Soluciones OIFem II Nivel 3

Entrenamiento 4

Guess the Array

Teoría

- Problemas interactivos
- Algoritmos constructivos

Solución

El algoritmo para resolver este problema lo vimos en clase. Está explicado en los apuntes.

Código

```cpp
#include <iostream>

using namespace std;
// 4 6 1 5 5
int main() {
    int n;
    cin >> n;
    int x_mas_y, x_mas_z, y_mas_z;
    cout << "? 1 2\n";
        fflush(stdout);
        cin >> x_mas_y;
        cout << "? 1 3\n";
        fflush(stdout);
        cin >> x_mas_z;
        cout << "? 2 3\n";
        fflush(stdout);
        cin >> y_mas_z;
        int nums[n];
        nums[0] = (x_mas_y - y_mas_z + x_mas_z)/2; // 2.5
        nums[1] = x_mas_y - nums[0];
        nums[2] = x_mas_z - nums[0];
        for (int i = 3; i < n; i++) {
            cout << "? 1 " << (i+1) << '\n';
            fflush(stdout);
            cin >> nums[i];
            nums[i] -= nums[0];
```

```
        }
        cout << "!";
        for (int i = 0; i < n; i++)
            cout << ' ' << nums[i];
        fflush(stdout);
    return 0;
}
```

Carnival

Teoría

- Problemas interactivos
- Búsqueda binaria
- Divide y vencerás

Solución

Podemos modelar los grupos de personas con el mismo disfraz con la estructura de datos UFDS. Tendremos en todo momento un conjunto (C++ set) con las raíces finales, es decir, por cada disfraz de 1 a C, una persona que sepamos que lleva ese disfraz. Inicialmente ese conjunto estará vacío.

Iteraremos sobre los índices 1 a N y, para cada persona, comprobaremos si está en el conjunto de alguna de las raíces descubiertas. Si es ese el caso, haremos búsqueda binaria para encontrar cuál es el conjunto del que forma parte y uniremos con UFDS esa persona al conjunto. Si no encontramos una raíz con la que unirla, entonces esta persona formará la raíz de un conjunto nuevo.

Código

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

vector<int> par;
vector<int> roots;

int root(int a) {
        if (a == par[a])
                return a;
        return par[a] = root(par[a]);
}

void merge(int i, int j) {
        if (i == j)
                return;
        par[j] = i;
        auto it = find(roots.begin(), roots.end(), j);
        roots.erase(it, it+1);
}

void solve(int N, int C) {
        if (C == 1) {
                par.assign(N, 0);
                return;
        }
        par = vector<int>(N);
```

```
        for (int i = 0; i < N; i++)
                par[i] = i;

        roots = {};
        int ans, costumesDiscovered = 0;
        for (int i = 0; i < N; i++) {
                if (N - i == C - costumesDiscovered)
                        continue;

        // comprobaremos si hay un conjunto al que añadir a la persona
                if(roots.size()) {
                        cout << (int)roots.size()+1 << " ";
                        for (int r: roots)
                                cout << r+1 << ' ';
                        cout << i+1 << '\n';
                        cin >> ans;
                        if (ans == (int) roots.size()) {
                            // hay un conjunto -> hacemos búsqueda binaria para
                            ↪  encontrar cuál
                            int lo = 0, hi = (int) roots.size()-1, mid, k = 0;
                            while(lo <= hi) {
                                    if (lo == hi) {
                                            k = lo;
                                            break;
                                    }
                                    mid = lo + (hi-lo)/2;
                                    cout << (mid-lo+2) << " ";
                                    for (int j = lo; j <= mid; j++)
                                            cout << roots[j]+1 << ' ';
                                    cout << i+1 << '\n';
                                    cin >> ans;
                                    if (ans == (mid-lo+1)) {
                                            hi = mid;
                                            k = mid;
                                    } else lo = mid+1;
                            }
                            merge(i, roots[k]);
                        }
                }

                // esta persona forma un conjunto nuevo
                if (par[i] == i) {
                        costumesDiscovered++;
                        roots.push_back(i);
                }
        }
}

int main() {
        int N, C;
        cin >> N;
        cout << N;
        for (int i = 1; i <= N; i++)
                cout << " " << i;
        cout << '\n';
```

```
        cin >> C;
        solve(N, C);
        map<int, int> roots;
        int c = 1;
        for (int i = 0; i < N; i++) {
                if (roots.find(root(i)) == roots.end()) {
                        roots[root(i)] = c++;
                }
        }
        int costumes[N];
        for (int i = 0; i < N; i++)
                costumes[i] = roots[root(i)];
        cout << "0";
        for (int i = 0; i < N; i++)
                cout << " " << costumes[i];
        cout << '\n';
        return 0;
}
```