```
Mazno2 Stnoni Lfoo2 Ref
 freopen("input.txt", "r", stdin);
 freopen("output.txt", "w", stdout);
# String.....................
transform(s.begin(), s.end(), s.begin(), ::tolower);
transform(s.begin(), s.end(), s.begin(), ::toupper);
read spaces or \n
cin.get(x);
isalpha(), isdigit(), isupper(), islower()
Tolower(),toupper()
Some string functions
Substr(pos,len)
Erase(pos,len)
Insert(pos, string)
Find(string s )  return string::npos if not find
.....................................................
#Binary Search ........................................
#binary search on index;.......
lower_bound returns a pointer that points on the smallest element larger than or
equal to 5.
  If there are several such elements it chooses the one with smallest index.
upper_bound returns a pointer that points on the smallest element strictly larger
than 5.
  If there are several such elements it chooses the one with smallest index.
imprtant note::
        Sometimes you need to check if the pointer is pointing on n. This means
that there is no such element.

lower_bound();   return x >= wanted value first find
upper_bound();   return x > wanted value
to return value
int p = * lower_bound(a,a+n, wanted value);
to return index of value
int i = lower_bound(a,a+n,wanted value) - a ;
to use this functions for subarray
lower_bound( a + start_idx , a + end_idx +1 , wanted value);
If you want the element that is strictly less than 5,
find the lower_bound of 5 and then subtract one from it.
You need to check that the pointer doesn't point on -1.
This means that there is no such element
int pos = lower_bound(a,a+n,5) - a - 1;
    if( pos != -1 )
        cout << a[pos] << endl;
/// Same thing as above if you want the value.

    if( lower_bound(a,a+n,5) - a - 1 != -1 )

    {

        int val = *(lower_bound(a,a+n,5)-1);

        cout << val << endl;

    }

/// If you want the element that is less than or equal to 5,
```

```
find the upper_bound of 5 then subtract one from it.

    int pos = upper_bound(a,a+n,5) - a - 1;

    if( pos != -1 )

        cout << a[pos] << endl;



    if( upper_bound(a,a+n,5) - a - 1 != -1 )
    {

        int val = *(upper_bound(a,a+n,5)-1);
        cout << val << endl;

    }
/// Same as sort function, if you have a 1-indexed array, you need to add 1 to the
beginning and the end of the array like this: upper_bound(a+1,a+n+1,5)
for vectors
   sort(v.begin(),v.end());
    int pos = lower_bound(v.begin(),v.end(),5) - v.begin();
    if( pos != (int)(v.size()) )
        cout << v[pos] << endl;
    /// Or:
    vector <int> :: iterator it = lower_bound(v.begin(),v.end(),5);
    if( it != v.end() )

    {
        cout << it-v.begin() << endl; /// pos
        cout << *it << endl; ///val
    }

 for sets
    sets are already sorted
    set <int> :: iterator it = s.lower_bound(5);
/// note that
lower_bound(s.begin(),s.end(),5) doesn.t work.

     if( it != s.end() )

    {
        cout << *it << endl; /// val
        /// no indexes so no pos     }

#Math.................................................
#factorization..............

vector <ll> factorization(ll x)

{

    vector <ll> ret;
    for(ll i=1;i*i<=x;i++)

    {
        if( x%i == 0 )
```

```
        {
            ret.push_back(i);
            if( i*i != x )    /// or i != x/i
                ret.push_back(x/i);
        }
    }
    sort(ret.begin(),ret.end()); /// if needed
