

# CS 170 Homework 5

Ninh Do

TOTAL POINTS

**62.5 / 100**

QUESTION 1

**1 Name Distances 3 / 3**

+ 3 Correct

+ 0 Incorrect/empty.

+ 12 Correct.

+ 6 Algorithm on right track, with insufficient runtime.

+ 0 Completely incorrect.

QUESTION 2

**2 Weighted Set Cover 10 / 10**

+ 10 Correct

+ 2.5 Correct algorithm, unsatisfactory proof.

+ 0 Completely incorrect/empty.

QUESTION 3

**3 Ternary Huffman 37.5 / 37.5**

+ 37.5 Completely correct.

+ 28 Correct algorithm, proof missing necessary component (e.g. prove the handle for odd and even # of characters; prove the tree is complete).

+ 18.75 Minor mistake in main idea or insufficient proof.

+ 0 Completely incorrect.

+ 35 minor error

QUESTION 4

**4 Birthday Surprise 0 / 37.5**

+ 37.5 Completely correct.

+ 35 Attempt at a swapping argument with a small error.

+ 28 Correct ordering, proof missing necessary component.

+ 18.75 Correct ordering, insufficient proof.

+ 0 Completely incorrect/empty.

+ 11.25 A correct brute force solution that doesn't use a greedy algorithm.

QUESTION 5

**5 Finding Maul 12 / 12**

Problem #1

## CS170 - Homework # 5

SID #25949105

First name : NINH  $\rightarrow$  NINHNINHNILast name : DO  $\rightarrow$  DODODODODO

Using the algorithm in the textbook gives, for the first letter pair :

N	-	N
-	D	D
Cost :	1	1

Subprob: INHNINHNI      NINHNINHNI      INHNINHNI  
 DODODODODO      DODODODODO      DODODODODO

$$\text{or } E(10, 10) = \min \{ 1 + E(9, 10), 1 + E(10, 9), 1 + E(9, 9) \}$$

In general :

$$E(i, j) = \min \{ 1 + E(i-1, j), 1 + E(i, j-1), 1 + E(i-1, j-1) \}$$

we come up with the table :

N	I	N	H	N	I	N	H	N	I		
V	1	2	3	4	5	6	7	8	9	10	
D	1	1	2	3	4	5	6	7	8	9	10
O	2	2	2	3	4	5	6	7	8	9	10
D	3	3	3	3	4	5	6	7	8	9	10
O	4	4	4	4	4	5	6	7	8	9	10
D	5	5	5	5	5	5	6	7	8	9	10
O	6	6	6	6	6	6	6	7	8	9	10
D	7	7	7	7	7	7	7	7	8	9	10
O	8	8	8	8	8	8	8	8	8	9	10
D	9	9	9	9	9	9	9	9	9	9	10
O	10	10	10	10	10	10	10	10	10	10	10

Since my names don't have any letter matched then any way is the same, say, the diagonal one.

The total cost is 10. (edit distance)

1 Name Distances 3 / 3

+ 3 Correct

+ 0 Incorrect/empty.

## Problem #2 : Weighted Set Cover

### Main idea :

Use greedy algorithm: repeatedly select set  $s_i$  with minimal relative weight  $\tilde{w}_i$ , where the relative weight  $\tilde{w}_i$  is defined as the ratio of weight  $w_i$  and the number of "uncovered" elements in  $s_i$ .

### Pseudo code :

CoverSet =  $\emptyset$  ; CoverElement =  $\emptyset$

while not cover all element :

    Compute # of elements in  $s_i$  but not in CoverElement  $\rightarrow n_i$

    Compute relative weight  $\tilde{w}_i = \frac{w_i}{n_i}$

    Add  $s_i$  w/ minimal relative weight  $\tilde{w}_i$

### Proof of Correctness :

Using the same strategy as in the lecture: Assume at the iteration  $i$ , there is at least  $n-i$  elements left. This is conservative since we may be able to cover more than one element at a time.

The total cost is  $k$ , so the cost of element at most  $\frac{k}{n-i}$  per element in average, otherwise the total cost will exceed  $k$  after cover all  $n-i$  element.

