El test Fizz Buzz

El test Fizz Buzz es un sencillo problema de programación que algunas empresas usan para descartar a los malos programadores. Esperemos que hoy todos lo paséis ☺

Dado un natural n, hay que escribir todos los números entre 1 y n, en orden, y uno por linea. Pero para los múltiplos de 3 hay que escribir "`Fizz`" en lugar del número, para los múltiplos de 5 hay que escribir "`Buzz`" en lugar del número, y para los múltiplos de 3 y 5 hay que escribir "`FizzBuzz`" en lugar del número.

Entrada

La entrada consiste en un natural n entre 1 y 100.

Salida

Escribid las n lineas que pide el test.

Ejemplo de entrada 1

```
15
```

Ejemplo de salida 1

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

Ejemplo de entrada 2

```
4
```

Ejemplo de salida 2

```
1
2
Fizz
4
```

Solución

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++){
        bool done = false;
        if (i%3 == 0){
                cout << "Fizz";
                done = true;
        }
        if (i%5 == 0){
                cout << "Buzz";
                done = true;
        }
        if (!done)
                cout << i;
        cout << '\n';
    }
}
```

// Autor de la solución: Javier Nistal (Miembro del comité organizador)

Fantasma Blitz

Para jugar al Fantasma Blitz se necesitan cinco objetos, cada uno con una forma y un color diferente. En cada ronda se destapa una carta, que tiene dos dibujos con formas diferentes y con colores también diferentes. El objetivo del juego es localizar rápidamente el objeto identificado por la carta. En el paquete básico hay dos tipos de cartas:

- En las cartas de tipo 1, uno de los dibujos tiene la combinación de forma y color de uno de los objetos, y el otro dibujo no. En este caso el objeto correcto es el que aparece dibujado en la carta.
- En las cartas de tipo 2, ninguno de los dibujos se corresponde a ningún objeto con su misma forma y color. De hecho, para cada objeto, no aparece simultáneamente en la carta su forma y su color, ni siquiera en dibujos diferentes. En este caso el objeto correcto es el único que no aparece ni en forma ni en color en la carta.

Además, todas las cartas del paquete básico cumplen que sus dos dibujos tienen formas y colores diferentes entre sí.

Por ejemplo, supongamos que los objetos son un fantasma blanco, una botella verde, un libro azul, un sofá rojo y un ratón gris. Si una carta contiene una botella verde y un sofá azul, el objeto correcto es la botella verde. Sin embargo, si en una carta aparecen un fantasma gris y una botella roja, entonces el objeto correcto es el libro azul.

Dada la lista de cinco objetos, y una lista de cartas, determinad para cada carta si pertenece al paquete básico. En caso afirmativo, determinad cuál es el objeto correcto correspondiente a dicha carta.

Entrada

La entrada empieza con cinco parejas de palabras que describen la forma y el color de cada objeto. No hay dos formas iguales ni dos colores iguales. Siguen diversas cartas, definidas con dos parejas de palabras. Cada pareja es la forma y el color de uno de sus dibujos. Todas las formas y colores de los dibujos se encuentran en la lista de los objetos. Todas las palabras de la entrada están formadas por entre 1 y 20 letras.

Salida

Para cada carta, si pertenece al paquete básico escribid la forma y el color del objeto correcto correspondientes a dicha carta. En otro caso, escribid "`Expansion`".

Puntuación

- **Test1:** Resolver entradas donde todas las cartas son del paquete básico. **50 Puntos**
- **Test2:** Resolver entradas de todo tipo **50 Puntos**

Ejemplo de entrada 1

```
Fantasma blanco
Botella verde
Libro azul
Sofa rojo
Raton gris
Sofa gris Libro blanco
Botella verde Libro gris
Raton rojo Libro verde
Botella blanco Sofa azul
Raton gris Botella rojo
```

Ejemplo de salida 1

```
Botella verde
Botella verde
Fantasma blanco
Raton gris
Raton gris
```

Ejemplo de entrada 2

```
Fantasma blanco
Botella verde
Libro azul
Sofa rojo
Raton gris
Sofa gris Libro blanco
Botella verde Libro gris
Raton rojo Libro verde
Botella blanco Sofa azul
Raton gris Botella rojo
Raton gris Raton rojo
Botella verde Botella verde
Fantasma azul Libro rojo
```

Ejemplo de salida 2

```
Botella verde
Botella verde
Fantasma blanco
Raton gris
Raton gris
Expansion
Expansion
Expansion
```

Ejemplo de entrada 3

```
A a
b B
c c
D D
eE fF
A a b c
A a b B
A B c D
D D D fF
eE c A D
A c c a
eE fF D B
```

Ejemplo de salida 3

```
A a
Expansion
eE fF
Expansion
b B
Expansion
eE fF
```

Solución

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    vector<string> formas(5);
    vector<string> colores(5);
    map<string, int> formasIdx;
    map<string, int> coloresIdx;
    for (int i = 0; i < 5; i++){
        cin >> formas[i] >> colores[i];
        formasIdx[formas[i]] = i;
        coloresIdx[colores[i]] = i;
    }
    string f1, c1, f2, c2;
    while (cin >> f1 >> c1 >> f2 >> c2){
        int forma1 = formasIdx[f1], color1 = coloresIdx[c1];
        int forma2 = formasIdx[f2], color2 = coloresIdx[c2];
        if (
              forma1 != forma2 and
              color1 != color2 and
              forma1 != color2 and
              forma2 != color1 and
              (forma1 != color1 or forma2 != color2)
        ){
               if (forma1 == color1)
                      cout << formas[forma1] << ' ' << colores[color1] << '\n';
               else if (forma2 == color2)
                      cout << formas[forma2] << ' ' << colores[color2] << '\n';
               else {
                      int indiceRestante = 10-forma1-color1-forma2-color2;
                      cout << formas[indiceRestante] << ' ';
                      cout << colores[indiceRestante] << '\n';
               }
        }
        else{
               cout << "Expansion\n";
        }
    }
}
```

