Using Treaps for Optimization of Graph Storage

Dharya Arora
Student (ME)
Department of CSE
Thapar University
Patiala, India

Shalini Batra
Assistant Professor
Department of CSE
Thapar University
Patiala, India

ABSTRACT
Adjacency matrix is an effective technique used to represent a graph or a Social network comprising of large number of vertices and edges. The intent is of this paper is to optimize the graph storage and mapping without using a large adjacency matrix to represent a large graph. A special data structure Treap, a combination of binary search tree and heaps has been used as a replacement to a large adjacency matrix. It has been experimentally evaluated that the proposed approach significantly improves the space occupied by adjacency matrix and helps the graph to grow dynamically without affecting the current data structure.

General Terms
Treaps, Adjacency matrix

Keywords
Storage optimization, Graph mapping, Treap data structure, Adjacency matrix

1. INTRODUCTION
The enormous growth of Internet and the development of new applications and services have dramatically increased the number of users, resulting in increased storage size [14]. Although the size of the social networks has grown exponentially, no standard method has been designed for efficient mapping of a graph or social network onto a compatible data structure. As internet in itself is a social network graph comprises of nodes and edges so with the increased size of graph day by day directly affect the storage being used such as adjacency matrix [1, 11]. Data structures used for adjacency matrices usually have two components: (*i*) an array that stores all the entries of the matrix, (*ii*) pointer to an array which would take care of increased size of entries in it [2].

Major problem associated with the use of adjacency matrix is that it has static array allocation and fixed entries for increased size of network creates problem during insertion. Secondly, dynamic array allocation requires more time to create a new array of increase size and then move the entries from previous array to new array and then deallocating the previous array [19, 20]. Finally, although the adjacency matrix for a graph is sparse, every null entry would take space. For dynamically increased network and efficient storage optimization [3, 10, 16, 17] this paper proposes the use of Treaps to store a graph with its mapping information in an efficient manner [4, 15, 18].

1.1 Treaps
A *treap* is a binary search tree in which each node has both a key and a priority [22, 23, 24]. Nodes are ordered in an in-order fashion by their keys and are heap-ordered by their priorities [5, 21]. The idea behind Treaps is to use randomness, to balance binary search trees [6, 13]. The Treap and the randomized binary search tree are two closely related forms of binary search tree data structures that maintain a dynamic set of ordered keys and allow binary searches among the keys. After any sequence of insertions and deletions of keys, the shape of the tree is a random variable with the same probability distribution as a random binary tree; in particular, with high probability its height is proportional to the logarithm of the number of keys, so that each search, insertion, or deletion operation takes logarithmic time to perform [7, 9].

2. PROPOSED DATA STRUCTURE: TREAPS
This paper proposes creation of Treaps for storing and mapping graph which can be a friendship network, a railway network, a chemical compound structure etc [12]. Various operations in a Treap can be performed such as insert new node, delete a node, traverse a node.

2.1 Algorithm for insertion of a node
To insert a node in a Treap:

Input:
- Key value of node as 'key'
- Priority of node as 'pr'
- Root node of Treap as 'root'

Output:
- Insert the new node to its appropriate position by treap rotations.

```
1.  Set new ← Getnode() // Create an empty new node
2.  Set new→key=key
3.  Set new→pr=pr
4.  if root = NULL then
5.          Set root=new
6.          return
7.  Set ptr=root
8.          Repeat  while ptr ≠ NULL do
9.                  Set prev=ptr
10.                 if new→key > ptr→key then
11.                    ptr=ptr→right
12.                 side=right
13.                 else
14.                  ptr=ptr→left
15.                  side=left
16. Set prev→side=new
17. Set ptr=new→parent
18. Repeat while new→pr < ptr→pr do
19.         temp=ptr
20.         ptr=new
21.         if temp→key < ptr→key then
22.           prev=ptr→left
```

```
23.              ptr→left=temp
24.              if prev→key < temp→key then
25.                       temp→left=prev
26.              else
27.                       temp→right=prev
28.           else
29.              prev=ptr→right
30.              ptr→right=temp
31.              if prev→key < temp→key then
32.                       temp→left=prev
33.              else
34.                       temp→right=prev
35. go to step 16 until new→key < new→parent→key
36. return
```

Algorithm 1: Insert a node in a Treap

3. IMPLEMENTATION DETAILS

Let G be a graph comprises of two main components (V, E) where V contains a set of vertices of graph and E contains a set of edges of graph [8]. To store the whole graph corresponds to its mapping information; consider a graph having few nodes as shown in figure 1;

