ISSN:2319-9873

Research & Reviews: Journal of Engineering and Technology

Matrix Search Algorithm Using Binary Search Trees

Shaik Kareem B*

Assistant Professor in CSE, HITAM, Hyderabad, India

Research Article

Received date: 22/01/2016

Accepted date: 19/03/2016

Published date: 16/04/2016

***For Correspondence**

Shaik Kareem B. Assistant Professor in CSE, HITAM, Hyderabad, India.

E-mail: kareembashas.cse@hitam.org

Keywords: Algorithm, Matrix, Binary search, Programming language.

ABSTRACT

Searching is a process of finding whether a key element belongs to search space or not. The proposed Matrix Search Algorithm using Binary Search Trees takes unsorted matrix of order mxn and key element to be searched as input, constructs two Binary Search Trees BST1 of lower triangular matrix including diagonal elements and BST2 of upper triangular matrix excluding diagonal elements of input matrix. Key element is searched in BST1, if a found return true otherwise key element is searched in BST2, if found returns true otherwise returns False. I followed the various stages of Software Development Life Cycle to demonstrate the proposed searching algorithm. In Section I, proposed algorithm is introduced. In section II, proposed algorithm is Analyzed and Designed. In section III, proposed algorithm is Implemented using C programming Language. In section IV, I tested the implementation of proposed algorithm using different test cases. In section V, I concluded the proposed algorithm.

INTRODUCTION

Searching is a process of finding whether a key element belongs to search space or not. There are many searching algorithms to search a key element within a search space. Each algorithm has its own advantages and disadvantages. The proposed Matrix Search Algorithm using Binary Search Trees takes matrix of order mxn and key element to be searched as input. It constructs two binary search trees BST1 and BST2 by using elements of input matrix. BST1 is constructed by using lower triangular matrix including diagonal elements of input matrix and BST2 is constructed by using upper triangular matrix excluding diagonal elements of input matrix. In the proposed algorithm, I am dividing the matrix into two parts as Lower Triangular Matrix and Upper Triangular Matrix. For each part of matrix, a binary search tree is constructed. First key element is searched in BST1 from root node, if found then algorithm returns true otherwise key element is searched in BST2 from root node, if found then algorithm returns true otherwise false is returned. I am using searching algorithm of Binary Search Tree to search key element within BST1 and BST2. A given key element is first searched in BST1, if not found then same key element is searched in BST2, if found returns true otherwise returns false. The searching time of key element by using proposed algorithm is O(log xy) where x is the number of nodes of BST1 and y is the number of nodes of BST2. The time taken to construct two binary search trees of elements of given input matrix is O(mn!) where m is the number of rows of input matrix and n is the number of columns per row of input matrix.

ANALYSIS AND DESIGN OF PROPOSED ALGORITHM

Figure 1 shows the searching of key element within unsorted matrix of order mxn. First the given input matrix of order mxn is divided into two parts. The first part of matrix is lower triangular matrix including diagonal elements and second part of matrix is upper triangular matrix excluding diagonal elements. Two Binary Search Trees BST1 and BST2 are constructed by using two parts of matrix. BST1 is constructed by using first part of matrix and BST2 is constructed by using second part of matrix. Key element is searched in BST1, if found then returns true otherwise key element is searched in BST2, if found then returns true otherwise false is returned.

Figure 2 shows the conversion of unsorted input matrix of order 3x3 into two binary search trees. The given input matrix consists of 9 elements {{1 4 3},{7 8 6},{2 5 9}}. It is divided into two parts. The first part is lower triangular matrix including diagonal

elements, which contains 6 elements {1 7 8 2 5 9}. The second part is upper triangular matrix excluding diagonal elements, contains 3 elements {4 3 6}. Using the elements of first part of matrix, Binary Search Tree BST1 is constructed and using the elements of second part of matrix, BST2 is constructed. Key element 5 is searched in BST1, returns true since key element 5 belongs to BST1. Key element 6 is searched in BST1 but element 6 does not belongs to BST1, so key element 6 is searched in BST2, returns true since key element 6 belongs to BST2.