    return ret;
}
#.........................................
#Seive.............

const int N = 1000100;
bool p[N];
vector <int> primes;
void seive()

{
    memset(p,1,sizeof p);
    p[0] = p[1] = 0;

    for(int i=2;i*i<=1000000;i++)

        if( p[i] )

            for(int j=i*i;j<=1000000;j+=i)

                p[j] = 0;

    for(int i=0;i<=1000000;i++)

        if( p[i] )

            primes.push_back(i);

}

#.........................................
#Prime factorization
const int N = 1000100;
bool p[N];
vector <int> primes;
void seive()

{
    memset(p,1,sizeof p);
    p[0] = p[1] = 0;
    for(int i=0;i*i<=1000000;i++)

        if( p[i] )

            for(int j=i*i;j<=1000000;j+=i)
```

```
  p[j] = 0;
    for(int i=0;i<=1000000;i++)
        if( p[i] )
            primes.push_back(i);

}
vector < pair < ll , int > > primeFactoriztion(ll x)

{

    vector < pair < ll , int > > ret;



    for(int i=0;primes[i]*primes[i]<=x;i++)

    {

        int cnt = 0;



        while( x%primes[i] == 0 )

        {

            x /= primes[i];

            cnt++;

        }



        if( cnt )

            ret.push_back( { primes[i] , cnt } );

    }



    if( x != 1 )

        ret.push_back( { x , 1 } );



    return ret;

}

#.................................................

#LCM
```

```
ll LCM(ll x,ll y) { return x/__gcd(x,y)*y; }



#.................................................
#Power with mod.........
ll mod = 1e9+7;
ll po(ll x,ll os)

{

    if( os == 0 )

        return 1;



    ll z = po(x,os/2);



    if( os&1 )

        return z*z%mod*x%mod;



    return z*z%mod;

}



#.................................................

#Graph..............

/** GRAPH */


/* BFS */
int n, m;
vector<int> adj[200200];
void addEdge(int u, int v) {
 // Undirected
 adj[u].push_back(v);
 adj[v].push_back(u);
 return;
}
// O(V+E)
vector<int> Bfs(int start) {
 queue<int> q;
 vector<int> Distance(n + 1, 1e8);
 q.push(start);
 Distance[start] = 0;
 while (!q.empty()) {
 int parent = q.front();
```

```
 q.pop();
 for (int son : adj[parent]) {
 if (Distance[son] > Distance[parent] + 1) {
 Distance[son] = Distance[parent] + 1;
 q.push(son);
 }
 }
 }
 return Distance;
}
int main() {
 n = 7;
 addEdge(3, 5);
 addEdge(5, 2);
 addEdge(2, 4);
 addEdge(6, 1);
 addEdge(1, 3);
 addEdge(3, 6);
 int sorce = 2;
 vector<int> Dist = Bfs(sorce);
 for (int i = 1; i <= n; i++) {
 if (Dist[i] == 1e8)
 cout << "There is no path from "
 << sorce << " to " << i << endl;
 else
 cout << "The length from " << sorce
 << " to " << i << " is " << Dist[i]
 << endl;
 }
 return 0;
}
.....................................................
/* DFS */
int n, m, vis[200200];
vector<int> adj[200200];
// O(V+E) maybe including for loop in main
void Dfs(int u) {
 if (vis[u]) return;
 vis[u] = 1;
 for (int v : adj[u]) Dfs(v);
}
void addEdge(int u, int v) {
 adj[u].push_back(v);
 adj[v].push_back(u);
 return;
}
int main() {
 int numberOfComponents = 0;
 n = 5;
 addEdge(1, 3);
 addEdge(5, 2);
 addEdge(2, 4);
 for (int i = 1; i <= n; i++) {
 if (!vis[i]) {
 Dfs(i);
 numberOfComponents++;
 }
```

```
 }
 cout << numberOfComponents;
 return 0;
}
.........................................

/** DIJKSTRA */
int n;
vector<ll> Distance[1010];
vector<pair<ll, int>> adj[1010];
void addEdge(int u, int v, ll w) {
 // Undirected and Weighted
 adj[u].push_back({w, v}); // first w then v
 adj[v].push_back({w, u}); // first w then u
