ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2022

Problem Set 2

Due: 23:59, Mar. 29, 2021

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" in Gradescope, set your FULL NAME to your Chinese name.
3. If you want to submit a handwritten version, scan it clearly.
4. When submitting your homework in Gradescope, match each of your solutions to the corresponding problem number.
5. Late homeworks submitted within 24 hours of the due date will be marked down 25%, submitted within 48 hours will be marked down 50%, and will not be accepted past 48 hours.
6. You are required to follow ShanghaiTech's academic honesty policies. You are allowed to discuss problems with other students, but you must write up your solutions by yourselves. You are not allowed to copy materials from other students or from online or published resources. Violating academic honesty can result in serious sanctions.

Problem 1:

Given a list of positive integers, we can choose certain sets of three values to form the sides of a triangle; in particular, the values need to satisfy the triangle inequality. Note that the same value can occur multiple times in the list, and we treat these occurrences as different instances. Design an algorithm to return the number of possible sets, and show the time complexity of your algorithm.
Example:
Input: [3, 3, 4, 5]
Output: 4 sets, $\{3,3,4\}, \{3,4,5\}, \{3,4,5\}, \{3,3,5\}$ (note that $\{3,4,5\}$ is repeated because 3 occurs twice in the input).

Solution:

For a triplet of positive integers a, b, c, we can take them as side lengths and form a valid triangle iff the sum of any two sides is greater than the third side alone, i.e., $a+b>c,\ b+c>a,\ a+c>b$. Thus, we need to check every possible triplet in the given array and check if it satisfies the three inequalities above.

If we sort the elements of the array in ascending order such that $a \leq b \leq c$, we don't need to check all the three inequalities since $a+c>b$ and $b+c>a$ are satisfied implicitly. In this way, we only need to check if $a+b>c$.

Given an array arr, for every pair $(arr[i],\ arr[j])$ with $i<j$, we traverse towards the right for choosing the index k that makes $arr[i]+arr[j]>arr[k]$. That is, we need to find the right limit of the index k for a pair of indices (i,j). Since arr is in ascending order, we can conclude that all the elements in arr in range $[j+1, k-1]$ satisfy the inequality. Thus, the count of triplets will be added by $(k-1)-(j+1)+1 = k-j-1$.

Notice that when we choose a higher value of index j for a particular i, we need not start from the index $j+1$. Instead, we can start off directly from the value of k where we left for the last index j. This saves redundant computations with a linear scan.

Time complexity: $O(n^2)$. Sorting takes $O(nlogn)$, and loop of i and j will be in $O(n^2)$. Thus the complexity will be $O(nlogn+n^2) = O(n^2)$.

Problem 2:

It's a hot day and you want to buy some ice cream. The shop is offering a discount where you can buy two ice creams and get the third one for free. However, the price of the free ice cream cannot be higher than the minimum

price of the two you bought. The shop has many types of ice cream. Given a list of their prices, design an algorithm to buy all the ice creams using the minimum cost. Prove your algorithm is optimal and show its time complexity.

Solution:

Sort the price of ice creams in descending order, then divide them into groups of three, buy the first two and pick the third for free.

Suppose the number of ice creams is n, the maximum number for free will be $\lfloor n/3 \rfloor$, which is equal to the upper bound of our algorithm.

First, for any plan that the number of free ice creams is less than $\lfloor n/3 \rfloor$, we can definitely find three ungrouped ice creams. For the three ice creams, we can buy two of them to get the remaining one for free. Thus, the plan with the minimum cost must be a plan with the most ice creams for free.

Second, we sort the price list in descending order and denote it as $cost$, then the price of the free ice cream should not be larger than $cost[2]$. Similarly, the price of the k-th free ice cream should not be larger than $cost[3k+2]$. Thus, our algorithm gives the optimal solution among the plans with the most ice creams for free.

As above, our algorithm is optimal.

Time complexity: $O(nlogn)$.

Problem 3:

Given a list of letters, you need to merge them into a string. However, there must be at least n characters between two occurrences of the same letter. The letters can be permuted in any order, and you can add '#' characters to the string to satisfy the requirement. Design an algorithm to return the minimum length string, and prove your algorithm is optimal and show its time complexity.

