

基礎圖論

sam571128

November 10, 2021

公告

這禮拜天 (11/7) 有 APCS 的考試，因此我今天會在社團的 group 裡面放個我幫大家出的模擬考試。詳細的東西我會在弄好的時候發訊息到社團的 Discord 群

一些建議

- ① APCS 的前兩題通常都不需要會任何演算法，p1 只要照著題目做就好，p2 在這兩年的考試中，幾乎有 90% 以上都是二維陣列的使用
- ② 前兩題一定要拿到! 這樣就有 3 級分了
- ③ 對於剩下的兩題，通常會有一題 *DP* 題，如果真的想不到作法，請看一下題目的子題，想辦法從剩下的兩題拿到 50 分 \Rightarrow 4 級分
- ④ 如果想要拿到 5 級分，最簡單的方式是全對，但是如果真的做不到，建議拿 3 題之後，再從剩下的一題拿到 50 分

另外一些重要的技巧

- `#define int long long`，可以避免整數溢位
- 暴力對拍，用一些時間複雜度更差的正確解去 debug 時間複雜度好的解
- 少用 Python，時間容易被卡

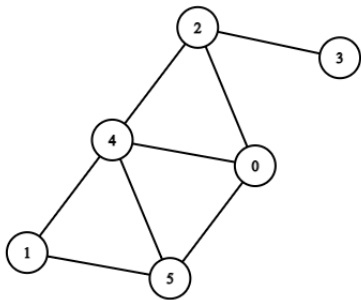
什麼是圖論？

圖論，在資訊科學中是一個滿重要的一部分，但是在高中數學根本不會遇到圖論的問題。不過圖論的問題其實十分有趣，有一些東西大家可能稍微聽過。



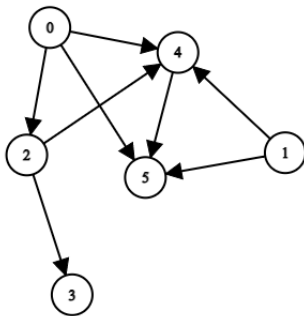
一張圖，是由許多的節點，與幾條邊所組成的一種圖形。

數學上寫成 $G = \{V, E\}$



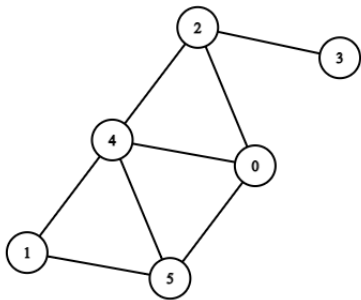


而圖有很多種不同的分類，依照方向，分為有向圖與無向圖



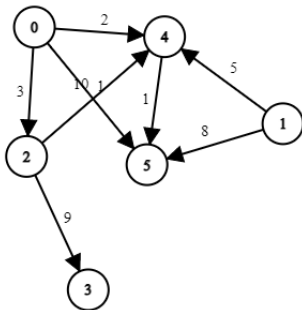


而圖有很多種不同的分類，依照方向，分為有向圖與無向圖





有時候，邊上會有權重。





這裡有幾個專有名詞要讓大家知道一下。

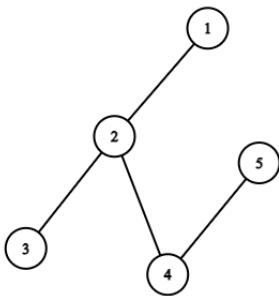
- 邊 (Edge) , $e = (u, v)$
- 節點 (Vertex)
- 邊權 (Weight)
- 度數 (Degree): 對於一個點，他連出去的邊數有多少

這裡有幾個專有名詞要讓大家知道一下。

- 路徑 (Path): 在一張圖上，由一連串的点從 u 走到 v
- 連通 (Connected): 在圖上，能從 u 走到 v ，則 u, v 連通
- 環 (Cycle): 從 u 經由一些路徑走回 u ，稱為環
- 自環 (Cycle): 當圖上有一條 u 到 u 的邊
- 重邊 (Multiple Edge): 在圖上有兩條 $e_i = e_j$ 的邊，稱為重邊