Thus, the cost for the whole process is at most

$$\frac{k}{n} + \frac{k}{n-1} + \frac{k}{n-2} + \dots + \frac{k}{2} + \frac{k}{1} = k \left( \frac{1}{n} + \frac{1}{n-1} + \dots + 1 \right)$$

for  $i = 0 \rightarrow n-1$ . Taylor's expansion goes:  $\ln n = 1 + \frac{1}{2} + \frac{1}{3} + \dots$

$\Rightarrow$  the sum  $< k(\ln n + 1) \approx O(k \log n)$

## 2 Weighted Set Cover 10 / 10

+ 10 Correct

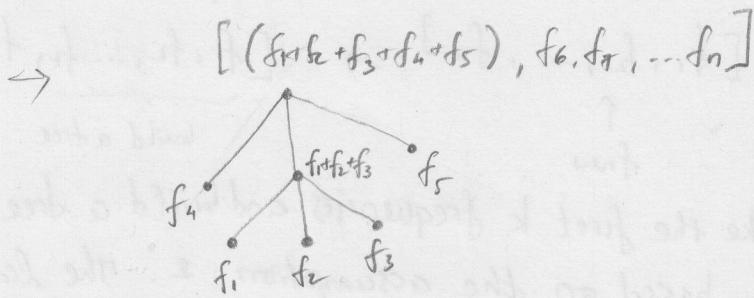
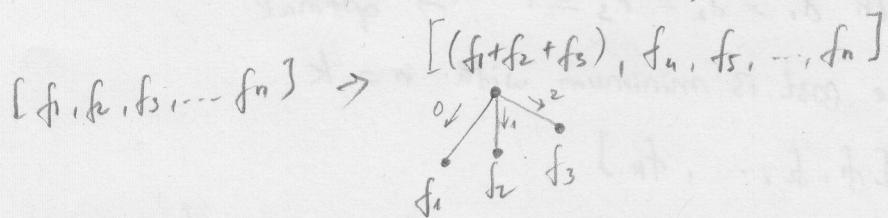
+ 2.5 Correct algorithm, unsatisfactory proof.

+ 0 Completely incorrect/empty.

### Problem #3 Ternary Huffman

Main idea:

Similar to binary Huffman tree, the ternary one uses greedy algorithm: find the three smallest frequencies  $f_1, f_2, f_3$ , make them leaves and add the new frequency  $(f_1 + f_2 + f_3)$  into the priority queue. This frequency is a common parent node of the three old ones. Pop three smallest ones and build the tree up with leaves and node, until the priority queue is empty.



Pseudo code:

$\emptyset = [f_1, f_2, f_3, \dots, f_n]$  priority queue

while  $\emptyset$  not empty :

$f_i = \text{deletemin}(\emptyset); f_j = \text{deletemin}(\emptyset); f_k = \text{deletemin}(\emptyset);$

create a node  $\lambda$  with children  $f_i, f_j, f_k$

$\emptyset \cdot \text{add}(f_i + f_j + f_k)$

## Proof of Correctness:

There are two invariants:

1. Each node has 3 children except the root node that may have 2 children.

2. The low frequency has depth  $d \leq$  that of the high frequency.

The cost of the tree is given by:

$$\text{Cost} = \sum_1^n f_i d_i \quad \text{where } d_i \text{ is depth of } f_i$$

We prove this cost is minimum by using induction:

1. for  $n=3$ :  $\text{cost} = f_1 d_1 + f_2 d_2 + f_3 d_3$

with  $d_1 = d_2 = d_3 = 1 \rightarrow \text{optimal}$

2. Assume the cost is minimum with  $n=k$

$$[f_1, f_2, \dots, f_k]$$

3. for  $n=k+1$ , we can add an arbitrary new frequency:

$$[f_1, f_2, \dots, f_k] \xrightarrow{\substack{\uparrow \\ \text{new}}} [\underbrace{f_1, f_2, \dots, f_k}_{\text{build a tree}}, f_{k+1}]$$

We can take the first  $k$  frequencies and build a tree. This tree is optimal based on the assumption 2. The last frequency  $f_{k+1}$  is leaf having depth 1, since it has highest frequency.

