University of Colombo School of Computing

SCS3206 Graph Theory

Assignment

20000103 - D.A. Amarasinghe

September 24, 2023

Introduction

Following report will explore and compare the efficiency of three well-known graph algorithms: Kruskal's Algorithm, Prim's MST Algorithm (Lazy Version), and Prim's MST Algorithm (Eager Version) concerning different graph representations (Adjacency List and Adjacency Matrix) as well as different densities.

Algorithm Implementation

Above algorithms were implemented using C++. Seperate implementations for each of the graph representations as adjacency list and adjacency matrix. To generate the dataset I've used another C++ script as 'generate.cpp'. The generated graphs will be stored in text file with respective names and those names will be added to the 'scritpt.sh' file The purpose of 'script.sh' file is to automatically run each algorithms and output the execution time to a csv file for further comparison.

After taking the execution time for each of ther algorithms in each representation, a comparison was made using 'matplotlib' library in python. The full dataset which i used to evaluate the efficiency of algorithms has been uploaded to a google drive because of the larger file size (nearly 1GB). You can find the full data set here. Download it and placed the content under the root(this) directory.

Logical Comparison

For the comparison, logrithamic values have been used to clearly visualize the exponential growth or decay of the execution time.

- **Kruskal's Algorithm** - It seems the kruskal's algorithm perform well with sparse trees in adjacency list representation. But as the graph becomes more dense the execution time growth linearly for adjacency lists. When the graph becomes dense adjacency matrix representation performs better than the adjacency list. (see Figure1)
- **Prim's Algorithm (Lazy Version)** - It seems the prim's lazy version perform well with sparse trees in adjacency list representation. But as the graph becomes more dense the execution time growth linearly for adjacency lists. Adjacency matrix representation for prim's lazy version performs equaly with both sparse and dense graphs. (see Figure2)
- **Prim's Algorithm (Eager Version)** - Prim's eager version perform well with sparse trees in adjacency list representation. But as the graph becomes more dense the execution time growth linearly for adjacency lists. Difference is the adjacency list representation of eager version performs better than adjacency list representation of lazy version Adjacency matrix representation for prim's eager version performs equaly with both sparse and dense graphs. (see Figure3)

