Answer to the question no-01

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long int
const ll N=3000;
vector<pair<ll,ll>>adj_list[N];
ll d[N];
const ll INF=1e15;
int n,m;
ll parent[N];
bool negative_cycle=false;
void Bellman_Ford(ll src)
{
   for(ll i=1;i<=n;i++)
   {
      d[i]=INF;
   }
   d[src]=0;
   for(ll i=1;i<=n;i++)
   {
      for(ll node=1;node<=n;node++)
      {
         for(pair<ll,ll>adj_node:adj_list[node])
         {
            ll u=node;
            ll v=adj_node.first;
            ll w=adj_node.second;
            if(d[u]+w<d[v])
            {
               d[v]=d[u]+w;
               parent[v]=u;
               if(i==n)
               {
                  negative_cycle=true;
               }
            }
         }
```

```
        }
    }
}
int main()
{
    cin>>n>>m;
    for(int i=0;i<m;i++)
    {
        int u,v,w;
        cin>>u>>v>>w;
        adj_list[u].push_back({v,w});
    }
    int src=1;
    Bellman_Ford(src);
    if(negative_cycle)
    {
        cout<<"YES\n";
    }
    else
    {
        cout<<"NO\n";
    }
}
```

Answer to the question no-02

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long int
const ll N=3000;
vector<pair<ll,ll>>adj_list[N];
ll d[N];
const ll INF=1e15;
int n,m,last_updated_node=-1;
ll parent[N];
bool negative_cycle=false;
void Bellman_Ford(int src)
{
```

```
    for(ll i=1;i<=n;i++)
    {
        d[i]=INF;
    }
    d[src]=0;
    for(ll i=1;i<=n;i++)
    {
        for(ll node=1;node<=n;node++)
        {
            for(pair<ll,ll>adj_node:adj_list[node])
            {
                ll u=node;
                ll v=adj_node.first;
                ll w=adj_node.second;
                if(d[u]+w<d[v])
                {
                    d[v]=d[u]+w;
                    parent[v]=u;
                    if(i==n)
                    {
                        negative_cycle=true;
                        last_updated_node=v;
                    }
                }
            }
        }
    }
}
void neg_cycle(int selected_node)
{
    cout<<"YES\n";
    for(int i=1;i<=n-1;i++)
    {
        selected_node=parent[selected_node];
    }
    deque<int>cycle;
    cycle.push_back(selected_node);
```

```
    int first_node=selected_node;
    while(1)
    {
        selected_node=parent[selected_node];
        cycle.push_front(selected_node);
        if(selected_node==first_node)
        {
            break;
        }
    }
    for(auto node:cycle)
    {
        cout<<node<<" ";
    }
    cout<<endl;
}
int main()
{
    cin>>n>>m;
    for(int i=0;i<m;i++)
    {
        int u,v,w;
        cin>>u>>v>>w;
        adj_list[u].push_back({v,w});
    }
    int src=1;
    Bellman_Ford(src);
    if(negative_cycle)
    {
        neg_cycle(last_updated_node);
    }
    else
    {
        cout<<"NO\n";
    }
}
```

Answer to the question no-03

```
#include<bits/stdc++.h>
using namespace std;
const int N=505;
#define ll long long int
const ll INF=1e18;
ll d[N][N];
int main()
{
   int n,m,q;
   cin>>n>>m>>q;
   for(int i=1;i<=n;i++)
   {
      for(int j=1;j<=n;j++)
      {
         d[i][j]=INF;
         d[i][i]=0;
      }
   }
   for(int i=1;i<=m;i++)
   {
      ll u,v,w;
      cin>>u>>v>>w;
      d[u][v]=min(d[u][v],w);
      d[v][u]=min(d[v][u],w);
   }
   for(int k=1;k<=n;k++)
   {
      for(int u=1;u<=n;u++)
      {
         for(int v=1;v<=n;v++)
         {
            d[u][v]=min(d[u][v],d[u][k]+d[k][v]);
         }
      }
   }
   while(q--)
```

```
    {
        int u,v;cin>>u>>v;
        d[u][v]==INF?cout<<-1<<endl:cout<<d[u][v]<<endl;
    }
}
```

