IJCSC 0973-7391 Volume 7 • Number 2 March 2016 - Sept 2016 pp. 197-196 available online at www.csjournalss.com

Graph Storage Optimization

Sandeep Kaur[1], Er. Navjot Kaur[2],
[1]Student M.Tech(CSE), [2]Assistant professor
Department of Computer Science and Engg, Sri Sai College of Engg & Tech, Amritsar.
sandeep,hundal1@gmail.com[1], navjot_bhullar_88@yahoo.co.in[2]

Abstract: Graph is a data structure widely used now a day in different fields of computer science. It represents the relationship between one and more nodes. With the growth of internet technology social networking websites are also increased and to store the information about friends it require a graph. The intent of this paper is to propose a solution to optimize the storage of graph without using adjacency matrix. Efficient mapping of graph in the binary tree with some constraints is its goal.
General terms: Adjacency matrix, Treaps, Binary tree.
Keywords: Storage optimization, Data Structure.

I. INTRODUCTION

With the growth of technology application of graph data structure is also grow and now a day there is no any area in computer science where graphs are not used .Graphs are widely used in data mining, social networking sites, scheduling the processes by operating system, in networking for finding shortest path and lot more. So it needed to optimize the storage of graph by using efficient data structures. The data structure commonly known for storing graph is adjacency matrix which is a squared matrix of n*n size for storing n nodes in the graph. Entry in matrix is 1 if there is an edge between the nodes and is 0 if no edge is present. It take very large space in main memory and database for storing a large graphs of thousands of nodes as these are present currently in the social networking sites. In social networking sites a node is linked with hundreds of other nodes and has no relation with thousands of nodes so it is wastage of memory to store the 0's for no link present. And there is also lot more problems with adjacency matrix such that insertion in static arrays requires changing the size of array dynamically. Deletion requires moving the all elements to shrink the array. Then there is a data structure treaps is used for optimizing the graph storage by mapping the graph into new data structure name as treaps. It is a random binary search tree which is a binary search tree for a key value and heap for other priority value. In this also a problem exist if mapping the complete tree using this data structure then it takes more space than adjacency matrix also. It is specially designed for social networking sites in which it is always true that the probability of complete graph is negligible. So a new approach is proposed in which graph mapped into a binary tree which reduces space almost half of the treaps. Basically treaps removes only zeros from the adjacency matrix but also require more space to store the nodes.

II. PROPOSED TECHNIQUE

This paper proposed technique to store and map the graph into a binary tree which optimize the space required for storage of graph for large friendship networks exist in social networking sites as well as a network of railway, roads and structure of chemical compounds.

Algorithm for Insertion

Input Data

1. Key value of node as 'key'.
2. Priority of node as 'pr'.
3. Root node as 'root'.
4. Adjacent key as 'adj' =adjacent[i] where I approaches from 0 to number of adjacent nodes to that key.

1. Set new->key=key;

2. Set new->pr=pr;

3. If root=NULL then

4. Set root=new;

5. Return;

6. Set ptr=root;

```
7. If(adj<key)
8.  Repeat while ptr!=NULL do
9.  par= search(adj); //search adjacent key location
10. if(par->left=NULL)
11.  par->left=new;
12.  return;
13. else
14.  if(par->right=NULL)
15.  par->right=new;
16.  return;
17.  else
18.  adj=adj[i+1];
19.  goto step 7;
20.else
21.  par=searchRight(); //search any key which do not have right child
22. par->right=new;
23. return;
```

III IMPLEMENTATION DETAIL

Let G be a graph having V a set of vertices and E a set of edges. Consider a graph shown in figure 1.

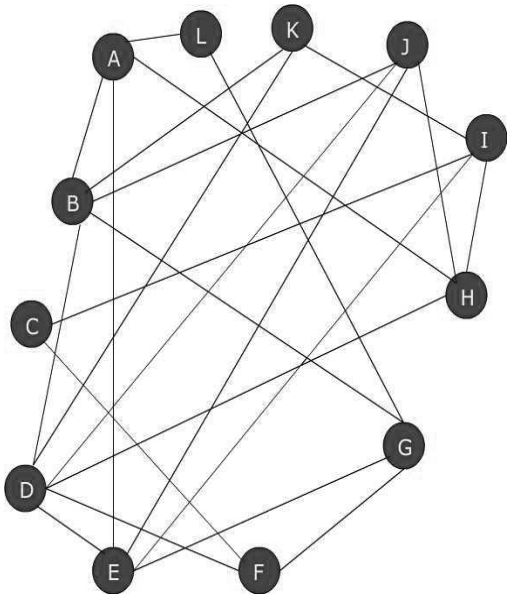

Figure 1.Graph

The corresponding adjacency matrix for above graph is shown in figure 2 .It shows as there are 12 nodes in V there needs to 12*12 matrix and which is a static array very difficult to insert new node and delete the existing one.

