主題: All-pairs Shortest Paths

- 基礎
- 應用
- 作業與自我挑戰

基礎

- The all-pairs shortest paths problem
- Floyd-Warshall's algorithm
- Transitive closure

The all-pairs shortest paths problem

- 給一個 directed weighted graph，對所有 pair (u, v)，求 u 到 v 的 shortest path
- Input: the weighted adjacency-matrix W

$$w[i,j] = \begin{cases} 0 & \text{if } i = j; \\ \text{length of } (i,j) & \text{if edge } (i,j) \text{ exists;} \\ \infty & \text{otherwise.} \end{cases}$$

- 可以用 graph 中的每一個 vertex 當起始點來算 single-source shortest paths
  - $O(n^3)$

Floyd-Warshall's algorithm

- 注意: 此方法適用於 negative edges，但不可以有 negative cycles

```
D ← W
for k = 1 ~ n
   for i = 1 ~ n
       for j = 1 ~ n
           D[i, j] = min{D[i, j], D[i, k] + D[k, j]};
```

- **注意 ∞ 的處理**
- **$O(n^3)$**

Floyd-Warshall vs. Dijkstra

- Floyd-Warshall algorithm 雖然一樣可以做 backtracking 建出 shortest paths，但是 backtrack 的步驟比較麻煩 (這裏省略)，所以若需要建 shortest paths，建議使用 Dijkstra's algorithm n 次

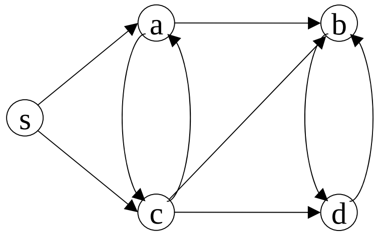
- 由於 Floyd-Warshall algorithm 極易寫，所以當 n 在 300 以下而且不需 backtrack 時，即使是 single-source shortest paths problem，也可以考慮寫 Floyd-Warshall's all-pairs algorithm

只需距離且 $n \le 300$ (???) $\Rightarrow$ Floyd-Warshall!!!

Transitive Closure

- 給一個 directed graph G 的 adjacency-matrix A，計算 G 的 *transitive closure* A^*

a graph G



a 有 edge 到 b

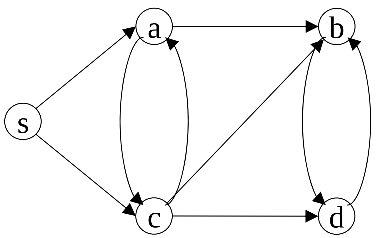
a 沒有 edge 到 d

A

	s	a	b	c	d
s	0	1	0	1	0
a	0	0	1	1	0
b	0	0	0	0	1
c	0	1	1	0	1
d	0	0	1	0	0

Transitive Closure (cont.)

- $A^*[i,j] = \begin{cases} 1 & \text{if (i = j) or (there is a path from i to j);} \\ 0 & \text{otherwise} \end{cases}$



A^*

	s	a	b	c	d
s	1	1	1	1	1
a	0	1	1	1	1
b	0	0	1	0	1
c	0	1	1	1	1
d	0	0	1	0	1

a 有 path 到 d

d 沒有 path 到 a

Method 1. Floyd-Warshall

- 將每條 edge 的 length 設為 1，沒有 edge 的部分則設為 ∞

$$w[i,j] = \begin{cases} 0 & \text{if i = j;} \\ 1 & \text{A[i, j] = 1;} \\ \infty & \text{otherwise.} \end{cases}$$

- 用這組 W 去執行 Floyd-Warshall's algorithm ，則:
  - $D[u, v] \neq \infty \Rightarrow A^*[u, v] = 1$
  - $D[u, v] = \infty \Rightarrow A^*[u, v] = 0$

Method 2. DFS

- For a directed graph G
  - A^* 可以跑 n 次 DFS 來計算
  - $O(n \times (n + m)) = O(n^3)$

每次算一個 row
(row i: 由 i 出發 DFS)

- For an undirected graph G
  - A^* 可以只跑一次 DFS 來計算
  - $O(n + m) = O(n^2)$

$A^*[i, j] = 1$ iff
i, j are in the same tree

- may be faster but slightly complicated

應用

- 應用一: A.247 Calling Circles
- 應用二: AT2003D The Geodetic Set Problem
- 應用三: AT2001E Hinge Node Problem

