BCSL-033 Data and File Structures Lab

Question 1 Write algorithm and program for the conversion of a Tree to a Binary Tree.
Ans 1.

Algorithm for the conversion of a Tree to a Binary Tree:

Step 1: use the root of the general tree as the root of the binary tree

Step 2: determine the first child of the root. This is the leftmost node in the general tree at the next level

Step 3: insert this node. The child reference of the parent node refers to this node

Step 4: continue finding the first child of each parent node and insert it below the parent node with the child reference of the parent to this node.

Step 5: when no more first children exist in the path just used, move back to the parent of the last node entered and repeat the above process. In other words, determine the first sibling of the last node entered.

Step 6: complete the tree for all nodes. In order to locate where the node fits you must search for the first child at that level and then follow the sibling references to a nil where the next sibling can be inserted. The children of any sibling node can be inserted by locating the parent and then inserting the first child. Then the above process is repeated.

Exit.

Program for the conversion of a Tree to a Binary Tree in C:

```
#include"stdio.h"
#include"conio.h"
```

```
#include"alloc.h"
#include"stdlib.h"
#define n 15
struct node
{
int data;
struct node *lptr;
struct node *rptr;
};
typedef struct node node;
struct stack
{
int top;
int number[n];
struct node *location[n];
};
typedef struct stack stack;
void convert(node **);
void display(node *);
void main()
{
node *root=NULL;
clrscr();
convert(&root);
printf("\ndata in binary tree :\n");
display(root);
getch();
}
void convert(node **root)
{
char c;
int pl,level,x;
stack s;
node *p=NULL,*ploc=NULL;
node *newnode=(node *)malloc(sizeof(node));
s.top=-1;
if(newnode==NULL)
{
printf("memory overflow");
return;
}
if(*root==NULL)
```

```
{
*root=newnode;
(*root)->lptr=(*root)->rptr=NULL;
printf("enter level and data>");
scanf("%d%d",&level,&x);
(*root)->data=x;
s.location[0]=*root;
s.number[0]=level;
s.top=0;
}
l:printf("to enter a data(y/n)?");
c=getch();
if(c=='y')
{
printf("\nenter level and data>");
scanf("%d%d",&level,&x);
p=(node *)malloc(sizeof(node));
p->lptr=p->rptr=NULL;
p->data=x;
pl=s.number[s.top];
ploc=s.location[s.top];
if(level>pl)
ploc->lptr=p;
else
{
while(pl>level)
{
s.top=s.top+1;
pl=s.number[s.top];
ploc=s.location[s.top];
}
ploc->rptr=p;
s.top=s.top-1;
}
s.top=s.top-1;
s.number[s.top]=level;
s.location[s.top]=p;
goto l;
}}
void display(node *p)
{
if(p!=NULL)
```

```
{
printf(" %d ",p->data);
display(p->lptr);
display(p->rptr);
}
}
```

Question2: Write algorithm and program for multiplication of two Sparse Matrices using Pointers.
Ans 2.

Algorithm for multiplication of two Sparse Matrices using Pointers:

Not Available.

Program for multiplication of two Sparse Matrices using Pointers in C:

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

#define MAX1 3
#define MAX2 3
#define MAXSIZE 20

#define TRUE 1
#define FALSE 2
```

```
struct sparse
{
   int *sp ;
   int row ;
   int *result ;
} ;

void initsparse ( struct sparse * ) ;
void create_array ( struct sparse * ) ;
int count ( struct sparse ) ;
void display ( struct sparse ) ;
void create_tuple ( struct sparse*, struct sparse ) ;
void display_tuple ( struct sparse ) ;
void prodmat ( struct sparse *, struct sparse, struct sparse ) ;
void searchina ( int *sp, int ii, int*p, int*flag ) ;
void searchinb ( int *sp, int jj, int colofa, int*p, int*flag ) ;
void display_result ( struct sparse ) ;
void delsparse ( struct sparse * ) ;

void main( )
{
   struct sparse s[5] ;
   int i ;

   clrscr( ) ;

   for ( i = 0 ; i <= 3 ; i++ )
      initsparse ( &s[i] ) ;

   create_array ( &s[0] ) ;

   create_tuple ( &s[1], s[0] ) ;
   display_tuple ( s[1] ) ;

   create_array ( &s[2] ) ;

   create_tuple ( &s[3], s[2] ) ;
   display_tuple ( s[3] ) ;

   prodmat ( &s[4], s[1], s[3] ) ;

   printf ( "\nResult of multiplication of two matrices: " ) ;
```

```
    display_result ( s[4] ) ;

    for ( i = 0 ; i <= 3 ; i++ )
        delsparse ( &s[i] ) ;

    getch( ) ;
}

/* initialises elements of structure */
void initsparse ( struct sparse *p )
{
    p -> sp = NULL ;
    p -> result = NULL ;
}

/* dynamically creates the matrix */
void create_array ( struct sparse *p )
{
    int n, i ;

    /* allocate memory */

    p -> sp = ( int * ) malloc ( MAX1 * MAX2 * sizeof ( int ) ) ;

    /* add elements to the array */
for ( i = 0 ; i < MAX1 * MAX2 ; i++ )
    {
            printf ( "Enter element no. %d: ", i ) ;
            scanf ( "%d", &n ) ;
            * ( p -> sp + i ) = n ;
    }
}

/* displays the contents of the matrix */
void display ( struct sparse s )
{
    int i ;

    /* traverses the entire matrix */
for ( i = 0 ; i < MAX1 * MAX2 ; i++ )
    {
```

```
        /* positions the cursor to the new line for every new row */
if ( i % 3 == 0 )
            printf ( "\n" ) ;
        printf ( "%d\t", * ( s.sp + i ) ) ;
    }
}

