主題: Graphs and DFS

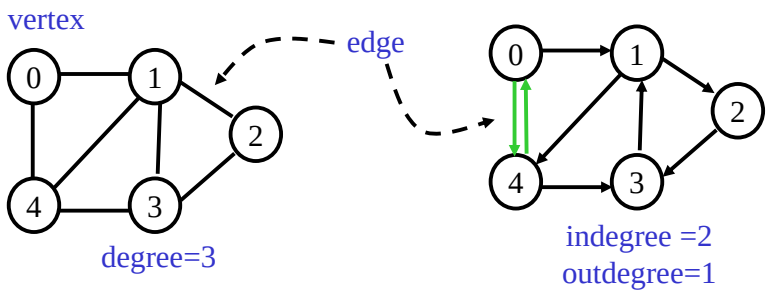
- 基礎
- 應用
- 作業與自我挑戰

基礎

- Definitions
- Adjacency matrix
- Adjacency-lists
- Usual formats of input
- DFS

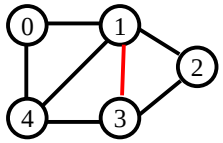
Definitions

undirected graph **directed graph**



Adjacency matrix

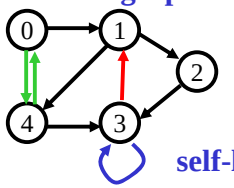
undirected graph



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	**1**	1
2	0	1	0	1	0
3	0	**1**	1	0	1
4	1	1	0	1	0

有時存 1

directed graph



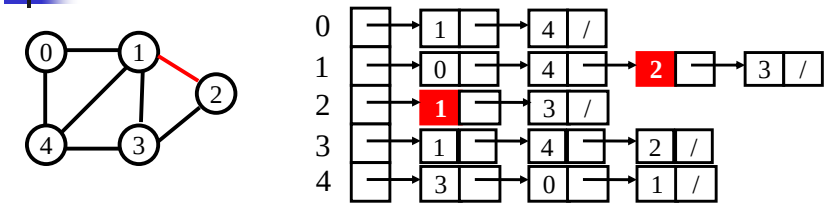
	0	1	2	3	4
0	0	1	0	0	1
1	0	0	1	**0**	1
2	0	0	0	1	0
3	0	**1**	0	**1**	0
4	1	0	0	1	0

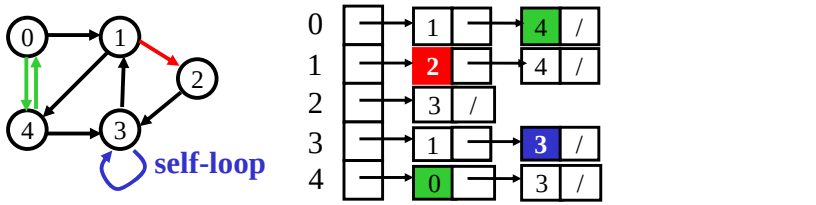
Adjacency matrix (cont.)

- an n × n matrix, where n = |V|
  - A[i, j] = 1: edge (i, j) 存在
  - A[i, j] = 0: edge (i, j) 不存在
- For an undirected graph, A[i, i] 存 0 或 1 由題目決定
- 當 edge 有 length 時， A[i, j] 存 edge (i, j) 的 length，∞ 表示這條 edge 不存在

use a special number to denote it

Adjacency-lists



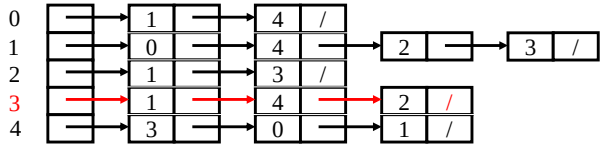


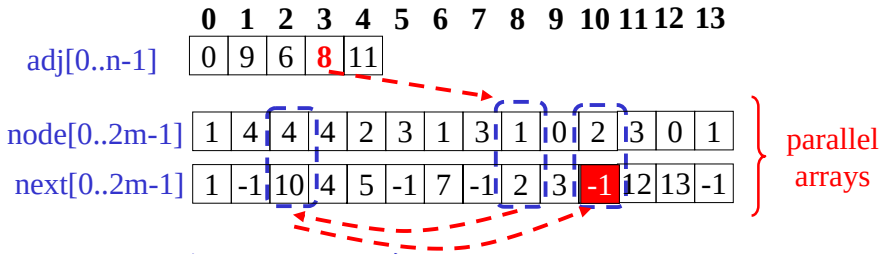
Adjacency-lists (cont.)

