REPUBLIKA E SHQIPËRISË
MINISTRIA E ARSIMIT
DHE SPORTIT

QENDRA E SHËRBIMEVE ARSIMORE

OLIMPIADA KOMBËTARE E INFORMATIKËS
NË ARSIMIN E MESËM TË LARTË

Faza e tretë I

Udhëzime për nxënësin: **Viti shkollor 2024 - 2025**

- 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 ose C++
- Zgjidhjet do të bëhen në kompjuter.

Për përdorim nga komisioni i vlerësimit

Pyetja	**1**	**2**	**3**	**4**	**5**
	9 pikë	**15 pikë**	**8 pikë**	**11 pikë**	**7 pikë**
Pikët e fituara					

Totali i pikëve të fituara ☐

KOMISIONI I VLERËSIMIT

1.....

2. .....

Ushtrim 1. Një qytet përbëhet nga N kryqëzime dhe M rrugë dykahëshe. Çdo rrugë ka një gjatësi të caktuar. Bashkia dëshiron të zbulojë sa më shpejt rrugën më të shkurtër midis dy pikave të rëndësishme të qytetit.

Detyra:
Jepet një graf i përbërë nga N nyje dhe M brinja. Çdo nyje përfaqëson një kryqëzim dhe çdo brinjë një rrugë me gjatësi W. Për një palë nyjesh të dhëna A dhe B, gjeni rrugën më të shkurtër midis tyre.

Formati i hyrjes(input):
Në rreshtin e parë janë dy numra N, M ($1 \le N \le 105$, $1 \le M \le 2 \times 105$). Pasojnë M rreshta me nga tre numra U, V, W, që tregojnë një rrugë dykahëshe midis kryqëzimeve U dhe V me gjatësi W. Në fund jepet një palë nyjesh A, B.

Formati i daljes(output):Një numër i vetëm që përfaqëson gjatësinë e rrugës më të shkurtër nga A te B ose -1 nëse nuk ka rrugë të tillë.

Zgjidhje

```
// Implementimi i algoritmit të Dijkstra për rrugën më të shkurtër
#include <iostream>
#include <vector>
#include <queue>
#include <limits>

using namespace std;

const int INF = numeric_limits<int>::max();
typedef pair<int, int> pii; // Çift (distancë, nyje)

vector<vector<pii>> graf; // Lista e fqinjeve për secilën nyje

vector<int> dijkstra(int N, int start) {
   priority_queue<pii, vector<pii>, greater<pii>> pq;
   vector<int> dist(N + 1, INF); // Inicializimi i distancave me vlerën INF

   dist[start] = 0; // Distanca nga vetja është 0
   pq.push({0, start});

   while (!pq.empty()) {
       int d = pq.top().first;
       int u = pq.top().second;
       pq.pop();
if (d > dist[u]) continue; // Nëse kemi një rrugë më të gjatë, e injorojmë

       for (auto edge : graf[u]) {
           int v = edge.first, w = edge.second;
           if (dist[u] + w < dist[v]) { // Përditësojmë distancën nëse gjejmë një
```

```
rrugë më të shkurtër
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
    return dist;
}

int main() {
    int N, M, A, B;
    cin >> N >> M;

    graf.resize(N + 1);
    for (int i = 0; i < M; i++) {
        int U, V, W;
        cin >> U >> V >> W;
        graf[U].push_back({V, W});
        graf[V].push_back({U, W}); // Grafi është dykahor
    }

    cin >> A >> B; // Nyjet për të cilat kërkojmë rrugën më të shkurtër
    vector<int> dist = dijkstra(N, A);

    cout << (dist[B] == INF ? -1 : dist[B]) << endl; // Printojmë distancën ose -1
nëse nuk ka rrugë
    return 0;
}
```

Ushtrim 2. **15 pikë**

Klasifikimi i Produkteve (Algoritmat e Makinës Mësuese - KNN)

Një kompani teknologjike dëshiron të klasifikojë produkte të ndryshme bazuar në veçoritë e tyre duke përdorur një model të thjeshtë të K-Nearest Neighbors (KNN).

