Advanced
Competitive Programming

國立成功大學ACM-ICPC程式競賽培訓隊
nckuacm@imslab.org

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Week 5
Sorting & Graph

排序、離散化、一些基礎圖論

Outline

- 排序
  - Merge Sort
  - Counting Sort
- 離散化
- 基礎圖論
  - Dfs
  - Bfs
  - Disjoint Set & Union

Sort

排序？那是什麼？可以吃嗎？

- 簡單來說，就是排序
- 像是把 1,4,5,3,2 排成 1,2,3,4,5

Sorting Algorithm

- Bubble Sort
- Merge Sort
- Counting Sort
- STL Sort

Bubble Sort

從最基礎的開始

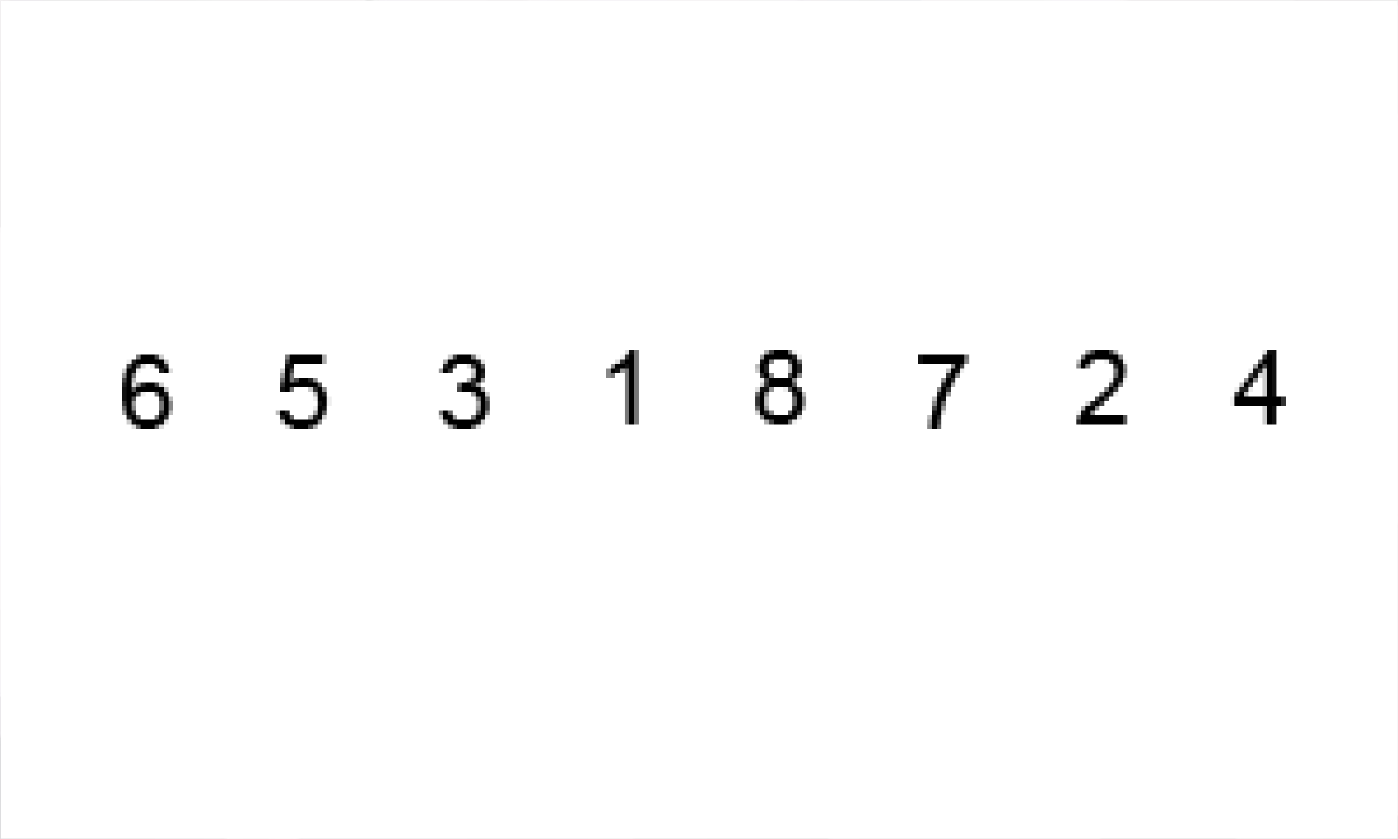
Bubble Sort

- 1. 比較兩個相鄰的元素，前面的比較大就swap
  2. 重複動作 1 直到序列結束
  3. 重複以上動作直到序列不需要再做調整

code

```
void bubbleSort ( int len ){
    for ( int i = 0 ; i < len ; i++ )
        for ( int j = 1 ; j < len ; j++ )
            if ( data[j - 1] > data[j] )
                swap ( data[j - 1], data[j] );
}
```

Bubble Sort Animation



分析一下複雜度

- 因為序列長度為 N ，所以內層迴圈要跑 N 次
- worst case 可能需要跑 N 次內層迴圈
- 所以複雜度為 $O(N^2)$

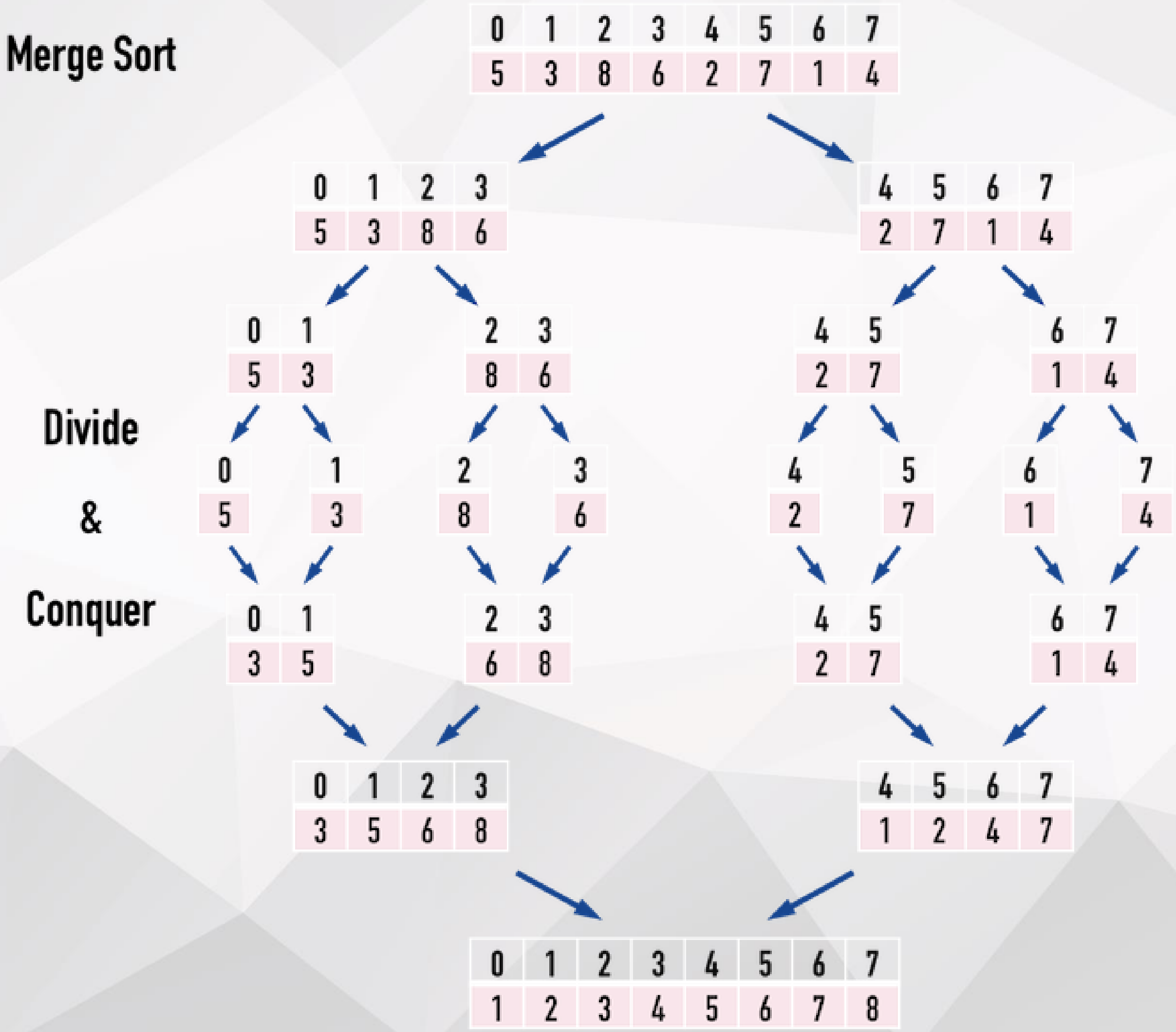
Merge Sort

Deja vu！加速囉

Merge Sort

- 分治的一種
- 不斷地把序列拆成左右子序列，並對他們遞迴
- 將兩邊的結果合併

For Example


Merge Sort
0 1 2 3 4 5 6 7
5 3 8 6 2 7 1 4
0 1 2 3
5 3 8 6
4 5 6 7
2 7 1 4
0 1
5 3
2 3
8 6
4 5
2 7
6 7
1 4
Divide
&
Conquer
0
5
1
3
2
8
3
6
4
2
5
7
6
1
7
4
0 1
3 5
2 3
6 8
4 5
2 7
6 7
1 4
0 1 2 3
3 5 6 8
4 5 6 7
1 2 4 7
0 1 2 3 4 5 6 7
1 2 3 4 5 6 7 8

