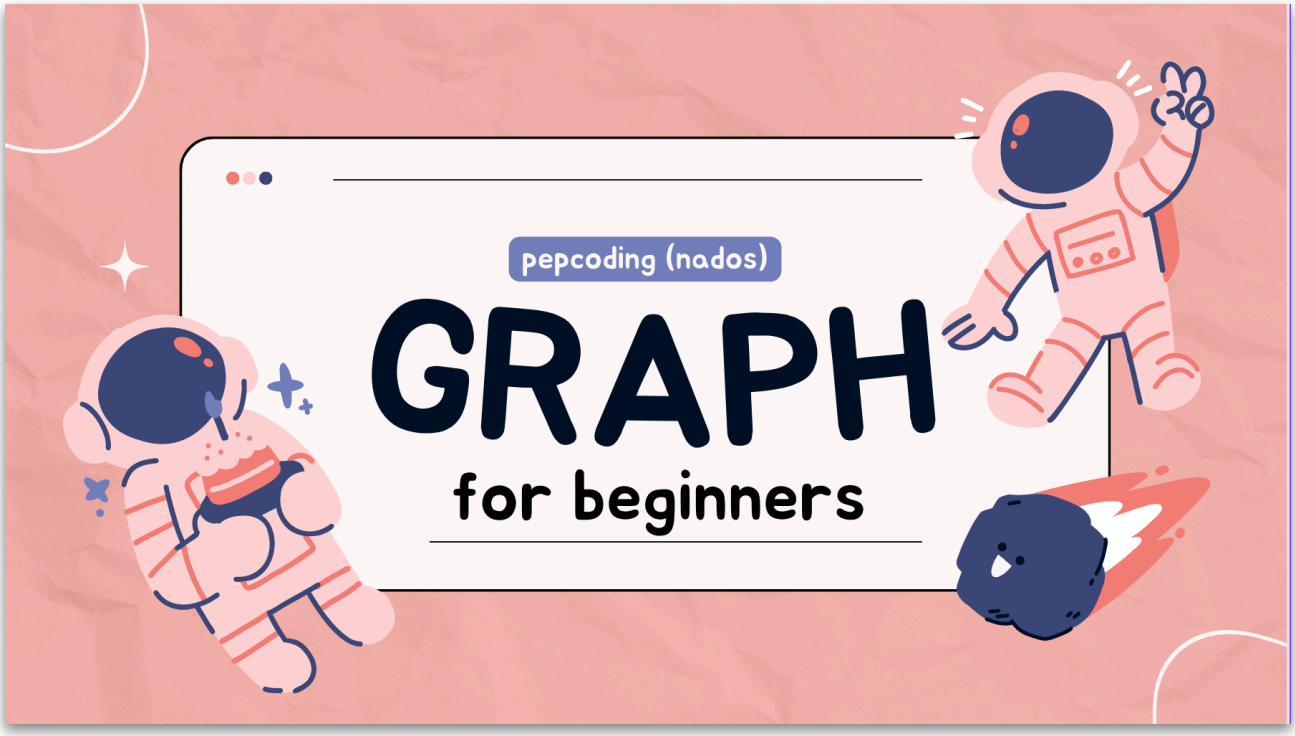
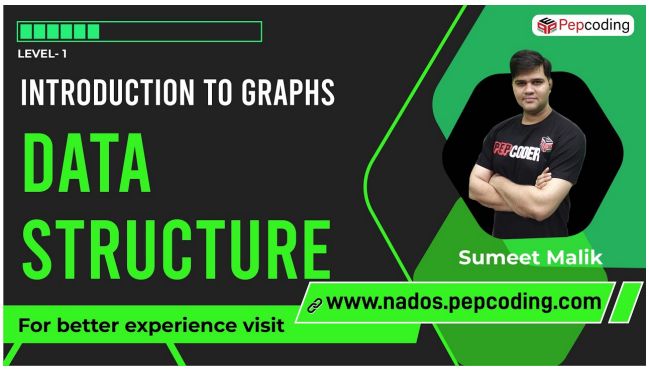


Introduction To Graph And Its Representation



Has Path?

Easy

1. You are given a graph, a src vertex and a destination vertex.
2. You are required to find if a path exists between src and dest. If it does, print true otherwise print false.

Constraints

None

Format

Input

Input has been managed for you

Output

true if path exists, false otherwise

Example

Sample Input

```
7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
0
6
```

Sample Output

```
true
```

```
#include<iostream>
#include<vector>
#include<iostream>
#include<iostream>
using namespace std;
   class Edge {
       public:
          int src;
          int nbr;
          int wt;

      Edge(int src, int nbr, int wt){
         this->src = src;
         this->nbr = nbr;
         this->wt = wt;
      }
   };
```

```cpp
bool hasPath(vector<Edge> graph [], int src , int dest,
vector<bool> &visited){
   if(src == dest ){
      return true;
   }
   visited[src] = true;
   for(auto e : graph[src]){
      if(visited[e.nbr] == false){
          bool hasNeighbour = hasPath(graph,e.nbr,dest,visited);
          if(hasNeighbour == true ){
            return true;
          }
      }
   }
   return false;
}

int main(){
   int ver{};
   cin>> ver ;
   vector<Edge> graph [ver]; //-> array of vector
//   vector < vector < Edge > >graph(ver);



   int edges;
   cin >>edges;
   for(int i = 0 ; i < edges; i++){
      int v1;
      int v2;
      int wt;
      cin>>v1>>v2>>wt;
      graph[v1].push_back(Edge(v1,v2,wt));
      graph[v2].push_back(Edge(v2,v1,wt));
   }

   // "has path" question between these vertices
   int src{};
   cin>>src;
   int dest{};
   cin>>dest;

  vector<bool> visited (ver,false);
  bool hasP = hasPath( graph, src,dest,visited);
  if(hasP){
     cout<<"true"<<endl;
  }else {
     cout<<"false"<<endl;
  }

}
```

Print All Paths

Easy

1. You are given a graph, a source vertex and a destination vertex. 2. You are required to find and print all paths between source and destination. Print them in lexicographical order. E.g. Check the following paths 012546 01256 032546 03256 The lexicographically smaller path is printed first.

