SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB EXERCISE 7

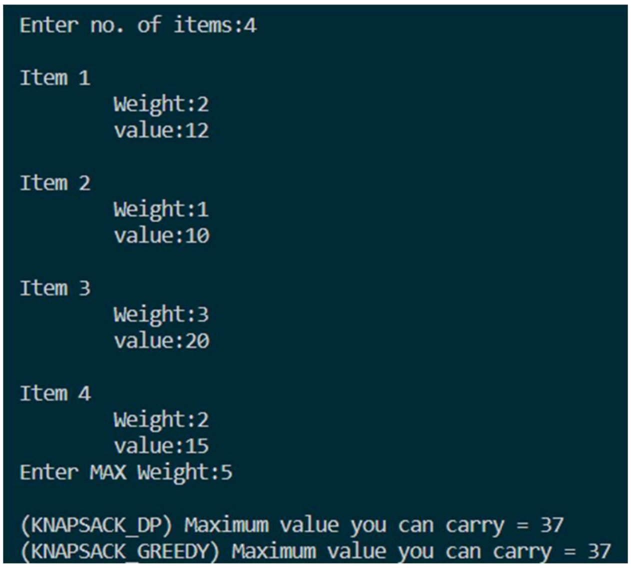
Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

1. To Implement Knapsack Algorithm in DP

Code:

```
#include <bits/stdc++.h>
using namespace std;

int KNAPSACK_GREEDY(int W, int v[], int w[], int n)
{
    if (n == 0 or W == 0)
    {
        return 0;
    }

    if (w[n - 1] > W)
    {
        return KNAPSACK_GREEDY(W, v, w, n - 1);
    }

    else
    {
        return max(KNAPSACK_GREEDY(W, v, w, n - 1), v[n - 1] + KNAPSACK_GREEDY(W - w[n -
1], v, w, n - 1));
    }
}

int KNAPSACK_DP(int W, int v[], int w[], int n)
{
    vector<vector<int>> F(n + 1, vector<int>(W + 1));

    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= W; j++)
        {
            if (i == 0 or j == 0)
            {
                F[i][j] = 0;
            }
            else if (w[i - 1] <= j)
            {
                F[i][j] = max(F[i - 1][j], v[i - 1] + F[i - 1][j - w[i - 1]]);
            }
            else
            {
                F[i][j] = F[i - 1][j];
            }
        }
    }
    return F[n][W];
}

int main()
{
    int n;
    cout << "\nEnter no. of items:";
    cin >> n;

    int v[n], w[n];
    for (int i = 0; i < n; i++)
    {
        cout << "\nItem " << i + 1 << endl;
        cout << "\tWeight:";
        cin >> w[i];
        cout << "\tvalue:";
        cin >> v[i];
    }

    int W;
    cout << "Enter MAX Weight:";
    cin >> W;

    cout << "\n(KNAPSACK_DP) Maximum value you can carry = " << KNAPSACK_DP(W, v, w, n);
    cout << "\n(KNAPSACK_GREEDY) Maximum value you can carry = " << KNAPSACK_GREEDY(W, v,
w, n);

    return 0;
}
```

Output:



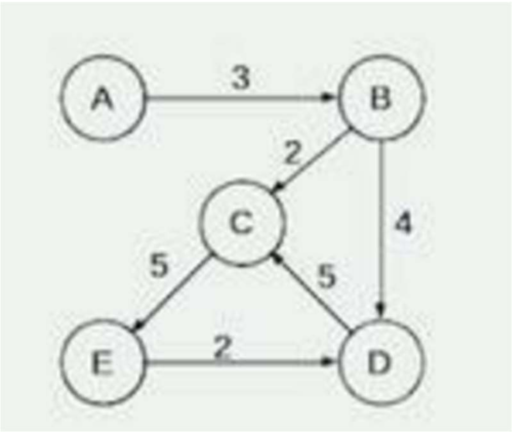
```
Enter no. of items:4

Item 1
        Weight:2
        value:12

Item 2
        Weight:1
        value:10

Item 3
        Weight:3
        value:20

Item 4
        Weight:2
        value:15
Enter MAX Weight:5

(KNAPSACK_DP) Maximum value you can carry = 37
(KNAPSACK_GREEDY) Maximum value you can carry = 37
```

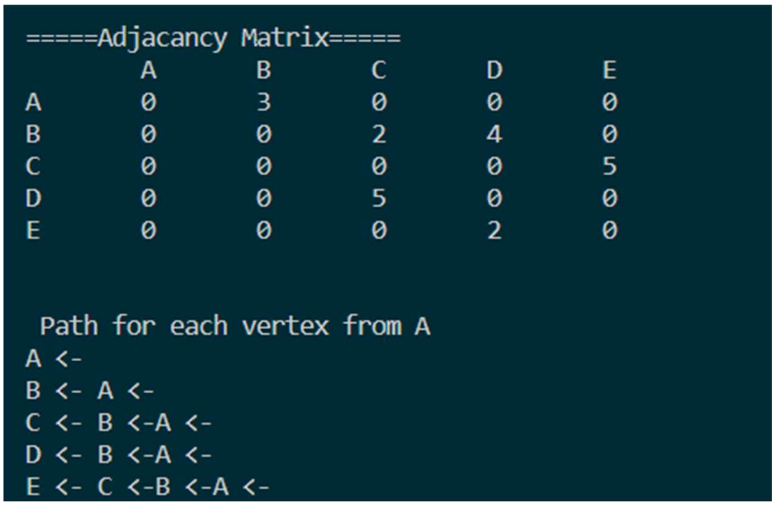
2. To Implement Dijkstra’s Algorithm for Shortest Path Algorithm

