CS430 HW5 part 2

Junzle Zheng
A20254389

Problem 4.

(a) Algorithm: Sort the tasks a_i in an increasing Order by their processing time p_i. This gives the minimized average ~~time~~ completion time.

Proof: For simplify, assume $p_i < p_j$ if $i < j$.

Given by algorithm, the processing sequence is:

$$a_1\ a_2\ a_3 \cdots a_{i-1}\ a_i\ a_{i+1} \cdots a_{j-1}\ a_j\ a_{j+1} \cdots a_n \quad (i<j)$$

~~$\text{So } \sum_{k=1}^{n} C_k = P_1 + P_2 + \cdots + P_i + \cdots P_j + \cdots P_n$~~

$$C_k = \sum_{l=1}^{k} P_l \quad 1 \le l \le n.$$

$$C_i = \sum_{l=1}^{i} P_l = \sum_{l=1}^{i-1} P_l + P_i \qquad C_j = C_i + \sum_{l=i+1}^{j} P_l = C_i + \sum_{l=i+1}^{j-1} P_l + P_j$$

Switch a_i, a_j

$$a_1\ a_2\ a_3 \cdots a_{i-1}\ a_j\ a_{i+1} \cdots a_{j-1}\ a_i\ a_{j+1} \cdots a_n$$

$$C_i' = \sum_{l=1}^{i-1} P_l + P_j + \sum_{l=i+1}^{j-1} P_l + P_i$$

$$C_j' = \sum_{l=1}^{i-1} P_l + P_j$$

$$C_i' + C_j' = \sum_{l=1}^{i-1} P_l + P_j + \sum_{l=i+1}^{j-1} P_l + P_i + \sum_{l=1}^{i-1} P_l + P_j$$

$$= \underbrace{\left(\sum_{l=1}^{i-1} P_l + P_i\right)}_{C_i} + \underbrace{\left(\sum_{l=1}^{i-1} P_l + P_i + \sum_{l=i+1}^{j-1} P_l + P_j\right)}_{C_j} + P_j - P_i$$

2

$$c_i' + c_j' = c_i + c_j + (p_j - p_i)$$
$$\geq c_i + c_j \qquad (\text{For } i \leq j \quad p_i \leq p_j)$$

~~Thus~~ Thus, for $i \leq$ $p_i < p_j$ $\mathcal{O}i$

We have $c_i + c_j$ less then any ~~other~~ ~~od~~ other order. This algorithm gives the minimized averager completion ~~tt~~ time.

For running time, ~~the~~ it should be seperated into two parts.

Part 1. sorting: we can use quicksort, which takes $O(n\log n)$

Part 2. Processing: it takes $O(n)$

Thus time complexity is $O(n\log n)$.

(b) As was proven in the previous one, the average completion time is minimized when all tasks are scheduled in an ascending time by their processing time.

1. we sort r_i's in an ascending order.
2. Then launch r_i by its order continously, means when r_i is finished, we lanuch next r_{i+1} after r_i immediately. Meanwhile, ~~we~~ we processes a_i when r_i is done.
3. If r_{i+1} is done but a_i is still running, we stop a_i, put the ~~reamined~~ remained a_i to the sorted task queue (order by p_i) in ascending order), ~~take ou~~ take out the task which requires the minimum processing time from ~~the~~ task queue and process it.

2

Junzle Zheng
A20254387

If a_i is done when r_{i+1} is still runing, ~~we~~ wait till r_{i+1} is done, then process a_{i+1} and ~~tanch~~ launch r_{i+2}.

The key ~~val~~ idea is to run the process in an ascending order by processing ~~a~~ time and remaining processing time.

Time Complexity.

The initial sort will run in $O(n\lg n)$.

For each new ~~take~~ ~~tast~~ task arrival,

worst: $O(n)$ best $O(1)$

~~We~~ We have n task arrival, thus

worst $O(n^2)$; best $O(n)$.

Thus, running ~~the~~ time of this algorithm in worst case is.

$$O(n\lg n) + O(n^2) = O(n^2)$$

Best case is: $O(n\lg n) + O(n) = O(n\lg n)$

3

CS430 HW5 Part 2

Junzle Zhang
A20254389

Problem 5.

