COL351 Fall 2021 Assignment 1

Tamajit Banerjee

TOTAL POINTS

54 / 60

QUESTION 1

Minimum Spanning Tree 20 pts

1.1 Unique MST 5 / 5

✓ **+ 5 pts Correct**

+ 0 pts Incorrect/Unattempted

1.2 MST in O(n) time 10 / 15

✓ **+ 15 pts Correct**

+ 0 pts Incorrect/Unattempted

+ 3 pts Algorithm without proof

✓ **- 5 pts Used claims not proved in lecture**

You need to explain as well as prove the "cycle property" that you have mentioned as it is not a standard result done in class.

1 C not G

QUESTION 2

Hufmann Encoding 20 pts

2.1 Frequency vector with Fibonacci numbers 10 / 10

+ 0 pts Incorrect/Unattempted

Identifying and proving $$\sum_{i=1}^{n-1} F_i < F_{n+1}$$ (or its stronger form $$\sum_{i=1}^{n-1} F_i = F_{n+1}-1$$)

+ 0 pts Incorrect

+ 1 pts Partial credit (minimal)

+ 3 pts Partial credit (halfway)

+ 4 pts Partial credit (minor issue)

✓ **+ 5 pts Full credit**

Rest of the analysis, identifying the minimum two frequencies in each step and the structure of the Huffman tree

+ 1.5 pts Partial credit

✓ **+ 3 pts Full credit**

Encoding of the two letters with frequency 1

+ 0 pts Incorrect

+ 1 pts Not explicitly mentioned

+ 1 pts Partial credit

✓ **+ 2 pts Full credit**

2.2 Frequency vector with similar values 9 / 10

+ 0 pts Incorrect/Unattempted

✓ **+ 3 pts F[i] + F[j] > F[n]**

✓ **+ 2 pts In each iteration i such that some original characters have not joined the tree, characters with frequencies F[2i-1] and F[2i] in the sorted frequency vector F are joined (to create dummy character with frequency F'[i]).**

+ 1 pts F'[i] >= F'[i-1]

✓ **+ 2 pts F'[n/2] < 2F'[1] (or, F'[n/2] < F'[1] + F'[2])**

✓ **+ 2 pts Recursive reasoning using frequency vector F'**

+ 10 pts Proof by contradiction: all correct

For this to reduce to the "same problem", you must also show that a_1^1 <= a_2^1 <= ... <= a_{n/2}^1.

QUESTION 3

Graduation Party of Alice 20 pts

3.1 Optimal set of invitees 15 / 15

✓ **- 0 pts Correct**

is G \ Sv = H ?

3.2 Dinner tables **5 / 5**

✓ **- 0 pts** **Correct**

COL351 Assignment 1

Tamajit Banerjee | Mustafa Chasmai
2019CS10408 | 2019CS10341

September 2021

1 Minimum Spanning Tree

It is assumed that $G = (V, E)$ is connected, and thus $\exists$ a minimum spanning tree $T = (V, E'), E' \subset E$.

(a) **To prove**: G has a unique MST

Proof: (By induction)

We sort the edges in ascending order and number them starting from 1 to m.

H(i) : The edges with index 1 through i have been processed and the edges selected by Kruskal's algorithm are present in every MST and the edges rejected by Kruskal's algorithm are not present in any MST.

- Base Case: H(0) is trivially true. We have processed 0 edges so there cannot be any edges which is selected by Kruskal's algorithm but not present in MST and vice versa.
- Induction Hypothesis : Let H(i - 1) be true for some i $\in [0, m]$.
- Induction Step: Let E_{i-1}^{sel} be the set of edges selected by Kruskal's algorithm before the current i^{th} edge is encountered. Let e_i be the current edge.
  - (I) Case I: $e_i \cup E_{i-1}^{sel}$ forms a cycle.
    According to Kruskal's algorithm, this edge will be rejected because of the cycle. Now, by hypothesis, every MST must have all the edges of E_{i-1}^{sel}. Thus, e_i cannot present in any MST, since it cannot have a cycle. Hence H(i) is true for Case I.
  - (II) Case II: $e_i \cup E_{i-1}^{sel}$ does not form a cycle.
    Since the edges are unique, weight of e_i will be strictly smaller than all edges yet to be processed.
    Assume: e_i is not present in some MST T. Because the MST is maximally acyclic, adding e_i to T will lead to some cycle. Since it does not form a cycle with E_{i-1}^{sel}, and by hypothesis, rejected edges willl not be in T, it has to form a cycle with edges yet to be processed. Thus, e_i will be the minimum weighted edge in the cycle, and by the cycle property, T cannot be an MST.
    Thus, by contradiction, e_i has to be present in every MST. Thus, H(i) is also true in this case.

