

Design and Analysis of Algorithms

3rd Semester

Batch of 2021-2025

Practical File

SUBMITTED BY- ARYAN PRAKASH

ROLL NO – 12112076

BRANCH- Computer Engineering

Submitted To-

Bhavna Chaudhary Ma’am

1.Implement Insertion Sort

Code:

#include <iostream>

using namespace std;

void insertionsort(int *arr*[], int *n*)

{

    int key, j;

    for (int i = 1; i < *n*; i++)

    {

        key = *arr*[i];

        j = i - 1;

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

        {

*arr*[j + 1] = *arr*[j];

            j--;

        }

*arr*[j+1]=key;

    }

}

int main()

{

    int n;

    cout << "Enter size of Array\n";

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    insertionsort(arr,n);

    // for printing Array

    for (int i = 0; i < n; i++)

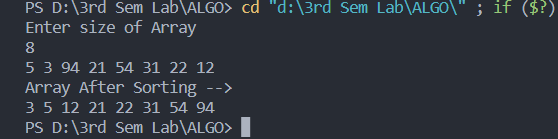
    {

        cout << arr[i] << " ";­­­

    }

}

Output:-



2.Implement Selection Sort

Code

#include <iostream>

using namespace std;

void swap(int \**x*, int \**y*)

{

    int temp = \**x*;

    \**x* = \**y*;

    \**y* = temp;

}

void selectionsort(int *arr*[], int *n*)

{

    for (int i = 0; i < *n* - 1; i++)

    {

        int min\_idx = i;

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

        {

            if (*arr*[j] < *arr*[min\_idx])

                min\_idx = j ;

        }

        if (*arr*[min\_idx] != *arr*[i])

        {

            swap(&*arr*[min\_idx], &*arr*[i]);

        }

    }

}

int main()

{

    int n;

    cout << "Enter size of Array\n";

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    selectionsort(arr, n);

    // printing array

    for (int i = 0; i < n; i++)

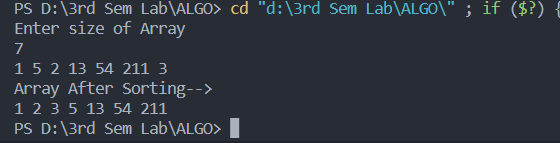
    {

        cout << arr[i] << " ";

    }

}

Output:-



3.Implement Bubble Sort

Code

#include<iostream>

using namespace std;

void swap(int \**x*,int \**y*){

    int temp=\**x*;

    \**x*=\**y*;

    \**y*=temp;

}

 void bubblesort(int *arr*[],int *n*){

    for (int i = 0; i < *n*-1; i++)

    {

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

        {

            if(*arr*[i]>*arr*[j])

            swap(&*arr*[i],&*arr*[j]);

        }

    }

 }

int main(){

    int n;

    cout<<"Enter Size  of Array\n";

    cin>>n;

    int arr[n];

    //array input

    for (int  i = 0; i < n; i++)

    {

        cin>>arr[i];

    }

    bubblesort(arr,n);

    cout<<"Array After Sorting\n";

    for (int i = 0; i < n; i++)

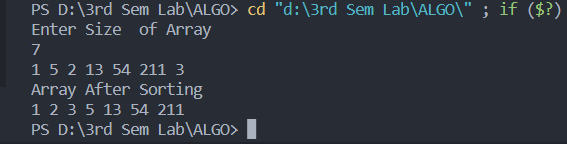
    {

        cout<<arr[i]<<" ";

    }

}

Output:-



4.Implement HeapSort

Code

 #include <iostream>

using namespace std;

// swap function

void swap(int \**x*, int \**y*)

{

    int temp = \**x*;

    \**x* = \**y*;

    \**y* = temp;

    return;

}

// heapify function

void heapify(int *arr*[], int *i*, int *n*)

{

    if (2 \* *i* <= *n* && (*arr*[*i*] < *arr*[2 \* *i*] && *arr*[2 \* *i*] >= *arr*[2 \* *i* + 1]))

    {

        swap(&*arr*[*i*], &*arr*[2 \* *i*]);

        heapify(*arr*, 2 \* *i*, *n*);

    }

    else if (2 \* *i* + 1 <= *n* && (*arr*[*i*] < *arr*[2 \* *i* + 1]))

    {

        swap(&*arr*[*i*], &*arr*[2 \* *i* + 1]);

        heapify(*arr*, 2 \* *i* + 1, *n*);

    }

    else

        return;

}

// max heap function

void build\_max\_heap(int *arr*[], int *n*)

{

    for (int i = *n* / 2; i >= 1; i--)

    {

        heapify(*arr*, i, *n*);

    }

}

// heapsort function

void heapsort(int *arr*[], int *n*)

{

    if (*n* == 1)

        return;

    else

    {

        build\_max\_heap(*arr*, *n*);

        swap(&*arr*[1], &*arr*[*n*]);

*n*--;

        heapify(*arr*, *n*, 1);

        heapsort(*arr*, *n*);

    }

}

int main()

{

    int n;

    cout << "Enter size of Array\n";

    cin >> n;

    int arr[n + 1];

    cout << "Enter Elements of Array\n";

    for (int i = 1; i <= n; i++)

    {

        cin >> arr[i];

    }

    // heap sorting

    heapsort(arr, n);

    cout<<"Array After Sorting\n";

    for (int i = 1; i <= n; i++)

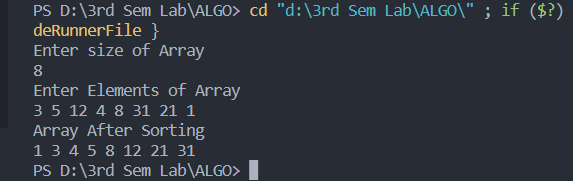
    {

        cout << arr[i] << " ";

