Name: Sanket Wakankar

Reg. No.: 2020BIT055

Practical No.4

\*\*Travelling Salesman Problem\*\*

#include <bits/stdc++.h>

using namespace std;

#define V 4

// implementation of traveling Salesman Problem

int travllingSalesmanProblem(int graph[][V], int s)

{

    vector<int> vertex;

    for (int i = 0; i < V; i++)

        if (i != s)

            vertex.push\_back(i);

    int min\_path = INT\_MAX;

    do {

        int current\_pathweight = 0;

        int k = s;

        for (int i = 0; i < vertex.size(); i++) {

            current\_pathweight += graph[k][vertex[i]];

            k = vertex[i];

        }

        current\_pathweight += graph[k][s];

        min\_path = min(min\_path, current\_pathweight);

    } while (

        next\_permutation(vertex.begin(), vertex.end()));

    return min\_path;

}

int main()

{

    int graph[][V] = { { 0, 10, 15, 20 },

                    { 10, 0, 35, 25 },

                    { 15, 35, 0, 30 },

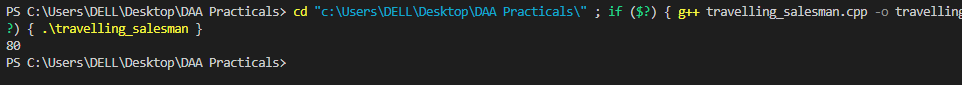
                    { 20, 25, 30, 0 } };

    int s = 0;

    cout << travllingSalesmanProblem(graph, s) << endl;

    return 0;

}

OUTPUT: 

\*\*BF String Matching Algorithm\*\*

#include<iostream>

#include<string>

using namespace std;

int BF(string text ,string pattern){

    int n = text.length();

    int m = pattern.length();

    for(int i=0;i<=n-m;i++){

        int j=0;

        while(j<m && text[i+j]==pattern[j]){

            j++;

        }

        if(j==m){

            return i;

        }

    }

    return -1;

}

int main(){

    string text = "sanket";

    string pattern = "an";

    int pos = BF(text,pattern);

    if(pos != -1){

        cout<<"pattern found at postion "<<pos<<endl;

    }

    else{

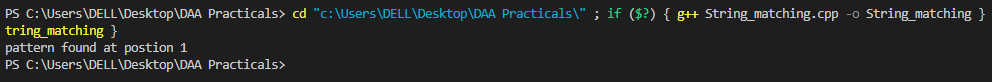
        cout<<"pattern not found "<<endl;

    }

    return 0;

}

OUTPUT:



\*\*Exhaustive Search Algorithm\*\*

// Sanket Wakankar

#include <bits/stdc++.h>

using namespace std;

int maxPackedSets(vector<int>& items,

                vector<set<int> >& sets)

{

int maxSets = 0;

// Loop through all the sets

for (auto set : sets) {

    int numSets = 0;

    // Loop through all the items

    for (auto item : items) {

    // Check if the current item is in the current set

    if (set.count(item)) {

        // If the item is in the set, increment

        // the number of sets that can be packed

        numSets += 1;

        // Remove the item from the set of items,

        // so that it is not counted again

        items.erase(remove(items.begin(),

                        items.end(), item),

                    items.end());

    }

    }

    // Update the maximum number of sets that can be

    // packed

    maxSets = max(maxSets, numSets+1);

}

return maxSets;

}

int main()

{

// Set of items

vector<int> items = { 1, 2, 3, 4, 5, 6,7,8 };

// List of sets

vector<set<int> > sets

    = { { 1, 2, 3 },{7,8}, { 4, 5 }, { 5, 6 }, { 1, 4 } };

// Find the maximum number of sets that

// can be packed into the given set of items

int maxSets

    = maxPackedSets(items, sets);

// Print the result

cout << "Maximum number of sets that can be packed: "

    << maxSets << endl;

return 0;

}

OUTPUT:

