指為原圖的子圖，且涵蓋圖上所有點。  
可應用於通訊傳遞，且跟隨同一個通訊協定 (同一圖中)  
要符合相連通 (加邊)、無迴圈 (減邊)  
spanning tree 為一個臨界點，達平衡

#### 7-6 生成樹的特性

- 1. 若已知是 connected，且有  $n$  個點，則至少有  $n-1$  個邊
- 2. 今若增加邊，無論加在那，若會出現 cycle



- 若有 4 個點 & 3 個邊 spanning tree 有幾種長法?

ex = 圖 - 3 邊同構 (isomorphic) R 算一種

同構亦可用 degree 判斷  $1 \{2, 2, 1, 1\}$   $2 \{3, 1, 1, 1\}$   $P_4$

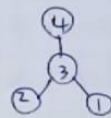
公式 =  $n^{n-2}$  ex = 4 nodes =  $4^{4-2} = 4^2 = 16$

### 7-7 以普呂弗序列記錄一棵樹 Prüfer sequence

- 將樹以唯一表示法存下: 設有  $n$  個點 而存下字串長為  $n-2$

- 方法 = 每個回合都找樹葉且層級最小者

並記錄該樹葉之父節點  $(3) \rightarrow (3, 3)$



即可將 16 種圖都以不同結果紀錄

- 亦可由字串還原出圖形 若為  $(3, 3)$  則知  $(3)$  為內部節點 故  $(3)$  的 degree 為  $1 + 2 = 3 \Rightarrow 1131$

在還原的同時要將 degree 的記錄扣除

- 長度為  $n-2$  的原因:  $\because 4 \times 4 = 16$  種數字排法 =  $n^{n-2}$   
且圖形種類為  $16 = n^{n-2}$

- 實作 = 

| degree | A | B |
|--------|---|---|
| 1      | A | B |
| 2      | B | A |
| 0      | C |   |
| ...    |   |   |

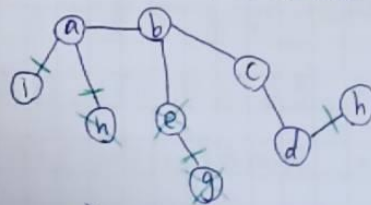
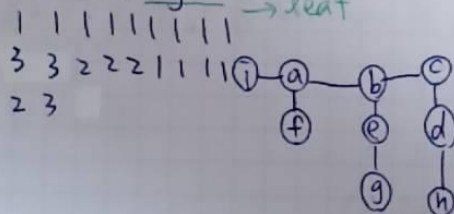
 先做 C  $\because$  degree 最小  
 $B \Rightarrow 1 \rightarrow 0$

### 7-8 普呂弗序列的轉換練習

- $a e b d c b a$

樹葉可從最大 or 最小的點

- 還原 =  $a b c d e f g h i$   $\rightarrow$  leaf



產生長度 7 的字串  
9 nodes



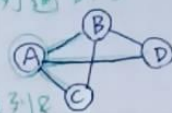
### 7-9 深度優先走訪建立生成樹

- 利用 BFS or DFS 即可找出一種生成樹

iterative DFS (Vertex  $V$ ) 資料建立同前  
第 6 章的 graph (主陣列 & 諸  $V \rightarrow \text{visited}$  ...)

```
s.createStack(); s.push(V); V → visited;
while (!s.isEmpty() && count < |V|-1) u = s.getTop();
 if (unvisited 且 符合要求的優先者, 並為鄰近者)
 s.push(w); count++; w → visited;
 else s.pop();
```

此例適 BFS



iterative BFS (Vertex  $V$ )

```
q.createQueue(); q.enqueue(V); V → visited;
while (!q.isEmpty() && count < |V|-1) q.dequeue(u);
 for (unvisited, adjacent, priority)
 w → visited; q.enqueue(w); count++;
```

### 7-10 最小生成數 → 並不唯一

- 某些應用可能有特殊規定, 所以並不是隨便一種型態都可  
此時 DFS or BFS 的走訪並不適用

- 應用 Steiner tree: 指定硬體能有更好的通訊

ex:  $k=3$   
k-minimum spanning tree: 在一群點中, 哪個樹  
只有三個點且最小(大) 權重

Steiner 車交區域, 且指定,  $k$  則是要找出適當區域

### 7-11 Prim 演算法求最小生成樹 (法 1)

- 類似於 DFS, 但在找 unvisited 邊時會選擇 權重小的

Prim Algorithm (Vertex  $v$ )

