Project Objective

The project objective is to analyze and implement the Depth First Search algorithm and optimize it to reduce its running time by providing it with different datasets and analyzing how it behaves when treated with differing amount of data. After careful analysis of the time taken for our program to execute and what affected that time in our dataset, we will update the program to optimize those edge cases as well.

Since the algorithm has a few flaws as well as its benefits all we can do is optimize its implementation to get as much of a boost in performance as we can get which is why we’re experimenting on this paper to implement the code for the

Applications of Depth First Search

Depth First Search has been used for years since its inception to solve both graph-based and tree based problems and has been a very handy tool for programmers to quickly solve any sort of traversal problem that comes their way.

Graph Traversal

Depth first Search mainly is used in Graph traversal algorithms and is used to traverse both weighted and unweighted, directed and undirected graphs even if they contain negative edges. The algorithm will still work.

State Space Problems

Depth first Search is used in AI to search through a state space to find a solution to a problem for that AI. Depth first search is perfect for Uninformed State Space search when looking for a certain state that can occur after multiple levels of the tree.

Tree Traversal

Depth first Search is used to traverse the Tree data structure in order to find a certain value if the tree is not binary in nature.

Detecting Cycles in a Graph

DFS is used to detect if a graph has cycles in by detecting back-edges

Path Finding

Depth first search can be specialized to be used as a path finding algorithm by just tracking the parent of each node.

Connected Component Problem

Depth first search is used to find connected components in a graph.

Related Work

Previously, since its inception DFS has been improved upon and specialized to complete the task it was required to complete. DFS is a very flexible algorithm and since it stores more information about the nodes, we can therefore specialize it.

DFS has only been specialized and there have been mathematical proofs for its average efficiency in running and people have therefore, not gone for optimizing the coding process and using the fastest possible components to get the best runtime possible in C++.

I have not been able to find related work that tries to this optimization in C++.

Experiments

Dataset 1:

In this dataset, we were testing how quickly can the algorithm actually find something if it is right at the start of the first node.

Nodes: 0, 1, 2, 3

Edges: 0 – 1, 1 – 2, 2 – 3 , 0 – 3

Node to find : 1

This should ideally take a very small amount of time and should be better than BFS since it takes the extra time to initialize its data

This dataset is the ideal situation for a DFS algorithm and would provide the fastest and the best results.

Time Taken 0.001s

For 50, 5000, 50000

The Time Taken was: 0.001, 0.001, 0.001 respectively

Dataset 2:

In this dataset we were testing what would happen if the element to find is the last element in our graph.

Nodes: 0, 1, 2, 3, 4, 5

Edges: 0 – 1, 1 – 2, 2 – 3, 3 – 4, 1 – 4, 2 – 4, 1 – 3, 0 – 5

Node to find: 5

This dataset is the worst case scenario for the algorithm as it would have to go through all the other nodes before arriving at the required node.

Since the dataset was small the difference wasn’t obvious in the delta Time but it is the worst case scenario for our Algorithm

Time Taken: 0.002s

For 50, 5000, 50000

The Time Taken was: 0.02, 0.421, 2.8 respectively

Dataset 3:

In this dataset we tested the average situation that occurs when looking at a graph or a tree traversal where the node is right in the middle somewhere for the algorithm to find

Nodes: 0, 1, 2, 3, 4

Edges: 0 – 1, 1 – 2, 0 – 3, 3 – 4, 1 – 3, 2 – 4

Node to Find: 3

This is the average case and should be pretty quick to find how long does it take.

Time Taken: 0.0015s

For 50, 5000, 50000

The Time Taken was: 0.01, 0.143, 1.12 respectively

Conclusion

Depth first search is already a good algorithm which can be specialized into different solutions to different problems such as topological sorting but when going down to code it in C++ we can use certain techniques to decrease the total runtime to save time when trying to work with very large datasets. We can achieve that by using core C++ components which are mapped directly to machine instructions and therefore usually take less time.

References

[1] S. Baswana, S. R. Chaudhury, K. Choudhary, and S. Khan. Dynamic DFS in undirected graphs: breaking the O(m) barrier. In Proceedings of the twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms, pages 730–739. SIAM, 2016.

[2] S. Baswana and K. Choudhary. On dynamic DFS tree in directed graphs. In International Symposium on Mathematical Foundations of Computer Science, pages 102–114. Springer, 2015. [3] S. Baswana, A. Goel, and S. Khan. Incremental DFS algorithms: a theoretical and experimental study. In Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 53–72. SIAM, 2018.

[4] S. Baswana, S. K. Gupta, and A. Tulsyan. Fault tolerant and fully dynamic DFS in undirected graphs: simple yet efficient. arXiv preprint arXiv:1810.01726, 2018.

[5] L. Chen, R. Duan, R. Wang, and H. Zhang. Improved algorithms for maintaining DFS tree in undirected graphs. CoRR, abs/1607.04913, 2016.

[6] Broder, A., R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata and A. Tomkins, J. Wiener, „Graph Structure in the Web,‟ Computer Networks, Vol. 33, pp. 309–320, June 2000. [7]. Aggarwal, A., R.J. Anderson, „A Random NC Algorithm for Depth First Search,‟ Cominatorica, 8(1), pp. 1–12,1988.

[8]. F. Corman. Real-time Railway Traffic Management, Dispatching in complex, large and busy railway networks. Ph.D. thesis, TechnischeUniversiteitDelft,The Netherlands, December 2010. 90-5584-133-1.