Thus, by the principle of mathematical induction, H(i) is true $\forall i \in [0, m]$. Thus, H(m) is true. Since Kruskal's algorithm is deterministic and unique, a single MST is produced by it. By H(m), this can be the only MST possible. Thus, G has a unique MST.

1.1 Unique MST **5 / 5**

✓ **+ 5 pts** **Correct**

+ 0 pts Incorrect/Unattempted

(b) Obtain a MST of $G = (V, E), m \leq n + 8$ in O(n) running time (n = $|V|, m = |E|$)

Algorithm 1: MST of G in O(n) running time

```
1 M ← G;
2 while M contains a cycle do
3 |   C ← a cycle in M ;
4 |   e ← largest weight edge in C ;
5 |   M ← T \ {e};
  end
```

Algorithm to find the cycle in line 3:

We have to perform a DFS on the graph and keep a record of the visited nodes and the parent node. If we encounter a visited node other than its parent from a node then we have found a cycle since we already have a treepath from the visited node to the current node. We can retrieve the cycle by maintaining a stack of nodes during DFS and return it when the cycle is detected.

Proof of correctness:

Cycle Property: Suppose G has all edges of distinct weight. Let C be any cycle in G, and let e be the most expensive edge in C. Then e does not belong to any MST of G.

For the purpose of proof, let M_i be the graph M after i iterations of the while loop.

INV: M_i is connected, spanning and contains all the edges belonging to MST of G. We can prove this by induction on i

- Base case : Assumption G is connected, otherwise no MST exists. By initialisation, M ← G. Thus the invariant holds.
- Induction Hypothesis: INV holds for (i-1) >= 0
- Induction Step: Consider any edge e = (u, v) removed by Algorithm in the i th iteration. Since e is the edge of maximum weight on a cycle C in M_{i-1}, by Cycle Property, e does not belong to any minimum spanning tree. Thus Mi contains all edges belonging to the MST of G. Further since e is part of a cycle, it is not a bridge. Thus removing e from M_{i-1} will not disconnect it. Thus M_i is connected. Note that every vertex of a cycle has degree atleast two. Thus after removal of e, no vertex degree becomes 0 so Mi is spanning. Loop Termination Condition: M is acyclic. Thus M is connected and acyclic → M is a tree . Further it is spanning and contains all edges belonging to the MST of G → M is the MST of G.

Time Complexity:

Since there are at max n+8 edges, and the MST will have n-1 edges, a maximum of 9 edges can be removed. Thus, the while loop in line 2 will have a maximum of 9 iterations (O(1)).

In line 3, a DFS is performed, which takes O(m) = O(n+9) = O(n) time. Again, in lines 4 and 5, the maximum edge of the cycle is found and removed, which takes O(no of edges in the cycle) = O(n) time.

Thus, each iteration takes O(n) time, and there are O(1) iterations. Thus, the total time complexity of the algorithm is O(n).

1.2 MST in O(n) time 10 / 15

✓ **+ 15 pts** **Correct**

+ 0 pts Incorrect/Unattempted

+ 3 pts Algorithm without proof

✓ **- 5 pts** **Used claims not proved in lecture**

You need to explain as well as prove the "cycle property" that you have mentioned as it is not a standard result done in class.

1 C not G

2 Huffman Encoding

(a)

Lemma 2.1. *The sum of the first n Fibonacci numbers satisfies the inequality*

$$u_1 + u_2 + ... + u_{n-1} + u_n < u_{n+2}$$

Proof. From the definition of the Fibonacci sequence, we know

$$u_1 = u_3 - u_2,$$

$$u_2 = u_4 - u_3,$$

$$u_3 = u_5 - u_4,$$

$$...$$

$$u_{n-1} = u_{n+1} - u_{n+2},$$

$$u_n = u_{n+2} - u_{n+1}.$$

We now add these equations to find $u_1 + u_2 + ... + u_{n-1} + u_n = u_{n+2} - u_2$. Recalling that u2 = 1, and since $u_{n-1} - u_2 = u_{n-1} - 1 < u_{n-1}$, we have the required inequality. □

Lemma 2.2. *In the i^{th} merging step in the Huffman algorithm , it will always merge the binary tree formed by the first i letters and $(i+1)^{th}$ letter.*

We can prove this by induction.

