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 can solve this problem by using dynamic programming. We can define an array dp, where $dp[i]$ is the maximum amount of money that can be robbed up to the i-th house.

The state **transition equation** is:

$$dp[i] = \max(dp[i-2] + \text{nums}[i], dp[i-1])$$

This equation means that for the i-th house, we can choose to rob it (in this case we can't rob the $(i-1)$-th house, so the maximum amount of money is $dp[i-2] + \text{nums}[i]$), or not rob it (in this case the maximum amount of money is $dp[i-1]$).

Algorithm Steps:

1. Initialize the dp array: $dp[0] = \text{nums}[0]$ (only one house, rob it), $dp[1] = \max(\text{nums}[0], \text{nums}[1])$ (two houses, rob the one with more money).

2. For $i = 2$ to $n-1$ (where n is the number of houses), update $dp[i]$ using the state transition equation $dp[i] = \max(dp[i-2] + \text{nums}[i], dp[i-1])$.

3. $dp[n-1]$ is the final result, i.e., the maximum amount of money that can be robbed.

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 can solve this problem using dynamic programming. We define $dp[i][j]$ as the minimum number of operations needed to transform the first i characters of `str1` into the first j characters of `str2`. Then, we can obtain the following recurrence relations:

1. When i is 0 (i.e., `str1` is empty), $dp[0][j]$ is j because we need to perform j insertions to transform `str1` into the first j characters of `str2`.
2. When j is 0 (i.e., `str2` is empty), $dp[i][0]$ is i because we need to perform i deletions to make the first i characters of `str1` empty.
3. When both i and j are not 0, we consider the ith character of `str1` (denoted as c_1) and the jth character of `str2` (denoted as c_2):
   - If c_1 is the same as c_2, then $dp[i][j] = dp[i-1][j-1]$ because we don't need to do any operations.
   - If c_1 is not the same as c_2, we can perform one of the insert, delete, or replace operations. Therefore, $dp[i][j]$ is the minimum number of operations among these three operations, i.e., $dp[i][j] = \min\{dp[i][j-1], dp[i-1][j], dp[i-1][j-1]\} + 1$.

By the above recurrence relations, we can compute all $dp[i][j]$ in a bottom-up manner. Finally, $dp[m][n]$ is the answer we want, where m and n are the lengths of `str1` and `str2`, respectively.

DP Transition Formula

Here are the dynamic programming transition formulas:

For the base cases:

1. When `str1` is empty, we perform j insertions:

$$dp[0][j] = j, \quad \text{for } j = 0, 1, 2, \ldots, n$$

2. When `str2` is empty, we perform i deletions:

$$dp[i][0] = i, \quad \text{for } i = 0, 1, 2, \ldots, m$$

For the general cases:

3. When the ith character of `str1` is the same as the jth character of `str2`:

$$dp[i][j] = dp[i-1][j-1], \quad \text{if } str1[i-1] = str2[j-1]$$

4. When the ith character of `str1` is not the same as the jth character of `str2`:

$$dp[i][j] = \min\{dp[i][j-1], dp[i-1][j], dp[i-1][j-1]\}+1, \quad \text{if } str1[i-1] \neq str2[j-1]$$

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.**

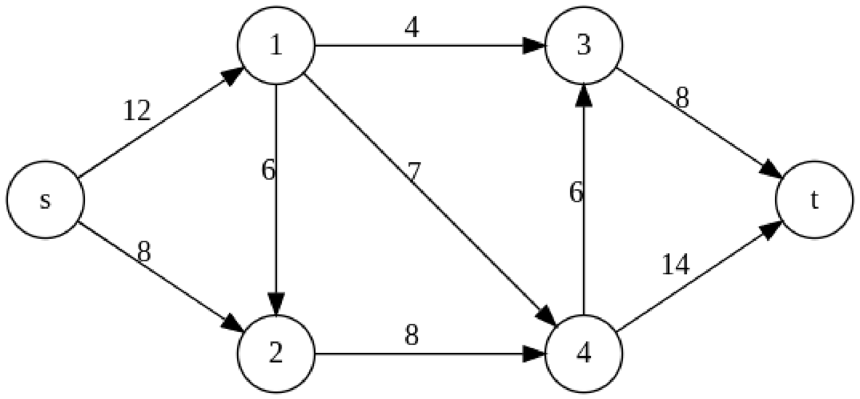
Solution:

We can solve this problem by using dynamic programming method. We use $\text{len}(s')$ to denote the length of string s', and $\text{DP}(s', i)$ to denote the number of possible string s for substring $s'[i]||s'[i+1]||...||s'[\text{len}(s')-1]$.

