Introduction to Algorithms Engineering Project

By - Sriteja Pashya (2021111019), Keval Jain (2021111030)

Project 6:

Modify the algorithm of Tarjan-Vishkin to incorporate Union-Find data structure as we add edges to G'. Check how this works in practice wrt other algorithms such as Tarjan's algorithm.

Project Ideas -

After applying Tarjan-Vishkin, any connected component in the auxiliary graph is a biconnected component in the original graph. By using Union-Find data structure, we can avoid adding extra edges by which the connectivity of the auxiliary graph does not change.
In the attached code for Tarjan-Vishkin with Union-Find, at every step of adding an edge, if the two vertices happen to be in the same component, we skip adding this edge.

Analysis of Tarjan-Vishkin with Union Find vs Tarjan Vishkin without Union Find vs Normal Tarjan code is done for various types of graphs. Analysis is also done for various spanning trees, and for each tree, various possibilities of selecting the rooted vertex. Comparison of time taken for every step of the algorithm is also done.

All codes used have been stress tested for correctness.

Notation Used -

General: N - Number of vertices, M - Number of edges

Algorithm Used: T - Tarjan, TV - Tarjan Vishkin, TVUF - Tarjan Vishkin with Union Find

Tree Used: _D - DFS Tree, _B - BFS Tree

Rooted Vertex: _R - Random rooted vertex, _4S - 4-sweep rooted vertex

General Syntax: AlgoUsed_TreeUsed_RootedVertex

Implementation -

- in adjacency list `vector<vector<int>> adj` is used to store the graph
- the vertices in each component are stored in `vector<set<int>> ans` to print all components neatly

Tarjan's - standard implementation, by calculating low and high values and using a stack `stack<pair<int, int>> st` to keep track of components

Tarjan-Vishkin - follows algorithm as described in the paper

An Efficient Parallel
Biconnectivity Algorithm, SIAM Journal of
Computing, 1985, Vol. 14, No. 4

Tarjan-Vishkin DSU - uses union-find data structure instead of adding edges in auxiliary graph

The values in the columns are program duration times in seconds.

Performance Testing -

These graphs have been self generated randomly.
The aim is to get a good idea on how the algorithms compare.

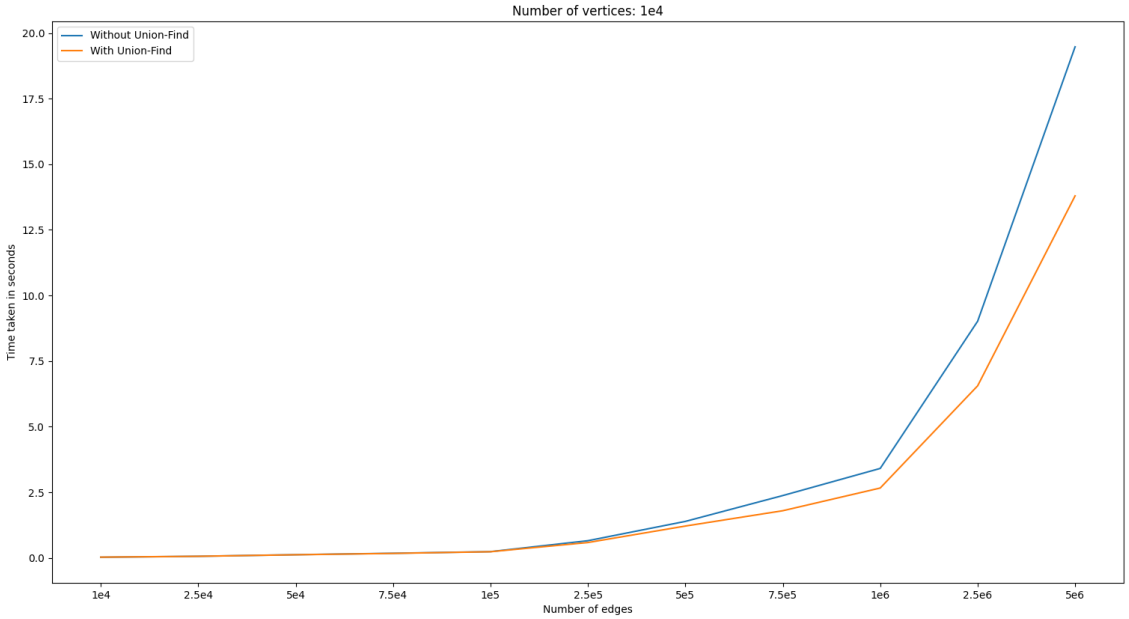
N	M	T	TV_D_R	TV_B_R	TVUF_D_R	TVUF_B_R
1e4	1e4	0.014024	0.026523	0.032426	0.030229	0.031513
1e4	2.5e4	0.0214	0.062584	0.076521	0.063223	0.076109
1e4	5e4	0.032649	0.123767	0.156185	0.121554	0.147609
1e4	7.5e4	0.042312	0.177779	0.23068	0.173659	0.209432
1e4	1e5	0.053944	0.240193	0.315788	0.236872	0.27015
1e4	2.5e5	0.104638	0.656884	1.07537	0.587502	0.67222
1e4	5e5	0.197214	1.39223	1.83952	1.21525	1.29294
1e4	7.5e5	0.286889	2.37697	2.92697	1.79901	1.98766
1e4	1e6	0.362357	3.41079	4.5328	2.66425	2.77235
1e4	2.5e6	0.890946	9.02061	11.3013	6.55929	6.93398
1e4	5e6	1.78095	19.4747	23.5644	13.7968	13.8937
1e5	1e4	0.028866	0.042806	0.04905	0.039524	1.09897
1e5	1e5	0.095954	0.316961	0.326603	0.328922	0.520348
1e5	1e6	0.711118	4.44724	4.65767	3.28888	3.69083

