Editorial Codemania 2023.

Mauricio Tapia

February 14, 2024

Development and Algorithmic Hub
DAH

Contents

0.-Consideraciones

- Durante la competencia se detecto un comportamiento extraño en la solución usada para generar casos de prueba del problema G, después de analizarla se ha decidido que la solución y por consecuencia los casos de prueba generados con ella son correctos, por lo que se recomienda que se intente resolverlo.
- Se asumirá que el lector sabe analizar la complejidad temporal de algoritmos, sin embargo si no se domina esto no sera impedimento para la comprensión de este texto.
- Algunos problemas son mas fáciles de resolver sí se tiene conocimiento previo de ciertos algoritmos, cuando sea el caso, sus nombres serán escritos al revés en la sección de hints.
- En el comprimido donde se encuentra esta editorial se encontraran las soluciones que se usaron para generar los casos de prueba, los casos de prueba y las soluciones descriptivas usadas en este documento.
- Para upsolvear los problemas (resolver después del concurso) recomendamos que se una al siguiente grupo privado. https://codeforces.com/group/r9FP7r2CAt/
- Si deseas representar a tu universidad/preparatoria en concursos nacionales contáctanos en https://www.facebook.com/people/DAH/100089564896614/

A.- An overly elaborate trap.

Hints.

- ¿Como se puede simular el derrame de la lava con un solo bloque fuente? Flood Fill
- ¿De las diferentes maneras de simularlo, cual es la adecuada si deseamos conocer el segundo en que el bloque fue tocado por primera vez por la lava? Flood Fill con BFS
- ¿Es posible simular el flujo de todos los bloques de lava a la vez?

Solución.

Simulación de un solo bloque de lava.

Dado un bloque de lava con ubicación en (r, c), es posible simular su flujo colocando sus vecinos inmediatos en una lista, extrayendolos uno por uno y colocando a su vez sus vecinos que aun no han estado en lista, de esta forma hasta que la lista quede vaciá. Ahora, el orden en el que la lista es accedida es importante pues se puede demostrar que, si extraemos primero los elementos que entraron primero a la lista, la cantidad de pasos necesarios para llegar a cada celda es la mínima. Esto se puede lograr de varias formas (priority queue, multiset, list, queue) pero considero que la forma mas limpia de hacerlo es usando una fila, pues dado su comportamiento los elementos que entran primero salen primero.

```
//           filas,columnas (r,c)
int tiempo[1000][1000];
bool visitado[1000][1000];
int filas, columnas;
bool esValido(int f, int c){
    if(f<0) return false;
    if(f>=filas) return false;
    if(c<0) return false;
    if(c>=columnas) return false;
    if(visitado[f][c]) return false;
    //si llego aqui entonces si es valido
    return true;
}
void simular(pair<int,int> padre){
    for(int f = 0; f<filas; f++){
        for(int c = 0; c<columnas; c++){
            visitado[f][c] = false;
            //como la lava fluye a lo sumo por 3 segundos, si una celda se queda
   con 4 significa que nunca es alcanzada
            tiempo[f][c] = 4;
        }
    }

    queue<pair<int,int>> porProcesar;
    pair<int,int> actual;
    visitado[padre.first][padre.second] = 1;
    //se tardan cero segundos en llegar a la celda padre
    tiempo[padre.first][padre.second] = 0;
    porProcesar.push(padre);

    //mientras queden celdas por procesar
    while(porProcesar.size()>0){
        //toma la celda mas reciente que aun este pendiente de procesar.
        actual = porProcesar.front();
```

```
34          porProcesar.pop();
35          //si el tiempo que tardo la lava en llegar a esta celda es 3, ya no se
    puede expandir mas por lo que se puede omitir esta celda
36          if(tiempo[actual.first][actual.second]==3) continue;
37          //---------------IZQUIERDA------------
38          //si tiene una celda izquierda y aun no ha sido visitada, introducirla a la
     lista de celdas por procesar
39          //y actualizar su tiempo de introduccion
40          if(esValido(actual.first,actual.second-1)){
41              porProcesar.push(make_pair(actual.first,actual.second-1));
42              visitado[actual.first][actual.second-1] = true;
43              tiempo[actual.first][actual.second-1] = tiempo[actual.first][actual.
    second]+1;
44          }
45          //---------------DERECHA--------------
46          //si tiene una celda derecha y aun no ha sido visitada, introducirla a la
    lista de celdas por procesar
47          //y actualizar su tiempo de introduccion
48          if(esValido(actual.first,actual.second+1)){
49              porProcesar.push(make_pair(actual.first,actual.second+1));
50              visitado[actual.first][actual.second+1] = true;
51              tiempo[actual.first][actual.second+1] = tiempo[actual.first][actual.
    second]+1;
52          }
53          //---------------ARRIBA---------------
54          //si tiene una celda arriba y aun no ha sido visitada, introducirla a la
    lista de celdas por procesar
55          //y actualizar su tiempo de introduccion
56          if(esValido(actual.first-1,actual.second)){
57              porProcesar.push(make_pair(actual.first-1,actual.second));
58              visitado[actual.first-1][actual.second] = true;
59              tiempo[actual.first-1][actual.second] = tiempo[actual.first][actual.
    second]+1;
60          }
61          //---------------ABAJO---------------
62          //si tiene una celda abajo y aun no ha sido visitada, introducirla a la
    lista de celdas por procesar
63          //y actualizar su tiempo de introduccion
64          if(esValido(actual.first+1,actual.second)){
65              porProcesar.push(make_pair(actual.first+1,actual.second));
66              visitado[actual.first+1][actual.second] = true;
67              tiempo[actual.first+1][actual.second] = tiempo[actual.first][actual.
    second]+1;
68          }
69      }
70
71  }
```

Simulación con varios bloques de lava.

Existen dos formas de simular el flujo simultaneo de dos o mas bloques de lava:

- Simular uno por uno los bloques de lava a colocar utilizando un $map<int,map<int,bool> >$ declarado localmente para registrar las celdas visitadas, este enfoque por su cuenta demuestra ser demasiado lento para los limites del problema (sin optimizaciones tarda hasta 3 segundos en terminar los casos de prueba mas exigentes), pero sí, durante la simulación se evita procesar aquellas celdas cuyo tiempo registrado sea menor o igual al tiempo que se tardo en ser alcanzada en la

simulación actual, este enfoque corre dentro del tiempo limite (≈500ms). La implementación de este enfoque no sera discutida en este documento, pero puede ver su implementación en la carpeta *codigosDesarrolloProblemario*.

- Agregando una celda imaginaria adyacente **SOLO** a cada una de las celdas padre con un tiempo de llegada de -1 y simular como si se tratase de un solo bloque de lava. Es fácil ver como el funcionamiento de la simulación de un solo bloque de lava se traduce bien a este caso.