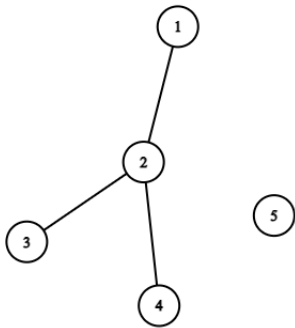
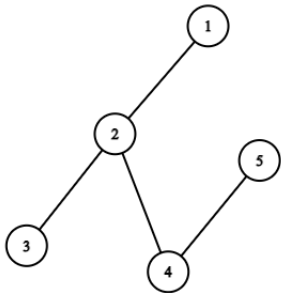


簡單圖 (Simple Graph): 在一張圖上沒有自環與重邊 (正常題目都是出簡單圖)





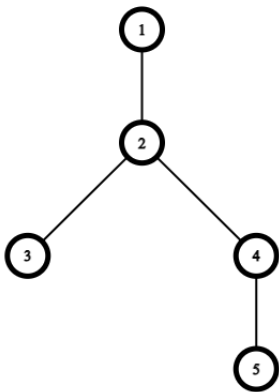
連通圖 (Connected Graph): 在一張圖上，任兩點都能走到對方





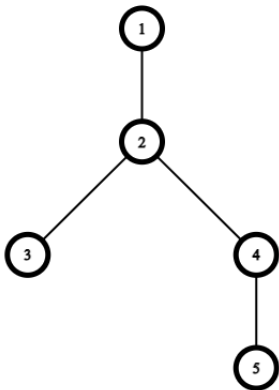
樹: 一張連通且無環的圖 (特點是有 n 個點的話有 $n - 1$ 條邊)

最上面是根，最下面是葉子



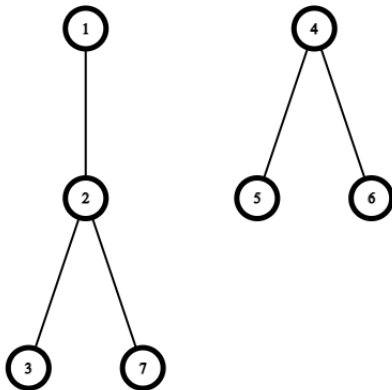


而樹又分為有根樹與無根樹，在這張圖 1 是根

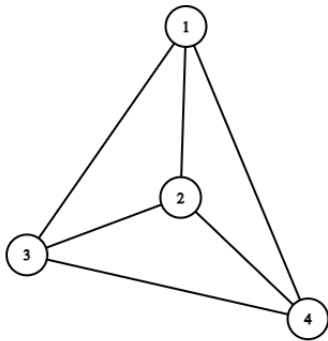




森林：一張由很多棵樹所組成的圖

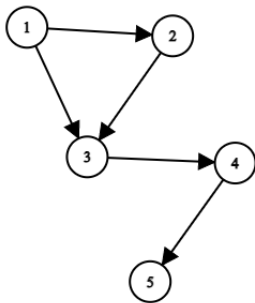


完全圖：圖上每個點兩兩都有一條邊





有向無環圖 (Directed Acyclic Graph, 簡稱 DAG), 沒有環的有向圖

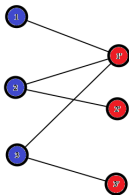




有向無環圖 (Directed Acyclic Graph, 簡稱 DAG), 沒有環的有向圖
可以化成一條直線, 後面的點無法走到前面的點 (可以用拓撲排序找)



二分圖 (Bipartite Graph): 可以塗色成兩種顏色，且相同顏色的點不相鄰



DAG

圖的儲存

鄰接矩陣 (Adjacency Matrix)

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

當 $a_{ij} = 1$ 時，表示 i 與 j 兩點之間有一條邊

反之，當 $a_{ij} = 0$ 時，表示 i 與 j 兩點之間沒有邊

圖的儲存

鄰接矩陣 (Adjacency Matrix)

```
const int N = 1e3+5;
int adj[N][N]

for(int i = 0; i < m; i++){
    int u,v;
    cin >> u >> v;
    adj[u][v] = 1; //有向圖只有這行
    adj[v][u] = 1; //若是無向圖
}
```

圖的儲存

鄰接串列 (Adjacency List)

1	2, 3
2	1
3	1

通常用 vector 或 linked list 儲存 (或一種大陸常用的鍊式前向星的存法)

圖的儲存

鄰接串列 (Adjacency List)

```
const int N = 1e6+5;
vector<int> adj[N];

for(int i = 0; i < m; i++){
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v); //有向圖只有這條
    adj[v].push_back(u); //若是無向圖
}
```