(a.) ⓐ First of all, I'd like to apply the algorithm to section 16.5 problem ~~ote~~ as follows:

Task

a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

← sorted by penalty weight in descending order.

Processing Order (slots list)

S (slot)	1	2	3	4	5	6	7
a_i	a_4	a_2	a_3	a_1			
w_i	40	60	50	70			

First, take out a_1, ~~witch~~ which $d_1 = 4$, we ~~seat~~ search in slots list slot4 is unoccupied thus we insert ~~a4~~ a_1 at S_4.
Then we look at a_2. $d_2 = 2$, we search in slots list, Slot_2 is unoccupied, thus we insert a_2 at S_2.
Then we look at a_3. $d_3 = 4$ we search in slots list, slot4 is occupied, but there are slots before Slot4 unoccupied (slot1 and slot3), we insert ~~S~~ a_3 at latest slot, which is ~~S4~~ S_3.
Similiar, we insert a_4 at S_1.

Then we look at a_5. $d_5 = 1$, searching the slots list. S_1 is occupied, and there is no empty slot ~~be for~~ before S_1, which means a_5 will suffer a penalty. $w_5 = 30$. In this situation, I prefer do not insert a_5 to slots list yet, but to another list which named penalty list.

Processing Order (slots list)

S	1	2	3	4	5	6	7
a_i	a_4	a_2	a_3	a_4		a_7	
w_i	40	50	60	70		10	

Penalty list.

P	1	2	3	4	5	6	7
a_i	a_5	a_6					
w_i	30	20					

Similar, we insert a_6 to Penalty list P_2.

Last, we insert a_7 to S_6.

Now, we note that S_5 is unoccupied, then we can move a_7 to S_5.

Processing Order (slots. list)

S	1	2	3	4	5	6	7
a_i	a_4	a_2	a_3	a_1	a_7	a_5	a_6
w_i	40	50	60	70	10	30	20

→ penalty = 30+20 = 50

Finally, we move all items from P to S, ~~fill to~~ fill all empty slots.

This algorithm always deal with a_i with most penalty at first, and put it at last possible slot, leave slots with before this possible slot to other a_i in case they ~~required~~ require a ~~early~~ earlier deadline time.

Thus, this algorithm can guarantee that we can insert as much as we can, and all the tasks with a higher penalty, which leads to a less penalty.

CS430 HW 5 Part 2.

Junzle Zleng
A20254389

Problem 5 (a)

Suppose A is ~~the~~ generated by the algorithm, $p(A)$ is the overall ~~ex~~penalties of A. B is one of the optimal ~~ans~~ answer. The first k jobs $k \in [0, 1, \dots n]$ ~~is~~ ~~equa~~ are same in A, and B, and $k+1$ is different.

If $k = n$, then A and B ~~are~~ are the same, so A is the optimal solution.

If $k < n$, there exist an optimal solution B' which first $k+1$ ~~①~~ tasks ~~are~~ equal to the positions in A.

Proof. B':

Let C has first k tasks time slot same as A, rest slots are zero. Suppose $k+1$ task is at pth time slot in B.

1). If there is time slot in C at or before $k+1$ th tasks deadline d_{k+1}, and it is the latest slot, let it be the qth slot. p is not equal to q. Assume the qth slot in B is task t. we swap p, and q slots in B, gives us B'.

① if $p < q$, the $k+1$ th task will still be executed before deadline. There is no penalty. $p(B') \le p(B)$.

② if $q > p$, $k+1$ th executed after deadline in B, but before in B' penalty ~~is~~ change is $-p(k+1\text{th})$. ~~W~~ Move task t from q to p

the penalty changes is at most $p(t)$. $p(t) \leq p(\cancel{k}+1 \text{ th})$

so $p(B') \leq p(B)$

2) If there is no such slot in C. q is the latest slot which is unoccupied. Assume job in qth slot of B is t. we swap pth and qth in B. yeilds ~~a~~ B'

Since $p<q$. ~~that~~ there is no penalty change. the job t moved forward and the penalty will not increase. so $p(B') \leq p(B)$

From above. we get optimal B' which has first $k+1$ ~~job~~ tasks are the same as in A respectively. The we use inductive method to get total n tasks positions which are equal to A.

