BIJIT - BVICAM's International Journal of Information Technology
Bharati Vidyapeeth's Institute of Computer Applications and Management (BVICAM), New Delhi (INDIA)

A Methodology to Find the Cycle in a Directed GraphUsing Linked List

Shubham Rungta[1], Samiksha Srivastava[2], Uday Shankar Yadav[3]and **Rohit Rastogi[4]**

Submitted in July 2013, Accepted in September 2014

Abstract - In computer science, cycle detection is the algorithmic problem of finding a cycle in a sequence of iterated functionvalues. The analysis of cycles in network has different application in the design and development in communication systems such as the investigation of topological features and consideration of reliability and fault tolerance. There are various problems related to the analysis of cycles in network among which the most important one is the detection of cycles in graph. In this paper, we proposed SUS_dcycle method which is a detection algorithm for detecting cycle in a directed graph, with the help of linked list in order to discover new lists in run time. This algorithm is used to detect the cycle in any type of directed graph. The proposed algorithm differs from other existing algorithms through its ability to count the total number of cycles present in any type of directed graphs. Also the study of earlier works says that this is a novel approach for the prescribed task and complex problems may use it as a subroutine application for effective results. In advanced computing, time-space trade-off is an important factor to efficiently deal with the problems. This method may solve the above said purpose.

Index Terms – Directed graph, Cycle, Linked list, Graph theory, and Data structure.

1.0 INTRODUCTION

1.1 Cycle

A cycle is repeating part in the sequence. In computer science, cycle detection is the algorithmic problem of finding a cycle in a sequence of iterated function values [1]. Suppose in a function f(x), if x repeats the same sequence of values once again, then there exist a cycle.

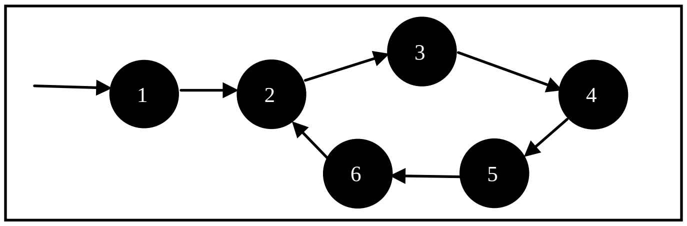

Figure 1: Example of a cycle with 1, 2, 3, 4, 5, and 6 as vertices of the graph.

[1, 2, 3] *B. Tech. CSE-IV Year, ABES Engineering College, Ghaziabad (U.P.), India.*
[4] *Sr. Asst. Professor, CSE Deptt. ABES Engineering College, Ghaziabad (U.P.), India.*
E-mail: [1]shubhamrungta93@gmail.com, [2]samikshasrivastava607@gmail.com, uday4792@gmail.com and [4]rohit.rastogi@abes.ac.in

Here f(x) is the function and [x: x is 1, 2, 3, 4, 5, 6, y… in sequence, where y is 2, 3, 4, 5, and 6 in sequence and is repeated], here cycle exists because x repeats the value 2, 3, 4, 5, 6 repeatedly. This paper proposes a SUS's cycle detection algorithm to detect cycles and to find number of cycles in any directed graph whether it is simple or multi digraph. This algorithm is also helpful in fetching out the number of cycles in the whole graph.

2.0 EARLIER WORKS IN THIS FIELD

2.1 Floyd's cycle-finding algorithm, also called the "tortoise and the hare" algorithm, is a pointer algorithm that uses only two pointers, which move through the sequence at different speeds. The algorithm is named for Robert W. Floyd, who invented it in the late 1960s.[9]

The key insight in the algorithm is that, for any integers $i \geq \mu$ and $k \geq 0$, $x_i = x_{i+k\lambda}$, where λ is the length of the loop to be found. In particular, whenever $i = m\lambda \geq \mu$, it follows that $x_i = x_{2i}$. Thus, the algorithm only needs to check for repeated values of this special form, one twice as far from the start of the sequence as the other, to find a period ν of a repetition that is a multiple of λ. Once ν is found, the algorithm retraces the sequence from its start to find the first repeated value x_μ in the sequence, using the fact that λ divides ν and therefore that $x_\mu = x_{\mu+2\nu}$. Finally, once the value of μ is known it is trivial to find the length λ of the shortest repeating cycle, by searching for the first position $\mu + \lambda$ for which $x_{\mu+\lambda} = x_\mu$.