6 5 3 1 8 7 2 4

code #1 - Basic version

```cpp
// basic version
void mergeSort ( int l, int r ){
    if ( l == r )
        return ;

    int m = ( l + r ) / 2, p = l, q = m + 1, idx = l;

    mergeSort ( l, m );
    mergeSort ( m + 1, r );

    while ( p <= m || q <= r )
        if ( p <= m && data[p] <= data[q] )
            swp[idx++] = data[p++];
        else
            swp[idx++] = data[q++];

    while ( p <= m )
        swp[idx++] = data[p++];

    while ( q <= r )
        swp[idx++] = data[q++];

    for ( int i = l ; i <= r ; i++ )
        data[i] = swp[i];
}
```

code #2 - advanced version

```
// advanced version
void mergeSort ( int l, int r ){
    if ( l == r )
        return ;

    int m = ( l + r ) / 2, p = l, q = m + 1, index = l;

    mergeSort ( l, m );
    mergeSort ( m + 1, r );

    while ( p <= m || q <= r )
        if ( p <= m && ( q > r || data[p] <= data[q] ) )
            swap ( swp[index++], data[p++] );
        else
            swap ( swp[index++], data[q++] );

    memcpy ( data + l, swp + l, sizeof ( int ) * ( r - l + 1 ) );
}
```

分析一下複雜度

- 每次都會把序列長度砍半
  - 也就是說有 $logN$層
    - 每一層最差需要處理 N 個數字
      - 複雜度為$O(NlogN)$

Counting Sort

"はやく！もっとはやく！"
（"快！還要更快！"）

Counting Sort

- 計算各個元素出現次數

code

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

using namespace std;

#define maxN 100005

int cnt[maxN];

int main(){
    int n, in;
    cin >> n;
    for ( int i = 0 ; i < n ; i++ ){
        cin >> in;
        cnt[in]++;
    }
    for ( int i = 0 ; i < maxN ; i++ )
        for ( int j = 0 ; j < cnt[i] ; j++ )
            cout << i << ' ';

    cout << '\n';
}
```

分析一下複雜度

- 因為要把整個陣列掃過一次 → $O(N)$
- 假設值域大小為 K，最後還要掃過整個值域 → $O(K)$
- 總複雜度 → $O(N + K)$

Discretization

既然學完排序了

假設今天有個題目

- 請計算出個元素出現的次數
- $1 \leq N \leq 10^5, |S_i| \leq 10^9$（沒錯！有負數！）

沒錯！就是離散化！

- 顧名思義只在乎元素之間的關係，並不在乎差距
- 像是我們可以把 $-1,5,9,11,2000$ 轉換成 $0,1,2,3,4$
- 於是我們就不用怕負數了 <3
- 不過還是可以用map就是了啦（小聲

先備知識

- STL函數
  - unique
  - sort（或是你要自己手寫也可以啦）
- 二分搜
  - 通常我是用 lower_bound

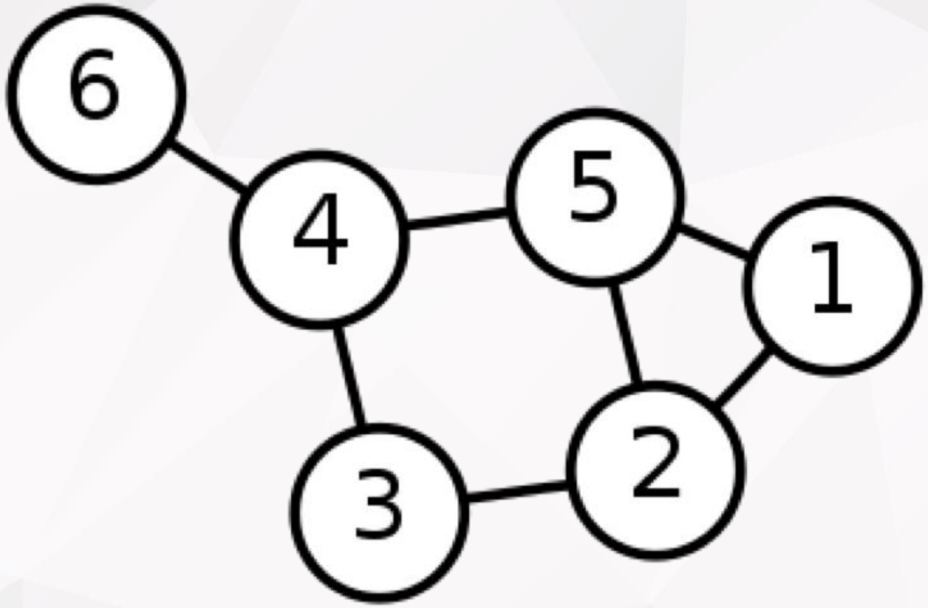
code

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

using namespace std;

#define pb push_back

int main(){
    int n, in;
    vector < int > data, lib
    cin >> n;
    for ( int i = 0 ; i < n ; i++ ){
        cin >> in;
        data.pb ( in );
    }
    lib = data;
    sort ( lib.begin(), lib.end() );
    lib.erase ( unique ( lib.begin(), lib.end() ), lib.end() );

    for ( auto i: data )
        cout << i << ' ';
    cout << '\n';
    for ( auto &i: data )
        i = lower_bound ( lib.begin(), lib.end(), i ) - lib.begin();
    for ( auto i: data )
        cout << i << ' ';
    cout << '\n';
}
```

基礎圖論

名詞解釋

- 圖：由數個節點以及邊組成
- 環：一個節點可以由不重複的路徑回到自己，則稱這條路徑為環
- 樹：沒有任何環的聯通圖
- 聯通塊：一群點中，所有點都可以直接或間接連接到其他點


6
4
5
1
3
2

圖的儲存

- 鄰接矩陣 → $G[u][v]$ 表示 u 與 v 之間有一條邊存在
- 鄰接表 → 把 u 會連接到的點都 push 進去 $G[u]$
  - 假設有一條單向邊從 u 到 v
    - G[u].push_back (v);
  - 假設有一條雙向邊於 u, v 之間，則需要
    - G[u].push_back (v), G[v].push_back (u);

假設現在有 n 個點 m 條邊

```cpp
// 2D array
#include<bits/stdc++.h>
using namespace std;
#define maxN 100005
bool G[maxN][maxN];
int main(){
    int n, m, u, v;
    cin >> n >> m;
    while ( m-- ){
        cin >> u >> v;
        G[u][v] = G[v][u] = true;
    }
}
```

```cpp
// vector
#include<bits/stdc++.h>
using namespace std;
#define maxN 100005
vector < int > edges[maxN];
int main(){
    int n, m, u, v;
    cin >> n >> m;
    while ( m-- ){
        cin >> u >> v;
        edges[u].push_back ( v );
        edges[v].push_back ( u );
    }
}
```

如果還有權重的話

```cpp
// 2D array
#include<bits/stdc++.h>
using namespace std;
#define maxN 100005
int G[maxN][maxN];
int main(){
    int n, m, u, v, w;
    cin >> n >> m;
    while ( m-- ){
        cin >> u >> v >> w;
        G[u][v] = G[v][u] = w;
    }
}
```

```cpp
// vector
#include<bits/stdc++.h>
using namespace std;
vector < pair < int, int > > edges[maxN];
// first -> 點編號, second -> 邊權重
int main(){
    int n, m, u, v, w;
    cin >> n >> m;
    while ( m-- ){
        cin >> u >> v >> w;
        edges[u].push_back ( make_pair ( v, w ) );
        edges[v].push_back ( make_pair ( u, w ) );
    }
}
```

Searching

dfs

- 全名：Depth-First Search，深度優先搜尋
- 由Hopcroft & Tarjan提出
  （以後你們就知道這一位有多煩了）
- 把根節點塞入 stack 中
  - while (stack != empty())
    - 取出第一個點，把未遍歷過的相鄰節點塞進 stack

code

```cpp
// 輸出 dfs 順序
// by. MiohitoKiri5474
#include<bits/stdc++.h>

using namespace std;

#define maxN 100005
vector < int > edges[maxN];
vector < int > output;
bool used[maxN];

// 用遞迴來模擬 stack
void dfs ( int n ){
    used[n] = true;
    output.push_back ( n );
    for ( auto i: edges[n] ){
        if ( used[i] )
            continue;
        dfs ( i );
    }
}
```

```cpp
int main(){
    int n, m, u, v, w;
    cin >> n >> m;
    while ( m-- ){
        cin >> u >> v >> w;
        edges[u].push_back ( v );
        edges[v].push_back ( u );
    }
    dfs ( 0 );
    for ( auto i: output )
        cout << i << ' ';
    cout << '\n';
}
```

bfs

- 全名：Breadth-First Search，廣度優先搜尋法
- 把根節點塞入queue中
  - while (queue != empty())
    - 取出第一個點，把未經歷過的相鄰節點塞進queue

code

```cpp
// 輸出 bfs 順序
// by. MiohitoKiri5474
#include<bits/stdc++.h>

using namespace std;

#define maxN 100005
vector < int > output, edges[maxN];
bool used[maxN];

int main(){
    int n, m, u, v, w, now;
    cin >> n >> m;
    while ( m-- ){
        cin >> u >> v >> w;
        edges[u].push_back ( v );
        edges[v].push_back ( u );
    }
```

```cpp
    queue < int > q;
    q.push ( 0 );
    while ( !q.empty() ){
        now = q.front();
        q.pop();
        used[now] = true;
        output.push_back ( now );
        for ( auto i: edges[now] ){
            if ( used[i] )
                continue;
            q.push ( i );
        }
    }
    for ( auto i: output )
        cout << i << ' ';
    cout << '\n';
}
```

Disjoint Set

Disjoint Set

- 可以在良好的複雜度內，查詢兩個元素是否在同一個集合
- 同一個元素不會同時出現在兩個集合內

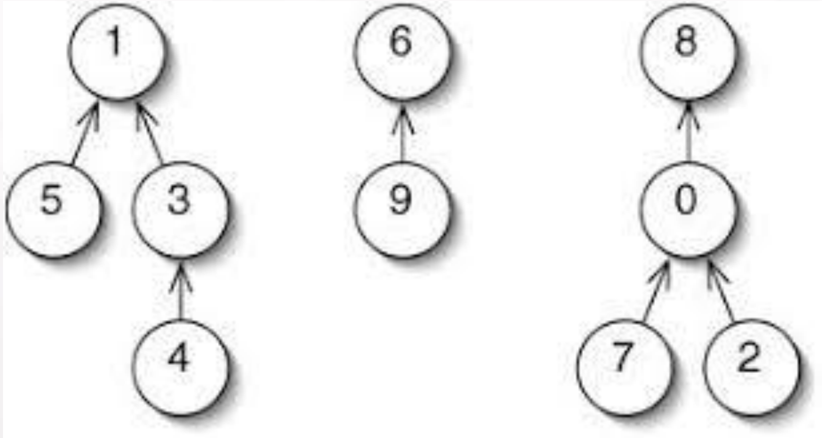
Disjoint Set 操作

- 查詢元素所屬組別
- 加入新元素進入集合
- 合併兩個集合
- 查詢集合大小

Disjoint Set 概念

- 用一顆樹表達一個組別

  → 如果兩個相異元素根節點相同，則兩元素屬於同一個集合



Disjoint Set Initialization

```
#define maxN 10005
int dis[maxN], sz[maxN];
inline void init ( void ){
    for ( int i = 0 ; i < maxN ; i++ ){
        dis[i] = i;
        sz[i] = 1;
    }
}
```