Constraints

None

Format

Input

Input has been managed for you

Output

Check sample output

Example

Sample Input

```
7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
0
6
```

Sample Output

```
0123456
012346
03456
0346
```

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

using namespace std;
   class Edge {
       public:
          int src;
          int nbr;
          int wt;
```

```
      Edge(int src, int nbr, int wt){
         this->src = src;
         this->nbr = nbr;
         this->wt = wt;
      }
   };

   void printAllPath(vector<Edge>graph[],int src, int
dest,vector<bool> &visited,string path){
      if(src == dest ){
         cout<<path<<endl;
         return ;
      }
      visited[src] = true;

      for(auto e : graph[src]){
         if(visited[e.nbr] == false){
             printAllPath(graph,e.nbr,dest,visited,path +
to_string(e.nbr));
         }
      }
      visited[src] = false;

   }

   int main(){
      int vtces;
      cin>>vtces;
      vector<Edge>graph[vtces];

      int edges;
      cin>>edges;
      for(int i = 0; i < edges; i++){
         int v1 ;
         int v2 ;
         int wt ;
         cin>>v1>>v2>>wt;
         graph[v1].push_back( Edge(v1, v2, wt));
         graph[v2].push_back( Edge(v2, v1, wt));
      }

      int src;
      cin>>src;
      int dest;
      cin>>dest;
      // write your code here
      vector<bool>visited(vtces,false);
      string path = to_string(src);
      printAllPath(graph ,src,dest,visited, path);

    }
```

Multisolver - Smallest, Longest, Ceil, Floor, Kthlargest Path

Easy

1. You are given a graph, a src vertex and a destination vertex.
2. You are give a number named "criteria" and a number "k".
3. You are required to find and print the values of
3.1 Smallest path and it's weight separated by an "@"
3.2 Largest path and it's weight separated by an "@"
3.3 Just Larger path (than criteria in terms of weight) and it's weight separated by an "@"
3.4 Just smaller path (than criteria in terms of weight) and it's weight separated by an "@"
3.5 Kth largest path and it's weight separated by an "@"

Constraints

None

Format

Input

Input has been managed for you

Output

Check sample output

Example

Sample Input

```
7
9
0 1 10
1 2 10
2 3 10
0 3 40
3 4 2
4 5 3
5 6 3
4 6 8
2 5 5
0
6
30
4
```

Sample Output

```
Smallest Path = 01256@28
Largest Path = 032546@66
Just Larger Path than 30 = 012546@36
Just Smaller Path than 30 = 01256@28
4th largest path = 03456@48
```

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

using namespace std;

class Edge {
public:
  int src;
  int nbr;
  int wt;
```

```
  Edge(int src, int nbr, int wt) {
    this->src = src;
    this->nbr = nbr;
    this->wt = wt;
  }
};


struct myComp {
  bool operator()(
    pair<int, string>& a,
    pair<int, string>& b)
  {
    return a.first > b.first;
  }
};
string spath;
int spathwt = INT_MAX;
string lpath;
int lpathwt = INT_MIN;
string cpath;
int cpathwt = INT_MAX;
string fpath;
int fpathwt = INT_MIN;

priority_queue<pair<int, string>, vector<pair<int, string>>,
myComp> pq;

void multisolver(vector<Edge> graph[], int src, int dest,
                   vector<bool> visited, int criteria, int k,
string psf, int wsf) {
  //write your code here
  if(src == dest ){
    // cout<<psf << " "<<wsf<<endl;
    if(wsf < spathwt) {
      spath = psf;
      spathwt = wsf;
    }
    if(wsf > lpathwt) {
      lpath = psf;
      lpathwt = wsf;

    }
    if(wsf > criteria && wsf < cpathwt){
      cpath = psf;
      cpathwt = wsf;

    }
    if(wsf < criteria && wsf > fpathwt){
      fpath = psf;
      fpathwt = wsf;
```

```
    }

     pair<int ,string> np ;
     np.first = wsf;
     np.second = psf;
     pq.push(np);
     if(pq.size()> k){
       pq.pop();
     }
  }

  visited[src] = true;
  for(Edge e : graph[src]){
    if(visited[e.nbr] == false){
      multisolver(graph,e.nbr,dest,visited,criteria,k,psf +
to_string(e.nbr),wsf +e.wt );
    }
  }
  visited[src] = false ;//

}

int main() {

  int vtces;
  cin >> vtces;
  vector<Edge>graph[vtces];

  int edges;
  cin >> edges;
  for (int i = 0; i < edges; i++) {
    int v1 ;
    int v2 ;
    int wt ;
    cin >> v1 >> v2 >> wt;
    graph[v1].push_back( Edge(v1, v2, wt));
    graph[v2].push_back( Edge(v2, v1, wt));
  }

  int src;
  cin >> src;
  int dest;
  cin >> dest;

  int criteria;
  cin >> criteria;
  int k;
  cin >> k;

  vector<bool> visited(vtces, false);
```

```
  multisolver(graph, src, dest, visited, criteria, k,
to_string(src) , 0);

  cout << "Smallest Path = " << spath << "@" << spathwt << endl;
  cout << "Largest Path = " << lpath << "@" << lpathwt << endl;
  cout << "Just Larger Path than " << criteria << " = " << cpath
<< "@" << cpathwt << endl;
  cout << "Just Smaller Path than " << criteria << " = " << fpath
<< "@" << fpathwt << endl;
  cout << k << "th largest path = " << pq.top().second << "@" <<
pq.top().first << endl;

  return 0;
}
```

