OLIMPIADA KOMBËTARE E INFORMATIKËS
NË ARSIMIN E MESËM TË LARTË

Faza e dytë

Klasa 11 **18 janar 2025**

Udhëzime për nxënësin:

- Olimpiada fillon në orën 10:00 dhe mbaron në orën 13:00.
- Testi përmban 5 pyetje.
- Për të zgjidhur secilin ushtrim nxënësi mund të përdorë gjuhën e programimit C++.

Për përdorim nga komisioni i vlerësimit

Pyetja	**1**	**2**	**3**	**4**	**5**
	7 pikë	**7 pikë**	**10 pikë**	**13 pikë**	**13 pikë**
Pikët e fituara					

Totali i pikëve të fituara ☐

KOMISIONI I VLERËSIMIT

1.....

2.....

1. Shkruani një program që përdor një algoritëm të kërkimit binar për të gjetur elementin çift më të vogël, i cili është më i madh se një numër i caktuar në një vektor të renditur në rendin zbritës. **7 pikë**

Zgjidhje:

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

int findSmallestEven(vector<int>& arr, int x) {
    int left = 0, right = arr.size() - 1;
    int result = -1; // Fillimisht nuk është gjetur

    while (left <= right) {
        int mid = (left + right) / 2;

        // Kontrollo nëse elementi është çift dhe më i madh se x
        if (arr[mid] > x && arr[mid] % 2 == 0) {
            result = arr[mid];
            right = mid - 1; // Shiko më tej në të majtë
        } else if (arr[mid] <= x) {
            left = mid + 1; // Shiko më tej në të djathtë
        } else {
            right = mid - 1; // Shiko në të majtë
        }
    }

    return result;
}
```

```
int main() {
    vector<int> arr = {20, 18, 15, 14, 12, 10, 8, 6, 4, 2}; // Renditur zbritës
    int x;
    cout << "Shkruani numrin: ";
    cin >> x;

    int result = findSmallestEven(arr, x);
    if (result != -1) {
        cout << "Elementi çift më i vogël që është më i madh se " << x << " është: " << result << endl;
    } else {
        cout << "Nuk ka element çift më të madh se " << x << endl;
    }

    return 0;
}
```

2. Shkruani një program që implementon një pemë binare dhe mundëson operacione si shtimi dhe kërkimi i dy elementëve të caktuar në këtë pemë. Pasi të gjenden, të fshihen këta elementë nga pema binare. **7pikë**

Zgjidhje:

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

// Struktura e nyjës së pemës binare
struct Node {
    int value;
    Node* left;
    Node* right;

    Node(int val) : value(val), left(nullptr), right(nullptr) {}
};
```

```
// Funksioni për të shtuar një element në pemë
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);
    }

    if (value < root->value) {
        root->left = insert(root->left, value);
    } else if (value > root->value) {
        root->right = insert(root->right, value);
    }

    return root;
}

// Funksioni për të kërkuar një element në pemë
bool search(Node* root, int value) {
    if (root == nullptr) {
        return false;
    }

    if (root->value == value) {
        return true;
    } else if (value < root->value) {
        return search(root->left, value);
    } else {
        return search(root->right, value);
    }
}
```

```
// Funksioni për të gjetur nyjën minimale (përdoret për fshirje)
Node* findMin(Node* root) {
    while (root->left != nullptr) {
        root = root->left;
    }
    return root;
}

// Funksioni për të fshirë një element nga pema
Node* deleteNode(Node* root, int value) {
    if (root == nullptr) {
        return root;
    }

    if (value < root->value) {
        root->left = deleteNode(root->left, value);
    } else if (value > root->value) {
        root->right = deleteNode(root->right, value);
    } else {
        // Rastet kur gjejmë nyjën për fshirje
        if (root->left == nullptr) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (root->right == nullptr) {
            Node* temp = root->left;
            delete root;
            return temp;
        }
```

```
        // Rast kur nyja ka dy fëmijë
        Node* temp = findMin(root->right);
        root->value = temp->value;
        root->right = deleteNode(root->right, temp->value);
    }

    return root;
}

// Funksioni për të printuar pemën (In-Order Traversal)
void inOrderTraversal(Node* root) {
    if (root != nullptr) {
        inOrderTraversal(root->left);
        cout << root->value << " ";
        inOrderTraversal(root->right);
    }
}

int main() {
    Node* root = nullptr;

    // Shtimi i elementëve
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);
```

```
    cout << "Pema fillestare (In-Order Traversal): ";
    inOrderTraversal(root);
    cout << endl;

    // Kërkimi për elementët
    int value1 = 20, value2 = 70;
    cout << "Kërko " << value1 << ": " << (search(root, value1) ? "Gjetur" : "Jo gjetur") << endl;
    cout << "Kërko " << value2 << ": " << (search(root, value2) ? "Gjetur" : "Jo gjetur") << endl;

    // Fshirja e elementëve
    root = deleteNode(root, value1);
    root = deleteNode(root, value2);

    cout << "Pema pas fshirjes (In-Order Traversal): ";
    inOrderTraversal(root);
    cout << endl;

    return 0;
}
```