    }

}

Output:-



5.Implement MergeSort

Code

#include <iostream>

using namespace std;

void merge(int *arr*[], int *lb*, int *mid*, int *ub*)

{

    int i = *lb*;

    int j = *mid* + 1;

    int k = 0;

    int t[*ub*-*lb*+1];// tempory array to store operation in array

    while (i <= *mid* && j <= *ub*)

    {

        if (*arr*[i] <= *arr*[j])

        {

            t[k] = *arr*[i];

            i++;

            k++;

        }

        else

        {

            t[k] = *arr*[j];

            j++;

            k++;

        }

    }

    if (i > *mid*)

    {

        while (j <= *ub*)

        {

            t[k] = *arr*[j];

            j++;

            k++;

        }

   }

    else

    {

        while (i <= *mid*)

        {

            t[k] = *arr*[i];

            i++;

            k++;

        }

    }

    int p=0;    //copy down the whole array in old array

    for (int x = *lb*; x <= *ub*; x++)

    {

*arr*[x] = t[p];

        p++;

    }

}

void mergesort(int *arr*[],int *lb*,int *ub*){

    if(*lb*<*ub*){

        int mid=(*lb*+*ub*)/2;

        mergesort(*arr*,*lb*,mid);

        mergesort(*arr*,mid+1,*ub*);

        merge(*arr*,*lb*,mid,*ub*);

    }

}

int main()

{

    int n;

    cout<<"Enter size of array\n";

    cin>>n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cin>>arr[i];

    }

    mergesort(arr,0,n-1);

    cout<<"Array after Sorting-->\n";

    for (int i = 0; i < n; i++)

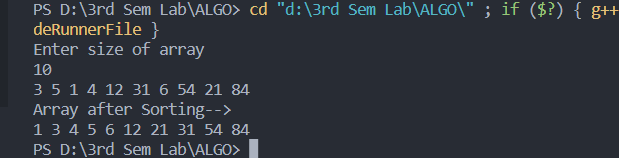
    {

        cout<<arr[i]<<" ";

    }

}

Output:-



6.Implement QuickSort

Code

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

int partition(int *arr*[], int *lb*, int *ub*)

{

    int pivot = *arr*[*lb*];

    int start = *lb*;

    int end = *ub*;

    while (start < end)

    {

        while (*arr*[start] <= pivot)

        {

            start++;

        }

        while (*arr*[end] > pivot)

        {

            end--;

        }

        if (start < end)

        {

            swap(*arr*[start], *arr*[end]);

        }

    }

    swap(*arr*[*lb*], *arr*[end]);

    return end;

}

int quicksort(int *arr*[], int *lb*, int *ub*)

{

    if (*lb* < *ub*)

    {

        int loc = partition(*arr*, *lb*, *ub*);

        quicksort(*arr*, *lb*, loc - 1);

        quicksort(*arr*, loc + 1, *ub*);

    }

}

int main()

{

    int n;

    cout << "Enter size of Array\n";

    cin >> n;

    int arr[n];

    cout << "Enter elements of Array\n";

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    quicksort(arr, 0, n - 1);

    cout<<"Array After Sorting-->\n";

    for (int i = 0; i < n; i++)

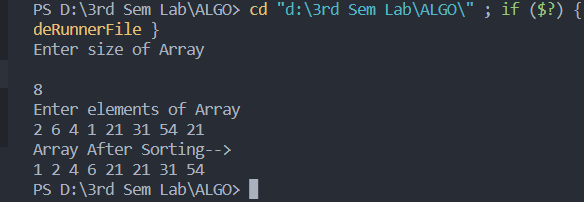
    {

        cout << arr[i] << " ";

    }

}

Output:-



7.Implement Counting Sort

Code

#include <iostream>

#include <math.h>

using namespace std;

void countingSort(int *arr*[], int *n*)

{

    int max = *arr*[0];

    for (int i = 1; i < *n*; i++)

    {

        if (*arr*[i] > max)

            max = *arr*[i];

    }

    int output[*n*];      // to store the sorted array

    int count[max + 1]; // to count the frequency of element

    for (int i = 0; i <= max; i++)

    {

        count[i] = 0; // initialiase

    }

    for (int i = 0; i < *n*; i++)

    {

        count[*arr*[i]]++;

    }

    // cumilative sum

    for (int i = 1; i <= max; i++)

    {

        count[i] += count[i - 1];

    }

    // placing ele at correct position in array

    for (int i = *n*-1; i >=0; i--)

    {

        output[--count[*arr*[i]]] = *arr*[i];

    }

    for (int i = 0; i < *n*; i++)

    {

*arr*[i] = output[i];

    }

}

void print(int *arr*[], int *n*)

{

    for (int i = 0; i < *n*; i++)

        cout << *arr*[i] << " ";

        cout<<endl;

}

void test()

{

    int arr[] = {192, 214, 5, 21, 4, 5, 74, 732, 235, 52};

    cout<<"Array before Sorting is-->\n";

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

    print(arr,n);

    countingSort(arr, n);

    cout<<"Array after Sorting-->\n";

    print(arr, n);

}

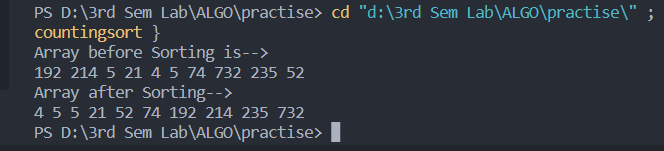
int main()

{

    test();

}

Output:-



8.Implement Radix Sort

Code

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

int max(int *arr*[], int *n*)

{

    int max = INT\_MIN;

    for (int i = 0; i < *n*; i++)

    {

        if (*arr*[i] > max)

        {

            max = *arr*[i];

        }

    }

    return max;

}

void countsort(int *arr*[], int *n*, int *pos*)