// Autor de la solución: Javier Nistal (Miembro del comité organizador)

4-SAT

El conocido problema 4-SAT consiste en lo siguiente: Tenemos m variables booleanas (sólo pueden valer *cierto* o *falso*). Además, tenemos n cláusulas, formadas cada una por cuatro variables, no necesariamente diferentes. Veamos un ejemplo de clásula:

$$x_1 \vee x_5 \vee \neg x_3 \vee x_4$$

El símbolo $\vee$ es la `OR` booleana, y el símbolo $\neg$ es la `NOT` booleana. Así pues, la cláusula es falsa sólo si x_1, x_5 y x_4 son falsas, y x_3 es cierta.

El problema 4-SAT original pide asignar valores a las variables de manera que todas las cláusulas sean ciertas, o decir que no hay solución. Como ese problema es demasiado difícil, os pedimos una versión más sencilla: ¿Podéis encontrar una asignación tal que al menos el 90% de las cláusulas sean ciertas?

Entrada

La entrada tiene diversos casos. Cada uno empieza con n y m, seguidas de las n claúsulas, codificadas con cuatro enteros entre $-m$ y m, sin ningún cero. Una i positiva indica x_i, y una i negativa indica $\neg x_i$. Se garantiza que la entrada se ha generado de forma **aleatoria**.

Salida

Escribid una línea para cada caso. Si no existe solución, escribid "`WA`". Si existe, escribid m caracteres indicando el valor de las variables en orden: '`0`' para falso y '`1`' para cierto. Si hay más de una posible asignación, cualquiera sirve.

Puntuación

- **Test1:** Casos donde $n \leq 10$ y $m \leq 4$ y se garantiza que existe solución. **20 Puntos**
- **Test2:** Casos donde $n \leq 10^6$ y $100 \leq m \leq 10^4$. **80 Puntos**

Ejemplo de entrada

```
2 4
-2 3 -4 3
-2 1 2 -2

2 3
-2 -1 -2 1
2 2 -3 -1

5 3
1 2 2 2
1 3 3 3
-1 -2 -2 -2
-1 2 2 2
1 3 3 3

2 5
-4 -5 -5 -4
2 1 -2 -4

4 6
-4 2 -2 -3
-4 -6 1 -6
-3 -6 -1 -1
1 -2 -3 6
```

Ejemplo de salida

```
1011
110
011
00000
101100
```

Solución

LOL

En la jerga de internet, LOL, el acrónimo inglés de "laughing out loud" (riendo a carcajadas) se usa con frecuencia para describir una situación supuestamente divertida. Veamos si este problema os parece divertido...

Dados dos naturales n y m, debéis escribir una matriz $n \times m$ con los caracteres 'L' y 'O', de manera que el número de "LOL"s que contenga sea el máximo posible, contando las apariciones horizontales, verticales y diagonales.

Por ejemplo, para $n = 3$ y $m = 7$ la solución óptima es

```
LLLLLLL
OOOOOOO
LLLLLLL
```

con 17 apariciones de "LOL". Como otro ejemplo, para $n = 1$ y $m = 2$

```
OL
```

es una de las cuatro soluciones posibles, todas con ninguna aparición de "LOL".

Entrada

La entrada consiste en dos naturales n y m, ambos entre 1 y 100.

Salida

Sea q el máximo número de "LOL"s que habéis encontrado para esta combinación de n y m. Escribid primero una línea con n, m y q separadas con un espacio. Escribid a continuación n lineas con m caracteres 'L' o 'O' cada una. La matriz debe contener exactamente q "LOL"s. Si hay más de una posible matriz, escribid cualquiera de ellas.

Puntuación

Hay 50 juegos de pruebas privados, todos diferentes, que se evalúan independientemente. Para cada uno, si el formato de vuestra salida no es exactamente el requerido, o si q no es exactamente el número de "LOL"s de la matriz escrita, tendréis cero puntos. En otro caso, recibiréis dos puntos si vuestra q es igual (o superior) a la mejor que ha sido capaz de encontrar el autor de este problema para esta combinación de n y m.

Ejemplo de entrada 1

```
3 7
```

Ejemplo de salida 1

```
3 7 17
LLLLLLL
OOOOOOO
LLLLLLL
```

Ejemplo de entrada 2

```
1 2
```

Ejemplo de salida 2

```
1 2 0
LL
```

Solución

Bombillas

En el piso del autor de este problema no se gana para sustos. Por ejemplo, la última factura de luz llegó en papel extra-grande para que cupiera el precio. Por eso se ha decidido invertir en n bombillas de bajo consumo, que se han instalado todas en la cocina.

Una bombilla está definida por dos parámetros: el consumo inicial i, que es la cantidad de julios que gasta para encenderse, y el consumo temporal t, que es la cantidad de julios que gasta por cada minuto que esté encendida. Así, si queremos iluminar la cocina durante poco tiempo, quizá convenga más encender una bombilla que tenga un consumo inicial bajo, aunque su consumo temporal sea más alto, mientras que si queremos iluminarla durante más tiempo tal vez convenga encender una bombilla cuyo consumo temporal sea menor, a pesar de que el inicial sea más alto. De hecho, puede que hasta nos interese dejar encendida una bombilla aunque durante un rato nadie esté usando la cocina, para ahorrarnos el coste de volver a encenderla más tarde. ¡Cada julio cuenta!