應用一: A.247 Calling Circles

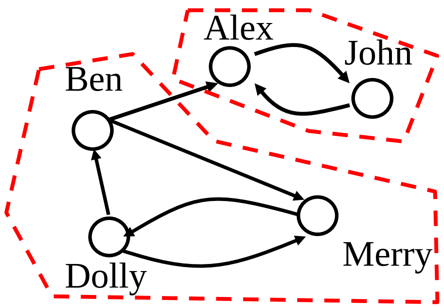
- 電話公司對 users 撥打電話有完整的紀錄，用 $A \rightarrow B$ 表示 A 曾打電話給 B
- 如果存在一條由 X 到 Y 的 call path ，同時存在一條由 Y 到 X 的 call path，則我們說 X, Y 在同一個 *calling circle* 上
- 給 users 撥打電話的完整紀錄，請將所有的 calling circles 找出來
- $n \leq 25$ users

Example

Sample input

5 users and 7 calls

5 7
Ben Alex
Merry Dolly
John Alex
Alex John
Dolly Ben
Dolly Merry
Ben Merry



共兩個 calling circles
(strongly connected components)

Strongly connected components

- Let G be a directed graph
- Let A^* be the transitive closure of G.
- Two vertices i, j are of the same *strongly connected component* iff (A^*[i, j] = 1)&(A^*[j, i] = 1)

Solution

- Step 1: Computing A^*
- Step 2: Output circles (strongly connected component)

A^*

	1	2	3	4	5
1	1	1	1	1	1
2	1	1	1	1	1
3	0	0	1	0	1
4	1	1	1	1	1
5	0	0	1	0	1

1	2	3	4	5
Ben	Merry	Alex	Dolly	John

共兩個 calling circles

Pseudo code for Step 2

```
for (i = 0; i < n; i++) flag[i]=1;        ← Initialization

for (i = 0; i < n; i++)
   if (flag[i] == 1) {    ← 找到一個尚未輸出的 vertex i
      flag[i] = 0;
      printf("%s", name[i])
      for (j = i + 1; j < n; j++)
         if ((A*[i, j] == 1)&(A*[j, i] = = 1))
                  flag[j]=0;
                  printf(", %s ", name[j])
      printf("\n");
   }
```

← 輸出 i 的 calling circle

只需由 i + 1 開始拉人

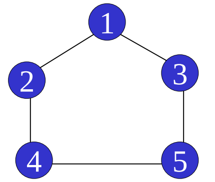
Analysis

- $O(n^3)$ /* $n \leq 25$ */
- 註: strongly connected components 用 DFS 有 O(n+m) time 的作法 (course of Algorithms)

應用二: AT2003D The Geodetic Set Problem

- 給一個 unweighted graph G ($n \le 40$)
  - 給兩個 nodes x, y ，$I(x, y) = \{z \mid x, y$ 有一條 shortest path 通過 $z\}$
  - For a vertex-set S，$I(S) = \cup_{x,y\in S} I(x, y)$
  - If I(S)=V(G), S is called a *geodetic set* (測量點集合)
- 給 G 和 ($S_1, S_2, ..., S_q$)，請判斷每一個 S_i 是否為 geodetic set

Example



- $I(2, 3) = \{1, 2, 3\}$
- $S_1 = \{3, 4, 5\} \Rightarrow$ No, since $1, 2 \notin I(\{3, 4, 5\})$
- $S_2 = \{1, 3, 4\} \Rightarrow$ Yes

Solution

- Step 1. 計算 D[i, j] (all-pairs shortest paths)
- Step 2. 暴力檢查 all v, x, y, 看 v 是否在 x-y shortest path 上

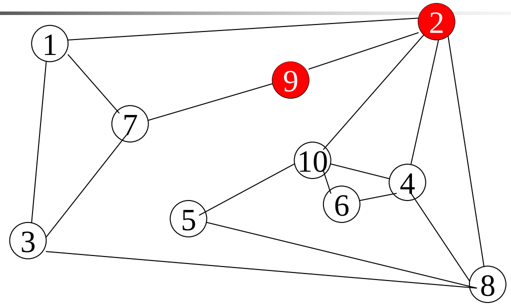
$O(n \times |S_i|^2)$
  - for each $v \in V$
    - for all $x, y \in S_i$
      - check whether $v \in I(x, y)$ $\boxed{\text{How ???}}$
- Step 3. If there is a vertex $v \notin I(x, y)$ for all $x, y \in S_i$, S_i is not a geodetic set
- $O(n \times |S_i|^2) = O(n^3)$ for check an S_i, where $n = 40$

  $\Rightarrow O(q \times n^3)$ (good for $q < 10^3$)

