

## Answer to the question no-01

Total nodes are 7.

The connected nodes are

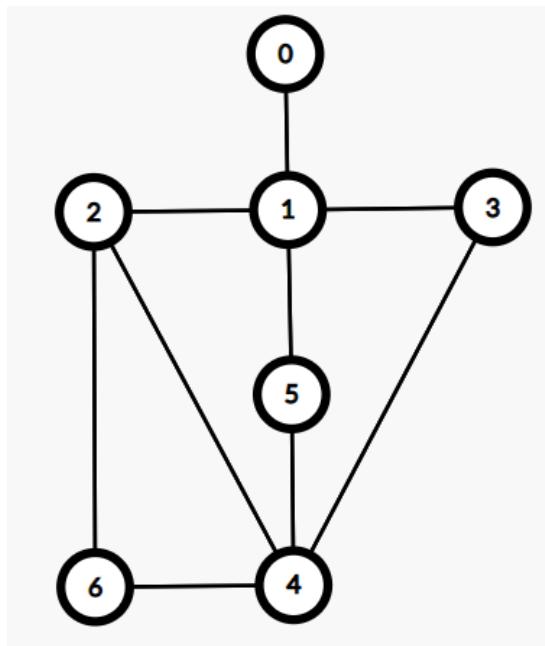
- (0,1),
- (1,0),(1,2),(1,3),(1,5),
- (2,1),(2,4),(2,6),
- (3,1),(3,4),
- (4,2),(4,3),(4,5),(4,6),
- (5,1),(5,4),
- (6,2),(6,4).

From the relation we can say that it is an undirected graph.

The adjacency list will be-

- 0->1.
- 1->0,2,3,4.
- 2->1,4,6.
- 3->1,4.
- 4->2,3,5,6.
- 5->1,4.
- 6->2,4.

The graph will look like:



## Answer to the question no-02

BFS Algorithm	DFS Algorithm
1. Full form is Breadth First Search	1. Full form is Depth First Search
2. It is level wise traversal.	2. It is a pre-order traversal.
3. We have to use queue data structure to implement it	3. We have to use a recursive function to implement it.
4. It visits all the adjacent nodes.	4. It goes to the last end of a node.
5. It is useful for finding the smallest path.	5. It is useful for finding paths between 2 nodes.
6. It is only one type.	6. It has 3 types: pre-order, post-order, in-order.

## Answer to the question no-03

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long int
const int N=1e5+5;
ll dp[N];
int x;
ll perfect_square(int n)
{
    if(n==0)
    {
        return 0;
    }
    if(dp[n]!=-1)
    {
        return dp[n];
    }
    ll ans=1e8;
    for(int i=1;i<=x;i++)
    {
        if(n>=i*i)
```

```

    {
        ans=min(ans,perfect_square(n-i*i)+1);
    }
}
dp[n]=ans;
return ans;
}
int main()
{
    int num;
    cin>>num;
    x=sqrt(num);
    memset(dp,-1,sizeof(dp));
    cout<<perfect_square(num);
}

```

### Answer to the question no-04

```

#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
#define ll long long
ll dp[N];
vector<int>money1,money2;
int x,m;
int rob1(int i)
{
    if(i==0)
    {
        return money1[0];
    }
    if(i<0)
    {
        return 0;
    }
    if(dp[i]!=-1)
    {
        return dp[i];
    }
    else

```

```

    }
    int ans1=0,ans2=0;
    if(i-2>=m)
    {
        ans1=money1[i]+rob1(i-2);
    }
    else
    {
        ans1=money1[i];
    }
    if(i-3>=m)
    {
        ans2=money1[i]+rob1(i-3);
    }
    else
    {
        ans2=money1[i];
    }

    dp[i]=max(ans1,ans2);
    return dp[i];
}
int rob2(int i)
{
    if(i==0)
    {
        return money2[0];
    }
    if(i<0)
    {
        return 0;
    }
    if(dp[i]!=-1)
    {
        return dp[i];
    }
    int ans1=0,ans2=0;

```

