|  |
| --- |
| **CST 203 LAB** |
| CST 203 Data Structure and Algorithms |
| Data Structure and Algorithms |
|  |
| **GAURAV KABRA (2016 UCP 1471)** |
| **12/11/2017** |

|  |
| --- |
|  |

Table of Contents

[Lab 1 [28 July 2017]](#_Toc498009956) 3

[< Find product of two numbers without multiplication operator.>](#_Toc498009957)

[< Perform matrix multiplication and transpose.>](#_Toc498009958)

[Lab 2 [11 August 2017]](#_Toc498009960) 7

[< Calculate power using recursion. >](#_Toc498009961)

[<Find if a number is palindrome or not.>](#_Toc498009962)

[< To print prime numbers in given range. >](#_Toc498009963)

< Sort an array using any method.>

[Lab 3 [18 August 2017]](#_Toc498009964) 13

[< Encrypt a string as per the given rule.>](#_Toc498009965)

<Apply Merge-Sort >

[<Practical Title 3>](#_Toc498009967)

Lab 4 [8 September 2017] 19

<Apply Selection - Sort.>

<Apply Insertion - Sort.>

< Implement the logic of Quick - Sort.>

Lab 5 [15 September 2017] 24

< Convert an infix expression to prefix expression.>

<Convert an infix expression to postfix expression.>

<Evaluate given postfix expression.>

Lab 6 [22 September 2017] 38

<Tower of Hanoi without recursion.>

<Stack using linked list and reversal of string.>

<Implement tree traversals.>

Lab 7 [27 October 2017] 57

<BFS>

<Find common ancestors of two nodes.>

Lab 8 [3 November 2017] 67

<Implement graph using adjacency list.>

<Minimum distance between given distance.>

Lab 9[10 November 2017] 82

<Find pairs of equal sum.>

<Detection of cycle.>

# Lab 1 [28 July 2017]

## <Find product of two numbers without multiplication operator.>

Code:

#include<stdio.h>

void mul(int num1,int num2)

{

int i,result=0;

for(i=1;i<=num2;i++)

{

result=result+num1;

}

printf("The value of %d x %d is %d\n",num1,num2,result);

}

int main()

{

int num1,num2;

printf("Enter num1\n");

scanf("%d",&num1);

printf("Enter num2\n");

scanf("%d",&num2);

mul(num1,num2);

}

## <Perform matrix multiplication and transpose.>

Code:

#include<stdio.h>

int A[3][3],B[3][3],M[3][3]; // Global Declaration

/\* Get The Matrix \*/

void getMatrix(int arr[3][3])

{

int i,j;

printf("Enter 9 elements for the matrix\n");

for(i=0;i<=2;i++)

{

for(j=0;j<=2;j++)

{

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

}

}

}

/\*Multiply The Matrix \*/

void Mul(int A[3][3],int B[3][3])

{

int i,j,k;

for(i=0;i<=2;i++)

{

for(j=0;j<=2;j++)

{

M[i][j]=0;

for(k=0;k<=2;k++){

M[i][j]=M[i][j]+A[i][k]\*B[k][j];

}

}

}

}

/\* Display Matrix \*/

void display(int arr[3][3]){

int i,j;

printf("The result is :\n");

for(i=0;i<=2;i++){

for(j=0;j<=2;j++){

printf("%2d ",arr[i][j]);

}

printf("\n");

}

printf("\n");

}

/\* Transpose the matrix \*/

void Trans(int arr[3][3]){

int T[3][3],i,j;

for(i=0;i<=2;i++){

for(j=0;j<=2;j++){

T[i][j]=arr[j][i];

}

}

display(T);

}

/\* Driver Block\*/

int main()

{

getMatrix(A);

getMatrix(B);

printf("The multiplication is:\n");

Mul(A,B);

display(M);

printf("The transpose of A is:\n");

Trans(A);

printf("The transpose of B is:\n");

Trans(B);

}

# Lab 2 [11 August 2017]

## <Calculate power using recursion. >

Code:

#include<stdio.h>

int calPow(int n,int k)

{

if(k==0)

return 1;

else if(k==1)

return n;

else

return(n\*calPow(n,k-1));

}

int main()

{

int n,k,p;

printf("Enter the value of n\n");

scanf("%d",&n);

printf("Enter the value of k\n");

scanf("%d",&k);

if(k>0)

p=calPow(n,k);

else if(k<0)

p=calPow(n,-k);

if(k<0)

printf(" %f \n",1/(float)p);

else

printf(" %d \n",p);

}

## <Find if a number is palindrome or not.>

Code:

#include<stdio.h>

#define MAX 20

int main()

{

int digit,arr[MAX]={0},i,result=1;

printf("How many digit is in your number ?\n");

scanf("%d",&digit);

printf("Enter the number\n");

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

{

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

}

for(i=0;i<=digit/2;i++)

{

if(arr[i]==arr[digit-1-i])

