OLIMPIADA KOMBËTARE E INFORMATIKËS
NË ARSIMIN E MESËM TË LARTË

Faza e dytë

Klasa 12 **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**
	10 pikë	**10 pikë**	**10 pikë**	**10 pikë**	**10 pikë**
Pikët e fituara					

Totali i pikëve të fituara ☐

KOMISIONI I VLERËSIMIT

1.....

2.....

1. Shkruani një program që përdor kthimin (rekursionin) për të gjetur të gjitha mundësitë për të vendosur M mbretëresha në një tavolinë shahu NxN, në mënyrë që asnjëra prej mbretëreshave të mos jetë në të njëjtin rresht, kolonë ose diagonale me njëra-tjetrën. **10 pikë**

Zgjidhje:

```
#include <iostream>

#include <vector>

using namespace std;

// Funksioni për të kontrolluar nëse një mbretëreshë mund të vendoset në rreshtin 'row' dhe kolonën 'col'

bool isSafe(int row, int col, const vector<int>& board, int N) {

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

        // Kontrollo nëse ndodhet në të njëjtën rresht, kolonë ose diagonale

        if (board[i] == row || abs(board[i] - row) == abs(i - col)) {

            return false;

        }

    }

    return true;

}



// Funksioni rekursiv për të gjetur të gjitha mundësitë për vendosjen e mbretëreshave

void solveNQueensUtil(int col, vector<int>& board, int N, vector<vector<int>>& solutions) {

    if (col == N) {

        // Kur arrihet fundi, regjistro zgjidhjen e mundshme

        solutions.push_back(board);

        return;

    }
```

```
    for (int row = 0; row < N; row++) {
        if (isSafe(row, col, board, N)) {
            // Vendos mbretëreshën në pozitat e mundshme
            board[col] = row;
            solveNQueensUtil(col + 1, board, N, solutions);
            // Kthehemi (backtrack) për të provuar mundësi të tjera
            board[col] = -1;
        }
    }
}

// Funksioni kryesor për të ndihmuar në gjetjen e mundësive për të vendosur M mbretëresha
void solveNQueens(int N) {
    vector<int> board(N, -1); // Tabella NxN, ku -1 do të thotë se nuk ka mbretëreshë
    vector<vector<int>> solutions; // Lista e zgjidhjeve

    solveNQueensUtil(0, board, N, solutions); // Fillon nga kolona 0

    // Shfaqim të gjitha zgjidhjet
    if (solutions.empty()) {
        cout << "Nuk ka zgjidhje." << endl;
    } else {
        for (const auto& solution : solutions) {
            for (int i = 0; i < N; i++) {
```

```
            for (int j = 0; j < N; j++) {
                if (solution[j] == i) cout << "Q ";  // Shfaqim mbretëreshën si 'Q'
                else cout << ". ";  // Pjesa e boshë e tabelës
            }
            cout << endl;
        }
        cout << endl;
    }
  }
}

int main() {
  int N;
  cout << "Jepni dimensionin e tabelës (NxN): ";
  cin >> N;

  solveNQueens(N);
  return 0;
}
```

2. Shkruani një program që implementon **KMP (Knuth-Morris-Pratt) String Matching Algorithm** për kërkimin e një substring-u në një tekst të dhënë. **10 pikë**

Zgjidhje:

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

// Funksioni për krijimin e tabelës LPS
void computeLPSArray(string pattern, vector<int>& lps) {
   int len = 0; // Gjatësia e prefix që është gjithashtu suffix
   lps[0] = 0;  // LPS për pozitën 0 është gjithmonë 0
   int i = 1;

   // Ndërtojmë LPS array
   while (i < pattern.length()) {
      if (pattern[i] == pattern[len]) {
         len++;
         lps[i] = len;
         i++;
      } else {
         if (len != 0) {
            len = lps[len - 1];  // Përdorim vlerën nga tabela LPS
         } else {
            lps[i] = 0;
            i++;
         }
      }
   }
```

```
}

// Funksioni KMP për kërkimin e substring-ut në tekst
void KMPSearch(string text, string pattern) {
   int n = text.length();
   int m = pattern.length();

   // Tabela LPS për pattern-in
   vector<int> lps(m);

   // Krijo tabelën LPS
   computeLPSArray(pattern, lps);

   int i = 0; // Indeksi për tekstin
   int j = 0; // Indeksi për pattern-in

   // Pjesa kryesore e algoritmit KMP
   while (i < n) {
      if (pattern[j] == text[i]) {
         i++;
         j++;
      }
      // Nëse pattern është gjetur
      if (j == m) {
         cout << "Pattern gjetur në pozitat " << i - j << " deri në " << i - 1 << endl;
         j = lps[j - 1];  // Përdorim tabelën LPS për të lëvizur përpara
      }
      // Kur nuk përputhet dhe kemi një match paraprak
      else if (i < n && pattern[j] != text[i]) {
         if (j != 0) {
            j = lps[j - 1];  // Përdorim LPS për të bërë skip
```