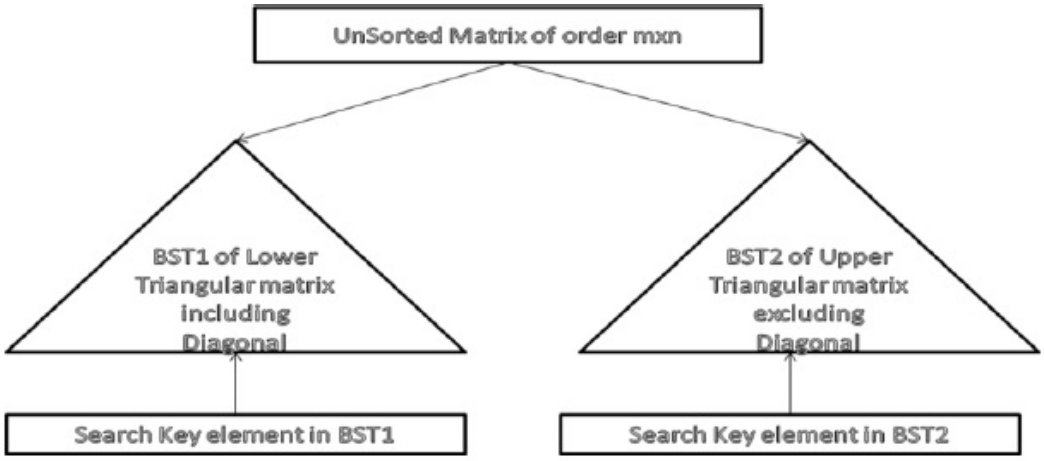

Figure 1. Conversion of unsorted Matrix into two Binary Search Trees and searching key element.

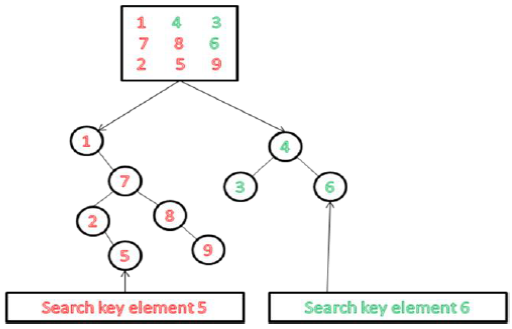

Figure 2. Searching key elements 5 and 6 in BST1 and BST2.

Algorithm 1.1 searches a key element within BST1, if a found return true otherwise key element is searched in BST2, if found returns true otherwise false are returned. It takes root node of BST1, root node of BST2 and key element to be searched as input[1].

```
Algorithm Matrix Search (tree1, tree2, key){l
//tree1 is the root node of BST1
//tree2 is the root node of BST2
//key is the element to be searched within input matrix of order mxn
 while(tree1!=NULL){ // if tree1 is not an empty node
if(tree1->data==key) then // if key element is equal to element of node of BST1
return True; // key element found within BST1
else if(key<tree1->data) then // if key element is smaller than element of node of BST1
tree1=tree1->left; // search key element within left sub tree of current node of BST1
else // if key element is greater than element of node of BST1
tree1=tree1->right; } // search key element within right sub tree of current node of BST1
while(tree2!=NULL){ // if tree2 is not an empty node
if(tree2->data==key) then // if key element is equal to element of node of BST2
return True; // key element found within BST2
else if(key<tree2->data) // if key element is smaller than element of node of BST2
tree2=tree2->left; // search key element within left sub tree of current node of BST2
```

else // if key element is greater than element of node of BST2

tree2=tree2->right; }//search key element within right sub tree of current node of BST2

return False; // key element not found in BST1 and BST2

}

Algorithm 1.1 searches key element within BST1 and BST2

Let m be the number of nodes of BST1, n be the number of nodes of BST2, T(m, n) be the time taken to search a key element within input matrix, f(m) be the time taken to search a key element within BST1 and g(n) be the time taken to search a key element within BST2.

f(m)= f(m/2) + A

f(m)= f(m/4) +2A

f(m)= f(m/8) + 3A

f(m)= f(m/23) + 3A

f(m)= f(m/2k) +kA

let m=2k k=log m

f(m)= f(1)+ A log m

f(m)= log m = O(log m)

g(n)= g(n/2) + A

g(n)= g(n/4) +2A

g(n)= g(n/8) + 3A

g(n)= g(n/23) + 3A

g(n)= g(n/2k) +kA

let n=2k k=log n

g(n)= g(1)+ A log n

g(n)= log n = O(log n)

The time taken by Algorithm 1.1 is

T(m, n)= f(m) + g(n)

T(m, n)=log m + log n = log mn

T(m, n)= O(log mn)

Algorithm 1.2 constructs two Binary Search Trees BST1 and BST2. BST1 is constructed by using lower triangular matrix including diagonal elements of input matrix of order mxn and BST2 is constructed by using upper triangular matrix excluding diagonal elements of input matrix of order mxn[2,3].