Disjoint Set Find

```
int find ( int n ){
    if ( dis[n] == n )
        return n;
    return find ( dis[n] );
}
```

```
int find ( int n ){
    if ( dis[n] == n )
        return n;
    return dis[n] = find ( dis[n] );
}
```

欸，好像怪怪的

假設一下

- 想一下，如果這一個結構是

$$1 \to 2 \to 3 \to 4 \to \ldots \to n$$

- 那麼每次 find (1) 就要跑 n 次，複雜度 $O(n)$ 聽起來很爛

路徑壓縮

- 因為第一次查詢的時候就已經知道最頂端的節點是什麼了
- 所以記錄下來，下一次就省掉許多時間了
- 有數學證明可以把複雜度從 $O(N)$ 壓到 $O(\alpha(N))$
- α 指的是反阿克曼函數，簡單來說就是成長速度非常慢的函數

Disjoint Set Union

```cpp
inline void Union ( int a, int b ){
    a = find ( a ), b = find ( b );
    if ( sz[a] > sz[b] )
        swap ( a, b );
    dis[a] = b;
    sz[b] += sz[a];
}
```

codes on github

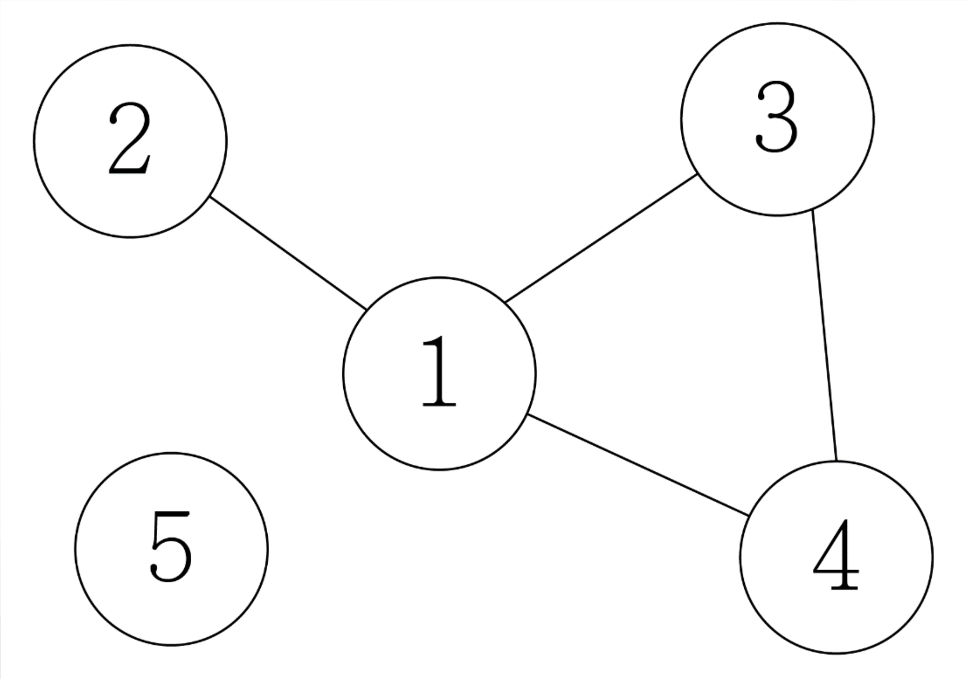
- https://github.com/MiohitoKiri5474/CodesBackUp/tree/master/ncku-icpc/2020/week5

- https://ppt.cc/fKPjIx

Topological Sort

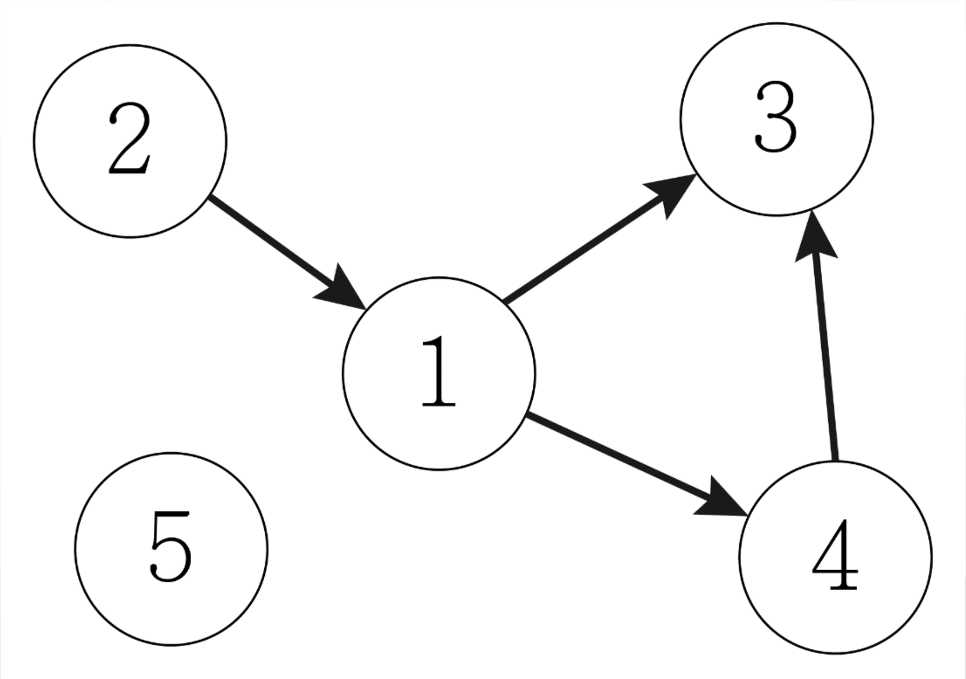
點的度數

- 在一張圖中，每個節點都有它的「度數」，一個節點的度數代表這個節點連接幾條邊
- 如右圖，
  點 1 的度數為 3 ，
  點 4 的度數為 2 ，
  點 5 的度數為 0 。



入度&出度

- 但如果這是一張有向圖，又可以將其細分為「入度」與「出度」，入度就是連進這個節點的邊，出度就是從這個點連出去的邊。
- 如右圖，
  點 1 的入度為 1 、出度為 2 ，
  點 4 的入度為 1 、出度為 1 ，
  點 5 的入度為 0 、出度為 0 。

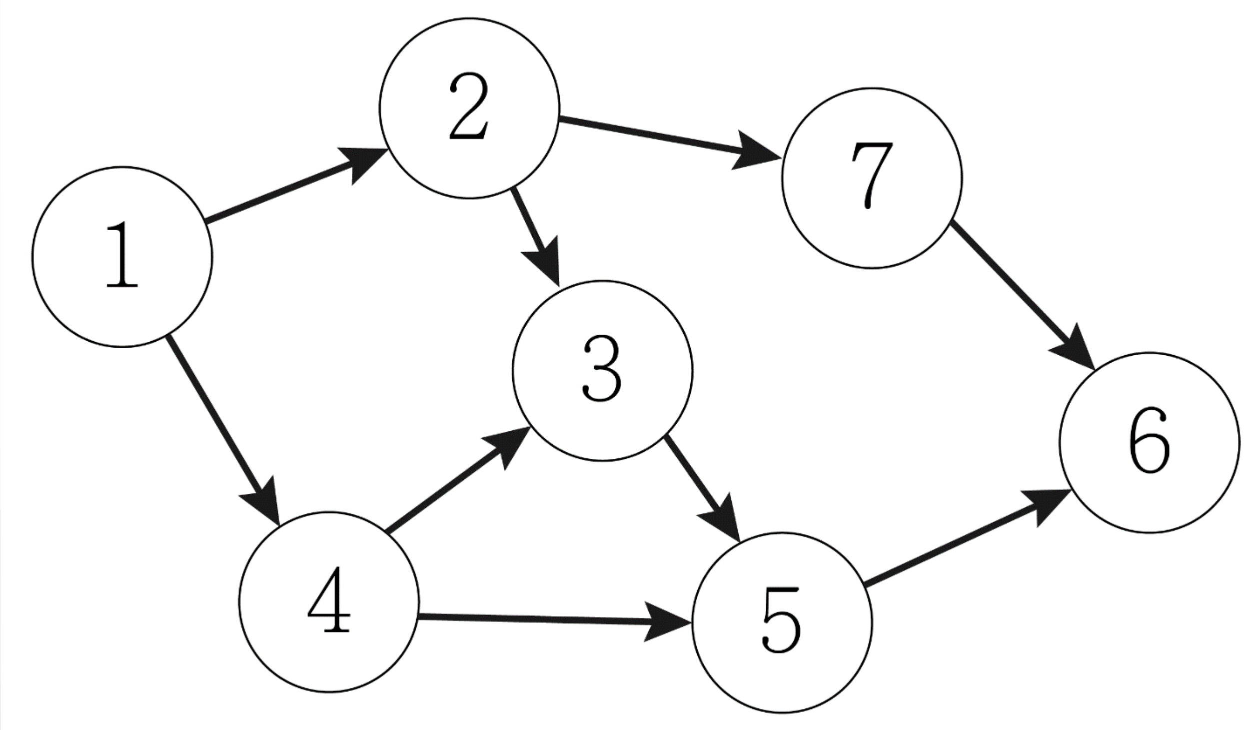


Topological Sort

- 給定一張有向圖，你要為所有節點訂定一個順序 $\{v_1, v_2, \cdots, v_n\}$，且這個順序滿足 $\forall i \leq j$ 皆不存在從 v_j 走到 v_i 的路徑，則此順序就是這張圖的拓樸排序。