0	1	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	1	0	0	1	1	0
0	0	0	0	0	1	0	0	1	0	0	0
0	1	0	0	1	1	0	1	0	1	1	0
1	0	0	1	0	0	1	0	1	1	0	0
0	0	1	1	0	0	1	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0	0	1
1	0	0	1	0	0	0	0	1	1	0	0
0	0	1	0	1	0	0	1	0	0	1	0
0	1	0	1	1	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0

Figure 2. Adjacency Matrix

STRUCTURE OF PROPOSED NODE

Each node store the information of key , its priority, pointer to the array of adjacent nodes, pointer to the sibling node, pointer to the left child and right child. As shown in figure 3.

IJCSC 0973-7391 Volume 7 • Number 2 March 2016 - Sept 2016 pp. 197-196 available online at www.csjournalss.com

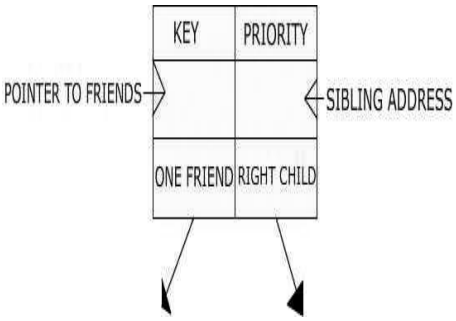

Figure 3. Node structure

Table 1 shows priority of different nodes and adjacent nodes to it.

A	B	C	D	E	F	G	H	I	J	K	L
4	5	2	6	5	3	4	4	4	4	3	2
B ,E , H ,L	A, D, G, J, K	F , I	B, E, F, H, J, K	A, D, G, I,J	C , D , G	B , E ,F , L	A , D ,I ,J	C ,E , H , K	B , D ,E , H	B , D , J	A , G

Table 1.

INSERTION OF NODES

From Table 1 key value is 'A' and priority is 4. So as root is Null so 'A' inserted in the root as shown below.

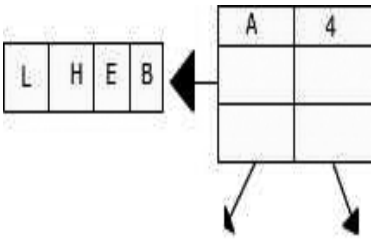

Figure 4.

Next node is B and is friend of A which have available left pointer NULL so it will be left of A and B deleted from list of A as shown below.

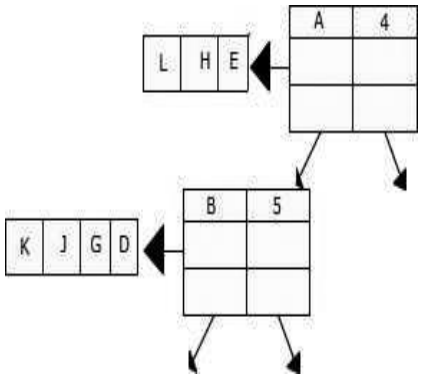

Figure 5.

Now next node is C have friends which have higher key value so it can be in right of A.

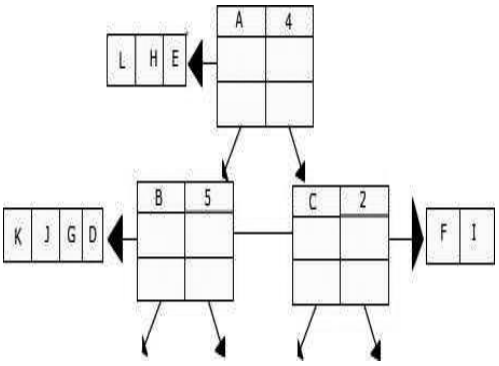

Figure 6.

