

# **Adaptive Probabilistic Skip Graph**

*Thesis submitted in partial fulfillment of the requirements for the  
award of degree of*

**Master of Engineering**  
in  
**Computer Science and Engineering**

*Submitted By*  
**Amit Goyal**  
**(Roll No. 801332002)**

Under the supervision of:  
**Dr. Shalini Batra**  
Assistant Professor, CSED



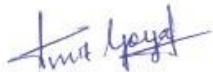
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT  
THAPAR UNIVERSITY  
PATIALA – 147004

**July 2015**

## Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Adaptive Probabilistic Skip Graph*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Shalini Batra* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

  
(Amit Goyal)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

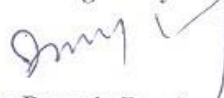
  
(Dr. Shalini Batra)

Assistant Professor

Computer Science and Engineering Department

Thapar University, Patiala

Countersigned by

  
(Dr. Deepak Garg)

Head

Computer Science and Engineering Department

Thapar University

Patiala

  
(Dr. S.S. Bhatia)

Dean (Academic Affairs)

Thapar University

Patiala

## Acknowledgement

---

No volume of words is enough to express my gratitude towards my guide, **Dr. Shalini Batra**, Assistant Professor, Computer Science and Engineering Department, Thapar University, Patiala, who has been very concerned and has aided for all the materials essential for the preparation of this thesis report. She has helped me to explore this vast field in an organized manner and provided me with all the ideas on how to work towards a research oriented venture.

I am also thankful to **Dr. Deepak Garg**, Head of Computer Science and Engineering Department and **Mr. Ashutosh Mishra**, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there in the need of the hour and provided with all the help and facilities, which I required, for the completion of my thesis work.

Most importantly, I would like to thank my parents, friends and the almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

**Amit Goyal**  
**(801332002)**

Mining of massive data sets is the need of the hour in present computer science industry. The exponential growth in the number of users on internet and volume of available data has lead to the invention of variants of the algorithms and data structures traditionally used in peer to peer networks. Many data structures like adjacency matrix, skip webs, hash tables, skip lists and skip graphs have been proposed to represent the peer to peer networks. The data structure to be used depends upon the requirements of the area where it is being implemented. Available data structures do not seem to be quite efficient in handling the current volume of big data. There is an imminent requirement of data structures which are dynamic and adaptive to the current trends of data.

This thesis explores usage of one of these data structures, skip graph, a variant of skip list for peer to peer networks. The existing algorithm for searching in skip graph has  $O(\log n)$  time complexity which can be further decreased by reforming the present structure as discussed in the proposed approach named as Adaptive Probabilistic Skip Graph. Modifications have been proposed in the scenario where a node is repeatedly being queried by a certain node and it has been experimentally verified that in such scenarios, the search time reduces drastically. The major focus of the thesis has been on optimizing the search algorithm by adding probability vector in the basic structure of the skip graph node.

## Table of Contents

---

|                                                                      |                                     |
|----------------------------------------------------------------------|-------------------------------------|
| Certificate.....                                                     | <b>Error! Bookmark not defined.</b> |
| Acknowledgement .....                                                | ii                                  |
| Abstract .....                                                       | iii                                 |
| Table of Contents .....                                              | iv                                  |
| List of Figures .....                                                | vii                                 |
| List of Tables .....                                                 | viii                                |
| Chapter 1 Introduction .....                                         | 1                                   |
| 1.1    Types of Distributed Systems .....                            | 1                                   |
| 1.1.1    Client Server Architecture .....                            | 1                                   |
| 1.1.2    3-Tier Architecture.....                                    | 2                                   |
| 1.1.3    Multitier Architecture .....                                | 2                                   |
| 1.1.4    Peer to Peer Architecture .....                             | 3                                   |
| 1.2    Peer to Peer Network.....                                     | 3                                   |
| 1.3    Data Structures Used to Represent Peer to Peer Networks ..... | 4                                   |
| 1.3.1    Adjacency Matrix.....                                       | 5                                   |
| 1.3.2    Adjacency List .....                                        | 6                                   |
| 1.3.3    Skip Lists .....                                            | 7                                   |
| 1.3.4    Skip Graphs.....                                            | 8                                   |
| 1.4    Structure of The Thesis .....                                 | 9                                   |
| Chapter 2 Literature Review .....                                    | 11                                  |
| Chapter 3 Problem Statement .....                                    | 17                                  |
| 3.1 Research Gaps .....                                              | 17                                  |
| 3.2 Problem Statement .....                                          | 17                                  |
| 3.3 Objectives.....                                                  | 18                                  |
| 3.4 Methodology .....                                                | 18                                  |
| Chapter 4 Design of Adaptive Probabilistic Skip Graph .....          | 19                                  |
| 4.1 Existing Skip Graph Data Structure .....                         | 19                                  |

|                                                                         |    |
|-------------------------------------------------------------------------|----|
| 4.1.1 Structure of Skip Graph Node .....                                | 19 |
| 4.1.2 Insertion in Skip Graph.....                                      | 21 |
| 4.1.3 Deletion in Skip Graph .....                                      | 26 |
| 4.1.4 Search in Skip Graph.....                                         | 27 |
| 4.2 Adaptive Probabilistic Skip Graph.....                              | 28 |
| 4.2.1 Structure of Adaptive Probabilistic Skip Graph Node .....         | 29 |
| 4.2.2 Insertion and Deletion in Adaptive Probabilistic Skip Graph ..... | 30 |
| 4.2.3 Search in Adaptive Probabilistic Skip Graph .....                 | 30 |
| 4.2.4 Replacement in Adaptive Probabilistic Skip Graph .....            | 32 |
| Chapter 5 Implementation Details .....                                  | 36 |
| 5.1 Functions Defined .....                                             | 36 |
| 5.2 Results and Analysis .....                                          | 39 |
| 5.3 Comparison of Basic and Adaptive Probabilistic Skip Graph .....     | 48 |
| Chapter 6 Conclusion and Future Scope.....                              | 49 |
| 6.1 Conclusion.....                                                     | 49 |
| 6.2 Summary of Contributions .....                                      | 49 |
| 6.3 Future Scope.....                                                   | 49 |
| References.....                                                         | 51 |
| List of Publications and Video Link .....                               | 55 |
| Reflective Dairy .....                                                  | 56 |
| Appendix A: Membership Vector of 1024 nodes .....                       | 59 |

## List of Figures

---

|                                                                           |    |
|---------------------------------------------------------------------------|----|
| Figure 1.1 Server based network and client server architecture .....      | 2  |
| Figure 1.2 3-tier architecture.....                                       | 2  |
| Figure 1.3 Replication and specialization in multitier architecture ..... | 3  |
| Figure 1.4 Peer to peer network.....                                      | 4  |
| Figure 1.5 Undirected graph of 5 nodes.....                               | 5  |
| Figure 1.6 Directed graph of 5 nodes.....                                 | 5  |
| Figure 1.7 Adjacency list of undirected graph.....                        | 7  |
| Figure 1.8 Adjacency list of directed graph.....                          | 7  |
| Figure 1.9 Skip list .....                                                | 8  |
| Figure 1.10 Skip graph .....                                              | 9  |
| Figure 4.1 Structure of skip graph node.....                              | 19 |
| Figure 4.2 Skip graph having 14 nodes .....                               | 20 |
| Figure 4.3 Insertion of nodes 1 and 16 .....                              | 22 |
| Figure 4.4 Insertion of nodes 8 and 4 .....                               | 22 |
| Figure 4.5 Insertion of nodes 12 and 3 .....                              | 22 |
| Figure 4.6 Insertion of node 15.....                                      | 23 |
| Figure 4.7 Insertion of node 5.....                                       | 23 |
| Figure 4.8 Insertion of node 7.....                                       | 24 |
| Figure 4.9 Insertion of node 10.....                                      | 24 |
| Figure 4.10 Insertion of node 2.....                                      | 25 |
| Figure 4.11 Insertion of node 13.....                                     | 25 |
| Figure 4.12 Deletion of node 1 .....                                      | 26 |
| Figure 4.13 Search of node 9 by node 2 .....                              | 28 |
| Figure 4.14 Modified skip graph node.....                                 | 29 |
| Figure 5.1 Insertion of node 1.....                                       | 39 |
| Figure 5.2 Insertion of node 8.....                                       | 40 |
| Figure 5.3 Insertion of node 12.....                                      | 40 |
| Figure 5.4 Insertion of node 7.....                                       | 41 |
| Figure 5.5 Insertion of node 13.....                                      | 42 |

|                                                                                                        |    |
|--------------------------------------------------------------------------------------------------------|----|
| Figure 5.6 Deletion of node 1 .....                                                                    | 42 |
| Figure 5.7 First search of node 9 by node 2.....                                                       | 43 |
| Figure 5.8 Fourth search of node 9 by node 2 .....                                                     | 44 |
| Figure 5.9 Fifth search of node 9 by node 2 .....                                                      | 45 |
| Figure 5.10 Skip graph after 14 calls from node 2 to node 9 and 12 calls from node 3 to<br>node 1..... | 46 |
| Figure 5.11 Skip graph when least recently used replacement policy is used.....                        | 47 |
| Figure 5.12 Skip graph when least frequently used replacement policy is used .....                     | 47 |

## List of Tables

---

|                                                                                      |    |
|--------------------------------------------------------------------------------------|----|
| Table 1.1 Adjacency matrix of undirected graph.....                                  | 6  |
| Table 1.2 Adjacency matrix of directed graph.....                                    | 6  |
| Table 5.1 Membership and probability vector of node 2 when it calls node 9.....      | 43 |
| Table 5.2 Membership and probability vector of node 512 when it searches node 513... | 46 |
| Table A.1 Membership vector of each node in 1024 nodes skip graph.....               | 59 |

## Chapter 1 Introduction

---

Computing started with desktop computing where each computer could only access its own resources. Then, it shifted to mainframe computing followed by client server computing and finally, today is the era of distributed computing. The first major distributed systems were ARPANET (invented in 1960s) and ARPANET email (invented in early 1970s). Local area network like Ethernet, invented in 1970s was also a specimen of distributed systems and internet is the best example of distributed systems [1]. The term *distributed* does not necessarily mean that the computing units are physically distributed within a geographical area but they may be a part of a single computer system. The computing units are also called computers or nodes. All the nodes might have a common objective to solve a large computational problem or the nodes may be treated as individual computers which are communicating with each other and using shared resources over the internet [2]. Distributed computing is often confused with parallel computing [3]. One of the criteria which can be used to roughly distinguish them is that parallel computing allows access to shared memory among processors while distributed computing does not allow this [4][5]. In distributed systems the processors have their own memory and message passing is used to share information [6].

### 1.1 Types of Distributed Systems

Distributed systems use a variety of hardware and software architectures. It is important that the nodes of the distributed system must be connected using some sort of network and they must be able to communicate with each other by some sort of communication system. Some of the architectures proposed for distributed systems are [1]:

#### 1.1.1 Client Server Architecture

A dedicated machine called server serves the clients. Server specializes in managing data and application logic while the client specializes in user interface. Asymmetric type of relationship exists between clients and server as clients predominantly make requests while server makes replies. Clients are active initiators of transaction while servers are passive. Client server architecture is also known as 2-tier architecture in which tier 1 is client platform which hosts a web browser while tier 2 is server platform which hosts all

server software components. Email server and chat servers use client server architecture. It suffers from the drawback of having a single point of failure and hence lacks robustness. There is heavy load on database, scalability is limited and separation of software components is also very poor. Traffic congestion is also a problem and it increases with the increasing size of network.

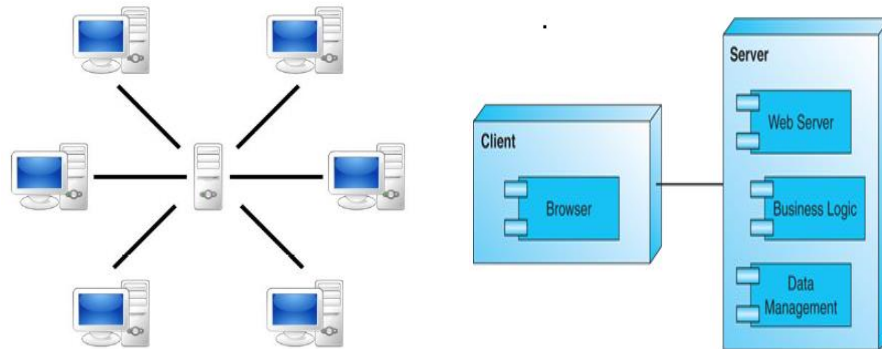


Figure 1.1 Server based network and client server architecture [7] [8]

### 1.1.2 3-Tier Architecture

In 3-tier architecture, tier 3 (middle tier) is introduced which takes over part of the tier 2 (server platform), typically management of data. In thin clients, the client intelligence may also be shifted to tier 3. It is used in small to medium business applications. It provides better scalability, improved performance and decreased coupling among server software components. The database overload on server in tier 2 is reduced drastically.

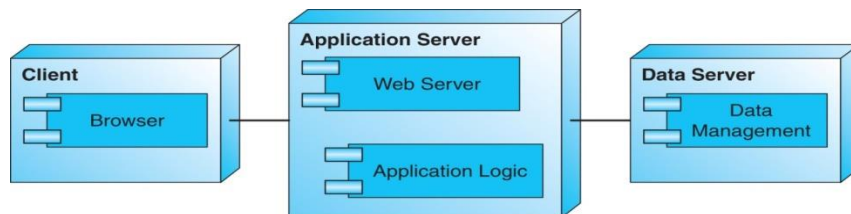


Figure 1.2 3-tier architecture [8]

### 1.1.3 Multitier Architecture

Multitier architecture or n-tier architecture is an extension of 3-tier architecture. Tiers may be introduced to replicate the function of a tier, to specialize the functionality within a tier or to provide portal services to handle incoming traffic. It provides better decoupling of the software components, greater scalability and less traffic congestion. It

can support large business organizations having thousands of users. It is also used in web services where the requests are forwarded to other enterprise services. It suffers from high design, cooling, electrical load and maintenance costs.

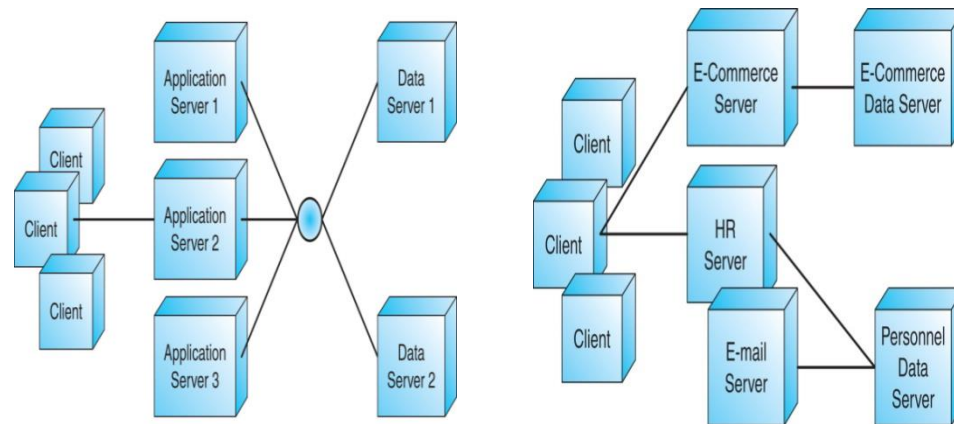


Figure 1.3 Replication and specialization in multitier architecture [8]

#### 1.1.4 Peer to Peer Architecture

In peer to peer architecture, there is no central server to provide services or to manage the network. Here the responsibilities are equally divided among all the nodes. All nodes can act as client or server and are thus peers to each other. It takes the advantage of the resources of all the nodes in the network and thus has gained huge popularity. Peer to peer network is dynamic in nature, i.e. nodes can enter or exit the system at any time. It is very robust as there is no central server to act as a single point of failure. It provides huge scalability and traffic congestion is also very less. In fact, the traffic congestion reduces after adding more nodes up to certain extent because of the added bandwidth of the new nodes. Cloud computing, web services and ad-hoc networks are some of the popular applications of peer to peer networks.

### 1.2 Peer to Peer Network

In peer to peer networks, data is stored at multiple locations and retrieved by passing messages between the nodes to find out the node which contains the required information. A few important characteristics of peer to peer network are fast and efficient query resolution, fast update, self stabilization, fault tolerance, load balancing, scalability, decentralization and better data availability than its counterparts [9]. A peer to peer network is shown in Figure 1.4.

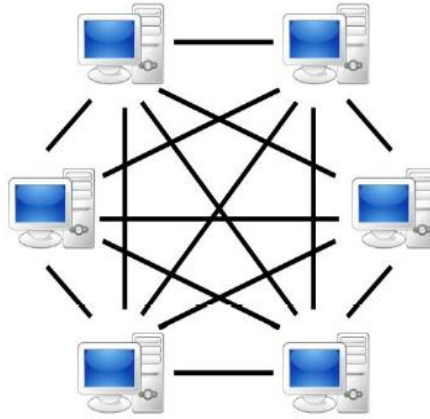


Figure 1.4 Peer to peer network [10]

Peer to peer networks use the collective power of resources of all the nodes in the network. The resources include the data, disk storage, cache storage, CPU cycles, networking connectivity, input and output devices. In peer to peer network every node does not have the information about the complete network instead it knows only a few nodes and using these nodes it can reach to any other node in the network. Hence, the nodes also behave as routers for forwarding the message from one node to other. New nodes in the system can be introduced by any of the existing nodes as there is no central server. In peer to peer networks nodes are autonomous and they all work with the same algorithm and collaborate directly with each other. As discussed in distributed systems, the nodes in peer to peer networks may be a part of the same system or may be geographically distributed.

### **1.3 Data Structures Used to Represent Peer to Peer Networks**

The data structure to be used to represent a peer to peer network must possess most of its characteristics mentioned in Section 1.2. The peer to peer network may be represented using a graph where the vertices represent the nodes and edges represent the link with the neighbors i.e. the known peers. The graph may be undirected in which the connected nodes are linked with each other in both the directions. As shown in Figure 1.5, nodes A and D are connected through a bidirectional link. All other links represented in this figure are also bidirectional.

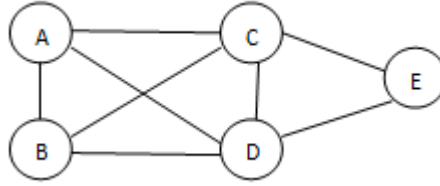


Figure 1.5 Undirected graph of 5 nodes

The degree of a vertex is the number of edges incident on it. The degree of a graph is equal to the maximum degree of its set of vertices. The sum of the degree of all vertices is always even as every edge contributes an addition of two and hence the number of odd degree vertices is always even [11]. In the above graph, degree of vertices A and B is 3, C and D is 4 and degree of vertex E is 2. The degree of the graph is 4, the maximum degree of the vertices. The sum of the degree of all vertices is 16 which is even and the number of odd degree vertices is 2 i.e. A and B.

The graph may be directed in which the nodes are connected only in the shown direction. As shown in Figure 1.6, A and D are connected through a unidirectional link *i.e.* only A is linked to D but D is not linked to A. Every vertex in directed graph has an in-degree and an out-degree representing the number of incoming and outgoing edges respectively.