Example:
Input: ['X', 'X', 'X', 'Y', 'Y', 'Y'], $n = 2$
Output: 8. One possible solution is "XY#XY#XY".

Solution:

First, for each letter, we record its remaining amount and the next possible index. We should always choose the letter which satisfies the rule and has the largest amount left. We prove the greedy algorithm is optimal by contradiction.

For an index t, suppose the letters a and b satisfy the rule, and they have p and q ($p > q$) left respectively. According to our algorithm above, we should

choose a first. Now assume that the algorithm is not optimal, i.e., we choose b instead. Denote $a_1, a_2, \cdots, a_p$ as the indices of a, and $b_1, b_2, \cdots, b_q$ as the indices of b, we have $a_1 > b_1 = t$. Following the rule, for any adjacent a_i, a_{i+1} and b_j, b_{j+1}, we must have $a_{i+1} - a_i > n$ and $b_{j+1} - b_j > n$.

- If there exists a smallest $k \in [2, q]$ that makes $a_k < b_k$, then we have $a_1 > b_1,\ a_2 > b_2,\ \cdots,\ a_{k-1} > b_{k-1},\ a_k < b_k$. We can swap the letters at $(a_1, b_1),\ (a_2, b_2),\ \cdots,\ (a_{k-1}, b_{k-1})$. Then,
  - the indices of a will be $b_1, b_2, \cdots, b_{k-1}, a_k, a_{k+1}, \cdots, a_p$,
  - the indices of b will be $a_1, a_2, \cdots, a_{k-1}, b_k, b_{k+1}, \cdots, b_q$.

  For the indices of a, we can find that $a_k - b_{k-1} > a_k - a_{k-1} > n$. Similarly, for the indices of b, we have $b_k - a_{k-1} > a_k - a_{k-1} > n$. Thus, swapping will not violate the rule.
- If for any $k \in [2, q]$, we have $a_k > b_k$. We can swap the letters of a and b. Then,
  - the indices of a will be $b_1, b_2, \cdots, b_q, a_{q+1}, \cdots, a_p$,
  - the indices of b will be $a_1, a_2, \cdots, a_q$.

  Since $a_{q+1} - b_q > a_{q+1} - a_q > n$, swapping will not violate the rule.

From above, we swap b and a at index t, which will not increase the total length of the string. Thus, always choosing the letter which satisfies the rule and has the largest amount left can lead to the string with minimum length, i.e., the algorithm is optimal.

Time complexity: $O(len \times type)$. len denotes the length of the list, $type$ denotes the type of letters.

Problem 4:

Given a list of non-negative integers, you want to merge all of them together in an arbitrary order into a single number. Design an algorithm which returns the minimum such number. Prove your algorithm is optimal and show its time complexity.

Example:
Input: [2, 14, 134, 56]
Output: 13414256

Solution:

Consider the integers as strings, then sort the strings in ascending order with the following rule. Given two strings x and y,

$$\begin{cases} x < y & \text{if } xy < yx \\ x = y & \text{if } xy = yx \\ x > y & \text{if } xy > yx \end{cases}$$

For example, '14' + '2' = '142', and '2' + '14' = '214', we have $142 < 214$, so '14' < '2'.

From the list of letters, suppose the integers a and b have m and n digits respectively, and satisfying $a > b$ according to the rule above. Then all the remaining integers form a string x with length of k. We assume that the greedy algorithm is not optimal, that a is in front of b with $a > b$ in the generated minimum number.

- If a and b are adjacent in the generated number, say xab, we can swap a and b and find that $xba < xab$, which contradicts the assumption.
- If a and b are not adjacent in the generated number, say axb. Since it is the minimum number, it should satisfy $ax < xa$ and $xb < bx$.
  - For $ax < xa$, we have $a \times 10^k + x < x \times 10^m + a$.
    Then, $a \times 10^k - a < x \times 10^m - x$.
    So, $a \times \frac{10^k-1}{10^m-1} < x$.
  - For $xb < bx$, we have $x \times 10^n + b < b \times 10^k + x$.
    Then, $x \times 10^n - x < b \times 10^k - b$.
    So, $x < b \times \frac{10^k-1}{10^n-1}$.