Analysis:

Tarjan Vishkin With Union Find vs Without Union Find

The general trend which can be seen is that -

- Tarjan-Vishkin with Union Find is slower than Tarjan-Vishkin without Union Find for sparse graphs, (where number of edges are comparable to the number of vertices).
  This is expected as in sparse graphs, we would not be skipping many edges, and the cost of maintaining the Union Find data structure would exceed any benefits of such skips.
- However, Tarjan_Vishkin with Union Find becomes faster than Tarjan-Vishkin without Union Find for dense graphs, (where number of edges are much greater than the number of vertices). In such graphs, many edges are skipped from being added into the auxiliary graph, overcoming any costs from the Union Find Data Structure.



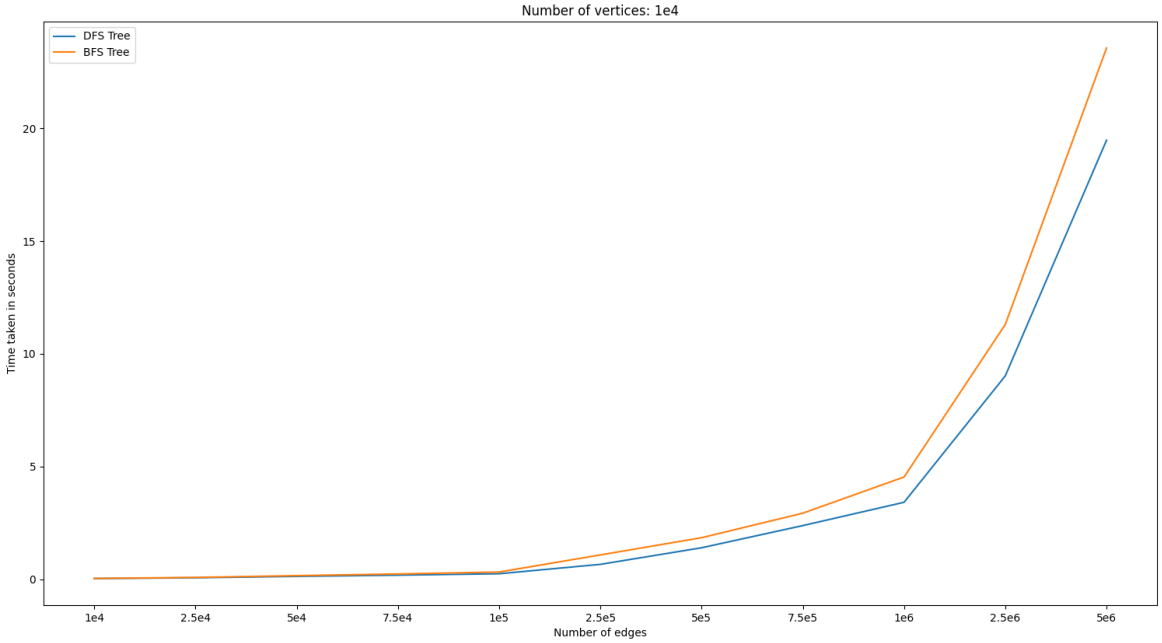
DFS Tree vs BFS Tree

- Building a DFS Tree using recursion is not a good implementation for very large graphs, because recursive DFS calls will lead to stack overflow.
- Although the stack limit can be increased, this is will have performance implications.
- We have increased our stack limit to 1GB to allow DFS tree to work on large graphs.

