Soluciones OIFem II Nivel 1

Entrenamiento 6

Bingo infantil

Teoría

- conjuntos ordenados

Solución

Construimos un conjunto ordenado que contiene las posibles diferencias entre dos bolas diferentes del Bingo. Debemos vigilar si la diferencia es negativa o positiva. Al final, recorremos el conjunto ordenado e imprimimos sus elementos.

Link al problema: Bingo infantil

Código

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

int main () {
    int t;
    while (cin >> t and t != 0) {
        vector<int> bolas(t);
        set<int> resta;
        for (int k = 0; k < t; ++k) cin >> bolas[k];
        for (int i = 0; i < t; ++i) {
            for (int j = i+1; j < t; ++j) {
                if (bolas[i] > bolas[j])
                    resta.insert(bolas[i] - bolas[j]);
                else resta.insert(bolas[j] - bolas[i]);
            }
        }

        bool primero = true;
        for (auto it : resta) {
            if (primero) {
                primero = false;
                cout << it;
```

```
            } else {
                cout << ' ' << it;
            }
        }
        cout << '\n';
    }
}
```

Abdicación de un rey

Teoría

- mapas

Solución

Creamos un mapa desordenado de pareja nombre-número para que guarde el último número utilizado en la dinastía para el nombre en cuestión. Así, al principio se actualiza el mapa desordenado con los reyes de la dinastía que ha habido, y después, con los sucesores, se debe actualizar y también escribir el número que le toca.

Link al problema: Abdicación de un rey

Código

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

int main () {
    int n;
    while (cin >> n and n!= 0) {
        unordered_map<string,int> dinastia(8192);
        string rey;
        while (n--) {
            cin >> rey;
            if (dinastia.find(rey) != dinastia.end()) ++dinastia[rey];
            else dinastia[rey] = 1;
        }
        int sucesor;
        cin >> sucesor;
        string s;
        while (sucesor--) {
            cin >> s;
            if (dinastia.find(s) != dinastia.end()) {
                ++dinastia[s];
            } else {
                dinastia[s] = 1;
            }
            cout << dinastia[s] << '\n';
        }
        cout << '\n';
```

```
    }
}
```

Potitos

Teoría

- conjuntos ordenados

Solución

Creamos dos conjuntos ordenados `ha_comido` y `no_ha_comido`, el primero con los alimentos que sí ha comido, y el otro con los que no. Cada vez que no come algún alimento, se añaden en `no_ha_comido`. Cada vez que come algún alimento, se añade en `ha_comido`, y si también está en `no_ha_comido`, entonces se borra. Esto podría pasar porque en un potito puede haber muchos alimentos, pero si alguno de ellos no le gusta, entonces no se lo come.

Link al problema: Potitos

Código

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

void ha_comido(set<string>& le_gusta, set<string>& no_le_gusta) {
    string comida;
    while (cin >> comida and comida != "FIN") {
        le_gusta.insert(comida);
        if (no_le_gusta.find(comida) != no_le_gusta.end()) {
            no_le_gusta.erase(comida);
        }
    }
}

void no_ha_comido(set<string>& le_gusta, set<string>& no_le_gusta) {
    string comida;
    while (cin >> comida and comida != "FIN") {
        if (le_gusta.find(comida) == le_gusta.end()) {
            no_le_gusta.insert(comida);
        }
    }
}

int main () {
    int t;
    while (cin >> t  and t != 0) {
        set<string> le_gusta;
        set<string> no_le_gusta;
        while (t--) {
            string s;
            cin >> s;
```

```
            if (s == "SI:") {
                ha_comido(le_gusta, no_le_gusta);
            } else {
                no_ha_comido(le_gusta, no_le_gusta);
            }
        }
        bool primero = true;
        for (auto it : no_le_gusta) {
            if (primero) {
                cout << it;
                primero = false;
            } else
                cout << ' ' << it;
        }
        cout << endl;
    }
}
```

Mediana dinámica

Teoría

- conjuntos ordenados

Solución

Construimos un conjunto ordenado con las palabras. Tiene que estar ordenado porque queremos encontrar la palabra mediana en el orden alfabético. Debemos separar en tres posibles casos:la primera entrada, si la palabra mediana anterior `mediana` es mayor (alfabéticamente) que la nueva palabra `s`, o si la palabra `mediana` es menor o igual (alfabéticamente) que la nueva palabra `s`. En la primera entrada, la mediana es el único número que hay.

Si `mediana` es mayor que `s`, entonces debemos mirar si tenemos un número par o impar de palabras (después de añadir `s`). Si tenemos un número impar, entonces la mediana no cambia, pero si tenemos un número par, entonces debemos bajar una posición en el conjunto ordenado para encontrar la nueva mediana.

Si `mediana` es menor o igual que `s`, entonces debemos mirar si tenemos un número par o impar de palabras (después de añadir `s`). Si tenemos un número par, entonces la mediana no cambia, pero si tenemos un número impar, entonces debemos subir una posición en el conjunto ordenado para encontrar la nueva mediana.

Link al problema: Mediana dinámica

Código

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

void median(set<string>& palabras, string& mediana, string s, bool& inicio) {
    palabras.insert(s);
```

```
    if (inicio) {
        mediana = s;
        inicio = false;
    } else {
        auto it = palabras.find(mediana);
        if (mediana > s) {
            if (palabras.size()%2 == 0) --it;
        } else {
            if (palabras.size()%2 == 1) ++it;
        }
        mediana = *it;
    }
}

int main () {
    set<string> palabras;
    string s, mediana;
    bool inicio = true;
    while (cin >> s and s != "END") {
        median(palabras, mediana, s, inicio);
        cout << mediana << '\n';
    }
}
```

