International Journal of Engineering Technology and Management Sciences[IJETMS]

Website: ijetms.in Issue:4, Volume No.3, November-2019

MAXIMUM SUM ARRAY FIRST SEARCH ALGORITHM FOR GRAPH TRAVERSAL

C.M.T.Karthigeyan[1], C.SatheeshPandian[2]

[1]Assistant Professor Department of CSE Government College of Engineering Bargur,Krishnagiri, Tamilnadu, India.

[2]Assistant Professor Department of CSE Government College of Engineering Bodinayakanur ,Theni, Tamilnadu, India.

Abstract

Graphs are the commonly used data structures that describe a set of objects as nodes and the connections between them as edges. Graph traversal is a technique to find all nodes reachable from a given set of root nodes. To traverse a graph is to process every node in the graph exactly once. The two most widely used algorithms used for traversing a graph are Breadth First Search and Depth First Search. This paper presents a new algorithm namely maximum sum array first search algorithm for traversing a graph. In this algorithm the node with the maximum number of edges if processed first and then its neighboring nodes are processed. This paper presents an algorithm to traverse an undirected or a directed graph and calculates the time and space complexity of the algorithm. The objective of proposed algorithm is to find a new alternative algorithm that can be applied to all types of graphs.

Keywords - Adjacency Matrix, Breadth First Search, Depth First Search, Graph Traversal

Introduction

Graphs are non-linear data structures comprising a finite set of nodes and edges. The nodes are the elements and edges are ordered pairs of connections between the nodes. Graphs are widely used data structure in computer science and different computer applications. Generally, a graph is represented as a pair of sets (V, E). V is the set of vertices or nodes. E is the set of Edges. The elements of a graph are connected through edges. A path or a line between two vertices in a graph is called edges. Two nodes are called adjacent if they are connected through an edge. Path is a sequence of edges between two nodes. It is essentially a traversal starting at one node and ending at another. An undirected graph is one where the edges do not specify a particular direction. The edges are bi-directional. A directed graph is one where the edges can be traversed in a specified direction only. A weighted graph is one where the edges are associated with a weight. This is generally the cost to traverse the edge.

With the development of computer and information system, the research on graph algorithm is wide opened. The two most widely used algorithms used for traversing a graph are Breadth First Search and Depth First Search. BFS is a vertex based technique for finding a shortest path in graph. It uses a Queue data structure which follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. DFS is a edge based technique. It uses the Stack data structure, performs two stages, first visited vertices are pushed into stack and second if there is no vertices then visited vertices are popped.

Graph Traversal Algorithms Breadth First Search (BFS) Algorithm

Breadth First Search (BFS) algorithm traverses a graph in a breadth ward motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration. BFS is a graph traversal algorithm and traversal method which visits all successors of a visited node before visiting any successors of any of those successors. BFS tends to create very wide and short trees. BFS is implemented using a queue, representing the fact that the first node visited is the first node whose successors are visited. BFS algorithm inserts a node into a queue, which we assume is initially empty. Every entry in the array mark is assumed to be unvisited. If the graph is not connected, BFS must be called on a node of each connected component. In BFS we must mark a node visited before inserting it into the queue, so as to avoid placing it on the queue more than once. The algorithm terminates when the queue becomes empty.

The algorithm for BFS is shown below:

- **Step 1:** SET STATUS = 1 (ready state) for each node in G
- **Step 2:**Enqueue the starting node A and set its STATUS = 2 (waiting state)
- **Step 3:** Repeat Steps 4 and 5 until QUEUE is empty
- **Step 4:** Dequeue a node N. Process it and set its STATUS = 3 (processed state).
- **Step 5:**Enqueue all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state) [END OF LOOP]
- **Step 6:** EXIT

