### **EVALUARE TEME LABORATOARE STRUCTURI DE DATE**

**03.06.2024**

#### **LABORATOR 1**

1. Descrieți pasul comun dintre algoritmii Selection Sort și Heapsort.
2. Descrieți pe scurt (dar complet), pașii algoritmului Heapsort.

**LABORATOR 2**

1. Scrieți codul pentru ștergerea corpului unei liste circulare simplu/dublu înlănțuite
2. Rulați un program pentru crearea unei liste simplu înlănțuite cu date citite de la tastatură, până la citirea numărului 0. Se va șterge primul nod al acestei liste. În document va fi introdus un screenshot care dovedește rezolvarea cerinței

#### **LABORATOR 3**

1. Se va furniza un screenshot pentru rularea programului pentru intrarea ( { ) }.
2. Descrieți pașii algoritmului pentru rezolvarea problemei 3.2.

#### **LABORATOR 4**

1. Se va furniza un screenshot pentru testarea programului din laboratorul 4 (crearea unui dicționar), pentru o intrare conținând cuvintele: Ana, are, Ana, carte, caiet, cuminte, inteligent, inimă, imagine.

#### **LABORATOR 5**

1. Scrieți pașii algoritmului pentru rezolvarea problemei 5.1  
    *(crearea unui arbore binar de căutare echilibrat pornind de la un vector sortat).*
2. Scrieți pașii algoritmului pentru rezolvarea problemei 5.2  
    *(interclasarea a doi arbori binari de căutare echilibrați).*

#### **LABORATOR 6**

1. Precizați care este diferența dintre construcția arborelui de cost minim folosind algoritmul Kruskal și algoritmul Prim.
2. Scrieți pașii algoritmului Prim.

**Atenție:**

*La sfârșitul fișierului va fi introdus codul pentru fiecare dintre problemele cerute la temă, pentru a fi evaluate.*

## ***LABORATOR 1***

***1. Descrieți pasul comun dintre algoritmii Selection Sort și Heapsort.*** *În ambele algoritmi, la fiecare iterație se* ***selectează un element*** *(cel mai mic, respectiv cel mai mare) și se plasează la poziția „finală” a zonei nesortate, astfel crescând treptat zona sortată.*

***2. Descrieți pe scurt (dar complet) pașii algoritmului Heapsort.***

1. ***Build‐heap****: transformă vectorul de intrare într-un max‐heap.*
2. *Pentru i = n–1 pana la 1:*
   * *Schimbă rădăcina (elementul maxim) cu arr[i].*
   * *Reduce „dimensiunea” heap‐ului cu 1.*
   * *„Heapify” la rădăcină pentru a restabili proprietatea de max‐heap.*
3. *La final, vectorul este sortat crescător.*

