

**Assignment-5**

**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**
* Job Assignment Problem using Brute Force.
* Algorithm
* Code
* Output
* Complexity
* Fractional Knapsack using Greedy Method
* Algorithm
* Code
* Output
* Complexity
* Complexity Chart

1. Write a C/C++ program to Implement the Job Assignment  Problem using Brute Force.

Algorithm: -

Algorithm JobAssign(CostMatrix, Cost, Assigned, index)

{

if(index==N)then//N is the size of 2-D matrix

{

if(Cost<min\_cost)

{

min\_cost:=cost;

for i:=0 to N do

{

min\_assignment[i]:=assigned[i];

}

}

return;

}

for i:=index to N do

{

swap(assigned[index],assigned[i]);

JobAssign(CostMatrix, Cost, Assigned, index);

swap(assigned[index],assigned[i]);

}

}

Time Complexity: -

O(n!)

Code: -

#include <stdio.h>

#include <limits.h>

#define N 4 // Number of workers and jobs

int minCost = INT\_MAX;

int minAssignment[N];

void swap(int \*a, int \*b)

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

void findMinCost(int costMatrix[N][N], int cost, int assigned[], int index)

{

    if (index == N)

    {

        if (cost < minCost)

        {

            minCost = cost;

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

            {

                minAssignment[i] = assigned[i];

            }

        }

        return;

    }

    for (int i = index; i < N; i++)

    {

        swap(&assigned[index], &assigned[i]);

        findMinCost(costMatrix, cost + costMatrix[index][assigned[index]], assigned, index + 1);

        swap(&assigned[index], &assigned[i]);

    }

}

int main()

{

    int costMatrix[N][N] = {

        {9, 2, 7, 8},

        {6, 4, 3, 7},

        {5, 8, 1, 8},

        {7, 6, 9, 4}};

    int assigned[N];

    for (int i = 0; i < N; i++)

    {

        assigned[i] = i;

    }

    findMinCost(costMatrix, 0, assigned, 0);

    printf("Minimum cost: %d\n", minCost);

    printf("Assignment: ");

    for (int i = 0; i < N; i++)

    {

        printf("(%d, %d) ", i + 1, minAssignment[i] + 1);

    }

    printf("\n");

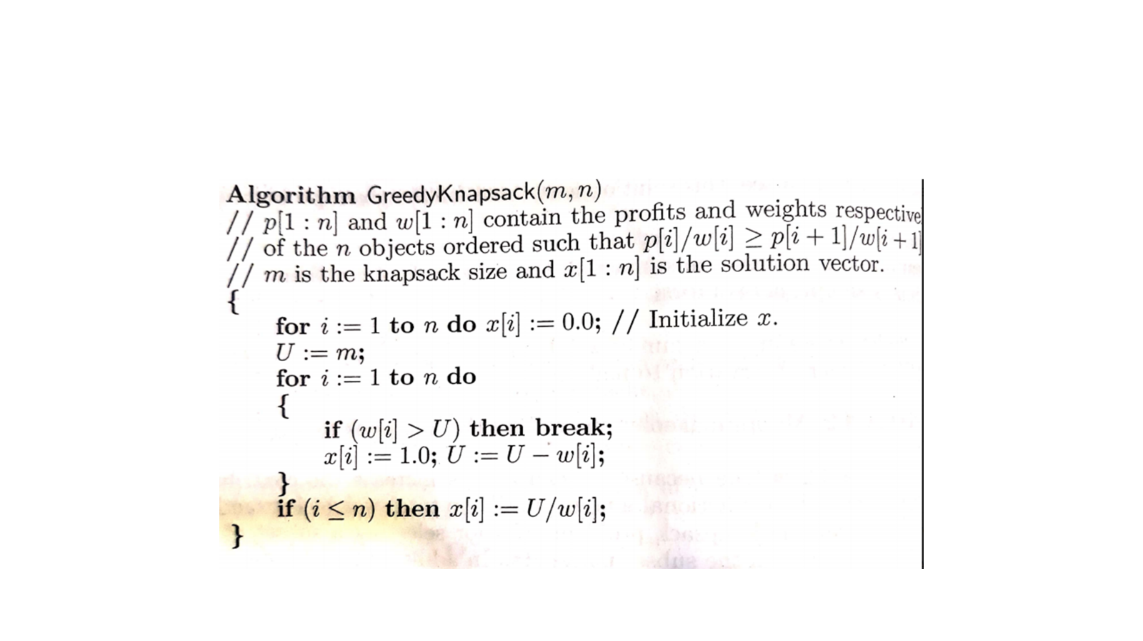
    return 0;

}

Output: -

1. Write a C/C++ program to Implement the Fractional Knapsack  Problem using the Greedy Method.

Algorithm: -



Time Complexity: -

O(2n)

Code: -

#include <stdio.h>

#include <stdlib.h>

typedef struct

{

    int weight;

    int value;

    float ratio;

} Item;

void swap(Item \*a, Item \*b)

{

    Item temp = \*a;

    \*a = \*b;

    \*b = temp;

}

void sortItemsByRatio(Item items[], int n)

{

    for (int i = 0; i < n - 1; i++)

    {

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

        {

            if (items[j].ratio < items[j + 1].ratio)

            {

                swap(&items[j], &items[j + 1]);

            }

        }

    }

}

float fractionalKnapsack(int capacity, Item items[], int n)

{

    float totalValue = 0.0;

    int currentWeight = 0;

    sortItemsByRatio(items, n);

    for (int i = 0; i < n; i++)

    {

        if (currentWeight + items[i].weight <= capacity)

        {

            currentWeight += items[i].weight;

            totalValue += items[i].value;

        }

        else

        {

            int remainingWeight = capacity - currentWeight;

            totalValue += items[i].ratio \* remainingWeight;

            break;

        }

    }

    return totalValue;

}

int main()

{

    int capacity = 50;

    Item items[] = {

        {10, 60, 0.0},

        {20, 100, 0.0},

        {30, 120, 0.0}};

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

    for (int i = 0; i < n; i++)

    {

        items[i].ratio = (float)items[i].value / items[i].weight;

    }

    float totalValue = fractionalKnapsack(capacity, items, n);

    printf("Maximum value in Knapsack = %.2f\n", totalValue);

    return 0;

}

Output: -

|  |  |  |
| --- | --- | --- |
| Algorithm | Time Complexity | Space Complexity |
| Job Assignment Brute Force | O(n!) | O(n) |
| Binary search | O(2n) | O(n) |