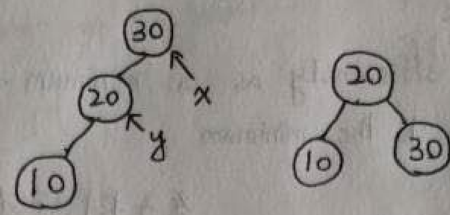
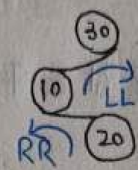
LL Rotation

```
nodeType rotateLL(nodeType x)
{
  nodeType y = x->left;
  x->left = y->right;
  y->right = x;
  return y;
}
```



LL . RR . LR . RL



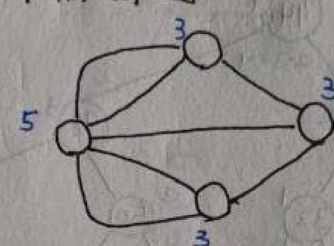
7橋問題

$G = \{V, E\}$

$V(G)$: vertex set

$E(G)$: edge set

degree number of edges



2 個 class

橋 . 陸地

$$\sum_{v_i \in V(G)} \text{degree}(v_i) = |E(G)| \times 2$$

Vertex types

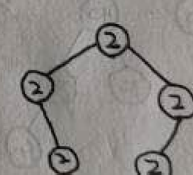
Odd or even degrees

degree 一定是偶数

Eulerian path (trial) / Euler walk

visits every edge exactly once

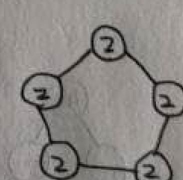
0 or 2 nodes with odd degree



Eulerian circuit (cycle) / Euler tour

begin and end at the same vertex

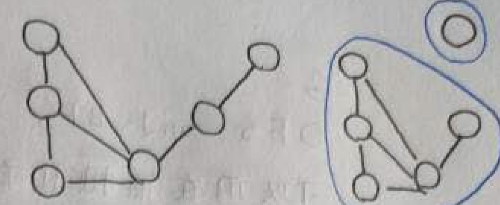
0 node with odd degree



connected graph 無向圖

There is a path between any two vertices

disconnected graph



Connected graph

connected components

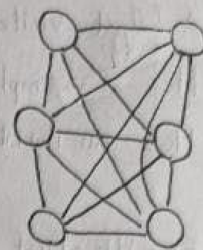
Two subgraphs

Complete graph |E|

There is an edge between any two vertices

Strong connected graph 有向圖

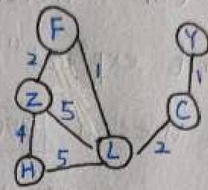
For any two vertices on a diagraph, there is a path from one vertex to the other



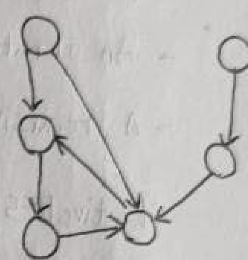
Complete graph

Weighted graph

the edges have numeric labels



weighted graph



strong connected graph

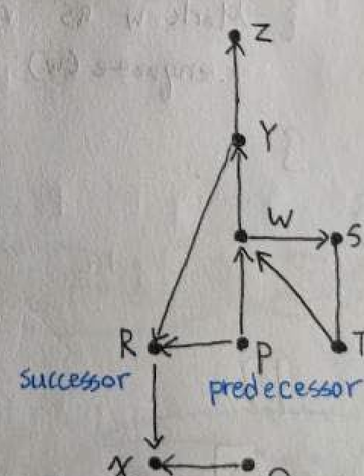
ADT graph Operations

int numVertices()
int numEdges()
int getNumVertices()
int getNumEdges()
int getWeight(Edge e)
void add(Edge e)
void remove(Edge e)
bool isEdge(Vertex u, Vertex v)
int getDegree(Vertex v)
bool isConnected(Graph g)
edgeList traverse(Graph g)

Most common implementations of a graph

Adjacency matrix

Adjacency list



	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

In-degree

被指向的

Out-degree

指向的

traverse(g) : $O(|V|^2)$ $9^2 = 81$

相鄰矩陣

DFS and BFS

可以用在無向和有向

Depth-First Search (DFS) Traversal　深度優先走訪

- A "last visited, first explored" strategy
- Has a simple recursive form　stack
- Has an iterative form that uses a $\boxed{\text{stack}}$

