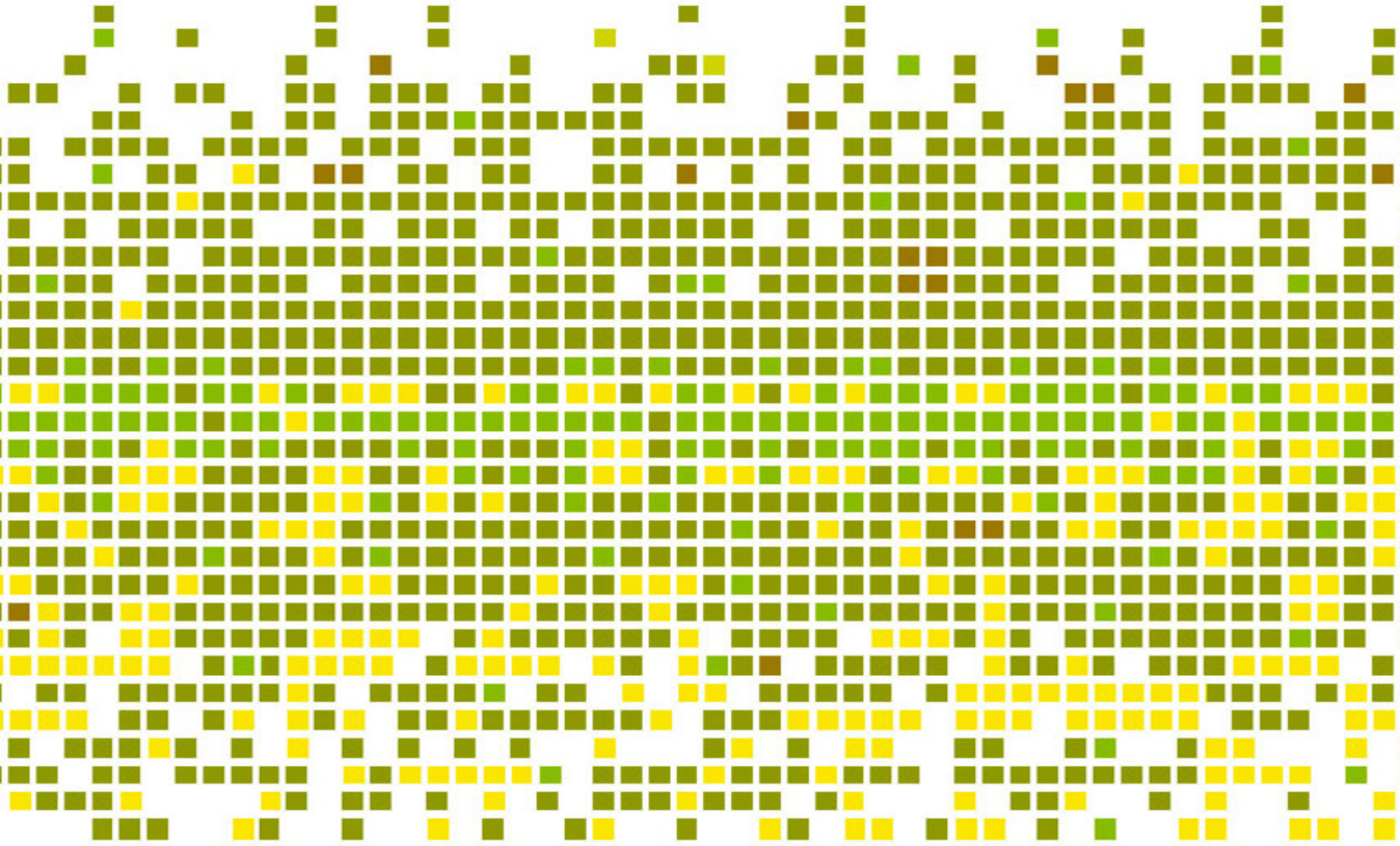
Volume 5 ▪ Issue 1 ▪ July 2015

INTERNATIONAL JOURNAL OF

EXPERIMENTAL ALGORITHMS (IJEA)

ISSN : 2180-1282

Publication Frequency: 6 Issues / Year



CSC PUBLISHERS
http://www.cscjournals.org

INTERNATIONAL JOURNAL OF EXPERIMENTAL ALGORITHMS (IJEA)

VOLUME 5, ISSUE 1, 2015

EDITED BY
DR. NABEEL TAHIR

ISSN (Online): 2180-1282

International Journal of Experimental Algorithms (IJEA) is published both in traditional paper form and in Internet. This journal is published at the website http://www.cscjournals.org, maintained by Computer Science Journals (CSC Journals), Malaysia.

IJEA Journal is a part of CSC Publishers
Computer Science Journals
http://www.cscjournals.org

INTERNATIONAL JOURNAL OF EXPERIMENTAL ALGORITHMS (IJEA)

Book: Volume 5, Issue 1, July 2015
Publishing Date: 31-07-2015
ISSN (Online): 1985-4129

IJEA Journal is a part of CSC Publishers
http://www.cscjournals.org

Published in Malaysia

Typesetting: Camera-ready by author, data conversation by CSC Publishing Services – CSC Journals, Malaysia

CSC Publishers, 2015

EDITORIAL BOARD

TABLE OF CONTENTS

Volume 5, Issue 1, July 2015

Pages

Algorithm for Edge Antimagic Labeling for Specific Classes of Graphs

Nissankara Lakshmi Prasanna *prasanna.manu@gmail.com*
Research scholar of ANU & Asst.professor of Vignan's Lara Institution of Technology and Science/CSE Guntur, India

Nagalla Sudhakar *suds.nagalla@gmail.com*
Professor & Principal of Bapatla Engineering College/CSE Guntur, India

Abstract

Graph labeling is a remarkable field having direct and indirect involvement in resolving numerous issues in varied fields. During this paper we tend to planned new algorithms to construct edge antimagic vertex labeling, edge antimagic total labeling, (a, d)-edge antimagic vertex labeling, (a, d)-edge antimagic total labeling and super edge antimagic labeling of varied classes of graphs like paths, cycles, wheels, fans, friendship graphs. With these solutions several open issues during this space can be solved.

