Tanislav Alexia
Grupa 132

TEST

Laborator 1:

Exercitiul 1:

Pasul comun algoritmilor Selection Sort si Heapsort este selectarea elementului maxim (sau minim) din partea nesortata a listei si plasarea acestuia la pozitia corecta.

Exercitiul 2:

Algoritmul Heapsort se desfasoara in doua faze principale:

1. Construirea unui Max-Heap:
    - Porneste de la ultimul nod care are copii (non-leaf nodes) si aplica procedura de heapify pentru a transforma lista intr-un Max-Heap
    - Continua heapify pentru toate nodurile pana la radacina, asigurandu-se ca toata structura este un Max-Heap
2. Sortarea:
    - Se schimba primul element (maxim) cu ultimul element din lista curenta.
    - Se reduce dimensiunea heap-ului (excluderea elementului maxim actual din heap)
    - Se aplica procedura de heapify pentru radacina pentru a reface structura Max-Heap
    - Se repeta pasii de sortare pana cand toate elementele sunt sortate

Laborator 2:

Exercitiul 1:

Tanislav Alexia
Grupa 132

```
#include <iostream>

//lista dublu inlantuita

using namespace std;



struct DoublyNode {

   int data;

   DoublyNode* next;

   DoublyNode* prev;

};



void insert(DoublyNode*& head, int value) {

   DoublyNode* newNode = new DoublyNode{value, head, nullptr};

   if (head != nullptr) {

      head->prev = newNode;

   }

   head = newNode;

}
```

Tanislav Alexia
Grupa 132

```
void remove(DoublyNode*& head, int value) {

   DoublyNode* temp = head;



   while (temp != nullptr && temp->data != value) {

      temp = temp->next;

   }



   if (temp == nullptr) return;



   if (temp->prev != nullptr) {

      temp->prev->next = temp->next;

   } else {

      head = temp->next;

   }



   if (temp->next != nullptr) {

      temp->next->prev = temp->prev;

   }
```

```
    delete temp;
}
void display(DoublyNode* head) {
    DoublyNode* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " <-> ";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}

int main() {
    DoublyNode* head = nullptr;
    int x,c;
    cin>>x;
    while (x!=0){
        insert(head, x);
```

Tanislav Alexia
Grupa 132

```
        c =x;

        cin>>x;

    }

    display(head);



    remove(head, c);

    display(head);




    return 0;

}
```

Exercitiul 2:

Tanislav Alexia
Grupa 132

```
    Node* temp = head;
    while (temp != nullptr && temp->data != value) {
        temp = temp->next;
    }
    return temp;
}

void display(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}

int main() {
    Node* head = nullptr;
    int x;
    int c;
    cin>>x;
    while (x!=0) {
        insert(head, x);
        c = x;
        cin>>x;
    }
    display(head);

    remove(head, c);
    display(head);


    return 0;
}
```

```
> 4 -> 3 -> 2 -> NULL
> 3 -> 2 -> NULL

Program finished with exit code 0
ss ENTER to exit console.
```

Laborator 3:

Exercitiul 1:

Tanislav Alexia
Grupa 132

```cpp
#include <iostream>
#include <stack>
#include <string>


bool verificaParanteze(const std::string& expresie) {
    std::stack<char> stiva;
    for (char paranteza : expresie) {
        if (paranteza == '(' || paranteza == '[' || paranteza == '{') {
            stiva.push(paranteza);
        } else if (paranteza == ')' || paranteza == ']' || paranteza == '}') {

            if (stiva.empty() ||
                (paranteza == ')' && stiva.top() != '(') ||
                (paranteza == ']' && stiva.top() != '[') ||
                (paranteza == '}' && stiva.top() != '{')) {
                return false;
            }
            stiva.pop();
        }
    }
    return stiva.empty();
}

int main() {
    std::string expresie;
    std::cout << "Introduceti o expresie: ";
    std::cin >> expresie;

    if (verificaParanteze(expresie)) {
        std::cout << "parantezele sunt corecte.\n";
    } else {
        std::cout << "parantezele nu sunt corecte.\n";
    }
```

```
input
ntroduceti o expresie: ({)}
arantezele nu sunt corecte.


..Program finished with exit code 0
ress ENTER to exit console.
```

Exercitiul 2:

Tanislav Alexia
Grupa 132

Laborator 4:

main.cpp

```
    cin >> nr;
    for (int i = 0; i < nr; i++){
        cout << "cuvantul pe care il vrei in tabel: ";
        cin >> wordToInsert;
        insert(hashTable, wordToInsert);
    }

    display(hashTable);

    cout << "vrei sa cauti vreun cuvant?y/n \n";
    cin >> raspuns;
    if (tolower(raspuns) == 'y'){
        cout << "care e cuvantul: ";
        cin >> searchWord;
        if (search(hashTable, searchWord)){
            cout << searchWord << " gasit\n";
        }else{
```

input

```
nr de cuv pe care le vrei introduse in table: 9
cuvantul pe care il vrei in tabel: am
cuvantul pe care il vrei in tabel: Are
cuvantul pe care il vrei in tabel: Ana
cuvantul pe care il vrei in tabel: carte
cuvantul pe care il vrei in tabel: caiet
cuvantul pe care il vrei in tabel: cuminte
cuvantul pe care il vrei in tabel: inteligent
cuvantul pe care il vrei in tabel: inima
cuvantul pe care il vrei in tabel: imagine
Index a: (a, am) -> (A, Are) -> (A, Ana) -> NULL
Index b: NULL
Index c: (c, carte) -> (c, caiet) -> (c, cuminte) -> NULL
Index d: NULL
Index e: NULL
Index f: NULL
Index g: NULL
Index h: NULL
Index i: (i, inteligent) -> (i, inima) -> (i, imagine) -> NULL
Index j: NULL
Index k: NULL
Index l: NULL
Index m: NULL
Index n: NULL
Index o: NULL
Index p: NULL
Index q: NULL
Index r: NULL
Index s: NULL
Index t: NULL
Index u: NULL
```

Laborator 5:

Exercitiul 1:

Pasii pentru problema 5.1:

Tanislav Alexia
Grupa 132

1. Identificarea radacinii: se selecteaza elementul median al vectorului ca radacina a arborelui

2. Construirea subarborilor:

   - Elementele din partea stanga a medianului devin subarborele stang
   - Elementele din partea dreapta a medianului devin subarborele drept

3. Repetarea recursiva: se aplica acelasi proces pentru subarborii stang si drept pana cand vectorul este gol

Exericitiul 2:

Pasii algoritmului de la problema 5.2:

1. Convertirea arborilor in liste ordonate:

   - Se realizeaza o traversare in ordine (inorder) pentru ambii arbori pentru a obtine doua liste ordonate

2. Interclasarea listelor:

   - Se interclaseaza cele doua liste ordonate intr-o singura lista ordonata

3. Construirea unui nou arbore echilibrat:

   - Se utilizeaza algoritmul pentru crearea unui arbore binar de cautare echilibrat dintr-o lista sortata pentru a construi noul arbore din lista interclasata

Laborator 6:

Tanislav Alexia
Grupa 132

Exercitiul 1:

Kruskal functioneaza pe un graf neorientat si sortat pe margini in ordinea costului, fata de Prim care creste arborele de la un nod initial. Kruskal adauga margini in ordine crescatoare a greutatii, evitand ciclurile, pana cand arborele este complet, fata de Prim care adauga margini cu cel mai mic cost care extinde arborele la noduri noi, repetand procesul pana cand arborele este complet.

Exercitiul 2:

Pasii algoritmului lui Prim:

1. Initializare:

- Se alege un nod initial si se marcheaza ca fiind inclus in MST
- Se initializeaza o structura de date pentru a gestiona marginile disponibile

2. Constructia arborelui:

- Pentru nodul curent, se adauga toate marginile care il conecteaza la nodurile neincluse in MST in structura de date
- Se selecteaza marginea cu cel mai mic cost din structura de date si se adauga la MST
- Se marcheaza nodul conectat de margine ca fiind inclus in MST

3. Repetare:

- Se repeta procesul pana cand toate nodurile sunt incluse in MST

Tanislav Alexia
Grupa 132

TEME

Laboratorul 1:

Metoda selectiei:

```
#include <iostream>

#include <vector>

using namespace std;

void selectionSort(vector<int>& v) {

   int n = v.size();

   for (int i = 0; i < n - 1; ++i) {

      int minIdx = i;

      for (int j = i + 1; j < n; ++j) {

         if (v[j] < v[minIdx]) {

            minIdx = j;

         }

      }

      swap(v[i], v[minIdx]);
```

```
    }

}

int main() {

    vector<int> v = {64, 34, 25, 12, 22, 11, 90, 34};

    selectionSort(v);

    cout << "vector sortat: ";

    for (int x : v) {

        cout << x << " ";

    }

    cout << endl;

    return 0;

}
```

Metoda HeapSort:

```
#include <iostream>

#include <vector>
```

Tanislav Alexia
Grupa 132

```
using namespace std;



void heapify(std::vector<int>& v, int n, int i) {

   int largest = i; // cel mai mare e radacina

   int left = 2 * i + 1;

   int right = 2 * i + 2;



   // daca copilul stg > radacina

   if (left < n && v[left] > v[largest]) {

      largest = left;

   }



   // daca copilul dr > radacina

   if (right < n && v[right] > v[largest]) {

      largest = right;

   }



   // cel mai mare nu e radacina
```

```
    if (largest != i) {

        std::swap(v[i], v[largest]);



        // heapify pe restu

        heapify(v, n, largest);

    }

}



void heapSort(std::vector<int>& v) {

    int n = v.size();



    // reface vectorul, face heap

    for (int i = n / 2 - 1; i >= 0; --i) {

        heapify(v, n, i);

    }



    // extrage elem cu elem din heap

    for (int i = n - 1; i >= 0; --i) {
```

```
        // Move current root to end

        std::swap(v[0], v[i]);



        // max heapify pe heapul redus

        heapify(v, i, 0);

    }

}



int main() {

    vector<int> v = {23,45,55,1,48,90};

    heapSort(v);

    for (int i : v) {

        cout << i << " ";

    }

    cout << endl;

    return 0;

}
```

Tanislav Alexia
Grupa 132

Laborator 2:

```
#include <iostream>

//lista simplu inlantuita

using namespace std;



struct Node {

   int data;

   Node* next;

};



void insert(Node*& head, int value) {

   Node* newNode = new Node{value, head};

   head = newNode;

}



void remove(Node*& head, int value) {

   Node* temp = head;
```

```
    Node* prev = nullptr;



    while (temp != nullptr && temp->data != value) {

        prev = temp;

        temp = temp->next;

    }



    if (temp == nullptr) return;



    if (prev == nullptr) {

        head = temp->next;

    } else {

        prev->next = temp->next;

    }



    delete temp;

}
```

Tanislav Alexia
Grupa 132

```
Node* search(Node* head, int value) {

   Node* temp = head;

   while (temp != nullptr && temp->data != value) {

      temp = temp->next;

   }

   return temp;

}



void display(Node* head) {

   Node* temp = head;

   while (temp != nullptr) {

      cout << temp->data << " -> ";

      temp = temp->next;

   }

   cout << "NULL" << endl;

}



int main() {
```

```
    Node* head = nullptr;

    insert(head, 1);

    insert(head, 2);

    insert(head, 3);

    display(head);



    remove(head, 2);

    display(head);



    Node* found = search(head, 3);

    if (found) {

        cout << "Gasit: " << found->data << endl;

    } else {

        cout << "nu exista" << endl;

    }



    return 0;

}
```

Tanislav Alexia
Grupa 132

```
#include <iostream>

//lista dublu inlantuita

using namespace std;



struct DoublyNode {

   int data;

   DoublyNode* next;

   DoublyNode* prev;

};



void insert(DoublyNode*& head, int value) {

   DoublyNode* newNode = new DoublyNode{value, head, nullptr};
```

```
    if (head != nullptr) {
        head->prev = newNode;
    }
    head = newNode;
}

void remove(DoublyNode*& head, int value) {
    DoublyNode* temp = head;

    while (temp != nullptr && temp->data != value) {
        temp = temp->next;
    }

    if (temp == nullptr) return;

    if (temp->prev != nullptr) {
        temp->prev->next = temp->next;
    } else {
```

```
        head = temp->next;
    }

    if (temp->next != nullptr) {
        temp->next->prev = temp->prev;
    }

    delete temp;
}

DoublyNode* search(DoublyNode* head, int value) {
    DoublyNode* temp = head;
    while (temp != nullptr && temp->data != value) {
        temp = temp->next;
    }
    return temp;
}
```

Tanislav Alexia
Grupa 132

```
void display(DoublyNode* head) {

    DoublyNode* temp = head;

    while (temp != nullptr) {

        cout << temp->data << " <-> ";

        temp = temp->next;

    }

    cout << "NULL" << endl;

}



int main() {

    DoublyNode* head = nullptr;

    insert(head, 1);

    insert(head, 2);

    insert(head, 3);

    display(head);



    remove(head, 2);

    display(head);
```