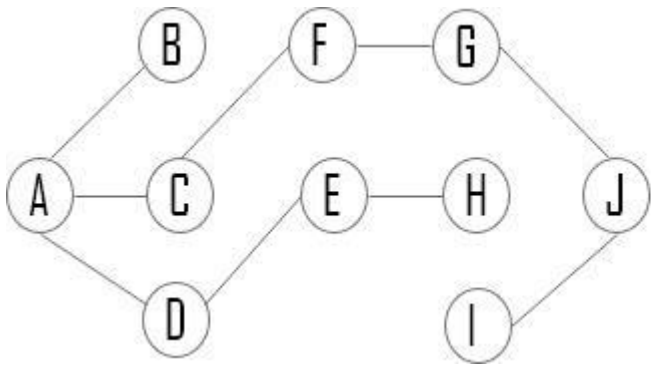

Fig 1: Directed graph

START FROM A

A -> B -> C -> D -> F -> E -> G -> H -> J -> I

So the output of above tree traversal is A,B,C,D,F,E,,G,H,J,I

Depth First Search (DFS) Algorithm

Depth First Search (DFS) algorithm traverses a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search when a dead end occurs in any iteration. A depth first search of a graph differs from a breadth first search in that the exploration of a vertex v is suspended as soon as a new vertex is reached. At this time, exploration of the new vertex u begins. When this new vertex has been explored, the exploration of u continues. The search terminates when all reached vertices have been fully explored. This search process is best described recursively. DFS uses a strategy that searches deeper in the graph whenever possible. The predecessor sub graph produced by DFS may be composed of several trees, because the search may be repeated from several sources. This predecessor sub graph forms a depth-first forest E composed of several depth-first trees and the edges in E are called tree edges.

The algorithm for BFS is shown below:

- **Step 1:** SET STATUS = 1 (ready state) for each node in G
- **Step 2:** Push the starting node A on the stack and set its STATUS = 2 (waiting state)
- **Step 3:** Repeat Steps 4 and 5 until STACK is empty
- **Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)
- **Step 5:** Push on the stack all the neighbors of N That are in the ready state(whose STATUS=1) and set their STATUS=2 (waiting state) [END OF LOOP]
- **Step 6:** EXIT

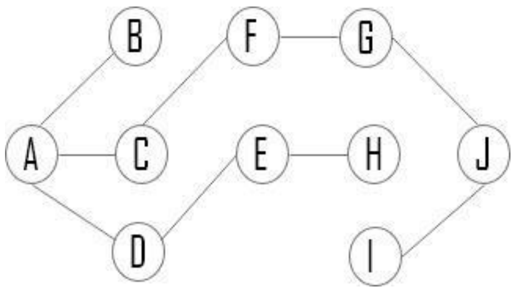

Fig 2: Directed graph

START FROM A

A->B->C->F->G->J->I->D->E->H

So the output of above tree traversal is A,B.C,F,G,J,I.D,E,H

Proposed Algorithm

So far, many different variants of BFS and DFS algorithm have been implemented sequentially as well as in parallel manner. In all parallel implementations, the unvisited nodes of the root node are visited, but in our implementation, the node with the maximum number of edges if processed first and then its neighboring nodes are processed. This paper presents a new algorithm namely maximum sum array first search algorithm for traversing a graph and then compares the traversal result of a directed and undirected graph using BFS, DFS and our proposed algorithm.

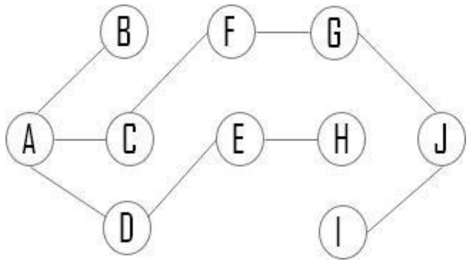

Fig 3: Directed graph

Consider the above graph. The algorithm for traversing the graph is as follows: The adjacency matrix of the graph along with an array SUM that corresponds to the summation of the edge count of different

nodes is given
Starting node: A
STEP 1:
Initially, construct an adjacency matrix of graph along with the SUM array

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**SUM**
A	0	1	1	1	0	0	0	0	0	0	**3**
B	1	0	0	0	0	0	0	0	0	0	1
C	1	0	0	0	0	1	0	0	0	0	2
D	1	0	0	0	1	0	0	0	0	0	2
E	0	0	0	1	0	0	0	1	0	0	2
F	0	0	1	0	0	0	1	0	0	0	2
G	0	0	0	0	0	1	0	0	0	1	2
H	0	0	0	0	1	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	1	1
J	0	0	0	0	0	0	1	0	1	0	2