Casino

Teoría

- funciones
- mapas ordenados

Solución

Creamos un mapa ordenado `jugadores` con parejas (nombre jugador dentro del casino, dinero que ha ganado). Debemos procesar de forma diferente la entrada o salida de un jugador en el casino, o si ha ganado dinero. Debemos vigilar en los casos que no tengan sentido (como si un jugador ya ha entrado al casino no puede volver a entrar, o si se ha marchado entonces no puede ganar dinero, o un jugador no puede salir de casino si no está dentro).

Si un jugador sale del casino, debemos escribir la cantidad de dinero que ha ganado y borrarlo del mapa `jugadores`. Si un jugador entra en el casino, debemos actualizar el dinero que ha ganado a 0. Si un jugador ha ganado dinero, se debe sumar a la cantidad que tenía anteriormente.

Link al problema: Casino

Código

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

void entrada(map <string, int>& jugadores, string name) {
    if (jugadores.find(name) != jugadores.end()) {
```

```
        cout << name << " is already in the casino" << endl;
    } else {
        jugadores[name] = 0;
    }
}

void salida(map <string, int>& jugadores, string name) {
    if (jugadores.find(name) == jugadores.end() ) {
        cout << name << " is not in the casino" << endl;
    } else {
        cout << name << " has won " << jugadores[name] << endl;
        jugadores.erase(name);
    }
}

void ganar(map <string, int>& jugadores, string name, int ingreso) {
    if (jugadores.find(name) == jugadores.end()) {
        cout << name << " is not in the casino" << endl;
    } else {
        jugadores[name] += ingreso;
    }
}

int main () {
    map <string, int> jugadores;
    string name;
    while (cin >> name) {
        string mid;
        cin >> mid;
        if (mid == "enters") {
            entrada(jugadores, name);
        } else if (mid == "leaves") {
            salida(jugadores, name);
        } else {
            int ingreso;
            cin >> ingreso;
            ganar(jugadores, name, ingreso);
        }
    }
    cout << "----------" << endl;
    for (auto it: jugadores) {
        cout << it.first << " is winning " << it.second << endl;
    }
}
```

Bolsa de palabras

Teoría

- mapas ordenados

Solución

Creamos un mapa ordenado `palabras` con parejas (palabra, número de veces que aparece). En caso de recibir una nueva palabra, se inicializa a 1. Si se borra una palabra, debemos mirar si hay más apariciones de la palabra o es la última. Si piden encontrar el mínimo, como el mapa está ordenado alfabéticamente, entonces sabemos que estará en la primera posición. De la misma forma, si piden la palabra lexicográficamente mayor, sabemos que estará en la última posición del mapa `palabras`.

Link al problema: Bolsa de palabras

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

void guardar(map<string, int>& palabras, string t) {
    if (palabras.find(t) != palabras.end()) ++palabras[t];
    else palabras[t] = 1;
}

void borrar(map<string,int>& palabras, string t) {
    if (palabras.find(t) != palabras.end()) {
        if (palabras[t] == 1) palabras.erase(t);
        else --palabras[t];
    }
}

void buscar_minimo(map<string,int>& palabras) {
    if (palabras.empty()) cout << "indefinite minimum\n";
    else {
        auto it = palabras.begin();
        cout << "minimum: " << it->first << ", " << it->second << " time(s)\n";
    }
}

void buscar_maximo(map<string,int>& palabras) {
    if (palabras.empty()) cout << "indefinite maximum\n";
    else {
        auto it = palabras.rbegin();
        cout << "maximum: " << it->first << ", " << it->second << " time(s)\n";
    }
}

int main () {
    map <string, int> palabras;
    string s;
    while (cin >> s) {
        if (s == "maximum?") buscar_maximo(palabras);
        else if (s == "minimum?") buscar_minimo(palabras);
        else if (s == "store") {
            string t;
            cin >> t;
            guardar(palabras, t);
        } else {
            string t;
            cin >> t;
            borrar(palabras, t);
        }
    }
}
```

Llenando la bolsa

Teoría

- conjuntos ordenados
- paso por referencia

Solución

Creamos dos conjuntos ordenados para guardar las gemas que Ali Baba se llevaría `se_lleva`, y las gemas que dejaría en la cueva `deja`. También tenemos que llevar la cuenta de la suma de las gemas que se lleva `suma` en todo momento.