```

if(i-2>=m)
{
    ans1=money2[i]+rob2(i-2);
}
else
{
    ans1=money2[i];
}
if(i-3>=m)
{
    ans2=money2[i]+rob2(i-3);
}
else
{
    ans2=money2[i];
}

dp[i]=max(ans1,ans2);
return dp[i];
}
int main()
{
    int n;cin>>n;
    int A[n];
    memset(dp,-1,sizeof(dp));
    for(int i=0;i<n;i++)
    {
        cin>>A[i];
    }
    if(n==1)
    {
        cout<<A[0]<<endl;
        return 0;
    }
    if(n==2)
    {
        cout<<max(A[0],A[1]);
    }
}

```

```

        return 0;
    }
    for(int i=0;i<n;i++)
    {
        if(i!=n-1)
        {
            money1.push_back(A[i]);
        }
        if(i!=0)
        {
            money2.push_back(A[i]);
        }
    }
    int a=rob1(n-2);
    memset(dp,-1,sizeof(dp));
    int b=rob2(n-2);
    cout<<max(a,b)<<endl;
}

```

### Answer to the question no-05

#### **////Memoization:**

```

#include<bits/stdc++.h>
using namespace std;
const int N=1005;
#define ll long long
ll dp[N][N];
int maze[N][N];
ll mod=1e9+7;
bool Maze(int x,int y)
{
    if(maze[x][y]==-1)
    {
        return false;
    }
    return true;
}
int unique_paths(int n,int m)

```

```

{
    if(n==0&&m==0)
    {
        return 1;
    }
    if(dp[n][m]!=-1)
    {
        return dp[n][m];
    }
    int ans=0;
    if(n-1>=0 && Maze(n-1,m) && Maze(n,m))
    {
        ans+=unique_paths(n-1,m);
        ans%=mod;
    }
    if(m-1>=0 && Maze(n,m-1) && Maze(n,m))
    {
        ans+=unique_paths(n,m-1);
        ans%=mod;
    }
    dp[n][m]=ans;
    return ans;
}
int main()
{
    int n;
    cin>>n;
    memset(dp,-1,sizeof(dp));
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            char c;
            cin>>c;
            if(c=='*')
            {
                maze[i][j]=-1;
            }
        }
    }
}
```

```

        }
    }
}
if(n==1 && maze[n-1][n-1])
{
    cout<<0;
    return 0;
}
cout<<unique_paths(n-1,n-1)<<endl;
}

```

**///Tabulation:**

```

#include<bits/stdc++.h>
using namespace std;
const int N=1005;
#define ll long long
ll dp[N][N];
int maze[N][N];
ll mod=1e9+7;
bool Maze(int x,int y)
{
    if(maze[x][y]==-1)
    {
        return false;
    }
    return true;
}
int main()
{
    int n;
    cin>>n;
    memset(dp,-1,sizeof(dp));
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            char c;

```

```

cin>>c;
if(c=='*')
{
    maze[i][j]=-1;
}
}

if(n==1 && maze[n-1][n-1])
{
    cout<<0;
    return 0;
}

dp[0][0]=1;
for(int i=0;i<n;i++)
{
    for(int j=0;j<n;j++)
    {
        if(i==0 && j==0)
        {
            continue;
        }
        if(maze[i][j]==-1)
        {
            continue;
        }
        int ans=0;
        if(i-1>=0 && Maze(i-1,j) && Maze(i,j))
        {
            ans+=dp[i-1][j];
            ans%=mod;
        }
        if(j-1>=0 && Maze(i,j-1) && Maze(i,j))
        {
            ans+=dp[i][j-1];
            ans%=mod;
        }
        dp[i][j]=ans;
    }
}

```

```

        }
    }
    cout<<dp[n-1][n-1]<<endl;
}

```

## Answer to the question no-06

### BFS Order::

