Journal of Basic and Applied Engineering Research
Print ISSN: 2350-0077; Online ISSN: 2350-0255; Volume 2, Number 10; April-June, 2015 pp. 835-838

http://www.krishisanskriti.org/jbaer.html

Popularity Tree Operation Algorithms

Priyanka Sehra Gaba[1] and Jyoti Arora[2]

[1,2]A.P in PDMCE Bahadurgarh
E-mail: [1]priyanka.mailid@gmail.com, [2]Jyoti.hans@gmail.com

Abstract—*Popularity tree is a binary tree which arranges nodes of the tree in such a manner that node on root is most visited node and node at the end is least visited node and thus BFS traversal shows the list of nodes in decreasing order of popularity. This paper shows the algorithms of various operations applied on popularity tree and its result as well. History of every node is being stored which decides popularity of every node and hence its positions in the tree.*

1. INTRODUCTION

Popularity tree is a type of Binary tree data structure that will arrange the nodes in a manner such that while BFS traversal we will get the nodes in decreasing order of popularity. The root node would be the most popular among all the nodes then its left child then right child and so on while moving downwards. A popularity factor is attached to every node and that popularity value of any node could be in terms of number of times that node has been visited

Every node contains:

"Info/Popularity factor"

"Alphabets" shows the value of that node and

"Numbers" shows the popularity factor of each node.

Example of a Popularity tree is shown below:

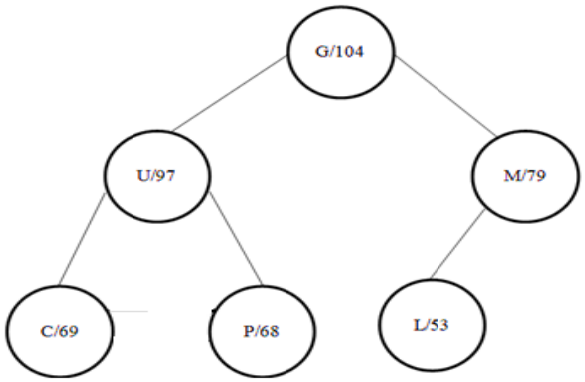

Fig. 1: Example of Popularity Tree

After BFS Traversal we got nodes in decreasing order of popularity.
G/104 U/97 M/79 C/69 P/68 L/53

Popularity tree will satisfy the following properties:

a) Create a binary tree that tree which can have at most two child nodes.
b) Maintains the popularity factor of every node.
c) Nodes are stored in a manner such that every node is:
   i) More popular than nodes which are after it in BFS traversal
   ii) Less popular than nodes before it in BFS traversal
d) Every visit of a node increases its popularity factor value by 1.
e) Nodes are exchanged if a node's popularity factor becomes more than the previous node in BFS traversal.
f) On Breadth First search traversal we will get a list of nodes in decreasing order of popularity.

2. BACKGROUND

Trees are used in databases to store huge amount of data. There are so many types of trees available each having its own functionality storing the nodes in different manner. Popularity tree is taking the advantages of binary tree in a manner that complexity of searching a node randomly is less as compared to any other type of data structure.

Although it is quite similar to a binary tree inheriting many of its features like In contrast to other binary trees which consider its info part, popularity tree stores the nodes by considering the popularity factor of various nodes.

Motivation of this tree also comes from splay tree, a tree which stores the most recently visited node as the root node so that next timeif we want to access the same node again we will be able to reach that node very fast so that tree just keep the record of last visited node but what about keeping the record of not only last node but history of all time operations to make best decision while choosing a node.

3. PROPOSED WORK

A node of a tree contains the following information:

Table 1: Node structure

Left child link	Value	Popularity factor	Parent node link	Previous node link	Next node link	Right child link

Various operations of popularity tree which would be implemented in this research are:

a) Insertion of a new node which will take place at the last position of the tree as because popularity factor of a new node in the staring is zero.
b) Traversing involves Breadth first search traversal which will show all the nodes in decreasing order of popularity.
c) Finding a node involves
   i) First searching that node in the tree.
   ii) Print it.
   iii) Increasing its popularity factor by 1.
   iv) Check its previous node popularity factor value. If popularity factor of current node becomes greater than previous node popularity factor then exchange two nodes.

4. ALGORITHM FOR VARIOUS OPERATIONS

a) Insert(int)

i. if AVAIL==NULL then
Write: OVERFLOW and Exit.

ii. Setnewn= AVAIL and AVAIL=Left[AVAIL]

iii. Set newn[value]=data
newn[pfactor]=0
newn[left]=NULL
newn[right]=NULL
newn[parent]=NULL
newn[previous]=NULL
newn[next]=NULL

iv. If root == NULL
Set root =newn and EXIT.

v. Set front=1, rear=1 and queue[rear]=root.

vi. Repeat while queue[front]!=NULL
- Set ptr=queue[front]
- ifptr[left] == NULL then

Set ptr[left]=newn
Newn[parent]=ptr
Newn[previous]=queue[rear]
queue[rear][next]=newn and EXIT.
- if ptr[right] == NULL then

