ABC086 / ARC089 解説

sigma425

For International Readers: English editorial starts on page 5.

A: Product

整数 a, b を読み込み、$a \times b$ を計算し、2 で割ったあまりを計算すればいいです。

```
int main(){
        int a,b;
        cin>>a>>b;
        int c = a * b;
        if(c%2 == 0) puts("Even");
        else puts("Odd");
        return 0;
}
```

B: 1 21

整数として読み込むよりも、文字列として読み込んで連結したものを整数に直すのが簡単かもしれません。C++ には標準で文字列を整数に直す stoi 関数があります。この整数は高々 100100 なので、平方数かどうかは、1×1, 2×2, .. , 1000×1000 くらいまでのどれかと一致するかループを回して確かめれば良いです。

```
int main(){
        string s,t;
        cin>>s>>t;
        int x = stoi(s+t);
        for(int i=1;i<=1000;i++){
                if(i*i == x){
                        puts("Yes");
                        return 0;
                }
        }
        puts("No");
}
```

C: Traveling

$t_0 = 0, (x_0, y_0) = (0, 0)$ と置きます。各 0 以上 $N-1$ 以下の i について、時刻 t_i に場所 (x_i, y_i) にいた後、時刻 t_{i+1} に場所 (x_{i+1}, y_{i+1}) にいることができるか、を判定して、どれか一つでも不可能なら No を、全て可能なら Yes を出力すれば良いです。

この判定は、$t := t_{i+1} - t_i$, $d := abs(x_i - x_{i+1}) + abs(y_i - y_{i+1})$ とおくと、$d \leq t \wedge t\%2 = d\%2$ で判定できます。($d \leq t$ でないと不可能なのは明らか、毎秒 $x + y$ の偶奇は変わるので 2 で割ったあまりは一致する必要がある。 逆に d, t がこれを満たすなら実際に条件のように動くことが可能)

D: Checker

マス (x, y) が白 と マス $(x, y + K)$ が黒 は同値なので、$(x, y,' W')$ という入力を $(x, y + K,' B')$ に置き換えても条件は同じです。これで入力の c_i を全て'B' に変換することが出来ます。

マス (x, y) の色とマス $(x, y + 2K), (x + 2K, y)$ の色は一致します。なので、入力の x, y を $x\%2K, y\%2K$ で置き換えても答えは変わりません。すると $0 \leq x, y < 2K$ と変換することが出来ます。

市松模様の黒い部分の左下のマスのうちひとつを定めるとマスの塗られ方全体が定まります。ここで、定めるマスは $[0, 2K) \times [0, 2K)$ の中にあると仮定して問題ありません (なぜなら、どんな市松の塗られ方でも黒い部分の左下のマスのうちひとつはこの中にあるため)。なので、左下マスを $(2K)^2$ すべて試すと、左下を決めれば各条件が満たされるかどうかは $O(1)$ で判定できるので、全体で $O(NK^2)$ になります。

ここから計算量を落とすには、条件 i を満たせるような左下マスの集合が $K * K$ の正方形区間のようになっていることに着目して、その範囲に $+1$ を足し、全体での max を取ります。ただし毎回範囲に $+1$ をしていては遅いので、差分を取って左下と右上に $+1$, 左上と右下に -1 を足しておき、あとで累積和を取ることで $O(N + K^2)$ で同じことが可能です。

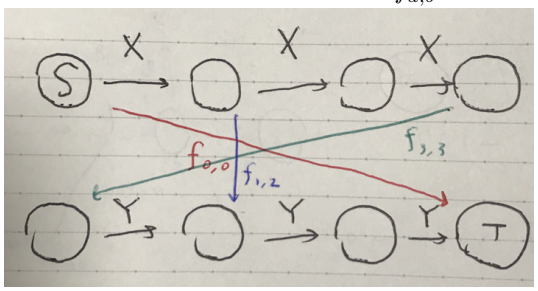
E: GraphXY

まずグラフ,S, T が与えられた時に,$X = x, Y = y$ の時の最短距離 $D_{x,y}$ がどういう振る舞いをするかを考えます。0 以上の整数の組 (a, b) に対して、「X の辺をちょうど a 回,Y の辺をちょうど b 回使った時に S から T への (整数が書いてある辺の重みのみを考慮した) 最短距離」 を $f_{a,b}$ とおきます。