Algorithm ConstructTrees(tree1,tree2,a[][],m,n) {

// tree1 is the root of BST1, constructed by using lower triangular matrix including diagonal //elements of input matrix of order mxn, tree2 is the root of BST2, constructed by using upper //triangular matrix excluding diagonal elements, a[][] is an input unsorted matrix of order mxn, //m is the number of rows of matrix and n is the number of columns per row of matrix.

for(row=1;row<=m;row=row+1)

for(col=1;col<=row;col=col+1)

insert(&tree1,a[row][col]); // inserting lower triangular matrix elements including diagonal //elements of matrix into binary search tree BST1

for(row=m-1;row>=1;row=row-1)

for(col=n;col>row;col=col-1)

insert(&tree2,a[row][col]);// inserting upper triangular matrix elements excluding diagonal //elements of matrix into binary search tree BST2

```
}
```

Algorithm 1.2 Constructing Binary Search Trees BST1 and BST2

Algorithm 1.2 divides the matrix into two parts. The first part is the lower triangular matrix including diagonal elements and second part is the upper triangular matrix excluding diagonal elements. The time taken to construct BST1 by using first part of matrix is mn!, since first part of matrix contains m rows and each row contains number of columns equal to row index. The time taken to construct BST2 by using second part of matrix is (m-1)(n-1)!, since second part of matrix contains (m-1) rows and each row contains number of columns equal to (row-1) index. The overall time taken by Algorithm 1.2 is as follows:

$T(m, n) = mn! + (m-1)(n-1)!$

$T(m, n)= mn! + m(n-1)! - (n-1)! = mn!$

$T(m, n)= O(mn!)$

IMPLEMENTATION OF PROPOSED ALGORITHM USING C

The Proposed Matrix Search Algorithm using Binary Search Trees is implemented by using C Programming Language[3], which is a robust, structure oriented programming language, which provides different concepts like functions, pointers, structures, arrays and so on, suitable to implement proposed Algorithm.

```
/* Implementation of Matrix Search Algorithm using Binary Search Trees */
#include <stdio.h>
#include <conio.h>
struct node { // structure of node of Binary Search Tree
int data;
struct node *left,*right;
};
struct node *tree1=NULL,*tree2=NULL; // root nodes of BST1 and BST2
int a[10][10],n; // input matrix of order nxn
void insert(struct node **p,int ele) { // inserting elements into Binary Search Tree
 if((*p)==NULL) { // if current node is empty
(*p)=(struct node *)malloc(sizeof(struct node)); // create new node within BST
(*p)->data=ele; // place element into new node of BST
(*p)->left=NULL; // left sub tree of new node is empty
(*p)->right=NULL;// right sub tree of new node is empty
} else if(ele<(*p)->data) // if element is smaller than element of current node of BST
 insert(&((*p)->left),ele);// insert element into left sub tree of current node of BST
else if(ele>(*p)->data)// if element is greater than element of current node of BST
insert(&((*p)->right),ele);// insert element into right sub tree of current node of BST
}
int MatrixSearch(int key){ // searches key element within BST1 and BST2
struct node *temp=tree1;
while(temp!=NULL){ // search key element within BST1
if(temp->data==key) // if key element found in BST1 then return true return 1;
else if(key<temp->data) // if key element is smaller than element of node of BST1
temp=temp->left; // search key element within left sub tree of current node of BST1
else// if key element is greater than element of node of BST1
temp=temp->right; }// search key element within right sub tree of current node of BST1
```

```
temp=tree2;
while(temp!=NULL){ // search key element within BST2
if(temp->data==key)// if key element found within BST2 then return true return 1;
else if(key<temp->data)//if key element is smaller than element of node of BST2
temp=temp->left;// search key element within left sub tree of current node of BST2
else//if key element is greater than element of node of BST2
temp=temp->right; }//search key element within right sub tree of current node of BST2
return 0; } // if key element not found within BST1 and BST2 then return false
void ConstructTrees() { // construct two binary search trees of elements of given input matrix int row,col;
for(row=1;row<=n;row++)
for(col=1;col<=row;col++)
insert(&tree1,a[row][col]); // inserting elements into BST1 using lower triangular matrix //including diagonal element of input matrix of order nxn
for(row=n-1;row>=1;row--)
for(col=n;col>row;col--)
insert(&tree2,a[row][col]); } // inserting elements into BST2 using upper triangular matrix //excluding diagonal elements of input matrix of order nxn
void main() {
 int i,j,key;
clrscr();
printf("\nSorting of matrix of order nxn,Enter n value:\n");
scanf("%d",&n);
printf("\nEnter %d elements of matrix:\n",n*n);
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]); // reading elements into input matrix of order nxn
ConstructTrees();// constructing two binary search trees of elements of input matrix
printf("\nEnter key element to search:\n");
scanf("%d",&key); // reading key element to be searched within input matrix
if(MatrixSearch(key)==1) // if key element belongs to input matrix
printf("\nelement %d found",key);
else // if key element does not exist within input matrix
printf("\nelement %d not found",key);
getch(); }
```

