|  |  |
| --- | --- |
| Name: Varun Mahendra Khadayate | Roll No: A016 |
| Subject: DAA | Date of Submission: 18-10-2021 |

# Practical 1 – Insertion Sort

## Code

#include <stdio.h>

int insertionSort(int size, int \*array);

int main()

{

int size, i , array[21];

printf("Enter total number of elements: ");

scanf("%d", &size);

printf("Enter %d elements: ", size);

for(i=0; i<size; i++)

scanf("%d", &array[i]);

insertionSort(size, array);

printf("The Sorted Array is :: ");

for(i=0; i<size; i++)

printf(" %d", array[i]);

printf("\n");

return 0;

}

int insertionSort(int size, int \*array)

{

int i, j;

int temp;

for(i=1; i<size; i++){

temp=array[i];

j= i-1;

while((temp < array[j])&&(j >= 0))

{

array[j+1] = array[j];

j= j-1;

}

array[j+1] = temp;

}

return 1;

}

## Output

Text

Description automatically generated

# Practical 2 – Merge Sort

## Code

#include <stdio.h>

void merge(int arr[], int start, int mid, int end)

{

int len1 = mid - start + 1;

int len2 = end - mid;

int leftArr[len1], rightArr[len2];

for (int i = 0; i < len1; i++)

{

leftArr[i] = arr[start + i];

}

for (int j = 0; j < len2; j++)

{

rightArr[j] = arr[mid + 1 + j];

}

int i, j, k;

i = 0;

j = 0;

k = start;

while (i < len1 && j < len2) {

if (leftArr[i] <= rightArr[j])

{

arr[k] = leftArr[i];

i++;

}

else

{

arr[k] = rightArr[j];

j++;

}

k++;

}

while (i < len1)

{

arr[k] = leftArr[i];

i++;

k++;

}

while (j < len2) {

arr[k] = rightArr[j];

j++;

k++;

}

}

void mergeSort(int arr[], int start, int end) {

if (start < end)

{

int mid = start + (end - start) / 2;

mergeSort(arr, start, mid);

mergeSort(arr, mid + 1, end);

merge(arr, start, mid, end);

}

}

void display(int arr[], int size) {

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

printf("%d ", arr[i]);

printf("\n");

}

int main()

{

int arr[] = {6, 5, 12, 10, 9, 1};

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

printf("Original array::\n");

display(arr, size);

mergeSort(arr, 0, size - 1);

printf("Sorted array after implementing Merge Sort is::\n");

display(arr, size);

}

## Output

Graphical user interface, text, application

Description automatically generated

# Practical 3 – Quick Sort

## Code

#include<stdio.h>

int main()

{

int arr[20], n, i;

printf("Enter the size of the array:: \n");

scanf("%d", &n);

printf("Enter the elements to be sorted of size %d::\n",n);

for(i = 0; i < n; i++)

scanf("%d", &arr[i]);

quicksort(arr, 0, n-1);

printf("\n\nSorted array after implementing Quick sort are as follows ::\n");

for(i = 0; i < n; i++)

printf("%d ", arr[i]);

return 0;

}

void quicksort(int \*arr, int low, int high)

{

int pivot, i, j, temp;

if(low < high) {

pivot = low;

i = low;

j = high;

while(i < j) {

while(arr[i] <= arr[pivot] && i <= high)

i++;

while(arr[j] > arr[pivot] && j >= low)

j--;

if(i < j) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

temp = arr[j];

arr[j] = arr[pivot];

arr[pivot] = temp;

quicksort(arr, low, j-1);

quicksort(arr, j+1, high);

}

}

## Output

Text

Description automatically generated

# Practical 4 – Knapsack

Fractional

## Code

#include <stdio.h>

#include <stdlib.h>

typedef struct Item

{

int itemId;

int weight;

int profit;

float PBW;

float xi;

} Item;

void margeRev(Item \*arr, int low, int mid, int high)

{

int i = low;

int j = mid + 1;

int c = 0;

int temp\_Arr\_size = high - low + 1;

