Soluciones OIFem II Nivel 1

Concurso de práctica 2

Elephant

Teoría

- Aritmética

Solución

Si hacemos a mano los primeros casos, pronto veremos el siguiente patrón: lo mejor es hacer saltos de longitud 5 hasta que lleguemos al destino o nos quede un último paso de longitud inferior a 5.

Código

```cpp
#include <iostream>

using namespace std;

int main()
{
    int x;
    cin >> x;
    cout << (x+4)/5;
    return 0;
}
```

Presents

Teoría

- Vectores

Solución

Este era un problema de implementación, es decir, programar cuidadosamente lo que pedía el enunciado.

Código

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

int main()
{
    int n;
    cin >> n;
    vector<int> regalos(n);
    for (int i = 1; i <= n; i++) {
        int j;
        cin >> j;
        regalos[j-1] = i;
    }
    cout << regalos[0];
    for (int i = 1; i < n; i++)
        cout << ' ' << regalos[i];
    return 0;
}
```

Games

Teoría

- Vectores
- Bucles

Solución

Este problema también es de implementación.

Código

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

int solve(vector<int> & unifCasa, vector<int> & unifVisitante) {
        int partidos = 0;
        for (size_t equipoCasa = 0; equipoCasa < unifCasa.size(); equipoCasa++) {
                for (size_t equipoVisitante = 0; equipoVisitante <
                ↪  unifVisitante.size(); equipoVisitante++) {
                        if (equipoCasa != equipoVisitante && unifCasa[equipoCasa]
                        ↪  == unifVisitante[equipoVisitante])
                                partidos++;
                }
        }
        return partidos;
}

int main() {
        ios::sync_with_stdio(false);
        cin.tie(NULL);
        int N;
        cin >> N;
        vector<int> unifCasa(N), unifVisitante(N);
        for (int i = 0; i < N; i++)
                cin >> unifCasa[i] >> unifVisitante[i];
        cout << solve(unifCasa, unifVisitante) << '\n';
        return 0;
}
```

Shuffle String

Teoría

- Strings
- Bucles

Solución

En este problema había que tener cuidado al programar lo que nos pedía usando los índices de las strings.

Código

```cpp
class Solution {
public:
    string restoreString(string s, vector<int>& indices) {
        string a(s.length(), '0');
        for (size_t i = 0; i < indices.size(); i++) {
            a[indices[i]] = s[i];
        }
        return a;
    }
};
```

Longest Palindrome

Teoría

- Mapas

Solución

Para resolver este problema era recomendable usar un mapa guardando, para cada carácter de s, cuántas veces aparecía.

Empezamos con un palíndromo vacío y vamos carácter a carácter añadiendo el máximo de ocurrencias de ese carácter al palíndromo actual. Si este carácter tiene un número par de ocurrencias, añadimos una mitad al principio y otra al final del palíndromo. Si tiene un número impar de ocurrencias, nos guardamos una y hacemos lo mismo que para un número par.

Si al terminar hemos encontrado algún carácter con un número impar de ocurrencias, lo añadiremos a la mitad del palíndromo.

Código

```
class Solution {
public:
    int longestPalindrome(string s) {
        map<char, int> occ;
        for (char c: s)
            occ[c]++;
        int len = 0;
        bool odd = 0;
        for (char c = 'a'; c <= 'z'; c++) {
            len += 2*(occ[c]/2);
            odd |= (occ[c]&1);
        }
        for (char c = 'A'; c <= 'Z'; c++) {
            len += 2*(occ[c]/2);
            odd |= (occ[c]&1);
        }
        return len+odd;
    }
};
```

Find the Difference

Teoría

- Mapas

Solución

Contamos cuántos hay de cada carácter en las dos palabras y lo guardamos en un mapa. Al terminar, comparamos hasta buscar cuál tiene un número diferente de ocurrencias.

Código

```cpp
class Solution {
public:
    char findTheDifference(string s, string t) {
        unordered_map<char, int> freq1, freq2;
        for (char c = 'a'; c <= 'z'; c++)
            freq1[c] = freq2[c] = 0;
        for (char c: s)
            freq1[c]++;
        for (char c: t)
            freq2[c]++;
        for (char c = 'a'; c < 'z'; c++)
            if (freq1[c] != freq2[c])
                return c;
        return 'z';
    }
};
```

Intersection of Two Arrays

Teoría

- Sets

Solución

Creamos un set con los elementos de `nums1` y otro con los elementos de `nums2`. Podemos encontrar la intersección iterando sobre uno de los dos conjuntos y viendo qué elementos están en el otro.

Código

```cpp
class Solution {
public:
    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
        set<int> unique1, unique2;
        for (int & n: nums1)
            unique1.insert(n);
        for (int & n: nums2)
            unique2.insert(n);
        vector<int> result;
        for (int n: unique1) {
            if (unique2.find(n) != unique2.end())
                result.push_back(n);
        }
        return result;
    }
};
```