- the size is O(n + m), where m = |E|
- 當 edge 有 length 時， 可以將 length 存在對應的 node 上
- 儘可能用 parallel arrays 去 implement ，避免使用 dynamic memory allocation

Implementation (I)

adj[0..n-1]

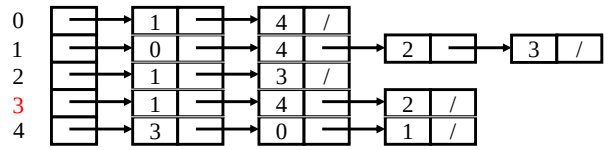




- 有 edge length 時使用另一個 array len[0..2m-1]

Implementation (II)

- 集中連續擺放



	0	1	2	3	4	5	6	7	8	9	10	11	12	13
degree[0..n-1]	2	4	2	3	3									
adj[0..n-1]	0	2	6	8	11									
node[0..2m-1]	1	4	0	4	2	3	1	3	1	4	2	3	0	1

Adjacency-matrix vs. adjacency-lists

- 儘可能用 adjacency-matrix
  - 比較簡單
  - 可直接查表知道 edge (i, j) 是否存在
  - 雖然 storage 和 time complexity 較高，不過大部分情況下用 adjacency-matrix 已足夠

Usual formats of input (I)

- 直接給 adjacency-matrix 或 adjacency-lists

```
0 1 0 0 1                2 5
1 0 1 1 1                1 5 3 4
0 1 0 1 0      或        2 4
0 1 1 0 1                2 5 3
1 1 0 1 0                4 1 2
```

- 直接讀進來存進 adjacency-matrix 或 adjacency-lists 中

Usual formats of input (II)

- 所有 edges 以任意順序輸入，如:
  (2, 9), (4, 7), (2, 8), (4, 6), (0, 1), (0, 2), (8, 10), (1, 5), (1, 3), (9, 11), (1, 4)
- 每次讀進一條 edge 之後，就把它加入 adjacency-matrix 或 adjacency-lists 中
  - 如果是 undirected edges，要記得雙向都加

Usual formats of input (III)

- 與 (II) 相同，不過 vertex 的編號可能很大，不是 0 到 n – 1，或 vertex 的 id 是字串

```
1  100000
100000  33333
33333  1
```

或

```
Mary Petter
Peter Allen
Mary William
George Tom
Mary Edward
Tom John
George Peter
Tom Mary
Tom Adam
```


Re-labeling

- 把不能當作 index 的文字 (或數字) id，轉換成 0 到 n – 1 之間的整數
  - 1 => 0, 33333 => 1, 100000 = > 2
  - Mary => 0, Peter => 1, Allen => 2, ...
- 準備一個對應表 (mapping)，每看到一個字串，就到表中查詢是否已經出現過，如果出現過就回報對應此字串的 index，否則就新增一筆字串

Re-labeling (cont.)

Mary Peter (0, 1)
Peter Allen (1, 2)
Mary William (0, 3)
George Tom
Mary Edward
Tom John
George Peter
Tom Mary
Tom Adam

ID → 名字

0	Mary
1	Peter
2	Allen
3	William
4	George
5	Tom
6	Edward
7	John
8	Adam

Linear Search

建完後 sort

名字 → ID

Adam	8
Allen	2
Edward	6
George	4
John	7
Mary	0
Peter	1
Tom	5
William	3

Binary Search

Two-phases construction

- 如果 vertices 的個數很多，讀進 input 時每次都用 linear search 去查 mapping，時間是 O(mn)，可能無法接受
- Two-phases:
  - 先讀進所有 input 存起來
  - sort 後造表 (拿掉重複的)
  - 最後再建 adjacency-matrix/adjacency-lists (binary search)

Two-phases

ID ↔ 名字

0	Adam
1	Allen
2	Edward
3	George
4	John
5	Mary
6	Peter
7	Tom
8	William

Binary Search

Phase 1: Sort (remove duplicates)

O(m lg m)

Mary Peter
Peter Allen
Mary William
George Tom
Mary Edward
Tom John
George Peter
Tom Mary
Tom Adam

