R05546030 **彭新翔**

一、簡介

瑞士數學家 Leonhard Euler 為了解決著名的七橋問題(Seven Bridges of Königsberg)，在 1735 年在所發表的論文《柯尼斯堡的七橋》中將此問題轉化為實質的一筆畫問題（一張圖是否能夠遍歷完所有的邊而沒有重複），此為圖論(Graph Theory)的濫觴。在電腦被發明後，圖論與演算法的結合，被廣泛地被應用於求解最佳化問題與存在性問題，比如通訊網路間的最短路徑問題、班機航班調配問題、快遞轉運站配置問題…等。

二、問題定義

此篇論文所要處理的問題為：

給定一棵擁有 n 個節點的樹(tree) $T=(V,E)$，其中每一條線段(edge) (i,j) 具有非負的長度 d_{ij} 且每個節點(vertex) i 都被賦予一個非負的權重值 w_i，定義線段 (i,j) 上存在一點(point)距離 i 節點 t 單位長度（也就是說距離 j 節點 $d_{ij}-t$ 單位長度），並記為 $x=(i,j:t)$。

對於此棵樹上，可以找到一個加權中心(weighted center)使下列結果最小：

$$r(x)=\max\{w_i d(x,i):i\in V\}$$

另外一個相關的問題可以將點限定為僅可能為節點的狀況，亦即：

$$r(x)=\max\{w_i d(x,i):i,x\in V\}$$

為了更清楚地理解問題與解法敘述，在此回顧圖論中關於樹(tree)的內容，並給出一些定義與符號：

A. 樹 (tree)

一種常被用以描述具有階層結構(hierarchical structure)問題的結構，可以理解成是將連接串列(linked list)推廣至多維度狀況的結構，主要由多個不形成迴路(cycle)的節點(vertex)和線段(edge)組成，記為 $T(V,E)$。

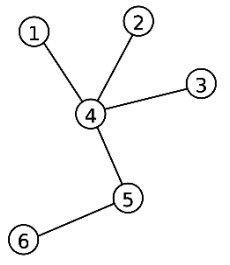

Figure 1 樹(Tree)

其中：

- **節點(vertex)**：圖形中的點，定義所有節點所形成的集合記為 V
- **線段(edge)**：圖形中連結兩點的連線，並沒有標記方向者，以一對節點表示，並定義所有線段所形成的集合記為 E

B. 路徑(path)、簡單路徑(simple path)

- **路徑(path)**：從一節點至另一節點間可能存在一連串前後相接的線段，將這些串接的線段稱為路徑
- **簡單路徑(simple path)**：若一條路徑之中，除了起始節點與結束節點之外，沒有其餘的節點被重複經過，稱為簡單路徑

R05546030 **彭新翔**

C. 相鄰(adjacent)、子樹(subtree)、根(root)、葉(leaf)

- **相鄰(adjacent)**：若從一節點至另一節點間存在一條線段，稱此兩個節點互為相鄰
- **子樹(subtree)**：一棵樹的節點可以切割成任意整數個互斥(disjoint)的集合 $T_1, T_2, \cdots, T_n$，其中每一個集合若亦滿足樹的定義，則稱這些集合為這棵樹的子樹
- **根(root)**：一棵樹中處於最上層的節點，稱該節點唯這棵樹的根
- **葉(leaf)**：對於沒有無法生成子樹的節點，稱該節點為這棵樹的葉

