WEEK - 6

ɪn a graph, design an algorithm & implement it using a program
nd if a path exists b/w two given vertices or not.

orithm :→

```
1) START
2) INPUT V
3) if i>=V goto step 7
4) input temp
5) adj[i].push back (temp)
6) if i<v goto step 3
7) input s,d
8) res = checkpath (adj,V,s-1,d-1)
9) if (res==1) print "Path exists"
10) else print "Path does not exist"
11) STOP
```

```
checkpath (vector <int> adj[] , int v, int s, int d) {
    visited [V] = false ;
    for (int i=0; i<v; i++ )
       dfs (adj, s, visited ,V)
       }
       return visited [d];
}

dfs(adj[], s, v, visited)
{
  visited [s] = true;
  for (int i=0 ; i<v; i++){
    if (adj[s][i] ! =0 && !visited [i]){
        dfs (adj,i , V, visited) ;
    }

  }
}
```

2. Given a graph, design an algorithm and implement it using c to find if a graph is Bipartite or not.

```
Algorithm :→ 1) START
             2) Input v
             3) if i>=V goto step 7
             4) input temp
             5) input G[i].push_back(temp)
             6) if i<v goto step 3
             7) res = (Bipartite (G,V))
             8) if(res == 1) print " Bipartite "
             9) else print " not Bipartite"
             10) STOP
```

```
Bipartite (vector<int> G[], int V) {
    color_arr[V];
    for(int i=0; i<V; i++)
        color_arr[i] = -1;
    for (int i=0; i<V; i++) {
        if (color_arr[i] == -1)
        if (isBipartite (G, i, color_arr, V) == false)
            return false;
    }
        return true;
}
```

```
~~is_B~~ isBipartite (G[], src, color_arr[], V) {
    color_arr[src] = 1;
    queue<int> q;
    q.push(src);
    while (!q.empty()) {
     int u = q.front();
      q.pop;
```

n a directed graph, design and implement an algorithm using program to find whether cycle exists or not.

Algorithm :→

```
1) START
2) Input V
3) Input u,v in adj[V]
4) res = DFS (adj, V)
5) if (res = = 1) , print "Cycle Exists"
6) else print " No cycle Exist "
7) STOP
```

```
DFS( vector <int> adj[], int V[], int dfsV[], int node){
        V[node] = 1;
        dfsV [node] =1 ;
       for (auto it : adj[node]) {
        if(!V[it])  {
        if(DFS(it, adj, V, dfsV))
             return True;
         }
          dfsV [node] = 0
          return false;
   }
isCycle (vector <int> adj [], int N ){
    int V [N+1], dfsV [N+1];
    memset ( u, 0, sizeof (dfsV));
   for (int i =1; i<N ; i++ ){
      if (!V[i]){
     if(DFS( i, adj , V, dfsV))
          return True ;
      }
    }
      return false;
 }
```

```
    if (G[u][u] == 1)
        return false;
    for (int v = 0; v< V; ++v) {
        if (G[u][v] != 0 && color-arr [v] == 1) {
            color-arr[v] = 1 - color-arr[u];
            q.push (v)
        }
    else if (G[u][v] != 0 && color-arr [v] == color-arr[
            return false;
        }
    }
        return true;
}
```