ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2023

Problem Set 3

Due: 23:59, April 6, 2023

1. Submit your solutions to the course Blackboard.
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:

Given an array A, find the longest continuous increasing subsequence in the array and return its length. For example, for the array [1, 3, 2, 4, 7, 6, 8, 9], the longest continuous increasing subsequence is [6, 8, 9] and its length is 3. Design an algorithm, prove its correctness, and analyze its time and memory complexity.

Solution:

Let $L[i]$ denote the length of the longest continuous increasing subsequence ending at $a[i]$. If $a[i] > a[i-1]$, then $L[i] = L[i-1] + 1$, otherwise $L[i] = 1$. When $i = 1$, $L[i] = 1$ and $maxLen = 1$, so the algorithm is correct for the case where there is only one element in the array.

Consider an array a of length $n+1$, and let $L[i]$ denote the length of the longest continuous increasing subsequence ending at $a[i]$. According to the algorithm, if $a[i] > a[i-1]$, then $L[i] = L[i-1] + 1$, otherwise $L[i] = 1$. $L[i]$ is correct for any $i \leq n$, therefore $L[n+1]$ is correct because the algorithm calculates $L[n+1]$ based on the comparison between $a[n+1]$ and $a[n]$.

Since the algorithm uses the max function to update $maxLen$, the value of $maxLen$ is equal to the maximum value of all $L[i]$. According to the induction hypothesis, $L[i]$ is correct, so $maxLen$ is also correct.

Problem 2:

Given a string containing just the characters '(' and ')', return the length of the longest valid (well-formed) parentheses substring.

For example, if the input is ")()())", the output is 4, because the longest valid parentheses substring is "()()".

Solution:

We define $dp[i]$ to represent the length of the longest valid parenthesis at the end of the following I character. We initialize the dp array all to 0. Obviously a valid substring must end with a) ' , so we know the value of the corresponding position in the dp array that corresponds to the (ending substring must have a dp value of 0, we just need to solve for) ' . We iterate through the string from front to back to get the dp value, and we check every two characters:

1. $s[i] = ')'$ and $s[i-1] = '('$, which is a string shaped like "... ()", we can deduce:

$$dp[i] = dp[i-2] + 2$$

We can do this because the "()" at the end is a valid substring and increases the length of the previously valid substring by 2.

2. $s[i] = ')'$ and $s[i-1] = ')'$, which is a string like "...))", we can deduce that if $s[i - dp[i-1] - 1] = '('$, then

$$dp[i] = dp[i-1] + dp[i - dp[i-1] - 2] + 2$$

We consider that if the penultimate' is part of a valid substring (called a sub_s) , for the last')', if it is part of a longer substring, then it must have a corresponding '(' , and its position is in the penultimate ')' before the valid substring (that is, before the sub_s). So, if the substring subs happens to be preceded by '(' , then we use 2 plus $sub_s(dp[i-1])$ to update the $dp[i]$. At the same time, we will also add the length of the effective substring before the effective substring"(sub_s)", that is, add $dp[i - dp[i-1] - 2]$. The final answer is the maximum value in the dp array.

Time complexity: $O(N)$, where n is the length of the string. We only have to go through the whole string once to get the dp array.

Spatial complexity:$O(N)$. We need a dp array of size N.

Problem 3:

The school is organizing a spring outing for students to Disney Amusement Park. There are M students, and park has N areas ($M, N \geq 1$). We want to know how many ways there are to divide the students so that each student goes to exactly one area (we allow some areas to not be visited by any students). Note that we only care how many students go to the different areas, not which areas they go to.

For example, if there are 3 students and 2 areas, the output should be 2, because the only divisions are (3,0) and (2,1). Note that here, we consider assigning 2 students to the first area and 1 student to the second, to be the same as assigning 1 student to the first area and 2 to the second; both assignments yield the division (2,1).

Design an algorithm for this problem and analyze its time and memory complexity.

Solution:

Denote $F(M, N)$ to be the DP function to solve the issue with M students and N facilities.

When $M = 0$ or $N = 1$, output should be 1.

When $M < N$, $F(M, N) = F(M, M)$, since there will always be $N - M$ facilities remained empty.

When $M \geq N$, the allocation method can be divided into two categories:

1. At least one facility remains empty, then $F(M, N) = F(M, N - 1)$.

2. All facilities have at least one student, we can remove one student from all facilities and this action will not influence the number of allocation methods: $F(M, N) = F(M - N, N)$.

An example code:

def $F(M, N)$:

    if $(N == 1)$ return 1

    elif $(M == 0)$ return 1

    elif $(N > M)$ return $F(M, M)$

    elif $(M \geq N)$ return $F(M, N - 1) + F(M - N, N)$

Both space and time complexity should be $O(MN)$.

Problem 4:

Suppose there are n players playing games. Every player plays exactly one game with every other player. Every game has a winner (there are no ties). Given an input vector $(v_1, ..., v_n)$, design an efficient algorithm to determine whether it is possible for player i to win exactly v_i games, for $1 \leq i \leq n$.