## ***LABORATOR 2***

### ***1. Ștergerea corpului unei liste circulare (simplu și dublu înlănțuite)***

*#include <iostream>*

*using namespace std;*

*// Nod de listă circulară simplu înlănțuită*

*struct NodeS {*

*int data;*

*NodeS\* next;*

*NodeS(int v): data(v), next(this) {}*

*};*

*// Șterge toate nodurile dintr-o listă circulară simplu înlănțuită*

*void clearCircularS(NodeS\*& head) {*

*if (!head) return;*

*NodeS\* cur = head->next;*

*while (cur != head) {*

*NodeS\* tmp = cur;*

*cur = cur->next;*

*delete tmp;*

*}*

*delete head;*

*head = nullptr;*

*}*

*// Nod de listă circulară dublu înlănțuită*

*struct NodeD {*

*int data;*

*NodeD\* next;*

*NodeD\* prev;*

*NodeD(int v): data(v), next(this), prev(this) {}*

*};*

*// Șterge toate nodurile dintr-o listă circulară dublu înlănțuită*

*void clearCircularD(NodeD\*& head) {*

*if (!head) return;*

*NodeD\* cur = head->next;*

*while (cur != head) {*

*NodeD\* tmp = cur;*

*cur = cur->next;*

*delete tmp;*

*}*

*delete head;*

*head = nullptr;*

*}*

### ***2. Crearea unei liste simplu înlănțuite, citire până la 0 și ștergerea primului nod***

*#include <iostream>*

*using namespace std;*

*struct Node {*

*int data;*

*Node\* next;*

*Node(int v): data(v), next(nullptr) {}*

*};*

*int main() {*

*Node\* head = nullptr;*

*Node\* tail = nullptr;*

*int x;*

*cout << "Introduceti valori (0 = stop): ";*

*while (cin >> x && x != 0) {*

*Node\* nou = new Node(x);*

*if (!head) head = tail = nou;*

*else {*

*tail->next = nou;*

*tail = nou;*

*}*

*}*

*// Ștergerea primului nod*

*if (head) {*

*Node\* tmp = head;*

*head = head->next;*

*delete tmp;*

*}*

*// Afișare listă rezultată*

*cout << "Lista dupa stergerea primului nod: ";*

*for (Node\* p = head; p; p = p->next)*

*cout << p->data << " ";*

*cout << "\n";*

*// Curățare rest de noduri*

*while (head) {*

*Node\* tmp = head;*

*head = head->next;*

*delete tmp;*

*}*

*return 0;*

*}*

## ***LABORATOR 3***

***1.*** *(Se va furniza screenshot pentru rularea programului cu intrarea ({)}.)*

***2. Descrieți pașii algoritmului pentru rezolvarea problemei 3.2.*** *(Verificarea echilibrării parantezelor folosind un stivă.)*

1. *Inițializează o stivă goală.*
2. *Pentru fiecare caracter c din șir:*
   * *Dacă c e (, { sau [, o* ***push*** *în stivă.*
   * *Dacă c e ), } sau ]:*
     1. *Dacă stiva e goală →* ***neechilibrat****, se oprește.*
     2. *Verifică vârful stivei: dacă tipul parantezei nu se potrivește (de ex. ( cu ]) →* ***neechilibrat****, se oprește.*
     3. *Altfel,* ***pop*** *din stivă.*
3. *După parcurgere, dacă stiva e goală →* ***echilibrat****, altfel →* ***neechilibrat****.*

## ***LABORATOR 4***

***Se va furniza un screenshot pentru testarea programului (dicționar) cu intrarea:  
 Ana, are, Ana, carte, caiet, cuminte, inteligent, inimă, imagine.***

*#include <iostream>*

*#include <map>*

*#include <string>*

*#include <sstream>*

*using namespace std;*

*int main() {*

*string line;*

*cout << "Introduceti sir de cuvinte separate prin spatiu:\n";*

*getline(cin, line);*

*istringstream in(line);*

*map<string,int> freq;*

*string w;*

*while (in >> w) {*

*// transformă la lowercase și elimină semne diacritice dacă e cazul*

*for (auto& c : w) c = tolower(c);*

*freq[w]++;*

*}*

*cout << "Dicționar (cuvânt : frecvență):\n";*

*for (auto& [cuv, cnt] : freq)*

*cout << cuv << " : " << cnt << "\n";*

*return 0;*

*}*

## ***LABORATOR 5***

***1. Pașii algoritmului pentru problema 5.1*** *(crearea unui arbore binar de căutare echilibrat pornind de la un vector sortat)*

1. *Dacă vectorul e gol → return nullptr.*
2. *Alege indexul de mijloc m = (l + r) / 2.*
3. *Creează un nod cu key = v[m].*
4. *Recursiv, construiește subarborele stâng din [l..m-1].*
5. *Recursiv, construiește subarborele drept din [m+1..r].*

***2. Pașii algoritmului pentru problema 5.2*** *(interclasarea a doi arbori BST echilibrați)*

1. *Pentru fiecare arbore, obține vectorul in‐ordine (sorted) prin recursivitate.*
2. *Interclasează cei doi vectori sorted într‐un singur vector sorted.*
3. *Aplică algoritmul de la 5.1 pe vectorul rezultat pentru a obține un BST echilibrat.*

## ***LABORATOR 6***

***1. Diferența dintre Kruskal și Prim***

* ***Kruskal*** *construiește arborele de cost minim (MST)* ***global****, alegând muchiile cele mai ieftine din întreg graf-ul (folosind Union-Find pentru a evita ciclurile).*
* ***Prim*** *pornește de la un nod inițial și extinde arborele* ***local****, adăugând la frontieră cea mai ieftină muchie care leagă arborele de restul nodurilor.*

***2. Pașii algoritmului Prim***

1. *Alege un nod arbitr­ar și adaugă-l în MST.*
2. *Marchează toate muchiile incident­e ca „frontieră”.*
3. *Repetă până când MST are n–1 muchii:*
   * *Din frontieră, selectează muchia de cost minim (u,v) cu u în MST și v în afara MST.*
   * *Adaugă v în MST și muchia (u,v) în soluție.*
   * *Adaugă în frontieră toate muchiile noi incidente în v.*

***Atenție:*** *La sfârșitul fișierului sursă C++ va fi introdus codul pentru fiecare problemă, astfel încât toate implementările să fie evaluate împreună.*

*Mai jos găsești toate implementările C++ necesare pentru fiecare cerință. Fiecare secțiune poate fi compilată și rulată independent (copiază blocul corespunzător într-un fișier .cpp și compilează cu g++).*

