圖論 I

Graph Theory I

2023 南 11 校聯合寒訓 - 演算法課程

2023.02.02

陳俊安 Colten

主辦單位：

SC ST

NHDK Ten Point Round

協辦單位：

iTSA 教育部 智慧創新跨域人才培育計畫

成功大學資訊工程學系暨研究所 Department of Computer Science and Information Engineering

贊助單位：

ytp 少年圖靈計畫 Young Turing Program

奧義智慧科技 Powered by CyCraft

TEAMT5

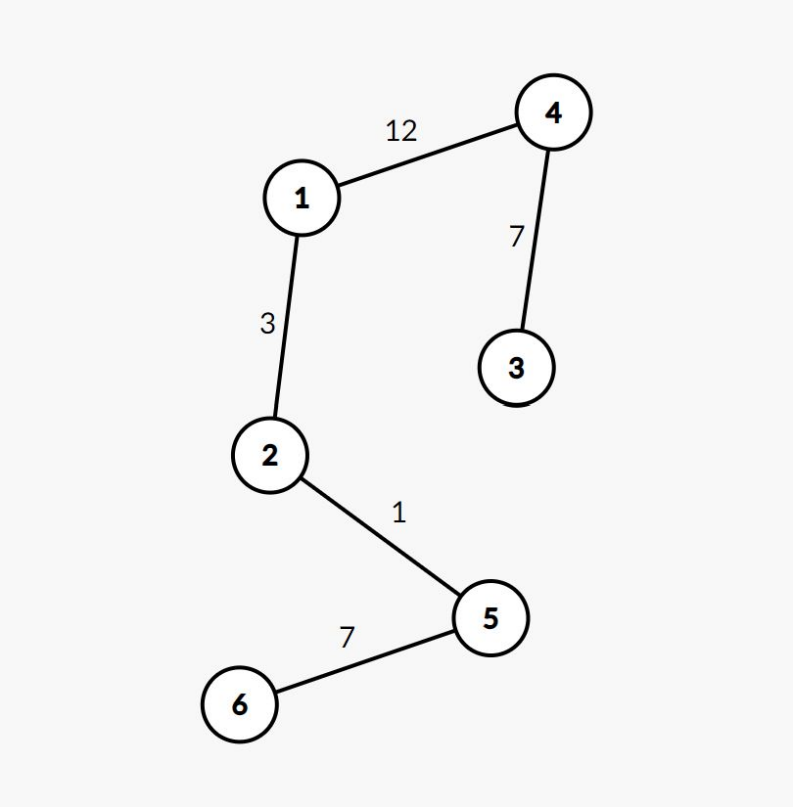
DEVCORE

Catalog

- Introduction to Graph and Graph Theory
- Graph and Creating Edges
  - Adjacency Matrix
  - Adjacency List
- DFS and BFS
- Topological Sort (拓撲排序)

Graph

- Vertex 節點
- Edge 邊
- Weight 權重
- Path 路徑



Graph Theory 圖論

- 離散數學的其中一個分支，跟圖有關的一些特性或性質
  - Dijkstra's Algorithm
  - Minumum Spanning Tree
  - Floyd-Warshall
  - and so on…

Creating Edges of Graph

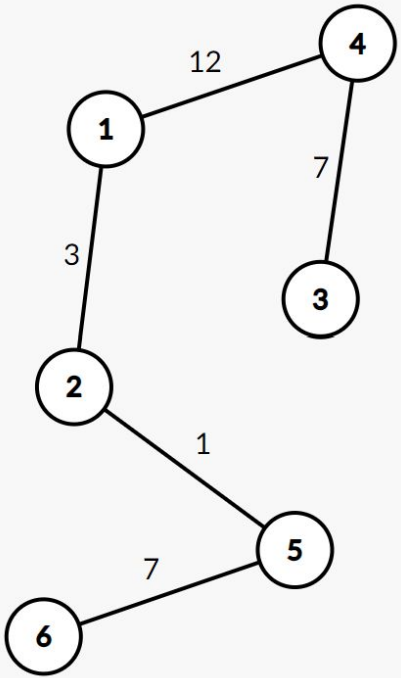
- 兩種方式
  - Adjacency Matrix (使用二維陣列實作)
  - Adjacency List (Linked-List)
    - 如果使用 C++ 的人，STL 有一些好用的工具可以取代 Linked-List (eg. vector)

