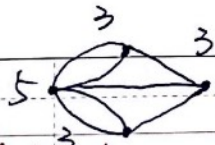


2021年6月6日

10827117 陳柏宇 資訊二甲

資料結構筆記

單元 6: 1 橋問題 \Rightarrow  degree: 有多少邊
edge: 邊

* 不可能會有單個奇數 degree \Rightarrow 不可能畫出來

$\sum_{v \in V(G)} \text{degree}(v_i) = |E(G)| \times 2 \Rightarrow$ 邊會有連接兩點 $\Rightarrow \times 2$

Adjacency List: 1. unweighted 2. weighted

在作業 4 中, adjacency list 圖形是 weighted 和有向
想法: 將資料放置於一個陣列中, 再慢慢
從此陣列拿東西, 接著創造另一陣列。每一
次拿到新資料: 1. 搜尋主陣列是否已存在
此 ID $\begin{cases} \text{存在: 在此 ID 的次陣列放入資料} \\ \text{不存在: 在主陣列放入此 ID 和它的資料} \end{cases}$

3. 排列 (在此因為檔案筆數可能很大, 所以使
用 quick sort 來增加速度) 4. 寫入檔案。

DFS: 深度優先 Depth-First Search Traversal

先走一條路徑到最底, 如果觸底了再回到
上一層, 往下一個路徑走。* 非常適合遞迴

\Rightarrow A "last visited, first explored" strategy

\Rightarrow 且要使用 stack (符合先進後出)

BFS: 廣度優先 Breadth-First Search Traversal

不再一路走到底, 要先以鄰近的為優先, 把
所有鄰近的點走過之後, 才走下一個節點。

* 比較適合迴圈, 遞迴有難度

\Rightarrow A "first visited, first explored" strategy \Rightarrow 要用 queue

單元 1. Topological Sort: 用在網路, 表示電腦 and 電腦的連接

2. 不可以是個 cycle! \Rightarrow 不然就不一定是對的 3. 有向的

ex. 遊戲, 電影, 工廠 \Rightarrow 開放式劇情

topsort 1: 1. 從沒有指出去的開始 2. 不斷從左邊放進去
3. 拿掉 edges 4. Repeat \Rightarrow 最後 graph 就會空了, 得 ans
out-degree = 0 && 有誰指向此矣 \Rightarrow 要存下來

topsort 2: 1. 從沒有指進來的開始 2. DFS 3. stack
4. 拿掉 edges 5. Repeat \Rightarrow 最後 graph 就會空了, 得 ans
in-degree = 0 && 有誰被此矣指向 \Rightarrow 要存下來

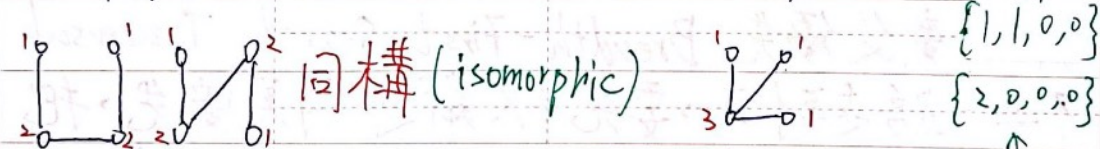
生成樹: 1. 點跟點是相連通的 2. 不可以是 cycles
3. 樹是圖的子圖, 涵蓋全部矣。

\Rightarrow 點: 電腦 / edges: 連線 \Rightarrow communication network

ex. CISCO, Spanning Tree Protocol 協定 \Rightarrow 保證每台皆可連線
相連通且不能有 cycles (再加越多越好)

特性: 在 n 個矣的圖中 \Rightarrow 必有 $n-1$ 個 edges

ex. 4 矣只可以是 3 個边 \Rightarrow 多的話有 cycle / ans 不唯一
在同數量的點和边會有不同的接法 兩種同構



$$\sum \text{degree}(V_i) = |E(G)| \times 2 \Rightarrow 3 \text{ edges} \times 2 - 4 \text{ 矣} = 2$$

$$\{1, 1, 0, 0\} \Rightarrow C(4, 2) = 12 > 16 \text{ 種}$$

$$\{2, 0, 0, 0\} \Rightarrow C(4, 1) = 4$$

Priifer sequence: 持續拿走 graph 中是葉子且 label 最小的
 然後記錄下其父節點的 label 形成一串數字 \Rightarrow 絕不會 repeat

ex. 33 \Rightarrow degree: 1111 \Rightarrow 1121 \Rightarrow 1131
 \Rightarrow 1121 = 0121 \Rightarrow 0011 \Rightarrow 0000



Minimum Spanning Tree: 找到一條路徑, 此路徑上每一邊的 weight 會是 最小的 (MST) \Rightarrow 希望點跟點之間的連結是最有效率的且成本最小的

