Soluciones OIFem 2020
Entrenamiento 6

Distinct numbers

Teoría

- Conjuntos desordenados

Solución

Este problema era una práctica con conjuntos desordenados. La solución consiste en insertar todos los números del input en un conjunto desordenado e imprimir su tamaño al final. La dificultad de este problema es principalmente el tiempo, ya que es algo escaso. Para resolver esto, basta con hacer entrada y salida rápida, además de usar `reserve()` (función explicada en los apuntes).

Código

C++

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

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

    int N, numero;
    cin >> N;
    unordered_set<int> conjunto;
    nums.reserve(7192);

    while (N--) {
        cin >> numero;
        conjunto.insert(numero);
    }

    cout << conjunto.size() << '\n';

    return 0;
}
```

Don't be Last

Teoría

- Mapas

Solución

Guardamos la producción total de cada vaca en un mapa, teniendo en cuenta que las vacas de las que no hay datos han producido 0 litros (y deben tenerse en cuenta). Proseguimos a encontrar la producción mínima.

Sabiendo la producción mínima, volvemos a iterar sobre el mapa, encontrando la segunda mínima y la cantidad de vacas que produjeron esta segunda, además del nombre de la vaca en caso de que sea solo 1 la cantidad.

En caso de un empate, imprimimos esto. Si no, imprimimos la vaca que produjo esta segunda mínima.

Código

C++

```cpp
#include <fstream>
#include <map>
#include <vector>
using namespace std;

int main() {
        ifstream fin;
        fin.open("notlast.in");
        ofstream fout;
        fout.open("notlast.out");
        int N, prod;
        string cow;
        map<string, int> cowProd;
        vector<string> cows = {"Bessie", "Elsie", "Daisy", "Gertie", "Annabelle",
        ↪  "Maggie", "Henrietta"};
        for (string cow: cows)
                cowProd[cow] = 0;
        fin >> N;
        while(N--) {
                fin >> cow >> prod;
                cowProd[cow] += prod;
        }
        fin.close();

        int produccionMinima = 1e9;
        for (auto it = cowProd.cbegin(); it != cowProd.cend(); it++) {
                produccionMinima = min(produccionMinima, it->second);
        }
```

```
        int segundaMinima = 1e9, cantidadSegunda = -1;
        string segundaVaca = "";
        for (auto it = cowProd.cbegin(); it != cowProd.cend(); it++) {
                if (it->second > produccionMinima) {
                        if (it->second < segundaMinima) {
                                segundaMinima = it->second;
                                cantidadSegunda = 1;
                                segundaVaca = it->first;
                        } else if (it->second == segundaMinima)
                                cantidadSegunda++;
                }
        }

