ShanghaiTech University

CS240 Algorithm Design and Analysis

Fall 2023

Problem Set 2

Due: 23:59, Nov. 21, 2023

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:

(**Dynamic Programming**) You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have a security system connected, and it will automatically contact the police if two adjacent houses were broken into on the same night. Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

Solution:

We define i as today, $i-1$ for the day before and $i-2$ for the previous two days.

there are two situations

first:

we can make the decsion,robbing in today . Then the total amount of money robbed should be the total amount of money robbed in the previous two days plus the money robbed today.

$$amount[i] = amount[i-2] + price[i] \tag{1}$$

second:

we can make the decsion, don't robbing in today. Then the total amount of money should be equal to the day before.

$$amount[i] = amount[i-1] \tag{2}$$

finally,we conclude the amount of money that is equal to

$$amount[i] = \max(\quad (1), \quad (2) \quad)$$

Problem 2:

(**Dynamic Programming**) Given two strings `str1` and `str2`, give an algorithm to compute the minimum number of operations required to transform `str1` into `str2`. You have the following three operations permitted on a word:

1. Insert a character
2. Delete a character
3. Replace a character

Solution:

We use dp[i][j] to represent str1[0:i] and str2[0:j] to achieve the same minimum number of operations.We have two cases.

one is that when the character of the current str1[i] are equal to the characters of str2[j], there is an equation

$$dp[i][j] = dp[i-1][j-1]$$

another is that when the characters of the current str1[i] is not equal to the character of str2[i],there is three situations:

Insert a character:

$$dp[i][j] = dp[i-1][j] + 1 \tag{1}$$

Delete a character:

$$dp[i][j] = dp[i][j-1] + 1 \tag{2}$$

Replace a character:

$$dp[i][j] = dp[i-1][j-1] + 1 \tag{3}$$

finally

$$dp[i][j] = min(\ (1),\ (2),\ (3)\ )$$

Time Complex:O(n*m)
Space Complex: O(m*n)

Problem 3:

(**Dynamic Programming**) Given a map $f : \{a, b, ..., z\} \to \{1, 2, ..., 26\}$ defined as $f(a) = 1, f(b) = 2, \cdots, f(z) = 26$. For any string s that only contains characters from $\{a, b, \cdots, z\}$, we can construct a natural map $g : \{a, b, ..., z\}^* \to \{1, 2, ..., 26\}^*$ by applying f to each character of s. That is

$$g(s) = f(s[0]) \ || \ f(s[1]) \ || \ ... \ || \ f(s[n]).$$

For example, $g(ab) = 12$.

Given a numeric string s' that only contains characters from $\{0, 1, \cdots, 9\}$. Please design an algorithm to find the number of possible string s such that $g(s) = s'$. For example, $s' = 1234$, then s only can be *abcd*, *lcd* and *awd*, so the number is 3.

Notice: **When you provide pseudocode for your algorithm, you must also provide the appropriate comments or it will not be considered correct.**

Solution:

```
def count_possible_strings(s):
    n = len(s)
    dp = [0] * (n + 1)
    dp[0] = 1 # initial dp[0]

    for i in range(1, n + 1): # dp has size = n+1
        for j in range(1, min(i, 2) +1): # Only the first two character
            intervals of i need to be considered
           if is_valid_mapping(s[i - j:i]):

               dp[i] += dp[i - j] # adding the previous dp[i-j]

    return dp[n] # return the outcome


def is_valid_mapping(substring): # judge substring in the range of number
    return 1 <= int(substring) <= 26
```

‘dp[i]‘ represents the number of possible strings, of length ‘i‘, that can be mapped to the given numeric string ‘s‘.

Within the loop, we iterate through each position in the numeric string, assuming the current position is ‘i‘.

In the inner loop, we consider up to two characters before the current position (‘min(i, 2)‘ ensures we consider at most two characters).

For each ‘j‘ (ranging from 1 to ‘min(i, 2)‘), we check if the substring ‘s[i - j:i]‘ can be mapped to a valid character.

