Soluciones OIFem II Nivel 1

Entrenamiento 3

Vector-Sort

Teoría

- vector
- ordenación

Solución

Sort usado en clase.

Link al problema: Sort

Código

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> miVector(n);
    for(int i = 0; i < n; i++)
        cin >> miVector[i];
    sort(miVector.begin(), miVector.end());
    for(int i = 0; i < n; i++)
        cout << miVector[i] << " ";
    return 0;
}
```

Variable Sized Arrays

Teoría

- vectores

Solución

Creamos un vector de dos dimensiones, donde guardamos todos los demás vectores, y luego simplemente imprimimos los valores que nos indican las queries.

Link al problema: Variable Sized Arrays

Código

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int n,q,c,j;
    cin>>n>>q;
    vector<vector<int> > misVectores(n);
    for(int i = 0; i < n; i++){
        int t;
        cin >> t;
        vector<int> vectori(t);
        for(int k = 0; k < t; k++)
            cin >> vectori[k];
        misVectores[i] = vectori;
    }
    for(int i = 0; i < q; i++){
        cin >> c >> j;
        cout << misVectores[c][j] << endl;
    }
    return 0;
}
```

Sort

Teoría

- ordenar vectores

Solución

Es el mismo código, que vimos en la clase, solo que hay que tener cuidado con el formato de la salida.
Link al problema: sort

Código

```cpp
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

bool miComparador(int elemento1, int elemento2){
    if(elemento1 > elemento2) return true;
    else return false;
}

int main(){
    int n;
    while(cin >> n){
        vector<int> miVector(n);
        for(int i = 0; i < n; i++)
            cin >> miVector[i];
        sort (miVector.begin(), miVector.end(), miComparador);
        if(n != 0)
            cout << miVector[0];
        for(int i = 1; i < n; i++)
            cout << " " << miVector[i];
        cout << endl;
    }
    return 0;
}
```

Cálculo de la Mediana

Teoría

- vector
- ordenación

Solución

Almacenamos los valores dados en un vector. Ordenamos el vector y ya nos queda escoger los valores centrales. En caso de ser impar dos veces el valor central, en caso de ser par los dos centrales.

Link al problema

Código- Inicializar un vector:

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

int main(){
        int x;
        while(cin>>x){
                if(x==0)break;
                vector<int>lista(x);
                for(int i=0;i<x;i++)cin>>lista[i];//leemos valores
                sort(lista.begin(),lista.end());//ordenamos la lista
                int ans= lista[x/2]+lista[(x-1)/2];//calculamos la mediana
                cout<<ans<<"\n";
        }
        return 0;
}
```

Siete picos

Teoría

- vector
- comparadores

Solución

Inicializamos un contador de picos en cero. Aquí guardaremos nuestra solución.Guardamos las alturas en un vector y recorremos el vector. Comprobamos si la altura actual es mayor que la anterior y la siguiente si es así sumamos uno al contador de picos.Es imporante notar que la montña rusa es circular y también hay que comprobar si la primera y la última altura son picos.

Código

```
#include <vector>
#include <queue>
#include <cmath>
#include <algorithm>
#include <utility>
#include <iostream>
using namespace std;

