Answer Script

Question No. 01

1) Write a c++ program to convert an Adjacency Matrix to an Adjacency List. **20**

Sample Input	Sample Output
6 0 1 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0	0: 1 2 1: 0 3 4 2: 0 5 3: 1 4 5 4: 1 3 5: 2 3

Answer No. 01

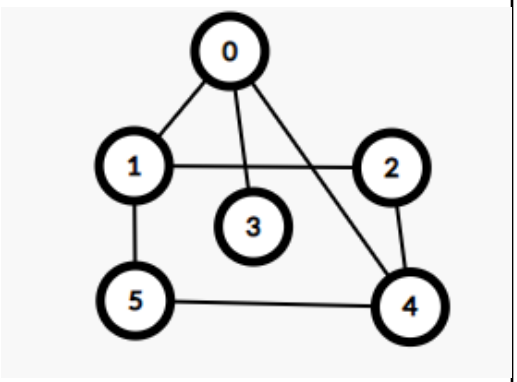
Code:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int nodes;
   cin>>nodes;

   int matrix[nodes][nodes];

   for(int i=0; i<nodes; i++)
   {
      for(int j=0; j<nodes; j++)
      {
         cin>>matrix[i][j];
      }
   }

   for(int i=0; i<nodes; i++)
   {
      cout<<i<<": ";
```

```
      for(int j=0; j<nodes; j++)
      {
         if(matrix[i][j] == 1)
         {
            cout<<j<<" ";
         }
     }
      cout<<"\n";
   }

   return 0;
}
```

Question No. 02

2) Write a c++ program to solve the single source shortest path(SSSP) problem using **BFS**.
Consider 0 as the **source node**. **20**

Sample Input	**Sample Output**
	node 0 -> level: 0 node 1 -> level: 1 node 2 -> level: 2 node 3 -> level: 1 node 4 -> level: 1 node 5 -> level: 2

Answer No. 02

Code:

```
#include<bits/stdc++.h>
using namespace std;

const int N = 2e5;

vector<int> adj_list[N];
int visited[N];
int level[N];

void BFS(int node)
{
   queue<int> q;

   visited[node] = 1;
   level[node] = 0;
   q.push(node);

   while(!q.empty())
   {
      int head = q.front();
```

```
        q.pop();

        for(auto adj_node: adj_list[head])
        {
            if(visited[adj_node] == 0)
            {
                visited[adj_node] = 1;
                level[adj_node] = level[head] + 1;
                q.push(adj_node);
            }
        }
    }
}

int main()
{
    int nodes = 6;
    adj_list[0] = {1, 3, 4};
    adj_list[1] = {0, 2, 5};
    adj_list[2] = {1, 4};
    adj_list[3] = {0};
    adj_list[4] = {0, 2, 5};
    adj_list[5] = {1, 4};

    int src = 0;
    BFS(src);

    for(int i=0; i<nodes; i++)
    {
        cout<<"node "<<i<<" -> level: "<<level[i]<<"\n";
    }

    return 0;
}
```

Question No. 03

3) Write a c++ program to solve **cycle detection** in a **directed graph** using **DFS.** **20**

Sample Input	Sample Output
5 5 0 1 1 2 2 3 3 4 4 1	YES
5 4 0 1 1 2 2 3 3 4	NO

Answer No. 03

Code:

```
#include<bits/stdc++.h>
using namespace std;

const int N = 2e5;
int visited[N];
vector<int> adj_list[N];

bool detect_cycle(int node)
{
   visited[node] = 1;

   for(auto adj_node: adj_list[node])
   {
      if(visited[adj_node] == 0)
      {
         bool got_cycle = detect_cycle(adj_node);
         if(got_cycle)
            return true;
      }
```

```
        else if(visited[adj_node] == 1)
        {
            return true;
        }
    }

    visited[node] = 2;
    return false;
}


int main()
{
    int nodes, edges;
    cin>>nodes>>edges;

    for(int i=0; i<edges; i++)
    {
        int u, v;
        cin>>u>>v;

        adj_list[u].push_back(v);
    }

    bool cycle_exixts = false;

    for(int i=1; i<=nodes; i++)
    {
        if(visited[i] == 0)
        {
            bool got_cycle = detect_cycle(i);
            if(got_cycle)
            {
                cycle_exixts = true;
                break;
            }
        }
    }

    if(cycle_exixts)
        cout<<"YES\n";
    else
        cout<<"NO\n";
    return 0;
}
```

Question No. 04

4) Write a c++ program to check if a graph is **Bipartite** or not. **20**

Sample Input	Sample Output
3 2 0 1 1 2	YES
3 3 0 1 1 2 2 0	NO

Answer No. 04

Code:

```
#include<bits/stdc++.h>
using namespace std;

const int N = 2e5;
int visited[N];
int color[N];

vector<int> adj_list[N];

bool DFS(int node)
{
   visited[node] = 1;

   for(auto adj_node: adj_list[node])
   {
      if(visited[adj_node] == 0)
      {
         if(color[node] == 1)
            color[adj_node] = 2;
         else
            color[adj_node] = 1;
```