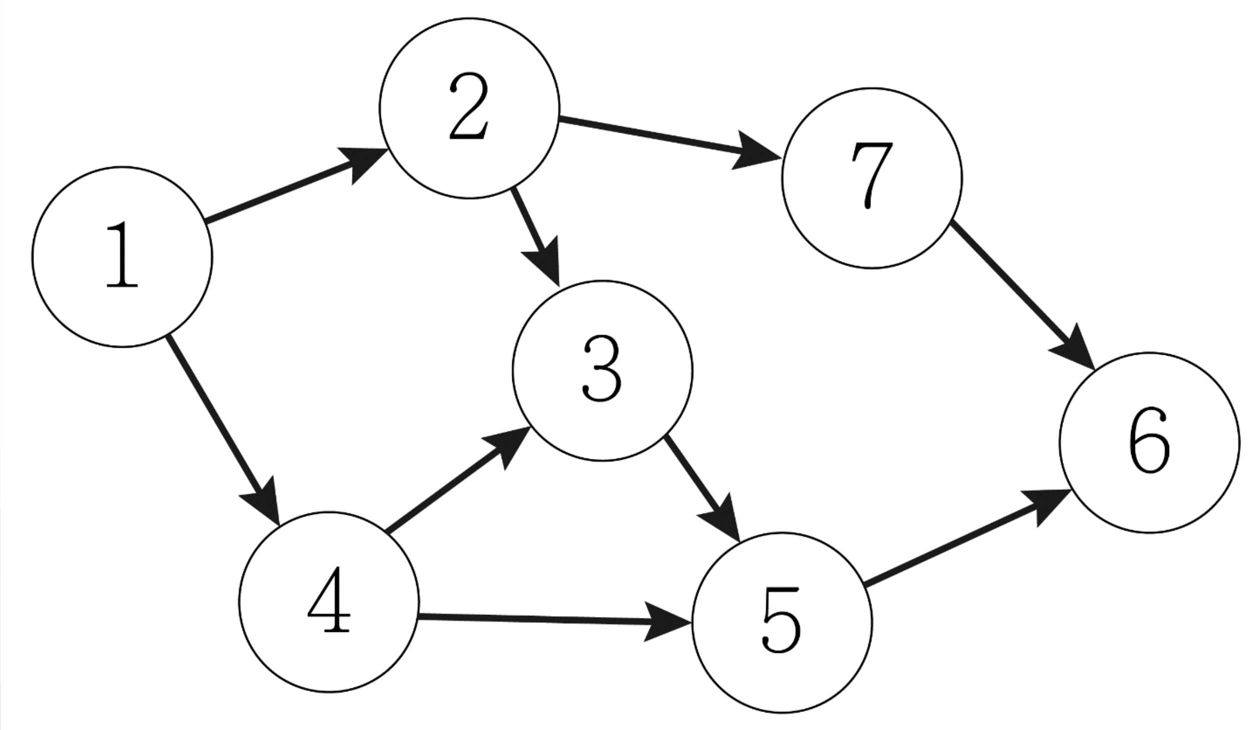
Topological Sort

- 如下圖中，{1,2,4,3,5,7,6} 就是一組合法的拓樸排序，因為不存在任一組 $i \leq j$ 且有任一條路徑可以從 v_j 走到 v_i 。



Topological Sort

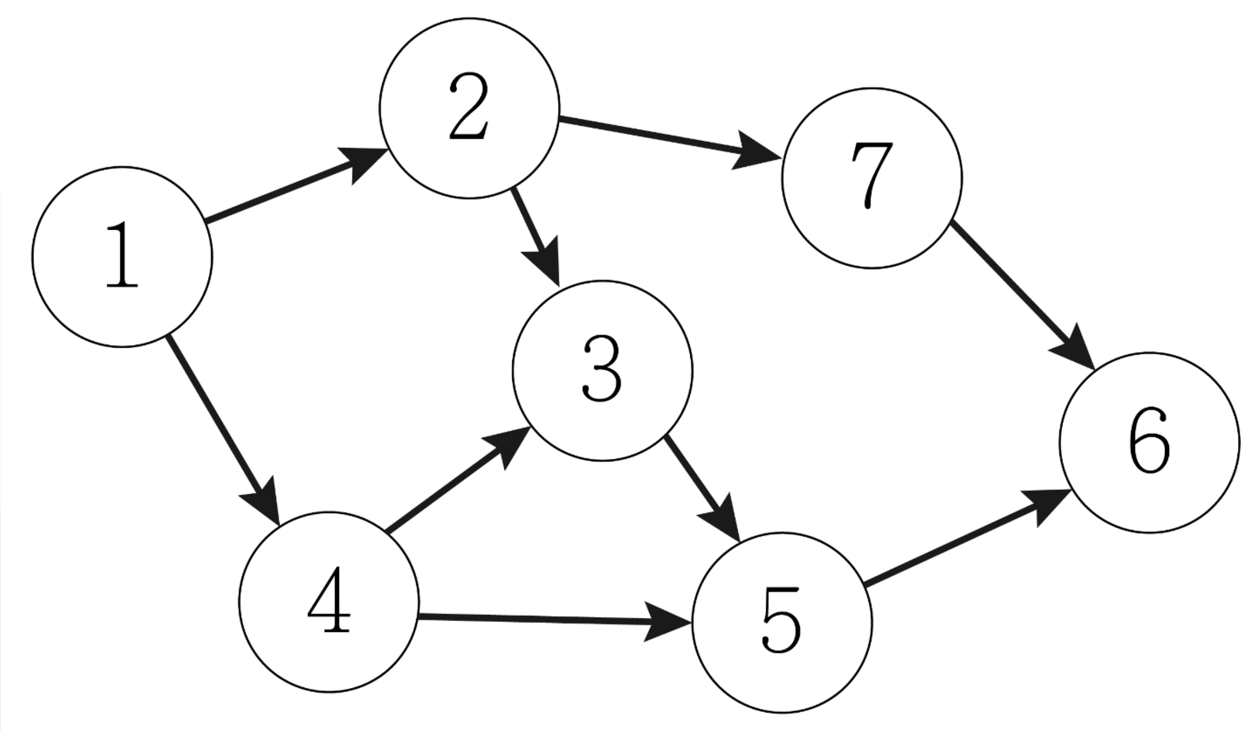
- 換個說法，一個點如果要被排進序列時，所有指向它的點都必須被排進去了，以下圖為例。
- 點 2 必須等點 1 被排進去後才能被排進去。
- 點 3 必須等點 2 與點 4 被排進去後才能被排進去。



- 一張圖可能存在多組拓樸排序。

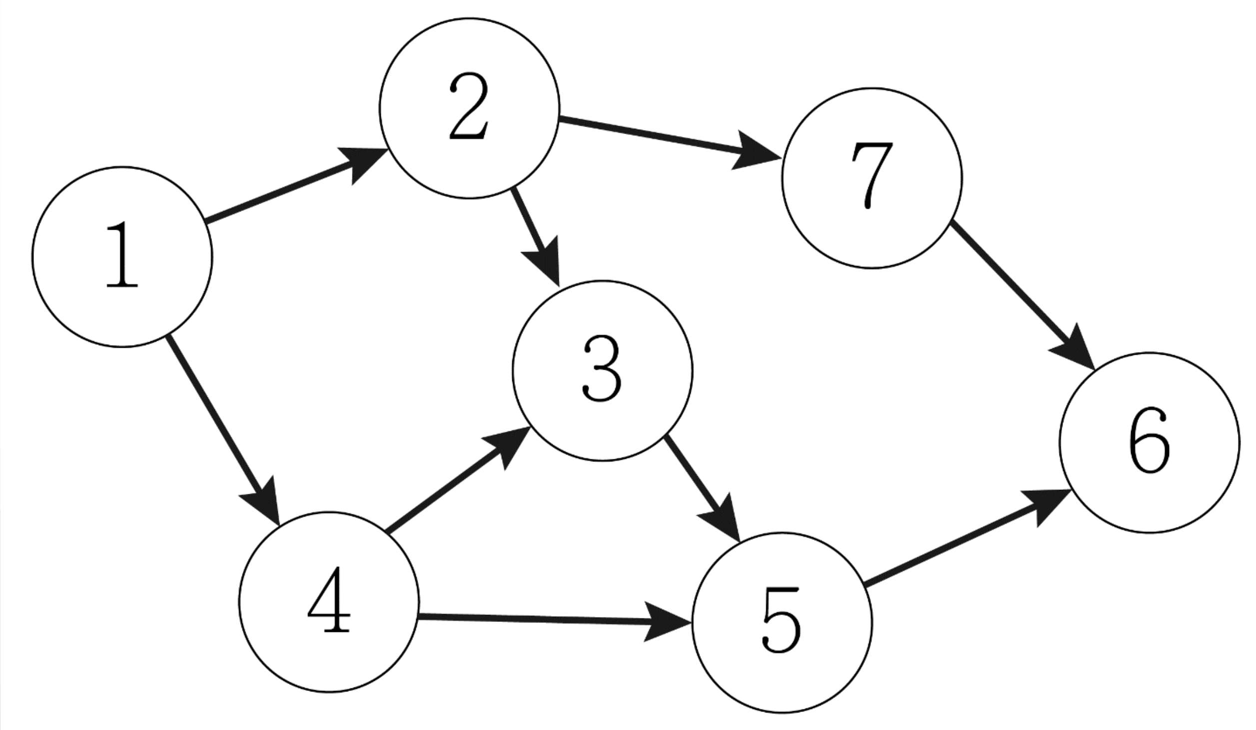
做法

- 那麼做法呼之欲出了，只要某個點的入度為 0 時，這個點就可以被排進去拓樸序列。



做法

- 只要每次都將入度為 0 的點排進拓樸序列尾端，並將該點及其連出去的邊移出這張圖，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。
- 實作過程可以使用 stack 或 queue。