Proof. Let us define H(i): i^{th} merging step in the Huffman algorithm merges the binary tree formed by the first i letters and $(i+1)^{th}$ letter.

- Base case: H(1) is true since the 1st and second elements are the two lowest ones, and thus the tree formed by first element is merged with the second element
- Induction Hypothesis: H(i) is true for some $i \in [1, n-1]$.
- Induction Step: the tree formed by the first i letters will have frequency $f_{merged} = u_1 + u_2 + ... + u_{n-1} + u_i$, which satisfies $f_{merged} < u_{i+2}$, as shown by Lemma 2.1. Now, since fibonacci numbers form an increasing sequence, we have, in general, $f_{merged} < u_{i+2} < u_j \forall j > i+2$ and $u_{i+1} < u_j \forall j > i+1$. Thus, both u_{i+1} and f_{merged} are lower than $u_j \forall j > i+2$, and hence are the two minimum frequencies to be merged during Huffman encoding. Thus, H(i) is true.

Thus, by the principle of mathematical induction, H(i) is true $\forall i \in [1, n]$.

□

Thus, u_n will be encoded as 0, u_{n-1} as 10, general i^{th} letter will be encoded as 1 appended to the left of the encoding for $i + 1^{th}$ letter, for all $i \in [2, n-1]$. The last letter will be encoded as 2^{n-1}, while the second last as $2^{n-1} - 1$. Thus, the two letters having the lowest frequencies will have encoding length n-1.

2.1 Frequency vector with Fibonacci numbers 10 / 10

+ **0 pts** Incorrect/Unattempted

Identifying and proving $$\sum_{i=1}^{n-1} F_i < F_{n+1}$$ (or its stronger form $$\sum_{i=1}^{n-1} F_i = F_{n+1}-1$$)

+ **0 pts** Incorrect
+ **1 pts** Partial credit (minimal)
+ **3 pts** Partial credit (halfway)
+ **4 pts** Partial credit (minor issue)

✓ **+ 5 pts Full credit**

Rest of the analysis, identifying the minimum two frequencies in each step and the structure of the Huffman tree

+ **1.5 pts** Partial credit

✓ **+ 3 pts Full credit**

Encoding of the two letters with frequency 1

+ **0 pts** Incorrect
+ **1 pts** Not explicitly mentioned
+ **1 pts** Partial credit

✓ **+ 2 pts Full credit**

(b) Let the character frequencies sorted in non-decreasing manner be $a_1, a_2, a_3...a_n$, such that $a_i \leq a_{i+1} \forall i \in [1, n-1]$.

Lemma 2.3. *For Huffman encoding of a set of characters, if the maximum character frequency is less than or equal to twice the minimum character frequency, all character frequencies are less than or equal to the sum of any two character frequencies.*

Proof. Following the notation defined above, it is given that $a_n ¡ 2 * a_1$. Now, since $a_1 \leq a_i \forall i \in [1, n]$, we can observe that $2 * a_1 = a_1 + a_1 \leq a_i + a_j \forall i, j \in [1, n]$. Now, we also know that $a_k \leq a_n \forall k \in [1, n]$.

Hence, from these two observations, we have $a_k \leq a_i + a_j \forall i, j, k \in [1, n]$, which is the statement of the lemma. □

Lemma 2.4. *For Huffman encoding of a set of characters, if the maximum character frequency is less than or equal to twice the minimum character frequency, then the maximum sum of two frequencies is less than or equal to twice the minimum sum of two frequencies.*

Proof. Following the notation defined above, it is given that $a_n \leq 2 * a_1$, and we need to prove that $a_n + a_{n-1} \leq 2 * (a_1 + a_2)$. Now from Lemma 2.3, we have $a_n \leq a_1 + a_2$ and $a_{n-1} \leq a_1 + a_2$. Thus, simply summing them up will give us the required result. □

Theorem 2.5. *For Huffman encoding of a set of total $n = 2^k$ characters for $k \in$, if the maximum character frequency is less than or equal to twice the minimum character frequency, then the algorithm will lead to a complete binary tree.*

Proof. According to the algorithm, in the first step, a_1 and a_2 will be merged. Now by Lemma 2.3, this new frequency $a_1 + a_2$ will be larger than all of a_i. Thus, it will not be merged as long as atleast two of a_i are left merging. Let $a_i^1 = a_{2i-1} + a_{2i}$. Thus, for even n, after n/2 merges, the frequencies will become $a_1^1, a_2^1...a_{n/2^1}^1$. Now, using Lemma 2.4, $a_{n/2}^1 \leq 2 * a_1^1$, and thus, the same problem gets reduced to that of size n/2, with each node being replaced by a binary tree of height 1. Iterating over this process, and getting a^2, a^3, ... we can reach upto a^k, where we will have a complete binary tree of height k.

