Linear search

#include <stdio.h>

int search(int arr[], int n, int x)

{

    int i;

    for (i = 0; i < n; i++)

        if (arr[i] == x)

            return i;

    return -1;

}

int main(void)

{

    int arr[] = { 2, 3, 4, 10, 40 };

    int x = 10;

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

    int result = search(arr, n, x);

    (result == -1)

        printf("Element is not present in array")

        printf("Element is present at index %d", result);

    return 0;

}

Binary search

#include <stdio.h>

int binarySearch(int arr[], int l, int r, int x)

{

    if (r >= l) {

        int mid = l + (r - l) / 2;

        if (arr[mid] == x)

            return mid;

        if (arr[mid] > x)

            return binarySearch(arr, l, mid - 1, x);

        return binarySearch(arr, mid + 1, r, x);

    }

    return -1;

}

int main(void)

{

    int arr[] = { 2, 3, 4, 10, 40 };

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

    int x = 10;

    int result = binarySearch(arr, 0, n - 1, x);

    (result == -1) ? printf("Element is not present in array")

                   : printf("Element is present at index %d",

                            result);

    return 0;

}

Quick sort

#include <iostream.h>

using namespace std;

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

{

    int t = \*a;

    \*a = \*b;

    \*b = t;

}

int partition (int arr[], int low, int high)

{

    int pivot = arr[high]; // pivot

    int i = (low - 1); // Index of smaller element and indicates the right position of pivot found so far

    for (int j = low; j <= high - 1; j++)

    {

        // If current element is smaller than the pivot

        if (arr[j] < pivot)

        {

            i++; // increment index of smaller element

            swap(&arr[i], &arr[j]);

        }

    }

    swap(&arr[i + 1], &arr[high]);

    return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

    if (low < high)

    {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

void printArray(int arr[], int size)

{

    int i;

    for (i = 0; i < size; i++)

        cout << arr[i] << " ";

    cout << endl;

}

int main()

{

    int arr[] = {10, 7, 8, 9, 1, 5};

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

    quickSort(arr, 0, n - 1);

    cout << "Sorted array: \n";

    printArray(arr, n);

    return 0;

}

Merge sort

#include <stdio.h>

#include <stdlib.h>

void merge(int arr[], int l, int m, int r)

{

    int i, j, k;

    int n1 = m - l + 1;

    int n2 = r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)

        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)

        R[j] = arr[m + 1 + j];

    i = 0;

    j = 0;

    k = l;

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        }

        else {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

while (j < n2) {

        arr[k] = R[j];

        j++;

        k++;

    }

}

void mergeSort(int arr[], int l, int r)

{

    if (l < r) {

        int m = l + (r - l) / 2;

        // Sort first and second halves

        mergeSort(arr, l, m);

        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);

    }

}

void printArray(int A[], int size)

{

    int i;

    for (i = 0; i < size; i++)

        printf("%d ", A[i]);

printf("\n");

}

int main()

{

    int arr[] = { 12, 11, 13, 5, 6, 7 };

    int arr\_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");

    printArray(arr, arr\_size);

    mergeSort(arr, 0, arr\_size - 1);

    printf("\nSorted array is \n");

    printArray(arr, arr\_size);

    return 0;

}

Insertion sort

#include <math.h>

#include <stdio.h>

void insertionSort(int arr[], int n)

{

    int i, key, j;

    for (i = 1; i < n; i++) {

        key = arr[i];

        j = i - 1;

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}

void printArray(int arr[], int n)

{

    int i;

    for (i = 0; i < n; i++)

        printf("%d ", arr[i]);

    printf("\n");

}

int main()

{

    int arr[] = { 12, 11, 13, 5, 6 };

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

    insertionSort(arr, n);

    printArray(arr, n);

    return 0;

}

BFS

#include<iostream.h>

#include <list.h>

using namespace std;

class Graph

{

    int V;

    list<int> \*adj;

public:

    Graph(int V);

    void addEdge(int v, int w);

    void BFS(int s);

};

Graph::Graph(int V)

{

    this->V = V;

    adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w);

}

void Graph::BFS(int s)

{

    bool \*visited = new bool[V];

    for(int i = 0; i < V; i++)

        visited[i] = false;

list<int> queue;

    visited[s] = true;

    queue.push\_back(s);

    list<int>::iterator i;

    while(!queue.empty())

    {

        s = queue.front();

        cout << s << " ";

        queue.pop\_front();

        for (i = adj[s].begin(); i != adj[s].end(); ++i)

        {

            if (!visited[\*i])

            {

                visited[\*i] = true;

                queue.push\_back(\*i);

            }

        }

    }

}

int main()

{

    Graph g(4);

    g.addEdge(0, 1);

    g.addEdge(0, 2);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

 g.addEdge(2, 3);

    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "

         << "(starting from vertex 2) \n";

    g.BFS(2);

    return 0;

}

DFS

 #include <bits/stdc++.h>

using namespace std;

class Graph

{

public:

    map<int, bool> visited;

    map<int, list<int>> adj;

    void addEdge(int v, int w);

    void DFS(int v);

};

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w);.

