Sherlock and Trip - Hackerearth

Wednesday, March 4, 2020 11:42 PM

The Problem:

Sherlock has decided to go on a trip with Watson. They are initially at position 0 and they plan to travel to position A. They have decided to travel in a car. The car they are using has a tank capacity of L litres. The car uses one litre petrol to travel one unit distance. There are m petrol pumps in the route from 0 to A. If at any position, the petrol tank is empty and there is no petrol pump at that position, then the car will stop and they couldn't complete the trip. They doesn't want the trip to fail and so they are finding the number of possible ways to successfully complete the trip. Since, the answer can be too large, Sherlock wants the answer modulo
10^9 + 7(1000000007).

Notes:-
1. Two ways are considered different, if there is atleast one petrol pump where petrol was filled in first way, while petrol was not filled in the second way.
2. At any instant the petrol in the car can't exceed the tank capacity of the car.
3. Even if the petrol tank becomes empty when the car reaches position A, it is considered a successfull trip.
4. Initially, the petrol tank is completely filled.
5. We can fill any amount of petrol at any petrol pump, but maximum petrol in the car can't exceed the tank capacity.
6. We don't consider different amount of petrol filled as different ways. For e.g.-Filling one litre petrol at position 1 or filling two litre petrol at position 1 will be considered the same way.

Note:- Use of fast I/O is recommended.

See the sample case for better understanding.

Input format:

First line contains an integer T, denoting the number of test-cases.

Next 4T lines contains the testcases as shown below:-

First line contains an integer A, denoting the destination.

Second line contains an integer L, denoting the tank capacity.

Third line contains an integer m, denoting the number of petrol pumps

Fourth line contains m integers p1, p2, ..., pm. Here, pi indicates location of ith petrol pump.

Output format:

For each testcase, print the number of ways to have a successfull trip modulo 10^9 + 7 in a new line.

Constraints:

1 <= T <= 10

1 <= m <= 1000000

1 <= A, L <= 1000000000(10^9)

0 < pi < A

It is guaranteed that no two petrol pumps will be at same location.

Subtasks:

Subtask #1 (10 points) : 1 <= m <= 20

Subtask #2 (10 points) : 1 <= m <= 1000

Subtask #3 (80 points) : Original Constraints

SAMPLE INPUT

```
1
10
5
3
4 6 2
```

SAMPLE OUTPUT

```
3
```

Explanation
There are three ways for a successfull trip. The ways are as follows :-
{4, 6, 2} - When we reach the petrol pump at position 2, our tank will have 3 litre petrol. We can fill one litre petrol here and then go to petrol pump at position 4. Now, our tank will have two litre petrol. We can again fill one litre petrol here. Now, when we reach petrol pump at position 6, our tank will have one litre petrol. We can fill three litre petrol here and successfully reach destination. Similarly, {6, 2} and {4, 6} will also help in a successfull trip. One of the way that can't help for a successfull trip is {4, 2} as the petrol tank will get empty at position 9 even if we fill the tank to the fullest at position 4 and so we cant reach our destination.

The Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MOD 1000000007

int comp (const void * elem1, const void * elem2)
{
   long int a = *((long int*)elem1);
   long int b = *((long int*)elem2);
   return (a > b) - (a < b);
}
```

```
int main(){
        int caseCount;
        long int dist, tankCapacity, bunksLen, resultSum, nextNSum, rightLimit, i;
        long int bunks[1000001];
        long int* sums = (long int*)malloc(1000001 * sizeof(long int));

        scanf("%d", &caseCount);

        while(caseCount > 0) {
           scanf("%ld %ld %ld", &dist, &tankCapacity, &bunksLen);

           for(i=0; i<bunksLen; i++) {
              scanf("%ld", &bunks[i]);
           }

           qsort(bunks, bunksLen, sizeof(long int), comp);

           bunks[bunksLen] = dist;
           sums[bunksLen] = 1;
           rightLimit = bunksLen;
           nextNSum = 1;
           resultSum = 0;
           for(i=bunksLen; i>=0; i--) {

              while(bunks[rightLimit] > (bunks[i] + tankCapacity)) {
                 nextNSum = (nextNSum - sums[rightLimit] + MOD) % MOD;
                 rightLimit--;
              }

              if(rightLimit <= i && i != bunksLen) {
                 resultSum = 0;
                 break;
              }

              sums[i] = nextNSum;

              if(i != bunksLen)
                 nextNSum = (nextNSum + sums[i])  % MOD;

              if(bunks[i] <= tankCapacity)
                 resultSum = (resultSum + sums[i]) % MOD;
           }

           printf("%ld\n", resultSum);
           caseCount--;
        }
}
```

The Stats:

Score

6.0

Time (sec)
4.21944

Memory (KiB)
23636

Language
C

Note: The score is not full because two test cases pass at 2.007 secs when the limit is 2.000 secs. Ridiculous. Using my custom merge sort results in the same. I've tried some other languages and optimizations but to no avail. Maybe I'll revisit it someday.