$v \rightarrow \text{visited}$ ; count = 0

while (count < |V|-1)

priority queue

- $u \rightarrow$  權重最小且 unvisited 的邊
- $u \rightarrow$  visited    Add( $u$ ) into MST; Count++;
- 資料結構類似於 DFS 並可將點的所有的邊針對權重由小到大排序
- 7-12 以 Kruskal 演算法求最小生成樹 (法2)
- 利用“合併”的觀念,  $n$  個點即為  $n$  棵樹, 兩棵兩棵合併判斷兩棵樹是否有邊使相連
- 當兩棵樹有多個邊使相連時, 擇權重小者用以連接則有  $n-1$  個回合 &  $n-1$  個邊
- 設 label 給不同 vertex, 以利合併, 此時對於邊所連到的兩 label 是否相同, 若否則列入考量 (priority queue)
- 合併後將兩端 label 統一 (or min) 即成為同一棵樹
- 資料的結構亦同, 答案固定

### 7-13 以 Sollin 演算法求最小生成樹 (法3)

- 為 7-12 的加速版, 多了 Edges[], 以記錄層找到的最短邊, 平行處理, 同時合併, 空間換時間
- while (size > 1)
  - for each vertex  $v$ ;  $L = v.label$ ;  $u = priority$ ;
  - if (Edges[L].weight >  $u.weight$ ) Edges[L] =  $u$ ;
- 找出每個點所連到的最優先權重, 方能同時合併多個此時答案亦同

#### 7-14 初探最短路徑

- 多應用於地圖資料 ex= 航班的設計, 將不同資訊視題意設為權重
- 可能出現長路線卻低成本, 因為有很多不同組合故圖若卻複雜組合變化
- Dijkstra 解決找最短路徑的問題  
origin 為起點, 找出某點到地圖上所有點的最短路徑
- 用到 vertexSet & weight

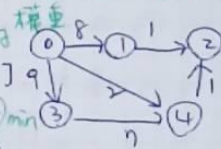
#### 7-15 以 Dijkstra 演算法求最短路徑

- 若有一條最短路徑且不只經過一點, 當拿掉兩端的其中一點, 則剩餘路徑仍為最短  $\Rightarrow$  遞迴關係
- VertexSet weight 到這些點的權重

0  $\rightarrow$  1: 0 (可轉機站)

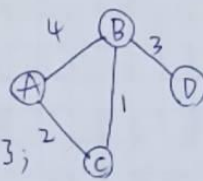
0  $\rightarrow$  4  $\rightarrow$  1: 0.4

|   | (0) | (1) | (2)      | (3) | (4)     |
|---|-----|-----|----------|-----|---------|
| 0 | 0   | 8   | $\infty$ | 9   | 4 (min) |
| 1 | 0   | 8   | 5        | 9   | 4       |



#### 7-16 最短路徑的樹

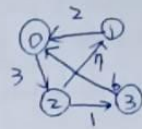
- 記錄 vertexSet & weight. 每回合加一個權重最小的為轉機點再更新權重記錄
- do { Add v into vertexSet  
 $weight[u] = \min \{ weight[u], weight[v] + edgeWeight[v, u] \};$   
 for u not in vertexSet  
 if ( $weight[u] < cheapest$ )  $v = u$ ;  $cheapest = weight[u]$   
 } while ( $cheapest < \infty$ )
- 亦可利用相鄰串列





## 7-20 任兩點之間的最短路徑 All Pairs

- 求兩點之間的最短路徑，目的同於 Dijkstra 但亦可從任何起點，原理相同但



利用矩陣

- 法1. 呼叫  $n^2$  每次以不同點作為起點  $\begin{matrix} 0, 1 \\ \vdots \\ 1, 0 \end{matrix}$   $0 \rightarrow 2 \rightarrow 1$

- 法2. 將已知資訊重覆利用，而不用重新計算

|   | 0 | 1  | 2 | 3 |          |
|---|---|----|---|---|----------|
| 起 | 0 | 10 | 3 | 4 | → 各擇一，若起 |
| 1 | 2 | 0  | 5 |   | 終點不一     |
| 2 | 4 | 7  | 0 | 1 | 則可合併。    |
| 3 | 6 | 16 | 9 | 0 |          |