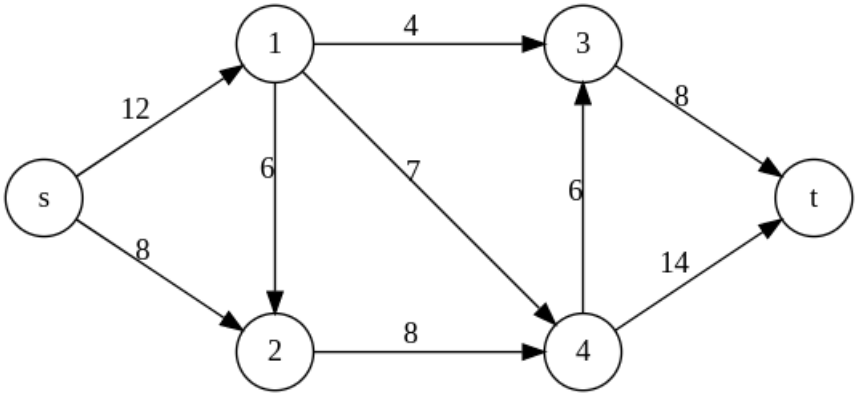
If it can be mapped, then ‘dp[i - j]‘ represents the number of possible strings considering the first ‘j‘ characters.

The statement ‘dp[i] += dp[i - j]‘ signifies that we accumulate this count into the possible solutions at the current position ‘i‘, updating ‘dp[i]‘.

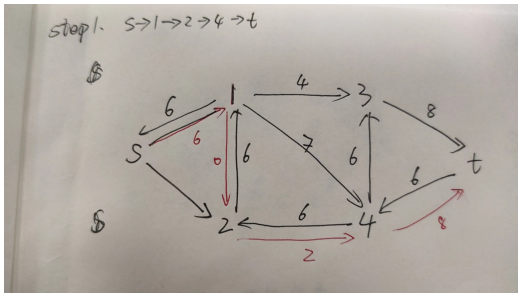
This process can be understood as follows: for each current position ‘i‘, we consider all possible prefix substrings and accumulate their solutions into the current position's solutions. Through this accumulation process, we ultimately determine the total number of possible strings of length ‘n‘ that can be mapped to the given numeric string ‘s‘.

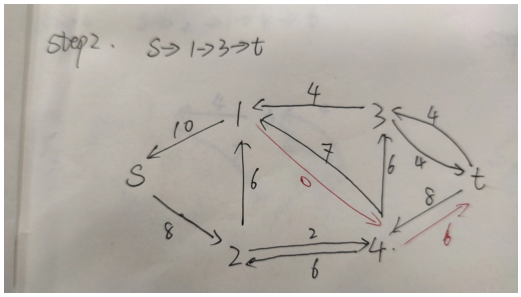
Problem 4:

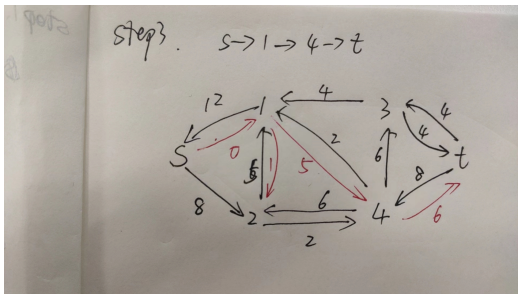
Run the Ford-Fulkerson algorithm on the flow network in the figure below, and show the residual network after each flow augmentation. For each iteration, pick the augmenting path that is lexicographically smallest. (e.g. if you have two augmenting paths $s \to 3 \to t$ and $s \to 4 \to t$, then you should choose $s \to 3 \to t$, because it is lexicographically smaller than $s \to 4 \to t$. Moreover for augmenting paths $s \to 3 \to t$ and $s \to 2 \to 4 \to t$, you should choose $s \to 2 \to 4 \to t$)