```
        } else {
            i++;
        }
      }
   }
}
int main() {
   string text = "ABABDABACDABABCABAB";
   string pattern = "ABABCABAB";
   KMPSearch(text, pattern);
   return 0;
}
```

3. Shkruani një program i cili llogarit fluksin minimal për të optimizuar menaxhimin e trafikut në një qytet. Krijoni një model të rrjetit urban, ku nodet janë rrugët dhe lidhjet janë kryqëzimet. Kërkohet të gjendet rruga më e shpejtë për automjetet që kalojnë nga një hyrje në një dalje, duke optimizuar trafikun dhe minimizuar vonesat. **10 pikë**

Zgjidhje:

```
#include <iostream>
#include <vector>
#include <climits>
#include <queue>

using namespace std;

// Krijimi i një strukture për të mbajtur grafët
class Graph {
public:
   int V; // Numri i vrimave
   vector<vector<pair<int, int>>> adj; // Lista e fqinjeve (graf i drejtuar)

   Graph(int V); // Konstruktor për të krijuar një graf me V vrima
```

```
    void addEdge(int u, int v, int weight); // Shto një lidhje nga u -> v me peshë
    void dijkstra(int src); // Algoritmi i Dijkstra-s për gjetjen e rrugës më të shpejtë
};

// Konstruktor për të inicializuar graf-in
Graph::Graph(int V) {
    this->V = V;
    adj.resize(V);
}

// Funksioni për të shtuar një lidhje në graf
void Graph::addEdge(int u, int v, int weight) {
    adj[u].push_back(make_pair(v, weight)); // Shto lidhje nga u në v me peshë
}

// Algoritmi i Dijkstra-s për gjetjen e rrugës më të shpejtë
void Graph::dijkstra(int src) {
    vector<int> dist(V, INT_MAX); // Vektor që mban distancat më të shpejta
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq; // Min-heap për të mbajtur nodet dhe distancat e tyre

    dist[src] = 0;
    pq.push(make_pair(0, src)); // Shto burimin me distancën 0

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        // Vizito fqinjtë e nodit u
        for (auto neighbor : adj[u]) {
```

```
            int v = neighbor.first;
            int weight = neighbor.second;

            // Nëse gjejmë një rrugë më të shpejtë për në v
            if (dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
                pq.push(make_pair(dist[v], v)); // Shto në min-heap me distancën e re
            }
        }
    }

    // Shfaq rezultatet (distancat minimale për çdo nod)
    cout << "Distanca minimale nga burimi " << src << ":\n";
    for (int i = 0; i < V; i++) {
        if (dist[i] == INT_MAX) {
            cout << "Nuk është e arritshme për " << i << "\n";
        } else {
            cout << "Për " << i << " është " << dist[i] << " minuta\n";
        }
    }
}

int main() {
    Graph g(5); // Krijo graf me 5 vrima (rrugë)

    // Shto lidhje (rrugë dhe kryqëzime me pesha)
    g.addEdge(0, 1, 5); // A -> B, 5 minuta
    g.addEdge(0, 2, 3); // A -> C, 3 minuta
    g.addEdge(1, 3, 2); // B -> D, 2 minuta
    g.addEdge(2, 3, 1); // C -> D, 1 minutë
```

```
    g.addEdge(3, 4, 4); // D -> E, 4 minuta

    // Gjej rrugën më të shpejtë nga burimi A (0) në çdo nod tjetër

    g.dijkstra(0);

    return 0;

}
```

4. Një kompani fluturimesh mund të përdorë një algoritëm të optimizuar për menaxhimin e rezervimeve të biletave. Kjo përfshin përdorimin e algoritmeve sorting për të renditur biletat sipas datës, dhe algoritme searching për të gjetur biletat e disponueshme që plotësojnë kriteret e kërkesave të përdoruesit. Programi duhet të mundësojë kërkime të shpejta të biletave dhe optimizimin e renditjes së tyre në mënyrë që të mund të përballojë një sasi të madhe të të dhënave. **10 pikë**

Zgjidhje:

```
#include <iostream>

#include <vector>

#include <algorithm>

#include <ctime>


using namespace std;


// Struktura për të ruajtur informacionin e biletës

struct Ticket {

    string flightDate;  // Data e fluturimit në formatin "YYYY-MM-DD"

    int ticketID;       // ID e biletës

    bool available;     // Nëse bileta është e disponueshme


    // Krijojmë një funksion për krahasimin e biletave sipas datës

    bool operator<(const Ticket& other) const {

        return flightDate < other.flightDate;

    }

};


// Funksioni për të kërkuar biletat e disponueshme
```

```
int binarySearch(const vector<Ticket>& tickets, const string& targetDate) {
    int left = 0, right = tickets.size() - 1;

