DFS
Timp: O(n+m)
Spatiu: O(n)

```
void dfs(int x)
{
   vector_vizitat[x] = true;
   for (int i=0; i<lista_adiacenta[x].size(); ++i)
      if (vector_vizitat[lista_adiacenta[x][i]]==0)
         dfs(lista_adiacenta[x][i]);
}
```

BFS
Timp: O(n+m)

```
void bfs(int x)
{
   memset(distanta_minima,-1, sizeof(distanta_minima));
   vector_vizitat[x]=true;
   coada_bfs.push_back(x);
   distanta_minima[x]=0;

   while(!coada_bfs.empty())
   {
      x = coada_bfs.front();
      coada_bfs.pop_front();

      for (int i=0; i<lista_adiacenta[x].size(); ++i)
      {
         if (distanta_minima[lista_adiacenta[x][i]]<0)
         {
            vector_vizitat[lista_adiacenta[x][i]] = true;
            coada_bfs.push_back(lista_adiacenta[x][i]);
            distanta_minima[lista_adiacenta[x][i]]=distanta_minima[x]+1;
         }
      }
   }
}
```

Sortare Topologica
Timp: O(n+m)
Spatiu: O(n)

```
void sortare_topologica(int x)
{
   vector_vizitat[x]=1;
   for(int i : lista_adiacenta[x])
   {
```

```
        if(vector_vizitat[i]==0)
            sortare_topologica(i);
    }
    lista_sorttop.push_front(x);
}

int main()
{
for(int i=n; i>=1; i--)
    {
        if(vector_vizitat[i]==0)
            sortare_topologica(i);
    }
    while(!lista_sorttop.empty())
        {
            int x=lista_sorttop.front();
            fout<<x<<" ";
            lista_sorttop.pop_front();
        }
}
```

Kosaraju
Time: O(n+m)

```
void DFS1(int x)
{
    vector_vizitat[x] = true;

    for (int i=0; i <lista_adiacenta[x].size(); ++i)
        if (vector_vizitat[lista_adiacenta[x][i]]==0)
            DFS1(lista_adiacenta[x][i]);

     stiva.push_back(x);
}

void DFS2(int x)
{
    vector_vizitat_2[x] = true;

    componente[nr_componente_tari_conexe].push_back(x);

    for (int i = 0; i < lista_adiacenta2[x].size(); ++i)
        if (vector_vizitat_2[lista_adiacenta2[x][i]]==0)
            DFS2(lista_adiacenta2[x][i]);

}

int main()
```

```
{
  for(int i=1; i<=n; i++)
      if(!vector_vizitat[i])
         DFS1(i);

  for(int i=stiva.size()-1;i>=0 ;i--)
   {
     if(!vector_vizitat_2[stiva[i]])
    {
       DFS2(stiva[i]);
       nr_componente_tari_conexe++;
    }
   }
}
```

Tarjan
Time: O(n+m)

Biconex

```
void biconex(int start, int tata)
{
   timp++;
   vizitat[start] = timp;
   low[start] = timp;
   for (int i=0; i<lista_adiacenta[start].size(); i++)
   {
      int conect=lista_adiacenta[start][i];
      //cout<<conect<<" "<<tata<<endl;
      if(conect!=tata)
      {
         if (vizitat[lista_adiacenta[start][i]]==0)
         {
            stackCBC.push({start, lista_adiacenta[start][i]});
            biconex(lista_adiacenta[start][i],start);
            low[start] = min(low[lista_adiacenta[start][i]],low[start]);
            if (low[lista_adiacenta[start][i]]>=vizitat[start])
            {
               set<int> elem;
               elem1 = stackCBC.top().first;
               elem2 = stackCBC.top().second;
               elem.insert(elem1);
               elem.insert(elem2);
               stackCBC.pop();
               while (elem1 != start || elem2 != lista_adiacenta[start][i])
               {
                  elem1 = stackCBC.top().first;
                  elem2 = stackCBC.top().second;
                  elem.insert(elem1);
```

```
                elem.insert(elem2);
                stackCBC.pop();
            }
            componente.push_back(elem);
        }
      }
      else
      {
        low[start] = min(vizitat[lista_adiacenta[start][i]],low[start]);
      }
    }
  }
}
```

