**PRACTICAL – 6**

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

**Time Complexity of both algorithms.**

**PROGRAM CODE:**

**OUTPUT:**

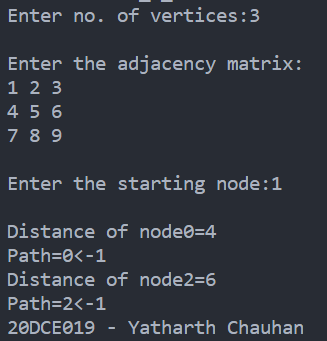
**CONCLUSION:**

**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**:

**OUTPUT**:



**CONCLUSION:** **Dijkstra’s algorithm is very similar to Prim’s algorithm for minimum spanning tree. Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks.Time Complexity: O(ElogV)**

**PRACTICAL – 7**

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

**PROGRAM CODE:**

**OUTPUT:**

**CONCLUSION:**

**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:**

**OUTPUT:**

**CONCLUSION:**

**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 <bits/stdc++.h>

#define int long long int

#define F first

#define S second

#define pb push\_back

using namespace std;

#define d 19

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

{

int m = strlen(pattern);

int n = strlen(text);

int i, j;

int p = 0;

int t = 0;

int h = 1;

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

h = (h \* d) % q;

// Calculate hash value for pattern and text

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;

}

if (j == m)

cout << "Pattern is found at position: " << i + 1 << endl;

}

if (i < n - m)

{

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

if (t < 0)

t = (t + q);

}

}

}

int32\_t main()

{

#ifndef ONLINE\_JUDGE

freopen("input.txt", "r", stdin);

freopen("output.txt", "w", stdout);

#endif

char text[] = "2359023141526739921";

char pattern[] = "31415";

int q = 13;

rabinKarp(pattern, text, q);

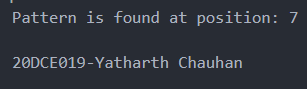
cout << "\n20DCE019-Yatharth Chauhan";

return 0;

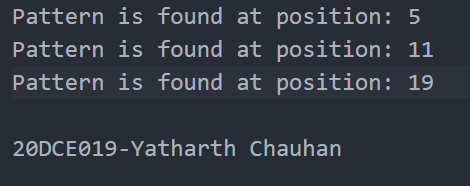
}

**OUTPUT:**

**Test Case-1:**

****

**Test Case-1:**

****

**CONCLUSION:** In this practical I learnt to implement Rabin-Karp algorithm and using the concept of hashes I ran the program for two test cases. The average and best-case complexity of this algorithm is O(m+n) and for worst case it is O(m\*n).