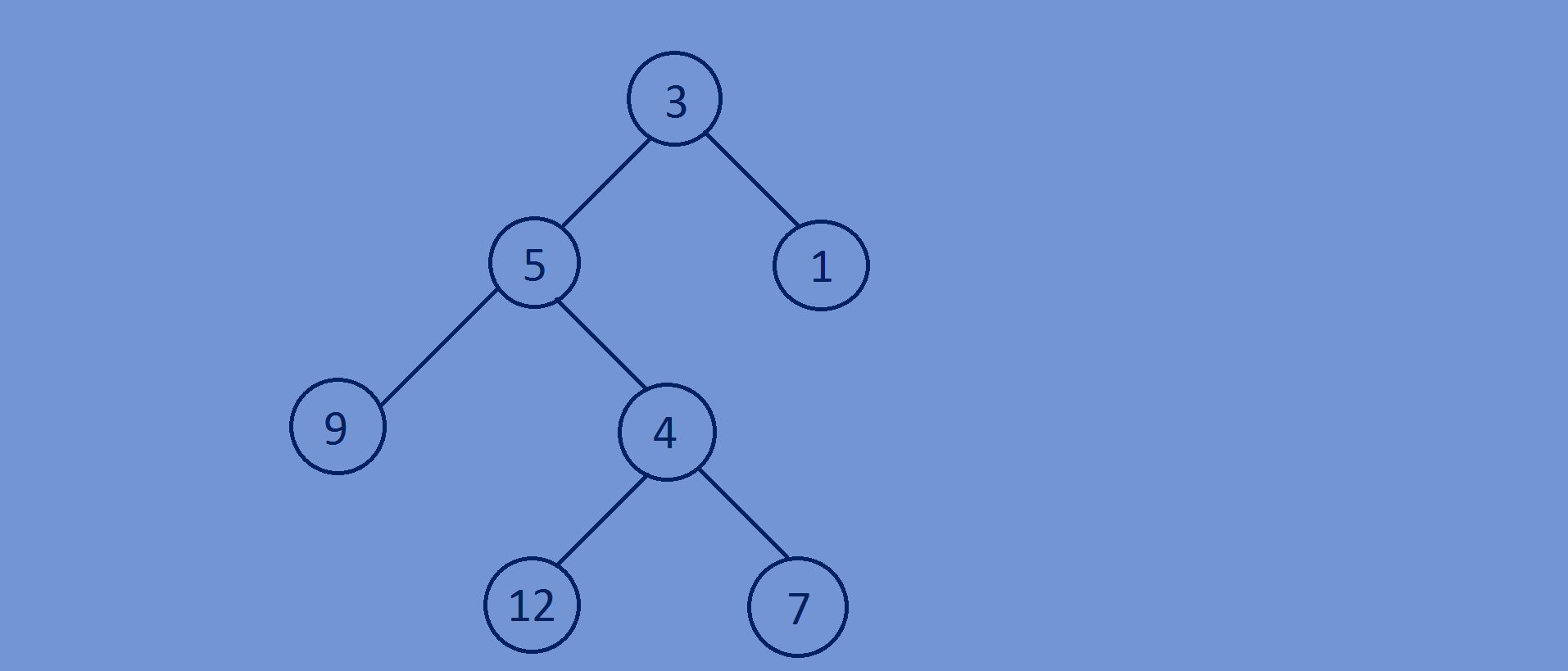
**Data Structures**

Implementation of a Binary Tree

**Representation of a node of a tree:**

* A binary tree can have a maximum of two children.
* A node has two links.
* A node is a structure which contains of following parts.
* Data.
* Pointer to the left child.
* Pointer to the right child.

**Example of tree node with integer data:**

typedef struct BTNODE{

int data;

struct BTNODE \*left;

struct BTNODE \*right;

}BTNODE;

