CS300, KAIST
Fall 2020
Homework 5
Due: 2020/11/20 (Fri) 17:59:99 KST

TA: Jinyoung Oh (jinyoungoh@kaist.ac.kr)

- I gave 0 points for solutions that I think wrong or cannot understand. If you believe your algorithm is right, send an email to me **with Python implementation** after testing with multiple test cases.

1. (40 points) There is a sequence S that consists of n integers, (i.e., $a_1, a_2, \cdots, a_n$). Given the sequence, you can pick any adjacent integers and multiply them to *synthesize* a new sequence. However, each integer could be multiplied with only one integer. For example, when a sequence is '$1, 1, 2, 3$', '$1, 1 \times 2, 3$' is a valid *synthesized* sequence but '$1 \times 1 \times 2, 3$' is not. With this multiplication, you need to maximize the sum of a sequence. For example, in case of '$1, 1, 2, 3$', the answer is '$1, 1, 2 \times 3$'. Give an **efficient algorithm to find such a *synthesized* sequence that maximizes the sum of its elements**, starting from a sequence S with n integers, (i.e., $a_1, a_2, \cdots, a_n$). You don't have to find all cases with the same (largest) sum.

   To get full credits from this problem, your algorithm should at least as efficient as TA's sample solution.

   (a) (5 points) What is the *synthesized* sequence that maximizes the sum of its elements, starting from $S := (1, 3, 2, 1, 2, 4, 1)$?

   Solution. $(1, 3 \times 2, 1, 2 \times 4, 1)$ or $(1, 6, 1, 8, 1)$

   (b) (5 points) Define the entries of your table for DP in words. You can use multiple tables if you wish. (e.g., $T(i, j, \cdots)$ is ... and $V(i, j, k)$ is ...)

   Note.
   - Time complexity of your algorithm should be $O(n)$. To achieve so, the easiest way is: memoizing the *partial sum* as well as '*synthesized* ***sub****sequence*'.
   - If you memoized only *partial sum*, you must provide $O(n)$ time algorithm to reconstruct the '*synthesized sequence*' with *partial sum*s in your pseudo-code. If you did not, you will be deducted **0,3,5,7,5** points from each subproblem. Else if you provided a wrong algorithm, you will be deducted **0,2,3,5,2** points from each subproblem.
   - If you memoized '*operation you did for each entry*' or '***accumulated*** *operation****s*** *you did for each entry*', you must provide $O(n)$ time algorithm to reconstruct the '*synthesized sequence*' in your pseudo-code. Otherwise, you will be deducted **0,2,3,5,2** points from each subproblem.
   - If you did not memoized *partial sum*, there is no way to compute *partial sum* within $O(1)$ time. The best time complexity you can achieve in this way is $O(n^2)$, so you will be deducted **0,0,0,5,5** points.
   - If your algorithm does not take care about negative elements, you will be deducted **0,0,2,3,0** points.

   Solution 1.
   - $T(i)$:= the sum of the synthesized sequence that maximizes its sum, starting from the subsequence $S[0 : i]$ (using up to a_i).
   - $S(i)$:= the synthesized sequence that maximizes its sum, starting from the subsequence $S[0 : i]$ (using up to a_i).

   Solution 2.
   - $T(i)$:= the sum of the synthesized sequence that maximizes its sum, starting from the subsequence $S[0 : i]$ (using up to a_i).
   - $O(i)$:= the last operations you took to achieve $T(i)$.

(c) (10 points) Define the relationship between entries in your table for DP, and briefly justify. (e.g., $T(i) = T(i-1)$ because I want it.)

Note.

- Each table out of the two tables counts 5-points.
- If your pseudo-code is based on the relationships based on the answers for this subproblem and the pseudo-code is correct, you can get full credits for this subproblem even though your answer for this problem is not stated below.

Solution 1. In this case, you need to reconstruct the synthesized sequence properly in your pseudo-code, and it would be challenging.

- $T(i) = \max\left(T(i-1) + a_i, T(i-2) + a_{i-1} \times a_i\right)$

Solution 2.

- $T(i) = \max\left(T(i-1) + a_i, T(i-2) + a_{i-1} \times a_i\right)$
- $S(i) = \begin{cases} (S(i-1),\ a_i) & \text{if } T(i-1) + a_i \geq T(i-2) + a_{i-1} \times a_i \\ (S(i-2),\ a_{i-1} \times a_i) & \text{if } T(i-1) + a_i \leq T(i-2) + a_{i-1} \times a_i \end{cases}$
  (you can put 'equal' case on either side.)

Solution 3. In this case, you need to reconstruct the synthesized sequence properly in your pseudo-code. To do so, you need to traverse $O(i)$ backward and ignore $O(i-1)$ if $O(i)$ was `multiplication`.