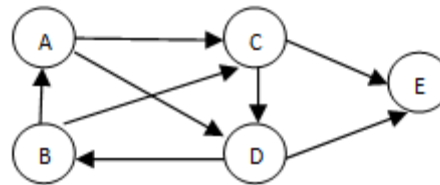


Figure 1.6 Directed graph of 5 nodes

Various data structures have been used to represent peer to peer networks. A few of them are:

### 1.3.1 Adjacency Matrix

It is a matrix of size  $n \times n$ , where  $n$  is the number of nodes in the network. When  $a_{ij}$  is 1, it represents the presence of an edge between the nodes  $i$  and  $j$ , while 0 represents its absence [11]. The adjacency matrix of all isomorphic graphs is same provided the right mapping of vertices is done. The adjacency matrix for undirected graph in Figure 1.5 is:

Table 1.1 Adjacency matrix of undirected graph

|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 1 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 1 | 1 | 0 |

In case of undirected graphs the adjacency matrix is symmetric and the same can be concluded from the above adjacency matrix. The adjacency matrix for the directed graph in Figure 1.6 is shown in Table 1.2.

Table 1.2 Adjacency matrix of directed graph

|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 0 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 0 | 0 |
| C | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 1 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

This representation is good in case of a central server to mark the connections between the different members in the network. But in case of peer to peer networks, as a node is connected to only some of its neighbors, the matrix will be very sparse and hence a huge space wastage. Also, as a node needs to store information about its few neighbors only, the storage of complete matrix at every node is not required. Adjacency matrix is a  $n \times n$  matrix, so with the growth of the network, the space complexity increases drastically by  $O(n^2)$ . Even deletion of a node is very difficult as all the subsequent rows and columns have to be moved up by one space each.

### 1.3.2 Adjacency List

To address the issue of scalability in adjacency matrix, adjacency lists are used. In adjacency list, all the vertices which are connected to a vertex are shown with a list

representation. Insertion in adjacency list is better than adjacency matrix but searching is very difficult. The adjacency list of the undirected graph in Figure 1.5 is:

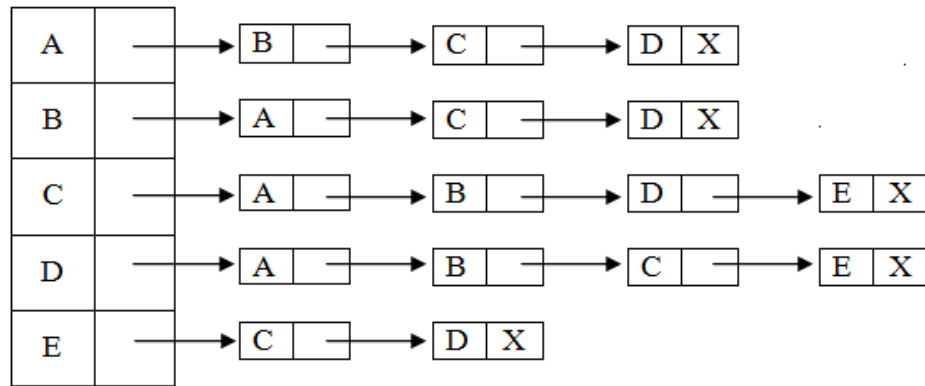


Figure 1.7 Adjacency list of undirected graph

The adjacency list of the directed graph in Figure 1.6 is:

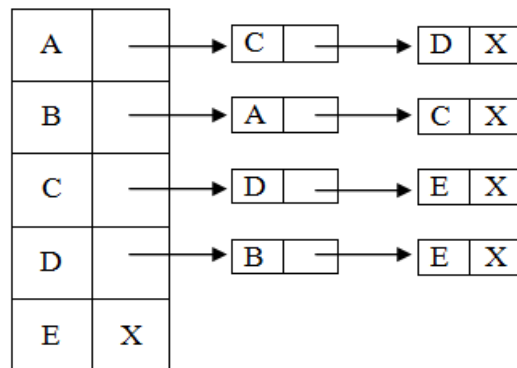


Figure 1.8 Adjacency list of directed graph

The implementation of the list may vary from application to application.

### 1.3.3 Skip Lists

Skip list, an ordered data structure is based upon the extension of linked list. It is built in layers where the last layer is ordinary doubly linked list while the layers above act as express lane for the layers below. The number of nodes decreases geometrically with the increasing levels. The maximum length of the search path for any key from the peak level of the skip list is known as height of skip list and is of the order of  $\log n$ , thus providing a search complexity of  $O(\log n)$  [12]. An example of skip list is shown in Figure 1.9.

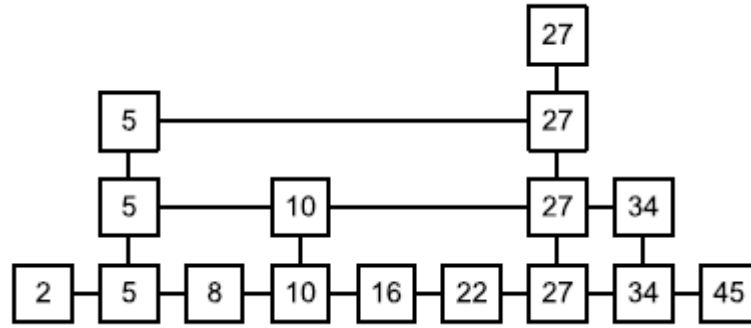


Figure 1.9 Skip list [13]

It is not fault tolerant because of the presence of a single point of failure which is the head node. Deletion of the head node from the graph leads to the loss of information regarding the starting point of search and further every node is present only in  $1/(1-p)$  lists. Moreover, skip list is not fully distributed as whenever a node wants to query in the network, it must know the address of the head node.

#### 1.3.4 Skip Graphs

It is a variant of skip list and one of the most suitable data structure for peer to peer network with search complexity of  $O(\log n)$  [14]. It provides fault tolerance [13] as all the nodes are present at all the levels. Skip graph offers various benefits like fast insertion, fast searching, efficient updates, load balancing and scalability [15][16]. Since, skip graph offers so many advantages; it is the primary focus of this thesis and efforts have been made to improve its search complexity. The major objective is to introduce probability information in the skip graph structure and modify the search algorithm to reduce the search complexity in case a node queries a certain node repeatedly. An example of skip graph is shown in the Figure 1.10. As depicted in the figure, every node is present at all the levels and the interconnection between the nodes is decided by the membership vector (discussed in Section 4.1).

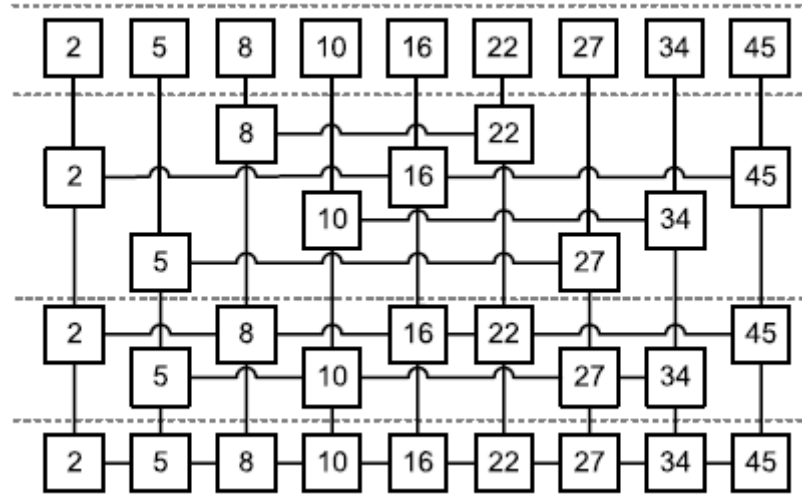


Figure 1.10 Skip graph [13]

Apart from the above discussed data structures, distributed hash tables are also used to represent peer to peer networks. It provides  $O(1)$  search complexity but cannot handle range queries and further, it suffers from collision. Many other data structures like skip nets [17] and skip webs [18] are also used in peer to peer networks.

## 1.4 Structure of The Thesis

The rest of the thesis is organized in the following order:

- **Chapter 2 Literature Review** - It surveys the history of peer to peer networks and various data structures used to represent it. Further, it discusses the benefits and applications of skip graphs.
- **Chapter 3 Problem Statement** - It discusses the research gaps and the resulting problem statement. Further, the objectives of the problem statement and the methodology used to solve it have been discussed.
- **Chapter 4 Design of Adaptive Probabilistic Skip Graph** - The first few pages have been devoted to explain the existing skip graph data structure while the rest of the pages explain in detail the proposed changes in the existing structure and algorithm.
- **Chapter 5 Implementation Details** - This chapter includes the experiments performed and analysis of the results. Further, it also compares the existing skip graph and adaptive probabilistic skip graph.

- **Chapter 6 Conclusion and Future Scope** - This concludes the thesis, the contribution it makes and the future scope.

Finally references, list of publications, reflective diary and appendix have been appended.

## Chapter 2 Literature Review

---

Since the invention of computers, humans have always tried to increase their computation power and produced supercomputers to perform trillions of calculations per second. While it was being tried to enhance the power of a single computer, in 1960 the term *distributed computing* came into existence where the nodes communicated via message passing [19]. It is often confused with parallel computing. Distributed system is a type of loosely coupled system while parallel computing is a type of tightly coupled systems [20]. In very little span of time, distributed systems have gained huge popularity and various architectures were proposed for the personal computers to take part in the network to leverage the benefits of distributed systems. Client server based systems were the first to come and were the main contributors for the growth of internet. Later, 3-tier and multitier architectures were proposed. Finally, peer to peer network was invented to overcome the limitation of these architectures. Peer to peer networks provided better scalability, fault tolerance, load balancing, decentralization, self -stabilization and data availability [9].

Today, optimization is a basic research tool used in all areas of science, engineering, medicine and management. With the growth in the knowledge of all the aspects of the field and the increasing interdisciplinary nature of the fields, new techniques have been developed to improve the optimization results. Almost in every sphere of human activity, decision making based on optimization procedures are being successfully applied. Memory usage and computation speed optimization is the major research concern in every field. Different techniques for optimization problems use various data structures depending on the nature of the problem. Distributed hash tables, skip lists and skip graphs are used in optimization problems where objectives and constraints can be only formulated using linear functions with respect to the decision variables. The great work done by James Aspnes on peer to peer networks created a lot of enthusiasm for the invention of various data structures, which provided a number of new techniques for large scale dynamic peer to peer networks [15].

To overcome scalability problem, recent systems like Tapestry [21], CAN [22], Viceroy [23], Pastry [24], Chord [25] use distributed hash table. Hashing of the key of the resource is used to determine where it will be stored. The main operation is to find out the identity of the node where a resource is stored. In overlay graph, the location of the node and resource is determined by hashed value of its identity and key respectively. Different routing algorithms are used for the resource location using overlay graphs. Plaxtons' algorithm based on hypercube routing is used by Pastry and Tapestry [21] [24]. In CAN, a d-dimensional space is partitioned into zones owned by nodes which stores keys mapped to their zones. Routing is performed by greedily forwarding messages to the closest neighbor of the target zone [22]. In Chord, nodes and resources are mapped to identities of  $m$  bits placed around a modulo  $2^m$  identifier circle and greedy routing is done to the farthest possible node stored in the routing table [25]. Most of these systems use  $O(\log^2 n)$  time for insertion and  $O(\log n)$  space and time for routing.

In optimization problems such as peer to peer networks, node entry, update, and failure are stochastic. In stochastic problems, apart from the challenges faced by deterministic problems, an additional challenge of dealing with uncertainties is to be dealt with which makes it necessary to implement probabilistic techniques along with the optimization techniques. This led to the invention of skip list and skip graph. The main goal was to design and control stochastic systems while optimizing the performance.

### **Skip Lists**

Skip list stores sorted elements by using a hierarchy of linked lists which contains increasingly sparse subsequence of items. The lowest level contains all the keys. Each level contains two special keys  $+\infty$  and  $-\infty$  and each list is a subsequence of the previous one and the highest level contains only the two special keys [26]. The running time of search, insertion and deletion algorithms is affected by height  $H$  of the skip list which is of order of  $O(\log n)$  [27]. Guaranteed properties are offered by deterministic skip list while high probability performance is offered by randomized skip list. Skip lists are as versatile as balanced trees and are simpler, faster and use less space as they use

probabilistic balancing rather than enforced balancing [28]. Skip lists can be used to represent a dictionary [29].

### **Skip Graphs**

Skip graphs were invented by James Aspnes and Gauri Shah in 2003. They are a kind of distributed data structure and are an extension of skip list consisting of tower of increasingly refined linked lists, where each list is doubly linked. In contrast to skip lists, in skip graphs every list contains all the elements and contains no head. This key difference enables skip graphs to provide better fault tolerance than skip lists [30]. Absence of head also allows carrying out parallel search in skip graphs. Skip graphs are one of the best suitable data structure to represent a peer to peer network where the entire data structure would be distributed among the nodes of the peer to peer network [31]. Algorithms to construct a skip graph, insert new nodes into it, search it, detect and repair errors caused by node failures are simple and straightforward [32].

### **Benefits of skip graphs**

#### **Concurrency and efficient operations**

Searching, insertion, deletion and update can be carried out in parallel in skip graph with a complexity of the order of  $O(\log n)$  [30]. Searching is always carried out before deletion, update and insertion.

#### **Fault tolerance**

Fault tolerance describes the number of nodes which are disconnected from the graph by the failure of other nodes [33]. Two types of failure models have been studied: Random and adversarial failures. In random failures, nodes fail independently from each other with some probability while in adversarial failures nodes are failed in a planned way such that the worst possible failure occurs at any instant. In adversarial failures, the structure of the entire skip graph is known and failures are chosen in a way to maximize node disconnection. Skip graphs are highly fault tolerant to random failures [13]. Because of the presence of redundant links, normal operations can be continued even when large number of nodes fails. Unless  $O(1/\log n)$  nodes fail, the skip graph does not fail

completely and continue to function normally. Simulations performed by James Aspnes have shown that a skip graph having 131072 nodes can tolerate up to sixty percent of its node failure before the surviving nodes are isolated [30]. Other data structures may achieve higher resilience but at the cost of greater complexity. As in a large network, it is difficult to find a worst case failure pattern and hence it is difficult to simulate adversarial failure. Resilience depends upon the vertex expansion ratio of the graph. For a set of nodes  $X$  in a graph  $G$ , expansion ratio is the number of nodes adjacent to the nodes in  $X$  (but not in  $X$ ) divided by the number of nodes in  $X$ . In skip graphs having large expansion ratio of  $\Omega(1/\log n)$ , at most  $O(w \log n)$  nodes may be separated even if up to  $w$  failures are specifically targeted [15].

### **Load Balancing**

If a large number of search operations are carried out on a node, it will lead to heavy load on the node and the other nodes which fall in the search path. But load balancing is due to the load spreading effects which are a result of assuming a random membership vector [15]. To balance the load, random walk of length  $\log n$  is carried out to record the number of data items held by each encountered node. A segment which splits the load of the most heavily loaded nodes is where insertion is carried out. The probability that a set of heavily loaded nodes would be hit by a walk of length  $\log n$  is same as its probability of being hit by  $\log n$  independent samples [16].

### **Locating highly replicated data items**

Suppose that some data item is very popular and appears at a large fraction of nodes. The node holding the popular item would be hit fast if a random walk is done. To search popular items, exhaustive search is more efficient rather than doing an exact search using distributed hash table. Further, a random walk is more efficient than exhaustive search if the underlying graph is an expander [15].

### **Applications of Skip Graph**

Skip graphs along with its variants like rainbow skip graphs [13], skip nets [17] and skip webs [18] are used in wide variety of applications:

- Dynamic overlay network is a logical network made by its members over wired or wireless networks. The number of users in this network grows exponentially. Thus, skip graph is a good option as it is highly scalable and provides fast and efficient searching. Apart from that, in overlay network node failure is a norm rather an exception. A data structure which can bear disturbances, inconsistencies and faults is required and skip graphs serves the purpose very well [34].
- Usage of skip graphs in multithreaded, multiprocessor or distributed systems show better results as discussed in [5] [31].
- A skip graph may serve as an excellent hybrid data structure; i.e., it may serve structured and unstructured P2P system simultaneously as skip graph provides high connectivity [35].
- The cloud computing has completely revolutionized the economics of computing because of its “Pay as you use” policy. In cloud, data is modified and retrieved in a purely distributed manner. Thus, a data structure which is persistent i.e. that allows frequent changes and allows to access the previous versions is required. Authentication is also an important requirement to maintain confidentiality of the information and to validate the received data. In such situation, skip graph is quite advantageous over traditional data structure [36].
- Web services have also gained huge popularity to serve various e-commerce, science and research communities. Service discovery is one of the major problems which is the search of a particular service or integrating several services which satisfies the need of the user. Skip graph provides an easy solution [37].
- Social networking analysis requires a dynamic scalable graph and skip graphs can be comfortably used [38]. Skip graphs are also used in applications which have complex queries rather than a simple dictionary where only membership queries need to be answered. DNA databases and location based queries are some of the application areas.
- Skip graphs can also be used in peer to peer networks as in file sharing (Napster, Kaza), multiplayer games [39], ad-hoc networks, grid computing [40], mobile sensor networks [41], biological networks and neural networks [42].

Thus, after carrying out the literature review it is very clear that skip graphs is a versatile data structure, used in a variety of applications. A number of variants of skip graphs have been invented to make it suitable for the application in which it would be used. But it has been found that little attention has been given to improve the search complexity of the skip graphs, in case a same node is queried again and again by a certain node. In this thesis, effort has been made to make the skip graphs more adaptive and smarter to the search trends in the network. The basic data structure remains the same with an addition of probability vector and modifications have been done in the search algorithm.

## Chapter 3 Problem Statement

---

### 3.1 Research Gaps

With the increasing number of users on the internet and growing amount of data, peer to peer networks, a class of distributed systems have gained huge popularity. Various data structures have been used from time to time to represent a peer to peer network. Adjacency matrix [11] provides a space complexity of  $O(n^2)$  which is very large and insertion and deletion of a node is also quite costly. Adjacency list reduces the space complexity but all the basic operations insertion, deletion and update have  $O(n)$  time complexity. Skip nets [17] and skip webs [18] have very high congestion. Although, distributed hash tables provides a search complexity of  $O(1)$  but they destroy the key ordering and could not handle range queries. Skip lists provide a search complexity of  $O(\log n)$  [12] but they are not very fault tolerant as deletion of the head nodes leads to the collapse of the entire network. Skip graphs also have  $O(\log n)$  search complexity [14] and it provides most of the necessary features of peer to peer networks including fast insertion, fast deletion, fast update, load balancing, fast and efficient query retrieval, self stabilization, decentralization, scalability, fault tolerance and data availability [13][15][16]. Thus, it is one of the most suitable data structure to represent a peer to peer network. Researchers keep on optimizing and introducing changes in the existing data structure to meet the changing needs of the applications. A lot of work has been done to make skip graphs more fault tolerant and secure. But very less work has been done on improving the search complexity of the skip graphs. A major research gap has been observed in cases where a node repeatedly searches a certain node. The search complexity remains the same that is  $O(\log n)$  and network congestion keeps on increasing with each search releasing  $\log n$  messages in the network.