{

    int output[*n*];

    int count[10];

    for (int i = 0; i < *n*; i++)

    {

        count[i] = 0;

    }

    for (int i = 0; i < *n*; i++)

    {

        count[(*arr*[i] / *pos*) % 10]++;

    }

    for (int i = 0; i < max(*arr*, *n*); i++)

    {

        count[i] += count[i - 1];

    }

    for (int i = *n* - 1; i >= 0; i--)

    {

        output[--count[((*arr*[i] / *pos*) % 10)]-1] = *arr*[i];

    }

    for (int i = 0; i < *n*; i++)

    {

*arr*[i] = output[i];

    }

}

void radixsort(int *arr*[], int *n*)

{

    int maxs = max(*arr*, *n*);

    for (int i = 1; maxs / i > 0; i \*= 10)

    {

        countsort(*arr*, *n*, i);

    }

}

int main()

{

    int n;

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    cout<<"Array Before sorting\n";

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

    radixsort(arr, n);

    cout<<"Array After sorting\n";

    for (int i = 0; i < n; i++)

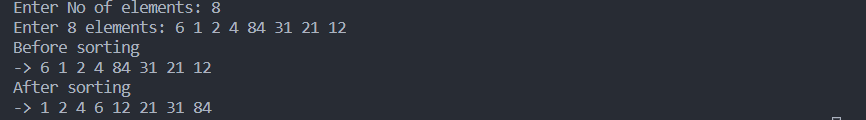
    {

        cout << arr[i] << " ";

    }

}

Output:-



9.Implement Infix to Postfix

Code

#include<stdio.h>

#include<stdbool.h>

#include<stdlib.h>

#include<limits.h>

struct Stack {

    char s[100];

    int top;

};

typedef struct Stack stack;

stack push(stack *st*, char *ch*){

*st*.s[++(*st*.top)]=*ch*;

return *st*;

}

char pop(stack *st*){

    char r=*st*.s[*st*.top];

    return r;

}

// precedence

int pre(char *c*){

    if(*c*=='/' || *c*=='\*'){

        return 4;

    }

    if(*c*=='+' || *c*=='-'){

        return 2;

    }

    if(*c*=='$'){

        return 3;

    }

    return INT\_MIN;

}

int main(){

    char str[100];

    printf("Enter an expression of maximum 100 characters : \n");

    scanf("%s",str);

    stack s1;

    s1.top=-1;

    char output[100];

    int curr=-1;

    for(int i=0;str[i]!='\0';i++){

        if((str[i]>='a' && str[i]<='z')|| (str[i]>='A' && str[i]<='Z')||(str[i]>='1' && str[i]<='9')){

            output[++curr]=str[i];

        }else{

            if(s1.top==-1){             // operator stack is empty

                s1=push(s1,str[i]);

            }

           else if(str[i]==')'){        // founded closing parenthesis

               char ch=pop(s1);

                s1.top--;

                while(ch!='(' && s1.top>-1){

                    output[++curr]=ch;

                     ch=pop(s1);

                     s1.top--;

                }

                ch=pop(s1);

            }

            else{

                    int check=s1.top;

                while(pre(s1.s[check])>pre(str[i]) && s1.top>-1){

                    char ch1=pop(s1);

                    if(ch1!='('){

                  output[++curr]=pop(s1);

                    }

                    s1.top--;

                  check--;

                }

            s1=push(s1,str[i]);

            }

        }

    }

    while(s1.top>=-1){

        output[++curr]=pop(s1);

        s1.top--;

    }

printf("Given postfix notation is : \n");

    printf("%s\n",output);

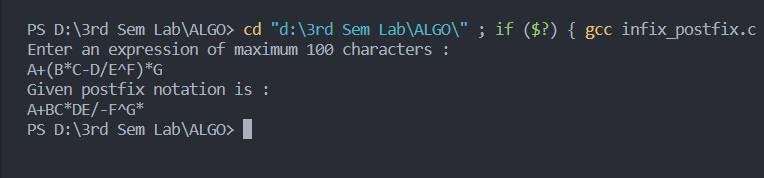
}

/\*

A+(B\*C-D/E^F)\*G

\*/

Output:-



10.Implement Postfix Evaluation

Code

#include<stdio.h>

int stack[30];

int top=-1;

void push(int *x*){

    stack[++top]=*x*;

}

int pop(){

    if(top==-1){

        return -1;

    }

    else

    return stack[top--];

}

int main(){

    char str[30];

    char \*e;

    printf("Enter Expression\n");

    scanf("%s",&str);

    e=str;

    while(\*e!='\0'){

        if((int)\*e>=48 && (int)\*e<=57){

            push((int)(\*e)-48);

        }

        else if((int)(\*e)==43){

            int a=pop();

            int b=pop();

            push(a+b);

        }

        else if((int)(\*e)==42){

            int a=pop();

            int b=pop();

            push(a\*b);

        }

        else if((int)(\*e)==45){

            int a=pop();

            int b=pop();

            push(b-a);

        }

        else if((int)(\*e)==47){

            int a=pop();

            int b=pop();

            push(b/a);

        }

        e++;

    }

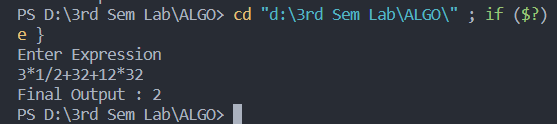
    printf("Final Output : ");

    printf("%d",pop());

    return 0;

}

Output-



11.Implement Binary Search Tree

Code

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

struct node

{

    int data;

    struct node \*left, \*right;

};

struct node \*search(struct node \**root*, int *key*)

{

    // if root is null or key is present at root node

    if (*root* == NULL || *root*->data == *key*)

    {

        return *root*;

    }

    if (*key* < *root*->data)

