Roy and Sweets - Hackerearth

Monday, March 23, 2020 11:53 PM

The Problem:

Its Diwali time and Little Roy's family is having a lot of guests. Guests come in families. Each guest family has M members. Roy's task is to serve sweets to them. Roy has N different sweets and each sweet has some specific sweetness level S and quantity Q.

Each guest family will give R rupees (as a token of gratitude) to Roy if following conditions are satisfied:

1. The sweetness level of sweet served is greater than or equal to the members in the guest family
2. Every member should get sweet of one particular sweetness level
3. Every member should get equal quantity of sweets

where R = 100 * Quantity of sweet(s) each member had.

After each guest family has left, Roy's Mom makes sure that the quantity of sweets that guest family had is restored to its original value.
Your task is to find R - maximum amount in rupees Roy has after all the guests have come and left.

Input:
First line will contain integer N - number of different sweets.
Next N lines will contain two space separated integers S and Q
S - Sweetness level of the sweet (this will be distinct for each sweet)
Q - Quantity of that sweet
This is followed by an integer G - number of guest families.
Next G lines will contain integer M, indicating number of members in the guest family.

Output:
Print integer R in single line.

Constraints:
1 <= N, S, Q, G, M <= 1000000

SAMPLE INPUT

```
5
5 8
3 6
10 7
4 6
2 5
2
5
8
```

SAMPLE OUTPUT

100

Explanation
We have 2 families, first family has 5 members. According to condition 1, Roy can serve two sweets with sweetness level 5 and 10. According to condition 2, Roy can serve only one of these two sweets. According to condition 3, Roy can serve only 1 sweet to each member.

So, after first family R = 1*100 i.e. R = 100

For second family, we have only one sweet with sweetness level greater than or equal to 8, which is sweetness level 10. Quantity of this sweet is 7, but Roy has to serve equal number of sweets to each of the 8 members.
Hence 0 sweets are served.

So after second family R = R + 0*100 (means he didn't get any rupees from second family)
Hence the output, 100.

The Code:

```
/* For any sweetness level between 1-1000000, store the max quantity you can
give >= to that
* sweetness level. For that:
* 1: Sort the array according to sweetness level.
* 2: Traverse it backwards and store maximum quantity encountered so far. ( I
stored in
* quantity array itself)
* 3: Fill the maxSweets array between 1-1000000. (Actually from 1 to max in
sweetness levels,
* as anything beyond the max is 0).
*/


#include <stdio.h>
#define MAX 1000000

int *s_copy, *q_copy;

void merge(int left, int right, int* sweets, int* quantity) {
    int l, r, mid, i, copy_count;
    l = left;
    mid = (left + right) / 2;
    r = mid+1;
    copy_count = 0;
    while(l <= mid && r <= right) {
        if(sweets[l] < sweets[r]) {
            s_copy[copy_count++] = sweets[l++];
            q_copy[copy_count-1] = quantity[l-1];
        } else {
            s_copy[copy_count++] = sweets[r++];
            q_copy[copy_count-1] = quantity[r-1];
        }
```

```c
    }
    while(l <= mid) {
        s_copy[copy_count++] = sweets[l++];
        q_copy[copy_count-1] = quantity[l-1];
    }
    while(r <= right) {
        s_copy[copy_count++] = sweets[r++];
        q_copy[copy_count-1] = quantity[r-1];
    }
    copy_count = 0;
    for(i=left; i<=right; i++) {
        sweets[i] = s_copy[copy_count++];
        quantity[i] = q_copy[copy_count-1];
    }
}
void mergeSort(int left, int right, int* sweets, int* quantity) {
    int mid = (left + right) / 2;
    if(left < right) {
        mergeSort(left, mid, sweets, quantity);
        mergeSort(mid+1, right, sweets, quantity);
        merge(left, right, sweets, quantity);
    }
}

int main(){
    int sweetCount, familyCount, familySize, max_sweet, i, j;
    long long int profit = 0;
    scanf("%d", &sweetCount);
    int* sweets = (int*)malloc(sweetCount * sizeof(int));
    int* quantity = (int*)malloc(sweetCount * sizeof(int));
    s_copy = (int*)malloc(sweetCount * sizeof(int));
    q_copy = (int*)malloc(sweetCount * sizeof(int));
    int* maxSweets = (int*)calloc(MAX, sizeof(int));
    for(i=0; i<sweetCount; i++) {
        scanf("%d %d", &sweets[i], &quantity[i]);
    }
    mergeSort(0, sweetCount-1, sweets, quantity);
    max_sweet = quantity[sweetCount-1];
    for(i=sweetCount-2; i>=0 ; i--) {
        if(quantity[i] > max_sweet) {
            max_sweet = quantity[i];
        } else {
            quantity[i] = max_sweet;
        }
    }
    j = 0;
    for(i=0; i<sweets[sweetCount-1]; i++) {
        if(((i+1) > sweets[j]) && ((j+1) < sweetCount) && ((i+1) <= sweets[j+1]))
            j++;
        if( j < sweetCount)
            maxSweets[i] = quantity[j];
    }
    scanf("%d", &familyCount);
    for(i=0; i<familyCount; i++) {
        scanf("%d", &familySize);
        profit += 100 * (maxSweets[familySize-1] / familySize);
    }
    printf("%lld", profit);
}
```

The Stats:

Score
30.0

Time (sec)
2.41711

Memory (KiB)
19856

Language
C