ShanghaiTech University

CS240 Algorithm Design and Analysis

Fall 2022

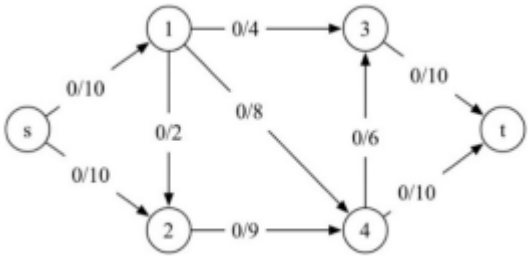
Problem Set 2

Due: 23:59, Oct. 25, 2022

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" of Gradescope, set your FULL NAME to your Chinese name and enter your STUDENT ID correctly.
3. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
4. When submitting your homework, match each of your solution to the corresponding problem number.

Problem 1:

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 path $1 \to 3 \to$ t and $1 \to 4 \to$ t, then you should choose $1 \to 3 \to$ t, which is lexicographically smaller than $1 \to 4 \to$ t)



Solution:

The augmenting path:$s124t$; $s13t$; $s14t$; $s214t$; $s24t$; $s243t$

Problem 2:

Given an $m \times n$ matrix in which every element is a positive integer, you need to select a subset of the elements in the matrix so that these selected elements are not adjacent. We define that element (i, j) is adjacent to elements $(i, j \pm 1)$ and $(i \pm 1, j)$ but is not adjacent to elements $(i \pm 1, j \pm 1)$. Design an efficient algorithm that maximizes the sum of the selected elements.

Solution:

Construct a network as following:

- Regard the matrix as a table and do black-white dyeing on that.
- Regard the matrix entries as nodes and add nodes s, t.
- "$\rightarrow$" from S to black entries and from white entries to T, set the capacity of the edges as the same as the value in corresponding entries.
- "$\rightarrow$" from all the black entries to the white entries with infity capacity.

Then run the Ford-Fulkson to get a max flow f. Then $v - f$ is the maximum we are searching for where v is the sum of all the entries in matrix. The selected matrix entries are those satisfying $f(e) \neq c(e)$ where e is an edge connecting the entry and s or t.

Problem 3:

Suppose there is Super Mario: the Game of eating Gold coins. The goal of this game is to get as many gold coins as possible under the rules.

Now let's introduce the game settings.

There are N $\times$ N grids as game map, in which there are only three categories for each grid, which are represented by values 0,1,2:

1. The grid value is 0: this is an open space, which can be directly passed.

2. The grid value is 1: there is a gold coin in the open space, which can be passed and the number of gold coins obtained increases by one.

3. The grid value is 2: this is an obstacle, unable to pass.

In the game, Mario needs to do these things:

Mario starts from grid position (0, 0) and arrives at (N-1, N-1). In the game, Mario can only walk down or right, and can only walk through the effective grid (the grid value is 0 or 1).

When Mario arrives at (N-1, N-1), Mario should continue to walk back to (0, 0). At this stage, Mario can only walk up or left, and can only cross the effective grid (the grid value is 0 or 1).

When Mario goes from (0, 0) to (N-1, N-1) and then from (N-1, N-1) to (0, 0), Mario finishes the game and counts the gold coins obtained. If there is no path that Mario can pass between (0, 0) and (N-1, N-1), the game ends and the number of gold coins obtained is 0.

NOTE: when Mario passes a grid and there is a gold coin in the grid (the grid value is 1), Mario will take the gold coin and the grid will be empty (the grid value becomes 0).

Design an effective algorithm to play the game and return the maximum gold coins which can be obtained. Analyze the algorithm's time complexity and space complexity.

Example:

$$Input: \text{grids} = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 2 & 1 & 0 \end{bmatrix}, N = 4$$

$$output: \text{ max number of gold coins} = 6$$

Solution:

Algorithm: Dynamic Programming

The path from (N-1, N-1) to (0, 0) can be seen as the other path from (0, 0) to (N-1, N-1). So this problem can be equivalent to the other problem: two Mario (named A and B) start from (0, 0) to (N-1, N-1) to get max gold coins.

Suppose A and B start at the same time and walk at the same speed.

Under the same time, the sum of their steps to the right plus the steps to the down is a fixed value (set as k).

Set the position of A as (x1, y1), and the position of B as (x2, y2), then x1+y1=x2+y2=k.

When x1 = x2, there is y1 = y2. A and B arrived at the same position.