    {

        return search(*root*->left, *key*);

    }

    else

        return search(*root*->right, *key*);

}

struct node \*newnode(int *value*)

{

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

    temp->left = temp->right = NULL;

    temp->data = *value*;

    return temp;

}

struct node \*insert(struct node \**node*, int *value*)

{

    if (*node* == NULL)

    {

        return newnode(*value*);

    }

    else if (*node*->data > *value*)

    {

*node*->left = insert(*node*->left, *value*);

    }

    else

*node*->right = insert(*node*->right, *value*);

    return *node*;

}

void inorder(struct node \**root*)

{

    if (*root* != NULL)

    {

        inorder(*root*->left);

        cout<<*root*->data<<" ";

        inorder(*root*->right);

    }

}

struct node \*minvalue(struct node \**node*)

{

    struct node \*current = *node*->left;

    while (current && current->left != NULL)

    {

        current = current->left;

    }

    return current;

}

struct node \*deletenode(struct node \**node*, int *key*)

{

    if (*node* == NULL)

        return *node*;

    else if (*node*->data > *key*)

    {

*node*->left = deletenode(*node*->left, *key*);

    }

    else if (*node*->data < *key*)

*node*->right = deletenode(*node*->right, *key*);

    else

    { // no left child maybe right child

        if (*node*->left == NULL)

        {

            struct node \*temp = *node*->right;

            free(*node*);

            return temp;

        } // left child may exist

        else if (*node*->right = NULL)

        {

            struct node \*temp = *node*->left;

            free(*node*);

            return *node*;

        }

        struct node \*temp = minvalue(*node*->right);

*node*->data = temp->data;

*node*->right = deletenode(*node*->right, temp->data);

    }

}

int main()

{

    struct node\* root=NULL;

    root=insert(root,10);

    root=insert(root,35);

    root=insert(root,73);

    root=insert(root,11);

    root=insert(root,112);

    inorder(root);

}

Output:-



# **Graph Code**

12.Implement Graph By Matrix

Code

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define N 20

// taking input

void read\_graph(int *n*, int *graph*[N][N])

{

    // making inital graph zero

    for (int i = 0; i < *n*; i++)

    {

        for (int j = 0; j < *n*; j++)

        {

*graph*[i][j] = 0;

        }

    }

    // now inserting values in graph

    //  arr[i][j]==1 represent a edge

    printf("Enter -1 to stop entering edges in graph\n");

    for (int i = 1; i < *n*; i++)

    {

        int j;

        printf("Enter Adjacent of vertex %d :", i);

        {

            do

            {

                scanf("%d", &j);

                if (j == -1)

                    break;

                else

*graph*[i][j] = 1;

            } while (1);

        }

    }

}

int count\_no\_edges(int *n*, int *arr*[N][N])

{

    int count = 0;

    for (int i = 1; i < *n*; i++)

    {

        for (int j = 1; j < *n*; j++)

        {

            if (*arr*[i][j] == 1)

                count++;

        }

        return count;

    }

}

void Display\_graph(int *n*, int *graph*[N][N])

{

    for (int i = 1; i < *n*; i++)

    {

        printf("Vertex %d ->:", i);

        for (int j = 1; j < *n*; j++)

        {

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

        }

        printf("\n");

    }

}

int vertex\_having\_highest\_outgoing\_edge(int *graph*[N][N], int *n*)

{

    int max = 0, count = 0;

    for (int i = 0; i < *n*; i++)

    {

        count = 0;

        for (int j = 0; j < *n*; j++)

        {

            if (*graph*[i][j] == 1)

                count++;

        }

        max = count > max ? count : max;

    }

    return max;

}

int vertex\_having\_no\_OutgoingEdge(int *graph*[N][N], int *n*)

{

    int count=0;

    for (int i = 1; i < *n*; i++)

    {   count=0;

        for (int j = 1; j < *n*; j++)

        {

            if(*graph*[i][j]==1)

            count++;

        }

        if(count<=1)

        return i;

    }

}

int main()

{

    int n;

    printf("Enter Number of Vertices of Graph\n");

    scanf("%d", &n);

    n++;

    int Adjmat[N][N];

    // taking input of Adjmat

    read\_graph(n, Adjmat);

    Display\_graph(n, Adjmat);

    printf("%d \n", vertex\_having\_highest\_outgoing\_edge(Adjmat, n));

    printf("%d \n",vertex\_having\_no\_OutgoingEdge(Adjmat,n));

}

//

/\*

3 1 2 -1

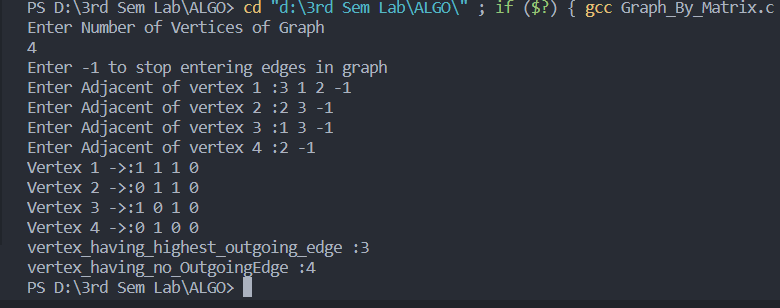
2 3 -1

1 3 -1

2 -1

\*/

Output:-



13.Graph by linked list

Code

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define N 20

#define MAX 20

int stack[100];

int top = -1;

int push(int *x*)

{

    top++;

    stack[top] = *x*;

}

int pop()

{

    return stack[top--];

}

typedef struct queuee{

    int size;

    int top;

    int \*arr;

}queuee;

queuee\* queue\_create(int *size*){

    queuee\* q=(queuee\*)calloc(1,sizeof(queuee));

    q->arr=(int\*)calloc(*size*,sizeof(int));

