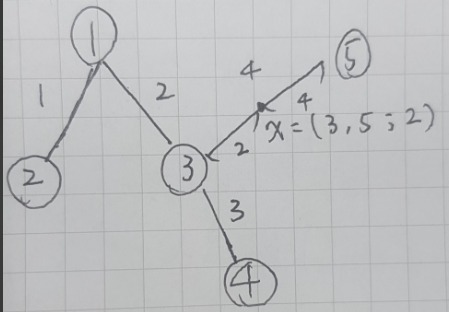
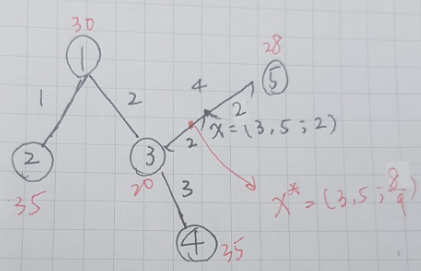
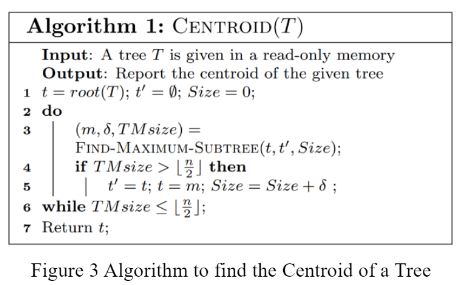
Reading Report:

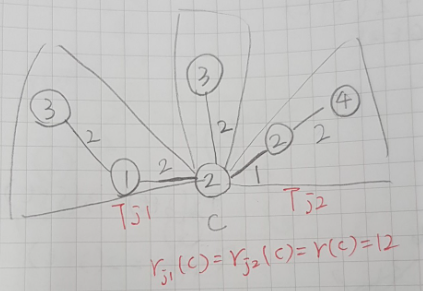
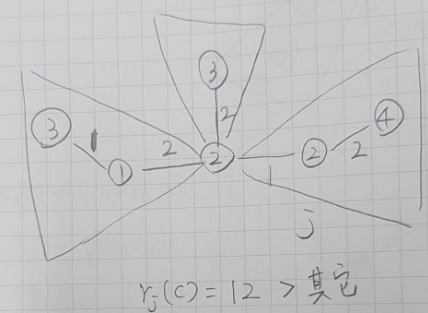
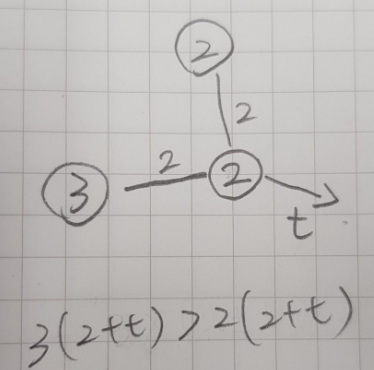
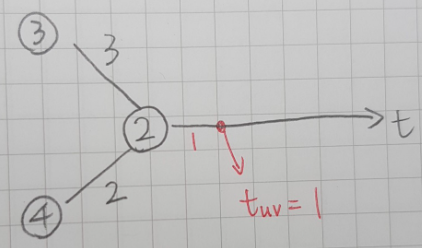
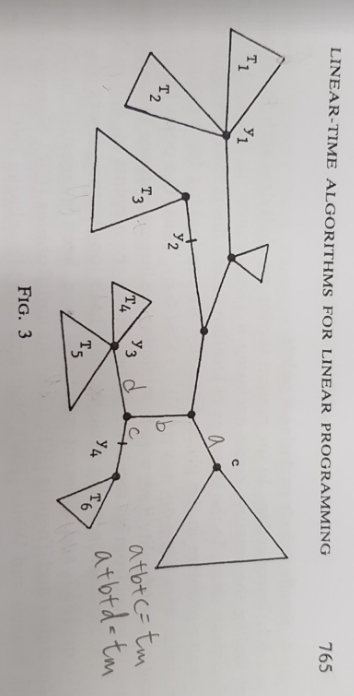
LINEAR-TIME ALGORITHMS FOR LINEAR PROGRAMMING IN R3AND RELATED PROBLEMS\*

學號: r08922136

姓名: 黃新予

1. 問題定義  
   問題名稱: The weighted center of a tree  
     
   **INPUT**: 一顆具備n個點的樹T = (V, E)  
   - 每一個節點具有不同權重(weight), 每一條邊長度不同dij，且樹上每一個點都可以連續地被表達。例如點x = (I, j; t)表示x與點i的距離為t; 與點j的距離為dij-t。  
   例如: 點x連接點3, 點5距離點3兩單位(勘誤: 圖中x距離點5距離應為2)。  
     
   (圖一、input範例)  
     
   **OUTPUT**:  
   找到一個加權中心點x\*使得r(x) = max{wi \* d(x, i), for all x in T} 最小，此解可以是連續的解，或是離散的解。連續的解為任何在樹上的點都可以，離散的解則限定在節點上。  
     
   (圖二、output範例)  
   以上面的例子來說，此圖的加權中心(weighted center)是x = (3,5; 16/9) 這個點。  
   此時的r(x\*) = 140/9 ~= 15.55
2. **解法描述**
   1. **尋找重心**

首先要先尋找樹的重心c，樹的重心定義在此不解釋，但由演算法得知找尋重心可利用DFS方法遍歷整棵樹，需要花費O(n)的時間。  
([樹的重心:定義,性質,算法分析,c++代碼,\_中文百科全書 (newton.com.tw)](https://www.newton.com.tw/wiki/%E6%A8%B9%E7%9A%84%E9%87%8D%E5%BF%83))  
(圖三、尋找重心pseudo-code)  
以上頁圖二為例，重心為點3。

* 1. **終止條件測試及判斷加權中心位置**  
     找到重心c之後，根據重心，可以切出多棵子樹，每顆子樹的節點定義為Vj(c)，計算所有相鄰節點j的rj(c) = max{wid(c, i)}。  
     其中r(c)表示點c加權距離最大的全域解，rj(c)表示在j子樹中，距離點c最大的加權距離。  
     接著進行以下討論決定是否達成終止條件，若否，則判斷加權中心位置在重心的哪一側。
     1. 運算結束:  
        若c相鄰節點中rj1(c) = rj2(c) = r(c)，表示兩個相鄰節點與重心c的加權距離皆為全域解，且因為兩個節點位於不同的位置，因此c點沒辦法繼續移動讓解變更小。  
        例如  
          
        (圖四、運算結束條件)  
        點c分出三顆子樹，但左邊和右邊的加權距離已經和r(c)相等了，點c沒辦法往左或往右移動，因此可以結束運算。
     2. 判斷加權中心位置:  
        若有一顆子樹j的加權距離比其他子樹k都大  
        rj(c) > rk(c)  
        那可知加權中心會位於此子樹內  
        例如  
          
        (圖五、判斷加權中心位於哪一顆子樹)  
        右側的加權距離最大，因此加權中心會落在右側的子樹中。
     3. 從圖五中可以知道最佳解x\* 應該從中心往右側移動，但要移動多少才能讓左右平衡，這就需要用到prune and search的方法了。
  2. **移動距離t的影響**  
     從步驟II中可以知道，加權中心要從重心移動，接著就要判斷要移動多少。假設加權中心位於目標子樹j內，則我們需要利用不在目標子樹內的兩點去做範圍限制。  
     假設點u, v 不屬於目標子樹內的節點，且點u, v到重心的加權距離為wud(u, c), wvd(v, c), 不失一般性假設wud(u, c) >= wvd(v, c)則考慮以下兩種情況:
     1. wu >= wv: 則不管移動多少，u的加權距離永遠會比v的加權距離還要大，因此可以把點v刪除。  
        例如  
          
        (圖六、p&s case 1-1)  
        重心要往右移動t，但不論怎麼移動，點3的加權距離永遠比上方點2的還要遠，因此移動時不需要考慮上方點2。
     2. 若wu < wv: 則移動的距離會影響要刪除哪一個點，臨界值tuv:   
        若移動>tuv，則刪除點u。  
        若移動<tuv，則刪除點v。  
        例如  
          
