109 臺大資演參考答案

Part1:

1. 3,4,5
2. 3
3. 2,4,5
4. 1,2,4,5
5. 2,3,5
6. 3,4,5
7. 1,3
8. 1,2
9. 1,4,5
10. 4.5

Part2:

11. (5 points) Given an array of n integers $A = (a_1, a_2, ..., a_n)$ $(n > 0)$, you are asked to find the maximum and minimum elements with minimum comparisons. A simple solution is to iteratively compare each element with current max and min. It takes $2n$ comparisons in total.

    (a) (3 points) Devise an algorithm that requires less than $1.67n$ comparisons.

    (b) (2 points) How many comparisons does your algorithm take? Show your derivation.

(a).

1. Create 2 sets *max* ={}, *min*={}
2. for each a_i, a_j *in array* A: //倆倆比較大小，分出
3. { if $a_i > a_j$: append a_i to max and a_j to *min* //max, min set
   { else: append a_i to min and a_j to *max* //這裡花共(n/2)
4. Create a new set max_{new}
5. for each a_i, a_j *in* max_{new}: //迭代的倆倆比較大小
   { if $a_i > a_j$: append a_i max_{new} //找出 max int
   { else: append a_j max_{new} //這裡共$\sum_{i=2}^{k} \frac{n}{2^i}$
6. repeat 4 to 5 until find max_{new} only has 1 element
7. Create a new set min_{new}
8. for each a_i, a_j *in* min_{new}: //同上，改為找 min int
   { if $a_i > a_j$: append a_i min_{new}
   { else: append a_j min_{new}
6. repeat 7 to 8 until find max_{new} only has 1 element

(b). 加起來共 1.5n 的比較次數

12. (15 points) There are n stations along a coastal railway ($n > 0$). You're planning to select some of them to open cafes. Three arrays S, L and R have been given, including

- $S = (s_1, s_2, ..., s_n)$: the list of the stations from s_1 (first) to s_n (last).
- $L = (l_1, l_2, ..., l_n)$: the locations of the stations, where l_i is the distance of s_i from the first station s_1. So $l_1 = 0$ and l_n is the length of the railway. $l_1 < l_2 < ... < l_n$.
- $R = (r_1, r_2, ..., r_n)$: the revenues of the cafes, where r_i (> 0) is the revenue for opening a cafe in s_i.

The only one constraint in your plan is that the distance of any pair of your selected stations should be longer than a given threshold T. If s_i and s_j ($i \neq j$) are selected, the total revenue would be $l_i + l_j$. Different selection leads to different total revenue. Given S, L, R and T, your goal is to pick up a subset of the stations to maximize the total revenue under the constraint. Suppose that $f(n)$ returns the maximum total revenue for the cafes you select from the first n stations. Please answer the following questions. **No code is required (code will not be graded).**

(a) (9 points) Give an $O(n^2)$ solution by defining a recurrence formula for $f(n)$. Clearly explain the meaning of your formula and why it can be computed in $O(n^2)$ time.

(b) (6 points) Consider the special case that the distance difference between two consecutive stations is 1, i.e., $l_{i+1} - l_i = 1$ ($1 \leq i \leq n-1$). Give an $O(n)$ solution by defining a recurrence formula for $f(n)$. Clearly explain the meaning of your formula and why it can be computed in $O(n)$ time.

(a)

當我們考慮第 i 個車站 (si) 時，我們只有兩個選擇：「選」 或 「不選」。我們的目標是看哪種選擇能讓總獲利最大。

定義 f(i) 為：從第 1 站考慮到第 i 站時，能賺到的最大利潤。

1. 選擇一：不選第 i 站
   - 既然不選這站，那利潤就跟「考慮到第 i-1 站」的情況一模一樣。
   - 獲利 = f(i-1)
2. 選擇二：選第 i 站
   - 如果選了這站，我們會先拿到這站的收入 ri。
   - 但因為距離限制 T，我們必須往回找。我們不能選 i-1 站（可能太近），也不能隨便選。
   - 我們必須回頭檢查所有之前的車站 j (其中 j < i)，找出那些距離大於 T (即 $l_i - l_j > T$) 的車站。
   - 在這些合法的「前一家店」候選人中，我們挑一個累積獲利 f(j) 最大的來接續。
   - 獲利 = r_i + max{ f(j) } (其中 j 必須滿足距離條件)