Get Connected Components Of A Graph

Easy

1. You are given a graph. 2. You are required to find and print all connected components of the graph.

Constraints

None

Format

Input

Input has been managed for you

Output

Check the sample output

Example

Sample Input

```
7
5
0 1 10
2 3 10
4 5 10
5 6 10
4 6 10
```

Sample Output

```
[[0, 1], [2, 3], [4, 5, 6]]
```

```
#include<bits/stdc++.h>
using namespace std;
   class Edge {
       public:
          int src;
          int nbr;
          int wt;

      Edge(int src, int nbr, int wt){
         this->src = src;
         this->nbr = nbr;
         this->wt = wt;
      }
   };
   void getAComponent(vector<Edge> graph[] , int i,
vector<bool>&visited ,vector<int> &component){
      visited [i] = true;
      component.push_back(i);
      for(Edge e : graph[i]){
        if(visited[e.nbr] == false){
          getAComponent(graph,e.nbr,visited,component);
        }
      }
   }
```

```
   vector<vector<int>> getConnectedComponents(vector<Edge>
graph[],int v){
        vector<vector<int>> ans;
        vector<int> component;
        vector<bool > visited (v,false);
        for(int i{};i < v;i++){
           //  cout<<"hh"<<endl;
          if(visited[i] == false){

            getAComponent(graph,i,visited,component);
            ans.push_back(component);
            component.clear();
          }
        }
        return ans ;
   }
   int main(){
      int vtces;
      cin>>vtces;
      vector<Edge>graph[vtces];

      int edges;
      cin>>edges;
      for(int i = 0; i < edges; i++){
         int v1 ;
         int v2 ;
         int wt ;
         cin>>v1>>v2>>wt;
         graph[v1].push_back( Edge(v1, v2, wt));
         graph[v2].push_back( Edge(v2, v1, wt));
      }
       vector<vector<int>> comps;

      // write your code here
        comps = getConnectedComponents(graph,vtces);

        cout<<"[";
        for(int i = 0 ; i<comps.size() ; i++){
            cout<<"[";
            for(int j = 0 ; j<comps[i].size() ; j++){
                if(j!=comps[i].size()-1)
                    cout<<comps[i][j]<<", ";
                else
                    cout<<comps[i][j];

            }
            cout<<"]";
            if(i!=comps.size()-1)cout<<", ";
        }
        cout<<"]";
    }
```

Is Graph Connected

Easy

1. You are given a graph.
2. You are required to find and print if the graph is connected (there is a path from every vertex to every other).

Constraints

None

Format

Input

Input has been managed for you

Output

true if the graph is connected, false otherwise

Example

Sample Input

```
7
5
0 1 10
2 3 10
4 5 10
5 6 10
4 6 10
```

Sample Output

```
false
```

```
#include<bits/stdc++.h>
using namespace std;
   class Edge {
       public:
          int src;
          int nbr;
          int wt;

      Edge(int src, int nbr, int wt){
         this->src = src;
         this->nbr = nbr;
         this->wt = wt;
      }
   };

   void visitAComponent(vector<Edge> graph[] , int i,
vector<bool>&visited){
      visited [i] = true;
      for(Edge e : graph[i]){
        if(visited[e.nbr] == false){
          visitAComponent(graph,e.nbr,visited);
        }
      }
   }
   int main(){
```

```
      int vtces;
      cin>>vtces;
      vector<Edge>graph[vtces];

      int edges;
      cin>>edges;
      for(int i = 0; i < edges; i++){
         int v1 ;
         int v2 ;
         int wt ;
         cin>>v1>>v2>>wt;
         graph[v1].push_back( Edge(v1, v2, wt));
         graph[v2].push_back( Edge(v2, v1, wt));
      }

      int src;
      cin>>src;
      int dest;
      cin>>dest;
      // write your code here
      int n{};
      vector<bool > visited (vtces,false);
       for(int i{};i < vtces;i++){
         if(visited[i] == false){
             n++;
             if(n > 1){
                  break;
             }
           visitAComponent(graph,i,visited);
         }

       }
       if(n> 1){
          cout<<"false"<<endl;
       }else{
          cout<<"true"<<endl;
       }
    }
    /*
    #include<bits/stdc++.h>
using namespace std;
   class Edge {
       public:
          int src;
          int nbr;
          int wt;

      Edge(int src, int nbr, int wt){
         this->src = src;
         this->nbr = nbr;
         this->wt = wt;
      }
```

```
   };
   void getAComponent(vector<Edge> graph[] , int i,
vector<bool>&visited ,vector<int> &component){
      visited [i] = true;
      component.push_back(i);
      for(Edge e : graph[i]){
        if(visited[e.nbr] == false){
          getAComponent(graph,e.nbr,visited,component);
        }
      }
   }
   vector<vector<int>> getConnectedComponents(vector<Edge>
graph[],int v){
       vector<vector<int>> ans;
       vector<int> component;
       vector<bool > visited (v,false);
       for(int i{};i < v;i++){
          //  cout<<"hh"<<endl;
         if(visited[i] == false){

           getAComponent(graph,i,visited,component);
           ans.push_back(component);
           component.clear();
         }
       }
       return ans ;
   }
   int main(){
      int vtces;
      cin>>vtces;
      vector<Edge>graph[vtces];

      int edges;
      cin>>edges;
      for(int i = 0; i < edges; i++){
         int v1 ;
         int v2 ;
         int wt ;
         cin>>v1>>v2>>wt;
         graph[v1].push_back( Edge(v1, v2, wt));
         graph[v2].push_back( Edge(v2, v1, wt));
      }


       vector<vector<int>> comps;

      // write your code here
        comps = getConnectedComponents(graph,vtces);

        cout<<"[";
        for(int i = 0 ; i<comps.size() ; i++){
            cout<<"[";
```

```
        for(int j = 0 ; j<comps[i].size() ; j++){
            if(j!=comps[i].size()-1)
                cout<<comps[i][j]<<", ";
            else
                cout<<comps[i][j];

        }
        cout<<"]";
        if(i!=comps.size()-1)cout<<", ";
    }
    cout<<"]";
}
*/
```

Number Of Islands

Easy

1. You are given a 2d array where 0's represent land and 1's represent water.
   Assume every cell is linked to it's north, east, west and south cell.
2. You are required to find and count the number of islands.