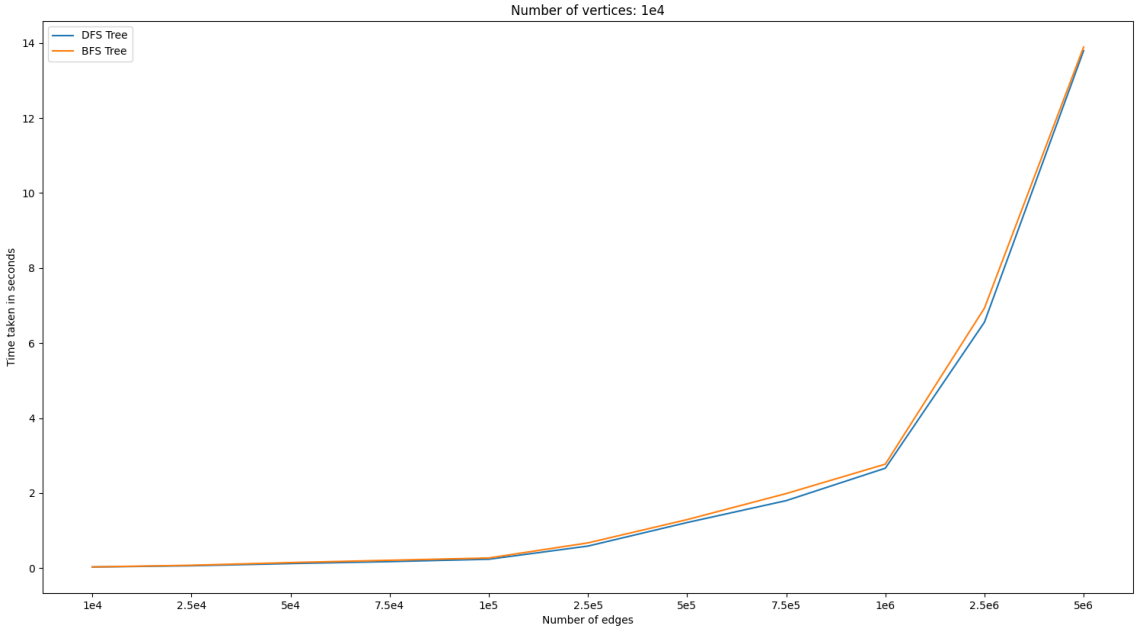
However the general trend which can be seen is that

- DFS tree implementation is faster than BFS Tree implementation for Tarjan Vishkin without Union Find
- DFS tree implementation is comparable to BFS Tree implementation for Tarjan Vishkin with Union Find

Tarjan Vishkin Without Union Find - DFS Tree vs BFS Tree



Tarjan Vishkin With Union Find - DFS Tree vs BFS Tree



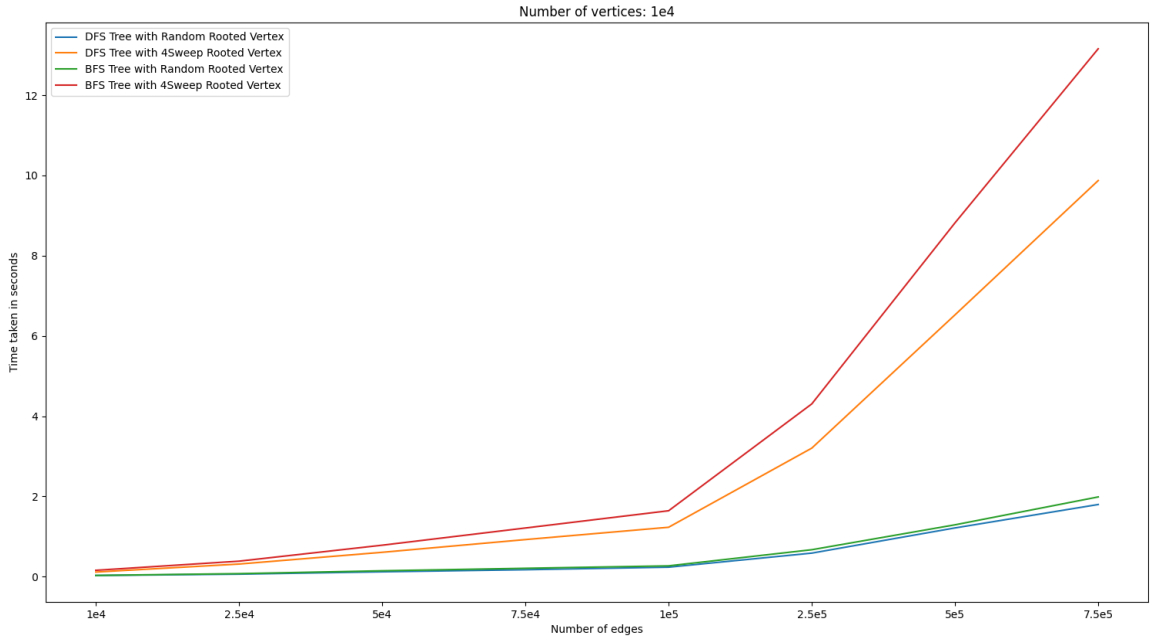
Choosing the root vertex: Random vs 4 Sweep -

