東京海上日動　プログラミングコンテスト 2020 解説

yutaka1999

2020 年 6 月 13 日

For International Readers: English editorial starts on page 7.

A: Nickname

適当な連続する 3 文字を出力すればよいです. 例えば先頭 3 文字を出力するのが簡単でしょう. C++ での実装例を以下に示します.

```
#include<iostream>
using namespace std;
int main(){
        string S;
        cin >> S;
        cout << S.substr(0,3) << endl;
        return 0;
}
```

B: Tag

$V \leqq W$ の場合は, 2 人の子供の距離は広がる一方ですので, 答えは NO となります. 一方で $V > W$ の場合は, 2 人の子供の距離は 1 秒あたり $V - W$ だけ縮まります. よって答えが YES となる条件は $\frac{|A-B|}{V-W} \leqq T$ です。

C++ での実装例を以下に示します.

```
#include<iostream>
using namespace std;
typedef long long int ll;
int main(){
        ll A, V;
        cin >> A >> V;
        ll B,W;
        cin >> B >> W;
        ll T;
        cin >> T;
        ll D=abs(A-B);
        ll D2=(V-W)*T;
        puts(D<=D2?"YES":"NO");
}
```

このように if 分岐を使わない実装をすることもできます.

C: Lamps

まず 1 回の操作をシミュレーションするのにかかる計算量を見積もります. シミュレーションの内容を整理すると, 各電球 i について $i - A_i$ から $i + A_i$ までの区間に 1 を足し, 最終的に各座標に書かれている値を計算する, というものです. これは累積和というテクニックにより $O(N)$ の計算量で行うことができます.

例えば C++ での実装は次のようになります. ここで数列 A, B の役割は問題文の通りです.

```
vector <int> B(N,0);
for(int i=0;i<N;i++)
{
        int l=max(0,i-A[i]);
        int r=min(N-1,i+A[i]);
        B[l]++;
        if(r+1<N) B[r+1]--;
}
for(int i=1;i<N;i++) B[i]+=B[i-1];
```

よって, 問題文の通りにシミュレーションをすると計算量は $O(NK)$ となります. しかし最悪ケースでは $NK \approx 4 \times 10^{10}$ なのでこれでは間に合いません. そこで次のような事実に気づく必要があります.

- $O(\log N)$ 回操作を繰り返すと数列は収束しすべて N になる.

これは大雑把には次のような理由で成立します. まず操作を 1 回行うと, どの項も 1 以上になります. さらに操作を 1 回行うと, 端は 2 以上, その他の項は 3 以上になります. さらにもう 1 回行うと ... というのを繰り返すと, $\log N$ 回ぐらいまでは操作するごとに各項の"下限"が約 2 倍になることが分かり, 最終的に電球の強さが N になって収束する項が現れます. その後は強さが N になって収束する項が増えていき, さらに $\log N$ 回ぐらいの操作を行うと全体が N になります. ここで使った性質は, $A'_i \leqq A_i$ が成り立つならば $B'_i \leqq B_i$ であるという性質であり, 各電球の強さが 0 の状態から $O(\log N)$ 回の操作を繰り返すとすべて N になる, という観察により, 常に $O(\log N)$ 回の操作ですべて N になるということが分かります. 実際には $N \leqq 2 \times 10^5$ の制約の下では 41 回操作を行えば十分です.

よって, シミュレーションを行う回数は $O(\log N)$ 回で十分なので $O(N \log N)$ の計算量でこの問題を解くことができます.

D: Knapsack Queries on a tree

簡単のため $N = 2^{2K} - 1$ とします. また $M := \max L_i$ とします. このとき $O(N + 2^K(M+Q))$ の時間計算量で動作するアルゴリズムを紹介します. この問題では $K = 9$, $M = 10^5$ であるので $2^K(M+Q) \approx 10^8$ となり, 確かに間に合います.

