Western University

CS 2210B - Data Structures and Algorithms
Winter 2022, Instructor: Dr. Saad Bin Ahmed

Assignment 2 (Programming)

Total Marks: 55
Due Date: February 16, 2022

Car Fuelling Problem

1 Overview

In this assignment you will implement a car fuelling problem. Suppose, you are travelling to another city that is located d miles away from your home city. Your car can travel,

1. At most m miles on a full tank and you start with a full tank.
2. Along your way, there are gas stations at distances $stop1$, $stop2$, . . . , $stop\ n$ from your home city. What is the minimum number of refills needed?

2 Problem Description

Input Format: The first line contains an integer d. The second line contains an integer m. The third line specifies an integer n. Finally, the last line contains integers $stop1$, $stop2$, ... , $stop\ n$.

Output Format: Assuming that the distance between the cities is d miles, a car can travel at most m miles.

- On a full tank, and there are gas stations at distances $stop1, stop2, ..., stop\ n$
- along the way, output the **minimum number of refills** needed. Assume that the car starts with a full tank. If it is **not possible** to reach the destination, **output** -1.

Constraints

$1 \leq d \leq 110, \quad 1 \leq m \leq 415, \quad 1 \leq n \leq 300, \quad 0 < stop1 < stop2 < ... < stop\ n < d$

Test Case 1:
Input:

>> 950
>> 400
>> 4
>> 200 375 550 750

Output:

>> 2

The distance between the cities is 950, the car can travel at most 400 miles on a full tank. It suffices to make two refills: at points 375 and 750. This is the minimum number of refills as with a single refill one would only be able to travel at most 800 miles.

3 Exercises

Q 1. (5 marks) Write a java program that will implement the scenario when the tank is full and there are gas stations at distances $stop1, stop2$, ... , $stop\ n$. (Hint, Test Case 1)

- You will design two test cases similar to "Test case 1". When your program will execute then it will first show the numbers selected for each test case. This will be the guidance for a user regarding input.

Implementation of Hash Table using Hash Function

Q 2. (14 marks) Assume you have started a new job at local bank as Java programming specialist and you have asked to design an application that will assist the staff to extract the client information, update client's record and delete it if necessary. But as there would be large number of clients and to manage all client's record efficiently would depend on the algorithm you choose to implement your idea. If you boss says to use hash table for the implementation of your application so how you will implement the functions? (Hint! Also consider collision strategy in your implementation)

- You will write a program that will depict the search, insertions and deletion functions. Design two test cases for verification of your implementation.

Find minimum number of coins that make a given value

Q 3. (7 marks) Given a value V, if we want to make a change for V cents, and we have an infinite supply of each of coin C = C1, C2, .., Cm valued coins, what is the minimum number of coins to make the change? If it's not possible to make a change, print -1.

```
Test Case

Input: coins[] = {25, 10, 5}, V = 30}
Output: Minimum 2 coins required
We can use one coin of 25 cents and one of 5 cents
```

Coin Change Problem

Q 4. (14 marks) Given a value N, if we want to make change for N cents, and we have infinite supply of each of $S = \{S1, S2, .., Sm\}$ valued coins, how many ways can we make the change? The order of coins doesn't matter. For example, for $N = 6$ and $S = \{1, 2, 3\}$, there are six solutions: $\{1, 1, 1, 1, 1, 1\}, \{1, 1, 2, 2\}, \{1, 2, 3\}$,
$\{2, 2, 2\}, \{3, 1, 1, 1\}, \{3, 3\}$. So output should be 6. For $N = 10$ and $S = \{2, 5, 3, 6\}$, there are five solutions: $\{2, 2, 2, 2, 2\}, \{2, 2, 3, 3\}, \{2, 2, 6\}, \{2, 3, 5\}$ and $\{5, 5\}$. So the output should be 5.

Optimal Substructure To count the total number of solutions, we can divide all set solutions into two sets.

1. Solutions that do not contain mth coin (or Sm).
2. Solutions that contain at least one Sm.
3. Let count(S[ ], m, n) be the function to count the number of solutions, then it can be written as sum of count(S[ ], m-1, n) and count(S[ ], m, n-Sm).

Therefore, the problem has optimal substructure property as the problem can be solved using solutions to subproblems.

Algorithm

Here each coin of a given denomination can come an infinite number of times. (Repetition allowed), this is what we call UNBOUNDED KNAPSACK. We have 2 choices for a coin of a particular denomination, either i) to include, or ii) to exclude.

But here, the inclusion process is not for just once; we can include any denomination any number of times until $N < 0$.

Basically, If we are at s[m-1], we can take as many instances of that coin (unbounded inclusion) i.e count(S, m, n – S[m-1]) ; then we move to s[m-2]. After moving to s[m-2], we can't move back and can't make choices for s[m-1] i.e count(S, m-1, n).

Finally, as we have to find the total number of ways, so we will add these 2 possible choices, i.e count(S, m, n – S[m-1]) + count(S, m-1, n) ; which will be our required answer.

4 Grading

Your grade will be computed as follows.

- Program must be written in java and should comply according to the provided instructions, compiles, and produces a meaningful output: (40 marks)
- Coding style including comments where necessary: (10 marks)
- Clearly present 2 test cases initially for each problem when execute the program: (5 marks)

5 Submission Instructions

- There will be four separate java files with the same name as question number like Q1.java, Q2.java .... so on.
- Submit your assignment 2's four java programs via OWL.

6 Important Note

- To be eligible for full marks, your programming assignment must run on the departmental computing equipment. You may develop assignments on your home computer, but you must allow for the amount of time it will take to get the final programs working on Computer Science's machines. All programming assignments must be submitted through OWL.

- The late penalty for programming assignment is [2.5i] (2.5 to the i-th power, rounded to the nearest integer), where i $>$ 0 is the number of days you are late. So if you hand in your assignment 1 day late, you will be penalized 3%, a delay of 2 days will decrease your grade by 6%, 3 days is penalized 16% and 4 days takes 39% off your grade. You cannot be more than 4 days late.