Keywords: Graph Labeling, Edge Antimagic Vertex Labelling, Edge Antimagic Total Labelling, Super Edge Antimagic Labelling, Paths, Cycles, Fan Graphs, Wheels, Friendship Graphs.

1. INTRODUCTION

A graph G = (V, E) is finite, straightforward and un-directed. G denotes graph then G includes a vertex and edge sets. Vertex set denoted by V = V (G) and edge set E = E (G). We followed the standard notations m = |E| and n = |V|. A typical graph theoretical notation is followed refer [13]. Labelled graphs are getting associate more and more helpful family of Mathematical Models for a broad vary of applications [4, 17]. It's terribly crucial impact in network communications explained in [5]. Several latest applications presented its usage to image authentication and frequency allocation.

Graph Labeling is a method of mapping that maps several set of elements of graph to a collection of numbers (usually +ve or non -ve integers). The foremost complete graph labeling latest survey is in [3] [14]. Sedlacek [9] introduce labelings that simplify the thought of a magic labeling. The magic labelings is outlined as a bijection of graph component to set of successive integers ranging from one, satisfying some reasonably "constant sum" property. If this Bijection involves vertices or edges or both as graph elements to a collection of integers yielding a constant sum known as magic constant, it'll be known as Vertex or Edge or Total Magic Labeling. Hartsfield along with Ringel in [6] introduced the idea of an Antimagic graphs. In step with them associate Antimagic labeling is a process of edge labeling of the graph with integers 1,2,…., m in order that weight at every vertex is totally different from the weight at every vertex.

In [7] Bodendiek with Walter outlined the thought of an (a, d)-Antimagic labeling as edge labeling during which the weights of vertex from an AP(arithmetic progression) ranging from a and have common distinction d. Martin Baca, Francois Bertault and MacDougall [8] initiated the notions of the Vertex Antimagic Total Labeling [VATL] and (a, d)-Vertex Antimagic Total Labeling [(a, d)-VATL], and conjuncture that every regular graphs are (a, d)- VATL. In the year 2004, K.A.Sugeng et al. [10] presented the concept of SVMTL (super vertex magic total labeling) & SEMTL (super edge magic total labeling). The existence of In [11] Antimagic vertex labeling of categories of hyper graphs like Cycles, Wheels and therefore the existence and non-existence of the Antimagic

vertex labeling of Wheels are mentioned in theorems. An idea to get antimagic labeling for trees given in [12]. In reference [2] they projected the procedure (algorithms) to build (a, d)-Antimagic labeling, Antimagic labeling, (a, d)-vertex Antimagic total labeling of complete graphs and vertex Antimagic total labeling that could be a generalization of many different kinds of labelings.

In [15], we deal with the magic labeling of vertices and edges of a graph. Again magic labeling is expressed in-terms of Vertex Magic Total Labeling (VMTL), Edge Total Magic Labeling (EMTL) and Total Magic Labeling (TML). We have studied existing approaches for magic labeling and we found some improvements can be done over existing VMTL algorithms and we design algorithm to find EMTLs. We proposed new and enhanced algorithms for VMTL, EMTL and TML. We applied these algorithms on different kinds of graphs like cycles, wheels, fans and friendship graphs. In[16] we proposed new algorithms to construct vertex antimagic edge labeling, (a, d)-vertex antimagic labeling, vertex antimagic total labeling and (a, d)-vertex antimagic total labeling and super vertex-antimagic total labeling of various classes of graphs like paths, cycles, wheels, Fan Graph and Friend Graphs.

In continuation to this, we have projected algorithms for EATL (Edge Antimagic Total labeling) on different categories of graphs like cycles, paths, wheels, and fan and Friendship graphs. These algorithms are similar all categories with minor changes because of structural variations. With these we have to study the behaviour of the graphs with specific graph size. For given Graph with size we will determine the attainable labelings, possible values of a and d to create (a, d) Edge opposed magic Total Labelings and also the chance of forming SEMTL (super edge magic total labeling).

2. PRELIMINARIES

Standard definitions of paths and wheels and cycles, fan and Friendship graphs are as follows. A Path (P_n) could be a cycle without an edge from initial vertex to final vertex. Cycle could be a graph wherever there's an edge amid the neighbouring vertices solely and therefore the vertex is adjacent to final one (Fig1a). Wheel could be a Cycle with central hub, wherever all vertices of cycle are neighbouring to that (Fig1b) Fans & Friendship graphs are variations of wheels. If a path is linked to central hub it's a Fan (Fig1c). A Friendship graph has n triangles with 1 common vertex known as hub and n is size of Friendship graph (Fig 1d)

FIGURE 1a: Path (P_4)

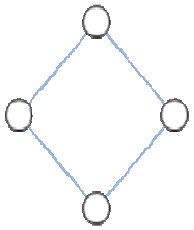


FIGURE 1b: Cycle(C4)

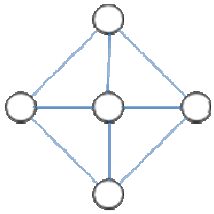


FIGURE 1c: Wheel(W4)

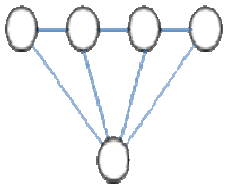


FIGURE 1d: Fan Graph(F4)

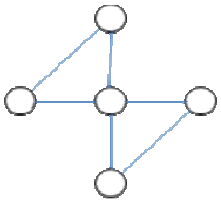


FIGURE 1e: FriendshipGraph(T2)

Labeling is the method of distribution integers to graph elements is often called as a mapping function from integers to Elements of Graph. Magic Labeling is outlined as a bijection from {1, 2, 3, 4, 5… n-1,n} to n - graph elements such the total of every element could be a magic constant K . If the element is vertex or edge or both it's referred to as Vertex or Edge or Total magic Total Labeling.