The algorithm thus maintains two pointers into the given sequence, one (the tortoise) at x_i, and the other (the hare) at x_{2i}. At each step of the algorithm, it increases i by one, moving the tortoise one step forward and the hare two steps forward in the sequence, and then compares the sequence values at these two pointers. The smallest value of $i > 0$ for which the tortoise and hare point to equal values is the desired value ν.

2.2Richard P. Brent et al. described an alternative cycle detection algorithm that, like the tortoise and hare algorithm, requires only two pointers into the sequence.[10] However, it is based on a different principle: searching for the smallest power 2^i that is larger than both λ and μ. For i = 0, 1, 2, etc., the algorithm compares x_{2^i-1} with each subsequent sequence value up to the next power of two, stopping when it finds a match. It has two advantages compared to the tortoise and hare algorithm: it finds the correct length λ of the cycle directly, rather than needing to search for it in a subsequent stage, and its steps involve only one evaluation of *f* rather the indices of saved sequence than three.

Brent [10] already describes variations of his technique in which values are powers of a number R other than two. By choosing R to be a number close to one, and storing the sequence values at indices that are near a sequence of consecutive powers of R, a cycle detection algorithm can use a number of function evaluations that is within an arbitrarily small factor of the optimum $\lambda+\mu$.[12] [13]

2.3Sedgewick, Szymanski, and Yao [14] provide a method that uses M memory cells and requires in the worst case only $(\lambda+\mu)(1+cM^{-1/2})$function evaluations, for some constant c, which they show to be optimal. The technique involves maintaining a numerical parameter d, storing in a table only those positions in the sequence that are multiples of d, and clearing the table and doubling d whenever too many values have been stored.
Several authors have described distinguished point methods that store function values in a table based on a criterion involving the values, rather than (as in the method of Sedgewick et al.) based on their positions. For instance, values equal to zero modulo some value d might be stored.[15][16] More simply, **Nivasch**[11] credits D. P. Woodruff with the suggestion of storing a random sample of previously seen values, making an appropriate random choice at each step so that the sample remains random.

2.4 Nivasch [11] describes an algorithm that does not use a fixed amount of memory, but for which the expected amount of memory used (under the assumption that the input function is random) is logarithmic in the sequence length. An item is stored in the memory table, with this technique, when no later item has a smaller value. As Nivasch shows, the items with this technique can be maintained using a stack data structure, and each successive sequence value need be compared only to the top of the stack. The algorithm terminates when the repeated sequence element with smallest value is found. Running the same algorithm with multiple stacks, using random permutations of the values to reorder the values within each stack, allows a time–space tradeoff similar to the previous algorithms. However, even the version of this algorithm with a single stack is not a pointer algorithm, due to the comparisons needed to determine which of two values is smaller.
Any cycle detection algorithm that stores at most M values from the input sequence must perform at least $(\lambda+\mu)(1+\frac{1}{M-1})$function evaluations.[17] [18]

3.0 PROPOSED IDEA

3.1 Proposed SUS_dcycle Algorithm

a) START

b) SUS_dcycle algorithm has parameters as information field, variables with their data types and identifiers for starting, processing the paths, temporary allocation and exchanging of the data, variables for counting the cycles in test graph, holding temporary data and functional mechanism to free the unused space.

c) Initialize the variables as per the mechanism chosen and allocate the data in them.

d) Define the vertices information and edges as per the mechanism/ approach chosen.

e) Put each and every node in a list say 'LIST0'.

f) Take out any node from LIST0 as a starting node (if any node exists.)

g) Get/start with the first vertex, let A and hold its storage location.

h) Now, if there are n directed paths from the first vertex A, then the possible new paths from here are n-1, so create the n-1 data structures (chosen by you) to hold these possible paths. Also remove these discovered node from List0.And whenever a new node discovered during traversal remove it from LIST0.

i) Define a mechanism to store the information of all next vertex traversed from the previous vertex and hold it anyhow.

j) Loop starts up to all the existing and uncovered paths.

k) In a particular path p, compare the new vertex presently being traversed with the all of previously traversed vertices starting from the first vertex in p and check whether the present information is being repeated.

l) If yes, then
  We are sure that a cycle exists.
  Store the vertices information comprising this cycle and may print their values as per need.
  Increment the counter by one.

m) If no vertex is repeated, then
  We can declare that there is not a cycle in that traversed path p.

