Homework #2

Problem 5 - Generals.io (Hand-Written) (25 points)

Note: In this problem, you are not allowed to write pseudocode. Please explain your algorithm in words.

As a general of CSIE kingdom, you have to defend your country from invasion. Specifically, you anticipate N waves of attack from the enemies, and the i-th wave will arrive on the day T_i. As a part of the defense measures, some military contractors proposed M weapon construction plans. The i-th construction can be done on the day t_i and costs p_i dollars. We need exactly one weapon to successfully defend against one wave of attack. Also, after each wave of attack, the used weapon will be destroyed and is no longer usable under the heavy gunfire.

Now, your goal is to minimize the cost for constructions while keeping the country safe from all hostility. (T_i and t_i are all distinct. $\langle T_i \rangle$ is in an ascending order. Also, you can assume that there exists at least one way to successfully defend against all attacks, which leads to $N \in O(M)$.)

(a) (2 points)

Given $N = 4,\ M = 7,\ T = [9, 21, 28, 32],\ t = [10, 3, 16, 39, 25, 31, 5],\ p = [4, 5, 6, 1, 2, 7, 3]$, find the optimal **choice**.

$t = 10, 3, 25, 5$

$p = 4, 5, 2, 3$

Choose $\{1, 2, 5, 7\}$-th constructions.

Total cost: 14

(b) (7 points)

Design an $O(M \log M)$ greedy algorithm to find the minimum cost and prove its correctness (including the greedy-choice property) and time complexity.

Greedy choice property

1. sort p's value along with t, the complexity is $O(M \log M)$. That is, p is in an ascending order.
2. check if the t_i is $< T.back()$
    a. if yes, push i-th construction into set S, and p_i (local OPT) is one of our choices. $T.popback()$.

b. if no, erase this index (can not defend any remaining waves).

c. Loop this step until $T.size() = 0$.

We at most check all the element in $t(p)$ one time, so the complexity is $O(M)$.

3. S is our choices. (Global OPT)
4. Total complexity $O(M \log M)$

OPT substructure

$$S_n = \min_j(p_j + S_{n-1}), \text{for } t_j < T_n$$

If Case k-th is in OPT S_n, n-th wave is the current wave.

OPT\k-th is an OPT of S_{n-1}

Proof

Greedy choice: select the smallest p_i where $t_i < T.back()$.

Prove via contradiction

Assume that there is no OPT including p_k, which is the smallest p_i where $t_i < T.back()$.

Choose p_x, for $x \neq k$, then either

case1: $p_x >= p_k$, and the total cost increases or remains the same. We can always choose k-th instead.

case2: $t_x > T.back()$ we can't choose it because we can not defend the waves.

Sadly, you find out that your country is too small to accommodate so many constructions. Thus, the number of weapons that are ready for service must not exceed K at any time.

(c) (2 points)

Given $N = 5,\ M = 9,\ K = 2,\ T = [9, 21, 31, 39, 45],\ t = [25, 10, 16, 43, 50, 5, 28, 32, 3],\ p = [9, 1, 2, 8, 7, 3, 6, 5, 4]$, find the optimal **choice**.

$t = 10, 16, 5, 28, 32$

$p = 1, 2, 3, 5, 6$

Choose $\{2, 3, 6, 7, 8\}$-th constructions.

Total cost: 17

(d) (7 points)

Design an $O(MK + M \log M)$ dynamic-programming algorithm to find the minimum cost and prove its correctness and time complexity.