```
            bool is_bipartite = DFS(adj_node);
            if(!is_bipartite)
                return false;
        }
        else
        {
            if(color[node] == color[adj_node])
                return false;
        }
    }
    return true;
}

int main()
{
    int nodes, edges;
    cin>>nodes>>edges;

    for(int i=0; i<edges; i++)
    {
        int u, v;
        cin>>u>>v;
        adj_list[u].push_back(v);
        adj_list[v].push_back(u);
    }

    bool is_bipartite = true;

    for(int i=1; i<=nodes; i++)
    {
        if(visited[i] == 0)
        {
            color[i] = 1;
            bool ok = DFS(i);
            if(!ok)
            {
                is_bipartite = false;
                break;
            }
        }
    }

    if(is_bipartite)
        cout<<"YES\n";
```

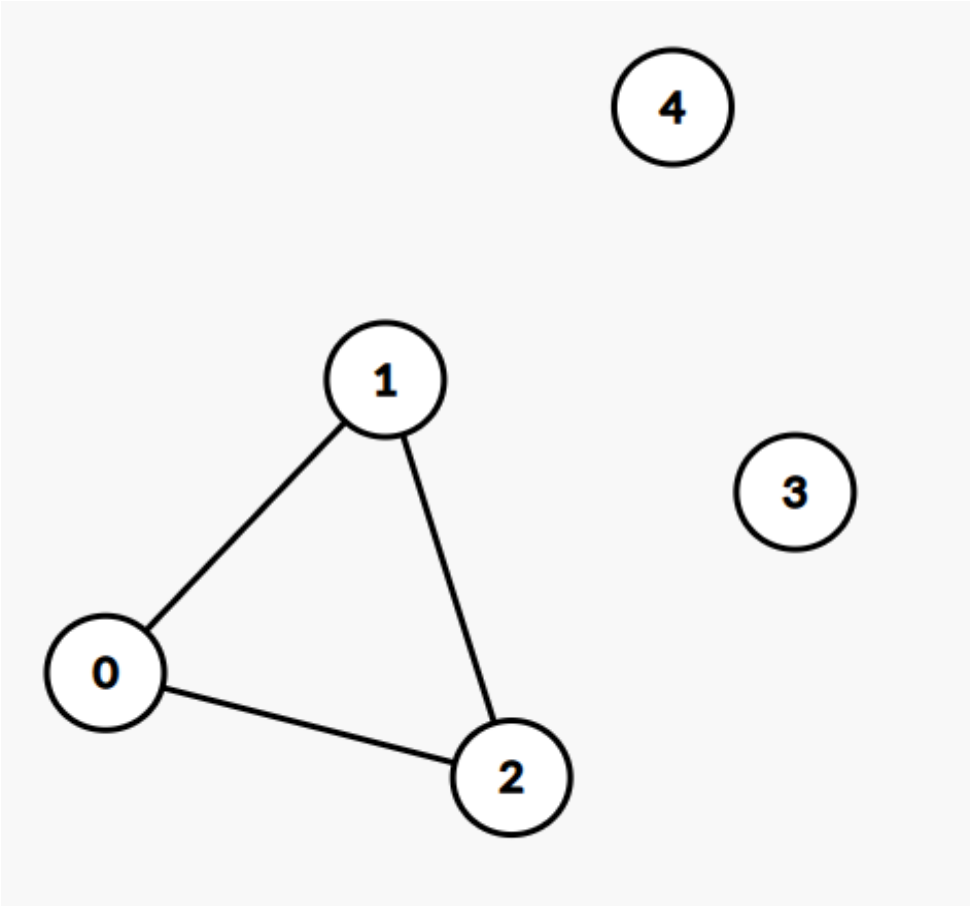
```
    else
        cout<<"NO\n";

    return 0;

}
```

Question No. 05

5) Write a c++ program to take an **undirected** graph as input and count the number of **connected** components in it. **20**

Sample Input	Sample Output
5 3 0 1 1 2 2 0	3

Explanations:



Sample Input graph is in above, we see that there are 3 components in this graph.

Answer No. 05

Code:

```
#include<bits/stdc++.h>
using namespace std;

const int N = 2e5;
int visited[N];
vector<int> adj_list[N];

void DFS(int node)
{
   visited[node] = 1;

   for(int adj_node: adj_list[node])
```

```
    {
        if(visited[adj_node] == 0)
        {
            DFS(adj_node);
        }
    }
}

int main()
{
    int nodes, edges;
    cin>>nodes>>edges;

    for(int i=0; i<edges; i++)
    {
        int u, v;
        cin>>u>>v;

        adj_list[u].push_back(v);
        adj_list[v].push_back(u);
    }

    int countConnected = 0;

    for(int i=0; i<nodes; i++)
    {
        if(visited[i] == 0)
        {
            DFS(i);
            countConnected++;
        }
    }

    cout<<countConnected<<"\n";

    return 0;
}
```