CS2040S: Data Structures and Algorithms

Discussion Group Problems for Week 13

For: April 14–April 18

Goals:

- Dynamic Programming

Problem 1. Fibonacci As Warmup

To get you warmed up, let's quickly re-cap Fibonacci yet again.

Problem 1.a. Here's some pseudocode for computing Fibonacci:

```
int fibVer1(int i) {
    if (i == 0 || i == 1) {
        return 1;
    }
    return fibVer1(i - 1) + fibVer1(i - 2);
}
```

Quick revision: What is the running time of `fibVer1`?

Problem 1.b. Let's consider the following version instead:

```
int fibVer2Helper(int i, int[] arr) {
    if (arr[i] != -1) {
        return arr[i];
    }
    if (i == 0 || i == 1) {
        arr[i] = 1;
        return 1;
    }
    int answer = fibVer2Helper(i - 1, arr) + fibVer2Helper(i - 2, arr);
    arr[i] = answer;
    return answer;
}
```

```
int fibVer2(int i) {
    int[] memo_table = new int[i + 1];
    for (int idx = 0; idx <= i; ++idx){
        memo_table[idx] = -1; // mark as unsolved
    }
    return fibVer2Helper(i, memo_table);
}
```

What is the running time of `fibVer2`? What is the space complexity of `fibVer2`?

To help you answer the question on running time, focus on the 2 following things:

1. To solve for input i, how many sub-problems are we solving?
2. How much does it cost to solve each subproblem?

Problem 1.c. Try coming up with an iterative version instead, without recursion.

Problem 2. Fancy Paintings

Xenon, who recently earned a fortune from teaching CS2040S, has decided to purchase n paintings, where the i-th painting has height h_i metres and width w_i metres. He now wishes to build a building to display his paintings. Unfortunately, he has quite limited land, and thus he needs to build multiple floors to display all the paintings.

Each floor of the building has one display wall k metres long. He can fit as many paintings side by side as long as the total width of the paintings does not exceed k metres. Furthermore, Xenon wishes to provide a curated experience, and so there are two constraints for the paintings:

- We cannot reorder the paintings. Painting i must come before painting $i+1$.
- We cannot stack paintings. Each floor will only have one row of paintings.

The height of each floor is solely determined by the height of the tallest painting for that floor. To save on construction costs, he needs to minimize the height of the building (i.e. the sum of the height of all the floors).

Example: Given $k = 10$ and the following paintings:

- Painting 1: $h_1 = 5, w_1 = 2$
- Painting 2: $h_2 = 3, w_2 = 6$
- Painting 3: $h_3 = 4, w_3 = 4$

All three paintings can't be on the same floor, since $w_1 + w_2 + w_3 = 12 > k$. Thus, there are three possible arrangments:

- All three paintings on different floors. Then the total height is $5 + 3 + 4 = 12$.
- Paintings 1 and 2 on the first floor, and painting 3 on the second. Then total height is $5 + 4 = 9$
- Painting 1 on the first floor, and paintings 2 and 3 on the second. Then total height is $5 + 4 = 9$.

Thus, the minimal height of the building is 9.

Problem 2.a. Show that the following greedy approach to this problem does not work:

- Keep inserting paintings to the current floor.
- If the next painting does not fit, create a new floor.

Problem 2.b. Let $dp(i)$ be the height of the building if we only consider the first i paintings. Write the recurrence relation and initial condition for $dp(i)$. If you need additional variables, please state them clearly.

Problem 2.c. Write pseudocode for your recurrence relation. What is the worst case time and space complexity for your code?

Problem 3. Road Trip

Relevant Kattis Problems:

- https://open.kattis.com/problems/adventuremoving4
- https://open.kattis.com/problems/highwayhassle
- https://open.kattis.com/problems/roadtrip

You are going on a road trip. You get in your trusty car and drive to Panglossia, where you will spend a nice vacation by the beach.

You have already found the best route from your home to Panglossia (using Dijkstra's Algorithm). Next, you need to determine where you can buy petrol along the way. On the road between your home and Panglossia, there are a set of n petrol stations, the last of which is in Panglossia itself. Assume that your trip is complete when you reach the last station.