\Rightarrow Prim's Algorithms: 1. 找到 least-cost edge 2. 標示該路徑的終點的點為 visited 3. 加入此 vertex 和 edge 4. Repeat \Rightarrow 就是每次都找最小的那一條路就好

\Rightarrow Kruskal Algorithms: 1. 一次合併兩顆樹 (只要連一個邊就可以把兩顆樹連起來, 挑 cost 最少的) 2. 合併加入最小生成樹 3. Repeat (P.S. 每個節點都看成是一顆樹 and 給編號)

\Rightarrow 連接起來的兩顆樹 \Rightarrow 一顆樹

\Rightarrow Sollin Algorithm: (Kruskal 改進版) 多了一個 DS 記錄已算出來的短邊 (P.S. 有對象是分成兩邊去進行 MST 然後再合併這兩邊的樹)

Dijkstra's Algorithm: 只要找到一條最短路徑, 那前面的也高 屏 鍊 都會是最短的

A \rightarrow B \rightarrow C \leftarrow (需要兩個 DS 來存資料)

ABC 是最短 \Rightarrow AB 也會是最短的

All-Pairs Shortest Paths: 要去記錄然後使用曾經算過的結果, 來去避免重覆計算。 (需要一個大矩陣記點和點之間的關係 \Rightarrow 相鄰矩陣) 1. 要看矩陣的行列 2. 然後跟其他行列連結 (加兩條的 cost)

\rightarrow (第 n 行, 第 n 列, 就是找以 n 起的路)

Critical Path: 可以讓我們知道在整圖當中, 那些路徑、活動是比較重要的。(最一開始IBM用此方法買電腦)

1. 算出最後一個事件的最晚時間 2. 往回推到每一個桌上

P.S. 如果遇到有分支 \Rightarrow 取比較少的那一個, 如此每條皆滿足

3. 算從頭(左 \rightarrow 右)的最晚時間 4. 相減 $(la - ea)$ la : 右 \rightarrow 左 ea : 左 \rightarrow 右

5. 把所有相減後是0的記下來 (0的代表沒有任何彈性)

Ford-Fulkerson algorithm: 1. 存成一資料結構(DS) 2. 照矩陣順序去找路徑(必須找 >0 的edge) 3. 找到後某路徑最小的, 對此路徑上每一個辺都減該數字。4. 最後找到所有的反向路徑, 終點相加 \Rightarrow max flow。

(反向路徑 \Rightarrow 將反向路徑的數字皆設定為該正向路徑中最小數字)

Edmonds-Karp algorithm (跟Ford-Fulkerson相差是在於有選路):

1. 每一次都選最大的那一條路 2. 其餘跟Ford-Fulkerson一樣

Eulerian Circuit (Euler Tour): 利用DFS走訪 \Rightarrow 找出一筆畫

TSP on the Web:

Brute-force algorithm: 1234567 \rightarrow 1234576 \rightarrow ... \rightarrow 1345... \rightarrow ...

程式好寫但速度慢

Greedy algorithm: 速度快但ans比較差 / 任選一個點做出發

Branch-and-bound: Brute-force 和 Greedy 折衷 (不一定是最佳答案)

Bi-connected Graph: 1. DFS走訪 2. 記流水號(從流水號判斷是不是孤島) (連接孤島的就是樞紐)

單元九:

External sort: 在這一次的作業五中, 使用了 external sort, 覺得很厲害, 可以在有硬體限制的情況下做出更多的可能性。

1. 首先將一個大筆資料的檔案分成一個個小筆資料的檔案。(且在分成小檔案前要先做 internal sort, 這樣 external sort 才會是正確的) 2. 而在第一次分好檔案後, 最重要的 external sort 就來了, 會使有一部份的 mergesort, 也就是取其合併的部分。(作業有限制不可一次把所有資料讀進記憶體) 所以在合併時, 只取兩個檔案各自一筆資料進來, 然後做比大小, 來決定放哪一個檔案進來的資料到新的檔案中。把所有資料放好後, 就要讀下兩個檔案的資料, 一路做到最後只剩一個檔案, external sort 便完成了!

第一次寫程式不同的硬體區塊在搭配運做, 真的很特別, 以往都是無腦全部讀進來做排序, 如此一來需要大量的處理空間和效能, 畢竟未來在工作時, 隨便一個檔案都是現在的資料筆數的好幾倍。如果能將此技能學好的話, 便可在有限中創造無限。

作業4補充: 承蒙老師在機測時的講解, 讓我了解到加速的方法不是只有排列加速和少做迴圈, 而是要多加利用自己前面算出來的答案來做搭配, 如果走到一個學号是之前已經做過處理的發訊者, 那我可以直接把這個結果拿來使用, 如此一來, 隨著處理完的資料越多, 程式就可以越跑越快。