 return;
}
// O(V + E*log(V))
vector<ll> Dijkstra(int sorce) {
 priority_queue<pair<ll, int>> q;
 vector<ll> Dist(n + 1, 1e18);
 Dist[sorce] = 0;
 q.push({Dist[sorce], sorce});
 while (!q.empty()) {
 pair<ll, int> Top = q.top();
 q.pop();
 int Parent = Top.second;
 ll DistParent = (-1) * Top.first;
 if (DistParent > Dist[Parent]) continue; // Very Important
 for (auto PairSon : adj[Parent]) {
 int son = PairSon.second;
 ll Weight = PairSon.first;
 if (DistParent + Weight < Dist[son]) {
 Dist[son] = DistParent + Weight;
 q.push({(-1) * Dist[son], son}); // first Dist then son
 }
 }
 }
 return Dist;
}
int main() {
 n = 5;
 addEdge(1, 3, 2);
 addEdge(1, 2, 2);
 addEdge(2, 4, 3);
 addEdge(4, 5, 8);
 addEdge(1, 4, 6);
 for (int i = 1; i <= n; i++) Distance[i] = Dijkstra(i);
 for (int i = 1; i <= n; i++) {
 for (int j = 1; j <= n; j++) {
 if (Distance[i][j] == 1e18)
 cout << "There is no path form " << i << " to " << j << endl;
 else
 cout << "Shortest Path from " << i << " to " << j << " is "
 << Distance[i][j] << endl;
 }
 cout << "\n--------------\n";
 }
```

```
 return 0;
}
.....................................................................
.....................................................................


/* FLOYD */
int n, m, Distance[555][555];
void addEdge(int u, int v, int w) {
 // Directed and Weighted
 Distance[u][v] = w;
 // if The Graph is Undirected add Distance[v][u] = w
 return;
}
void Initiate() {
 for (int i = 1; i <= n; i++)
 for (int j = 1; j <= n; j++)
 Distance[i][j] = 1e8; // should be number larger than the longest
path
 for (int i = 1; i <= n; i++) Distance[i][i] = 0;
 return;
}
void Floyd() {
 for (int k = 1; k <= n; k++)
 for (int i = 1; i <= n; i++)
 for (int j = 1; j <= n; j++)
 Distance[i][j] = min(Distance[i][j], Distance[i][k] +
Distance[k][j]);
 return;
}
int main() {
 n = 5;
 Initiate();
 addEdge(1, 3, 2);
 addEdge(1, 2, 2);
 addEdge(2, 4, 3);
 addEdge(4, 5, 8);
 addEdge(1, 4, 6);
 Floyd();
 for (int i = 1; i <= n; i++) {
 for (int j = 1; j <= n; j++) {
 if (Distance[i][j] == 1e8)
 cout << "There is no path form " << i << " to " << j << endl;
 else
 cout << "Shortest Path from " << i << " to " << j << " is "
 << Distance[i][j] << endl;
 }
 cout << "\n--------------\n";
 }
 return 0;
}

.....................................................................
.....................................................................

/** Topgological Sort */
vector<int> adj[300100];
```

```
int inD[300100];
int dp[300100];
vector<int> topoSort;
int n, m;
string s;
// TOPOLOGICAL SORT ONLY WORKS ON DIRECTED ACYCLIC GRAPH (DAG)!
// TOPOLOGICAL SORT IS QUIET CLOSE TO DP!!
int main() {
 speed;
 // Separator
 cin >> n >> m;
 cin >> s;
 s = '*' + s;
 for (int i = 0; i < m; i++) {
 int u, v;
 cin >> u >> v;
 adj[u].push_back(v);
 inD[v]++;
 }
 int cnt = 0;
 queue<int> q;
 for (int i = 1; i <= n; i++)
 if (inD[i] == 0) q.push(i);
 while (!q.empty()) {
 int u = q.front();
 q.pop();
 cnt++;
 topoSort.push_back(u);
 for (int v : adj[u]) {
 inD[v]--;
 if (inD[v] == 0) q.push(v);
 }
 }
 if (cnt < n) {
 // cycle!
 cout << -1 << endl;
 return 0;
 }
 int ans = 0;
 for (char c = 'a'; c <= 'z'; c++) {
 memset(dp, 0, sizeof(dp));
 for (int u : topoSort) {
 if (s[u] == c) dp[u]++;
 for (int v : adj[u]) dp[v] = max(dp[v], dp[u]);
 ans = max(ans, dp[u]);
 }
 }
 cout << ans << endl;
 // Separator
 return 0;
}
```

