# Index - 7

**Graphs**



**1) Breadth First Search**

```
public ArrayList<Integer> bfsOfGraph(int v, ArrayList<ArrayList<Integer>> adj)
{
      ArrayList<Integer> answer = new ArrayList<>();
      Queue<Integer> q = new LinkedList<>();

      q.add(0);
      boolean vis[] = new boolean[v];
      vis[0] = true;
      while(!q.isEmpty()){
         int cur = q.poll();
         answer.add(cur);

         for(int i=0;i<adj.get(cur).size();i++){
            if(!vis[adj.get(cur).get(i)]){
               vis[adj.get(cur).get(i)] = true;
               q.add(adj.get(cur).get(i));
            }
         }
      }
      return answer;
}
```

**T.C = O(N + E)**
**S.C = O(N + E) + O(N)**

**2) Depth First Search**

```
public ArrayList<Integer> dfsOfGraph(int v, ArrayList<ArrayList<Integer>> adj)
   {
      boolean vis[] = new boolean[v];
      ArrayList<Integer> answer = new ArrayList<>();
```

```
        for(int i=0;i<v;i++){
            if(!vis[i]){
                dfs(i, adj, vis, answer);
            }
        }
        return answer;
    }
    public void dfs(int vertex, ArrayList<ArrayList<Integer>> adj, boolean
vis[],ArrayList<Integer> answer){
        vis[vertex] = true;
        answer.add(vertex);

        for(int i=0;i<adj.get(vertex).size();i++){
            if(!vis[adj.get(vertex).get(i)]){
                dfs(adj.get(vertex).get(i),adj, vis, answer);
            }
        }
    }
```

**T.C = O(N + E)**
**S.C = O(N + E) + O(N) + O(N)**

**3) Detect cycle in an undirected graph Using BFS**

```
public boolean isCycle(int v, ArrayList<ArrayList<Integer>> adj)
    {
        boolean vis[] = new boolean[v];

        for(int i=0;i<v;i++){
            if(!vis[i]){
                if((bfsCycleDetection(i, adj, vis)))
                    return true;
            }
        }
        return false;
    }
    public boolean bfsCycleDetection(int cur,ArrayList<ArrayList<Integer>> adj, boolean
vis[]){

        Queue<Pair> q = new LinkedList<>();
        q.add(new Pair(cur,-1));
        vis[cur] = true;

        while(!q.isEmpty()){
            Pair element = q.poll();
            int pt = element.parent;
            int v = element.cur;

            for(int i=0;i<adj.get(v).size();i++){
                int vertex = adj.get(v).get(i);
                if(!vis[vertex]){
```

```
                q.add(new Pair(vertex, v));
                vis[vertex] = true;
            }
            else if(vertex!=pt)
                return true;
        }
    }
    return false;
}
```

**T.C = O(N + E)**
**S.C = O(N + E) + O(N)**

**4) Detect cycle in an undirected graph using DFS**

```
public boolean isCycle(int v, ArrayList<ArrayList<Integer>> adj)
{
    boolean vis[] = new boolean[v];

    for(int i=0;i<v;i++){
        if(!vis[i]){
            if(dfsCycleDetection(i,-1 ,adj, vis))
                return true;
        }
    }
    return false;
}
public boolean dfsCycleDetection(int cur, int parent,ArrayList<ArrayList<Integer>> adj,
boolean vis[]){
    vis[cur] = true;

    for(int i=0;i<adj.get(cur).size();i++){
        int vertex = adj.get(cur).get(i);
        if(!vis[vertex]){
            if(dfsCycleDetection(vertex, cur ,adj, vis))
                return true;
        }
        else if(vertex!=parent)
            return true;
    }
    return false;
}
```

**T.C = O(N + E)**
**S.C = O(N + E) + O(N) + O(E)**

**5) Bipartite Check Using BFS**

```
public boolean isBipartite(int v, ArrayList<ArrayList<Integer>>adj)
```

```
    {
        int color[] = new int[v];
        Arrays.fill(color,-1);

        for(int i=0;i<v;i++){
            if(color[i]==-1){
                if(!bfsBipartiteCheck(i, adj, color)){
                    return false;
                }
            }
        }

        return true;
    }
    public boolean bfsBipartiteCheck(int cur,ArrayList<ArrayList<Integer>> adj, int color[]){
        Queue<Integer> q = new LinkedList<>();
        q.add(cur);
        color[cur] = 1;

        while(!q.isEmpty()){
            int vertex = q.poll();

            for(int i=0;i<adj.get(vertex).size();i++){
                int v = adj.get(vertex).get(i);
                if(color[v]==-1){
                    q.add(v);
                    color[v] = 1 - color[vertex];
                }
                else if(color[v]==color[vertex])
                    return false;
            }
        }
        return true;
    }
```

**T.C = O(N + E)**
**S.C = O(N + E) + O(N)**

**6) Bipartite Check Using DFS**

```
 public boolean isBipartite(int v, ArrayList<ArrayList<Integer>>adj)
    {
        int color[] = new int[v];
        Arrays.fill(color,-1);

        for(int i=0;i<v;i++){
            if(color[i]==-1){
                if(!dfsBipartiteCheck(i, adj, color))
                    return false;
            }
        }
```

```
        return true;
    }
    public boolean dfsBipartiteCheck(int cur,ArrayList<ArrayList<Integer>> adj, int vis[]){
        if(vis[cur]==-1)
            vis[cur] = 1;

        for(int i=0;i<adj.get(cur).size();i++){
            int v = adj.get(cur).get(i);
            if(vis[v]==-1){
                vis[v] = 1 - vis[cur];
                if(!dfsBipartiteCheck(v ,adj, vis))
                    return false;
            }
            else if(vis[cur]==vis[v])
                return false;
        }
        return true;
    }
```

