ABC 127 解説

writer: tozangezan, drafear, satashun, potetisensei

2019 年 5 月 25 日

A: Ferris Wheel

if 文を用いて、高橋君が 13 歳以上なのか、6 歳以上 12 歳以下なのか、5 歳以下なのかに応じて、それぞれ答え (B 円、B/2 円、0 円) を出力しましょう。

C++ での実装例は以下の通りです (include 等は省いています)。

```
int main(){
  int a,b;scanf("%d%d",&a,&b);
  if(a<6)printf("0\n");
  else if(a<13)printf("%d\n",b/2);
  else printf("%d\n",b);
}
```

B: Algae

for ループを回すことによって漸化式の通りにシミュレーションしましょう。配列 `x[i]` を用意して `x[2001]`, ..., `x[2010]` を順に求め、出力してもよいですし、変数 x を随時更新し、更新するごとに出力してもよいです。

C++ での実装例は以下の通りです。

```
int main(){
  int r,K;
  int x;
  scanf("%d%d%d",&r,&K,&x);
  for(int i=0;i<10;i++){
    x=x*r-K;
    printf("%d\n",x);
  }
}
```

C: Prison

考え方 1

各 $i = 1, 2, ..., N$ について、i 番目の ID カードで全てのゲートを通過できるかを考えます。すると、全ての $j = 1, 2, ..., M$ について、 $L_j \leq i \leq R_j$ であるときに限り i 番目の ID カードで全てのゲートを通過できます。このような i の個数を数え上げることで答えを求めることができますが、このままでは時間計算量が $O(NM)$ となり間に合いません。

そこで、各 i について、全ての j について $L_j \leq i \leq R_j$ が成り立つかを高速に判定したいです。この条件は、

$$i \geq L_1, L_2, ..., L_M$$

かつ

$$i \leq R_1, R_2, ..., R_M$$

と表すことができます。すなわち、

$$L' = \max\{L_1, L_2, ..., L_M\}$$
$$R' = \min\{R_1, R_2, ..., R_M\}$$

とすると、各条件は $L' \leq i \leq R'$ と表すことができます。このような i の個数が答えになります。これは $L' \leq R'$ のとき $R' - L' + 1$、そうでないとき 0 です。時間計算量は $O(M)$ です。

考え方 2

全ゲートを通過できる ID カードの番号は区間 (連番) になります。また、全ゲートを通過できる ID カードは、$M - 1$ 番目までの全ゲートに通過できる ID カードのうち M 番目のゲートにも通過できるものです。そこで、i 番目までの全ゲートに通過できる ID カードの番号を $l_i, l_i + 1, ..., r_i$ とします。すると、l_{i+1}, r_{i+1} は l_i, r_i から計算できます。初期値も適当に、$l_0 = 1, r_0 = N$ や $l_1 = L_1, r_1 = R_1$ とすれば l_M, r_M を時間計算量 $O(M)$ で計算することができます。考え方 1 と同様に、$l_M \leq i \leq r_M$ であるような i の個数が答えになります。

D: Integer Cards

書き換える操作は好きな順番で行っても構いません。なぜなら、同じカードに対して 2 回以上書き換えるのは無駄だからです。また、C_i に B_i 枚まで書き換える操作は、C_i に 1 個まで書き換える操作を B_i 回行えると考えます。各 C_i を B_i 個ずつ並べた列を $D_1, D_2, ..., D_{\sum B_i}$ とします。すると、操作は「選んだことのない i を選んで 1 枚まで D_i に書き換える」と考えることができます。同じカードに対して 2 回以上書き換えるのは無駄なので、この操作は合計で N 回しか行いません。したがって、D_i の大きい方から N 個まで取り出した列を新しい D として考えても構いません。

X 枚書き換えるとします。すると、書き換え元は小さい方から X 枚書き換えるのが最適です。書き換え先は D_i の大きいものから順に X 個選ぶのが最適です。X を 1 ずらすことによって書き換え元と書き換え先はそれぞれ 1 つずつしか変化しないため、X を順に捜査することで $O(MlogM + NlogN)$ の計算時間で解くことができます。

実装では、D の大きい方から N 個まで取り出す部分が難しいかもしれません。これは、$(B_1, C_1), ..., (B_M, C_M)$ を C_i の大きい順にソートして[*1]配列 D が N 未満である間 C_i を B_i 個まで 1 個ずつ D に追加していく実装が楽でしょう。

最適な X を求めることもできます。A, D をソートして $A_1 \leq A_2 \leq ... \leq A_N$ かつ $D_1 \geq D_2 \geq ... \geq D_|D|$ とします。すると、前述のアルゴリズムでは X を $i-1$ から i にずらすことによって合計値は $D_i - A_i$ 変化しますが、$D_i - A_i$ は単調非増加なので $D_i - A_i \geq 0$ すなわち $D_i \geq A_i$ である間だけ操作を続けるのが最適です。すなわち、そのような i の最大値 (なければ 0) を X とするのが最適です。

別解

残す整数と書き換え先とする整数を合計で N 個選んで N 個の整数の合計を最大化することを考えます。$(X_1, Y_1), ..., (X_{N+M}, Y_{N+M})$ を $(1, A_1), ..., (1, A_N), (B_1, C_1), ..., (B_M, C_M)$ とすると、問題は「整数 Y_i を X_i 個まで選ぶことができ、合計 N 個の整数を選ぶときその和の最大値を求めてください」となります。これは、Y_i の大きい順に合計 N 個となるように選ぶのが最適です。この方法では $O((N+M)log(N+M))$ の時間計算量で求めることができます。