    // q->size=0;

    // q->top=0;

    return q;

}

void popq(queuee\* *q*){

    (*q*->top)++;

    (*q*->top)=(*q*->top)%100;

    (*q*->size)--;

}

void pushq(queuee\* *q*,int *a*){

    (*q*->size)++;

    (*q*->arr)[((*q*->top)+(*q*->size)-1)%100]=*a*;

}

int isemptyq(queuee\* *q*){

    return ((*q*->size)==0?1:0);

}

int frontq(queuee\* *q*){

    if(isemptyq(*q*))

        return INT\_MIN;

    return (*q*->arr)[(*q*->top)];

}

typedef struct gnode

{

    int vertex;

    struct gnode \*next;

} gnode;

// int visited[MAX];

int parent[MAX] = {-1};

void DFS\_visit(gnode \**graph*[], int *i*, int *trav\_status*[])

{

*trav\_status*[*i*] = 1;

    printf("%d ", *i*);

    gnode \*temp = *graph*[*i*];

    while (temp != NULL)

    {

        int adj = temp->vertex;

        if (*trav\_status*[adj] == 0)

        {

            parent[adj] = *i*;

            DFS\_visit(*graph*, adj, *trav\_status*);

        }

        temp = temp->next;

    }

}

void DFS(gnode \**graph*[], int *n*)

{

    int trav\_status[*n*];

    for (int i = 1; i < *n*; i++)

    {

        trav\_status[i] = 0;

    }

    for (int i = 1; i < *n*; i++)

    {

        if (trav\_status[i] == 0)

        {

            DFS\_visit(*graph*, i, trav\_status);

        }

    }

}

void DFS\_toposort\_visit(gnode \**graph*[], int *i*, int *trav\_status*[])

{

*trav\_status*[*i*] = 1;

    gnode \*temp = *graph*[*i*];

    int adj;

    while (temp != NULL)

    {

        adj = temp->vertex;

        if (*trav\_status*[adj] == 0)

        {

            parent[temp->vertex] = *i*;

            DFS\_toposort\_visit(*graph*, adj, *trav\_status*);

        }

        temp = temp->next;

    }

    push(*i*);

}

void DFS\_toposort(gnode \**graph*[], int *n*)

{

    int trav\_status[*n*];

    for (int i = 1; i < *n*; i++)

    {

        trav\_status[i] = 0;

    }

    for (int i = 1; i < *n*; i++)

    {

        DFS\_toposort\_visit(*graph*, i, trav\_status);

    }

}

void BFS\_visit(gnode\* *graph*[],int *parent*[],int *trav\_status*[],int *i*,queuee\* *q*){

    // printf("\*\*%d\n",empty(q));

    pushq(*q*,*i*);

    // printf("%d\n",front(q));

*trav\_status*[*i*]=1;

*parent*[*i*]=-1;;

    while(!isemptyq(*q*)){

        int a=frontq(*q*);

        printf("%d ",a);

        popq(*q*);

        gnode\* temp=*graph*[a];

        while(temp!=NULL){

            if(*trav\_status*[temp->vertex]==0){

*parent*[temp->vertex]=a;

                pushq(*q*,temp->vertex);

*trav\_status*[temp->vertex]=1;

            }

            temp=temp->next;

        }

    }

}

void BFS(gnode\* *graph*[],int *n*){

    int trav\_status[*n*];

    int parent[*n*];

    struct queuee \*q=queue\_create(50);

    for(int i=1;i<*n*;i++){

        trav\_status[i]=0;

        parent[i]=-1;

    }

    for(int i=1;i<*n*;i++){

        if(trav\_status[i]==0){

            BFS\_visit(*graph*,parent,trav\_status,i,q);

        }

    }

}

void insert(gnode \**graph*[], int *i*, int *adj*)

{

    gnode \*temp = (gnode \*)malloc(sizeof(gnode));

    temp->vertex = *adj*;

    temp->next = NULL;

    if (*graph* == NULL)

    {

*graph*[*i*] = temp;

    }

    else

    {

        temp->next = *graph*[*i*];

*graph*[*i*] = temp;

    }

}

void read\_vertices\_at\_once(gnode \**graph*[], int *n*)

{

    for (int i = 1; i < *n*; i++)

    {

*graph*[i] = 0;

    }

    int adj;

    for (int i = 1; i < *n*; i++)

    {

        printf("Enter Adjacent of vertex %d :", i);

        do

        {

            scanf("%d", &adj);

            if (adj == -1)

                break;

            insert(*graph*, i, adj);

        } while (1);

    }

}

void print\_edge(gnode \**graph*[], int *n*)

{

    for (int i = 1; i < *n*; i++)

    {

        printf("Adjacent of Vertex %d --> ", i);

        gnode \*temp = *graph*[i];

        while (temp != NULL)

        {

            printf("%d ", temp->vertex);

            temp = temp->next;

        }

    printf("\n");

    }

}

void read\_adjanceny\_list(gnode \**graph*[], int *n*)

{

    int val;

    printf("Enter 1 if their exist an edge between vertex else 0\n");

    for (int i = 1; i < *n*; i++)

    {

        printf("Enter Adjacent vertex %d :\n", i);

        for (int j = 1; j < *n*; j++)

        {

            scanf("%d", val);

            if (val == 1)

                insert(*graph*, i, val);

        }

    }

}

int count\_adjacentof\_vertex(gnode \**graph*[], int *vertex*)

{

    int count = 0;

    gnode \*temp;

    temp = *graph*[*vertex*];

    while (temp != NULL)

    {

        temp = temp->next;

        count++;

    }

    return count;

}

int vertex\_having\_max\_edge(gnode \**list*[], int *n*)

{

    int max = INT\_MIN;

    int ver;

    int count;

    for (int i = 1; i < *n*; i++)