.....

.....

MST Prim

dense graph n^2

```cpp
int n;
vector<vector<int>> adj; // adjacency matrix of graph
const int INF = 1000000000; // weight INF means there is no edge

struct Edge {
    int w = INF, to = -1;
};

void prim() {
    int total_weight = 0;
    vector<bool> selected(n, false);
    vector<Edge> min_e(n);
    min_e[0].w = 0;

    for (int i=0; i<n; ++i) {
        int v = -1;
        for (int j = 0; j < n; ++j) {
            if (!selected[j] && (v == -1 || min_e[j].w < min_e[v].w))
                v = j;
        }

        if (min_e[v].w == INF) {
            cout << "No MST!" << endl;
            exit(0);
        }

        selected[v] = true;
        total_weight += min_e[v].w;
        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to << endl;

        for (int to = 0; to < n; ++to) {
            if (adj[v][to] < min_e[to].w)
                min_e[to] = {adj[v][to], v};
        }
    }

    cout << total_weight << endl;
}
```

.....

MST Prim

sparse graphs m log n

```cpp
const int INF = 1000000000;

struct Edge {
    int w = INF, to = -1;
    bool operator<(Edge const& other) const {
        return make_pair(w, to) < make_pair(other.w, other.to);
    }
};

int n;
vector<vector<Edge>> adj;
```

```cpp
void prim() {
    int total_weight = 0;
    vector<Edge> min_e(n);
    min_e[0].w = 0;
    set<Edge> q;
    q.insert({0, 0});
    vector<bool> selected(n, false);
    for (int i = 0; i < n; ++i) {
        if (q.empty()) {
            cout << "No MST!" << endl;
            exit(0);
        }

        int v = q.begin()->to;
        selected[v] = true;
        total_weight += q.begin()->w;
        q.erase(q.begin());

        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to << endl;

        for (Edge e : adj[v]) {
            if (!selected[e.to] && e.w < min_e[e.to].w) {
                q.erase({min_e[e.to].w, e.to});
                min_e[e.to] = {e.w, v};
                q.insert({e.w, e.to});
            }
        }
    }

    cout << total_weight << endl;
}
```

.....

.....

Alaa

.....