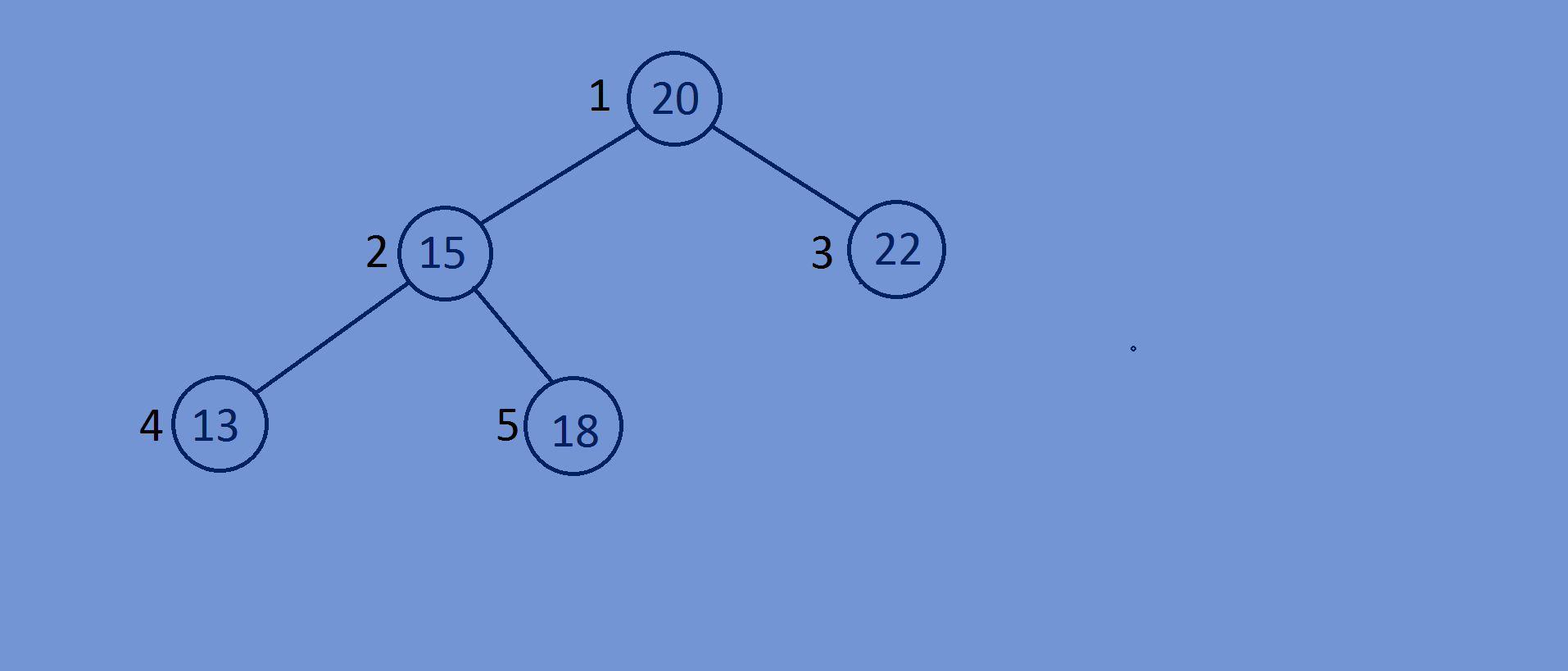
**Inserting nodes into a Binary Tree:**

* Implement complete binary tree.
* Write a insert function that adds a new node in the last level and at the left most

Available position.

* Give integer positions to the nodes starting from root node in level order fashion.

So that root node is at position 1 and left child of root node will be at position 2

****And the right child of the root node will be at position 3,while inserting node into

The tree we will follow the same order.

