

# ABC 130 解説

satashun, DEGwer, yuma000, yosupo, gazelle

2019/06/15

*For International Readers: English editorial starts on page 8.*

## A: Rounding

問題文通りに、 $X < A$  なら 0,  $x \geq A$  なら 10 を出力すれば良いです。  
実装例をソースコード 1 に示します。

Listing 1 Rounding 実装例

---

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int X, A; cin >> X >> A;
6     puts(X < A ? "0" : "10");
7     return 0;
8 }
```

---

## B: Bounding

実際に  $D_1, D_2, \dots, D_{N+1}$  を漸化式に従って生成し、それぞれの要素が  $X$  以下かどうか判定すればよいです。時間計算量と空間計算量は  $\mathcal{O}(N)$  です。

実装例をソースコード 2 に示します。

Listing 2 Bounding 実装例

---

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int N, X; cin >> N >> X;
6     vector<int> D(N + 1);
7     D[0] = 0;
8     for (int i = 0; i < N; ++i) {
9         int x; cin >> x;
10        D[i + 1] = D[i] + x;
11    }
12    int ans = 0;
13    for (int i = 0; i <= N; ++i) {
14        if (D[i] <= X) {
15            ans++;
16        }
17    }
18    cout << ans << endl;
19    return 0;
20 }
```

---

## C: Rectangle Cutting

与えられた点  $(x, y)$  と長方形の中心を両方通るような直線で長方形を切って二つの部分に分けると、大きくない方の面積は全体の半分の面積になり、これが最大です。また、中心を通らないような直線で切った場合、中心を含む方の部分の面積の方が大きくなります。

以上の考察から、与えられた点が長方形の中心でない場合は問題文の条件を満たす切り方は一意に定まりません。そうでないなら、どの直線に沿って長方形を切っても、できる両方の部分の面積は等しくなるので、以下のようにして答えを求めることができます。

---

```
1 #include<stdio.h>
2 #include<vector>
3 #include<algorithm>
4 using namespace std;
5 typedef long long ll;
6 int main()
7 {
8     int a, b, x, y;
9     scanf("%d%d%d%d", &a, &b, &x, &y);
10    printf("%lf %d\n", double(a)*double(b) / 2, x + x == a&&y + y == b);
11 }
```

---

## D. Enough Arrays(writer : yuma000)

有り得る部分列の左端と右端を列挙して、それぞれの区間和が  $K$  以上かを確認する方法では、 $O(N^2)$  かってしまいます (累積和を用いることで、区間和は  $O(1)$  で求まります)。よって、もっと効率の良い解法を見つける必要があります。

$S(l, r) = \sum_r^l A_k$  としたとき、

- $S(a, b) < S(a, b + 1)$
- $S(a, b) > S(a + 1, b)$

が成り立ちます。

つまり、ここから

- ある  $l, r$  に対して、 $S(l, r) \geq K$  ならば、全ての  $x (x \geq r)$  に対して、 $S(l, x) \geq K$  である。

といえます。つまり、部分列の左端  $l$  を固定したとき、 $S(l, r) \geq K$  を満たすような最小の  $r$  を見つけることができれば、部分列の左端が  $l$  の時に、条件を満たすものの個数を求めることができます。(具体的には、 $N - r + 1$  です。)

具体的に  $r$  を求める方法としては

- 尺取り法 ( $O(N)$ )
- 二分探索 ( $O(N \log N)$ )

が挙げられるので、そのどちらかで実装すればよいです (個人的には尺取り法の方が計算量も少なく、実装も軽いと思うので、おすすめです)。

以下が、尺取り法のコードです。

---

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int N;long long int K;
6     cin>>N>>K;
7     vector<long long int>A(N);
8     for(int i=0;i<N;++i){
9         cin>>A[i];
10    }
11    long long int answer=0;
12    long long int sum=0;
13
14    int r=0;
15    for(int l=0;l<N;++l){
16
17        while(sum<K){
18            if(r==N)break;
```