Initially Visited array will be

null	null	null	null	null	null	null	null	null	null

STEP 2:
A row has the maximum sum, so choose it and visit all the nodes. Then make the maximum sum row A and visited columns B, C, D as 0

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**SUM**
A	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	1	0	0	0	0	1
D	0	0	0	0	1	0	0	0	0	0	1
E	0	0	0	0	0	0	0	1	0	0	1
F	0	0	0	0	0	0	1	0	0	0	1
G	0	0	0	0	0	1	0	0	0	1	**2**
H	0	0	0	0	1	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	1	1
J	0	0	0	0	0	0	1	0	1	0	2

The visited array will become:

A	B	C	D	null	null	null	null	null	null

STEP 3:
G row has the maximum sum, so choose it and visit all the nodes. Then make the maximum sum row G and visited columns F, J as 0

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**SUM**
A	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0

C	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	1	0	0	0	0	0	1
E	0	0	0	0	0	0	0	1	0	0	1
F	0	0	0	0	0	0	1	0	0	0	1
G	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	1	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	1	0	1	0	**2**

The visited array will become:

A	B	C	D	G	F	J	null	null	null

STEP 4:

J row has the maximum sum, so choose it and visit all the nodes. Then make the maximum sum row J and visited columns G, I as 0

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**SUM**
A	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	1	0	0	0	0	0	**1**
E	0	0	0	0	0	0	0	1	0	0	1
F	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	1	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0

The visited array will become:

A	B	C	D	G	F	J	I	null	null

STEP 5:

D row has the maximum sum, so choose it and visit all the nodes. Then make the maximum sum row D and visited columns A, E as 0

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**SUM**
A	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	1	0	0	**1**
F	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0

The visited array will become:

A	B	C	D	G	F	J	I	E	null

STEP 6: E row has the maximum sum, so choose it and visit all the nodes. Then make the maximum sum row E and visited columns D, H as 0

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**SUM**
A	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0

The visited array will become:

A	B	C	D	G	F	J	I	E	H

As all the nodes get visited, the proposed algorithm traverses the graph. The order of traversal is A, B, C, D, G, F, J, I, E, H

Analysis of Time and Space Complexity

Time complexity and space complexity of BFS, DFS and our proposed algorithm shown in below figure.

Algorithm	Time Complexity	Space Complexity
Breadth First Search	O(V+E)	O(V)
Depth First Search	O(V+E)	O(V)
Maximum Sum Array Search	O(V+E)	$O(V^{2)}$

Fig 4: Comparison of Time and Space Complexity

Future Scope and Conclusion

The proposed algorithm can be further improved in terms of running time and space. The time complexity is same in the worst case when the algorithm traverses each vertex along each path. The algorithm sometimes cannot traverse the graph accurately. In case of directed graph the proposed algorithm sometime fails to traverse it accurately similar to BFS and DFS. However, the proposed algorithm has a time complexity much less than BFS or DFS when it does not have to go through each edge to traverse the nodes. This also reduces the space complexity as well. However, the accuracy can still be improved. The proposed algorithm can still be made time and space efficient.

References

[1] Introduction to Algorithms (2nd ed) by Cormen Thomas H, Leiserson ,Charles E,Clifford(2001)

[2] Danny Sleator, "Parallel and Sequential Data Structures and Algorithms,15-210 (fall 2013) ", Sept. 24 , 2013.

[3] Graph theory and applications by Paul Van Dooren

[4] Applications of graph theory by S.G.Shirinivas S.Vetrivel , Dr.N.M. Elango

[5] Nykamp DQ, "Undirected graph definition" on Math Insight. Available: http://mathinsight.org/definition/undirected_graph

[6] F. Y. Chin, J. Lam, and I.-N. Chen. Efficient parallel algorithms for some graph problems. Communications of the ACM, 25(9):659–65, 1982

[7] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. Journal of Computer and System Sciences, 38(1):86–124, February 1989.

[8] Wiki books contributors, Available: https://en.m.wikibook.org/wiki/Data Structures/Graphs