Aquí es donde necesitamos vuestra ayuda. Tenemos la información de las n bombillas, y una lista con los m intervalos en los que la cocina está ocupada a lo largo del día. Las horas empiezan a las 00:00 y acaban a las 23:59. Al principio del día todas las luces están apagadas. Siempre que haya alguien en la cocina tiene que haber al menos una luz encendida. ¿Cuál es la mínima energía necesaria para iluminar todos los intervalos?

Entrada

La entrada consiste en diversos casos. Cada caso empieza con n y m, seguidas de n pares de enteros i y t describiendo las bombillas, seguidos de m pares de horas en el formato `hh:mm`, indicando el momento inicial y final de cada intervalo. Las $2m$ horas de cada caso aparecen en orden estrictamente creciente. Podéis suponer $1 \leq n \leq 2000$, $m \geq 1$, $1 \leq i \leq 2 \cdot 10^5$, $1 \leq t \leq 2000$, $00 \leq hh \leq 23$, y $00 \leq mm \leq 59$.

Salida

Para cada caso, escribid el mínimo consumo de energía necesario.

Puntuación

- **Test1:** Resolver entradas donde $n = 1$. **20 Puntos**
- **Test2:** Resolver entradas donde $n \leq 2$. **20 Puntos**
- **Test3:** Resolver entradas donde $n \leq 100$. **30 Puntos**
- **Test4:** Resolver entradas de todo tipo. **30 Puntos**

Ejemplo de entrada 1

```
1 1
1000 10
08:00 09:00
1 2
1000 10
09:00 10:00
11:00 12:00
1 2
1000 10
10:00 11:00
13:00 14:00
```

Ejemplo de salida 1

```
1600
2800
3200
```

Ejemplo de entrada 2

```
2 1
1000 10
200 100
10:00 10:05
2 1
1000 10
200 100
10:00 10:30
2 2
1000 10
200 100
10:00 10:05
12:00 12:30
```

Ejemplo de salida 2

```
700
1300
2000
```

Ejemplo de entrada 3

```
4 4
1000 20
500 15
300 18
150 150
10:00 10:01
10:02 10:05
10:10 10:30
11:15 13:20
```

Ejemplo de salida 3

```
3215
```

Solución

```
#include <bits/stdc++.h>

using namespace std;

typedef pair<int, int> pii;

const int INF = 1e9;

int main() {
	int n, m;
	while (cin >> n >> m){
	 vector<int> init(n);
	 vector<int> temp(n);
	 for (int i = 0; i < n; i++)
		cin >> init[i] >> temp[i];
	 vector<pii> intervals(m);
	 for (int i = 0; i < m; i++){
		string a, b;
		cin >> a >> b;
		intervals[i].first = (a[0]-'0')*600 + (a[1]-'0')*60 + (a[3]-'0')*10 +
(a[4]-'0');
		intervals[i].second = (b[0]-'0')*600 + (b[1]-'0')*60 + (b[3]-'0')*10 +
(b[4]-'0');
	 }
	 vector<vector<int>> dp(m, vector<int>(n));
	 vector<int> best(m, INF);
	 for (int i = 0; i < n; i++){
		dp[0][i] = init[i] + (intervals[0].second-intervals[0].first)*temp[i];
		best[0] = min(dp[0][i], best[0]);
	 }
	 for (int i = 1; i < m; i++){
		for (int j = 0; j < n; j++){
			dp[i][j] =  min(dp[i-1][j] + (intervals[i].second -
intervals[i-1].second)*temp[j], best[i-1]+init[j]+(intervals[i].second-
intervals[i].first)*temp[j]);
			best[i] = min(best[i], dp[i][j]);
		}
	 }

	 cout << best[m-1] << '\n';
	}
}
```

// Autor de la solución: Javier Nistal (Miembro del comité organizador)

Malditos unos

Dada una secuencia de naturales $x_1 \dots x_n$ entre 0 y 9, escoged como operarlos mediante sumas y productos, poniendo los paréntesis que queráis, de manera que el resultado final sea el máximo posible. Por ejemplo, para la secuencia

$$1\ 3\ 2\ 0\ 9\ 1\ 1\ 3\ 1\ 5$$

el máximo resultado posible es

$$(1+3)*(2+0)*9*(1+1)*(3+1)*5 = 2880\,.$$

Fijáos que no está permitido cambiar el orden de los números dados, ni unirlos entre si para formar números de dos o más dígitos.

Entrada

La entrada consiste en diversos casos. Cada caso comienza con n, seguida de n números entre 0 y 9. Asumid $1 \leq n \leq 10^4$.

Salida

Para cada caso, escribid el máximo resultado posible módulo $10^9 + 7$.

Puntuación

- **Test1:** **5 Puntos**

  Resolver casos donde todas las x_i están entre 2 y 9, como los del ejemplo 1.

- **Test2:** **15 Puntos**

  Resolver casos donde $n \leq 10$ y el resultado sin hacer módulos no sería superior a 10^9, como los del ejemplo 2.

- **Test3:** **20 Puntos**

  Resolver casos donde $n \leq 30$ y el resultado sin hacer módulos no sería superior a 10^{18}, como los del ejemplo 3.

- **Test4:** **60 Puntos**

  Resolver casos de todo tipo.