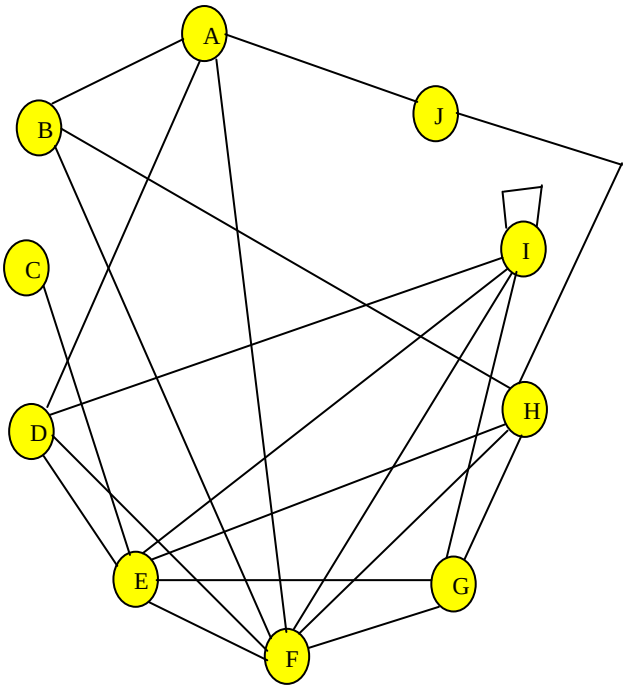

Figure 1: A Random Graph

The corresponding adjacency matrix for the above graph as shown in figure 1 would be given as in figure 2. If the given adjacency matrix is used to store and map the graph information then it is clear from the figure 2 that the storage schema used is not scalable for dynamic graph approach. But as the applications are increasing day by day where graph has a strong impact to represent a problem domain, in that case using static allocation pays no attention and cause problems during dynamic allocation.

To implement the concept, for easy understanding of graph consider its corresponding adjacency matrix so that it can be easy to understand how a graph can be stored and mapped to a Treap.

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 2: Adjacency matrix of a graph

This is an adjacency matrix corresponding to a graph of 10 nodes. Each row of matrix depicts the adjacency information of each node of graph respectively. Since Treap require a key and a priority, key is taken from the unique name of a node and priority would be the no. of links corresponding to a each node i.e. no. of 1's to its row is its priority.

Table 1: Adjacency table

A	B	C	D	E	F	G	H	I	J
4	3	1	4	6	7	4	5	5	2

3.1.1 Structure of a node

Each node of Treap has a specific internal representation. Each node will store the information about its key and priority, pointers to left and right child and a pointer to an array containing the name of its adjacent node such as in figure 3:

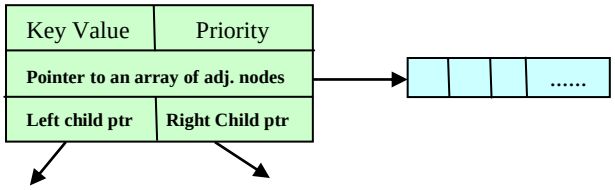

Figure 3: Node representation of a Treap

3.1.2 Insertion of node

From Table 1, key value would be 'A' and its priority is '4' so initially there is no node in the Treap so new node inserted in itself act as a root of the Treap such as:

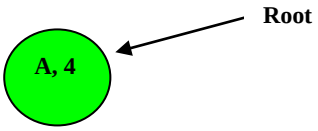

Figure 4: First node of Treap

When another node from table 1, is inserted then the Treap would be:

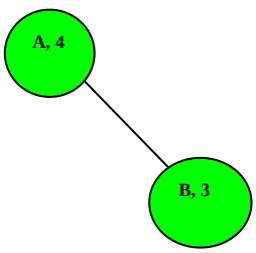

Figure 5: Insertion after new node

Nodes are inserted according to BST based on their key value. Here A is less and B is more so more value node comes at right child. Now according to Treap rule, rotation would takes place because priority of B is less than A and min heap is being considered.