3. Shkruani një program që përdor një strukturë të dhënash graf dhe implementon një algoritëm të kërkimit në gjerësi (BFS) për të gjetur të gjitha lidhjet e mundshme midis dy nodeve të caktuar. **10 pikë**

Zgjidhje:

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

// Funksioni për të shtuar një lidhje në graf
void addEdge(vector<int> adj[], int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u); // Grafi është jo i drejtuar
}

// Funksioni për BFS
void BFS(vector<int> adj[], int start, int target, int V) {
    vector<bool> visited(V, false); // Përdoret për të mbajtur gjurmët e nyjeve të vizituara
    queue<int> q;
    vector<int> parent(V, -1); // Për të ndjekur lidhjet

    visited[start] = true;
    q.push(start);

    while (!q.empty()) {
        int node = q.front();
        q.pop();

        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                parent[neighbor] = node;
                q.push(neighbor);
```

```
            if (neighbor == target) { // Ndalu kur gjendet nyja target
                vector<int> path;
                for (int v = target; v != -1; v = parent[v]) {
                    path.push_back(v);
                }
                cout << "Rruga më e shkurtër është: ";
                for (int i = path.size() - 1; i >= 0; i--) {
                    cout << path[i] << (i == 0 ? "\n" : " -> ");
                }
                return;
            }
        }
    }
}

    cout << "Nuk ka rrugë midis " << start << " dhe " << target << endl;
}

int main() {
    int V = 6; // Numri i nyjeve në graf
    vector<int> adj[V];

    // Ndërtojmë graf
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 2);
    addEdge(adj, 1, 3);
    addEdge(adj, 1, 4);
    addEdge(adj, 2, 4);
    addEdge(adj, 3, 5);
```

```
    addEdge(adj, 4, 5);

    int start = 0, target = 5;
    cout << "Kërkimi BFS nga nyja " << start << " te nyja " << target << ":\n";
    BFS(adj, start, target, V);