### 3.2 Problem Statement

Considering the research gaps stated above, the problem statement is to propose and implement a probabilistic technique to improve the search efficiency of the skip graph, in case a node frequently searches for a certain node in the graph. This will reduce the unnecessary message injection into the network for searching the same resource, time and

again. Minor changes needs to be done in the available structure and algorithm keeping in mind that the space utilized by the new variables is not very high.

### **3.3 Objectives**

The objective of the thesis is:

- To study, analyze and explore the various probabilistic techniques which can be implemented in the skip graphs.
- To propose a technique for improvement in the search time complexity of skip graphs for a case when the same node queries a certain node again and again i.e. to enable it to learn the frequency of search in the network.
- To test and validate the proposed algorithm and compare the results with the existing algorithm.

### **3.4 Methodology**

The methodology to be followed is as under:

- Implement the proposed changes in C language by using skip graph as the base data structure and modify it to add the probability vector.
- Modify the search algorithm to update the probability vector of the caller whenever searching takes place.
- Change the membership vector of the caller if the probability vector indicates regarding the change. This will help in making skip graphs smarter and adaptive to the current trends in data. The threshold for changing the probability will be set at run time.
- Running similar search queries on adaptive probabilistic skip graph and skip graph to verify that the search time reduces drastically in the former when the same node queries a certain node again and again.

## Chapter 4 Design of Adaptive Probabilistic Skip Graph

### 4.1 Existing Skip Graph Data Structure

As discussed in literature review, peer to peer networks have gained huge popularity because of their efficient load balancing and utilization of resources. Various data structures have been proposed to represent peer to peer networks like adjacency matrix, adjacency list, skip lists, skip graphs, skip webs, skip nets and distributed hash tables. Skip graph has evolved as one of the best candidate to represent a peer to peer network as it offers most of the necessary features required to implement a peer to peer network.

#### 4.1.1 Structure of Skip Graph Node

The basic structure of skip graph node having 4 levels is as follows:

| Key                        |          |
|----------------------------|----------|
| Levels                     |          |
| Left[3]                    | Right[3] |
| Left[2]                    | Right[2] |
| Left[1]                    | Right[1] |
| Left[0]                    | Right[0] |
| Membership Vector          |          |
| Any other Parameter (flag) |          |

Figure 4.1 Structure of skip graph node [30]

The major elements are:

1. **Key:** Key stores the unique value that distinguishes a node from all the other nodes. Nodes are sorted in increasing order (or decreasing order) on the basis of their key attribute.
2. **Levels:** It defines the maximum levels till which a node is present and is equivalent to the current number of levels present in the skip graph. Its value is equal to  $\text{ceil}(\log n)$  where  $n$  is the number of nodes in the skip graph.
3. **Left and Right Pointers:** These are the pointers of node type that point to the left and right neighbors of the node at all the levels.
4. **Membership vector:** This is a random vector of infinite length over a fixed alphabet. Generally, the size is kept one less than the maximum number of levels in the skip

graph and the alphabets chosen are 0 and 1. That is, membership vector is a boolean vector. A node is present in any sub-list (lists on a level), if and only if its prefix matches with the label of that sub-list. The sub-list at level 0 has no label. Level 1 sub-lists have label of length 1, level 2 sub lists have label of length 2 and so on. The label of the highest level is the complete membership vector.

5. Extra Parameter: This is required to store any other parameter or flag that is required as per the application.

Figure 4.2 shows a skip graph having 14 nodes:

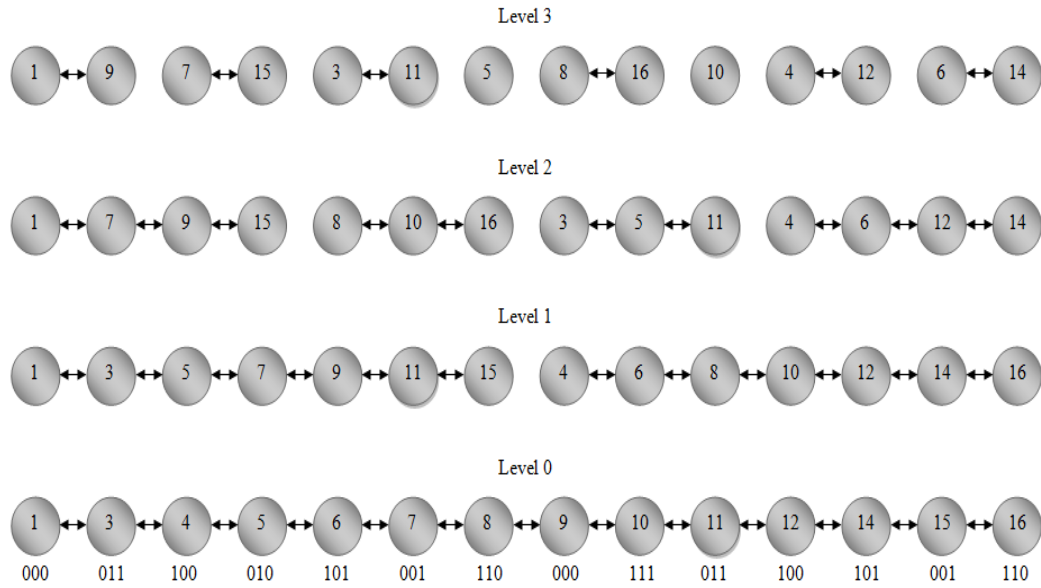


Figure 4.2 Skip graph having 14 nodes

Figure 4.2 shows a skip graph of 14 nodes each having its key and membership vector. The left and right pointers of each node have also been shown. Number of levels is four equal to  $\text{ceil}(\log n)$  i.e.  $\text{ceil}(\log 14)$ . The membership vector is three bit long i.e. one less than the maximum number of levels. All the nodes are connected at level 0. At level 1, nodes having same most significant bit are connected. At level 2, nodes having same two most significant bits are attached. At level 3, nodes having the same membership vector are connected. Level 0 one has 1 sub-lists, level 1 has 2 sub-lists, level 2 and 3 have 4 and 8 sub-lists respectively i.e. each level has  $2^k$  sub-lists where  $k$  is the level number.

#### 4.1.2 Insertion in Skip Graph

Insertion in skip graph is carried out from the bottommost level. Initially, the node is allocated memory and if the introducer and the new node are same and it is the first node to be inserted in the skip graph, then the node is directly added at all the levels with all the left and right pointers as NULL. For all other nodes, find the left and right neighbors of the node at all the levels, based upon its membership vector and insert it at all the levels. The complexity of insertion is  $O(\log n)$ . The detailed insertion algorithm 1 [32] is as follows:

**Input:** Introducer, Key

**Output:** The node is added in the skip graph

```
insert (introducer, key)

Start searching from introducer to find max  $s < key$  and
to find minimum  $y > key$ 
set right neighbor equal to  $y$  at level 0
set left neighbor equal to  $s$  at level 0

set  $s'$  equal to  $s$ 
set  $y'$  equal to  $y$ 
Initialize: level = 0
while level < maxlevels-1 do
    starting from  $s'$  search backward at level to find  $s'$ 
    such that  $m(s')|(level + 1) = m(key)|(level+1)$ 
    starting from  $y'$  search forward at level to find  $y'$ 
    such that  $m(y')|(level + 1) = m(key)|(level+1)$ 

    if  $s'$  doesn't exists and  $y'$  exists then
        set right neighbor at level+1 equal to  $y'$ 
        set left neighbor at level+1 equal to null
    else if  $y'$  doesn't exists and  $s'$  exists then
        set right neighbor at level+1 equal to null
        set left neighbor at level+1 equal to  $s'$ 
    else if  $s'$  and  $y'$  exists then
        set right neighbor at level+1 equal to  $y'$ 
        set left neighbor at level+1 equal to  $s'$ 
    end
    level = level+1
end
```

Algorithm 1 Insertion in skip graph

Let's consider an example in which 16 nodes are added one after the other in the skip graph:

1. First node 1 with membership vector 000 is added by node 1 and then node 16 with membership vector 110 is added by node 1.



Figure 4.3 Insertion of nodes 1 and 16

2. Then, node 8 (110) is added by node 16 and node 4 (100) is added by node 1.

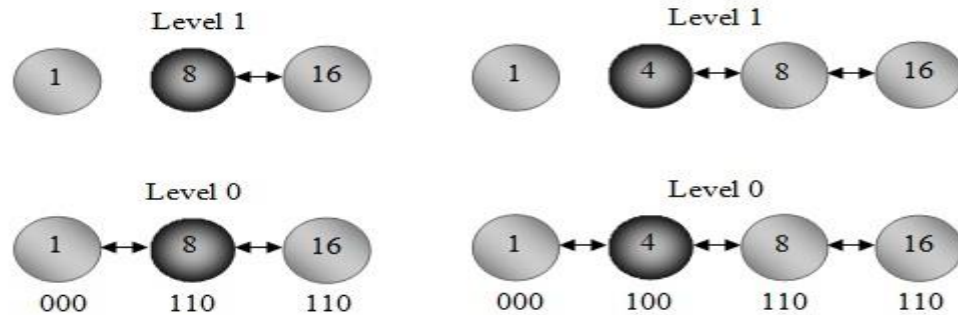


Figure 4.4 Insertion of nodes 8 and 4

3. Then, node 12 (100) is added by node 16. As the number of nodes becomes equal to 5 ( $2^2 + 1$ ), a new level 2 is added. After this, node 3 (011) is added by node 1.

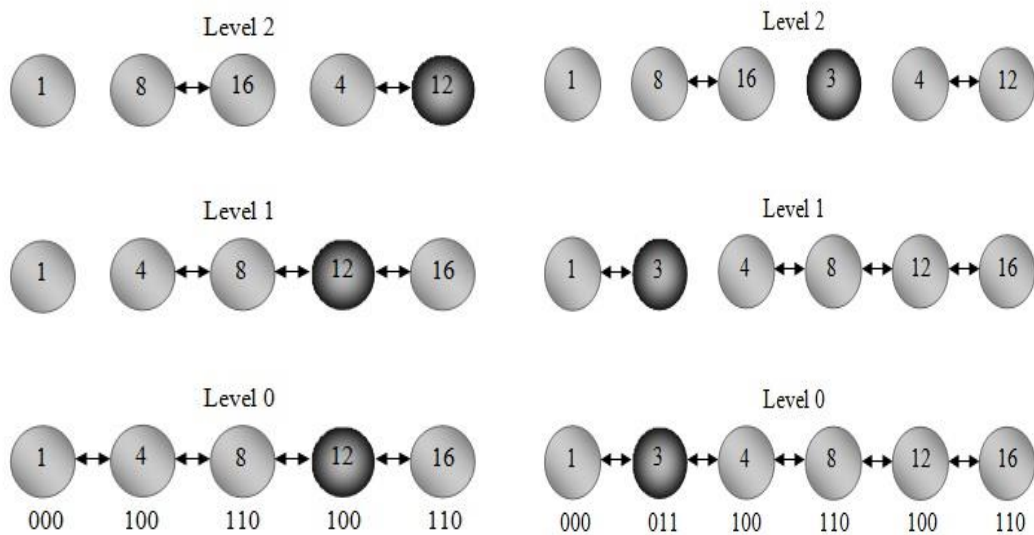


Figure 4.5 Insertion of nodes 12 and 3

4. Node 15 (001) is added by node 16

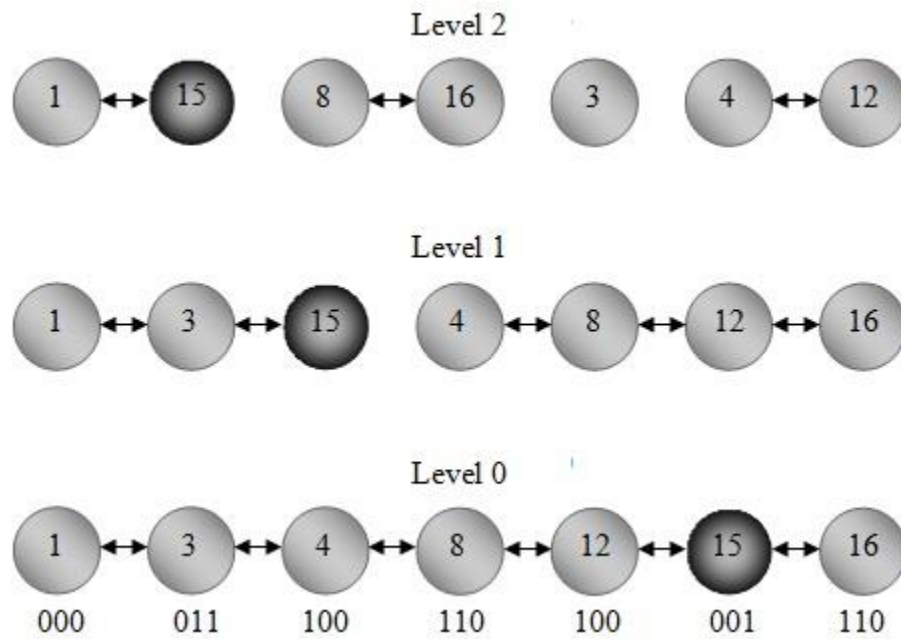


Figure 4.6 Insertion of node 15

5. Node 5 (010) being added by node 3.

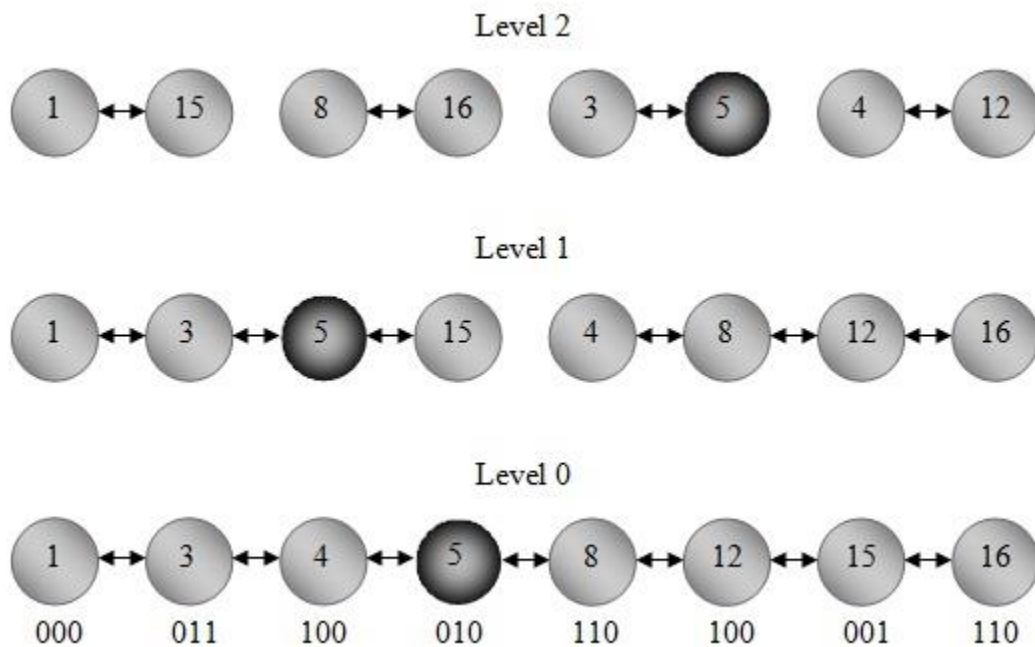


Figure 4.7 Insertion of node 5

6. Then, node 7 (001) is added by node 12. As the number of nodes becomes equal to 9 ( $2^3 + 1$ ), a new level 3 is added.

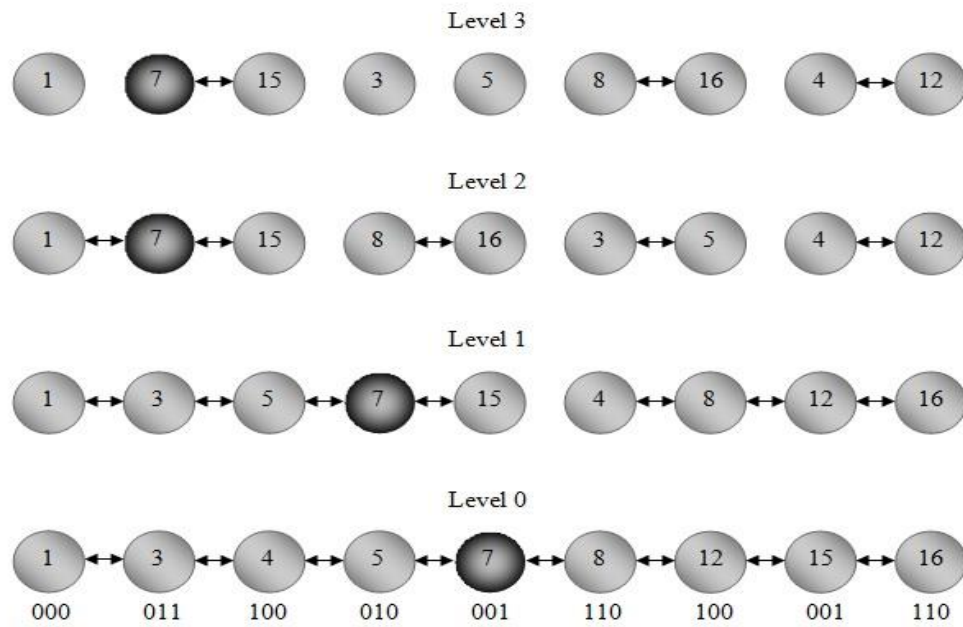


Figure 4.8 Insertion of node 7

7. Then, node 10 (111) is added by node 4.

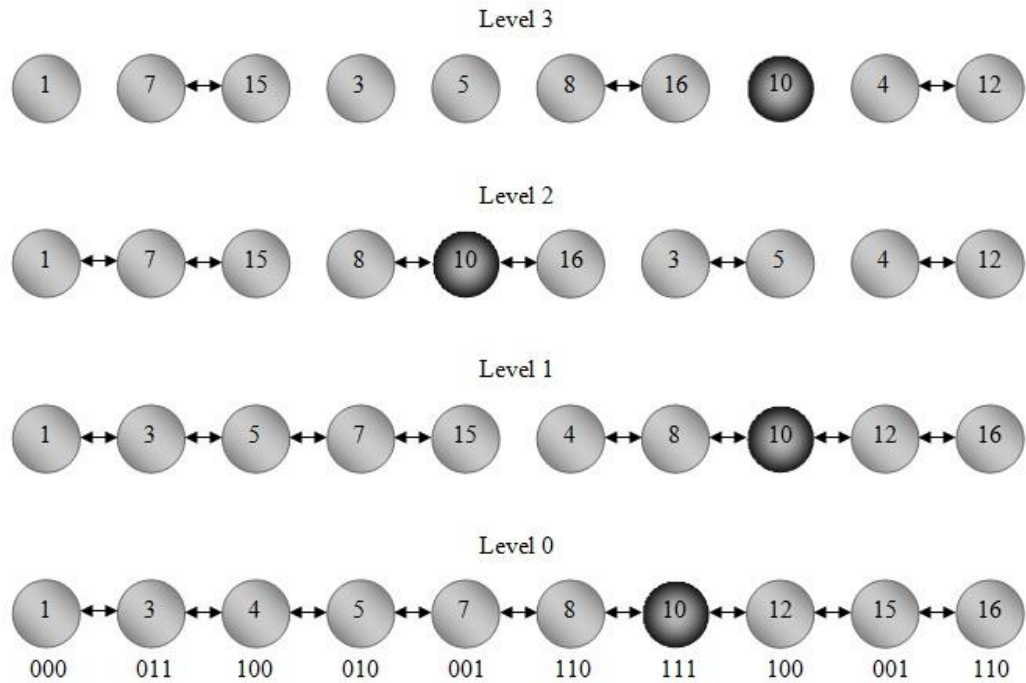


Figure 4.9 Insertion of node 10

8. Then, node 6 (101) is inserted by node 10, node 9 (000) by node 5, node 14 (101) by node 15, node 11 (011) by node 4 and node 2 (111) by node 10.