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

        if (tickets[mid].flightDate == targetDate && tickets[mid].available)
            return mid;  // Kemi gjetur biletën e disponueshme

        if (tickets[mid].flightDate < targetDate)
            left = mid + 1;
        else
            right = mid - 1;
    }

    return -1;  // Nuk ka bileta të disponueshme për këtë datë
}

int main() {
    // Krijojmë një listë biletash
    vector<Ticket> tickets = {
        {"2025-03-12", 1, true},
        {"2025-01-15", 2, false},
        {"2025-05-09", 3, true},
        {"2025-03-12", 4, true},
        {"2025-02-20", 5, true}
    };

    // Rendi biletat sipas datës
    sort(tickets.begin(), tickets.end());
```

```
    // Shfaqim biletat pas renditjes
    cout << "Biletat e renditura sipas dates:\n";
    for (const auto& ticket : tickets) {
        cout << "TicketID: " << ticket.ticketID << ", Date: " << ticket.flightDate
             << ", Available: " << (ticket.available ? "Yes" : "No") << endl;
    }

    // Kërkojmë biletat e disponueshme për një datë të caktuar
    string targetDate = "2025-03-12";
    int result = binarySearch(tickets, targetDate);

    if (result != -1) {
        cout << "Bileta e disponueshme për datën " << targetDate << " është gjetur. TicketID: "
             << tickets[result].ticketID << endl;
    } else {
        cout << "Nuk ka bileta të disponueshme për datën " << targetDate << endl;
    }

    return 0;
}
```

5. Shkruani një program që menaxhon informacionin mbi pajisje elektronike, duke përfshirë kompaninë prodhuese, modelin dhe vitin e prodhimit. Programi duhet të mundësojë regjistrimin e pajisjeve elektronike, krahasimin e tyre përmes operatorëve të mbivendosur për të krahasuar pajisjet elektronike sipas kompanisë prodhuese dhe modelit. **10 pikë**

Zgjidhje:

```
#include <iostream>
#include <vector>
#include <string>
```

```
using namespace std;

// Struktura për pajisjet elektronike
struct ElectronicDevice {
    string company; // Kompania prodhuese
    string model;   // Modeli i pajisjes
    int year;       // Viti i prodhimit

    // Konstruktori
    ElectronicDevice(string c, string m, int y) : company(c), model(m), year(y) {}

    // Mbivendosja e operatorit < për krahasimin sipas kompanisë dhe modelit
    bool operator<(const ElectronicDevice& other) const {
        if (company == other.company) {
            return model < other.model;  // Krahasimi sipas modelit
        }
        return company < other.company;  // Krahasimi sipas kompanisë
    }

    // Mbivendosja e operatorit == për krahasimin e pajisjeve
    bool operator==(const ElectronicDevice& other) const {
        return company == other.company && model == other.model;
    }

    // Mbivendosja e operatorit > për krahasimin sipas kompanisë dhe modelit
    bool operator>(const ElectronicDevice& other) const {
        if (company == other.company) {
            return model > other.model;  // Krahasimi sipas modelit
        }
        return company > other.company;  // Krahasimi sipas kompanisë
```

```
    }
};

// Funksioni për të shtuar pajisje të reja
void addDevice(vector<ElectronicDevice>& devices, const string& company, const string& model, int year) {
    devices.push_back(ElectronicDevice(company, model, year));
}

// Funksioni për të shfaqur pajisjet
void displayDevices(const vector<ElectronicDevice>& devices) {
    for (const auto& device : devices) {
        cout << "Company: " << device.company << ", Model: " << device.model << ", Year: " << device.year << endl;
    }
}

int main() {
    vector<ElectronicDevice> devices;  // Lista e pajisjeve

    // Shtojmë disa pajisje
    addDevice(devices, "Apple", "iPhone 13", 2021);
    addDevice(devices, "Samsung", "Galaxy S21", 2021);
    addDevice(devices, "Apple", "iPad Pro", 2020);
    addDevice(devices, "Sony", "PlayStation 5", 2020);

    // Shfaqim pajisjet para renditjes
    cout << "Pajisjet para renditjes:\n";
    displayDevices(devices);

    // Renditja e pajisjeve sipas kompanisë dhe modelit
```

```
    sort(devices.begin(), devices.end());

    // Shfaqim pajisjet pas renditjes
    cout << "\nPajisjet pas renditjes:\n";
    displayDevices(devices);

    // Krahasoni dy pajisje për shembull
    ElectronicDevice device1("Apple", "iPhone 13", 2021);
    ElectronicDevice device2("Samsung", "Galaxy S21", 2021);

    if (device1 < device2) {
        cout << "\n" << device1.model << " është më e vogël se " << device2.model << endl;
    } else if (device1 == device2) {
        cout << "\n" << device1.model << " është e barabartë me " << device2.model << endl;
    } else {
        cout << "\n" << device1.model << " është më e madhe se " << device2.model << endl;
    }

    return 0;
}
```