$$\Rightarrow \text{Cost} = \sum_1^{k+1} f_i d_i = \underbrace{\sum_1^k f_i d_i}_{\substack{\text{optimal} \\ (\text{minimum})}} + \underbrace{f_{k+1}}_{\substack{\text{depth } d=1}}$$

Now we see that if we swap  $f_{k+1}$  w/ any other leaf, the cost will increase since:

$$f_{k+1} d_i + f_i \geq f_i d_i + f_{k+1} \Leftrightarrow (f_{k+1} - f_i)(d_i - 1) \geq 0$$

thus, cost is optimal.

### 3 Ternary Huffman 37.5 / 37.5

+ **37.5** Completely correct.

+ **28** Correct algorithm, proof missing necessary component (e.g. prove the handle for odd and even # of characters; prove the tree is complete).

+ **18.75** Minor mistake in main idea or insufficient proof.

+ **0** Completely incorrect.

+ **35** minor error

## Problem #4

### Main Idea:

We sort the tasks in the increasing order of execution time and we choose the ones which have smaller time to do first, i.e. in increasing time order.

If both tasks have the same execution time, we choose the one with higher cost. The basic idea of this breaking tie is if we delay the task with higher cost, the total penalty will add up.

## Proof of Correctness

Tasks :  $T_i$  Credit :  $V_i$  Cost :  $P_i$  Time :  $R_i \quad i=1, n$

The award of task  $i$  is :  $V_i - t_i P_i$ ,  $t_i \geq R_i$

The total award is  $\sum_{i=1}^n (V_i - t_i P_i)$

The credit does not change so to maximize the award, we have to minimize the delay time associated w/ cost.

First, consider short  $\rightarrow$  long task order. The table of penalty having  $R_1 \leq R_2 \leq R_3 \leq \dots \leq R_n$  is as below

$$\text{Day 1} : P_1 R_1$$

$$\text{Day 2} : P_2 R_1 + P_2 R_2$$

$$(T1) \quad \text{Day 3} : P_3 R_1 + P_3 R_2 + P_3 R_3$$

$$\text{Day 4} : P_4 R_1 + P_4 R_2 + P_4 R_3 + P_4 R_4$$

$$\text{Day } n : P_n R_1 + P_n R_2 + P_n R_3 + \dots + P_n R_n$$

In term of matrix :

$$\begin{bmatrix} P_1 & & & \\ P_2 & P_2 & & \\ P_3 & P_3 & P_3 & \\ \vdots & & & \\ P_n & P_n & \dots & P_n \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \vdots \\ R_n \end{bmatrix} =$$

Now we try to swap tasks, if the total penalty increases, don't swap, otherwise swap it. We will see that the rule of swapping is as follow:

$$\text{Day } i : P_i R_1 + P_i R_2 + P_i R_3 + \dots + P_i R_i$$

$$\text{Day } j : P_j R_1 + P_j R_2 + P_j R_3 + \dots + P_j R_j$$

After swapping :

$$\text{Day } i : P_j R_1 + P_j R_2 + P_j R_3 + \dots + P_j R_j$$

$$\text{Day } j : P_i R_1 + P_i R_2 + P_i R_3 + \dots + P_i R_i$$

By adding terms vertically, the total penalty changes from : (before swapping)

$$(I) \quad R_1 \left( \underbrace{\sum_1^n P_k}_{\text{all } P's \text{ term}} \right) + R_2 \left( \underbrace{\sum_2^n P_k}_{\text{missing } P_1} \right) + \dots + R_i \left( \underbrace{\sum_i^n P_k}_{\overset{\uparrow}{\text{missing } P_i \rightarrow P_{i-1}}} \right) + \dots + R_j \left( \underbrace{\sum_j^n P_k}_{\overset{\uparrow}{\text{missing } P_j \rightarrow P_{j-1}}} \right)$$

to : (after swapping )

$$(II) \quad R_1 \left( \underbrace{\sum_1^n P_k}_{\text{missing } P_1 \rightarrow P_{i-1}} \right) + \dots + R_j \left( \underbrace{\sum_i^n P_k}_{\text{missing } P_i \rightarrow P_j} \right) + \dots + R_i \left( \underbrace{\sum_1^n P_k}_{\text{have } P_i} \right)$$