應用三: AT2001E Hinge Node Problem

- 給一個 graph G ($3 \le n \le 100$)
  - all edges have unit length (unweighted)
  - 給一個 node g，如果拿掉 g 後有某兩個 vertices x, y 的 shortest path 變長，則稱 g 為 *hinge node* (關鍵點)
- 找出 G 的所有 hinge nodes
- $q < 6$ 組測資

Example



- 2 is a hinge node, since $d_{G-\{2\}}(8, 9) = 3 > d_G(8, 9) = 2$
- 9 is a not a hinge node, since $d_{G-\{9\}}(x, y) = d_G(x, y)$ for all x, y

Solution 1

- Step 1. 計算 D[x, y] (all-pairs shortest paths)
- Step 2. For each $v \in V$, 拔掉 v 看有無 x, y 的 distance 增加
  - Compute $D_{G-\{v\}}[x, y]$
  - Compare D[x, y] and $D_{G-\{v\}}[x, y]$

How???

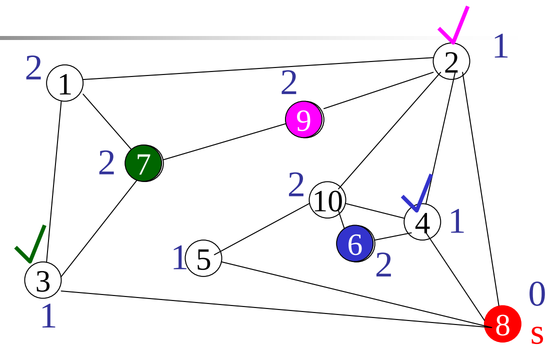
- $O(n^3)$ for checking a node v
  - $O(n \times (n + m))$ if using adjacency-lists and BFS
- $O(n^4) \approx 10^8$ for a graph (n = 100)
- $O(q \times n^4) \approx 5 \times 10^8$ in total (Pass ???)

Solution 2

- 固定一個 node s，找出所有 s-v shortest paths 上的 hinge nodes
  - Step 1. 利用 BFS (或 single-source shortest paths) 計算所有 d(s, v)
  - Step 2. for each u with d(s, u) = k $\geq$ 2
    - if u has only one neighbor g with d(s, g) = k-1, mark g as a hinge node

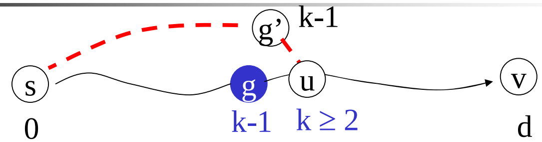
Example

- 計算所有 d(s, v)



- 6 marks 4 as a hinge node
- 7 marks 3
- 9 marks 2
- 1, 10 do not mark any node

Correctness



- 假設 g 是 s-v shortest path 上的 hinge node
- 令 u 為 g 的下一個 vertex, depth 是 k
- g 的 depth 是 k - 1
- g 是 u 的 neighbors 中，唯一 depth = k - 1 的 node
- g 必然也是 s-u shortest path 上的 hinge node

- 如何推廣到 weighted graph ???

Analysis

- $O(n^2)$ for checking a node s
- $O(n^3) \approx 10^6$ for a graph (n = 100)
- $O(q \times n^3) \approx 5 \times 10^6$ in total

Solution 3

- modify Solution 2，用 all-pairs shortest paths 取代 n 次 BFS
  - ~~Step 1. 利用 BFS 計算所有 d(s, v) (single-source)~~
    $\Rightarrow$ 計算 D[x, y] (all-pairs shortest paths)
  - Step 2. for each node s
    - 找出所有 s-v shortest paths 上的 hinge nodes
      ($\Rightarrow$ row s of D, D[s, ·], stores all shortest s-v paths)

- $O(n^3) \approx 10^6$ for a graph
- $O(q \times n^3) \approx 5 \times 10^6$ in total

作業與自我挑戰

- 作業
  - 練習題
    - A.247 Calling Circles
      http://uva.onlinejudge.org/external/2/247.html
  - 挑戰題
    - A. 10269 Adventure of Super Mario
      http://uva.onlinejudge.org/external/102/10269.html
    - A.718 Skyscraper Floors
      http://uva.onlinejudge.org/external/7/718.html
- 自我挑戰
  - A.10113 Exchange Rates
- 其它有趣題目:
  - A.334 Identifying Concurrent Events
  - A.869 Airline Comparison
  - A.273 Jack Straws
  - A.10331 The Flyover Construction