Breadth-First Search (BFS) Traversal　寬度優先走訪

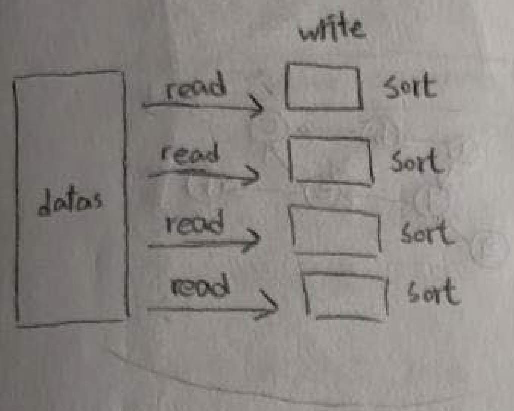
- A "first visited, first explored" strategy　queue
- An iterative form uses a $\boxed{\text{queue}}$
- A recursive form is possible, but not simple

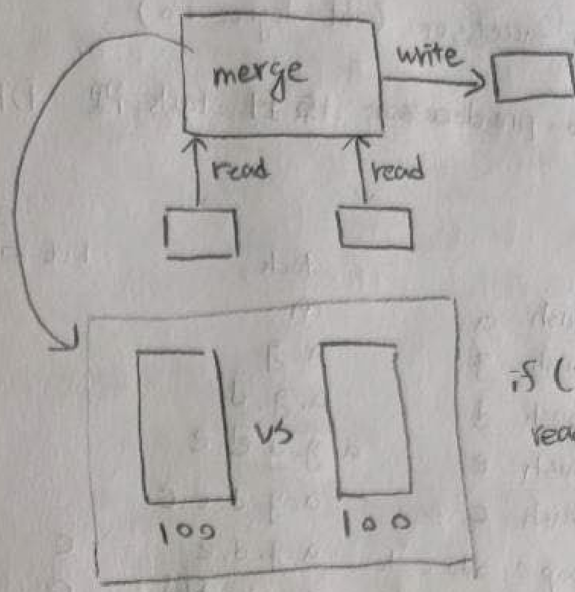
```
iterativeBFS(Vertex v)
q.createQueue();
q.enqueue(v);
Mark v as visited;
while(!q.isEmpty())
{   q.dequeue(u);
    for(each unvisited vertex w adjacent to u)
    { Mark w as visited;
      q.enqueue(w);

    }
}
```