Detyra:
Jepen një listë me N produkte me disa karakteristika numerike, si dhe një produkt i ri për t'u klasifikuar. Përdorni metodën e KNN për të gjetur klasën e tij bazuar në shumicën e klasave të produkteve më të afërta.

Formati i hyrjes(input):
Numri N ($1 \leq N \leq 105$) dhe K ($1 \leq K \leq 50$). N rreshta me D+1 numra (D karakteristika dhe një etiketë klase). Një rresht me D numra që përfaqësojnë produktin për t'u klasifikuar.

Formati i daljes(output):
Një numër që përfaqëson klasën e produktit të ri.

Zgjidhje

```
// Implementimi i KNN për klasifikimin e një produkti të ri
#include <iostream>
#include <vector>
#include <cmath>
#include <map>
#include <algorithm>

using namespace std;

struct Produkt {
   vector<double> karakteristika;
   int klasa;
};

// Funksion për llogaritjen e distancës Euklidiane midis dy produkteve
double distance(const vector<double>& a, const vector<double>& b) {
   double sum = 0;
   for (size_t i = 0; i < a.size(); i++) {
       sum += (a[i] - b[i]) * (a[i] - b[i]);
   }
   return sqrt(sum);
}

int main() {
   int N, K, D;
   cin >> N >> K >> D;

   vector<Produkt> produkte(N);
   for (int i = 0; i < N; i++) {
       produkte[i].karakteristika.resize(D);
       for (int j = 0; j < D; j++) cin >> produkte[i].karakteristika[j];
       cin >> produkte[i].klasa; // Lexojmë klasën e produktit
   }

   vector<double> produkt_i_ri(D);
   for (int j = 0; j < D; j++) cin >> produkt_i_ri[j]; // Lexojmë produktin e ri
për klasifikim

   vector<pair<double, int>> distanca;
   for (const auto& p : produkte) {
```

```
        distanca.push_back({distance(p.karakteristika, produkt_i_ri), p.klasa});
    }
sort(distanca.begin(), distanca.end()); // Rendisim produktet sipas distancës

    map<int, int> numri_klasave;
    for (int i = 0; i < K; i++) numri_klasave[distanca[i].second]++;

    int klasa_max = 0, frekuenca_max = 0;
    for (auto& p : numri_klasave) {
        if (p.second > frekuenca_max) {
            frekuenca_max = p.second;
            klasa_max = p.first;
        }
    }

    cout << klasa_max << endl; // Printojmë klasën më të zakonshme
    return 0;
}
```

Ushtrim 3. **8 pikë**

Kodimi i Mesazheve (Teoria e Numrave - RSA i thjeshtuar)

Një sistem sekret përdor një version të thjeshtëzuar të RSA për të koduar mesazhet. Çdo karakter konvertohet në ASCII dhe pastaj kodohet sipas formulës: C = ME mod N.

Detyra:

Duke pasur një mesazh të koduar dhe çelësin privat (D, N), rikuperoni mesazhin origjinal.

Formati i hyrjes(input):

Dy numra D, N (1 ≤ D, N ≤ 109). Një rresht me M numra të ndarë me hapësirë që përfaqësojnë mesazhin e koduar.

Formati i daljes(output):Një varg me karakteret e mesazhit të deshifruar.

Zgjidhje

```
// Implementimi i RSA për dekriptimin e mesazheve
#include <iostream>
#include <vector>

using namespace std;

// Funksion për eksponent modular (modular exponentiation)
long long mod_exp(long long base, long long exp, long long mod) {
```

```
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) result = (result * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }


return result;
}


int main() {
    long long D, N;
    cin >> D >> N; // Lexojmë çelësin privat RSA
vector<int> mesazhi;
    int koduar;
    while (cin >> koduar) {
        mesazhi.push_back(koduar); // Lexojmë karakteret e koduara
    }
    for (int c : mesazhi) {
        cout << char(mod_exp(c, D, N)); // Dekriptojmë dhe konvertojmë në
karaktere
    }
    cout << endl;
    return 0;
}
```

Ushtrim 4. **11 pikë**

Rindërtimi i ADN-së (Grafet - Eulerian Path)

Shkencëtarët kanë ndërtuar një grup fragmente të një ADN-je dhe dëshirojnë të rindërtojnë renditjen e saktë të bazave. Duke supozuar se fragmentet janë të lidhura në mënyrë Euleriane, gjeni sekuencën origjinale.