公式的意思：

f(i) = max(不選的獲利, 選的獲利)

O(n^2):

- 我們要算 n 個車站的 f(i)，要跑一次迴圈(n 次)。
- 在決定「選第 i 站」的時候，我們必須回頭掃描前面所有的車站 j，去檢查距離並找最大值，這又是一層迴圈（最多 i 次）。
- 雙層迴圈，$1 + 2 + ⋯ + n，所以是 O($n^2$)。

(b).

這個情況變簡單了，因為車站是均勻排列的（距離都是 1）。

$l_1 = 0,\ l_2 = 1,\ l_3 = 2$ ...

這意味著，車站 i 和車站 j 的距離，直接就是它們的編號差| i - j |。

1. 選擇一：不選第 i 站
   - 獲利 = f(i-1)。
2. 選擇二：選第 j 站
   - 拿到收入r_i。
   - 這時不需要像 (a) 那樣回頭一個一個檢查誰符合距離。
   - 因為距離固定是 1，若規定距離要大於 T，很清楚知道，第 i-T-1 站（或是題目定義的邊界）就是離我最近且合法的那個邊界。
   - 因為 f(x) 是一個累積最大值的函數（越後面的站累積的機會越多，數值只會變大或持平），我們不需要找更前面的，直接抓「合法範圍內最近的那一站」的答案即可。
   - 獲利 = r_i + f(i - T)（假設間隔 T 是安全距離）。

O(n):

- 我們要算 n 個車站的 f(i)，這是一層迴圈。
- 在迴圈內部，只需要做一次陣列讀取，O(1)。
- 總運算量 = n * 1 = O(n)。

13. (15 points) A disjoint-set data structure supports two operations. One is UNION(x, y), which merges the two roots of the trees containing x and y. The other is FIND(x), which returns the root of the tree containing x. Union-by-height is a union heuristic, which keeps track of the heights of the trees to attach the shorter tree to the root of the taller tree.

    (a) (9 points) Given an undirected graph $G = (V, E)$, where $V = \{v_1, ..., v_m\}$ $(m > 0)$ and $E = \{e_1, ..., e_n\}$ $(n > 0)$, devise an algorithm to check if the graph G is a tree or not by using UNION(v_i, v_j) with union-by-height and FIND(v_k) with no path compression, where v_i, v_j and $v_k \in V$. Clearly describe your solution and analyze the time complexity of your algorithm in the worst case. **No code is required (code will not be graded).**

    (b) (6 points) The union-by-height heuristic prevents the height of a tree from growing linearly. Consider another heuristic, called union-by-descendant, which always attaches the tree with fewer descendants to the root of the tree with more descendants. Suppose that path compression is not applied. Which of the two heuristics is asymptotically better? Why?

(a)

G 必須滿足沒有 cycle，且是連通圖

1. 初始化 (Initialization)：

   針對圖中 m 個頂點 (v_1 到 v_m)，建立 m 個獨立的集合。此時每個點都是自己集合的 root。

2. 遍歷 Edges 並檢查 Cycle：

   對於圖中的每一條邊 e_k = (u, v)，執行以下檢查：

   - 分別找出 u 和 v 所屬集合的根節點：root_u = FIND(u)，root_v = FIND(v)。
   - 如果 root_u == root_v，表示 u 和 v 已經在同一個集合中，這條邊會形成 Cycle。
     - 直接回傳 G 不是樹 。
   - 合併：如果 root_u != root_v，則執行 UNION(root_u, root_v) 將兩個集合合併。