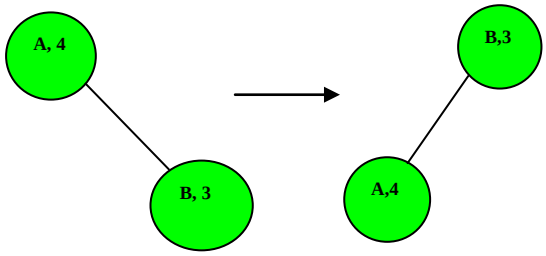

Figure 6: Treap after Rotation

Now B becomes the root of the Treap and A comes at left to B because key value is less so according of BST rule, it becomes left child. After inserting another node from table 1, structure of Treap would be shown as in Figure 7.

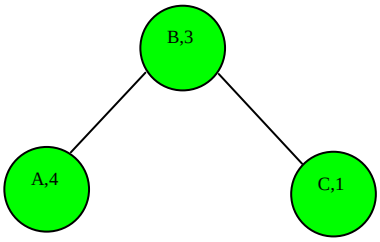

Figure 7: Treap after insertion of new node

Again the same rotation would take place because priority of C is less than B so after rotation treap formed would be shown as in Figure 8:

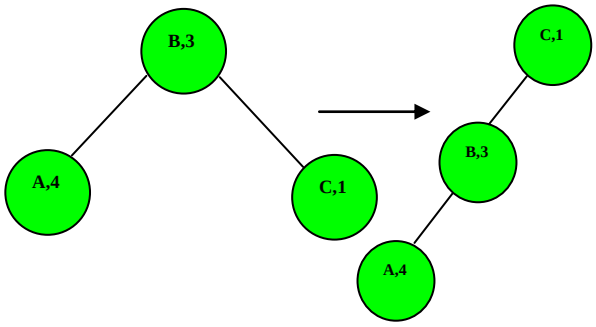

Figure 8: Treap after rotation

Final structure of the Treap after insertion of series of nodes from table 1, one by one would be shown in Figure 9:

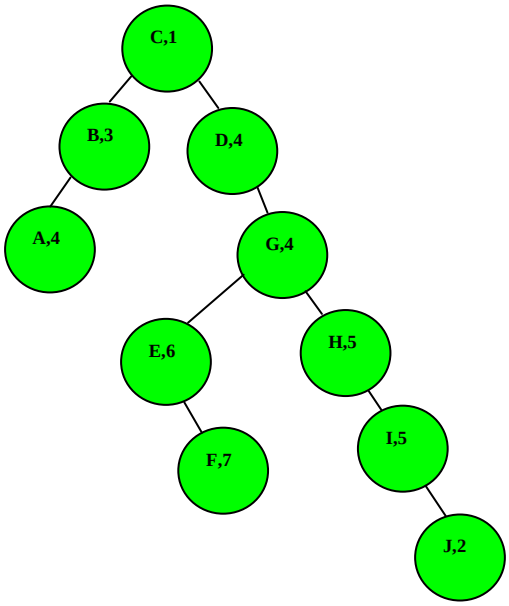

Figure 9: Treap after insertion of last node

Since the newly inserted node makes the Treap unbalanced as the priority of J is less than to its parent node as well as to its ancestor nodes, so after performing a series of transformations the final result which will be achieved is shown in Figure 10.

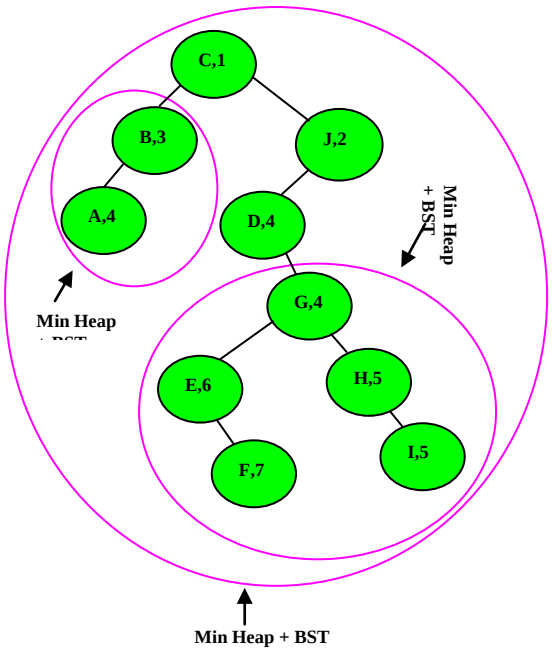

Figure 10: Final output

As seen in Figure 10, every sub-tree and sub-sub-tree in itself satisfy the property of Treaps. Thus the graph which would be actually stored in adjacency matrix can now be stored in Treaps where it is easy to modify and scalable to large networks which adjacency matrix fails for storing graphs of dynamic nature.

