

**Assignment-4**

**Name: - Mann Kothari**

**Reg. No.: - 22BCB7064**

**Course Title: - Design and Analysis of Algorithm (Embedded Lab)**

**Course Code: - CSE3023**

**Slot: - L21+L22**

**Submitted to: - Prof. Tanikella Divya Naga Pavani**

* **INDEX**
* Travelling Salesman Problem Using Brute Force.
* Algorithm
* Code
* Output
* Complexity
* 0/1 Knapsack Problem Using Brute Force
* Algorithm
* Code
* Output
* Complexity
* Complexity Chart

1. Write a C/C++ program to Implement the Travelling Salesperson Problem (TSP) using Brute Force.

Algorithm: -

Algorithm TSP(graph, current, visited, path, cost, min\_cost)

{

If(all vertices visited)

cost = CalculateTotalCost(path);

if(cost<min\_cost)

{

min\_cost = cost;

return;

}

for(vertex in graph(

{

if (vertex is not visited)

{

Add vertex to path;

Mark vertex as visited;

TSP(graph, current, visited, path, cost, min\_cost);

}

Remove last vertex from path;

Mark vertex a unvisited;

}

}

Time Complexity: -

O(2n\*n2)

Code: -

#include <stdio.h>

#include <limits.h>

#define V 4

int next\_permutation(int arr[], int size)

{

    int i = size - 1;

    while (i > 0 && arr[i - 1] >= arr[i])

    {

        i--;

    }

    if (i <= 0)

    {

        return 0;

    }

    int j = size - 1;

    while (arr[j] <= arr[i - 1])

    {

        j--;

    }

    int temp = arr[i - 1];

    arr[i - 1] = arr[j];

    arr[j] = temp;

    j = size - 1;

    while (i < j)

    {

        temp = arr[i];

        arr[i] = arr[j];

        arr[j] = temp;

        i++;

        j--;

    }

    return 1;

}

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

{

    int vertex[V - 1];

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

    {

        if (i != s)

        {

            vertex[k] = i;

            k++;

        }

    }

    int min\_path = INT\_MAX;

    do

    {

        int current\_pathweight = 0;

        int k = s;

        for (int i = 0; i < V - 1; i++)

        {

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

            k = vertex[i];

        }

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

        if (current\_pathweight < min\_path)

        {

            min\_path = current\_pathweight;

        }

    } while (next\_permutation(vertex, V - 1));

    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;

    printf("The optimal cost is: %d\n", travllingSalesmanProblem(graph, s));

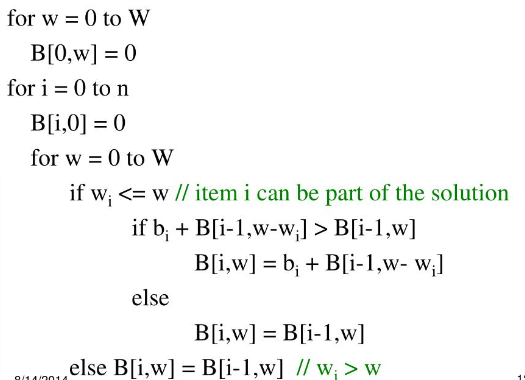
    return 0;

}

Output: -

1. Write a C/C++ program to Implement the 0/1 Knapsack  Problem using Brute Force.

Algorithm: -



Time Complexity: -

O(2n)

Code: -

#include <stdio.h>

int max(int a, int b) { return (a > b) ? a : b; }

int knapSack(int W, int wt[], int val[], int n)

{

    if (n == 0 || W == 0)

        return 0;

    if (wt[n - 1] > W)

        return knapSack(W, wt, val, n - 1);

    else

        return max(

            val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1),

            knapSack(W, wt, val, n - 1));

}

int main()

{

    int profit[] = {42, 12, 40, 25};

    int weight[] = {7, 3, 4, 5};

    int W = 50;

    int n = sizeof(profit) / sizeof(profit[0]);

    printf(" The maximum profit within given weight %d is: %d", W, knapSack(W, weight, profit, n));

    return 0;

}

Output: -

|  |  |  |
| --- | --- | --- |
| Algorithm | Time Complexity | Space Complexity |
| TSP(Brute Force) | O(2n\*n2) | O(n) |
| 0/1 Knapsack (Brute Force) | O(2n) | O(n) |