### ***LABORATOR 1 – Heapsort***

*// lab1\_heapsort.cpp*

*#include <iostream>*

*#include <vector>*

*using namespace std;*

*void heapify(vector<int>& a, int n, int i) {*

*int largest = i;*

*int l = 2\*i + 1, r = 2\*i + 2;*

*if (l < n && a[l] > a[largest]) largest = l;*

*if (r < n && a[r] > a[largest]) largest = r;*

*if (largest != i) {*

*swap(a[i], a[largest]);*

*heapify(a, n, largest);*

*}*

*}*

*void heapsort(vector<int>& a) {*

*int n = a.size();*

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

*heapify(a, n, i);*

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

*swap(a[0], a[i]);*

*heapify(a, i, 0);*

*}*

*}*

*int main() {*

*vector<int> a;*

*int x;*

*cout << "Introduceti elementele (EOF pentru terminare):\n";*

*while (cin >> x) a.push\_back(x);*

*heapsort(a);*

*cout << "Vector sortat: ";*

*for (int v : a) cout << v << " ";*

*cout << "\n";*

*return 0;*

*}*

### ***LABORATOR 2***

#### ***2.1 Ștergerea corpului unei liste circulare simplu/dublu înlănțuite***

*// lab2\_clear\_circular.cpp*

*#include <iostream>*

*using namespace std;*

*// Simplu înlănțuită*

*struct NodeS {*

*int data;*

*NodeS\* next;*

*NodeS(int v): data(v), next(this) {}*

*};*

*void clearCircularS(NodeS\*& head) {*

*if (!head) return;*

*NodeS\* cur = head->next;*

*while (cur != head) {*

*NodeS\* tmp = cur;*

*cur = cur->next;*

*delete tmp;*

*}*

*delete head;*

*head = nullptr;*

*}*

*// Dublu înlănțuită*

*struct NodeD {*

*int data;*

*NodeD\* next;*

*NodeD\* prev;*

*NodeD(int v): data(v), next(this), prev(this) {}*

*};*

*void clearCircularD(NodeD\*& head) {*

*if (!head) return;*

*NodeD\* cur = head->next;*

*while (cur != head) {*

*NodeD\* tmp = cur;*

*cur = cur->next;*

*delete tmp;*

*}*

*delete head;*

*head = nullptr;*

*}*

*int main() {*

*// Exemplu simplu de folosire:*

*NodeS\* headS = new NodeS(1);*

*headS->next = headS; // un singur nod*

*clearCircularS(headS);*

*cout << "Lista circulara simplu inlantuaita curatata.\n";*

*NodeD\* headD = new NodeD(2);*

*headD->next = headD;*

*headD->prev = headD;*

*clearCircularD(headD);*

*cout << "Lista circulara dublu inlantuaita curatata.\n";*

*return 0;*

*}*

#### ***2.2 Listă simplu înlănțuită: citire până la 0 și ștergerea primului nod***

*// lab2\_singly\_list.cpp*

*#include <iostream>*

*using namespace std;*

*struct Node {*

*int data;*

*Node\* next;*

*Node(int v): data(v), next(nullptr) {}*

*};*

*int main() {*

*Node\* head = nullptr;*

*Node\* tail = nullptr;*

*int x;*

*cout << "Introduceti valori (0 = stop): ";*

*while (cin >> x && x != 0) {*

*Node\* nou = new Node(x);*

*if (!head) head = tail = nou;*

*else {*

*tail->next = nou;*

*tail = nou;*

*}*

*}*

*// Ștergere primul nod*

*if (head) {*

*Node\* tmp = head;*

*head = head->next;*

*delete tmp;*

*}*

*// Afișare*

*cout << "Lista dupa stergerea primului nod: ";*

*for (Node\* p = head; p; p = p->next) cout << p->data << " ";*

*cout << "\n";*

*// Curățare rest*

*while (head) {*

*Node\* tmp = head;*

*head = head->next;*

*delete tmp;*

*}*

*return 0;*

*}*

### ***LABORATOR 3 – Verificare echilibrare paranteze***

*// lab3\_brackets.cpp*

*#include <iostream>*

*#include <stack>*

*#include <string>*

*using namespace std;*

*bool isMatching(char open, char close) {*

*return (open=='(' && close==')') ||*

*(open=='{' && close=='}') ||*

*(open=='[' && close==']');*

*}*

*bool checkBalanced(const string& s) {*

*stack<char> st;*

*for (char c : s) {*

*if (c=='('||c=='{'||c=='[') {*

*st.push(c);*

*} else if (c==')'||c=='}'||c==']') {*

*if (st.empty() || !isMatching(st.top(), c)) return false;*

*st.pop();*

*}*

*}*

*return st.empty();*

*}*

*int main() {*

*string s;*

*cout << "Introduceti secventa de paranteze: ";*

*getline(cin, s);*

*cout << (checkBalanced(s) ? "ECHILIBRAT\n" : "NEECHILIBRAT\n");*

*return 0;*

*}*

### ***LABORATOR 4 – Dicționar de frecvență***

*// lab4\_dictionary.cpp*

*#include <iostream>*

*#include <map>*

*#include <string>*

*#include <sstream>*

*#include <algorithm>*

*using namespace std;*

*int main() {*

*string line;*

*cout << "Introduceti cuvintele separate prin spatiu:\n";*

*getline(cin, line);*

*istringstream in(line);*

*map<string,int> freq;*

*string w;*

*while (in >> w) {*

*// lowercase*

*transform(w.begin(), w.end(), w.begin(), ::tolower);*

*freq[w]++;*

*}*

*cout << "Dicționar (cuvant : frecventa):\n";*

*for (auto& [cuv, cnt] : freq)*

*cout << cuv << " : " << cnt << "\n";*

*return 0;*

*}*

### ***LABORATOR 5 – Arbore BST echilibrat și interclasare***

*// lab5\_bst\_merge.cpp*

*#include <iostream>*

*#include <vector>*

*using namespace std;*

*// Nod BST*

*struct Node {*

*int key;*

*Node\* left;*

*Node\* right;*

*Node(int k): key(k), left(nullptr), right(nullptr) {}*

*};*

*// 5.1 Construiește BST echilibrat din vector sortat*

*Node\* buildBalanced(const vector<int>& a, int l, int r) {*

*if (l > r) return nullptr;*

*int m = (l + r) / 2;*

*Node\* root = new Node(a[m]);*

*root->left = buildBalanced(a, l, m - 1);*

*root->right = buildBalanced(a, m + 1, r);*

*return root;*

*}*

*// In-order traversal în vector*

*void inorder(Node\* root, vector<int>& out) {*

*if (!root) return;*

*inorder(root->left, out);*

*out.push\_back(root->key);*

*inorder(root->right, out);*

*}*

*// 5.2 Interclasare a doi BST echilibrați*

*Node\* mergeBST(Node\* A, Node\* B) {*

*vector<int> va, vb, merged;*

*inorder(A, va);*

*inorder(B, vb);*

*// merge sorted vectors*

*int i=0, j=0;*

*while (i<va.size() && j<vb.size()) {*

*if (va[i] < vb[j]) merged.push\_back(va[i++]);*

*else merged.push\_back(vb[j++]);*

*}*

*while (i<va.size()) merged.push\_back(va[i++]);*

*while (j<vb.size()) merged.push\_back(vb[j++]);*

*// build balanced BST*

*return buildBalanced(merged, 0, (int)merged.size()-1);*

*}*

*// Traversare pre-order (pentru test)*

*void preorder(Node\* root) {*

*if (!root) return;*

*cout << root->key << " ";*

*preorder(root->left);*

*preorder(root->right);*

*}*

*int main() {*

*// Exemplu: vector sortat*

*vector<int> v1 = {1,3,5,7};*

*vector<int> v2 = {2,4,6,8,9};*

*Node\* A = buildBalanced(v1, 0, v1.size()-1);*

*Node\* B = buildBalanced(v2, 0, v2.size()-1);*

*Node\* M = mergeBST(A, B);*

*cout << "Preorder merged BST: ";*

*preorder(M);*

*cout << "\n";*

*return 0;*

*}*

### ***LABORATOR 6 – Algoritmul Prim (MST)***

*// lab6\_prim.cpp*

*#include <iostream>*

*#include <vector>*

*#include <limits>*

*using namespace std;*

*int main() {*

*int n;*

*cout << "Numar noduri: ";*

*cin >> n;*

*vector<vector<int>> cost(n, vector<int>(n));*

*cout << "Matrice cost (0 daca nu exista muchie):\n";*

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

*for (int j = 0; j < n; ++j)*

*cin >> cost[i][j];*

*vector<bool> inMST(n,false);*

*vector<int> dist(n, numeric\_limits<int>::max());*

*vector<int> parent(n,-1);*

*dist[0] = 0;*

*for (int k = 0; k < n; ++k) {*

*int u = -1;*

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

*if (!inMST[i] && (u==-1 || dist[i] < dist[u]))*

*u = i;*

*inMST[u] = true;*

*// update frontieră*

*for (int v = 0; v < n; ++v) {*

*if (cost[u][v] > 0 && !inMST[v] && cost[u][v] < dist[v]) {*

*dist[v] = cost[u][v];*

*parent[v] = u;*

*}*

*}*

*}*

*cout << "Muchii MST (u - v : cost):\n";*

*for (int v = 1; v < n; ++v)*

*cout << parent[v] << " - " << v << " : " << cost[parent[v]][v] << "\n";*

*return 0;*

*}*