{

continue;

}

else

{

result=0;

break;

}

}

if(result==1)

printf("The number is palindrome\n");

else if(result==0)

printf("The number is not palindrome\n");

}

## <To print prime numbers in given range. >

Code:

#include<stdio.h>

#include<stdlib.h>

int main()

{

int n,i,j,arr[30]={0},index,count=0,value;

printf("How many prime numbers you want to get ?\n");

scanf("%d",&n);

if(n<0)

{

printf("Invalid Input\a\n");

exit(1);

}

else

{

for(i=2;count<n;i++)

{

value=1;

for(j=2;j<i;j++)

{

if(i%j==0)

{

value=0;

break;

}

}

if(value==1)

{

arr[count++]=i;

}

}

}

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

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

}

## <Sort an array using any method. >

Code:

#include<stdio.h>

#define MAX 20

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

{

int temp=\*a;

\*a=\*b;

\*b=temp;

return;

}

int main()

{

int arr[MAX]={0},i,n,j;

printf("How many numbers are there?\n");

scanf("%d",&n);

printf("Enter %d numbers\n",n);

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

{

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

}

for(i=0;i<n-1;i++)

{

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

{

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

{

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

}

}

}

printf("The sorted array is:\n\t");

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

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

}

# Lab 3 [18 August 2017]

## <Encrypt a string as per the given rule.>

Code:

//Encryption

#include<stdio.h>

#include<string.h>

#define MAX1 7

#define MAX2 6

char str[MAX1][MAX2];

int main()

{

int i,j,k=0,len;

char stat[50],key[MAX2];

char ch;

printf("Enter the statement\n");

scanf("%[^\n]s",stat);

printf("Enter the key\n");

scanf("%s",key);

len=strlen(key);

for(i=0;i<MAX1;i++)

{

key[i]=toupper(key[i]);

}

i=0;

for(j=0;j<MAX2;j++)

{

str[0][j]=key[i];

i++;

}

for(i=1;i<MAX1;i++)

{

for(j=0;j<MAX2;j++)

{

str[i][j]=stat[k];

k++;

if(k==strlen(stat))

goto label;

}

}

label:

i=0;

for(ch=65;ch<=97;ch++)

{

for(i=0;i<MAX2;i++)

{

if(str[0][i]==ch)

{

for(j=1;j<MAX1;j++)

{

printf("%c",str[j][i]);

}

}

}

}

printf("\n");

}

## <Apply Merge-Sort >

Code:

//Sort half array

#include<stdio.h>

#define MAX 11

int mid=MAX/2;

void SortIt2O(int \*p)

{

int i,j,temp;

for(i=mid+1;i<MAX-1;i++)

{

for(j=i+1;j<MAX;j++)

{

if(\*(p+i)<\*(p+j))

{

temp=\*(p+j);

\*(p+j)=\*(p+i);

\*(p+i)=temp;

}

}

}

}

void SortIt1O(int \*p)

{

int i,j,temp;

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

{

for(j=i+1;j<mid+1;j++)

{

if(\*(p+i)>\*(p+j))

{

temp=\*(p+j);

\*(p+j)=\*(p+i);

\*(p+i)=temp;

}

}

}

}

void SortIt2E(int \*p)

{

int i,j,temp;

for(i=mid;i<MAX-1;i++)

{

for(j=i+1;j<MAX;j++)

{

if(\*(p+i)<\*(p+j))

{

temp=\*(p+j);

\*(p+j)=\*(p+i);

\*(p+i)=temp;

}

}

}

}

void SortIt1E(int \*p)

{

int i,j,temp;

for(i=0;i<mid-1;i++)

{

for(j=i+1;j<mid;j++)

{

if(\*(p+i)>\*(p+j))

{

temp=\*(p+j);

\*(p+j)=\*(p+i);

\*(p+i)=temp;

}

}

}

}

void SortItA(int \*p)

{

int i,j,temp;

for(i=0;i<MAX-1;i++)

{

for(j=i+1;j<MAX;j++)

{

if(\*(p+i)>\*(p+j))

{

temp=\*(p+j);

\*(p+j)=\*(p+i);

\*(p+i)=temp;

}

}

}

if(mid\*2!=MAX)

{

SortIt1O(p);

SortIt2O(p);

}

else

{

SortIt1E(p);

SortIt2E(p);

}

}

int main()

{

int arr[MAX],i;

printf("Enter %d numbers for the array\n",MAX);

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

{

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

}

SortItA(arr);

printf("\n");

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

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

}

# Lab 4 [8 September 2017]

## <Apply Selection-Sort.>

Code:

//Selection Sort

#include<stdio.h>

#define MAX 10

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

{

int temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

int findMin(int arr[],int index)

{

int min,i,ind;

min=arr[index];

ind=index;

for(i=index+1;i<MAX;i++)

{

if(arr[i]<min)

{

min=arr[i];

ind=i;

