

CS430 HW 5 part 2.

Junze 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 of completion time.

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

Given by algorithm, the processing sequence is:

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

~~$$\text{So } \sum_{k=1}^n C_k = p_1 + p_2 + \dots + p_i + \dots + p_j + \dots + p_n$$~~

$$C_k = \sum_{l=1}^k p_l, \quad 1 \leq k \leq n.$$

$$C_i = \sum_{l=1}^i p_l, \quad C_j = C_i + \sum_{l=i+1}^j p_l = C_i + \sum_{l=i+1}^{j-1} p_l + p_j$$

$$= \sum_{l=1}^{i-1} p_l + p_i$$

Switch a_i, a_j

$a_1, a_2, a_3, \dots, a_{i-1}, a_j, a_{i+1}, \dots, a_i, \dots, a_n$

$$C_i' = \sum_{l=1}^{i-1} p_l + p_j + \sum_{l=i+1}^{j-1} p_l + p_i$$

$$C_j' = \sum_{l=1}^{i-1} p_l + p_j$$

$$C_i' + C_j' = \sum_{l=1}^{i-1} p_l + p_j + \sum_{l=i+1}^{j-1} p_l + p_i + \sum_{l=1}^{i-1} p_l + p_j$$

$$= \underbrace{\left(\sum_{l=1}^{i-1} p_l + p_i \right)}_{C_i} + \underbrace{\left(\sum_{l=1}^{i-1} p_l + p_i + \sum_{l=i+1}^{j-1} p_l + p_j \right)}_{C_j} + p_j - p_i$$

①

Q

$$C_i' + C_j' = C_i + C_j + (P_j - P_i) \\ \geq C_i + C_j \quad (\text{For } i \leq j \quad P_i \leq P_j)$$

~~Thus~~ Thus, for $i < j$ $P_i < P_j$

We have $C_i + C_j$ less than any ~~other~~ other order. This algorithm gives the minimized average completion time.

~~For~~ For running time, ~~the~~ it should be separated 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 continuously, means when r_i is finished, we launch next r_{i+1} after r_i immediately.
Meanwhile, we process 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 ~~remained~~ remained a_i to the sorted task queue (order by p_i in ascending order), ~~take~~ take out the task which requires the minimum processing time from ~~the~~ task queue and process it.

②

Junze Zheng
A2025438P

If A_i is done when r_{i+1} is still running, ~~the~~ wait till r_{i+1} is done, then process A_{i+1} and ~~then~~ launch r_{i+2} .

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

Time Complexity.

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

For each new ~~task~~ 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.

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

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

(3)

CS430 HW 5 Part 2.

Junze Zeng
A2025438P

Problem 5 (a)

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

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

If $k < n$, there exist an optimal solution B' which first $k+1$ 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 p th 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 q th slot. p is not equal to q . Assume the q th 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. $p(B') \leq p(B)$.

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

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

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 q th slot of B is t .

We swap p th and q th in B . yields a B'

Since $p < q$, ~~the~~ 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 $k+1$ jobs tasks are the same as in A respectively. Then 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 \oplus Initialization

Each position $0, 1, 2 \dots n$ is different set and

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

a Job list J . $J[i] = 0, 0 \leq i \leq n$.

Step 2 \oplus 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 , assign this 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 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 it is $O(n)$

CS430 HW5 Part 2.

Jungle Zeng

A2025438f

Problem 6.

(a) 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 does not have it.

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 . ~~where~~ In addition, C in B ~~has~~ has an edge e_2 with a weight greater than e_1 .

Removing e_2 will generate a new B with a less weight than original B and A .

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

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

(b) Let's consider most extreme situation. all edges have a same weight, obviously ~~then~~ all there are many possible MSTs.

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

If e_1 and e_2 have a same weight and not in a cycle, we choose e_1 or e_2 or both of them ~~but unless unless if there~~ no cycle will be generated. ~~Whether~~ 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 ~~find~~ get multiple possible edges with a same weight, we can pick ~~any one~~ any one of them ~~as long as~~ as long as picking it ~~doesn't~~ ^{won't} violate any algorithm and MST rules.

(5)

CS430 HW5 Part 2

Junze Zhang.

A2025438P

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 denote the number of 25c, 10c, 5c, 1c.

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

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

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

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

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

Sup: Pseudocode:

Change (V, d)

d is array contains denominations

~~for $i = 1$ to $d.size$~~

change[]; ~~;~~

declare array to hold number of changes.

for ($i = 1$ to change.size)

for each denomination.

change[i] = 0.

for ($i = 1$ to d.size)

{ if $V > 0$

{ change[i] = $V / d[i]$;

$V = V \% d[i]$

}

→ backpage

(P6)

```

else
    break;
}
return charge[ ];
}

```



10