Detyra:
Jepen N fragmente si nyje të një grafi. Lidhjet midis tyre paraqiten si një graf Eulerian. Duhet të rindërtoni sekuencën origjinale duke përdorur Eulerian Path.

Formati i hyrjes(input):
Numri N ($1 \leq N \leq 105$). M rreshta që përfaqësojnë lidhjet midis fragmenteve.

Formati i daljes(output):
Një varg me renditjen e fragmenteve që formojnë ADN-në.

Zgjidhje

```
// Implementimi i algoritmit Hierholzer për rrugën Euleriane
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

vector<vector<int>> graf;
vector<int> rezultati;
vector<int> shkalle;

// Funksion për të ndërtuar rrugën Euleriane
void eulerian_path(int v) {
    stack<int> s;
    s.push(v);
    while (!s.empty()) {
        int u = s.top();
        if (!graf[u].empty()) {
            int next = graf[u].back();
            graf[u].pop_back();
            s.push(next);
        }

else {
            rezultati.push_back(u);
            s.pop();
        }
    }
}
int main() {
    int N, M;
    cin >> N >> M;

    graf.resize(N);
    shkalle.resize(N, 0);

    for (int i = 0; i < M; i++) {
        int u, v;
        cin >> u >> v;
        graf[u].push_back(v);
        graf[v].push_back(u);
        shkalle[u]++;
        shkalle[v]++;
    }

    int start = 0;
```

```
    for (int i = 0; i < N; i++) {
        if (shkalle[i] % 2 == 1) {
            start = i;
            break;
        }
    }

    eulerian_path(start);

    for (int i = rezultati.size() - 1; i >= 0; i--) {
        cout << rezultati[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Ushtrimi 5. **7 pikë**

Transporti Optimal i Robotëve (Algoritmat e Rrjedhës Maksimale - Ford-Fulkerson)

Një kompani teknologjike dëshiron të automatizojë transportin e robotëve mes fabrikave duke përdorur një sistem inteligjent të menaxhimit të trafikut. Çdo rrugë ka një kapacitet të caktuar transportimi.

Detyra:
Jepet një rrjet me N nyje dhe M lidhje me kapacitete të caktuara. Duhet të gjeni rrjedhën maksimale nga nyja S (depoja) në nyjën T (destinacioni final).

Formati i hyrjes (input):
Në rreshtin e parë janë N, M ($1 \le N \le 500$, $1 \le M \le 104$). Pasojnë M rreshta me nga tre numra U, V, C, ku U dhe V përfaqësojnë lidhjen mes nyjeve dhe C është kapaciteti. Në fund jepen S, T.
Formati i daljes (output):
Një numër që përfaqëson rrjedhën maksimale nga S në T.

Zgjidhje

```
// Implementimi i algoritmit Ford-Fulkerson për rrjedhën maksimale

#include <iostream>
#include <vector>
#include <queue>
#include <cstring>

using namespace std;

const int MAX = 500;
int kapaciteti[MAX][MAX], rrjedha[MAX][MAX], prindi[MAX];
vector<int> graf[MAX];

// Funksion për kërkimin e një rrugë të re me BFS
bool bfs(int S, int T) {
    memset(prindi, -1, sizeof(prindi));
```

```
    queue<int> q;
    q.push(S);
    prindi[S] = S;

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

        for (int v : graf[u]) {
            if (prindi[v] == -1 && kapaciteti[u][v] > rrjedha[u][v]) {
                prindi[v] = u;
                if (v == T) return true;
                q.push(v);
            }
        }
    }
    return false;
}