Thus, Huffman encoding with this condition will lead to a complete binary tree.

□

Now, it is given in the question that the maximum character frequency is strictly less than twice the minimum character frequency. Thus, by Theorem 2.5, Huffman encoding will lead to a complete binary tree, with each leaf node at depth $k = log_2(n)$, and thus, each having k digits to encode it. Thus, this is equivalent to having a fixed k-length encoding. Thus, the compression in this case is same as that of the ordinary fixed-length encoding.

2.2 Frequency vector with similar values 9 / 10

+ **0 pts** Incorrect/Unattempted

✓ **+ 3 pts** **F[i] + F[j] > F[n]**

✓ **+ 2 pts** **In each iteration i such that some original characters have not joined the tree, characters with frequencies F[2i-1] and F[2i] in the sorted frequency vector F are joined (to create dummy character with frequency F'[i]).**

+ **1 pts** F'[i] >= F'[i-1]

✓ **+ 2 pts** **F'[n/2] < 2F'[1] (or, F'[n/2] < F'[1] + F'[2])**

✓ **+ 2 pts** **Recursive reasoning using frequency vector F'**

+ **10 pts** Proof by contradiction: all correct

For this to reduce to the "same problem", you must also show that a_1^1 <= a_2^1 <= ... <= a_{n/2}^1.

3 Graduation Party of Alice

(a) Suppose each friend of Alice is a node and if 2 of her friends know each other it is represented by an undirected edge between them.

We define property P(G, v) as: the degree of node v in G is ≥ 5 and $\leq |G| - 6$.

Algorithm 2:

```
1 V ← 1,2,3..., n;
2 min_priority_queue ← deg[x] ∀x ∈ V;
3 max_priority_queue ← deg[x] ∀x ∈ V;
4 while ∃ v ∈ V s.t. not P(G, v) do
5 |   V ← V \ {v} ;
6 |   for x in adj[v] do
7 |   |   deg[x] ← deg[x] - 1 ;
8 |   |   update the degree of the resp. vertex in min_priority_queue;
9 |   |   update the degree of the resp. vertex in max_priority_queue ;
  |   end
  end
```

For checking if there exists a vertex such that it does not satisfy the property P, we need only check the vertex with minimum degree and that with the maximum degree. If these do not violate the property P, neither will the other vertices.

For this, we need to maintain two priority queue , one for the maximum and the other for the minimum. We also have to maintain an additional array to store the position of all the nodes in the two priority queues which allows us to decrease a vertex's degree by 1.

Lemma 3.1. *In a graph G, Let S_v be the set of vertices not satisfying property P(G,v).*

If H is a subgraph with all the nodes satisfying the property P then H intersection S_v is null.

Proof. If H intersection S_v is not null then not all nodes in H satisfy the property P.

Case 1 : The intersection has a node whose degree was < 5 in G.

Since H is a subgraph of G and the degree of a node can only decrease if its neighbor is not present in H. So the node will have a degree < 5 in H too. Thus, a contradiction.

Case 2 : The intersection has a node whose degree was $>$ | G | - 6 in G.

Let the initial degree of the node be deg.

Suppose the degree of the node in H is x. Then we have removed deg - x of its neighbors. Thus,

|H| <= |G| - (deg - x)

|H| - |G| + deg <= x

Since, deg > |G| - 6

|H| - |G| + |G| - 6 < x

|H| - 6 < x

So the node will not satisfy the property in H too.

Hence proved

□

Theorem 3.2. *S be the set of nodes that do not belong to any optimal subgraph of G where all nodes satisfy the property P.*

INV *:: At the start of the i^{th} iteration, S contains i vertices and S intersection with any optimal solution is null.*

Proof. **Initialisation :** According to the invariant, S should contain 0 vertices at the start of 0^{th} iteration which is true and S intersection with any set is null.

Maintenance : Let us assume that the invariant is true at the start of i^{th} iteration.

In the i^{th} iteration, if all the nodes satisfy the condition then we break and we reach termination

Else we remove a node that does not satisfy the condition P.

Since, S intersection with any optimal solution is null , so the optimal solution has to be a subgraph of the current graph.

