みんなのプロコン 2019 予選 解説

DEGwer

2019/02/09

For International Readers: English editorial starts on page 5.

A: Anti-Adjacency

差が 1 の整数をともに選ばないように K 個の整数を選ぶとき、最大のものと最小のものの差は $2(K-1)$ 以上になります。逆に、正の奇数を小さい順に選べば、その差が $2(K-1)$ となるような例を構成できるので、$N \geq 2(K-1)+1$ かどうか判定すればよいです。

```
#include<stdio.h>
#include<vector>
#include<algorithm>
using namespace std;
int main()
{
        int n, k;
        scanf("%d%d", &n, &k);
        if (k * 2 - 1 <= n)printf("YES\n");
        else printf("NO\n");
}
```

B: Path

与えられる街のつながり方は、条件を満たすつながり方か、ある街が存在して他の全ての街と直接つながっているようなつながり方です。後者の場合には入力中に 3 回現れる整数があり、前者の場合はないので、同じ整数が入力中に 3 回現れるかどうかを判定すればよいです。

```
#include<stdio.h>
#include<vector>
#include<algorithm>
using namespace std;
int main()
```

```
{
        int d[10];
        for (int i = 1; i <= 4; i++)d[i] = 0;
        for (int i = 0; i < 6; i++)
        {
                int z;
                scanf("%d", &z);
                d[z]++;
        }
        if (d[1] <= 2 && d[2] <= 2 && d[3] <= 2 && d[4] <= 2)printf("YES\n");
        else printf("NO\n");
}
```

C: When I hit my pocket...

$B - A \leq 2$ のとき、ビスケットをお金に変換する操作とお金をビスケットに変換する操作を組合せても操作回数分のビスケットしか増えないので、お金を得る操作をする意味はありません。よってこの場合、答えは $K + 1$ です。

そうでない場合、ビスケットが A 枚以上あり、かつ残りの操作回数が 2 回以上あるなら、ビスケット A 枚をお金を介して操作 2 回でビスケット B 枚に変換してよいことも分かります。このお金を介する操作が可能な回数は $\max(0, \lfloor (K - (A - 1))/2 \rfloor)$ 回なので、あとは簡単な計算によりこの問題を解くことができます。

D: Ears

散歩後にすぬけ君の $1, 2, ..., L$ 番目の耳に入っている石の個数を順に $X_1, X_2, ..., X_L$ とします。

すぬけ君が座標 S で散歩を開始し、座標 T で終了したとします。また、散歩中に通った座標の最小値が D であり、最大値が U であるとします。このとき、

- $i <= D$ に対し、$X_i = 0$
- $D < i <= S$ に対し、X_i は 0 でない偶数
- $S < i <= T$ に対し、X_i は奇数
- $T < i <= U$ に対し、X_i は 0 でない偶数
- $U < i$ に対し、$X_i = 0$

となります。逆に、列 X' がある D, S, T, U に対し上の条件を満たすなら、各 i に対して i 個目の耳に X'_i 個の石が入っているような散歩の方法を簡単に構成できます。

以上より、DP[i][t]: i 番目の耳まで見て、D, S, T, U のうち t 個の地点を既に通過したときのりんごさんの必要な操作回数の最小値 として DP 配列を $O(L)$ 時間で更新していくことで、この問題を解くことができます。

E: Odd Subrectangles

行集合 A と列集合 B に対し、その交わりに属する整数の合計を 2 で割ったあまりを $f(A,B)$ とします。列集合 B を固定して考えましょう。

このとき、B との共通部分に奇数個の 1 が含まれる行を A に追加することにより $f(A,B)$ は必ず反転し、また、B との共通部分に偶数個の 1 が含まれる行を A に追加することにより $f(A,B)$ は反転しません。もし 1 つでも反転が起こるような行があれば、その行を A に入れるかどうかで $f(A,B)$ の値が変わるため、A の選び方のうちちょうど半分が $f(A,B)$ を 1 にすることになります。

逆に、反転が起こるような行がなければ、$f(A,B)$ の値は常に 0 です。よって、どの行も B との共通部分に偶数個の 1 を含むような列集合 B、すなわち B の各列をベクトルとして見たときにその和が ($\mathbb{F}_2^N$ 上で)0 になるような列集合 B の個数を求めればよいです。