n) So, go to next possible path starting from first vertex.
  Continue this process till all the paths are covered.
  Loop Ends

o) After working with every path from Starting node ('A'). Check LIST0 if any node is present there, if yes then once again take out any node from remaining node and follow step 8 to 14.

p) Print the counter value as the no. of cycles in the test graph and as per need can print the vertices contained in those cycles respectively.

q) END

3.2 Proposed SUS_dcycle Algorithmwith the Linked List Implementation

SUS_dcycle (INFO, LINK, START1, START2, LIST0, LIST1, LIST2, PTR1, PTR2, PTR3, ITEM, Counter, FREE(x), TEMP)

INFO-Stores the information field of the node in linked list.

LINK-Address field of the node that contains the address of the next node in the linked list.

LIST0- List to store all the nodes of a graph. Use of this list is to find out the unreachable nodes (if exist) from a starting node (that we choose randomly from LIST0) in a digraph.

LIST1-Linked list to store the base address of all the linked lists formed during runtime.

LIST2-Linked list to store nodes discover during traversal.

START1-Pointer which points to the first node of linked LIST1.
START2-Pointer which points to the first node of linked LIST2.
Counter-A global variable to count the number of cycle.
SAVE, **PTR**, **PTR2**, **PTR3**,**PTR4**, **and TEMP**: They are the pointer variables.
ITEM-It contains the info character.
FREE(x)-This function will remove the node x.
Start with:

1. Put each and every node in a LIST0.
2. Take out any node from LIST0 as a starting node (if any node exists.).
3. Whenever a new node discovered during traversal just remove it from LIST0.

Step1: Insert the first node in linked list LIST2. (Let the first node be A.)

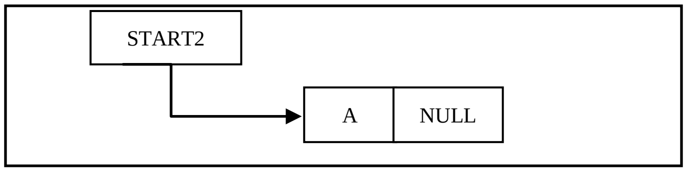

Fig: 2 Insertion of first

Step2: Put the base address of the LIST2 in LIST1.

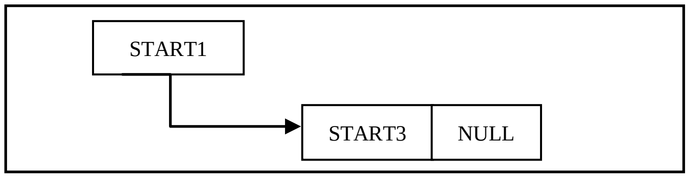

Figure 3: Insertion

Step3: Now, if there are n directed paths from A, then total number of new linked list will be n-1 and they all will be duplicate of LIST2.

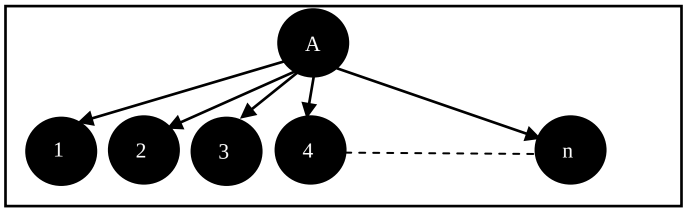

Figure 4: Example of n directed path from A

Step-4: Now put each next node of the graph directed from the last node into separate linked lists.