//////////////////////DP

// logest increasing subsequance

```cpp
inline void solve()
{
    vector<int> v = {-9, 3, 4, 5, 2, 2, 2};

    vector<vector<int>> dp(7);
    for (int i = 0; i < 7; i++)
    {
        dp[i].push_back(v[i]);
    }
    int j = 0;
    for (int i = 0; i < 7; i++)
    {
        j = 0;
        while (j < i)
        {
            if (v[i] >= v[j])
            {
```

```
                if (dp[j].size() >= dp[i].size())
                {
                    for (auto e : dp[j])
                    {
                        dp[i].push_back(e);
                    }
                }
            }
            j++;
        }
    }
    for (auto i : dp)
    {
        for (auto j : i)
        {
            cout << j << " ";
        }
        cout << endl;
    }
}
////LCS Longest Common Subsequence
inline void solve()
{
    string a = "abracadabra", b = "avadakedavra";
    cin >> a >> b;

    a = " " + a;
    b = " " + b;

    int v[30][30];

    for (int i = 0; i <= a.length(); i++)
        for (int j = 0; j <= b.length(); j++)
            v[i][j] = 0;

    for (int i = 1; i < a.length(); i++)
    {
        for (int j = 1; j < b.length(); j++)
        {

            if (i == 0 || j == 0)
            {
                v[i][j] = 0;
            }
            else
            {
                if (a[i] == b[j])
                    v[i][j] = v[(i - 1)][(j - 1)] + 1;
                else
                    v[i][j] = max(v[(i - 1)][j], v[i][(j - 1)]);
            }
```

```
        }
    }

    // for (int i = 0; i < a.length(); i++)
    // {
    //     for (int j = 0; j < b.length(); j++)
    //         cout << v[i][j] << " ";
    //     cout << endl;
    // }
    int x = a.length() - 1, y = b.length() - 1;
    string ans = "";
    while (x && y)
    {
        if (v[x][y] == v[x - 1][y])
            x--;
        else if (v[x][y] == v[x][y - 1])
            y--;
        else
        {

            ans.push_back(a[x]);
            x--;
            y--;
        }
    }
    for (int i = ans.length() - 1; i > -1; i--)
        cout << ans[i];
    // cout <<endl<< v[a.length() - 1][b.length() - 1];
}
// longest common subarray
vector<char> longest_common_subarray(vector<char> &arr1, vector<char> &arr2)
{
    int m = arr1.size(), n = arr2.size();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
    int max_len = 0, end_idx = 0;

    for (int i = 1; i <= m; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (arr1[i - 1] == arr2[j - 1])
            {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                if (dp[i][j] > max_len)
                {
                    max_len = dp[i][j];
                    end_idx = i - 1;
                }
            }
        }
    }
```

```
    return vector<char>(arr1.begin() + end_idx - max_len + 1, arr1.begin() +
end_idx + 1);
}
////KnapSack
inline void solve()
{
    ll n, Weight;
    cin >> n >> Weight;
    ll value[n + 1];
    ll weight[n + 1];

    ll dp[2][Weight + 1];

    for (int i = 1; i <= n; i++)
    {
        cin >> weight[i] >> value[i];
    }

    value[0] = 0;
    weight[0] = 0;

    for (int i = 0; i <= Weight; i++)
        dp[0][i] = 0;

    for (int i = 1; i <= n; i++)
    {
        for (int w = 0; w <= Weight; w++)
        {
            if (weight[i] <= w)
            {
                dp[i % 2][w] = max(dp[(i - 1) % 2][w], dp[(i - 1) % 2][w -
weight[i]] + value[i]);
            }
            else
            {
                dp[i % 2][w] = dp[(i - 1) % 2][w];
            }
        }
    }
    cout << dp[n % 2][Weight];
}
//// knapSack two
inline void solve()
{
    ll n, weight;
    cin >> n >> weight;
    vector<ll> val(n + 1, 0);
    vector<ll> wet(n + 1, 0);
    ll sum = 0;
    for (int i = 1; i <= n; i++)
```

```
    {
        cin >> wet[i] >> val[i];
        sum += val[i];
    }
    wet[0] = oo;
    val[0] = oo;
    vector<vector<ll>> dp(n + 1, vector<ll>(sum + 1, oo));

    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j <= sum; j++)
        {
            dp[i][j] = dp[i - 1][j];
            if (val[i] <= j)
                dp[i][j] = min(dp[i - 1][j], wet[i] + dp[i - 1][j - val[i]]);
        }
    }
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j <= sum; j++)
        {
            cout << dp[i][j] - oo << " ";
        }
        cout << endl;
    }
    ll ans = 0;
    for (int i = sum; i >= 0; i--)
    {
        if (dp[n][i] >= weight)
        {
            ans = dp[n][i];
            break;
        }
    }
    cout << ans;
}
// bitmask
//  Simple assignment
/* bitset<2> arr; */
/* arr[0] = 1; */
/* arr[1] = 0; */
/* cout << arr << '\n'; // 01 */

// Integer to bitset and vice-versa
/* bitset<4>a_int(8); */
/* cout << a_int << '\n'; // 1000 */
/* int n = (int) a_int.to_ulong(); */
/* cout << n << '\n'; // 8 */

// String to bitset
/* string str = "1010110100"; */
/* bitset<10> brr(str); */
/* cout << brr[0] << '\n'; // 0 */
```

```
/* string new_str = brr.to_string(); */
/* cout << new_str << '\n'; // 1010110100 */

// Count no of ones
/* cout << brr.count() << '\n'; // 5 */

// Basic operations..
/* bitset<4> a(string("0101")); */
/* bitset<4> b(string("1010")); */
/* cout << (a & b) << '\n'; // 0000 */
/* cout << (a | b) << '\n'; // 1111 */
/* cout << (a ^ b) << '\n'; // 1111 */
/* cout << (~a) << '\n'; // 1010 */
/* cout << (a << 1) << '\n'; // 1010 */
/* cout << (b >> 1) << '\n'; // 0101 */

/* string str1 = "1010110100"; */
/* istringstream stream(str1); */
/* bitset<2> s1; */
/* bitset<6> s2; */
/* stream >> s1; */
/* cout << s1 << '\n'; // 10 */
/* stream >> s2; */
/* cout << s2 << '\n'; // 101101 */

// Check if any bit is set
/* bitset<4> a1(string("1010")); */
/* bitset<4> b1(string("0000")); */
/* cout << a1.any() << '\n'; // true */
/* cout << b1.any() << '\n'; // false */

// Check if none of the bits are set
/* cout << a1.none() << '\n'; // false */
/* cout << b1.none() << '\n'; // true */

// Check if all bits are set
/* bitset<4> ball(string("1111")); */
/* cout << ball.all() << '\n'; // True */

// Flip all or any particular bit:
/* bitset<4> a3(string("1010")); */
/* cout << a3.flip() << '\n'; // 0101 */
/* cout << a3 << '\n'; */
/* cout << a3.flip(1) << '\n'; // 0111 */

// Reset all or any particular bit:
/* bitset<4> a4(string("1010")); */
/* cout << a4.reset(1) << '\n'; // 1000 */
/* cout << a4.reset() << '\n'; // 0000 */

// Set all or any particular bit:
// bitset<4> a5(string("1010"));
// cout << a5.set(0) << '\n'; // 1011
// cout << a5.set() << '\n'; // 0000

..................................................
Checking a graph for acyclicity and finding a cycle in
```