According to lemma, the node which cannot satisfy the property can not be present in a subgraph which does so we can remove the node.

S $\leftarrow$ S U node.

So S before the $(i+1)^{th}$ iteration contains (i+1) nodes and S intersection with H is null.

Termination : In each step either we have find the optimal answer or we remove a node not satisfying the property.

Since the number of nodes are finite, the algorithm must terminate.

Proof the solution is Optimal : Since S has the property that it contains all the nodes that cannot be in an optimal solution.

Let H be an optimal solution.

Therefore | H | <= | G / S |

But our solution outputs a solution whose size is equal to | G / S |

So our solution is also optimal □

Time Complexity :

We have to store the degrees of the vertex in two data structures like priority queue. Min priority queue and Max Priority queue

We have to handle queries of Minimum degree and Maximum degree and decrease degree of a vertex.

There at atmost n iterations and in each iterations we iterate over all the edges of that vertex.

To determine if all vertices satisfies the property or not. We have to do two checks O(log n) for all iteration O(n*logn)

Iterate over all edges in all iterations and decrease the respective degree is O(m*logn).

So the time complexity is O((n+m)*logn)

3.1 Optimal set of invitees **15 / 15**

✓ **- 0 pts** **Correct**

is G \ Sv = H ?

(b) Initially Perform count sort to sort the array A. Friend (Fi) has Age A[i]

Algorithm 3:

```
1  Perform count sort to sort the elements in ascending order ;
2  cur ← 0 ;
3  T ← null ;
4  while cur < n_0 do
5  |   min_age ← A[cur] ;
6  |   start ← cur ;
7  |   cur_grp ← null;
8  |   while start < n_0 ∧ start - cur <= 10 ∧ A[start] <= min_age + 10 do
9  |   |   cur_grp ← cur_grp U {start} ;
10 |   |   start ← start + 1 ;
   |   end
11 |   cur ← start;
12 |   T ← T U {cur_grp};
   end
13 Output | T | ;
```

Lemma 3.3. *F_0 belongs to A so it will be present in a group. Consider the size of the Group = S. Let G' be a group consisting of friends [F_0 , F_1 , ... F_{S-1}] There exists an optimal (ie. of minimum possible size) arrangement say T_0 that contains G' as one of its group*

Proof. Let T be an optimal arrangement and G be the group to which F_0 belongs.

We can convert G into G' by keeping |G| the same.

We have to perform the following steps while G does not become equal to G'

Let F_i be the smallest i such that it does not belong to G. Let F_i belongs to a Group R.

F_i has the smallest i among all members present in group R as members with index 0 to (i-1) are present in G.

Also, max_age in the group R has to be <= F_i + 10.

Also, we also have max_age in the group G<= F_i + 10 and max_age in the group G>= F_i.

We can swap F_i with the max_age in the group G, we will still get a valid solution as

(R \ { F_i}) U {max_age in the group G} is still a valid group as

| max_age in the group R - max_age in the group G | <= 10 and | min_age in the group R - max_age in the group G | <= 10 (which is trivially true by the above conditions)

Let T_0 be this solution and from the above solution | T_0 | = | T |

So T_0 is also an optimal agreement which contains G'

Hence proved.

□

Theorem 3.4. *F_0 belongs to A so it will be present in a group.*

Consider the size of the Group be S.

Let G' be a group consisting of friends [F_0 ,F_1 , ... F_{S-1}]

And A' = A \ G'

Then OPT(A) = OPT(A') + 1

Proof. (I) OPT(A) <= OPT(A') + 1

Suppose T' is the optimal solution of A'

And T be an optimal solution of A

T' U G' is non overlapping by def of G' and T'.

|T' U G'| = |T'| + 1

This implies OPT(A)<= 1 + OPT(A').

Hence proved.

(II) OPT(A) >= OPT(A') + 1

From Lemma 3.3, we know that there exists an optimal solution of A which contains G'

Let T_0 be an optimal solution containing G'

T_0 \ G' is a valid solution for A \ G'

A' = A \ G'

OPT(A') <= | T_0 \ G' |

OPT(A') <= | T_0 | - 1

OPT(A') <= OPT(A) -1

OPT(A) >= OPT(A') + 1

Hence proved.

□

Time Complexity Analysis :

For sorting it takes O(n_0) time as we use count sort.

Then just need to iterate over all the friends in ascending order and assign each of them a group accordingly which takes O(n_0).

So, the total time complexity is O(n_0).

3.2 Dinner tables **5 / 5**

✓ **- 0 pts** **Correct**