    return 0;
}
```

4. Shkruani një program që implementon një sistem menaxhimi të produkteve në një dyqan online, ku përdoruesi mund të shtojë produkte, t'i rendisë sipas çmimit në rendin rritës dhe t'i filtrojë në bazë të gamës së çmimeve ose sasisë në gjendje. Programi duhet të shfaqë listën e produkteve të filtruar dhe renditur. **13 pikë**

Zgjidhje:

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

// Struktura për produktin
struct Product {
    string name;
    double price;
    int quantity;

    Product(string n, double p, int q) : name(n), price(p), quantity(q) {}
};
```

```
// Funksioni për të shtuar një produkt
void addProduct(vector<Product>& products, string name, double price, int quantity) {
    products.push_back(Product(name, price, quantity));
}

// Funksioni për të renditur produktet sipas çmimit në rendin rritës
void sortProducts(vector<Product>& products) {
    sort(products.begin(), products.end(), [](Product& a, Product& b) {
        return a.price < b.price;
    });
}

// Funksioni për të filtruar produktet
void filterProducts(vector<Product>& products, double minPrice, double maxPrice, int minQuantity) {
    cout << "Produktet e filtruara:\n";
    for (auto& product : products) {
        if (product.price >= minPrice && product.price <= maxPrice && product.quantity >= minQuantity) {
            cout << "Emri: " << product.name << ", Çmimi: " << product.price
                 << ", Sasia: " << product.quantity << endl;
        }
    }
}

int main() {
    vector<Product> products;

    // Shtimi i disa produkteve
    addProduct(products, "Laptop", 1000.50, 10);
    addProduct(products, "Mouse", 25.99, 50);
    addProduct(products, "Keyboard", 45.89, 30);
```

```
    addProduct(products, "Monitor", 150.00, 20);


    // Renditja e produkteve
    sortProducts(products);
    cout << "Produktet e renditura sipas çmimit:\n";
    for (auto& product : products) {
        cout << "Emri: " << product.name << ", Çmimi: " << product.price
             << ", Sasia: " << product.quantity << endl;
    }


    // Filtrimi i produkteve
    double minPrice = 30, maxPrice = 200;
    int minQuantity = 20;
    filterProducts(products, minPrice, maxPrice, minQuantity);


    return 0;
}
```

5. Shkruani një program që implementon algoritmin e Bellman-Ford për gjetjen e rrugës më të shkurtër nga një nënstacion energjie në një rrjet të shpërndarjes së energjisë elektrike. Programi duhet të trajtojë mundësinë e cikleve negative, të identifikojë nëse ka cikle negative në rrjet dhe të ofrojë rrugën më të shkurtër të mundshme për shpërndarjen e energjisë nga një nënstacion në destinacionin e dëshiruar. **13 pikë**

Zgjidhje:

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


// Struktura për lidhjen (brinjën) në graf
```

```
struct Edge {
    int src, dest, weight;
};

// Funksioni për algoritmin Bellman-Ford
void bellmanFord(int V, int E, vector<Edge>& edges, int start) {
    vector<int> distance(V, numeric_limits<int>::max());
    distance[start] = 0;

    // Relakso të gjitha brinjët (V - 1) herë
    for (int i = 1; i <= V - 1; i++) {
        for (auto& edge : edges) {
            int u = edge.src;
            int v = edge.dest;
            int weight = edge.weight;

            if (distance[u] != numeric_limits<int>::max() && distance[u] + weight < distance[v]) {
                distance[v] = distance[u] + weight;
            }
        }
    }

    // Kontrollo për cikle negative
    for (auto& edge : edges) {
        int u = edge.src;
        int v = edge.dest;
        int weight = edge.weight;

        if (distance[u] != numeric_limits<int>::max() && distance[u] + weight < distance[v]) {
            cout << "Grafi përmban cikle negative!" << endl;
```

```
            return;
        }
    }

    // Shfaq distancat më të shkurtra
    cout << "Distancat më të shkurtra nga nyja burimore " << start << " janë:" << endl;
    for (int i = 0; i < V; i++) {
        if (distance[i] == numeric_limits<int>::max()) {
            cout << "Nyja " << i << ": Pa rrugë" << endl;
        } else {
            cout << "Nyja " << i << ": " << distance[i] << endl;
        }
    }
}

int main() {
    int V = 5; // Numri i nyjeve
    int E = 8; // Numri i brinjëve
    vector<Edge> edges;

    // Shto lidhjet në graf
    edges.push_back({0, 1, -1});
    edges.push_back({0, 2, 4});
    edges.push_back({1, 2, 3});
    edges.push_back({1, 3, 2});
    edges.push_back({1, 4, 2});
    edges.push_back({3, 2, 5});
    edges.push_back({3, 1, 1});
    edges.push_back({4, 3, -3});
```

```
    int start = 0; // Nyja burimore

    bellmanFord(V, E, edges, start);


    return 0;
}
```