Set ptr[right]=newn
Newn[parent]=ptr
Newn[previous]=ptr[left]
Ptr[left[next]]=newn and EXIT.
- rear++

queue[rear]=ptr[left]
rear++
queue[rear]=ptr[right]
front++

b) find(int)

i) if root == NULL
Set loc=NULL and
Write: Tree empty and Exit.

ii) if root[value]==data
Set loc=root
loc[pfactor]++
Write: node found at root
And goto step (vi)

iii) top++
stack[top]=root[right]
ptr=root[left]

iv) Repeat while top!=0 || ptr!=NULL
a) if ptr!=NULL
- if ptr[value]==data then
Setloc=ptr
loc[pfactor]++

Write: node found
And goto step (vi)
- if ptr[right]!=NULL then
Settop++;
stack[top]=ptr[right]
- Set ptr=ptr[left]

b) Set ptr=stack[top];
top- -

v)Write: node not found and Exit.

vi) Repeat while loc[previous]!=NULL
- ifloc[pfactor] >loc[previous[pfactor]]

thencall Function
exchange(loc[previous],loc) and Exit.

c) Exchange(treeNode*node1, treeNode *node2)

i) Set t[right]=node2[right]
ii) if node1== root then
Set node2[right]=root->right
node1[right]=t->right
node1[left]=node2[left]
node2[left]=node1
node1[left[parent]=node1
node1[right[parent]]=node1
node2[left[parent]]=node2
node2[right[parent]]=node2
node1[parent]= node2
node2[parent]=NULL
node1[previous]= node2
node2[previous]= NULL
node1[next]=node2[next]
node1[next[previous]]=node1
node2[next]=node1

node2[next[previous]]=node2
root=node2 and Exit.

iii) Set t1[left]=node1[left]
t1[right]=node1[right]
node1[left]=node2[left]
node1[right]=node2[right]
node2[left]=t1[left]
node2[right]=t1[right]
node1[left[parent]]=node1
node1[right[parent]]=node1
node2[left[parent]]=node2
node2[right[parent]]=node2

iv) if node1[parent]==node2[parent] then
Set ptr=node1[parent]
ptr[left]=node2
ptr[right]=node1
andgoto step (vi)

v) Set node1[parent[right]=node2
node2[parent[left]]=node1
tempn=node1[parent]
node1[parent]=node2[parent]
node2[parent]=tempn

vi) Set
node2[previous]=node1[previous]
node1[previous]=node2
node1[next]=node2[next]
node2[next]=node1
node2[previous[next]]=node2
node1[next[previous]]=node1

d) bfs()

i) if root==NULL then
Write: No node exist in the tree and Exit.

ii) Set front=rear=1
queue[rear]=root

iii) Write: various details of node.

iv) Repeat while queue[front]!=NULL then
Set ptr=queue[front]
front++
Write: various details of node.
rear++;
queue[rear]=ptr->left;
rear++;
queue[rear]=ptr->right;

e) **draw()**

Repeat while l<level

i) max=pow(2,l);
ii) diff=60/(2*max);
iii) for(i=1;i<=max;i++)
iv) if(m<count)
cout<<setw(diff)<<nlist[m];
m++;
cout<<setw(diff)<<" ";
else
break;
v) l++;

f) **level()**

i) ptr=root
ii) Repeat while ptr!=NULL
Set ptr=ptr->left;
c++;

5. RESULTS & ANALYSIS

Perform operations on tree as per given sequence:

i. Insert 10:
As Node 10 is the first node to be inserted in the tree so it becomes the root node.
ii. Insert 20:
Node 20 will be inserted as left child of root Node i.e Node 10.
iii. Insert 30:
Node 30 will be inserted as right child of root Node i.e Node 10.
iv. Insert 40:
Node 40 will be inserted as left child of Node 20.
v. Insert 50:
Node 50 will be inserted as right child of Node 20.
vi. Draw the Tree:
After all the insertions in the tree it will look like as shown in Fig: 2

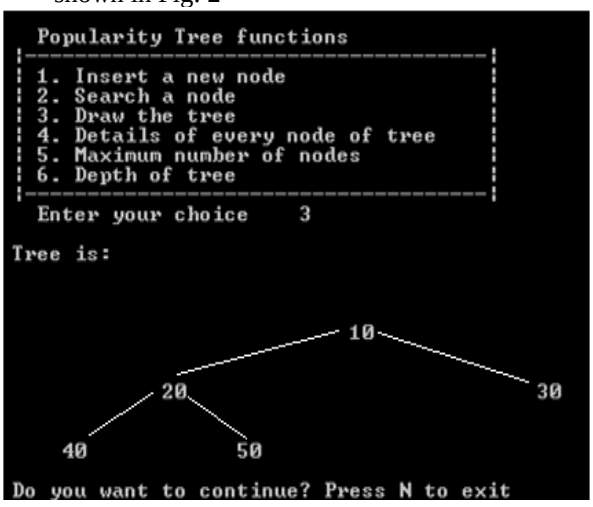

Fig. 2: Draw the Tree

vii. Details of every node:
This operation gives the complete detail of every node of the tree as shown in Fig: 3

