**PRACTICAL – 6**

**6.1 AIM: Write a program to implement BFS and DFS in Graph. Compare**

**Time Complexity of both algorithms.**

**PROGRAM CODE**:

#include <iostream>

#include <list>

#include <limits.h>

using namespace std;

class Graph

{

int V;

list<int> \*adj;

bool isCyclicUtil(int v, bool visited[], int parent);

public:

Graph(int V);

void addEdge(int v, int w);

bool isCyclic();

};

Graph::Graph(int V)

{

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

adj[v].push\_back(w);

adj[w].push\_back(v);

}

bool Graph::isCyclicUtil(int v, bool visited[], int parent)

{

visited[v] = true;

list<int>::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

{

if (!visited[\*i])

{

if (isCyclicUtil(\*i, visited, v))

return true;

}

else if (\*i != parent)

return true;

}

return false;

}

bool Graph::isCyclic()

{

bool \*visited = new bool[V];

for (int i = 0; i < V; i++)

visited[i] = false;

for (int u = 0; u < V; u++)

if (!visited[u])

if (isCyclicUtil(u, visited, -1))

return true;

return false;

}

int main()

{

int i, n, v;

cout << "Enter total number of vertices :";

cin >> v;

cout << "Enter total number of edges :";

cin >> n;

Graph g(v);

int a, b;

cout << "Enter edges : ";

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

{

cin >> a >> b;

g.addEdge(a, b);

}

if (g.isCyclic())

cout << "Graph contains cycle";

else

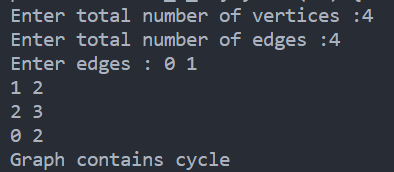
cout << "Graph doesn't contain cycle";

return 0;

cout << "\n20DCE019-Yatharth Chauhan";

}

**OUTPUT**:



**6.2 AIM: From a given vertex in a weighted graph, implement a program to find shortest paths to other vertices using Dijkstra’s algorithm.**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Adjacency Matrix of graph** | **Start Vertex** |
| 1 |  | 1 |
| 2 |  | 3 |

**PROGRAM CODE**:

#include<bits/stdc++.h>

using namespace std;

int minDistance(int dist[],bool visited[],int V)

{

int min\_index = 0;

int min = INT\_MAX;

for(int i=0;i<V;i++)

{

if(!visited[i] && dist[i]<=min)

{

min\_index = i;

min = dist[i];

}

}

return min\_index;

}

void dijkstra(int V,int source\_index)

{

int graph[V][V];

memset(graph,0,sizeof(graph));

cout<<"Consider Undirected Graph starts with index 0.\n";

for(int i=0;i<V;i++)

{

int n;

cout<<"Enter how many edges are connected with vertex "<<i<<": ";

cin>>n;

for(int j=0;j<n;j++)

{

int vertex,weight;

cout<<"Enter which vertex is connected with "<<i<<" and weight of it: ";

cin>>vertex>>weight;

graph[i][vertex] = weight;

}

cout<<"\n";

}

bool visited[V];

int dist[V];

for(int i=0;i<V;i++)

dist[i] = INT\_MAX;

dist[source\_index] = 0;

for(int i=0;i<V;i++)

{

int u = minDistance(dist,visited,V);

visited[u] = true;

for(int v=0;v<V;v++)

{

if(!visited[v] && graph[u][v] && dist[u]!=INT\_MAX &&

dist[u]+graph[u][v] < dist[v])

dist[v] = dist[u]+graph[u][v];

}

}for(int i=0;i<V;i++)

{

if(dist[i] == INT\_MAX)

cout<<"Distance from "<<source\_index<<" to "<<i<<" is infinite\n";

else

cout<<"Distance from "<<source\_index<<" to "<<i<<" is: "<<dist[i]<<"\n";

}

}

int main()

{

int V;

cout<<"Enter total vertex: ";

cin>>V;

int source;

cout<<"Enter Source Index: ";

cin>>source;