Codigo AC(AC.cpp)

```cpp
#include "bits/stdc++.h"
using namespace std;
#define endl '\n'

//           filas,columnas (r,c)
int tiempo[1000][1000];
bool visitado[1000][1000];
int filas, columnas;
bool esValido(int f, int c){
    if(f<0) return false;
    if(f>=filas) return false;
    if(c<0) return false;
    if(c>=columnas) return false;
    if(visitado[f][c]) return false;
    //si llego aqui entonces si es valido
    return true;
}
void simular(vector<pair<int,int>> padres){
    for(int f = 0; f<filas; f++){
        for(int c = 0; c<columnas; c++){
            visitado[f][c] = false;
            //como la lava fluye a lo sumo por 3 segundos, si una celda se queda con 4 significa que nunca es alcanzada
            tiempo[f][c] = 4;
        }
    }

    queue<pair<int,int>> porProcesar;
    pair<int,int> actual;

    for(pair<int,int> padre: padres){
        visitado[padre.first][padre.second] = 1;
        //se tardan cero segundos en llegar a cada celda padre
        tiempo[padre.first][padre.second] = 0;
        porProcesar.push(padre);
    }


    //mientras queden celdas por procesar
    while(porProcesar.size()>0){
        //toma la celda mas reciente que aun este pendiente de procesar.
        actual = porProcesar.front();
        porProcesar.pop();
        //si el tiempo que tardo la lava en llegar a esta celda es 3, ya no se puede expandir mas por lo que se puede omitir esta celda
        if(tiempo[actual.first][actual.second]==3) continue;
        //---------------IZQUIERDA------------
```

```
46          //si tiene una celda izquierda y aun no ha sido visitada, introducirla a la
    lista de celdas por procesar
47          //y actualizar su tiempo de introduccion
48          if(esValido(actual.first,actual.second-1)){
49              porProcesar.push(make_pair(actual.first,actual.second-1));
50              visitado[actual.first][actual.second-1] = true;
51              tiempo[actual.first][actual.second-1] = tiempo[actual.first][actual.
    second]+1;
52          }
53          //---------------DERECHA--------------
54          //si tiene una celda derecha y aun no ha sido visitada, introducirla a la
    lista de celdas por procesar
55          //y actualizar su tiempo de introduccion
56          if(esValido(actual.first,actual.second+1)){
57              porProcesar.push(make_pair(actual.first,actual.second+1));
58              visitado[actual.first][actual.second+1] = true;
59              tiempo[actual.first][actual.second+1] = tiempo[actual.first][actual.
    second]+1;
60          }
61          //---------------ARRIBA---------------
62          //si tiene una celda arriba y aun no ha sido visitada, introducirla a la
    lista de celdas por procesar
63          //y actualizar su tiempo de introduccion
64          if(esValido(actual.first-1,actual.second)){
65              porProcesar.push(make_pair(actual.first-1,actual.second));
66              visitado[actual.first-1][actual.second] = true;
67              tiempo[actual.first-1][actual.second] = tiempo[actual.first][actual.
    second]+1;
68          }
69          //---------------ABAJO---------------
70          //si tiene una celda abajo y aun no ha sido visitada, introducirla a la
    lista de celdas por procesar
71          //y actualizar su tiempo de introduccion
72          if(esValido(actual.first+1,actual.second)){
73              porProcesar.push(make_pair(actual.first+1,actual.second));
74              visitado[actual.first+1][actual.second] = true;
75              tiempo[actual.first+1][actual.second] = tiempo[actual.first][actual.
    second]+1;
76          }
77      }
78
79  }
80
81  bool comp(const pair<int,int> a, const pair<int,int> b){
82      if(a.first == b.first){
83          return a.second<b.second;
84      }
85      return a.first > b.first;
86  }
87  int main(){
88      ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
89      cin>>filas>>columnas;
90      vector<pair<int,int>> padres;
91      pair<int,int> actual;
92      int bloquesLava;
93      cin>>bloquesLava;
94      while(bloquesLava-->0){
95          cin>>actual.first>>actual.second;
96          padres.push_back(actual);
```

```
    }
    simular(padres);

    int mejorTiempo = 0;

    for(int f = 0; f<filas; f++){
        for(int c = 0; c<columnas; c++){
            mejorTiempo = max(mejorTiempo,tiempo[f][c]);
        }
    }

    if(mejorTiempo==0){
        cout<<-1<<endl;
        return 0;
    }
    vector<pair<int,int>> res;

    for(int f = 0; f<filas; f++){
        for(int c = 0; c<columnas; c++){
            if(tiempo[f][c] == mejorTiempo){
                res.push_back(make_pair(f,c));
            }
        }
    }


    sort(res.begin(),res.end(),comp);
    cout<<res[0].first<<" "<<res[0].second<<endl;
    return 0;
}
```

B.- Brainfuck.

Hints.

- La correspondencia entero → carácter es igual al del código ascii, ¿existe alguna forma de hacer la conversión con algún método nativo del lenguaje?
- El tamaño máximo del arreglo global es 10^9, ¿El limite de memoria nos permitirá crear un arreglo tan grande?
- En caso de no poder crear el arreglo directamente, ¿Que alternativas podrían existir?

Solución.

Implementación del interprete

Por simplicidad, supongamos que para nuestra implementación un arreglo del tamaño necesario fue declarado, en la siguiente sección se darán detalles de como resolver la anotación hecha en el hint #2. La implementación del interprete muestra ser una tarea relativamente sencilla dada la naturaleza secuencial de esta versión de brainfuck, pues basta con iterar cada carácter de la cadena que indica el programa llevando solamente registro del numero de celda en la que esta el puntero. **Nota:** Los arreglos tienen indices de 0 a $longitud - 1$.

```
//supongamos que la lectura de programa y longitud fueron hechas.
//tambien que fue creado un arreglo global arr con todas sus posiciones
   inicializadas con 32
string programa;
int longitud;
//el puntero debe iniciar en la primera celda de memoria
int puntero = 0;
for(char instruccion: programa){
    //mueve el puntero a la izquierda.
    if(instruccion == '<'){
        puntero--;
        //si el puntero es menor a cero
        //significa que se movio a la izquierda desde la posicion cero
        if(puntero<0){
            puntero = longitud -1;
        }
    }
    //mueve el puntero a la derecha.
    if(instruccion == '>'){
        puntero++;
        //si el puntero es mayor o igual a longitud
        //significa que se movio a la derecha desde la posicion mas derecha posible
        if(puntero>= longitud){
            puntero = 0;
        }
    }
    //Aumenta el valor de la casilla.
    if(instruccion == '+'){
        arr[puntero]++;
        //si es mayor al valor maximo permitido, regresar a 32
        if(arr[puntero] > 126){
            arr[puntero] = 32;
        }
    }
```

```
    //Disminuye el valor de la casilla.
    if(instruccion == '-'){
        arr[puntero]--;
        //si es menor al valor minimo permitido, regresar a 126
        if(arr[puntero] < 32){
            arr[puntero] = 126;
        }
    }
    //Imprime el valor debajo del puntero.
    if(instruccion == '.'){
        //Como la tabla del problema coincide con la tabla ascii
        //podemos hacer la conversion directamente con un casteo de la forma
        cout<< (char)arr[puntero];
    }
}
```

Creación/alternativas al arreglo global.