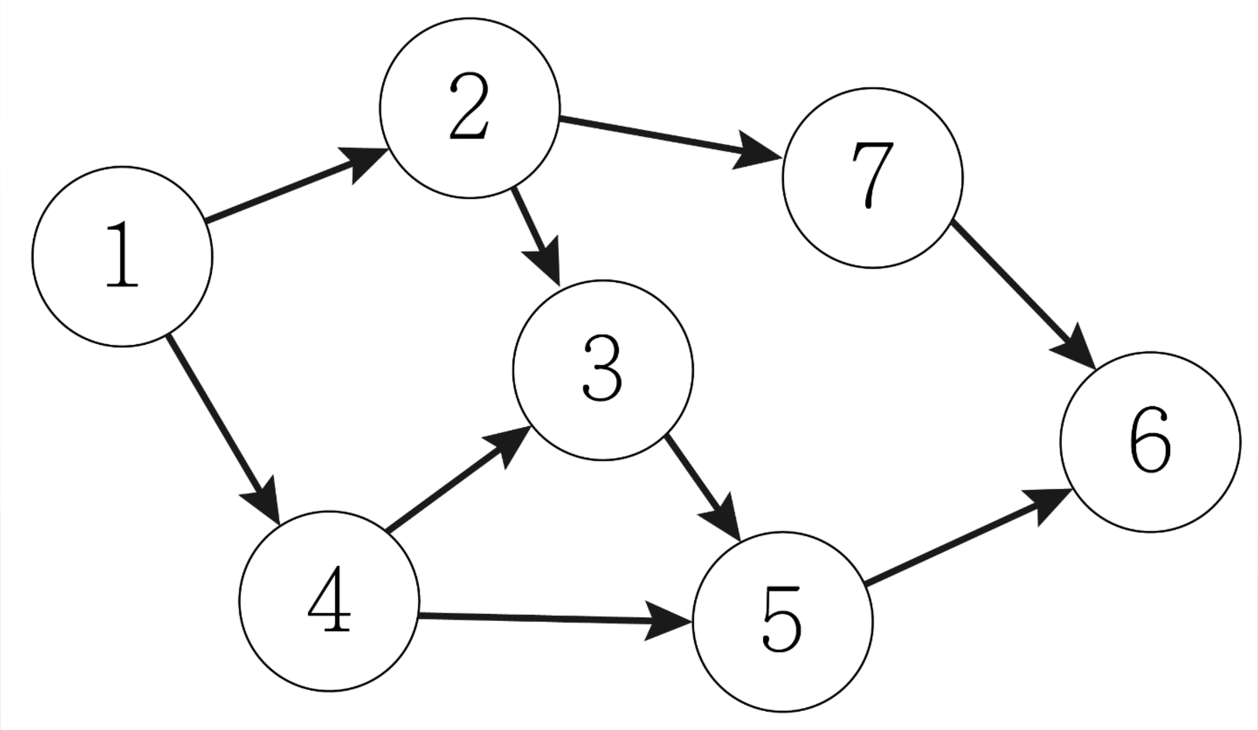
- 本篇教學使用 queue。

做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

Topological Sort



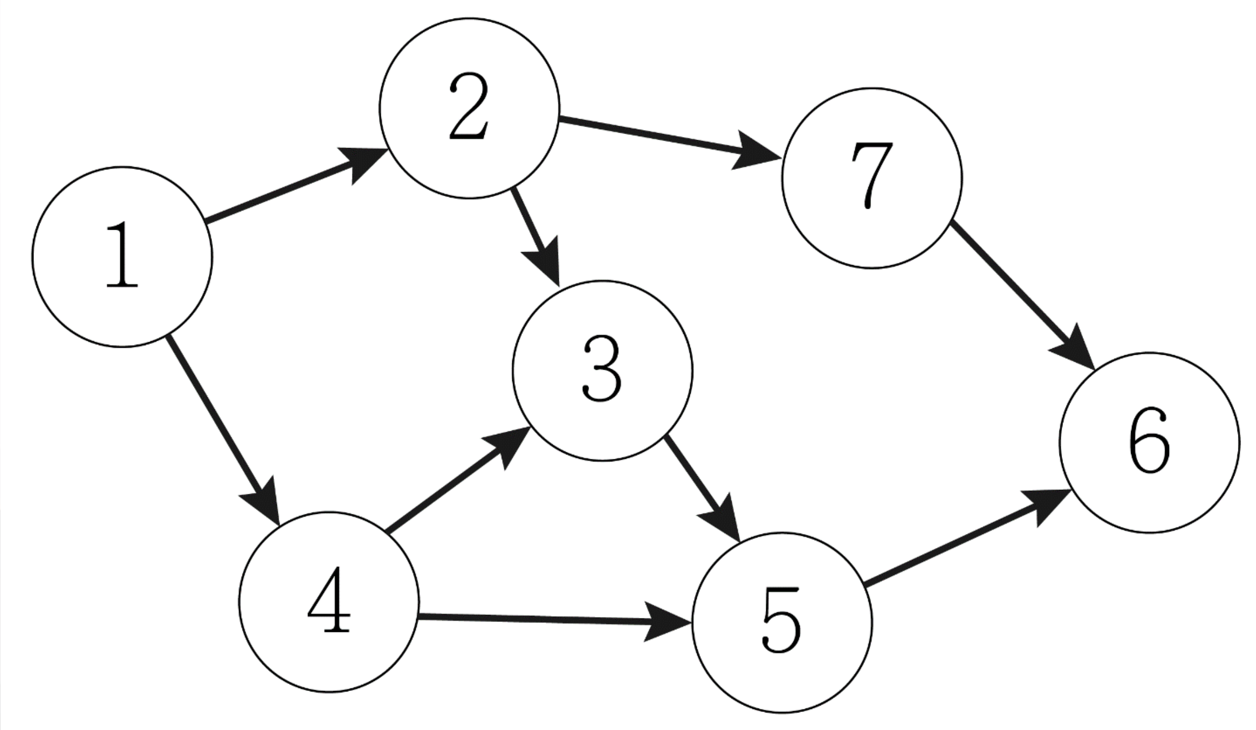
做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

1

Topological Sort



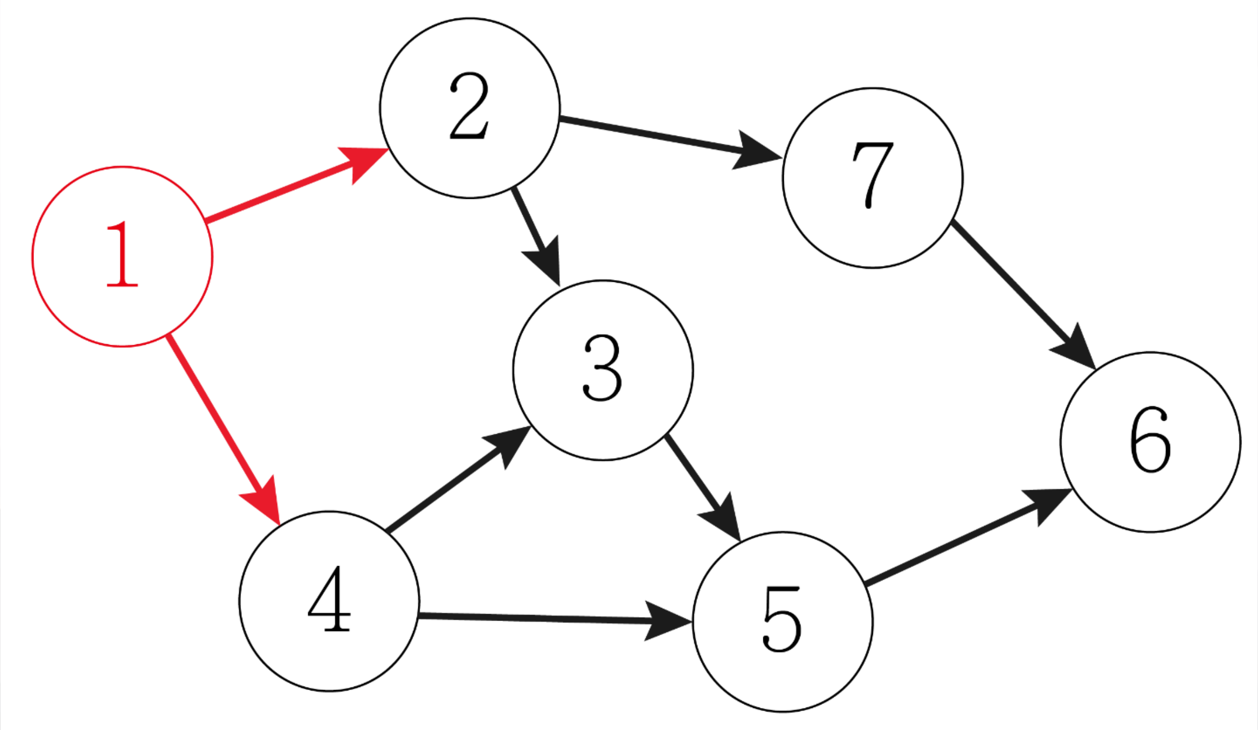
做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