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
using namespace std;
typedef struct Graph *graph;

typedef struct Graph
{
    int nv;
    int **am;
    // Array to store constructed MST
    int *parent;
    // Key values used to pick minimum weight edge in cut
    int *key;
    // To represent set of vertices included in graph
    bool *MST_Set;

} Graph;

graph creategraph(int v)
{
    graph g = (graph)malloc(sizeof(Graph));

    g->nv = v;

    g->am = (int **)malloc(v * sizeof(int *));
    g->parent = (int *)malloc(v * sizeof(int *));
    g->key = (int *)malloc(v * sizeof(int *));
    g->MST_Set = (bool *)malloc(v * sizeof(bool *));
    for (int i = 0; i < v; i++)
    {
        g->am[i] = (int *)malloc(v * sizeof(int));
    }
    // initialising graph
    for (int i = 0; i < v; i++)
    {
        for (int j = 0; j < v; j++)
        {
            g->am[i][j] = 0;
        }
    }

    // Initialize all keys as INFINITE
    for (int i = 0; i < v; i++)
    {
        g->key[i] = INT_MAX;
        g->MST_Set[i] = false;
    }

    return g;
}

graph fillmatrix(graph g, int i, int j, int w)
{
    if (i < g->nv && j < g->nv)
    {
        g->am[i][j] = w;
        // g->am[j][i] = w;
    }
    return g;
}

graph getgraph(graph g)
{
    char v1, v2;
    int width;
    printf("\nEdge:\n\tvertice 1:");
    scanf(" %c", &v1);
    printf("\tvertice 2:");
    scanf(" %c", &v2);
    printf("\tweight:");
    scanf("%d", &width);

    while (v1 != '0' && v2 != '0')
    {
        int vv1 = v1 - 'A';
        int vv2 = v2 - 'A';
        g = fillmatrix(g, vv1, vv2, width);
        printf("\nEdge:\n\tvertice 1:");
        scanf(" %c", &v1);
        printf("\tvertice 2:");
        scanf(" %c", &v2);
        printf("\tWidth:");
        scanf("%d", &width);
    }
    return g;
}

void printadjmat(graph g)
{
    printf("\n=====Adjacancy Matrix=====\n");
    printf("\t");
    for (int i = 0; i < g->nv; i++)
    {
        printf("%c\t", 'A' + i);
    }
    printf("\n");

    for (int i = 0; i < g->nv; i++)
    {
        printf("%c\t", 'A' + i);
        for (int j = 0; j < g->nv; j++)
        {
            printf("%d\t", g->am[i][j]);
        }
        printf("\n");
    }
}

// function to pick minimum key from vertex
int minkey(graph g)
{
    int min_val = INT_MAX;
    int min_index;
    for (int i = 0; i < g->nv; i++)
    {
        if (g->MST_Set[i] == false and g->key[i] < min_val)
        {
            min_val = g->key[i];
            min_index = i;
        }
    }
    return min_index;
}

void printDijk(graph g, int src)
{
    cout << "\n\n Path for each vertex from " << (char)(src + 65) << endl;
    for (int i = 0; i < g->nv; i++)
    {
        cout << (char)(i + 65) << " <- ";
        if (i != src)
        {
            int j = i;
            do
            {
                j = g->parent[j];
                cout << char(j + 65) << " <-";
            } while (j != src);
        }
        cout << endl;
    }

}
void Dijkstra(graph g, int src)
{
    // g->MST_Set[src] = true;
    g->key[src] = 0;
    g->parent[src] = -1;
    for (int i = 0; i < g->nv - 1; i++)
    {
        // picking minimum key from vertex and not included in MST
        int min_key = minkey(g);
        // set the min_key as added to MST
        g->MST_Set[min_key] = true;

        // Updating the key value and parent index of the adjacent vertices of the
selected vertex
        for (int j = 0; j < g->nv; j++)
        {
            if ((g->am[min_key][j]) and (g->MST_Set[j] == false) and (g->am[min_key][j] <
g->key[j]))
            {
                g->parent[j] = min_key;
                g->key[j] = g->am[min_key][j];
            }
        }
    }
    printDijk(g, src);
}

int main(int argc, char const *argv[])
{
    int n;
    printf("\nEnter no. of vertices:");
    scanf("%d", &n);
    graph g;
    g = (graph)malloc(sizeof(Graph));
    g = NULL;
    g = creategraph(n);
    printf("\nEnter Edges(Enter '0 0 0' to exit):\n");
    g = getgraph(g);
    printadjmat(g);
    Dijkstra(g, 0);
}
```

Output:





```
=====Adjacency Matrix=====
        A       B       C       D       E
A       0       3       0       0       0
B       0       0       2       4       0
C       0       0       0       0       5
D       0       0       5       0       0
E       0       0       0       2       0


 Path for each vertex from A
A <-
B <- A <-
C <- B <-A <-
D <- B <-A <-
E <- C <-B <-A <-
```