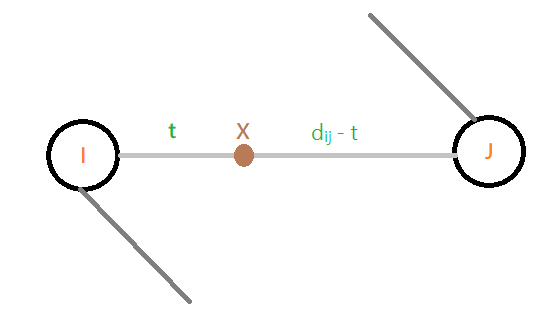
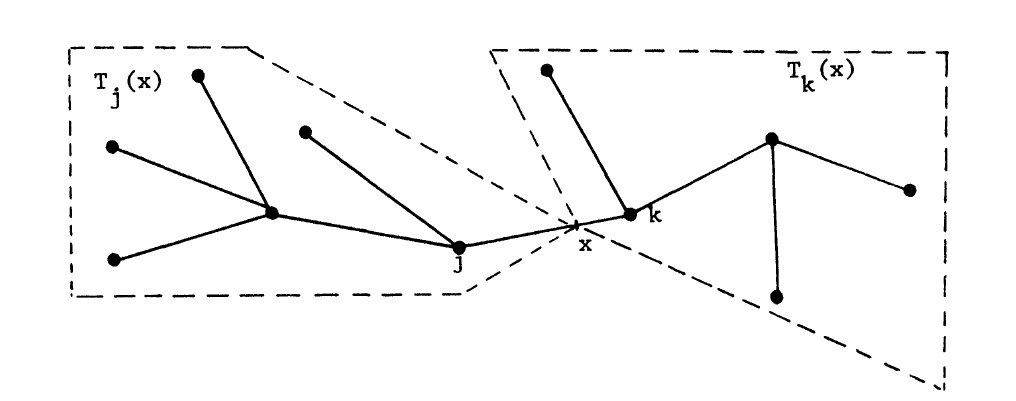
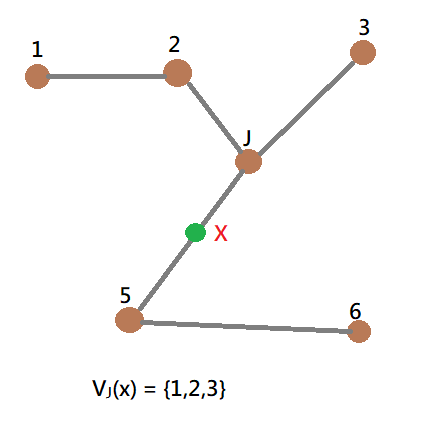
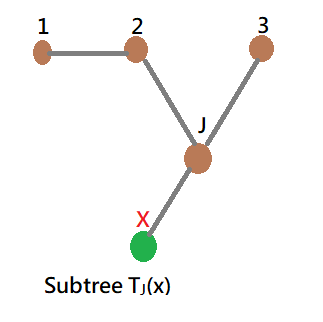
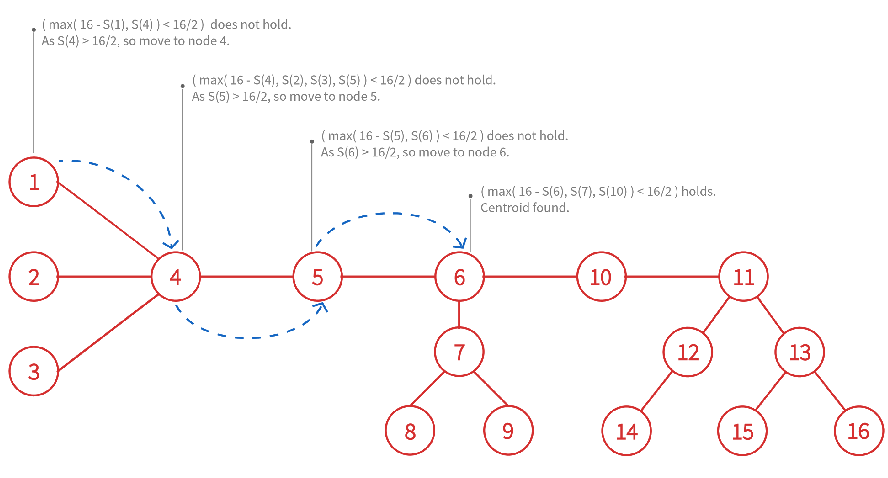
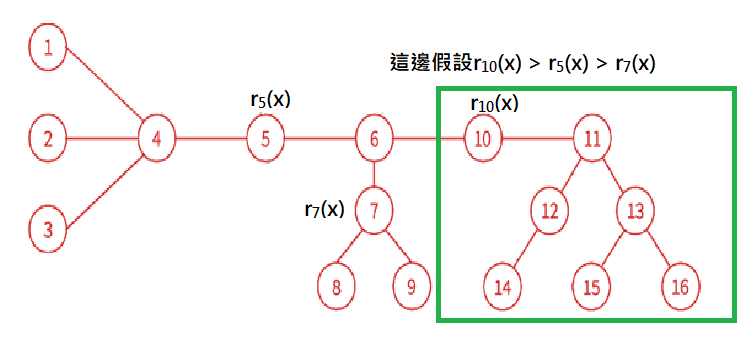
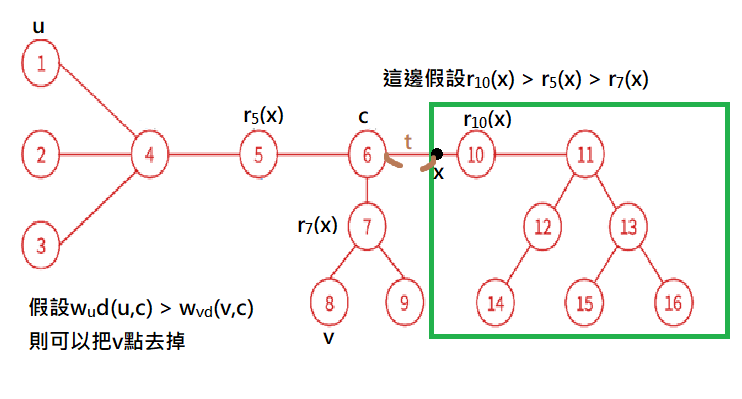
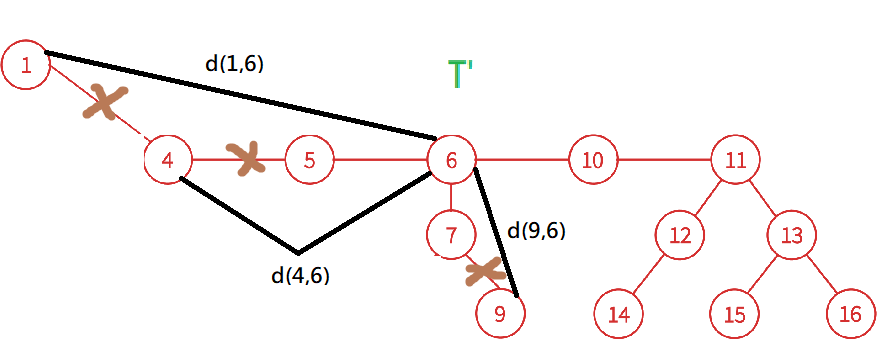
***The weighted center of a tree***

1. **問題定義**給定一棵具有n個節點的樹T = ( V , E )，其中每條邊( i , j )皆為一非負的長度d i j且每個節點ｉ具有一個非負的權重值ｗｉ­。  
   定義一個點ｘ= ( i , j : t ) 是位在邊( i , j )上的一點，並且與節點ｉ距離ｔ單位長度，與節點ｊ距離為ｄi j－ｔ單位長度，如下圖表示。   
   ****  
   對於此棵樹T上，我們可以找到一個加權中心使下列結果最小：   
    　　 ｒ(x)＝ max｛ｗid(x,i)，ｉ∈Ｖ｝  
     
   令ｘ為樹上的任一個點，若節點ｊ與ｘ相鄰，則ｘ位在與ｊ相鄰的一條邊上。我們將點 ｘ 經simple path到ｉ節點上所會經過的節點ｊ所形成的集合記做ｖｊ(x)，而將 ｖj(x) U｛ｘ｝ 所生成的子樹，記做Ｔj(x)，  
   如下圖所示。  
      
   最後定義一個函數，ｒj(x)＝max｛ｗid(ｘ, i)，ｉ∈ ｖj(x)｝，令ｊ1,ｊ2, … , ｊl 為所有與ｘ相鄰的節點，則我們可以發現上述的  
    ｒ(x) ＝ max｛ｒj1(x),… ,ｒjl(x)｝。  
   根據上述定義我們可以得到結論，假設存在兩個節點ｊa; ｊb，使得  
   ｒja(x) =ｒjb(x) =ｒ(x)，則x 即為此問題的解，因為r(x) 為一個 convex function。
2. **解法敘述**  
   此問題中所要求解的ｒ(x)在考慮簡單路徑的狀況下為一convex function。也就是說針對一點 x 若為 r(x) = max｛ｒj1(x), ｒj2(x), … ,ｒjl(x)｝的一個局部極值， 根據convex function的性質可以知道此解即為所求的加權中心點。論文作者以Prune and Search 的方式提出了以下的演算法：
3. Step 1:  
   首先，令c 為centroid of T，根據此篇論文的定義，c 是一個節點使得跟它相鄰的所有節點ｊ， Vj(c) ≤ n／2，n為此樹的節點數。  
   這一步只需要O(n) 的時間就可以找到c。找到c 的方法如下，先跑一次DFS 來找到每個節點的subtree 節點個數，接下來再跑一次DFS 來找到c。在第二次DFS當中，我們會不斷往subtree 節點個數 ≥ n/2 的節點移動，並且記錄該節點，直到移動到某個節點使得該節點subtree 的節點個數均< n/2，該節點即為 centroid。而我們知道DFS 可在線性時間內完成，所以可以在O(n) 的時間就可以找到c。  
   
