ABC 130 解説

satashun, DEGwer, yuma000, yosupo, gazelle

2019/06/15

A: Rounding

問題文通りに、$X < A$ なら 0, $x >= A$ 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

```
#include <bits/stdc++.h>
using namespace std;

int main() {
  int X, A; cin >> X >> A;
  puts(X < A ? "0" : "10");
  return 0;
}
```

B: Bounding

実際に $D_1, D_2, ..., D_{N+1}$ 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

```
#include <bits/stdc++.h>
using namespace std;

int main() {
  int N, X; cin >> N >> X;
  vector<int> D(N + 1);
  D[0] = 0;
  for (int i = 0; i < N; ++i) {
        int x; cin >> x;
    D[i + 1] = D[i] + x;
  }
  int ans = 0;
  for (int i = 0; i <= N; ++i) {
        if (D[i] <= X) {
      ans++;
    }
  }
  cout << ans << endl;
  return 0;
}
```

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き、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す。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。

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

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 以上かを確かめる方法では、$O(N^2)$ かかってしまいます (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、$S(l,r) >= K$ ならば、全ての $x(x >= r)$ に対して、$S(l,x) >= K$ 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、$S(l,r) >= K$ 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(NlogN))

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。

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    int N;long long int K;
    cin>>N>>K;
    vector<long long int>A(N);
    for(int i=0;i<N;++i){
        cin>>A[i];
    }
    long long int answer=0;
    long long int sum=0;

    int r=0;
    for(int l=0;l<N;++l){

        while(sum<K){
            if(r==N)break;
```

```
            else{
                sum+=A[r];
                r++;
            }
        }
        if(sum<K)break;
        answer+=N-r+1;
        sum-=A[l];
    }
    cout<<answer<<endl;
    return 0;
}
```

E: Common Subsequence

ある整数 k に対して、$S_{i_1} = T_{j_1}, ..., S_{i_k} = T_{j_k}$ となるように添字の集合 $1 \leq i_1 < i_2 < ... < i_k \leq N$ と $1 \leq j_1 < j_2 < ... < j_k \leq M$ 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。