    {

        count = count\_adjacentof\_vertex(*list*, i);

        if (count > max)

        {

            max = count;

            ver = i;

        }

    }

    return ver;

}

int highest\_outgoing\_edge(gnode \**graph*[], int *n*)

{

    int count = 0;

    int max = 0;

    int vertex = 0;

    for (int i = 1; i < *n*; i++)

    {

        gnode \*temp = *graph*[i];

        while (temp != NULL)

        {

            count++;

            temp = temp->next;

        }

        if (count > max)

        {

            max = count;

            vertex = i;

        }

    }

    return vertex;

}

int highest\_incoming\_edge(gnode \**graph*[], int *n*)

{

    int vertex[*n*];

    int ver = 0;

    int max = INT\_MIN;

    for (int i = 0; i < *n*; i++)

    {

        vertex[i] = 0;

    }

    for (int i = 0; i < *n*; i++)

    {

        gnode \*temp = *graph*[i];

        while (temp != NULL)

        {

            int adj = temp->vertex;

            vertex[adj]++;

            temp = temp->next;

        }

    }

    for (int i = 1; i < *n*; i++)

    {

        if (max < vertex[i])

        {

            max = vertex[i];

            ver = i;

        }

        return i;

    }

}

int main()

{

    int n;

    printf("Enter the Number of vertex\n");

    scanf("%d", &n);

    n++;

    gnode \*graph[n];

    read\_vertices\_at\_once(graph, n);

    printf("DFS topo traversal of graph is\n");

    DFS\_toposort(graph, n);

    while (top != -1)

    {

        printf("%d ", pop());

    }

    printf("\n\n");

    printf("DFS visit of Graph is :->\n");

    DFS(graph, n);

    printf("\n\n");

    print\_edge(graph, n);

    printf("Breadth First Search :-->");

    BFS(graph,n);

    printf("\n\n");

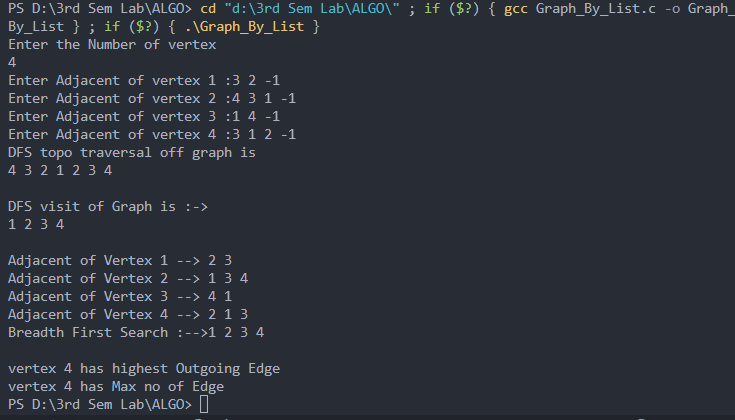
    printf("vertex %d has highest Outgoing Edge\n", highest\_outgoing\_edge(graph, n));

    printf("vertex %d has Max no of Edge\n", highest\_outgoing\_edge(graph, n));

    printf("Vertex %d has highest Incoming Edges\n", highest\_incoming\_edge(graph, n));

}

Output:



Code

#include<stdio.h>

#include<stdlib.h>

#include<limits.h>

#include<stdbool.h>

typedef struct pq\_node{

    int vertex;

    int key;

}pq\_node;

typedef struct edge\_node{

    int vertex;

    int wt;

    struct edge\_node\* next;

}edge\_node;

typedef struct graph\_node

{

    int parent;

    int ind\_in\_pq;

    bool in\_pq;

    struct edge\_node\* adj\_list;

}graph\_node;

void insert\_edge(graph\_node *graph*[],int *i*,int *adj*,int *weight*){

    edge\_node\* temp=(edge\_node\*)malloc(sizeof(edge\_node));

    temp->vertex=*adj*;

    temp->wt=*weight*;

    temp->next=NULL;

    if(*graph*[*i*].adj\_list==NULL){

*graph*[*i*].adj\_list=temp;

    }

    else{

        temp->next=*graph*[*i*].adj\_list;

*graph*[*i*].adj\_list=temp;

    }

}

void read\_graph(graph\_node *graph*[],int *n*){

    for(int i=1;i<=*n*;i++){

*graph*[i].adj\_list=NULL;

    }

    printf("Enter -1 to stop entering adjacents of asked vertex\n");

    for(int i=1;i<=*n*;i++){

        printf("Enter adjacents of vertex %d and wt of this edge: ",i);

        int adj,wt;

        do{

            scanf("%d",&adj);

            if(adj==-1)

                break;

            scanf("%d",&wt);

            insert\_edge(*graph*,i,adj,wt);

        }while(1);

    }

}

void vertex\_adjacents(graph\_node *graph*[],int *n*){

    for(int i=1;i<*n*;i++){

        printf("Vertex : %d -> ",i);

        edge\_node\* temp=*graph*[i].adj\_list;

        while(temp!=NULL){

            printf("%d(%d) ",temp->vertex,temp->wt);

            temp=temp->next;

        }

        printf("\n");

    }

}

void swap(graph\_node *graph*[],pq\_node \**a*,pq\_node \**b*){

    int temp1=*graph*[*a*->vertex].ind\_in\_pq;

*graph*[*a*->vertex].ind\_in\_pq=*graph*[*b*->vertex].ind\_in\_pq;

*graph*[*b*->vertex].ind\_in\_pq=temp1;

    pq\_node temp=\**a*;

    \**a*=\**b*;\**b*=temp;