Constraints

None

Format

Input

Input has been managed for you

Output

Number of islands

Example

Sample Input

```
8
8
0 0 1 1 1 1 1 1
0 0 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 0 0 0 1 1 0
1 1 1 1 0 1 1 0
1 1 1 1 0 1 1 0
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 0
```

Sample Output

```
3
```

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

using namespace std;
void visitLand(vector<vector<int>> arr ,vector<vector<bool>>
&visited,int i, int j){
      if(i < 0 || j < 0 || i >= arr.size() || j >= arr[0].size()
|| visited[i][j] == true || arr[i][j] == 1){
        return ;
      }
      visited [i][j] = true;
       visitLand(arr,visited,i,j+1);
       visitLand(arr,visited,i+1,j);
       visitLand(arr,visited,i,j-1);
       visitLand(arr,visited,i-1,j);
   }
int main()
{
    int n, m;
    cin >> n;
    cin >> m;

    vector<vector<int>> arr;
```

```
    for (int i = 0; i < n; i++)
    {
        vector<int> ans;
        for (int j = 0; j < m; j++)
        {
            int res;
            cin >> res;
            ans.push_back(res);
        }
        arr.push_back(ans);
    }

    //write your code here
    vector<vector<bool>> visited (n , vector<bool>(n,false));
    int numberOfIsland {};
    for(int i{};i<n;i++){
        for(int j{} ; j < arr[i].size() ; j++) {
            if(arr[i][j] == 0 && visited[i][j] == false){
              numberOfIsland++;
              visitLand(arr,visited,i,j);
            }
        }
    }

    cout<<numberOfIsland<<endl;

}
```

Perfect Friends

Easy

1. You are given a number n (representing the number of students). Each student will have an id from 0 to n - 1.
2. You are given a number k (representing the number of clubs)
3. In the next k lines, two numbers are given separated by a space. The numbers are ids of students belonging to same club.
4. You have to find in how many ways can we select a pair of students such that both students are from different clubs.

Constraints

None

Format

Input

Input has been managed for you

Output

Check the sample output

Example

Sample Input

```
7
5
0 1
2 3
4 5
5 6
4 6
```

Sample Output

```
16
```

```
#include<bits/stdc++.h>
using namespace std;
class Edge{
    public:
        int src;
        int nbr;
        Edge(int src, int nbr){
            this->src = src;
            this->nbr = nbr;
        }
};
void getAComponent(vector<Edge> graph[] , int i,
vector<bool>&visited ,vector<int> &component){
      visited [i] = true;
      component.push_back(i);
      for(Edge e : graph[i]){
        if(visited[e.nbr] == false){
          getAComponent(graph,e.nbr,visited,component);
        }
      }
}
```

```
int main(){
   int n;
   cin>>n;

   int k;
   cin>>k;

   // write your code here
   vector<Edge> graph [n];
   for(int i {};i<k;i++) {
     int u;
     int v;
     cin>>u>>v;
     graph[u].push_back(Edge(u,v));
     graph[v].push_back(Edge(v,u));
   }


    vector<vector<int>> c_comp;
    vector<int> component;
    vector<bool > visited (n,false);

    for(int i{};i < n;i++){
      if(visited[i] == false){
        getAComponent(graph,i,visited,component);
        c_comp.push_back(component);
        component.clear();
      }
    }
    // cout<<" sizse "<<c_comp.size()<<endl;
    // for(int i{}; i < c_comp.size() ;i++){
    //     for(int j {0} ; j < c_comp[i].size() ; j++){
    //         cout<<c_comp[i][j]<<" ";
    //     }
    //     cout<<endl;
    // }
    int ans {};
    for(int i{}; i < c_comp.size() ;i++){
        for(int j {i+1} ; j < c_comp.size() ; j++){
            ans += c_comp[i].size() * c_comp[j].size();
        }
    }
    cout<<ans<<endl;


   return 0;
}
```

Hamiltonian Path And Cycle

Easy

1. You are given a graph and a src vertex.
2. You are required to find and print all hamiltonian paths and cycles starting from src. The cycles must end with "*" and paths with a "."

Note -> A hamiltonian path is such which visits all vertices without visiting any twice. A hamiltonian path becomes a cycle if there is an edge between first and last vertex.
Note -> Print in lexicographically increasing order.

Constraints

None

Format

Input

Input has been managed for you

Output

Check sample output

Example

Sample Input

```
7
9
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
2 5 10
0
```

Sample Output