Critical Connection

```
class Solution {
public:
  vector<vector<int>> criticalConnections(int n, vector<vector<int>>& connections)
  {
    disc = vector<int>(n);
    low = vector<int>(n);
    for (auto muchie : connections) {
      lista_adiacenta[muchie[0]].push_back(muchie[1]);
      lista_adiacenta[muchie[1]].push_back(muchie[0]);
    }
    dfs_muchie_critica(0, -1);
    return ans;
  }
  void dfs_muchie_critica(int curr, int prev)
  {
    disc[curr] = ++time;
    low[curr] = time;
    for (int i=0; i<lista_adiacenta[curr].size(); i++)
    {
      if (disc[lista_adiacenta[curr][i]] == 0)
      {
        dfs_muchie_critica(lista_adiacenta[curr][i], curr);
        low[curr] = min(low[curr], low[lista_adiacenta[curr][i]]);
      }
      else if (lista_adiacenta[curr][i] != prev)
      {
        low[curr] = min(low[curr], disc[lista_adiacenta[curr][i]]);
      }
      if (low[lista_adiacenta[curr][i]] > disc[curr])
        ans.push_back({curr, lista_adiacenta[curr][i]});
    }
  }
  vector<int> disc;
```

```
    vector<int> low;
    int time = 0;
    vector<vector<int>> ans;
    unordered_map<int, vector<int>> lista_adiacenta;
};
```

Paduri de Multimi Disjuncte
Time: O(n) amortizat

```
int find_node(int x)
{
    while(x!=par[x])
        x=par[x];
    return x;
}

void unite(int x,int y)
{
    int find_x=find_node(x);
    int find_y=find_node(y);
    if(dim[find_x]>=dim[find_y])
    {
        par[find_y]=find_x;
        dim[find_x]+=dim[find_y];
    }
    else
    {
        par[find_x]=find_y;
        dim[find_y]+=dim[find_x];
    }
}
```

Kruskal
Time: O(m log n)

```
int kruskall()
{
    int cost = 0;
    sort(muchii.begin(), muchii.end());
    for(auto muchie : muchii)
    {
        if(find_node(muchie.second.first) != find_node(muchie.second.second))
        {
            unite(muchie.second.first, muchie.second.second);
            cost += muchie.first;
            sol_m.push_back({muchie.second.first, muchie.second.second});
        }
    }
```

```
    return cost;
}
```

Dijkstra
Time: O(n+m log n)

```
void dijkstra(int startNod)
{
    d[startNod]=0;
    priority_queue<pair<int,int>> q;
    q.push({0,startNod});
    while(!q.empty())
    {
        int nod=q.top().second;
        q.pop();
        if(viz[nod]==true) continue;
        else viz[nod]=true;
        for(auto vecin : g[nod])
        {
            if(d[nod]+vecin.second<d[vecin.first])
            {
                d[vecin.first]=d[nod]+vecin.second;
                q.push({-d[vecin.first],vecin.first});
            }
        }
    }
}
```

BellmanFord
Time: O(mn)

```
void bellmanFord(int startNod, int n)
{
    d[startNod]=0;
    priority_queue<pair<int,int>> q;
    q.push({0,startNod});
    while(!q.empty())
    {

        int nod=q.top().second;
        viz[nod]++;
        if(viz[nod]>=n)
        {
            fout<<"Ciclu negativ!";
            ok=0;
            return;
```

```
        }
        q.pop();
        for(auto vecin : g[nod])
        {
            if(d[nod]+vecin.second<d[vecin.first])
            {
                d[vecin.first]=d[nod]+vecin.second;
                q.push({-d[vecin.first],vecin.first});
            }
        }
    }
}
```

Roy-Floyd
Time: O(n^3)
Space: O(n^2)

```
for(int i=1; i<=n; i++)
        for(int j=1; j<=n; j++)
        {
            fin>>m[i][j];
            if(m[i][j]==0) m[i][j]=INFINIT;
        }

    for(int k=1; k<=n; k++)
        for(int i=1; i<=n; i++)
            for(int j=1; j<=n; j++)
                if(m[i][k]!=INFINIT and m[k][j]!=INFINIT and m[i][j]>m[i][k]+m[k][j])
                    m[i][j]=m[i][k]+m[k][j];
    for(int i=1; i<=n; i++)
    {
        for(int j=1; j<=n; j++)
        {
            if(i!=j and m[i][j]!=INFINIT) fout<<m[i][j]<<" ";
            else fout<<"0 ";
            if(j==n) fout<<'\n';
        }
    }
```

DFS Darb
Time: O(m)

```
void DFS(int curr)
{
    for(int i = 0; i < lista_adiacenta[curr].size(); i++)
    {
        if(vizitat[lista_adiacenta[curr][i]] == 0)
```

```
    {
        vizitat[lista_adiacenta[curr][i]]=vizitat[curr];
        vizitat[lista_adiacenta[curr][i]]++;
        if(vizitat[lista_adiacenta[curr][i]] > dist_max)
        {
            nod_departe = lista_adiacenta[curr][i];
            dist_max=vizitat[lista_adiacenta[curr][i]];
        }
        DFS(lista_adiacenta[curr][i]);
    }
```