O(M)

for directed graph.

.....

```
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

.....

for undirected graph.

.....

```cpp
int n;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v, int par) { // passing vertex and its parent vertex
    visited[v] = true;
    for (int u : adj[v]) {
        if(u == par) continue; // skipping edge to parent vertex
        if (visited[u]) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
        parent[u] = v;
        if (dfs(u, parent[u]))
            return true;
    }
    return false;
}

void find_cycle() {
    visited.assign(n, false);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (!visited[v] && dfs(v, parent[v]))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

.....

.....

Finding a negative cycle in the graph

.....

Using Bellman-Ford algorithm

.....

```cpp
struct Edge {
    int a, b, cost;
};
```

```
int n, m;
vector<Edge> edges;
const int INF = 1000000000;

void solve()
{
    vector<int> d(n);
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (Edge e : edges) {
            if(d[e.a] < INF){
                if (d[e.a] + e.cost < d[e.b]) {
                    d[e.b] = max(-INF, d[e.a] + e.cost);
                    p[e.b] = e.a;
                    x = e.b;
                }
            }
        }
    }

    if (x == -1) {
        cout << "No negative cycle found.";
    } else {
        for (int i = 0; i < n; ++i)
            x = p[x];

        vector<int> cycle;
        for (int v = x;; v = p[v]) {
            cycle.push_back(v);
            if (v == x && cycle.size() > 1)
                break;
        }
        reverse(cycle.begin(), cycle.end());

        cout << "Negative cycle: ";
        for (int v : cycle)
            cout << v << ' ';
        cout << endl;
    }
}
.....................
Using Floyd-Warshall algorithm
.....................
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        for (int t = 0; t < n; ++t) {
            if (d[i][t] < INF && d[t][t] < 0 && d[t][j] < INF)
                d[i][j] = - INF;
        }
    }
}
.....................................................
.....................................................
```