Supongamos que creamos un arreglo global con la longitud que cada caso de prueba nos pide, en el peor de los casos el arreglo puede ser de hasta 10^9 posiciones, asumiendo que usamos el tipo de dato mas compacto *char* ocupando un byte por posicion en el arreglo, su tamaño sería de 953 MiB, lo cual excede el limite de memoria. Existen dos formas de superar esta limitante:

- Utilizando una estructura de datos asociativa (i.e. Maps, HashTables, Dictionaries, etc.) donde solo las posiciones visitadas por brainfuck son creadas, si este enfoque es elegido se debe tener cuidado con el valor por defecto con el cual se inicializa cada posición, pues en C++ suele ser 0, por lo cual si una celda es visitada por primera vez su valor deberá ser cambiado a 32.
- Igualar a *longitud* con el mínimo entre *longitud* y 10^5 para después crear un arreglo global con este nuevo valor, esto funciona debido a que el programa es de a lo sumo 10^5 instrucciones, por lo que es imposible que el puntero interactué con mas de 10^5 posiciones del arreglo global.

Las dos alternativas anteriormente mencionadas usan 10^5 posiciones de memoria, suponiendo que usamos *int* como tipo de dato, ambas soluciones usan menos de 1 MiB de memoria.

Codigo usando map. (bfMap.cpp)

```
#include "bits/stdc++.h"

using namespace std;
#define endl '\n'

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    //arreglo global
    map<int,int> arr;
    string programa;
    int longitud;
    cin>>programa;
    cin>>longitud;
    //el puntero debe iniciar en la primera celda de memoria
    int puntero = 0;
    //la primera posicion es la primera en ser visitada, por lo que se inicializa
   en 32
    arr[puntero] = 32;
```

```
    for(char instruccion: programa){
        //mueve el puntero a la izquierda.
        if(instruccion == '<'){
            puntero--;
            //si el puntero es menor a cero
            //significa que se movio a la izquierda desde la posicion cero
            if(puntero<0){
                puntero = longitud -1;
            }
            //si su valor es cero, significa que es la primera vez que la celda es visitada, inicializar en 32
            if(arr[puntero]==0){
                arr[puntero] = 32;
            }
        }
        //mueve el puntero a la derecha.
        if(instruccion == '>'){
            puntero++;
            //si el puntero es mayor o igual a longitud
            //significa que se movio a la derecha desde la posicion mas derecha posible
            if(puntero>= longitud){
                puntero = 0;
            }

            //si su valor es cero, significa que es la primera vez que la celda es visitada, inicializar en 32
            if(arr[puntero]==0){
                arr[puntero] = 32;
            }
        }
        //Aumenta el valor de la casilla.
        if(instruccion == '+'){
            arr[puntero]++;
            //si es mayor al valor maximo permitido, regresar a 32
            if(arr[puntero] > 126){
                arr[puntero] = 32;
            }
        }
        //Disminuye el valor de la casilla.
        if(instruccion == '-'){
            arr[puntero]--;
            //si es menor al valor minimo permitido, regresar a 126
            if(arr[puntero] < 32){
                arr[puntero] = 126;
            }
        }
        //Imprime el valor debajo del puntero.
        if(instruccion == '.'){
            //Como la tabla del problema coincide con la tabla ascii
            //podemos hacer la conversion directamente con un casteo de la forma
            cout<< (char)arr[puntero];
        }
    }
}
```

Codigo usando min (bfMin.cpp)

```
#include "bits/stdc++.h"
```

```cpp

using namespace std;
#define endl '\n'

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    //arreglo global
    int *arr;
    string programa;
    int longitud;
    cin>>programa;
    cin>>longitud;
    longitud = min(longitud,100000);

    //se crea e inicializa el arreglo global
    arr = new int[longitud];
    for(int i = 0; i<longitud; i++){
        arr[i] = 32;
    }

    //el puntero debe iniciar en la primera celda de memoria
    int puntero = 0;
    for(char instruccion: programa){
        //mueve el puntero a la izquierda.
        if(instruccion == '<'){
            puntero--;
            //si el puntero es menor a cero
            //significa que se movio a la izquierda desde la posicion cero
            if(puntero<0){
                puntero = longitud -1;
            }
        }
        //mueve el puntero a la derecha.
        if(instruccion == '>'){
            puntero++;
            //si el puntero es mayor o igual a longitud
            //significa que se movio a la derecha desde la posicion mas derecha
   posible
            if(puntero>= longitud){
                puntero = 0;
            }
        }
        //Aumenta el valor de la casilla.
        if(instruccion == '+'){
            arr[puntero]++;
            //si es mayor al valor maximo permitido, regresar a 32
            if(arr[puntero] > 126){
                arr[puntero] = 32;
            }
        }
        //Disminuye el valor de la casilla.
        if(instruccion == '-'){
            arr[puntero]--;
            //si es menor al valor minimo permitido, regresar a 126
            if(arr[puntero] < 32){
                arr[puntero] = 126;
            }
        }
        //Imprime el valor debajo del puntero.
```

```
60          if(instruccion == '.'){
61              //Como la tabla del problema coincide con la tabla ascii
62              //podemos hacer la conversion directamente con un casteo de la forma
63              cout<< (char)arr[puntero];
64          }
65      }
66  }
```

C.- Cost of Doing Business.

Hints.

- ¿El mínimo costo de llegada a un país p_1 afecta el costo para llegar a otro país p_2 ($p_1 < p_2$) si están conectados por un tratado?
- Dado un tratado que va del país p_1 al país p_2, ¿Se puede llegar a p_2 con un costo mínimo a través de algún otro país entre p_1 y p_2?
- ¿Como construirías la respuesta usando funciones recursivas sí no te importa la eficiencia?
- ¿Si a la función recursiva le agregar un arreglo que almacena el mínimo costo de llegar a cada país, se haría mas eficiente?
- ¿El orden en que calculamos la respuesta para cada país afecta?
- Construye la respuesta del país cero al país M
- ¿Realmente es necesario calcular la respuesta para todos los países del planeta? ¿Que cambiaría si solo la calculamos para los países en el extremo derecho de un tratado?
- Sí un tratado va mas allá de M, ¿cambiara la respuesta si cambiamos su extremo derecho a M?
- ¿DP + Range queries?

Solución.

Este fue uno de los problemas mas difíciles de codemania, pues no solo requería encontrar una solución de programación dinámica, si no también implementarla usando una estructura de datos especializada en range queries (ie. segment tree, fenwick tree, sparse table, etc.) para lograr una respuesta en $O((M \log M) + (T \log M))$. Para llegar a la solución utilizando programación dinámica, es necesario partir de una solución recursiva (e ineficiente) como la siguiente:

```
struct tratado{
    int ini;
    int fin;
    long long int costo;
};
//se iguala a un costo que no se puede obtener
long long int inf = 1000000000000000LL;
vector<tratado> tratados;
long long int solve(int paisActual){
    //llegar hasta el pais cero es gratis, pues
    if(paisActual == 0) return 0;
    long long int respuesta = inf;
    for(tratado actual: tratados){
        //si el tratado no afecta al pais actual, continua con el siguiente
        if(actual.ini> paisActual) continue;
        if(actual.fin< paisActual) continue;
        //por cada pais en el tratado menor al pais actual
        //igualar la respuesta al minimo costo de cada pais mas el costo del
   tratado que nos ayudo a acceder a dicho pais
        for(int p = actual.ini; p<paisActual; p++){
            respuesta = min(respuesta, solve(p)+actual.costo)
        }
    }
```

```
    return respuesta;
}
```

Existen dos optimizaciones importantes que se pueden hacer a este código:

- Memoización, para evitar que la función *solve* con un argumento en particular se evalué varias veces, es necesario almacenar su valor en una tabla de memoización una vez calculado por primera vez, esto es relativamente sencillo pues basta con inicializar la tabla de memoización con el valor inobtenible de infinito(inf) y usarlo como una bandera que indica si la función en cierto valor ya fue calculada.