Item \*tempArr = (Item \*)malloc(sizeof(arr[0]) \* temp\_Arr\_size);

while (i <= mid && j <= high)

if (arr[i].PBW > arr[j].PBW)

tempArr[c++] = arr[i++];

else

tempArr[c++] = arr[j++];

while (i <= mid)

tempArr[c++] = arr[i++];

while (j <= high)

tempArr[c++] = arr[j++];

for (i = low, c = 0; i <= high; i++)

arr[i] = tempArr[c++];

}

void margeSortRev(Item \*arr, int low, int high)

{

if (low < high)

{

int mid = (high + low) / 2;

margeSortRev(arr, low, mid);

margeSortRev(arr, mid + 1, high);

margeRev(arr, low, mid, high);

}

}

float fractionalknapsack(Item \*items, int n, int capacity)

{

margeSortRev(items, 0, n - 1);

float profit = 0;

int i = 0;

int takenWeight = 0;

for (i = 0; i < n; i++)

{

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

{

profit += items[i].profit;

takenWeight += items[i].weight;

items[i].xi = 1;

}

else

{

items[i].xi = ((float)capacity - takenWeight) / items[i].weight;

takenWeight += items[i].xi \* items[i].weight;

profit += items[i].xi \* items[i].profit;

break;

}

}

return profit;

}

int main()

{

int n, i, knapsackCapacity;

printf("\tKNAPSACK USING GREEDY APPROACH\n");

printf("\nEnter total number of items :: ");

scanf("%d", &n);

Item \*items = (Item \*)malloc(sizeof(Item) \* n);

for (i = 0; i < n; i++)

{

printf("\nEnter itemId, weight, profit for %d :: ",i+1);

scanf("%d %d %d", &items[i].itemId, &items[i].weight, &items[i].profit);

items[i].PBW = (float)items[i].profit / items[i].weight;

}

printf("\n\nEnter the Knapsack Capacity :: ");

scanf("%d", &knapsackCapacity);

float totalProfit = fractionalknapsack(items, n, knapsackCapacity);

printf("\n\nTotal Profit is :: %f", totalProfit);

return 0;

}

## Output

Text

Description automatically generated

0-1

## Code

## Output

# Practical 5 – Dijkstra’s Algorithm

## Code

#include<stdio.h>

#include<conio.h>

#include<time.h>

#define INFINITY 9999

#define MAX 10

void ALGO(int G[MAX][MAX],int n,int startnode);

int main()

{

int G[MAX][MAX],i,j,n,u;

printf("\tDIJKSTRA ALGORITHM!!!\n");

printf("\nEnter the total no. of nodes :: ");

scanf("%d",&n);

printf("\nEnter the adjacency matrix ::\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%d",&G[i][j]);

}

}

printf("\nEnter the starting node :: ");

scanf("%d",&u);

printf("\n\n\tTHE DIJKSTRA ALGORTIHM IS LOADING...");

delay(8);

printf("\n\n\tLoaded 100%%");

delay(3);

printf("\n\nThe answers for individual nodes along with their respective path are as follows :: \n");

delay(1);

ALGO(G,n,u);

printf("\n\n\n");

return 0;

}

void delay(int second)

{

int milsec = 1000 \* second;

clock\_t startTime = clock();

while(clock() < (startTime + milsec));

}

void ALGO(int G[MAX][MAX],int n,int startnode)

{

int cost[MAX][MAX],distance[MAX],pred[MAX];

int visited[MAX],count,mindistance,nextnode,i,j;

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

if(G[i][j]==0)

cost[i][j]=INFINITY;

else

cost[i][j]=G[i][j];

}

for(i=0;i<n;i++)

{

distance[i]=cost[startnode][i];

pred[i]=startnode;

visited[i]=0;

}

distance[startnode]=0;

visited[startnode]=1;

count=1;

while(count<n-1)

{

mindistance=INFINITY;

for(i=0;i<n;i++)

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

{

mindistance=distance[i];

nextnode=i;

}

visited[nextnode]=1;

for(i=0;i<n;i++)

if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i])

{

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode;

}

count++;

}

for(i=0;i<n;i++)

if(i!=startnode)

{

printf("\n\nDistance of node [%d] :: %d",i,distance[i]);

printf("\nAnd the Path for the same is :: %d",i);

j=i;

do

{

j=pred[j];

printf(" <- %d",j);

}

while(j!=startnode);

}

}

## Output

Text

Description automatically generated

# Practical 6 – Kruskal’s Algorithm

## Code

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

void delay(int);

int main()

{

printf("\n\tKRUSKAL ALGORITHM\n");

printf("\nEnter the total no. of nodes :: ");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=999;

}

}

printf("\n\n\tTHE KRUSKAL ALGORTIHM IS LOADING...");

printf("\n\n\tLoaded 100%%");

printf("\n\nThe edges of Minimum Cost Spanning Tree are\n");

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

{

for(j=1;j <= n;j++)

{

if(cost[i][j] < min)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u);

v=find(v);

if(uni(u,v))

{

printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

cost[a][b]=cost[b][a]=999;

}

printf("\n\tMinimum cost = %d\n",mincost);

return 0;

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

return 1;

}

return 0;

}

## Output

Text

Description automatically generated

# Practical 7 – Tree Traversal

## Code

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct node

{

struct node \*left;

int data;

struct node \*right;

};

struct node \*tree=NULL;

struct node\* insertelement(struct node \*tree,int n)

{

struct node \*newnode,\*nodeptr,\*parentptr;

newnode=(struct node \*)malloc(sizeof(struct node));

newnode->data=n;

newnode->left=NULL;

newnode->right=NULL;

if(tree==NULL)

{

tree=newnode;

}

else

{

parentptr=NULL;

nodeptr=tree;

while(nodeptr!=NULL)

{

parentptr=nodeptr;

if(n<nodeptr->data)

nodeptr=nodeptr->left;

else

nodeptr = nodeptr->right;

}

if(n<parentptr->data)

parentptr->left=newnode;

else

parentptr->right=newnode;

}

return tree;

}

int inorder(struct node \*tree)

{

while(tree!=NULL)

{

inorder(tree->left);

printf("\t%d",tree->data);

inorder(tree->right);

return tree;

}

}

int postorder(struct node \*tree)

{

while(tree!=NULL)

{

preorder(tree->left);

preorder(tree->right);

printf("\t%d",tree->data);

return tree;

}

}

int preorder(struct node \*tree)

{

while(tree!=NULL)

{

printf("\t%d",tree->data);

preorder(tree->left);

preorder(tree->right);

return tree;

}

}

int main()

{

int n,ch;

printf("\tBinary Search Tree Traversal\n");

printf("\n\tMenu:\n");

printf("1.Insert Elements\n2.In-Order\n3.Pre-Order\n4.Post-Order\n5.Exit");

while(ch!=5)

{

printf("\n\nEnter your Choice :: ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nEnter the Element :: ");

scanf("%d",&n);

tree=insertelement(tree,n);

break;

case 2: printf("\tThe In-Order sequence for the given string is as follows::\n");

n=inorder(tree);

break;

case 3: printf("\tThe Pre-Order sequence for the given string is as follows::\n");

n=preorder(tree);

break;

case 4: printf("\tThe Post-Order sequence for the given string is as follows::\n");

n=postorder(tree);

break;

case 5: printf("\tSure Boss Exiting!!\n\tGood Bye\n\n");

break;

}

}

return 0;

}

## Output

Text

Description automatically generated Text

Description automatically generated

# Practical 8 – Floyd Warshall

## Code

#include<stdio.h>

#include<conio.h>

void floyd(int cost[10][10],int n)

{

int i,j,k,t,x[10][10];

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

x[i][j]=cost[i][j];

}

}

for(k=0;k<n;k++)

{

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

if((x[i][k]==1000) || (x[k][j]==1000))

t=1000;

else

t=x[i][k]+x[k][j];

x[i][j]=(x[i][j]>t?t:x[i][j]);

}

}

}

printf("\n\n\tThe Final answer after implementing Floyd-Warshall Algorithm is:\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

printf("%d ",x[i][j]);

}

printf("\n");

}

}

int main()

{

int i,j,n,a[10][10];

printf("\tFloyd-Warshall's Algorithm!!");

printf("\nEnter the no. of vertices: ");

scanf("%d",&n);

printf("\nEnter the Adjacency matrix:\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%d",&a[i][j]);

}

}

floyd(a,n);

getch();

return 0;

}

## Output

Text

Description automatically generated