Solution:

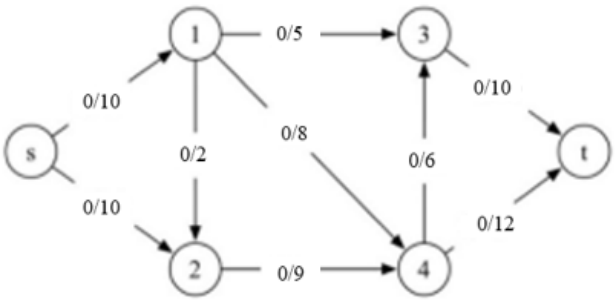
First we check an easy condition: altogether, players play $n(n-1)/2$ times, so the total number of wins is $n(n-1)/2$, so we check if $\prod_{i=1}^{n} v_i$ is equal to $n(n-1)/2$. If not, we can answer in the negative.

Otherwise we persevere. We create a network flow instance such that the vector $(v_1, \ldots, v_n)$ is a possible outcome if and only that network has a flow with value $n(n-1)/2$. We have 4 "layers" of nodes in the network: (a) source s, (b) game nodes $g_{i,j}$ where $1 \leq i < j \leq n$, (c) player nodes p_i where $1 \leq i \leq n$, (d) sink node t. Edges $(s, g_{i,j})$ have capacity 1; edges (p_i, t) have capacity v_i, and from $g_{i,j}$ we have two edges of capacity 1: $(g_{i,j}, p_i)$ and $(g_{i,j}, p_j)$.

The idea is that the flow is an accounting of game results. Because capacities are integers, a maximum flow has integer values. We must have an inflow of 1 to every game node. Then the outflow from a game node goes to the winning player. The number of wins that player accumulates is the inflow of his node, and the outflow has to be equal to v_i.

Problem 5:

Run the Ford-Fulkerson algorithm on the flow network in the figure below, and show the residual network after each flow augmentation. For each iteration, pick the augmenting path that is lexicographically smallest. (e.g. if you have two augmenting paths $1 \to 3 \to t$ and $1 \to 4 \to t$, then you should choose $1 \to 3 \to t$, because it is lexicographically smaller than $1 \to 4 \to t$).



Solution:

The augmenting path:$s \to 1 \to 2 \to 4 \to t$; $s \to 1 \to 3 \to t$;$s \to 1 \to 4 \to t$; $s \to 2 \to 1 \to 4 \to t$; $s \to 2 \to 4 \to t$; $s \to 2 \to 4 \to 3 \to t$

After path $s \to 1 \to 2 \to 4 \to t$:

s→1 = 2/10, 1→2 = 2/2, 2→4 = 2/9, 4→t = 2/12, v(f) = 2

After path $s \to 1 \to 3 \to t$:

s→1 = 7/10, 1→3 = 5/5, 3→t = 5/10, v(f) = 2 + 5 = 7

After path $s \to 1 \to 4 \to t$:

s→1 = 10/10, 1→4 = 3/8, 4→t = 5/12, v(f) = 7 + 3 = 10

After path $s \to 2 \to 1 \to 4 \to t$:

s→2 = 2/10, 2→1 = 0/2, 1→4 = 5/8, 4→t = 7/12, v(f) = 10 + 2 = 12

After path $s \to 2 \to 4 \to t$:

s→2 = 7/10, 2→4 = 7/9, 4→t = 12/12, v(f) = 12 + 5 = 17

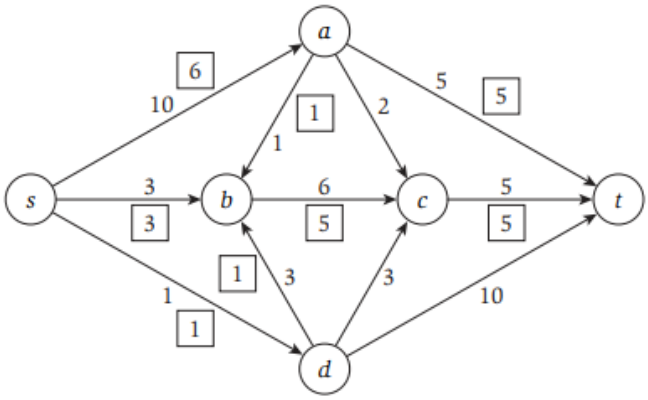
After path $s \to 2 \to 4 \to 3 \to t$:

s→2 = 9/10, 2→4 = 9/9, 4→3 = 2/6, 3→t = 7/10, v(f) = 17 + 2 = 19

Final maximum flow result is 19, s→1 = 10/10, s→2 = 9/10, 1→2 = 0/2, 1→3 = 5/5, 1→4 = 5/8, 2→4 = 9/9, 3→t = 7/10, 4→3 = 2/6, 4→t = 12/12

Problem 6:

The figure shows a flow network in which an s-t flow has been computed. The capacity of each edge appears as a label next to the edge, and the numbers in boxes give the amount of flow sent on each edge (edges without boxed numbers have no flow being sent on them). Find a minimum s-t cut in the flow network, and say what its capacity is.



Solution:

The minimum cut is $(\{s, a, b, c\}, \{d, t\})$. Its capacity is 11.