4. CONCLUSIONS

The current work done has a strong impact on the storage behavior of social network or graph. The approach used here gives promising results and the storage schema in itself has a

strong implementation impact for optimization of space and query time. The technique discussed in this paper is easy to implement and scalable to a large network. By using Treaps the space used for storing a graph, as compared to an adjacency matrix, is tremendously decreased because of the use of linked list and pointers. Above all this technique can work for other social network when there are less parameters such as transport network, railway network, chemical compounds, etc.

5. REFERENCES

[1] J.B. Shearer and R. J. McEliece. 1977: "There is no Mac Williams identity for convolutional codes"; IEEE Trans. Inform. Theory, IT-23:775-776.

[2] Bernard Elspas and James Turner. 1970: "Graphs with circulant adjacency matrices", 297

[3] A. Cohen and V. Lefebvre. 1988: "Optimization of storage mappings for graphs" Technical Report 1998/46, PRiSM, U. of Versailles.

[4] H. Orsila, E. Salminen, M. H¨annik¨ainen, T.D. H¨am¨al¨ainen. 2007: "Optimal Subset Mapping And Convergence Evaluation of Mapping Algorithms for Distributing Task Graphs on Multiprocessor"; SoC, Symposium on SoC.

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

[6] R. W. Irving and L. Love. 2003: "The suffix binary search tree and suffix AVL tree"; J. Discrete Algorithms, 1(5-6):387-408.

[7] G. D. Forney and M. D. Trott. 2004: "The Dynamics of Group Codes: Dual Abelian Group Codes and Systems"; IEEE Trans. Inform. Theory, IT-50:2935-2965.

[8] A. A. Nanavati, S. Gurumurthy, G. Das, D. Chakraborty, K. Dasgupta, S. Mukherjee, A. Joshi. 2006: "On the structural properties of massive telecom call graphs: findings and implications"; In CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management, New York, NY, USA, ACM, 435–444

[9] M. Newman, A.-L. Barabasi, D. J. Watts. 2006: "The Structure and Dynamics of Networks"

[10] G. Pike. 2002; "Reordering and Storage Optimizations for graphs"; PhD thesis, University of California, Berkeley.

[11] Frank O. 1981; "A Survey of Statistical Methods for Graph Analysis. Sociological Methodology", 110-155.

.

[12] Brewer, D.D., Webster, C.M.. 1999; "Forgetting of friends and its effects on measuring friendship networks. Social Networks" 21, 361–373.

[13] Pattison, P.E., Robins, G.L. 2002; "Neighbourhood-based models for social networks. Sociological Methodology" 32, 301–337.

[14] Watts, D.J. 1999; "Small Worlds: The Dynamics of Networks between Order and Randomness"; Princeton University Press, Princeton, NJ.

[15] Handcock, M.S., Hunter, D.R., Butts, C.T., Goodreau, S.M., Morris, M. 2008. Statnet: "Software Tools for the Representation, Visualization, Analysis and Simulation of Network Data"; Journal of Statistical Software 24 (1), URL, http://www.jstatsoft.org/v24/i01.

[16] A. Cohen. 1999; "Parallelization via constrained storage mapping optimization";

[17] A. Cohen and V. Lefebvre. 1988; "Optimization of storage mappings for parallel programs"; Technical Report 1998/46, PRiSM, U. of Versailles.

[18] P. Feautrier. 2001; "The use of farkas lemma in memory optimization".

[19] A. W. Lim, S.-W. 2001; "Liao, and M. S. Lam. Blocking and array contraction across arbitrarily nested loops using affine partitioning"; In Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming, pages 103-112, ACM Press.

[20] W. Thies, F. Vivien, J. Sheldon, and S. Amarasinghe. 2001; "A unified framework for schedule and storage optimization"; In Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation, pages 232-242, ACM Press.

[21] Blelloch, Guy E, Reid-Miller, Margaret, (1998), "Fast set operations using Treaps", Proc. 10th ACM Symp. Parallel Algorithms and Architectures (SPAA 1998), New York, NY, USA: ACM, pp. 16–26.

[22] Martinez, Conrado, Roura, Salvador. 1997, "Randomized binary search trees", Journal of the ACM 45 (2): 288–323

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

[24] D. D. Sleator and R. E. Tarjan. 1985; "Self-adjusting binary trees"; Journal of the Association for Computing Machinery,