まず深さが $K-1$ 以下の頂点 v 及び M 以下の非負整数 k に対して, v 及び v の先祖にあるアイテムから, 重さの合計が高々 k になるようにいくつか選んだときの, 選んだアイテムの価値の合計の最大値 $D_{v,k}$ を計算します. この値は根に近い頂点から順に計算していくことができ, $O(2^K M)$ の時間計算量ですべての $D_{v,k}$ を求めることができます.

次にクエリに答えることを考えます. 指定された頂点を v とします. ここで v 及び v の先祖のうち, 深さが K 以上のものを考えると, これは高々 K 個しかありません. よって $O(2^K)$ の時間計算量をかけることで, 深さが K 以上の頂点にあるアイテムを選ぶかどうか全探索することができます. 深さが K 以上の頂点から選ぶアイテムを決めると, 深さ $K-1$ 以下の頂点にあるアイテムから最適に選ぶ方法が先ほど構成した $D_{*,*}$ から分かるので, 各場合について $O(1)$ の時間計算量で最適解が分かります. よって $O(2^K)$ の時間計算量で 1 つのクエリに答えることができます.

以上により, 全体で $O(N + 2^K(M+Q))$ の時間計算量でこの問題を解くことができます.

E: O(rand)

簡単のため $S = 0$, $T = 2^K - 1$, $0 \leqq A_i < 2^K$ としてよいです. さまざまな解法が考えられますが, ここでは $O(2^K N)$ の時間計算量の解法を紹介します.

まず $0 \leqq U < 2^K$ なる各 U に対して $f(U)$ を次のように定めます.

- 与えられた数から 1 個以上 K 個以下の数を選ぶ方法であって, U とのビットごとの論理積がすべて等しくなるような方法の数を $f(U)$ とする.

このとき $\sum_U (-1)^{popcount(U)} f(U)$ が求める答えとなっています. ここで $popcount(U)$ は U に属するビットの個数です. まずこれを証明します.

与えられた数から 1 個以上 K 個以下の数を選ぶ方法を 1 つ固定します. この選び方が $\sum_U (-1)^{popcount(U)} f(U)$ において"何回"数えられているかを考えます. まず選ばれた数の U とのビットごとの論理積がすべて等しくなる, という U に関する条件を整理します. これは U に含まれるビットの状態が, どの 2 つの選ばれた数についても一致するということであるので, 選ばれた数のビットごとの論理積を S', ビットごとの論理和を T' とすると $U \subset S' \cup (\{0, 1, \ldots, K-1\} \backslash T')$ と同値です.

これより $S' \cup (\{0, 1, \ldots, K-1\} \backslash T') = \phi$ であれば, この選び方は $\sum_U (-1)^{popcount(U)} f(U)$ において 1 回数えられており, $S' \cup (\{0, 1, \ldots, K-1\} \backslash T') \neq \phi$ であれば 0 回であることが分かります. (一般に $\sum_{U \subset A} (-1)^{popcount(U)}$ は $A = \phi$ のとき 1, そうでないならば 0 です.) $S' \cup (\{0, 1, \ldots, K-1\} \backslash T') = \phi \Leftrightarrow S' = 0, T' = 2^K - 1$ であるので, $\sum_U (-1)^{popcount(U)} f(U)$ が求める答えであることが分かりました.

後は $f(U)$ を求めることができればよいです. $O(2^K)$ の空間計算量を使いまわせば, 各 U について $O(N)$ の時間計算量で $f(U)$ を求めることができます. よって, 全体として $O(2^K N)$ の時間計算量でこの問題を解くことができます.

別解

$O(N^2 + 2^K NK)$ の時間計算量の別解も軽く紹介します. まず $O(N)$ の時間計算量をかけて選ぶ数を 1 つ固定します. すると, 各ビットについて, 論理和と論理積の条件のいずれか一方は必ず満たされます. よって満たすべき条件は, 各ビットについて, そのビットを含まない, またはそのビットを含む数を少なくとも 1 つ選ぶ, というように言い換えることができ, これは包除原理により $O(N + 2^K K)$ の時間計算量で解くことができます. よって全体として $O(N^2 + 2^K NK)$ の時間計算量で解くことができます.