}

}

return ind;

}

int main()

{

int arr[MAX],i,min,j;

printf("Enter %d numbers\n",MAX);

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

{

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

}

for(i=0;i<MAX-1;i++)

{

for(j=i+1;j<MAX;j++)

{

min=findMin(arr,j);

if(arr[i]>arr[min])

swap(&arr[i],&arr[min]);

}

}

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

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

}

## <Apply Insertion Sort.>

Code:

//Insertion Sort

#include<stdio.h>

#define MAX 4

int main()

{

int arr[MAX],i,temp,j,k,m;

printf("Enter %d numbers\n",MAX);

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

{

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

}

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

{

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

{

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

{

for(k=i;k>j;k--)

{

arr[k]=arr[k-1];

}

arr[j]=arr[i];

}

}

}

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

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

}

## <Apply Quick Sort.>

Code:

#include<stdio.h>

#define MAX 4

int partition(int \*arr, int init, int tot) {

int pivot = \*(arr+tot - 1);

int pIndex = init,i,temp;

for (i = init; i < tot; i++) {

if (\*(arr + i) < pivot) {

temp = \*(arr + i);

\*(arr + i) = \*(arr + pIndex);

\*(arr + pIndex) = temp;

pIndex++;

}

}

temp = \*(arr + pIndex);

\*(arr + pIndex) = \*(arr + tot - 1);

\*(arr + tot - 1) = temp;

return pIndex;

}

void QuickSort(int \*arr, int init, int tot) {

int pIndex ;

if (init < tot) {

pIndex = partition(arr, init, tot);

QuickSort(arr, init, pIndex - 1);

QuickSort(arr, pIndex + 1, tot);

}

else

return;

}

int main() {

int arr[MAX];

int i;

printf("Enter %d numbers\n", MAX);

for (i = 0; i < MAX; i++) {

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

}

QuickSort(arr,0,MAX);

printf("\n");

for (i = 0; i < MAX; i++) {

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

}

}

# Lab 5 [15 September 2017]

## <Convert an infix expression to prefix expression.>

Code:

// Gaurav Kabra

// Infix to Prefix

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#define MAX 50

char stack[MAX];

char infix[MAX];

char output[MAX];

int top;

char \*t;

void initialise()

{

strcpy(infix,"");

strcpy(output,"");

top=-1;

t=output;

}

void push(char ch)

{

top++;

stack[top]=ch;

}

char pop()

{

char item=stack[top];

top--;

return item;

}

int precede(char ch)

{

if('\*'==ch ||'/'==ch)

return 1;

else

return 0;

}

void prefix\_setter()

{

int i,len=strlen(infix);

char stack\_top;

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

{

if(isalpha(infix[i]) || isdigit(infix[i]))

{

while(isalpha(infix[i]) || isdigit(infix[i]))

{

\*(t)=infix[i];

t++;

i++;

}

}

if(')'==infix[i])

{

push(infix[i]);

}

if(infix[i]=='+' || infix[i]=='-' || infix[i]=='\*' || infix[i]=='/')

{

stack\_top=pop();

while(precede(stack\_top)>precede(infix[i]))

{

\*(t)=stack\_top;

t++;

stack\_top=pop();

}

push(stack\_top);

push(infix[i]);

}

if('('==infix[i])

{

stack\_top=pop();

while(')'!=stack\_top)

{

\*(t)=stack\_top;

t++;

stack\_top=pop();

}

}

}

while(-1!=top)

{

stack\_top=pop();

\*(t)=stack\_top;

t++;

}

}

void reverse(char \*r)

{

int i;

char temp;

int l=strlen(r);

for(i=0;i<=l/2;i++)

{

temp=\*(r+i);

\*(r+i)=\*(r+l-1-i);

\*(r+l-1-i)=temp;

}

}

int main()

{

initialise();

printf("Enter the infix expression\n");

scanf("%s",infix);

reverse(infix);

prefix\_setter();

reverse(output);

printf("\n\n%s",output);

}

## <Convert an infix expression to postfix expression.>

Code:

// Gaurav Kabra

// Infix to Postfix

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#define MAX 50

char stack[MAX];

char output[MAX];

char infix[MAX];

int top;

char \*t;

void initialise()

{

top=-1;

strcpy(infix,"");

strcpy(output,"");

t=output;

}

void push(char ch)

{

top++;

stack[top]=ch;

}

char pop()

{

char item=stack[top];

top--;

return item;

}

int precede(char ch)

{

if('\*'==ch ||'/'==ch)

return 1;

else

return 0;

}

void postfix\_setter()

{

int i,len=strlen(infix);

char stack\_top;

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

{

if(isalpha(infix[i])||isdigit(infix[i]))

{

while(isalpha(infix[i])||isdigit(infix[i]))

{

\*(t)=infix[i];

t++;

i++;

}

}

if(infix[i]=='(')

{

push(infix[i]);