  From above, we have

  $$a \times \frac{10^k - 1}{10^m - 1} < x < b \times \frac{10^k - 1}{10^n - 1}$$

  Then,

  $$\frac{a}{10^m - 1} < \frac{b}{10^n - 1}$$
  $$a \times (10^n - 1) < b \times (10^m - 1)$$

  Thus,

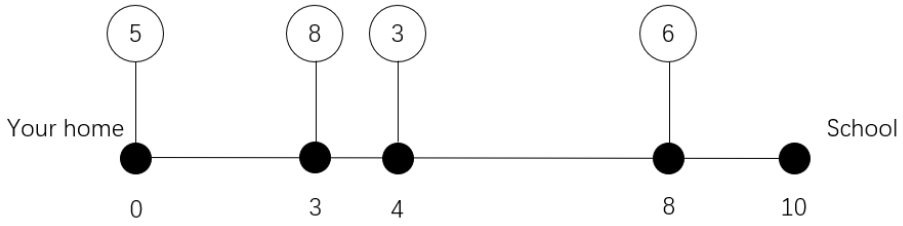
  $$a \times 10^n + b < b \times 10^m + a$$

  We find that $ab < ba \Rightarrow a < b$, which contradicts the assumption.

So we prove that the algorithm is optimal by contradiction.
Time complexity: $O(nlogn)$.

Problem 5:

There is a road from your home to school. The length of road is L kilometers, and your home is located at coordinate 0, and the school is located at coordinate L. There are n signs along the road, where the i'th sign has a value a_i, indicating that you must travel at a speed of exactly a_i minutes per kilometer until you reach the next sign. There is also a sign at coordinate 0 which sets the initial speed. We can use the signs to calculate the time to travel from home to school. For example, in Fig. 1, the total time is $3 \cdot 5 + 1 \cdot 8 + 4 \cdot 3 + 2 \cdot 6 = 47$ minutes.



We now want to remove at most k signs, in a way which minimizes the time to travel from home to school. You cannot remove the sign at coordinate 0. Design an efficient algorithm for this problem and give its time complexity.

Solution:
The time complexity is $O(n^3)$.
Need to know that you can't just transfer dp[i-1][j-1] to dp[i][j] because the sign before sign i may be removed already.

Problem 6:

You have two strings A and B of equal length. Both strings consist of lower case letters. You now want to change A to B. In each move you can change a segment of characters in A to any other character you want. That is, the new segment consists of only one kind of character. For example, for $A = ddddffee$ and $B = ddbbbcce$, you can first change A to $ddbbbfee$, then to $ddbbbcce$, using two

Algorithm 1 Problem 5

Define $dp[i][j]$ as minimum time when get to i-th sign and remove j signs.
$a[i]$ is the speed from i-th sign to j-th sign.
$d[i]$ is the distance from 0 to sign i.
for $i = 1$ to n **do** \\When get into i-th sign.
    for $p = 0$ to k **do** \\Remove p signs.
        for $j = 1$ to i **do** \\ Try every situation from sign j to sign i
            $len \leftarrow i - j - 1$ \\ The number of signs from i-th sign to j-th sign.
            if $p \geq len$ **then** \\ Make sure the move succeeds.
                $dp[i][p] = min(dp[i][p], dp[j][p - len] + a[j](d[i] - d[j]))$
            end if
        end for
    end for
end for
The answer is $dp[n][k]$.

moves in total. Give an efficient algorithm to calculate the minimum number of moves you need, and show the time complexity of your algorithm.

Solution:
This problem need to be split into two part.
Part 1: Get the minimum step change a empty string to string B.
Part 1: Then we need to use the result to change A to B.
That is because we can see the parts of A that different from B as the empty string. We don't need to change the same parts. So we can use the result in part 1 to finish part 2.
The time complexity is $O(n^3)$.

Problem 7:

You are going to the supermarket to buy some water. There are n types of water, where the i'th type has price p_i and weighs c_i kilograms. There are an infinite number of bottles of each kind of water. You want to know the least amount of money M needed to buy at least X kilograms of water, and the actual weight Y of water you will get. If there are multiple solutions with the same minimum cost, return the one with maximum weight. Design an efficient algorithm for this problem and give the algorithm's complexity.

