By Anando Zaman

Adjacency list vs Matrix comparison and use case:

If we are given a graph to query frequently to check if an edge exists with the available options of using adjacency list or adjacency matrix without considering memory usage, then adjacency matrix is best since it can do this at constant time by checking if something is connected. Adjacency list would be effective if we were concerned about memory as the matrix takes up quite a lot of memory in comparison.

Shortest paths for directed graphs

For shortest paths, there are many algorithms to use such as Topological sort, BellMan-Ford, BFS or Dijkstras. If it is a directed path with the possibility of both positive/negative weights, then it is best to use BellMan-Ford Algo aslong as there are NO NEGATIVE CYCLES. If it is an Acyclic graph however, then it may be better to use Topological sort as it works fastest assuming it is a DAG. Dijkstras has a complexity of $O(ELogV)$ which is linearithmic, but only works if the edges are positive weighted. BFS works for cases regardless of the weighting aslong as all the weights are the same. In other words, if they are all constants. Recall, Dijkstras and BFS are quite similar but Dijkstras supports the ability to work with different weightings which can only be positive while BFS has either neg/pos weights but have to be the same sign and magnitude. This is because BFS does not consider weights in its computation. Another thing to note is that BFS and Dijkstra's both work on directed/undirected graphs.

Design a data structure to perform efficient spell check

To do this, it is good to use a 26-way trie(key=word, val=bit) if the memory usage does not matter. This is because tries have very fast look up times with the downside of space usage as each node will have R null links. Search misses are $O(log_R N)$. If memory does matter, then it may be useful to use TST(Ternary Search Trees) since they are faster than hashing especially for search misses. They are also more flexible than Red-black BSTs and have better space usage than R-way tries. However, they have worse search hit and miss compared to R-way tries. Its kind of like the comparison of linear probing and seperate chaining. Linear probing takes up more memory but can give fast searches assuming uniform hashing assumption while seperate-chaining can conserve space at the cost of slower search time due to linkedlists formed during collisions.

Cases for heaviest edge in MST

Recall this was done in one of the exercise problems in Section/chapter 4. Assuming the MST is computed using Kruskal's algorithm, there could be cases where heaviest edge could belong, if it is the only edge connecting a node to rest of the graph, then this edge has to exist on the MST.