}

if(infix[i]=='+' || infix[i]=='-' || infix[i]=='\*' || infix[i]=='/')

{

stack\_top=pop();

while(precede(stack\_top)>precede(infix[i]))

{

\*(t)=stack\_top;

t++;

stack\_top=pop();

}

push(stack\_top);

push(infix[i]);

}

if(infix[i]==')')

{

stack\_top=pop();

while(stack\_top!='(')

{

\*(t)=stack\_top;

t++;

stack\_top=pop();

}

}

}

while(-1!=top)

{

stack\_top=pop();

\*(t)=stack\_top;

t++;

}

}

int main()

{

initialise();

printf("Enter the infix expression\n");

scanf("%s",infix);

postfix\_setter();

printf("\n\n%s",output);

}

## <Evaluate a postfix expression.>

Code:

// Gaurav Kabra

// Infix to Postfix & evaluate it

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#define MAX 50

char stack[MAX],stack2[MAX];

char output[MAX];

char infix[MAX];

float value[MAX];

int top,top2;

char \*t;

void initialise()

{

top=-1;

strcpy(infix,"");

strcpy(output,"");

t=output;

}

void push(char ch)

{

top++;

stack[top]=ch;

}

char pop()

{

char item=stack[top];

top--;

return item;

}

void push2(char ch)

{

top2++;

stack2[top2]=ch;

}

char pop2()

{

char item=stack2[top2];

top2--;

return item;

}

int precede(char ch)

{

if('\*'==ch ||'/'==ch)

return 1;

else

return 0;

}

void postfix\_setter()

{

int i,len=strlen(infix);

char stack\_top;

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

{

if(isalpha(infix[i])||isdigit(infix[i]))

{

while(isalpha(infix[i])||isdigit(infix[i]))

{

\*(t)=infix[i];

t++;

i++;

}

}

if(infix[i]=='(')

{

push(infix[i]);

}

if(infix[i]=='+' || infix[i]=='-' || infix[i]=='\*' || infix[i]=='/')

{

stack\_top=pop();

while(precede(stack\_top)>precede(infix[i]))

{

\*(t)=stack\_top;

t++;

stack\_top=pop();

}

push(stack\_top);

push(infix[i]);

}

if(infix[i]==')')

{

stack\_top=pop();

while(stack\_top!='(')

{

\*(t)=stack\_top;

t++;

stack\_top=pop();

}

}

}

while(-1!=top)

{

stack\_top=pop();

\*(t)=stack\_top;

t++;

}

}

void eval(char output[],float data[])

{

int i=0;

float op2,op1;

if(isalpha(output[i]))

{

value[i]=output[i];

}

if(output[i]=='+' || output[i]=='-' || output[i]=='\*' || output[i]=='/')

{

op2=pop2();

op1=pop2();

if(output[i]=='+') push2(op1+op2);

if(output[i]=='-') push2(op1-op2);

if(output[i]=='\*') push2(op1\*op2);

if(output[i]=='/') push2(op1/op2);

}

printf("The value is :%f",pop2());

}

int main()

{

int i,l;

initialise();

printf("Enter the infix expression\n");

scanf("%s",infix);

postfix\_setter();

printf("\n\n%s",output);

l=strlen(output);

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

{

if(isalpha(output[i]))

{

printf("Enter the value of %c\n",output[i]);

scanf("%f",&value[i]);

}

}

eval(output,value);

# } Lab 6 [22 September 2017]

## <Implement Tower of Hanoi without recursion.>

Code:

// Tower of Hanoi

#include<stdio.h>

#include<math.h>

int A[100],B[100],C[100],topA=-1,topB=-1,topC=-1;

void pushA(int num)

{

topA++;

A[topA]=num;

}

int popA()

{

int num;

if(-1==topA)

return 0;

num=A[topA];

topA--;

return num;

}

void showA()

{

int i;

printf("Tower A====>\t");

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

{

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

}

printf("\n");

}

void pushB(int num)

{

topB++;

B[topB]=num;

}

int popB()

{

int num;

if(-1==topB)

return 0;

num=B[topB];

topB--;

return num;

}

void showB()

{

int i;

printf("Tower B====>\t");

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

{

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

}

printf("\n");

}

void pushC(int num)

{

topC++;

C[topC]=num;

}

int popC()

{

int num;

if(-1==topC)

return 0;

num=C[topC];

topC--;

return num;

}

void showC()

{

int i;

printf("Tower C====>\t");

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

{

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

}

printf("\n");

}

int main()

{

int N\_disk,N\_move,i,num1,num2;

printf("Enter the number of disks\n");

scanf("%d",&N\_disk);

for(i=N\_disk;i>=1;i--)

{

pushA(i);

}

showA();

showB();

showC();

printf("\n");

N\_move=pow(2,N\_disk)-1;

if(0!=N\_disk%2)

{

for(i=1;i<=N\_move;i++)