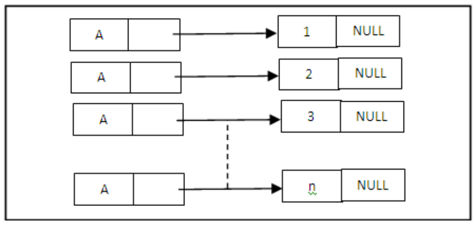

Figure 5: Insert 1 in LIST2 and others in the copies in LIST2

Step-5: Put the base address of each newly created list into the LIST1 in a consecutive manner.

```
// Iteration of outer loop
Step-6: Repeat Step (7) to (10) while (START1! = NULL)
/*Compare the element at last node of LIST2 with all its previous nodes starting from starting node.*/

Step 7: [Initialize the value of pointer PTR and SAVE with the value of pointer START2.]
        PTR<-START2
        SAVE<-START2
Step 8: [Repeat Steps (a) to (b) until (PTR! =NULL)]
        a) PTR<-LINK [PTR]
        b) TEMP<-INFO [PTR]
Step 9: [Initialize PTR2 with START2]
        PTR2<-START2

/*Compare the last element with all the elements of LIST2.*/

Step10: [Repeat the following steps (a) to (b) and 11 to 13 until (PTR2! =NULL)]
        SAVE=PTR2
      IF (TEMP=INFO [SAVE])
      THEN
        i.   Counter<- Counter+1

/*If the counter is incremented then drop the wholeLIST2 list if cycle exist in addition, the node with the base address of the dropped list is removed from List1 and START1 points the next node to the deleted node in List1*/

ii. [Display the detected cycle by repeating the following step until PTR3! =NULL]
          a)  ITEM=INFO [START2]
          b)  DISPLAY [ITEM]
          c)  PTR3=LINK [START2]
[In addition freeing the displayed nodes]
          d)  FREE (START2)
          e)  START2=PTR3
        iii. [Remove the node containing the base address ofLIST2 list in which cycle occurs or null node appears]
a) PTR4=LINK [START1]
b) FREE (START1)
          c) START1=PTR4
      ELSE
```

```
        PTR2=LINK [PTR2]
ENDIF
```

Step 11: [Enter the next node directed by the last node in LIST2 using Step (3)-(12) until NULL node encountered] [If NULL is encountered then GOTO STEP 11-(ii)]
After working with every path from Starting node ('A'). Check LIST0 if any node is present there, if yes then once again take out any node from remaining node and follow **Step** 1 to 11.

If no node is left in LIST0 then GOTO Step12.

Step12: [The final value of counter will result in total number of cycles in the test graph].
DISPLAY [Counter]
Step 13: END

3.3 Example

Let's take a directed graph with n (V) =5

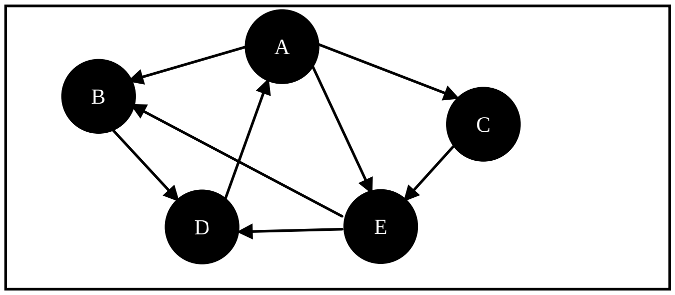

Figure 6: A directed graph with A, B, C, D and E as its vertices.

Let us take two-linked list LIST1, LIST2 withSTART1, START2 as their pointers to their base address respectively.
And LIST0 for storing each and every node of digraph (Figure 6)
Start with:

A	B	E	C	D

Figure 7: LIST0 storing all the nodes of digraph (Figure 6)

Take out any node from LIST0 as a starting node (if any node exists.).
Remember, whenever a new node discovered during traversal just remove it from LIST0.

Step-1: Insert the first node in linked list LIST2. Let it be node A. Remove 'A' from LIST0.

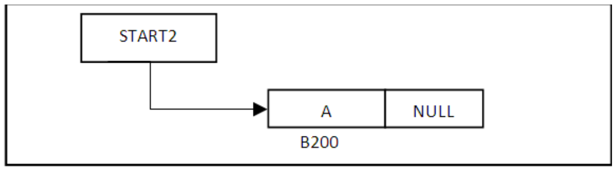

Figure 8: Linked list LIST2 with A in its first info field.

In addition, put the base address of this linked list LIST2 in LIST1 in its info field.

In addition, put the base address of this linked list LIST2 in LIST1 in its info field.

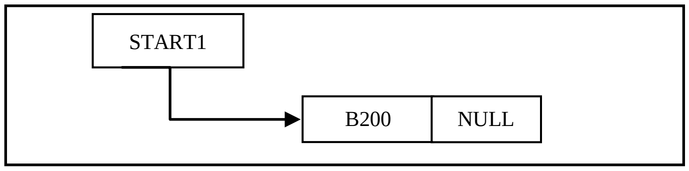

Figure 9: START1 points to the base address of LIST1 with base address of LIST2 in its info field.

Step-2: Now, there are four directed paths from A i.e. B,C,Eand Dso, total number of new linked lists will be three (4-1=3).And they will be duplicate of lastly implemented linked list. Now Put each next node of the graph directed from the last node (B,C,E&D) into separate linked lists and put the base address of each newly created linked lists into the LIST1 in a consecutive manner. Also, Remove B, C, E & D from LIST0.

