# Soluciones OIFem 2020

# Concurso de Práctica 2

## Tienda de teléfonos

### Teoría

- I/O
- Operadores aritméticos

### Solución

Imprimimos el producto de el número de móviles que queremos comprar y el precio de cada uno, 10000.

### Código

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n;
    cin >> n;
    cout << n * 10000 << '\n';

    return 0;
}
```

Autora de la solución: Helena Folia Cots, concursante

# Adulto

## Teoría

- I/O
- Condicionales

## Solución

Una vez sabida la edad de la persona, vemos si es mayor o igual a 18 con un if, e imprimimos un mensaje acorde a esto.

## Código

```cpp
#include <iostream>
using namespace std;

int main() {
        int edad=0;
        cin>>edad;
        if(edad>=18) cout<<"adulto"<<endl;
        else cout<<"joven"<<endl;
}
```

Autora de la solución: Laia Mayné Arévalo, concursante

# Watermelon

## Teoría

- I/O
- Condicionales
- Operador módulo

## Solución

Se podrá obtener una solución en el caso de que $n$ sea par y mayor que 2, ya que siempre podremos partir en 2 y $n-2$. En cambio, si $n$ es impar, nunca podrá formarse con la suma de dos pares y el número 2 tampoco, ya que solo hay dos sumas que hagan 2: 1+1 y 0+2 y ninguna de estas dos está compuesta por dos pares positivos.

## Código

```cpp
#include <iostream>

using namespace std;

int main (){
    int n;
    cin >> n;
    if (n % 2 == 0 and n>2) cout << "SI\n";
    else cout << "NO\n";
}
```

Autora de la solución: Adriana Aguiló Martínez, concursante

# ABC

## Teoría

- I/O
- Condicionales
- Ordenamiento
- Simulación

## Solución

Ordenamos los tres números de la entrada e imprimimos lo que nos piden, utilizando condicionales para hacerlo en el orden adecuado.

## Código

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    vector<int> v(3);
    cin >> v[0] >> v[1] >> v[2];

    sort(v.begin(), v.end());

    char a, b, c;

    cin >> a >> b >> c;

    if (a == 'A') cout << v[0] << ' ';
    else if (a == 'B') cout << v[1] << ' ';
    else cout << v[2] << ' ';

    if (b == 'A') cout << v[0] << ' ';
    else if (b == 'B') cout << v[1] << ' ';
    else cout << v[2] << ' ';

    if (c == 'A') cout << v[0] << '\n';
    else if (c == 'B') cout << v[1] << '\n';
    else cout << v[2] << '\n';

    return 0;
}
```

Autora de la solución: Helena Folia Cots, concursante

# El ladrón precavido

## Teoría

- Programación dinámica
- Backtracking

## Solución

Este problema es una variación de knapsack, cuyo código es casi igual al de knapsack. Solo hay que hacer dos cambios: si cogemos el lingote con índice $i$, no podemos coger el lingote con índice $i-1$. Además, como queremos que la salida esté en orden lexicográfico, invertiremos el orden de los lingotes, empezando al final en vez de al principio, ya que un menor orden lexicográfico de izquierda a derecha es equivalente a un mayor orden lexicográfico de derecha a izquierda. Es mucho más fácil atacar el problema de esta forma, ya que se soluciona escogiendo de forma voraz el lingote actual si esto es posible.

Finalmente, debemos imprimir la solución de izquierda a derecha. Encontraremos estos índices haciendo backtracking sobre la tabla de programación dinámica.

## Código

```
#include <iostream>
#include <vector>
using namespace std;

int N, C;
vector<int> L;
vector<vector<int>> pre;
vector<int> solucion;

void rellenarTabla() {
        pre.assign(N, vector<int>(C + 1, -1));
        pre[0][L[0]] = 0;
        pre[1][L[0]] = 0;
        pre[1][L[1]] = 1;
        for (int i = 2; i < N; i++) {
                for (int j = 0; j < L[i]; j++)
                        pre[i][j] = pre[i - 1][j];
                pre[i][L[i]] = i;
                for (int j = L[i] + 1; j <= C; j++) {
                        if (pre[i - 2][j - L[i]] != -1)
                                pre[i][j] = i;
                        else if (pre[i - 1][j] != -1)
                                pre[i][j] = pre[i - 1][j];
                }
        }
}

void backtrack(int i, int j) {
  if (i < 0 || pre[i][j] == -1)
    return;
  if (pre[i][j] == i) {
```

```
            solucion.push_back(N-i-1);
            backtrack(i-2, j - L[i]);
    } else backtrack(i-1, j);
}


void imprimirSolucion() {
        for (int k = C; k > 0; k--) {
                if (pre[N - 1][k] != -1) {
                        cout << k << '\n';
                        backtrack(N - 1, k);
                        cout << solucion[0];
                        for (wchar_t i = 1; i < solucion.size(); i++)
                                cout << ' ' << solucion[i];
                        cout << '\n';
                        return;
                }
        }
        cout << "0\n";
}

int main() {
        ios::sync_with_stdio(false);
        cin.tie(NULL);
        cin >> N >> C;
        L = vector<int>(N);
        for (int i = 0; i < N; i++)
                cin >> L[N-i-1];
        rellenarTabla();
        imprimirSolucion();
        return 0;
}
```

Autora de la solución: Blanca Huergo Muñoz, organizadora

# El castillo

## Teoría

- DFS (búsqueda en profundidad)
- Componentes conexas

## Solución

Usamos DFS para encontrar las componentes conexas del castillo y sus respectivos tamaños. La dificultad de este problema está en saber qué pared destruir para juntar dos componentes. Podemos descubrirla viendo qué componentes son contiguas y qué pareja de componentes contiguas tiene la suma mayor, imprimiendo una pared que las conecte.