{

if(i%3==1) // A and C

{

num1=popA();

num2=popC();

if(0==num1)

{

pushA(num2);

}

else if(0==num2)

{

pushC(num1);

}

else if(num1>num2)

{

pushA(num1);

pushA(num2);

}

else if(num1<num2)

{

pushC(num2);

pushC(num1);

}

showA();

showB();

showC();

printf("\n");

}

else if(i%3==2) // A and B

{

num1=popA();

num2=popB();

if(0==num1)

{

pushA(num2);

}

else if(0==num2)

{

pushB(num1);

}

else if(num1>num2)

{

pushA(num1);

pushA(num2);

}

else if(num1<num2)

{

pushB(num2);

pushB(num1);

}

showA();

showB();

showC();

printf("\n");

}

else if(i%3==0) // B and C

{

num1=popB();

num2=popC();

if(0==num1)

{

pushB(num2);

}

else if(0==num2)

{

pushC(num1);

}

else if(num1>num2)

{

pushB(num1);

pushB(num2);

}

else if(num1<num2)

{

pushC(num2);

pushC(num1);

}

showA();

showB();

showC();

printf("\n");

}

}

}

else

{

for(i=1;i<=N\_move;i++)

{

if(i%3==1) // A and B

{

num1=popA();

num2=popB();

if(0==num1)

{

pushA(num2);

}

else if(0==num2)

{

pushB(num1);

}

else if(num1>num2)

{

pushA(num1);

pushA(num2);

}

else if(num1<num2)

{

pushB(num2);

pushB(num1);

}

showA();

showB();

showC();

printf("\n");

}

else if(i%3==2) // A and C

{

num1=popA();

num2=popC();

if(0==num1)

{

pushA(num2);

}

else if(0==num2)

{

pushC(num1);

}

else if(num1>num2)

{

pushA(num1);

pushA(num2);

}

else if(num1<num2)

{

pushC(num2);

pushC(num1);

}

showA();

showB();

showC();

printf("\n");

}

else if(i%3==0) // B and C

{

num1=popB();

num2=popC();

if(0==num1)

{

pushB(num2);

}

else if(0==num2)

{

pushC(num1);

}

else if(num1>num2)

{

pushB(num1);

pushB(num2);

}

else if(num1<num2)

{

pushC(num2);

pushC(num1);

}

showA();

showB();

showC();

printf("\n");

}

}

}

}

## <Implement stack using linked list and do reversal of string.>

Code:

// Stack using Linked List and reversal of string

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

struct node

{

char ch;

struct node \*link;

};

struct node \*root=NULL;

void insert(char item)

{

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

temp->ch=item;

temp->link=NULL;

if(NULL==root)

{

root=temp;

}

else

{

struct node \*p=root;

while(p->link)

{

p=p->link;

}

p->link=temp;

}

}

void pop()

{

struct node \*temp=root;

if(NULL==root->link)

{

root=NULL;

return;

}

root=root->link;

temp->link=NULL;

free(temp);

}

struct node\* reverse()

{

struct node \*t1=NULL,\*t2=NULL;

while(root)

{

t2=root->link;

root->link=t1;

t1=root;

root=t2;

}

return t1;

}

void show()

{

struct node \*temp=root;

while(temp)

{

printf("%c",temp->ch);

temp=temp->link;

}

}

int main()

{

char ch;

int choice;

while(1)

{

printf("Enter the choice :\t1. Push\t2. Pop\t3. Show\t4. Reverse\t5. Exit\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter the string char-by-char , enter digit for exit\n");

while(1)

{

fflush(stdin);

scanf("%c",&ch);

if(isdigit(ch))

break;

insert(ch);

}

break;

case 2:

pop();

pop();

break;

case 3:

printf("Stacked string :\t");

show();

printf("\n");

break;

case 4:

root=reverse();

printf("The reversed string is :\t");

show();

printf("\n");

break;

case 5:

exit(0);

default:

printf("Invalid Key\n");

break;

}

}

}

## <Implement tree treversals.>

Code:

// BST and it's traversal

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*root=NULL;

struct node\* insert(struct node\* root,int num)

{

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

temp->data=num;

temp->left=NULL;

temp->right=NULL;

if(NULL==root)

{

root=temp;

}

else

{

if(root->data>temp->data)

{

root->left=insert(root->left,num);

}

else

{

root->right=insert(root->right,num);

}

}

return root;

}

void preorder(struct node \*root)

{

if(NULL==root) return;

printf("%d ",root->data);

preorder(root->left);

preorder(root->right);

}

void inorder(struct node \*root)

{

if(NULL==root) return;

inorder(root->left);

printf("%d ",root->data);

inorder(root->right);

}

void postorder(struct node \*root)

{

if(NULL==root) return;

postorder(root->left);

postorder(root->right);

printf("%d ",root->data);

}

int main()

{

int ch,num;

while(1)