Ejemplo de entrada 1

```
6    2 3 4 5 6 7
12   9 9 9 9 9 9 9 9 9 9 9 9
```

Ejemplo de salida 1

```
5040
429534507
```

Ejemplo de entrada 2

```
10   1 3 2 0 9 1 1 3 1 5
9    2 1 1 2 1 1 1 1 3
9    3 1 1 2 1 2 1 1 3
2    0 0
```

Ejemplo de salida 2

```
2880
108
216
0
```

Ejemplo de entrada 3

```
15   2 1 1 1 0 2 1 1 1 0 1 1 2 2 2
```

Ejemplo de salida 3

```
648
```

Ejemplo de entrada 4

```
30   2 1 1 2 1 1 2 2 1 1 2 1 3 2 1 1 2 1 1 1 1 3 2 1 1 2 1 2 1 2
```

Ejemplo de salida 4

```
11337408
```

Solución

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const ll MOD = 1e9+7;

int main(){
    int n;
    while (cin >> n){
        vector<int> v(n);
        for (int i = 0; i < n; i++){
                cin >> v[i];
                if (v[i] == 0){
                        v.pop_back();
                        n--;
                        i--;
                }
        }
        if (n == 0){
                cout << "0\n";
                continue;
        }
        ll ans = 1;
        int idx = n-1;
        while (idx >= 0){
                if (v[idx] >= 3){
                        if (idx == 0 or v[idx-1] > 1 or (idx > 1 and v[idx-2] < v[idx])){
                                ans = ans*v[idx]%MOD;
                                v.pop_back();
                                idx--;
                        }
                        else{
                                ans = ans*(v[idx]+1)%MOD;
                                v.pop_back();
                                v.pop_back();
                                idx -= 2;
                        }
                }
                else if (v[idx] == 2){
                        if (idx == 0 or v[idx-1] > 1){
                                ans = ans*v[idx]%MOD;
                                v.pop_back();
                                idx--;
                        }
                        else{
                                v.pop_back();
                                idx--;
                                v[idx] = 3;
                        }
```

```
                }
                else{
                        v.pop_back();
                        idx--;
                        if (idx >= 0){
                                v[idx]++;
                        }
                }
        }
        cout << ans << '\n';
    }
}
```

// Autor de la solución: Javier Nistal (Miembro del comité organizador)

Anhelos

Se dice que siempre deseamos lo que no tenemos... Se han reunido n amigos, y el amigo i-ésimo trae a_i caramelos. Sin embargo, a nadie le gustan sus propios caramelos, sino únicamente los de los demás. Se os pide que calculéis, para cada amigo, cuántos caramelos tienen los demás.

Entrada

La entrada consiste en diversos casos. Cada caso empieza con n, seguido de los n a_i en orden. Suponed $1 \leq n \leq 50000$, y $1 \leq a_i \leq 100$.

Salida

Para cada caso, escribid n enteros: Para cada i, escribid el número de caramelos que tienen todos sus amigos en total.

Puntuación

- **Test1:** Resolver entradas donde $n \leq 500$. **50 Puntos**
- **Test2:** Resolver entradas de todo tipo. **50 Puntos**

Ejemplo de entrada 1

```
3   1 2 3
2   4 5
1   6
5   12 53 24 31 99
10  10 10 10 10 10 10 10 10 10 10
```

Ejemplo de salida 1

```
5 4 3
5 4
0
207 166 195 188 120
90 90 90 90 90 90 90 90 90 90
```

Ejemplo de entrada 2

```
1  1
1  100
2  1 1
2  100 100
2  1 100
2  100 1
```

Ejemplo de salida 2

```
0
0
1 1
100 100
100 1
1 100
```

Solución

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    int n;
    while (cin >> n){
        vector<int> v(n);
        int suma = 0;
        for (int& x : v){
                cin >> x;
                suma += x;
        }
        for (int i = 0; i < n-1; i++){
                cout << suma-v[i] << ' ';
        }
        cout << suma-v[n-1] << '\n';
    }
}
```

// Autor de la solución: Javier Nistal (Miembro del comité organizador)

Jardines imperiales

En Tokio, una de las muchas cosas destacables son los Jardines imperiales. Sin embargo, es difícil saber cuando están abiertos, ya que en la puerta hay un cartel que, traducido, dice lo siguiente (y esto es totalmente cierto, como demuestra la foto que se puede ver a la derecha):

Los Jardines están abiertos los días que cumplan alguno de los puntos siguientes:

1. *Miércoles.*
2. *Jueves.*
3. *Sábados.*
4. *Domingos.*
5. *Fiestas Nacionales, sin contar el Cumpleaños del Emperador, el 23 de diciembre.*
6. *Lunes siguiendo inmediatamente una Fiesta Nacional que caiga en domingo.*
7. *Martes, sin contar los martes inmediatamente después de un lunes de los puntos 5 ó 6.*

Sin embargo, los Jardines están siempre cerrados en el periodo entre el 28 de diciembre y el 3 de enero.

Dada una lista con las n Fiestas Nacionales, ¿podéis determinar si los Jardines Imperiales están abiertos en ciertas fechas dadas?

Entrada

La entrada empieza con n, seguido de las n Fiestas Nacionales en orden. Siguen diversas fechas en orden. Todas las Fiestas Nacionales y fechas están entre el 1-1-2014 y el 31-12-2015 y siguen estrictamente el formato de los ejemplos. Como el Cumpleaños del Emperador siempre es Fiesta Nacional, en la entrada aparece como tal.

Salida

Para cada fecha, escribid en una línea "`ABIERTOS`" o "`CERRADOS`". Asumid que no hay Fiestas Nacionales antes del día 1-1-2014.