For example , if we swap task 2 and task 4 , the penalty table turns into :

$$\text{Day 1} : P_1 R_1$$

$$\text{Day 2} : P_4 R_1 + P_4 R_4$$

$$\text{Day 3} : P_3 R_1 + P_3 R_4 + P_3 R_3$$

$$(T_2) \quad \text{Day 4} : P_2 R_1 + P_2 R_4 + P_2 R_3 + P_2 R_2$$

$$\text{Day } n-1 : P_{n-1} R_1 + P_{n-1} R_4 + P_{n-1} R_3 + P_{n-1} R_2 + P_{n-1} R_5 + \dots$$

$$\text{Day } n : P_n R_1 + P_n R_4 + P_n R_3 + P_n R_2 + P_n R_5 + \dots$$

If we add terms vertically, we will see exactly the series (II)

We want to minimize series (I). We see that the factors  $(\sum_{k=1}^n P_k)$  are decreasing as it has one term  $P$  less than the factor  $(\sum_{k=1}^n P_k)$  before and one term  $P$  more than the factor  $(\sum_{k=1}^n P_k)$  after. So we have to associate factor  $(\sum_{k=1}^n P_k)$  with the increasing order of factor  $R_i$  from left to right, and this is also the order of executing tasks: favor shortest time tasks.

Quantitatively, take table  $(T_2) - (T_1)$ :

$$ET = R_4 \left( \underbrace{\sum_{k=2}^3 P_k}_{\text{col 2 of } T_2 - \text{col 4 of } T_1} \right) + R_3 \left( \underbrace{P_4 - P_2}_{\text{col 3 of } T_2 - \text{col 3 of } T_1} \right) - R_2 \left( \underbrace{\sum_{k=3}^4 P_k}_{\text{col 4 of } T_2 - \text{col 2 of } T_1} \right) \quad (\text{extra time after swapping})$$

We prove the above Extra-time is always positive given  $R_2 \leq R_3 \leq R_4$ .

Two cases:

Case I : If  $P_4 > P_2$

$$\begin{aligned} \Rightarrow ET &\geq R_4 \left( \sum_{k=2}^3 P_k \right) + R_2 (P_4 - P_2) - R_2 \left( \sum_{k=3}^4 P_k \right) \\ &\geq R_4 (P_2 + P_3) - R_2 (P_2 + P_3) \\ &\geq (R_4 - R_2) (P_2 + P_3) \geq 0 \end{aligned}$$

Case II : If  $P_4 < P_2$

$$\begin{aligned} \Rightarrow ET &\geq R_4 \left( \sum_{k=2}^3 P_k \right) + R_4 (P_4 - P_2) - P_2 \left( \sum_{k=3}^4 P_k \right) \\ &\geq R_4 (P_4 + P_3) - P_2 (P_4 + P_3) \\ &\geq (R_4 - R_2) (P_4 + P_3) \geq 0 \end{aligned}$$

So swapping task 2 & task 4 increasing the total penalty.

Similarly for  $i \leftrightarrow j$  swapping, the ET is a little bit more complicated but pretty much the same.

$$ET_{i \leftrightarrow j} = R_j \left( \sum_{k=1}^{j-1} P_k \right) + R_{i+1} (R_j - R_i) + \dots + R_{j-1} (R_j - R_i) \\ - R_i \left( \sum_{k=i+1}^j P_k \right)$$

We can consider two case to get either

$$ET_{i \leftrightarrow j} \geq (R_j - R_i) \left( \sum_{k=1}^{j-1} P_k \right) \geq 0$$

$$\text{or } (R_j - R_i) \left( \sum_{k=i+1}^j P_k \right) \geq 0$$

This confirm that the task ordered in increasing execution time is optimal.