Graph Traversal Sequences

DFS的序列

BFS的序列　visited的不用在拜訪





if (isEmpty)
read next 100 piece of data

merge 兩兩合併

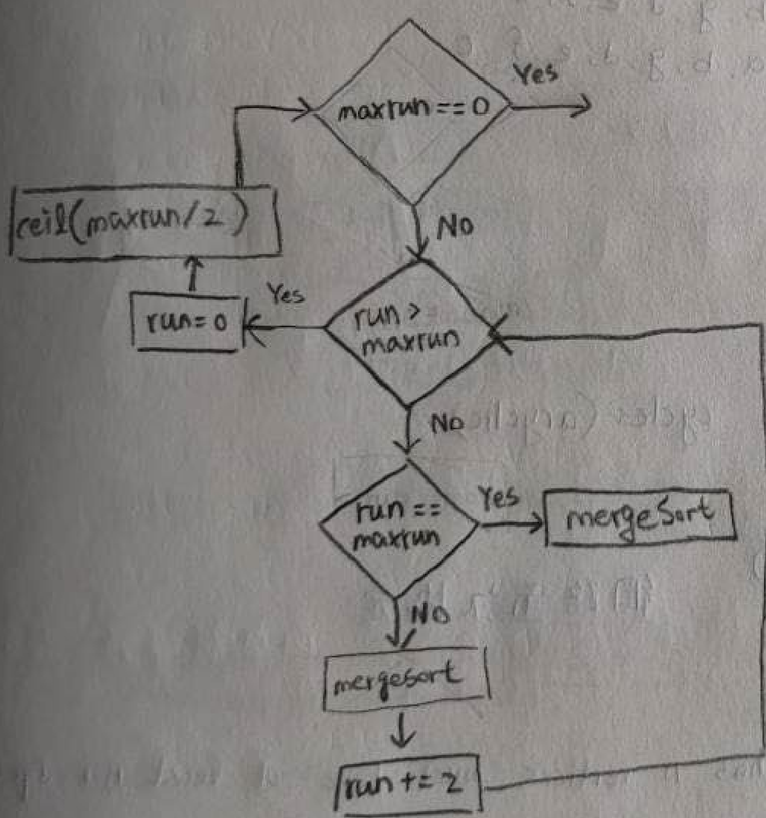
1. 切割成一小塊一小塊
2. 2筆2筆合在一起

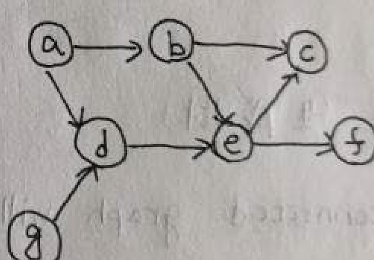
merge → mergesort
還剩幾筆

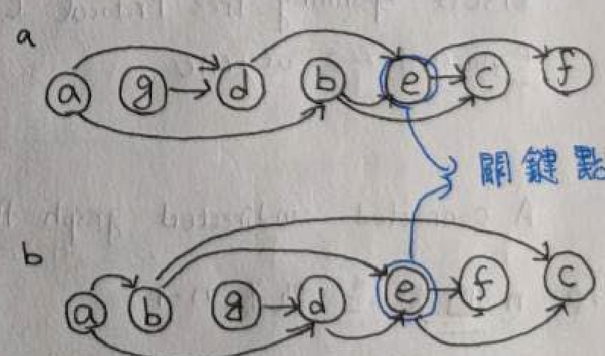
Topological order 拓樸

directed graph without cycles

(Acyclic Digram or Directed Acyclic Graph, DAG)





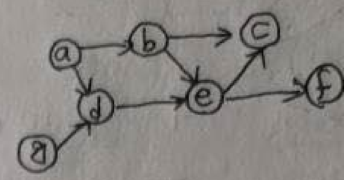


關注資料重點在點 Activity-on-Vertex (AOV) Network

重點在邊 Activity-on-edge (AOE) Network

no successor (out-degree = 0)

no-predecessor 丟到 stack 裡 DFS



	stack	List aList
push a	a	
push g	a, g	
push d	a, g, d	
push e	a, g, d, e	
push c	a, g, d, e, c	
pop c, add c	a, g, d, e	c
push f	a, g, d, e, f	c
pop f, add f	a, g, d, e	f, c
pop e, add e	a, g, d	e, f, c
pop d, add d	a, g	d, e, f, c
pop g, add g	a	g, d, e, f, c
push b	a, b	g, d, e, f, c
pop b, add b	a	b, g, d, e, f, c
pop a, add a	(empty)	a, b, g, d, e, f, c

Spanning tree 生成樹

undirected connected graph without cycles (acyclic)

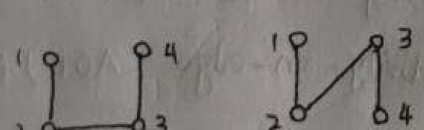
CISCO Spanning Tree Protocol (STP) 網路通訊協定