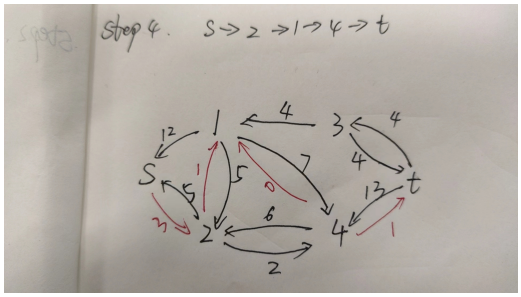


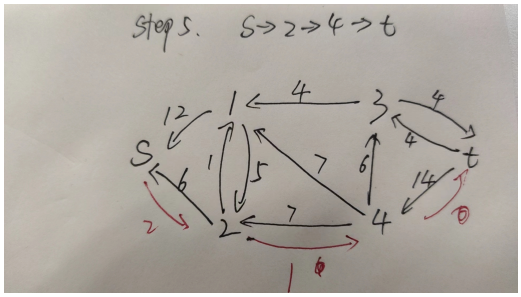
Solution:

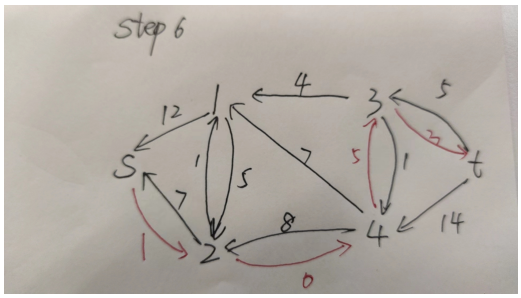












Final result is show below. The value of max flow is 19.

Problem 5:

Given a directed graph $G = (V, E)$, and source s, sink t, and all edge capacities in G are positive integers. Design an algorithm in polynomial time to determine if G has a unique minimal s-t cut.

Solution:

1. Let's first find the maximum flow for the original image

2. For the residual network, we start from S and find all the points that S can reach, and then start from T to find all the points that can reach T.

3. Determine whether there is a point in the original network that has not been accessed, if not, it is the only one, if not, it is not the only one!

```
function Max_Flow(graph, source, sink)
    // implementation of maximum flow algorithm
end function
function BFS(graph, source, visitedS, visitedT)
    // implementation of breadth first search
end function
function All_Points_Visited(graph, visitedS, visitedT)
    // check if all points are visited
end function


visitedS = {} // empty set for visited nodes from S
visitedT = {} // empty set for visited nodes to T
maxFlow = Max_Flow(graph, S,T) // calculate maximum flow
BFS(graph,S, visitedS, visitedT) // find nodes reachable from S starting
BFS(graph, T, visitedS, visitedT) // find nodes reachable from T starting
if All_Points_Visited(graph, visitedS, visitedT) then
    unique = true // unique minimum cut exists
else
    unique = false // multiple minimum cuts exist
end if
```

Problem 6:

Given a $n * n$ chess board, and there are k obstacles in k squares. You can not put any chess in square with obstacle. You need to put as many knight pieces as you can, such as no knight can attack another knight. Given a knight at (x,y), it can attack (x+1,y+2),(x+1,y-2),(x-1,y+2),(x-1,y-2),(x+2,y-1),(x+2,y+1),(x-2,y-1),(x-2,y+1). Design a minimal s-t cut algorithm to output the maxinum number of knights. (Hint:Divide the chessboard into white squares and black squares, Bipartite Matching)

Solution:

we can use a Bipartite Matching approach. Here's the outline of the algorithm:

1. Create a Bipartite Graph: - Divide the chessboard into white squares and black squares. Each square will be a node in the graph. - Connect each white square to its attackable black squares (according to the knight's movements).

2. Add Source and Sink Nodes: - Add a source node 's' and connect it to all white squares. - Add a sink node 't' and connect all black squares to it.

3. Add Obstacles: - If there are obstacles, remove the corresponding edges between the nodes representing the obstacles and their reachable squares.

4. Apply Maximum Flow Algorithm: - Apply a maximum flow algorithm (Ford-Fulkerson) to find the maximum flow from 's' to 't'.

5. Output the Maximum Flow: - The maximum flow value as ans will be the maximum number of knights that can be placed on the board without attacking each other.

$$ans = maximum \quad flow \quad value$$

n

$$\textbf{(maximum)}_{knights} = n * n - ans - k$$

if there are obstacles, you should remove the corresponding edges in the graph to ensure that knights cannot be placed on those squares.