Si los ladrones dejan una gema más valiosa que las que se llevaba, entonces añadimos la nueva gema, y cambiamos la gema menos valiosa de `se_lleva`, y la insertamos en `deja`. Si los ladrones cogen una joya de la cueva y no estaba en `se_lleva`, entonces la borramos de `deja`. Sin embargo, si la joya sí está en `se_lleva`, entonces, la borramos del conjunto y añadimos la joya con más valor de `deja`.

Link al problema: Llenando la bolsa

Código

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

using ll = long long int;

void dejar(set<ll>& s1, set<ll>& s2, ll& suma, int n, ll k) {
    if ((int) s1.size() < n) {
        s1.insert(k);
        suma += k;
    } else {
        ll aux = *s1.begin();
        if (k > aux) {
            s1.insert(k);
            suma += k;
            s1.erase(aux);
            suma -= aux;
            s2.insert(aux);
        } else {
            s2.insert(k);
        }
    }
}

void coger(set<ll>& s1, set<ll>& s2, ll& suma, ll k) {
    if (s1.find(k) != s1.end()) {
        s1.erase(k);
        suma -= k;
        if (not s2.empty()) {
            ll aux = *s2.rbegin();
            s1.insert(aux);
            suma += aux;
```

```
            s2.erase(aux);
        }
    } else {
        s2.erase(k);
    }
}

int main () {
    set <ll> s1, s2;
    int n;
    ll k, suma = 0;
    cin >> n;
    string t;
    while (cin >> t >> k) {
        if (t == "leave") {
            dejar(s1, s2, suma, n, k);
        }
        else {
            coger(s1, s2, suma, k);
        }
        cout << suma << endl;
    }
}
```

Rol clasificatorio

Teoría

- conjuntos y mapas ordenados

Solución

Creamos un mapa ordenado `elo` para guardar las parejas (nombre jugador, número elo), en el que se encuentran todos los jugadores que se han conectado en algún momento. Creamos otro mapa ordenado `conectado` de parejas (nombre jugador, `true` si está conectado o `false` si no). Así podemos controlar si dos jugadores puede jugar o hay alguno no conectado. También creamos un conjunto ordenado `ranking` de parejas (número elo, nombre jugador). De esta forma, tenemos ordenados por número elo a los jugadores. Sin embargo, guardamos con un valor negativo el valor de elo para que estén ordenados de mayor elo a menor elo.

Link al problema: Rol clasificatorio

Código

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

int valor_inicial = 1200;

void login(map<string,int>& elo, map<string,bool>& conectado,
↪  set<pair<int,string>>& ranking, string player) {
```

```
    conectado[player] = true;
    if (elo.find(player) == elo.end()) {
        elo[player] = valor_inicial;
        ranking.insert(make_pair(-valor_inicial, player));
    }
}

void logout(map<string,int>& elo, map<string,bool>& conectado, string player) {
    if (elo.find(player) != elo.end())
        conectado[player] = false;
}

void play(map<string,int>& elo, map<string,bool>& conectado,
 ↪  set<pair<int,string>>& ranking, string winner, string loser) {
    if (elo.find(winner) == elo.end() or elo.find(loser) == elo.end() or not
     ↪  conectado[winner] or not conectado[loser]) cout << "player(s) not
     ↪  connected" << '\n';
    else {
        ranking.erase(make_pair(-elo[winner], winner));
        ranking.erase(make_pair(-elo[loser], loser));
        elo[winner] += 10;
        if (elo[loser] - 10 >= valor_inicial) {
            elo[loser] -= 10;
        }
        ranking.insert(make_pair(-elo[winner], winner));
        ranking.insert(make_pair(-elo[loser], loser));
    }
}

void get_elo(map<string,int>& elo, string player) {
    cout << player << ' ' << elo[player] << '\n';
}

int main () {
    map<string,int> elo;
    map<string,bool> conectado;
    set< pair<int,string> > ranking;
    string instruccion;
    while (cin >> instruccion) {
        if (instruccion == "LOGIN") {
            string player;
            cin >> player;
            login(elo, conectado, ranking, player);
        } else if (instruccion == "LOGOUT") {
            string player;
            cin >> player;
            logout(elo, conectado, player);
        } else if (instruccion == "PLAY") {
            string winner, loser;
            cin >> winner >> loser;
            play(elo, conectado, ranking, winner, loser);
        } else { // instruccion == "GET_ELO"
            string player;
            cin >> player;
            get_elo(elo, player);
```

```
            }
        }
        cout << '\n';
        cout << "RANKING" << '\n';
        for (auto it : ranking) {
            cout << it.second << ' ' << -it.first << '\n';
        }
    }
```