**T.C = O(N + E)**
**S.C = O(N + E) + O(N) + O(N)**

**7) Detect cycle in a directed graph using DFS**

```
public boolean isCyclic(int v, ArrayList<ArrayList<Integer>> adj)
{
        boolean vis[] = new boolean[v];
        boolean inList[] = new boolean[v];

        for(int i=0; i<v; i++)
        {
                if(!vis[i])
                {
                        if(dfsCycleDetection(i, adj, vis, inList))
                                return true;
                }
        }

        return false;
}

boolean dfsCycleDetection(int cur, ArrayList<ArrayList<Integer>> adj, boolean vis[],
                                                                boolean inList[])
{
        vis[cur] = true;
        inList[cur] = true;

        for(int i=0; i<adj.get(cur).size(); i++)
        {
                int v = adj.get(cur).get(i);
```

```
                if(!vis[v])
                {
                        if(dfsCycleDetection(v, adj, vis, inList)
                                return true;

                }

                else if(inList[v])
                        return true;
        }

        inList[cur] = false;
        return false;
}
```

**T.C = O(N + E)**
**S.C = O(N + E) + O(N) + O(N)**

**8) Detect cycle in a directed graph using BFS**

```
public boolean isCyclic(int v, ArrayList<ArrayList<Integer>> adj) {

        int indegree[] = new int[v];

        for(int i=0;i<v;i++)
            for(int vertex:adj.get(i))
                indegree[vertex]++;

        Queue<Integer> q = new LinkedList<>();
        for(int i=0;i<v;i++)
            if(indegree[i]==0)
                q.add(i);
        int count = 0;
        while(!q.isEmpty()){
            int cur = q.poll();
            count++;

            for(int vertex:adj.get(cur)){
                indegree[vertex]--;
                if(indegree[vertex]==0)
                    q.add(vertex);
            }
        }

      return count!=n;
}
```

**9) Topological Sorting using DFS**

```
static int[] topoSort(int v, ArrayList<ArrayList<Integer>> adj) {
        Stack<Integer> s = new Stack<>();
        boolean vis[] = new boolean[v];

        for(int i=0;i<v;i++){
            if(!vis[i])
                dfsTopoSort(i, adj, vis, s);
        }

        int answer[] = new int[v];
        int index=0;
        while(!s.isEmpty())
            answer[index++] = s.pop();

        return answer;
    }
    static void dfsTopoSort(int cur,ArrayList<ArrayList<Integer>> adj, boolean
vis[],Stack<Integer> s){
        vis[cur] = true;

        for(int i=0;i<adj.get(cur).size();i++){
            int vertex = adj.get(cur).get(i);
            if(!vis[vertex])
                dfsTopoSort(vertex, adj, vis, s);
        }
        s.push(cur);
    }
```

**10) Topological Sorting using BFS**

```
 static int[] topoSort(int v, ArrayList<ArrayList<Integer>> adj) {

        int indegree[] = new int[v];
        for(int i=0;i<v;i++){
            for(int j=0;j<adj.get(i).size();j++){
                indegree[adj.get(i).get(j)]++;
            }
        }
        Queue<Integer> q = new LinkedList<>();

        for(int i=0;i<v;i++)
            if(indegree[i]==0)
                q.add(i);

        int answer[] = new int[v];
        int index = 0;

        while(!q.isEmpty()){
            int cur = q.poll();
            answer[index++] = cur;
```

```
            for(int i=0;i<adj.get(cur).size();i++){
                int vertex = adj.get(cur).get(i);
                indegree[vertex]--;
                if(indegree[vertex]==0)
                    q.add(vertex);
            }
        }

        return answer;
    }
```

**11) Shortest distance in unweighted graph with unit weights using BFS (From source to all other vertices)**

```
public int[] shortestPath(int src, ArrayList<ArrayList<Integer>> adj, int v)
{

        int dist[] = new int[v];
        Arrays.fill(dist, Integer.MAX_VALUE);

        Queue<Integer> q = new LinkedList<>();
        dist[src] = 0;
        q.add(src);

        while(!q.isEmpty())
        {
                int node = q.poll();

                for(int vertex : adj.get(node))
                {
                        if(dist[node]+1<dist[vertex])
                        {
                                dist[vertex] = dist[node]+1;
                                q.add(vertex);
                        }
                }
        }

        return dist;
}
```

**12) Shortest path in weighted directed acyclic graph**

```
class Pair{
        int node;
        int weight;
        Pair(int n, int w)
        {
                node = n;
```

```
            weight = w;
        }
}

int[] shortestPath(int src, int v,  ArrayList<ArrayList<Pair>> adj)
{
        Stack<Integer> s = new Stack<>();
        boolean vis[] = new boolean[v];

        for(int i=0; i<v; i++)
            if(!vis[i])
                dfsToposort(i, adj, vis, s);

        int dist[] = new int[v];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[src] = 0;

        while(!s.isEmpty())
        {
            int cur = s.pop();

            if(dist[cur]!=Integer.MAX_VALUE)
            {
                for(Pair it : adj.get(cur))
                {
                    if(dist[cur]+it.weight<dist[it.v])
                    {
                        dist[it.v] = dist[cur] + it.weight;
                    }
                }
            }
        }

        return dist;
}

void dfsTopoSort(int cur, ArrayList<ArrayList<Integer>> adj, boolean vis[], Stack<Integer>
s)
{
        vis[cur] = true;

        for(int i : adj.get(cur))
        {
            if(!vis[i])
                dfsTopoSort(i, adj, vis, s);
        }

        s.push(cur);
}
```