Connected ⟷ acyclic

A connected undirected graph that has n vertices must have at least n-1 edges

定義 n 個點，邊數為 n-1

同構 isomorphic



Various vertex labeling n^{n-2}

2 nodes → $2^{2-2} = 1$

3 nodes → $3^{3-2} = 3$

4 nodes → $4^{4-2} = 16$

Prüfer sequence 普呂弗序列

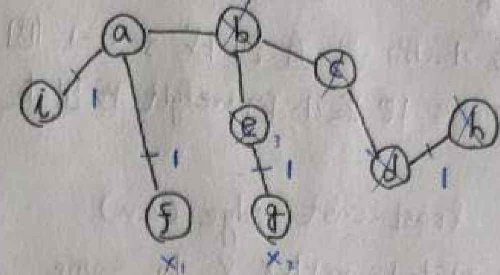
可以把樹存成字串

$\boxed{\text{Tree} \rightarrow \text{Prüfer sequence}}$

9-2 =7 會產生7個點

先找樹葉 degree = 1 的

拿掉順序最小的



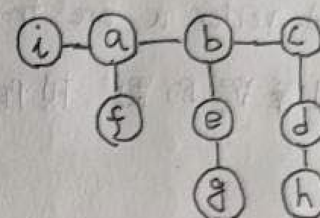
a, e, b, d, c, b, a, 直到剩下最後一個邊

a, e, b, d, c, b, a

degree [a, b, c, d, e, f, g, h, i]
1 1 1 1 1 1 1 1 1
+2 +2 +1 +1 +1
3. 3. 2. 2. 2. 1. 1. 1. 1

⇒ [a, b, c, d, e, f, g, h, i] 先拿 degree 最小的最小字符
3 3 2 2 2 1 1 1 1

a - f
e - g
b - e
d - h
c - d
b - c
a - b



DFS in iterative form (stack)

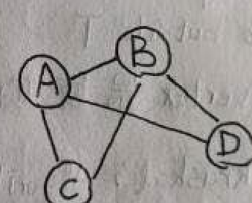
visited

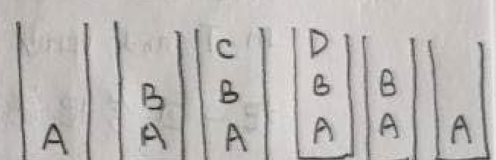
T A → B → C → D

F B → A → C → D

F C → A → B

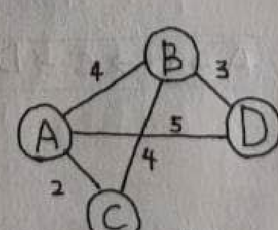
F D → A → B





BFS 把 stack 改 queue 就好

Minimum Spanning Tree $\boxed{\text{很基本}}$



DFS : 4+4+3 =11

BFS : 4+2+5 =11

MST : 4+2+3 =9

Cost of spanning tree

- Sum of the edge weights

Other variations

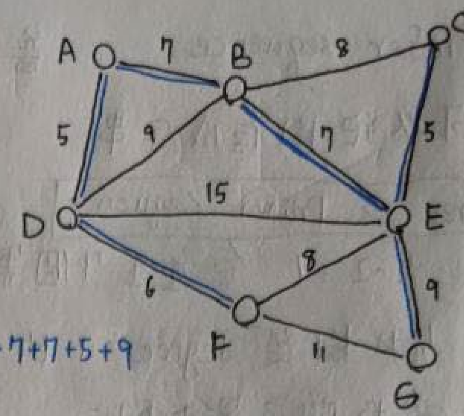
- (minimum) steiner tree 區域
- K-minimum spanning tree 整個拿出來看，哪裡適合

Prim's Algorithm 普林演算法

每次找最小的邊直到找了 n-1 個邊

priority queue 把最小的 weight 找出來

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

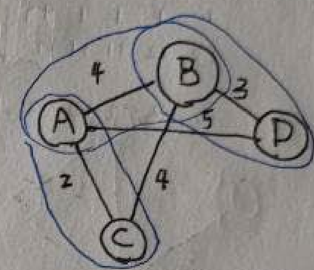


MST : 5+6+7+7+5+9 =39

Kruskal's Algorithm

Find the least-cost edge (v,u) where vertex v and vertex u are from two different trees

一個單獨的點,視作一棵最小生成子數



AC
BD
AB(or BC)