```
19         else{
20             sum+=A[r];
21             r++;
22         }
23     }
24     if(sum<K)break;
25     answer+=N-r+1;
26     sum-=A[l];
27 }
28 cout<<answer<<endl;
29 return 0;
30 }
```

---

## E: Common Subsequence

ある整数  $k$  に対して、 $S_{i_1} = T_{j_1}, \dots, S_{i_k} = T_{j_k}$  となるように添字の集合  $1 \leq i_1 < i_2 < \dots < i_k \leq N$  と  $1 \leq j_1 < j_2 < \dots < j_k \leq M$  を選ぶ場合の数を求める問題です。以下ではこの集合を前から決めていくことを考えます。まず、素直な解法として次のような dp が考えられます。 $dp[i][j]$  :  $S$  の  $i$  文字目までと  $T$  の  $j$  文字目までを考慮し、この 2 文字をペアにするときの場合の数とすると、 $S_i = T_j$  であるとき  $dp[i][j] = (\sum_{k=1}^{i-1} \sum_{l=1}^{j-1} dp[k][l]) + 1$ 、そうでないとき 0 として計算できますが、この時間計算量は  $\mathcal{O}(N^2 * M^2)$  です。和を取る部分に着目すると、2 次元累積和の考え方を適応することができ、 $\mathcal{O}(NM)$  に改善できます。具体的には、 $sum[i][j] = \sum_{k=1}^i \sum_{l=1}^j dp[k][l]$  と置いて、 $sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + dp[i][j]$  として更新していくことができます。

## F: Minimum Bounding Box

以下  $(x_{max} - x_{min}) \times (y_{max} - y_{min})$  を bounding box と呼ぶことにします。

点の移動開始後、 $x_{max}$  の変化量は減少、不変、増加と推移していきます（各区間が 0 に潰れる場合もあります）。変化量が変わるタイミングは、単調性を利用した二分探索などで求めることができます。同様のことは  $x_{min}$  や  $y_{max}$  や  $y_{min}$  でも成り立ちます。

この性質を利用すると、点の移動開始後の時間軸を各最大最小値の変化量が不変ないくつかの区間に分割できます。そして実は bounding box が最小値をとるタイミングは、そのような区間の端点のいずれかです。

このことを示します。

まず  $dx = x_{max} - x_{min}$ ,  $dy = y_{max} - y_{min}$  のように定めます。 $dx$  および  $dy$  が広義単調増加するような区間では、bounding box は増加します。よってそのような区間では、区間の開始地点で bounding box は最小値をとります。同様に  $dx$  および  $dy$  が広義単調減少するような区間では、区間の終了地点で bounding box は最小値をとります。

次に  $dx$  が狭義増加  $dy$  が狭義減少である場合を考えます（逆の場合も同様です）。このときの bounding box の変化量は  $dx$  と  $dy$  の比率に左右されます。 $dx$  が  $dy$  と比べて一定以上小さい間は bounding box は増加します。 $dx$  が増加を続けると（あるいは最初から）減少の影響力のほうが強くなり、bounding box は減少します。結局このような区間では bounding box の値は上に凸になり、区間端点のいずれかで最小値をとります。

以上より、先述の事実が示せました。

# ABC 130

satashun, DEGwer, yuma000, yosupo, gazelle

06/15/2019

## A: Rounding

For this problem, implement the instructions in the problem statement. If  $X < A$ , print 0. Otherwise, print 10. An implementation is below.

Listing 1: Rounding

---

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int X, A; cin >> X >> A;
6     puts(X < A ? "0" : "10");
7     return 0;
8 }
```

---



## B: Bounding

For this problem, we care about the location of the ball at each bounce. The location of the ball at the first bounce is always 0. The location of the ball at for subsequent bounce  $i$  is the location of the ball at the previous bounce plus  $L[i]$ , the length traveled from the previous bounce. This is the same thing as taking prefix sums. Now that we know the location of all the bounces, we can check each location and count the number of locations that are  $\leq X$ . The runtime is  $O(N)$ .

Listing 2: Bounding

---

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int N, X; cin >> N >> X;
6     vector<int> D(N + 1);
7     D[0] = 0;
8     for (int i = 0; i < N; ++i) {
9         int x; cin >> x;
10        D[i + 1] = D[i] + x;
11    }
12    int ans = 0;
13    for (int i = 0; i <= N; ++i) {
14        if (D[i] <= X) {
15            ans++;
16        }
17    }
18    cout << ans << endl;
19    return 0;
20 }
```

---

## C: Rectangle Cutting

A rectangle can be cut into two equal parts by any line that passes through the center of the rectangle. The points  $(x, y)$  and the center can form a line that cuts through the rectangle, so it is always possible to cut the rectangle exactly in half.

A line that doesn't pass through the center of the rectangle cannot split the rectangle in half. If  $(x, y)$  is the center of the rectangle, there are multiple ways to cut the rectangle in half passing through  $(x, y)$  because any line passing through the center works. If  $(x, y)$  is not the center, there is exactly one way to cut the rectangle in half because there is exactly one line that passes through both  $(x, y)$  and the center.

Listing 3: Rectangle Cutting

---