- $T(i) = \max\left(T(i-1) + a_i, T(i-2) + a_{i-1} \times a_i\right)$
- $O(i) = \begin{cases} \texttt{no-op} & \text{if } T(i-1) + a_i \geq T(i-2) + a_{i-1} \times a_i \\ \texttt{multiplication} & \text{if } T(i-1) + a_i \leq T(i-2) + a_{i-1} \times a_i \end{cases}$
  (you can put 'equal' case on either side.)

(d) (15 points) Write pseudo-code for your algorithm.

Note.

- If your base case is wrong, you will be deducted 5 points.
- You must interpret your relationship into pseudo-code properly. Otherwise, you will be deducted 3 points per error.
- The output of your algorithm must be a *synthesized sequence* rather than the sum of its elements.

(e) (5 points) Analyze time-complexity of the algorithm.

Note.

- To get credits, your pseudo-code should be correct or have only minor errors.
- I gave full points for those who stated "the answer is $O(n^2)$ since copying string takes $O(n)$ time" as long as your algorithm is right. Of course string manipulation may take long time in the real world, so you have to reconstruct synthesized sequence only once at the time after the DP process is done rather than always copying the synthesized sequence. However, it's too harsh, so let's just assume it takes $O(1)$ time in this question.

Solution. $O(n)$

2. (60 points) A country named *the United States of Wonderland(USW)* uses a method similar to that of the United States of America to elect its president (indirect election). There are n states in the USW and each state i has p_i people who can vote for their preferred candidate. Each state selects a single candidate by Popular Vote (the candidate who gets the most votes wins in the state i). When a candidate wins in a state i, he or she can nominate e_i *electors* who pledged to vote for him or her at the final election. Unlike the USA, the USW is not democratic, so e_i is not proportional to p_i. The candidate who gets the most votes from the *electors* becomes the president (to make your life easier, let's assume there is no 'faithless elector'.)

   Currently, there are only two candidates in the election, Mr. Cake from Birthday Party and Ms. Wine from Surprise Party. You are working at Birthday Party, and your Party's candidate Mr. Cake wants to reduce the campaign cost. Obviously, the easiest way to reduce the cost should be focusing on **the smallest number of people who can make him win.** Give an algorithm to find out the smallest **total number of people** who can make him win in the election.

   (a) (10 points) Define the entries of your table for DP in words. You can use multiple tables if you wish. (e.g., $T(i, j, \cdots)$ is ... and $V(i, j, k)$ is ...)

   Solution 1. $T(i, j) :=$ the smallest number of people who can bring **exactly** (or 'at least', depending on your recurrence) j electors for Mr. Cake when we consider only state $1, 2, \ldots, i$.

   This is a Knapsack-like problem:

   - In the original Knapsack problem, the constraint was: the sum of picked items' weights should be equal or smaller than the capacity of a knapsack. In this problem, the contraint is: the sum of picked states' (i.e. where Mr. Cake won) number of electors should be larger than the half of the total number of electors (i.e. $\sum e_i$).
   - In the original Knapsack problem, we had to maximize the sum of picked items' value. In this problem, we need to minimize the sum of picked states' population.

   Solution 2. $K(i, j) :=$ (assuming we count **up to only** $\lceil \frac{p_i+1}{2} \rceil$ **for each state rather than** p_i) the largest number of people who can bring **at most** j electors for Ms. Wine when we consider only state $1, 2, \ldots, i$.

   We can use Knapsack to filter out the most inefficient states where we can give up the campaign. I know that it is difficult to define $K(i, j)$ in words, so I'll give full credits even if your definition is ambiguous as long as your pseudo-code is (almost) correct. However, if your definition is ambiguous and your pseudo-code has any crucial error, you cannot get any points.

   (b) (15 points) Define the relationship between entries in your table for DP, and briefly justify. (e.g., $T(i) = T(i-1)$ because I want it.)

   Solution 1. $T(i, j) = \min\left(T(i-1, j), T(i-1, j-e_i) + \lceil \frac{p_i+1}{2} \rceil\right)$.
   The first term stands for the case where state i is cost-inefficient while the latter stands for the case where state i is cost-efficient.

   - As I mentioned (on KLMS) you can assume every p_i is an odd number, $\lceil \frac{p_i}{2} \rceil$, $\frac{p_i+1}{2}$, and $\lfloor \frac{p_i}{2} \rfloor + 1$ is correct as well. However, if you wrote $\frac{p_i}{2}$, $\lceil \frac{p_i}{2} \rceil + 1$, or something similar, you will be deducted 5 points and I'll not deduct points from other subproblems.
   - If you wrote p_i instead of $\lceil \frac{p_i+1}{2} \rceil$, you will be deducted 10 points, but I'll not deduct points from other subproblems.