Tanislav Alexia
Grupa 132

```
    DoublyNode* found = search(head, 3);

    if (found) {

        cout << "Gasit: " << found->data << endl;

    } else {

        cout << "nu exista" << endl;

    }



    return 0;

}
```

```
#include <iostream>

//lista circulara

using namespace std;
```

Tanislav Alexia
Grupa 132

```
struct CircularNode {
    int data;
    CircularNode* next;
};


void insert(CircularNode*& head, int value) {
    CircularNode* newNode = new CircularNode{value, nullptr};
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
    } else {
        CircularNode* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
```

```
    }

}



void remove(CircularNode*& head, int value) {

    if (head == nullptr) return;



    CircularNode* temp = head;

    CircularNode* prev = nullptr;



    do {

        if (temp->data == value) {

            if (prev == nullptr) {

                CircularNode* last = head;

                while (last->next != head) {

                    last = last->next;

                }

                last->next = head->next;

                CircularNode* toDelete = head;
```

```
            head = head->next;
            delete toDelete;
        } else {
            prev->next = temp->next;
            delete temp;
        }
        return;
    }
    prev = temp;
    temp = temp->next;
  } while (temp != head);
}

CircularNode* search(CircularNode* head, int value) {
  if (head == nullptr) return nullptr;

  CircularNode* temp = head;
  do {
```

```
        if (temp->data == value) {

            return temp;

        }

        temp = temp->next;

    } while (temp != head);



    return nullptr;

}



void display(CircularNode* head) {

    if (head == nullptr) return;



    CircularNode* temp = head;

    do {

        cout << temp->data << " -> ";

        temp = temp->next;

    } while (temp != head);

    cout << "(head)" << endl;
```

```
}



int main() {

    CircularNode* head = nullptr;

    insert(head, 1);

    insert(head, 2);

    insert(head, 3);

    display(head);



    remove(head, 2);

    display(head);



    CircularNode* found = search(head, 3);

    if (found) {

        cout << "Gasit: " << found->data << endl;

    } else {

        cout << "nu exista" << endl;

    }
```

Tanislav Alexia
Grupa 132

```
    return 0;
}
```

Laborator 3:

Exercitiul 3.1:

```
#include <iostream>
#include <stack>
#include <string>


bool verificaParanteze(const std::string& expresie) {
    std::stack<char> stiva;
    for (char paranteza : expresie) {
        if (paranteza == '(' || paranteza == '[' || paranteza == '{') {
            stiva.push(paranteza);
        } else if (paranteza == ')' || paranteza == ']' || paranteza == '}') {
```

```
            if (stiva.empty() ||
                (paranteza == ')' && stiva.top() != '(') ||
                (paranteza == ']' && stiva.top() != '[') ||
                (paranteza == '}' && stiva.top() != '{')) {
                return false;
            }
            stiva.pop();
        }
    }
    return stiva.empty();
}


int main() {
    std::string expresie;
    std::cout << "Introduceti o expresie: ";
    std::cin >> expresie;

    if (verificaParanteze(expresie)) {
```

```
        std::cout << "parantezele sunt corecte.\n";

    } else {

        std::cout << "parantezele nu sunt corecte.\n";

    }



    return 0;

}
```

Exercitiul 3.2:

```
#include <iostream>

#include <stack>

#include <queue>

#include <vector>



using namespace std;



vector<int> f(vector<int>& v) {
```

Tanislav Alexia
Grupa 132

```
    int n = v.size();

    vector<int> vr(n, -1);  // vector deafult cu elem -1

    stack<int> st;  // stiva pt indicii elem



    for (int i = 0; i < n; i++) {

        while (!st.empty() && v[i] > v[st.top()]) {

            vr[st.top()] = v[i];  // modif rez pt indicele din vf stivei

            st.pop();  // pop indice

        }

        st.push(i);  // punem indice elem curent pe stiva

    }



    return vr;

}




int main() {

    int nr;
```

Tanislav Alexia
Grupa 132

```
   cout << "scrie nr de elem pt vector: ";

   cin >> nr;

     vector<int> v(nr, 0);

   for (int i = 0; i < nr; i++){

      cout << "elem: ";

      cin >> v[i];

   }

   vector<int> vr = f(v);  // vectorul final cu rezultate

   queue<int> q;    // coada cu rezultate din vector



   for (int elem : vr) {

      q.push(elem);

   }



   while (!q.empty()) {      //af

      cout << q.front() << " ";

      q.pop();

   }
```