1. if $s'[0] = 0$, then $\text{DP}(s', 0) = 0$. i.e., we cannot find a s for string 09.
2. if $\text{len}(s') = 1$, $\text{DP}(s', 0) = 1$.
3. if $\text{len}(s') = 2$.
   (a) if $s'[1] \neq 0$ and $s' \leq 26$, then $\text{DP}(s', 0) = 2$.
   (b) if $s'[1] = 0$ and $s' \leq 26$, then $\text{DP}(s', 0) = 1$.
   (c) if $s'[1] = 0$ and $s' > 26$, then $\text{DP}(s', 0) = 0$. i.e., we cannot find a s for string 30.
   (d) otherwise, $\text{DP}(s', 0) = 1$.
4. if $\text{len}(s') > 2$, for any $i \geq 0$,
   (a) if $s'[i]||s'[i+1] \leq 26$ and $s'[i+1] \neq 0$, then $\text{DP}(s', i) = \text{DP}(s', i+1) + \text{DP}(s', i+2)$
   (b) if $s'[i]||s'[i+1] \leq 26$ and $s'[i+1] = 0$, then $\text{DP}(s', i) = \text{DP}(s', i+2)$.
   (c) if $s'[i]||s'[i+1] > 26$ and $s'[i+1] \neq 0$, then $\text{DP}(s', i) = \text{DP}(s', i+1)$.
   (d) otherwise, $\text{DP}(s', 0) = 0$. i.e., we cannot find a s for string 990.

Problem 4:

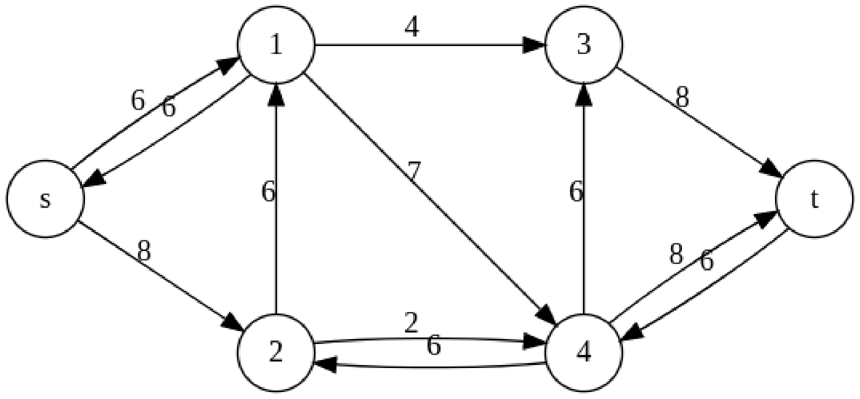
Run the Ford-Fulkerson algorithm on the flow network in the figure below, and show the residual network after each flow augmentation. For each iteration, pick the augmenting path that is lexicographically smallest. (e.g. if you have two augmenting paths $s \to 3 \to t$ and $s \to 4 \to t$, then you should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



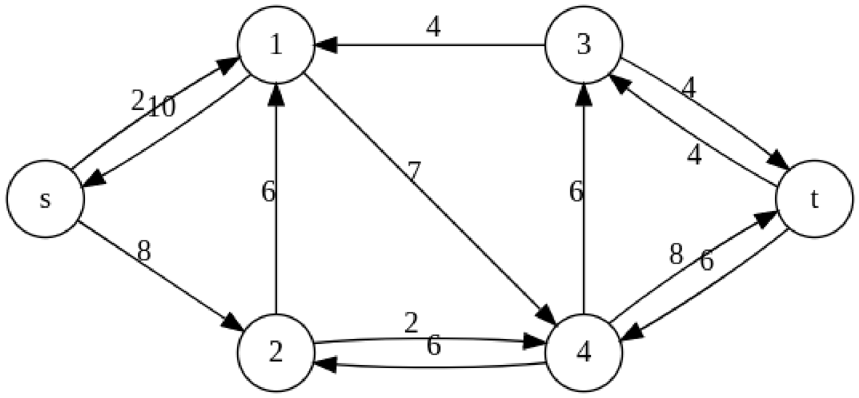
Solution:

We use $v(f)$ to denote the value of current flow f.

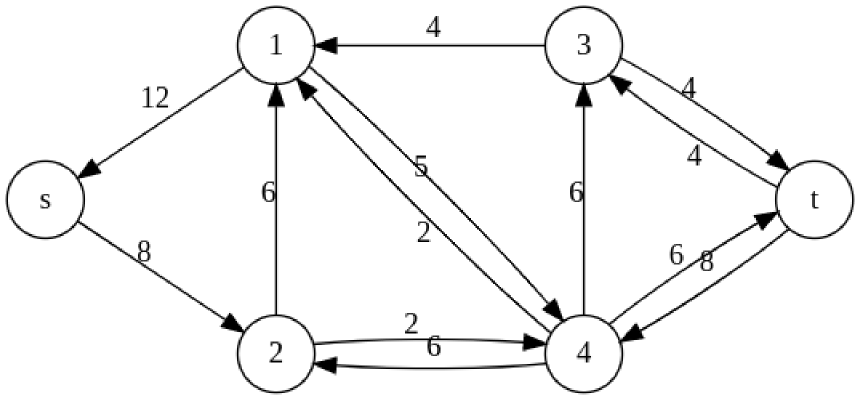
(1) Take augmenting path $s \to 1 \to 2 \to 4 \to t$, then $v(f) = 6$.



