ADIT LUHADIA

190911112

IT A

**LAB 4 – GREEDY I**

1. Container Loading Problem

#include <bits/stdc++.h>

using namespace std;

int counter;

int main()

{

    counter = 0;

    int n, c;

    cout << "enter the no. of boxes and total capacity\n";

    cin >> n >> c;

    int weight[1000], index[1000];

    cout << "enter the weights of boxes: \n";

    for (int i = 0; i < n; i++, counter++)

    {

        cin >> weight[i];

        counter++;

        index[i] = i + 1;

    }

    for (int i = 0; i < n; i++, counter++)

    {

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

        {

            counter++;

            if (weight[j] > weight[j + 1])

            {

                counter = counter + 3;

                swap(weight[j], weight[j + 1]);

                swap(index[j], index[j + 1]);

            }

        }

    }

    counter++;

    int k = 0, currWeight = 0;

    for (int i = 0; i < n; i++, counter++)

    {

        counter++;

        if (currWeight + weight[i] <= c)

        {

            counter++;

            currWeight += weight[i];

            counter++;

            k++;

        }

    }

    cout << "boxes included are: ";

    for (int i = 0; i < k; i++)

    {

        cout << index[i] << " ";

    }

    cout << endl;

    cout << "Time complexity is: " << counter;

    return 0;

}

Text

Description automatically generated

2. 0/1 Knapsack Problem

#include <bits/stdc++.h>

using namespace std;

int counter;

int main()

{

    counter = 0;

    int n, c;

    cout << "enter the no. of boxes and capacity:\n";

    cin >> n >> c;

    int weight[1000], prices[1000], index[1000], density[1000];

    cout << "enter the weights:\n";

    for (int i = 0; i < n; i++, counter++)

    {

        counter++;

        index[i] = i + 1;

        cin >> weight[i];

    }

    cout << "enter the prices:\n";

    for (int i = 0; i < n; i++, counter++)

        cin >> prices[i];

    for (int i = 0; i < n; i++, c++)

    {

        counter++;

        density[i] = prices[i] / weight[i];

    }

    for (int i = 0; i < n; i++, counter++)

    {

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

        {

            counter++;

            if (density[j] > density[j + 1])

            {

                counter = counter + 3;

                swap(density[j], density[j + 1]);

                swap(index[j], index[j + 1]);

                swap(prices[j], prices[j + 1]);

                swap(weight[j], weight[j + 1]);

            }

        }

    }

    counter++;

    int k = 0, currWeight = 0;

    for (int i = 0; i < n; i++, counter++)

    {

        counter++;

        if (k < n && currWeight + weight[i] <= c)

        {

            counter++;

            currWeight += weight[i];

            counter++;

            k++;

        }

    }

    cout << "the boxes included are: ";

    for (int i = 0; i < n; i++)

    {

        cout << i << " ";

    }

    cout << "\n";

    cout << "the complexity is: " << counter;

}

Text

Description automatically generated

3. Topological Sorting

#include <bits/stdc++.h>

using namespace std;

int c;

int main()

{

    c = 0;

    int v, e;

    cout << "enter the total vertixes and edges\n";

    cin >> v >> e;

    int \*\*adj = new int \*[v];

    for (int i = 0; i < v; i++, c++)

        adj[i] = new int[v];

    for (int i = 0; i < v; i++, c++)

    {

        for (int j = 0; j < v; j++, c++)

            adj[i][j] = 0;

    }

    cout << "enter the elements of the matrix:\n";

    for (int i = 0; i < e; i++, c++)

    {

        cout << "enter left and right of:" << i << "\n";

        int l, r;

        cin >> l >> r;

        adj[l][r] = 1;

    }

    c++;

    int indegree[1000] = {0};

    for (int j = 0; j < v; j++, c++)

    {

        for (int i = 0; i < v; i++, c++)

            indegree[j] += adj[i][j];

    }

    c++;

    queue<int> q;

    for (int i = 0; i < v; i++, c++)

    {

        c++;

        if (indegree[i] == 0)

        {

            c++;

            q.push(i);

        }

    }

    while (!q.empty())

    {

        c++;

        c++;

        int curr = q.front();

        c++;

        q.pop();

        cout << curr << " ";

        for (int i = 0; i < v; i++, c++)

        {

            c++;

            if (adj[curr][i] == 1)

            {

                c++;

                c++;

                indegree[i]--;

                c++;

                if (indegree[i] == 0)

                {

                    c++;

                    q.push(i);

                }

            }

        }

    }

    cout << "\n";

    cout << "the complexity is: " << c;

    return 0;