F: Triangles

簡単のため, 3 頂点が $(x,0)$, $(0,y_1)$, $(W,y_1+s)(0<x<W,\ 0\leqq s<H-1,\ 0<y_1<H-s)$ と書けるような Δ を数えます. 実際は $s=0$ と $s>0$ の場合で係数が異なりますが, 以下では s を固定して条件を満たす Δ を数えるので, 係数の違いはここでは無視します.

まず Δ の面積を S, Δ の内部に含まれる格子点の個数を b, 辺上（頂点を除く）にある格子点の個数を i とおくと, ピックの定理より $S=b+\frac{i}{2}+\frac{1}{2}$ です. よって $b\leqq K$ という条件は $2S-i-1\leqq 2K$ と書き直すことができます. 簡単な計算により $2S=Wy_1+sx$, $i=\gcd(s,W)+\gcd(x,y_1)+\gcd(W-x,y_1+s)-3$ とわかるので, 条件を整理すると

$$Wy_1-\gcd(x,y_1)-\gcd(W-x,y_1+s)\leqq 2K+\gcd(s,W)-sx-2$$

となります.

ここで s と x の値を固定すると, 条件の右辺の値は一定になります. この値を R とします. また, 左辺の値は $W(y_1-1)$ 以上 Wy_1 以下であるので, $y_1\leqq\frac{R}{W}$ の場合は常に条件を満たし $y>\frac{R}{W}+1$ の場合は常に条件を満たしません. よって $\frac{R}{W}<y_1\leqq\frac{R}{W}+1$ なる y_1 についてのみ実際に条件を確かめればよく, 結局 s と x の値を固定すると gcd の計算のみに求める個数が分かります. つまり $O(\log(H+W))$ で対応する Δ の個数が求まります.

しかし $0<x<W$, $0\leqq s<H-1$ なる (s,x) をすべて調べるのは $O(HW)$ となり間に合いません. そこで (s,x) についての条件を考えると, 対応する R の値が正であることが必要なので $2K+\gcd(s,W)-sx-2\geqq 0$ という条件が得られます. 特に $\gcd(s,W)\leqq W$ より $sx\leqq 2K+W-2$ が必要です. このような条件を満たす (s,x) のペアは $O((2K+W)\log(2K+W))$ 個しかないので, それらを全探索し, 各々について対応する Δ の個数を求め足し合わせることで, この問題を解くことができます.

全体の時間計算量は $O((K+W+H)\log(K+W+H)\log(H+W))$ となります.

A: Nickname

We can print any three consecutive characters. One of the simplest choices would be printing the first three characters.

Sample C++ implementation follows:

```
#include<iostream>
using namespace std;
int main(){
    string S;
    cin >> S;
    cout << S.substr(0,3) << endl;
    return 0;
}
```

B: Tag

If $V \leq W$, the distance between the two children keeps increasing, so the answer is NO. On the other hand, if If $V > W$, the distance decreases by $V - W$ per second. Thus, the answer is YES if and only if $\frac{|A-B|}{V-W} \leq T$.

Sample C++ implementation follows:

```
#include<iostream>
using namespace std;
typedef long long int ll;
int main(){
        ll A, V;
        cin >> A >> V;
        ll B,W;
        cin >> B >> W;
        ll T;
        cin >> T;
        ll D=abs(A-B);
        ll D2=(V-W)*T;
        puts(D<=D2?"YES":"NO");
}
```

As seen here, we can do without if statements.

C: Lamps

First, let us evaluate the time it takes to simulate one operation. The operation can be rephrased as follows: for each i, we add 1 to the segment from $i-A_i$ to $i+A_i$, then we compute the value written at each coordinate. This can be done in $O(N)$ time with a technique called accumulated sums.