Algorithm 2 Problem 6 part 1

Define $dp[i][j]$ as minimum step to change the empty section from i to j into $B[i...j]$.
for $len = 2$ to n **do** \\The length of the section we want change in one step.
    for $l = 1$ to $n - len + 1$ **do** \\Left margin of section.
        $r \leftarrow l + len - 1$; \\Right margin of section.
        $dp[l][r] = 1 + dp[l + 1][r]$; \\ Just change the leftmost one.
        for $k = l + 1$ to r **do**
            if $B[l] == B[l]$ **then** \\ Need to change the rest part following the leftmost one.
                $dp[l][r] = min(dp[l][r], dp[l][k - 1] + dp[k + 1][r])$;
            end if
        end for
    end for
end for

Algorithm 3 Problem 6 part 2

Define $d[i]$ as minimum step to change the $A[0...i]$ to $B[0...i]$.
$d[0] = 0$ \\ Initial d[0].
for $i = 1$ to n **do** \\The length of the section we want change in one step.
    if $A[i] == B[i]$ **then** \\ This part doesn't need to change.
        $d[i] = d[i - 1]$;
    else
        for $k = 1$ to i **do**
            $d[i] = min(d[i], d[k - 1] + dp[k][i])$;\\ Use the result in part 1 to get the minimum step.
        end for
    end if
end for

Solution:

This problem is a variant of the Knapsack problem. Need to notice that this problem want you to calculate the **minimum** cost that buy at least X kilograms water. That means X is known by us. Some students misunderstood the problem. Other students didn't give the answer the problem needed, but only the transfer function of dp.

Assume the maximum kilograms of water you can buy is C_{max}. The time complexity is $O(nC_{max}$. Because the problem don't provide specific number, we consider $O(nX)$ as the true answer since C_{max} is related to X. Other answers related to x are also considered correct.

Algorithm 4 Problem 7

Assume the maximum kilograms of water you can buy is C_{max}.
Define $dp[i]$ as minimum cost that have i kilograms of water left to buy.
$a[i]$ is the speed from i-th sign to j-th sign.
$d[i]$ is the distance from 0 to sign i.
for $i = 1$ to n **do** \\ i-th type of water.
    for $j = w[i]$ to C_{max} **do** \\ j kilograms left.
        $dp[j] = min(dp[j - w[i]] + c[i], dp[j])$;
    end for
end for
Find the minimum cost.
$ans = dp[X]$, $weight = X$ \\ Need to buy at least X kilograms water.
for $i = X$ to C_{max} **do**
    if $dp[i] \leq ans$ **then**
        ans=dp[i],wight=i;
    end if
end for

Problem 8:

The multiplication puzzle is played with a row of cards, each containing a single positive integer. During each move a player takes one card out of the row, and his score increases by the product of the number on the card taken and the numbers on the cards to the left and right of it. The player cannot take the first or last cards in the row. The game ends when there are only two cards left.

For example, if cards are initially [10, 1, 50, 20, 5], then player can can take cards in the order 1, 20, 50, and get a score of $10*1*50+50*20*5+10*50*5 = 500+5000+2500 = 8000$. If he took the cards in the opposite order, i.e. 50, 20, 1, his score would be $1*50*20+1*20*5+10*1*5 = 1000+100+50 = 1150$.

Design an efficient algorithm to calculate the minimum score the player can get, and show the complexity of your algorithm.

Solution:

The time complexity is $O(n^3)$.

Algorithm 5 Problem 8

```
Define dp[i][j] as minimum result from i to j.
num[i] is the number of card.
for i = 1 to n do \\ i-th type of water.
    for j = i + 1 to n do \\ j kilograms left.
        if j==i+1 then
            dp[i][j] = 0; \\ Initial dp.
        end if
        if j == i + 2 then
            dp[i][j] = num[i] * num[i + 1] * num[j];
        else
            for k = i + 1 to j do
                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j] + num[i] * num[k] * num[j]);
            end for
        end if
    end for
end for
```