

演算法設計方法論 (Design Strategies for Computer Algorithms)

Homework 2

DUE DATE: NOVEMBER 27, 2017

學號: b03902129 系級: 資工四 姓名: 陳鵬宇

1 問題定義

問題: *The weighted center of a tree.*

輸入: 一棵擁有 n 個點的樹 $T = (V, E)$

- $d_{ij} = \text{len}(\text{edge}(i, j)) \geq 0, \forall \text{ edge}(i, j)$
- $w_i \geq 0, \forall \text{ vertex } i$
- 我們可以連續的表示樹上的每一個點, 舉例來說: 點 $x = (i, j; t)$ 代表 x 與點 i 的距離為 t ; 與點 j 的距離為 $d_{ij} - t$

輸出:

- 連續點解: 重心 c , 使得 $r(x) = \max(w_i d(x, i) : i \in V)$ 為最小
- 離散點解: vertex j , 使得 $r(x) = \max(w_i d(x, i) : i \in V)$ 為最小

2 解法敘述

2.1 Top-down 觀點

令 c 為 T 之重心, 對所有鄰近 c 的 j (以下用 $\text{adj}(c)$ 表示之, 為集合的概念), $|V_j(c)| \leq n/2$ (n 為 T 中點的數量), 由 [HM] 我們知道 c 可以透過遍歷所有的點, 在 $O(n)$ 的時間內被找出來。

首先, 我們對所有 $j \in \text{adj}(c)$ 計算 $r_j(c)$, 這同時也會計算所有的 $d(c, i) [i \in \text{adj}(c)]$ 。

- 終止條件: 當有兩點 $j_1, j_2 (j_1 \neq j_2)$ 且 $j_1, j_2 \in \text{adj}(c)$, 則: $r_{j_1}(c) = r_{j_2}(c) = r(c)$. 那麼重心 c 即為 *center*, 得解。
- 假設 $j \in \text{adj}(c)$, 且:

$$r_j(c) > r_k(c), \forall k \in \text{adj}(c) \text{ 且 } k \neq j.$$

那麼 $\text{center} \in T_j(c)$ 。

2.2 數學式推導

當 $u \notin V_j(c)$ 、 $x \in T_j(c)$ 、 $d(c, x) = t$ 時:

$$d(u, x) = d(u, c) + d(c, x) = d(u, c) + t. \quad (1)$$

如果 $u, v \notin V_j(c)$, 透過計算

$$w_u(d(u, c) + t) = w_v(d(v, c) + t), \quad (2)$$

不失一般性我們可以假設:

$$w_u d(u, c) \geq w_v d(v, c). \quad (3)$$

$\exists t_{uv} (0 \leq t_{uv} \leq \infty)$ 使得:

$$w_u d(u, x) \geq w_v d(v, x), \forall x \in T_j(c), d(x, c) = t \Leftrightarrow 0 \leq t \leq t_{uv}. \quad (4)$$

從以上的數學式我們可以知道:

1. $d(\text{center}, c) < t_{uv}$: 刪除點 v (prune)
2. $d(\text{center}, c) > t_{uv}$: 刪除點 u (prune)

2.3 解法發想

給定 leaf x 和 t ($t > 0, t \in \mathcal{R}$), 我們可以在 $O(n)$ 的時間內找到點 y_1, \dots, y_l 使得 $d(x, y_v) = t, v = 1, \dots, l$.

1. 計算所有的 $d(x, i)$ ($i \in T$)

2. $\forall \text{ edge}(i, j)$, 如果 $d(x, i) \leq t \leq d(x, j)$, 那麼點 $y_v \in \text{edge}(i, j)$,

$$d(y_v, i) = t - d(x, i) \Rightarrow d(x, y_v) = t$$

3. $\forall z, d(x, z) \geq t$ 可以被表示成根為 y_1, \dots, y_l 的子樹 (每一個點 y 可能貢獻 ≥ 2 子樹)

4. 令這些子樹為 T_1, \dots, T_m , 根為 u_1, \dots, u_m ($\{u_1, \dots, u_m\} \subseteq \{y_1, \dots, y_l\}$)

5. 令 V_i 為 T_i 中所有點形成的集合, 但不包含 $\{u_i\}$, 即

$$V_i = T_i - \{u_i\}.$$

6. 定義:

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

7. $\because V_i$ 兩兩不相交, \therefore 我們可在 $O(n)$ 時間內求出所有的 $R_i(u_i), \forall i = 1, \dots, m$, 分成以下三種情況討論:

(a) $R_i(u_i) < R$: $c \notin T_i$.