Define f[k][x1][x2]: Max gold coins which A and B get, when A starts from (0, 0) to (x1, k-y1) and B starts from (0, 0) to (x2, k-y2)

When (x1, k-y1) or (x2, k-y2) is obstacle, f[k][x1][x2] = -∞

Enumerate the previous steps of A and B, there are four transfer states to compute f[k][x1][x2]:

- A and B both go right: using f[k-1][x1][x2] to compute f[k][x1][x2]
- A goes down, B goes right: using f[k-1][x1-1][x2] to compute f[k][x1][x2]
- A goes right, B goes down: using f[k-1][x1][x2-1] to compute f[k][x1][x2]
- A and B go down: using f[k-1][x1-1][x2-1] to compute f[k][x1][x2]

Take the maximum value of this four cases, then add grids[x1][k-x1] and grids[x2][k-x2] to get f[k][x1][x2]. (If x1=x2, only add grids[x1][k-x1], because gold coins in a position can only be obtained once.)

The initial state: f[0][0][0] = grids[0][0]

The final answer: max(f[2N-2][N-1][N-1],0)

Time Complexity : $O(N^3)$.

Space Complexity : $O(N^3)$.

Problem 4:

Suppose that we wish to know which stories in a 36-storey building are safe to drop the new iPhone 14 from, and which will cause the iPhone to break on landing. We make a few assumptions:

1. An iPhone that survives a fall can be used again.
2. A broken iPhone must be discarded.
3. The effect of a fall is the same for all iPhones.
4. If an iPhone breaks when dropped, then it would break if dropped from a higher floor.
5. If an iPhone survives a fall then it would survive a shorter fall.
6. It is not ruled out that the first-floor windows break iPhones, nor is it ruled out that the 36th-floor do not cause an iPhone to break.

If only one iPhone is available and we wish to be sure of obtaining the right result, the experiment can be carried out in only one way. Drop the iPhone from the first-floor window; if it survives, drop it from the second-floor window. Continue upward until it breaks. In the worst case, this method may require 36 droppings. Suppose there are n iPhones that are available and the problem is about k-storey building. What is the least number of droppings that is guaranteed to work?

Solution:

When we drop an iPhone from a floor x, there can be two cases (1) The iPhone breaks (2) The iPhone doesn't break.

1. If the iPhone breaks after dropping from 'xth' floor, then we only need to check for floors lower than 'x' with remaining iPhones as some floor should exist lower than 'x' in which iPhone would not break; so the problem reduces to $x-1$ floors and $n-1$ iPhones.
2. If the iPhone doesn't break after dropping from the 'xth' floor, then we only need to check for floors higher than 'x'; so the problem reduces to '$k-x$' floors and n iPhones.

Since we need to minimize the number of trials in worst case, we take the maximum of two cases. We consider the max of above two cases for every floor and choose the floor which yields minimum number of trials.

In Dynamic Programming, we work on the same idea as described above neglecting the case of calculating the answers to sub-problems again and again. The approach will be to make a table which will store the results of sub-problems so that to solve a sub-problem, it would only require a look-up from the table which will take constant time, which earlier took exponential time.

Formally for filling $DP[i][j]$ state where 'i' is the number of iPhones and 'j' is the number of floors:

We have to traverse for each floor 'x' from '1' to 'j' and find minimum of:

$$(1 + max(DP[i-1][j-1], DP[i][j-x])).$$

Problem 5:

Given three sequences L1, L2 and L, whose length is m, n and m+n respectively. Design an efficient algorithm to check if L1 and L2 can be merged into L such that all the letter orders in L1 and L2 stay unchanged with optimal time complexity. Analyze the algorithm's time complexity and space complexity.
Example 1: L1 = aabb, L2 = cba, L = acabbab, then return true.
Example 2: L1 = aabb, L2 = cba, L = aaabbbc, then return false.

Solution:

Algorithm:

1. Create a DP array (matrix) of size M*N, where m is the size of the first string and n is the size of the second string. Initialize the matrix to false.
2. If the sum of sizes of smaller strings is not equal to the size of the larger string then return false and break the array as they cant be the interleaved to form the larger string.
3. Run a nested loop the outer loop from 0 to m and the inner loop from 0 to n. Loop counters are i and j.
4. If the values of i and j are both zeroes then mark dp[i][j] as true. If the value of i is zero and j is non zero and the j-1 character of B is equal to j-1 character of C the assign dp[i][j] as dp[i][j-1] and similarly if j is 0 then match i-1 th character of C and A and if it matches then assign dp[i][j] as dp[i-1][j].
5. Take three characters x, y, z as (i-1)th character of A and (j-1)th character of B and (i + j – 1)th character of C.
6. If x matches with z and y does not match with z then assign dp[i][j] as dp[i-1][j] similarly if x is not equal to z and y is equal to z then assign dp[i][j] as dp[i][j-1].
7. If x is equal to y and y is equal to z then assign dp[i][j] as bitwise OR of dp[i][j-1] and dp[i-1][j].
8. return value of dp[m][n].

Time Complexity: O(M*N).
Space Complexity: O(M*N).