Puntuación

- **Test1:** Casos donde sólo se preguntan miércoles, jueves, sábados y domingos, como en el Ejemplo 1. **20 Puntos**
- **Test2:** Casos donde sólo se preguntan Fiestas Nacionales, nunca en lunes ni martes, como en el Ejemplo 2. **20 Puntos**
- **Test3:** Casos donde sólo se preguntan lunes, como en el Ejemplo 3. **20 Puntos**
- **Test4:** Casos de todo tipo, como en el Ejemplo 4. **40 Puntos**

Ejemplo de entrada 2

```
8
23-4-2014
10-12-2014
23-12-2014
24-1-2015
15-10-2015
23-10-2015
13-12-2015
23-12-2015

23-4-2014
10-12-2014
23-12-2014
24-1-2015
15-10-2015
23-10-2015
13-12-2015
23-12-2015
```

Ejemplo de salida 2

```
ABIERTOS
ABIERTOS
ABIERTOS
ABIERTOS
ABIERTOS
ABIERTOS
ABIERTOS
ABIERTOS
```

Ejemplo de entrada 1

```
8
10-1-2014
8-3-2014
9-3-2014
5-10-2014
23-12-2014
7-6-2015
9-12-2015
23-12-2015

16-2-2014
17-9-2014
21-6-2015
19-9-2015
24-9-2015
30-9-2015
31-12-2015
```

Ejemplo de salida 1

```
ABIERTOS
ABIERTOS
ABIERTOS
ABIERTOS
ABIERTOS
ABIERTOS
CERRADOS
```

Ejemplo de entrada 3

```
10
30-1-2014
21-2-2014
17-6-2014
5-10-2014
26-11-2014
18-12-2014
23-12-2014
18-10-2015
30-10-2015
23-12-2015

9-6-2014
11-8-2014
29-12-2014
27-4-2015
4-5-2015
25-5-2015
19-10-2015
```

Ejemplo de salida 3

```
CERRADOS
CERRADOS
CERRADOS
CERRADOS
CERRADOS
CERRADOS
ABIERTOS
```

Ejemplo de entrada 4

```
24
6-4-2014
8-5-2014
19-5-2014
22-5-2014
31-7-2014
25-8-2014
22-12-2014
23-12-2014
29-12-2014
6-1-2015
31-1-2015
11-2-2015
26-4-2015
6-5-2015
10-6-2015
12-6-2015
14-6-2015
28-6-2015
27-7-2015
19-9-2015
26-10-2015
18-11-2015
18-12-2015
23-12-2015

15-2-2014
11-3-2014
14-4-2014
20-8-2014
24-8-2014
5-12-2014
13-2-2015
2-3-2015
9-4-2015
10-5-2015
20-6-2015
15-7-2015
20-9-2015
9-10-2015
29-11-2015
9-12-2015
10-12-2015
```

Ejemplo de salida 4

```
ABIERTOS
ABIERTOS
CERRADOS
ABIERTOS
ABIERTOS
CERRADOS
CERRADOS
CERRADOS
ABIERTOS
ABIERTOS
ABIERTOS
ABIERTOS
ABIERTOS
CERRADOS
ABIERTOS
ABIERTOS
ABIERTOS
```

Solución

```
#include <bits/stdc++.h>
using namespace std;
int diasMes[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
struct fecha{
    int d, m, a;
    fecha(){ d = m = a = 0; }
    fecha (int D, int M, int A){
        d = D;
        m = M;
        a = A;
    }
    fecha prev(){
        int D = d, M = m, A = a;
        D--;
        if (D == 0){
                M--;
                if (M == 0){
                        M = 12;
                        A--;
                }
                D = diasMes[M];
        }
        return fecha(D, M, A);
    }
    fecha next(){
        int D = d, M = m, A = a;
        D++;
        if (D > diasMes[M]){
                M++;
                D = 1;
                if (M == 13){
                        A++;
                        M = 1;
                }
        }
        return fecha(D, M, A);
    }
    bool read(){
        d = m = a = 0;
        string s;
        if (cin >> s){
                int idx = 0;
                while (s[idx] != '-'){
                        d = 10*d + s[idx]-'0';
                        idx++;
                }
                idx++;
                while (s[idx] != '-'){
                        m = 10*m + s[idx]-'0';
                        idx++;
                }
                idx++;
```

```
                a = 1000*(s[idx]-'0') + 100*(s[idx+1]-'0') + 10*(s[idx+2]-'0') +
(s[idx+3]-'0');
                return true;
        }
        return false;
    }
    int id(){
        return 31*12*(a-2014)+31*(m-1)+d-1;
    }
};
int main(){
    map<int, int> diaSem;
    fecha actFecha = fecha(1, 1, 2014);
    int actDiaSem = 2;
    while (actFecha.a < 2016){
        diaSem[actFecha.id()] = actDiaSem;
        actFecha = actFecha.next();
        actDiaSem = (actDiaSem+1)%7;
    }
    int n;
    cin >> n;
    map<int, bool> fiesta;
    while(n--){
        fecha F;
        F.read();
        fiesta[F.id()] = true;
    }
    fecha F;
    while (F.read()){
        int dSem = diaSem[F.id()];
        bool ans = false;
        if ((28 <= F.d and F.m == 12) or (3 >= F.d and F.m == 1)){
                ans = false;
        }
        else if (dSem == 2 or dSem == 3 or dSem == 5 or dSem == 6){
                ans = true;
        }
        else if (fiesta[F.id()] and (F.d != 23 or F.m != 12)){
                ans = true;
        }
        else if (dSem == 0 and fiesta[F.prev().id()]){
                ans = true;
        }
        else if (dSem == 1){
                fecha prev = F.prev();
                fecha prev2 = prev.prev();
                ans = not ((fiesta[prev.id()] and (prev.d != 23 or prev.m != 12)) or
fiesta[prev2.id()]);
        }
        cout << (ans ? "ABIERTOS\n" : "CERRADOS\n");
    }
}
```

// Autor de la solución: Javier Nistal (Miembro del comité organizador)

Muestreo de velociraptores (2)

Es bien sabido que los velociraptores son una amenaza real y por ello hemos trazado un gran plan desde la OIE. Se ha descubierto un lugar donde, curiosamente, duermen cada noche n velociraptores en fila india. Sin embargo, la última misión fue un completo fracaso y vamos a probar de nuevo, cambiando un poco el objetivo.

