CH4、Graph Alogrithm

圖論相關的演算法

考題重點(目錄)：

1. DFS(Cormen 版本)[code]
2. Minimum Spanning Tree[證明]
3. Shortest Path Problem
4. Flow Network[ex]
5. 其他問題

DFS 及其應用

一、Depth First Search 的目的為將 Graph 上的點，走訪過一次，以：若有未 visit 的鄰居，則去 visit，若無，則退回上一步

二、DFS 時需要的變數

假設考慮某一點 u

1. color(u)：目前 u 的狀態：
   (1) white：初始值，尚未被 visit
   (2) grey：被 visit，但未 finish
   (3) black：已 finish
2. d(u)：discover time，即第一次被 visit 的時間
3. f(u)：finish time，即 finish 的時間

三、DFS 的演算法(p4-6)

1. DFS(G)：自 G 中任一點起做 DFS
2. DFS-visist(u)：去 visit 點 u

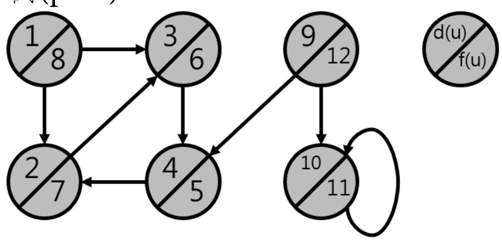
程式：

```
DFS(G)
{
        for each u 屬於 V[G]
                color(u) <- white;        //初始化

        time <- 0;            //Global clock，記錄目前做到第幾步

        for each u 屬於 V[G]
                if(color(u)=white)
                        DFS-visit(u);       //Graph 中若有尚未 visit 的點，則 visit 之
}

DFS-visit(u)
{
        color(u) <- grey;
        time <- time+1;
        d(u) <- time;       //被 visit 時的狀態設定

        for each v 屬於 adj(u);
                if(color(v) = white)
                        DFS-visit(v);       //去 visit 尚未被 visit 的鄰居

        color(u) <- black;
        time <- time+1;
        f(u) <- time;       //已 finish 的狀態設定
}
```

Time Complexity：Theta(V+E)　*//Cormen 寫法，但加入絕對值較佳*

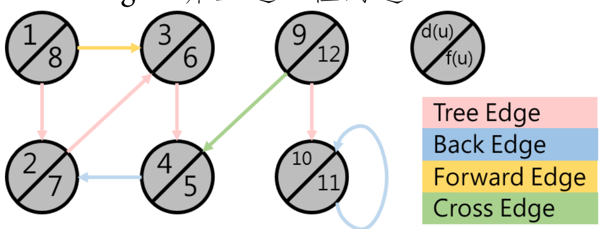
Note：此為將 Graph 整個 Scan 一次的複雜度

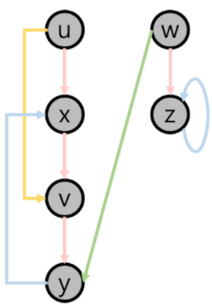
例(p4-8)：



四、DFS 的應用：邊的分類

1. 對於一種 DFS 的過程，可將 Graph 上的邊，分成以下 4 類：
   (1) Tree Edge：u 透過(u, v)去 visit v => (u, v)為 Tree Edge
       Note：可將 visit 的過程表示成數個 DFS Tree
   (2) Back Edge：不為 Tree Edge，但在 DFS Tree 上，由子孫->祖先的邊
       (Self-loop 亦為此 Edge)
   (3) Forward Edge：從祖先->子孫的邊
   (4) Cross Edge：非上述三種的邊





2. 在實務上，可藉由觀察以下特性，在 DFS 時，那可完成邊的分類：
   (1) Tree Edge：u(灰) -> v(白)
   (2) Back Edge：u(灰) -> v(灰)
   (3) Forward Edge：u(灰) -> v(黑)，且 d(u) < d(v)
   (4) Cross Edge：u(灰) -> v(黑)　，且 d(u) > d(v)

例(96 台大)(p4-12.2.3)：

五、DFS 的應用：判斷 G 是否為 Acyclic

有 Back Edge ⇔ 有 Cycle

演算法：p4-14.2.4(94、99 台大)

(在 DFS-visit(u)的 line 10~11 判斷是否有 Back Edge

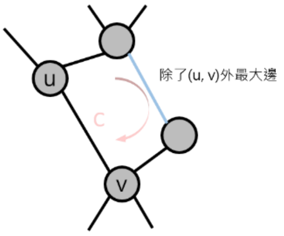
Time Complexity

1. 有向圖：Theta(V+E)
2. 無向圖：Theta(V) *//在無向圖(只有 Tree/Back Edge)中，最多看到第 V 個邊，即知其有 Cycle*

