ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2024

Problem Set 2

Zhou Shouchen

StudentID: 2021533042

Due: 23:59, April 11, 2024

1. Submit your solutions to the course Gradescope.
2. If you want to submit a handwritten version, scan it clearly.
3. Late homeworks submitted within 24 hours of the due date will be marked down 25%. Homeworks submitted more than 24 hours after the due date will not be accepted unless there is a valid reason, such as a medical or family emergency.
4. 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 penalties.

Problem 1:

Suppose you are given a set of intervals $I_1 = [s_1, t_1], \ldots, I_n = [s_n, t_n]$, which can possibly overlap. Your task is to select a set of points $p_1, \ldots, p_m$ so that each interval intersects at least one of the points. Give an algorithm to minimize the number of selected points m, and prove that your algorithm is optimal.

Solution:

The time consumption is sorting takes $O(n \log n)$ time, and ..... takes $O(n)$ time, so the total time complexity is $O(n \log n)$.
The space complexity is $O(n)$ for storing .

Problem 2:

Suppose you need to perform n tasks, and you can only perform one task at a time. It takes t_i time to perform the i'th task, and the task has an importance of $w_i > 0$. Let C_i be the completion time of the i'th task, i.e. the time when you finish performing the task. Find an ordering of the tasks to minimize their weighted completion time, defined as $\sum_{i=1}^{n} w_i C_i$. Your algorithm should work in $O(n \log n)$ time, and you need to analyze the algorithm's time complexity and prove its correctness.

Solution:

The time consumption is sorting takes $O(n \log n)$ time, and ..... takes $O(n)$ time, so the total time complexity is $O(n \log n)$.
The space complexity is $O(n)$ for storing .

Problem 3:

A thief is planning to rob houses along a street. Each house has a certain amount of money stashed, and the thief can rob any set of houses, as long as he does not rob any adjacent houses. Determine the maximum amount of money the thief can steal.

Solution:

For loop in the pseudo-code is form '**for** $i \leftarrow 1$ **to** n' represents $i = 1, 2, \cdots, n$ sequentially.

Algorithm 1 Maximum stolen money

1: **Input:** Number of house n, each house's money number $a_1, a_2, \cdots, a_n$.
2: **Output:** The maximum number of money could steal ans.
3: $dp(1,0) \leftarrow 0, dp(1,1) \leftarrow a_1$
4: **for** $i \leftarrow 2$ **to** n **do**
5: $\quad dp(i,0) \leftarrow \max\{dp(i-1,0), dp(i-1,1)\}$
6: $\quad dp(i,0) \leftarrow dp(i-1,0) + a_i$
7: **end for**
8: $ans \leftarrow \max\{dp(n,0), dp(n,1)\}$
9: **return** ans

The time complexity is $O(n)$.
The space complexity is $O(n)$.
Where n is the number of houses.

Problem 4:

Given three sequences L_1, L_2 and L, design an efficient algorithm to check if L_1 and L_2 can be interleaved to produce L. For example, the sequences $L_1 =$ `aabb` and $L_2 =$ `cba` can be interleaved into sequence $L =$ `acabbab`, but L_1 and L_2 cannot be interleaved into sequence $L =$ `aaabbbc`. Analyze the time and space complexity of your algorithm as a function of the lengths of L_1 and L_2.

Solution:

For loop in the pseudo-code is form '**for** $i \leftarrow 1$ **to** n' represents $i = 1, 2, \cdots, n$ sequentially.

The time complexity is $O(nm)$.
The space complexity is $O(nm)$.
Where n is the length of L_1, m is the length of L_2.

Algorithm 2 Interleaved the object sequence

Input: Two sequences L_1, L_2 and L.
Output: Whether L_1 and L_2 can be interleaved to produce L.
$dp(0,0) \leftarrow 1$
for $i \leftarrow 1$ **to** $|L_1|$ **do**
    if $L_1[i] == L[i]$ **then**
        $dp(i,0) \leftarrow 1$
    else
        $dp(i,0) \leftarrow 0$
    end if
end for
for $j \leftarrow 1$ **to** $|L_2|$ **do**
    if $L_2[j] == L[j]$ **then**
        $dp(0,j) \leftarrow 1$
    else
        $dp(0,j) \leftarrow 0$
    end if
end for
for $i \leftarrow 1$ **to** $|L_1|$ **do**
    for $j \leftarrow 1$ **to** $|L_2|$ **do**
        $dp(i,j) \leftarrow 0$
        if $L_1[i] == L[j]$ **then**
            $dp(i,j) \leftarrow dp(i,j) \vee dp(i-1,j)$
        end if
        if $L_2[j] == L[j]$ **then**
            $dp(i,j) \leftarrow dp(i,j) \vee dp(i,j-1)$
        end if
    end for
end for
return $dp(|L_1|, |L_2|)$