Topological Sort

1



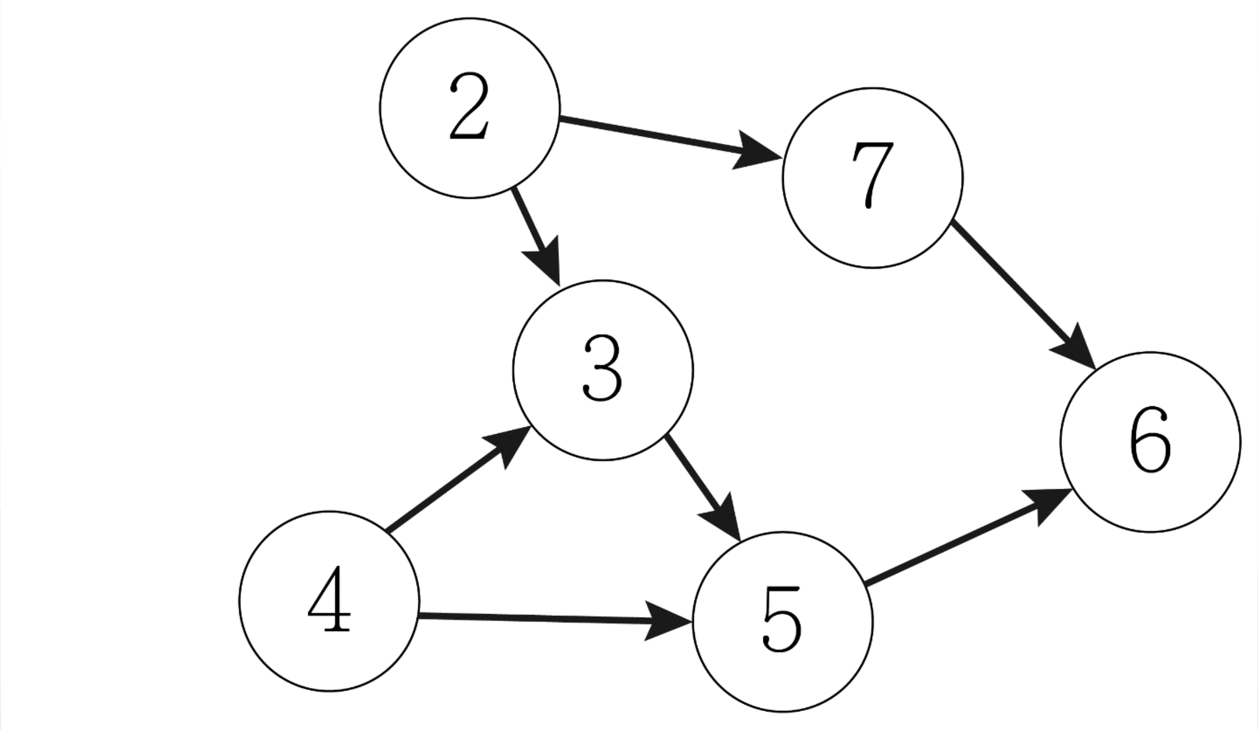
做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

Topological Sort

1



做法

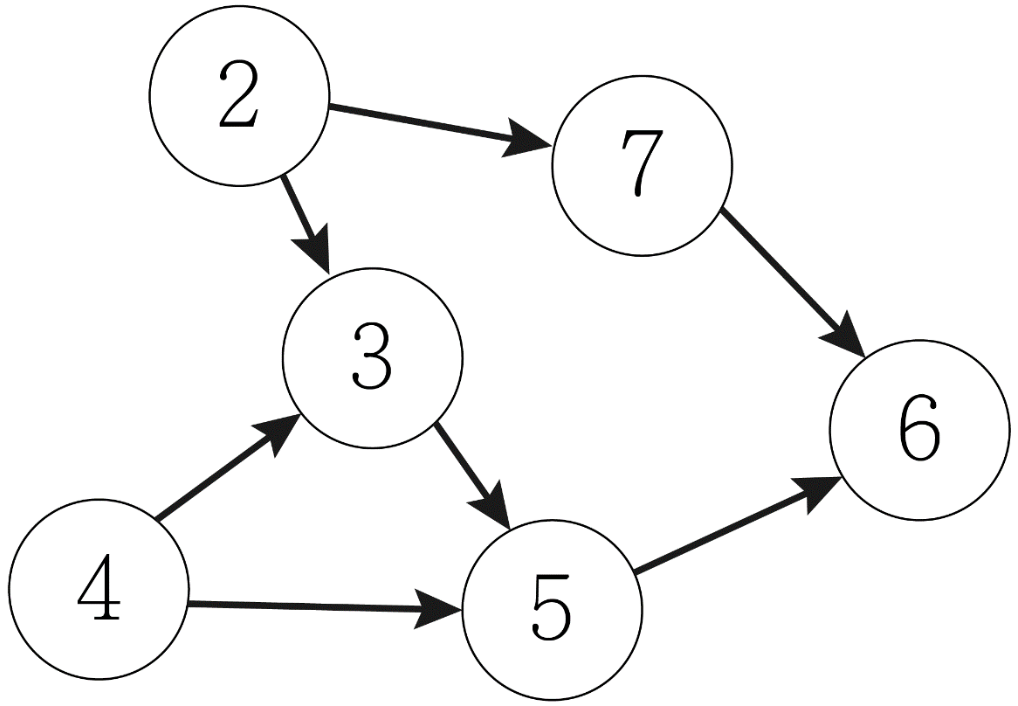
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

2 4

Topological Sort

1



做法

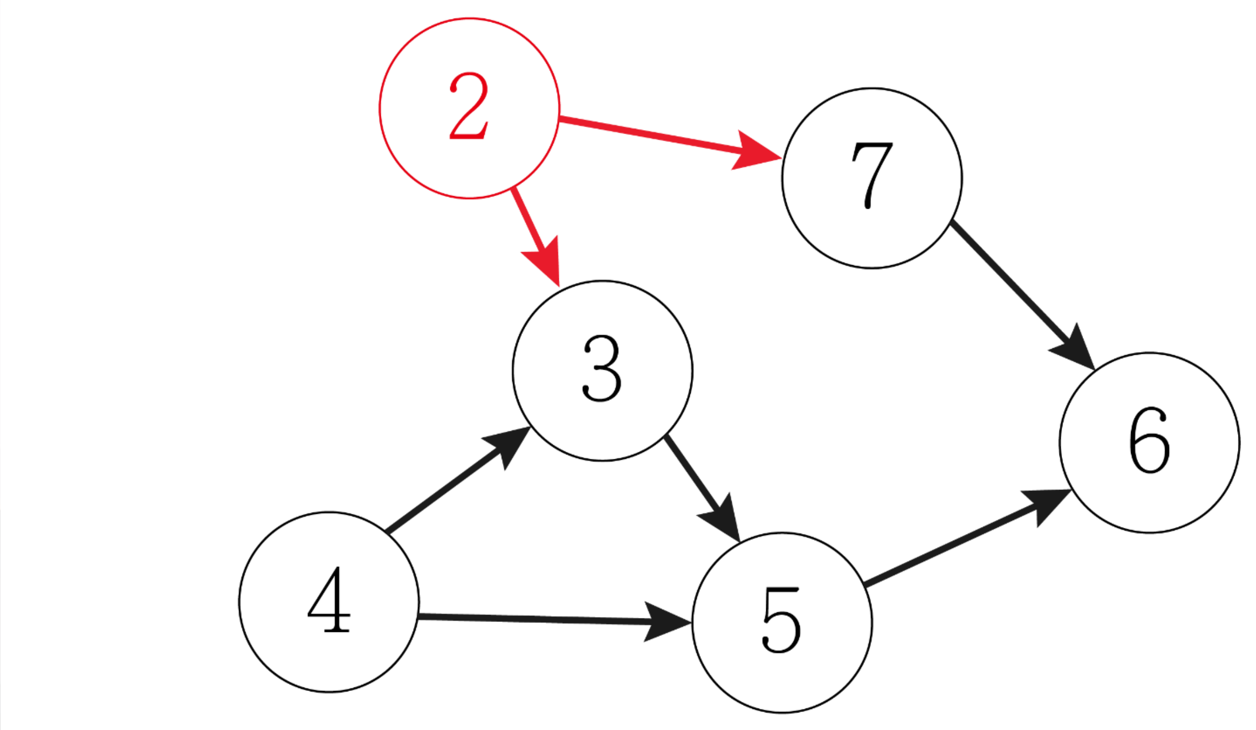
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

4

Topological Sort

1 2



做法

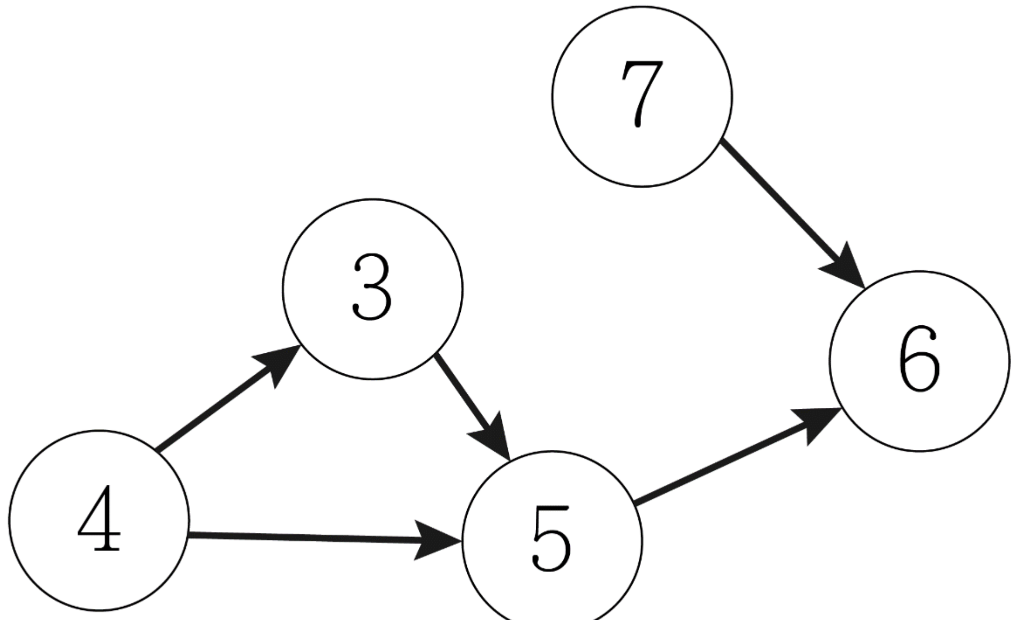
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