## 7-21 Floyd 演算法的範例

- 起點、終點、跳板，可排除循環問題因只求最小
- 若無向圖，其作法相同且以對角線對稱且只需要算一半。
- Topological: 針對有向圖、無權重、每個點有 label
- spanning tree: 針對無向圖、有權重、求權重總合
- shortest path = 類似於生成樹，求權重合最小的路徑。

## 7-22 以 A\* 演算法求最短路徑

- Dijkstra 功能強大，但實際應用時通常不用那麼多功能通常是固定起點 & 終點，且能省略不必要的計算
- Best-first search = 一個已排序的路徑大小順序  
Dijkstra = 記錄
- Greedy = 選擇最好的方向進行，但不能保證最短

- $g(v)$  (從起點累計的距離) +  $h(v)$  (與目前有關的猜測)

- $h$  的計算不 - ex = 小於線段

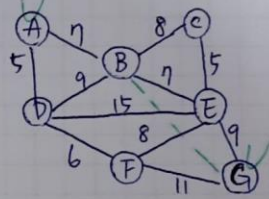
$h(B)=4, h(C)=12, \dots$

- $f(B) = g(B) + h(B) = 7 + 14 = 21$ .

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

∴ 預估最短路徑以利找到正確方向

- 要求多次、得真正最小路徑, [走到終點  $f(G) = g(G) = 11 + 11 = 22$




## 8 圖形問題

### 8-1 初探關鍵路徑分析 似拓撲

- Activity-on-Arrow (AOA) Network 耗時
- 起床  $\xrightarrow{\text{母: 讀書}} \text{吃飯} \xrightarrow{\text{母: 吃}} \text{上學}$  邊有向、點某事件  
某時刻  $\text{父: 起床}$   $\text{父: 吃}$  必無循環
- Path length = 所有權重和
- 若第一人起床就開燈直到最後一人出門，求開燈時長。  
得一路徑 Critical Path

### 8-2 關鍵路徑分析方法

- 求每個活動開始的時間且耗時固定 但因有誤差。  
故求最早 & 最晚的開始時間  $l_{\text{earliest}} = l_e[x]$
- 由後往前求  $l_{\text{ate}}[x]$  &  $l_e[y]$   $l_{\text{earliest}} l_e[y]$   
並由路徑取  $\max \& \min$   A, B, C 的  $\max - \min$   
 $l_{\text{ate}}[x] - l_e[y]$  越小 (彈性小) 有優先權
- 若要縮短總時長，從關鍵活動著手。  
所有路徑都會經過的路徑

### 8-3 關鍵路徑分析的向前階段

- 利用拓撲有向 = 找到最後的點並拿掉
- 改從左側開始 (in-degree=0) 求每個連出去的邊。  
求其最早發生時間 & 下個點發生時間。
- 若某點有多個 in-degree，則求開始時間時應取最晚者。因為要前面的都完成才能統一開始
- 資料結構有主陣列，後方接 out degree 的邊 & 權重。  
亦要有一份以線為主，記錄其起點 & 終點。  
主陣列存 in-degree 數



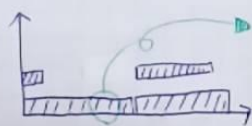


- $ee=0, 5, 4, 6, 7, 9, 16, 15, 18$  全部完成最短時間  
 $ea=0, 0, 0, 5, 4, 6, 7, 7, 9, 16, 15$

#### 8-4 關鍵路徑分析的向後階段

- 最晚開始時間 = 最晚結束時間 - 耗時  
 某點有多個 in-degree, 則取最早的 "最晚開始時間"
- 從尾做回來主陣列存 out degree 數, 後方記錄前一個節點 & 權重
- 用 queue or stack 記錄均可, 並依序處理 並要改變主陣列的數字記錄
- Free float = 最晚 - 最早  $\neq 0$  故有彈性, 不為關鍵節點。

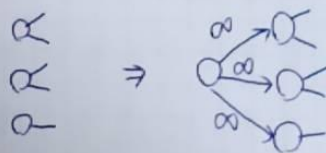
• 圖示:



無重複, 故可將其縮時  
即可縮短整體時間

#### 8-5 最大流量的問題 Maximum Flow Problem

- 亦為有向圖且類似於前者  
 $\hookrightarrow$  抽水龍頭, 抽水桶  
 線段上的數字為水管口徑  
 求某時間內最多能流多少水
- Capacity  $c(u, v)$  容量 flow  $f(u, v)$  流量  
 single source (sink) 為單一起點終點 但沒必要強調, 但若有多个起點 就在前面多加一個起點  
 並將出水量設  $\infty$  即可。