```
0123456.
0123465.
0125643*
0346521*
```

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <stack>
#include <unordered_set>

using namespace std;

class Edge{
    public:
        int src;
        int nbr;
        int wt ;
```

```
        Edge(int src, int nbr,int wt){
            this->src = src;
            this->nbr = nbr;
            this->wt = wt;
        }
};

template<typename T>
bool allTrue(std::vector<T> const &v) {

    for(int i{} ; i<v.size();i++){
        if(v[i] == false){
            return false;
        }
    }
    return true;
}

void printHamiltonianPathAndCycle( vector<Edge>*graph ,int
vtces,int src, int u,unordered_set<int> &visited, string path){
    // cout<<u<<".  n"<<endl;
    visited.insert(u);
    if(visited.size() == vtces){
        bool for_cycle = false;
        for(Edge e : graph[src]){
            if(e.nbr == u){
                for_cycle = true;
                break;
            }
        }
        if(for_cycle){
            cout<<path<<"*"<<endl;
        }else{
            cout<<path<<"."<<endl;
        }
    }
    for(Edge e : graph[u]){
        if(visited.find(e.nbr) == visited.end()){

printHamiltonianPathAndCycle(graph,vtces,src,e.nbr,visited,path +
to_string(e.nbr));

        }
    }
    visited.erase(u);
}
int main(){

    int vtces;
    cin>>vtces;
    vector<Edge>graph[vtces];
```

```
    int edges;
    cin>>edges;
    for(int i = 0; i < edges; i++){
        int v1 ;
        int v2 ;
        int wt ;
        cin>>v1>>v2>>wt;
        graph[v1].push_back( Edge(v1, v2, wt));
        graph[v2].push_back( Edge(v2, v1, wt));
    }
    int src{};
    cin>> src;
    // vector<bool> visited(vtces, false);
    unordered_set<int> visited;

printHamiltonianPathAndCycle(graph,vtces,src,src,visited,to_string
(src));




    return 0;
}
```

Knights Tour

Easy

1. You are given a number n, the size of a chess board.
2. You are given a row and a column, as a starting point for a knight piece.
3. You are required to generate the all moves of a knight starting in (row, col) such that knight visits all cells of the board exactly once.
4. Complete the body of printKnightsTour function - without changing signature - to calculate and print all configurations of the chess board representing the route of knight through the chess board. Use sample input and output to get more idea.

Note -> When moving from (r, c) to the possible 8 options give first precedence to (r - 2, c + 1) and move in clockwise manner to explore other options.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
n = 5
0 <= row < n
0 <= col < n
```

Format

Input

A number n
A number row
A number col

Output

All configurations of the chess board representing route of knights through the chess board starting in (row, col)
Use displayBoard function to print one configuration of the board.

Example

Sample Input

```
5
2
0
```

Sample Output

```
25 2 13 8 23
12 7 24 3 14
1 18 15 22 9
6 11 20 17 4
19 16 5 10 21

19 2 13 8 21
12 7 20 3 14
1 18 15 22 9
6 11 24 17 4
25 16 5 10 23

25 2 13 8 19
12 7 18 3 14
```

```
1 24 15 20 9
6 11 22 17 4
23 16 5 10 21

19 2 13 8 25
12 7 18 3 14
1 20 15 24 9
6 11 22 17 4
21 16 5 10 23

21 2 17 8 19
12 7 20 3 16
1 22 13 18 9
6 11 24 15 4
23 14 5 10 25

23 2 17 8 25
12 7 24 3 16
1 22 13 18 9
6 11 20 15 4
21 14 5 10 19

……..many more
```

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

//function to display the 2-d array
void display(vector<vector<int>> &chess){
    for(int i=0;i<chess.size();i++){
        for(int j=0;j<chess.size();j++){
            cout << chess[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

void printKnightsTour(vector<vector<int>> &chess,int n,int r,int
c,int upcomingMove){
  //write your code here
    if(r < 0 || c < 0 || r >= n || c >= n || chess[r][c] != 0 ){
        return;
    }

    if(upcomingMove == 25 ){
        chess[r][c] = upcomingMove;
        display(chess);
        chess[r][c] = 0;
        return;
    }
```

```
    chess[r][c] = upcomingMove;
        printKnightsTour(chess,n,r-2,c+1,upcomingMove + 1);
        printKnightsTour(chess,n,r-1,c+2,upcomingMove + 1);
        printKnightsTour(chess,n,r+1,c+2,upcomingMove + 1);
        printKnightsTour(chess,n,r+2,c+1,upcomingMove + 1);
        printKnightsTour(chess,n,r+2,c-1,upcomingMove + 1);
        printKnightsTour(chess,n,r+1,c-2,upcomingMove + 1);
        printKnightsTour(chess,n,r-1,c-2,upcomingMove + 1);
        printKnightsTour(chess,n,r-2,c-1,upcomingMove + 1);
        chess[r][c] = 0;
    return ;

}

int main(){
    int n{};
    cin >>n;

    vector<vector <int> > chess (n,vector<int>(n));
    int r{};
    int c{};
    cin>>r;
    cin>>c;
    printKnightsTour(chess, n,r,c, 1);


    return 0;
}
```

Breadth First Traversal

Easy

1. You are given a graph, and a src vertex.
2. You are required to do a breadth first traversal and print which vertex is reached via which path, starting from the src.

Note -> for output, check the sample output and question video.

Constraints

None

Format

Input

Input has been managed for you

Output

Check the sample output

Example

Sample Input

```
7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
2
```

Sample Output

```
2@2
1@21
3@23
0@210
4@234
5@2345
6@2346
```

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

class Edge
{
public:
  int src = 0;
  int nbr = 0;
```

```
  Edge(int src, int nbr)
  {
    this->src = src;
    this->nbr = nbr;
  }
};




int main() {
  int vtces;
  cin >> vtces;
  vector<vector<Edge>> graph(vtces, vector<Edge>());


  int edges;
  cin >> edges;

  for (int i = 0; i < edges; i++ ) {
    int u, v, w;
    cin >> u >> v >> w;
    graph[u].push_back(Edge(u, v));
    graph[v].push_back(Edge(v, u));
   }

  int src;
  cin >> src;
  // write your code here
  //REMOVE , MARK, WORK , ADD unMARKED NEIGHBOUR
  queue<pair<int,string>> q;
  pair<int,string> s(src,to_string(src));
  q.push(s);
  vector<bool> visited(vtces, false);
  visited[src] = true;
  // cout<<"reached"<<endl;
  while(!q.empty()){
        pair<int, string> fn = q.front();q.pop();
        cout<<fn.first<<"@"<<fn.second<<endl;
        for(Edge e: graph[fn.first]){
            if(visited[e.nbr] == false){
                visited[e.nbr] = true;
                pair<int,string>  np (e.nbr,fn.second +
to_string(e.nbr));
                q.push(np);
            }
        }
  }

  return 0;
}
```

Is Graph Cyclic

Easy

1. You are given a graph.
2. You are required to find and print if the graph is cyclic.

Constraints

None

Format

Input

Input has been managed for you

Output

true if the graph is cyclic, false otherwise

Example

Sample Input