4. Step2:  
   已知重心c並計算所有每一相鄰節點 j 的ｒｊ(c)，此部分可以於 O(n) 時間內完成，並作以下討論：   
   (1) 若存在兩相鄰節點 j1 和 j2（其中 j1 != j2）滿足：ｒj1(c) =ｒj2(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：ｒj(c) > ｒk(c) 則可知所求加權重心會位於子樹 Tj(c)上。  
   
5. Step3:  
   對於任意兩節點 u, v ∉ Vj(c)和與重心c距離 t 的點 x ∈ Tj(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 wu(d(u, c) + t) = wv(d(v, c) + t) 的過程中，不失一般性的狀況下可假設 wud(u, c) ≥ wvd(v, c) 若且唯若存在一非負的 tuv 滿足 0 ≤ t ≤ tuv，tuv = (wud(u, c) – wvd(v, c)) / (wv – wu)，其中：  
   若加權中心與重心位置小於 tuv，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 tuv，則在尋找加權中心的過程中可以不用考慮節點 u。  
     
   除此之外，計算完所有的tuv 後，我們計算這些值的中位數tm，然後對所有的節點i 計算d(c, i)，這步驟能在O(n)的時間內可以完成，如果能找到一個邊(i, j)使得d(x, i) ≤ tm ≤ d(x, j)，那我們就知道在(i, j)之間有一個y 使得d(x, y) = tm。所有與c距離超過tm 的點可以組成以y1, y2,…, yl 為root 的subtrees，令他們映射到subtrees分別為T1,T2, …, Tm，而這些subtrees 的root 分別u1,u2,… ,um。定義Ri(x) = max { wkd(x, k), k∈ Vi } and R = max{ Ri (ui) , i = 1,…,m }，以上計算Ri的動作也只需線性時間即可完成，之後分成以下幾種情形，如果Ri(ui) < R，那此問題的解自然不會在Ti 當中。然後，可以知道解應該存在與c 距離小於tm 的點當中。最後，如果存在一個i 使得Ri(ui) = R，在這個情況下，我們只需要從rj(y) 來判斷，而j為任何一個與ui 相鄰的點，那y = ui。
6. Step4:  
   對所有沒被刪掉的節點u，新增一條邊(u, c)，並從T 當中算出d(u, c) 以及沿用wu，將這些節點加回Tj(c) 當中，組成一顆新的樹T’。重複流程直到找到解為止。  
   
7. Time Complexity  
   根據c 的定義，不在Tjmax(x) 的節點至少有n/2 個，所以至少會有n/4 個配對，在每個配對中，我們至少會刪除其中一個節點， 以在每一輪中，至少會有n/8 個節點被刪除，又每輪的時間複雜度為O(n)，則 T(n) ≤ T(7n/8) + O(n)，最後我們可以得到T(n) = O(n)。
8. **讀後心得**Prune and Search為Divide and Conquer的一種特例，常被用於求解最佳化問題。problem2 就是採用此策略去解2-Dimensional Linear Programming，將原本的線性規劃問題進行簡化，直到problem set夠小後，直接去解他，進而免去了對矩陣進行高斯消去法的繁複計算，我也在網路上稍微查詢了一下，使用Prune and Search 概念求解的問題其實不算多，大致上有：Linear Programming in the Plane、The Weighted Center of Tree、Smallest Circle enclosing n Points與Linear Programming in R­­­3。在寫作業的過程中發現其實整體演算法的內容其實十分相近，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。在本篇論文出來之前，The Weighted Center of Tree問題的最佳解為O(n log n)，而本篇論文利用prune and search方法，找到一個O(n) 的解法，關鍵便是在每一輪的計算中，都會去除掉一部分的節點，使得此演算法能在線性時間內完成。  
   另外，我覺得這篇論文有些地方並沒有解釋得很清楚，讓我有些地方一看再看才能理解，閱讀這篇論文的時候讓我體會到教授提到要舉例的重要性，之後的作業我會更注意在如何淺顯易懂的讓人理解論文。