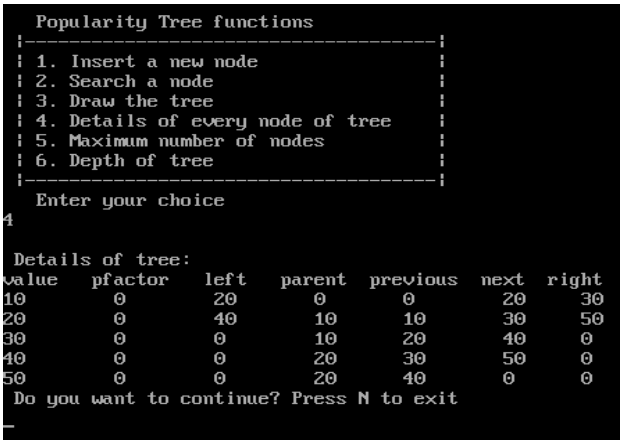

Fig. 3: Details of Tree

viii. Search Node 40:
After applying the search node 40 its popularity factor will increase from 0 to 1 and it will become the root node of the tree because of highest popularity factor according to proposed algorithm.

ix. Search Node 30:
After searching node 30 its popularity factor will increase from 0 to 1 and it will become the left child node of root node of the tree because of equal popularity factor as the root node according to proposed algorithm.

x. Search Node 30 again:
After searching node 30 again its popularity factor will increase from 1 to 2 and it will become the root node of the tree because of highest popularity factoraccording to proposed algorithm.

xi. Draw the Tree:
After all the operations in the tree it will look like as shown in Fig: 4

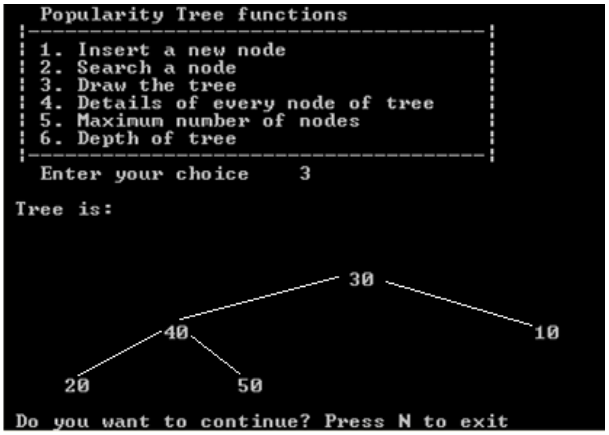

Fig. 4: Draw the Tree

xii. Details of every node:
This operation gives the complete detail of every node of the tree as shown in Fig: 5

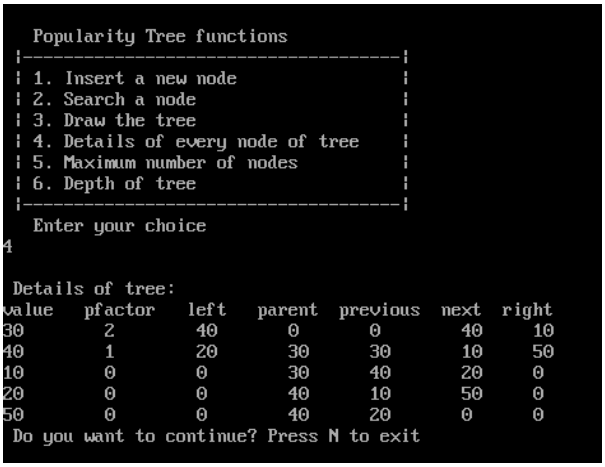

Fig. 5: Details of Tree

REFERENCES

[1] Akram Ai-Rawi, AzzedineLansari, FaouziBouslama "A NEW NON-RECURSIVE ALGORITHM FOR BINARY SEARCH TREE TRAVERSAL"

[2] C. Euis, "CONCURRENT SEARCH AND INSERTION IN AVL TREES" IEEE *Trans. Comput*.,Vol. C-29, Pp. 811-817, Sept. 1980.

[3] Daniel Dominic Sleator and Robert EndreTarjan "SELF-ADJUSTING BINARY SEARCH TREES" *Journal of the Association for Computing Machinery*. Vol. 32, No. 3, July 1985, pp. 652-686.

[4] E. Haq, Y. Cheng and S. S. Iyengar "NEW ALGORITHMS FOR BALANCING BINARY SEARCH TREES"*IEEE*, June 1988.

[5] FabrizioLuccio and Linda Pagli "REBALANCING HEIGHT BALANCED TREES"*IEEE Transactions on Computers*, Vol. C-27, No. 5, May 1978

[6] Ravi TejaCheruku "SPLAY TREE" *Indiana State University*

[7] UdiManber "CONCURRENT MAINTENANCE OF BINARY SEARCH TREES" *IEEE Transactions on Software Engineering*, Vol. Se-10, No. 6, November 1984

[8] Kiran Jain, PriyankaSehra "POPULARITY TREE, DATA STRUCTURE" *International Journal of Computer Science and Communication* (ISSN 0973-7391), Volume-IV,Number-I of March 2013.