Tanislav Alexia
Grupa 132

```
    return 0;
}
```

Laboratorul 4:

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

struct Node {
    char key;
    string value;
    Node* next;

    Node(char k, string v) : key(k), value(v), next(nullptr) {}
};
```

Tanislav Alexia
Grupa 132

```
int hashFunction(char key) {

    return tolower(key) - 'a';

}



bool search(vector<Node*>& hashTable, string value) {

    char key = value[0];

    int hashIndex = hashFunction(key);

    Node* temp = hashTable[hashIndex];



    while (temp != nullptr) {

        if (temp->value == value) {

            return true;

        }

        temp = temp->next;

    }



    return false;

}
```

Tanislav Alexia
Grupa 132

```
void insert(vector<Node*>& hashTable, string value) {
    char key = value[0];
    int hashIndex = hashFunction(key);
    Node* newNode = new Node(key, value);

    if (!search(hashTable, value)){
        if (hashTable[hashIndex] == nullptr) {
            hashTable[hashIndex] = newNode;
        } else {
            Node* temp = hashTable[hashIndex];
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }
```

```
}



void remove(vector<Node*>& hashTable, string wantedValue) {

   char key = wantedValue[0];

   int hashIndex = hashFunction(key);

   Node* temp = hashTable[hashIndex];

   Node* prev = nullptr;



   while (temp != nullptr && temp->value != wantedValue) {

      prev = temp;

      temp = temp->next;

   }



   if (temp == nullptr) {

      cout << "cuvantul nu exista" << endl;

      return;

   }
```

```
    if (prev == nullptr) {
        hashTable[hashIndex] = temp->next;
    } else {
        prev->next = temp->next;
    }

    delete temp;
}

void display(const vector<Node*>& hashTable) {
    for (int i = 0; i < hashTable.size(); ++i) {
        cout << "Index " << char(i + 'a') << ": ";
        Node* temp = hashTable[i];
        while (temp != nullptr) {
            cout << "(" << temp->key << ", " << temp->value << ") -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
```

```
    }

}



int main() {

    int size = 26;

    int nr;

    string wordToInsert, searchWord, removeWord;

    char raspuns;

    vector<Node*> hashTable(size, nullptr);

    cout << "nr de cuv pe care le vrei introduse in table: ";

    cin >> nr;

    for (int i = 0; i < nr; i++){

        cout << "cuvantul pe care il vrei in tabel: ";

        cin >> wordToInsert;

        insert(hashTable, wordToInsert);

    }



    display(hashTable);
```

Tanislav Alexia
Grupa 132

```
    cout << "vrei sa cauti vreun cuvant?y/n \n";
    cin >> raspuns;
    if (tolower(raspuns) == 'y'){
        cout << "care e cuvantul: ";
        cin >> searchWord;
        if (search(hashTable, searchWord)){
            cout << searchWord << " gasit\n";
        }else{
            cout << searchWord << " nu a fost gasit\n";
        }
    }

    cout << "vrei sa stergi vreun cuvant? y/n ";
    cin >> raspuns;
    if (tolower(raspuns) == 'y'){
        cout << "care e cuvantul: ";
        cin >> removeWord;
```

```
        remove(hashTable, removeWord);

    }



    display(hashTable);



    return 0;

}
```

Laboratorul 5:

Exercitiul 5.1:

```
#include <iostream>

#include <vector>



using namespace std;



struct TreeNode {

    int data;
```

```
    TreeNode* left;

    TreeNode* right;



    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}

};



TreeNode* buildBalancedBST(const vector<int>& sortedArray, int start, int end) {

    if (start > end) {

        return nullptr;

    }



    int mid = (start + end) / 2;

    TreeNode* root = new TreeNode(sortedArray[mid]);



    root->left = buildBalancedBST(sortedArray, start, mid - 1);

    root->right = buildBalancedBST(sortedArray, mid + 1, end);



    return root;
```

```
}



void printInOrder(TreeNode* root) {
    if (root == nullptr) {
        return;
    }

    printInOrder(root->left);
    cout << root->data << " ";
    printInOrder(root->right);
}

int main() {
    vector<int> sortedArray = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    TreeNode* root = buildBalancedBST(sortedArray, 0, sortedArray.size() - 1);

    cout << "BST: ";
```

```
    printInOrder(root);

    cout << endl;



    return 0;

}
```

Exercitiul 5.2:

```
TreeNode* mergeTwoBSTs(TreeNode* root1, TreeNode* root2) {

    std::vector<int> inorder1, inorder2;

    inorderTraversal(root1, inorder1);

    inorderTraversal(root2, inorder2);



    std::vector<int> mergedInorder = mergeSortedArrays(inorder1, inorder2);



    return sortedArrayToBST(mergedInorder, 0, mergedInorder.size() - 1);

}



// Functie pentru a afisa un BST (inordine)
```

Tanislav Alexia
Grupa 132

```
void printInOrder(TreeNode* root) {

   if (root == nullptr) return;

   printInOrder(root->left);

   std::cout << root->val << " ";

   printInOrder(root->right);

}
```

Laborator 6:

Prim:

```
// Prim

#include <bits/stdc++.h>

#include <iostream>

#include <vector>

using namespace std;



#define V 5  //nr vf



//functie pt a gasi min key value care nu sunt in mst
```

Tanislav Alexia
Grupa 132

```
int minKey(int key[], bool mstSet[])

{

	int min = INT_MAX, min_index;



	for (int v = 0; v < V; v++)

		if (mstSet[v] == false && key[v] < min)

			min = key[v], min_index = v;



	return min_index;

}



void printMST(int parent[], int graph[V][V])

{

	cout << "Margini\t Adancime\n";

	for (int i = 1; i < V; i++)

		cout << parent[i] << " - " << i << " \t"

			<< graph[i][parent[i]] << " \n";

}
```

Tanislav Alexia
Grupa 132

```
// construieste si af mst prin matrice adiacenta/lista de adiacenta

void primMST(int graph[V][V])

{

	int parent[V];

	int key[V];

	bool mstSet[V];

	for (int i = 0; i < V; i++)

		key[i] = INT_MAX, mstSet[i] = false;



	key[0] = 0;

	parent[0] = -1;

	for (int count = 0; count < V - 1; count++) {

		int u = minKey(key, mstSet);

		mstSet[u] = true;

		for (int v = 0; v < V; v++)

			if (graph[u][v] && mstSet[v] == false

				&& graph[u][v] < key[v])
```

```
                                parent[v] = u, key[v] = graph[u][v];

        }



        printMST(parent, graph);

}



int main()

{

        int graph[V][V] = { { 0, 2, 0, 6, 0 },

                                                { 2, 0, 3, 8, 5 },

                                                { 0, 3, 0, 0, 7 },

                                                { 6, 8, 0, 0, 9 },

                                                { 0, 5, 7, 9, 0 } };



        primMST(graph);



        return 0;

}
```

Tanislav Alexia
Grupa 132

Kruskal:

```
//kruskal alg

#include <bits/stdc++.h>

using namespace std;



class DSU {

	int* parent;

	int* rank;



public:

	DSU(int n)

	{

		parent = new int[n];

		rank = new int[n];



		for (int i = 0; i < n; i++) {

			parent[i] = -1;
```

```
                rank[i] = 1;

        }

}



int find(int i)

{

        if (parent[i] == -1)

                return i;



        return parent[i] = find(parent[i]);

}



void unite(int x, int y)

{

        int s1 = find(x);

        int s2 = find(y);



        if (s1 != s2) {
```

```
                        if (rank[s1] < rank[s2]) {

                                parent[s1] = s2;

                        }

                        else if (rank[s1] > rank[s2]) {

                                parent[s2] = s1;

                        }

                        else {

                                parent[s2] = s1;

                                rank[s1] += 1;

                        }

                }

        }

};



class Graph {

        vector<vector<int> > edgelist;

        int V;
```

```
public:

    Graph(int V) { this->V = V; }



    void addEdge(int x, int y, int w)

    {

        edgelist.push_back({ w, x, y });

    }



    void kruskals_mst()

    {



        sort(edgelist.begin(), edgelist.end());



        DSU s(V);

        int ans = 0;

        cout << "margini: "

            << endl;

        for (auto edge : edgelist) {
```

```
                int w = edge[0];

                int x = edge[1];

                int y = edge[2];




                if (s.find(x) != s.find(y)) {

                        s.unite(x, y);

                        ans += w;

                        cout << x << " -- " << y << " == " << w

                                << endl;

                }

        }

        cout << "MST: " << ans;

    }

};



int main()

{
```

```
    Graph g(4);

    g.addEdge(0, 1, 10);

    g.addEdge(1, 3, 15);

    g.addEdge(2, 3, 4);

    g.addEdge(2, 0, 6);

    g.addEdge(0, 3, 5);



    g.kruskals_mst();



    return 0;

}
```