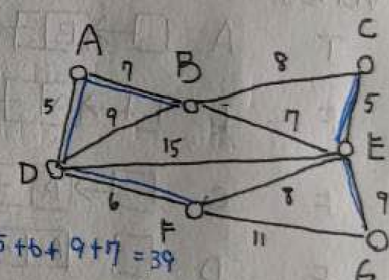
4棵樹 → 1棵樹

Sollin's Algorithm

Find the least-cost edge (v,u) where vertex v is in T and vertex u is outside T

第一回合把每個 vertex 最短的邊連起來

加快 kruskal's Algorithm



MST 5+7+5+6+9+7 = 39

最短路徑

Dijkstra's Algorithm

Shortest Paths

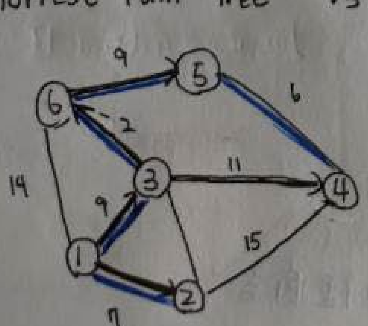
A → B → C A到C如果是最短

A到B一定也是

找直接到和最短的 $\boxed{A \to B}\to C \to D \to E$ A到E是最短 前面都必須是最短

A到B、A到C…都是

Shortest Path Tree vs Minimum Spanning Tree

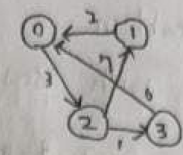


最小生成樹:保證整個拓樸圖的所有路徑之和最小,但不能保證任意兩點之間是最小路徑

最短路徑:從一點出發,到達目的地的路徑最小

All Pairs Shortest Paths　Floyd's Algorithm

考慮一行一列



D^{-1}	0	1	2	3
0	0	∞	3	∞
1	2	0	~~∞~~ 5	∞
2	∞	7	0	1
3	6	∞	∞	0

A* algorithm　Best-First Search　Expected total cost $f(v) = g(v) + h(v)$

Dijkstra's algorithm: favor vertices close to the origin

Greedy best-first search: favor vertices close to the goal

有起點有目的地

$f(n) = g(n) + h(n)$

$g(n)$: 從起始點到目前節點的距離

$h(n)$: 預測目前節點到結束點的距離（此為A*演算法主要評價公式）

$f(n)$: 目前節點評價分數

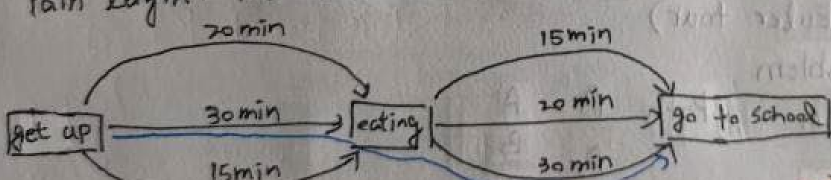
Activity-on-Edge (AOE)

Directed edge

vertex: event to signal the completion of certain activities

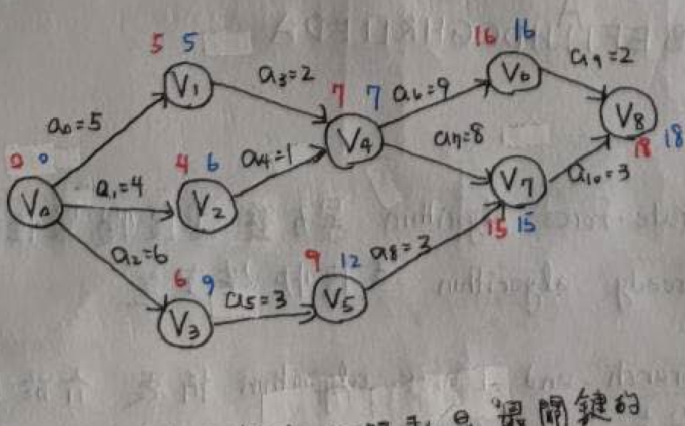
Edge weight: the time required to perform an activity

Path length: the total time from the start to the last event



關鍵路徑

取 maximum.