- Calcular *solve* solamente para los extremos derechos de cada tratado, es posible notar que dos países que comparten el mismo conjunto de tratados tienen el mismo costo de llegada, por lo que calcular el costo minimo de llegada de todos los países es redundante, en cambio resulta optimo solo calcular este valor en el ultimo pais de cada tratado.

Implementados estos cambios, la función recursiva(aun ineficiente) luce de la siguiente forma:

```
struct tratado{
    int ini;
    int fin;
    long long int costo;
};
//se iguala a un costo que no se puede obtener
long long int inf = 10000000000000000LL;
//todo inicializado en inf
long long int dp[100001];
vector<tratado> tratados;
long long int solve(int paisActual){
    //llegar hasta el pais cero es gratis, pues
    if(paisActual == 0) return 0;
    //si esta instancia de la funcion ya fue calculada, retorna su resultado
    if(dp[paisActual]!= inf) return dp[paisActual];
    long long int respuesta = inf;
    for(tratado actual: tratados){
        //si el paisActual no es el fin del tratado actual, no hagas nada
        if(actual.fin != paisActual) continue;
        //por cada pais en el tratado menor al pais actual
        //igualar la respuesta al minimo costo de cada pais mas el costo del
   tratado que nos ayudo a acceder a dicho pais
        for(int p = actual.ini; p<paisActual; p++){
            respuesta = min(respuesta, solve(p)+actual.costo)
        }
    }
    dp[paisActual] = respuesta;
    return respuesta;
}
```

Existen dos fuentes de complejidad en esta función recursiva, el ciclo que verifica cuantos tratados tienen como valor derecho a *paisActual* y el ciclo buscando entre el extremo izquierdo y el *paisActual* el país con costo de llegada mas bajo. Para corregir esto, solo basta con solo iterar por los extremos derechos de cada tratado de menor a mayor país inicial, y para buscar el mínimo en el rango que el tratado cubre, usar una estructura de datos para hacer range queries en $\log M$

```
#include "bits/stdc++.h"
//assert(x>0) si falla da RTE
using namespace std;
#define endl '\n'
```

```
#define DBG(x) cerr<<#x<< "=" << (x) << endl;
#define RAYA cerr<<"======================"<<endl;
#define RAYAS cerr<<"......................"<<endl;
//#define DBG(x) ;
//#define RAYA ;
//#define RAYAS ;

// definicion del segment tree
long long int* arr;
template<class T> class SegmentTree {
private:
  unsigned long long int stHeight, maxNumLeaves;
  T* sTree;
  int origSize;
  void createSTree(int arrLen);
  T valorInex;
  std::function<T(T, T, int)> funcionLlenadora;
  //Nodo segTree, Index inicio , index final
  T fill(int STIndex, int startIndex, int endIndex);
  //Consulta de L a R, nodo STIndex del segtree, rango cubierto actual de
    startindex a endindex
  T search(int L, int R, int STIndex, int startIndex, int endIndex);
  //valor,indice, nodo del segtree, rango cubierto actual de startindex a endindex
  void updateValue(T value, int valueInd, int STIndex, int startIndex, int endIndex
    );
public:
  //CONSTRUCTORES
  SegmentTree(int n, T inex, const std::function<T(T, T, int)>& llen);
  //PROCEDIMIENTOS
  T query(int L, int R);
  void update(T value, int valueInd);
  void print();
  void resizeSTree(int arrLen);
  //DESTRUCTOR
  ~SegmentTree();
};
template<class T> void SegmentTree<T>::createSTree(int arrLen) {
  //CALCULA LA ALTURA, CANTIDAD DE HOJAS Y EN BASE A ELLO CERA EL ARREGLO
  stHeight = ceil(log2(arrLen));
  maxNumLeaves = (2 * pow(2, stHeight)) - 1;
  sTree = new T[maxNumLeaves];
  //ALMACENA EL TAMANO ORIGINAL DEL ARREGLO
  origSize = arrLen;
}
template<class T> void SegmentTree<T>::resizeSTree(int arrLen) {
  //CALCULA LA ALTURA, CANTIDAD DE HOJAS Y EN BASE A ELLO CERA EL ARREGLO
  stHeight = ceil(log2(arrLen));
  maxNumLeaves = (2 * pow(2, stHeight)) - 1;
  //ALMACENA EL TAMANO ORIGINAL DEL ARREGLO
  origSize = arrLen;
  fill(0, 0, arrLen - 1);
}
template<class T> SegmentTree<T>::SegmentTree(int n, T inex, const std::function<T(
    T, T, int)>& llen) {
  funcionLlenadora = llen;
  valorInex = inex;
  //CONSTRUYE EL SEGMENT TREE
  createSTree(n);
  //RELLENA EL SEGMENT TREE
```

```
  fill(0, 0, n - 1);

}

template<class T> T SegmentTree<T>::fill(int STIndex, int startIndex, int endIndex)
    {
  //Si el nodo representa un solo elemento regresa el valor de ese elemento
  if (startIndex == endIndex) {
    sTree[STIndex] = arr[startIndex];
    return sTree[STIndex];
  }
  //Caso contrario busca en el hijo izquierdo y en el derecho
  int middleIndex = startIndex + ((endIndex - startIndex) / 2);
  int leftChildIndex = STIndex * 2 + 1;
  int rightChildIndex = STIndex * 2 + 2;
  T leftChild = fill(leftChildIndex, startIndex, middleIndex);
  T rightChild = fill(rightChildIndex, middleIndex + 1, endIndex);
  // para despues alamacenar el menor y devolverlo
  sTree[STIndex] = funcionLlenadora(leftChild, rightChild, log2(endIndex -
   startIndex + 1));
  return sTree[STIndex];
}

template<class T> T SegmentTree<T>::search(int L, int R, int STIndex, int
   startIndex, int endIndex) {
  //SI LA SECCION BUSCADA NO CORRESPONDE AL SEGMENTO REGRESAR EL MAXIMO NUMERO
   POSIBLE
  if (endIndex<L || startIndex>R) {
    return  valorInex;
  }
  //SI LOS INDICES DEL QUERY CUBREN TOTALMENTE AL SEGMENTO REGRESAR EL MINIMO DEL
   SEGMENTO
  if (L <= startIndex && endIndex <= R) {
    return sTree[STIndex];
  }
  //SI ESTA CUBIERTO PARCIALMENTE POR EL QUERY PREGUNTA A SUS HIJOS
  int middleIndex = startIndex + ((endIndex - startIndex) / 2);
  int leftChildIndex = STIndex * 2 + 1;
  int rightChildIndex = STIndex * 2 + 2;
  T leftChild = search(L, R, leftChildIndex, startIndex, middleIndex);
  T rightChild = search(L, R, rightChildIndex, middleIndex + 1, endIndex);
  //PARA DESPUES DEVOLVER EL MENOR DE SUS HIJOS
  return funcionLlenadora(leftChild, rightChild, log2(endIndex - startIndex + 1));
}
template<class T> T SegmentTree<T>::query(int L, int R) {
  //HACE LA PRIMERA LLAMADA RECURSIVA
  return search(L, R, 0, 0, origSize - 1);
}
template<class T> void SegmentTree<T>::updateValue(T value, int valueInd, int
   STIndex, int startIndex, int endIndex) {
  //SI EL NODO REPRESENTA UN SOLO ELEMENTO SE IGUALA EL VALOR AL NUEVO
  if (startIndex == endIndex) {
    sTree[STIndex] = value;
    return;
  }
  //CASO CONTRARIO BUSCARA EN SUS HIJOS
  int middleIndex = startIndex + ((endIndex - startIndex) / 2);
  int leftChildIndex = STIndex * 2 + 1;
  int rightChildIndex = STIndex * 2 + 2;
```

```
  //REVISA EN CUAL HIJO SE ENCUENTRA EL ELEMENTO PARA DESPUES ACTUALIZAR MEDIANTE RECURSION
  if (startIndex <= valueInd && valueInd <= middleIndex) {
    updateValue(value, valueInd, leftChildIndex, startIndex, middleIndex);
  }
  else {
    updateValue(value, valueInd, rightChildIndex, middleIndex + 1, endIndex);
  }
  //UNA VEZ ACTUALIZADO TOMA EL VALOR MENOR DE LOS DOS HIJOS
  T LeftChild = sTree[leftChildIndex];
  T RightChild = sTree[rightChildIndex];
  sTree[STIndex] = funcionLlenadora(LeftChild, RightChild, log2(endIndex - startIndex + 1));
}
template<class T> void SegmentTree<T>::update(T value, int valueInd) {
  //PRIMERA LLAMADA RECURSIVA
  updateValue(value, valueInd, 0, 0, origSize - 1);
}
template<class T> void SegmentTree<T>::print() {
  //RECORRE EL SEGTREE E IMPRIME CADA ELEMENTO SEPARADO POR UN ESPACIO
  for (T curr : sTree) {
    cout << curr << " ";
  }
  cout << endl;
}
//DESTRUCTOR
template<class T>  SegmentTree<T>::~SegmentTree() {
  delete[] sTree;
  sTree = nullptr;
  origSize = 0;
  stHeight = 0;
  maxNumLeaves = 0;
}



//lenadora T left, T right, int lvl
long long int bitllena(long long int left, long long int right, int lvl) {
  return min(left,right);
}

struct permiso{
  //pais izquierdo, pais derecho, costo
  long long int l,r,w;
};
bool comp( permiso a,permiso b){
    if(a.l == b.l){
        return a.r > b.r;
    }
    return a.l < b.l;
}
//--------------------SOLBEGIN---------------------------------
int main() {
  ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
    long long int  m,c;
    cin>>m;

    //arr representa el costo minimo para las distancias en el intervalo cerrado [0-m]
```

```
    arr = new long long int[m+1];
    //el costo para cubrir el envio al pais 0 es cero, pues jaime vive ahi
    arr[0] = 0;
    //se inicializa en +infinito
    for(int i = 1; i<=m; i++){
        arr[i] = 1000000000000000000LL;
    }

    permiso aux;
    vector<permiso> tratados;
    int q;
    cin>>q;
    while(q--){
        cin>>aux.l>>aux.r>>aux.w;
        //solo nos interesa si el tratado cubre hasta m
        aux.r = min(aux.r,m);
        tratados.push_back(aux);
    }

    sort(tratados.begin(),tratados.end(),comp);

  SegmentTree<long long int> st(m+1,10000000000000000LL,bitllena);

  for(permiso el: tratados){
    //busca el minimo costo en el rango que cubre el tratado, y le suma el costo del tratado pues fue necesario usarlo para acceder a los elementos del rango
    c = st.query(el.l,el.r)+el.w;
    //si el costo para llegar al extremo derecho del tratado actual es mayor al costo calculado en esta iteracion, actualizalo
    if(st.query(el.r,el.r)>c){
      st.update(c,el.r);
    }
  }

  //pregunta al segment tree el costo de llegar a m
  long long int res = st.query(m,m);

  //si cuesta infinito, significa que es imposible llegar, si es cualquier otra cosa ese es el costo de llegar
  cout<<(res==1000000000000000000LL?-1:res)<<endl;

}
//--------------------EOSOLUTION---------------------------------
```