すると、

$$D_{x,y} = \min_{a,b \geq 0} xa + yb + f_{a,b}$$

逆に次のようなグラフを考えれば、$f_{a,b}$ を 0 以上の整数値に自由に決めて上のように $D_{x,y}$ を設定できます。



なので、$f_{a,b}$ を決めることでこの D を入力の d と一致させることが目標です。

入力 d が与えられた時,d の値の範囲が高々 100 であることから、a や b が 100 より大きいと $xa + yb + f_{a,b}$ が 100 より大きくなってしまい D を d と一致させる役には立ちません。

よって $0 \leq a, b \leq 100$ をみたす整数の組 (a, b) に対して $f_{a,b}$ を決めることにします。$f_{a,b}$ は、任意の $1 \leq x \leq A, , 1 \leq y \leq B$ に対し $d_{x,y} \leq xa + yb + f_{a,b}$ を満たす、という条件のもとで最も小さくする (ただし 0 以上である必要はある) のがベストです。

なので、

$$f_{a,b} = \max(0, \max_{0 \leq a,b \leq 100}(d_{x,y} - xa + yb))$$

として、これで計算した D と入力の d が一致していればこれを出力、していなければ Impossible です。

F: ColoringBalls

以下では赤、青、白色のボールをそれぞれ R,B,W で表します。まずボールの列が与えられた時に、それが実現できるか判定することを考えます。例えば”WRRBRBBWWRRRRWRBBWWRRBBRR” というボール列を考えます。まず’W’ で区切って R,B からなる列の (多重) 集合に変換します。

{“RRBRBB”, “RRRR”, “RBB”, “RRBBRR”}

次に、RR を R に、BB を B にする操作をできるだけ繰り返します (つまり連続している R(B) をひとつの R(B) に置き換える操作をします)。

{“RBRB”, “R”, “RB”, “RBR”}

さらに、これらが次のうちどのグループに属しているかをカウントします：

グループ 1：“R”
グループ 2：“B”, “RB”, “BR”, “RBR”
グループ 3：“BRB”, “RBRB”, “BRBR”, “RBRBR”
グループ 4：“BRBRB”, “RBRBRB”, “BRBRBR”, “RBRBRBR”
:
(以下同様)

グループ id を降順にならべた配列を f とおきます。

上の例だと $f = [3, 2, 2, 1]$ です。

重要なのは、この配列 f が一緒になるようなどんなボール列も実現できるかどうかは変わらないということです。以下説明します。

まず列の集合に変換した結果が同じなら (つまり列の順番を並び替えたり無駄な W を挿入したりするだけ) 結果も明らかに同じなのでここは OK です。

次に連続する R の個数を増やしたりしてもそういう塗り方が出来るかは変わらないのでこれも OK です。

最後にグループ分けですが、赤で塗る操作を r, 青で塗る操作を b, どちらでもいいので塗る操作を?とかくと、

グループ 1 は r で実現できる
グループ 2 は rb で実現できる
グループ 3 は rb? で実現できる
グループ 4 は rb?? で実現できる
グループ 5 は rb??? で実現できる
:

ということがわかります。つまり同グループに属していれば操作列によって実現可能かどうかは一緒です。

最終的に、配列 f が同じならば、実現可能性は同じであることがわかりました。

配列 f となるようなボール列が実現可能かどうかは次のように判定できます。操作列を s とおきます。

f の各要素 x に、操作列の部分列であってグループ x を実現できるもの を割り当てられるかの判定をすれば良いです。(操作列の各文字はひとつにしか割り当てられません)

実は最適な割当は以下の貪欲法で構成することが出来ます (以下の貪欲で構成できなければ不可能です)。

はじめに、s の要素を左から見ていって、k 個目にでてきた r を f の k 個目の要素に割り当てます。これが出てきた場所を r_k とおきます。

次に、f の要素を左から見ていって、もし値が 2 以上なら、r_k より右にあってまだ割り当てていないうち最も左の b を割り当てます。これが出てきた場所を b_k とおきます。