        if (cantidadSegunda != 1)
                fout << "Tie\n";
        else
                fout << segundaVaca << '\n';
        fout.close();
    return 0;
}
```

Cities and States

Teoría

- Mapas
- Búsqueda binaria

Solución

Primero usamos un mapa como diccionario, o sea, para evitar estar comparando strings (operación con coste $O(longitud)$), convertimos todos los substrings que nos encontramos de longitud 2 a un número, comparando en $O(1)$. Guardamos entonces cada pareja (ciudad, estado) con sus respectivos códigos.

Ordenamos esta lista y hacemos un pase de izquierda a derecha. Al encontrarnos una nueva pareja, pasamos a la siguiente si su segundo elemento es mayor que el izquierdo, ya que de existir su inversa ya la habríamos contado.

Sabiendo que es una pareja nueva, seguimos iterando sobre la lista hasta encontrarnos con una pareja diferente, sabiendo así cuántas veces aparece nuestra pareja. Usamos búsqueda binaria para saber cuántas veces aparece su inversa y sumamos el producto de estos dos valores a la respuesta final.

Si los dos elementos de una pareja coinciden, significa que cualquier otra pareja con estos mismos elementos estará situada en el mismo estado y, por tanto, no sumará nada a la respuesta. Continuamos si es este el caso.

Extra: la función `cantidad()` puede cambiarse para ser más eficiente usando búsqueda binaria una segunda vez para encontrar el último índice donde la pareja es igual a `key` en vez de hacer ese pase lineal al final. ¿Sabrías hacer este cambio? La solución está escrita con el pase lineal porque el juez la acepta así y es más sencillo, pero en concursos es recomendable hacer la forma difícil, ya que te aseguras de que no se pase de tiempo.

Código

C++

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

int cantidad (vector<pair<int, int>> & parejas, pair<int, int> key) {
        int lo = 0, hi = (int) parejas.size()-1, mid, indice = -1;
        while(lo <= hi) {
                mid = lo + (hi-lo)/2;
                if (parejas[mid] >= key) {
                        indice = mid;
                        hi = mid-1;
                } else lo = mid+1;
        }
        if (indice == -1 || parejas[indice] != key) return 0;
        int i = indice;
        while(parejas[i] == key) {
                i++;
                if (i == (int) parejas.size()) break;
        }
        return i-indice;
}

long long int parejasEspeciales(vector<pair<int, int>> & parejas) {
    sort(parejas.begin(), parejas.end());
        long long int respuesta = 0, actual;
        int i = 0, primero, segundo;
        while (i < (int) parejas.size()) {
                if (parejas[i].first > parejas[i].second) {
                        // evitamos contar la misma pareja dos veces
                        i++;
                        continue;
                }
                actual = 0;
                primero = parejas[i].first;
                segundo = parejas[i].second;
                while(parejas[i].first == primero && parejas[i].second == segundo)
                ↪  {
                        i++;
                        actual++;
                        if (i == (int) parejas.size()) break;
                }
                if (primero == segundo) continue; // son del mismo estado -> no
                ↪  las contamos
                else respuesta += actual*cantidad(parejas, make_pair(segundo,
                ↪  primero));
        }
        return respuesta;
}
```

```cpp
int main() {
        ifstream fin;
        ofstream fout;
        fin.open("citystate.in");
        fout.open("citystate.out");
        int N;
        fin >> N;
        string city, state, cityStart;
        map<string, int> codeDict = map<string, int>();
        vector<pair<int, int>> parejas(N);
        for (int i = 0; i < N; i++) {
                fin >> city >> state;
                cityStart = city.substr(0, 2);
                if (codeDict.find(state) == codeDict.end())
                        codeDict[state] = (int) codeDict.size();
                if (codeDict.find(cityStart) == codeDict.end())
                        codeDict[cityStart] = (int) codeDict.size();
                parejas[i] = make_pair(codeDict[cityStart], codeDict[state]);
        }
        fout << parejasEspeciales(parejas) << '\n';
        fin.close();
        fout.close();
        return 0;
}
```

Juegos con Queta

Teoría

- Algoritmos voraces
- Pilas (stacks)

Solución

Primero, ordenamos las Quetas de izquierda a derecha, de forma que podamos procesarlas por orden de coordenada. Declaramos una pila de números, que contendrá en todo momento las Quetas que aún no se han aniquilado. Vamos de izquierda a derecha, procesando las Quetas una a una. Si la pila está vacía (no hay Quetas a la izquierda de la actual sin aniquilar), la Queta anterior se movía hacia el extremo izquierdo, o la Queta actual se mueve hacia la derecha, la añadiremos a la pila, ya que no hay posibilidades de que se destruya con otra vista anteriormente. Sin embargo, si ninguna de estas tres condiciones se cumple, significa que la Queta actual va hacia la izquierda y que la Queta más cercana de las ya vistas va hacia la derecha. Por lo tanto, estas dos Quetas se van a aniquilar. Quitamos entonces esta Queta anterior de la pila y no añadimos la actual.

Código

C++

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


int main() {
    int T, N, m1, m2, direccionActual;
    cin >> T;
    vector<pair<int, int>> quetas;
    stack<int> pila;
    while(T--) {
        cin >> N >> m1 >> m2;
        quetas.assign(N, pair<int, int>());
        for (int i = 0; i < N; i++) cin >> quetas[i].first;
        for (int i = 0; i < N; i++) cin >> quetas[i].second;
        sort(quetas.begin(), quetas.end());
        pila = stack<int>();
        for (int i = 0; i < N; i++) {
            direccionActual = quetas[i].second;
            if (pila.empty() || pila.top() == 1 || direccionActual == 2)
            ↪  pila.push(direccionActual);
            else pila.pop();
        }
        cout << pila.size() << '\n';
    } return 0;
}
```

Where Am I?

Teoría

- Conjuntos
- Método de bisección

Solución

Usamos el método de bisección para encontrar el mínimo K válido. Para comprobar si un K de prueba es válido, basta con usar un conjunto al que le insertamos todos los substrings de longitud K de prueba. Si no hay ningún substring repetido, ese K será válido (y si no, no).

Código

C++