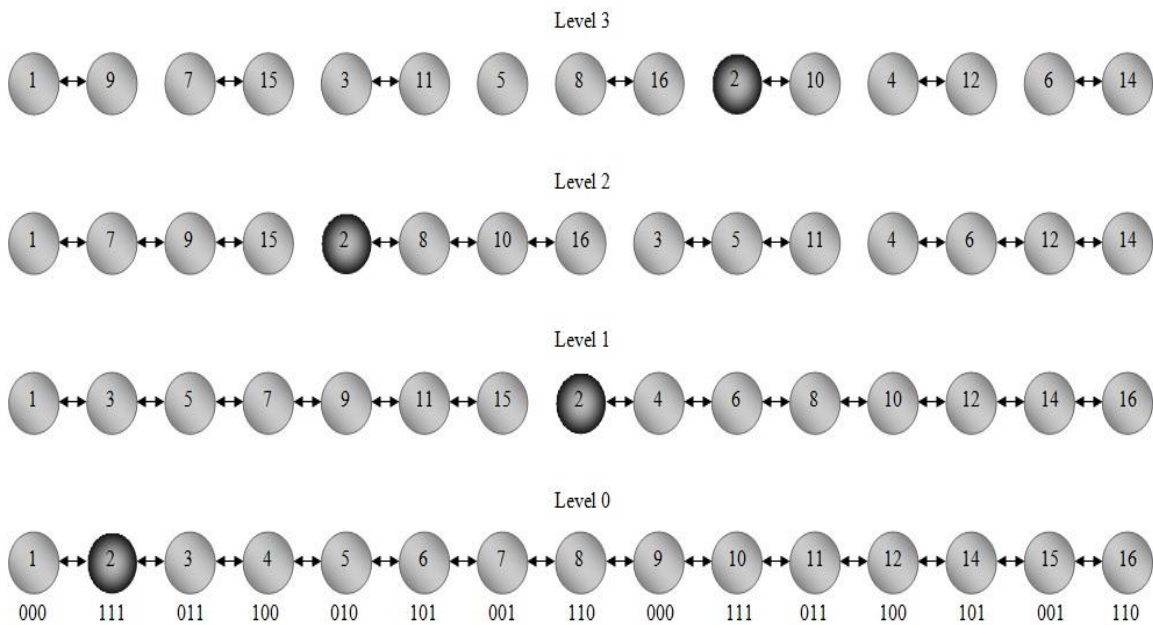


Figure 4.10 Insertion of node 2

9. And finally node 13 (010) is added by node 1.

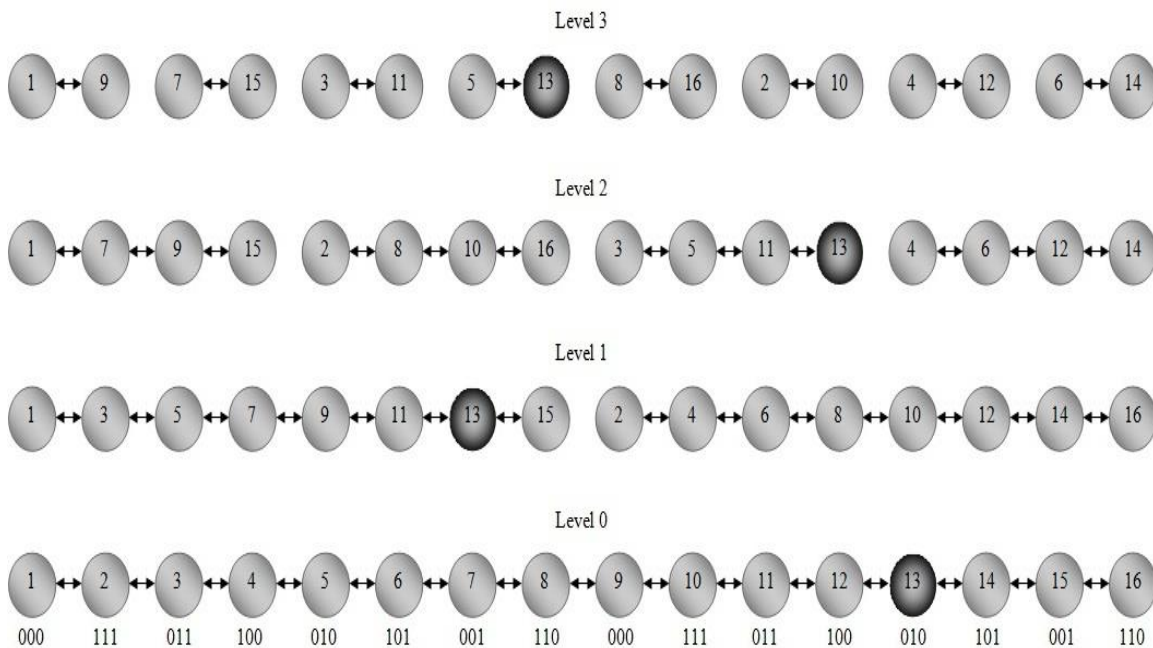


Figure 4.11 Insertion of node 13

### 4.1.3 Deletion in Skip Graph

The deletion algorithm first searches the node to be deleted by starting the search from the head node at the maximum level. Once the address is obtained, delete the node from all the levels by adjusting the left and right pointers of its right and left neighbors respectively. The average complexity of deletion is  $O(\log n)$ . The delete algorithm 2 [32] is as follows:

**Input:** Key of the node to be deleted

**Output:** The node is deleted from the skip graph

```
delete (node)

for le = 0 to maxlevels-1 do
    delete node from list at level le
end
```

Algorithm 2 Deletion in skip graph

The example shown below deletes node 1 from the skip graph shown in Figure 4.11 created after insertion of all the 16 nodes in the graph.

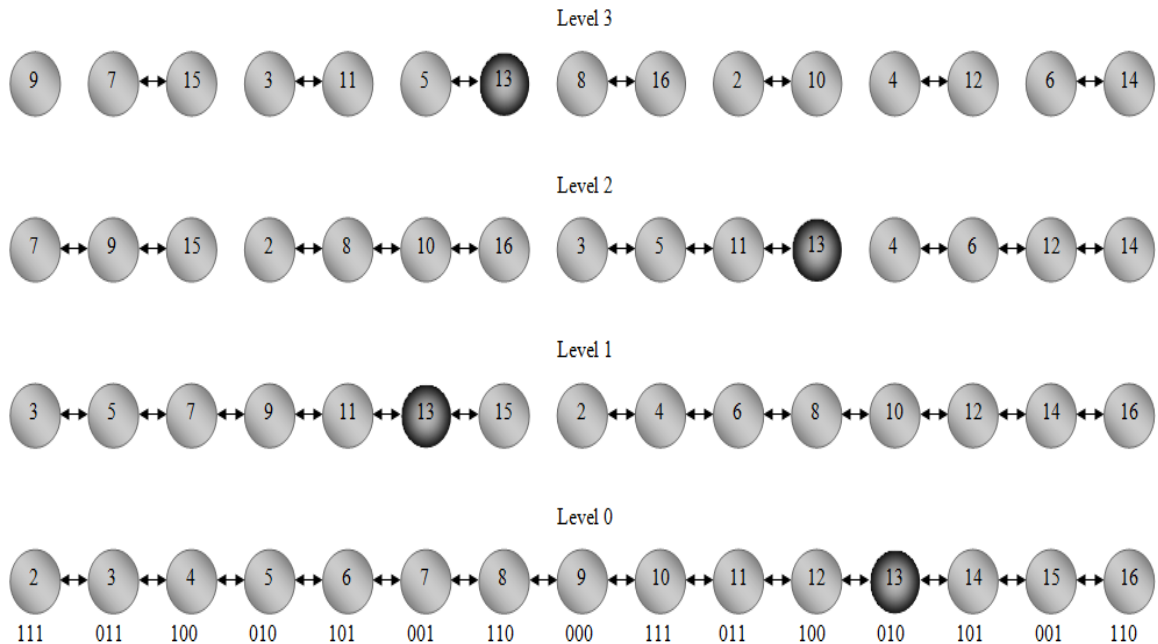


Figure 4.12 Deletion of node 1

#### 4.1.4 Search in Skip Graph

In search algorithm, the search starts from the topmost level. If the start node has the same key as that of the search node, then it is returned while if the start node is greater than the search node one moves towards left until the node is still greater than or equal to the search node and decrease the level if it becomes smaller. If the start node is smaller than the search node one moves towards right until the node is still less than or equal to the search node and decrease the level if it becomes greater. If the level becomes less than zero then it means the searched node does not exist. The average complexity of searching is  $O(\log n)$ .

The search algorithm 3 [32] is as follows:

**Input:** Node who wants to search (start node), key of the node to be searched, level

**Output:** Address of the node if the node exists and NULL otherwise

```
struct node * search (startnode, skey, level)

if (startnode.key == skey) then
    return startnode
else if (startnode.key < skey) then
    while level >= 0 do
        if ((startnode.right[level]).key <= skey) then
            return search(startnode.right[level], skey, level)
        else
            level = level - 1
        end
    end
end
else
    while level >= 0 do
        if ((startnode.left[level]).key >= skey) then
            return search(startnode.left[level], skey, level)
        else
            level = level - 1
        end
    end
end
end

if (level < 0) then
    return NULL
end
end
```

Algorithm 3 Search in skip graph

The search of node 9 by node 2 is shown below:

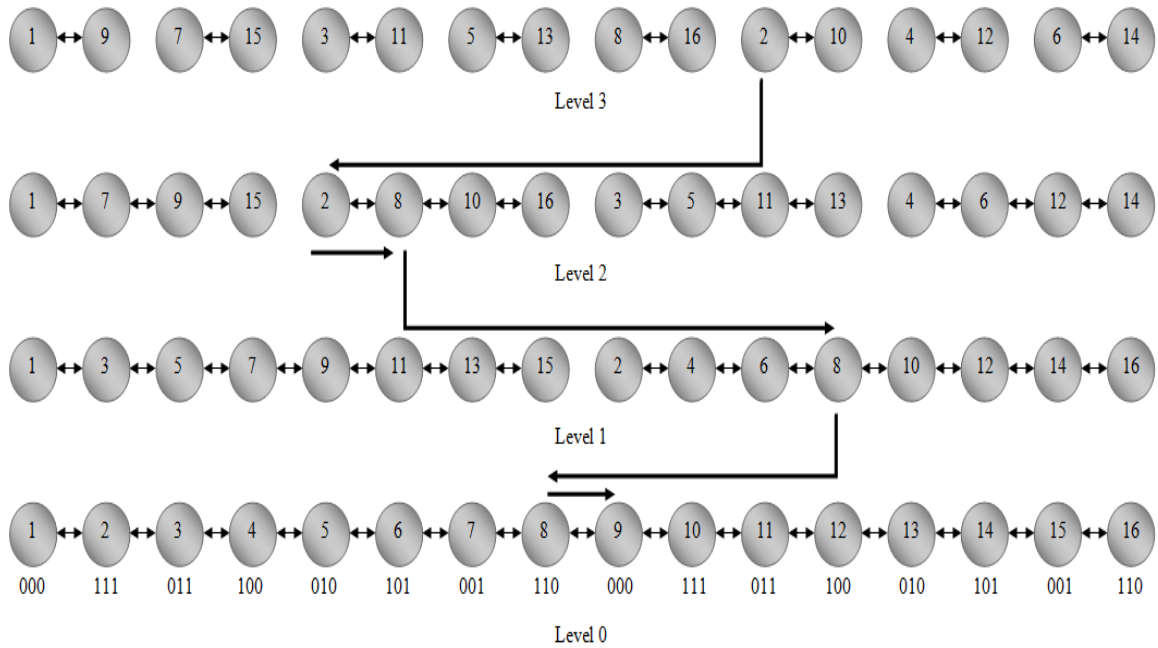


Figure 4.13 Search of node 9 by node 2

Here the search starts at the topmost level. As 2 is less than 9, it scans in the right. Node 10 is greater than 9, therefore level is decreased. Then again at level 2, node 2 scans in the right. Node 8 is smaller than 9 therefore, search moves to node 8. In the right of 8 is node 10 which is greater than 9 therefore, level is decreased. Again at level 1 on scanning the right node 10, level is further decreased to reach at the last level. Node 8 then scans the node in its right and search moves to node 9. When node 9 searches itself, it returns its address to the starting node recursively.

## 4.2 Adaptive Probabilistic Skip Graph

The skip graph discussed in section II is quite efficient and is used as such in many applications but one of the major limitation is that the search time of the skip graph remains  $O(\log n)$  even if a node is searched again and again by the same node. Moreover, as the size of the network increases, the amount of messages passed between different layers to find a particular node will definitely go on increasing. If the same node is searched for about  $n$  times, then the network would be congested with  $n \times l$  messages where 'l' is number of levels in the skip graph. If the repeated call is being made,

proposed scheme reduces the time complexity by introducing probability vector and changing the existing membership vector. If the frequency of call from a caller node to a callee node crosses a defined threshold, determined by the probability vector, the membership vector of the caller should be changed and mapped as same with that of callee. The condition for changing the membership vector is controlled by the probability vector. As the proposed skip graph adapts itself according to the frequent searches being carried out in the graph, the name **Adaptive Probabilistic Skip Graph** has been proposed.

#### 4.2.1 Structure of Adaptive Probabilistic Skip Graph Node

In order to implement the above discussed improvement, minor changes need to be done in the basic skip graph data structure. Figure 4.14 shows how a node would after applying the proposed changes:

|                           |          |
|---------------------------|----------|
| Key                       |          |
| Levels                    |          |
| Left[3]                   | Right[3] |
| Left[2]                   | Right[2] |
| Left[1]                   | Right[1] |
| Left[0]                   | Right[0] |
| Membership Vector         |          |
| <b>Probability vector</b> |          |
| Flag                      |          |

Figure 4.14 Modified skip graph node

The structure of the modified skip graph node is same as the structure of the basic skip graph node with the addition of probability vector whose size is equal to the size of the membership vector and it stores the probability of the corresponding membership vector bit to be 1. The probability of the membership vector bit to be 0 is  $1 - (\text{probability vector component of that bit})$ . The algorithms discussed in the next section, have been modified to incorporate the improvements.

#### 4.2.2 Insertion and Deletion in Adaptive Probabilistic Skip Graph

The steps followed in the insertion of a node in adaptive probabilistic skip graph are same as in skip graph with an additional initialization of the probability vector. Initially it is assumed that if the membership vector bit is 1 then probability of that bit is 0.9 and if it is 0 then probability is 0.1. The deletion in adaptive probabilistic skip graph is exactly same as the deletion in basic skip graph.

#### 4.2.3 Search in Adaptive Probabilistic Skip Graph

Searching in adaptive probabilistic skip graph is carried out as it was carried out earlier. Once the original search algorithm completes its operation, the function *updateprob* is called if the caller node and the callee node are not same. The function updates the probability vector of the caller node to make it similar to that of the callee node by comparing their membership vector bit by bit. The algorithm 4 for *updateprob* is:

**Input:** Introducer address (caller node), key address (callee node) and per (percentage of calls after which the membership vector must change)

**Output:** Updates probability vector of the caller node

```
updateprob(intro, key, per)
set calls equal to per*n
set delta equal to .4/calls
for (i=0; i<maxlevels-1; i++)
    if (intro.mem[i]==0 && key.mem[i]==0)
        Subtract delta from intro.probmem[i]
        if (intro.probmem[i]<.1)
            set intro.probmem[i] equal to .1
        end
    else if (intro.mem[i]==1 && key.mem[i]==1)
        Add delta into intro.probmem[i]
        if (intro.probmem[i]>.9)
            set intro.probmem[i] equal to .9
        end
    else if (intro.mem[i]==0 && key.mem[i]==1)
        Add delta into intro.probmem[i]
    else if (intro.mem[i]==1 && key.mem[i]==0)
        Subtract delta from intro.probmem[i]
    end
end
end
checkprob(intro)
```

Algorithm 4 updateprob in adaptive probabilistic skip graph

In the end, *checkprob* function is called. It checks if membership vector of the caller needs to be changed. This is done by checking whether membership vector is in accordance with the probability vector. If it is not, then the node is deleted and a new node with same key, same probability and new membership vector is inserted. The *checkprob* function is discussed in Algorithm 5:

**Input:** Caller node

**Output:** Caller node deleted and inserted with updated membership vector (if threshold is crossed)

```

checkprob(node)

set flag equal to 0
initialize memc vector
initialize double probmemc vector

for (i=0;i<maxlevels-1;i++)
    if(node.mem[i]==0&&node.probmem[i]>.5)
        set memc[i] equal to 1
        set probmemc[i] equal to node.probmem[i]
        set flag equal to 1
    else if(node.mem[i]==1&&node.probmem[i]<.5)
        set memc[i] equal to 0
        set probmemc[i] equal to node.probmem[i]
        set flag equal to 1
    else
        set memc[i] equal to node.mem[i];
        set probmemc[i] equal to node.probmem[i]
    end
end

if (flag==1)
    delete (node)
    insert(head,createnode(node,memc,probmemc))
end

```

Algorithm 5 checkprob in adaptive probabilistic skip graph

The search complexity is  $O(\log n)$  and it reduces drastically when a node is searched again and again by the same node. The number of calls after which the membership vector changes depends upon the input variable *per* in the *updateprob* function.

#### 4.2.4 Replacement in Adaptive Probabilistic Skip Graph

The algorithm discussed in the previous section helps in reducing the search complexity to a large extent when the same search is carried out by a node multiple times. A scenario may be such that all the nodes search for a particular node in the graph leading to a situation in which large number of nodes will be connected at the highest level. To avoid that, two replacement policies have also been proposed. When nodes equal to  $\log n$  are connected at the highest level, and a new node is to be inserted with the same membership vector, then either of the two replacement policies can be used.