la − ea is called (total) float or slack

	ea	la	la − ea
[0]	0	0	$\boxed{0}$
[1]	0	2	2
[2]	0	3	3
[3]	5	5	$\boxed{0}$
[4]	4	6	2
[5]	6	9	3
[6]	7	7	$\boxed{0}$
[7]	7	7	$\boxed{0}$
[8]	9	12	3
[9]	16	16	$\boxed{0}$
[10]	15	15	$\boxed{0}$

critical activity

最關心的（最不能 delay 的）

走哪條↓

在關鍵路徑上的活動是最關鍵的

所有路徑中都會經過的更是關鍵中的關鍵

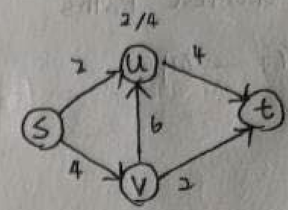
到該點時的時間

ea → 最晚發生的時間

la → 最晚開始的時間

Maximum Flow Problem

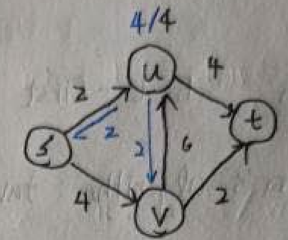
flow network G with source s and sink t



Maximum-flow min-cut theorem
(最大流就會是最小的cut)

Ford-Fulkerson algorithm, 1955

– Residual graph 剩餘圖



↓ 送過去,將來如果要反悔,可以照這個返回

如果走錯能夠反悔

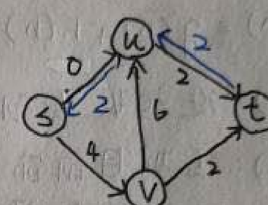
Edmonds-Karp algorithm, 1972

Heuristic to find augmenting path

Gr	s	u	v	t
s	0	$\cancel{2}^{0}$	4	0
u	$\cancel{0}^{2}$	0	$\cancel{0}^{2}$	$\cancel{4}$ 2 0
v	$\cancel{0}_{2}$	$\cancel{6}$ 4	0	2
t	0	$\cancel{0}_{2}$	0	0

P: s → u → t

cs(P) = 2



等於0的邊越多,越早結束

Edmonds-Karp algorithm

Find a path P from s to t by a heuristic

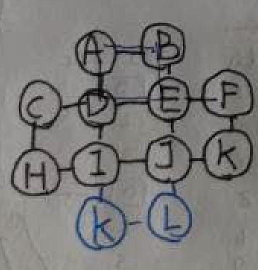
建立在前者,希望能更快找到路徑

Heuristic 1. max-capacity first

Heuristic 2. breadth first

Eulerian circuit (Euler tour)

DFS-based Algorithm



ABEDA

× ABEFJIEDA

ABEFJIHDCGHKLIEDA

○

Hamilton circuit

Brute-force algorithm 暴力法 速度慢最佳解

Greedy algorithm 速度快 答案差

Branch-and-bound algorithm 折衷 介於 Brute-force 和 Greedy algorithm 之間

(Branch: 分支; bound: 上限)

vertex Coloring (Edge Coloring)

Sequential ordering algorithms — Heuristics for a specific ordering of vertices

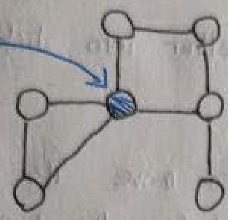
BFS — Welsh-Powell algorithm (greedy coloring)