Let a G be a Graph with vertices v and edges e if there's a 1 to 1 function from set of integers {1,2,3…..,n-2,n-1,n} to edges of Graph & vertices are appointed label as total of edges incident to that. If the edge weights are completely different then it'll be EAVL (Edge Antimagic Vertex

Labeling) .For any 2 integers a, d if the sides are assign with labels is { a , a + d ,…..…. , a+(e-1)d }, where a is greater than 0 i.e. a>0 and d greater than equal to 0 d>=0 is termed (a, d) EATL. If the vertices are appointed with then successive numbers then, it'll be SEATL (Super Edge Antimagic Labeling).

Likewise for a G graph with vertices V and edges E if there's a 1 to 1 operate from set of integers {1,2,3,4,……, v + e} to vertex set V & edge set E of graph and edge weight is total of labels appointed to edge & labels appointed to its (V) vertices . If edge weights completely different then it'll be Edge Antimagic Total Labeling (EATL). For any 2 integers a, d if the vertices appointed with labels then , wherever a is greater than 0 i.e. a>0 and d greater than equal to 0 d>=0 is termed as (a, d) EATL. If the vertices are indicated with successive numbers then, it'll be Super Edge Antimagic Total Labeling (SEATL). Following section provides procedure to spot above all for various topologies of Graphs. Accumulative variety of all such potentialities is calculated. And open issues are solved & observed properties of graphs

3. PROPOSED WORK

During this section we have a tendency to discuss algorithmic rules to spot varied features of edge Antimagic labeling. We have a tendency to offer generalized algorithmic rule in common to any or all forms of Graphs mentioned during this paper. As a result of topological variations every graph structure needs some modifications to algorithmic rule.1st we have a tendency to discuss algorithmic rule then needed changes for every kind of graph are given during this section. the subsequent functions are utilized in designing algorithmic rule.

npx : Generates various sizes x from set of n to set of numbers that are obtainable and unused labels.

is_Duplicate (array, size) : The function returns Boolean value if array have duplicate values other wise false

is_RegDiff (array, size) : The functions returns change of rate if the array have adjacent values along with a common difference given by AP else it returns negative one (-).

is_SEATL (array) : The function returns Boolean value if is_RegDiff (array, size) function returns true then it also returns true otherwise false

Input : Graph G with vertices V & Edges E

Output : Possible variety of total Edge Anti Magic Vertex Labeling, Edge Anti Magic Vertex Labeling, (a, d) Edge Anti Magic Vertex Labeling with a, d values, Total Edge Anti Magic Total Labeling, Edge Anti Magic Total Labeling, (a, d) Edge Anti Magic Total Labeling with a, d values and checks existence of super Edge Antimagic Labeling

Algorithmic procedure for EAVL

Read Graph with size n & set labels range {1,2,3,4….,r}.
for (i=1; i<=r ;i++)
    If there exist rP_n & it's not an isomorphic then set them as labels of hub and spokes.
    for (j=1;j<=r;j++)
    if there exist rP_n, then continue process
    Otherwise display "all possible assignments are checked".
    Set them as labels of edges.
    For all edges calculate weight.
    Wei_ght[e] = addition of labels of its vertices.
    If(is_Duplicate(wei_ght, n)) EAVL_cnt++;
    Set d= is_RegDiff(wei_ght, n)
    If(d>=1) adEAVL_cnt++;
    If(d==1) SEAVL_cnt++;
Stop.

Algorithm for EATL

Read Graph with size n and set labels range {1,2,3,4,5….r}.
For(i=1;i<=r;i++)
If there exist rP_n & it's not an isomorphic then set them as labels of hub & spokes.
for (j=1;j<=r;j++)
if there exist ar rP_3 then continue process.
Otherwise display "all possible assignments are checked".
Link1: Set them as labels of last edge, 1st vertex and 1st edge.
Previous vertex=1 current vertex=2
for (i=1;i<=r;i++)
if there exist a rp_2 then continue process
else go to Link1.
Link2: Set them as labels of current vertex and current edge.
Previous vertex=current vertex Current vertex= curr_vertex+1
If current vertex=n then go to Link2.
else
reset labels assigned to previous vertex as available.
Previous vertex=prev_vertex-1 Current vertex= curr_vertex-1
For all edges calculate weight.
Wei_ght[e] = addition of labels assigned to it and labels assigned to its vertices.
If (is_Duplicate(wei_ght, n)) EATL_cnt++;
Set d= is_RegDiff(wei_ght, n)
If (d>=1) adEATL_cnt++;
If (d==1) SEATL_cnt++;
Stop.

Modifications to be done For Path:
For EAVL Range R is {1,2,.....n}.
For EATL Range R is {1,2,.....2n-1}.
We can avoid Step 2 for Paths and set label of first edge as zero.
Modifications to be done For Cycle:
For EAVL Range R is {1,2,.....n}.
For EAT Range R is {1,2,.....2n}.
We can avoid Step 2 for Cycle.
Modifications to be done For Wheel:
For EAVL Range R is {1,2,.....n+1}.
For EATL Range R is {1,2,.....3n+1}.
Modifications to be done For Fan Graph:
For EAVL Range R is {1,2,.....n}.
For EATL Range R is {1,2,.....3n}.
Set label of first edge as zero.
Modifications to be done For Friendship Graph:
For EAVL Range R is {1,2,.....2*n+1}.
For EATL Range R is {1,2,.....5n+1}.
Set labels of even edges as zero.