// Funksion për llogaritjen e rrjedhës maksimale
int ford_fulkerson(int S, int T) {
    int rrjedha_maks = 0;
    while (bfs(S, T)) {
        int rrjedha_sakte = 1e9;
        for (int v = T; v != S; v = prindi[v]) {
            int u = prindi[v];
            rrjedha_sakte = min(rrjedha_sakte, kapaciteti[u][v] - rrjedha[u][v]);
        }
for (int v = T; v != S; v = prindi[v]) {
            int u = prindi[v];
            rrjedha[u][v] += rrjedha_sakte;
            rrjedha[v][u] -= rrjedha_sakte;
        }

        rrjedha_maks += rrjedha_sakte;
    }
    return rrjedha_maks;
}
int main() {
    int N, M, S, T;
    cin >> N >> M;

    memset(kapaciteti, 0, sizeof(kapaciteti));
    memset(rrjedha, 0, sizeof(rrjedha));

    for (int i = 0; i < M; i++) {
        int U, V, C;
        cin >> U >> V >> C;
        graf[U].push_back(V);
        graf[V].push_back(U);
        kapaciteti[U][V] = C;
    }
    cin >> S >> T;
    cout << ford_fulkerson(S, T) << endl;
    return 0;
}
```

REPUBLIKA E SHQIPËRISË
MINISTRIA E ARSIMIT
DHE SPORTIT
QENDRA E SHËRBIMEVE ARSIMORE

OLIMPIADA KOMBËTARE E INFORMATIKËS

NË ARSIMIN E MESËM TË LARTË

Faza e tretë II

Udhëzime për nxënësin: **Viti shkollor 2024 - 2025**

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

Për përdorim nga komisioni i vlerësimit

Pyetja	**1**	**2**	**3**	**4**	**5**
	7 pikë	**10 pikë**	**8 pikë**	**15 pikë**	**10 pikë**
Pikët e fituara					

Totali i pikëve të fituara ☐

KOMISIONI I VLERËSIMIT

1.……

2. ……

Ushtrim 1.
Një kompani dëshiron të ndërtojë rrjetin elektrik më ekonomik duke minimizuar kostot e ndërtimit të kabllove mes qyteteve. Përdorni algoritmin e **Kruskal-it** për të gjetur pemën minimale lidhëse.

7 pikë

Zgjidhje

```
#include <bits/stdc++.h>
using namespace std;

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

vector<int> parent, rank_set;

int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (rank_set[a] < rank_set[b])
            swap(a, b);
        parent[b] = a;
        if (rank_set[a] == rank_set[b])
            rank_set[a]++;
    }
}

int main() {
    int n, m;
    cin >> n >> m;
    vector<Edge> edges(m);
    for (int i = 0; i < m; i++) {
        cin >> edges[i].u >> edges[i].v >> edges[i].weight;
    }

    sort(edges.begin(), edges.end());
    parent.resize(n);
    rank_set.resize(n);
```

```
    for (int i = 0; i < n; i++) {
        parent[i] = i;
        rank_set[i] = 0;
    }

    int cost = 0;
    vector<Edge> result;
    for (Edge e : edges) {
        if (find_set(e.u) != find_set(e.v)) {
            cost += e.weight;
            result.push_back(e);
            union_sets(e.u, e.v);
        }
    }

    cout << "Kosto minimale: " << cost << endl;
    return 0;
}
```

Ushtrim 2.

Një kompani taksish dëshiron të gjejë rrugën më të shkurtër midis dy lokacioneve duke përdorur algoritmin e Dijkstra-s.

10 pikë

Zgjidhje

```
#include <bits/stdc++.h>
using namespace std;

const int INF = 1e9;

int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<pair<int, int>>> adj(n);
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
    }
    int src;
    cin >> src;

    vector<int> dist(n, INF);
    dist[src] = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    pq.push({0, src});

    while (!pq.empty()) {
        int d = pq.top().first;
        int u = pq.top().second;
```

```
        pq.pop();
        if (d > dist[u]) continue;
        for (auto edge : adj[u]) {
            int v = edge.first;
            int w = edge.second;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }

    for (int i = 0; i < n; i++) {
        cout << "Distanca nga " << src << " te " << i << " eshte " << dist[i] << endl;
    }
    return 0;
}
```

Ushtrim 3.

Një sistem kërkimi dëshiron të gjejë shpejt një nënvarg në një tekst të madh duke përdorur algoritmin KMP.

8 pikë

Zgjidhje