degree 奇數會比偶數需要更多顏色 max-degree first

Articulation point

關節點

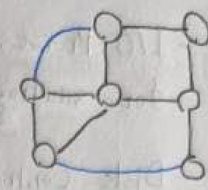
有一個點被拿掉會變成 disconnected



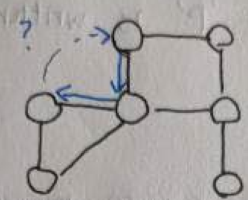
$\longleftrightarrow$

Bi-connected

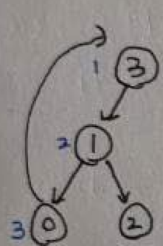
有一個點被拿掉不會造成 disconnected



- Graph traversal algorithm
- DFS-tree based algorithm

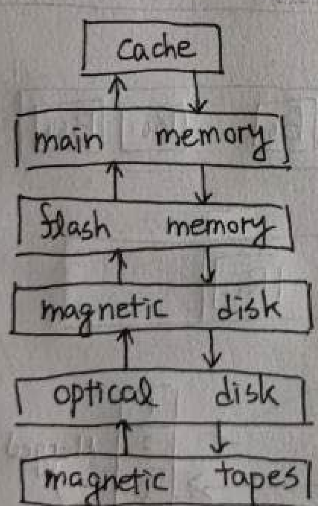


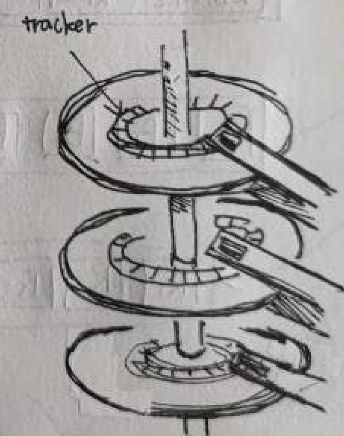
走不走的回來用來判斷是否為 Articulation point



帶不帶得來更小的數字

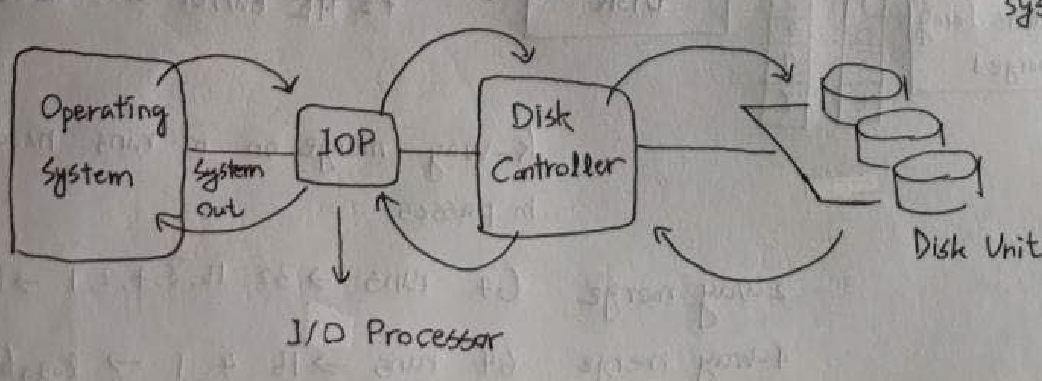
Secondary storage





CPU time vs I/O time (seek + latency + transfer)

Cluster: a number of contiguous sectors



System I/O Buffer

暫存的地方

I/O Processor

- Wait for an external data path to become available (CPU is faster)