{

printf("Enter the choice:\t1. Insert\t2. Preorder\t3. Inorder\t4.Postorder\t5. Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter the number\n");

scanf("%d",&num);

root=insert(root,num);

break;

case 2:

printf("Pre-order traveral is:\n");

preorder(root);

printf("\n");

break;

case 3:

printf("In-order traveral is:\n");

inorder(root);

printf("\n");

break;

case 4:

printf("Post-order traveral is:\n");

postorder(root);

printf("\n");

break;

case 5:

exit(0);

default:

printf("Invalid choice");

break;

}

}

}

## Lab 7 [27 October 2017]

## <BFS >

Code:

/\* Also known as LEVEL - ORDER TRAVERSAL \*/

#include<stdio.h>

#include<stdlib.h>

#define MAX 10

int front = -1;

int rear = -1;

int queue[MAX] = { 0 };

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*root = NULL;

struct node\* insert(struct node \*root, int number)

{

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

temp->data = number;

temp->left = NULL;

temp->right = NULL;

if (NULL == root)

{

root = temp;

return;

}

else

{

if (root->data > number)

{

root->left = insert(root->left, number);

}

else

{

root->right = insert(root->right, number);

}

}

return root;

}

void inorder(struct node \*root)

{

if (root)

{

inorder(root->left);

printf(" %d ", root->data);

inorder(root->right);

}

}

void enqueue(int data)

{

if (rear == -1)

front = 0;

rear++;

queue[rear] = data;

}

int is\_there(int number)

{

int i;

for(i=front;i<=rear;i++)

{

if(queue[i]==number)

{

return 1;

}

}

return 0;

}

void BFS(struct node \*root)

{

if (root) {

if(!is\_there(root->data))

enqueue(root->data);

if(root->left){

if(!is\_there(root->left->data))

enqueue(root->left->data);

}

if(root->right){

if(!is\_there(root->right->data))

enqueue(root->right->data);

}

if(root->left){

BFS(root->left);

}

if(root->right){

BFS(root->right);

}

}

}

void look\_up() //dequeue

{

printf(" The elements in BFS are : ");

while (front <=rear)

{

printf(" %d ", queue[front]);

front++;

}

}

int main()

{

int ch, number, dig1, dig2, i, j;

while (1)

{

printf("1. Insert\n2. LookUp In-Order\n3. BFS\n4. Exit\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter the number\n");

scanf("%d", &number);

root = insert(root, number);

break;

case 2:

inorder(root);

printf("\n");

break;

case 3:

BFS(root);

look\_up();

printf("\n");

break;

case 4:

exit(0);

}

}

}

## <Find common ancestors of given nodes. >

Code:

#include<stdio.h>

#include<stdlib.h>

#define MAX 10

int a1[MAX] = { 0 };

int a2[MAX] = { 0 };

int c1 = -1;

int c2 = -1;

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*root = NULL;

struct node\* insert(struct node \*root, int number)

{

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

temp->data = number;

temp->left = NULL;

temp->right = NULL;

if (NULL == root)

{

root = temp;

return;

}

else

{

if (root->data > number)

{

root->left = insert(root->left, number);

}

else

{

root->right = insert(root->right, number);

}

}

return root;

}

void inorder(struct node \*root)

{

if (root)

{

inorder(root->left);

printf(" %d ", root->data);

inorder(root->right);

}

}

void FCA1(struct node\* root,int n) // FCA = Find Common Ancestor

{

if (root)

{

if (root->data == n) {

return;

}

else

{

++c1;

a1[c1] = root->data;

if (root->data > n)

{

FCA1(root->left, n);

}

else

{

FCA1(root->right, n);

}

}

}

}

void FCA2(struct node\* root, int n)

{

if (root)

{

if (root->data == n) {

return;

}

else

{

++c2;

a2[c2] = root->data;

if (root->data > n)

{

FCA2(root->left, n);

}

else

{

FCA2(root->right, n);

}

}

}

}

int main()

{

int ch,number,dig1,dig2,i,j;

while (1)

{

printf("\n1. Insert\n2. LookUp In-Order\n3. Find Common Ancestor(s)\n4. Exit");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter the number\n");

scanf("%d", &number);

root = insert(root,number);

break;

case 2:

inorder(root);

printf("\n");

break;

case 3: // This code matters the most

printf("Enter the first number\n");

scanf("%d", &dig1);

printf("Enter the second number\n");

scanf("%d",&dig2);

FCA1(root,dig1);

printf("Ancestors of %d: ", dig1);

for (i = 0; i <= c1 ; i++)

{

printf(" %d ", a1[i]);

}

printf("\n");

FCA2(root,dig2);

printf("\nAncestors of %d: ", dig2);

for (i = 0; i <= c2 ; i++)

{

printf(" %d ", a2[i]);

}

printf("\n");

printf(" Common ancestors of %d and %d are:\n",dig1,dig2);

for (i = 0; i <= c1; i++)

{

for (j = 0; j <= c2; j++)