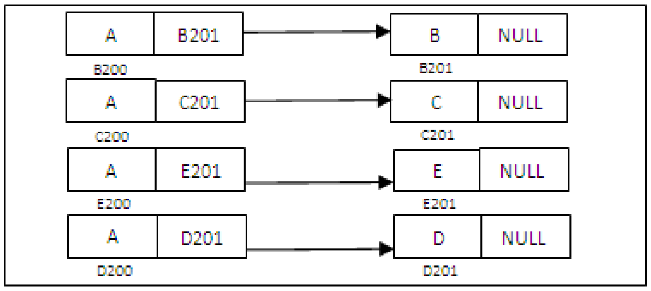

Figure 10: Insert B in LIST2 and C, E and D in the newly created copies of LIST2.

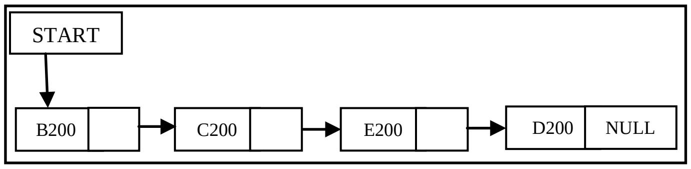

Figure 11: Insertion of base address of the above-created lists in LIST1

Step-3: Repeat step (4)-(9) until START1! = NULL
Step-4: Take PTR as a temporary variable.
PTR=INFO [START1] (Here, PTR = B200)
Step-5: Now as PTR points to the LIST2, compare the element at last node of LIST2 with all its previous nodes from starting.
Step-6: Since, cycle is not detected; insertion of node directed by B in LIST2 will take place. Since B directs only one node (D), there is no need to discover new lists, if there will be two say x & y then new nodes will be formed with A linked with B and B linked with x and second list will be A linked with B and B linked with y.

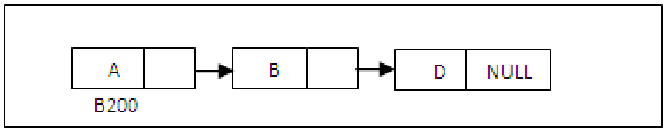

Figure 12: Insertion of D node in the LIST2

Step-7: use Step-5 and 6 for further traversing path.

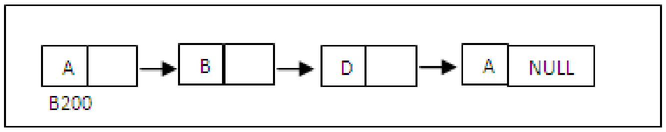

Figure 13: Insertion of node A in List2

Step-8: Use step-5 and check it has cycle or not.
Here, there is a cycle hence counter incremented and the list displayed and then deleted. In addition, if first info field of the LIST1 detects NULL then there is no cycle, counter does not incremented but list deleted.

Step-9: Now, PTR should points to next node of LIST1.
PTR =LINK [PTR]

Step-10: Since, there are no nodes left in LIST0, displaying counter results in number of cycle in the graph.

HERE, there are 7 cycles in the graph (Figure 5)
Therefore, using the algorithm we can find the node that forms the cycles and number of cycle in any digraph.

3.4 Complexity

3.4.1 Worst Case

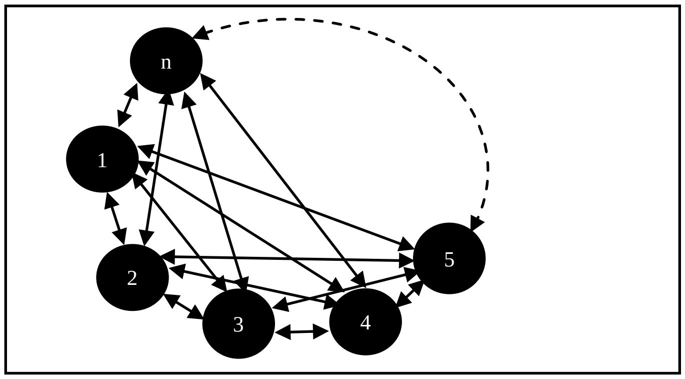

Figure 14: Example of complete directed graph in order to calculate the worst case of algorithm.