D.- Damage Control.

Fe de erratas.

Mientras hacían upsolving de este problema, un equipo notó que el delta en algunas filas de la tabla mostrada como nota en el enunciado del problema estaba equivocado. La versión ya corregida fue subida tanto a OmegaUp como a Codeforces, y recomendamos que vuelvan a leer la nota:

A	B	Resulting Packages	Δ sum
$[3,8]$	$[8,2]$	$[(3,8),(8,2)]$	11
	$[8,9]$	$[(3,8),(8,9)]$	6
	$[2,9]$	$[(3,2),(8,9)]$	2
$[8,1]$	$[8,2]$	$[(8,8),(1,2)]$	1
	$[8,9]$	$[(8,8),(1,9)]$	8
	$[2,9]$	$[(8,2),(1,9)]$	14
$[3,1]$	$[8,2]$	$[(3,8),(1,2)]$	6
	$[8,9]$	$[(3,8),(1,9)]$	13
	$[2,9]$	$[(3,2),(1,9)]$	9

Hints.

- Observa los limites, ¿Es posible resolver el problema por fuerza bruta?
- Si la solución pensada fue por backtracking, ¿Crees que se puedan eliminar ramas recursivas con un k mayor al que recibiste? ¿Mejorara el tiempo de ejecución sí generas los subconjuntos de la segunda banda solamente cuando alcanzas un estado con k objetos en la primera?
- Si pensaste en una solución iterativa, ¿Existe una forma eficiente de generar y representar subconjuntos de forma iterativa?subset generation using bitmasking

Solución.

Este problema fue pensado para ser resuelto usando un enfoque de fuerza bruta en $O(\binom{n}{k}^2)$, pero es necesario notar que un tester descubrió un enfoque usando programación dinámica con complejidad $O(n^3)$ esta solución sera colocada en *codigosDesarrolloProblemario*. Sus soluciones por fuerza bruta son las siguientes.

Solución recursiva.

```
#include "bits/stdc++.h"
using namespace std;
#define endl '\n'

vector<int> a,b;
int n,k;
vector<int> nuevoA,nuevoB;
long long int respuesta;