By searching on the internet, you find for each station, its location on the road and the cost of petrol. That is, the input to the problem is n stations, $s_0, s_1, \ldots, s_{n-1}$, along with $c(s_i)$, which specifies the cost of petrol at station s_i, and $d(s_i)$, which specifies the distance from your home to station s_i.

The tank of your car has a capacity of L liters. You begin your trip at home with a full tank of petrol. Your car uses exactly 1 liter per kilometer. Along the way, you must ensure that your car always has petrol (though you may arrive at a station just as you run out of petrol). Note that you do not need to fill your tank at a station. You can buy any amount of petrol, as long as it's within the capacity of your tank.

Your job is to determine **how much petrol to buy at each station** along the way so as to minimize the cost of your trip.

You may assume, for simplicity, that L and all the given distances are integers, i.e., all the quantities are integers.

Here are two examples, a simple one and a more complicated one.

Example 1: Your tank holds 6 liters, and there are 3 stations:

```
5km: $1
6km: $2
7km: $4
```

In this case, the best solution costs one dollar, where you purchase one liter of petrol at the first station at a cost of 1 per liter.

Example 2: Your tank holds 20 liters, and there are 10 stations:

```
7km:  $3
17km: $5
20km: $2
23km: $1
39km: $5
48km: $4
66km: $9
83km: $9
88km: $4
92km: $1
```

In this case, the best solution is to purchase petrol at each station as follows, for a total cost of 327 dollars:

```
0
0
3
20
5
20
15
5
4
0
```

Hint: To solve this problem, think about how to calculate $DP(s_j, k)$, the minimum cost to get from station s_j to the destination, assuming you have k liters of petrol left.

Problem 4. Plagiarism Panic!

By now, you should be well aware of the Collaboration Policy on solving the Problem Sets. We hope that you've been following this policy throughout the semester!

Let's say (hypothetically!) that two students decide to cheat anyway. How might we detect that their codes are similar? In other words, given two strings of code A and B, how do we determine their similarity factor?

One way we can define the similarity factor is through shared substrings. We say the string S is a *substring* of string T if S appears as a consecutive sequence of characters within T. For example, for the string `CS2040S`, some possible substrings are `CS`, `2040`, `S204`, `CS2040S`, etc.

Two strings A and B are said to have a shared substring C if C is a substring of A and C is a substring of B. For example, `CS2040S` and `CS2030S` both share the substring `S20`.

Problem 4.a. Given two strings A and B, come up with an algorithm to determine the length of their longest shared substring. For example, the length of the longest shared substring of `dynamic` and `programming` is 2. There are two such shared substrings: `am` and `mi`.

Let $f(i, j)$ be the length of the longest shared substring that ends at the i-th character of string A and ends at the j-th character of string B.

Write the recurrence relation and base cases for $f(i, j)$. What is the answer to the original problem?

Write pseudocode for your recurrence relation. What is the time complexity of your code?

Problem 4.b. Another way we can define the similarity factor is through shared subsequences. We say the string S is a *subsequence* of string T if S can be obtained by removing some (or no) characters of T. For example, for the string `CS2040S`, some possible subsequences are `C4`, `SS`, `S40S`, `CS2040S`, etc.

Given two strings A and B, come up with an algorithm to determine the length of their longest shared subsequence. For example, the length of the longest shared subsequence of `dynamic` and `programming` is 3 (the string `ami`).

Similar to the previous part, let $g(i, j)$ be the length of the longest shared subsequence that ends at the i-th character of string A and ends at the j-th character of string B. What is the recurrence relation now?

Write pseudocode for your recurrence relation. What is the time complexity of your code?

Problem 4.c. Let's use a different approach for the DP. Define $h(i, j)$ to be the length of the longest shared subsequence that ends *before* (or at) the i-th character of string A and ends *before* (or at) the j-th character of string B.

Write the recurrence relation and pseudocode for $h(i, j)$. What is the time complexity of your code?

Note: The moral of the story is that choosing the right subproblem can make your algorithm more efficient!

- This is it for CS2040S this semester. All the best! -