D. 長度(length)、點(point)、權重(weight)

對於論文中所要求解的問題，必須定義以下名詞，值得一提的是此處對於權重的定義可能與一般教科書中有所不同：

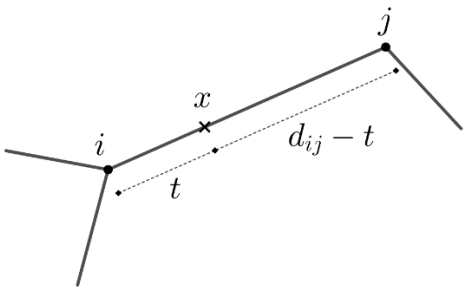

Figure 2 Length and Point

- **長度(length)**：對於任意兩節點 i 和 j 所形成的線段 (i, d) 賦予了非負長度 d_{ij}
- **點(point)**：對於任意兩節點 i 和 j 所形成的線段 (i, d) 上，可以找到一點 x 與 i 節點的距離為 t 而與 j 節點的距離為 $d_{ij} - t$
- **權重(weight)**：與一般常見地對線段加權有所不同，此處加權是針對任意節點 i 給予一個非負的加權值 w_i

結合此處定義的名詞與上述相鄰(adjacent)、簡單路徑(simple path)和子樹(subtree)的概念，我們將點 x 經簡單路徑到 i 節點上所會經過的節點 j 所形成的集合記做 $V_j(x)$；而將 $V_j(x) \cup \{x\}$ 所生成的子樹記做 $T_j(x)$

三、解法敘述

顯而易見地，此問題中所要求解的 $r(x)$ 在考慮簡單路徑的狀況下為一凸函數(convex function)。也就是說針對一點 x 若為 $r(x) = \max\{r_{j1}(x), r_{j2}(x), \cdots, r_{jl}(x)\}$ 的一個局部極值，根據凸函數的性質可以知道此解即為所求的加權中心點。以此概念出發，論文作者以 Prune and Search 的思想提出了以下的演算法：

R05546030 **彭新翔**

假設節點 c 為給定樹 $T(V,E)$ 的重心（注意此處並非所要求解的加權中心），亦即對於所有與 c 相鄰的節點 j 而言滿足 $|V_j(c)| \leq \frac{n}{2}$，並且可以於 $O(n)$ 時間內找出[1]，以下敘述過程中皆假定重心 c 為已知：

Algorithm 1: CENTROID(T)

Input: A tree T is given in a read-only memory
Output: Report the centroid of the given tree

```
1 t = root(T); t' = ∅; Size = 0;
2 do
3 |   (m, δ, TMsize) = FIND-MAXIMUM-SUBTREE(t, t', Size);
4 |   if TMsize > ⌊n/2⌋ then
5 |   |   t' = t; t = m; Size = Size + δ ;
6 while TMsize ≤ ⌊n/2⌋;
7 Return t;
```

Figure 3 Algorithm to find the Centroid of a Tree

A. 終止條件或判斷加權中心位置

已知重心 c 並計算所有每一相鄰節點 j 的 $r_j(c) = \max\{w_i d(c,i)\}$，此部分可以於 $O(n)$ 時間內完成，並作以下討論：

(1) 運算終止

若存在兩相鄰節點 j_1 和 j_2（其中 $j_1 \neq j_2$）滿足：

$$r_{j1}(c) = r_{j2}(c) = r(c)$$

則此重心 c 即為所求加權中心。

(2) 判斷加權中心位置

若存在相鄰節點 j 對於其他所有相鄰節點 k 而言，滿足：

$$r_j(c) > r_k(c)$$

則可知所求加權重心會位於子樹 $T_j(c)$ 上

B. 刪除(prune)不必要的節點

對於任意兩節點 $u, v \notin V_j(c)$ 和與重心 c 距離 t 的點 $x \in T_j(c)$ 而言，則此兩點與重心的距離即為 $d(u,x) = d(u,c) + t$ 和 $d(v,x) = d(v,c) + t$，則在求解方程式

$$w_u(d(u,c) + t) = w_v(d(v,c) + t)$$

的過程中，不失一般性的狀況下可假設 $w_u d(u,c) \geq w_v d(v,c)$ 若且唯若存在一非負的 t_{uv} 滿足 $0 \leq t \leq t_{uv}$，其中：

- 若加權中心與重心位置小於 t_{uv}，則在尋找加權中心的過程中可以不用考慮節點 v
- 若加權中心與重心位置大於 t_{uv}，則在尋找加權中心的過程中可以不用考慮節點 u