}

void min\_heapify(graph\_node *graph*[],pq\_node *pq*[],int *i*,int *n*){

    int min=*i*;

    int left=2\**i*;

    int right=2\**i*+1;

    if(left<=*n*&&*pq*[min].key>*pq*[left].key)

        min=left;

    if(right<=*n*&&*pq*[min].key>*pq*[right].key)

        min=right;

    if(min!=*i*){

        swap(*graph*,&*pq*[min],&*pq*[*i*]);

        min\_heapify(*graph*,*pq*,min,*n*);

    }

}

void buildminheap(graph\_node *graph*[],pq\_node *pq*[],int *n*){

    for(int i=*n*/2;i>0;i--){

        min\_heapify(*graph*,*pq*,i,*n*);

    }

}

pq\_node extract\_min(graph\_node *graph*[],pq\_node *pq*[],int\* *n*){

    pq\_node ans=*pq*[1];

*pq*[1]=*pq*[\**n*];

*graph*[*pq*[1].vertex].ind\_in\_pq=1; // critical mistake

    (\**n*)--;

    min\_heapify(*graph*,*pq*,1,\**n*);

    return ans;

}

void decrease\_key(graph\_node *graph*[],pq\_node *pq*[],int *i*,int *n*,int *new\_wt*){

*pq*[*i*].key=*new\_wt*;

    while(*i*>1 && *pq*[*i*/2].key>*new\_wt*){

        swap(*graph*,&*pq*[*i*],&*pq*[*i*/2]);

*i*/=2;

    }

}

void PRIM\_MST(graph\_node *graph*[],pq\_node *pq*[],int *n*,int *source*){

    // graph\_node ans\_set[n];

    int mst\_sum=0;

    for(int i=1;i<=*n*;i++){

*graph*[i].parent=-1;

*graph*[i].ind\_in\_pq=i;

*graph*[i].in\_pq=true;

*pq*[i].key=INT\_MAX;

*pq*[i].vertex=i;

    }

*pq*[*source*].key=0;

    buildminheap(*graph*,*pq*,*n*);

    while(*n*>0){

        pq\_node ans=extract\_min(*graph*,*pq*,&*n*);

    // printf("okk\n");

        int u=ans.vertex;

*graph*[u].in\_pq=false;

        // min weight of the edge

        mst\_sum+=ans.key;

        // u's adjacency list

        edge\_node\* temp=*graph*[u].adj\_list;

        while(temp){

            int v=temp->vertex;

            int wt=temp->wt;

            int index=*graph*[v].ind\_in\_pq;

            if(*graph*[v].in\_pq==true && *pq*[index].key>wt){

                printf("%d %d\n",*pq*[index].key,wt);

                // pq[index].key=wt; use decrease function( what is this non - sense)

                decrease\_key(*graph*,*pq*,index,*n*,wt);

*graph*[v].parent=u;

            }

            temp=temp->next;

        }

    }

    printf("MST having minimum total weight = %d.\n",mst\_sum);

}

int main(){

    int n;

    printf("Total No of vertices?: ");

    scanf("%d",&n);

    graph\_node graph[n+1];

    pq\_node pq[n+1];

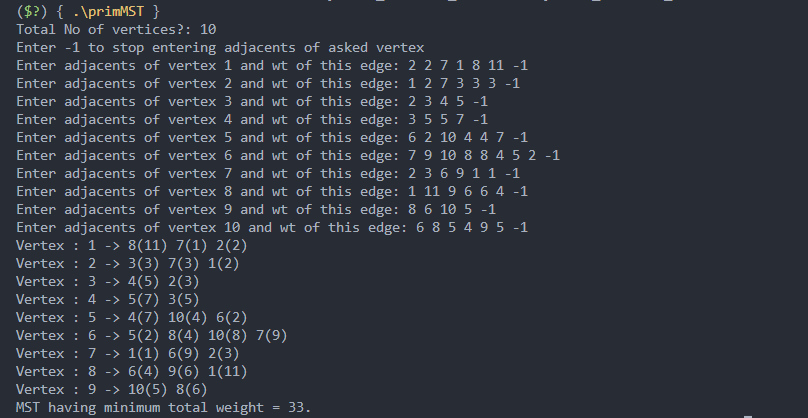
    read\_graph(graph,n);

    vertex\_adjacents(graph,n);

    PRIM\_MST(graph,pq,n,1);

}

Output:-



15.Implement Bellman Ford

Code

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <limits.h>

typedef struct EdgeNode

{

    int vertex;

    int weight;

    struct EdgeNode \*next;

} EdgeNode;

typedef struct vertexNode

{

    int distance;

    int predecessor;

    EdgeNode \*adjptr;

} vertexNode;

int ans[1000]; // global answer array for storing the results

bool bellmanFord(vertexNode \**graph*[], int *vertices*, int *sv*)

{

    for (int i = 1; i <= *vertices*; i++) // initializing the distance which is already there in vertex node

    {

*graph*[i]->distance = INT\_MAX;

    }

*graph*[*sv*]->distance = 0; // making the distance of source vertex as 0

    for (int i = 1; i < *vertices*; i++)  // v-1 times loop

    {

        for (int i = 1; i <= *vertices*; i++)

        {

            EdgeNode \*temp = *graph*[i]->adjptr;

            while (temp != NULL)

            {

                if (*graph*[temp->vertex]->distance > *graph*[i]->distance + temp->weight)

                {

*graph*[temp->vertex]->distance = *graph*[i]->distance + temp->weight;

*graph*[temp->vertex]->predecessor = i;

                }

                temp = temp->next;

            }

        }

    }

    // checking for negative cycle

    for (int i = 1; i <= *vertices*; i++) // again  times loop

    {

        EdgeNode \*temp = *graph*[i]->adjptr;

        while (temp != NULL)

        {

            if (*graph*[temp->vertex]->distance > *graph*[i]->distance + temp->weight)

            {

                printf("\nNegative Cycle exists in the graph \n");

                return false;

            }

            temp = temp->next;

        }

    }

    return true;

}

int main()