int main(){
        int n;
        while(cin>>n){
                if(n==0)break;
                vector<int>alturas(n);
                for(int i=0;i<n;i++)cin>>alturas[i];
                int picos=0;
                for(int i=1;i<n-1;i++){
                        if(alturas[i]>alturas[i-1]&&alturas[i]>alturas[i+1])picos++;//comprobamos
                        ↪  si altura actual pico
                }
                if(alturas[0]>alturas[1]&&
                ↪  alturas[0]>alturas[n-1])picos++;//comprobamos si la primera
                ↪  altura pico
                else if(alturas[n-1]>alturas[n-2]&&
                ↪  alturas[n-1]>alturas[0])picos++;//comprobamos si la última
                ↪  altura pico
                cout<<picos<<"\n";
        }
        return 0;
}
```

Entrando al cine

Teoría

- vector
- bool

Solución

Guardamos los asientos de las personas de la fila en un vector. Tenemos un bool impar, inicializado en falso, y un contador de impares, inicializado en 0. Recorremos la lista de principio a final, si nos topamos con un impar, ponemos el bool impar a true y aumentamos el contador por uno. Si despúes nos topamos con un par, significa, que no todos los impares están al final de la lista y por tanto no pueden abrir la segunda puerta.En este caso tendemos que imprimir "NO". En caso contrario imprimos "SI" y el número de personas que se quedan en la primera fila, es decir (total de personas)-(contador de impares)

Link al problema

Código- Inicializar un vector:

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

int main(){
        int casos;
        cin>>casos;
        while(casos--){
                int personas;
                cin>>personas;
                vector<int>sitio(personas);//vector de la fila
                bool impar=false;
                bool pos=true;
                int con=0;
                for(int i=0;i<personas;i++){
                cin>>sitio[i];//leemos la fila
                }
                for(int i=0;i<personas;i++){
                if(sitio[i]%2==0){//comprobamos paridad
                        if(impar){//significa que hay por lo menos una persona
                        ↪  impar entre personas pares, por tanto no cumple
                        ↪  condición
                                pos=false;
                                break;
                        }
                }
                else{
                        impar=true;
                        con++;//contamos los impares
```

```
                }
                }
                if(pos){
                        cout<<"SI"<<" "<<personas-con<<"\n";//los que se quedan en
                        ↪  la fila son el número total de personas menos las
                        ↪  personas con asientos pares al final de la cola
                }
                else{
                        cout<<"NO"<<"\n";
                }
        }
        return 0;
}
```

Clasificación OIE

Teoría

- vectores
- if y for

Solución

Para solucionar este proplema vamos a crear un vector de dos dimensiones. Aquí vamos a guardar para cada participante cuantos puntos tiene en cada problema. Además tenemos un vector que guarda la puntuación total de cada participante. Nos guardamos la puntuación máxima y el ganador actual. Leemos cada entrada c, p, s y miramos si el participante ha coseguido más puntos que en un envio anterior, a este número lo llamamos x. Si es el caso hacemos restamos x de el número de puntos total y le sumamos s. Comparamos la puntación actualizada del participante c, pc con la maxima puntuación mx. Si $mx = pc$ entonces hay empate, si $mx < pc$ entonces c va ganando y si $mx > pc$ nada cambia.

Link al problema: calsificación OIE

Código

```cpp
#include <cmath>
#include <iostream>
#include <cstdio>
#include <vector>

using namespace std;

int main(){
    int N,P,K;
    int c,p,s;
    int ganador = -1, maxp = 0;
    cin >> N >> P >> K;
    if(N == 1) ganador = 0;
    vector<vector<int> > tabla(N, vector<int> (P,0));
    vector<int> puntuacionTotal(N,0);
```

```
    while(K--){
        cin >> c >> p >> s;
        if (tabla[c][p] < s) {
            puntuacionTotal[c] -= tabla[c][p];
            puntuacionTotal[c] += s;
            tabla[c][p] = s;
            if (puntuacionTotal[c] > maxp){
                maxp = puntuacionTotal[c];
                ganador = c;
            }
            else if (puntuacionTotal[c] == maxp) {
                ganador = -1;
            }
        }
    }
    cout <<maxp <<" " <<ganador <<endl;
    return 0;
}
```

Maraton

Teoría

- vectores
- for, if y bool

Solución

Vamos posición a posición en el tablero comprobando, si hay un tres en raya empezando por esa casilla de manera horizontal, vertical o en diagonal. Una vez encontamos una, paramos de buscar e imprimimos la posición inicial. Si no emcontramos ningún tres en raya imprimimos un "ongoing".