/* counts the number of non-zero elements */
int count ( struct sparse s )
{
    int cnt = 0, i ;

    for ( i = 0 ; i < MAX1 * MAX2 ; i++ )
    {
        if ( * ( s.sp + i ) != 0 )
            cnt++ ;
    }
    return cnt ;
}

/* creates an array that stores information about non-zero elements */
void create_tuple ( struct sparse *p, struct sparse s )
{
    int r = 0 , c = -1, l = -1, i ;

    /* get the total number of non-zero elements */

    p -> row = count ( s ) + 1 ;

    /* allocate memory */

    p -> sp = ( int * ) malloc ( p -> row * 3 * sizeof ( int ) ) ;

    /* store information about
      total no. of rows, cols, and non-zero values */

    * ( p -> sp + 0 ) = MAX1 ;
    * ( p -> sp + 1 ) = MAX2 ;
    * ( p -> sp + 2 ) = p -> row - 1 ;

    l = 2 ;
```

```
    /* scan the array and store info. about non-zero values
       in the 3-tuple */
for ( i = 0 ; i < MAX1 * MAX2 ; i++ )
    {
        c++ ;

        /* sets the row and column values */
if ( ( ( i % 3 ) == 0 ) && ( i != 0 ) )
        {
            r++ ;
            c = 0 ;
        }

        /* checks for non-zero element,
           row, column and non-zero value
           is assigned to the matrix */
if ( * ( s.sp + i ) != 0 )
        {
            l++ ;
            * ( p -> sp + l ) = r ;
            l++ ;
            * ( p -> sp + l ) = c ;
            l++ ;
            * ( p -> sp + l ) = * ( s.sp + i ) ;
        }
    }
}

/* displays the contents of the matrix */
void display_tuple ( struct sparse s )
{
    int i, j ;

    /* traverses the entire matrix */

    printf ( "\nElements in a 3-tuple: " ) ;

    j = ( * ( s.sp + 2 ) * 3 ) + 3 ;

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

```
      /* positions the cursor to the new line for every new row */
if ( i % 3 == 0 )
         printf ( "\n" ) ;
      printf ( "%d\t", * ( s.sp + i ) ) ;
   }
   printf ( "\n" ) ;
}

/* performs multiplication of sparse matrices */
void prodmat ( struct sparse *p, struct sparse a, struct sparse b )
{
   int sum, k, position, posi, flaga, flagb, i , j ;
   k = 1 ;

   p -> result = ( int * ) malloc ( MAXSIZE * 3 * sizeof ( int ) ) ;

   for ( i = 0 ; i < * ( a.sp + 0 * 3 + 0 ) ; i++ )
   {
      for ( j = 0 ; j < * ( b.sp + 0 * 3 + 1 ) ; j++  )
      {
         /* search if an element present at ith row */

         searchina ( a.sp, i, &position, &flaga ) ;
         if ( flaga == TRUE )
         {
            sum = 0 ;

            /* run loop till there are element at ith row
               in first 3-tuple */
while ( * ( a.sp + position * 3 + 0 ) == i )
            {
               /* search if an element present at ith col.
                  in second 3-tuple */

               searchinb ( b.sp, j, * ( a.sp + position * 3 + 1 ),
                     &posi, &flagb ) ;

               /* if found then multiply */
if ( flagb == TRUE )
                  sum = sum + * ( a.sp + position * 3 + 2 ) *
                        * ( b.sp + posi * 3 + 2 ) ;
               position = position + 1 ;
```

```
            }

            /* add result */
if ( sum != 0 )
            {
               * ( p -> result + k * 3 + 0 ) = i ;
               * ( p -> result + k * 3 + 1 ) = j ;
               * ( p -> result + k * 3 + 2 ) = sum ;
               k = k + 1 ;
            }
         }
       }
     }

   /* add total no. of rows, cols and non-zero values */

   * ( p -> result + 0 * 3 + 0 ) = * ( a.sp + 0 * 3 + 0 ) ;
   * ( p -> result + 0 * 3 + 1 ) = * ( b.sp + 0 * 3 + 1 ) ;
   * ( p -> result + 0 * 3 + 2 ) = k - 1 ;
}

/* searches if an element present at iith row */
void searchina ( int *sp, int ii, int *p, int *flag )
{
   int j ;
   *flag = FALSE ;
   for ( j = 1 ; j <= * ( sp + 0 * 3 + 2 ) ; j++ )
   {
      if ( * ( sp + j * 3 + 0 ) == ii )
      {
         *p = j ;
         *flag = TRUE ;
         return ;
      }
    }
}

/* searches if an element where col. of first 3-tuple
   is equal to row of second 3-tuple */
void searchinb ( int *sp, int jj, int colofa, int *p, int *flag )
{
    int j ;
```

```
    *flag = FALSE ;
    for ( j = 1 ; j <= * ( sp + 0 * 3 + 2 ) ; j++ )
    {
      if ( * ( sp + j * 3 + 1 ) == jj && * ( sp + j * 3 + 0 ) == colofa )
      {
         *p = j ;
         *flag = TRUE ;
         return ;
      }
   }
}

/* displays the contents of the matrix */
void display_result ( struct sparse s )
{
   int i ;

   /* traverses the entire matrix */
for ( i = 0 ; i < ( * ( s.result + 0 + 2 ) + 1 ) * 3 ; i++ )
   {
      /* positions the cursor to the new line for every new row */
if ( i % 3 == 0 )
         printf ( "\n" ) ;
      printf ( "%d\t", * ( s.result + i ) ) ;
   }
}

/* deallocates memory */
void delsparse ( struct sparse *s )
{
   if ( s -> sp != NULL )
      free ( s -> sp ) ;
   if ( s -> result != NULL )
      free ( s -> result ) ;
}
```