4

Topological Sort

1 2



做法

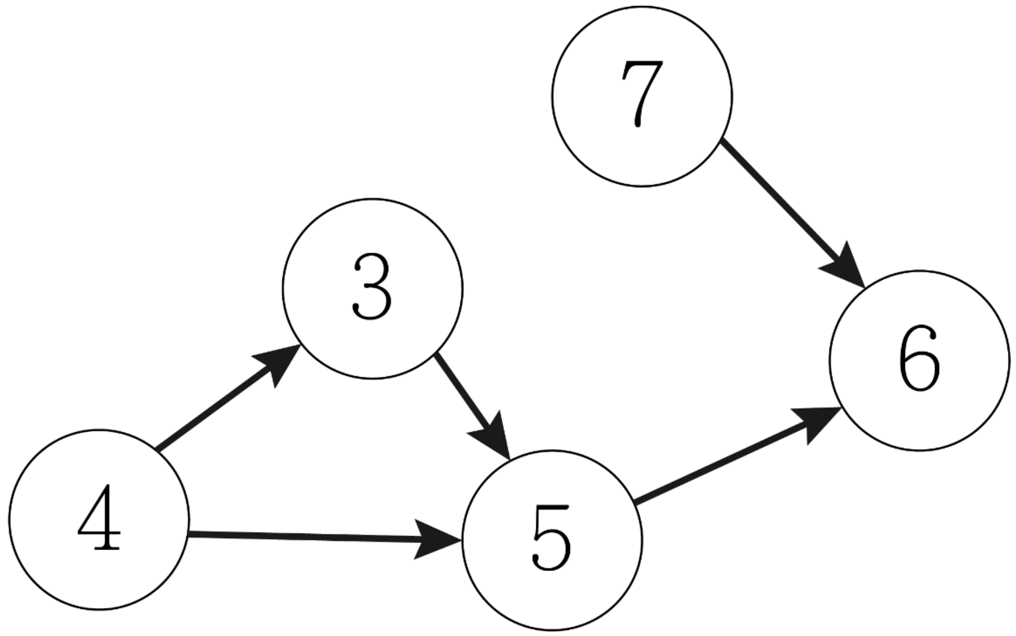
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

4 7

Topological Sort

1 2



做法

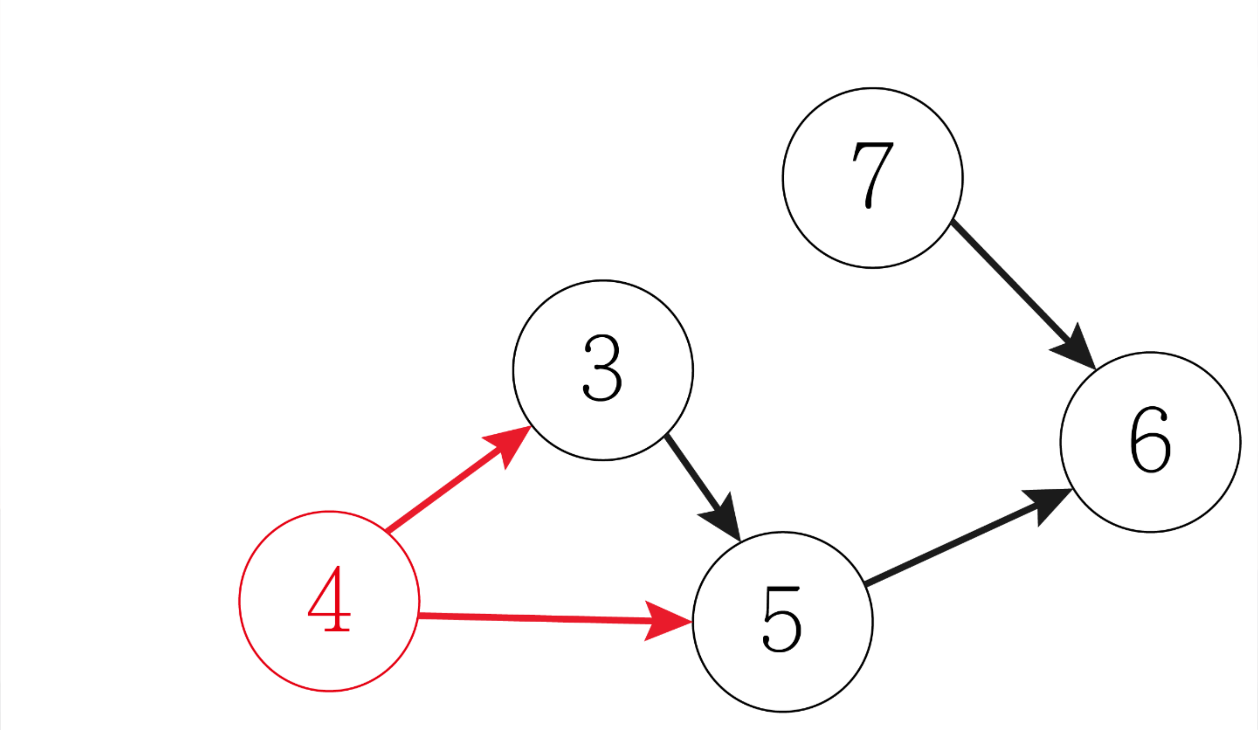
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

7

Topological Sort

1 2 4



做法

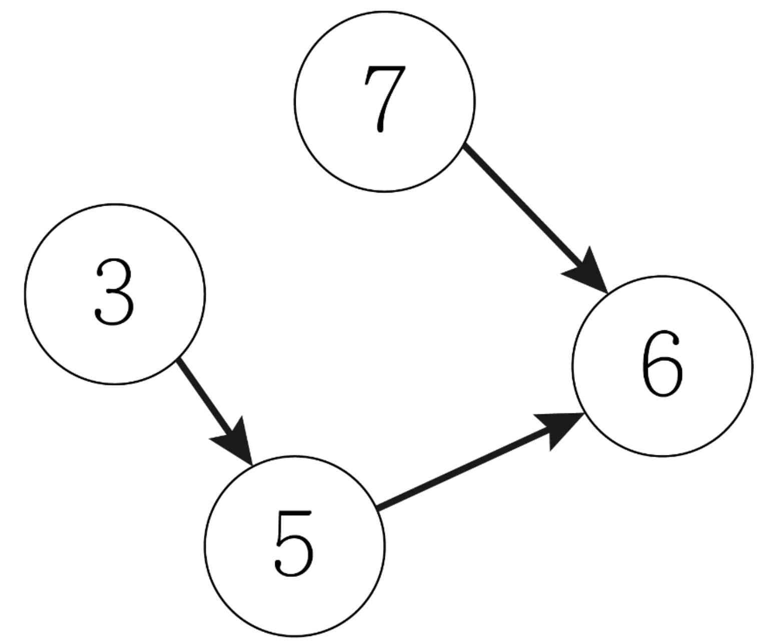
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

7

Topological Sort

1 2 4



做法

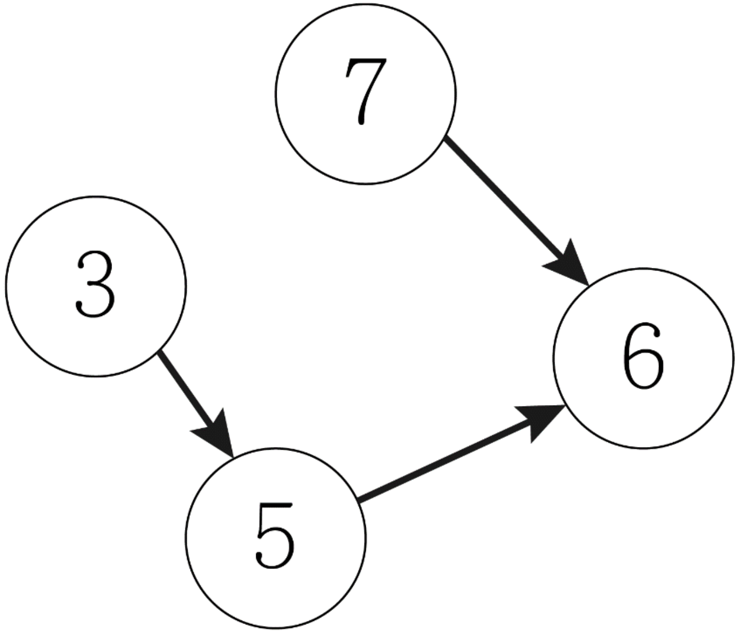
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

7 3

Topological Sort

1 2 4



做法

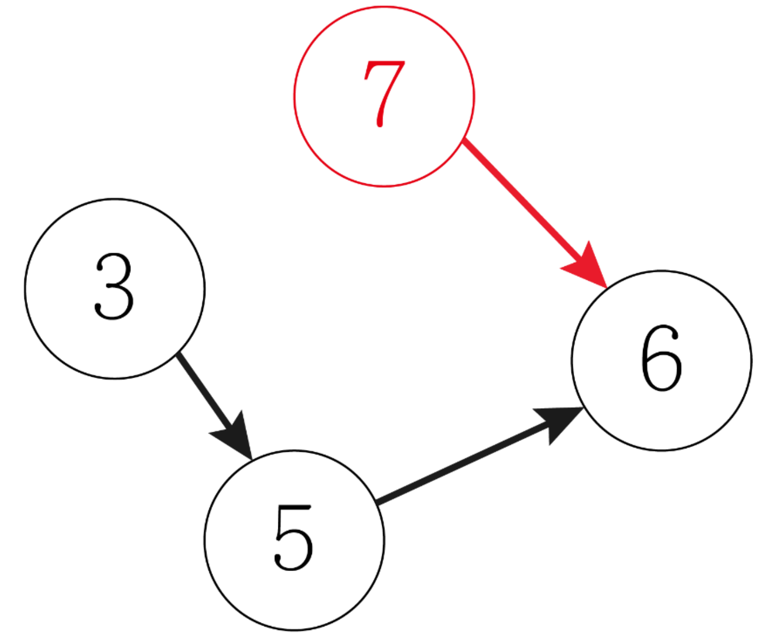
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

3

Topological Sort

1 2 4 7



做法

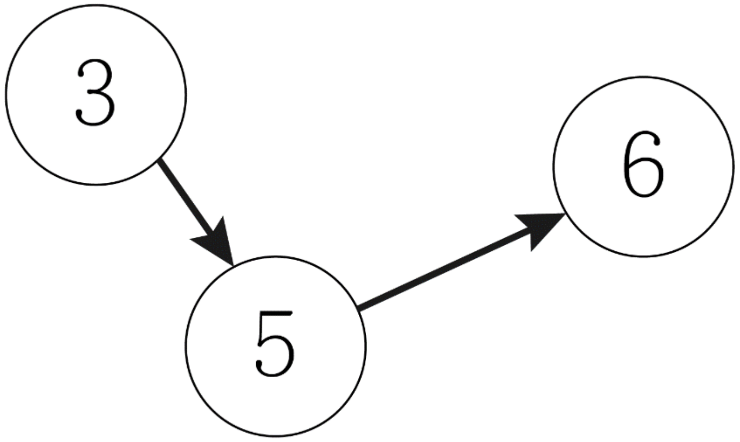
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