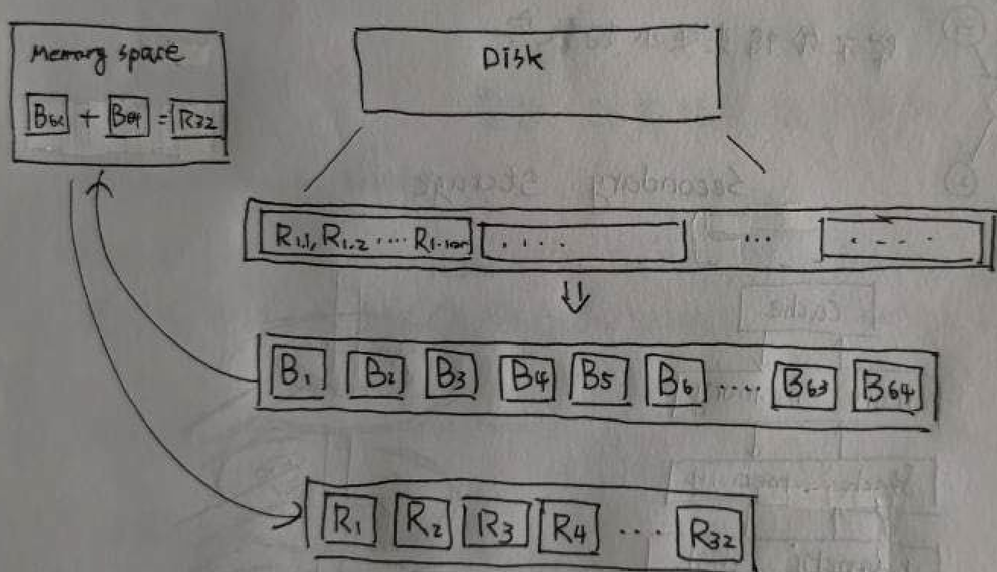
Direct Memory Access Input/Output DMA

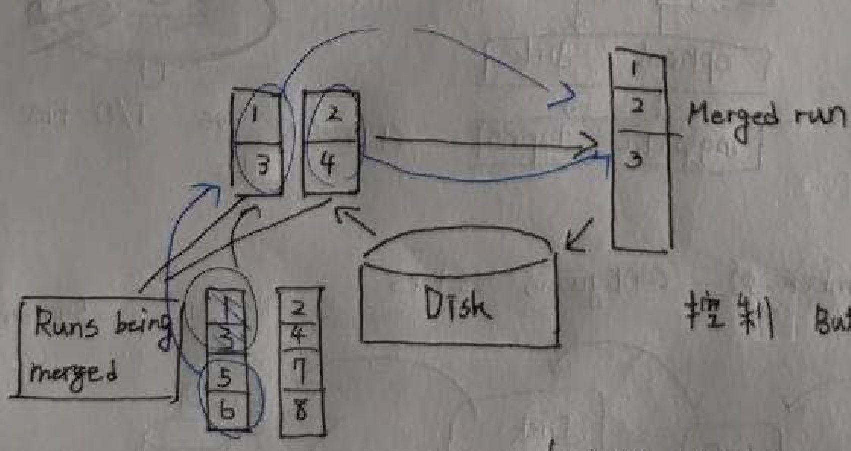
Disk Controller

- I/O Processor asks the disk controller whether into disk drive is available for writing
- Disk Controller instructs the disk drive to move its read/write head to the right track (seek time) and then wait for the desired sector (latency time)
- Disk spins to right location and 'P' is written (transfer time)

Internal Sort : main memory

External Sort : secondary storage + main memory





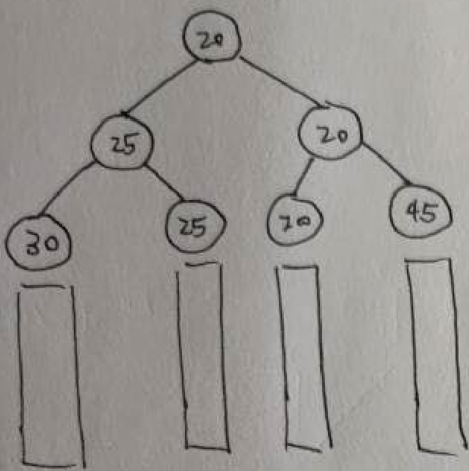
控制 Buffer size 在允許內

k-way merge on m runs needs $\log_k m$ passes

2way merge 64 runs → 32, 16, 8, 4, 2, 1 → $\log_2 64 = 6$ passes

4-way merge 64 runs → 16, 4, 1 → $\log_4 64 = 3$ passes

K-way Merge : select Tree



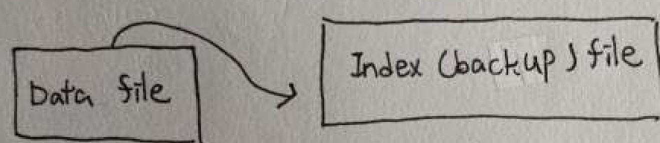
CPU-time

$O(\log k)$

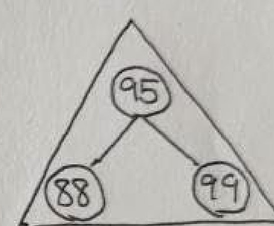
Record - Field
- name
- value

1. Force the field into a predictable length (byte)
2. Begin each field with a length indicator
3. Separate the fields with delimiters
4. Use a "fieldname = value" expression to identify each field and its content

Index Secondary Index



資料檔案與索引檔分開存



AVL Tree

B-tree Index

Balanced Search Tree

- If the entire tree fits the memory, no file is needed
- Otherwise, number of node accesses $\cong$ tree height
- $O(\log_2 n) > O(\log_m n)$ for $m > 2$

Balanced m-way search tree = B-tree of order m

- A generalization of 2-3 trees and 2-3-4 trees
- Order m: maximum number of children (m-1 keys)
- Given the order m and tree height h, the number of keys N in the B-tree $\le m^h - 1$