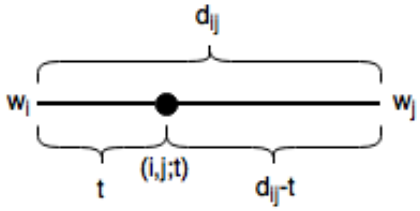
The weighted center of a tree

問題定義

首先定義一棵樹的 weighted center。假設一棵樹 $T = (V, E)$ 有 n 個頂點，每條邊 (i,j) 有一個非負的長度 d_{ij}，每個頂點 i 有一個非負的權重 w_i，(i,j) 邊上可以定義一個點 $(i,j;t)$，代表距離 i 的長度為 t，距離 j 的長度為 $d_{ij} - t$（如下圖）。



則 weighted center 定義為 T 上的一個點 x^*，使得所有頂點到 x^* 的距離乘上權重之最大值為最小的，數學表示如下：$\text{minimize } r(x) = \max\{w_i d(x,i) | i \in V\}$。以下簡稱 weighted center 為 center，或以 x^* 表示。

解法敘述

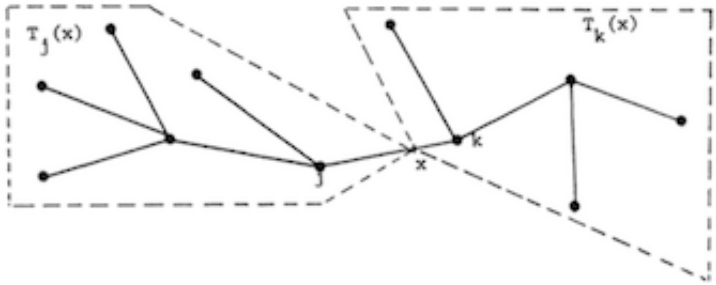
首先，有三個重要的觀察：

1. $r(x)$ 為 piecewise linear 且為 convex，因為 $r(x)$ 是一群 convex functions 的最大值。Convex function 的特性在於，如果 x^* 是 local minimum，則它也是 global minimum。所以在以下的觀察也會利用到這個性質。

2. 在 T 上選定一個點 x，假設與 x 相鄰的頂點有 l 個，分別是 $j_1, j_2, \ldots, j_l$，則我們可以把一棵樹切成 l 棵 subtrees，切法如下：

    - 把所有的頂點分成 l 群 $V_1(x), V_2(x), \ldots, V_l(x)$，其中 $V_n(x)$ for $n = 1, 2, \ldots, l$ 包含所有的頂點，使得 j_n 落在任一頂點 $i \in V_n(x)$ 到 x 的 simple path 上。
    - 所有的 subtrees 分別為 $T_1(x), T_2(x), \ldots, T_l(x)$，其中 $T_n(x)$ for $n = 1, 2, \ldots, l$ 為 $V_n(x) \cup \{x\}$ 所形成的 spanning subtree。

    下圖所示，為 x 剛好落在一條邊的內部，則 x 一定只有兩個相鄰的頂點 j、k。



定義 $r_j(x) = \max\{w_i d(x,i) | i \in V_j(x)\}$，則原本的 $r(x)$ 可以表示成 $r(x) = \max\{r_{j_1}(x), r_{j_2}(x), \ldots, r_{j_l}(x)\}$，跟原本的公式只差在把 V 分成 l 群。考慮以下兩種情形：

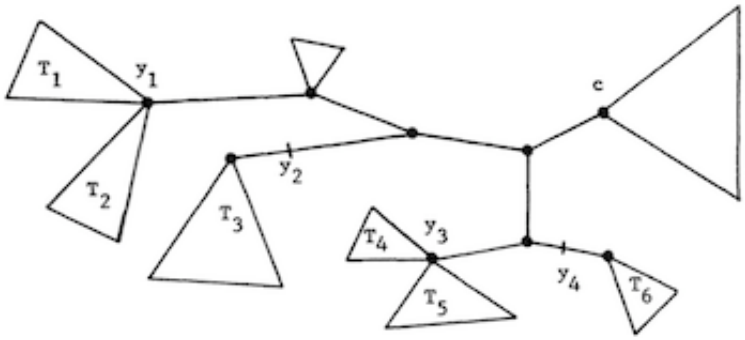
- 若最大值的 index 不只一個，則 x 必然是 T 的 center。為什麼？因為不管 x 往哪個方向移動，都會使得其中一個 $r_j(x)$ 增加，則所得到的 $r(x)$ 就不是最小值了，故 x 是 local minimum，也是 global minimum。
- 若最大值的 index 只有一個，則 T 的 center 必然落在那個 subtree。為什麼？因為如果我們把 x 往那個 subtree 移動，則可以使得目前最大的 $r_j(x)$ 減少，那麼又可以得到更小的 $r(x)$。

結論：給定一個點 x，我們可以在 $O(n)$ 的時間內決定 x^* 落在哪一個 subtree。

3. 給定一個點 x 和一個正實數 t，我們可以在 linear-time 找出所有的點 $y_1, y_2, \ldots, y_l$ 使得 $d(x, y_v) = t$ for $v = 1, 2, \ldots, l$。步驟如下：

   - 計算所有的 $d(x,i)$ for $i \in V$。
   - 對於所有的邊 (i,j)，如果 $d(x,i) \le t \le d(x,j)$，則 (i,j) 必然包含一個唯一的 y_v 使得 $d(x, y_v) = t$。

