Problem Set 2: Dynamic Programming and Greedy Algorithm

Name: Isaac Boaz (Solo)

May 2, 2024

Q1: You're helping Santa Clauses to distribute gifts to children. For ease of delivery, you are asked to divide n gifts into two groups such that the weight difference of these two groups is minimized. The weight of each gift is a positive integer.

(a) Please design an algorithm to find an optimal division minimizing the value difference. The algorithm should find the minimal weight difference and the groupings in $O(nS)$ time, where S is the total weight of these n gifts.

We can simplify this question down by realizing that it is similar to a problem we've seen prior in class: the Knapsack Problem.

Since we're splitting the n gifts into two groups, the *best* weight to minimize the sum of weight differences for each group is $S/2$. Thus, we can imagine this problem as the Knapsack Problem with a maximum weight of $S/2$ (i.e we want to get as close as possible to $S/2$ without going over). One small simplification we can additionally apply is ignoring the value part of the Knapsack Problem.

(b) Please write down the recurrence relation you derived in (a), and use it to complete a DP table for solving the following problem instance.

Gift i	1	2	3	4
Weight w_i	2	2	4	3

Constructing a DP table for this problem, we can use the following recurrence relation:

$$dp[i,w] = \begin{cases} 0 & \text{if } i = 0 \text{ or } w = 0 \\ dp[i-1,w] & \text{if } w_i > w \\ \max(dp[i-1,w], dp[i-1,w-w_i] + w_i) & \text{if } i > 0 \text{ and } w \geq w_i \end{cases}$$

Calculating our dp table, let's set $w = \frac{2+2+4+3}{2} = \frac{11}{2} = 5.75$ Let's just round this down to 5 for simplicity (considering we don't have fractional weights).

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	2	2	2	2
2	0	0	2	2	4	4
3	0	0	2	2	4	4
4	0	0	2	3	4	5

We've successfully found an optimal solution of 5, which means that the two groups of gifts will have a weight difference of.

$$\begin{aligned} |5 - (S - 5)| &= |5 - (11 - 5)| \\ &= |5 - 6| \\ &= 1 \end{aligned}$$

To find the first group of gifts, we can backtrack through our DP table to find the gifts that were included in the optimal solution. Starting at the bottom right corner of the table, we can check if the value at $dp[i, w]$ is equal to $dp[i - 1, w]$. If it is, then the gift was not included in the optimal solution. If it is not, then the gift was included in the optimal solution. Thus we subtract the weight of the gift from w and north-west to $dp[i - 1, w - w_i]$. We continue this process until we reach the top left corner of the table.

Doing this, we find that the first group of gifts is {2, 3} and the second group of gifts is {2, 4}.

Q2: Due to a zombie virus outbreak, some cities have been occupied by zombies and are no longer safe. You and your survivor team need to travel through several cities to get to a far away shelter.
There are n cities forming a line topology. You are at city 1 now, and the shelter is at city n. The location of city i is $L[i]$, and $L[i] < L[j], \forall 1 \le i < j \le n$. $z[i] = 1$ indicates city i has been occupied by zombies; otherwise, $z[i] = 0$ indicates the city is still safe to stop at night.

If you plan to move at most 100 miles a day, and you need to rest at a safe city at night, please design a greedy algorithm to pick the cities for resting at night so that you can arrive at the shelter as soon as possible. Your algorithm should run in O(n) time. Please show that your algorithm has the greedy choice property.

For a greedy algorithm, we want to travel the maximum distance possible each day. This means we want to travel as far as possible each day, and rest at the closest safe city.

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ \max_{i \le j \le n} j \ \forall L[j] < L[c[i]] + 100 \text{ and } z[j] = 0 & \text{otherwise} \end{cases}$$

```
c = []

def get_city(i):
    if i == 1:
        return 1
    if len(c) > i:
        return c[i]
    n = len(cities)
    farthestDist = -1
    farthestCity = -1
    for j in range(i, n):
        if infected[j]:
            continue
        if cities[j] - cities[get_city(i - 1)] > 100:
            break
        dist = cities[j] - cities[get_city(i - 1)]
        if dist > farthestDist:
            farthestDist = dist
            farthestCity = j
    if farthestCity == -1:
        raise Exception("No city found")
    c.append(farthestCity)
    return farthestCity
```

To show that this algorithm has the greedy choice property, we can show that the optimal substructure is maintained. This is done by showing that the subproblem of finding the optimal solution for the next city is the same as the subproblem of finding the optimal solution for the current city. This is demonstrated in the algorithm above, where we find the farthest city that is safe to rest at. This is the greedy choice, as we want to travel as far as possible each day.