Let n be the number of vertices in the directed complete graph.
=>T (n) =Pointers assigning+ New lists+ Cycle detection
=>T (n) =O (n-1) +O ((n-1) ^ (n-1)) +O (n-1)
=>T (n) =O (n^n)

3.4.2 Best Case

=>T (n) = Pointers assigning+ Cycle detection
=>T (n) =O (1) +O (n-1)
=>T (n) =O (n)

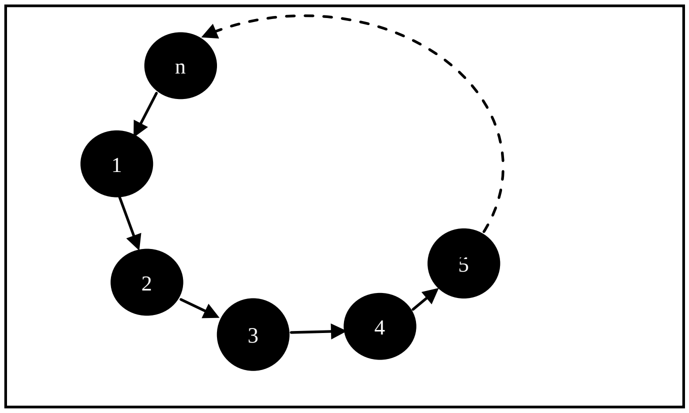

Figure 15: Example of the directed graph in order to find the best case of algorithm

4.0 APPLICATIONS OF CYCLE

a) Cycle detection may be helpful as a way of discovering infinite loops in certain types of computer programs [2].
b) Use of wait-for graphs to detect deadlocks in concurrent system [3].
c) Periodic configurations in cellular automaton simulations may be found by applying cycle detection algorithms to the sequence of automaton states [4].
d) In cryptographic applications, the ability to find two distinct values $x_{\mu-1}$ and $x_{\lambda+\mu-1}$ mapped by some cryptographic function f to the same value x_μ may indicate a weakness in f. For instance, Quisquater and Delescaille [5] apply cycle detection algorithms in the search for a message and a pair of Data Encryption Standard keys that map that message to the same encrypted value; Kaliski, Rivest, and Sherman [6] also use cycle detection algorithms to attack DES. The technique may also use to find a collision in a cryptographic hash function.
e) Analysis of electrical networks, periodic scheduling, analysis of chemical and biological pathways.

5.0 RECOMMENDATION

The proposed SUS's cycle detection method is not only an easier method to detect cycle in any digraph but also very helpful in finding number of cycles in the graph. So, thisalgorithmcan be used in detecting infinite loops in various computer programs, analysis of electrical networks, periodic scheduling and in many more places where there is a need to detect cycle.

6.0 LIMITATION

In this paper, the cycle detection done with the help of linked list. However, this method is easier to implement, in worst case its complexity reaches to O (n^n), which is much higher and because of the formation of new lists in run time it needs large space to act upon. Although, this algorithm removes that linked lists in which traversal is completed, computers with large space used here to execute the proposed algorithm.

7.0 FUTURE SCOPE

The above-proposed algorithmhelps in detecting cycle in any digraph takes much space to execute and its complexity is much higher in case of worst case. Therefore, using this method and logic, in future new logics may define to overcome the complexity and space problems.

8.0 CONCLUSION

In this paper, we have developed a new technique to detect the number of cycles in a directed graph and showed the entire traversed node that forms cycle by displaying it at the time of using counter that incremented at the time of detecting cycle. In addition, when the cycle detects, the same time traversed list is deleted hence, that saved the space. Insertion of nodes from directed graph inserts in the singly linked list. The procedure of this algorithm is much easier to implement and execute for digraph and directed multigraph. This algorithm can be beneficial in detecting infinite loops in certain computer program [2].The proposed algorithm expected to be of great interest in theory and practice alike.

9.0 NOVELTY IN THIS PAPER

In this paper, we have not only presented the new way to detect the cycle in any simple or strongly connected digraph but also presented the new way to count number of cycles in the graph. We used here an efficient data structure named linked list to form new nodes in run-time. It is also used in storing the base address of the newly form linked list in run-time. Hence, we can say that all the application of this algorithm executes in run-time.