Phase 2

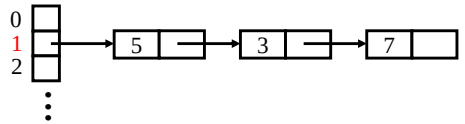
(5, 6)
(6, 1)
(5, 8)
(3, 7)
(5, 2)
(7, 4)
(3, 6)
(7, 5)
(7, 0)

O(m lg n)

Time: O(m lg m) = O(m lg n)

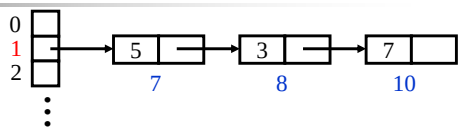
Usual formats of input (VI)

- 所有 edges 以任意順序輸入，如:
  (2, 9), (4, 7), (2, 8), (4, 6), (0, 1), (0, 2),
  (8, 10), (1, 5), (1, 3), (5, 11), (1, 7)
- 規定 adjacency-list 中的 neighbors 需按出現順序排列



- 如何完成: 記住目前 的最後一個, 每次都加到尾巴

按出現順序排列



- last [i] 記錄目前 adj(i) 的最後一個 node

(1, 5)
(1, 3)
(5, 11)
(1, 7)

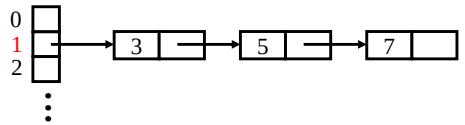
	0	1	2	3	4	5	6	7	8	9
adj[0..n-1]	4	7	0		1	9			6	
last[0..n-1]	5	10	2	-1	3	9	-1	-1	6	-1

	0	1	2	3	4	5	6	7	8	9	10	11	12
node[0..m-1]	9	7	8	6	1	2	10	5	3	11	7		
next[0..m-1]	2	3			5			8	10				

used

Usual formats of input (V)

- 所有 edges 以任意順序輸入，如:
  (2, 9), (4, 7), (2, 8), (4, 6), (0, 1), (0, 2),
  (8, 10), (1, 5), (1, 3), (9, 11), (1, 7)
- 規定 adjacency-list 中的 neighbors 需按 id 大小順序排列



- 如何完成: 利用 sort

利用 sort 整理

- 記住每條 edge 的 head 和 tail 以做比較
  - 整理之前

edge	0	1	2	3	4	5	6	7	8	9	10
head	2	4	2	4	0	0	8	1	1	9	1
tail	9	7	8	6	1	2	10	5	3	11	4

  - 整理之後 (sort: 先比 head, 平手時再比 tail)

edge	0	1	2	3	4	5	6	7	8	9	10
head	0	0	1	1	1	2	2	4	4	8	9
tail	1	2	3	4	5	8	9	6	7	10	11

vertex 順序

依 id 大小

轉換成 adjacency-lists

edge	0	1	2	3	4	5	6	7	8	9	10
head	0	0	1	1	1	2	2	4	4	8	9
tail (node)	1	2	3	4	5	8	9	6	7	10	11

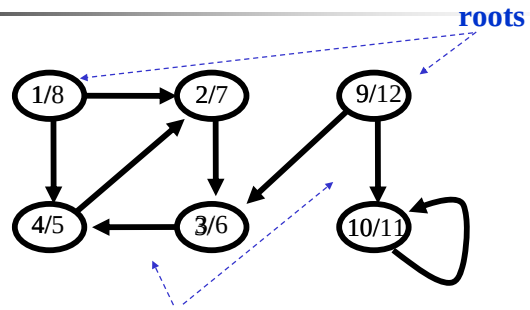
	0	1	2	3	4	5	6	7	8	9
adj[0..n-1]	0	2	5	-1	7	-1	-1	-1	9	10
degree[0..n-1]	2	3	2	0	2	0	0	0	1	1

整理成連續擺放形式

- 對 edge 陣列做 sort，可以同時完成符合規定順序與連續擺放兩種效果
- Sort 的規則
  - 如規定依 id 順序: 先比 head, 平手時再比 tail
    - $e_3 = (4, 7)$, $e_{11} = (2, 8)$ ⟹ $(2, 8) < (4, 7)$
    - $e_5 = (2, 9)$, $e_{11} = (2, 8)$ ⟹ $(2, 8) < (2, 9)$
  - 如規定依出現順序: 先比 head, 平手時再比出現順序
    - $e_5 = (2, 9)$, $e_{11} = (2, 8)$ ⟹ $(2, 9) < (2, 8)$ (e_5 e_{11})