4. RESULTS

Here we tend to designed algorithms to supply additive variety of edge anti magic vertex or total labelings. we tend to conjointly known doable variety of (a, d) EAVL or Edge anti magic total labelings for various values of a and d. Also we tend to calculated so many of super Antimagic labelings if exists. Many authors analyzed behaviour of a specific graph structure for a few a and d values. However here these algorithms turn out all such sequence of arrangement of vertices and edges labels. So, we will simply and visually perceive behaviour of any structure. Observations made are as follows.

Paths:
All paths of size n>=3 is Anti magic. For Associate in Nursing instance if the path size is eight there are 6024 (EAVL) Edge anti magic vertex labelings are possible and it's 64 (a, d) Edge anti magic vertex labelings and fifty six among them are super edge magic. It has (3, 2) and (6, 1) Edge anti magic vertex labelings for several sequences. Paths with any length consists several Edge anti magic total labelings. For instance the path with length five has 233520 Edge anti magic total labelings. it's 4680 (a, d) Edge anti magic total labelings among 1840 are super. This sort of study is often done on any path. for each path there's no (a, d)-EAT labeling with d > 6.

Cycles:
In cycles, all cycles with size >=3 is antimagic. These algorithmic rules are estimated for many values of n. For n=5 it resulted thirty Edge anti magic vertex labelings. It also has 10 (a, d) Edge anti magic vertex labelings for a few values of a and d and each one of those are super. For a cycle of size five we tend to observed 1858600 Edge anti magic total labelings. It also has 10460 (a ,d) Edge anti magic total labelings among 4980 are super. feasible values of (a, d) are (6,5), (7,5), (8,4)etc.
In the similar way we will analyze any cycle with any given size.

Wheels:
Wheel with size≥ three, has no edge magic vertex labeling. But it has edge magic total labelings. For given wheel size we are able to show the attainable Edge anti magic total labelings. If the wheel size is three, then there exist 1128768 Edge anti magic total labelings. It's 4512 (a, d) Edge anti magic total labelings for values among 3840 are super Edge anti magic total labelings. we've got (15,1), (16,1), (17,1), (18,1), (10,3) and (11,3) etc attainable (a, d) Edge anti magic total labelings

Fan Graphs:
Fan Graphs are Antimagic with size three to six. Although the number of such sequences are very less all those are (a, d) Edge anti magic vertex labelings. Fan graphs with size three, four and five are giving 8 (a, d) Edge Anti Magic Vertex Labelings however all are super Edge anti magic vertex labelings & all are of (3,1) Edge anti magic vertex labelings. For Fan Graphs we tend to indicate some shocking results. If the fan size is three, then there exist 152152 Edge anti magic total labelings. It's 1344(a, d) Edge anti magic total labeling for various values among 672 are super Edge anti magic total labelings. we've got (14,2),(9,3) ,(16,1),(8,3) (11,3), (15,1), (12,2) and (11,3) etc attainable (a, d) Edge anti magic total labelings.

Friendship Graphs:
Friendship graph also has edge anti magic labeling. Friendship graphs also are observed as every (a, d) edge anti magic vertex labeling is super. Friendship graph with size 2 is having 24 edge anti magic vertex labelings and none of them is (a, d) edge anti magic vertex labeling. However Friendship graphs with size 1,3,4,5 are producing edge anti magic vertex labelings,(a, d) edge anti magic vertex labelings. For these graphs all are super edge magic. Friendship graph with size three producing 192 edge anti magic vertex labelings and 48 (a, d) edge anti magic vertex labelings and every one area unit super. Friendship graph with size four manufacturing 3072 edge anti magic vertex labelings and 2304 (a, d) edge anti magic vertex labelings and all of them are super edge anti magic vertex labelings. Each Friendship graph could be a super edge anti magic total labeling. For Friendship graphs with size 3 has huge number of edge anti magic total labeling (a, d) edge anti magic total labeling for d=2. These are some observations created by us.

5. CONCLUSIONS
In this paper, we tend to provide algorithms to enumerate all Edge Antimagic labelings on wheel graphs, Fan Graphs, cycle Graphs and Friendship graphs. The thought of the algorithms can be applied to alternative categories of graphs or adopted to develop algorithms for other form of labeling. within the in the meantime, we still engaged on algorithm for other form of labeling like edge anti magic and total, harmonious, swish etc. we tend to present the number of non

isomorphic completely different anti magic labelings on every graph for a some small size graphs. The number of non-isomorphic labeling on larger size of the remaining graphs is still an open problem.

6. REFERENCES

[1] Magic labelings on cycles and wheels, by Baker A., J. Sawada, 2008, COCOA LNCS, 361-373.

[2] "Algorithms to Construct Anti Magic Labeling of Complete Graphs ", by Krishnappa H K, N K Srinath, S Manjunath .IJCIT Volume 02– Issue 03, May 2013.

[3] J. Gallian, A dynamic survey of graph labeling, Electronic J. Combin., 14 (2007), DS6.

[4] Narsingh Deo, "Graph theory with applications to Engineering and Computer Science", 56 (2000).

[5] "applications of graph labeling in communication networks" by N. Lakshmi prasanna, k. Sravanthi and nagalla sudhakar's oriental journal of Computer science & technology issn: 0974-6471 april 2014,vol. 7, no. (1):pgs. 139-145.

[6] N. Hartsfield and G. Ringel, Pearls in Graph Theory, Academic Press, Boston-San Diego-New York- London, 1990.

[7] R. Bodendiek and G. Walther, Arithmetisch antimagische graphen, in K. Wagner and R. Bodendiek, Graphentheorie III, BI-Wiss. Verl. Mannheim, 1993.

[8] M. Ba˘ca, F. Bertault, J. MacDougall, M. Miller, R. Simanjuntak, and Slamin, Vertex Antimagic total labelings of graphs, Discuss. Math. Graph Theory, 23 (2003) 67–83.

[9] J.Sedlacek, Problem 27 in Theory of graphs and its applications, Proc. Symp. Smolenice, June 1963, Praha (1964), p.162.