```
7
6
0 1 10
1 2 10
2 3 10
3 4 10
4 5 10
5 6 10
```

Sample Output

```
false
```

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

class Edge
{
public:
  int src = 0;
  int nbr = 0;

  Edge(int src, int nbr)
  {
    this->src = src;
    this->nbr = nbr;
  }
};


int main() {
  int vtces;
  cin >> vtces;
  vector<vector<Edge>> graph(vtces, vector<Edge>());
```

```cpp
  int edges;
  cin >> edges;

  for (int i = 0; i < edges; i++ ) {
    int u, v, w;
    cin >> u >> v >> w;

    graph[u].push_back(Edge(u, v));
    graph[v].push_back(Edge(v, u));

  }

 // write your code here
 //REMOVE , MARK, WORK(if unmarked) , ADD unMARKED NEIGHBOUR =>
sir strategy
 // using sir strategy
 queue<int> q;
  vector<bool> visited(vtces, false);
  // cout<<"reached"<<endl;
  bool is_cycle = false;
  for(int i{}; i<vtces;i++){
    //if graph is not connected
    if(visited[i] == false && is_cycle == false){
      q.push(i);

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

        if(visited[fn] == true){
          is_cycle = true;
          break;
        }

        visited[fn] = true;
        // cout<<fn.first<<"@"<<fn.second<<endl;
        for(Edge e: graph[fn]){
            if(visited[e.nbr] == false){
                q.push(e.nbr);
            }
        }
      }
    }
  }
  if(is_cycle){
    cout<<"true"<<endl;
  }else{
    cout<<"false"<<endl;
  }
  return 0;
}
```

Is Graph Bipartite

Easy

1. You are given a graph.
2. You are required to find and print if the graph is bipartite

Note -> A graph is called bipartite if it is possible to split it's vertices in two sets of mutually exclusive and exhaustive vertices such that all edges are across sets.

Constraints

None

Format

Input

Input has been managed for you

Output

true if the graph is bipartite, false otherwise

Example

Sample Input

```
7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
```

Sample Output

```
false
```

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

class Edge
{
public:
  int src = 0;
  int nbr = 0;

  Edge(int src, int nbr)
  {
    this->src = src;
    this->nbr = nbr;
  }
};
class Pair{
  public:
```

```
  int u;
  string path;
  int level;
  Pair(int u, string path ,int level){
    this->u = u;
    this->path = path ;
    this->level = level;
  }

};
bool isAComponentBipartite(vector<vector<Edge>> graph, int i,
vector<int> &visited){
    queue<Pair> q;
    Pair p (i,to_string(i),0) ;
    q.push(p);

      while(!q.empty()){
        Pair fn = q.front();q.pop();

        if(visited[fn.u] != -1){ //putting level in the visited
          if(visited[fn.u] != fn.level){
            return false;
          }
          else{
            continue;
          }
        }

        visited[fn.u] = fn.level;
        for(Edge e: graph[fn.u]){
            if(visited[e.nbr] == -1){
              Pair np (e.nbr,fn.path+to_string(e.nbr) ,fn.level +
1) ;
                q.push(np);
            }
        }
      }
      return true;
}

int main() {
  int vtces;
  cin >> vtces;
  vector<vector<Edge>> graph(vtces, vector<Edge>());


  int edges;
  cin >> edges;

  for (int i = 0; i < edges; i++ ) {
    int u, v, w;
    cin >> u >> v >> w;
```

```
    graph[u].push_back(Edge(u, v));
    graph[v].push_back(Edge(v, u));

  }

  // write your code here
  /*
  acyclic graph is always biparted
  in cyclic is length of cycle is even then it is biparted
  otherwise if the lenght of cycle is odd it is not biparted
  */
  //REMOVE , MARK, WORK(if unmarked) , ADD unMARKED NEIGHBOUR =>
sir strategy
 // using sir strategy

  vector<int> visited(vtces, -1);//-1 implies not visited
  // cout<<"reached"<<endl;
  bool is_biparted = false;
  for(int i{}; i<vtces;i++){
    //if graph is not connected
    if(visited[i] == -1){
      is_biparted = isAComponentBipartite(graph,i,visited);
    }
    if(!is_biparted){
      cout<<"false"<<endl;
      return 0;
    }
  }


  //otherwise it will be true
    cout<<"true"<<endl;

   return 0;
}
```

Spread Of Infection

Easy

1. You are given a graph, representing people and their connectivity.
2. You are also given a src person (who got infected) and time t.
3. You are required to find how many people will get infected in time t, if the infection spreads to neighbors of infected person in 1 unit of time.

Constraints

None

Format

Input

Input has been managed for you

Output

count of people infected by time t

Example

Sample Input

```
7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
6
3
```

Sample Output