Minimu Spanning Tree

一、假設(u, v)是 G 中 weight 最小的邊，則(u, v)必在 G 的某一個 MST 中

證：設(u, v)不在 G 的任一個 MST 中，則任取一個 G 的 MST：T



將(u, v)加入 T 中，形成一個 Cycle C，令 e 為 C 中除了(u, v)外的 weight 最大邊，將 e 自 C 中去除，則形成一個 Spanning Tree T’，wt(T’) <= wt(T)，結論矛盾

<：可靠出一個比 MST weight 更小的 ST => 與 T 為 MST 矛盾

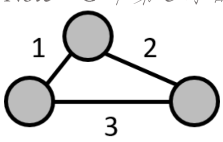
=：可造出一個有(u, v)的 MST => 與假設(u, v)不在任一 MST 矛盾

二、設(u, v)是 G 中第 2 小的邊，則(u, v)必在 G 的某一 MST 中

證：因為一個 Cycle 的邊數 >= 3　　*//利用(u, v)為第 2 小的邊*

所以 C 中必有一個邊 e，其 weight >= (u, v)的 weight

Note：G 中第 3 小的邊不一定在 G 的某一 MST 中

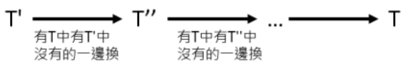


三、Kruskal’s 演算法的正確性證明(p4-25.3.3)

概念：T：用 Kruskal 找出的 Spanning Tree

T’：真正的 MST

1. if T=T’ => ok
2. else => 必有邊在 T 有，但在 T’沒有



且其過程 weight 不會上升 => 可證 wt(T’)=wt(T) => T 是 MST

Flow Network

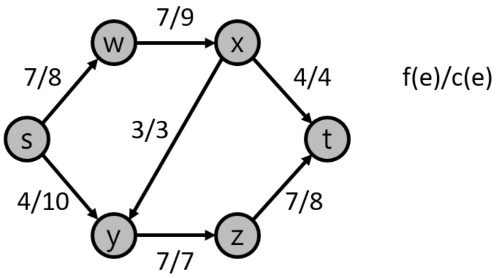
一、Def：Flow Network 是一個有向圖 G=(V, E)，滿足：

1. 只有唯一一個 in-degree=0 的點 s(source)
2. 只有唯一一個 out-degree=0 的點 t(sink)
3. 對於每個邊 e 屬於 E，定義一個 Capacity c(e)>=0

另外定義一個 Flow 函數 f(e)，滿足：

1. 對每個點而言，流入=流出
2. 對每個邊而言，0<=f(e)<=c(e)

例：



二、Max-Flow Problem

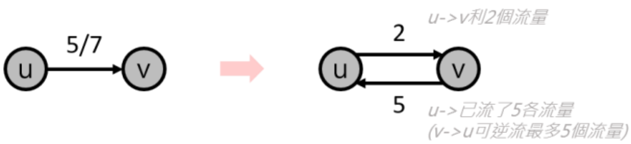
Input：一個 Flow Network G=(V, E)

Output：此 Network 的最大流量(source 流出流量和的最大值)

例：上例的 Network 的 Max-Flow 即為 11(即 7+4)

三、Ford-Fulkerson

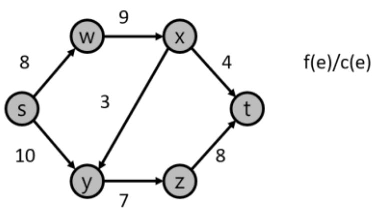
1. 使用 Residual Network 表示法



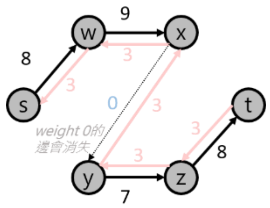
2. 自 s->t 找一條 path P，令 P 上最小的 weight 為 a。順向 P 上的每個邊：weight-a；逆向 P 上的每個邊：weight+a
3. Ans=指向 s 的邊的 weight 和