```

```
10
11 void generaB(int pos){
12     //si ya no quedan items en B por considerar
13     if(pos==n){
14         //si en el subconjunto de B que tomamos hay exactamente 'k' elementos
15         //calculamos la respuesta para esta combinacion subconjunto A + subconjunto
     B
16     if(nuevoB.size() == k){
17             long long int actual = 0;
18             for(int i = 0; i<k; i++){
19         actual+=abs(nuevoA[i]-nuevoB[i]);
20       }
21       respuesta = min(respuesta,actual);
22     }
23     return;
24   }
25     //si el tamano del subconjunto B alcanzo a 'k', termina de buscar mas
     posibilidades y calcula la respuesta para esta combinacion
26   if(nuevoB.size() == k){
27     long long int actual = 0;
28         for(int i = 0; i<k; i++){
29       actual+=abs(nuevoA[i]-nuevoB[i]);
30     }
31     respuesta = min(respuesta,actual);
32     return;
33   }
34     //agrega un elemento al subconjnto actual, genera mas posibilidades
35   nuevoB.push_back(b[pos]);
36   generaB(pos+1);
37     //genera las demas posibilidades sin ese elemento
38   nuevoB.pop_back();
39   generaB(pos+1);
40   return;
41 }
42 void generaA(int pos){
43     //si ya no quedan items en A por considerar
44     if(pos==n){
45         //si en el subconjunto de A que tomamos hay exactamente 'k' elementos,
     generamos los posibles subconjuntos de b
46     if(nuevoA.size() == k){
47       generaB(0);
48     }
49     return;
50   }
51     //si el tamano del subconjunto A alcanzo a 'k', termina de buscar mas
     posibilidades y  genera los posibles subconjuntos de b
52   if(nuevoA.size() == k){
53     generaB(0);
54     return;
55   }
56     //agrega un elemento al subconjnto actual, genera mas posibilidades
57   nuevoA.push_back(a[pos]);
58   generaA(pos+1);
59     //genera las demas posibilidades sin ese elemento
60   nuevoA.pop_back();
61   generaA(pos+1);
62   return;
63 }
64
```

```
int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>n>>k;
    int elemento;
    //lee los elementos en la cinta a
    while(a.size()<n){
        cin>>elemento;
        a.push_back(elemento);
    }
    //lee los elementos en la cinta b
    while(b.size()<n){
        cin>>elemento;
        b.push_back(elemento);
    }
    //se inicializa en un valor inobtenible
    respuesta = 100000000000LL;

    //Genera todas las combinaciones de A y B
    generaA(0);
    cout<<respuesta<<endl;

}
```

Solución iterativa.

```
#include "bits/stdc++.h"
using namespace std;
#define endl '\n'

//esta solucion en particular puede dar TLE si los vectores son declarados dentro de cada solve
vector<int> a,b;
int n,k;
vector<int> nuevoA,nuevoB;
long long int solve(int mascaraA, int mascaraB){
  nuevoA.clear();nuevoB.clear();
  //genera subconjunto A
  for(int i = 0; i<n; i++){
        //si el bit actual esta encendido, meter el elemento que representa al subconjunto actual
    if(mascaraA%2==1){
      nuevoA.push_back(a[i]);
    }
        //recorrer la mascara un bit a la derecha para continuar con la siguiente posicion
    mascaraA>>=1;
  }

    //genera subconjunto B
  for(int i = 0; i<n; i++){
        //si el bit actual esta encendido, meter el elemento que representa al subconjunto actual
    if(mascaraB%2==1){
      nuevoB.push_back(b[i]);
    }
        //recorrer la mascara un bit a la derecha para continuar con la siguiente posicion
    mascaraB>>=1;
  }
```

```
30
31      //calcula el resultado
32    long long int resultado = 0;
33    for(int i = 0; i<k; i++){
34      resultado+=abs(nuevoA[i]-nuevoB[i]);
35    }
36
37    return resultado;
38  }
39  //--------------------SOLBEGIN----------------------------------
40  int main() {
41    ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
42    int elemento;
43    cin>>n>>k;
44
45      //lee los contenidos de A
46    while(a.size()<n){
47      cin>>elemento;
48      a.push_back(elemento);
49    }
50
51      //lee los contenidos de B
52    while(b.size()<n){
53      cin>>elemento;
54      b.push_back(elemento);
55    }
56
57      //todas las combinaciones involucrando n elementos pueden ser mapeadas de forma
        unica usando una
58      //mascara de bits en el rango [0-2**n)
59    int upp = pow(2,n);
60    vector<int> mascaras;
61    for(int i = 0; i<upp; i++){
62          //si la mascara actual tiene k bits, significa que representa un estado de
       una cinta con k elementos, por lo que se guarda
63      if(__builtin_popcount(i)==k){
64        mascaras.push_back(i);
65      }
66    }
67
68      //se inicializa el resultado con un valor inobtenible
69    long long int resultado = 100000000000LL;
70    for(int mascaraA: mascaras){
71      for(int mascaraB: mascaras){
72        resultado = min(resultado,solve(mascaraA,mascaraB));
73      }
74    }
75    cout<<resultado<<endl;
76  }
77  //--------------------EOSOLUTION----------------------------------
```

E.- Elimination sort.

Hints.

- ¿Es realmente necesario implementar el algoritmo tal y como se describe?
- Si un numero no es eliminado de la lista, ¿Que determina cuales de sus números a su derecha si lo sean?

Solución.

Para este problema, los limites de memoria y de tiempo permiten resolverlo tal y como el algoritmo lo describe o con una implementación mas limpia.

Implementación directa.

```
#include "bits/stdc++.h"
using namespace std;
#define endl '\n'

vector<int> eliminar(vector<int>arr,int pos){
    vector<int> aux;
    for(int i = 0; i<arr.size(); i++){
        if(i==pos){
            continue;
        }
        aux.push_back(arr[i]);
    }
    return aux;
}
//--------------------SOLBEGIN---------------------------------
int main() {
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int n,elemento;
    bool estaOrdenado;
    vector<int> arreglo;
    cin>>n;

    while(arreglo.size()<n){
        cin>>elemento;
        arreglo.push_back(elemento);
    }

    estaOrdenado = false;
    while(estaOrdenado == false){
        //inicia en 1 porque 0 no tiene antecesor
        for(int i = 1; i<arreglo.size(); i++){
            //si un numero es menor a su antecesor entonces debe de ser eliminado
            if(arreglo[i]<arreglo[i-1]){
                arreglo = eliminar(arreglo,i);
                //como el numero en la posicion i fue eliminado, se tiene que
   volver a checar ese elemento, por lo que es necesario decrementar a i
                i--;
            }
        }
        //esta ordenado hasta que se demuestre lo contrario
        estaOrdenado = true;
```

```
        for(int i = 1; i<arreglo.size(); i++){
            //si un elemento es menor a su antecesor, entonces el arreglo no esta ordenado
            if(arreglo[i]<arreglo[i-1]){
                estaOrdenado = false;
                break;
            }
        }
    }

    bool primero = true;
    for(int el: arreglo){
        //Imprime el espacio solo si el elemento no es el primero en ser impreso
        if(primero){
            primero = false;
        }else{
            cout<<" ";
        }
        //imprime cada elemento del arreglo
        cout<<el;
    }
    cout<<endl;
}
//--------------------EOSOLUTION---------------------------------
```

Implementación corta.

Es posible crear un algoritmo equivalente al descrito en el enunciado del problema pero con una implementación mucho mas corta. Basta con hacer algunos casos de prueba a mano para observar que si un numero en el arreglo original tiene un elemento mas grande a su izquierda, este numero eventualmente será removido y estará ausente en el arreglo final.

```
#include "bits/stdc++.h"
using namespace std;
#define endl '\n'

vector<int> eliminar(vector<int>arr,int pos){
    vector<int> aux;
    for(int i = 0; i<arr.size(); i++){
        if(i==pos){
            continue;
        }
        aux.push_back(arr[i]);
    }
    return aux;
}
//--------------------SOLBEGIN---------------------------------
int main() {
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int n,elemento;
    vector<int> arreglo;
    cin>>n;

    while(arreglo.size()<n){
        cin>>elemento;
        arreglo.push_back(elemento);
    }
    int valorMax = arreglo[0];
    bool primero = true;
```

```
28      for(int el: arreglo){
29          if(el>=valorMax){
30              if(primero == true){
31                  primero = false;
32              }else{
33                  cout<<" ";
34              }
35              valorMax = el;
36              cout<<el;
37          }
38      }
39      cout<<endl;
40  }
41  //---------------------EOSOLUTION----------------------------------
```

F.- F(r)ontier.