3

Topological Sort

1 2 4 7



做法

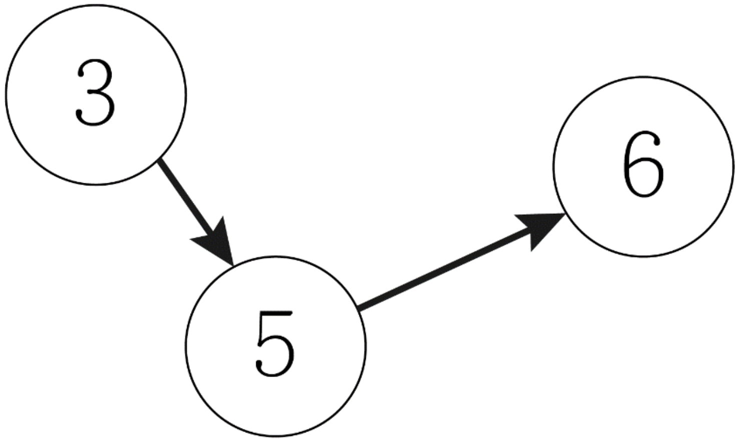
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

3

Topological Sort

1 2 4 7



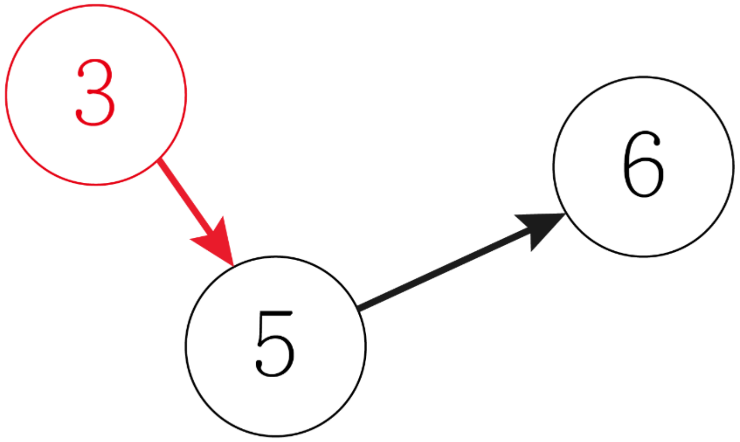
做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

Topological Sort

1 2 4 7 3



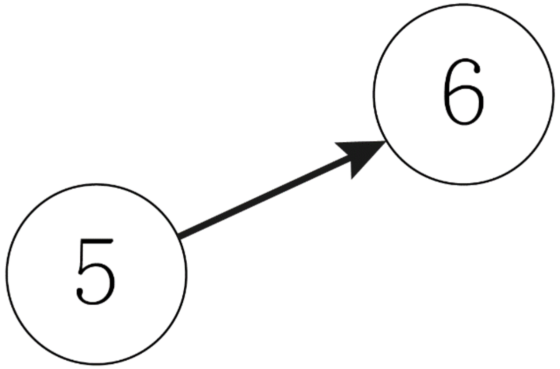
做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

Topological Sort

1 2 4 7 3



做法

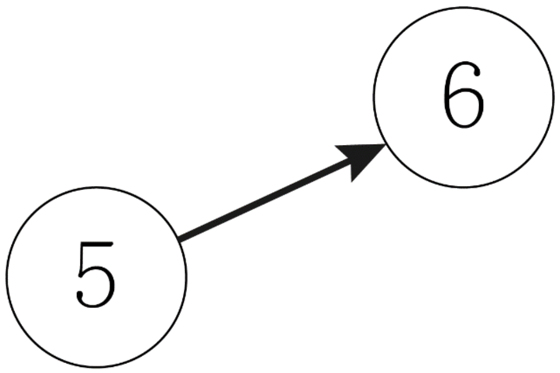
- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

5

Topological Sort

1 2 4 7 3



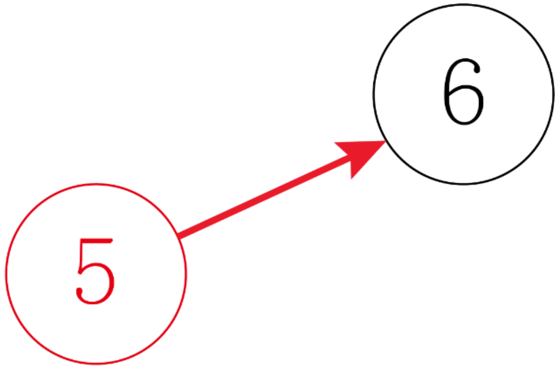
做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

Topological Sort

1 2 4 7 3 5



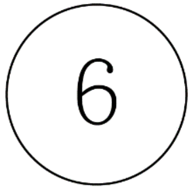
做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

Topological Sort

1 2 4 7 3 5



做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

6

Topological Sort

1 2 4 7 3 5

做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

Topological Sort

1 2 4 7 3 5 6

做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Queue

Topological Sort

1 2 4 7 3 5 6

做法

- 只要每次都將入度為 0 的點放在當前序列尾端，並將該點移出圖中，重複下去直到圖上所有點都被移除時，就完成該圖的拓樸排序了。

Topological Sort

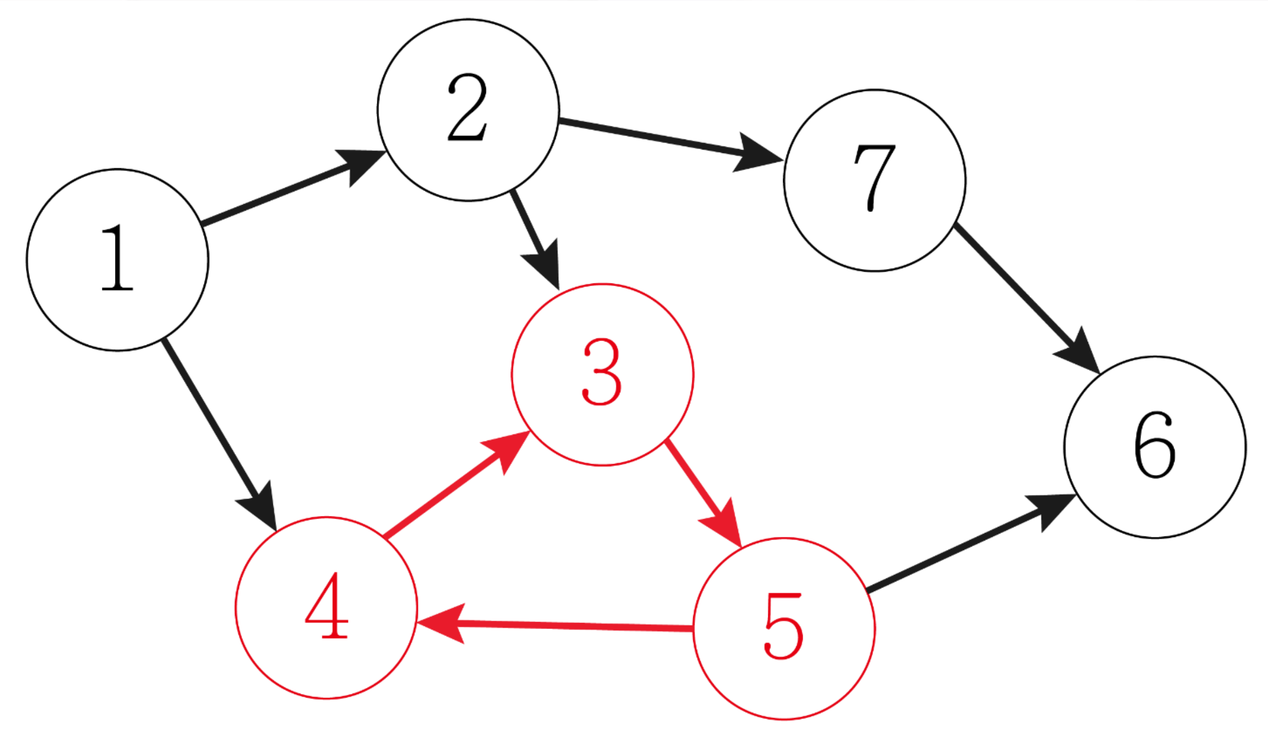
1 2 4 7 3 5 6

做法

- 只要開一個陣列記錄每個節點的入度，在每一次要拔一個點時，將該點指到的所有點入度都減 1 ，如果那個點入度為 0 時，將它丟進 queue 中。
- 演算法的複雜度：
  遍歷所有點， $O(V)$ 。
  遍歷所有邊， $O(E)$ 。
  總複雜度， $O(V+E)$ 。

Topological Sort

- 欸那一張圖一定有拓樸排序嗎？
- 其實不一定
- 可以看看右圖



Topological Sort

- 當一張圖出現環時，這張圖不存在拓樸排序。
- 在程式中，當你的 queue 中沒任何元素且整張圖未遍歷完時，這張圖不存在拓樸排序。
- 一張圖存在拓樸排序若且唯若這張圖為有向無環圖，通常簡稱為 DAG(Directed Acylic Graph)。

Source Code

```
for(int i = 1; i <= n; i++)
    if(in[i] == 0)
        q.push(i);  //將入度為 0 的點丟進 queue
while(!q.empty()){
    tmp = q.front(), q.pop();
    topo.push_back(tmp);  //將現在處理的點排進序列
    for(int &i : v[tmp]){
        in[i]--;  //將指到的點入度減 1
        if(in[i] == 0)  //若該點入度為 0 則丟進 queue
            q.push(i);
    }
}
if(topo.size() < n)  //當拓樸序列長度不為 n 時, 代表圖沒遍歷完
    cout << "No topological sort";
```

Questions?