**Binary Representation**

1-1

2-10

3-11

4-100

5-101

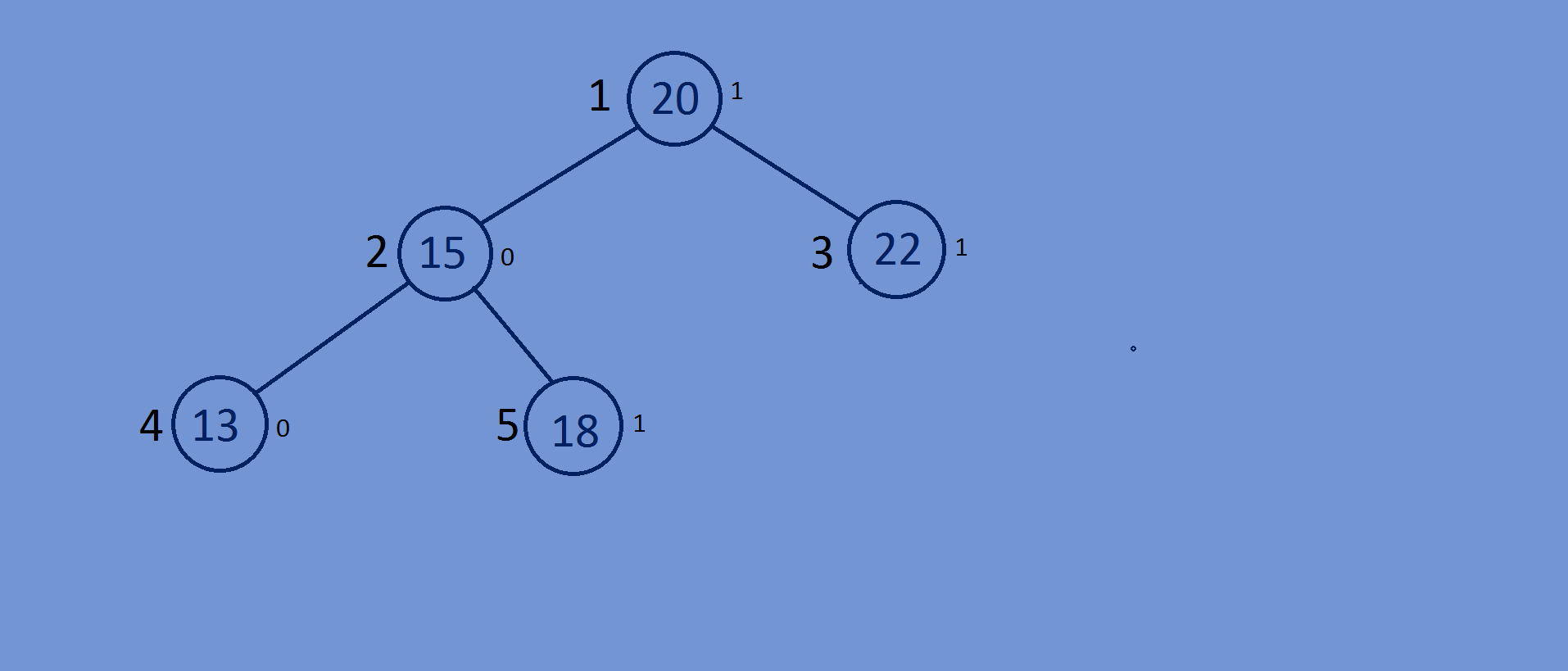
* For inserting n nodes we traverse through the loop from i=1 to i<=n(including n)

And while traversing we take “i” as reference and and if i==1 we insert root node

And then we insert new nodes in level order from left to right.

* What we do is we convert the integers into binary no’s and store the binary digits

Of each integer In a linked list,now if i==1 we make the new node as root else

We traverse the tree using the binary digits=>if we encounter 0 we traverse to the left child else if we encounter 1 we traverse to right child,here we skip the data in first node of the linked list which is always one .

Binary Representation of integers:

1-|1|

2-|1||0|

3-|1||1|

4-|1||0||0|

5-|1||0||1|

**Function for converting decimal no into binary digits:**

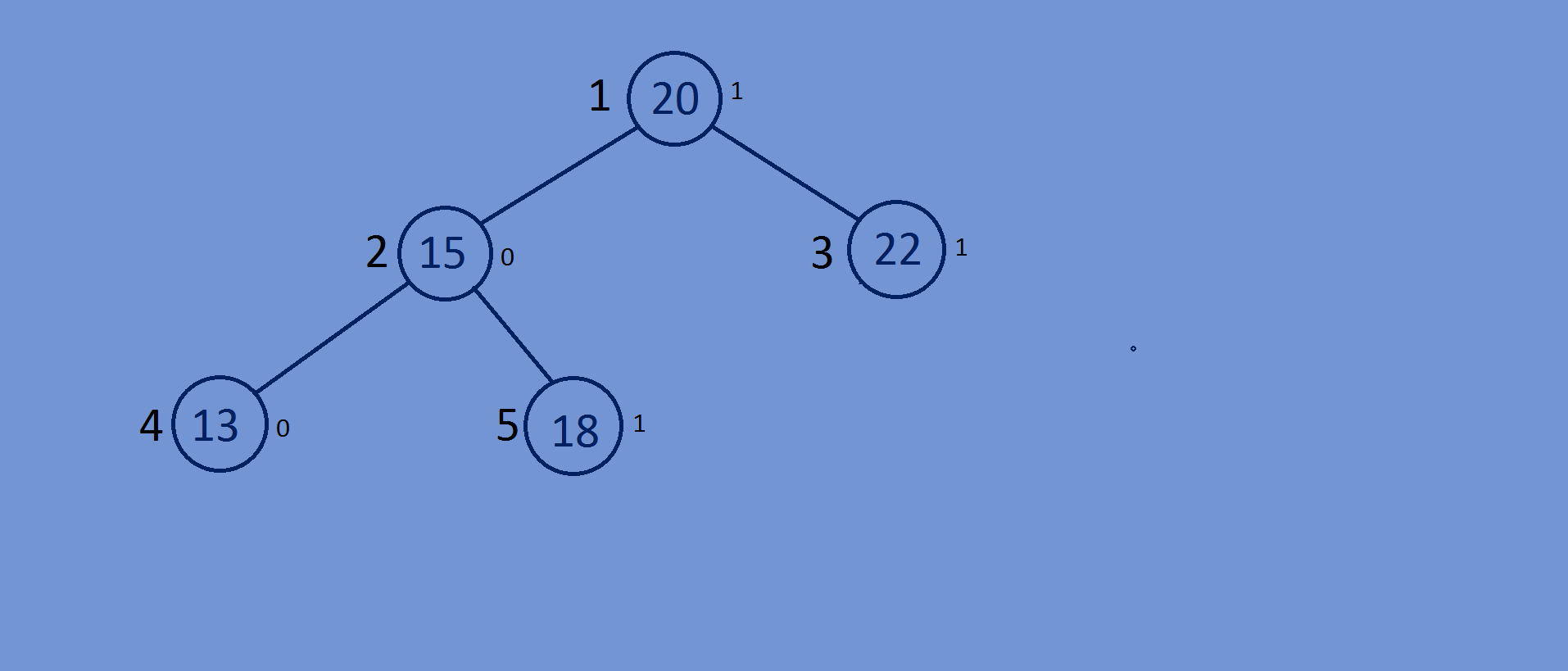
//converting a given number into binary digits and storing them into a linked list  
lin\_list \*binary(**int** i){  
 lin\_list \*head=NULL;  
 **while**(i!=0){  
 lin\_list \*ptr=(lin\_list\*)malloc(**sizeof**(lin\_list));  
 ptr->data=i%2;  
 ptr->next=NULL;  
 i=i/2;  
 ptr->next=head;  
 head=ptr;  
   
 }  
 **return** head;

}

**Function for inserting nodes into a Binary Tree:**

//inserting nodes into a tree.  
BTNODE \*InsertNode(BTNODE \*Btree,**int** n){  
 lin\_list \*Binaryno=NULL;  
 **for**(**int** i=1;i<=n;i++){  
 BTNODE \*temp=(BTNODE\*)malloc(**sizeof**(BTNODE)),\*temp1=Btree;  
 temp->data=i\*2;  
 temp->left=NULL;  
 temp->right=NULL;  
 Binaryno=binary(i);  
 Binaryno=Binaryno->next;  
 **if**(i==1){  
 Btree=temp;  
 }  
 **else**{  
 **while**(Binaryno->next){  
 **if**(Binaryno->data==0){  
 temp1=temp1->left;  
 }  
 **else if**(Binaryno->data==1){  
 temp1=temp1->right;  
 }  
 Binaryno=Binaryno->next;  
 }  
 **if**(Binaryno->data==0){  
 temp1->left=temp;  
 }  
 **else if**(Binaryno->data==1){  
 temp1->right=temp;  
 }  
 }  
 }  
 **return** Btree;  
}