1. Sort t's value along with p $O(M \log M)$
2. **Dynamic Programming strategy**
   a. $dp[i][j]$ for the minimum cost on time i and having j weapons constructed, where for all$\text{num of waves hitted} \leq j \leq \min(K + \text{num of waves hitted}, N)$.
   b. Building dp strategy:
      i. $dp[0][0] = 0$,
      ii. **def Check**: If $dp[i][j]$ doesn't exist a valid j, then $dp[i][j] = \infty$.
      iii. **If** $i = T_{\text{weapon}}$:

$$dp[i][j] = \min(dp[i-1][j-1] + p_i, dp[i-1][j])$$

      and **Check().**
      iv. **Else if** $i = T_{wave}$: **Check().**
      v. **Else** $dp[i][j] = dp[i-1][j]$: and **Check().**
   c. total cost = $dp[T_{\text{last}}][t.size()]$.
3. Because we only run through a part of this dp, and the size of the part is $O((M+N)K) = O(MK)$, consider
   a. $M + N$events.
   b. the range of j in each row is $\leq cK$ for some c, because $\text{num of waves hitted} \leq j \leq \min(K + \text{num of waves hitted}, N)$.
   c. $N = O(M)$

   so the complexity of building this dp is $O(MK)$.
4. In conclusion, the total complexity is $O(MK + M \log M)$.

Back tracking

From $dp[T_{\text{last}}][t.size()]$, check when $i = T_{\text{weapon}}$ if $dp[i][j] = dp[i-1][j-1]$, record i and then we can know when we buy the weapons.

Proof

Prove via contradiction:

Prove it's in OPT:

If $i = T_{\text{weapon}}$:

$$dp[i][j] = \min(dp[i-1][j-1] + p_i, dp[i-1][j])$$

Case 1: $dp[i-1][j-1] + p_i < dp[i-1][j]$, and $dp[i-1][j-1]$ is in OPT

Assume $d[i-1][j]$ is in OPT'.

The total cost of $OPT < OPT'$, we can always choose buying p_i instead.

Case 2: $dp[i-1][j-1] + p_i > dp[i-1][j]$, and $dp[i-1][j]$ is in OPT

Assume $d[i-1][j-1]$ is in OPT'.

The total cost of $OPT < OPT'$, we can always choose not buying p_i instead.

Prove it won't violate the rule:

In the Algorithm, every time we update the dp, we check $\text{num of waves hitted} \leq j \leq \min(K + \text{num of waves hitted}, N)$. Thus, at every time point, the remaining weapons always $\leq K$ and ≥ 0.

(e) (bonus 5 points)

Design an $O(M \log M)$ greedy algorithm to find the minimum cost and prove its correctness (including the greedy-choice property) and time complexity.

1. Sort t's value along with p. $O(M \log M)$

Clever as you are, you come up with the idea of requisitioning private residence to resolve the land-size restriction. Now, you can construct as many weapons as you want, but you have to pay the residents 1 dollar per day per weapon as long as the weapon is ready for service.

(f) (7 points)

Design an algorithm to find the minimum cost that runs as fast as possible and prove its correctness and time complexity.

Greedy choice property

1. update p's value: $p'_i = p_i - t_i$
2. sort p's value along with t (just 1 time), the complexity is $O(M \log M)$. That is, p is in an ascending order.
3. check if the t_i is $< T.back()$
   a. If yes, push i-th construction into set S, and p'_i (local OPT) is one of our choices. $T.popback()$.
   b. if no, erase this index (can not defend any remaining waves).
   c. Loop this step until $T.size() = 0$. at

   We at most check all the element in $t\&p'$ one time, so the complexity is $O(M)$.
4. S is our choices. (Global OPT)
5. Total cost: $\sum p_i$(original)$+(\sum T - \sum t_i)$(idle time) for all i in set S.
6. Total complexity $O(M \log M)$

OPT substructure

$$p'_i = p_i - t_i$$
$$S_n = \min_j(p'_j + S_{n-1}), \text{for } t_j < T_n$$

If Case k-th is in OPT S_n, n-th wave is the current wave.

OPT\k-th is an OPT of S_{n-1}

Proof

Greedy choice: select the smallest p'_i where $t_i < T.back()$.

Prove via contradiction

Assume that there is no OPT including p_k, which is the smallest p'_i where $t_i < T.back()$.