```

#include<bits/stdc++.h>
using namespace std;
#define ll long long int
const int N=1005;
int maze[N][N];
int dx[]={1,1,0,-1,-1,-1,0,1};
int dy[]={0,-1,-1,-1,0,1,1,1};
int visited[N][N];
int n;
bool is_inside(int x,int y)
{
    if(x>0 && y>0 && x<=n && y<=n)
    {
        return true;
    }
    return false;
}
void BFS(int c,int d)
{
    visited[c][d]=1;
    queue<pair<int,int>>pq;
    pq.push({c,d});
    while(!pq.empty())
    {
        pair<int,int>head;
        head=pq.front();
        pq.pop();
        for(int i=0;i<8;i++)
        {

```

```

int x=head.first;
int y=head.second;
int new_x=x+dx[i];
int new_y=y+dy[i];

if(maze[new_x][new_y]!=-1&&is_inside(new_x,new_y)&&visited[new_x][new_y]==0)
{
    visited[new_x][new_y]=1;
    pq.push({new_x,new_y});
}
}

int main()
{
    cin>>n;
    int a,b;
    cin>>a>>b;
    int c,d;
    cin>>c>>d;
    int e,f;
    cin>>e>>f;
    for(int i=1;i<=n;i++)
    {
        maze[a][i]=-1;
    }
    for(int i=1;i<=n;i++)
    {
        maze[i][b]=-1;
    }
    int x=a,y=b;
    while(x<=n && y<=n)
    {
        x++;y++;
        maze[x][y]=-1;
    }
    x=a;y=b;
}

```

```

while(x>1&&y>1)
{
    x--;
    maze[x][y]=-1;
}
x=a,y=b;
while(x>1&&y<n)
{
    x--;
    y++;
    maze[x][y]=-1;
}
x=a,y=b;
while(x<n && y>1)
{
    x++;
    y--;
    maze[x][y]=-1;
}
BFS(c,d);
visited[e][f]==1?cout<<"YES\n":cout<<"NO\n";
}

}

```

### **DFS order::**

```

#include<bits/stdc++.h>
using namespace std;
#define ll long long int
const int N=1005;
int maze[N][N];
int dx[]={1,1,0,-1,-1,-1,0,1};
int dy[]={0,-1,-1,-1,0,1,1,1};
int visited[N][N];
int n;
bool is_inside(int x,int y)
{
    if(x>0 && y>0 && x<=n && y<=n)
    {
        return true;
    }
}

```

```

        }
        return false;
    }
void DFS(int c,int d)
{
    visited[c][d]=1;
    for(int i=0;i<8;i++)
    {
        int new_x=c+dx[i];
        int new_y=d+dy[i];

        if(maze[new_x][new_y]!=-1&&is_inside(new_x,new_y)&&visited[new_x][new_y]==0)
        {
            DFS(new_x,new_y);
        }
    }
}
int main()
{
    cin>>n;
    int a,b;
    cin>>a>>b;
    int c,d;
    cin>>c>>d;
    int e,f;
    cin>>e>>f;
    for(int i=1;i<=n;i++)
    {
        maze[a][i]=-1;
    }
    for(int i=1;i<=n;i++)
    {
        maze[i][b]=-1;
    }
    int x=a,y=b;
    while(x<n && y<n)
    {

```