- **Least frequently used:** It replaces the node which has sent or received least number of calls within its sub-list at highest level. To implement it two functions have been defined:
  - a) *update\_frequency*: It is called before calling the *updateprob* function. It updates the frequency of a node to call or being called in its sub-list at highest level by comparing the membership vector of the caller and callee node bit by bit. If their membership vector is same then the frequency of both of them is incremented otherwise the frequency of the caller is decremented. The algorithm 6 for *update\_frequency* is:

**Input:** Caller node (introducer), callee node (key node)

**Output:** Updates frequency of the caller and callee node

```
update_frequency(introducer, keynode)

set check equal to 0
for (i=0; i<maxlevels-1; i++)
    if (introducer.mem[i] != keynode.mem[i])
        set check equal to 1
        break
    end
end

if (check==1)
    introducer.frequency = introducer.frequency-1
else
    introducer.frequency = introducer.frequency+1
    keynode.frequency = keynode.frequency+1
end
```

Algorithm 6 *update\_frequency* in adaptive probabilistic skip graph

- b) *check\_frequency*: It is called once *checkprob* function finds that the membership vector of the caller needs to be changed. It returns the node which has sent or received least number of calls in its sub-list at highest level. The algorithm 7 for *check\_frequency* is:

**Input:** Target node (a node in the target sub-list)

**Output:** Least frequently used node

```
struct node * check_frequency(target)

set temp equal to target
set least equal to target

temp = temp.right[maxlevels-1]
while (temp!=NULL)
    if (temp.frequency<least.frequency)
        least = temp
    end
    temp = temp.right[maxlevels-1]
end

return least
```

Algorithm 7 *check\_frequency* in adaptive probabilistic skip graph

The *checkprob* function is also modified. When it is found that the membership vector of the caller needs to be changed, then it first checks the number of nodes in target sub-list (the sub-list in which the caller node wants to go). If it is less than  $\log n$  then the caller is directly inserted in the target sub-list and if it is equal to  $\log n$  then least frequently used existing node is replaced by the caller node.

- **Least recently used:** It replaces the node which has sent or received least recent calls in its sub-list at highest level. To implement it, two functions have been defined:
  - a) *update\_timestamp*: It is called before calling the *updateprob* function. It updates the timestamp of a node to call or being called in its sub-list at highest level by comparing the membership vector of the caller and callee node bit by bit. If their membership vector is same, then the timestamp of both of them is updated. The algorithm 8 for *update\_timestamp* is:

**Input:** Caller node (introducer), callee node (key node)

**Output:** Updates timestamp of the caller and callee node

```
update_timestamp(introducer, keynode)

set check equal to 0
for (i=0; i<maxlevels-1; i++)
    if (introducer.mem[i]!=keynode.mem[i])
        set check equal to 1
        break
    end
end

if (check==1)
    do nothing
else
    update introducer.timestamp
    update keynode.timestamp
end
```

Algorithm 8 update\_timestamp in adaptive probabilistic skip graph

- b) *check\_timestamp*: It is called once *checkprob* function finds that the membership vector of the caller needs to be changed. It returns the node which has sent or received least recent calls in its sub-list at highest level. The algorithm 9 for *check\_timestamp* is:

**Input:** Target node (a node in the target sub-list)

**Output:** Least recently used node

```
struct node * check_timestamp(target)

set temp equal to target
set least equal to target

temp = temp.right[maxlevels-1]
while (temp!=NULL)
    if (temp.timestamp<least.timestamp)
        least = temp
    end
    temp = temp.right[maxlevels-1]
end

return least
```

Algorithm 9 check\_timestamp in adaptive probabilistic skip graph

Modifications that were done in the *checkprob* function during the implementation of least frequently used replacement policy, also needs to be incorporated here.

Along with this, the basic skip graph node structure should also store frequency and timestamp for the least frequently used and least recently used replacement policy respectively. Also, for every insertion the timestamp and frequency needs to be initialized with current timestamp and zero respectively. The replacement policy to be used may be decided based upon the requirements of the application. The proposed changes have been validated and compared with the existing data structure in the next chapter.

## Chapter 5 Implementation Details

---

The proposed algorithm has been implemented in Dev C++ 5.4.2 on windows 7. The basic structure of the node is defined as follows:

```
struct node
{
    int key;
    int levels;
    bool flag;
    struct node * left[maxlevels];
    struct node * right[maxlevels];
    bool mem[maxlevels-1];
    double probmem[maxlevels-1];
    int frequency;
    LARGE_INTEGER timestamp;
};
```

The number of left and right pointers is equal to the number of maximum levels in the skip graph and the size of membership and probability of membership vector is one less than the number of maximum levels in the graph. To store the timestamp, “Query Performance Counter Utility” of windows has been used. It gives the number of clock cycles passed since the system boot time. Its precision is in microseconds. If the LONG\_INTEGER range is not sufficient, then one can use a counter which is incremented on every insertion and when a caller and callee have the same membership vector.

### 5.1 Functions Defined

The following functions have been defined:

1. `bool chkcount ()`

**Input:** None

**Output:** Returns 0 if the number of nodes is equal to maximum number of nodes, otherwise it returns 1.

2. `bool chkkey (int nodekey)`

**Input:** Key

**Output:** Checks if the key already exists in the skip graph. It returns 1, if the node is not present in the skip graph and 0 otherwise.

3. `struct node * create_node (int nodekey)`

**Input:** Key

**Output:** Creates the node with the passed key

4. `struct node * create_node (int nodekey, bool *mem, double *prob)`

**Input:** Key, membership vector, probability vector

**Output:** Creates the node with passed key, membership vector and probability vector

5. `struct node * find (int key)`

**Input:** Key

**Output:** Returns the address of the key. This is required to find the address of the introducer

6. `void insert (struct node * introducer, struct node * newnode)`

**Input:** Introducer address, new node address

**Output:** Updates the links and adds the created node in the skip graph

7. `bool inc_check()`

**Input:** None

**Output:** Returns 1 if the number of nodes is of the form  $2^i + 1$

8. `void inc_level()`

**Input:** None

**Output:** Adds one more level in the skip graph

9. `bool dec_check()`

**Input:** None

**Output:** Returns 1 if the number of nodes is of the form  $2^i$

10. `void dec_level()`

**Input:** None

**Output:** Decreases one level in the skip graph

11. `void display (struct node * newnode)`

**Input:** Node address

**Output:** Displays details of the passed node

12. `void remove (int nodekey)`

**Input:** Key

**Output:** Deletes the node from the skip graph

13. `struct node * search (struct node *introducer, struct node *temp, int key, int level)`

**Input:** Introducer node address, current pointer, key, level

**Output:** Returns the address of the key

14. `void updateprob(struct node * introduceradd, struct node * keyadd)`

**Input:** Caller node address, callee node address

**Output:** Updates the probability of the caller node

15. `void checkprob(struct node * introduceradd)`

**Input:** Address of the caller node

**Output:** Checks if the node's membership vector needs to be changed and if the check is satisfied, changes the membership vector of the node

16. `void traverse ()`

**Input:** None

**Output:** Displays the skip graph at a glance

17. `int bintodec(bool * memb)`

**Input:** Membership vector

**Output:** Returns the decimal equivalent of the membership vector

18. `void postsearch(struct node * introduceradd, struct node * keyadd)`

**Input:** Caller and callee address

**Output:** Calls *updatefrequency*, *updatetimestamp* and *updateprob* in sequence

19. `void updatefrequency(struct node * introduceradd, struct node * keyadd)`

**Input:** Caller and callee address

**Output:** Updates the frequency of caller and callee

20. `void updatetimestamp(struct node * introduceradd, struct node * keyadd)`

**Input:** Caller and callee address

**Output:** Updates the timestamp of caller and callee

21. struct node \* checkfrequency(struct node \*target)

**Input:** Target node address

**Output:** Returns the address of the least frequently used node within the target sub-list

22. struct node \* checktimestamp(struct node \*target)

**Input:** Target node address

**Output:** Returns the address of the least recently used node within the target sub-list

23. struct node \* look (bool\* memb)

**Input:** Membership Vector

**Output:** Returns the smallest node having the passed membership vector

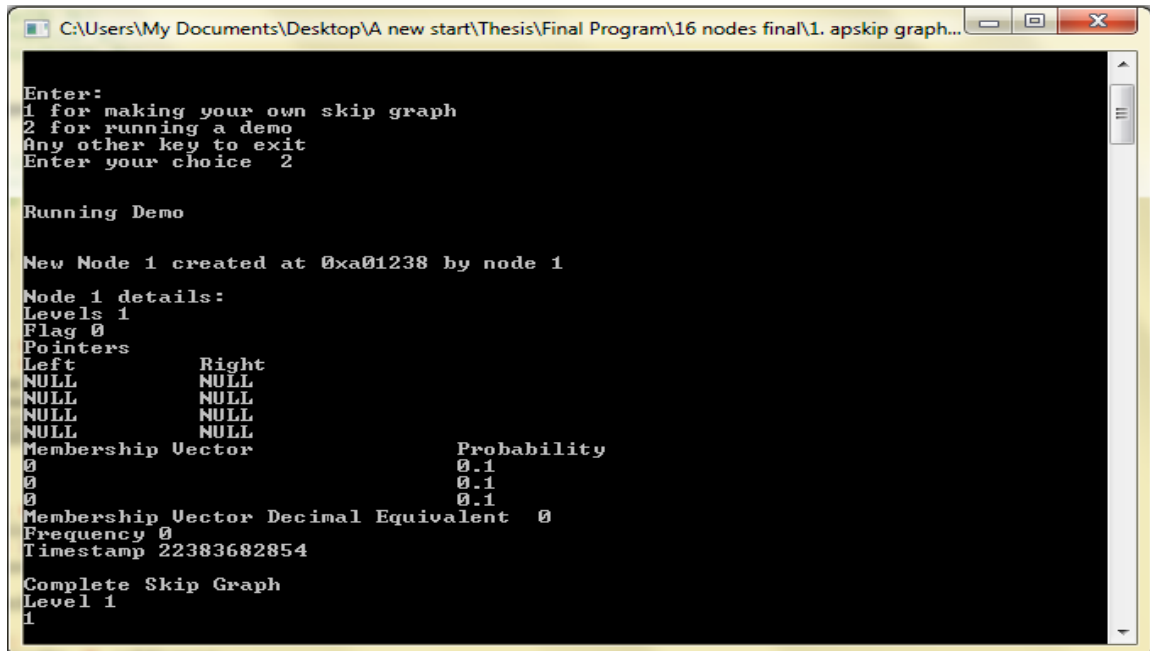
24. bool isreplace(struct node \* target)

**Input:** None

**Output:** Returns 1 if the target sub-list has log n elements, otherwise it returns 0

## 5.2 Results and Analysis

The insertion example discussed in section 4.1.2 has been implemented to create a skip graph of 16 nodes. Initially node 1 is introduced by node 1 as shown in Figure 5.1.



```
C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\1. apskip graph...
Enter:
1 for making your own skip graph
2 for running a demo
Any other key to exit
Enter your choice 2

Running Demo

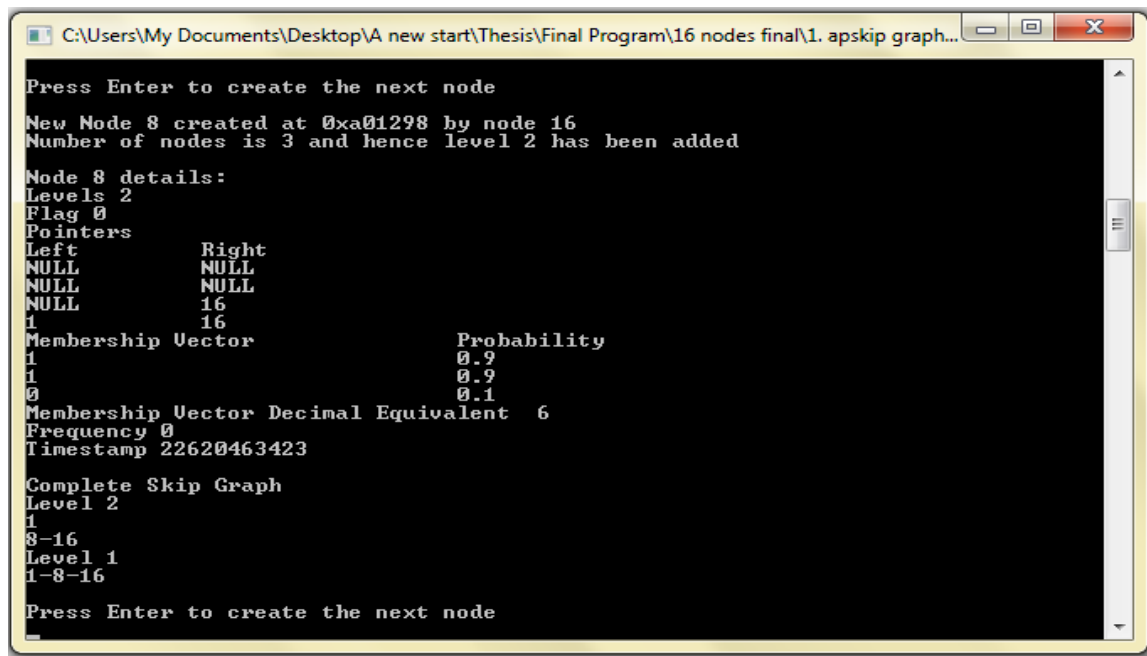
New Node 1 created at 0xa01238 by node 1

Node 1 details:
Levels 1
Flag 0
Pointers
Left      Right
NULL      NULL
NULL      NULL
NULL      NULL
NULL      NULL
Membership Vector      Probability
0                      0.1
0                      0.1
0                      0.1
0                      0.1
Membership Vector Decimal Equivalent  0
Frequency 0
Timestamp 22383682854

Complete Skip Graph
Level 1
1
```

Figure 5.1 Insertion of node 1

Then, node 16 was introduced, followed by insertion of node 8. Number of levels in skip graph increases to 2 as the number of nodes is 3 as shown in Figure 5.2.



```

C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\1. apskip graph...

Press Enter to create the next node
New Node 8 created at 0xa01298 by node 16
Number of nodes is 3 and hence level 2 has been added

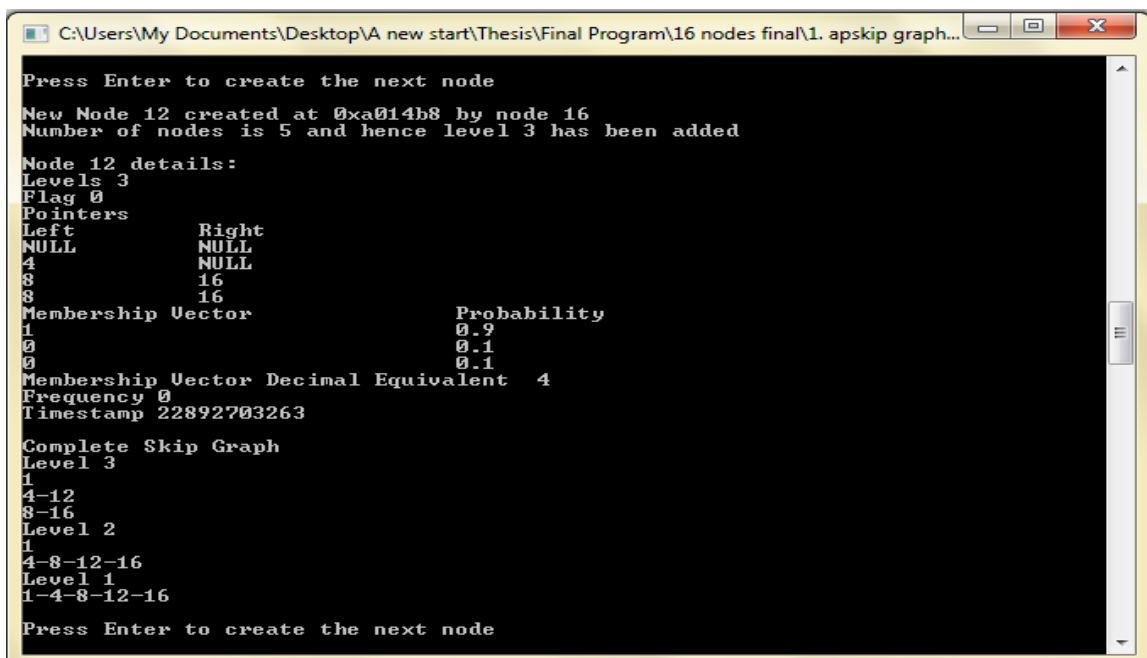
Node 8 details:
Levels 2
Flag 0
Pointers
Left      Right
NULL      NULL
NULL      NULL
NULL      16
1         16
Membership Vector      Probability
1                     0.9
1                     0.9
0                     0.1
Membership Vector Decimal Equivalent 6
Frequency 0
Timestamp 22620463423

Complete Skip Graph
Level 2
1
8-16
Level 1
1-8-16
Press Enter to create the next node

```

Figure 5.2 Insertion of node 8

Then, nodes 4 and 12 were introduced. Number of levels increases to 3 as number of nodes is 5 now as shown in Figure 5.3.



```

C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\1. apskip graph...

Press Enter to create the next node
New Node 12 created at 0xa014b8 by node 16
Number of nodes is 5 and hence level 3 has been added

Node 12 details:
Levels 3
Flag 0
Pointers
Left      Right
NULL      NULL
4         NULL
8         16
8         16
Membership Vector      Probability
1                     0.9
0                     0.1
0                     0.1
Membership Vector Decimal Equivalent 4
Frequency 0
Timestamp 22892703263

Complete Skip Graph
Level 3
1
4-12
8-16
Level 2
1
4-8-12-16
Level 1
1-4-8-12-16
Press Enter to create the next node

```

Figure 5.3 Insertion of node 12

Then, nodes 3, 15, 5 and 7 were introduced. Number of levels increases to 4 as number of nodes is 9 now (Figure 5.4).

```

C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\1. apskip graph...
Press Enter to create the next node
New Node 7 created at 0xa01518 by node 12
Number of nodes is 9 and hence level 4 has been added

Node 7 details:
Levels 4
Flag 0
Pointers
Left      Right
NULL      15
1         15
5         15
5         8
Membership Vector      Probability
0                      0.1
0                      0.1
1                      0.9
Membership Vector Decimal Equivalent 1
Frequency 0
Timestamp 23161287972

Complete Skip Graph
Level 4
1
3
4-12
5
7-15
8-16
Level 3
1-7-15
3-5
4-12
8-16
Level 2
1-3-5-7-15
4-8-12-16
Level 1
1-3-4-5-7-8-12-15-16

Press Enter to create the next node

```

Figure 5.4 Insertion of node 7

Then, nodes 10, 6, 9, 14, 11, 2 and 13 were introduced to create the final skip graph of 16 nodes (Figure 5.5).

```

C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\1. apskip graph...
Press Enter to create the next node
New Node 13 created at 0xa03a88 by node 1
Node 13 details:
Levels 4
Flag 0
Pointers
Left      Right
5         NULL
11        NULL
11        15
12        14
Membership Vector      Probability
0                     0.1
1                     0.9
0                     0.1
Membership Vector Decimal Equivalent  2
Frequency 0
Timestamp 23486202624

Complete Skip Graph
Level 4
1-9
2-10
3-11
4-12
5-13
6-14
7-15
8-16
Level 3
1-7-9-15
2-8-10-16
3-5-11-13
4-6-12-14
Level 2
1-3-5-7-9-11-13-15
2-4-6-8-10-12-14-16
Level 1
1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16
Complete Skip Graph Created_

```

Figure 5.5 Insertion of node 13

Deletion of node 1 was done to test the deletion function (Figure 5.6).

```

C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\1. apskip graph...
Press enter to delete node 1
Node 1 deleted successfully.

Complete Skip Graph
Level 4
2-8-16
3-11
4-13
5-6-9
7-15
10
12
14
Level 3
2-8-10-16
3-4-11-13
5-6-7-9-15
12-14
Level 2
2-8-10-12-14-16
3-4-5-6-7-9-11-13-15
Level 1
2-3-4-5-6-7-8-9-10-11-12-13-14-15-16

```

Figure 5.6 Deletion of node 1

Then, node 1 is again inserted with membership vector 000. Initially membership vector of node 2 is 111 and its probability vector is {0.9, 0.9, 0.9}. When node 2 searches node 9 having membership vector 000, the probability vector of node 2 changes. The threshold for changing the membership vector is set to 20% i.e. after 3.2 calls the probability vector should decrease by a factor of 0.4 leading to a change in the membership vector. Thus, on every call the probability vector is decremented by 0.125 (0.4/3.2). The membership and probability vector of node 2 when it calls node 9, four times is shown in Table 5.1.