Choose p'_x, for $x \neq k$, then either

case1: $p'_x >= p'_k$, and the total cost increases or remains the same. We can always choose k-th instead.

or

case2: $t_x > T.back()$ we can't choose it because we can not promise that we can defend all the waves.

Problem 6 - $(N+1)$-Legged Race (Hand-Written) (25 points)

The ADA Kingdom is about to hold an $(n+1)$-legged race ! Everyone in the ADA Kingdom has to participate in this race, and each person has his/her own velocity v_i.

To compete in the race, all participants form several teams. There is **no limit** to the number of people in a team. That is, you can form a team with only one person or even all people in the kingdom. The **speed difference of a group**: δ, is defined as the speed of the fastest person in that group minus the speed of the slowest person in that group (i.e., for a group g, $\delta(g) = \max_{i \in g} v_i - \min_{i \in g} v_i$). Note that a team consisting of only one person has a speed difference of 0. The **total speed difference** is the sum of the speed differences in all groups (i.e., $\sum_g \delta(g)$). You can assume that $v_i \neq v_j$ if $i \neq j$. (i.e. There are no two people having the same velocity.)

Please solve each subproblem based on the above premises. Also, you can assume that basic integer operations $(+, -, \times, \div)$ can be done in $O(1)$ time and you don't need to worry about integer overflow.

In all subproblems, We recommend you to explain the algorithm in words. If you want to answer it in pseudo code, make sure it is correct, readable (comment or explain if needed), and **less than 35 lines**.

(a) (4 points)

The king of the ADA kingdom wants to separate the participants into teams for the $(n+1)$-legged race, but he does not know how many groups to divide the people into.

Please design an algorithm which runs in $O(N \log N)$ time to calculate the **minimum total speed difference** for every i from 1 to N, if he separates all people into i groups. Briefly explain the correctness and time complexity of your algorithm.

Greedy choice property

1. Sort the value of v_i so v is in ascending order. The complexity = $O(N \log N)$
2. Calculate the difference between v_i and v_{i+1} for all i, let it be d_i. The complexity = $O(N)$

3. We can seperate all people into i groups by dividing v $i-1$ times. Also, the speed difference of a group is $\sum_{i \in g} d_i$.

   For example, if we have $v = \{5, 8, 9, 12\}$, we can divide them like $g_1 = \{5\}, g_2 = \{8, 9\}, g_3 = \{12\}$, and the speed difference of them is 0, 1, and 0. The total speed difference = (3+1+3)-(3+3)=1.

4. We can sort d_i's from large to small, the complexity = $O(N \log N)$, and pick the biggest = d_k, seperate the group into 2 group with $g_1 = \{v_1, v_2, ..., v_i\}$ and $g_2 = \{v_{i+1}, .., v_{N-1}, v_N\}$, and the total speed difference decreases by d_k. Remove d_k from d

5. If we require i groups, then we can do the above strategy $i-1$ times and then get i groups. And the total speed difference = $v_{max} - v_{min} - \sum_{j=1}^{i-1} d_j$. Because we always pick the i with biggest d_i to seperate, the total speed difference should be minimun.

6. Total complexity $O(N \log N)$

Greedy choice

Choose the i with biggest d_i, and then cut v at position i.

OPT substructure

$$D_n = \min_j(D_{n-1} - d_k), \text{choose } d_k, \text{which is biggest } d.$$

If Case k-th is in OPT D_n. n is the group number.

OPT\k-th is an OPT of D_{n-1}

Proof

Prove via contradiction

Assume there is no OPT including d_k, which is the biggest d.

Case 1: If d_x is in OPT and $d_x < d_k$,

then $g_1 = \{v_1, v_2, ..., v_x\}$ $g_2 = \{v_{x+1}, ..., v_{N-1}, v_N\}$

Becuase v is in ascending order, so difference of

$g_1 = v_x - v_1 = \sum_{i=1}^{x-1} d_i$, $g_2 = v_N - v_{x+1} = \sum_{i=x+1}^{N-1} d_i$

The total difference = $\sum_{i=1}^{N-1} d_i - d_x$. However, if we choose d_k instead, then the total difference = $\sum_{i=1}^{N-1} d_i - d_k$ which is smaller.