   Solution 2. $K(i, j) = \max\left(T(i-1, j), T(i-1, j-e_i) + \lceil \frac{p_i+1}{2} \rceil\right)$.
   The first term stands for the case where state i is cost-efficient while the latter stands for the case where state i is cost-inefficient.

   - The same policy as **Solution 1**.

(c) (30 points) Write pseudo-code for your algorithm.

Solution. Either Algorithm 1 or Algorithm 2 in the next page is correct.

Note.

- Base case is for 5 points. If only one is wrong, you can get 2 points.
- Filling DP table is for 15 points: 10 points for `if` case and 5 points for `else` case.
- If you iterate loop on slightly wrong range (e.g. wrong E_{lose}), you will be deducted 5 points.
- However, in case of Algorithm 1, you need to iterate j until E unless you did like L15 in Algorithm 1. If you iterated until only E_{win} or E_{lose} for Algorithm 1 (e.g. what if there was a state α where $p_\alpha = 1$ but $e_\alpha > E_{win}$? L15 can eliminate such corner cases), you cannot get any point.
- Finding the final solution is for 10 points.

Algorithm 1 Knapsack-Election

Input: n, $p_1, p_2, \ldots, p_n$, $e_1, e_2, \ldots, e_n$(electors allocated for each state)
Output: P_{min} := the smallest number of people who can make Mr. Cake win

1: $E \leftarrow \sum e_i$ // total number of electors
2: $E_{win} \leftarrow \lceil \frac{E+1}{2} \rceil$ // the minimum number of electors who can make Mr. Cake win
3: **for** $i = 1, 2, \ldots, n$ **do**
4: $\quad T[i, 0] \leftarrow 0$
5: **end for**
6: **for** $j = 1, 2, \ldots, E$ **do**
7: $\quad T[0, j] \leftarrow \infty$
8: **end for**
9: **for** $i = 1, 2, \ldots, n$ **do**
10: $\quad$ **for** $j = 1, 2, \ldots, E$ **do**
11: $\quad\quad$ **if** $j \geq e_i$ **then**
12: $\quad\quad\quad T[i, j] \leftarrow \min\left(T[i-1, j], T[i-1, j-e_i] + \lceil \frac{p_i+1}{2} \rceil\right)$
13: $\quad\quad$ **else**
14: $\quad\quad\quad T[i, j] \leftarrow T[i-1, j]$
15: $\quad\quad\quad$ // if you do 'min $\left(T[i-1, j], \lceil \frac{p_i+1}{2} \rceil\right)$', you don't have to iterate until E nor do L19-L22
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end for**
19: $P_{min} \leftarrow \infty$ // now we need to find the smallest number of people that can earn
20: **for** $j = E_{win}, \ldots, E$ **do** // at least E_{win} electors for Mr.Cake
21: $\quad P_{min} \leftarrow \min\left(P_{min}, T[n, j]\right)$
22: **end for**
23: **return** P_{min}

Algorithm 2 Knapsack-Election

Input: n, $p_1, p_2, \ldots, p_n$(population of each state), $e_1, e_2, \ldots, e_n$(electors allocated for each state)
Output: P_{min} := the smallest number of people who can make Mr. Cake win

$E \leftarrow \sum e_i$ // total number of electors
$E_{lose} \leftarrow \lceil \frac{E}{2} \rceil - 1$ // the largest number of electors who cannot make Mr. Cake cannot win
for $i = 1, 2, \ldots, n$ **do**
    $T[i, 0] \leftarrow 0$
end for
for $j = 1, 2, \ldots, E_{lose}$ **do**
    $T[0, j] \leftarrow 0$
end for
for $i = 1, 2, \ldots, n$ **do**
    for $j = 1, 2, \ldots, E_{lose}$ **do**
        if $j \geq e_i$ **then**
            $T[i, j] \leftarrow \max\left(T[i-1, j], T[i-1, j-e_i] + \lceil \frac{p_i+1}{2} \rceil\right)$
        else
            $T[i, j] \leftarrow T[i-1, j]$
        end if
    end for
end for
$P \leftarrow \sum \lceil \frac{p_i+1}{2} \rceil$
$P_{min} \leftarrow P - T[n, E_{lose}]$
return P_{min}

(d) (5 points) Analyze time-complexity of the algorithm.

Solution. $O(nE)$ where $E := \sum e_i$. It is because we need to iterate over n and E to fill the table and each entry takes $O(1)$ time.

Note.

- To get credits, your pseudo-code should be correct or have only minor errors.