Sample C++ implementation of the simulation is shown below. Here, the sequences A and B play the same role as in the problem statement.

```
vector <int> B(N,0);
for(int i=0;i<N;i++)
{
    int l=max(0,i-A[i]);
    int r=min(N-1,i+A[i]);
    B[l]++;
    if(r+1<N) B[r+1]--;
}
for(int i=1;i<N;i++) B[i]+=B[i-1];
```

If we naively repeat this process K times, it takes $O(NK)$ time. However, we have $NK \approx 4 \times 10^{10}$ in the worst case, so we will run out of time with this. To get it done in time, we need to notice the following fact:

- After $O(\log N)$ operations, the sequence converges to $N, \ldots, N$.

We will roughly explain why. After the first operation, every term becomes at least 1. After one more operation, the terms at both ends become at least 2, and the other terms become at least 3. If we continue this way, we can see that the "lower limit" of each term becomes approximately twice until approximately the $(\log N)$-th operation, then there will be a term that becomes N and converges there. After that, more terms become N and converge there, and all terms become N after approximately $(\log N)$ more operations. This behavior is based on the following property: if $A'_i \leqq A_i$, then $B'_i \leqq B_i$. We can observe from this property, that, if the intensity of every bulb is 0, it becomes N after $O(\log N)$ operations. Thus, we can see that the intensity of every bulb always becomes N after $O(\log N)$ operations. Actually, under the constraint $N \leqq 2 \times 10^5$, we need at most 41 operations before the convergence.

Therefore, we need to simulate at most $O(\log N)$ operations, so we can solve the problem in $O(N \log N)$ time.

D: Knapsack Queries on a tree

For simplicity, let $N = 2^{2K} - 1$. Also, let $M := \max L_i$. We will introduce an algorithm that works in $O(N + 2^K(M+Q))$ time. In this problem, $K = 9$ and $M = 10^5$, so $2^K(M+Q) \approx 10^8$, and thus it will run in time.

First, for each vertex v at depth $K-1$ or shallower and each non-negative integer k at most M, we compute $D_{v,k}$: the maximum possible total value of items when we choose items from v and its ancestors so that the total weight is at most k. By computing them in the order from distance from the root, we can find all $D_{v,k}$ in $O(2^K M)$ time.

Then, we will process the query. Let v be the specified vertex. Among the vertices that are v or its ancestors, at most K have depths of K or deeper. Thus, we can try all subsets of the items in the vertices at depth K or deeper in $O(2^K)$ time. After deciding the subset to choose from the vertices at depth K or deeper, we can use $D_{*,*}$ to know the optimal subset to choose from the vertices at depth $K-1$ or shallower in $O(1)$ time. Thus, we can process each query in $O(2^K)$ time.

Therefore, we can solve the problem in a total of $O(N + 2^K(M+Q))$ time.

E: O(rand)

We can assume $S = 0$, $T = 2^K - 1$, $0 \leqq A_i < 2^K$ for simplicity. There are various solutions, and here we will introduce one with the time complexity of $O(2^K N)$.

First, for each U such that $0 \leqq U < 2^K$, we define $f(U)$ as follows:

- $f(U)$ is the number of ways to choose a set of between 1 and K numbers from the given ones so that $U\ AND\ x$ is the same for all chosen numbers x.

Then, the answer is $\sum_U (-1)^{popcount(U)} f(U)$ where $popcount(U)$ is the number of standing bits in U. Let us prove it first.

Let us fix one way to choose a subset of between 1 and K numbers from the given ones, and count "the number of times" this subset is counted in $\sum_U (-1)^{popcount(U)} f(U)$. We will rephrase this condition on U: "$U\ AND\ x$ is the same for all chosen numbers x." This means that, for a bit position where the bit is standing in U, the chosen numbers all have the same bit. Thus, the condition is equivalent to $U \subset S' \cup (\{0, 1, \ldots, K-1\} \backslash T')$ where S' and T' are the bitwise AND and OR of the chosen numbers.