[10] J.A.MacDougall, M.Miller, K.A.Sugeng, "Super Magic Total Labelings of graphs", Proceeding of the Fifteenth Australian workshop on Combinatorial Algorithms 2004, Balina, NSW, (2004), 222-229.

[11] Ko-Wei-Lih,s "The Magic labelings for Wheels, Prisms and Friendship graphs" ,1983.

[12] Krishnaa.A's "A study of the major graph labelings of the trees", Informatica, **15**(4), 515–524, 2004.

[13] D.B. West, An Introduction to Graph Theory (Prentice-Hall, 1996).

[14] Kiki A. Sugeng's "Survey of edge antimagic labelings of graphs", J. Indones. Math. Soc. (MIHMI).

[15] Nissankara Lakshmi Prasanna ,Nagalla Sudhakar, "Algorithms For Magic Labeling On Graphs",Journal Of Theoretical And Applied Information Technology,10th August 2014. Vol. 66 No.1.

[16] Nissankara Lakshmi Prasanna,Nagalla Sudhakar, K. Sravanthi," Algorithm For Vertex Anti-Magic Total Labeling On Various Classes Of Graphs",Contemporary Engineering Sciences, Vol. 7, 2014, No. 19, 915 – 922, http://Dx.Doi.Org/10.12988/Ces.2014.4665.

[17] A. A. Samah, M. Z. Matondang, H. Haron, H. A. Majid & N. K. Ibrahim." Role Model of Graph Coloring Application in Labelled 2D Line Drawing Object", International Journal of Experimental Algorithms (IJEA), Volume (2), Issue (3), 2011.

Skip List: Implementation, Optimization and Web Search

Godwin Adu-Boateng *gaduboateng@umc.edu*
Center of Biostatistics and Bioinformatics
Department of Medicine
University of Mississippi Medical Center
Jackson, MS 39216, USA

Matthew N. Anyanwu *matany58@yahoo.com*
Department of Computer Science
University of Memphis
Memphis, TN 38152, USA

Abstract

Even as computer processing speeds have become faster and the size of memory has also increased over the years, the need for elegant algorithms (programs that accomplish such tasks/operations as information retrieval, and manipulation as efficiently as possible) remain as important now as it did in the past. It is even more so as more complex problems come to the fore. Skip List is a probabilistic data structure with algorithms to efficiently accomplish such operations as search, insert and delete. In this paper, we present the results of implementing the Skip List data structure. The paper also addresses current Web search strategies and algorithms and how the application of Skip List implementation techniques and extensions can bring about optimal search query results.

Keywords: Skip List, Efficient Algorithms, Web Search.

1. BACKGROUND & INTRODUCTION

One of the fundamentals in the study of computer science is the representation of stored information. Even as computer hardware has become more sophisticated, and the size of memory has increased over the decades and sustains an upward trajectory, the resources required for computer programs (memory space and time) to execute implemented programs continue to be in the fore-front. Programs that accomplish such tasks/operations as information retrieval, and manipulation as efficiently as possible have garnered considerable attention from computer scientists [1]. In Galil and Italiano [2], the authors' survey data structures and algorithms proposed as a solution for a set union problem. Weiss [3] analyzes a disjoint set problem and provides a real world scenario of its application in a computer network. Both Galil and Italiano, and Weiss investigate the memory space and time complexity of proposed algorithmic solutions. Collins [4] also discusses it with the Java programming language as a back drop. Literature exists that approach this fundamental problem in other ways. Review [5, 6, 7] for further discussion on algorithm analysis and efficiency. To achieve optimal efficiency in data manipulation, the type of data structure used plays an important role. Shaffer [1] defines a data structure as "…any data representation and its associated operations." He goes further to state that "...a data structure is meant to be an organization or structuring for a collection of data items." A scheme of organizing related data items is how Levitin [7] defines a data structure. In essence, a data structure is any organization of related data items and its associated operations. The data is organized such that it can be used optimally. Hence, data structures are an integral part of algorithm execution, and it can determine how efficient or otherwise an algorithm is.

Today the World Wide Web or simply the Web or its contemporary name: Social web has become part of everyday life. Each day millions of people interact with the Web in several ways, for instance, sending and receiving e-mails, interacting with friends and colleagues, reviewing and

rating visited restaurants, searching for travel deals and merchandise. It is anticipated that this trend would continue [8]. Searching the internet via search engines has become quite poplar as it usually serves as an introductory point to internet usage. In fact reports show that *84%* of internet users have used search engines [9]. Many internet search engines exists today for example, Yahoo, Bing, Google and many other more targeted ones, like Amazon and EBay, among others. Each has its own structure in terms of query execution, information gathering/corpus construction, and display/presentation of the relevant pages/retrieved corpus. However, the baseline goal of retrieving relevant information remains consistent in all search engines.

As information on the Web is forecasts to increase and more especially as big data comes to the forefront, it is increasingly going to be essential for search engines to not only retrieve relevant documents and Web pages, but also, be optimal in ad hoc query execution. For decades now, Text REtrieval Conference (TREC) has remained the benchmark for Information Retrieval (IR). The main focus of this benchmark is to test for effectiveness (accuracy) in the resulting corpus of a search query, and not efficiency [10]. A review of the literature shows that to be the case [11]. Cao et al. [12] also examine the retrieval of handwritten document images.

As earlier stated the quest for optimality is an ongoing and important one. It is even more important when considered in the context of the ever increasing complexity of problems that require computer solutions. The emergence of big data: A contemporary problem/opportunity presents varying degrees of complexity [13]. It also presents challenges in the search and retrieval of data, and data analytics, among others. Reports suggest that Web content continues to increase as Web usage increases [13, 14] and more users across all age grades and nations become conversant with using it. Therefore, an efficient way for search and retrieval in the face of this continuous increase in Web data/information is in order.