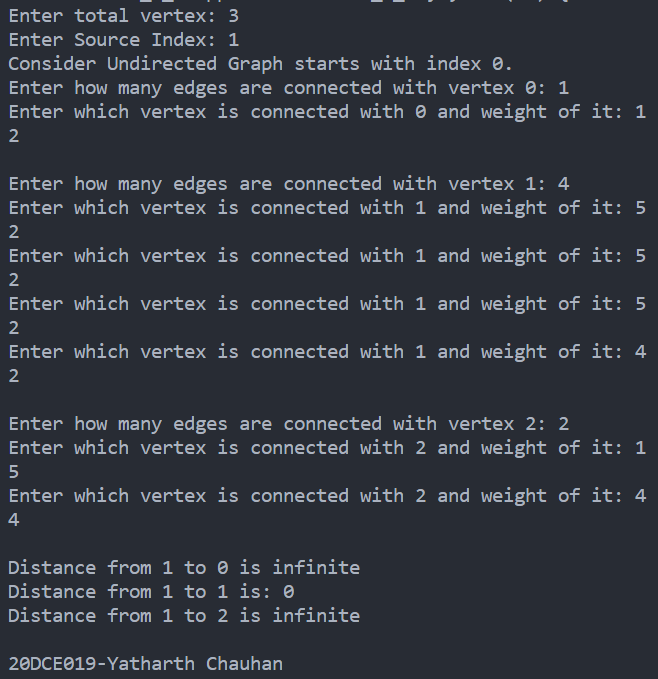
dijkstra(V,source);

cout << "\n20DCE019-Yatharth Chauhan";

return 0;

}

**OUTPUT**:



**PRACTICAL – 7**

**AIM: Program to implement 8-Queen’s problem using Backtracking.**

**PROGRAM CODE**:

#include<stdio.h>

#include<math.h>

void queen(int row, int p);

int chess[8],count;

int main()

{

int p = 8;

queen(1,p);

return 0;

printf("\n20DCE019-Yatharth Chauhan");

}

void print(int p)

{

int i,j;

printf("\n\nThis is Solution no. %d:\n\n",++count);

for(i=1;i<=p;++i)

printf("\t%d",i);

for(i=1;i<=p;++i){

printf("\n\n%d",i);

for(j=1;j<=p;++j){

if(chess[i]==j)

printf("\tQ");

else

printf("\t-");

}

}

printf("\n\n\nThere are total 92 solutions for 8-queens problem.");

}

int place(int row,int column)

{

int i;

for(i=1;i<=row-1;++i)

{

if(chess[i]==column)

return 0;

else

if(abs(chess[i]-column)==abs(i-row))

return 0;

}

return 1;

}

void queen(int row,int p)

{

int column;

for(column=1;column<=p;++column)

{

if(place(row,column))

{

chess[row]=column;

if(row==p)

print(p);

else

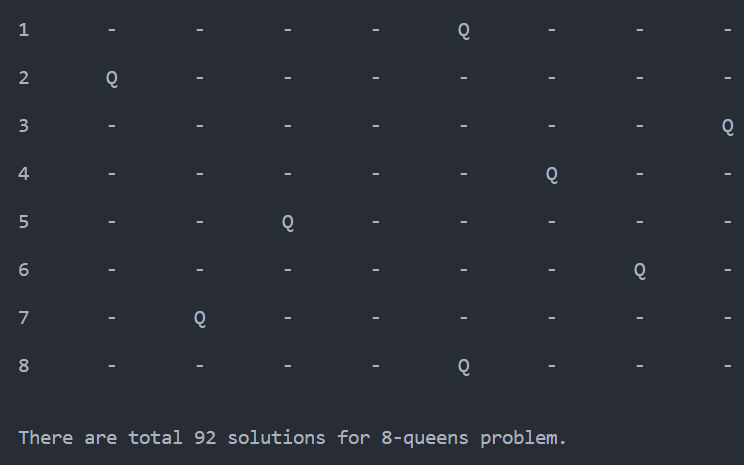
queen(row+1,p);