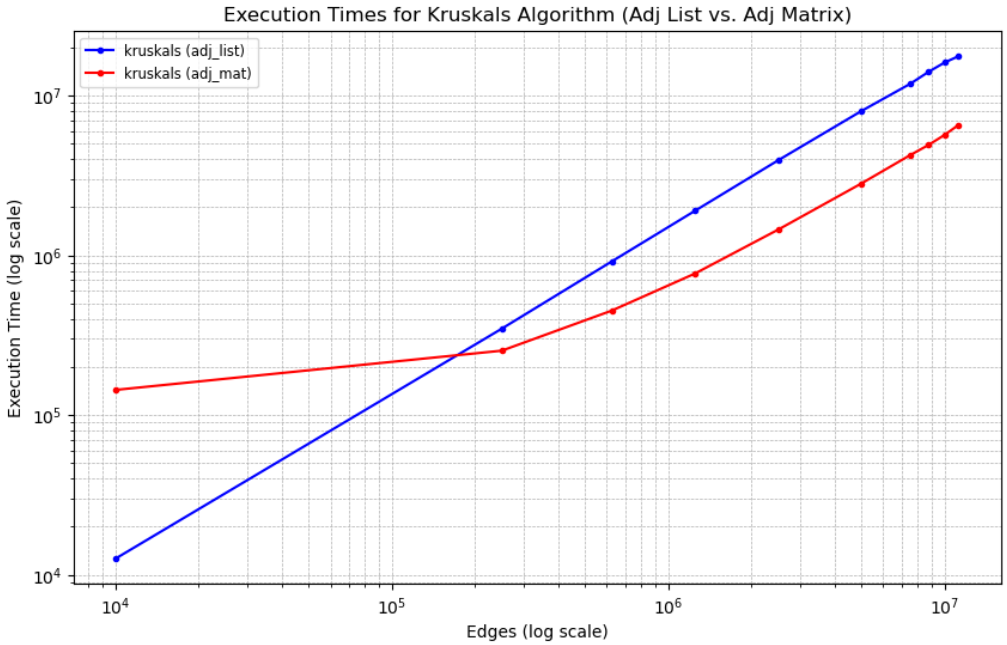

Figure 1: Kruskal's algorithm performace adj-list vs adj-mat

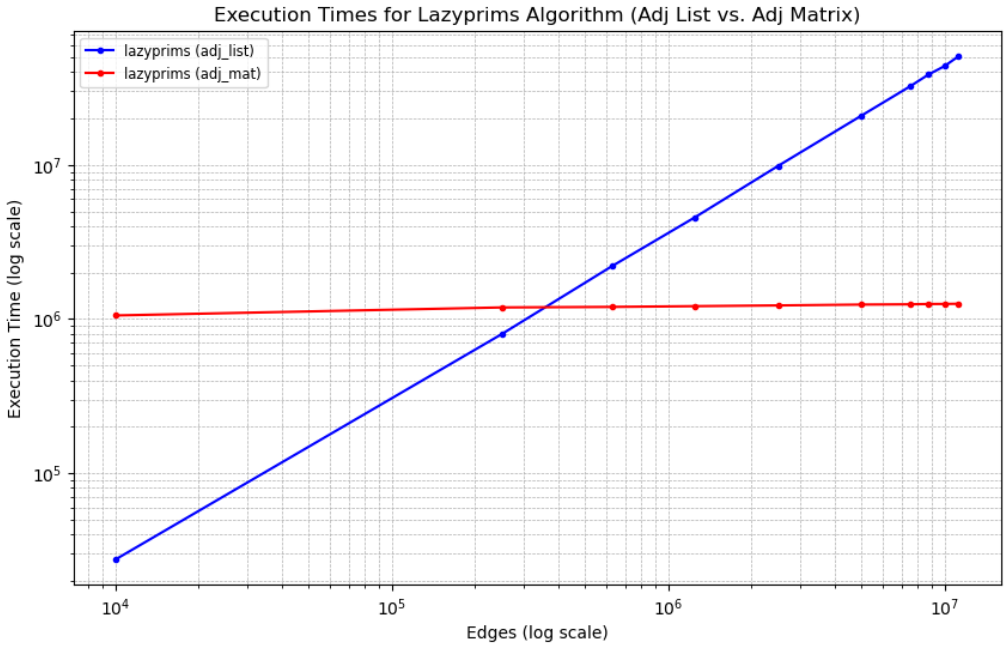

Figure 2: Prim's (lazy) algorithm performace adj-list vs adj-mat

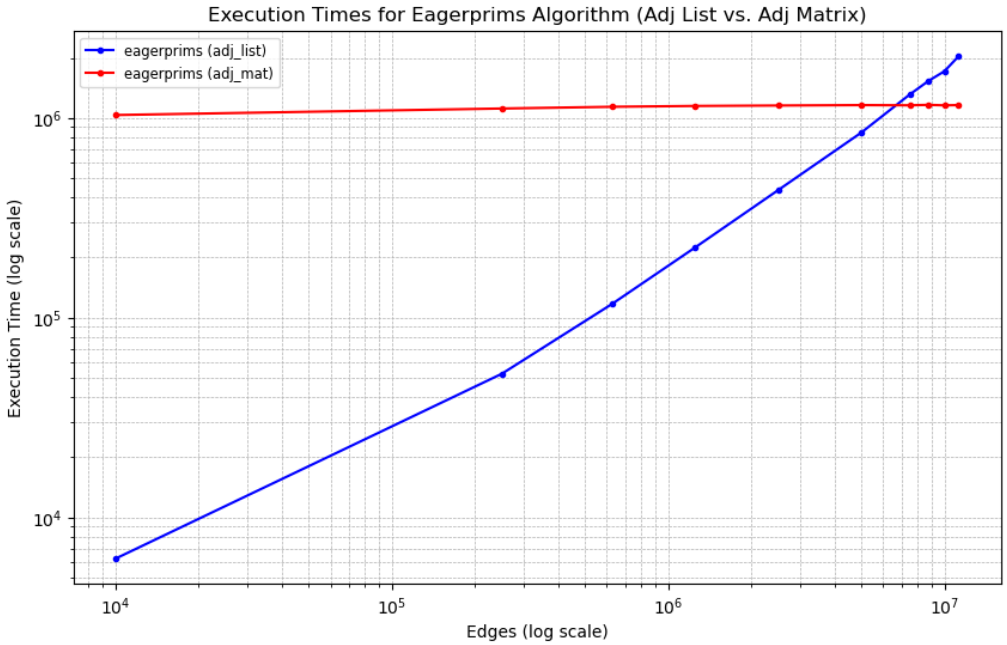

Figure 3: Prim's (eager) algorithm performace adj-list vs adj-mat

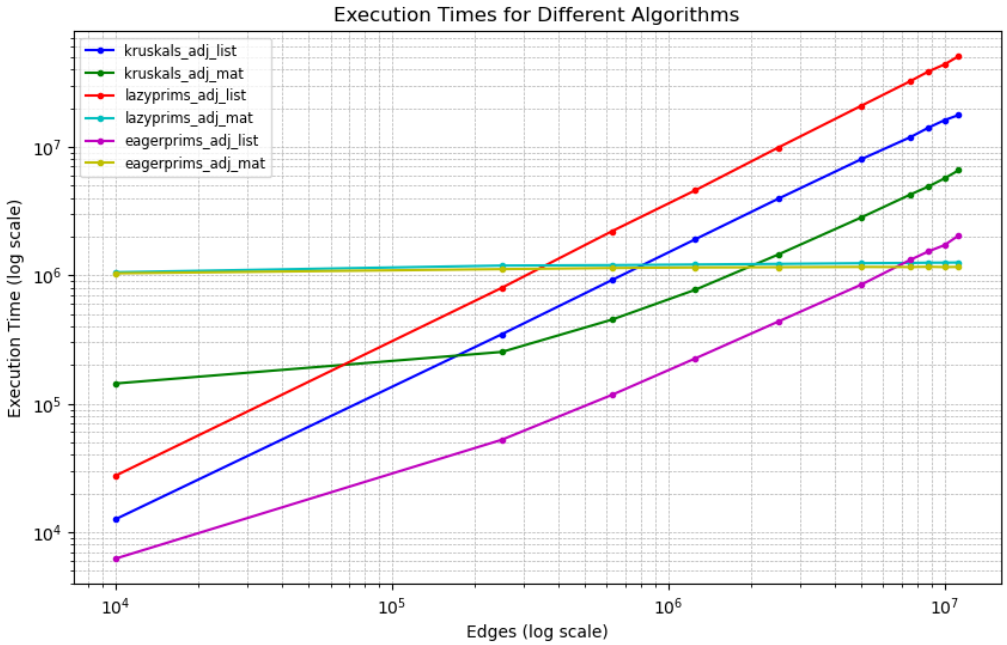

Figure 4: All algorithm sparse vs dense

Execution Time

edges	Kruskals Adj-list	Kruskals Adj-matrix	Lazyprims Adj-list	Lazyprims Adj-matrix	Eagerprims Adj-list	Eagerprims Adj-matrix
10,002	12,604	143,600	27,509	1,054,823	6,220	1,034,789
250,050	347,350	252,851	798,235	1,189,128	52,395	1,115,453
625,125	915,720	450,636	2,197,843	1,198,472	116,975	1,137,840
1,250,250	1,899,821	768,975	4,570,254	1,211,832	224,513	1,149,109
2,500,500	3,935,408	1,448,063	9,850,363	1,225,510	435,349	1,154,874
5,001,000	8,005,084	2,817,011	20,859,161	1,240,978	845,393	1,160,269
7,501,500	11,839,005	4,235,022	32,308,722	1,245,903	1,314,237	1,157,702
8,751,750	14,082,782	4,909,452	38,572,570	1,251,501	1,533,504	1,163,443
10,002,000	16,112,940	5,673,281	43,764,995	1,252,803	1,715,370	1,156,917
11,252,250	17,671,339	6,564,123	50,733,838	1,256,216	2,039,929	1,160,732

Figure 5: Execution time (in microseconds)

Conclusion

In conclusion, the efficiency of Kruskal's algorithm, Prim's MST Algorithm (Lazy Version), and Prim's MST Algorithm (Eager Version) depends on various factors, including the graph's density and the chosen data representation.

The adjacency list representation performs well with less number of edges(sparse) with compared to the number of nodes where the adjacency matrix representation performs well with large number of edges(dense). Both the prims algorithms performs eqaully in adjacency matrix representation while lazy prims performs well than eager prims in the adjacency list representation, spaecially with sparse graphs. Kruskal's algorithm in adjacency list representation works well with sparse graphs, but when the density increases adjacency matrix representation of the kruskal algorithm performs better than it's adjacency list representation.

Overall, understanding the trade-offs and characteristics of each algorithm is crucial for selecting the most efficient one for a given scenario.

Note* : Please follow the instructions given in README.md file to use the algorithms.