{

    // input graph;

    int vertices, e;

    printf("Enter the number of vertices: \n");

    scanf("%d%d", &vertices, &e);

    vertexNode \*graph[vertices + 1];

    for (int i = 1; i <= vertices; i++)

    {

        graph[i] = (vertexNode \*)malloc(sizeof(vertexNode));

        graph[i]->adjptr = NULL;

        graph[i]->predecessor = -1;

    }

    printf("Enter the all possible edge\n");

    int a, b, wt;

    for (int i = 0; i < e; i++)

    {

        scanf("%d%d%d", &a, &b, &wt);

        EdgeNode \*newnode = malloc(sizeof(EdgeNode));

        newnode->next = graph[a]->adjptr;

        newnode->vertex = b;

        newnode->weight = wt;

        graph[a]->adjptr = newnode;

    }

    if (bellmanFord(graph, vertices, 1))    // taking 1 as the source vertices

    {

             printf("\nShortest distance is  :\n");

        for (int i = 1; i <= vertices; i++)

        {

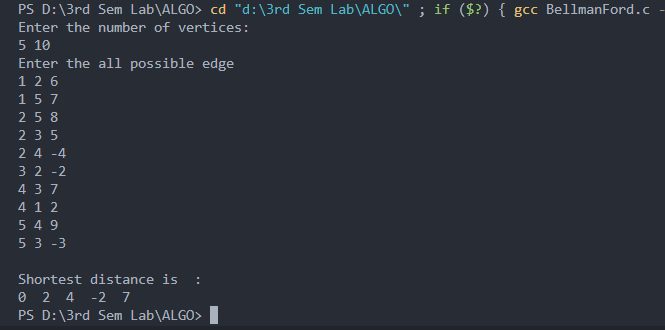
            printf("%d  ", graph[i]->distance);

        }

    }

}

Output:-



Dynamic Programming Code

1.Implement Rod Cut Problem

Code

#include<stdio.h>

int max(int *a*,int *b*){

    return (*a*>*b*?*a*:*b*);

}

int rod\_cut(int *profit*[],int *result*[],int *n*){

*result*[0]=0;

    for(int i=1;i<=*n*;i++){

        for(int j=1;j<=i;j++){

*result*[i]=max(max(*profit*[i],*result*[i]),*result*[j]+*result*[i-j]);

        }

    }

    return *result*[*n*];

}

int main(){

    int n;

    printf("Enter total size of rod : ");

    scanf("%d",&n);

    int profit[n+1],result[n+1];

    printf("Enter profit of rods of respective sizes : ");

    for(int i=1;i<=n;i++){

        scanf("%d",&profit[i]);

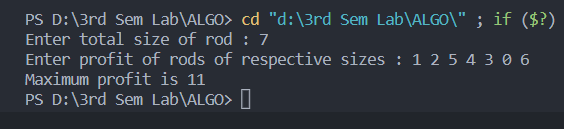
        result[i]=0;

    }

    printf("Maximum profit is %d",rod\_cut(profit,result,n));

}

Output:-



2.Implement Longest Common Substring

Code

#include<stdio.h>

#include<stdlib.h>

void main()

{

    int n,m;

    printf("Enter no. of letters in string 1 and 2 : ");

    scanf("%d%d",&n,&m);

    char s1[n+1],s2[m+1];

    scanf("%s",s1);

    scanf("%s",s2);

    int arr[n+1][m+1];

    for(int i=0;i<=n;i++)

    {

        arr[i][0]=0;

    }

    for(int i=0;i<=m;i++)

    {

        arr[0][i]=0;

    }

    for(int i=1;i<=n;i++)

    {

        for(int j=1;j<=m;j++)

        {

            if(s1[i-1]==s2[j-1])

            {

                arr[i][j]=arr[i-1][j-1]+1;

            }

            else

            {

                arr[i][j]=arr[i-1][j]>arr[i][j-1]?arr[i-1][j]:arr[i][j-1];

            }

        }

    }

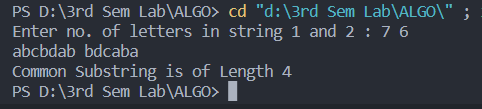
    printf("Common Substring is of Length %d",arr[n][m]);

}

// 7 6 abcbdab bdcaba

// lcs bottom up

Output:-



3. Implement Matrix Chain Multiplication (MCM)

Code

// Bottom Up Approach

#include <stdio.h>

#include <string.h>

#include <math.h>

#define N 3

int R[N+1][N+1];

int S[N][N];

int top=0;

int MCM(int *P*[],int *i*,int *j*)

{

    for (int length = 2; length <= N; length++)

    {

        for (int k = 1; k <= N-length+1; k++)

        {

            int l=k+length-1;

            for (int m = k; m < l; m++)

            {

                int temp=R[k][m]+R[m+1][l]+*P*[k-1]\**P*[m]\**P*[l];

                // printf("%d\n",temp);

                if(R[k][l] > temp)

                {

                    R[k][l]=temp;

                    S[*i*][*j*]=k;

                }

            }

        }

    }

    return R[1][N];

}

int MCMBottomUp(int *P*[],int *i*,int *j*)

{

    for (int k = 0; k < N+1; k++)

    {

        for (int l = 0; l < N+1; l++)

        {

            S[k][l]=0;

            if (l==k)

            {

                R[l][k]=0;

            }

            else

            {

                R[k][l]=100000;

            }

        }

    }

    return MCM(*P*,*i*,*j*);

}

int main()

{

    printf("For %d matrices.\n", N);

    int P[N+1];

    for (int i = 0; i < N+1; i++)

    {

        scanf("%d", P+i);

    }

    printf("Minimum Multiplication is %d.\n", MCMBottomUp(P,1,N));

    return 0;

}

/\* 10 5 100 50 \*/

Output:-