```
4
```

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

using namespace std;

class Edge
{
public:
  int src = 0;
  int nbr = 0;

  Edge(int src, int nbr)
  {
    this->src = src;
    this->nbr = nbr;
  }
};
class Pair{
```

```
  public:
  int u;
  string path;
  int level;
  Pair(int u, string path ,int level){
    this->u = u;
    this->path = path ;
    this->level = level;
  }

};
int inThisComponent(vector<vector<Edge>> graph, int i,int time,
vector<bool> &visited){
    queue<Pair> q;
    Pair p (i,to_string(i),1) ; //starting with level 1
    // first person also takes one time
    q.push(p);
    int infected_people{};
      while(!q.empty()){
        Pair fn = q.front();q.pop();
        if(fn.level > time){ //putting level in the visited
          break;
        }
        if(visited[fn.u] == true){ //putting level in the visited
          continue;
        }
        // cout<<fn.u<<"->"<<fn.level<<" "<<visited[fn.u]<<endl;
        infected_people++;
        visited[fn.u] = true;
        for(Edge e: graph[fn.u]){
            if(visited[e.nbr] == false){
              Pair np (e.nbr,fn.path+to_string(e.nbr) ,fn.level +
1) ;
                q.push(np);
            }
        }
        // cout<<"q"<<q.size()<<endl;
      }
      return infected_people;
}

int main() {

  int vtces;
  cin >> vtces;
  vector<vector<Edge>> graph(vtces, vector<Edge>());


  int edges;
  cin >> edges;

  for (int i = 0; i < edges; i++ ) {
```

```
    int u, v, w;
    cin >> u >> v >> w;

    graph[u].push_back(Edge(u, v));
    graph[v].push_back(Edge(v, u));

  }
  int src,t;
  cin >> src;
  cin >> t;

  //write your code here
  vector<bool> visited(vtces, false);//-1 implies not visited

  int numberOfInfectedPeople {};
  numberOfInfectedPeople = inThisComponent(graph,src,t,visited);
  // for(int i{}; i<vtces;i++){
  //   //if graph is not connected
  //   if(visited[i] == -1){
  //     numberOfInfectedPeople +=
inThisComponent(graph,i,visited);
  //   }
  // }

cout<<numberOfInfectedPeople<<endl;

  return 0;
}
```

Shortest Path In Weights

Easy

1. You are given a graph and a source vertex. The vertices represent cities and the edges represent distance in kms.
2. You are required to find the shortest path to each city (in terms of kms) from the source city along with the total distance on path from source to destinations.

Note -> For output, check the sample output and question video.

Constraints

None

Format

Input

Input has been managed for you

Output

Check the sample output

Example

Sample Input

```
7
9
0 1 10
1 2 10
2 3 10
0 3 40
3 4 2
4 5 3
5 6 3
4 6 8
2 5 5
0
```

Sample Output

```
0 via 0 @ 0
1 via 01 @ 10
2 via 012 @ 20
5 via 0125 @ 25
4 via 01254 @ 28
6 via 01256 @ 28
3 via 012543 @ 30
```

```
//I think ans is correct , but it doesn't "match" with pepcoding
    //solution
#include <iostream>
#include <vector>
#include <string>
#include <unordered_set>
#include <queue>
#include <algorithm>
#include <limits.h>
```

```cpp
using namespace std;

class Edge
{
public:
  int src = 0;
  int nbr = 0;
  int wt = 0;

  Edge(int src, int nbr, int wt)
  {
    this->src = src;
    this->nbr = nbr;
    this->wt = wt;
  }
};


class Pair{
  public:
    int u;
    int wsf;
    string path;
    Pair (int u, int wsf, string path){
      this-> u = u;
      this-> wsf = wsf; // from source
      this-> path = path;
    }
};
struct myComp {
  bool operator()(Pair& a, Pair& b) {
    return a.wsf > b.wsf;
  }
};

void dijkstraAlgo(vector<vector<Edge>> graph,int src,
                  vector<bool> &visited,vector<int> &distance) {
    priority_queue<Pair, vector<Pair>, myComp> pq;
    distance[src] = 0;
    Pair s (src,0,to_string(src));
    pq.push(s);

    while(pq.empty() == false){
      Pair front = pq.top();pq.pop();
      if(visited[front.u]){
        continue;
      }
      visited[front.u] = true;
      cout<<front.u<<" via "<<front.path<<" @ "<<front.wsf<<endl;
      for(Edge e: graph[front.u]){
```

```
        if(visited[e.nbr] == false){
          int w = min(distance[front.u] + e.wt , distance[e.nbr]);
          distance[e.nbr] = w;
          Pair np (e.nbr,w,front.path + to_string(e.nbr));
          pq.push(np);
        }
      }

    }

}


int main() {

  int vtces;
  cin >> vtces;
  vector<vector<Edge>> graph(vtces, vector<Edge>());


  int edges;
  cin >> edges;

  for (int i = 0; i < edges; i++ ) {
    int u, v, w;
    cin >> u >> v >> w;

    graph[u].push_back(Edge(u, v, w));
    graph[v].push_back(Edge(v, u, w));

  }
  int src;
  cin >> src;

  //write your code here
  int inf = INT_MAX;
  vector <int> distance(vtces,inf);//similar to visited
  vector<bool> visited(vtces,false);
  dijkstraAlgo(graph,src,visited,distance);

  return 0;
}
```

Minimum Wire Required To Connect All Pcs

Easy

1. You are given a graph and a source vertex. The vertices represent computers and the edges represent length of LAN wire required to connect them.
2. You are required to find the minimum length of wire required to connect all PCs over a network. Print the output in terms of which all PCs need to be connected, and the length of wire between them.

Note -> For output, check the sample output and question video.

Constraints

None

Format

Input

Input has been managed for you

Output

Check the sample output

Example

Sample Input

```
7
8
0 1 10
1 2 10
2 3 10
0 3 40
3 4 2
4 5 3
5 6 3
4 6 8
```

Sample Output

```
[1-0@10]
[2-1@10]
[3-2@10]
[4-3@2]
[5-4@3]
[6-5@3]
```

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

using namespace std;

class Edge
{
public:
  int src = 0;
  int nbr = 0;
  int wt = 0;
```

```
  Edge(int src, int nbr, int wt)
  {
    this->src = src;
    this->nbr = nbr;
    this->wt = wt;
  }
};

struct myComp {
  bool operator()(Edge& a, Edge& b) {
    return a.wt > b.wt;
  }
};

 void primsAlgo(vector<vector<Edge>> graph,int src,
                  vector<bool> &visited) {
    priority_queue<Edge, vector<Edge>, myComp> pq;
    visited[src] = true;
    for(Edge e: graph[src]){
        if(visited[e.nbr] == false){
          // Pair np (e.nbr,front.wsf + e.wt,front.path +
to_string(e.nbr));
          pq.push(e);
        }
      }

    while(pq.empty() == false){
      Edge front = pq.top();pq.pop();
      if(visited[front.nbr]){
        continue;
      }
      visited[front.nbr] = true;

cout<<"["<<front.nbr<<"-"<<front.src<<"@"<<front.wt<<"]"<<endl;
      for(Edge e: graph[front.nbr]){
        if(visited[e.nbr] == false){
          // Pair np (e.nbr,front.wsf + e.wt,front.path +
to_string(e.nbr));
          pq.push(e);
        }
      }