```cpp
#include <fstream>
#include <set>
using namespace std;

bool posibleK(string & S, int K) {
        set<string> subs;
        string actual;
        for (int p = 0; p <= (int) S.length()-K; p++) {
                actual = S.substr(p, K);
                if (subs.find(actual) != subs.end()) return false;
                subs.insert(actual);
        }
        return true;
}

int minimoK(string & S) {
        int N = (int) S.length();
        int lo = 1, hi = N, mid, ans=N;
        while(lo <= hi) {
                mid = lo + (hi-lo)/2;
                if (posibleK(S, mid)) {
                        ans = mid;
                        hi = mid-1;
                } else lo = mid+1;
        }
        return ans;
}
```

```
int main() {
        ifstream fin;
        ofstream fout;
        fin.open("whereami.in");
        fout.open("whereami.out");
        int N;
        string S;
        fin >> N;
        fin >> S;
        fout << minimoK(S) << '\n';
        fin.close();
        fout.close();
        return 0;
}
```

Deciphering the Mayan Writing

Teoría

- Método de la ventana deslizante

Solución

En vez de guardar las dos strings al completo, usamos el método de la ventana deslizante. Primero, introducimos el "pattern" carácter a carácter, guardando en un vector la cantidad que contiene de cada carácter.

Para la palabra larga, introducimos primero la primera ventana de longitud igual al "pattern", guardándola en un vector similar al del "pattern". Con cada ventana incrementamos el contador si los vectores coinciden.

Vamos carácter a carácter borrando el carácter $i - N$ y añadiendo el nuevo a la ventana hasta llegar al final de la segunda palabra. Finalmente, imprimimos el contador.

Código

C++

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

int main() {
        ios::sync_with_stdio(false);
        cin.tie(NULL);
        int N, M, respuesta;
        vector<int> pattern(52, 0);
        vector<int> window(52, 0);
        string curSeq = "";
        cin >> N >> M;
        cin.ignore();
        char c;
        // construimos el vector del "pattern"
        for (int i = 0; i < N; i++) {
                cin.get(c);
                if (c >= 'a') {
                        pattern[c - 'a' + 26]++;
                }
                else {
                        pattern[c - 'A']++;
                }
        }
        cin.ignore();
```

```
        respuesta = 0;
        // recogemos los primeros N caracteres de la palabra larga
        for (int i = 0; i < N; i++) {
                cin.get(c);
                curSeq += c;
                if (c >= 'a') window[c - 'a' + 26]++;
                else window[c - 'A']++;
                if (pattern == window) respuesta++;
        }
        // recogemos los últimos M-N carácteres y usamos el método de la ventana
        ↪ deslizante
        for (int i = N; i < M; i++) {
                cin.get(c);
                curSeq += c;
                if (c >= 'a') window[c - 'a' + 26]++;
                else window[c - 'A']++;
                if (curSeq[0] >= 'a') window[curSeq[0] - 'a' + 26]--;
                else window[curSeq[0] - 'A']--;
                curSeq.erase(0, 1);
                if (window == pattern)
                        respuesta++;
        }
        cout << respuesta;
        return 0;
}
```

Team Queue

Teoría

- Colas
- Mapas

Solución

La solución consiste en aprovechar colas y mapas juntos. Representaremos la cola de equipos usando dos estructuras de datos:

- `queue<int> colaEquipos`- Esta cola guarda los índices (el índice de un grupo es el orden en el que lo metió el usuario en la entrada) de los equipos a la cola, sin tener en cuenta cuántas personas hay en cada equipo.
- `unordered_map<int, queue<int>> encoladasDelEquipo`- Este mapa guarda, para cada equipo a la cola, qué personas están a la cola de este, y en qué orden están dentro del equipo, facilitando el encolar y decolar usando una cola para guardar esta lista.

Cada vez que queramos encolar a alguien, lo encolaremos al final de su equipo si este ya está a la cola y, si este no está a la cola, lo añadiremos a la cola de equipos y pondremos a esta persona en cabeza de la cola.

Cada vez que queramos decolar a alguien, vamos al frente de la cola de equipos e imprimimos la persona al frente de ese primer equipo. Decolamos la cola de ese equipo dentro del mapa y comprobamos si esta era la única persona a la cola que formara parte de su equipo. Si es este el caso, decolamos también al equipo de la cola de equipos.