Program searching key element in matrix of order mxn by using proposed algorithm.

TESTING OF PROPOSED ALGORITHM

Different Test cases are considered to test the result of Program. Consider the following test case in which, a matrix of order 3x3 with elements {{7 8 6}, {2 5 9}, {1 4 3}} and key element 18 are considered as input. **Figure 3** shows the output screen shot of program proposed algorithm using C.

CONCLUSION

I conclude that the proposed Matrix Search Algorithm using Binary Search Trees, takes unsorted matrix of order mxn and key

element to be searched as input, divides the matrix into two parts. The first part is the lower triangular matrix including diagonal elements and second part is the upper triangular matrix excluding diagonal elements. BST1 is constructed by using first part of matrix and BST2 is constructed by using second part of matrix. Key element is searched in BST1, if found returns True otherwise key element is searched in BST2, if found returns True otherwise False is returned. The searching time of key element by using proposed algorithm is O(log xy) where x is the number of nodes of BST1 and y is the number of nodes of BST2.

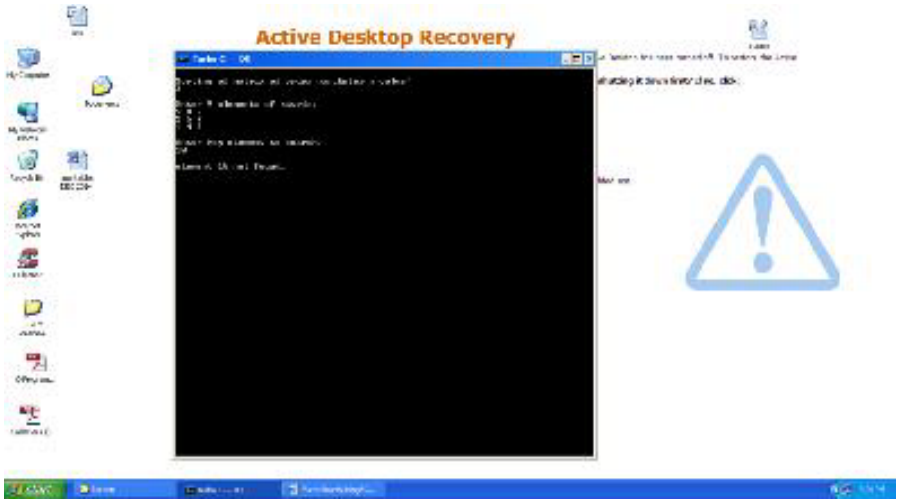

Figure 3. output screen shot of program.

REFERENCES

1. Aho AV, et al. Data Structures and Algorithms.(First Edition). Addison-Wesley Longman Publishing Co, USA. 1983.
2. Thomas H, et al. Introduction to Algorithms. (Second Edition) MIT Press Cambridge, England. 2001.
3. Donald E and Knuth. The Art of Computer Programming Sorting and Searching. (Second Edition). Addison-Wesley Longman Publishing Co, USA. 1998;3.