最後に、f の要素を左から見ていって、値 x が 3 以上なら、b_k より右にあってまだ割り当てられていないもののうち左にある $x-2$ 個 (r,b どちらでもよい) を割り当てます。

これらはそれぞれグループ分け部分で説明した必要な文字列のうち、r,b,? の部分に対応するものをとる操作です。

この貪欲の証明は、

i) r を左から貪欲にとってよい

ii) b を (対応する r より右で) 左から貪欲にとってよい

iii) グループ番号が大きいものをできるだけ左に対応させるのがよい

iv) 残りの?に対応する部分も、対応する b より右で左から貪欲にとってよい

の 4 つを示せば完了します。省略しますがこれは証明できます。

この貪欲は $O(NK)$ で走ります。

ところで $N=70$ でありうるボールの色の列から生成される f は 418662 通りしか無いことがわかります。なので、実際に f を生成し、上の貪欲アルゴリズムで可能かどうか判定し、可能なら,f を生成するボール列が何通りあるかを求めて (これは適切な combination をかけ合わせるとできます) 十分高速に答えが求まります。

N, K の多項式時間で走る DP で計算することも出来ます。f にある 1 の個数、1 以外の個数 を全探索して、そのあと f の要素を小さい方から追加していく (1 を何個追加するか、2 を何個追加するか・・・ を順番に決めていく) のですが、「今追加しようとしている値」「これまで追加した値の総和」「これまで何個値を追加したか」を DP のキーに持てば可能です。

ABC086 / ARC089 Editorial

sigma425

A: Product

```
int main(){
        int a,b;
        cin>>a>>b;
        int c = a * b;
        if(c%2 == 0) puts("Even");
        else puts("Odd");
        return 0;
}
```

B: 1 21

```
int main(){
        string s,t;
        cin>>s>>t;
        int x = stoi(s+t);
        for(int i=1;i<=1000;i++){
                if(i*i == x){
                        puts("Yes");
                        return 0;
                }
        }
        puts("No");
}
```

C: Traveling

Let $t_0 = 0, (x_0, y_0) = (0, 0)$. For each i between 0 and $N-1$, we check the following: can we reach (x_{i+1}, y_{i+1}) at time t_{i+1} if we start (x_i, y_i) at time t_i? If the answer is no for at least one i, we print "No", otherwise we print "Yes".

We can check this as follows. Let $t := t_{i+1} - t_i$, $d := abs(x_i - x_{i+1}) + abs(y_i - y_{i+1})$. If $d \leq t$ and $t\%2 = d\%2$, the answer is yes, otherwise the answer is no.

D: Checker

The cell (x, y) is white iff the cell $(x, y + K)$ is black. Thus, we can replace a query $(x, y,' W')$ with $(x, y + K,' B')$. Assume that all c_i are 'B's.

The colors of (x, y) and $(x, y+2K), (x+2K, y)$ are same. Thus, we can replace a query $(x, y,' B')$ with $(x\%2K, y\%2K,' B')$. Assume that $0 \leq x, y < 2K$ for all queries.

If we fix a black cell at the lower-left corner of a $K \times K$ block, we can uniquely determine the colors of all cells. Here, we can assume that the black cell is inside $[0, 2K) \times [0, 2K)$. Therefore, by trying all $(2K)^2$ possibilities, and by checking the colors of all N cells that appear in the queries, we get an $O(NK^2)$ solution.

To make it faster, after we fix the black cell at the corner, we should compute the number of queried cells inside certain rectangles in $O(1)$. This can be done by pre-computing rectangle sums. Now the solution becomes $O(N + K^2)$.

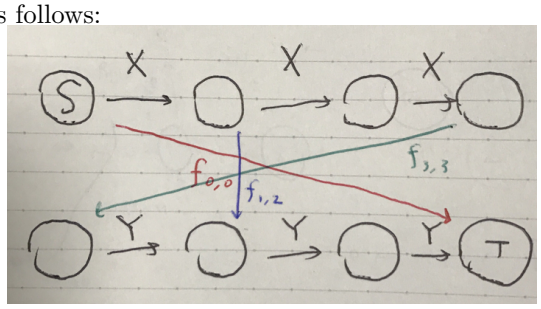
E: GraphXY

Let's fix a graph (and S, T). How does $D_{x,y}$ (the shortest path when $X = x, Y = y$) behave?