Thus A is the optimal solution.

Problem 5.

(b) Algorithms.

Step 1. Initization

Each position 0, 1, 2 ... n is different set and

$F(\{i\}) = i$, $0 \le i \le n$. $\{i\}$ is a set.

a Job List J. $J[i] = 0$, $0 \le i \le n$.

Step 2. ~~For~~ Start from first task in the ordered task list, ~~if~~ assume this task with a deadline d.

Find the set that contains d, ~~for~~ assuming it is the set K. 1. assign this ~~take~~ task to $J[F(K)]$

2. Find the set that contains $F(K) - 1$. call it set L.

3. Union ~~Merge~~ set K and L. $F(\text{Union}(K, L)) = F(L)$

(There is no set K ~~that any no~~ after merge).

repeat step 2 till last task.

Step 3. return J

Time Complexity. There are at most $2n$ Find operations, n make set operations and $n-1$ union operations.

Thus ~~O~~ it is $O(n)$

CS430 HW5 Part 2.

Junzle Zleng

A20254389

Problem 6.

(a) ~~Prof~~ Proof by contradiction.

Assuming A and B are both MST of a graph G.

assuming A has an edge e_1 with least weight. but B ~~do~~ does have not.

Add e_1 to B. then B must have a cycle C that contains edge e_1.

Because A, B are MST of G, e_1 is also the edge with least weight in B. ~~which~~ In addition, C in B ~~have~~ has an edge e_2 ~~wh~~ with a weight greater than e_1.

Removing e_2 will generate a new B ~~w~~ has a less weight than original B and A.

But at the begining we assumed A and B are both MSTs of G. ~~There~~

There is a contradiction. Thus MST of G is unique.

4

(b) Let's consider most exertrem situation. all edges have a same weight, obeviously ~~them~~ ~~all~~ there are many possible MSTs.

If e_1 and e_2 have a same weight and both of them ~~at~~ are in a cycle. we can use any one of them. ~~the~~ this will generate two different MSTs

If e_1 and e_2 have a same weight and not ~~a~~ in a cycle. we choose e_1 or e_2 or both of them ~~be~~ ~~a~~ ~~ules~~ unless if ~~there~~ no cycle will be generated. Wether a different MSTs will be generated or not can not be decided.

Thus, ~~But~~ multiple spanning trees can be generated, but not ~~not~~ always.

We can use Kruskal's algorithm. ~~we~~ When we ~~have~~ ~~got~~ get multiple possible edges with a same weight. we can pick ~~an~~ ~~any~~ ~~one~~ any one of them ~~a~~ ~~u~~ as long as picking it ~~won't~~ won't violate any algrothm and MST rules.

5

CS430 HW5 Part 2

Junzhe Zhang.
A20254389.

Problem 7.

~~Let k_1 = number of 25c.~~

~~k_2 =~~

To use minimum number of coins, we just ~~have to~~ need to use larger value coin as much as possible.

Let k_1, k_2, k_3, k_4 donate the number of 25c, 10c, 5c, 1c.

$$k_1 = \lfloor V/25 \rfloor$$

~~$k_2 = V - \lfloor V/25 \rfloor$~~

$$k_2 = \lfloor (V - k_1 \cdot 25)/10 \rfloor$$

$$k_3 = \lfloor (V - k_1 \cdot 25 - k_2 \cdot 10)/5 \rfloor$$

$$k_4 = \lfloor (V - k_1 \cdot 25 - k_2 \cdot 10 - k_3 \cdot 5)/1 \rfloor$$

Pseudocode:

```
Charge (V, d)          d is array contains denominations
{  ~~for (i = 1 to d.size)~~
   change [ ];            declare array to hold number of changes.
                          for each denomination.
   for (i = 1 to change.size)
       change[i] = 0;
   for (i = 1 to d.size)
   { if V > 0
       { change[i] = V / d[i];
         V = V % d[i]
       }
```

P6

→ backpage

```
else
    break;
}
reture change[ ];

}
```