```

x++;y++;
maze[x][y]=-1;
}
x=a;y=b;
while(x>1&&y>1)
{
    x--;y--;
    maze[x][y]=-1;
}
x=a;y=b;
while(x>1&&y<n)
{
    x--;y++;
    maze[x][y]=-1;
}
x=a;y=b;
while(x<n && y>1)
{
    x++;y--;
    maze[x][y]=-1;
}
DFS(c,d);
visited[e][f]==1?cout<<"YES\n":cout<<"NO\n";
}

```

## Answer to the question no-07

Here source node is 2.

1. From node 2 we can go node 3 and 1 and put the weights in the table as they are smaller than infinity.
2.  $3 < 18$  so the second node is 1, from node 1 we can go to node 2 3 6 8 4. For node 3,  $3+22 > 18$ , it will not change, and  $0+3 > 0$  so this also does not change for node 2. others are smaller than infinity, we put it in the table.
3. Next node is node 4. From node 4 we can go to nodes 1 3 7 8. For node 1-  $1+4 > 4$ , so no change, for node 3-  $4+2 < 18$  value will change, for node 7-  $4+4 < -$  value will change, for node 8-  $4+10 > 6$ , no change.

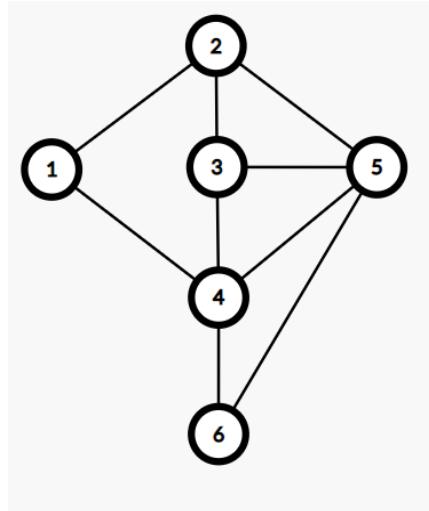
4. Next node is 3. From node 3 we can go 2 1 7 4. For node 2-  $0+6>0$  no change, for node 1-  $6+22>3$  no change, for node 7-  $6+1<8$ , value will change, for node 4-  $6+2>4$  no change.
5. Next node is 6. From node 6 we can go 1 7 8. For node 1-  $6+3>3$  no change, for node 7-  $6+2>7$  no change, for node 8-  $6+2>6$  no change.
6. Next node is 8. From it we can traverse nodes 1 4 6. For node 1- $>6+3>3$  no change, for node 4-  $6+10>4$  no change, for node 6-  $6+2>6$  no change.
7. Last node is 7. And there is no node which is able to change.

The table looks like this.

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
<b>2</b>	3	<b>0</b>	18	-	-	-	-	-
<b>1</b>	<b>3</b>	<b>0</b>	18	4	-	6	-	6
<b>4</b>	<b>3</b>	<b>0</b>	6	<b>4</b>	-	6	8	6
<b>3</b>	<b>3</b>	<b>0</b>	<b>6</b>	<b>4</b>	-	6	7	6
<b>6</b>	<b>3</b>	<b>0</b>	<b>6</b>	<b>4</b>	-	<b>6</b>	7	6
<b>8</b>	<b>3</b>	<b>0</b>	<b>6</b>	<b>4</b>	-	<b>6</b>	7	<b>6</b>
<b>7</b>	<b>3</b>	<b>0</b>	<b>6</b>	<b>4</b>	-	<b>6</b>	<b>7</b>	<b>6</b>
-	-	-	-	-	-	-	-	-

### Answer to the question no-08

After rearranging the graph it will look like:



We know that BFS traverse level wise:

Here the levels are 3. 2 is a source node so its level is 0.

We have to use queue data structure to do it.

The steps are:

1. we get the source node which is 2 and we push it to the queue. Now print it and put it into an integer H and pop the node.
2. When exploring 2 we get the 1,3,5 as child nodes. So we push them into the queue [1 3 5].
3. Now our node is 1. Print it and pop it. When exploring node 1 we get the node 4 as a child node. So push it into the queue [3 5 4].
4. This time the top node is 3. Print it and pop it. When exploring it we get nodes 4 and 5. But we do not push it because we already visit it. So the queue is [5 4].
5. Node 5 is at the top. Print it and pop it. When exploring node 5 we get node 6. As it is unvisited, push it into the queue [4 6].
6. 4 is a top node. Print it and pop it. When exploring node 4 we get node 5 and 6 but they are visited. So our queue will be [6].
7. Now 6 is at the top. Print it and pop it. As all nodes are visited so it will end the traversal.

The output will look like —>2 1 3 5 4 6.