10.0 ACKNOWLEDGEMENT

The completion of this paper would not have been possible without the help and guidance of our rev. HOD Prof. Dr.R.Radhakrishnan who was always there to support us and correct us at every step. We would also like to thank our Prof. Rohit Rastogi who guided us on this work. Next, we would like to thank our friends, family and seniors for their motivation and encouragement. Last but definitely not the least we would thank the almighty god without whose grace this paper would not have achieved success.

11.0 REFERENCES

[1]. Piotr Puczynski, "The cycle detection algorithms", Wroclaw University of Technology, Faculty of Management.

[2]. Van Gelder, Allen (1987), "Efficient loop detection in Prolog using the tortoise-and-hare technique", Journal of Logic Programming 4 (1): 23–31, doi: 10.1016/0743-1066(87)90020-3.

[3]. Silberschatz, Abraham; Peter Galvin, Greg Gagne (2003). Operating System Concepts. John Wiley & Sons, INC. p. 260. ISBN 0-471-25060-0.

[4]. Nivasch, Gabriel (2004), "Cycle detection using a stack", Information Processing Letters 90 (3): 135–140, doi: 10.1016/j.ipl.2004.01.016.

[5]. Quisquater, J.-J.; Delescaille, J.-P., "How easy is collision search? Application to DES", Advances in Cryptology – EUROCRYPT '89, Workshop on the Theory and Application of Cryptographic Techniques, Lecture Notes in Computer Science 434, Springer-Verlag, pp. 429–434.

[6]. Kaliski, Burton S., Jr.; Rivest, Ronald; Sherman, Alan T. (1988), "Is the Data Encryption Standard a group? (Results of cycling experiments on DES)", Journal of Cryptology 1 (1): 3–36, doi: 10.1007/BF00206323.

[7]. H.D. Rozenfeld et al. Statistics of cycles: how loopy is your network? J.Phys. A: Math.Gen. 38:4589, 2005.

[8]. M. Medard and S. S. Lumetta. Network reliability and fault tolerance. In J. Proakis, editor, Wiley Encyclopaedia of Engineering.

[9]. Floyd describes algorithms for listing all simple cycles in a directed graph in a 1967 paper: Floyd, R.W. (1967), "Non-deterministic Algorithms", J. ACM14 (4): 636–644, doi:10.1145/321420.321422.

[10]. Brent, R. P. (1980), "An improved Monte Carlo factorization algorithm", BIT20 (2): 176–184, doi: 10.1007/BF01933190.

[11]. Nivasch, Gabriel (2004), "Cycle detection using a stack", Information Processing Letters90 (3): 135–140, doi: 10.1016/j.ipl.2004.01.016.

[12]. Schnorr, Claus P.; Lenstra, Hendrik W. (1984), "A Monte Carlo Factoring Algorithm With Linear Storage", Mathematics of Computation (American Mathematical Society) 43 (167): 289–311, doi:10.2307/2007414, JSTOR 2007414.

[13]. Teske, Edlyn (1998), "A space-efficient algorithm for group structure computation", Mathematics of Computation67 (224): 1637–1663, doi: 10.1090/S0025-5718-98-00968-5.

[14]. Sedgewick, Robert; Szymanski, Thomas G.; Yao, Andrew C.-C. (1982), "The complexity of finding cycles in periodic functions", SIAM Journal on Computing11 (2): 376–390, doi: 10.1137/0211030.

[15]. Van Oorschot, Paul C.; Wiener, Michael J. (1999), "Parallel collision search with cryptanalytic applications", Journal of Cryptology12 (1): 1–28, doi: 10.1007/PL00003816.

[16]. [16] Quisquater, J.-J.; Delescaille, J.-P., "How easy is collision search? Application to DES", Advances in Cryptology – EUROCRYPT '89, Workshop on the Theory and Application of Cryptographic Techniques, Lecture Notes in Computer Science 434, Springer-Verlag, pp. 429–434.

[17]. Fich, Faith Ellen (1981), "Lower bounds for the cycle detection problem", Proc. 13th ACM Symp. Theory of Computation, pp. 96–105, doi: 10.1145/800076.802462.

[18]. Allender, Eric W.; Klawe, Maria M. (1985), "Improved lower bounds for the cycle detection problem",

Theoretical Computer Science36 (2–3): 231–237, doi: 10.1016/0304-3975(85)90044-1.

[19]. Pollard, J. M. (1975), "A Monte Carlo method for factorization", BIT15 (3): 331–334, doi: 10.1007/BF01933667.