Thus, if $S' \cup (\{0, 1, \ldots, K-1\} \backslash T') = \phi$, the subset is counted once in $\sum_U (-1)^{popcount(U)} f(U)$; if $S' \cup (\{0, 1, \ldots, K-1\} \backslash T') \neq \phi$, the subset is counted zero tiems. (In general, $\sum_{U \subset A} (-1)^{popcount(U)}$ is 1 if $A = \phi$ and 0 otherwise.) Since $S' \cup (\{0, 1, \ldots, K-1\} \backslash T') = \phi \Leftrightarrow S' = 0, T' = 2^K - 1$, we see that the answer is $\sum_U (-1)^{popcount(U)} f(U)$.

Now, we just have to find $f(U)$. By using the memory space of size $O(2^K)$ multiple times, we can find $f(U)$ in $O(N)$ time for each U. Thus, we can solve the problem in a total time of $O(2^K N)$.

Another solution

We will briefly introduce another solution with the time complexity of $O(N^2 + 2^K NK)$. We first fix one number to choose, multiplying the time complexity by $O(N)$. Then, for each bit, at least one of the conditions on the bitwise OR and AND is satisfied. We can represent the condition that we need to satisfy as follows: for each bit, we have to choose at least one number with 0 in that bit, or at least one number with 1 in that bit. The number of such ways to choose numbers can be computed in $O(N + 2^K K)$ time with the inclusion-exclusion principle, and thus we can solve the problem in a total of $O(N^2 + 2^K NK)$ time.

F: Triangles

For simplicity, let us count Δ such that its three vertices can be written as $(x, 0)$, $(0, y_1)$, $(W, y_1 + s)(0 < x < W,\ 0 \leqq s < H - 1,\ 0 < y_1 < H - s)$. We need to multiply the result by different coefficients in the cases $s = 0$ and $s > 0$, but we ignore the difference since we will count Δ after fixing s.

First, let S be the area of Δ, b be the number of grid points within Δ, and i be the number of grid points on the sides of Δ (excluding the vertices). From the Pick's theorem, $S = b + \frac{i}{2} + \frac{1}{2}$, so we can rephrase the condition $b \leqq K$ as $2S - i - 1 \leqq 2K$. After simple calculations, we can see $2S = Wy_1 + sx$ and $i = \gcd(s, W) + \gcd(x, y_1) + \gcd(W - x, y_1 + s) - 3$, so the condition can be further rephrased to the following:

$$Wy_1 - \gcd(x, y_1) - \gcd(W - x, y_1 + s) \leqq 2K + \gcd(s, W) - sx - 2$$

If we fix the values of s and x here, the right side has the constant value, which we call R. Also, the left side has a value between $W(y_1 - 1)$ and Wy_1, so the condition is always satisfied if $y_1 \leq \frac{R}{W}$ and never satisfied if $y > \frac{R}{W} + 1$. Thus, we only need to verify the condition for y_1 such that $\frac{R}{W} < y_1 \leq \frac{R}{W} + 1$. After fixing s and x, we just have to compute gcd to count y_1 satisfying the condition, that is, we can count Δ in $O(\log(H + W))$ time.

However, there are $O(HW)$ candidates (s, x) such that $0 < x < W$ and $0 \leqq s < H - 1$, and checking all of them runs out of time. Let us consider what condition (s, x) must satisfy. Since the corresponding value of R needs to be positive, we have $2K + \gcd(s, W) - sx - 2 \geq 0$. Particularly, from $\gcd(s, W) \leqq W$, it is neccesary that $sx \leqq 2K + W - 2$. Since only $O((2K + W)\log(2K + W))$ pairs (s, x) satisfy this condition, so we can solve the problem by enumerating all such pairs and summing up the numbers of Δ for those pairs.

The total time complexity of this solution is $O((K + W + H)\log(K + W + H)\log(H + W))$.