Skip List (SL) is a data structure with properties that appropriately satisfies the computer science fundamental of stored information representation. An experiment conducted clearly shows the desirability of using SL as the preferred structure for data processing. It can also play an important role in Web search as it also provides a data organization scheme that yields better efficiency in operations such as search [15].

2. SKIP LIST OVERVIEW & OPERATIONS

Skip lists (SL) was invented by William Pugh in 1989 and proposed by him in 1990. SL was proposed as an alternative to the Binary Search Trees (BST) and other balanced trees. It is a probabilistic data structure based on parallel linked list where elements are kept by key. But, as opposed to linked list, a node can contain more than 1 pointer. The number of pointers in a node determines its level. Each pointer points to the next node in the list that has an equal or greater level. If there isn't any forward node having this property, the pointer points to NIL [16]. The level of a Skip list corresponds to the highest level between the nodes. When a node is inserted to the list, a node is given a random level [16]. Figure 1 provides a pictorial depiction of a SL.

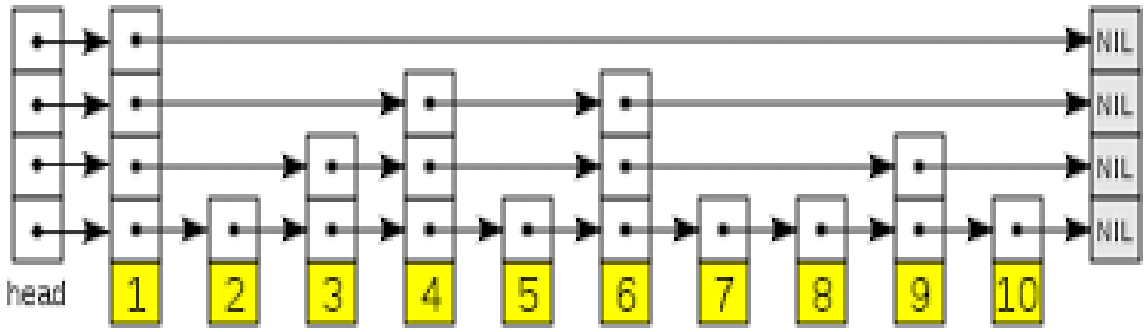


FIGURE 1: Skip List. Perfect Skip List.

The main motivation for using SL include among others is that it ensures a more natural representation than a tree, it is simple to implement, it has both practical and theoretical applications and its efficiency is comparable to a balanced tree where the expected time for an operation (such as searching for an element) is $O(\log n)$[1], where n is the node/random level [1, 16].

SL can be used as a replacement for other data structures. It also has other operational efficiencies as noted by Pugh [16];

- It is relatively easy to design and implement the algorithm
- It is efficient
- The randomization is not dependent on the input keys, thus it will be difficult for an advisory to access using the input keys
- It serves as an alternative to Binary search trees which become unbalanced after several insertion and deletion operations
- It requires a random number generator at the time of insertion
- The burden on memory management is due to varying size nodes

2.1 Skip List Operations

Like any data structure, SL supports such operations as *search*, *insert* and *delete*.
The search for an element starts at the header on the highest level, then traverses forward pointers that don't overshoot the element being searched for. If that happens, move down to the next level. A brief pseudo code description of the search algorithm is shown in Figure 2.

This algorithm is very simple to implement, it searches for the right place for an element to be inserted. Splice the list and insert the element with a random level.

Deleting/removing an element is also like inserting an element, it searches for the right place for an element to be deleted. Splice the list and delete the element with a given random level [17].

```
1. x <- list.header
2. for i <- list.level downto 1
3.     do    while x.forward[i].key < searchKey
4.               do    x <- x.forward[i]
5. x <- x.forward[1]
6. if x.key = searchKey
7.     then return x.value
8.     else return failure
```

FIGURE 2: Skip List Search Algorithm.

2.2 Practical Application of Skip List

A cursory search for practical application of the SL algorithm yielded varying levels of its application in the real world. Some of the practical applications are as follows:

- Parallel Computing: Skip List can be applied to parallel computation where insertions and searching of items can be accomplished in different parts of the list and in parallel without rebalancing of the data structure [18].

[1] This is a notation for identifying the order of growth of an algorithm's basic operation. In this particular case $O(\log n)$ represents the order of growth of SL search algorithm.

- Operating Systems: A common problem in computer science is the blocking of resources for processors to complete tasks. This can lead to race conditions. Skip list can be applied to this problem with a lock-free data structure which is more efficient and it is easy to implement [19].
- Forecasting/Prediction: Ge and Stan [20] used it for predicting queries by employing Pre-built Models (PM). The 'closest' PM is selected for use in the forecasting. Predicting can be applied to scheduling, resource acquisition, and resource requirement determination.
- Graphics and Visualization: Skip list-like data structure has been used to optimize the rendering of graphics and images [21].

It is important to state here that the list provided is by no means an exhaustive one as that would be beyond the scope of this paper.

3. EXPERIMENT

An experiment was conducted to determine the cost of searching for an element at a given random level.

The experiments were conducted on the algorithm using the java program on different values for p = 1/2, 1/4 and 1/8.

For each value of p, 1000 lists were randomly created for different sizes of n: 10, 100, 1000 and 10000.

For each case, the average number of steps to find 1000 random elements in the list was found. The average is then compared to $\log(1/p(n))$ and to $L(n)/p + 1/(1-p)$. Then the maximum number of steps recorded between 1000 searches was added. The result is shown in the following tables with the different p values and different n sizes.

3.1 Results

The results shown in the proceeding section illustrates well enough the analysis of the expected search cost.

In general, for the 3 different values of p, the average number of steps grows logarithmically to the number of elements n. In fact, for p = ½, the average is very close to log (1/2(n)). On average, the algorithm performs better for p = ½. But, as demonstrated in [17],
"…choosing p = ¼ slightly improves the constant of factors of the speed of the algorithm." This means that the algorithm is more consistent, and there is a higher probability to obtain a search cost close to the average when choosing ¼.