}

Text

Description automatically generated

**LAB 5 – GREEDY II**

1. Dijkstra’s Algorithm

#include <iostream>

#include <climits>

using namespace std;

int counter = 0;

int minDistance(int shortest[], bool visited[], int V)

{

    int minIndex = -1;

    counter++;

    for (int i = 0; i < V; i++)

    {

        counter++;

        if (minIndex == -1)

        {

            if (!visited[i])

            {

                minIndex = i;

                counter++;

            }

            counter++;

        }

        else if (!visited[i] && shortest[i] < shortest[minIndex])

        {

            minIndex = i;

            counter++;

        }

        counter++;

    }

    counter++;

    counter++;

    return minIndex;

}

void dijkstra(int a[][100], int V, int source)

{

    bool visited[100];

    counter++;

    int shortest[100], temp = V, u;

    counter++;

    for (int i = 0; i < V; i++)

    {

        counter++;

        visited[i] = false;

        counter++;

    }

    counter++;

    for (int i = 0; i < V; i++)

    {

        counter++;

        shortest[i] = INT\_MAX;

        counter++;

    }

    counter++;

    shortest[source] = 0;

    counter++;

    while (temp != 0)

    {

        counter++;

        u = minDistance(shortest, visited, V);

        counter++;

        for (int i = 0; i < V; i++)

        {

            counter++;

            if (a[u][i] != -1 && !visited[i])

            {

                if (shortest[u] + a[u][i] < shortest[i])

                {

                    shortest[i] = shortest[u] + a[u][i];

                    counter++;

                }

                counter++;

            }

            counter++;

        }

        counter++;

        visited[u] = true;

        counter++;

        temp--;

        counter++;

    }

    cout << "\nThe shortest distance from source to nodes are" << endl;

    cout << "Vertex Distance" << endl;

    for (int i = 0; i < V; i++)

        cout << (i + 1) << " " << shortest[i] << endl;

    cout << "\n\nNumber of steps : " << counter;

}

int main()

{

    int v, source;

    cout << "\nEnter the number of vertices" << endl;

    cin >> v;

    int a[100][100];

    cout << "\nEnter the adjacency matrix (-1 if not adjacent)" << endl;

    for (int i = 0; i < v; i++)

    {

        for (int j = 0; j < v; j++)

        {

            cin >> a[i][j];

        }

    }

    cout << "\nEnter the source" << endl;

    cin >> source;

    dijkstra(a, v, source - 1);

    counter++;

    return 0;

}

Text

Description automatically generated

2. Prim’s algorithm

#include <bits/stdc++.h>

using namespace std;

int c;

int findMinVertex(int \*distance, bool \*visited, int v)

{

    c++;

    int minVal = INT\_MAX, minIndex;

    for (int i = 0; i < v; i++, c++)

    {

        c++;

        if (!visited[i] && distance[i] < minVal)

        {

            c++;

            minIndex = i;

            c++;

            minVal = distance[i];

        }

    }

    c++;

    return minIndex;

}

int main()

{

    int v, e;

    cout << "enter the vertices and edges: \n";

    cin >> v >> e;

    cout << endl;

    int \*\*adj = new int \*[v];

    for (int i = 0; i < v; i++)

        adj[i] = new int[v];

    for (int i = 0; i < v; i++, c++)

        for (int j = 0; j < v; j++, c++)

        {

            c++;

            adj[i][j] = 0;

        }

    cout << "enter the elements and the weights";

    for (int i = 0; i < e; i++, c++)

    {

        int l, r, w;

        cin >> l >> r >> w;

        c++;

        adj[l][r] = w;

        c++;

        adj[r][l] = w;

    }

    bool \*visited = new bool[v];

    for (int i = 0; i < v; i++, c++)

    {

        c++;

        visited[i] = false;

    }

    int \*distance = new int[v];

    int \*parent = new int[v];

    for (int i = 1; i < v; i++, c++)

    {

        c++;

        distance[i] = INT\_MAX;

    }

    c++;

    distance[0] = 0;

    for (int i = 0; i < v - 1; i++, c++)

    {

        c++;

        int curr = findMinVertex(distance, visited, v);

        c++;

        visited[curr] = true;

        for (int j = 0; j < v; j++, c++)

        {

            c++;

            if (!visited[j] && adj[curr][j] != 0)

            {

                c++;

                if (distance[j] > adj[curr][j])

                {

                    c++;

                    distance[j] = adj[curr][j];

                    c++;

                    parent[j] = curr;

                }

            }

        }

    }

    for (int i = 1; i < v; i++)

        cout << min(parent[i], i) << " -> " << max(parent[i], i) << " : " << distance[i] << "\n";

    cout << "\n";

    cout << "the complecity is: " << c;

    return 0;

}