The comparison below is done for Tarjan Vishkin with Union Find when we the rooted vertex chosen is random vs the vertex chosen is through the 4-sweep method.

N	M	TVUF_D_R	TVUF_D_4S	TVUF_B_R	TVUF_B_4S
1e4	1e4	0.030229	0.115622	0.031513	0.159256
1e4	2.5e4	0.063223	0.315149	0.076109	0.386802
1e4	5e4	0.121554	0.60786	0.147609	0.784995
1e4	7.5e4	0.173659	0.924847	0.209432	1.21358
1e4	1e5	0.236872	1.23198	0.27015	1.6432
1e4	2.5e5	0.587502	3.2041	0.67222	4.30634
1e4	5e5	1.21525	6.52545	1.29294	8.81841
1e4	7.5e5	1.79901	9.87171	1.98766	13.1558
1e5	1e4	0.039524	0.240244	1.09897	18.3857
1e5	1e5	0.328922	1.32603	0.520348	4.76137
1e5	1e6	3.28888	15.4204	3.69083	20.1549

As it is clearly visible from above, algorithms taking the rooted vertex from the 4-sweep method appear to be significantly slower than randomly choosing the vertex. This is because the cost of doing the 4 BFS traversals for every connected component is significant compared to any advantage it may provide.

Tarjan Vishkin With Union Find DFS Tree - Random Root vs 4Sweep



Comparing Different Steps: DFS Tree vs BFS Tree -

The comparison below is done for Tarjan Vishkin with Union Find, between BFS tree and DFS tree methods, for the time taken to

1. building tree and calculating low and high values
2. building auxiliary graph
3. Finding connected components in auxiliary graph

TVUF_D:

N	**M**	Building tree and calculating low and high values	Building auxiliary graph	Finding connected components in auxiliary graph
1e4	1e4	0.010812	0.010294	0.006186
1e4	5e4	0.025562	0.069587	0.02729
1e4	1e5	0.038382	0.142512	0.053812
1e4	5e5	0.115975	0.842056	0.265769
1e5	1e4	0.019738	0.004259	0.01551
1e5	1e5	0.07516	0.155199	0.155199
1e5	1e6	0.487234	2.57165	0.839774

TVUF_B:

N	M	Building tree and calculating low and high values	Building auxiliary graph	Finding connected components in auxiliary graph
1e4	1e4	0.014808	0.014808	0.006995
1e4	5e4	0.023471	0.104571	0.02766
1e4	1e5	0.03639	0.205845	0.053155
1e4	5e5	0.111007	1.03657	0.26997
1e5	1e4	1.01649	0.004436	0.021008
1e5	1e5	0.255068	0.18089	0.091321
1e5	1e6	0.389736	3.03636	0.815768

- One particular thing to note is that the time taken for the BFS Tree is significantly higher for the case of 1e5 vertices and 1e4 edges, for Tarjan Vishkin with Union Find for both randomly chosen vertex and vertex chosen through the 4Sweep method.
  This could be because at this condition the graph is extremely unconnected, and most vertices do not have an edge joining them.

Conclusions Observed

- Tarjan Vishkin with Union Find appears to be significantly faster than Tarjan Vishkin without Union Find, for dense graphs, while it being slightly slower for sparse graphs.
- Tarjan Vishkin with DFS tree appears to be faster than with BFS Tree
- It appears to be better to chose a random vertex for rooting the tree vs any other algorithm to find the root.