圖的儲存

鄰接矩陣

空間較大

可以快速看出 u, v 是否相連

在某些狀況比較快

鄰接串列

空間較小

可以快速得到與 u 相鄰的點

一般情況都比較快

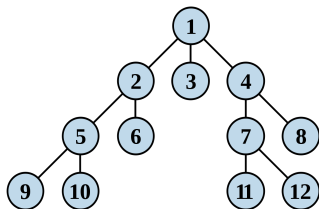
圖的遍歷

圖的遍歷有兩種方式:

- DFS
- BFS

圖的遍歷

BFS (廣度優先搜尋) 就是照著離起點的順序去搜尋。



圖的遍歷

DFS:

```
void dfs(int u){
    visited[u] = true;
    for(auto v : adj[u]){
        if(visited[v]) continue;
        dfs(v);
    }
}
```

圖的遍歷

BFS:

```
queue<int> q;  
q.push(s);  
  
while(!q.empty()){  
    int u = q.front(); q.pop();  
    for(auto v : adj[u]){  
        if(!visited[v]) q.push(v);  
    }  
}
```

圖的遍歷

BFS:

```
queue<int> q;  
q.push(s);  
  
while(!q.empty()){  
    int u = q.front(); q.pop();  
    for(auto v : adj[u]){  
        if(!visited[v]) q.push(v);  
    }  
}
```

圖的遍歷

網格圖:

```
//網格圖可以把 adj 的遍歷改成以下
for(int dx : {-1,0,1}){
    for(int dy : {-1,0,1}){
        if(abs(dx)==abs(dy)) continue;
        //Do something
    }
}
```

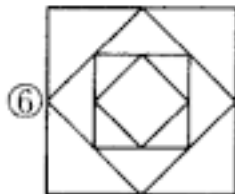
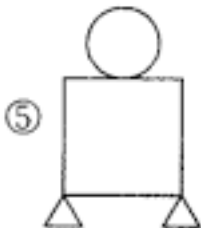
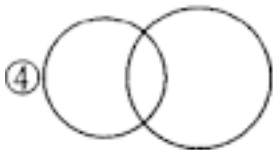

圖的遍歷

練習題:

- CSES Graph Algorithms 前幾題
- 可以在 cf, atcoder 等等找圖論題目寫

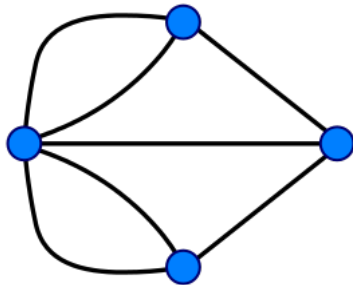
一筆畫問題

哪些可以在一筆畫 (不使用重複的邊) 走完一張圖



一筆畫問題

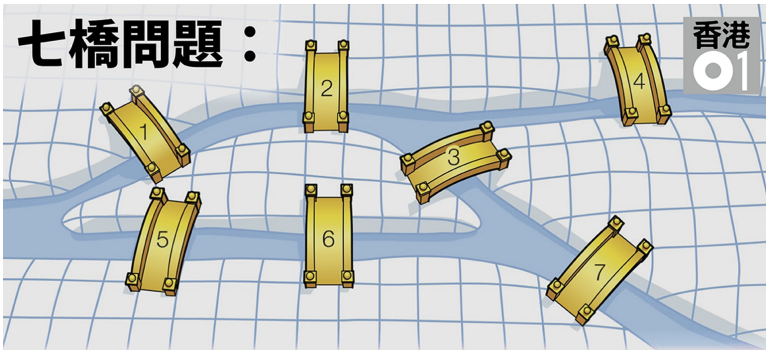
我們會發現，這張圖上有 4 個奇數度數的節點



一筆畫問題

事實上，我們在討論的這個一筆畫問題，在 18 世紀的歐洲，有一個很有名的問題，被稱為「七橋問題」

七橋問題：



香港 01

【小島與河兩岸由七條橋連接，如何每條橋都只走一次的前提下，把所有地方走遍？】

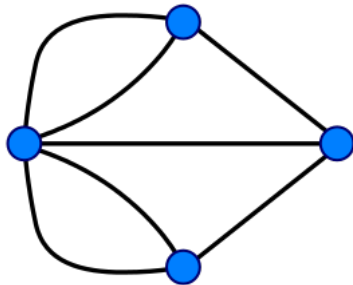
你能解答嗎？

一筆畫問題

而這個問題，被知名的數學家**歐拉**所解決，並統整出一種可以解決一筆畫問題的一種方式。

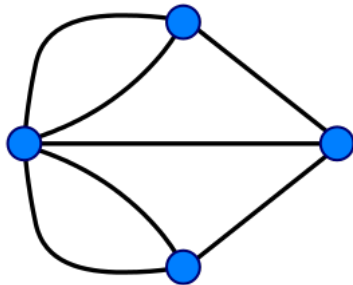
一筆畫問題

我們將剛剛的圖形畫為我們圖論上討論的圖



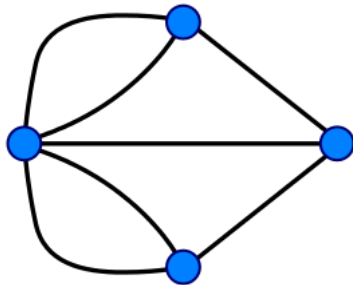
一筆畫問題

我們會發現，這張圖上有 4 個奇數度數的節點



一筆畫問題

因此這張圖，最少需要 2 筆畫才能畫完!



一筆畫問題

而這裡，我們要介紹兩個名詞：

- 歐拉路徑：從起點恰好經過一張圖的所有邊並結束在與起點不同的點
- 歐拉迴路：從起點恰好經過一張圖的所有邊並結束在起點

一筆畫問題

由歐拉所提出的「一筆畫定理」，如果一張連通圖有**歐拉路徑**，則這張圖的奇數節點必須為 0, 2 個，而如果一張圖有**歐拉迴路**，則這張圖必須只能有偶數節點

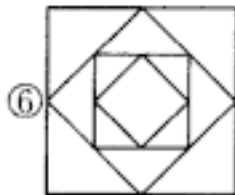
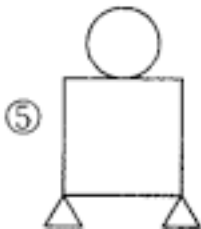
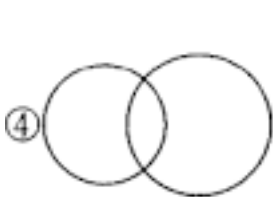
一筆畫問題

而一張連通圖最少需要的筆畫為 (V_o 為奇數度數的節點的數量)

$$\max(1, V_o/2)$$

一筆畫問題

所以上面這幾張圖，哪些可以在一筆畫畫完，又每張最少要幾筆畫呢？



一筆畫問題

接著，我們來談談要怎麼構造一條歐拉路徑或歐拉迴路

一筆畫問題

- 一條歐拉路徑，他的起點與終點的度數一定是奇數。
- 一條歐拉迴路，每個節點的度數一定是偶數

一筆畫問題

- 在歐拉路徑上，除了起點與終點外，每個點的進入的次數必須要等於離開的次數，而起點和終點會各自多一個出度與入度。
- 在歐拉迴路上，每個點的進入的次數必須要等於離開的次數，因此節點的度數皆為偶數

一筆畫問題

如果要找一組歐拉路徑，我們的起點選在其中一個奇數節點，並每次去走沒走過的邊即可找到

一筆畫問題

如果要找一組歐拉迴路，我們的起點任意選一個節點，每次走沒走過的邊即可

一筆畫問題

如果要找一組歐拉迴路，我們的起點任意選一個節點，每次走沒走過的邊即可

一筆畫問題

使用鄰接矩陣找歐拉路徑/迴路

```
int adj[N][N], vis[N][N];

void dfs(int u){
    cout << u << "\n";
    for(int v = 1; v <= n; v++){
        if(adj[u][v] && !vis[u][v])
            dfs(v);
    }
}
```