Time Complexity：O(|f*| E) //|f*|為最大流量的值

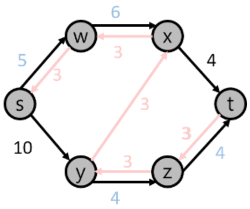
例：用 Ford-Fulkerson



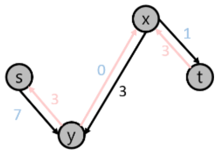
1. 第一條：



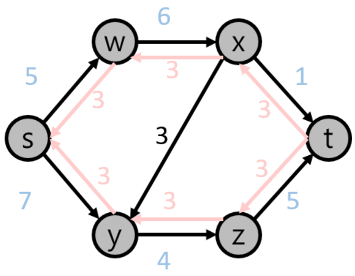
變成：



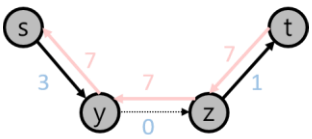
2. 第 2 條：



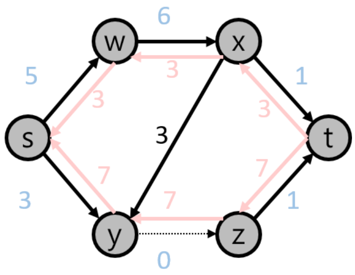
變成：



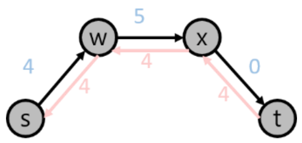
3. 第 3 條：



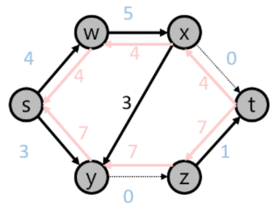
變成：



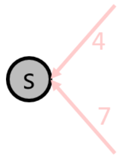
4. *第 4 條：*



變成：



結果：



MAX-Flow = 7+4 = 11

Note：可利用求完 Max-Flow 的 Residual Network 求本來的 Flow Network 的 Min-cut

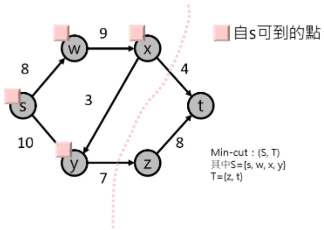
Min-cut：
一個 Graph 的 min-cut：(s, t)滿足：
1. S 聯集 T=V，且 S 交集 T=0
2. S->T 的邊的 weight 和，為所有 cut 中最小

在求完 Max-Flow 的 Residual Network 中，令

S：{自 s 可到的點}

T：{自 s 不可到的點}

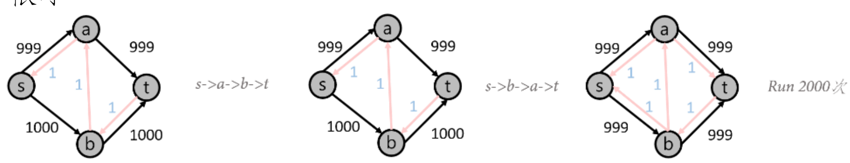
則(S, T)是原本 Flow Network 的 min-cut



Min-cut=(S, T)，其中 S={s, w, x, y}、T={z, t}

四、Edmand-Karp

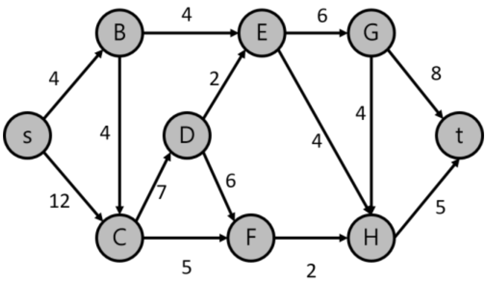
1. 因為 Ford-Fulkerson 在 worst case 下，可能會跑很久，即使 Network size 很小



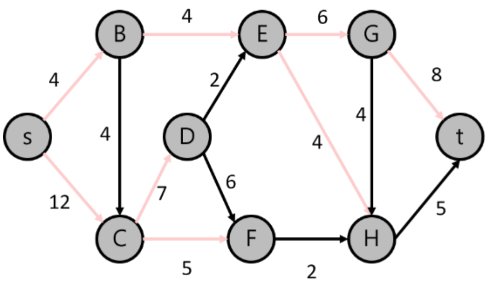
2. 和 Ford-Fulkerson 相同，唯有在選自 s->t 的路徑時，用 BFS 來選(自 s 做 BFS，當 visit 到 t 時，所產生 s->t 的路徑即為所求)

Time Complexity：$O(VE^2)$ *//只和 Network size 有關*

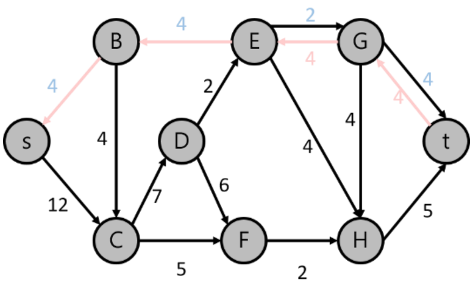
例(99 交大)：用 Edmand-Karp，求前 2 次的 Augmenting Path