Adjacency Matrix

- 鄰接矩陣
- 一種很直接的建邊方式
  - Step 1. 宣告一個二維陣列
  - Step 2. 定義 array[x][y] = 權重
    - 如果 array[x][y] != 0，表示 x，y 之間有一條邊，這一條邊的權重是 array[x][y]
    - 否則，x,y 之間沒有邊

Adjacency Matrix

- For example:
  - array[1][4] = 12
  - array[2][1] = 3
  - array[5][6] = 7
  - array[4][1] = 12
  - and so on...



Adjacency Matrix Implementation

- Now, we want to input N vertices and M edges
  - The first line input N (<= 500) and M (<= 1000).
  - Next, we will input M lines.
    - every line have three integers X,Y,W
    - that mean there have an edge between X and Y, and the weight is W.
    - the graph is undirected (無向圖).

Adjacency Matrix Implementation

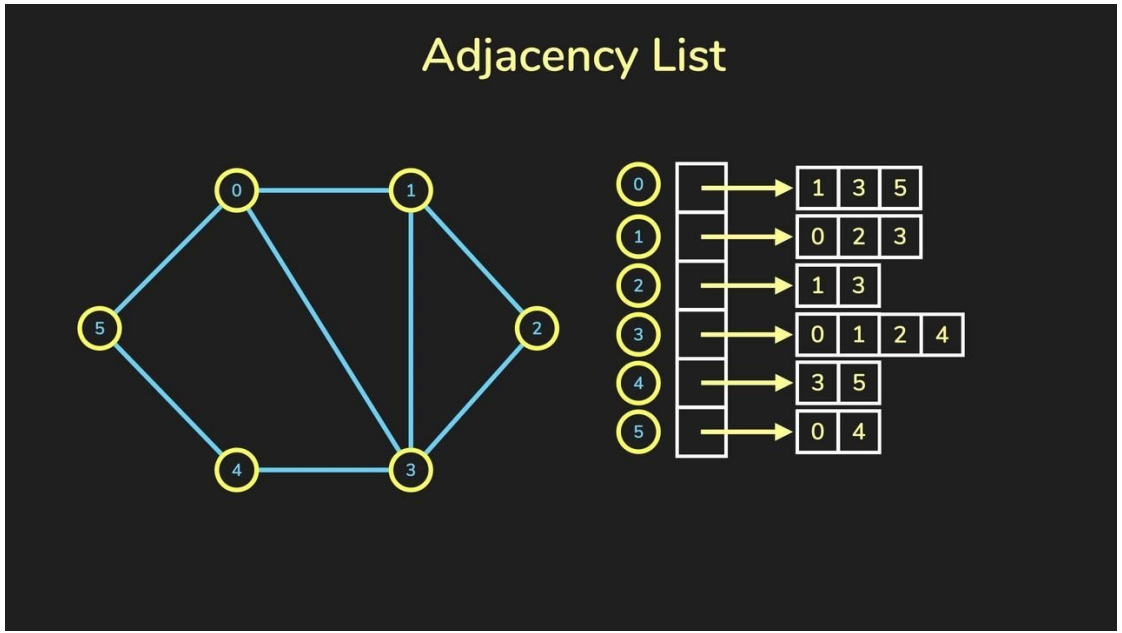
```cpp
#include <bits/stdc++.h>

#define int long long

using namespace std;

int a[501][501];

signed main(void)
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    int n,m;
    cin >> n >> m;

    for(int i=0;i<m;i++)
    {
        int X,Y,W;
        cin >> X >> Y >> W;

        a[X][Y] = W;
        a[Y][X] = W;
    }

}
```

Adjacency Matrix

- 我們需要 O(N^2) 的空間去宣告陣列
- 因此如果點的數量很多，這一個建邊的方法是不好的
- 如果點的數量很多，我們就必須使用另外一種方法來建邊
  - Adjacency List

Adjacency List

- When edge need a new space, and we just give it !



