ShanghaiTech University

# CS240 Algorithm Design and Analysis

## Fall 2022

## Problem Set 2

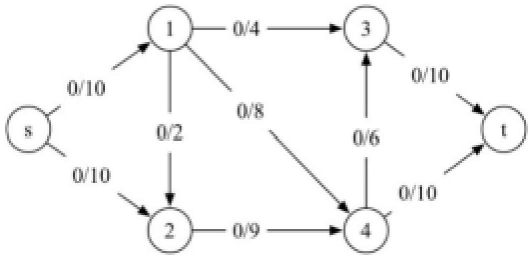
Dai ZiJia 2022233158

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.

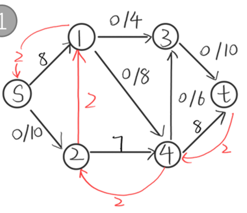
## Problem 1:

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 $1 \to 4 \to t$)



Result as the following:

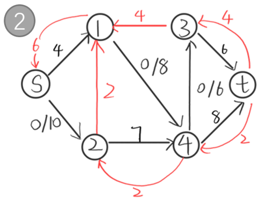
1



$S \to 1 \to 2 \to 4 \to t$

Flow value = 2

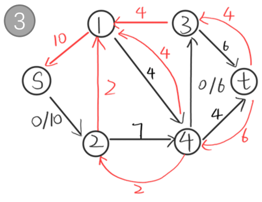
2



$S \to 1 \to 3 \to t$

Flow value = 6

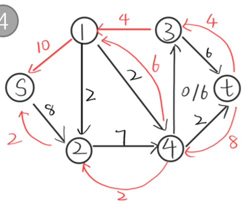
3



$S \to 1 \to 4 \to t$

Flow value = 10

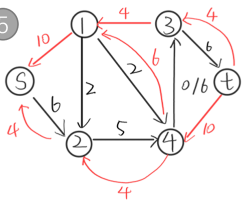
4



$S \to 2 \to 1 \to 4 \to t$

Flow value = 12

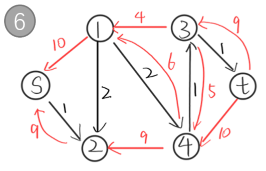
5



$S \to 2 \to 4 \to t$

Flow value = 14

6



$S \to 2 \to 4 \to 3 \to t$

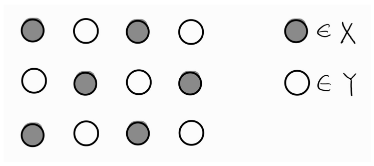
Flow value = 19

## Problem 2:

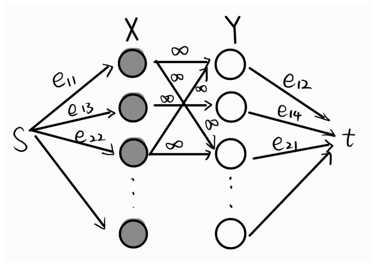
Given an $m \times n$ matrix in which every element is a positive integer, you need to select a subset of the elements in the matrix so that these selected elements are not adjacent. We define that element $(i, j)$ is adjacent to elements $(i, j \pm 1)$ and $(i \pm 1, j)$ but is not adjacent to elements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

**Solution:**

This can be seen as a maximum-weight independent set problem, we can use min cut to obtain the result. First we separate the elements into a bipartite graph. That means adjacent elements belong to different set $X, Y$.



Add $s$ and $t$. Connect a directed edge from $s$ to each vertex in the set $X$ with capacity equal to the integer of elements. Then make a directed edge from each vertex in the set $Y$ to $t$ with capacity equal to the integer of elements. Finally, adjacent elements $x_i, y_j$ are connected from $x_i$ to $y_j$ by a directed edge of $\infty$ capacity.



According to "Max-flow min-cut theorem", we can directly caculate the maximum flow on the network, and the result equals to

$$Result = \sum_{i=1}^{M} \sum_{j=1}^{N} e_{ij} - flow_{max}$$

**Analysis:** Use Ford-Fulkerson to caculate Max-flow, so the time complexity is $O(mn)$

## 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 × 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:**

Use the dynamic programming to solve the problem. There is a matrix $dp[n, n]$ to save the result. The elements in $dp$ symbolize the maximum gold coins can be obtained. Because Mario goes from (0,0) to (N-1,N-1) and then from (N-1,N-1) to (0,0), there will be two times dynamic programming in the algorithm, and **if the coin is taken from (0,0) to (N-1,N-1), the grid value must set to zero**. The first state transition equation is as following($dp[i,j] = -1$ symbolize there is no way to grid[i,j] or here is an obstacle):

$$dp_1[i,j] = \begin{cases} \max(dp_1[i-1,j], dp[i,j-1]) + grid[i,j], & dp_1[i-1,j] \neq -1 \wedge grid[i,j] \neq 2 \\ \max(dp_1[i-1,j], dp[i,j-1]) + grid[i,j], & dp_1[i,j-1] \neq -1 \wedge grid[i,j] \neq 2 \\ -1, & else. \end{cases} \tag{1}$$

The second state transition equation is as following:

$$dp_2[i,j] = \begin{cases} \max(dp_2[i+1,j], dp[i,j+1]) + grid[i,j], & dp_2[i+1,j] \neq -1 \wedge grid[i,j] \neq 2 \\ \max(dp_2[i+1,j], dp[i,j+1]) + grid[i,j], & dp_2[i,j+1] \neq -1 \wedge grid[i,j] \neq 2 \\ -1, & else. \end{cases} \tag{2}$$

$$Coin_{max} = max(dp_1[n,n], 0) + max(dp_2[0,0], 0)$$

**Analysis:**

The computation of subproblem cost $O(1)$ and there are 2*n*n subproblems, so the time complexity is $O(n^2)$. Since we make two n*n matrix, so the space complexity is $O(n^2)$.

## 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 egg 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:**

Use the dynamic programming to solve the problem. The state transition function is $f(k, n)$, which symbolize the least number of droppings that is guaranteed to work. Suppose we throw our first iphone at layer $x(1 <= x <= k)$, there are two cases will occur:

- The iphone broken. The function $f$ becomes $f(k-x, n)+1, 1 <= x <= k$.
- The iphone didn't breaks. We need to test from last 1 to $x-1$ floors. The function $f$ becomes $f(x-1, n-1)+1, 1 <= x <= k$.

Based on the above situation, The state transition function $f$ is

$$f(k, n) = min(max(f(k-x, n), f(x-1, n-1)) + 1), 1 <= x <= k$$

**Analysis:** The time to solve subproblems cost $O(n)$ and the number of subproblems is $k * n$, so the time complexity is $O(kn^2)$. The space complexity is $O(kn)$.

## 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:**

Use the dynamic programming to solve the problem. There is a matrix $dp[m, n]$ to save the result. The elements in $dp$ symbolize whether the first $i$ letters in L1 and the first $j$ letters in L2 can be merge into the first $i + j$ letters in L. The state transition equation is as following:

$$dp[i, j] = \begin{cases} True, & dp[i-1, j] = True \wedge L1[i] = L[i+j] \\ True, & dp[i, j-1] = True \wedge L2[j] = L[i+j] \\ False, & else. \end{cases} \tag{3}$$

**Analysis:**

The algorithm's time complexity is $O(mn)$, for we just compute the value of elements in matrix $dp[m, n]$ only once. The space complexity is $O(mn)$ too, because we just make a size of $m * n$ matrix.