**Whole program:**

#include<stdio.h>  
#include<stdlib.h>  
//creating a node of binary tree.  
**typedef struct** BTNODE{  
 **int** data;  
 **struct** BTNODE \*left;  
 **struct** BTNODE \*right;  
}BTNODE;  
//creating a node for linked list  
**typedef struct** lin\_list{  
 **int** data;  
 **struct** lin\_list \*next;  
}lin\_list;  
//converting a given number into binary digits and storing them into a linked  
lin\_list \*binary(**int** i){  
 lin\_list \*head=NULL;  
 **while**(i!=0){  
 lin\_list \*ptr=(lin\_list\*)malloc(**sizeof**(lin\_list));  
 ptr->data=i%2;  
 ptr->next=NULL;  
 i=i/2;  
 ptr->next=head;  
 head=ptr;  
 }  
 **return** head;  
}  
//inserting nodes into a tree.  
BTNODE \*InsertNode(BTNODE \*Btree,**int** n){  
 lin\_list \*Binaryno=NULL;  
 **for**(**int** i=1;i<=n;i++){  
 BTNODE \*temp=(BTNODE\*)malloc(**sizeof**(BTNODE)),\*temp1=Btree;  
 temp->data=i\*2;  
 temp->left=NULL;  
 temp->right=NULL;  
 Binaryno=binary(i);  
 Binaryno=Binaryno->next;  
 **if**(i==1){  
 Btree=temp;  
 }  
 **else**{  
 **while**(Binaryno->next){  
 **if**(Binaryno->data==0){  
 temp1=temp1->left;  
 }  
 **else if**(Binaryno->data==1){  
 temp1=temp1->right;  
 }  
 Binaryno=Binaryno->next;  
 }  
 **if**(Binaryno->data==0){  
 temp1->left=temp;  
 }  
 **else if**(Binaryno->data==1){  
 temp1->right=temp;  
 }  
 }  
 }  
 **return** Btree;  
}  
//printing the data of the nodes.  
**void** printdata(BTNODE \*Btree){  
 **if**(Btree){  
 printf("%d ",Btree->data);  
 printdata(Btree->left);  
 printdata(Btree->right);  
  
 }  
}  
//main function  
**int** main(){  
 BTNODE \*Btree=NULL;  
 **int** n=10;  
 Btree=InsertNode(Btree,n);  
 printdata(Btree);  
 **return** 0;  
}

2 4 8 16 18 10 20 6 12 14

#include<stdio.h>

#include<stdlib.h>

#include<limits.h>

#define N 4

typedef struct Node

{

// stores parent node of current node

// helps in tracing path when answer is found

struct Node\* parent;

// stores matrix

int mat[N][N];

// stores blank tile cordinates

int x, y;

// stores the number of misplaced tiles

int cost;

// stores the number of moves so far

int level;

}Node;

//creating a node

typedef struct Bstree{

int data;

int ton;

struct Bstree \*left;

struct Bstree \*right;

Node \*Slidenode;

}Bstree;

typedef struct lin\_list{

int data;

struct lin\_list \*next;

}lin\_list;

//converting a given number into binary digits and storing them into a linked

lin\_list \*binary(int i){

lin\_list \*head=NULL;

while(i!=0){

lin\_list \*ptr=(lin\_list\*)malloc(sizeof(lin\_list));

ptr->data=i%2;

ptr->next=NULL;

i=i/2;

ptr->next=head;

head=ptr;

}

return head;

}

//creating new nodes

Bstree \*createnewnode(int data){

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

newnode->data=data;

newnode->left=NULL;

newnode->right=NULL;

return newnode;

}

//swaping data of two nodes

void swapData(Bstree \*temp1,Bstree \*temp2){

Node \*temp3;

temp3=temp1->Slidenode;

temp1->Slidenode=temp2->Slidenode;

temp2->Slidenode=temp3;

}

//recursive function for bubbling the nodes based on comparison.

void BubbleDown(Bstree \*node){

Bstree \*smallestNode;

if(node==NULL){

return;

}

if(node->left!=NULL && ((node->left->Slidenode->cost + node->left->Slidenode->level) < (node->Slidenode->cost + node->Slidenode->level))){

//printf("\n%d????????????????????\n",node->left->Slidenode->cost);

smallestNode=node->left;

swapData(node,smallestNode);

BubbleDown(smallestNode);

}

if(node->right!=NULL && ((node->right->Slidenode->cost + node->right->Slidenode->level) < (node->Slidenode->cost + node->Slidenode->level))){

//printf("\n%d&&&&&&&&&&&&&&&&&&&&\n",node->right->Slidenode->cost);

smallestNode=node->right;

swapData(node,smallestNode);

BubbleDown(smallestNode);

}

}

//function for heapifying of a complete binary tree

void heapify(Bstree \*root){

if (root == NULL) {

return;

}

heapify(root->left);

heapify(root->right);

BubbleDown(root);

}

//print all the nodes of a tree in preorder fashion

void preorder(Bstree \*root){

if(root){

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

preorder(root->left);

preorder(root->right);