Case 2: If d_x is in OPT and $d_x > d_k$,

it's not valid because d_k must be the largest one.

(b) (5 points)

The king wants to divide all participants into **two unordered groups**: A and B, and conducts an $(n+1)$-legged race such that the total speed difference of these two groups is **not greater than** a given K (i.e., $\delta(A) + \delta(B) \leq K$).

Please design an algorithm that runs in $O(N \log N)$ to calculate how many different ways he can divide the participants. Note that two ways are different if and only if there exists two people a, b such that they are in the same group in one way but in different ones in another. Briefly explain the correctness and time complexity of your algorithm.

1. Sort the value of v_i so v is in ascending order. The complexity = $O(N \log N)$
2. Choose v_i and v_j s.t. the total difference = $(v_{max} - v_i) + (v_j - v_{min}) \leq K \rightarrow (v_{max} - v_{min}) - K \leq (v_i - v_j)$, and $\delta(A) = v_{max} - v_i,$ and $\delta(B) = v_j - v_{min}$
3. Goal: find (v_i, v_j) satisfying the above rule.
    a. Case1: $v_i > v_j$, v_i must equal to v_{j+1} otherwise, the v between v_i , v_j can't be groupped.

        In this case, we can check all v and find satisfied pair.

    b. Case2: $v_i < v_j$, we can find all satisfied pairs by checking all v_i $O(N)$,and use binary search to find v_j $O(\log N)$. If v_j is staisfid and v_{j-1} is not, then all the case is v_k for $k >= j$.

        When we've done finding these pairs the solution will follow the following rules

        i. v_k for all $k < j$ is in group B

        ii. v_k for all $k > i$ is in group A

        iii. v_k for all $j < k < i$ can either be in group A or in B → 2^{j-i+1} choices.
4. Finally, we can calculate number of all the possible solution. Complexity: $O(N \log N)$
5. $\text{Num of ways} = N - 1(\text{Case1}) + \sum_{(i,j),\text{ for j > i}} 2^{j-i+1}(\text{Case2})$

6. **Special Case:** If $v_{max} - v_{min} \leq K$, all people can be in one group, and then select one and put him into another group.

Proof

Prove via contradiction.

Case1: $v_i > v_j$, v_i must equal to v_{j+1}

Assume we don't select the proper $v_i = v_k$ for $k > 1$.

then $\delta(A) = v_{max} - v_k$ $\delta(B) = v_j - v_{min}$

Now j must be $k-1$ because v_{k-1} is not in group A

However, then v_i = v_{j+1} → contradiction.

Case2: $v_i < v_j$

1. v_k for all $k < j$ is in group B, otherwise $v_k = v_i$ →contradiction
2. v_k for all $k > i$ is in group A, otherwise $v_k = v_j$ →contradiction
3. v_k for all $j < k < i$ is in group B can either be in group A or in B → 2^{j-i+1} choices, otherwise → 1. or 2.

(c) (1 point)

Please list **all different ways** for grouping **four people** into any number of groups without further restriction.

You can use i to denote the i-th person, e.g., $\{\{1,2,3,4\}\}, \{\{1\},\{2\},\{3\},\{4\}\}, \ldots$

$\{\{1,2,3,4\}\},$
$\{\{1,2,3\},\{4\}\}, \{\{1,2,4\},\{3\}\}, \{\{1,3,4\},\{2\}\}, \{\{2,3,4\},\{1\}\},$
$\{\{1,2\},\{3,4\}\}, \{\{1,3\},\{2,4\}\}, \{\{1,4\},\{2,3\}\}$
$\{\{1,2\},\{3\},\{4\}\}, \{\{1,3\},\{2\},\{4\}\}, \{\{1,4\},\{2\},\{3\}\}, \{\{2,3\},\{1\},\{4\}\},$
$\{\{2,4\},\{1\},\{3\}\}, \{\{3,4\},\{1\},\{2\}\}$
$\{\{1\},\{2\},\{3\},\{4\}\}$