Código

C++

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

void reset(queue<int> & colaEquipos, unordered_map<int, int> & equipoDe,
 ↪  unordered_map<int, queue<int>> & encoladasDelEquipo) {
        colaEquipos = queue<int>();
        equipoDe.clear();
        equipoDe.reserve(7192);
        encoladasDelEquipo.clear();
        encoladasDelEquipo.reserve(7192);
}
```

```cpp
void encola(int idPersona, queue<int> & colaEquipos, unordered_map<int, int> &
    equipoDe, unordered_map<int, queue<int>> & encoladasDelEquipo) {
        if (encoladasDelEquipo[equipoDe[idPersona]].empty())
                colaEquipos.push(equipoDe[idPersona]);
        encoladasDelEquipo[equipoDe[idPersona]].push(idPersona);
}

void decola(queue<int> & colaEquipos, unordered_map<int, queue<int>> &
    encoladasDelEquipo) {
        cout << encoladasDelEquipo[colaEquipos.front()].front() << '\n';
        encoladasDelEquipo[colaEquipos.front()].pop();
        if (encoladasDelEquipo[colaEquipos.front()].empty())
                colaEquipos.pop();
}

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

        int numeroEquipos, tamanoEquipo, idPersona, caso = 1;
        queue<int> colaEquipos;
        unordered_map<int, int> equipoDe;
        unordered_map<int, queue<int>> encoladasDelEquipo;
        string instruccion;

        while(true) {
                cin >> numeroEquipos;
                if (!numeroEquipos)
                        break;

                cout << "Scenario #" << caso << '\n';
                reset(colaEquipos, equipoDe, encoladasDelEquipo);

                for (int i = 0; i < numeroEquipos; i++) {
                        encoladasDelEquipo[i] = queue<int>();
                        cin >> tamanoEquipo;
                        while(tamanoEquipo--) {
                                cin >> idPersona;
                                equipoDe[idPersona] = i;
                        }
                }
                while(true) {
                        cin >> instruccion;
                        if (instruccion[0] == 'E') {
                                cin >> idPersona;
                                encola(idPersona, colaEquipos, equipoDe,
                                    encoladasDelEquipo);
                        } else if (instruccion[0] == 'D') {
                                decola(colaEquipos, encoladasDelEquipo);
                        } else break;
                }
                caso++;
                cout << "\n";
        }
        return 0;
}
```

Borrando letras del SMS

Teoría

- Colas
- Algoritmos voraces

Solución

Para este problema, usaremos una cola para cada letra del alfabeto. Esta cola contendrá en orden de menor a mayor las posiciones en las que aparece esa letra en el mensaje.

Una vez construida este vector de 26 colas, crearemos el mensaje final letra a letra. Esto se hará de manera voraz, es decir, para cada posición en el mensaje final escogeremos la letra más cercana a la 'a' que nos permita terminar el mensaje (por ejemplo, si hay una 'a' en la penúltima posición pero nos faltan 3 letras, no podremos seleccionarla). Quitamos esta posición de la cola de esa letra y pasamos a la siguiente posición del mensaje final. De esta forma, obtendremos el mensaje final lexicográficamente menor en todos los casos.

Código

C++

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

int main() {
        ios::sync_with_stdio(false);
        cin.tie(NULL);
        int t, n, k;
        string mensaje;
        cin >> t;
        vector<queue<int>> colaAlfabeto;
        while (t--) {
                cin >> n >> k >> mensaje;
                colaAlfabeto.assign(26, queue<int>());
                // añadimos todos los caracteres del mensaje a sus respectivas colas
                for (int i = 0; i < n; ++i)
                        colaAlfabeto[mensaje[i] - 'a'].push(i);

                int posicionActual = 0;
                // vamos creando el mensaje final letra a letra
                for (int letrasSeleccionadas = 0; letrasSeleccionadas < k; letrasSeleccionadas++) {
                        // tratamos de escoger la letra lexicográficamente más pequeña
```

```
                        for (int letra = 0; letra < 26; letra++) {
                            // nos liberamos de los caracteres anteriores a la
                            ↪  posición actual
                            while (!colaAlfabeto[letra].empty() and
                            ↪  colaAlfabeto[letra].front() < posicionActual)
                                colaAlfabeto[letra].pop();
                            // no aparece esta letra más en el mensaje
                            ↪  original
                            if (colaAlfabeto[letra].empty())
                                continue;
                            int siguientePosicionDeLaLetra =
                            ↪  colaAlfabeto[letra].front();
                            if (n - siguientePosicionDeLaLetra >= k -
                            ↪  letrasSeleccionadas) {
                                // quedan suficientes letras para poder
                                ↪  rellenar el mensaje si escogemos esta
                                ↪  letra
                                // por lo que la escogemos de manera voraz
                                posicionActual =
                                ↪  siguientePosicionDeLaLetra;
                                colaAlfabeto[letra].pop();
                                cout << char('a' + letra);
                                break;
                            }
                        }
                    }
                    cout << '\n';
                }
            }
```