```
#include <bits/stdc++.h>
using namespace std;

void computeLPSArray(string pat, vector<int>& lps) {
    int m = pat.size(), len = 0;
    lps[0] = 0;
    int i = 1;
    while (i < m) {
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;
            i++;
        } else {
            if (len != 0) {
                len = lps[len - 1];
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }
}

void KMPSearch(string txt, string pat) {
    int n = txt.size(), m = pat.size();
    vector<int> lps(m);
```

```
    computeLPSArray(pat, lps);
    int i = 0, j = 0;
    while (i < n) {
        if (pat[j] == txt[i]) {
            i++, j++;
        }
        if (j == m) {
            cout << "Pattern gjendur ne indeksin " << i - j << endl;
            j = lps[j - 1];
        } else if (i < n && pat[j] != txt[i]) {
            if (j != 0)
                j = lps[j - 1];
            else
                i++;
        }
    }
}

int main() {
    string txt, pat;
    cin >> txt >> pat;
    KMPSearch(txt, pat);
    return 0;
}
```

Ushtrim 4.

Një kompani teknologjike ka një rrjet serverash të lidhur mes tyre, ku çdo lidhje ka një kapacitet të caktuar për të transmetuar të dhëna. Qëllimi është të gjendet sasia maksimale e të dhënave që mund të transmetohen nga serveri burim (S) te serveri destinacion (T).
Duke përdorur algoritmin **Ford-Fulkerson,** gjeni rrjedhën maksimale të të dhënave që mund të transmetohen në rrjet.

Input:

- Numri i serverëve N dhe numri i lidhjeve M.
- M rreshta ku secili përmban U, V, C (lidhje nga serveri U te serveri V me kapacitet C).
- Serveri burim S dhe serveri destinacion T.

Output:

- Një numër që tregon rrjedhën maksimale nga S te T. **15 pikë**

Zgjidhje

```
#include <bits/stdc++.h>
using namespace std;

void printPermutations(string str, int l, int r) {
    if (l == r) {
        cout << str << endl;  // Shtyp permutimin aktual
    } else {
```

```
        for (int i = l; i <= r; i++) {
            swap(str[l], str[i]);  // Ndërrojmë karakteret për të krijuar një permutim të ri
            printPermutations(str, l + 1, r);
            swap(str[l], str[i]);  // Rikthejmë ndryshimin (backtracking)
        }
    }
}

int main() {
    string str;
    cin >> str;  // Lexojmë vargun nga përdoruesi
    printPermutations(str, 0, str.size() - 1);
    return 0;
}
```

Ushtrimi 5.

Një kompani dëshiron të sigurojë komunikimet mes serverëve duke përdorur **kriptimin AES me çelës 128-bit.**

Implementoni një metodë XOR për të kriptuar dhe dekriptuar një mesazh duke përdorur një çelës simetrik.

10 pikë

Zgjidhje

```
#include <bits/stdc++.h>

using namespace std;


int countWays(vector<int>& coins, int sum) {

    vector<int> dp(sum + 1, 0);

    dp[0] = 1;  // Ka vetëm një mënyrë për të krijuar shumën 0 (duke mos përdorur asnjë monedhë)


    for (int coin : coins) {  // Kalojmë përmes secilës monedhë

        for (int j = coin; j <= sum; j++) {

            dp[j] += dp[j - coin];  // Shtojmë mënyrat për të krijuar shumën aktuale

        }

    }

    return dp[sum];  // Kthejmë numrin total të mënyrave për të arritur shumën
```

```
}

int main() {
    int n, sum;
    cin >> n >> sum;  // Lexojmë numrin e monedhave dhe shumën e synuar
    vector<int> coins(n);
    for (int i = 0; i < n; i++) cin >> coins[i];  // Lexojmë monedhat

    cout << countWays(coins, sum) << endl;  // Printojmë rezultatin
    return 0;
}
```