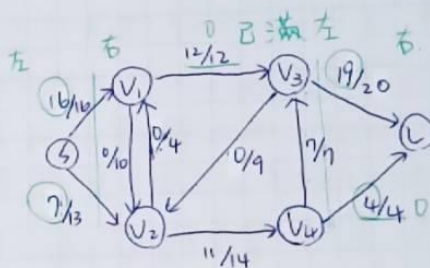
- Maximum-flow min-cut theorem

在圖上的線上切出一條線，則被切到的線段總稱為 cut，並將同一方向的數據進行總和（往右1 + 往右2... - 往左1 - 往左2...）描述某一 cut 能往哪個方向進帳多少，但找最大流恰為 min cut

- 但當線段太多，則計算紊亂，結果會被取代

### 8-6 找出最大流量

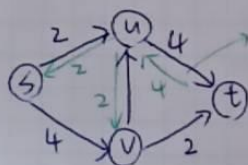
- flow = 23
- 求 cut 的邊即可得知  
並不斷更新可輸送的最大流量



### 8-7 剩餘圖

- 有一起點 & 終點，求其最大載容量
- Ford-Fulkerson algorithm  $\rightarrow$  Residual graph  
residual capacity = 在找答案的過程中隨時掌握剩下的容量

- Edmonds-Karp algorithm = 有許多 Heuristic 策略以找到路徑  
不能保證效果最好，但較靠直覺  $\rightarrow$  放大 augmenting path  
(排除一些不能增加流量的路徑)



反向，因為怕原本找的路徑不是最好的，所以反悔

- $s \rightarrow u \rightarrow t = c(s, u) = 2, c(u, t) = 4 \rightarrow flow(u, v) = 2$
- $s \rightarrow v \rightarrow t = c(s, v) = 4, c(v, t) = 2 \rightarrow flow(u, v) = 2$
- $s \rightarrow v \rightarrow u \rightarrow t = flow(u, v) = \min\{2, 6, 2\} = 2$

### 8-8 以 Ford Fulkerson 演算法求最大流量

- 將資料存於矩陣

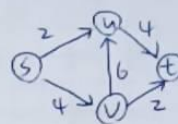
$$s \rightarrow u \rightarrow t \quad C_f(P) = 2$$

先找某路徑上的最小運

輸量，並將路徑上所有運輸量扣除最小值。

並將反向記錄最小值

| $C_f$ | $s$ | $u$ | $v$ | $t$ |
|-------|-----|-----|-----|-----|
| $s$   |     | 2   | 4   |     |
| $u$   | 2   |     |     | 4   |
| $v$   |     | 6   |     | 2   |
| $t$   |     | 2   |     |     |

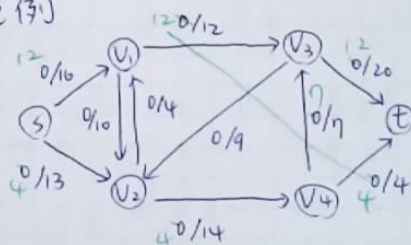


### 8-9 以 Edmondskarp 演算法求最大

- 加快前一個方法所不能做到的部分，在找路徑多了 heuristic 用以選擇路徑 (最大容量優先) 但無法保證
- $s \rightarrow v \rightarrow u \rightarrow t$ ,  $C_f(P) = 4$ ,  $s \rightarrow u \rightarrow v \rightarrow t$   $C_f(P) = 2$ .
- 另一策略 = 寬度優先 若一條路徑不長但可以有很大容量即可快速運作. ex:  $s \rightarrow v \rightarrow u \rightarrow t$

### 8-10 Edmondskarp 演算法的範例

|       | $s$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $t$ |
|-------|-----|-------|-------|-------|-------|-----|
| $s$   |     | 16    | 13    |       |       |     |
| $v_1$ | 2   |       | 10    | 12    |       |     |
| $v_2$ |     | 4     |       |       | 14    |     |
| $v_3$ |     |       |       |       |       | 20  |
| $v_4$ |     |       |       | 7     |       | 4   |
| $t$   |     |       |       |       |       |     |