(b) $R_{i_1}(u_{i_1}) = R_{i_2}(u_{i_2}) = R, i_1 \neq i_2$: $c \notin T_i, \forall i = 1, \dots, m$.

(c) $R_i(u_i) = R$ 且 $i (1 \leq i \leq m)$ 是唯一的: $c \in T_i$

情況 (c) 等價於計算 $r_j(y)$, 其中 $j \in \text{adj}(u_i)$ 且 $y = u_i$, 這些步驟一樣可以在 $O(n)$ 的時間算出。

總括以上, 可以在 $O(n)$ 的時間內得知

$$d(c, x) \leq t \text{ 或}$$

$$d(c, x) > t$$

2.4 解法流程

原問題中, 我們有樹 T 、重心 c 和子樹 $T_j(c)$ ($c \in T_j(c)$)。

重新排列 vertices $\notin T_j(c)$, 即:

$$(u_1, v_1), (u_2, v_2), \dots, (u_s, v_s)$$

$\because |v \in T_j(c)| \leq n/2, \therefore s > \lfloor n/4 \rfloor - 1$ (s 代表有 $|s|$ 對 pair), $\forall \text{ pair}(u, v)$, 考慮 2.2(2):

$$w_u(d(u, c) + t) = w_v(d(v, c) + t), \quad (2)$$

我們一樣可得 2.2(3):

$$w_u d(u, c) \geq w_v d(v, c). \quad (3)$$

• $w_u \geq w_v$: 刪除點 v (prune)。

• 否則: $t_{uv} = (w_u(d(u, c) - w_v d(v, c)) / (w_v - w_u))$.

步驟如下:

1. 計算所有的 $t_{u_i v_i}$, 直到不再有點會被 prune。

2. 尋找 $t_{u_i v_i}, \forall i$ 的中位數, 這可以在 $O(n)$ 的時間內被求出, 以實作上來看可以參考 c++ 的 `std::nth_element`。

3. 計算是否 $d(\text{center}, c) < t_m$ ($\text{center} \in T_j(c)$)

4. 如果有一對 pair(u, v), 使得 $t_{uv} \geq t_m$, 可以由:

$$w_u d(u, x^*) \geq w_v d(v, x^*)$$

來決定要刪除點 u 或是點 v 。

2.5 時間複雜度

由 prune and search 的過程，每一次 prune 會有大約一半的 pair 中的一個點被刪除，也就是約有 $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2}n = \frac{1}{8}n$ 個點在樹 T 中被刪除掉，這產生了一個遞迴的新問題，考慮 T' (T' 中點的數量大約是 T 的 $7/8$ 倍)，每次 prune 的過程我們假設花費 $C(n)$ 的時間，可得以下遞迴式：

$$T(n) \leq T\left(\frac{7n}{8}\right) + Cn,$$

由 Master Theorem，我們知道 $T(n) = O(n)$ 。

2.6 後記：離散版本

點非連續的，即： $c' \in V$ (c : 連續解的重心， c' : 離散解的重心)，由 $r(x)$ 的凸性我們知道離散的解：

1. $c \in V$: $c' = c$
2. $c \in \text{edge}(i, j)$
 - $c' = i$
 - $c' = j$

3 閱讀心得

因為用到了 prune and search 的技巧，使得每一個步驟都能在線性的時間內被完成 ($O(n)$)，是一個效率很高的演算法。

The weighted center of a tree 的應用層面很廣，例如當貨運公司，要決定自己的倉庫要放在一個國家中哪個位置，可以由每個可能被運送的城市 (vertex) 的消費強度 ($w_i, \forall \text{vertex } i$) 以及每個城市彼此間的交通費 ($\text{edge}(i, j)$)，由此計算出倉庫位置 (重心 c)。

當每個城市的消費強度相當時，題目會被簡化 (reduce) 成最小生成樹 (*minimum spanning tree*, MST)，當然也可以在線性的時間內算出倉庫位置。