   所有滿足 $d(x,z) \ge t$ 之點 z 的集合就是以 $y_1, y_2, \ldots, y_l$ 為 root 之 subtrees 的聯集（union）。令這些 subtrees 是 $T_1, T_2, \ldots, T_m$，並分別以 $u_1, u_2, \ldots, u_m$ 為 root，並且令 V_i 是 T_i 內除了 u_i 所有頂點的集合。顯然地，V_i 彼此不相交（disjoint）。如下圖所示：



   定義 $R_i(x) = \max\{w_k d(x,k) | k \in V_i\}$，令 $R = \max\{R_i(u_i) | i = 1, 2, \ldots, m\}$。考慮 $R_i(u_i)$ 跟 R 的關係：

   - 如果 $R_i(u_i) < R$，則 x^* 不會落在 T_i。
   - 如果存在 $i_1 \ne i_2$ 使得 $R_{i_1}(u_{i_1}) = R_{i_2}(u_{i_2}) = R$，則 x^* 不會落在 $T_1, T_2, \ldots, T_m$ 中的任一。
   - 否則有一個唯一的 u_i 使得 $R_i(u_i) = R$，則 x^* 有可能落在 T_i。要如何判斷是否落在 T_i？那就要用到觀察 2 的方法，計算 $r(u_i)$ 來決定 x^* 究竟是落在哪邊。

   結論：給定一個點 x 和一個正實數 t，我們可以在 $O(n)$ 的時間內檢查 x^* 是否落在距離 x 長度為 t 的範圍內。

接著，就是敘述解決此問題的 linear-time 演算法，包含了以下步驟：

1. 找出 T 的 centroid c。
2. 假設 $j_1, j_2, \ldots, j_l$ 為與 c 相鄰的頂點，計算 $r_{j_1}(c), r_{j_2}(c), \ldots, r_{j_l}(c)$：

- 如果有任兩個與 c 相鄰的頂點 j_1, j_2 使得 $r_{j_1}(c) = r_{j_2}(c) = r(c)$，則 c 就是 x^*。
- 否則最大值是唯一的。假設最大值是 $r_j(c)$，繼續執行以下步驟。

3. 將 $T_j(c)$ 以外所有的頂點，兩兩一組配對形成 $(u_1, v_1), (u_2, v_2), \ldots, (u_s, v_s)$（最多只會有一個頂點落單），而且最少會有 $\lfloor n/4 \rfloor - 1$ 組配對，因為 $T_j(c)$ 最多只有 $n/2$ 個頂點。對於每組配對 (u, v)，假設 $w_u d(u, c) \geq w_v d(v, c)$，考慮 $w_u(d(u, c) + t_{uv}) = w_v(d(v, c) + t_{uv})$ 這個等式：
   - 如果 $w_u \geq w_v$，則可以直接把頂點 v 捨棄。
   - 否則將等式解出來 $t_{uv} = (w_u d(u, c) - w_v d(v, c))/(w_v - w_u)$。
4. 找出所有 t_{uv} 的中間值 t_m。
5. 檢查 x^* 在 $T_j(c)$ 內是否落在距離 c 長度為 t_m 的範圍內（根據觀察 3）。
6. 考慮 x^* 落在距離 c 長度為 t_m 的範圍內或外兩種情形：
   - 範圍內：對於每組配對 (u, v)，如果 $t_{uv} \geq t_m$，則可以把 v 捨棄。
   - 範圍外：對於每組配對 (u, v)，如果 $t_{uv} \leq t_m$，則可以把 u 捨棄。
7. 重複執行 1-6 直到滿足 x^* 的條件出現。

分析此演算法的時間複雜度：對於每一輪，我們可以捨棄至少約 $n/8$ 的頂點，根據步驟 3 和步驟 6，我們可以保證在至少 $\lfloor n/4 \rfloor - 1$ 組的配對中，(u, v) 的其中之一可以被捨棄，而步驟 1-6 所需要的時間複雜度都是 $O(n)$，故此演算法的時間複雜度 $T(n) = T(7n/8) + O(n)$，故 $T(n) = O(n)$，是一個 linear-time 演算法。

讀後心得

在上完課並讀完本篇論文，體會到 prune and search 最重要的是如何將多餘的資訊捨棄，是一個去蕪存菁的過程。以本演算法為例，許多頂點即使被捨棄也不會影響最後 weighted center 的計算，每一個 iteration 都可以使得 search space 不斷減少，直到得出最後的答案。我認為如何去蕪存菁是相當困難的，包括課堂中已經提過的 2D 線性規劃，以及最小圓，都以神乎其技的方式將 input size 不斷縮減。歸納去蕪存菁的過程，不外乎將 input data 配對，然後在每組配對中，將其中不可能影響答案的一個 data 捨棄。其中又有一個重要的步驟，就是尋找中間值，中間值的重要性在於將 input 一分為二，為後面的 prune 埋下伏筆，來確保可以捨棄最多的 input。而有趣的是，尋找中間值的演算法過程又是 prune and search，由此可見 prune and search 對於許多問題的重要性，如果能夠看透一個問題之中不重要的 input 將之捨棄，時間複雜度必然可以得到大幅改善，所得到的演算法也會令人相當驚艷。