Codes

BFS(with Modifications):

```
#include<bits/stdc++.h>

using namespace std;

int ad[100][100];

int mark[100];

int dist[100];

int parent[100];

int visited[100];

void initadj()

{

   int i,j;

   for(i=0;i<100;i++)

   {

      for(j=0;j<100;j++)

      {

         ad[i][j] = 0;

      }

   }

}

void initmark()

{

   int i;

   for(i=0;i<100;i++)
```

```
    {
        mark[i] = 0;
        visited[i] = 0;
    }
}
void initdist()
{
    int i;
    for(i=0;i<100;i++)
    {
        dist[i] = -1;
    }
}
void initparent()
{
    int i;
    for(i=0;i<100;i++)
    {
        parent[i] = -1;
    }
}
void bfs(int start,int n)
{
    queue<int> q;
    q.push(start);
```

```
    int king;
    while(q.size()!=0)
    {
        king = q.front();
        q.pop();
        cout<<king<<" ";
        for(int i=0;i<n;i++)
        {
            if(ad[king][i]==1 && mark[i]==0)
            {
                q.push(i);
                visited[i]=1;
                mark[i] = 1;
                dist[i] = dist[king]+1;
                parent[i] = king;
            }
        }
    }
}
int main()
{
    initadj();
    initmark();
    int n,e,i,j,c=0;
    cin>>n>>e;
```

```
    for(i=0;i<e;i++)
    {
        int x,y;
        cin>>x>>y;
        ad[x][y] = 1;
    }
    for(i=0;i<n;i++)
    {
        if(visited[i]==0)
        {   c++;
            bfs(i,n);
        }
    }
    cout<<"Number of water supply stations: "<<c<<"\n";
    /*for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            cout<<ad[i][j]<<" ";
        }
        cout<<"\n";
    }*/
}
```

BFS(more Modifications):

```
#include<bits/stdc++.h>
using namespace std;
int n = 9;
```

```
int main()
{
    int v[n];
    int g[n][n];
    int color[n];
    int parent[n];
    int dist[n];
    int i;
    for(i=0; i<n; i++)
    {
        v[i] = 0;
        color[i] = 0;
        parent[i] = 0;
        dist[i] = 0;
    }
    for(i=0; i<n; i++)
    {
        for(int j = 0; j<n; j++)
        {
            g[i][j] = 0;
        }
    }
    g[0][1] = 1;
    g[0][2] = 1;
    g[0][3] = 1;
    g[1][4] = 1;
    g[1][5] = 1;
    g[2][3] = 1;
    g[2][6] = 1;
    g[3][7] = 1;
    g[3][8] = 1;
    g[7][8] = 1;
    int c = 0;
    queue<int>q;
    q.push(0);
    color[0] = 1;
    parent[0] = -1;
    v[0] = 1;
    while(!q.empty())
    {
        int k = q.front();
        cout<<k<<" "<<dist[k]<<"\n";
        q.pop();
        for(i=0; i<n; i++)
```

```
        {
            if(g[k][i]==1 && color[i] == 0)
            {
                q.push(i);
                color[i] = 1;
                parent[i] = k;
                dist[i] = dist[k] + 1;
                v[i] = 1;
            }
            else if(g[k][i]==1 && color[i] == 1 && v[i]==1 && parent[k]!=i)
            {
                c++;
                g[k][i] = 0;
            }
        }
        color[k] = 2;

    }
    cout<<"Number of cycles: "<<c<<"\n";
    for(i=0;i<n;i++)
    {
        if(parent[i]!=-1)
        {
            cout<<"Child: "<<i<<" -> "<<"Parent: "<<parent[i]<<"\n";
        }
        if(parent[i]==-1)
        {
             cout<<"Root Child: "<<i<<" -> "<<"NULL(since its the root)\n";
        }
    }
    int x;
    cout<<"the node to find: ";
    cin>>x;
    vector<int>s;
    while(x!=-1)
    {
        s.push_back(x);
        x = parent[x];
    }
    for(i=0;i<s.size();i++)
    {
        cout<<s[i]<<" ";
    }

}
```

BFS(character):

```
#include<bits/stdc++.h>
using namespace std;
int n = 10;

int main()
{
    int v[n];
    int g[n][n];
    int color[n];
    int parent[n];
    int dist[n];
    int i;
    for(i=0; i<n; i++)
    {
        v[i] = -1;
        color[i] = 0;
        parent[i] = 0;
        dist[i] = 0;
    }
    for(i=0; i<n; i++)
    {
        for(int j = 0; j<n; j++)
        {
            g[i][j] = 0;
        }
    }
    g[0][4] = 1;
    g[0][5] = 1;
    g[4][6] = 1;
    g[4][7] = 1;
    g[4][8] = 1;
    g[5][1] = 1;
    queue<int>q;
    q.push(0);
    color[0] = 1;
    parent[0] = -1;
    while(!q.empty())
    {
        int k = q.front();
        char x = k + 65;
        cout<<x<<" "<<dist[k]<<"\n";
        q.pop();
        for(i=0; i<n; i++)
        {
```

```
            if(g[k][i]==1 && color[i] == 0)
            {
                q.push(i);
                color[i] = 1;
                parent[i] = k;
                dist[i] = dist[k] + 1;
            }
        }
        color[k] = 2;

    }
    for(i=0;i<n;i++)
    {
        if(parent[i]!=0 && parent[i]!=-1)
        {   char y = i + 65;
            char z = parent[i] + 65;
            cout<<y<<" -> "<<z<<"\n";
        }
        if(parent[i]==-1)
        {   /*char y = i + 65;
            char z = parent[i] + 65;*/
            cout<<i<<" -> "<<"NULL(since its the root)\n";
        }
    }

}
```