Thus, the maximum number of steps is relatively low for high instances of n. For p = ½ (see Table 1), it is only 38 steps for a list of size 10000. (38 is only the maximum number of steps in the experiment that we did, and it is possible to obtain a higher maximum). This implies that it is very unlikely to obtain a list largely unbalanced that will give us an extremely poor performance.

n	Average Number of Steps	Log (1/p(n))	L(n)/p + 1/(1-p)	Maximum Number of Steps
10	3.45	3.32	8.64	10
100	6.56	6.64	15.2	26
1000	9.19	9.96	21.9	35
10000	9.92	13.29	28.5	38

TABLE 1: Average number for steps for finding 1000 random list elements. P=1/2.

n	Average Number of Steps	Log (1/*p*(*n*))	L(*n*)/*p* + 1/(1-*p*)	Maximum Number of Steps
10	4.01	1.667	7.9	10
100	8.56	3.32	14.6	45
1000	12.57	4.98	21.9	59
10000	13.63	6.64	27.9	60

TABLE 2: Average number for steps for finding 1000 random list elements. *P*=1/4.

n	Average Number of Steps	Log (1/*p*(*n*))	L(*n*)/*p* + 1/(1-*p*)	Maximum Number of Steps
10	4.67	1.12	10	10
100	11.43	2.21	18.8	67
1000	17.86	3.32	27.7	112
10000	19.31	14.43	36.6	116

TABLE 3: Average number for steps for finding 1000 random list elements. *P*=1/8.

4. WEB SEARCH: KNOWN & UNKNOWN

Not all search engines are created equal and today an online search involves Web ‘crawlers’, ‘spiders’ or ‘robots’. These are algorithms that innocuously navigate web pages and links to gather information. They undertake the processes that make up the dimension of a web IR, and they can provide different results set [11].

Although algorithms for web searches are mostly proprietary, research shows they rely on inverted indexing in the processing of search requests and corpus construction. Many search algorithms and strategies use this approach as the basis of conducting a document search [22, 23]. Inverted indexing creates a list of index terms along with a logical linked list referred to by Grossman [10] as a *posting list*. Each linked list pointer references a unique term in the index, which is created before query execution.

Again, current web search results are represented in some form or fashion. The documents relevance to the query, the number of times the search term is found in the document, or the interval between search terms in the document are some of the representation criteria discussed by Li et al. [22].

As previously stated the benchmark for IR is on the effectiveness and not efficiency. Hence, there is little to no mention of the optimality of search engine algorithms. It is our belief that an efficient data structure for the inverted index can improve performance.

5. CONCLUSION & FUTURE WORK

SL has an expected time of $O(\log n)$ for an operation such as search or insertion. Based on the experiments that we did in this project, we can conclude that SL is a good alternative to balanced tree. Taking the same argument from the author, SL is very easy to implement which is the biggest advantage over balanced trees.

Also, SL can be applied to inverted indexing; which features prominently in web searches. Research has shown that SL, when applied to web search offers improved efficiency. Boldi and Vigna [24] discuss embedding SL in inverted index. Inverted index search improves the efficiency of SL by reducing the space requirement of web searches. The reduction in the space

requirement further improves the expected time of search of SL. Also inverted index ensures that all occurrences of a query is retrieved. Chierichetti et al. [25] report on using different ways of determining the precise locations of skips to be placed in an inverted index with an embedded SL. They report the time complexity on an optimal algorithm as being linearithmic. Also, Campinas et al. [26] extend the basic Skip list for a block-based inverted list and found a lower search cost.

There appears to be promise in the continued application of skip list techniques and extensions as it pertains to search engine optimization. This is a step in the right direction when considered in the context of current trends in web searches and document retrieval, where the amount of documents on the web is estimated to be in the billions. Google, one the most used search engines has over 2 billion indexed documents. The current estimate of online documents is over 500 billion today. Li et al. [22] report on the feasibility of a peer-to-peer web search. Future work will focus on determining SL performance for TREC as well for efficiency.

6. ACKNOWLEDGEMENT

We are grateful to Serge Salan from University of Memphis TN for his contribution in performing the Skip-list experiment.

7. REFERENCES

[1] C.A. Shaffer. *A Practical Introduction to Data Structures and Algorithm Analysis – Java Edition.* Prentice-Hall, Inc., pp. 365-371, 1998.

[2] Z.Galil and G.F. Italiano. Data Structures and Algorithms for Disjoint Set Union Problems. ACM Computing Surveys vol. 23, pp. 319-344.

[3] M.A. Weiss, *Data Structures and Algorithm Analysis in C++,* Benjamin/Cummings, Redwood City, Calif., Chap 8, pp. 287, 1994.

[4] W.J. Collins. *Data Structures and the Java Collections Framework*. McGraw Hill. pp. 389 – 432, 2002.

[5] A.V. Aho, J.E. Hopcroft, J.Ullman. *Data Structures and Algorithms*. Addison-Wesley Longman Publishing Co., 1983.

[6] T. Kanungo, D.M. Mount, N.S. Netanyahu, C. Piatko, R. Silverman, A.Y. Wu. “An Efficient *k*-Means Clustering Algorithm: Analysis and Implementation.” IEEE Trans. Patt. Anal. Mach. Intell, v. 24, no.7, pp. 881-892, 2002.

[7] A. Levintin. *Introduction to the Design & Analysis of Algorithms*. Addison-Wesley, 2006.

[8] J. Porter. *Designing for the social web*. Peachpit Press, 2010.

[9] H.R. Varian. *The economics of Internet search*. University of California at Berkeley, 2006.

[10] D.A. Grossman. *Information retrieval: Algorithms and heuristics*. Vol. 15. Springer, 2004.