Table 5.1 Membership and probability vector of node 2 when it calls node 9

| Scenario     | Membership Vector | Probability Vector  |
|--------------|-------------------|---------------------|
| Initial      | 111               | {0.9,0.9,0.9}       |
| After Call 1 | 111               | {0.775,0.775,0.775} |
| After Call 2 | 111               | {0.65,0.65,0.65}    |
| After Call 3 | 111               | {0.525,0.525,0.525} |
| After Call 4 | 000               | {0.4,0.4,0.4}       |

When node 2 searches node 9 for the first time, it has to scan all the 4 levels (Figure 5.7):

```

C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\1. apskip graph...
Complete Skip Graph Created
Press enter to search node 9 from node 2
Node 9 found at 0xa038e8 after searching at 4 levels.
Node 2 details:
Levels 4
Flag 1
Pointers
Left Right
NULL 10
NULL 8
NULL 4
1 3
Membership Vector Probability
1 0.775
1 0.775
1 0.775
Membership Vector Decimal Equivalent 7
Frequency -1
Timestamp 23485122205
Press enter to search node 9 from node 2

```

Figure 5.7 First search of node 9 by node 2

Then, node 9 is searched for 3 more times by node 2. The output on the fourth search is (Figure 5.8):

```

C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\1. apskip graph...
Press enter to search node 9 from node 2
Node 9 found at 0xa038e8 after searching at 4 levels.
Node 2 details:
Levels 4
Flag 1
Pointers
Left      Right
NULL      10
NULL      8
NULL      4
1         3
Membership Vector      Probability
1                     0.4
1                     0.4
1                     0.4
Membership Vector Decimal Equivalent 7
Frequency -4
Timestamp 2348512205
Node 2 deleted successfully.

Complete Skip Graph
Level 4
1-9
3-11
4-12
5-13
6-14
7-15
8-16
10
Level 3
1-7-9-15
3-5-11-13
4-6-12-14
8-10-16
Level 2
1-3-5-7-9-11-13-15
4-6-8-10-12-14-16
Level 1
1-3-4-5-6-7-8-9-10-11-12-13-14-15-16

New Node 2 created at 0xa03a28 by node 1
Node 2 details:
Levels 4
Flag 0
Pointers
Left      Right
1         9
1         7
1         3
1         3
Membership Vector      Probability
0                     0.4
0                     0.4
0                     0.4
Membership Vector Decimal Equivalent 0
Frequency 0
Timestamp 24008053584

Complete Skip Graph
Level 4
1-2-9
3-11
4-12
5-13
6-14
7-15
8-16
10
Level 3
1-2-7-9-15
3-5-11-13
4-6-12-14
8-10-16
Level 2
1-2-3-5-7-9-11-13-15
4-6-8-10-12-14-16
Level 1
1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16

Press enter to search node 9 from node 2

```

Figure 5.8 Fourth search of node 9 by node 2

After fourth call, membership vector of caller node 2 becomes equal to membership vector of callee node 9. Node 2 is deleted and inserted with membership vector of 000 and probability vector of {0.4, 0.4, 0.4}. Now, node 2 & node 9 are connected at the highest level. If node 2 again searches node 9, it is found at the topmost level. Fifth search of node 9 by node 2 is shown in Figure 5.9:

```

C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\1. apskip graph...
Press enter to search node 9 from node 2
Node 9 found at 0xa038e8 after searching at 1 levels.
Node 2 details:
Levels 4
Flag 1
Pointers
Left      Right
1         9
1         7
1         3
1         3
Membership Vector      Probability
0                     0.275
0                     0.275
0                     0.275
Membership Vector Decimal Equivalent 0
Frequency 1
Timestamp 26571568442
Press enter to search node 9 from node 2

```

Figure 5.9 Fifth search of node 9 by node 2

Similar experiment is performed for 1024 nodes having 10 levels, membership vector of 9 bits and threshold set equal to 10% of the calls. The membership vector of each node is given in Appendix A. Membership vectors have been decided keeping in concern that the all the similar membership vectors are at the same maximum distance at level 0 and membership vector of a node's next neighbor at level 0 is the compliment of its own vector, so that they are not connected at any other level. Membership vector of node 512 is 000000000 and node 513 is 111111111. When node 512 searches nodes 513, it is found after searching all the 10 levels. This process is repeated for 103 times and on 103<sup>rd</sup> call, the threshold is crossed and membership vector of node 512 becomes equal to the membership vector of node 513 and then they are connected at the highest level. Membership and probability vector of node 512 during this process is given in Table 5.2. The experiment is also repeated after setting the threshold to 5% of the calls. In this case after 52 calls, the threshold is crossed and on 53<sup>rd</sup> call the search is done in O(1) time. Similar experiments were repeated for different values of threshold and random membership vectors and the results were verified.

Table 5.2 Membership and probability vector of node 512 when it searches node 513

| Scenario       | Membership Vector | Probability Vector                       |
|----------------|-------------------|------------------------------------------|
| Initial        | 000000000         | {0.1,0.1,...,0.1}                        |
| After Call 1   | 000000000         | {0.10390625, 0.10390625,...,0.10390625}  |
| After Call 2   | 000000000         | {0.1078125, 0.1078125,...,0.1078125}     |
| After Call 3   | 000000000         | {0.11171875, 0.11171875...., 0.11171875} |
| After Call 101 | 000000000         | {0.49453125, 0.49453125....,0.49453125}  |
| After Call 102 | 000000000         | {0.4984375, 0.4984375....,0.4984375}     |
| After Call 103 | 111111111         | {0.50234375,0.50234375....,0.50234375}   |

Replacement algorithms suggested in Section 4.2.4 have also been implemented successfully. Initially node 2 calls node 9 for 14 times and as a result membership vector of node 2 becomes same as node 9 and they are connected at the highest level. Then, node 3 calls node 1 for 12 times. Finally, the membership vector of nodes 1, 2, 3 and 9 becomes same and the skip graph would look like (Figure 5.10):

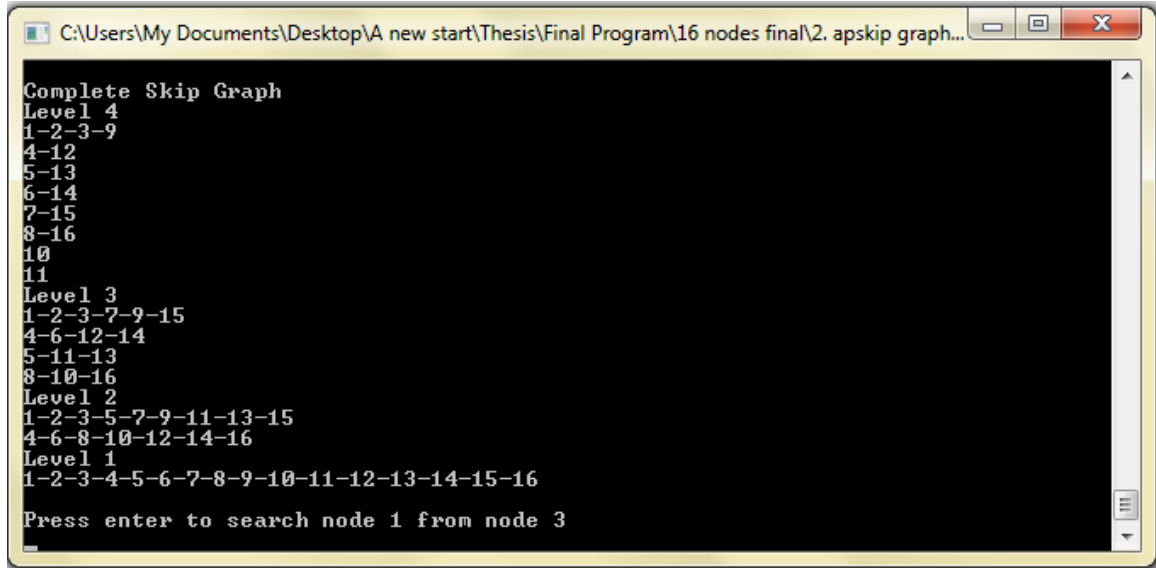
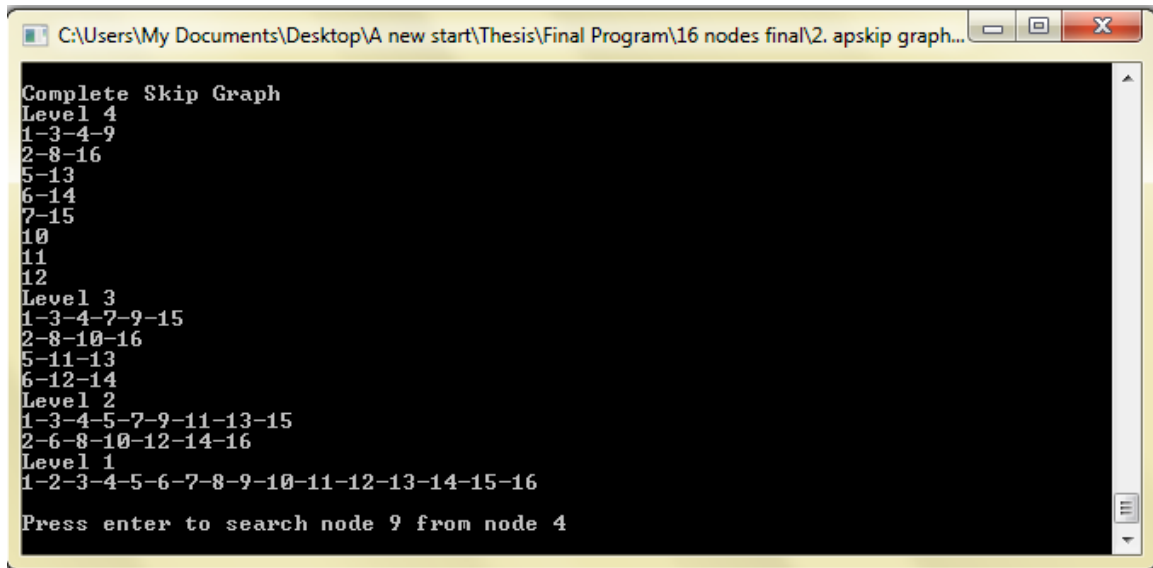


Figure 5.10 Skip graph after 14 calls from node 2 to node 9 and 12 calls from node 3 to node 1

Now, when node 4 calls node 9 multiple times, on 4<sup>th</sup> call, if least recently used replacement policy is used, node 2 is replaced as shown in Figure 5.11.



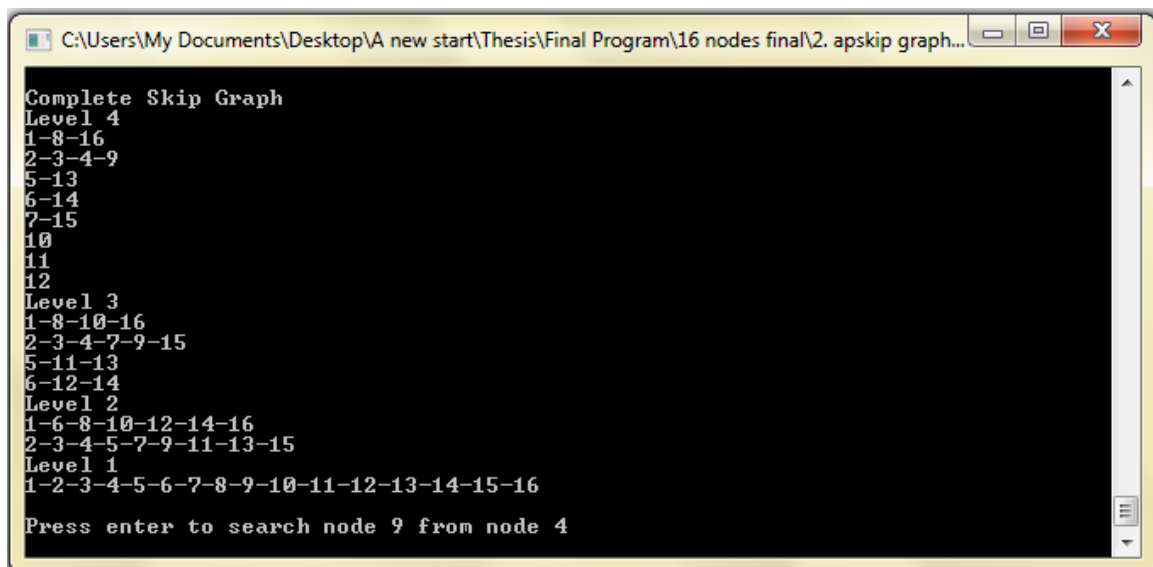
```

C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\2. apskip graph...
Complete Skip Graph
Level 4
1-3-4-9
2-8-16
5-13
6-14
7-15
10
11
12
Level 3
1-3-4-7-9-15
2-8-10-16
5-11-13
6-12-14
Level 2
1-3-4-5-7-9-11-13-15
2-6-8-10-12-14-16
Level 1
1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16
Press enter to search node 9 from node 4

```

Figure 5.11 Skip graph when least recently used replacement policy is used

If least frequently used replacement policy is used, node 1 is replaced as shown in Figure 5.12.



```

C:\Users\My Documents\Desktop\A new start\Thesis\Final Program\16 nodes final\2. apskip graph...
Complete Skip Graph
Level 4
1-8-16
2-3-4-9
5-13
6-14
7-15
10
11
12
Level 3
1-8-10-16
2-3-4-7-9-15
5-11-13
6-12-14
Level 2
1-6-8-10-12-14-16
2-3-4-5-7-9-11-13-15
Level 1
1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16
Press enter to search node 9 from node 4

```

Figure 5.12 Skip graph when least frequently used replacement policy is used

Thus, all the changes proposed in Section 4.2 have been successfully implemented.

### 5.3 Comparison of Basic and Adaptive Probabilistic Skip Graph

Skip graphs and adaptive probabilistic skip graphs can be compared as follows:

1. Search Complexity: In skip graphs, the search complexity is  $O(\log n)$  and it does not change even when the same node is searched again and again by a certain node. It always starts the search from the scratch, no matter how many times the search is carried out. Search complexity reduces drastically in the above situation if adaptive probabilistic skip graphs are used.
2. Extra variables: Adaptive probabilistic skip graphs use a probability vector having size equal to the size of membership vector. It also uses frequency and timestamp as extra variables. Skip graphs do not use any such variables.
3. Congestion: Large number of packets leads to congestion in the network while using skip graphs. Adaptive probabilistic skip graphs reduce the number of packets in the network drastically.

## Chapter 6 Conclusion and Future Scope

---

### 6.1 Conclusion

Skip graphs are one of the best candidates for peer to peer networks because of their dynamic nature, fault tolerance, self-stabilization, data availability and load balancing. The complexity for searching a node in skip graph is  $O(\log n)$ . The skip graph does not adapt itself according to the most frequent search queries. The search complexity remains the same in all the situations which leads to congestion in the network. Thus, a new data structure “Adaptive Probabilistic Skip Graph” has been proposed which adapts itself according to the search queries using a probability vector. It has been proved that in a skip graph of 16 nodes, search complexity reduces drastically when any node is queried for more than 4 times by the same node if the threshold for changing the membership vector is set to 20% of the calls. For a skip graph having 1024 nodes, threshold set at 10% of the calls, the search complexity reduces significantly when the same search is carried out for more than 103 times. Two replacement policies, least recently used and least frequently used have also been proposed and implemented to avoid a situation in which number of nodes greater than  $\log n$  have the same membership vector.

### 6.2 Summary of Contributions

By reducing the search complexity of the most frequent searches, the congestion in the network reduces drastically. Reduction in search complexity of the most frequent searches would also help to increase the efficiency of all the varied applications in which skip graphs are used including dynamic overlay networks, peer to peer networks, cloud computing, web services, social networking, file sharing, multiplayer games, ad-hoc networks, grid computing, mobile sensor networks, biological networks and neural networks.

### 6.3 Future Scope

Currently, the implementation has been done on a single system. It does not consider factors like internet congestion and jitter. However, the experiment can be performed in real world scenario by doing minor changes in the program. The same can also be done on Hadoop. Further, in future the probability should also consider weighted averages i.e.

when the same node is queried again and again, on every subsequent call, the change in probability should increase. Moreover, to further validate the results testing could have been done on more number of nodes.