Adjacency List

- 在 C++ 中我們可以不使用 Linked-List 來實作
- 為了方便我們通常都會使用 STL-vector 來實作 Adjacency List

Adjacency List

- 首先，我們宣告一些 vector 出來 (vector<pair<int,int>>)
  - 點的數量有幾個我們就宣告幾個 vector
- Pair
  - v[x].first = 這一條邊連接的另外一個點
  - v[x].second = 這一條邊的權重

Adjacency List

- 如果 (x,y) 之間有一條邊，這一條邊的權重是 w
  - v[x].push_back({ y , w })
  - v[y].push_back({ x , w }) // if the graph is undirected

Adjacency List Implementation

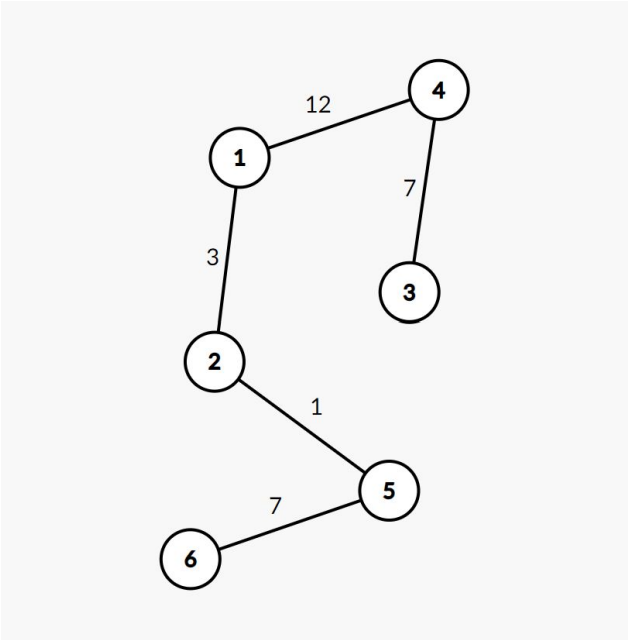
```
20    vector<int> a[20];
21    int n,m;
22    cin >> n >> m;
23    for(int i=0;i<m;i++)
24    {
25        int x,y,w;
26        cin >> x >> y >> w;
27        a[x].push_back({y,w});
28        a[y].push_back({x,w});
29    }
```

Adjacency List

- 我們需要 O(E) 空間去創建 Adjacency List

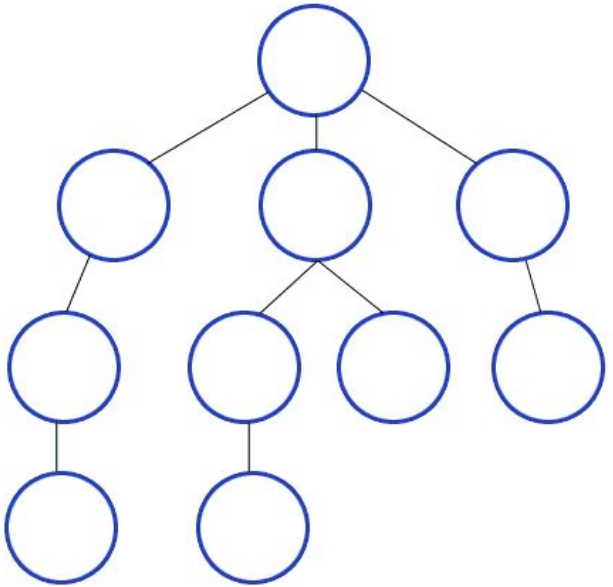
Practice !