(需記住出現順序來比較或使用 stable sort)

Depth-first search (DFS)

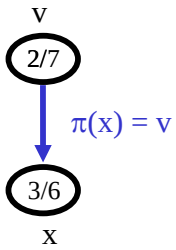


Depth-first forest

- start 和 finish time 不一定會用到，如果沒有需要可以省略 (algorithms 課程中會介紹一些很妙的用法)

Variables

- color 陣列: 記錄 vertex 是否被走過
  - WHITE: un-discovered
  - GRAY: expanding
  - BLACK: finish
- start: 記錄 vertex 的 discovery time
- finish: 記錄 vertex 的 finish time



- π: 記錄 vertex 在 depth-first forest 中的 parent (depth-first forest)

Pseudo code

```
DFS (G) {
    for each v ∈ V[G] {
        color[v] = WHITE; π(v) = NIL
    }

    time = 0

    for each v ∈ V[G]
        if (color[v] == WHITE)
            DFS_Visit(v)
}
```

找到一個起點當 root
往下 recursive 展開得到一棵 tree

Pseudo code (cont.)

```
DFS_Visit(v) {
    color(v) = GRAY
    time = time + 1;  start[v] = time

    for each u ∈ adj[v]
        if (color[u] == WHITE) {
            π(u) = v
            DFS_Visit(u)
        }

    color[v] = BLACK;
    time = time + 1; finish[v] = time
}
```

```
adjacency-matrix
    for (u = 0; u < n; u++)
        if (adj[v][u] = = 1)
```

- Adjacency matrix: $O(n^2)$ Adjacency-list: $O(n + m)$

應用

- 應用一: A.459 Graph Connectivity
- 應用二: A.10608 Friends

應用一: A. 459 Graph Connectivity

http://uva.onlinejudge.org/external/4/459.html

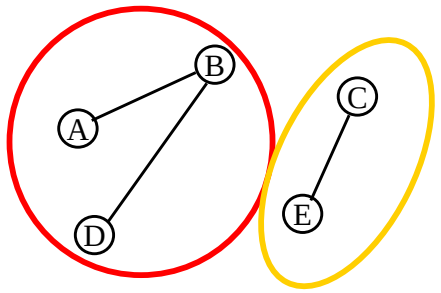
- 給一個 undirected graph，vertex 是由 A 到 Z 來編號
- 請問: 這個 undirected graph 中共有幾個 connected components

Sample input/output

Sample input

vertex 編號最大會到 E

E
AB
CE
DB



Sample output

2

共兩個 connected component

Solution

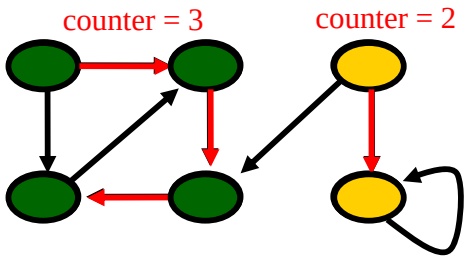
- 作 DFS，depth-first forest 中的每一棵 tree 都是一個 connected component
  - 用一個 counter 去計算 有幾棵 tree (外層)
- 因為是 undirected graph，讀進一條 edge 時要記得兩個方向的 edge 都要存
- adjacency matrix or adjacency-lists???

應用二: A.10608 Friends

- 給有 n 個人，m 個關係 (某人跟某人是否為好朋友)，互相為好朋友的人就是在同一個小團體，請問最大的小團體人數為何
  - 等同於給一個 n 個 nodes 與 m 條 edge 的 graph，問最大的 connected component 有幾個 nodes
- $n \le 30000$, $m \le 500000$

Solution

- 作 DFS, 最大的 component size 就是答案
  - 用一個 counter 去計算每一棵 tree 的 size (內層)



- adjacency matrix or adjacency-lists???

作業與自我挑戰

- 作業
  - 練習題
    - A.10608 Friends
      http://uva.onlinejudge.org/external/106/10608.html
  - 挑戰題
    - A. 315 Network
      http://uva.onlinejudge.org/external/3/315.html
- 自我挑戰
  - A.782 Contour Painting
    http://uva.onlinejudge.org/external/7/782.html
  - A.776 Monkeys in a Regular Forest
    - http://uva.onlinejudge.org/external/7/776.html
- 其它有趣的題目
  - A. 599 The Forest for the Trees
    - http://uva.onlinejudge.org/external/5/599.html