Hints.

- Para leer lineas con espacios en C++ recuerda usar *getline(cin,cadena);*
- Si se mezcla el uso de *cin* con el de *getline*, recuerda poner un *cin.ignore()* despues de cada *cin*.
- ¿Ayudara mantener un registro de que lado tiene una linea vertical para cada letra?
- ¿Puedes pensar en una solución que cuente TVL y despues VVL?
- ¿Consideraste casos especiales (I, HIM, II, IM,HH)?

Solución.

La solución a este ejercicio resulta ser bastante directa, solo es necesario hardcodear (Incrustar directamente en el código) que letras tienen una linea izquierda y cuales una linea derecha para después calcular el *Slickness* de la cadena tal y como lo describe el enunciado del problema. **Nota:** se debe tener cuidado a la hora de contar la letra 'I', pues esta se puede considerar que tiene una linea tanto a la izquierda como en la derecha, lo cual puede llevar a contar la misma linea dos veces.

```
#include "bits/stdc++.h"
using namespace std;
#define endl '\n'

//Por defecto todos son falso
bitset<256> palDer;
bitset<256> palIzq;

int contarVertTot(string s){
  int res = 0;
  for(char l: s){
    if(l!=' '){
      if(palDer[l]){
        res++;
      }
            //evita contar la linea vertical de la I dos veces
      if(l=='I'){
        continue;
      }
      if(palIzq[l]){
        res++;
      }
    }
  }
  return res;
}
int contarVertVis(string s){
    //se agrega un espacio al final para generalizar la logica (no tratar con el caso especial del ultimo caracter)
  s+=' ';
    //se inicializa el resultado dependiendo si la primera letra tiene una linea vertical o no.
  int res = (palIzq[s[0]]?1:0);

  for(int i = 1; i<s.size(); i++){
        //Si la letra anterior tiene una linea derecha,la actual una izquierda
```

```
35          //y si la letra anterior no es I (para evitar contar doble)
36      if((palDer[s[i-1]]||palIzq[s[i]]) && s[i-1]!='I'){
37        res++;
38      }
39    }
40
41    return res;
42  }
43  //--------------------SOLBEGIN----------------------------------
44  int main() {
45    ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
46      //se enlistan todos los que tienen palo izquierdo;
47    palIzq['A'] = 1;palIzq['B'] = 1;palIzq['C'] = 1;palIzq['D'] = 1;palIzq['E'] = 1;
       palIzq['F'] = 1;palIzq['G'] = 1;palIzq['H'] = 1;palIzq['I'] = 1;palIzq['K'] = 1;
       palIzq['L'] = 1;palIzq['M'] = 1;palIzq['N'] = 1;palIzq['O'] = 1;palIzq['P'] = 1;
       palIzq['Q'] = 1;palIzq['R'] = 1;palIzq['T'] = 1;palIzq['U'] = 1;palIzq['W'] = 1;
48    //se enlistan las que tienen palo derecho
49    palDer['A'] = 1;palDer['H'] = 1;palDer['I'] = 1;palDer['J'] = 1;palDer['M'] = 1;
       palDer['N'] = 1;palDer['O'] = 1;palDer['U'] = 1;palDer['V'] = 1;palDer['W'] = 1;

50    int n;
51    int tvl,vvl;
52    string s;
53
54      //si se mezclan cin con getline, es necesario poner cin.ignore() por cada cin
55    cin>>n;cin.ignore();
56
57    while(n--){
58      getline(cin,s);
59      tvl = contarVertTot(s);
60      vvl = contarVertVis(s);
61      cout<<tvl-vvl<<endl;
62    }
63
64  }
65  //--------------------EOSOLUTION----------------------------------
```

G.- Growing Bacteria.

Hints.

- Simula el problema para una bacteria con D pequeño, ¿La cantidad de bacterias en un momento dado se comporta como alguna secuencia de enteros?
- ¿¿¿Fibonacci???
- Si el tiempo aumenta, la cantidad de bacterias puede aumentar o permanecer igual.
- ¿¿¿Busqueda binaria sobre el tiempo???

Solución.

Primero, simulemos el problema para bacterias con $D = 10$, donde cada elemento en la columna de lista de bacterias representa el timer interno de cada bacteria en un momento dado.

Tiempo (Milisegundos)	Lista de Bacterias	Cantidad
0	5	1
5	10,20	2
15	10,10,20	3
25	10,10,10,20,20	5
35	10,10,10,10,10,20,20,20	8

Es fácil notar que la cantidad de bacterias cada fila de la tabla tiene un parecido a la sucesión de Fibonacci, esto se debe al hecho de que la distancia entre el timer de una nueva bacteria y de su descendiente es exactamente D, por lo que cada D segundos las bacterias *viejas* generaran una nueva bacteria nueva, mientras que las anteriores bacterias *nuevas* se convertirán en *viejas*.

De la observación anterior es posible concluir lo siguiente: Sí se tiene una bacteria con un valor de tiempo inicial t_i y se observa a ella y a sus descendientes en un momento M, el número total de bacterias en el grupo será de $fib((M/D) + 2)$ si el residuo de la división de M por D es mayor o igual a t_i. Si el residuo de la división de M por D es menor a t_i, entonces el número total de bacterias será de $fib((M/D) + 3)$. **Nota:** $fib(0) = 0$, $fib(1) = 1$, $fib(2) = 1$, $fib(3) = 2$ ...

También es fácil notar que sí el numero de milisegundos transcurrido aumenta, la cantidad de bacterias puede mantenerse igual o crecer, esto implica que es posible hacer búsqueda binaria sobre el tiempo para encontrar el primer momento donde la cantidad de bacterias es mayor o igual a las requeridas.

Implementación.

```
#include "bits/stdc++.h"
//assert(x>0) si falla da RTE
using namespace std;
#define endl '\n'

unsigned long long int fib(int n){
  unsigned long long int a = 1,b = 1,aux;
  if(n<=2){
    return 1;
```

```
  }
  for(int i = 3; i<=n; i++){
    aux = a+b;
    a = b;
    b = aux;
  }
  return b;
}

unsigned long long int D;
map<unsigned long long int, unsigned long long int> bacteriaCount;


unsigned long long int calcular(unsigned long long int milisegundos){
    unsigned long long int res = 0;
    if(milisegundos == 0){
        for(pair<unsigned long long int, unsigned long long int> el: bacteriaCount)
   {
            res+=el.second;
        }
        return res;
    }

  for(pair<unsigned long long int, unsigned long long int> el: bacteriaCount){
    //f(0) = 0, f(1) = 1, f(2) = 1, f(3) = 2
    if(el.first <= milisegundos%D){
            //para evitar hacer el procedimiento por cada bacteria con timer
   inicial en el.first
            //se multiplica el de una sola bacteria
      res+=fib((milisegundos/D)+3)*el.second;
    }else{
            //para evitar hacer el procedimiento por cada bacteria con timer
   inicial en el.first
            //se multiplica el de una sola bacteria
      res+=fib((milisegundos/D)+2)*el.second;
    }
  }
  return res;
}
//--------------------SOLBEGIN----------------------------------
int main() {
  ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
    unsigned long long int n, necesarias;
  cin>>n>>necesarias>>D;
  unsigned long long int e;
  for(int i = 0; i<n; i++){
    cin>>e;
    bacteriaCount[e]++;
  }


  unsigned long long int l = 0,m;
    //fib(70) es aproximadamente 10**14, en el peor de los casos, si nos dan una
   sola bacteria y se necesitan 10**13
    //esta garantizado que no tomara mas de D*70 dias
    unsigned long long int r = D*70;

  while(r>l){
    m = l+r; m/=2ULL;
```

```
65      if(calcular(m)<necesarias){
66        l = m+1;
67      }else{
68        r = m;
69      }
70    }
71      cout<<r<<endl;
72  }
73  //---------------------EOSOLUTION----------------------------------
```

H.- Harolds CUBG Conondrum.