- 寫一個 C++ 的程式碼建出這一個圖



Depth-First-Search

- 深度優先搜尋
- 有路就走，沒有路就回來找其他路



Depth-First-Search Implementation

- 通常我們使用 遞迴 來實作 DFS，除了遞迴以外 DFS 也可以透過 stack 的方式去實現，但遞迴是一個最方便的做法
- DFS 的 Function 我們通常會放兩個參數
    - 當前所在的點
    - 上一個我們走到的點是誰 (避免一直來回走)

Depth-First-Search Implementation

- 如果圖是 樹 Tree (沒有環)

```
5 vector<int> edge[100005];
6
7 void dfs(int idx,int last)
8 {
9     for(int i=0;i<edge[idx].size();i++)
10    {
11        if( edge[idx][i] == last ) continue;
12
13        dfs( edge[idx][i],idx );
14    }
15 }
```

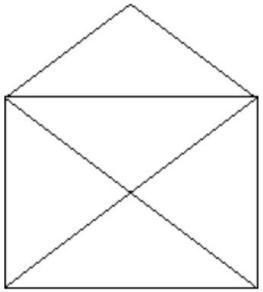
Depth-First-Search Implementation

- 如果圖可能有環

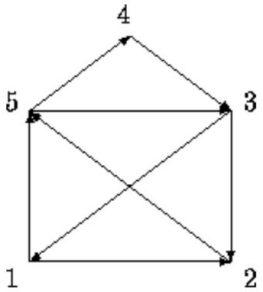
```
 5 vector<int> edge[100005];
 6
 7 bool visited[100005];
 8
 9 void dfs(int idx)
10 {
11     for(int i=0;i<edge[idx].size();i++)
12     {
13         if( visited[ edge[idx][i] ] == true ) continue; // 以前走過那個點
14
15         visited[ edge[idx][i] ] = true; // 標記成有走過
16         dfs( edge[idx][i],idx );
17         visited[ edge[idx][i] ] = false;
18         // 接下來要枚舉下一個走法了，但下一個走法還沒有走過這一個點
19         // 所以我們要把它還原成沒有走過
20     }
21 }
```

The House Of Santa Claus

不知道你們小時候有沒有玩過一筆畫，畫聖誕老人的房子的遊戲。畫出來的圖就像下面這樣：



經過這麼多年了，現在我們想要寫一個程式來知道如果我們從左下角開始畫起，共有多少種一筆畫聖誕老人房子的方法。下面的圖中的5個點用1到5來編號（從編號1開始畫起），箭頭代表一筆畫時筆的路徑。這個例子中的路徑為：153125432



先建邊，建完邊才能 DFS

- 利用我們之前學到的鄰接陣列來建邊

```
48 void edge()
49 {
50     pre[1][5] = true;
51     pre[1][3] = true;
52     pre[1][2] = true;
53
54     pre[2][1] = true;
55     pre[2][3] = true;
56     pre[2][5] = true;
57
58     pre[3][4] = true;
59     pre[3][5] = true;
60     pre[3][1] = true;
61     pre[3][2] = true;
62
63     pre[4][3] = true;
64     pre[4][5] = true;
65
66     pre[5][1] = true;
67     pre[5][2] = true;
68     pre[5][3] = true;
69     pre[5][4] = true;
70
71     return;
72 }
```

DFS 過程，有路就走，沒有路就退回來

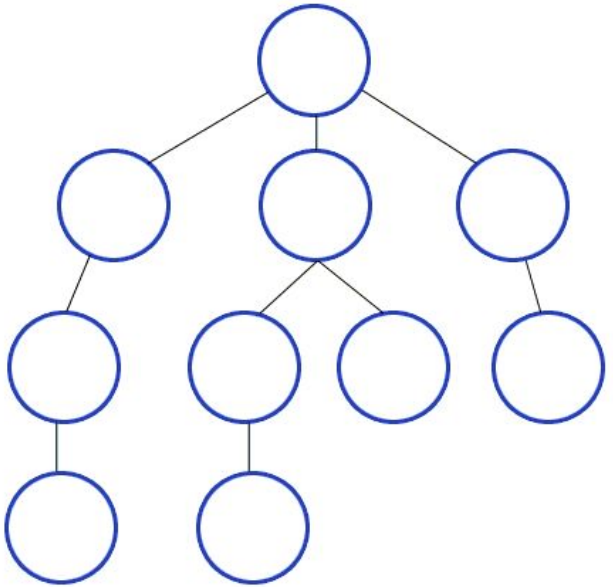
```
 9 int ans[9]; // 紀錄每一步到達的點是哪一個
10
11 bool visited[6][6]; // 紀錄那一條邊我們有沒有畫過了
12
13 void dfs(int point,int index) // 現在正在 point 這一個點，這次是第 index 次畫畫
```