}

}

int getInvCount(int arr[])

{

int inv\_count = 0;

for (int i = 0; i < N \* N - 1; i++)

{

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

{

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

inv\_count++;

}

}

return inv\_count;

}

int findXPosition(int puzzle[N][N])

{

for (int i = N - 1; i >= 0; i--){

for (int j = N - 1; j >= 0; j--){

if (puzzle[i][j] == 0){

                return N - i;

            }

        }

    }

}

int isSolvable(int puzzle[N][N])

{

int invCount = getInvCount((int\*)puzzle);

if (N & 1)

return !(invCount & 1);

else

{

int pos = findXPosition(puzzle);

if (pos & 1)

return !(invCount & 1);

else

return invCount & 1;

}

}

Bstree\* pop(Bstree \*root){

Bstree \*left=root->left;

Bstree \*right = root->right;

Bstree \*temp=(Bstree\*)malloc(sizeof(Bstree)),\*temp1=root;

lin\_list \*Binaryno=NULL;

Binaryno=binary(root->ton);

Binaryno=Binaryno->next;

if(root->left == NULL && root->right == NULL){

return NULL;

}

else{

while(Binaryno->next){

if(Binaryno->data==0){

temp1=temp1->left;

}

else if(Binaryno->data==1){

temp1=temp1->right;

}

Binaryno=Binaryno->next;

}

if(Binaryno->data==0){

temp=temp1->left;

temp1->left=NULL;

}

else if(Binaryno->data==1){

temp=temp1->right;

temp1->right=NULL;

}

temp->left = root->left;

temp->right = root->right;

temp->ton=root->ton - 1;

}

return temp;

}

//inserting nodes into a tree.

Bstree \*push(Bstree \*Btree,Node \*root){

lin\_list \*Binaryno=NULL;

Bstree \*temp=(Bstree\*)malloc(sizeof(Bstree)),\*temp1=Btree;

temp->Slidenode=root;

temp->left=NULL;

temp->right=NULL;

if(Btree!=NULL){

Btree->ton=Btree->ton + 1;

Binaryno=binary(Btree->ton);

Binaryno=Binaryno->next;

//printf("hai surya\n");

}

if(Btree==NULL){

Btree=temp;

Btree->ton = 1;

//printf("hai surya\n");

}

else{

while(Binaryno->next){

if(Binaryno->data==0){

temp1=temp1->left;

}

else if(Binaryno->data==1){

temp1=temp1->right;

}

Binaryno=Binaryno->next;

}

if(Binaryno->data==0){

temp1->left=temp;

}

else if(Binaryno->data==1){

temp1->right=temp;

}

}

return Btree;

}

void matcpy(int child[N][N],int parent[N][N]){

for(int i=0;i<N;i++){

for(int j=0;j<N;j++){

child[i][j]=parent[i][j];

}

}

}

void swaptiles(int mat[N][N],int newX,int newY,int oldX,int oldY){

int temp;

temp = mat[oldX][oldY];

mat[oldX][oldY]=mat[newX][newY];

mat[newX][newY]=temp;

}

int calculateCost(int initial[N][N], int final[N][N])

{

int count = 0;

for (int i = 0; i < N; i++){

for (int j = 0; j < N; j++){

if (initial[i][j] && initial[i][j] != final[i][j])

count++;

}

}

return count;

}

// Function to allocate a new node

Node\* newNode(int mat[N][N], int x, int y, int newX,

int newY, int level, Node\* parent)

{

Node\* node = (Node\*)malloc(sizeof(Node));

//(Bstree\*)malloc(sizeof(Bstree))

// set pointer for path to root

node->parent = parent;

// copy data from parent node to current node

//memcpy(node->mat, mat, sizeof node->mat);

matcpy(node->mat,mat);

// move tile by 1 postion

//swap(node->mat[x][y], node->mat[newX][newY]);

swaptiles(node->mat,newX,newY,x,y);

// set number of misplaced tiles

node->cost = INT\_MAX;

// set number of moves so far

node->level = level;

// update new blank tile cordinates

node->x = newX;

node->y = newY;

return node;

}

// Function to print N x N matrix

void printMatrix(int mat[N][N])

{

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

{

for (int j = 0; j < N; j++){

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

}

printf("\n");