}

void Graph::DFS(int v)

{

    visited[v] = true;

    cout << v << " ";

    list<int>::iterator i;

    for (i = adj[v].begin(); i != adj[v].end(); ++i)

        if (!visited[\*i])

            DFS(\*i);

}

int main()

{

        Graph g;

    g.addEdge(0, 1);

    g.addEdge(0, 9);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

    g.addEdge(2, 3);

    g.addEdge(9, 3);

    cout << "Following is Depth First Traversal"

            " (starting from vertex 2) \n";

    g.DFS(2);

    return 0;

}

Prims

#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

#define V 5

int minKey(int key[], bool mstSet[])

{

    int min = INT\_MAX, min\_index;

    for (int v = 0; v < V; v++)

        if (mstSet[v] == false && key[v] < min)

            min = key[v], min\_index = v;

    return min\_index;

}

int printMST(int parent[], int graph[V][V])

{

    printf("Edge \tWeight\n");

    for (int i = 1; i < V; i++)

        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);

}

void primMST(int graph[V][V])

{

    int parent[V];

    int key[V];

      bool mstSet[V];

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

        key[i] = INT\_MAX, mstSet[i] = false;

    key[0] = 0;

    parent[0] = -1;

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

        int u = minKey(key, mstSet);

        mstSet[u] = true;

        for (int v = 0; v < V; v++)

            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

                parent[v] = u, key[v] = graph[u][v];

    }

    printMST(parent, graph);

}

int main()

{

    int graph[V][V] = { { 0, 2, 0, 6, 0 },

                        { 2, 0, 3, 8, 5 },

                        { 0, 3, 0, 0, 7 },

                        { 6, 8, 0, 0, 9 },

                        { 0, 5, 7, 9, 0 } };

    primMST(graph);

    return 0;

}

Dijkstra’s

#include <limits.h>

#include <stdio.h>

int minDistance(int dist[], bool sptSet[])

{

    int min = INT\_MAX, min\_index;

    for (int v = 0; v < V; v++)

        if (sptSet[v] == false && dist[v] <= min)

            min = dist[v], min\_index = v;

    return min\_index;

}

array

void printSolution(int dist[])

{

    printf("Vertex \t\t Distance from Source\n");

    for (int i = 0; i < V; i++)

        printf("%d \t\t %d\n", i, dist[i]);

}

void dijkstra(int graph[V][V], int src)

{

    int dist[V];

    bool sptSet[V]; for (int i = 0; i < V; i++)

        dist[i] = INT\_MAX, sptSet[i] = false;

    dist[src] = 0;

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

        int u = minDistance(dist, sptSet);

        sptSet[u] = true;

vertex.

        for (int v = 0; v < V; v++)

from

            u is

                        if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

                && dist[u] + graph[u][v] < dist[v])

                dist[v] = dist[u] + graph[u][v];

    }

    printSolution(dist);

}

int main()

{

        int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },

                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },

                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },

                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },

                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },

                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },

                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },

                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);

    return 0;

}

LCS

#include<bits/stdc++.h>

int max(int a, int b);

int lcs( char \*X, char \*Y, int m, int n )

{

   if (m == 0 || n == 0)

     return 0;

   if (X[m-1] == Y[n-1])

     return 1 + lcs(X, Y, m-1, n-1);

   else

     return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));

}

int max(int a, int b)

{

    return (a > b)? a : b;

}

int main()

{

  char X[] = "AGGTAB";

  char Y[] = "GXTXAYB";

  int m = strlen(X);

  int n = strlen(Y);

  printf("Length of LCS is %d", lcs( X, Y, m, n ) );

  return 0;

}

Knapsack fractional and 0/1 both

#include <bits/stdc++.h>

using namespace std;

struct Item {

    int value, weight;

    Item(int value, int weight)

    {

       this->value=value;

       this->weight=weight;

    }

};

bool cmp(struct Item a, struct Item b)

{

    double r1 = (double)a.value / (double)a.weight;

    double r2 = (double)b.value / (double)b.weight;

    return r1 > r2;

}

double fractionalKnapsack(int W, struct Item arr[], int n)

{

    sort(arr, arr + n, cmp);

    for (int i = 0; i < n; i++)

    {

        cout << arr[i].value << "  " << arr[i].weight << " :

    "

             << ((double)arr[i].value / arr[i].weight) <<

    endl;

}

    int curWeight = 0;

    double finalvalue = 0.0;

    for (int i = 0; i < n; i++) {

        if (curWeight + arr[i].weight <= W) {

            curWeight += arr[i].weight;

            finalvalue += arr[i].value;

        }

        else {

            int remain = W - curWeight;

            finalvalue += arr[i].value

                          \* ((double)remain

                             / (double)arr[i].weight);

            break;

        }

    }

    return finalvalue;

}

int main()

{

    int W = 50;

    Item arr[] = { { 60, 10 }, { 100, 20 }, { 120, 30 } };

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

    cout << "Maximum value we can obtain = "

         << fractionalKnapsack(W, arr, n);

    return 0;

}