- 除陣列，還有一個 queue 為了 BFS 先進先出

$\begin{matrix} \text{頭} & v_1 & \Rightarrow & v_2 & \Rightarrow & v_1 \rightarrow v_2 & \Rightarrow & v_1 \rightarrow v_3 \\ & v_2 & & v_1 \rightarrow v_2 & & v_1 \rightarrow v_3 & & v_2 \rightarrow v_1 \\ \text{tail} & & & v_1 \rightarrow v_3 & & v_2 \rightarrow v_1 & & v_2 \rightarrow v_4 \\ & & & & & v_2 \rightarrow v_4 & & v_1 \rightarrow v_2 \rightarrow v_4 \end{matrix}$

- 形成 min cut 因流量全部用完，故得  $12 + 7 + 4 = 23$ .

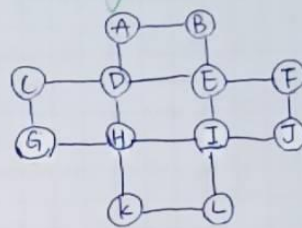


### 8-11 Eulerian Circuit → Euler tour

- 解開最原始的七橋問題，利用深度優先的方法  
同一起終點 (walk 不用) 並一條路走完 不可以有奇數 degree
- 若為 Hamilton circuit <sup>Decision Problem</sup> 則求一路經過所有點

$ABEDA \Rightarrow AB \downarrow EFJI \downarrow EDA$   
 $\quad \quad \quad [F, I] \quad \quad \quad [H, L]$

$\Rightarrow AB \downarrow EFJI \downarrow H \downarrow DC \downarrow GH \downarrow KL \downarrow IE \downarrow DA$

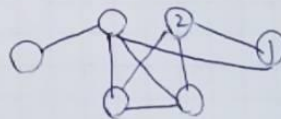


### 8-12 漢彌爾頓迴路

- 暴力法：每個可能的走法都走一遍，保留最好者
- Greedy = 快速但不是一個好的答案
- Branch-and-bound <sup>分支 上限</sup> = 品質 & 速度的折中
- ex = 有 16 個石頭，R 能拿 1 or 2 or 3 個，拿到最後一個即輸
- 非圖形塗色問題

### 8-13 非圖形塗色問題

- Sequential ordering = 先將顏色編上數字並每次都從小的開始選，使其相鄰點都不同色。能找出最少種顏色的種類 (BFS)
- 與 degree 數有明顯關係
- 法 2 從 degree 最多著開始著手。選色的原則同上。
- 但 2 種方法都不能保證為最佳解



#### 8-14 以貪婪策略染圖形

- 有本機會用最少顏色
- 從最多 degree 的點下手並依編號顏色最小開始選
- 若為完整圖 (任 2 點有一邊), 則其顏色種類為  $n$   
若為 cycle, 若點數為偶數  $= n/2$ , 奇數  $= n/2 + 1$   
若為 wheel (即為 cycle 多了中心), 則顏色數為 cycle 各加一



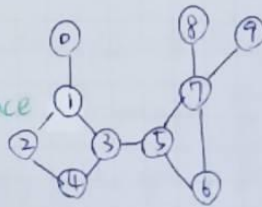
#### 8-15 雙連通圖 connected Graph

- 若為一網路線, 但不安全, 有些點被拿掉會摧斷網路線  $\rightarrow$  關節 Articulation point
- Bi-connected graph = 任一點被拿掉都不會
- 利用 BFS or DFS 找到關節 (是否為樞紐點是由其邊的點決定) 若沒有該點, 其旁邊的點是否還能到另一側, 若無法, 該點即為關節



#### 8-16 以深度優先走訪找出圖形的關節點

- 求 3 是否為一樞紐點, 用 DFS  
 $3 \rightarrow 1 \rightarrow 0$  記錄行走順序 會 backtrace  
 得知 0 會是一孤島, 故 1 為樞紐  
 $3 \rightarrow 1 \rightarrow 2 \rightarrow 4$  非孤島  $3 \rightarrow 1 \rightarrow 7 \rightarrow 5$
- 1, 7 為樞紐  
 $\downarrow \quad \downarrow$   
 0      8, 9

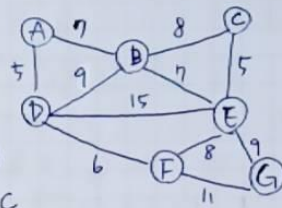


Prim's = Kruskal (結果)