Hints.

- ¿Es realmente necesario generar todo el mapa para poder explorarlo?
- ¿Como sabes si un nodo es elegible como punto de reunión?
- Si un nodo no es elegible como punto de reunión ¿es realmente necesario simular cuanto tardan los amigos en llegar al nodo en cuestión?
- ¿Existe una manera mas eficiente de encontrar cuanto tiempo tardan los amigos en llegar?

Solución.

Imagina que cada amigo hace todo el recorrido desde su nodo de aparición hasta el centro del mapa (nodo 1) y registra su recorrido. Ahora, si invirtieras esa lista por cada amigo y compararas cada elemento, el primero seria el nodo 1, el segundo seria el nodo 2, el tercero ... y asi de forma sucesiva, si se siguiese comparando la lista hasta que la lista de un amigo acabe o difiera en algún valor, podemos encontrar el ultimo punto en el que coinciden, o dicho de otra manera el nodo en comun X que minimiza la distancia desde los puntos de aparición de cada amigo.

Una vez obtenido eso, el tiempo que Harold debe esperar se puede obtener solamente simulando los caminos de todos los amigos hasta el punto de reunion, si Harold es el ultimo en llegar entonces el espera 0 segundos, si alguien mas llega ultimo, Harold debe esperar desde que el llega al nodo de reunión hasta que el ultimo amigo llegue.
La implementación demuestra ser relativamente sencilla.

```
#include "bits/stdc++.h"
//assert(x>0) si falla da RTE
using namespace std;
#define endl '\n'


vector<unsigned long long int>*arr;
void sim(unsigned long long int n, unsigned long long int meta,int ind){
  while(n!=meta){
    arr[ind].push_back(n);
        if(n%2LL==0LL){
            n/=2LL;
        }else{
            n*=3LL;
            n++;
        }
    }
    arr[ind].push_back(1);
}
//--------------------SOLBEGIN----------------------------------
int main() {
  ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);

  vector<unsigned long long int> cam;
    //Este tipo de dato se usa porque los nodos pueden tener un ID tan grande como
    10**18
  unsigned long long int n,spawnHarold,spawnAmigo,x;

```

```
  cin>>n>>spawnHarold;
  arr = new vector<unsigned long long int>[n+1]();
    sim(spawnHarold,1,n);


    for(int i = 0; i<n; i++){
        cin>>spawnAmigo;
        sim(spawnAmigo,1,i);
    }

    bool allSame = true;
    while(allSame){
        //Lo iguala a x, por si este es el ultimo nodo en comun
        x = arr[0].back();
        for(int i = 0; i<=n; i++){
            arr[i].pop_back();
        }

        //revisa que todos los amigos sigan en el mismo nodo
        //o que no se haya acabado la lista para algun amigo
        for(int i = 0; i<=n; i++){
            if(arr[i].size()==0){
                allSame = false;
                break;
            }
            if(i>0){
                if(arr[i].back()!=arr[i-1].back()){
                    allSame = false;
                    break;
                }
            }
        }
    }

    int tamCaminoHarold,maximoCamino;
    //Harold tiene que recorrer los nodos que quedaron
    tamCaminoHarold = arr[n].size();
    maximoCamino = 0;

    //busca el amigo, incluyendo a Harold, con el camino pendiente mas largo
    for(int i = 0; i<=n; i++){
        maximoCamino = max(maximoCamino,(int)arr[i].size());
    }
    cout<<"Harold must reach node "<<x<<" and wait "<<maximoCamino-tamCaminoHarold
   <<" second(s) for his friend(s) to arrive"<<endl;
}
//--------------------EOSOLUTION---------------------------------
```

I.- Integer Reversal.

Existen dos principales formas de hacer esto, usando funciones de strings nativas del lenguaje, o con módulos y divisiones.

Solución

Usando funciones de strings.

```
#include "bits/stdc++.h"
using namespace std;
#define endl '\n'

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    string numero;
    cin>>numero;
    reverse(numero.begin(),numero.end());
    cout<<stoi(numero)<<endl;
}
```

Usando modulos y divisiones

```
#include "bits/stdc++.h"
using namespace std;
#define endl '\n'

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int numero;
    int resultado = 0;
    cin>>numero;

    while(numero>0){
        resultado*=10;
        resultado+=numero%10;
        numero/=10;
    }
    cout<<resultado<<endl;
}
```

J.- Permutation Checker.

Se puede resolver de muchas maneras, considerando que el arreglo que se te da en la entrada del problema esta ordenado, una solucion posible seria:

```cpp
#include "bits/stdc++.h"

using namespace std;
#define endl '\n'

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    int n;
    cin>>n;

    int numero;
    bool valido = true;

    for(int i = 1; i<=n; i++){
        cin>>numero;
        if(numero!=i){
            valido = false;
        }
    }
    cout<<(valido?"YES":"NO")<<endl;
}
```

K.- Kodemania?

Existen dos formas de implementar este problema: Con un if por cada letra valida de 'kodemania2023' o usando cadenas para evitar escribir todos los ifs.

Solución

Con un if por letra

```cpp
#include "bits/stdc++.h"

using namespace std;
#define endl '\n'

bool esValida(char letra){
    if(letra == 'k') return true;
    if(letra == 'o') return true;
    if(letra == 'd') return true;
    if(letra == 'e') return true;
    if(letra == 'm') return true;
    if(letra == 'a') return true;
    if(letra == 'n') return true;
    if(letra == 'i') return true;
    if(letra == 'a') return true;
    if(letra == 'K') return true;
    if(letra == 'O') return true;
    if(letra == 'D') return true;
    if(letra == 'E') return true;
    if(letra == 'M') return true;
    if(letra == 'A') return true;
    if(letra == 'N') return true;
    if(letra == 'I') return true;
    if(letra == 'A') return true;
    if(letra == '2') return true;
    if(letra == '0') return true;
    if(letra == '3') return true;
    return false;
}
int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    char letra;
    cin>>letra;
    cout<<(esValida(letra)?"YES":"NO")<<endl;
}
```

Usando Strings

```cpp
#include "bits/stdc++.h"

using namespace std;
#define endl '\n'

bool esValida(char letra){
    string s = "KODEMANIAkodemania2023";
    for(char l: s){
        if(l== letra) return true;
    }
```

```
    return false;
}
int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    char letra;
    cin>>letra;
    cout<<(esValida(letra)?"YES":"NO")<<endl;
}
```