Existen k especies distintas de velociraptor, y queremos capturar al menos un ejemplar de m especies diferentes. Por miedo a despertarlos, un requisito esencial es que todos los velociraptores capturados duerman consecutivamente. ¿Podéis decir el menor número que habrá que capturar?

Entrada

La entrada contiene diversos casos. Cada caso empieza con n, k y m. Finalmente, viene la especie de cada uno de los n velociraptores, en orden. Suponed $1 \leq m \leq k \leq n \leq 10^6$, que las especies se numeran entre 0 y $k-1$, y que entre los n velociraptores hay al menos m de diferentes especies.

Salida

Para cada caso, escribid el menor número de velociraptores consecutivos que es necesario capturar.

Puntuación

- **Test1:** Resolver casos con $m = 1$ como el Ejemplo 1. **5 Puntos**
- **Test2:** Resolver casos con $n \leq 100$. **15 Puntos**
- **Test3:** Resolver casos con $n \leq 1000$. **15 Puntos**
- **Test4:** Resolver casos con $n \leq 10^5$. **15 Puntos**
- **Test5:** Resolver casos de todo tipo. **50 Puntos**

Ejemplo de entrada 1

```
4 3 1
0 1 1 2
```

Ejemplo de salida 1

```
1
```

Ejemplo de entrada 2

```
10 6 3
1 4 1 5 3 5 5 4 5 2
10 5 5
3 4 0 4 2 2 1 3 4 1
10 8 7
4 7 2 1 6 4 5 0 2 1
```

Ejemplo de salida 2

```
3
6
7
```

Solución

```
#include <bits/stdc++.h>

using namespace std;

int main(){
    int n, k, m;
    while (cin >> n >> k >> m){
        vector<int> v(n);
        for (int& x : v)
                cin >> x;
        vector<int> s(k, 0);
        int l = 0, r = 0;
        int ans = n;
        while (true){
                if (m > 0){
                        if (r == n)
                                break;
                        if (s[v[r++]]++ == 0)
                                m--;
                }
                else{
                        ans = min(ans, r-l);
                        if (s[v[l++]]-- == 1)
                                m++;
                }
        }
        cout << ans << '\n';
    }
}
```

// Autor de la solución: Javier Nistal (Miembro del comité organizador)

Monstruos en un mapa (2)

Haced un programa que, dados un mapa con monstruos, y unas posiciones inicial y final, diga si es posible ir desde la una hasta la otra sólo con movimientos horizontales y verticales, y manteniendo siempre una distancia de seguridad con los monstruos. Aquí usaremos la distancia Manhattan: dos casillas (a, b) y (c, d) se encuentran a distancia $|a - c| + |b - d|$. Por ejemplo, la distancia entre $(2, 8)$ y $(5, 1)$ es $|2 - 5| + |8 - 1| = 3 + 7 = 10$.

Entrada

La entrada consiste en diversos casos. Cada caso comienza con el número de filas $n > 0$ y el número de columnas $m > 0$ del mapa. Siguen n filas con m caracteres cada una. Un punto indica una posición vacía. Los monstruos se indican con dígitos, letras minúsculas y letras mayúsculas, que codifican la distancia de seguridad mínima que hay que mantener con ellos. Los dígitos (entre '`1`' y '`9`') indican distancias entre 1 y 9. Las minúsculas (entre '`a`' y '`z`') indican distancias entre 10 y 35. Las mayúsculas (entre '`A`' y '`Z`') indican distancias entre 36 y 61. La posición inicial se indica con '`*`', y la posición final con '`+`'. Siempre hay exactamente uno de cada, y en posiciones no amenazadas por ningún monstruo.

Salida

Para cada caso, escribid "`SI`" o "`NO`" dependiendo de si es posible o no llegar a la posición final desde la posición inicial.

Puntuación

- **Test1:** **5 Puntos**

  Resolver casos con $n = 1$, como los del ejemplo 1.

- **Test2:** **15 Puntos**

  Resolver casos donde todas las distancias de seguridad son 1, como los del ejemplo 2.

- **Test3:** **30 Puntos**

  Resolver casos donde todas las distancias de seguridad están entre 1 y 4, como los del ejemplo 3.

- **Test4:** **50 Puntos**

  Resolver casos de todo tipo, como los del ejemplo 4.

Ejemplo de entrada 1

```
1 10
....1+*...

1 10
+..3..*...
```

Ejemplo de salida 1

```
SI
NO
```

Ejemplo de entrada 2

```
3 4
...+
....
*...

3 4
1..+
.1..
*.1.
```

Ejemplo de salida 2

```
SI
NO
```

Ejemplo de entrada 3

```
4 5
*..3.
.....
.....
2...+

4 5
*..3.
.....
.....
3...+

5 12
..4........2
.....4.4....
+...4.4....*
......3.....
.3........3.

2 7
+.2.3.1
1*.....
```