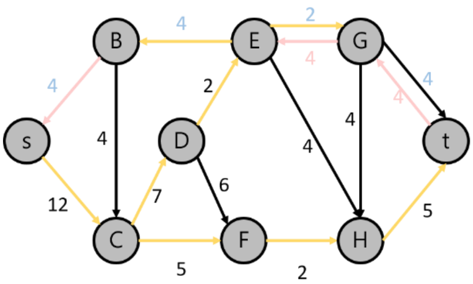
第 1 步：



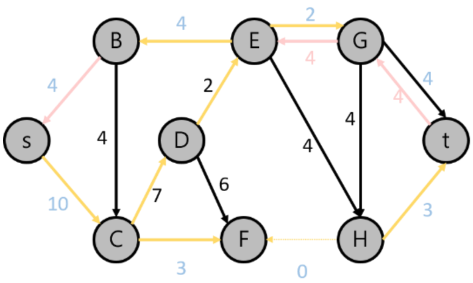
BFS visit的過程
1st Path：s B E G t



第 2 步：



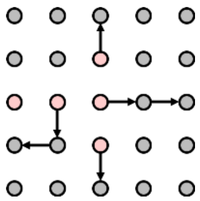
BFS visit的過程
2nd Path：s C F H t



BFS visit的過程
2nd Path：s C F H t

例(100 台大)(p4-80.20)：

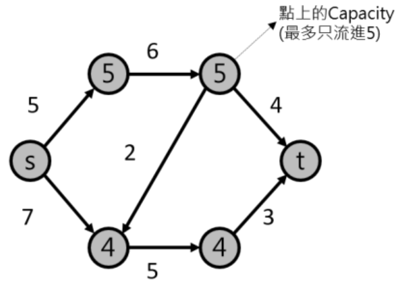
Escape Problem：給 n*n 的 grid，m 個起點，問 m 個人是否逃生成功



1. ◦找一條 Path 到
2. Path 彼此不可重複

Question：設計一個演算法解 Escape Problem

將 Escape Problem model 成一個”在點和邊均有 Capacity 的 Flow Network”，找 Max-Flow 的問題



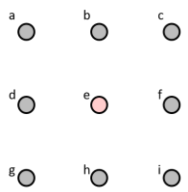
步驟如下：

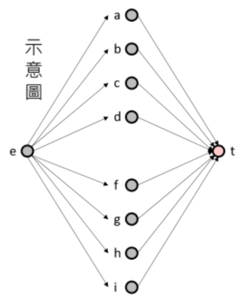
1. *建立一個 source，連接到 m 個起點*
2. *建立一個 sink t，每個在邊上的點(共有 4n-4 個)連到 t*
3. *對每個 grid 邊(u, v)，在 Flow Network 上建立(u, v)和(v, u)兩邊 //用有向圖模擬無向圖*
4. *設每邊、每點的 Capacity 均為 1*

若 MAX Flow = m => m 個人均可逃出

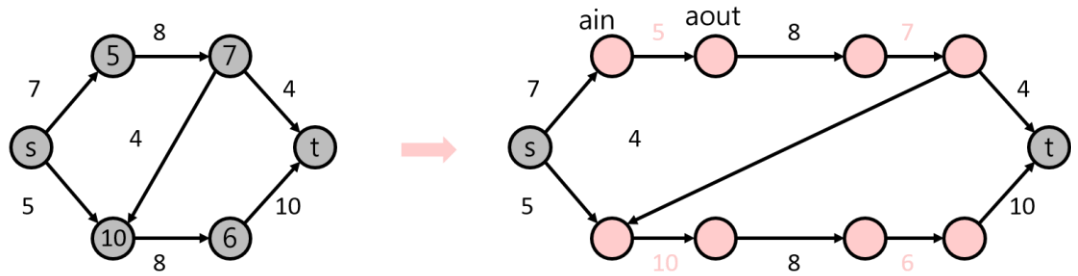
< m => 有人無法逃生

例：





Note：點、邊均有 Capacity 的 Flow Network 可以用傳統的 Flow Network 實現



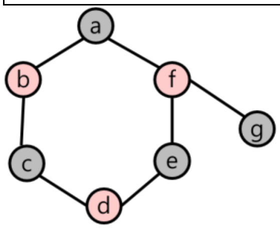
其他問題

一、問題的要求即使有小變化，可能使其難度改變很大

1. Shortest Path Problem：P
   Longest Path Problem：NPC
2. Min-cut：P
   Max-cut：
3. Euler Circuit：P
   Hamiltonian Cycle：NPC

二、同一個問題，若給的環境不相同，難度亦可能差很多

	Graph	Tree
找 Longest Path	NPC	Linear Time(p4-66.6)
找 Minimum Vertex Cover	NPC	Linear Time(p4-75.15)



Vertex-Cover：點集，會和 Graph 上所有邊相連

{b, d, f}為 Minimum Vertex Cover