[*1] C++ 言語では (C_i, B_i) の pair を降順にソートする実装が簡単です。より読みやすいコードを書きたいなら構造体を用いると良いでしょう。

E: Cell Distance

$N \times M$ のマス目のうち K マス選ぶ時のマンハッタン距離の和を求めよという問題です。式は明らかに X と Y について独立なので、X の差の絶対値の和と Y の差の絶対値の和をそれぞれ求めることにします。

以下では X の差について考えます。ある 2 マスの組み合わせを固定したとき、これら以外から $K-2$ マス選ぶ場合全てに 1 度ずつこれらの差が寄与するので、この組を固定して考えると $\binom{N\times M-2}{K-2}$ 通りあります。さらに、X が同じ場合は差が 0 なので、X が異なると仮定すると、X の差の絶対値が d となるように 2 マス選ぶ方法は $(N-d)\times M^2$ 通りあります。これを全ての d に対して足し合わせると X についての答えが求まります。Y についても N と M を入れ替えると同様に解くことができ、時間計算量としては二項係数を求めるパートがボトルネックとなり $\mathcal{O}(NM)$ でこの問題が解けました。

F: Absolute Minima

N 個の関数が与えられており、$f(x) = \sum_{i=1}^{N}(|x - a_i| + b_i)$ となっている時を考えます。まず、$f(x) = \sum_{i=1}^{N}|x - a_i| + \sum_{i=1}^{N} b_i$ と分解でき、項 $\sum_{i=1}^{N} b_i$ は定数であるため、最小値を考える際には影響を与えない事が分かります。よって、$\sum_{i=1}^{N}|x - a_i|$ の最小値を与える x を求め、その最小値に $\sum_{i=1}^{N} b_i$ を足したものを出力すれば良いです。

以下、簡単のため、$\{a_i\}$ が昇順に整列されていると仮定し、$a_0 = -\infty, a_{N+1} = \infty$ とおくことにします。

$\sum_{i=1}^{N}|x - a_i|$ の最小値はどのようにして求めれば良いでしょうか？今、i 番目の絶対値関数 $|x - a_i|$ は $x < a_i$ において傾き -1 の 1 次関数、$x > a_i$ において傾き 1 の 1 次関数になっています。1 次関数同士を足し合わせても 1 次関数になる事を考えると、$i = 0, \cdots, N$ それぞれにおいて、区間 $[a_i, a_{i+1}]$ 内で $f(x)$ は 1 次関数と見做すことができます。

更に、x が小さいほど、N 個の関数の内、傾きが -1 であるような絶対値関数が増え、x が大きいほど傾きが 1 であるような絶対値関数が増えることを考えると、$f(x)$ の各区間 $[a_i, a_{i+1}]$ 内における 1 次関数の傾きは単調に増加します（これは凸関数の和は凸関数ということから考えても良いです）。よって、これは凸関数であり、傾きが 0 であるような区間（実際には 1 点のみからなる区間になっている可能性もあります）を求め、その区間内の点を 1 つ出力すれば良い、ということが分かります。

傾きが 0 であるような区間をどのようにして求めれば良いでしょうか？ここで、絶対値関数 $|x - a_i|$ において、傾きの変化点は $x = a_i$ であり、$x = a_i$ を “またいだ” 時に傾きが 2 だけ増えることに注目します。このことは、複数の絶対値関数を足し合わせても変わることはありません。すなわち、関数 $f(x)$ においても、$x = a_i$ を “またいだ” 際、傾きは 2 だけ増加します（ただし、同じ絶対値関数が複数含まれる時、すなわち $a_i = a_j$ となる i, j が存在するような場合には、含まれる個数分だけ増加することに注意してください）。

そして、区間 $[a_0 = -\infty, a_1]$ において、$f(x)$ の傾きは $-N$ であるため、大雑把に言えば、$\frac{N}{2}$ 個の変化点を “またぐ” と傾きが 0 となることが分かります。すなわち、昇順に並べた全ての変化点の内、ちょうど真ん中にある変化点（N が偶数の場合には真ん中にある 2 つの変化点の間）において $f(x)$ は傾きが 0 となり、最小値を取ります。

この計算方法を実装する際には、変化点をソートした状態で格納してくれるデータ構造（以下セットと呼ぶ）を 2 個用います（例えば C++ では multiset です）。片方のセットには昇順に並べた変化点の内、左半分を、もう片方のセットには右半分を全て管理させます。この時、1 つ目のセットの最右点と 2 つ目のセットの最左点の間の区間において $f(x)$ は傾き 0 となります。

更新クエリ (a_i, b_i) が来た際には、値 a_i を **2 個**、元と同じ条件（左半分を片方のセット、右半分をもう片方のセット）を満たすよう適切にセットに挿入すれば良いです（変化点をあえて 2 個挿入することによって、「1 つの変化点につき傾きが 1 変わる」と思うことが出来、シンプルになります）。

さて、最小値を与える点は分かったので、具体的な最小値を求めます。これは、

- 新しく挿入された点が、元々最小値を与えた区間に含まれるならば 0
- 区間から外れているならば、区間の端点との距離（どちらか短い方）

を元の最小値に足したものになります。

以上から、更新・求値両方を高々定数回のセットの操作のみによって実現する事ができ、$O(Q \log Q)$ で問題を解くことができます。また、凸性に基づいた値自体に対する三分探索や、Binary Indexed Tree を用いた

変化点の個数に関する二分探索を行う解法も存在します。