}

}

// print path from root node to destination node

void printPath(Node\* root)

{

if (root == NULL){

return;

}

printPath(root->parent);

printMatrix(root->mat);

printf("\n");

}

// Function to check if (x, y) is a valid matrix cordinate

int isSafe(int x, int y)

{

return (x >= 0 && x < N && y >= 0 && y < N);

}

int checkmatrix(int olist[N][N],int child[N][N]){

int count=0;

for(int i=0;i<N;i++){

for(int j=0;j<N;j++){

if(olist[i][j]==child[i][j]){

count++;

}

}

}

if(count == 16){

return 1;

}

else{

return 0;

}

}

int findinopenlist(Bstree \*olist,Node \*child){

if(olist){

return 0;

}

//printf("hai surya\n");

printMatrix(olist->Slidenode->mat);

if(checkmatrix(olist->Slidenode->mat,child->mat)){

if(olist->Slidenode->cost+ olist->Slidenode->level > child->cost + child->level){

olist->Slidenode->cost=INT\_MIN;

return 1;

}

}

findinopenlist(olist->left,child);

findinopenlist(olist->right,child);

}

int findclosedlist(Bstree \*clist,Node \*child){

if(clist){

return 0;

}

if(checkmatrix(clist->Slidenode->mat,child->mat)){

if(clist->Slidenode->cost+ clist->Slidenode->level > child->cost + child->level){

clist->Slidenode->cost=INT\_MIN;

return 1;

}

}

findclosedlist(clist->left,child);

findclosedlist(clist->right,child);

}

void solve(int initial[N][N], int x, int y,int final[N][N])

{

// botton, left, top, right

int row[] = { 1, 0, -1, 0 };

int col[] = { 0, -1, 0, 1 };

// Create a priority queue to store live nodes of

// search tree;

//priority\_queue<Node\*, std::vector<Node\*>, comp> pq;

Bstree \*pq=NULL;

Bstree \*clist=NULL;

// create a root node and calculate its cost

Node\* root = newNode(initial, x, y, x, y, 0, NULL);

root->cost = calculateCost(initial, final);

// Add root to list of live nodes;

//pq.push(root);

pq=push(pq,root);

//printf("%d-----------\n",pq->ton);

heapify(pq);

// Finds a live node with least cost,

// add its childrens to list of live nodes and

// finally deletes it from the list.

//while (!pq.empty())

int j=2;

while (pq)

{

// Find a live node with least estimated cost

// Node\* min = pq.top();

Node\* min = pq->Slidenode;

printf("%d~~~~~~\n",min->cost);

clist = push(clist,min);

pq=pop(pq);

//heapify(pq);

// The found node is deleted from the list of

// live nodes

//pq.pop();

// if min is an answer node

if (min->cost == 0)

{

// print the path from root to destination;

printPath(min);

return;

}

// do for each child of min

// max 4 children for a node

for (int i = 0; i < 4; i++)

{

if (isSafe(min->x + row[i], min->y + col[i]))

{

// create a child node and calculate

// its cost

Node\* child = newNode(min->mat, min->x,

min->y, min->x + row[i],

min->y + col[i],

min->level + 1, min);

child->cost = calculateCost(child->mat, final);

// Add child to list of live nodes

//pq.push(child);

//printf("hai surya\n");

if(pq!=NULL && findinopenlist(pq,child)){

heapify(pq);

pq = pop(pq);

pq = push(pq,child);

}

else if(clist!=NULL && findclosedlist(clist,child)){

heapify(clist);

clist = pop(clist);

pq = push(pq,child);

}

else{

pq=push(pq,child);

}

//pq=push(pq,child);

printf("\n%d++++++++\n",child->cost);

printf("%d----------------",pq->Slidenode->cost);

}

}

heapify(pq);

}

}

//main function

int main(){

//Bstree \*tree=NULL;

// Initial configuration

// Value 0 is used for empty space

int initial[N][N] =

{

{5, 1, 3,4},

{9, 2, 7,8},

{13, 6, 10,12},

{14, 0, 11, 15}

};

// Solvable Final configuration

// Value 0 is used for empty space

int final[N][N] =

{

{1, 2, 3,4},

{5, 6, 7,8},

{9, 10, 11,12},

{13, 14, 15, 0}

};

int x = 3, y = 1;

solve(initial, x, y, final);

return 0;

}