}

}

}

**OUTPUT:**

****

**PRACTICAL – 8**

**8.1 AIM: Suppose you are given a source string S[0 ..n − 1] of length n, consisting of symbols a and b. Suppose that you are given a pattern string P[0 ..m − 1] of length m < n, consisting of symbols a, b, and \*, representing a pattern to be found in string S. The symbol \* is a “wild card” symbol, which matches a single symbol, either a or b. The other symbols must match exactly. The problem is to output a sorted list M of valid “match positions”, which are positions j in S such that pattern P matches the substring S [j..j + |P|− 1]. For example, if S = ababbab and**

**P = ab\*, then the output M should be [0, 2]. Implement a straightforward, naive algorithm to solve the problem.**

**PROGRAM CODE:**

#include <iostream>

#include <string.h>

using namespace std;

int main()

{

char t[100], p[100];

int tn, pn, shift[20] = {0}, s = 0, i, j = 0, count = 0, m = 0;

cout << "\nEnter The Text : ";

cin >> t;

cout << "\nEnter The Pattern : ";

cin >> p;

tn = strlen(t);

pn = strlen(p);

while (s != (tn - pn + 1))

{

j = 0;

for (i = s; i < pn + s; i++)

{

if (p[j] == t[i])

{

count++;

if (count == pn)

{

count = 0;

shift[m] = s;

m++;

}

}

else

{

count = 0;

break;

}

j++;

}

s++;

}

if (m > 0)

{

cout << "\n\nValid Shifts: \n";

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

cout << shift[i] << "\n";

}

else

{

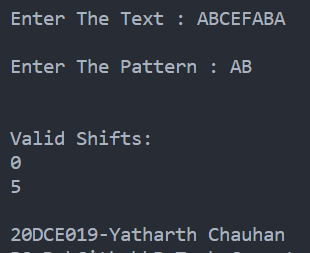
cout << "\n\nNo Valid Shifts.";

}

cout << "\n20DCE019-Yatharth Chauhan";

}

**OUTPUT:**

S

**8.2 AIM: Implement Rabin karp algorithm and test it on the**

**following test cases:**

|  |  |  |
| --- | --- | --- |
| **Test**  **Case** | **String** | **Pattern** |
| 1 | 2359023141526739921 | 31415 q=13 |
| 2 | ABAAABCDBBABCDDEBCABC | ABC q=101 |

**PROGRAM CODE:**

#include <string.h>

#include <iostream>

using namespace std;

#define d 10

void rabinKarp(char pattern[], char text[], int q)

{

int m = strlen(pattern);

int n = strlen(text);

int i, j;

int c = 0;

int p = 0;

int t = 0;

int h = 1;

for (i = 0; i < m - 1; i++)

h = (h \* d) % q;

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

{

p = (d \* p + pattern[i]) % q;

t = (d \* t + text[i]) % q;

}

for (i = 0; i <= n - m; i++)

{

if (p == t)

{

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

{

if (text[i + j] != pattern[j])

break;

c++;

}

if (j == m)

cout << "\n\nPattern is found at position: " << i + 1 << endl;

}

if (i < n - m)

{

c++;

t = (d \* (t - text[i] \* h) + text[i + m]) % q;

if (t < 0)

t = (t + q);

}

}

cout << "\ncounter:" << c;

}

int main()

{

char text[] = "2359023141526739921";

char pattern[] = "31415";

cout << "Text : " << text;

cout << "\nPattern :" << pattern;

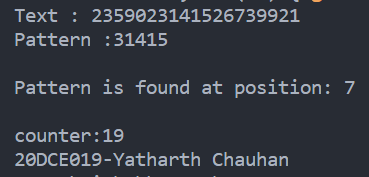
int q = 13;

rabinKarp(pattern, text, q);

cout << "\n20DCE019-Yatharth Chauhan";

}

**OUTPUT:**

****