ShanghaiTech University

CS240 Algorithm Design and Analysis

Fall 2021

Problem Set 2

Due: 23:59, Oct.29, 2021

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:

You have a garden and want to plant some apple trees in your garden, so that they produce as many apples as possible. There are n adjacent spots numbered 1 to n in your garden where you can place a tree. Based on the quality of the soil in each spot, you know that if you plant a tree in the i_{th} spot, it will produce exactly x_i apples. However, each tree needs space to grow, so if you place a tree in the i th spot, you can't place a tree in spots i - 1 or i + 1. What is the maximum number of apples you can produce in your garden?

Solution:

f(1) = x1, f(2) = $max\{x1, x2\}$, If we don't plant a tree in spot i, then the best way to plant trees in spots 1 to i is the same as the best way to plant trees in spots 1 to i - 1. Then, f(i) = f(i - 1). If we plant a tree in spot i, then we get xi apples from it. However, we cannot plant a tree in spot i - 1, so we are only allowed to place trees in spots 1 to i - 2. In turn, in this case we can pick the best way to plant trees in spots 1 to i - 2 and then add a tree at i to this solution to get the best way to plant trees in spots 1 to i. So we get f(i) = f(i - 2) + x_i Initialize a length n array, where the i th entry of the array will store f(i). Fill in f(1), and then use the formula f(i) = $max\{f(i-1), x_i + f(i-2)\}$ to fill out the rest of the table in order. Then, return f(n) from the table.

Problem 2:

You are given three sequences A, B and C. The length of the three sequences is m, n and m+n respectively. In other words, the length of C is the sum of the length of A and B. Design an efficient algorithm to check if A and B can be merged into C such that the order of all the letters in A and B is preserved.

Example 1: A = aabb, B = cba, C = acabbab, then your algorithm should return true. Example 2: A = aabb, B = cba, C = aaabbbc, then your algorithm should return false.

Solution:

We use Dynamic Programming to solve this problem.

1. Create a two-dim array W[m+1][n+1] (initialize 0)
2. Process all characters in A and B, using nested loop structure, and the pointers are i(in A) and j(in B)
3. If i == 0 and j == 0 then w[i][j] = 1

if i = 0 and B[j-1] = C[j-1] then w[i][j] = w[i][j-1]

if j = 0 and A[i-1] = C[j-1] then w[i][j] = w[i-1][j]
if A[i-1] = C[i+j-1] and B[j-1] != C[i+j-1] then w[i][j] = w[i-1][j]
if A[i] != C[i+j-1] and B[j-1] = C[i+j-1] then w[i][j] = w[i][j-1]
if A[i] = C[i+j-1] and B[j] = C[i+j-1] then w[i][j] = (w[i][j-1] or w[i][j-1])
After this nested loop finished, if W[len(A)][len(B)] = 1, return True. Else, Return false.

Problem 3:

You are given an integer array *coins* representing coins of different denominations and an integer *amount* representing a total amount of money.

Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.
Example:

Input: coins = [1,2,5], amount = 11

output: 3

Explanation: 11 = 5 + 5 + 1

Solution:

We use Dynamic Programming to solve this problem.
Let m = amount, n = len(coin) - 1
$dp[m][n] = min\{dp[m - coin[n]][n] + 1, dp[m][n-1]\}$

Condition is $m, n \geq 0$.

Problem 4:

There is an unknown $m \times n$ matrix A ($A_{i,j} \in \mathbb{N}$), given a_i ($a_i = \sum_{t=0}^{n-1} A_{i,t},\ i \in [0, m-1]$) and b_j ($b_j = \sum_{t=0}^{m-1} A_{t,j},\ j \in [0, n-1]$). Design an algorithm to assign values to each $A_{i,j}$. Brute force is forbidden.

Solution:

Let $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$ be the sums of the entries in the rows and columns of A, respectively. Consider the complete bipartite graph $G = (P \cup Q, E)$, where $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_m\}$. Let us make a directed network out of G. Orient all edges from P to Q, and set their capacity to

infinity. Add a source s, connect it to all vertices of P by oriented edges of capacity a_i for the vertex p_i. Analogously, add a sink t and connect all vertices of Q to it by oriented edges of capacity b_j for the vertex q_j.

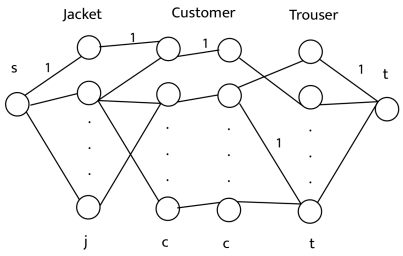
Clearly, the minimum capacity of a cut is $a_1 + \cdots + a_n = b_1 + \cdots + b_m$. Since all capacities are integral, the Ford-Furkerson theorem shows that there is an integer flow with maximum value. We can then define the entry A_{ij} of the matrix A to be the value of this flow on the edge $p_i q_j$.

Problem 5:

Suppose you are the manager of a clothing shop with j different jackets and t different trousers. One day, c customers come to the shop, each of them has their own favorite jackets and trousers but only wants to buy one jacket and one trouser of his favorite. We say that a deal is 'successful' if you sell one jacket and one trouser to a customer which both are his favorite. Design an efficient max-flow algorithm that maximizes the number of successful deals. Draw a sketch of your network and explain how you construct it.

Solution:

Construct the network with 6 layers of nodes. (1) Source s. (2) j jacket nodes, each is connected to the source node with capacity 1. (3) c customer nodes and each is split into two nodes which is connected by an edge with capacity 1, define as c_{i-left} and $c_{i-right}$. c_{i-left} connects to the jacket nodes which the i_{th} customer like, the same as $c_{i-right}$ to trouser nodes, and the capacity is also 1. (4) t trouser nodes, each is connected to the sink node with capacity 1 (5) Sink node t. See Figure.1
Run a maxflow, which is the maximum number of successful deals.



Problem 6:

Given a sequence of positive integers $x_1, ...x_n$, find

1. The length L of the longest strictly ascending subsequence in $O(n^2)$.(Hint: It can be solved by dynamic programming.)

2. How many strictly ascending subsequences of length L can be extracted from the input sequence? These subsequences must be non-overlapping. In other words, if x_i is contained in one subsequence, it cannot be contained in another subsequence.

Note: A subsequence is different from a substring in that a subsequence can be inconsecutive. For examle, $1, 3, 5$ is a subsequence of $1, 2, 3, 4, 5$.

Solution:

(1) Let $f(i)$ be the length of longest strict ascending subsequence end with x_i. $f(i) = 1 + \max_{j<i, x_j<x_i} f(j)$. Time complexity is $O(n^2)$.

(2) Construct a graph $G = (V, E)$, s.t $V = l_1, r_1, l_2, r_2, ..., l_n, r_n, s, t$,

1. $e(l_i, r_i) = 1$
2. $e(r_i, l_j) = 1$, if $j < i, x_j < x_i$,f(j)=f(i)+1
3. $e(s, l_i) = 1$, if $f(i) = 1$
4. $e(r_i, t) = 1$, if $f(i) = L$

Run a max flow on graph G.