Assignment 5

Michele Fattoruso - Z1840898

November 2018

1 (4 points) Gas stations optimization

There are n gas station S_1, ... , S_n along I-80 from San Francisco to New York. On a full tank of gas your car goes for D miles. Gas station S_1 is in San Francisco, each gas station S_i, for $2 \leq i \leq n$, is $d_i \leq D$ miles after the previous gas station $Si - 1$, and gas S_n is in New York.

1.1 Explanation

Considering the case where we do a full at the gas station S_1, in this case we know that we will be able to drive for D miles. Said so, we can approach the problem by decreasing the remaining mileage for each gas station S_i. When re reach that the remaining mileage minus the distance with the next gas station is less than zero, it means that we won't have enough gas to reach the next station, fo we full the tank. We can iterate this process until we reach the end location, so if the number of miles D necessary to reach the gas station S_n, is less than D. This means that we reach the destination.

1.2 Code

let's consider that the vector station, contains the distance between the current station and previous one, and the value tankMileage contains the number of miles that the car is able to drive with a full tank.

```
int getMinGasStops(vector<double> station, int tankMileage){
    int numOfStops = 0;
    int n = station.length();
    //remainingMileage = -1 if first station is a stop,0 otherwise
    int remainingMileage = -1;
    for(int i=0; i < n; i++){
        if( remainingMileage - station[i] < 0){
            /*We consider as we refilled gas at i-1
              station, and we reached the station i*/
            remainingMileage = tankMileage - station[i];
            numOfStops++;
        }
    }
    return numOfStops;
}
```

1.3 Why it gives the optimal result

Considering that with the following approach we fill the tank of the car only when the next gas station can't be reached with the current remaining mileage.

So, let's consider that our algorithm is able to drive from S_1 to S_i with x=1 refills, saying that out algorithm doesn't returns the optimal solution, means that it does exist a way to travel from S_1 to S_i withouth refilling the car. Given that our algorithm drive for the longest possible distance without refilling the tank, it means that there isn't any other solution to reach S_i in less refills, this means that x is the optimal solution for the distance between S_1 and S_i. This reduces the problem on calculating the minimum number of refills between S_i ans S_n. We can iterate the same process mentioned before, to demonstrate that out algorithm will always return the optimal solution.

1.4 Complexity

If we consider the number of gas stations to be equal to n, We can se that the inner loop executes n times, so the time complexity of this algorithm is $\mathcal{O}(n)$, as we go through the list of all the stations just once. The algorithm doesn't use any additional data structure, beside some variables, so the space complexity is equal to $\mathcal{O}(1)$.

2 Coin changing

Consider the problem of making changes for n cents using the fewest number of coins. Assume that each coin's value is an integer.

2.1 (2 points) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies.

A greedy approach to making change for n cents, is given by using always the biggest coin possible between the available coin values. for example, given the coin values [1, 5, 10, 25] and n=43:

n=43 $\rightarrow$ 1 Quarter and 18 Remainder
Remainder 18 $\rightarrow$ 1 Dime and 8 Remainder
Remainder 8 $\rightarrow$ 1 Nickel and 3 Remainder
Remainder 3 $\rightarrow$ 3 Pennies

2.1.1 Recurrence

Given n cois, we start form the greatest coin, and we take

$$X = \left\lfloor \frac{n}{coinValue} \right\rfloor \tag{1}$$

where X is the number of coins we take for that coin value, and then we subtract X*coinValue to the original problem value, and keep iterating over it, until we reach that the remainder of the operation is equal to zero.

2.2 (2 points) Suppose that the available coins are in the denominations that are powers of c

i.e., the denominations are $c^0, c^1, ..., c^k \ \forall c > 1, k \geq 1$.

2.2.1 Describe a Greedy Algorithm

So, as the problem before, we can apply the same tecnique, of using the biggest coin denomiation first, and then use the other coins available to change the remainder

for example, for $c = 3$ and $k = 3$,the coin values are [1, 3, 9, 27].Given $n = 33$:

n=33 $\rightarrow$ 1 coin 27 and 6 Remainder
Remainder 6 $\rightarrow$ 0 coin 9 and 6 Remainder
Remainder 6 $\rightarrow$ 2 coin 3 and 0 Remainder

This will calculate than we need 3 coins to change n=33 with the current denomination.

2.2.2 2 bonus points: Argue that the greedy algorithm always yields an optimal solution.

Given a specific coin denomination, we can see that the greedy algorithm doesn't return the optimal solution, when given a denomination V_k, we have that for the denomination immediately less than it is $V_k < 2 * V_(k-1)$

What we know from the coin denominations is thst each coin is multiple of each other

- for $c = 2$, The system will always yields an optimal result, for any value of k. If a number n can be composed of chage with the coin denomination V_i, then the previous coin denomination V_{i-1} can produce the same result by using $2 * (coinsusedfor V_i denomination)$, so using the double of coins used by V_i.This proves that our greedy approach returns the optimal result
- for $c = 3$, we have that, if a value n can be changed by a coin V_i, then we know that the same amount can be changed with $c * V_{i-1}$ coins
- We can iterate this approach for each $c > 1$ and $k \geq 1$.

As long as each coin denomination is multiple of the previous one, the system will always yields the optimal solution

2.3 Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution

Example 1:
$Coins = [1, 5, 6, 9]$
$n = 11$
$GreedyResult = 9 + 1 + 1 \rightarrow 3$ Coins
$OptimalResult = 6 + 5 \rightarrow 2$ Coins
Example 2
$Coins = [1, 15, 25]$
$n = 30$
$GreedyResult = 25 + 1 + 1 + 1 + 1 + 1 \rightarrow 6$ Coins
$OptimalResult = 15 + 15 \rightarrow 2$ Coins