    }

}

int main() {

  int vtces;
  cin >> vtces;
  vector<vector<Edge>> graph(vtces, vector<Edge>());
  int edges;
```

```
  cin >> edges;

  for (int i = 0; i < edges; i++ ) {
    int u, v, w;
    cin >> u >> v >> w;

    graph[u].push_back(Edge(u, v, w));
    graph[v].push_back(Edge(v, u, w));

  }

  //write your code here
  vector<bool> visited(vtces,false);
  primsAlgo(graph,0,visited);


  return 0;
}
```

Order Of Compilation

Easy

1. You are given a directed acyclic graph. The vertices represent tasks and edges represent dependencies between tasks.
2. You are required to find and print the order in which tasks could be done. The task that should be done at last should be printed first and the task which should be done first should be printed last. This is called topological sort. Check out the question video for details.

Topological sort -> A permutation of vertices for a directed acyclic graph is called topological sort if for all directed edges uv, u appears before v in the graph

Note -> For output, check the sample output and question video.

Constraints

None

Format

Input

Input has been managed for you

Output

Check the sample output

Example

Sample Input

```
7
7
0 1
1 2
2 3
0 3
4 5
5 6
4 6
```

Sample Output

```
4
5
6
0
1
2
3
```

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

class Edge
{
public:
  int src = 0;
  int nbr = 0;
```

```
  Edge(int src, int nbr)
  {
    this->src = src;
    this->nbr = nbr;

  }
};
void topologicalSort (vector<vector<Edge>> &graph,int
i,vector<bool>&visited,stack<int> &st){

  visited[i] = true;
  for(Edge e: graph[i]){
    if(visited[e.nbr] == false){
      topologicalSort(graph,e.nbr,visited,st);
    }
  }
  st.push(i);
}


int main() {

  int vtces;
  cin >> vtces;
  vector<vector<Edge>> graph(vtces, vector<Edge>());


  int edges;
  cin >> edges;

  for (int i = 0; i < edges; i++ ) {
    int u, v;
    cin >> u >> v;

    graph[u].push_back(Edge(u, v));
  }

  //write your code here
  vector<bool> visited (vtces,false);
  stack<int> st;
  for(int i{} ;i<vtces ;i++){
    if(visited[i] == false){
      topologicalSort(graph,i,visited,st);
    }
  }

  while(st.empty() == false){
    int a = st.top();st.pop();
    cout<<a<<endl;
  }
  /*
```

```
  why we didn't printed in post order
  that is the reverse of what we want that tell us in which order
works should
  should be done but topological is its reverse

  that's why we used stack in post order




  why we didn't printed in pre order
  because pre say that:- mai jis par dependent hu mai unse pehele
print ho jau
  but isme ye chez satisfy nhi ho rhi  ki jo mujhe par dependent
hai
  wo sare mujhse pehele print ho chuke hai ==> that may create
wrong output

  that's why we used stack in post order
  */

  return 0;
}
```

Iterative Depth First Traversal

Easy

1. You are given a graph, and a source vertex.
2. You are required to do a iterative depth first traversal and print which vertex is reached via which path, starting from the source.

Note -> For output, check the sample output and question video. Iterative depth first traversal should mimic "Reverse preorder" i.e. nbr with highest value should be visited first and should be printed on the way down in recursion.

Constraints

None

Format

Input

Input has been managed for you

Output

Check the sample output

Example

Sample Input

```
7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
2
```

Sample Output

```
2@2
3@23
4@234
6@2346
5@23465
0@230
1@2301
```

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

class Edge
{
public:
  int src = 0;
  int nbr = 0;
  int wt = 0;
```

```
  Edge(int src, int nbr,int wt)
  {
    this->src = src;
    this->nbr = nbr;
    this->wt = wt;

  }
};
void iterativeDfs(vector<vector<Edge>> graph,int src,vector <bool>
visited){

  stack<pair<int,string>> st;
  pair<int,string> s(src,to_string(src));
  st.push(s);
  while(st.empty() == false){
    pair <int,string>top = st.top(); st.pop();
    if(visited[top.first]){
      continue;
    }

    visited[top.first] = true;
    cout<<top.first<<"@"<<top.second<<endl;

    for(Edge e: graph[top.first]){
      if(visited[e.nbr] == false){
          pair<int,string>  np (e.nbr,top.second +
to_string(e.nbr));
          st.push(np);
      }
    }

  }
}

int main() {

  int vtces;
  cin >> vtces;
  vector<vector<Edge>> graph(vtces, vector<Edge>());


  int edges;
  cin >> edges;

  for (int i = 0; i < edges; i++ ) {
    int u, v, w;
    cin >> u >> v >> w;

    graph[u].push_back(Edge(u, v, w));
    graph[v].push_back(Edge(v, u, w));
```

```
  }
    int src;
    cin >> src;


    vector <bool> visited (vtces, false);
    iterativeDfs (graph, src,visited);
  //write your code here

  /*
  why iterative dfs
  because as dfs use recursion function call stack is created
  which don't have much space as they are not on the heap

  in iterative dfs we use stack which we can create on the heap
  */
  return 0;
}
```