C. 判斷加權中心與葉的距離

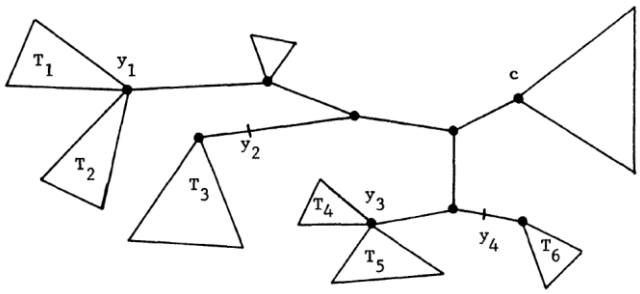

Figure 4 Subtree of all z without roots

[1] Gabriel Y. Handler And Pitu B. Mirchandani, Location on Networks: Theory and Algorithms.

R05546030 **彭新翔**

如上圖所示，給定樹上的一葉 x 和一非負距離 t，我們可以在 $O(n)$ 的時間內找到所有滿足 $d(x, y_v) = t$ 的點 $y_1, y_2, \cdots, y_l$，而對於滿足 $d(x, z) \geq t$ 的所有點 z 所形成的集合則可以表示為以 $y_1, y_2, \cdots, y_l$ 作為根的子樹 $T_1, T_2, \cdots, T_m$，取 V_i 為 T_i 中去掉其根 u_i 的所有點所形成的集合，並定義：

$$R_i(x) = \max\{w_k d(x, k) : k \in V_i\}$$

根據上述定義可以知道 V_i 彼此互斥，因此可在 $O(n)$ 的時間內計算所有 $R_i(u_i)$ 後令 $R = R_i(x) = \max\{R_i(u_i)\}$ 並作以下討論：

(1) $R_{i1}(u_{i1}) = R_{i2}(u_{i2}) = R$：加權中心並不在所有的子樹 $T_i (i = 1, 2, \cdots, m)$ 中
(2) 存在唯一的$R_i(u_i) = R$：加權中心位於此個子樹 T_i 中
(3) $R_i(u_i) < R$：加權中心並不在此個子樹 T_i 中

D. **總結**

最後將上述內容彙整，針對給定的樹 $T(V, E)$ 而言，求解其加權重心的具體步驟如下：

(1) 於 $O(n)$ 的時間內找出重心 c（即與之相鄰的節點 j 皆滿足 $|V_j(c)| \leq \frac{n}{2}$）
(2) 若其中 $r_{j1}(c) = r_{j2}(c) = r(c)$ 則終止運算，否則根據 $r_j(c) > r_k(c)$ 決定子樹 $T_j(c)$並進入以下的遞迴形式
(3) 將不屬於子樹 $T_j(c)$ 的所有節點兩兩配對成 $(u_1, v_1), (u_2, v_2), \cdots, (u_s, v_s)$，則至少會有 $s = \lfloor \frac{n}{4} \rfloor - 1$ 對節點
(4) 對於每一對節點，考慮 $w_u(d(u, c) + t) = w_v(d(v, c) + t)$ 是否刪除其中一節點，或是計算：

$$t_{uv} = \frac{w_u d(u, c) - w_v d(v, c)}{w_v - w_u}$$

(5) 於 $O(n)$ 的時間內找出所有 t_{uv} 值的中位數 t_m
(6) 比較 t_{uv} 與 t_m 大小，並由 $w_u d(u, x^*)$ 和 $w_v d(v, x^*)$ 大小關係決定刪除哪一節點

上述的演算法即蘊含了 Prune and Search 的概念，首先在尋找到重心並判斷加權中心位於哪一子樹後，節點數量可以減半至 $\frac{n}{2}$ 個；再將不屬於子樹中的節點兩兩配對後，可知至少共有 $\lfloor \frac{n}{4} \rfloor - 1$ 對節點；而每一對節點又可以根據判斷刪去其中一點，因此每次的遞迴過程中約有 $\frac{n}{8}$ 個節點被刪除。此外論問中亦有附註表示此題可以將欲求的加權中心限制於節點上，則為其離散形式，就概念上而言並無差異，簡單來說只要求出上述演算法所求的連續解：若所得加權中心位於節點上，則離散解與連續解相等；若所得加權中心位於一條線段上，則此線段兩端點即為離散解。