DFS 過程，有路就走，沒有路就退回來

```
27      for(int i=1;i≤5;i++) // 枚舉接下來要走到哪一個點
28      {
29          if( pre[point][i] == true ) // (point,i) 之間有邊
30          {
31              if( visited[point][i] == 1 || visited[i][point] == 1 )
32              {
33                  continue; // 走過了，不要再走了
34              }
35
36              ans[index] = i; // 藉由這一次畫畫，我們的筆到達的點是 i
37
38              visited[point][i] = 1;
39              visited[i][point] = 1; // 1 走過 , 0 沒有走過
40
41              dfs(i,index+1); // 枚舉下一筆畫
42
43              // 枚舉完後，別的 Case 還沒有用過 (point,i) 這一條邊，所以要還原
44
45              visited[point][i] = 0; // 還原
46              visited[i][point] = 0; // 還原
47          }
48      }
49  }
```

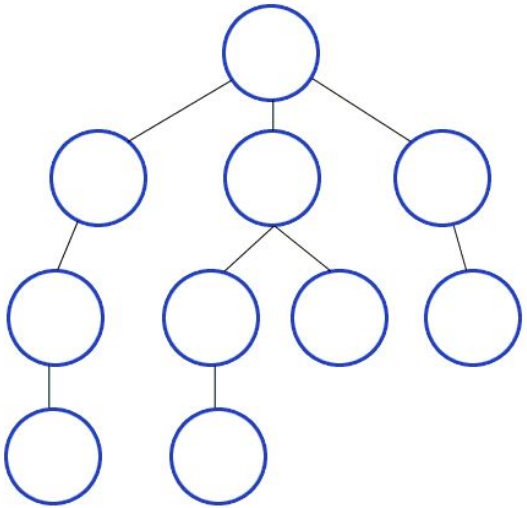
Breadth-First Search

- 廣度優先搜尋
- 從一個起點開始，向外擴散



Breadth-First Search

- 開一個 queue 維護
- 先把起點拉近 queue 裡面
- 1. 接下來從 queue 拿出 front()
- 2. 看看 queue 的 front() 能走到誰
- 3. 把那些點全部拉近 queue 裡面
- 4. 重複以上步驟，直到 queue 沒東西



Breadth-First Search

```
80      int n,m;
81      cin >> n >> m;
82
83      for(int i=0;i<m;i++)
84      {
85          int x,y;
86          cin >> x >> y;
87
88          e[x].push_back(y);
89          e[y].push_back(x); // 如果圖是無向圖
90      }
```

Breadth-First Search

```
92      queue<int> q;
93      q.push(1); // 假設 1 是起點
94
95      while( q.size() ≠ 0 )
96      {
97          int v = q.front();
98          q.pop(); // 記得刪掉
99
100         for( auto i : e[v] ) // 看看 v 可以走到哪些點
101         {
102             if( visited[i] == 0 ) // 如果 i 沒有走過
103             {
104                 visited[i] = 1;
105                 q.push(i);
106             }
107         }
108     }
```

BFS 練習題 - 倒水時間

範例輸入 #1

```
1
4 6
0 0 0 1 0 0
0 0 1 1 1 0
1 1 1 0 1 0
0 1 0 0 0 0
1
5 7
1 0 0 0 0 0 0
1 1 1 1 1 1 0
0 1 0 0 1 0 1
1 1 1 0 1 1 1
1 0 1 0 1 1 1
2
6 6
0 1 0 0 0 0
1 1 0 0 1 1
0 1 1 1 1 0
1 1 1 1 1 1
0 0 0 0 1 0
0 0 0 1 1 0
```