Answer to the question no-04

```
// this is for directed dijkstra algorithm
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
const long long int INF=1e18;
vector<pair<int,int>>adj_list[N];
long long int d[N];
int visited[N],parent[N];
int nodes,edges;
void Dijkstra(int src)
{
    for(int i=1;i<=nodes;i++)
    {
        d[i]=INF;
    }
    d[src]=0;
    priority_queue<pair<long long int,int>>pq;
    pq.push({0,src});
    while(!pq.empty())
    {
        pair<long long int ,int>head=pq.top();
        pq.pop();
        int selected_node=head.second;
        if(visited[selected_node])
        {
            continue;
        }
        visited[selected_node]=1;
        for(auto adj_entry:adj_list[selected_node])
        {
```

```
            int adj_node=adj_entry.first;
            int edge_cst=adj_entry.second;
            if(d[selected_node]+edge_cst<d[adj_node])
            {
               d[adj_node]=d[selected_node]+edge_cst;
               parent[adj_node]=selected_node;
               pq.push({-d[adj_node],adj_node});
            }
         }
      }
}
int main()
{
   memset(visited,0,sizeof(visited));
   cin>>nodes>>edges;
   for(int i=1;i<=edges;i++)
   {
      int u,v,w;
      cin>>u>>v>>w;
      adj_list[u].push_back({v,w});
      adj_list[v].push_back({u,w});
   }
   int src=1;
   Dijkstra(src);
   if(visited[nodes]==0)
   {
      cout<<-1;
      return 0;
   }
   int current_node=nodes;
   deque<int>path;
   while(1)
   {
      path.push_front(current_node);
      if(current_node==src)
      {
         break;
```

```
        }
        current_node=parent[current_node];
    }
    for(auto node:path)
    {
        cout<<node<<" ";
    }
}
```

Answer to the question no-05

```
#include<bits/stdc++.h>
using namespace std;
int r,c;
const int N=2000;
int visited[N][N];
int maze[N][N];
int dx[4]={0,0,-1,1};int dy[4]={1,-1,0,0};
int longest=-1;
pair<int,int>find_unvisited()
{
    for(int i=0;i<r;i++)
    {
        for(int j=0;j<c;j++)
        {
            if(visited[i][j]==0 && maze[i][j]==0)
            {
                return {i,j};
            }
        }
    }
    return {-1,-1};
}

bool is_inside(pair<int,int>coord)
{
    int x=coord.first,y=coord.second;
    if(x>=0&&x<r&&y>=0&&y<c)
```

```
    {
        return true;
    }
    return false;
}

bool is_safe(pair<int,int>coord)
{
    int x=coord.first,y=coord.second;
    if(maze[x][y]==-1)
    {
        return false;
    }
    return true;
}

void BFS(pair<int,int>src)
{
    queue<pair<int,int>>q;
    visited[src.first][src.second]=1;
    q.push(src);
    int cnt=1;
    while(!q.empty())
    {
        pair<int,int>head=q.front();
        q.pop();
        int x=head.first,y=head.second;
        for(int i=0;i<4;i++)
        {
            int new_x=x+dx[i];
            int new_y=y+dy[i];
            pair<int,int>adj_node={new_x,new_y};
            if(is_safe(adj_node)&&is_inside(adj_node) && visited[new_x][new_y]==0)
            {
                visited[new_x][new_y]=1;
                q.push(adj_node);
                cnt++;
```

```
            }
        }
    }
    if(longest<cnt)
    {
        longest=cnt;
    }
}
int main()
{
    int cnt=0;
    cin>>r>>c;
    for(int i=0;i<r;i++)
    {
        for(int j=0;j<c;j++)
        {
            char x;cin>>x;
            if(x=='#')
            {
                maze[i][j]=-1;
                visited[i][j]=-1;
            }
            else
            {
                maze[i][j]=0;
            }
        }
    }
    for(int i=0;i<r;i++)
    {
        for(int j=0;j<c;j++)
        {
            if(visited[i][j]==0 && maze[i][j]==0)
            {
                BFS({i,j});
                cnt++;
            }
```

```
        }
    }
    cout<<"Rooms - "<<cnt<<endl;
    cout<<"Length of the longest room - "<<longest<<endl;
}
```

Answer to the question no-06

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string s;
    cin>>s;
    unordered_map<char,int>str;
    for(int i=0;i<s.size();i++)
    {
        str[s[i]]++;
    }
    int odd=0;
    for(auto p:str)
    {
        if(p.second%2==1)
        {
            odd++;
            if(odd>1)
            {
                cout<<"NO\n";
                return 0;
            }
        }
    }
    cout<<"YES\n";
}
```

Answer to the question no-07

```
#include<bits/stdc++.h>
using namespace std;
int digit_count(string s,int a)
{
   if(s.size()==1)
   {
      return s[0]-'0';
   }
   a+=s[s.size()-1]-'0'+digit_count(s.substr(0,s.size()-1),a);
   return a;
}
int main()
{
   string s;
   cin>>s;
   cout<<digit_count(s,0)<<endl;
}
```