則此算法的時間複雜度：

$$T(n) \leq T(\tfrac{7n}{8}) + Cn \text{ (where c is big-O constant)}$$

$$T(n) = O(n)$$

R05546030 **彭新翔**

四、閱讀心得

刪尋法(Prune and Search)被視為分治法(Divide and Conquer)的一種特例，常被用於求解最佳化問題。當初期中考試中的第二小題 2-Dimensional Linear Programming 就是採用了 Prune and Search 的方式將原本的線性規劃問題進行簡化，免去了最常見的將係數存為矩陣後進行高斯消去法的繁複計算，在撰寫與查找相關資料的過程中其實就有查閱到這篇由 IBM 研究人員 Nimrod Megiddo 所寫的論文，其中使用了 Prune and Search 的概念求解了多則問題：平面上的線性規劃(Linear Programming in the Plane)、樹狀結構的節點加權中心(The Weighted Center of Tree)、最小覆蓋圓(Smallest Circle enclosing n Points)與三維空間的線性規劃(Linear Programming in $\mathbb{R}^3$)，其中又以最小覆蓋圓問題最為人所熟知。在這次作業的過程中也發現了其實整體演算法的大略內容其實十分相近，皆是藉由配對後找出中位數，並依條件刪去掉不需要納入考慮的限制式或者是節點。

雖然目前遇到的許多問題（不論是作業、考試或是一些 Online Judge 上的題目）其實能夠實際應用到 Prune and Search 的機會並不多，常見的僅有：凸包問題(Convex-Hull Problem)[i]、八皇后問題(Eight-Queen Problem)…等。但其實單就目前所學習到的方法與問題便可以應用到其他層面上了。

舉例而言，以這次作業所必須閱讀的部分：求解樹狀結構的節點加權中心(The Weighted Center of Tree)來說便可以應用到多數的問題上面去，可以被視作是一個區位與分配問題(Location-Allocation Problem, LAP)的簡化版本，在現實中有著諸多應用。比如超商的物流通路、有線電視的纜線鋪設、基地台的設立位置…等，在多數的狀況下這類問題擁有一個以上的聯絡站，因此早已被證明屬於 NP-Hard 問題而無法快速得到答案，但倘若化簡為多個不同重要度的卸貨點（節點上具備加權值），其中不同節點兩兩間有固定距離（給定線段長度），欲在圖中設立一個中樞轉運倉儲問題，採用文章中的方法可以有效地得到一個近似解。

五、參考資料

[1] 初學者寫給初學者的演算法教學 – Tree: Intro, http://alrightchiu.github.io/SecondRound/treeshu-introjian-jie.html
[2] 初學者寫給初學者的演算法教學 – Graph: Intro, http://alrightchiu.github.io/SecondRound/graph-introjian-jie.html
[3] 臺大數學系, 張鎮華, 蔡牧村 – 圖論及其演算法, http://www.math.ntu.edu.tw/~gjchang/courses/2008-02-graph-theory-II/graph.pdf
[4] 演算法筆記 – Graph, http://www.csie.ntnu.edu.tw/~u91029/Graph.html
[5] 演算法筆記 – Tree, http://www.csie.ntnu.edu.tw/~u91029/Tree.html
[6] 2014, Binay K. Bhattacharya, Minati De, Subhas C. Nandy, Sasanka Roy – Facility location problems in the constant work-space read-only memory model, https://arxiv.org/pdf/1409.4092.pdf
[7] 2007, Frank Dehne - Algorithms and Data Structures: 10th International Workshop

[i] Which can be solved with Kirkpatrick-Seidel's Prune-and-Search Convex Hull Algorithm