範例輸出 #1

```
Case 1:
0 0 0 1 0 0
0 0 3 2 3 0
6 5 4 0 4 0
0 6 0 0 0 0
Case 2:
1 0 0 0 0 0 0
2 3 4 5 6 7 0
0 4 0 0 7 0 11
6 5 6 0 8 9 10
7 0 7 0 9 10 11
Case 3:
0 1 0 0 0 0
3 2 0 0 0 0
0 3 4 5 6 0
5 4 5 6 7 8
0 0 0 0 8 0
0 0 0 10 9 0
```

BFS 練習題 - 倒水時間

- 由於水只會 上、下、左、右 這樣子流
- 因此我們可以預先開好兩個陣列儲存上下左右移動的變量
- 這樣到時候就可以直接依靠 for 迴圈而不用寫一堆 if/else

```
7 int add_index1[4] = {0,0,-1,1};
8
9 int add_index2[4] = {1,-1,0,0};
```

BFS 練習題 - 倒水時間

- 接下來我們先找出整張圖的起點的位置
- mp[i][k] 表示 (i,k) 的狀態，ans[i][k] 表示 (i,k) 的答案

```
23        int start;
24
25        for(int i=0;i<n;i++)
26        {
27            for(int k=0;k<m;k++)
28            {
29                cin >> mp[i][k];
30
31                if( i == 0 && mp[i][k] == 1 ) start = k;
32
33                ans[i][k] = 0;
34            }
35        }
```

BFS 練習題 - 倒水時間

- 將起點拉近 queue 裡面，並且更新答案
- pair 的 first 與 second 儲存點的位置

```
37          queue <pair<int,int>> q;
38
39          q.push(make_pair(0,start));
40
41          ans[0][start] = 1;
```

BFS 練習題 - 倒水時間

- 開始 BFS，一開始先將 queue 的 front 拿出來

```
while( q.size() ≠ 0 )
{
    int u1 = q.front().first;

    int u2 = q.front().second;

    q.pop();
```

BFS 練習題 - 倒水時間

- 接下來枚舉接下來要往 上下左右 這四個方向流

```
for(int i=0;i<4;i++)
{
    int u3 = u1 + add_index1[i];

    int u4 = u2 + add_index2[i];
```

BFS 練習題 - 倒水時間

- 特別注意如果輸入的最後一個數字是 2，那麼水是不能往上流的，這個時候就必須特判掉

```
57                  if( s == 2 && i == 2 ) continue;
```

BFS 練習題 - 倒水時間

- 除此之外，也要記得水不能出界，且流到的地方要是水管
  - mp[i][k] = 1，表示 (i,k) 是水管

```
59              if( u3 < 0 || u4 < 0 || u3 ≥ n || u4 ≥ m ) continue;
60
61              if( mp[u3][u4] ≠ 1 ) continue;
```

BFS 練習題 - 倒水時間

- 除此之外，也要記得水不能出界，且流到的地方要是水管
  - mp[i][k] = 1，表示 (i,k) 是水管

```
59              if( u3 < 0 || u4 < 0 || u3 ≥ n || u4 ≥ m ) continue;
60
61              if( mp[u3][u4] ≠ 1 ) continue;
```

BFS 練習題 - 倒水時間

- 最後看看這一個點有沒有被走過了
  - 沒有被走過就拉近 queue
  - 被走過了就直接忽略

```
61                  if( mp[u3][u4] ≠ 1 ) continue;
62
63                  if( ans[u3][u4] ≠ 0 ) continue;
64
65                  q.push({u3,u4});
66
67                  ans[u3][u4] = ans[u1][u2] + 1;
```