## References

---

- [1] Andrews, Gregory R., *"Foundations of multithreaded, parallel, and distributed programming"*, Addison–Wesley, 2000
- [2] Linial, Nathan, *"Locality in distributed graph algorithms"*, SIAM Journal on Computing, 1992, pp. 193–201
- [3] Keidar I., *"Distributed computing column 32 – The year in review"*, ACM SIGACT News, 2008
- [4] Garg, Vijay K., *"Elements of distributed computing"*, Wiley-IEEE, 2002
- [5] Herlihy, Maurice P., Shavit, Nir N., *"The art of multiprocessor programming"*, ACM, 2008
- [6] Coulouris G., Dollimore J., Kindberg T., Blair G., *"Distributed systems: Concepts and design"*, (5th Edition), Boston, Addison-Wesley, 2011
- [7] *"Peer to peer networks"* available [online]: *"[http://sop.inria.fr/mistral/personnel/Robin/Groenevelt/Publications/Peer-to\\_Peer\\_Introduction\\_Feb.ppt](http://sop.inria.fr/mistral/personnel/Robin/Groenevelt/Publications/Peer-to_Peer_Introduction_Feb.ppt)"*, [Accessed 10 June, 2015]
- [8] *"Client Server Architecture"* available [online]: *"<http://calstatela.edu/faculty/areed2/CIS561/PPTs/Chapter2Slides.ppt>"*, [Accessed 10 June, 2015]
- [9] Rudiger S., *"A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications"*, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE, 2002
- [10] *"Peer to peer routing"* available [online]: *"<http://project-iris.net/talks/dht-toronto-03.ppt>"*, [Accessed 10 June, 2015]
- [11] Deo N., *"Graph theory with applications to engineering and computer science"*, PHI Learning Pvt. Ltd., 2004, pp. 1-8
- [12] Pugh W., *"Skip lists: a probabilistic alternative to balanced trees"*, Communications of the ACM 33.6, 1990, pp. 668-676
- [13] Goodrich M.T., Nelson M.J., Sun J.Z., *"The rainbow skip graph: A fault-tolerant constant-degree distributed data structure"*, SIAM, May 2009
- [14] Aspnes J., Shah G., *"Skip graphs"*, Proceedings of The Fourteenth Annual ACM-SIAM Symposium on Discrete algorithms, 2003, pp. 384– 393

- [15] Aspnes J., Wieder U., *"The expansion and mixing time of skip graphs with applications"*, Proceedings of The Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2005, pp. 126–134
- [16] Patange V., Gatade D.D., *"Survey of load balancing approaches in peer-to-peer network"*, International Journal of Soft Computing and Engineering (IJSCE), 3.2, 2013, pp. 422-424
- [17] Harvey N., Dunagan J., Jones M.B., Saroiu S., Theimer M., Wolman A., *"Skipnet: A scalable overlay network with practical locality properties"*, 4th USENIX Symposium on Internet Technologies and Systems, USITS, 2003
- [18] Arge L., Eppstein D., Goodrich M.T., *"Skip-webs: Efficient distributed data structures for multi-dimensional data sets"*, Proceedings of the Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, 2009, pp. 69–76
- [19] Ghosh S., *"Distributed Systems – An algorithmic approach"*, Chapman & Hall/CRC, 2007
- [20] Attiya, Hagit, Welch, Jennifer, *"Distributed computing: fundamentals, simulations, and Advanced Topics"*, Wiley-Interscience, 2004
- [21] Joseph R., Kubiawicz J., Rao S., Zhao B.Y., *"Tapestry: An infrastructure for fault tolerant wide area location and routing"*, Anthony Technical Report UCB/CSD-01-1141 , University of California, Berkeley, 2001
- [22] Ratnasamy S., Francis P., Handley M., Carp R., Shenker S., *"A scalable content addressable network"*, In Proceedings of the ACM SIGCOMM, 2001, pp. 161-170
- [23] Malkhi D., Naorand M., Ratajczk D., *"Viceroy: A scalable and dynamic emulation of the butterfly"*, 21st ACM Symposium on Principles of Distributed Computing, 2002, pp. 183-192
- [24] Rowstron A., Druchel P., *"Pastry: scalable, distributed object location and routing for large scale peer-to-peer systems"*, Proceedings of 18th IFIP /ACM International Conference of Distributed Systems Platform, 2001
- [25] Stoica I., Morris R., Karger D., Kaashoek F., Balakrishna H., *"Chord: A scalable peer-to-peer lookup service for internet applications"*, Proceedings of the SIGCOMM, 2001, pp. 149-160

- [26] Brown M.R., Tarjan R.E., *"Design and analysis of a data structure for representing sorted lists"*, SIAM Journal of Computing, 9.3, 1980, pp. 594-614
- [27] Papadakis T. , Munro J.I., Poblete P.V., *"Analysis of the expected search cost in skip lists"*, Proceedings of The Second Scandinavian Workshop on Algorithm Theory, 1990, pp. 160-172
- [28] Pugh W., *"Skip list cookbook"*, available [online]: *"http://drum.lib.umd.edu/handle/1903/544"*, 1998 [Accessed 15 January, 2015]
- [29] Gabarro J., Martnez C., Messeguer X., *"A top-down design of a parallel dictionary using skip lists"*, Theoretical Computer Science 158.12, 1996, pp. 75-82
- [30] Aspnes J., Shah G., *"Skip graphs"* available [online]: *"http://www.cs.yale.edu/Skip Graphs"*, Department of Computer Science, Yale University, 2006 [Accessed 15 January, 2015]
- [31] Mendes H., Fernandes C.G., *"A concurrent implementation of skip graphs"*, Electronic Notes in Discrete Mathematics, 2009, pp. 263-268
- [32] Bidyarthi A., *"Skip graphs and its applications"*, available [online]: *"http://www.academia.edu/6421291/Skip\_Graphs\_and\_its\_Applications"*, Department of Mathematics, IIT Guwahati, 2012 [Accessed 15 January, 2015]
- [33] Aspnes J., ZoDiamadi, Shah G., *"Fault-tolerant routing in peer-to-peer systems"* Proceedings of The Twenty-First ACM Symposium on Principles of Distributed Computing (PODC), 2002, pp. 223-232
- [34] Clouser T., Nesterenko M., Scheideler C., *"A self-stabilizing deterministic skip list and skip graph"*, Elsevier, 2012, pp. 18-35
- [35] Fuminori M., Tatsuhiko T., Tohru K., *"Balance and proximity-aware skip graph construction"*, First International Conference on Networking and Computing, 2010
- [36] Shabeera T.P., Chandran P., Kumar M., *"Authenticated and persistent skip graph: A data structure for cloud based data-centric applications"*, ACM, 2012
- [37] Yu J., Su H., Zhou G., KeXu, *"SNet: skip graph based semantic web services discovery"*, ACM, 2007
- [38] Wasserman S., Faust K., *"Social network analysis"*, Cambridge University Press, Cambridge, UK, 1994

- [39] Jiang X., *"A feasibility study of using peer-to-peer architecture to support massively multiplayer online role playing games"*, available [online]: *"http://ro.uow.edu.au/theses/3931"*, Doctor of Philosophy thesis, School of Electrical, Computer and Telecommunications Engineering, University of Wollongong, 2012 [Accessed 15 January, 2015]
- [40] Morgan K., Plaszcak, P., Rich W., *"Grid Computing The Savvy Manager's Guide"*, ACM, 2006.
- [41] Brault G.J., Augeri C.J., Mullins B.E., Baldwin R.O., Mayer C.B., *"Enabling skip graphs to process k-dimensional range queries in a mobile sensor network"*, Network Computing and Applications, 2007, pp. 273-282
- [42] Hopfield J.J., *"Neural networks and physical systems with emergent collective computational abilities"*, Proceedings of National Academy of Sciences, 1982

## List of Publications and Video Link

---

### Publications

- **Accepted**

Goyal A., Batra S., “*Adaptive probabilistic skip graph*”, IEEE International Advance Computing Conference, June 2015.

### Video Link

[http://bit.do/adaptive\\_probabilistic\\_skip\\_graph](http://bit.do/adaptive_probabilistic_skip_graph)

### **January 2015 – A New Start**

After understanding the basics of recommender system and doing a little research on natural language processing for Sanskrit in the past year, I could not find a very good problem to work upon. In January, I read a paper suggested by my guide, written by James Aspnes and Gauri Shah on Skip Graphs. I understood the importance and working of the skip graphs from that paper. Then, I read few more papers to understand the applications and benefits of skip graphs and to find out what kind of research is going on in this area. I found a lot of research is being carried out to modify the skip graphs so that they suit the requirements of the application concerned. But no attention has been paid to improve the search complexity of the skip graph in case same search is carried out multiple times which may lead to congestion in the network. I thought of finding out a solution of this problem. So, by the end of January, I was able to find out a good problem to work upon. Along with this, I became the Proctor of Hostel J for the second consecutive tenure. This happened for the very first time in Hostel J, that Proctor's tenure was extended. This time the challenge was different. In the previous tenure, I learnt how to work and understood the system. In this tenure, I had to work in a direction to further improve the system.

### **February 2015 – In Search for a Solution**

Next, I started looking out for a probabilistic solution of the problem. Initially, I thought of applying Bayes but could not find its relevance here as space complexity would have increased drastically. This is because Bayes would have required storing the probability of a node being called by every other node in the graph. I had a lot of discussion and brainstorming with my guide. We thought to store the probability of the membership vectors which will require very less space complexity. A lot of discussion was carried out to find out when the probability should change and by what amount. Finally, we developed a simple probabilistic approach to provide a quick solution. By the end of February, I was able to find out the solution of the problem, the next task was to formulate it properly and I also started writing paper on the proposed approach. Meanwhile, my tenure as the President of IEEE Student Chapter, Thapar University

expired. I conducted a number of workshops and seminars during my tenure and I learnt a lot regarding leadership and management skills from that. Moreover, the application process for Ph.D. admission at IITs and IIITs also started. I filled a few applications in February. I also got selected as Assistant Professor in Lovely Professional University.

### **March 2015 - The First Paper and the First Interview**

I formulated the approach, named it Adaptive Probabilistic Skip Graph and wrote a paper on the same. I came to know about IEEE International Advance Computing Conference 2015 to be conducted in Bangalore and submitted my paper there. Simultaneously, I started the implementation of the proposed approach. I tried a lot to assign the maximum number of levels and nodes in the skip graph at run time but could not find any solution even after discussing with the top Codechef rankers in Thapar University. In the end, I took them as macros. I added the functionality of changing the number of current levels dynamically in it. Most of the implementation was done by the end of March. Also, as the Student Coordinator of Linux User Group, Thapar University, I conducted CryptoNyt 2015, a 3 day technical fest. By the end of March, I filled a total of 22 forms for Ph. D. admission at IISC, IITs and IIITs and appeared for the interview at IIT Roorkee which went very well.

### **April 2015 – Completed the Implementation**

In the first few days, I completed the implementation of adaptive probabilistic skip graph and then started testing it. My paper got accepted in IACC and a few modifications were asked by the reviewers which I did by the end of the month. Spardha 2015, a month long sports fest was conducted in the hostel. I also appeared for the interview at IIIT Delhi. I got selected at IIT Roorkee and IIIT Delhi on April 30, a very special day as it is my parents' wedding anniversary.

### **May 2015 – Mission Accomplished**

In May, I tried to resolve one more problem in skip graphs, a situation when more than  $\log n$  number of nodes has the same membership vector. Many ideas came into my mind including nested skip graphs and hybrid skip graphs. For this situation, I finally developed two replacement algorithms: Least recently used and Least frequently used. The major challenge was to find out when the timestamp and frequency needs to be

updated. After extensive discussion with my guide, we finally arrived at a solution. Then, I started implementing it. It was quite a difficult task and took far more time than expected. The major concern was that after adding the new functionality, the behavior of the previous program must not change. For this, I had to do integration testing multiple times. Storing the timestamp was also a tough task. I looked at various alternatives that can serve the purpose. I found few methods which had precision in milliseconds but better precision was required. Few other options which I found could only be implemented on Linux. Finally, I found Query Performance Counter which records the numbers of clock cycles elapsed since the system has started and has precision in microseconds. This also may fail when its value exceeds the range of `LARGE_INTEGER`. For this I thought of using a global clock which gets updated whenever an insertion or a search operation is performed. I implemented both Query Performance Counter and global clock. Finally after implementation, I verified the results. Besides this, executive committee of the hostel organized Hostel J Annual Night which was highly appreciated by the guests. During this month, I also received calls from many institutes for Ph.D. interviews. It was a tough call to decide which interviews to attend since many dates were clashing. I also had an incredible experience of my first airplane ride when I went to IIT Bombay and finally on May 27, I achieved the greatest success of my life till date, I was selected for Ph. D. at IIT Bombay.

### **June 2015 – The Final Touch**

After successful implementation and testing, I started writing text of the thesis. Writing literature review took the maximum time. Then, I made presentation, video, poster and wrote this diary entry. Besides this, I attended Ph. D. interview at IISc Bangalore and it was one of my best interviews ever. I also presented my paper in IACC 2015 via Skype on 13 June, 2015.

This was all about my last semester at Thapar University. Whatever I have done and managed to achieve would not have been possible without the prudent guidance of my guide, Dr. Shalini Batra. I shall always be thankful to her.

## Appendix A: Membership Vector of 1024 nodes

Table A.1 Membership vector of each node in 1024 nodes skip graph

| Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector |
|------|----------------------|------|----------------------|------|----------------------|------|----------------------|
| 0    | 000000000            | 25   | 100011000            | 50   | 011101000            | 75   | 101100110            |
| 1    | 111111111            | 26   | 011111111            | 51   | 100010111            | 76   | 000001000            |
| 2    | 000000001            | 27   | 100000000            | 52   | 010110111            | 77   | 111110111            |
| 3    | 111111110            | 28   | 010111000            | 53   | 101001000            | 78   | 011100100            |
| 4    | 000011010            | 29   | 101000111            | 54   | 011100101            | 79   | 100011011            |
| 5    | 111100101            | 30   | 001100010            | 55   | 100011010            | 80   | 010110110            |
| 6    | 000110011            | 31   | 110011101            | 56   | 001100001            | 81   | 101001001            |
| 7    | 111001100            | 32   | 000011011            | 57   | 110011110            | 82   | 001100000            |
| 8    | 001000010            | 33   | 111100100            | 58   | 000110100            | 83   | 110011111            |
| 9    | 110111101            | 34   | 011100110            | 59   | 111001011            | 84   | 000011100            |
| 10   | 000000011            | 35   | 100011001            | 60   | 011111100            | 85   | 111100011            |
| 11   | 111111100            | 36   | 001000011            | 61   | 100000011            | 86   | 000110101            |
| 12   | 001100011            | 37   | 110111100            | 62   | 011111110            | 87   | 111001010            |
| 13   | 110011100            | 38   | 000000010            | 63   | 100000001            | 88   | 001000100            |
| 14   | 001100100            | 39   | 111111101            | 64   | 011111101            | 89   | 110111011            |
| 15   | 110011011            | 40   | 001100101            | 65   | 100000010            | 90   | 000000100            |
| 16   | 000001101            | 41   | 110011010            | 66   | 010111011            | 91   | 111111011            |
| 17   | 111110010            | 42   | 010111010            | 67   | 101000100            | 92   | 001100111            |
| 18   | 001110110            | 43   | 101000101            | 68   | 001100110            | 93   | 110011000            |
| 19   | 110001001            | 44   | 001110111            | 69   | 110011001            | 94   | 010111100            |
| 20   | 010010111            | 45   | 110001000            | 70   | 001111000            | 95   | 101000011            |
| 21   | 101101000            | 46   | 010000000            | 71   | 110000111            | 96   | 000110010            |
| 22   | 010111001            | 47   | 101111111            | 72   | 010000001            | 97   | 111001101            |
| 23   | 101000110            | 48   | 010011000            | 73   | 101111110            | 98   | 001111001            |
| 24   | 011100111            | 49   | 101100111            | 74   | 010011001            | 99   | 110000110            |

| Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector |
|------|----------------------|------|----------------------|------|----------------------|------|----------------------|
| 100  | 010011010            | 128  | 000001110            | 156  | 001011101            | 184  | 011100000            |
| 101  | 101100101            | 129  | 111110001            | 157  | 110100010            | 185  | 100011111            |
| 102  | 011101001            | 130  | 011101010            | 158  | 011100001            | 186  | 010111111            |
| 103  | 100010110            | 131  | 100010101            | 159  | 100011110            | 187  | 101000000            |
| 104  | 011111011            | 132  | 001011110            | 160  | 010111110            | 188  | 010110010            |
| 105  | 100000100            | 133  | 110100001            | 161  | 101000001            | 189  | 101001101            |
| 106  | 011100011            | 134  | 010111101            | 162  | 010110011            | 190  | 010000100            |
| 107  | 100011100            | 135  | 101000010            | 163  | 101001100            | 191  | 101111011            |
| 108  | 001011111            | 136  | 011100010            | 164  | 000011110            | 192  | 000011111            |
| 109  | 110100000            | 137  | 100011101            | 165  | 111100001            | 193  | 111100000            |
| 110  | 010110101            | 138  | 011111001            | 166  | 010000011            | 194  | 000101110            |
| 111  | 101001010            | 139  | 100000110            | 167  | 101111100            | 195  | 111010001            |
| 112  | 000011101            | 140  | 010110100            | 168  | 001000110            | 196  | 010011101            |
| 113  | 111100010            | 141  | 101001011            | 169  | 110111001            | 197  | 101100010            |
| 114  | 000110110            | 142  | 000110111            | 170  | 000101111            | 198  | 001111101            |
| 115  | 111001001            | 143  | 111001000            | 171  | 111010000            | 199  | 110000010            |
| 116  | 001000101            | 144  | 011111010            | 172  | 001101001            | 200  | 001101011            |
| 117  | 110111010            | 145  | 100000101            | 173  | 110010110            | 201  | 110010100            |
| 118  | 010000010            | 146  | 000110000            | 174  | 001101010            | 202  | 001011011            |
| 119  | 101111101            | 147  | 111001111            | 175  | 110010101            | 203  | 110100100            |
| 120  | 001101000            | 148  | 000001111            | 176  | 001111100            | 204  | 011000000            |
| 121  | 110010111            | 149  | 111110000            | 177  | 110000011            | 205  | 100111111            |
| 122  | 000110001            | 150  | 001111011            | 178  | 001011100            | 206  | 000000110            |
| 123  | 111001110            | 151  | 110000100            | 179  | 110100011            | 207  | 111111001            |
| 124  | 001111010            | 152  | 010011100            | 180  | 011101011            | 208  | 011101100            |
| 125  | 110000101            | 153  | 101100011            | 181  | 100010100            | 209  | 100010011            |
| 126  | 010011011            | 154  | 000000101            | 182  | 011111000            | 210  | 011110111            |
| 127  | 101100100            | 155  | 111111010            | 183  | 100000111            | 211  | 100001000            |

| Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector |
|------|----------------------|------|----------------------|------|----------------------|------|----------------------|
| 212  | 011011111            | 240  | 010000110            | 268  | 010000111            | 296  | 001011000            |
| 213  | 100100000            | 241  | 101111001            | 269  | 101111000            | 297  | 110100111            |
| 214  | 011001111            | 242  | 000101101            | 270  | 000100000            | 298  | 001001001            |
| 215  | 100110000            | 243  | 111010010            | 271  | 111011111            | 299  | 110110110            |
| 216  | 010000101            | 244  | 000111001            | 272  | 000111010            | 300  | 000111011            |
| 217  | 101111010            | 245  | 111000110            | 273  | 111000101            | 301  | 111000100            |
| 218  | 000111000            | 246  | 001001000            | 274  | 001011001            | 302  | 001101110            |
| 219  | 111000111            | 247  | 110110111            | 275  | 110100110            | 303  | 110010001            |
| 220  | 001000111            | 248  | 001101100            | 276  | 001101101            | 304  | 011110000            |
| 221  | 110111000            | 249  | 110010011            | 277  | 110010010            | 305  | 100001111            |
| 222  | 011110110            | 250  | 001011010            | 278  | 011101111            | 306  | 011000100            |
| 223  | 100001001            | 251  | 110100101            | 279  | 100010000            | 307  | 100111011            |
| 224  | 001111110            | 252  | 001111111            | 280  | 011110101            | 308  | 000000111            |
| 225  | 110000001            | 253  | 110000000            | 281  | 100001010            | 309  | 111111000            |
| 226  | 000010000            | 254  | 011101110            | 282  | 011000011            | 310  | 010101110            |
| 227  | 111101111            | 255  | 100010001            | 283  | 100111100            | 311  | 101010001            |
| 228  | 000001001            | 256  | 011000010            | 284  | 011110100            | 312  | 000010010            |
| 229  | 111110110            | 257  | 100111101            | 285  | 100001011            | 313  | 111101101            |
| 230  | 011000001            | 258  | 000010001            | 286  | 011010010            | 314  | 010010110            |
| 231  | 100111110            | 259  | 111101110            | 287  | 100101101            | 315  | 101101001            |
| 232  | 011010000            | 260  | 011010001            | 288  | 010101111            | 316  | 010100000            |
| 233  | 100101111            | 261  | 100101110            | 289  | 101010000            | 317  | 101011111            |
| 234  | 011101101            | 262  | 011011101            | 290  | 000101100            | 318  | 000010101            |
| 235  | 100010010            | 263  | 100100010            | 291  | 111010011            | 319  | 111101010            |
| 236  | 011011110            | 264  | 010110000            | 292  | 010001000            | 320  | 001010111            |
| 237  | 100100001            | 265  | 101001111            | 293  | 101110111            | 321  | 110101000            |
| 238  | 010110001            | 266  | 010011110            | 294  | 010011111            | 322  | 000100001            |
| 239  | 101001110            | 267  | 101100001            | 295  | 101100000            | 323  | 111011110            |

| Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector |
|------|----------------------|------|----------------------|------|----------------------|------|----------------------|
| 324  | 001001010            | 352  | 000101000            | 380  | 001110001            | 408  | 000101001            |
| 325  | 110110101            | 353  | 111010111            | 381  | 110001110            | 409  | 111010110            |
| 326  | 000111100            | 354  | 0011110000           | 382  | 0111110010           | 410  | 010101011            |
| 327  | 111000011            | 355  | 110001111            | 383  | 100001101            | 411  | 101010100            |
| 328  | 001101111            | 356  | 010010011            | 384  | 011011011            | 412  | 011010101            |
| 329  | 110010000            | 357  | 101101100            | 385  | 100100100            | 413  | 100101010            |
| 330  | 000001010            | 358  | 010010100            | 386  | 000101010            | 414  | 011001001            |
| 331  | 111110101            | 359  | 101101011            | 387  | 111010101            | 415  | 100110110            |
| 332  | 011000101            | 360  | 011000110            | 388  | 011010100            | 416  | 010100100            |
| 333  | 100111010            | 361  | 100111001            | 389  | 100101011            | 417  | 101011011            |
| 334  | 010101101            | 362  | 010101100            | 390  | 011011010            | 418  | 001010100            |
| 335  | 101010010            | 363  | 101010011            | 391  | 100100101            | 419  | 110101011            |
| 336  | 010010101            | 364  | 000010011            | 392  | 010100011            | 420  | 000010110            |
| 337  | 101101010            | 365  | 111101100            | 393  | 101011100            | 421  | 111101001            |
| 338  | 000101011            | 366  | 010100010            | 394  | 001010101            | 422  | 010001011            |
| 339  | 111010100            | 367  | 101011101            | 395  | 110101010            | 423  | 101110100            |
| 340  | 010100001            | 368  | 011000111            | 396  | 010001010            | 424  | 010010000            |
| 341  | 101011110            | 369  | 100111000            | 397  | 101110101            | 425  | 101101111            |
| 342  | 011010011            | 370  | 001010110            | 398  | 011001000            | 426  | 000100011            |
| 343  | 100101100            | 371  | 110101001            | 399  | 100110111            | 427  | 111011100            |
| 344  | 010001001            | 372  | 001001100            | 400  | 000010100            | 428  | 001001101            |
| 345  | 101110110            | 373  | 110110011            | 401  | 111101011            | 429  | 110110010            |
| 346  | 011011100            | 374  | 000100010            | 402  | 010010001            | 430  | 000111110            |
| 347  | 100100011            | 375  | 111011101            | 403  | 101101110            | 431  | 111000001            |
| 348  | 011110001            | 376  | 000100111            | 404  | 000111101            | 432  | 001110011            |
| 349  | 100001110            | 377  | 111011000            | 405  | 111000010            | 433  | 110001100            |
| 350  | 001001011            | 378  | 010010010            | 406  | 001110010            | 434  | 010101010            |
| 351  | 110110100            | 379  | 101101101            | 407  | 110001101            | 435  | 101010101            |

| Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector |
|------|----------------------|------|----------------------|------|----------------------|------|----------------------|
| 436  | 011010110            | 464  | 001010010            | 492  | 011001100            | 520  | 001000010            |
| 437  | 100101001            | 465  | 110101101            | 493  | 100110011            | 521  | 110111101            |
| 438  | 011001010            | 466  | 010001110            | 494  | 010101000            | 522  | 000000011            |
| 439  | 100110101            | 467  | 101110001            | 495  | 101010111            | 523  | 111111100            |
| 440  | 010100101            | 468  | 010100110            | 496  | 010100111            | 524  | 001100011            |
| 441  | 101011010            | 469  | 101011001            | 497  | 101011000            | 525  | 110011100            |
| 442  | 001010011            | 470  | 011011000            | 498  | 001110101            | 526  | 001100100            |
| 443  | 110101100            | 471  | 100100111            | 499  | 110001010            | 527  | 110011011            |
| 444  | 010001111            | 472  | 000010111            | 500  | 000011000            | 528  | 000001101            |
| 445  | 101110000            | 473  | 111101000            | 501  | 111100111            | 529  | 111110010            |
| 446  | 010101001            | 474  | 000100110            | 502  | 001010000            | 530  | 001110110            |
| 447  | 101010110            | 475  | 111011001            | 503  | 110101111            | 531  | 110001001            |
| 448  | 010001100            | 476  | 011001110            | 504  | 001000001            | 532  | 010010111            |
| 449  | 101110011            | 477  | 100110001            | 505  | 110111110            | 533  | 101101000            |
| 450  | 011011001            | 478  | 000100100            | 506  | 000100101            | 534  | 010111001            |
| 451  | 100100110            | 479  | 111011011            | 507  | 111011010            | 535  | 101000110            |
| 452  | 011110011            | 480  | 001001111            | 508  | 000011001            | 536  | 011100111            |
| 453  | 100001100            | 481  | 110110000            | 509  | 111100110            | 537  | 100011000            |
| 454  | 001001110            | 482  | 001000000            | 510  | 000001100            | 538  | 011111111            |
| 455  | 110110001            | 483  | 110111111            | 511  | 111110011            | 539  | 100000000            |
| 456  | 000111111            | 484  | 001110100            | 512  | 000000000            | 540  | 010111000            |
| 457  | 111000000            | 485  | 110001011            | 513  | 111111111            | 541  | 101000111            |
| 458  | 000001011            | 486  | 011001101            | 514  | 000000001            | 542  | 001100010            |
| 459  | 111110100            | 487  | 100110010            | 515  | 111111110            | 543  | 110011101            |
| 460  | 011010111            | 488  | 001010001            | 516  | 000011010            | 544  | 000011011            |
| 461  | 100101000            | 489  | 110101110            | 517  | 111100101            | 545  | 111100100            |
| 462  | 011001011            | 490  | 010001101            | 518  | 000110011            | 546  | 011100110            |
| 463  | 100110100            | 491  | 101110010            | 519  | 111001100            | 547  | 100011001            |

| Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector |
|------|----------------------|------|----------------------|------|----------------------|------|----------------------|
| 548  | 001000011            | 576  | 011111101            | 604  | 001100111            | 632  | 001101000            |
| 549  | 110111100            | 577  | 100000010            | 605  | 110011000            | 633  | 110010111            |
| 550  | 000000010            | 578  | 010111011            | 606  | 010111100            | 634  | 000110001            |
| 551  | 111111101            | 579  | 101000100            | 607  | 101000011            | 635  | 111001110            |
| 552  | 001100101            | 580  | 001100110            | 608  | 000110010            | 636  | 001111010            |
| 553  | 110011010            | 581  | 110011001            | 609  | 111001101            | 637  | 110000101            |
| 554  | 010111010            | 582  | 001111000            | 610  | 001111001            | 638  | 010011011            |
| 555  | 101000101            | 583  | 110000111            | 611  | 110000110            | 639  | 101100100            |
| 556  | 001110111            | 584  | 010000001            | 612  | 010011010            | 640  | 000001110            |
| 557  | 110001000            | 585  | 101111110            | 613  | 101100101            | 641  | 111110001            |
| 558  | 010000000            | 586  | 010011001            | 614  | 011101001            | 642  | 011101010            |
| 559  | 101111111            | 587  | 101100110            | 615  | 100010110            | 643  | 100010101            |
| 560  | 010011000            | 588  | 000001000            | 616  | 011111011            | 644  | 001011110            |
| 561  | 101100111            | 589  | 111110111            | 617  | 100000100            | 645  | 110100001            |
| 562  | 011101000            | 590  | 011100100            | 618  | 011100011            | 646  | 010111101            |
| 563  | 100010111            | 591  | 100011011            | 619  | 100011100            | 647  | 101000010            |
| 564  | 010110111            | 592  | 010110110            | 620  | 001011111            | 648  | 011100010            |
| 565  | 101001000            | 593  | 101001001            | 621  | 110100000            | 649  | 100011101            |
| 566  | 011100101            | 594  | 001100000            | 622  | 010110101            | 650  | 011111001            |
| 567  | 100011010            | 595  | 110011111            | 623  | 101001010            | 651  | 100000110            |
| 568  | 001100001            | 596  | 000011100            | 624  | 000011101            | 652  | 010110100            |
| 569  | 110011110            | 597  | 111100011            | 625  | 111100010            | 653  | 101001011            |
| 570  | 000110100            | 598  | 000110101            | 626  | 000110110            | 654  | 000110111            |
| 571  | 111001011            | 599  | 111001010            | 627  | 111001001            | 655  | 111001000            |
| 572  | 011111100            | 600  | 001000100            | 628  | 001000101            | 656  | 011111010            |
| 573  | 100000011            | 601  | 110111011            | 629  | 110111010            | 657  | 100000101            |
| 574  | 011111110            | 602  | 000000100            | 630  | 010000010            | 658  | 000110000            |
| 575  | 100000001            | 603  | 111111011            | 631  | 101111101            | 659  | 111001111            |

| Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector |
|------|----------------------|------|----------------------|------|----------------------|------|----------------------|
| 660  | 000001111            | 688  | 001111100            | 716  | 011000000            | 744  | 011010000            |
| 661  | 111110000            | 689  | 110000011            | 717  | 100111111            | 745  | 100101111            |
| 662  | 001111011            | 690  | 001011100            | 718  | 000000110            | 746  | 011101101            |
| 663  | 110000100            | 691  | 110100011            | 719  | 111111001            | 747  | 100010010            |
| 664  | 010011100            | 692  | 011101011            | 720  | 011101100            | 748  | 011011110            |
| 665  | 101100011            | 693  | 100010100            | 721  | 100010011            | 749  | 100100001            |
| 666  | 000000101            | 694  | 011111000            | 722  | 011110111            | 750  | 010110001            |
| 667  | 111111010            | 695  | 100000111            | 723  | 100001000            | 751  | 101001110            |
| 668  | 001011101            | 696  | 011100000            | 724  | 011011111            | 752  | 010000110            |
| 669  | 110100010            | 697  | 100011111            | 725  | 100100000            | 753  | 101111001            |
| 670  | 011100001            | 698  | 010111111            | 726  | 011001111            | 754  | 000101101            |
| 671  | 100011110            | 699  | 101000000            | 727  | 100110000            | 755  | 111010010            |
| 672  | 010111110            | 700  | 010110010            | 728  | 010000101            | 756  | 000111001            |
| 673  | 101000001            | 701  | 101001101            | 729  | 101111010            | 757  | 111000110            |
| 674  | 010110011            | 702  | 010000100            | 730  | 000111000            | 758  | 001001000            |
| 675  | 101001100            | 703  | 101111011            | 731  | 111000111            | 759  | 110110111            |
| 676  | 000011110            | 704  | 000011111            | 732  | 001000111            | 760  | 001101100            |
| 677  | 111100001            | 705  | 111100000            | 733  | 110111000            | 761  | 110010011            |
| 678  | 010000011            | 706  | 000101110            | 734  | 011110110            | 762  | 001011010            |
| 679  | 101111100            | 707  | 111010001            | 735  | 100001001            | 763  | 110100101            |
| 680  | 001000110            | 708  | 010011101            | 736  | 001111110            | 764  | 001111111            |
| 681  | 110111001            | 709  | 101100010            | 737  | 110000001            | 765  | 110000000            |
| 682  | 000101111            | 710  | 001111101            | 738  | 000010000            | 766  | 011101110            |
| 683  | 111010000            | 711  | 110000010            | 739  | 111101111            | 767  | 100010001            |
| 684  | 001101001            | 712  | 001101011            | 740  | 000001001            | 768  | 011000010            |
| 685  | 110010110            | 713  | 110010100            | 741  | 111110110            | 769  | 100111101            |
| 686  | 001101010            | 714  | 001011011            | 742  | 011000001            | 770  | 000010001            |
| 687  | 110010101            | 715  | 110100100            | 743  | 100111110            | 771  | 111101110            |

| Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector |
|------|----------------------|------|----------------------|------|----------------------|------|----------------------|
| 772  | 011010001            | 800  | 010101111            | 828  | 010100000            | 856  | 010001001            |
| 773  | 100101110            | 801  | 101010000            | 829  | 101011111            | 857  | 101110110            |
| 774  | 011011101            | 802  | 000101100            | 830  | 000010101            | 858  | 011011100            |
| 775  | 100100010            | 803  | 111010011            | 831  | 111101010            | 859  | 100100011            |
| 776  | 010110000            | 804  | 010001000            | 832  | 001010111            | 860  | 011110001            |
| 777  | 101001111            | 805  | 101110111            | 833  | 110101000            | 861  | 100001110            |
| 778  | 010011110            | 806  | 010011111            | 834  | 000100001            | 862  | 001001011            |
| 779  | 101100001            | 807  | 101100000            | 835  | 111011110            | 863  | 110110100            |
| 780  | 010000111            | 808  | 001011000            | 836  | 001001010            | 864  | 000101000            |
| 781  | 101111000            | 809  | 110100111            | 837  | 110110101            | 865  | 111010111            |
| 782  | 000100000            | 810  | 001001001            | 838  | 000111100            | 866  | 001110000            |
| 783  | 111011111            | 811  | 110110110            | 839  | 111000011            | 867  | 110001111            |
| 784  | 000111010            | 812  | 000111011            | 840  | 001101111            | 868  | 010010011            |
| 785  | 111000101            | 813  | 111000100            | 841  | 110010000            | 869  | 101101100            |
| 786  | 001011001            | 814  | 001101110            | 842  | 000001010            | 870  | 010010100            |
| 787  | 110100110            | 815  | 110010001            | 843  | 111110101            | 871  | 101101011            |
| 788  | 001101101            | 816  | 011110000            | 844  | 011000101            | 872  | 011000110            |
| 789  | 110010010            | 817  | 100001111            | 845  | 100111010            | 873  | 100111001            |
| 790  | 011101111            | 818  | 011000100            | 846  | 010101101            | 874  | 010101100            |
| 791  | 100010000            | 819  | 100111011            | 847  | 101010010            | 875  | 101010011            |
| 792  | 011110101            | 820  | 000000111            | 848  | 010010101            | 876  | 000010011            |
| 793  | 100001010            | 821  | 111111000            | 849  | 101101010            | 877  | 111101100            |
| 794  | 011000011            | 822  | 010101110            | 850  | 000101011            | 878  | 010100010            |
| 795  | 100111100            | 823  | 101010001            | 851  | 111010100            | 879  | 101011101            |
| 796  | 011110100            | 824  | 000010010            | 852  | 010100001            | 880  | 011000111            |
| 797  | 100001011            | 825  | 111101101            | 853  | 101011110            | 881  | 100111000            |
| 798  | 011010010            | 826  | 010010110            | 854  | 011010011            | 882  | 001010110            |
| 799  | 100101101            | 827  | 101101001            | 855  | 100101100            | 883  | 110101001            |

| Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector |
|------|----------------------|------|----------------------|------|----------------------|------|----------------------|
| 884  | 001001100            | 912  | 000010100            | 940  | 001001101            | 968  | 000111111            |
| 885  | 110110011            | 913  | 111101011            | 941  | 110110010            | 969  | 111000000            |
| 886  | 000100010            | 914  | 010010001            | 942  | 000111110            | 970  | 000001011            |
| 887  | 111011101            | 915  | 101101110            | 943  | 111000001            | 971  | 111110100            |
| 888  | 000100111            | 916  | 000111101            | 944  | 001110011            | 972  | 011010111            |
| 889  | 111011000            | 917  | 111000010            | 945  | 110001100            | 973  | 100101000            |
| 890  | 010010010            | 918  | 001110010            | 946  | 010101010            | 974  | 011001011            |
| 891  | 101101101            | 919  | 110001101            | 947  | 101010101            | 975  | 100110100            |
| 892  | 001110001            | 920  | 000101001            | 948  | 011010110            | 976  | 001010010            |
| 893  | 110001110            | 921  | 111010110            | 949  | 100101001            | 977  | 110101101            |
| 894  | 011110010            | 922  | 010101011            | 950  | 011001010            | 978  | 010001110            |
| 895  | 100001101            | 923  | 101010100            | 951  | 100110101            | 979  | 101110001            |
| 896  | 011011011            | 924  | 011010101            | 952  | 010100101            | 980  | 010100110            |
| 897  | 100100100            | 925  | 100101010            | 953  | 101011010            | 981  | 101011001            |
| 898  | 000101010            | 926  | 011001001            | 954  | 001010011            | 982  | 011011000            |
| 899  | 111010101            | 927  | 100110110            | 955  | 110101100            | 983  | 100100111            |
| 900  | 011010100            | 928  | 010100100            | 956  | 010001111            | 984  | 000010111            |
| 901  | 100101011            | 929  | 101011011            | 957  | 101110000            | 985  | 111101000            |
| 902  | 011011010            | 930  | 001010100            | 958  | 010101001            | 986  | 000100110            |
| 903  | 100100101            | 931  | 110101011            | 959  | 101010110            | 987  | 111011001            |
| 904  | 010100011            | 932  | 000010110            | 960  | 010001100            | 988  | 011001110            |
| 905  | 101011100            | 933  | 111101001            | 961  | 101110011            | 989  | 100110001            |
| 906  | 001010101            | 934  | 010001011            | 962  | 011011001            | 990  | 000100100            |
| 907  | 110101010            | 935  | 101110100            | 963  | 100100110            | 991  | 111011011            |
| 908  | 010001010            | 936  | 010010000            | 964  | 011110011            | 992  | 001001111            |
| 909  | 101110101            | 937  | 101101111            | 965  | 100001100            | 993  | 110110000            |
| 910  | 011001000            | 938  | 000100011            | 966  | 001001110            | 994  | 001000000            |
| 911  | 100110111            | 939  | 111011100            | 967  | 110110001            | 995  | 110111111            |

| Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector | Node | Membership<br>Vector |
|------|----------------------|------|----------------------|------|----------------------|------|----------------------|
| 996  | 001110100            | 1003 | 101110010            | 1010 | 001110101            | 1017 | 110111110            |
| 997  | 110001011            | 1004 | 011001100            | 1011 | 110001010            | 1018 | 000100101            |
| 998  | 011001101            | 1005 | 100110011            | 1012 | 000011000            | 1019 | 111011010            |
| 999  | 100110010            | 1006 | 010101000            | 1013 | 111100111            | 1020 | 000011001            |
| 1000 | 001010001            | 1007 | 101010111            | 1014 | 001010000            | 1021 | 111100110            |
| 1001 | 110101110            | 1008 | 010100111            | 1015 | 110101111            | 1022 | 000001100            |
| 1002 | 010001101            | 1009 | 101011000            | 1016 | 001000001            | 1023 | 111110011            |