1 UVA Problem 10561: Ferry Loading

Background

Before bridges were common, ferries were used to transport cars across rivers. River ferries, unlike their larger cousins, run on a guide line and are powered by the river's current. Two lanes of cars drive onto the ferry from one end, the ferry crosses the river, and the cars exit from the other end of the ferry.

The cars waiting to board the ferry form a single queue, and the operator directs each car in turn to drive onto the port (left) or starboard (right) lane of the ferry so as to balance the load. Each car in the queue has a different length, which the operator estimates by inspecting the queue. Based on this inspection, the operator decides which side of the ferry each car should board, and boards as many cars as possible from the queue, subject to the length limit of the ferry. Your job is to write a program that will tell the operator which car to load on which side so as to maximize the number of cars loaded.

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The first line of input contains a single integer between 1 and 100: the length of the ferry (in metres). For each car in the queue there is an additional line of input specifying the length of the car (in cm, an integer between 100 and 3000 inclusive). A nal line of input contains the integer 0. The cars must be loaded in order, subject to the constraint that the total length of cars on either side does not exceed the length of the ferry. Subject to this constraint as many cars should be loaded as possible, starting with the rst car in the queue and loading cars in order until no more can be loaded.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

The first line of outuput should give the number of cars that can be loaded onto the ferry. For each car that can be loaded onto the ferry, in the order the cars appear in the input, output a line containing "port" if the car is to be directed to the port side and "starboard" if the car is to be directed to the starboard side. If several arrangements of the cars meet the criteria above, any one will do.

1.1 Mathematical Formulation

We are given an input of L for length of ferry, followed by an input of N cars in the form $l_1, l_2, ..., l_N$ of which $n, 1 \leq n \leq N$ will fit onto the ferry if optimally placed. The algorithm will decide what the number n is as well as whether each of these n cars should go to port or starboard in the order seen.

1.2 Solution

Important Confusing Data Structures:

- boolean[2][LENGT+1] **lastUsed** : keeps track of the last seen possible car lengths
- boolean[202][LENGT+1] **solutions** : yes
- int[202] **carLen** : Keeps track of the car lengths that have been seen thus far. carLength[1] $= l_1$

We treat this problem as the classic napsack dynamic programming problem except, instead of having one napsack we have two. Therefore the relationship we will be doing is for each open state and given a new value v_i:

$$state(i, v_1, v_2) = \begin{cases} state(i+1, v_1 + v_i, v_2), \ if \ v_1 \ + \ v_i \ \leq \ L \\ state(i+1, v_1, v_2 + v_i), \ if \ v_2 \ + \ v_i \ \leq \ L \end{cases}$$

If we use this exsact model we will have to use data structures for storage in the 3^{rd} degree, so as a trick to stay on a 2-Dimentional grid we use a trick. Since we know that $|v_1| + |v_2| + |v_i| = \ell_i$ when item v_i has been added where $\ell_i = \sum_{i=1}^{k} l_i$ and $k \leq n$. This fact is useful because then we just have to keep track of two things: ℓ_i and either $|v_1|$ *or* $|v_2|$. For this specific problem, $v_1 = port$ and $v_2 = starbord$ we can have the following:

$$state(i, v_2) = \begin{cases} state(i+1, v_2 + v_i), \ if \ |v_2| \ + \ |v_i| \ \leq \ L \implies starboard \\ state(i+1, v_2), \ if \ \ell_i \ - \ |v_2| \ \leq \ L \implies port \end{cases}$$

This way we only need to know v_2 length of starboard side, v_i length of current car, and ℓ length of total cars seen thusfar. This is only to build the table which we will in turn backtrack to determine for each car wheather to go to starboard or port.

Algorithm 1 Main and Build

```
procedure BUILD
    for numberOfCases do
        LENGTH ← Integer.parseInt(line) * 100
        lastUsed, solutions, carLen ← initialized (see above)
        booleandone ← true
        curRow, invRow, curCar, n, sumLen, lastLen ← initialized to 0
        while true do
            currentLen ← Integer.parseInt(in.nextLine)
            if currentLen == 0 then
                break
            if done then
                continue
            invRow ← curRow
            curRow ← (curRow + 1) % 2
            curCar++
            carLen[curCar] ← currentLen
            sumLen += currentLen
            Arrays.fill(lastUsed[curRow], false)
            boolean canLoad ← false;
            for len ∈ 0..LENGTH do
                if !lastUsed[invRow][len] then
                    continue
                intpos ← len + currentLen
                if pos ≤ LENGTH then // Starboard
                    lastUsed[curRow][pos] = true
                    lastLen = pos
                    canLoad = true
                if sumLen − len ≤ LENGTH then // Port
                    lastUsed[curRow][len] = true
                    solutions[curCar][len] = true
                    lastLen = len
                    canLoad = true
            if !canLoad then
                done = true
            else
                n++
        BACKTRACK(n, lastLen, carLen, solutions)
```

The next and final step then is to bakctrack through the built solutions 2-D array and determine (from the last element) whether each car should go to starboard or port side. Now from the build phase we note that we have only been marking down true on the solutions array if the specified car can go to port. Another thing that we keep in mind is that we break out of the search once we have found the max number of cars we can place on the ferry, we will simply traceback through the algorithm starting with our last seen option.

Algorithm 2 Backtrack and Print

```
procedure BACKTRACK(int n, int lastLen, int[ ] carLen, boolean[ ][ ] solutions)
    backtrack ← new boolean[n+1]
    for i = n..1 do
        if !solutions[i][lastLen] then
            lastLen -= carLen[i]
            backtrack[i] = false
        else
            backtrack[i] = true
    PRINT(n)
    for i = 1..n do
        if !backtrack[i] then
            PRINT(starboard)
        else
            PRINT(port)
```

1.3 Correctness

Proposition 1.
propose

Proof.
Using the fact that □

1.4 Analysis

For the following analysis, we will say that..

Proposition 2. *The space complexity of this algorithm is* ***O(..)***

Proof.
This is due to the fact that all of our data is stored in data structures:

- cause: reason $\Longrightarrow$ *complexity*

Giving us a space complexity of **O(..)**

□

Proposition 3. *The time complexity of this algorithm is* ***O(..)***

Proof. This is the case because our algorithm...

Giving us a time complexity of **O(..)**

□

1.5 An Example