        (圖七、移動距離t的影響)  
        若移動超過1, 則點3可被刪除，否則可刪除點4。
  3. **配對並決定t值**  
     我們接著將不屬於目標子樹的所有節點兩兩配對成(u1, v1), (u2, v2),…. 至少可以分成組，每一對節點都可以刪除一個點或者算出一個tuv，所有t的集合{t1, t2, t3…}取中位數tm當成是目前需要移動的距離。  
     接著我們要計算最佳的加權中心x\*，是在tm距離之內，還是tm距離之外。如果在tm距離之內，則tm之後的節點對，可以刪除點u，若在tm距離之外，則tm之前的節點對可以刪除點v。  
     如此就可以刪除約n/8個節點。
  4. 判斷x\*與tm的關係  
     先從重心開始往目標子樹畫出tm的距離，再將超過tm距離的點集合成一顆顆的子樹，例如  
       
     (圖八、根據tm將目標子樹再切成更多子樹)  
     圖八中每個y值都離重心tm的距離，接著我們去計算各個y點到底下子樹的最大加權距離R(y)。如果y點分出多顆子樹，則根據分出的子樹數量分別計算。定義R = max{R(y1), R(y1)…} (因為y1分成兩顆所以有兩個不同數值)  
     此時就會有以下的case
     1. 某顆子樹i的加權距離Ri < R: 表示yi到子樹i內任意點的距離都不是最大的，則加權中心必不在此顆子樹內。
     2. 有兩顆子樹的加權距離相等且為最大加權距離, i.e. Ri = Rj = R: 表示說最佳解x\*不會落在任何子樹之內，因為有至少兩顆子樹目前處於平衡狀態，所以x\* 會在tm範圍以內。
     3. 只有唯一一顆子樹Ri = R: 表示x\*落在從重心到此顆子樹的路徑上。接著就直接以yi去計算是否達成終止條件，若沒有達成終止條件，則判斷最佳解在yi的哪一側。判斷完之後就可以知道x\*是在tm範圍內還是外。

用以上三種case可以求出x\*與tm的關係，用此關係即可刪除節點對中的某一點

* 1. **總結以上的演算法**
     1. 找出重心c -> O(n)
     2. 測試終止條件 -> O(n)
     3. 兩兩配對並計算個別的臨界t值或刪除點，並計算tm -> O(n)
     4. 根據tm找出所有的點y -> O(n)
     5. 計算所有y點與其底下子樹的加權距離Ri -> O(n)
     6. 根據所有Ri值討論3種case，並計算出x\*與tm的關係，若為case 3則測試是否達成終止條件 -> O(n)
     7. 根據x\*與tm關係，刪除節點對中的某一點。
     8. 回到步驟iii 。
  2. 由於步驟iii中刪除的點至少有1/8個點，因此時間複雜度為T(n) = T(7/8n) + cn  
     因此T(n) = O(n)

1. **讀後心得**不得不說我覺得這篇論文寫的真的有夠難懂，除了命題本身解法要自己想像subtree，且subtree又要再分出subtree之外，還有很多相同的符號夾雜，r(x), ri(x), ri(xi), R, Ri(u), Ri(ui), Ri(yi)….更慘的是這種跟圖有關的內容卻沒有用適當的圖去解釋。  
   我覺得最難懂的部份應該是在解釋如何判斷x\*跟tm的距離關係，那邊跳出整個命題，leaf x 和還有t去解釋如何篩選。作者用一種不同的觀點先闡述想法，但最後帶回來用原命題時卻沒有好好解釋case3要怎麼判斷，讀起來總覺得卡卡的很不順，那裏我讀了滿久才知道原來要使用到原本的終止條件去計算。  
   整體來說我覺得整篇使用的prune and search技巧都還滿像的，不管是線性規劃、找圓的的中心或是找樹的加權中心，都是利用配對的方式討論點對中需要刪除哪些部分。覺得能使用類似的想法解開乍看不同的問題很不錯。看完之後才發現老師講義真的是整理得很清楚，把作者沒講沒畫的部分都補足了。