Ejemplo de salida 3

```
SI
NO
NO
NO
```

Solución

```
#include <bits/stdc++.h>

using namespace std;

const pair<int, int> INV = {-1, -1};
vector<vector<pair<int, int>>> UF;

pair<int, int> F(pair<int, int> p){
    return UF[p.first][p.second] == INV ? p : UF[p.first][p.second] =
F(UF[p.first][p.second]);
}

void U(pair<int, int> p1, pair<int, int> p2){
    p1 = F(p1);
    p2 = F(p2);
    if (p1 != p2)
        UF[p1.first][p1.second] = p2;
}

int main(){
    int n, m;
    while (cin >> n >> m){
        vector<vector<int>> v(n, vector<int>(m));
        UF = vector<vector<pair<int, int>>>(n, vector<pair<int, int>>(m, INV));
        int x1 = 0, y1 = 0, x2 = 0, y2 = 0;
        vector<queue<pair<int, int>>> vQ(62);
        for (int i = 0; i < n; i++){
               for (int j = 0; j < m; j++){
                      char c;
                      cin >> c;
                      if (c == '.' || c == '*' || c == '+') v[i][j] = 0;
                      else if (c >= '1' and c <= '9') v[i][j] = c - '1' + 1;
                      else if (c >= 'a' and c <= 'z') v[i][j] = c - 'a' + 10;
                      else v[i][j] = c - 'A' + 36;
                      if (c == '*'){
                             x1 = i;
                             y1 = j;
                      }
                      else if (c == '+'){
                             x2 = i;
                             y2 = j;
                      }
                      if (v[i][j] > 1)
                             vQ[v[i][j]].push({i, j});
               }
        }
        int danger = 61;
        while (danger > 1){
               queue<pair<int, int>>& Q = vQ[danger];
               while (!Q.empty()){
                      pair<int, int> it = Q.front();
```

```
                        Q.pop();
                        int& x = it.first;
                        int& y = it.second;
                        if (danger < v[x][y])
                                continue;
                        if (x > 0 and v[x-1][y] < danger-1){
                                v[x-1][y] = danger-1;
                                if (danger > 2)
                                        vQ[danger-1].push({x-1, y});
                        }
                        if (y > 0 and v[x][y-1] < danger-1){
                                v[x][y-1] = danger-1;
                                if (danger > 2)
                                        vQ[danger-1].push({x, y-1});
                        }
                        if (x < n-1 and v[x+1][y] < danger-1){
                                v[x+1][y] = danger-1;
                                if (danger > 2)
                                        vQ[danger-1].push({x+1, y});
                        }
                        if (y < m-1 and v[x][y+1] < danger-1){
                                v[x][y+1] = danger-1;
                                if (danger > 2)
                                        vQ[danger-1].push({x, y+1});
                        }
                }
                danger--;
        }
        for (int i = 0; i < n; i++){
                for (int j = 0; j < m; j++){
                        if (v[i][j] > 0)
                                continue;
                        if (i > 0 and v[i-1][j] == 0)
                                U({i, j}, {i-1, j});
                        if (j > 0 and v[i][j*1] == 0)
                                U({i, j}, {i, j-1});
                }
        }
        cout << (F({x1, y1}) == F({x2, y2}) ? "SI\n" : "NO\n");
    }
}
```

// Autor de la solución: Javier Nistal (Miembro del comité organizador)

Cartel bilingüe (2)

Como sabréis, en Kazajistán se hablan dos lenguas: kazajo y ruso. Por eso típicamente los carteles están en ambos idiomas: primero en kazajo y a continuación en ruso. Además, a veces ponen todas las palabras juntas, una detrás de otra, sin ni siquiera separar entre los dos idiomas. Vuestra tarea es intentar descifrar uno de esos carteles.

Disponéis de un diccionario de kazajo y un diccionario de ruso, el primero con k palabras y el segundo con r palabras. (Aunque ambos idiomas se escriben en cirílico, nosotros usaremos una transliteración al alfabeto latino.) Como el kazajo y el ruso son lenguas totalmente diferentes, es posible que la traducción de uno a otro no tenga nada que ver, ni en número de palabras ni en longitud. Además, podría ser que el cartel estuviera en un solo idioma, ya sea kazajo o ruso. ¿De cuantas maneras posibles se podría descifrar el cartel?

Entrada

La entrada empieza con k, seguido de las k palabras kazajas, seguido de r, seguido de las r palabras rusas. Al final viene un cartel de longitud c. Ambos diccionarios tienen entre 1 y 20000 palabras. Tanto las palabras como el cartel contienen sólo letras minúsculas. La longitud de las palabras está entre 1 y 200, y c está entre 1 y 20000. No hay palabras repetidas dentro del mismo diccionario.

Salida

Escribid el número de formas de descifrar el cartel, módulo $10^9 + 7$.

Puntuación

- **Test1:** **50 Puntos**

  Resolver entradas donde k y r son como mucho 1000, y c es como mucho 10.

- **Test2:** **20 Puntos**

  Resolver entradas donde c es como mucho 10.

- **Test3:** **30 Puntos**

  Resolver entradas de todo tipo.

Ejemplo de entrada 1

```
2
a
aa
2
b
bb
aaaabbb
```

Ejemplo de salida 1

```
15
```

Ejemplo de entrada 2

```
1
a
2
a
aa
aa
```

Ejemplo de salida 2

```
4
```

Ejemplo de entrada 3

```
4
a
aa
aaa
aaaa
4
a
aa
aaa
aaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Ejemplo de salida 3