In the case we have two task with the same time, their swap results in (say, swap Task 3 & task 4 w/  $\begin{cases} R_3 = R_4 \\ P_3 > P_4 \end{cases}$ )

$$ET_{4 \leftrightarrow 3} = R_4 P_3 - R_3 P_4 = R_4 (P_3 - P_4)$$

so we should take  $T_3$  over  $T_4$  so that their swap has  $ET > 0$ .

Final case: if some task are incontinuous, we see that it is even the worst case, since if we execute task  $i$  some days than switch to task  $j$ , then go back to task  $i$ .

the ET is more than the ET of swapping  $i \leftrightarrow j$  as job  $i$  is delayed  $R_j$  day more adding a little bit extra task  $j$  delay compared to the  $i \leftrightarrow j$  swapping case.

#### 4 Birthday Surprise 0 / 37.5

- + **37.5** Completely correct.
- + **35** Attempt at a swapping argument with a small error.
- + **28** Correct ordering, proof missing necessary component.
- + **18.75** Correct ordering, insufficient proof.
- + **0** Completely incorrect/empty.
- + **11.25** A correct brute force solution that doesn't use a greedy algorithm.

## Problem #5 Finding Maul

### Main Idea:

Assume we have both matrix & adjacency list representations of the graph, we first loop through the vertices to find the rank of each vertex, where  $\text{rank}(S)$  is defined as the # of edges the vertex  $S$  has.

Put  $(S: \text{rank}(S))$  into a dictionary or hashmap  $\text{HM}$  if  $\text{rank}(S) \geq 20$ .

and put  $(S': \text{rank}(S'))$  into a set  $SS$ /queue if  $\text{rank}(S') < 20$ .

Go through each vertex  $S'$  of  $SS$  to update the rank of vertex  $S$  in  $\text{HM}_S$ , that is, reduce the rank of a vertex  $S$  by 1 if it shares an edge with vertex  $S'$  in  $SS$ , and move  $S$  to the set if  $\text{rank}(S)$  drops below 20. The  $\text{HM}$  of crime-lords consists of those vertices whose rank  $\geq 20$ .

### Pseudocode:

1. for  $i = 1 \rightarrow n$  :

$r(S_i) = \# \text{ edges of } S_i$

if  $r(S_i) \geq 20$  : add  $(S_i: r(S_i))$  to hashmap  $\text{HM}$

else: add  $(S_i: r(S_i))$  to set  $SS$

2. while ( $SS$  is not empty)

$v = SS.pop()$

for each  $v_i$  is neighbor of  $v$ :

$\text{rank}(v_i) -= 1$

if  $\text{rank}(v_i) < 20$ :

move  $v_i$  into  $S$

return all  $S_i$  with  $r(S_i) \geq 20$  (or return  $\text{HM}$ )

### Proof of Correctness :

Set  $SS$  consists of those vertices whose are not crime-lords for sure.

Hashmap  $HM$  consists of those vertices who are potentially crime-lords.

When we go through  $V$  in  $SS$ , then going through neighbors of  $V$ , we disregard the edges representing "trivial" relationships by reducing  $\text{rank}(S)_{S \in HM}$  by 1, where we define "trivial" relationship as relationship / edge having at least one non-crime lord vertex.

Indeed, assume there is any "trivial" rls of  $S$  in  $HM$  that we do not disregard. This rls / edge connect  $S-S'$  where  $S' \in SS$ . We must have gone through  $S'$  already since we loop until  $SS$  is empty, so we must have reduced  $\text{rank}(S)$  by 1 due to this "trivial" rls.

Thus  $\text{rank}(S)$  represents the # of crime-lord relationship and in  $HM$  left only vertices whose rank is  $\geq 20$ , otherwise they has been moved to the set  $SS$  and been processed.

$\Rightarrow HM$  is the collection of crime-lords.

### Run time :

The for loop has runtime  $O(|V|)$

The while loop has runtime  $O(|E|)$  at worst since we just care about the relationship / edge.

This is based on the constant time to access vertices and update their rank using hashmap  $HM$ .

$\Rightarrow O(|V| + |E|)$

## 5 Finding Maul 12 / 12

- + 12 Correct.
- + 6 Algorithm on right track, with insufficient runtime.
- + 0 Completely incorrect.