[20]. Pollard, J. M. (1978), "Monte Carlo methods for index computation (mod p)", Math. Comp. (American Mathematical Society) 32 (143): 918–924, doi: 10.2307/2006496, JSTOR 2006496.

[21]. Kaliski, Burton S., Jr.; Rivest, Ronald L.; Sherman, Alan T. (1988), "Is the Data Encryption Standard a group? (Results of cycling experiments on DES)", Journal of Cryptology1 (1): 3–36, doi: 10.1007/BF00206323.

Mr. Shubham Rungta is from Ghughli, a town in district Maharajganj (U.P-India). He had received his high school education from Don Bosco School, Nainital (Uttarakhand-India) and intermediate from Gorakhpur (U.P.). At present, he is an IV year student of Computer Science Engineering in ABES Engineering College, Ghaziabad (U.P. - India). His area of interests includes several languages such as C, JAVA, Web Technologies and several subjects such as DBMS, Data Structure, Graph theory, Software Engineering. He is an author certified with two top most journals namely, Springer and IEEE. He is a philanthropist as an active member of NGO named Help Us to Help Child (HUHC).

Ms. Samiksha Srivastava is currently pursuing her graduation in B.Tech in Computer Science and Engineering (IV year)from ABES Engineering College, Ghaziabad (U.P.-India), affiliated to Uttar Pradesh Technical University.Her field of interest includes network security,DBMS, Website Designing, Date Compression and Data Structures. She is an active member of NGO named.

Mr. Uday Shankar Yadav is currently an IV year student of Computer Science Engineering in ABES Engineering College, Ghaziabad (U.P. - India), presently affiliated to Uttar Pradesh Technical University. His area of interests includes DBMS, Data Mining, Pattern Recognition, Date Compression, Soft Computing, and Data Structure. Currently, he completely focused upon the field of Graph theory.

Mr. Rohit Rastogireceived his B.E. degree in Computer Science and Engineering from C.C.S.Univ. Meerut in 2003, the M.E. degree in Computer Science from NITTTR-Chandigarh (National Institute of Technical Teachers Training and Research-affiliated to MHRD, Govt. of India), Punjab Univ. Chandigarh in 2010.

He was Asst. Professor at IMS College, Ghaziabad in computer Sc. Dept. His research interests include Data ware Housing and Data Mining, Design Analysis of Algorithm, Theory of Computation & Formal Languages and Data Bases.

He is a Sr. Asst. Professor of CSE Dept. in ABES Engineering. College, Ghaziabad (U.P.-India), affiliated to Gautam Buddha Tech. University and Mahamaya Tech. University (earlier Uttar Pradesh Tech. University) at present and is engaged in Clustering of Mixed Variety of Data and Attributes with real life application applied by Genetic Algorithm, Pattern Recognition and Artificial Intelligence.

He has served as the technical reviewer of 7 papers in 3rd International Conference on Computing, Communications and Informatics (IC3-2014) at GCET, Greater Noida, NOIDA, India on September, 24-27, 2014 And Currently working as the reviewer for the SPICES-2015 at NIT Kerala, Kojhicode for international conf. of Signal Processing and Communication…

Also currently working as the reviewer in the technical reviewer committee for theINDIA-2015 is Second International Conference on Information System Design and Intelligent Applications organized by Faculty of Engineering, Technology and Management, University of Kalyani, Kalyani-741235, West Bengal, India.

He has authored/co-authored, participated and presented research papers in various Science and Management areas in around 40 International Journals and International conferences including prestigious IEEE and Springer and 10 national conferences including SRM Univ., Amity Univ. and Bharti Vidyapeetha etc. He has guided five ME students in their thesis work and students of UG and PG in around 100 research papers. He has developed many commercial applications and projects and supervised around 30 B.E. students at graduation level projects.

At present, he is a Sr. Asst. Professor of CSE Dept. in ABES Engineering. College, Ghaziabad (U.P.-India), affiliated to Gautam Buddha Tech. University and Mahamaya Tech. University (earlier Uttar Pradesh Tech. University).

His research interests include Data ware Housing and Data Mining, Design Analysis of Algorithm, Theory of Computation & Formal Languages and Data Bases. At present, He is engaged in Clustering of Mixed Variety of Data and Attributes with real life application applied by Genetic Algorithm, Pattern Recognition and Artificial Intelligence.

Also, He is preparing some interesting algorithms on Swarm Intelligence approaches like PSO, ACO and BCO etc.