Similarly all the nodes inserted in the tree the conditions are only as following:

1. If a node has to insert and root is empty then insert it on the root.
2. Tree is Min heap with key values Minimum value of key is at the top and others are at the bottom, it also helps in searching the node.
3. If a node has friend have all child then check for next friend else it can be inserted first in left then in right.
4. For checking the list of friend first go to list of adjacent friends, if it has less number of friends then priority, than check its left child, if it is present then it also its friend. If again it is less in number than go check whether it is left of its parent, if it is than parent is also a friend, if again the number of friends is less then priority then check its right friend. Calculation of priority and finding the friends in the tree is as follows:

1. *Priority = Number of elements in adjacent friend list of node.*

If it is less then,

2. *Priority = Number of friends in adjacent list + Left child.*

If again it is less then,

3. *Priority = Number of friends in adjacent list + Left child + Parent node (if node is left child of its parent).*

If again it is less,

4. *Priority = Number of friends in adjacent list + Left child + Parent node (if node is left child of its parent) + Right child.*

Then final tree of figure 1 graph is mapped as shown in figure 7.

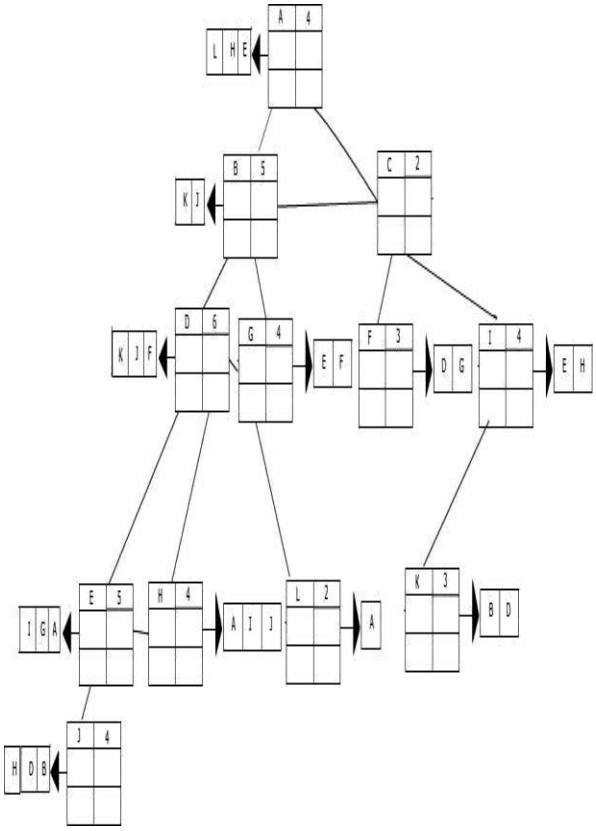

Figure 7.

III. CONCLUSION

The proposed approach has promising result and required very less storage than adjacency matrix as well as treaps for large graphs. The technique implemented in this paper is also easy to implement and tree can be scaled for a large network. The proposed solution tremendously reduces the storage requirement. For above graph in figure 1 where adjacency matrix require 144 bytes for storage proposed approach require only 97 bytes. The given solution can be used for any network or graph application such as railway network, routing network, chemical compounds etc.

REFERENCES:

[1] Deepak Garg and MeghaTyagi 2012:"Comparative analysis of dynamic graph techniques and data structure";IJCA 45-5.

[2] S.G. Shirinivas 2010: " Applications of graph theory in computer science an overview"; IJEST Vol.2(9).

[3] R.Seidel and C.R. Aragon. 1996;"Randomized search trees";Algorithmica,16:464-497.

[4] Dharya Arora and ShaliniBatra;" Using treaps for optimization of graph storage" ,IJCA vol. 41-no. 14.

[5] Chris Lattner: "Heap Data Structure Analysis and Optimization ", Ph.D. Thesis.

[6] Day, A. C. 1976, "Balancing a Binary Tree, Computer Journal",19,360-361.

[7] Vinod Prasad 2011 "A Forest of Hashed Binary Search Trees with Reduced Internal Path Length and better Compatibility with the Concurrent Environment";IJCAvol 33-n0. 10.