Ford-Fulkerson
Time: O(n*flux)

```
vector<int> lista_adiacenta[flowmax];
bool BFSflow(int s, int fin, int f[flowmax][flowmax], int c[flowmax][flowmax], int tata[flowmax])
{
    bool vizitat[flowmax]={false};
    queue<int> q;
    q.push(s);
    vizitat[s] = true;
    while(!q.empty())
    {
        int nod = q.front();
        q.pop();
        for(auto i : lista_adiacenta[nod])
        {
            if(c[nod][i]-f[nod][i]>0 and vizitat[i] == false)
            {
                vizitat[i] = true;
                q.push(i);
                tata[i] = nod;
                if(i == fin)
                    return true;
            }
        }
    }

    return false;
}



int main()
{
    int f[flowmax][flowmax]={0};
    int c[flowmax][flowmax]={0};
    int tata[flowmax]={0};
```

```
    int n, m;
    fin >> n >> m;
    for(int i = 1; i <= m; ++i)
    {
        int a, b, fluxx;
        fin >> a >> b >> fluxx;
        lista_adiacenta[a].push_back(b);
        lista_adiacenta[b].push_back(a);
        c[a][b] = fluxx;
    }
    int flow = 0;

    while(BFSflow(1,n,f,c,tata)==true)
    {
        int fmin = INT_MAX;
        int nod = n;
        while(nod != 1)
        {
            fmin = min(fmin, c[tata[nod]][nod] - f[tata[nod]][nod]);
            nod = tata[nod];
        }
        flow += fmin;
        nod = n;
        while(nod != 1)
        {
            f[tata[nod]][nod] += fmin;
            f[nod][tata[nod]] -= fmin;
            nod = tata[nod];
        }

    }

    fout << flow;
    return 0;
}
```

Hopcroft Karp
Time: O(sqrt(m) * n)

```
bool cuplaj(int k)
{
    if (vector_vizitat[k]==1) return false;
    vector_vizitat[k]=1;
    for (int i:lista_adiacenta[k])
        if (dr[i] == 0)
        {
            st[k] = i;
            dr[i] = k;
```

```
            return true;
        }
    for (int i : lista_adiacenta[k])
        if (cuplaj(dr[i]))
        {
            st[k] = i;
            dr[i] = k;
            return true;
        }
    return false;
}

int main()
{
    fin>>n1>>n2>>m;
    for (int i=1; i<=m; i++)
    {
        int a,b;
        fin>>a>>b;
        lista_adiacenta[a].push_back(b);
    }
    bool ok=1;
    while(ok)
    {
        ok=0;
        memset(vector_vizitat, 0, sizeof(vector_vizitat));
        for (int i=1; i<=n1; i++)
            if(st[i]==0 and cuplaj(i)==true)
            {
                cupmax++;
                ok=1;
            }
    }
    fout<<cupmax<<"\n";
    for(int i=1; i<=n1; i++)
        if(st[i]>0) fout<<i<<" "<<st[i]<<'\n';
    return 0;
}
```

Euler
Conditii: conex si toate nodurile au grad par
Time: O(n+m)?

```
euler (nod v)
     cat timp (v are vecini)
        w = un vecin aleator al lui v
        sterge_muchie (v, w)
        euler (w)
     sfarsit cat timp
```

adauga v la ciclu

Ciclu Hamiltonian
Time: O(m*2^n)

```
int main()
{
   vector <vector <pair<int, int>>> costuri_hamilton;
   fin>>n>>m;
   costuri_hamilton.resize(n+1);
   for(int i=1; i <=m; i++)
   {
      int x,y,ct;
      fin>>x>>y>>ct;
      costuri_hamilton[x].push_back({y, ct});
   }

   int cost_final = INFINIT;
   int costuri[(1<<n)+5][n+5];

   for(int i = 0; i < (1<<n); i++)
      for(int j = 0; j < n; j++)
         costuri[i][j] = INFINIT;

   costuri[1][0] = 0;

   for(int i=0; i <(1<<n); i++)
      for(int j=0; j<n; j++)
         for(int k=0; k<costuri_hamilton[j].size(); k++)
            costuri[i][j] = min(costuri[i][j], costuri[i
^(1<<j)][costuri_hamilton[j][k].first]+costuri_hamilton[j][k].second);

   for(int i=0; i<costuri_hamilton[0].size(); i++)
      cost_final = min(cost_final, costuri[(1<<n)-1][costuri_hamilton[0][i].first] +
costuri_hamilton[0][i].second);

   if(cost_final != INFINIT)
      fout<<cost_final;
   else
      fout<<"Nu exista solutie";

   return 0;
}
```

Havel Hakimi
Time: O(n^2 log n)
Space: O(1)

```
while (true)
   {
      sort(grade.begin(), grade.end());
      reverse(grade.begin(), grade.end());
      if (grade[0] == 0)
         return true;

      int nodactual=grade[0];
      grade.erase(grade.begin() + 0);

      if (grade.size() < nodactual)
         return false;

      for (int i=0; i<nodactual; i++)
      {
         grade[i]--;
         if (grade[i] < 0)
            return false;
      }
   }
```