BFS x APCS - 第四題 蓋步道

內容

有一個大小為 $n \times n$ 的方形區域，h_{ij} 代表位於座標 (i, j) 的格子該處的海拔高度。

工程團隊想要從該區域的左上角 $(1, 1)$ 鋪設一條步道到右下角 (n, n)，鋪設的步道可以視為在該區域內上下左右四個方向從左上角走到右下角的一條路徑。

考量到行人在步道上行走的安全，必須要注意步道每一步之間的高低落差，並希望可以建立出一個最大高度差最小的步道鋪設方案。

請輸出該鋪設方案最大高度差的最小值和在該最大高度差的前提下步道的最短路徑長度。

BFS x APCS - 第四題 蓋步道

- 先觀察一些性質，我們可以發現答案具有單調性
  - 如果高低差 x 是合法的，那麼 > x 的高度差也一定合法

BFS x APCS - 第四題 蓋步道

- 既然答案具有單調性，且枚舉的時間複雜度不好
- 那我們就可以利用對答案二分搜來優化枚舉
  - 每一次去看看二分搜的 mid 是不是合法的
  - 是不是合法的判斷方式可以簡單地透過 BFS 完成
  - 如果是合法的就縮小右界
  - 反之縮小左界

BFS x APCS - 第四題 蓋步道

- 既然答案具有單調性，且枚舉的時間複雜度不好
- 那我們就可以利用對答案二分搜來優化枚舉
  - 每一次去看看二分搜的 mid 是不是合法的
  - 是不是合法的判斷方式可以簡單地透過 BFS 完成
  - 如果是合法的就縮小右界
  - 反之縮小左界

BFS x APCS - 第四題 蓋步道

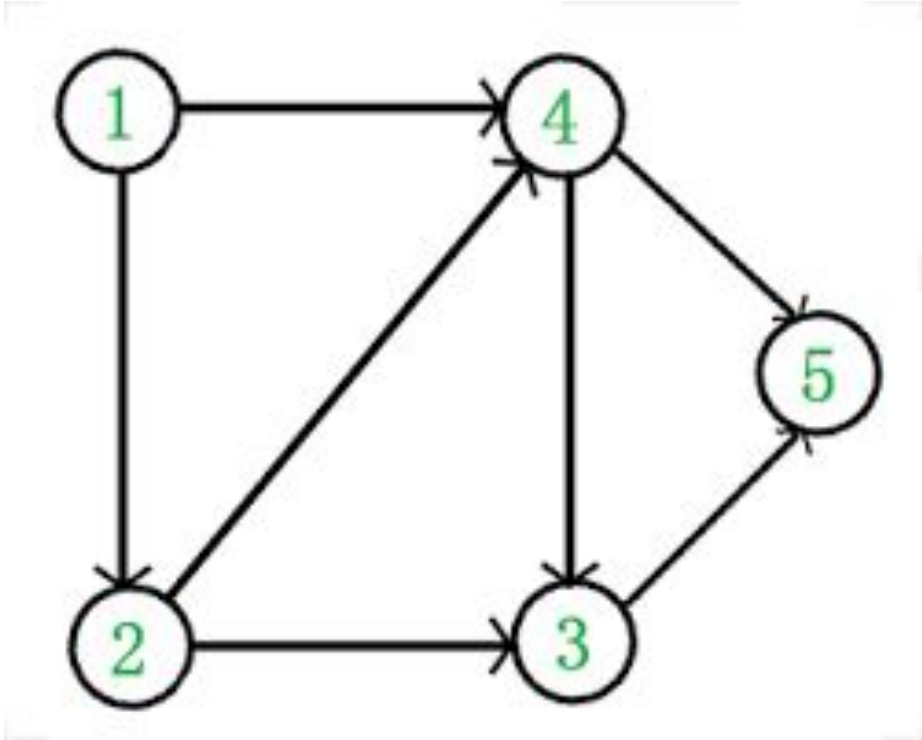
- 這邊我將我二分搜的寫法提供給大家
- check 函式做的事情是 BFS 看看高度差 mid 可不可以
- 這一個部分給大家當作練習

```
60      int l = 0,r = 1e6;
61      while( l ≤ r )
62      {
63          int mid = ( l + r ) / 2;
64
65          if( check(mid) == true ) r = mid - 1;
66          else l = mid + 1;
67      }
68
69      // answer = l
```