```
1 #include<stdio.h>
2 #include<vector>
3 #include<algorithm>
4 using namespace std;
5 typedef long long ll;
6 int main()
7 {
8     int a, b, x, y;
9     scanf("%d%d%d%d", &a, &b, &x, &y);
10    printf("%lf %d\n", double(a)*double(b) / 2, x + x == a
        &&y + y == b);
11 }
```

---

## D. Enough Arrays (writer: yuma000)

If the sum of elements in the range from  $l$  to  $r$  is  $\geq K$ , then the sum of elements in the range from  $l$  to  $r + 1$  is also  $\geq K$  because all the elements are positive. This means that for each index  $l$ , if we can find the first index  $r$  such that  $sum(a[l..r]) \geq K$ , then we can add  $N - r + 1$  to the answer.  $N - r + 1$  is the number of ranges that start with  $l$ . The ranges can end at any location from  $r..N$ .

There are two ways to find the first value  $r$  such that the range  $sum(a[l..r]) \geq K$  for each  $l$ .

The first way is binary search. First, we take the prefix sums of the array  $A$ . With prefix sums, we can query the sum of a range in  $O(1)$ . For every starting index  $l$ , we can binary search to find  $r$ . The runtime for this method is  $O(N \log N)$ .

The second method is two pointers. This is the method shown in the code below. If we know that  $r$  is the first index such that  $sum(a[l..r]) \geq K$  (this means  $sum(a[l+1..r]) < K$ ), then the range from  $l+1$  to  $r-1$  will not work (this means  $sum(a[l+1..r-1]) < K$ ). The brute force solution is to try each possible  $r$  for every  $l$ . This runs in  $O(N^2)$ . We can speed this up by only trying a possible value of  $r$  if it is greater than or equal to the value of  $r$  for the previous  $l$ . This speeds the runtime up to  $O(N)$  because we only check each possible value of  $r$  once total while going through all possible values of  $l$ .

Listing 4: Enough Arrays

---

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int N;long long int K;
6     cin>>N>>K;
7     vector<long long int>A(N);
8     for(int i=0;i<N;++i){
9         cin>>A[i];
10    }
11    long long int answer=0;
12    long long int sum=0;
13
14    int r=0;
15    for(int l=0;l<N;++l){
16
```

```
17     while(sum<K){
18         if(r==N)break;
19         else{
20             sum+=A[r];
21             r++;
22         }
23     }
24     if(sum<K)break;
25     answer+=N-r+1;
26     sum-=A[1];
27 }
28 cout<<answer<<endl;
29 return 0;
30 }
```

---

## E: Common Subsequence

Let  $dp[i][j]$  be the number of ways to create subsequences only using the first  $i$  elements of  $S$  and the first  $j$  elements of  $T$  such that the subsequences are the same and the  $i$ th element of  $S$  and the  $j$ th element of  $T$  are part of the subsequences. Basically,  $dp[i][j]$  is the answer to the problem if we only consider the first  $i$  elements of  $S$  and the first  $j$  elements of  $T$  and we have to use the  $i$ th element of  $S$  and the  $j$ th element of  $T$ . If  $S_i \neq T_j$ , then  $dp[i][j] = 0$  because no subsequence will end by using the  $i$ th element of  $S$  and the  $j$ th element of  $T$ . If  $S_i = T_j$  then  $dp[i][j] = (\sum_{k=1}^{i-1} \sum_{l=1}^{j-1} dp[k][l]) + 1$  because the previous index of  $S$  can be any index  $\leq i$  and the previous index of  $T$  can be any index  $\leq j$ . As a base case, we can say  $dp[0][0] = 1$ . This represents the case where we don't take any elements. The runtime of this is  $\mathcal{O}(N^2 * M^2)$ , but we can speed this up by precomputing the sums. Let  $sum[i][j] = \sum_{k=1}^i \sum_{l=1}^j dp[k][l]$ .  $sum[i][j]$  is a 2D prefix sum of the  $dp$  array.  $sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + dp[i][j]$ . With  $sum[i][j]$ , we can now calculate each state,  $dp[i][j]$ , in  $\mathcal{O}(1)$ . Since there are  $NM$  states, the runtime is  $\mathcal{O}(NM)$ .

## F: Minimum Bounding Box

Let's call  $(x_{max} - x_{min}) \times (y_{max} - y_{min})$  the bounding box. We want to minimize the area of the bounding box.

Consider the value of  $x_{max}$  as the points start moving.  $x_{max}$  will decrease for some amount of time (which may be 0), then it will stay the same for some amount of time (which may be 0), and then it will increase for some amount of time (which may be 0). We can binary search to determine when  $x_{max}$  stops decreasing and starting staying the same and when  $x_{max}$  stops staying the same and start increasing. We can repeat this process for  $x_{min}$ ,  $y_{max}$ , and  $y_{min}$ .

We care about the interval of time when the values (for example  $x_{max}$ ) stay the same because those will lead to the smallest bounding box. Consider time as a number line and place the intervals when the values are the same on the numberline. We have now divided the numberline into several sections. For each endpoint of a section, let  $dx = x_{max} - x_{min}$ ,  $dy = y_{max} - y_{min}$ . Note that all 4 values may not be constant in every section.

In the sections where  $dx$  and  $dy$  are increasing, the bounding box will also increasing. In the sections where  $dx$  and  $dy$  are decreasing, the bounding box will also decrease. In both these cases, the area of the bounding box is minimized at an endpoint of the section.

We now have to consider the sections where  $dx$  is decreasing and  $dy$  is increasing. Consider the ratio that  $dx$  changes by and the ratio that  $dy$  changes by. If the ratio that  $dx$  decreases by is more than the ratio  $dy$  increases by, then the area of the bounding box decreases. The area of the bounding box is convex in these sections, so the minimum area will be at the endpoint of one of these sections. We can symmetrically consider the sections where  $dx$  is increasing and  $dy$  is decreasing.