Lab -

DIP : Design and Analysis of Algorithm

AIM : Implement the following Algorithms :To understand the difference between

(1) String Edit Distance (Dynamic Programming)

(2) Prim's Algorithm for Minimum Spanning Tree (Greedy Algorithms)

Name : Ambalia Harshit

Department : Computer Engineering (M.Tech sem II)

Roll no : MT001

Date : Jan 2024

Program 01 : Prim's Algorithm for Minimum Spanning Tree (Greedy Algorithms)

- Description :

Prim's Algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be Minimized.

Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

How does the prim's algorithm work? Prim's algorithm is a greedy algorithm that starts from one vertex and continues to add the edges with the smallest weight until the goal is reached.

The steps to implement the prim's algorithm are given as follows

- First, we have to initialize an MST with the randomly chosen vertex.
- Now, we have to find all the edges that connect the tree in the above step with the new vertices. From the edges found, select the minimum edge and add it to the tree.
- Repeat step 2 until the minimum spanning tree is formed.

- Algorithm :

MST-PRIM (G, w, r)
1. for each u ∈ V [G]
2. do key [u] ← ∞
3. π [u] ← NIL
4. key [r] ← 0
5. Q ← V [G]
6. While Q ? ∅
7. do u ← EXTRACT - MIN (Q)
8. for each v ∈ Adj [u]
9. do if v ∈ Q and w (u, v) < key [v]
10. then π [v] ← u
11. key [v] ← w (u, v)

- Time Complexity : O((V + E)log V)

- Code :

```
// prog 01 : Prims algrithm

#include <stdio.h>
#define INF 99999

void prims_algorithm(int cost[], int parent[], int isVisited[], int start,
int n, int graph[n][n] ) {
   cost[start] = 0;
   parent[start] = -1;

   for( int v=0;v<n-1;v++ ) {
       int minCost = INF, minIndex;
       for( int i=0;i<n;i++ )  {
           if( isVisited[i]==0 && cost[i]<minCost ) {
               minCost = cost[i];
               minIndex = i;
           }
       }

       isVisited[minIndex] = 1;

       for( int i=0;i<n;i++ ) {
           if (graph[minIndex][i] && isVisited[i]==0 &&
graph[minIndex][i]<cost[i] ) {
               parent[i] = minIndex;
               cost[i] = graph[minIndex][i];
           }
       }
   }

   printf("Edge\tWeight\n");
   for( int i=1;i<n;i++ ) {
       printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
   }
}

int main() {
```

```
    int n = 5;
    int graph[5][5] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };

    int parent[n];
    int cost[n];
    int isVisited[n];

    for( int i=0;i<n;i++ ) {
        cost[i] = INF;
        isVisited[i] = 0;
    }

    prims_algorithm(cost, parent, isVisited, 0, n, graph);

    return 0;
}
```

- Output Screen-shots :

```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_05$ cd "
cuments/Semester_10/Lab_DAA/Lab_05/"ex01
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_05$
```

Program 02 : String Edit Distance (Dynamic Programming):

- Description :

  The algorithm begins by creating a table with rows and columns, where the rows represent the characters of the first string and the columns represent the characters of the second string.

  Each cell in the table represents the minimum number of edits required to transform the substring of the first string up to that row into the substring of the second string up to that column.

  The algorithm then proceeds by filling in the cells of the table row by row, using the following rules:

1. If the characters in the corresponding positions of the two strings are the same, the edit distance is the same as the edit distance between the two substrings without that character.

2. If the characters in the corresponding positions of the two strings are different, the edit distance is the minimum of the three possible operations: a substitution, a deletion, or an insertion.

3. The substitution operation is represented by the cell diagonally up and left of the current cell, plus 1.

4. The deletion operation is represented by the cell above the current cell, plus 1.
5. The insertion operation is represented by the cell to the left of the current cell, plus 1.

The final edit distance is stored in the bottom-right corner of the table, and the algorithm can also be used to trace the optimal sequence of edits to transform the first string into the second string.

The time complexity of the dynamic programming algorithm is O(nm) where n is the length of the first string and m is the length of the second string. The space complexity is also O(nm) as it uses a table to store the solutions.

- Algorithm :

Algorithm :
Begin
if initLen = 0, then
    return finalLen
if finalLen := 0, then
    return initLen
if initStr[initLen - 1] = finalStr[finalLen - 1], then
    return editCount(initStr, finalStr, initLen – 1, finalLen - 1)
    answer := 1 + min of (editCount(initStr, finalStr, initLen , finalLen - 1)),
    (editCount(initStr, finalStr, initLen – 1, finalLen),
    (editCount(initStr, finalStr, initLen – 1, finalLen - 1)
    return answer

End

- Code :
    - Following code does not represent the given algorithm, instead it recursively finds the solution while utilizing the memory table.

```
// prog 02
#include <stdio.h>
#include <string.h>

int minimum_of_three_integer(int x, int y, int z) {
   if (x <= y && x <= z) return x;
   if (y <= x && y <= z) return y;
   return z;
}

int edit_string_distance(char *init_string, char *final_string, int
init_length, int final_length, int memo[][final_length+1], int n) {
   // Base cases
   if (init_length == 0)
       return final_length;
   if (final_length == 0)
       return init_length;
```

```
   // If already computed, return memoized value
   if (memo[init_length][final_length] != -1)
       return memo[init_length][final_length];

   // If last characters are same, ignore last characters and recur for
remaining strings
   if (init_string[init_length - 1] == final_string[final_length - 1])
       return memo[init_length][final_length] =
edit_string_distance(init_string, final_string, init_length-1,
final_length-1, memo, n);

   // If last characters are not same, consider all three operations
   return memo[init_length][final_length] = 1 + minimum_of_three_integer(
       edit_string_distance(init_string, final_string, init_length,
final_length-1, memo, n), // Insert
       edit_string_distance(init_string, final_string, init_length-1,
final_length, memo, n), // Remove
       edit_string_distance(init_string, final_string, init_length-1,
final_length-1, memo, n) // Replace
   );
}

int main() {
   char init_string[] = "harshit";
   char final_string[] = "itharsh";
   int init_length = strlen(init_string);
   int final_length = strlen(final_string);

   int n = 100;
   int memo[n][n];

   memset(memo, -1, sizeof(memo));

   printf("Operations required : %d\n", edit_string_distance(init_string,
final_string, init_length, final_length, memo, n));
   return 0;
}
```

- Output Screen-Shots :

```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_05$ cd "
cuments/Semester_10/Lab_DAA/Lab_05/"pr02
Operations required : 4
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_05$
```

- Output Screen-shots :

```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_03$ cd "/home/hr/Docum
ex02.c -o ex02 && "/home/hr/Documents/Semester_10/Lab_DAA/Lab_03/"e
Value : 141
Coins Are : 2 5 10 25 50
We need 2 Coin of rupees 50 = 100
We need 1 Coin of rupees 25 = 25
We need 1 Coin of rupees 10 = 10
We need 1 Coin of rupees 5 = 5
Can generate change for 140 only with given coins, 1 is remaining.
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_03$
```

```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_03$ cd "/home/hr/Docum
ex02.c -o ex02 && "/home/hr/Documents/Semester_10/Lab_DAA/Lab_03/"e
Value : 142
Coins Are : 2 5 10 25 50
We need 2 Coin of rupees 50 = 100
We need 1 Coin of rupees 25 = 25
We need 1 Coin of rupees 10 = 10
We need 1 Coin of rupees 5 = 5
We need 1 Coin of rupees 2 = 2
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_03$
```