Link al problema: maraton

Código

```
#include <cmath>
#include <iostream>
#include <cstdio>
#include <vector>

using namespace std;

int main(){
    int n;
    cin>>n;
    vector<vector<char> > tablero(n, vector <char> (n, '.'));
    for(int i = 0; i < n; i++){
        for(int k = 0; k < n; k++){
            cin >> tablero[i][k];
        }
```

```
    }
    bool encontrado = false;
    char ganador = '.';
    for(int i = 0; (!encontrado) && (i < n); i++){
        for(int k = 0; (!encontrado) && (k < n); k++){
            if(tablero[i][k] != '.'){
                if(i+2 <= n-1){
                    if(tablero[i][k] == tablero[i+1][k] &&
                        tablero[i][k] == tablero[i+2][k]){
                            encontrado = true;
                            ganador = tablero[i][k];
                    }
                }
                if(k+2 <= n-1){
                    if(tablero[i][k] == tablero[i][k+1] &&
                        tablero[i][k] == tablero[i][k+2]){
                            encontrado = true;
                            ganador = tablero[i][k];
                    }
                }
                if(k+2 <= n-1 && i+2 <= n-1){
                    if(tablero[i][k]==tablero[i+1][k+1] &&
                        tablero[i][k]==tablero[i+2][k+2]){
                            encontrado = true;
                            ganador = tablero[i][k];
                    }
                }
                if(k-2 >= 0 && i+2 <= n-1){
                    if(tablero[i][k] == tablero[i+1][k-1] &&
                        tablero[i][k] == tablero[i+2][k-2]){
                            encontrado = true;
                            ganador = tablero[i][k];
                    }
                }
            }
        }
    }
    if (encontrado) cout << ganador <<endl;
    else  cout << "ongoing" <<endl;
    return 0;
}
```

Karte

Teoría

- vectores
- for, if, bool y string

Solución

Codificamos los palos con números del 0 al 3. Para cada palo guardamos en un vector de bool que cartas nos hemos encontrado. Este lo iniciaizamos como niguna carta encontrada. Después tenemos una variable por palo que cuenta las cartas, que nos faltan por encontrar. Estas las inicializamos con 13. Vamos leyendo el string y con ello las cartas. Para cada carta comprobamos si ya la habíamos encontrado. Si este es el caso paramos el programa e imprimimos GRESKA. En caso contrario marcamos la carta como encontrada y restamos uno a las cartas que faltan por encontar del corresponiente palo.

Link al problema: Karte

Código

```cpp
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    vector <vector<bool> > Palos(4, vector <bool> (13,false)); // al principio no tenemos ninguna carta, cuando tengamos una pondremos en su palo y número un true
    string s;
    vector <int> numPalos (4,13);
    int palo;
    // las cartas que faltan de cada palo
    cin >> s; // el string de entrada
    bool error = false; //si error=true significa que tenemos una carta dos veces y tenemos que imprimir "GRESKA"
    int x = s.size();
    for(int i = 0; (!error) && (i < x/3); i++){ //Pasamos el string de entrada
        //mirmamos primero el palo y luego en comprabarcarta el número
        int numero = (s[i*3+1]-'0')*10+(s[i*3+2]-'0');
        if (s[i*3]=='P') {
          palo=0;
        } else if (s[i*3]=='K') {
          palo=1;
        } else if (s[i*3]=='H') {
          palo=2;
        } else {
          palo=3;
        }
        if(Palos[palo][numero]) {//si ya hemos leido el número antes, ahora lo tenemos repetido-->error=true
```

```
29              error = true;
30          }
31          else{
32              Palos[palo][numero] = true;
33               //si no lo habíamos leido antes, ahora lo marcamos como leido
34              numPalos[palo]--;//nos falta una carta menos de ese palo
35          }
36      }
37      if(!error){
38          cout << numPalos[0] << " " << numPalos[1] << " " << numPalos[2] << " " <<
            ↪  numPalos[3];
39      }
40      else{
41          cout << "GRESKA";
42      }
43      return 0;
44  }
```