- Shortest Path Tree vs Minimum Spanning Tree  
都會涵蓋所有點且邊最少，一個一個擴張。  
每次都加入最小邊，但 Prim 是求其總合。

### 7-17 Dijkstra 演算法的範例

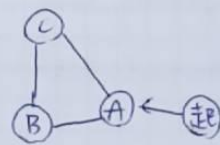
- 為其他演算法的基礎。
- 求  $A \rightarrow F$  的  $\min s+b$ ，若都以整個圖的最短邊作為組合到不同的點不一定正確。ex:  $A \rightarrow C$   
 $7+7+5 > 7+8$



- vertex set  $\rightarrow$  要轉機的城市, weight  $\{0, 0, \dots\} \rightarrow Ans$ .  
 $V = \{A\}$ ,  $wei = \{0, 7, \infty, 5, \infty, \infty, \infty\}$   
 $V = \{A, D\}$ ,  $wei = \{0, 7, \infty, 5, 20, 11, \infty\}$   
 $V = \{A, D, B\}$ ,  $wei = \{0, 7, 15, 5, 14, 11, \infty\}$   
 $V = \{A, D, B, F, E, C\}$ ,  $wei = \{0, 7, 15, 5, 14, 11, 22\}$

### 7-18 Dijkstra 演算法的正確性

- 在 ABCD 路徑中，ABC 為 prefix 且為  $A \rightarrow C$  的最短路徑
- 由  $A \rightarrow B$  求最短，故證  $A \rightarrow B < A \rightarrow C \rightarrow B$ . (pf  $A \rightarrow B < A \rightarrow C$ )



### 7-19 Dijkstra 演算法的應用

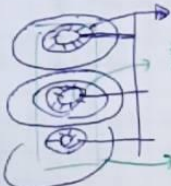
- 要規劃路線，將不同路線的路況或車程為依據並不斷合併
- 但耗時， $\therefore$  有些已知並非最佳路徑的組合，程式還是都要算。



## 9. 次要儲存體

### 9-1 次要儲存體 secondary storage

- 當資料量大時，要存放在外部空間。
- 越底部的資料可存較大空間 (磁帶...) 上方 (快速記憶體)。希望將大量資料存放在成本低、但速度慢的地方，而高成本、快速的地方用於執行最重要的部分  
ex: 程式 (上方) CPU time v.s I/O time (下方) <sup>硬碟</sup>

-  位置區分成很多磁區  
同心圓 = 磁軌，同步旋轉讀取。  
磁柱

- seek time 移動讀寫頭, latency time 旋轉, transfer <sup>量</sup>資料  
希望減少 seek time 以提高效率
- 作業系統能針對要讀寫的部分進行處理，並非處理整個檔案 Block access 希望能一次讀寫完防止一直移動讀寫頭。

### 9-2 硬碟存取

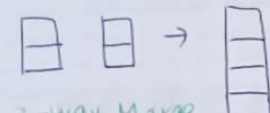
- 同一筆資料可能被存放在分散的位址，因不斷新增刪除故僅找未使用的記憶體進行存放 Cluster (非連續)
- I/O Buffer 緩衝區 (部分 memory) 放好給 CPU 讀  
將 Buffer 的資料寫進 disk
- I/O Processor 等待分配空間，DMA 是一記憶體，暫存要讀寫的資料

### 9-3 外部排序

- 因資料量過大或資料本來就存在外部檔案
- 一個 block 有 100 筆資料，將每筆利用任意方法排序再寫回檔案， $\Rightarrow$  再將兩<sup>2</sup>進行 Merge 合併。

$$10M + 10M = 20M.$$

### 9-4 多重管道合併

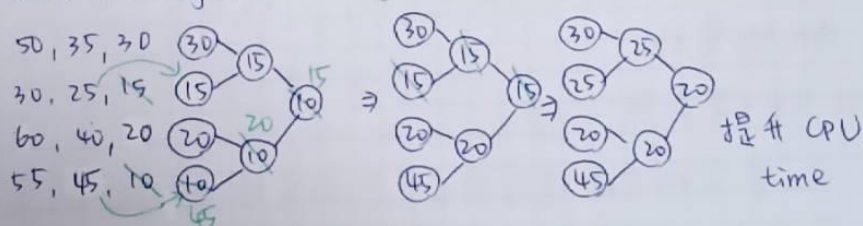
-  固定緩衝區 buffer 存取量  
若 buffer 空間多  $\Rightarrow$  空間換時間