Problem 5:

Suppose you are given a propositional logic formula containing only the terms $\wedge$, $\vee$, T and F, without any parentheses. You want to find out how many different ways there are to correctly parenthesize the formula so that the resulting formula evaluates to true. For example, the formula $\mathrm{T} \vee \mathrm{F} \vee \mathrm{T} \vee \mathrm{F}$ can be correctly parenthesized in 5 ways:

$$(\mathrm{T} \vee (\mathrm{F} \vee (\mathrm{T} \vee \mathrm{F})))$$
$$(\mathrm{T} \vee ((\mathrm{F} \vee \mathrm{T}) \vee \mathrm{F}))$$
$$((\mathrm{T} \vee \mathrm{F}) \vee (\mathrm{T} \vee \mathrm{F}))$$
$$(((\mathrm{T} \vee \mathrm{F}) \vee \mathrm{T}) \vee \mathrm{F})$$
$$((\mathrm{T} \vee (\mathrm{F} \vee \mathrm{T})) \vee \mathrm{F})$$

Of these, 3 evaluate to true: $((\mathrm{T} \vee \mathrm{F}) \vee (\mathrm{T} \vee \mathrm{F}))$, $(\mathrm{T} \vee ((\mathrm{F} \vee \mathrm{T}) \vee \mathrm{F}))$ and $(\mathrm{T} \vee (\mathrm{F} \vee (\mathrm{T} \vee \mathrm{F})))$.

Give a dynamic programming algorithm to solve this problem. Describe your algorithm, including a clear statement of your dynamic programming equation, show that it is correct, and prove its running time.

Solution:

For loop in the pseudo-code is form '**for** $i \leftarrow 1$ **to** n' represents $i = 1, 2, \cdots, n$ sequentially.

The time complexity is $O(n^3)$.
The space complexity is $O(n^2)$.
Where n is the number of propositions.

Algorithm 3 Number of True Statements

Input: Number of propositions n, propositions $a_1, a_2, \cdots, a_n$, operators $b_1, b_2, \cdots, b_{n-1}$
Output: The number of True Statements
for $i \leftarrow 1$ **to** n **do**
    $dp(i, i, a_i) \leftarrow 1$
    $dp(i, i, \neg a_i) \leftarrow 0$
end for
for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow i + 1$ **to** n **do**
        for $k \leftarrow i$ **to** $j - 1$ **do**
            if $b_i == \wedge$ **then**
                $dp(i, j, 0) \leftarrow dp(i, k, 0) * dp(k + 1, j, 0)$
                $+ dp(i, k, 0) * dp(k + 1, j, 1)$
                $+ dp(i, k, 1) * dp(k + 1, j, 0)$
                $dp(i, j, 1) \leftarrow dp(i, k, 1) * dp(k + 1, j, 1)$
            else
                $dp(i, j, 0) \leftarrow dp(i, k, 0) * dp(k + 1, j, 0)$
                $dp(i, j, 1) \leftarrow dp(i, k, 1) * dp(k + 1, j, 1)$
                $+ dp(i, k, 0) * dp(k + 1, j, 1)$
                $+ dp(i, k, 1) * dp(k + 1, j, 0)$
            end if
        end for
    end for
end for
return $dp(1, n, 1)$

Problem 6:

Consider a weighted directed graph G with n vertices and m edges, where each edge (i, j) has a positive integer weight $w_{i,j}$. A walk is a sequence of not necessarily distinct vertices $v_1, v_2, ..., v_k$, such that any two consecutive vertices v_i, v_{i+1} are connected by an edge. The length of the walk is the sum of the weights of the edges in the walk. Design an algorithm to find the number of different walks from a vertex s to another vertex t which have length exactly L, and analyze the time complexity of your algorithm.

Solution:

For loop in the pseudo-code is form '**for** $i \leftarrow 1$ **to** n' represents $i = 1, 2, \cdots, n$ sequentially.

Algorithm 4 Number of different walks

Input: Graph $G = (V, E)$ with n vertices and m edges with weight $w_{i,j}$
Output: The number of different walks from s to t with length L
$dp(s, 0) \leftarrow 1$
$dp(x, t) \leftarrow 0, \forall (x, t) \neq (s, 0)$
for $val \leftarrow 1$ **to** L **do**
    for $(i, j) \in E$ **do**
        $dp(j, val) \leftarrow dp(j, val) + dp(i, val - w_{i,j})$
    end for
end for
return $dp(t, L)$

The time complexity is $O(mL)$.
The space complexity is $O(nL)$.
Where n is the number of vertices, m is the number of edges, L is the length of the walk.