3. 檢查連通性 (Check Connectivity)：

   當所有邊都檢查完畢且沒有發現迴路後，檢查目前的「集合總數」。

   - 檢查是否只剩下 1 個集合。若集合數 > 1，代表圖不連通。
     - 回傳 G 不是樹。

4. 結論：

   若上述檢查皆通過，則回傳 "G 是樹 (True)"。

(b)

Union by height 會使樹高為 logm，如此單次 union 的花費為 logm，

有 n 條邊，於是 time complexity 為 nlogm。

Union by descendent 將 node 少的樹合併到 node 多的樹之下，

樹高也為 logm，兩者 asymptotic complexity 相同。

14. (20 points) A tree T is assumed to be simple, undirected, and with **positive edge-weights**. Let $d_T(u, v)$ denote the distance between u and v on T. For a vertex v, the *eccentricity* of v is the maximum of the distance to any vertex in the tree, i.e., $\max_{u \in V}\{d_T(v, u)\}$. The *diameter* of a tree is the maximum of the eccentricity of any vertex in the tree. (The term "diameter" is overloaded. It is defined as the maximum eccentricity and also as the path of length equal to the maximum eccentricity.) The *radius* of a tree is the minimum eccentricity among all vertices in the tree, and a *center* of a tree is a vertex with eccentricity equal to the radius.

    (a) (5 points) Prove or disprove that there exists a tree with three different diameters and two centers.

    (b) (5 points) Prove or disprove that there exists a tree with three centers.

    (c) (10 points) Prove or disprove that any center of a given tree T must lie in the diameter of T.

(a)

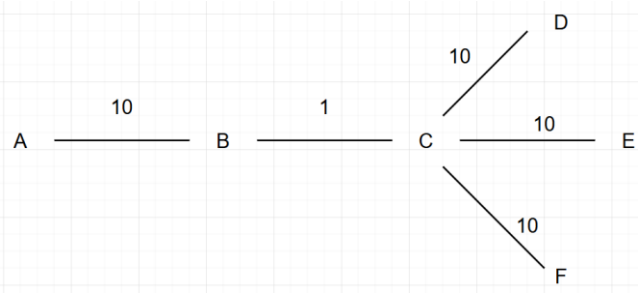
prove ，如下圖有三種 path 的長度都為 diameter=21:

$A \rightarrow B \rightarrow C \rightarrow D$

$A \rightarrow B \rightarrow C \rightarrow E$

$A \rightarrow B \rightarrow C \rightarrow F$

有兩個點(B, C)的 eccentricity 都為 radius=11:



(b)

disprove，證明不可能有三個中心:

每次將 tree T 上所有的 leaf node（degree 為 1 的點）同時刪除，得到一棵新樹 T'。

刪除所有 leaf 後，tree 上剩餘每個點的 eccentricity 都會剛好減少此 edge 的大小，且相對大小關係不變。因此，T' 的中心位置與原樹 T 相同。

重複上述刪除葉節點的步驟。由於樹是有限的，最後必定會收斂到以下兩種情況之一：

1.只剩下一個頂點（該點即為唯一的中心）。

2.剩下兩個相鄰的頂點（這兩點即為唯二的中心）。

因為過程最後不可能剩下三個獨立的點（那樣不連通）或三個相鄰的點（因為兩端的點會被視為 leaf 而在該步驟被修剪掉），所以樹不可能有三個中心。

(C)

prove，矛盾證:

1. P 是 tree T 上的一條 diameter，其端點為 x, y。
2. 設 tree T 有一個 center c，且 c 不在這條 diameter 上。

   因為 T 是連通的樹，所以必然存在一條路徑連接 c 到 diameter P；

   令 v 為 P 上距離 c 最近的點。
3. 因為 c 在 P 之外，對於直徑的任一端點 x, y 來說，從 c 到 x 的路徑必須經過 v:

   $d(c, x) = d(c, v) + d(v, x)$

   $d(c, y) = d(c, v) + d(v, y)$
4. eccentricity 比較：
   - v 的 eccentricity 至少是 d(v, x) 或 d(v, y) 其中之一（因為 x, y 是全樹距離最遠的一對點）。
   - c 的 eccentricity 是 c 到樹上最遠點的距離，所以 c 的 eccentricity $\geq \max(d(c, x), d(c, y))$
5. 將 3. 代入 4.，產生矛盾:

c 的 eccentricity $\geq d(c, v) + \max(d(v, x), d(v, y))$ → c 的 eccentricity $\geq d(c, v)$ + v 的 eccentricity

→ c 的 eccentricity > v 的 eccentricity。如此得出 c 的 eccentricity 不是最小，c 不是 center，產生矛盾；因此任一 center c 必定在 diameter 上。

15. (15 points) A Single Nucleotide Polymorphism (SNP, pronounced *snip*) is a single nucleotide variation in the genome that recurs in a significant proportion of the population of a species. The patterns of *Linkage Disequilibrium* (LD) observed in the human population reveal a block-like structure. LD refers to the association that particular alleles at nearby sites are more likely to occur together than would be predicted by chance. The entire chromosome can be partitioned into high LD regions interspersed by low LD regions. The high LD regions are usually called "haplotype blocks," and the low LD ones are referred to as "recombination hotspots." Since there is little or no recombination within a haplotype block, these SNPs are highly correlated. Consequently, a small subset of SNPs, called tag SNPs or haplotype tagging SNPs, is sufficient to categorize the haplotype patterns of the block. It has been shown that we can recast the tag SNP selection problem as **Problem W**, which is,"Given a universal set $\mathcal{U} = \{u_1, u_2, ..., u_n\}$ and a family $\mathcal{F} = \{F_1, F_2, ..., F_m\}$ of subsets of $\mathcal{U}$, find a minimum-size subset $\widehat{\mathcal{C}}$ of $\mathcal{F}$, such that every element of $\mathcal{U}$ belongs to at least one subset in $\widehat{\mathcal{C}}$."

(a) (5 points) Prove or disprove that a greedy approach always delivers an optimal solution for **Problem W**.

(b) (10 points) Prove or disprove that **Problem W** is NP-complete.

(a)

這是一個 set covering problem，要找到 S 的一個最小的子集，使得他們的併集等於全集。

例如 u={1,2,3,4,5}，S={{1,2,3}, {2,4}, {3,4}, {4,5}}，

我們可以找到一個 S 的子集{{1,2,3}, {4,5}}其併集是 u，稱此為一個集合覆蓋。

disprove，可以舉出反例證明 greedy approach 不是最佳方法:

u={1,2,3,4,5,6}，S={{1,2,3}, {4,5,6}, {2,3,4,5}}，使用 greedy 方法，

會先選涵蓋最多元素的{2,3,4,5}，接著再選{1,2,3}和{4,5,6}，總共選了 3 個集合。

但是 optimal solution 是選{1,2,3}和{4,5,6}，只需選 2 個集合。

(b)

prove: 證明 W is NP-complete

W ∈ NP:

可在多項式時間內快速檢查 W 有沒有覆蓋住所有元素。

W ∈ NP hard:

將 vertex cover 問題化簡到 W，令一個 vertex cover 為 G=(V, E), k，將其轉化為 u, S, k'

- 把 E 看作 u 中的元素：

最終的目標是要覆蓋所有的邊，就像 Problem W 中要覆蓋所有的元素。

- 把 V 看作集合 S：

每一個頂點連接著好幾條邊，這就像是一個集合裡面包含了好幾個元素。選了一個頂點，等於蓋住了它連出去的所有邊。

- 目標不變 (k'=k)：

如果在 Vertex Cover 問題中，我們想問「能不能用 k 個頂點蓋住所有邊？」

這就完全等同於在 Problem W 中問「能不能用 k 個集合蓋住所有元素？」

如此證明一個 NP-Complete 問題可在多項式時間內化為 W，W 為 NP-hard。