2-way Merge

- $64 \text{ runs} \rightarrow 32, 16, 8, 4, 2, 1 \rightarrow \log_2 64 = 6+1$   $\because$  資料一開始分割排序也讀寫一次
- k-way merge ex = 4 個資料同時進行合併

### 9-5 以選擇樹合併

- $O(k) \rightarrow O(\log k)$  利 binary tree 記錄最小者



- 實作時將 4 個 buffer 編號 ABCD 並在樹狀結構上記錄在該 buffer 的 index.
- index 的計算可用 heap 的方式  $\begin{matrix} 0 \\ 1 \ 2 \\ 3 \ 4 \dots \end{matrix}$

將 tree 記錄於陣列

#### 9-6 檔案結構中的欄位表

- 檔案有結構的目的是為了方便取得某部分的資料
- 一筆資料 (Record) (ex: 一個球員) 由一個欄位 (field) 組成 (名稱 & 內容)
- 可預留空間 (ex: `char id[10]` 固定大小)
  - 在資料前放好長度 ex: 6Jordan
  - 加入間隔符號
  - 將檔案內資料完整化 ex: `lastname = John`  
當用於不同應用程式間交換資料

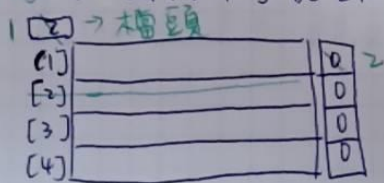
#### 9-7 檔案結構中的紀錄表示法

- file header = 存放整筆資料最前端, 用於區分每筆資料, 表 offset 位移量 (位移量以讀取下一筆資料)
- 將 offset 統一存在一 file 中 (index) 記錄  
當要讀取某筆資料時, 利用 index file 直接讀取

#### 9-8 刪除檔案的一筆記錄

- 要刪除檔案中第二筆資料:
  - 將 3 → 2 4 → 3 做搬動 (逐筆)
  - 將最後一筆遞補上刪除位置

3. 用一鏈結串列記錄 空缺位置 offset 偏移量



刪除 2 → 刪除 1 → 新增 (讀檔頭知 [1] 可放)

→ 新增 (讀 [1] 的尾部知 [2] 可放)



#### 9-9 以鍵值將檔案排序

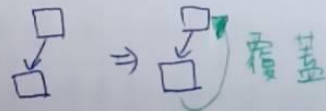
- 從 IN-PUT file 根據 key 排序後寫到 OUT-PUT file  
RRN 記錄第幾筆 (offset 位移量) 可再移動讀寫頭至指定資料讀寫
- 利用 key Array (RRN) 做為資料索引

#### 9-10 索引的基本觀念

- Secondary index (索引) Primary index (整份資料的主索引)  
ex: 同分數者記一起 ex: 學號
- 若不想每資跑都重建索引則將索引寫成檔 Index file
- Balanced Search Tree = 可直接存在一檔案中, 燕-來自  
= 元搜尋樹 (B-Tree) 2-3 Tree
- Hash = 未排序的索引

#### 9-11 以 B-Tree 作為索引

- = 元搜尋樹衍生 2-3 Tree  $\Rightarrow$  B-tree 衍生 2-3 tree  
Balance m-way search tree (不 $\times$  2-3-4)  
m 越大, 搜尋效率越好, 分裂次數越少
- 同分者可利用 bucket 記錄, 另建一同分者的 file
- 可在檔頭設 root 以利走訪 緩衝區因要走訪的緣故只需兩個節點大小



- 在 buffer 中, 可用陣列記錄資料, 且記錄不結點可用 index

### 9.12 各種索引的結構

- 變形:  $B^+tree$ : 當資料量(key)少於  $\frac{2}{3}$  時就要合併樹高每層空間使用效果好

$B^+Tree$ : 所有資料都存在最底層(有順序的鏈結串列)(不同結點亦串接)而往上建的是索引, 以利搜尋走訪

- Hash: 亦可作索引, 將 key 運算後存入檔案  
也可用 bucket

- Extensible Hash (2→4→8→16) Linear (2+1+1+1)  
hash 大小可動態控制

- $R^{regen}-tree$ : 由 B-tree 衍化用於搜尋幾何的平面圖形  
矩最小的矩形包住範圍, 且矩形只需記錄左上  
及右下的坐標

- 先求目標點與矩形的最短距離, 並針對最近的  
矩形內部進行比較