Texto, Carta

Descripción generada automáticamente

#include <iostream>

#include <unordered\_map>

#include <string>

#include <bits/stdc++.h>

using namespace std;

class TrieNode

{

public:

    unordered\_map<char, TrieNode \*> children;

    bool isEndOfWord;

    TrieNode()

    {

        isEndOfWord = false;

    }

};

// Clase para el Trie

class Trie

{

private:

    TrieNode \*root;

public:

    Trie()

    {

        root = new TrieNode();

    }

    TrieNode \*getTrieNode()

    {

        return root;

    }

    // Inserta una palabra en el Trie

    void insert(string word)

    {

        TrieNode \*current = root;

        for (char c : word)

        {

            if (current->children.find(c) == current->children.end())

            {

                current->children[c] = new TrieNode();

            }

            current = current->children[c];

        }

        current->isEndOfWord = true;

    }

    // Busca una palabra en el Trie

    bool search(string word)

    {

        TrieNode \*current = root;

        for (char c : word)

        {

            if (current->children.find(c) == current->children.end())

            {

                return false;

            }

            current = current->children[c];

        }

        return current->isEndOfWord;

    }

    int countTelephone(string word)

    {

        TrieNode \*current = root;

        for (char c : word)

        {

            if (current->children.find(c) == current->children.end())

            {

                return false;

            }

            current = current->children[c];

        }

        return current->children.size();

    }

    // Verifica si una palabra es un prefijo en el Trie

    bool startsWith(string prefix)

    {

        TrieNode \*current = root;

        for (char c : prefix)

        {

            if (current->children.find(c) == current->children.end())

            {

                return false;

            }

            current = current->children[c];

        }

        return true;

    }

};

int main()

{

    int t;

    do

    {

        cin >> t;

    } while (t < 0 || t > 50);

    vector<string> identificadores;

    int search1, search2;

    cin >> search1 >> search2;

    identificadores.push\_back(to\_string(search1));

    identificadores.push\_back(to\_string(search2));

    Trie trie;

    set<string> st;

    // Recorrer la cantidad de casos

    for (int i = 0; i < t; i++)

    {

        int n;

        // Validacion de la cantidad de lineas de numero de telefono

        do

        {

            cin >> n;

        } while (n < 0 || n > 5000);

        for (int i = 0; i < n; i++)

        {

            char add;

            int id1, id2, id3;

            // Validacion de que se ingrese la letra a

            do

            {

                cin >> add;

            } while (add != 'a');

            string telephone;

            // Validacion de que el numero de telefono no se repita

            do

            {

                cin >> id1 >> id2 >> id3;

                telephone = to\_string(id1) + to\_string(id2) + to\_string(id3);

            } while (st.find(telephone) != st.end());

            trie.insert(telephone);

        }

        // Mostrar la cantidad de numeros de telefono que contiene el identificador

        cout << trie.countTelephone(identificadores[i]) << endl;

    }

    return 0;

}

In.txt: es el ejemplo de prueba

2

911

985

4

a 911 124 345

a 925 625 888

a 911 452 639

a 987 321 134

5

a 911 334 265

a 985 123 340

a 920 567 440

a 914 845 245

a 981 745 346

Texto

Descripción generada automáticamente

2. Diagrama

Descripción generada automáticamente

#include <iostream>

#include <vector>

#include <queue>

#include <limits>

using namespace std;

// Estructura para representar un nodo y su distancia desde el origen

struct Nodo

{

    int indice;

    int distancia;

    bool operator<(const Nodo &otro) const

    {

        return distancia > otro.distancia;

    }

};

// Función auxiliar para agregar una arista a un grafo no dirigido

void agregarArista(vector<vector<pair<int, int>>> &grafo, int u, int v, int peso)

{

    grafo[u].push\_back({v, peso});

    grafo[v].push\_back({u, peso});

}

int sumarPesosRango(vector<vector<pair<int, int>>> &grafo, int inicio, int fin)

{

    int pesoTotal = 0;

    for (int i = inicio; i < fin; i++)

    {

        for (int j = 0; j < grafo[i].size(); j++)

        {

            int v = grafo[i][j].first;

            int peso = grafo[i][j].second;

            // Validacion para suma los vertices y que esten en el rango

            if (v >= inicio && v <= fin && i < v)

            {

                pesoTotal += peso;

            }

        }

    }

    return pesoTotal;

}

int multiplicarPesosRango(vector<vector<pair<int, int>>> &grafo, int inicio, int fin)

{

    int pesoTotal = 0;

    for (int i = inicio; i < fin; i++)

    {

        for (int j = 0; j < grafo[i].size(); j++)

        {

            int v = grafo[i][j].first;

            int peso = grafo[i][j].second;

            // Validacion para multiplicar los vertices y que esten en el rango

            if (v >= inicio && v <= fin && i < v)

            {

                if (pesoTotal == 0)

                {

                    pesoTotal += peso;

                }

                else

                {

                    pesoTotal \*= peso;

                }

            }

        }

    }

    return pesoTotal;

}

vector<int> menorPeso(vector<vector<pair<int, int>>> &grafo)

{

    int pesoMenor = 1000;

    vector<int> verticesPeso;

    for (int i = 0; i < grafo.size(); i++)

    {

        for (int j = 0; j < grafo[i].size(); j++)

        {

            int v = grafo[i][j].first;

            int peso = grafo[i][j].second;

            // Validacion para que no se repita un vertice desde el otro nodo

            if (i < v)

            {

                if (peso < pesoMenor)

                {

                    pesoMenor = peso;

                    verticesPeso = {i, v, peso};

                }

            }

        }

    }

    return verticesPeso;

}

int main()

{

    // Cantidad de nodos en el grafo

    int n = 6;

    // Crear el grafo

    vector<vector<pair<int, int>>> grafo(n);

    // Agregar las aristas al grafo

    for (int i = 0; i < n; i++)

    {

        int v1, v2, peso;

        cin >> v1 >> v2;

        // Validacion que este en el rango del 1-100

        do

        {

            cin >> peso;

        } while (peso < 1 || peso > 100);

        agregarArista(grafo, v1, v2, peso);

    }

    char letra;

    cout << "Ingrese una operacion:" << endl;

    do

    {

        cin >> letra;

    } while (letra != 's' && letra != 'm' && letra != 'c');

    if (letra == 's')

    {

        int inicio, fin;

        cin >> inicio >> fin;

        cout << "Peso Total sumado basado en rango es: " << sumarPesosRango(grafo, inicio, fin) << endl;

    }

    else if (letra == 'm')

    {

        int inicio, fin;

        cin >> inicio >> fin;

        cout << "Pesos Total multiplicado basado en rango es: " << multiplicarPesosRango(grafo, inicio, fin) << endl;

    }

    else if (letra == 'c')

    {

        vector<int> verticesMenorPeso = menorPeso(grafo);

        cout << "Los vertices con menor peso son : (" << verticesMenorPeso[0] << "," << verticesMenorPeso[1] << ") y el peso es " << verticesMenorPeso[2] << endl;

    }

}

Texto

Descripción generada automáticamente