B に含まれる列に対応するところが 1 であり、そうでないところが 0 であるような $\mathbb{F}_2$ 上の M 次のベクトル $\boldsymbol{v}_B$ を考えます。入力を $\mathbb{F}_2$ 上の $N \times M$ 行列と見れば、B の各列をベクトルとして見たときにその和が 0 になることは、入力の行列と $\boldsymbol{v}_B$ の積が 0 になること、すなわち $\boldsymbol{v}_B$ が入力を線形変換として見たときのカーネルに含まれていることと同値です。

カーネルの要素数はカーネルの次元を求めることで求まり、カーネルの次元は入力の行列のランクを求めることで求まるので、この問題は入力の行列のランクを掃き出し法などで求めることで解くことができます。特に、入力の行列のランクを r とすれば、答えは $2^{N+M-1} - 2^{N+M-r-1}$ です。

F: Pass

まず最初に、すべてのボールを区別して考えましょう。$i < j$ に対し、できる列の前から i 番目には、j 番目のすぬけ君が持っていたボールが並ぶことがないということは簡単に分かります。

逆に、どの i に対してもできる列の前から i 番目にあるボールを元々持っていたすぬけ君が前から i 番目以内のすぬけ君であるようなどの列も作ることができることが、以下のように証明できます。この条件を P とおきましょう。

帰納法を用いましょう。すぬけ君が 0 人の場合、明らかです。

そうでない場合、P を満たすような列は、以下のようにして先頭のすぬけ君を除いた列に対して条件 P を満たすような長さの 2 短い列に変換できます。

- 1 番目のすぬけ君が持っていたボールである、1 個目のボールを消去する
- ボールの列を先頭から見ていき、次に 1 番目のすぬけ君が持っていたボール (k 番目のボールとする) が現れるまでは、何もしない
- k 番目のボールを削除する
- それ以降、以下を繰り返す。
  - i 番目のボールを元々 i 番目のすぬけ君が持っているような最小の i を取る (そのような i がない場合、終了する)
  - このとき、i 番目より後ろに $i-1$ 番目以前のすぬけ君が持っていたボールがあるため、そのボールを取り除き、$i-1$ 番目のボールと i 番目のボールの間に挿入する

帰納法の仮定より、先頭のすぬけ君は自分を除いた列に対して条件 P を満たすような順にボールを受け取ることに注意すれば、先頭のすぬけ君の行動を適切に設定することで作ることができ、帰納法が回ります。

さて、ここで同じ色のボールを同一視しましょう。条件が言い換えられているので、できる可能性のある列の場合の数は、DP$[n][i]$: 作る列の先頭 n 個のボールを並べ終わり、そのうち i 個が赤であるような場合の数として更新していくことで求めることができます。時間計算量は $O(N^2)$ です。

Yahoo Programming Contest 2019 Editorial

DEGwer

February 9, 2019

A: Anti-Adjacency

When we choose K different integers so that no two of them differ by 1, the different between the largest and smallest chosen integer is at least $2(K-1)$. On the other hand, by choosing the smallest positive odd numbers, we can make a choice with the difference $2(K-1)$, so the problem can be solved by checking if $N \geq 2(K-1)+1$.

```
#include<stdio.h>
#include<vector>
#include<algorithm>
using namespace std;
int main()
{
        int n, k;
        scanf("%d%d", &n, &k);
        if (k * 2 - 1 <= n)printf("YES\n");
        else printf("NO\n");
}
```

B: Path

The possible ways in which the towns are connected in input are the ones that satisfy the condition, and the ones where there is a town that is directly connected to all other towns. In the latter case we have an integer that appears three times in input, and in the former case there is no such integer, thus the problem can be solved by checking if the same integer appears three times in input.

```
#include<stdio.h>
#include<vector>
#include<algorithm>
using namespace std;
int main()
{
        int d[10];
        for (int i = 1; i <= 4; i++)d[i] = 0;
        for (int i = 0; i < 6; i++)
        {
                int z;
                scanf("%d", &z);
                d[z]++;
        }
        if (d[1] <= 2 && d[2] <= 2 && d[3] <= 2 && d[4] <= 2)printf("YES\n");
        else printf("NO\n");
}
```

C: When I hit my pocket...

When $B - A \leq 2$, we don't need to perform the operation to gain money, since we can only gain at most one biscuit per operation from the combination of exchanging biscuits to money and exchanging money to biscuits. Thus, the answer in this case is $K + 1$.

Otherwise, it can be seen that, if we have A or more biscuits and two or more operations remaining, we can safely "exchange" A biscuits to B biscuits in two operations via money. This exchange via money can be performed $\max(0, \lfloor (K - (A - 1))/2 \rfloor)$ times, and the rest of the problem is an easy calculation.