[11] M. Gordon and P. Pathak. “Finding information on the World Wide Web: The retrieval effectiveness of search engines.” Information Processing and Management*,* 35(2), pp. 141–180, 1999.

[12] H. Cao, A.Bhardwaj and V. Govindaraju. “A probabilistic method for keyword retrieval in handwritten document images.” Pattern Recognition, 42(12), pp. 3374-3382, 2009.

[13] S. Lohr. (2012). “The Age of Big Data.” The New York Times. Retrieved Jan 2013, available online: http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html?_r=2&scp=1&sq=Big%20Data&st=cse&.

[14] A. McAfee and E. Brynjolfsson. “Big data: the management revolution.” Harvard Business Review, 90(10), pp. 60-66, 2012.

[15] R. Delbru, S. Campinas and G. Tummarello. Searching web data: An entity retrieval and high-performance indexing model. *Web Semantics: Science, Services and Agents on the World Wide Web*, *10*, pp. 33-58.

[16] W. Pugh. “Skip Lists: A Probabilistic alternative to balanced trees.” Communications of the ACM, v. 33. pp. 668-676, 1990.

[17] W. Pugh. “A skip list cookbook.” University of Maryland, Department of Computer Science Report CS-TR-2286.1, 1989.

[18] J. Gabarro, C. Martinez and X. Messeguer. “A design of a parallel dictionary using skip lists.” Theoretical Computer Science 158, pp. 1-33, 1996.

[19] F. Keir. *Practical lock-freedom*. Diss. University of Cambridge, 2004.

[20] T. Ge and S. Zdonik. “A Skip-list approach for efficiently processing forecasting queries.” Proceedings of the VLDB Endowment 1.1, pp. 984-995, 2008.

[21] J. El-Sana, E. Azanli, and A. Varshney. Skip Strips: Maintaining Triangle Strips for View-Dependent Rendering. In IEEE Visualization ’99 Proceedings, pp. 131–138, 1999.

[22] J. Li, B.T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Karger, and R. Morris. “On the feasibility of peer-to-peer web indexing and search.” In Peer-to-Peer Systems II. Springer Berlin Heidelberg. pp. 207-215, 2003.

[23] L.A. Barroso, J. Dean, and U. Holzle. "Web search for a planet: The Google cluster architecture." Micro, IEEE 23.2, pp. 22-28, 2003.

[24] P. Boldi and V. Sebastiano. "Compressed perfect embedded skip lists for quick inverted-index lookups." String Processing and Information Retrieval. Springer Berlin Heidelberg, 2005.

[25] F. Chierichetti, R. Kumar and P. Raghavan. "Compressed web indexes." Proceedings of the 18th international conference on World Wide Web. ACM, 2009.

[26] S. Campinas, R. Delbru, and G. Tummarello. "SkipBlock: self-indexing for block-based inverted list." Advances in Information Retrieval. Springer Berlin Heidelberg, pp. 555-561, 2011.

INSTRUCTIONS TO CONTRIBUTORS

Experimental Algorithmics studies algorithms and data structures by joining experimental studies with the more traditional theoretical analyses. With this regard, the aim of The International Journal of Experimental Algorithms (IJEA) is (1) to stimulate research in algorithms based upon implementation and experimentation; in particular, to encourage testing, evaluation and reuse of complex theoretical algorithms and data structures; and (2) to distribute programs and testbeds throughout the research community and to provide a repository of useful programs and packages to both researchers and practitioners. IJEA is a high-quality, refereed, archival journal devoted to the study of algorithms and data structures through a combination of experimentation and classical analysis and design techniques. IJEA contributions are also in the area of test generation and result assessment as applied to algorithms.

To build its International reputation, we are disseminating the publication information through Google Books, Google Scholar, Directory of Open Access Journals (DOAJ), Open J Gate, ScientificCommons, Docstoc and many more. Our International Editors are working on establishing ISI listing and a good impact factor for IJEA.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Started with Volume 5, 2015, IJEA appears with more focused issues. Besides normal publications, IJEA intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

We are open to contributions, proposals for any topic as well as for editors and reviewers. We understand that it is through the effort of volunteers that CSC Journals continues to grow and flourish.

IJEA LIST OF TOPICS

The realm of International Journal of Experimental Algorithms (IJEA) extends, but not limited, to the following:

- Algorithm Engineering
- Algorithmic Code
- Algorithmic Engineering
- Algorithmic Network Analysis
- Analysis of Algorithms
- Approximation Techniques
- Cache Oblivious algorithm
- Combinatorial Optimization
- Combinatorial Structures and Graphs
- Computational Biology
- Computational Geometry
- Computational Learning Theory
- Computational Optimization
- Data Structures
- Distributed and Parallel Algorithms
- Dynamic Graph Algorithms
- Heuristics
- Mathematical Programming For Algorithms
- Metaheuristic Methodologies
- Network Design
- Parallel Processing
- Randomized Techniques in Algorithms
- Routing and Scheduling
- Searching and Sorting
- Topological Accuracy
- Visualization Code
- VLSI Design
- Graphics

- Experimental Techniques and Statistics
- Graph Manipulation

CALL FOR PAPERS

Volume: 5 - **Issue: 2**

i. Paper Submission: October 31, 2015 **ii. Author Notification:** November 30, 2015

iii. Issue Publication: December 2015

CONTACT INFORMATION

Computer Science Journals Sdn BhD
B-5-8 Plaza Mont Kiara, Mont Kiara
50480, Kuala Lumpur, MALAYSIA

Phone: 006 03 6204 5627

Fax: 006 03 6204 5628

Email: cscpress@cscjournals.org

COMPUTER SCIENCE JOURNALS (CSC JOURNALS)
B-5-8 PLAZA MONT KIARA, MONT KIARA
50480, KUALA LUMPUR, MALAYSIA
PHONE: 00603 6204 5627
FAX: 00603 6204 5628
URL: http://www.cscjournals.org