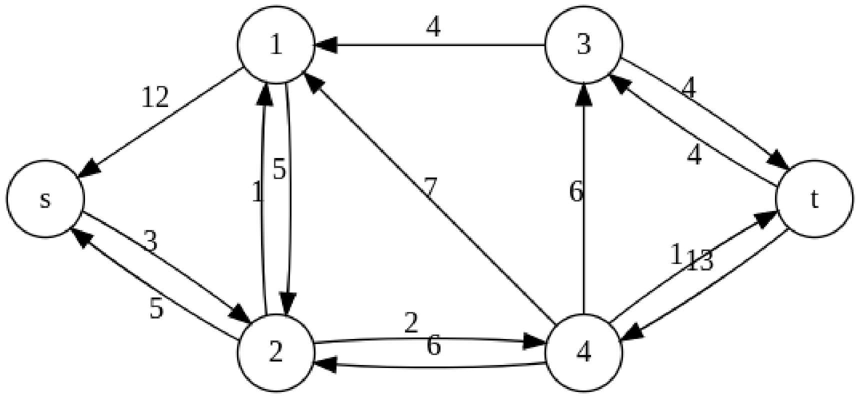
(2) Take augmenting path $s \to 1 \to 3 \to t$, then $v(f) = 6 + 4 = 10$.



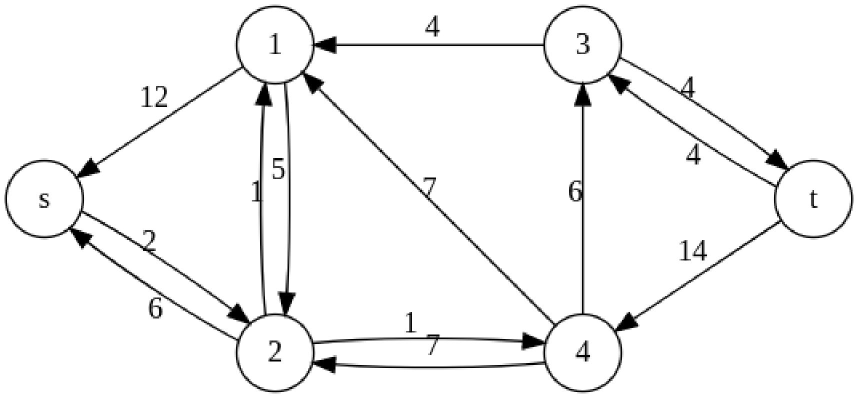
(3) Take augmenting path $s \to 1 \to 4 \to t$, then $v(f) = 10 + 2 = 12$.



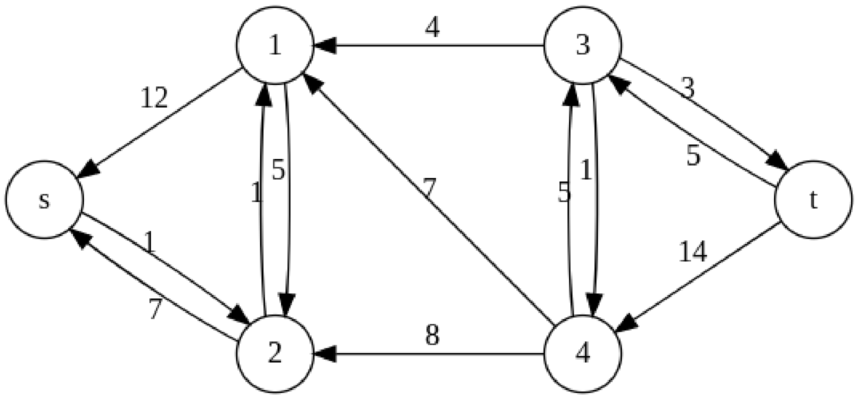
(4) Take augmenting path $s \to 2 \to 1 \to 4 \to t$, then $v(f) = 12 + 5 = 17$.



(5) Take augmenting path $s \to 2 \to 4 \to t$, then $v(f) = 17 + 1 = 18$.



(6) Take augmenting path $s \to 2 \to 4 \to 3 \to t$, then $v(f) = 18 + 1 = 19$.



Final maximum flow result 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:

First, calculate the minimal s-t cut of G. Assume the edge in minimal s-t cut is $e_1, e_2, ..., e_k$. For every edge in minimal s-t cut, reduce its capacities by one, and then recalculate the minimal s-t cut. If the new minimal s-t cut equals the original cut, G doesn't have a unique minimal s-t cut. If after traversing k edges, no minimal s-t cut equals to the original cut, G has a unique minimal s-t cut.

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:

For all squares with no obstacles, connect the source s with an edge with capacity 1 if square is white, connect the sink t with an edge with capacity 1 if square is black. For all black square and white square pairs, if a knight can attack black square from white square, the capacity is infinite, else no edge between the square pair. Calculate the minimal s-t cut. Return $n * n - k -$minimal s-t cut