D: Ears

Let $X_1, X_2, ..., X_L$ be the number of stones contained in the 1-st, 2-nd, ..., L-th ear after the walk, respectively.

Assume that Snuke started walking at coordinate S and finished at coordinate T. Also assume that the minimum and maximum coordinate visited while walking are D and U, respectively. Then, we have:

- For each i such that $i \leq D$, $X_i = 0$.
- For each i such that $D < i \leq S$, X_i is even and not zero.
- For each i such that $S < i \leq T$, X_i is odd.
- For each i such that $T < i \leq U$, X_i is even and not zero.
- For each i such that $U < i$, $X_i = 0$.

On the other hand, if a sequence X' satisfies the above conditions for some D, S, T, U, we can easily construct a walk that put X'_i stones in the i-th ear for each i.

Therefore, the problem can be solved by dynamic programming in OL time, where DP[i][t]: (the minimum number of operations required considering up to the i-th ear when we already passed t of the four points D, S, T, U).

E: Odd Subrectangles

Let $f(A, B)$ be the sum of the integers contained in the intersection of the row set A and the column set B, modulo 2. Let us first fix B.

Then, $f(A, B)$ gets inverted whenever a row with an odd number of 1 in the intersection with B is added, and does not get inverted when a row with an even number of 1 in the intersection with B is added. If there is a row (or more) that cause this inversion, exactly half of the ways to choose A have $f(A, B) = 1$, since whether or not this row is contained in A changes the value of $f(A, B)$.

On the other hand, if there is no such row, $f(A, B)$ is always 0. Thus, we need to find the number of the column sets B such that every row has an even number of 1 in the intersection with B. In other word, we need to find the column sets B such that when each column in B is seen as a vector, the sum of those vectors is 0 (in $\mathbb{F}_2^N$).

Let us consider a M-dimensional vector $\boldsymbol{v}_B$ where the elements corresponding to columns contained in B are 1, and the other elements are 0. If we regard the input as an $N \times M$-matrix in $\mathbb{F}_2$, the sum of the columns in B regarded as vectors is 0 if and only if the product of the given matrix and $\boldsymbol{v}_B$ is 0, that is, $\boldsymbol{v}_B$ is contained in the kernel of the input regarded as a linear map.

The number of the elements of the kernel can be found by finding the dimension of the kernel, which can be found by finding the rank of the input regarded as a matrix, for which we can use Gaussian elimination, for example. More specifically, the answer is $2^{N+M-1} - 2^{N+M-r-1}$ where r is the rank of the given matrix.

F: Pass

First, let us distinguish all the balls. It can be easily seen that, for $i < j$, the balls originally held by the j-th Snuke never comes as the i-th ball from the front in the resulting sequence.

On the other hand, it can be proved we can make any sequence of balls such that, for each i, the i-th ball from the front was originally held by some of the frontmost i Snukes, as follows. Let P denote this condition.

Let us prove it by induction on the number of Snukes. If there is zero Snuke, the proposition is obviously true.

Otherwise, any sequence of balls satisfying P can be converted to a sequence with two less balls that satisfies P for the sequence of Snukes without the frontmost Snuke, as follows:

- Erase the first ball, which was originally held by the frontmost Snuke.
- Let us go through the sequence of balls from front to back. Change nothing until we reach the next ball originally held by the frontmost Snuke (let this ball be the k-th ball from the front).
- Erase the k-th ball.
- For the remaining balls (the $(k+1)$-th and subsequent balls), repeat the following:
  - Take the smallest i $(i > k)$ such that the i-th ball from the front was originally held by the i-th Snuke from the front. (Terminate if there is no such i.)
  - Then, after the i-th ball, there must be a ball originally held by the $(i-1)$-th Snuke or someone ahead of him. Remove that ball and insert it between the $(i-1)$-th ball and the i-th ball.

From the inductive assumption, the frontmost Snuke can receive a sequence of balls that satisfies the condition P for the sequence of Snukes without himself. By noticing this and the conversion above, we can properly decide his actions to make any sequence of balls that satisfies the condition P, and thus we have a proof by induction.

Let us stop distinguishing the balls of the same colors here. The condition is now rephrased, and we can find the number of the possible sequences of balls that can be made by dynamic programming, where DP$[n][i]$: (the number of the possible prefixes of length n with i red balls), in $\mathrm{O}(N^2)$ time.