拓撲排序

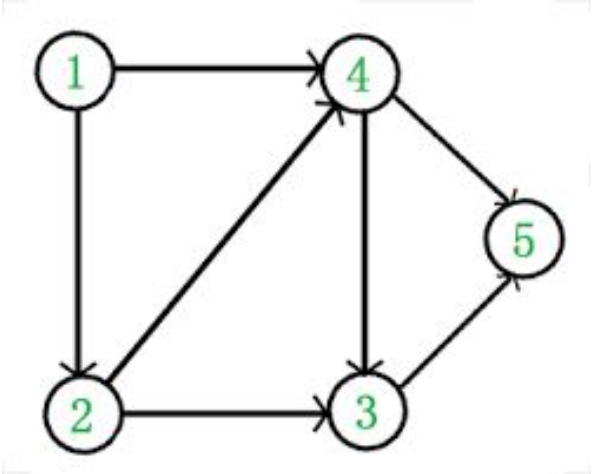
- 一種排序有向圖的排序方法
- 如果有一條邊是 A -> B
- 那就表示 B 一定要等 A 被走過才能被走
- 有一點像是 A 是 B 的前置任務的概念

拓撲排序



拓撲排序的實作

- 額，在這之前，我們再多認識一些圖論術語好了
  - 有向圖、無向圖
  - 度數
    - 出度
    - 入度



拓撲排序的實作

- 我們在 BFS 之前，多額外開一個陣列
- 紀錄每一個點的 入度
- 每次 BFS 的時候，A 走到某一個點 B，就把 B 在那一個陣列的數字 -1
    - 因為當前走到 A，而 A 是 B 的前置
    - 那就表示 B 的前置完成了一個

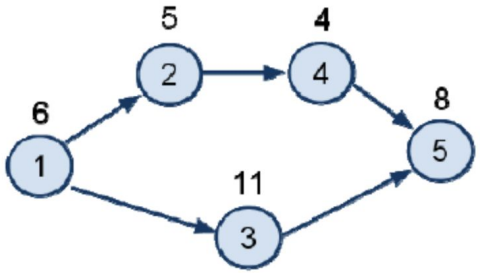
拓撲排序的實作

```
51    for(int i=0;i<m;i++)
52    {
53        int x,y;
54        cin >> x >> y;
55        e[x].push_back(y);
56        in[y]++;
57    }
58
59    queue<int> q;
60    q.push(1);
61
62    while( q.size() != 0 )
63    {
64        int v = q.front();
65        q.pop();
66
67        for( auto i : e[v] )
68        {
69            in[i]--;
70            if( in[i] == 0 ) q.push(i); // i 的前置都走過了，可以走 i 這一個點了
71        }
72    }
```

拓撲排序 - 2010 資奧初選第二題 - 專案時程

- 拓樸排序裸題

通常在開發一個專案時，整個專案會被分割為許多個項目，並同時分配給多組程式設計師去開發。但這些項目是有順序關係的，只有當順序在前方的項目完成後，才能夠開始開發順序在後方的項目。我們利用一個有向圖，來表示這些項目的開發順序。圖上的每一個節點代表一個項目，節點內的數字為節點編號，上方的數字代表開發這個項目所需的天數；圖上的邊則表示開發的順序，以右圖為例，只有在節點 2 完成後，才能夠開始節點 4 的開發。右圖為範例測試資料中的第二組專案有向圖。



有一間軟體公司目前正有許多的專案準備開始開發，但是這間公司的前一任專案管理人(PM)因不堪壓力離職了，在臨走之前他留下了當初初略畫出的開發流程圖。現在你是這間公司新進的專案管理人，而你的老闆正迫切的想知道這些專案能不能在他所限制的時間內完工，請你寫一個程式依照這些專案的開發流程圖回答老闆的問題。

註: 這間公司有非常充足的程式設計師，因此並不需要擔心人手不夠的問題。

拓撲排序 - 2010 資奧初選第二題 - 專案時程

- 就只要正常的實作一次拓樸排序就可以了
- 大家回去可以拿這一題來練習實作一次看看！