```
674769970
```

Ejemplo de entrada 4

```
2
abc
a
1
cd
abcd
```

Ejemplo de salida 4

```
0
```

Solución

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const ll MOD = 1e9 + 7;

struct node{
    string s = "";
    bool isEnd = true;
    map<char, node*> child;
    node(string r){
        s = r;
    }

    node* add(string& s1, int i) {
        for(int j = 0; j < s.size() and i+j < s1.size(); ++j) {
               if(s[j] != s1[i+j]) {
                      node* nuevo1 = new node(s.substr(0, j));
                      node* nuevo2 = new node(s1.substr(i+j+1));
                      nuevo1->isEnd = false;
                      nuevo2->isEnd = true;
                      nuevo1->child[s1[i+j]] = nuevo2;
                      nuevo1->child[s[j]] = this;
                      s = s.substr(j+1);
                      return nuevo1;
               }
        }

        if(s1.size() - i == s.size()){
               this->isEnd = true;
               return this;
        } else if(s1.size() - i < s.size()){
               node* nuevo = new node(s.substr(0, s1.size() - i));
               nuevo->isEnd = true;
               nuevo->child[s[s1.size() - i]] = this;
               s = s.substr(s1.size() - i+1);
               return nuevo;
        }

        if(child[s1[i+s.size()]]) {
               child[s1[i+s.size()]] = child[s1[i+s.size()]]->add(s1, i+s.size()+1);
        } else {
               child[s1[i+s.size()]] = new node(s1.substr(i+s.size()+1));
        }
        return this;
    }
};

typedef node* trie;
```

```
vector<ll> traducir(string &s, trie root){
    int n = s.size();
    vector<ll> dp(n+1, 0);
    dp[0] = 1;
    int last = 0;
    for (int i = 0; i <= last; i++){
        if (dp[i] == 0)
                continue;
        trie act = root;
        int idx = i;
        while (act and idx <= n){
                bool works = act->s.size() <= s.size() - idx;
                for (int j = 0; j < act->s.size() and works; j++){
                        if (act->s[j] != s[j+idx])
                                works = false;
                }
                if (!works)
                        break;
                idx += act->s.size();
                if (act->isEnd){
                        dp[idx] = (dp[idx] + dp[i])%MOD;
                        last = max(last, idx);
                }
                if (idx < n)
                        act = act->child[s[idx]];
                idx++;
        }
    }
    return dp;
}

int main(){
    int k, r;
    cin >> k;
    trie rootK = new node("");
    rootK->isEnd = false;
    while (k--){
        string s;
        cin >> s;
        rootK->add(s, 0);
    }

    trie rootR = new node("");
    rootR->isEnd = false;
    cin >> r;
    while (r--){
        string s;
        cin >> s;
        reverse(s.begin(), s.end());
        rootR->add(s, 0);
    }

    string s;
```

```
    cin >> s;
    int n = s.size();
    vector<ll> tradK = traducir(s, rootK);
    reverse(s.begin(), s.end());
    vector<ll> tradR = traducir(s, rootR);
    ll ans = 0;
    for (int i = 0; i <= n; i++){
        ans = (ans + tradK[i]*tradR[n-i])%MOD;
    }
    cout << ans << '\n';
}
```

// Autor de la solución: Javier Nistal (Miembro del comité organizador)

Pintando poliedros

Pedro tiene un *poliedro convexo de caras triangulares* y quiere pintar sus caras sin que dos caras adyacentes tengan el mismo color. Dispuesto a comprar pintura, se pregunta cuál es el menor número de colores necesarios. Tras fracasar buscando la respuesta, su amigo, un turista bielorruso, le dice lo siguiente:

– "Te daré una pista para que lo puedas pintar. Si pones un vértice en cada cara y unes los vértices que estén en caras adyacentes, obtienes un grafo. Tu problema es equivalente a colorear ese grafo con el menor número de colores. Y por si no te has dado cuenta, por venir de un poliedro, el grafo será siempre *cúbico*, *planar* y *tres-conexo*."

Pedro sigue sin tener ni idea, así que os ha dejado el problema a vosotros. Recordad que un grafo cúbico es aquel en que todo vértice tiene exactamente tres vecinos, un grafo planar es aquel que se puede pintar en un papel sin que dos aristas se crucen, y un grafo tres-conexo es un grafo conexo al que se le han de quitar por lo menos tres vértices para desconectarlo.

Entrada

La entrada tiene diversos grafos, todos provinientes de un poliedro como se ha descrito anteriormente. Cada grafo empieza con su número de vértices n, seguido de n triplas con los vecinos de cada vértice i. Al final viene la palabra `CUENTA` o la palabra `PINTA`. Podéis suponer $4 \leq n \leq 20000$. Los vértices se numeran desde cero.

Salida

Para cada caso de tipo `CUENTA`, escribid el mínimo número de colores para pintar el grafo. Para los de tipo `PINTA`, a continuación escribid los colores de los vértices en orden. Seguid estrictamente el formato de los ejemplos.

Pista

Es conocido que todo grafo planar se puede pintar con cuatro colores o menos.

Puntuación

- **Test1:** Resolver casos de tipo `CUENTA` donde $n \leq 10$. **10 Puntos**
- **Test2:** Resolver casos de tipo `CUENTA`. **30 Puntos**
- **Test3:** Resolver casos de tipo `PINTA` donde $n \leq 10$. **10 Puntos**
- **Test4:** Resolver casos de tipo `PINTA`. **50 Puntos**

Ejemplo de entrada 1

```
4
1 2 3
0 2 3
0 1 3
0 1 2
CUENTA

6
1 2 4
0 2 5
0 1 3
2 4 5
0 3 5
1 3 4
CUENTA

8
1 2 6
0 3 7
0 3 4
1 2 5
2 5 6
3 4 7
0 4 7
1 5 6
CUENTA
```

Ejemplo de salida 1

```
4
3
2
```

Ejemplo de entrada 2

```
4
1 2 3
0 2 3
0 1 3
0 1 2
PINTA

6
1 2 4
0 2 5
0 1 3
2 4 5
0 3 5
1 3 4
PINTA

8
1 2 6
0 3 7
0 3 4
1 2 5
2 5 6
3 4 7
0 4 7
1 5 6
PINTA
```

Ejemplo de salida 2

```
4
1234
3
312312
2
12211221
```

Solución