## Código

```
#include <iostream>
#include <vector>
using namespace std;

int cont;
void dfs(int a,vector<vector<int> >& v,vector<bool>& visto,int id,vector<int>&
 ↪  iden){
        visto[a]=true;
        iden[a]=id;
        cont++;
        for(int x:v[a]){
                if(!visto[x])dfs(x,v,visto,id,iden);
        }
}

int main(){
        int n,m,mod;
        cin>>n>>m;
        vector<vector<int> >v(n*m,vector<int>());
        for(int i=0;i<n;i++){
                for(int j=0;j<m;j++){
                        cin>>mod;
                        if(mod>=8){
                                mod-=8;
                        }
                        else{
                                v[i*m+j].push_back(i*m+j+m);
                                v[i*m+j+m].push_back(i*m+j);
                        }
                        if(mod>=4){
                                mod-=4;
                        }
                        else{
                                v[i*m+j].push_back(i*m+j+1);
                                v[i*m+j+1].push_back(i*m+j);
                        }
                        if(mod>=2){

```

```
                                mod-=2;
                        }
                        else{
                                v[i*m+j].push_back(i*m+j-m);
                                v[i*m+j-m].push_back(i*m+j);
                        }
                        if(mod>=1){
                                mod=0;
                        }
                        else{
                                v[i*m+j].push_back(i*m+j-1);
                                v[i*m+j-1].push_back(i*m+j);
                        }
                }
        }

        int maxi=0,habitaciones=0;
        vector<bool>visto(n*m,false);
        vector<int> identificar(n*m);
        vector<int> valor(n*m);
        for(int i=0;i<n*m;i++){
                if(!visto[i]){
                        habitaciones++;
                dfs(i,v,visto,habitaciones,identificar);
                if(cont>maxi)maxi=cont;
                for(int i=0;i<n;i++){
                        for(int j=0;j<m;j++){
                                if(identificar[i*m+j]==habitaciones){
                                        valor[i*m+j]=cont;
                                }
                        }
                }
                cont=0;
        }
        }
        cout<<habitaciones<<' '<<maxi<<'\n';
        pair<int,int> sol;
        maxi=0;
        for(int i=0;i<n;i++){
                for(int j=0;j<m;j++){
                        if(j){
                                if(identificar[i*m+j]!=identificar[i*m+j-1] &&
                                ↪  maxi<valor[i*m+j]+valor[i*m+j-1]){
                                        maxi=valor[i*m+j]+valor[i*m+j-1];
                                        sol=make_pair(i*m+j,i*m+j-1);
                                }
                        }
                        if(i){
                                if(identificar[i*m+j]!=identificar[i*m+j-m] &&
                                ↪  maxi<valor[i*m+j]+valor[i*m+j-m]){
                                        maxi=valor[i*m+j]+valor[i*m+j-m];
                                        sol=make_pair(i*m+j,i*m+j-m);
                                }
                        }
                }
```

```
90          }
91          if(sol.first==sol.second+1){
92                  cout<<sol.second/m+1<<' '<<sol.second%m+1<<" E\n";
93          }
94          else{
95                  cout<<sol.second/m+1<<' '<<sol.second%m+1<<" S\n";
96          }
97  }
```

Autora de la solución: María Lucía Aparicio García, concursante

# Chocolates

## Teoría

- Método de bisección

## Solución

Usamos el método de bisección para ver cuál es el máximo de chocolates que puede comer Christian cada día. Simulamos el proceso descrito en el enunciado para cada número de prueba, viendo si es posible o no el comer ese número de chocolates diario.

## Código

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> choc;

bool posible (int m) {
    long long int guard = 0;
    for (int i = 0; i < (int)choc.size(); i++) {
        if (choc[i] + guard < m) return false;
        if (choc[i] < m)
            guard -= (m - choc[i]);
        else
            guard += (choc[i] - m);
    }
    return true;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int n; cin >> n;
    choc.assign(n, 0);
    for (int i = 0; i < n; i++)
        cin >> choc[i];