For a pair of non-negative integers (a, b), define $f(a, b)$ as the smallest possible sum of integers on an $S - T$ path that contains exactly a edges with 'X' and exactly b edges with 'Y'. Then,

$$D_{x,y} = \min_{a,b \geq 0} xa + yb + f_{a,b}$$

On the other hand, if D satisfies the equation above (for some $f_{a,b}$), we can construct a desired graph as follows:



Thus, our objective is to find $f_{a,b}$ that matches with given d.

Since each integer in d is at most d, we don't need to care the values of $f(a, b)$ for $a > 100$ or $b > 100$ because $xa + yb + f_{a,b} > 100$ in this case. Thus, we want to decide $f_{a,b}$ for each pair (a, b) such that $0 \leq a, b \leq 100$. We should minimize the value of $f_{a,b}$ under the constraint that $d_{x,y} \leq xa + yb + f_{a,b}$ for all $1 \leq x \leq A, , 1 \leq y \leq B$.

Therefore, define

$$f_{a,b} = \max(0, \max_{0 \leq a,b \leq 100} (d_{x,y} - xa + yb))$$

and compute D. If it matches with the input, we should print the graph described above. Otherwise, the answer is "Impossible".

F: ColoringBalls

We start with a slow solution: generate all 3^N sequences of balls of length N, and for each sequence, check if the sequence can be constructed by given operations.

However, we don't need to check all 3^N sequences. Some sequences turn out to be equivalent regardless of the operations.

Let R,B,W denote red, blue, and white, respectively. As an example, we use a sequence of balls "WRRBRBBWWRRRRWRBBWWRRBBRR". We want to transform it into a "canonlcal form" (while keeping the string equivalent to the original string).

First, obviously, we can split the string by 'W':

{"RRBRBB", "RRRR", "RBB", "RRBBRR"}

Next, we can replace "RR" with "R" and "BB" with "B":

{"RBRB", "R", "RB", "RBR"}

Next, replace each string with its group number:

Group 1 : "R"

Group 2 : "B", "RB", "BR", "RBR"

Group 3 : "BRB", "RBRB", "BRBR", "RBRBR"

Group 4 : "BRBRB", "RBRBRB", "BRBRBR", "RBRBRBR"

:

Here, the groups represent strings that can be obtained by the following sequences of operations:

Group 1 : r

Group 2 : rb

Group 3 : rb?

Group 4 : rb??

Group 5 : rb???

:

Finally, we can arbitrary shuffle the elements of the array of integers we get. In this case, we get $f = [3, 2, 2, 1]$.

It turns out that the number of distinct final arrays we get is $O(partition_number(N) * N)$. In particular, when $N = 70$, there are only 418662 possible fs.

Now, our solution is as follows. We generate all 418662 possible fs. For each f, check if it can be obtained by the given sequence of operations. If yes, count the number of sequences of balls that lead to f (this is simply the product of some binomial coefficients), and add it to the answer.

How to check if we can get f with the given sequence of operations?

Let s be the sequence of operations. For each integer x in f, we want to assign a subsequence of s that matches with x (for example, if $x = 4$, the subsequence must be "rb??"). Here, the subsequences must be disjoint.

We can assign subsequences greedily as follows:

- Suppose that f is sorted in non-decreasing order.
- For each k, assign the k-th 'r' in s (suppose that this is the r_k-th character in s) to $f[k]$.
- For each $f[k] = x$, (we check elements of f from left to right) if $x \geq 2$, assign the leftmost 'b' in s to the right of the r_k-th character of s to $f[k]$. Suppose that this is the b_k-th character.
- Finally, for each $f[k] = x$, if $x \geq 3$, we assign the leftmost $x - 2$ unused letters of s to the right of b_k-th to $f[k]$.

We omit the proof here, but it is relatively straightforward.

The solution above is fast enough to get accepted.

If you want a polynomial solution, we can do DP. We first fix the total number of 1s in f and the number of elements in f. Then, we add integers to f in the increasing order. Here, the keys of DP is (the ineger we will add next), (the sum of all integers we've added so far), and (the number of integers we've added so far). Since it's complicated and not significantly faster than the brute force above, we omit details.