{

if (a1[i] == a2[j])

printf(" %d ", a1[i]);

}

}

printf("\n");

break;

case 4:

exit(0);

}

}

}

## Lab 8 [3 November 2017]

## <Implement graph using adjacency list. >

Code:

// adjacency list representation & implementation of directed graph

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct pointers

{

int data;

struct node \*next;

struct pointers \*down;

};

struct pointers \*root=NULL;

void insert(int number)

{

struct pointers \*temp=(struct pointers \*)malloc(sizeof(struct pointers)); // don't forget to amlloc memory !

temp->data=number;

temp->next=NULL;

temp->down=NULL;

if(NULL==root)

root=temp;

else

{

struct pointers \*p=root;

while(p->down)

{

p=p->down;

}

p->down=temp;

}

}

void lookUp()

{

struct pointers \*temp=root;

struct node \*p;

while(temp)

{

p=temp->next;

printf(" %d :: ",temp->data);

while(p)

{

printf("--> %d -->",p->data);

p=p->next;

}

temp=temp->down;

printf("\n");

}

}

int main()

{

int ch,number,count=0,dig1,dig2;

int flag;

struct pointers \*temp;

struct node \*p,\*q;

while(1)

{

printf("\n1. Insertion the numbers\n2. Insert the connections\n3. Deletion\n4. Path Available\n5. lookUp\n6. Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter the number to be inserted\n");

scanf("%d",&number);

insert(number);

break;

case 2:

temp=root;

while(temp)

{

do

{

printf("Enter the connections for %d-->\nPress 0 for exit connections\n",temp->data);

scanf("%d",&number);

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

p->data=number;

p->next=NULL;

q=temp->next;

if(NULL==q)

temp->next=p;

else{

while(q->next)

{

q=q->next;

}

q->next=p;

}

}while(number!=0);

temp=temp->down;

}

break;

case 3:

printf("Enter the first number\n");

scanf("%d",&dig1);

printf("Enter the second number\n");

scanf("%d",&dig2);

temp=root;

while(temp)

{

count=0;

if(temp->data==dig1)

{

p=temp->next;

while(p)

{

q=p;

if(p->data==dig2)

{

if(count==0)

{

temp->next=p->next;

}

else

{

q=p->next;

count++;

}

}

p=p->next;

}

}

temp=temp->down;

}

break;

case 4:

flag=0;

printf("To find path availability\n");

printf("Enter the first number\n");

scanf("%d",&dig1);

printf("Enter the second number\n");

scanf("%d",&dig2);

temp=root;

while(temp)

{

if(temp->data==dig1)

{

p=temp->next;

while(p)

{

if(p->data==dig2)

{

printf(" Yes Available\n");

flag=1;

break;

}

p=p->next;

}

if(flag==1)

break;

if(flag==0){

printf("Not available\n");

break;

}

}

else

{

temp=temp->down;

}

}

break;

case 5:

lookUp();

break;

case 6:

exit(0);

}

}

}

## <Find the minimum distance between given nodes.>

Code:

// Find minimum distance between any two given nodes of a tree

#include<stdio.h>

#include<stdlib.h>

#define MAX 10

int a1[MAX] = { 0 };

int a2[MAX] = { 0 };

int a3[MAX]={0};

int c1 = -1;

int c2 = -1;

int length1,length2,length3;// len1 for dig1, len2 for dig2 and len3 for last common ancestor

int count;

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*root = NULL;

struct node\* insert(struct node \*root, int number)

{

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

temp->data = number;

temp->left = NULL;

temp->right = NULL;

if (NULL == root)

{

root = temp;

return;

}

else

{

if (root->data > number)

{

root->left = insert(root->left, number);

}

else

{

root->right = insert(root->right, number);

}

}

return root;

}

void inorder(struct node \*root)

{

if (root)

{

inorder(root->left);

printf(" %d ", root->data);

inorder(root->right);

}

}

void FCA1(struct node\* root,int n) // FCA = Find Common Ancestor

{

if (root)

{

if (root->data == n) {

return;

}

else

{

++c1;

a1[c1] = root->data;

if (root->data > n)

{

FCA1(root->left, n);

}

else

{

FCA1(root->right, n);

}

}

}

}

void FCA2(struct node\* root, int n)

{

if (root)

{

if (root->data == n) {

return;

}

else

{

++c2;

a2[c2] = root->data;

if (root->data > n)

{

FCA2(root->left, n);

}

else

{

FCA2(root->right, n);

}

}

}

}

void search(struct node \*root,int number)

{

struct node \*temp=root;

if(NULL==temp)

{

printf(" %d not found\a!\n",number);

return 0;

}

if(temp->data>number)

{

count++;

search(root->left,number);

}

else if(temp->data<number)

{

count++;

search(root->right,number);

}

else if(temp->data==number)

{

return;

}

}

int main()

{

int ch,number,dig1,dig2,i,j,last;

while (1)