一筆畫問題

使用鄰接串列找歐拉路徑/迴路

```
struct edge{
    int u, v;
    bool vis;
};

vector<edge> edges;
vector<int> adj[N];

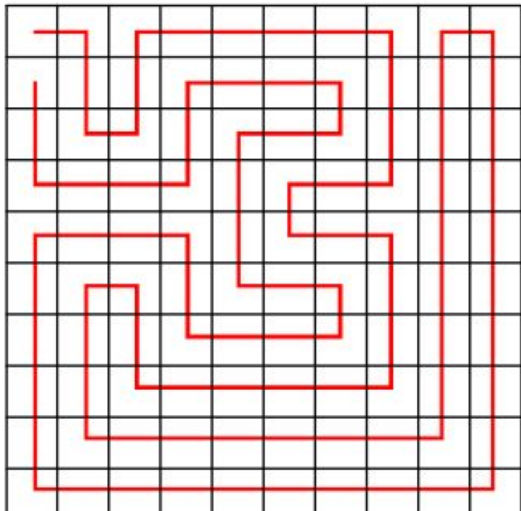
void dfs(int u){
    cout << u << "\n";
    for(auto eid : adj[u]){
        edge &e = edges[eid];
        if(e.vis) continue;
        e.vis = 1;
```

哈密頓路徑

- **哈密頓路徑**的定義是「經過不重複的點走完一張圖的路徑」
- **哈密頓迴路**的定義是「經過不重複的點走完一張圖的迴路」

哈密頓路徑

而哈密頓路徑則不如歐拉路徑那麼簡單，他是一個 **NP** 問題



哈密頓路徑

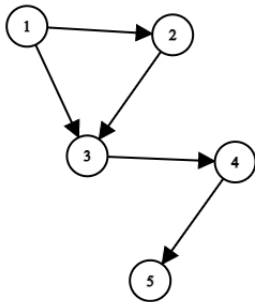
也就是說我們沒有一個可以快速找到哈密頓路徑的方法，不過，我們之前在講 DP 的時候提過一個問題「旅行推銷員問題」。

哈密頓路徑

而當時，我們是使用位元 DP 的做法去找，所以哈密頓路徑與迴路的問題可以在 $O(n^2 2^n)$ 的時間使用位元 DP 找到答案。

有向無環圖 (DAG)

接下來，讓我們回到有向無環圖 (DAG)



有向無環圖 (DAG)

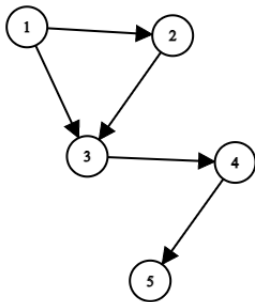
如果給你一張有向圖，你要怎麼確定他是不是 DAG?

拓樸排序

因此，我們要來談到所謂的**拓樸排序**。

拓樸排序

這張圖的拓樸序為 1, 2, 3, 4, 5，而順序比較後面的節點並不會連到前面



拓樸排序

要判斷一張圖是不是 DAG，很簡單，用節點的度數判斷!

拓樸排序

拓樸排序 (BFS):

- 1 開一個 queue，將所有入度為 0 的節點推進 queue
- 2 依序去做 BFS
- 3 移除目前走到的點
- 4 將所有目前入度為 0 的節點推進 queue

拓樸排序

拓樸排序 (BFS):

```
vector<int> topo;
queue<int> q;
q.push(0);

while(!q.empty()){
    int u = q.front(); q.pop();
    topo.push_back(u);
    for(auto v : adj[u]){
        deg[v]--;
        if(deg[v]==0) q.push(v);
    }
}
```

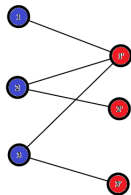
拓樸排序

拓樸排序 (BFS):

如果最後在 vector 裡面的點數少於 n ，則這張圖有環，否則為 DAG，順序已經被存在 vector 中了

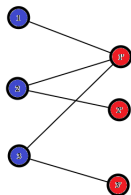
二分圖著色

如果要判斷一張圖是不是二分圖呢？



二分圖著色

很簡單，隨便從一個起點開始，去 DFS 所有點並塗色，如果完全沒有矛盾產生，則這張圖是一張二分圖



二分圖著色

```
int color[N], visited[N];

void dfs(int u, int c){
    color[u] = c;
    visited[u] = 1;
    for(auto v : adj[u]){
        if(color[v] == c){
            //The graph is not a bipartite
        }
        if(visited[v]) continue;
        dfs(v, c^1);
    }
}
```

暫時總結

今天講了歐拉路徑、哈密頓迴路、DAG、二分圖。我們明天會來講一個叫做並查集的資料結構，與圖論有關。