    int lo = 0, mid, hi = 1000000000, res = 0;
    while (lo <= hi) {
        mid = lo + (hi-lo) / 2;
        if (posible(mid)) {
            lo = mid+1;
            res = mid;
        } else {
            hi = mid-1;
        }
    }
    cout << res << '\n';
    return 0;
}
```

Autora de la solución: Amal Dokkar El Hamoudi, concursante

# Androides

## Teoría

- Vectores
- Simulación

## Solución

Simulamos el proceso descrito en el enunciado.

**Extra: una solución más eficiente se consigue usando un árbol de segmentos (si no sabes lo que es, mira en los apuntes de la sección de Entrenamientos de la web). ¿Sabrías programarla?**

## Código

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int n, m, k, a, b, total;
    cin >> n >> m;
    vector<int> andr(n, 0);
    while (m--) {
        cin >> k >> a >> b;
        a--;
        if (k == 0) {
            for (int i = a; i < b; i++) andr[i] = 1 - andr[i];
        } else {
            total = 0;
            for (int i = a; i < b; i++) if (andr[i]) total++;
            cout << total << '\n';
        }
    }
    return 0;
}
```

Autora de la solución: Amal Dokkar el Hamoudi, concursante

# Donaciones

## Teoría

- Árboles de segmentos

## Solución

Resolveremos este problema mediante la creación de dos árboles de segmentos de suma. En el primer árbol (SL) meteremos el inicio de cada intervalo y en el segundo árbol (SR) el final. En todo momento conocemos la suma de todas las donaciones. Para calcular la respuesta a una query (A..B), restamos a este total todos los intervalos con un inicio superior a B, que calcularemos con SL, y todos los intervalos con un final inferior a A, que calcularemos con SR. Esa será la respuesta que imprimimos.

## Código

```cpp
#include <bits/stdc++.h>
using namespace std;

using ll = long long;
using vl = vector<ll>;

struct Ival {
    int l, r, c;
};

const int STSIZE = 1e6+1;

struct SegTree {
    vl t;

    SegTree() {
        t = vl(4*STSIZE);
    }

    void build(int v, int tl, int tr) {
        if (tl == tr) {
            t[v] = 0;
        } else {
            int tm = (tl + tr) / 2;
            build(v*2, tl, tm);
            build(v*2+1, tm+1, tr);
            t[v] = t[v*2] + t[v*2+1];
        }
    }

    ll sum(int v, int tl, int tr, int l, int r) {
        if (l > r) return 0;
        if (l == tl and r == tr) return t[v];
        int tm = (tl + tr) / 2;
        return sum(v*2, tl, tm, l, min(r, tm)) + sum(v*2+1, tm+1, tr, max(l,
        ↪  tm+1), r);
    }
```

```
37
38      void update(int v, int tl, int tr, int pos, int new_val) {
39          if (tl == tr) {
40              t[v] += new_val;
41          }
42          else {
43              int tm = (tl + tr) / 2;
44              if (pos <= tm)
45                  update(v*2, tl, tm, pos, new_val);
46              else
47                  update(v*2+1, tm+1, tr, pos, new_val);
48              t[v] = t[v*2] + t[v*2+1];
49          }
50      }
51  };
52
53  int main() {
54      ios::sync_with_stdio(0);
55      cin.tie(0);
56
57      int n;
58      cin >> n;
59      ll csum = 0;
60      vector<Ival> v(n);
61      SegTree SL, SR;
62      for (int i = 0; i < n; ++i) {
63          cin >> v[i].l >> v[i].r >> v[i].c;
64          csum += v[i].c;
65          SL.update(1, 0, STSIZE-1, v[i].l, v[i].c);
66          SR.update(1, 0, STSIZE-1, v[i].r, v[i].c);
67      }
68      int q;
69      cin >> q;
70      while (q--) {
71          int t;
72          cin >> t;
73          if (t == 1) {
74              int a, b;
75              cin >> a >> b;
76              ll ans = csum;
77              if (b+1 <= STSIZE-1) ans -= SL.sum(1, 0, STSIZE-1, b+1, STSIZE-1);
78              if (a-1 >= 0) ans -= SR.sum(1, 0, STSIZE-1, 0, a-1);
79              cout << ans << endl;
80          }
81          else {
82              int i;
83              cin >> i; --i; // to 0-idx
84              Ival Iold = v[i];
85              cin >> v[i].l >> v[i].r >> v[i].c;
86              csum += v[i].c - Iold.c;
87              SL.update(1, 0, STSIZE-1, Iold.l, -Iold.c);
88              SR.update(1, 0, STSIZE-1, Iold.r, -Iold.c);
89              SL.update(1, 0, STSIZE-1, v[i].l, v[i].c);
90              SR.update(1, 0, STSIZE-1, v[i].r, v[i].c);
91          }
```

```
92          }
93      }
```

Autor de la solución: Izan Beltrán Ferreiro, organizador

# Juego

## Teoría

- Programación dinámica

## Solución

Para resolver este problema, la clave está en que los grupos cuyo signo puedes cambiar son de tamaño $n$, la mitad de la longitud de la lista, $2n$.

Podemos demostrar que en la solución óptima, las parejas de elementos a distancia $n$ tienen que, o en todas ellas haberse cambiado uno de los dos elementos de signo o un número par de elementos de signo en todas ellas.

Con esta observación, es sencillo implementar la solución.

El código de la solución oficial en C++ es **este**.