{

printf("\n1. Insert\n2. LookUp In-Order\n3. Find Common Ancestor(s)\n4. Minimum Distance\n5. Exit");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter the number\n");

scanf("%d", &number);

root = insert(root,number);

break;

case 2:

inorder(root);

printf("\n");

break;

case 3: // This code matters the most

printf("Enter the first number\n");

scanf("%d", &dig1);

printf("Enter the second number\n");

scanf("%d",&dig2);

FCA1(root,dig1);

printf("Ancestors of %d: ", dig1);

for (i = 0; i <= c1 ; i++)

{

printf(" %d ", a1[i]);

}

printf("\n");

FCA2(root,dig2);

printf("\nAncestors of %d: ", dig2);

for (i = 0; i <= c2 ; i++)

{

printf(" %d ", a2[i]);

}

printf("\n");

printf(" Common ancestors of %d and %d are:\n",dig1,dig2);

for (i = 0; i <= c1; i++)

{

for (j = 0; j <= c2; j++)

{

if (a1[i] == a2[j])

printf(" %d ", a1[i]);

}

}

printf("\n");

break;

case 4:

// case 1: both less than root->data

printf("Enter the first number\n");

scanf("%d", &dig1);

printf("Enter the second number\n");

scanf("%d",&dig2);

count=0;

search(root,dig1);

length1=count;

count=0;

search(root,dig2);

length2=count;

FCA1(root,dig1);

FCA2(root,dig2);

for (i = 0; i <= c1; i++)

{

for (j = 0; j <= c2; j++)

{

if (a1[i] == a2[j])

last=a1[i];

}

}

count=0;

search(root,last);

length3=count;

if(dig1<root->data && dig2<root->data || dig1>root->data && dig2>root->data )

printf("The minimum distance between %d and %d is %d\n",dig1,dig2,length1+length2-2\*length3);

else // nodes lie on opp. sides of the root->data

printf("The minimum distance between %d and %d is %d\n",dig1,dig2,length1+length2);

break;

case 5:

exit(0);

}

}

}

## Lab 9 [10 November 2017]

## < Find pairs of equal sum.>

Code:

// Pair whose sum is equal using Hashing

#include<stdio.h>

#include<stdlib.h>

#define MAX 8

#define HF(a,b) (a+b-19)%13

int store[MAX];

struct addition

{

int sum;

int i;

int j;

};

struct addition s1[MAX\*MAX]={0};

struct addition s2[MAX\*MAX]={0};

struct addition H1[MAX\*MAX]={0};

struct addition H2[MAX\*MAX]={0};

int count\_s1=0;

int count\_s2=0;

void HF1()

{

int l;

for(l=0;l<count\_s1;l++)

{

H1[l].sum=HF(s1[l].sum,l);

H1[l].i=s1[l].i;

H1[l].j=s1[l].j;

}

}

void HF2()

{

int l;

for(l=0;l<count\_s2;l++)

{

H2[l].sum=HF(s2[l].sum,l);

H2[l].i=s2[l].i;

H2[l].j=s2[l].j;

}

}

int main()

{

int i,j,l,m;

printf("Enter %d numbers for the array\n",MAX);

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

{

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

}

for(l=0;l<MAX-3;l++)

{

for(m=l+1;m<MAX-2;m++)

{

s1[count\_s1].sum=store[l]+store[m];

s1[count\_s1].i=l;

s1[count\_s1].j=m;

count\_s1++;

}

}

for(l=2;l<MAX-1;l++)

{

for(m=l+1;m<MAX;m++)

{

s2[count\_s2].sum=store[l]+store[m];

s2[count\_s2].i=l;

s2[count\_s2].j=m;

count\_s2++;

}

}

HF1();

HF2();

for(l=0;l<count\_s1;l++)

{

if(H1[l].sum==H2[l].sum)

{

if(H1[l].i!=H2[l].i && H1[l].j!=H2[l].j && H1[l].i!=H2[l].j && H1[l].j!=H2[l].i )

{

printf(" (%d,%d) & (%d,%d) \n",store[H1[l].i],store[H1[l].j],store[H2[l].i],store[H2[l].j]);

}

}

}

}

## <Detection of cycle.>

// DFS :: Graph Cycle

#include<stdio.h>

#include<stdlib.h>

#define MAX 5

int arr[MAX][MAX];

int visited[MAX]={0};

int dimension;

void DFS(int index)

{

int j;

visited[index]++;

if(visited[index]>1)

{

printf("The cycle exists\n");

exit(0);

}

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

{

if(arr[index][j]==1)

{

DFS(j);

}

}

}

int main()

{

int i,j;

printf("Enter the dimensions of the adjacency matrix of Graph\n");

scanf("%d",&dimension);

printf("Enter %d elements\n",dimension\*dimension);

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

{

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

{

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

}

}

DFS(0);

printf("The cycle does not exists\n");

}