(d) (7 points)

Design an algorithm using **dynamic programming** with time complexity $O(N^2)$ to solve the following problem:

Without any other restriction, **how many ways** are there to group N people into any number of groups? We recommend writing your solution in the following form:

1. Define the **subproblem** of dynamic programming and define **each variable** in it. (2 points)
2. Write down the **recurrence relationship** between subproblems. (2 points)
3. Briefly explain the **correctness** and both **time** and **space** complexity of your algorithm. (3 points)

1. **subproblem $\rightarrow i$ people being groupped into j groups.**

   let S_j^i denotes number of ways to group i people into j groups, and there are N people:

$$\text{total} = \sum_{j=1}^{N} S_j^N$$

$$\begin{cases} S_j^i = 0 \text{ , for all } i \text{ or } j \ = 0 \ || \ i < j \\ S_j^i = 1 \text{ , for all } i \text{ or } j \ = 1 \\ S_j^i = S_{j-1}^{i-1} + j \times S_j^{i-1} \end{cases}$$

   That is, $i-1$ people being groupped, whenever another person joins, either he can be put into a new group where only he's in it, or be put into one of original groups and there are j choices.

2. $dp[i][j]$
   a. i-th row : i people being grouped
   b. g-th column: j groups

$$\begin{cases} dp[i][j] = 0 \text{ , for all } i \text{ or } j \ = 0 \ || \ i < j \\ dp[i][j] = 1 \text{ , for all } i \text{ or } j \ = 1 \\ dp[i][j] = dp[i-1][j-1] + j \times dp[i-1][j] \end{cases}$$

3.
   a. Time complexity: With Bottom-Up method, we need $O(N^2)$ to build the dp table, and $O(N)$ to find $\text{total} = \sum_{j=1}^{N} S_j^N$. Thus, total time complexity is $O(N^2)$
   b. Space complexity: dp table $\rightarrow O(N^2)$.
   c. Proof:

$i-1$ people being groupped, whenever another person joins,

1. either he can be put into a new group where only he's in it. $dp[i-1][j-1]$
2. or be put into one of original groups and there are j choices. $dp[i-1][j-1] \times j$

By the way, in the dp table, we have covered all the case of $i < j$ and $dp[i][j] = dp[i-1][j-1] + j \times dp[i-1][j]$ covered the left-down triangle of the table. Thus, there should be no case being ignored.

(e) (1 point)

There are four people with their velocity $1, 2, 3, 5$, respectively. Assuming the total speed difference cannot be greater than **3**, list **all different ways** of grouping them into any number of groups.

You can use a person's velocity to represent this person, e.g., $\{\{1,2\},\{3,5\}\}, \{\{1\},\{2\},\{3\},\{5\}\}, \ldots$

$\{\{1,2,3\},\{5\}\}, \{\{2,3,5\},\{1\}\}, \{\{1,2\},\{3,5\}\}$
$\{\{1,2\},\{3\},\{5\}\}, \{\{1,3\},\{2\},\{5\}\}, \{\{2,3\},\{1\},\{5\}\},$
$\{\{2,5\},\{1\},\{3\}\}, \{\{3,5\},\{1\},\{2\}\}$
$\{\{1\},\{2\},\{3\},\{5\}\}$

(f) (7 points)

Design an algorithm using **dynamic programming** with time complexity $O(N^2K)$ to solve the following problem:

The king limits the total speed difference to be not greater than K. Given the velocity v_i of every person, **how many ways** are there to group N people into any number of groups? We recommend write your solution in the form below:

1. Define the **subproblem** of dynamic programming and define **each variable** in it. (2 points)
2. Write down the **recurrence relationship** between subproblems. (2 points)
3. Briefly explain the **correctness** and both **time and space** complexity of your algorithm. (3 points)

1. Create a $dp[i][j][k]$, denotes