DFS(with modifications):

```
#include<bits/stdc++.h>
using namespace std;
int c=0,d=0,n=9,t=1;
int g[9][9]= {0};
int v[9];
int color[9]= {0};
int dist[9]= {0};
int parent[9]= {0};
int post[9];
int pre[9];
void dfs_visit(int k)
{
    //char x = k + 65;
    cout<<k<<" ";
    color[k]=1;
    v[k] = 1;
```

```
    pre[k] = t;
    t++;
    for(int i=0; i<n; i++)
    {
        if(g[k][i]==1 && color[i]==0)
        {
            dist[i]=dist[k]+1;
            parent[i] = k;
            dfs_visit(i);
        }
        else if(g[k][i]==1 && v[i]==1 && parent[k]!=i)
        {
            c++;
            g[k][i] = 0;
        }
    }
    color[k]=2;
    post[k] = t;
    t++;




}
int main()
{

    g[0][1]=1;
    g[0][2]=1;
    g[0][3]=1;
    g[1][4]=1;
    g[1][5]=1;
    g[2][3]=1;
    g[2][6]=1;
    g[3][7]=1;
    g[3][8]=1;
    g[7][8]=1;
    dist[0]=0;
    for(int i=0; i<n; i++)
    {
        if(v[i]==0)
        {
            d++;
            dfs_visit(i);
        }
```

```
    }
    cout<<"\n";
    cout<<"Cycle: "<<c<<"\n";
    cout<<"Number of water supply stations: "<<d<<"\n";
    for(int i=0; i<n; i++)
    {
        cout<<"Pre time: "<<pre[i]<<" "<<"Post Time: "<<post[i]<<"\n";
    }
```

Hashtable:

```
#include<bits/stdc++.h>
using namespace std;
class Hashtable
{
public:
    static const int hashgroups = 10;
    list<pair<int,string>>table[hashgroups];
    bool isEmpty();
    int HashFunction(int key);
    void Insertitem(int key,string value);
    void RemoveItem(int key);
    string Searchtable(int key);
    void Printable();


};
bool Hashtable::isEmpty()
{
    int sum = 0,i;
    for(i=0; i<hashgroups; i++)
    {
        sum = sum + table[i].size();
    }
    if(!sum)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
int Hashtable::HashFunction(int key)
{
    return key%hashgroups;
```

```
}
void Hashtable::Insertitem(int key,string value)
{
    int hashvalue = HashFunction(key);
    auto &cell = table[hashvalue];
    auto bitr = begin(cell);
    bool Keyexists = false;
    /*for(; bitr!=end(cell); bitr++)
    {
        if(bitr->first==key)
        {
            Keyexists = true;
            bitr->second = value;
            break;
        }
    }*/
    if(!Keyexists)
    {
        cell.emplace_back(key,value);
    }
    /*else
    {
        return;
    }*/
}
void Hashtable::RemoveItem(int key)
{
    int hashvalue = HashFunction(key);
    auto &cell = table[hashvalue];
    auto bitr = begin(cell);
    bool Keyexists = false;
    for(; bitr!=end(cell); bitr++)
    {
        if(bitr->first==key)
        {
            Keyexists = true;
            bitr = cell.erase(bitr);
            break;
        }
    }
    if(!Keyexists)
    {
        cout<<"Value removed\n";
    }
    else
```

```
    {
        cout<<"key does not exist\n";
    }


}
string Hashtable:: Searchtable(int key)
{
    int hashvalue = HashFunction(key);
    auto &cell = table[hashvalue];
    auto bitr = begin(cell);
    bool Keyexists = false;
    for(; bitr!=end(cell); bitr++)
    {
        if(bitr->first==key)
        {
            Keyexists = true;
            return bitr->second;
            break;
        }
    }
    if(!Keyexists)
    {
        cout<<"Key does  exist\n";
    }
    else
    {
        cout<<"Key does not exist\n";
    }

}
void Hashtable:: Printable()
{
    int i;
    for(i=0;i<hashgroups;i++)
    {
        if(table[i].size()==0)
        {
            continue;
        }
        else
        {
            auto bitr = table[i].begin();
            for(;bitr!=table[i].end();bitr++)
            {
```

```
                cout<<bitr->first<<"->"<<bitr->second<<" ";
                cout<<"\n";
            }
        }
    }
}
int main()
{
    Hashtable ht;
    if(ht.isEmpty())
    {
        cout<<"no problem\n";
    }
    else
    {
        cout<<"there is a hidden problem\n";
    }
    int i;
    for(i=0;i<10;i++)
    {
        int k;
        string s;
        cin>>k>>s;
        ht.Insertitem(k,s);
    }
    ht.Printable();

}
```