Assignment One Report
Bipartite Graph Data Structures Evaluation

For this assignment, four different data structures were used to store information on a bipartite graph and perform operations on this information: Array-Linked List, Linked List-Linked List and Binary Search Tree for an Adjacency List[1] representation, and a 2D Array representing an Adjacency Matrix[2]. In each, basic operations were performed on the same set of data: Insertion, Search and Deletion.

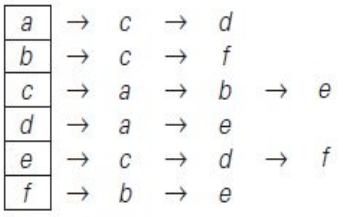


Figure 1: Adjacency List

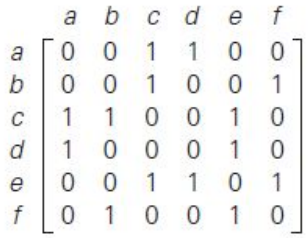


Figure 2: Adjacency Matrix

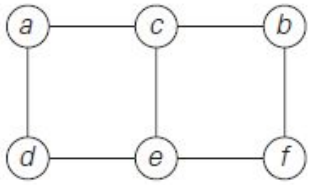
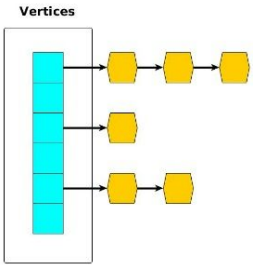


Figure 3: An undirected graph

Array - Linked List



In this data structure, an array of Linked Lists was used to represent vertices, and each position in the array had a Linked List representing edges from that vertex. This was supported by an array that represented vertex existence.

Insertion

Vertex insertion is given by the average case for insertion in a value-indexed array) which is O(n) complexity for all cases. This shares the same complexity as an unsorted linked list (O(n)) , however if there was an index for the end of the list then the complexity becomes O(1) *constant* complexity. For edge insertion in the linked list array, complexity is given by O(n + c), where c is vertex id. In the best case where the linked list also has an index, complexity becomes O(2).

Search

Vertex search O(n) complexity given the average-case of a basic array. This is the same as the worst case, however in the best case where the array is sorted, search becomes O(1) given that it's just getting a value at index. Edge search is O(n + c) for average case in our array-linked list scenario, given that we have O(n) complexity for vertex lookup, and O(n) complexity for linked list searching. The worst case doesn't differ, however if the vertex array were to be sorted then the best case would be O(n+1) complexity.

Deletion

Vertex deletion is more involved because it also requires deletion of all the edges. The average case then is O(n + c) , where c is the amount of edges to be delete. The best case is where the Vertex has no edges, and the complexity becomes O(n) for finding the vertex to be deleted, and O(1) if the array is sorted. Average case for edge insertion is also O(n+c), for c being the vertex to be found. If the vertex array is sorted, it would become O(n+1).

Linked List - Linked List

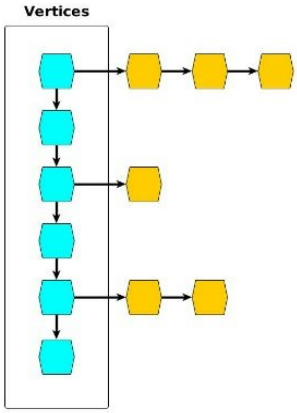


Figure 5: Linked List - Linked List diagram

In this data structure, both vertices and edges were stored in linked lists. This creates a higher space requirement and remove the possibility of fast indexing lookup from the array. However, it does allow for more dynamic data management that arrays lacked.

Insertion

Vertex insertion average case is O(n) , worst case is unchanged but best case could have an index for the end of the list and have O(1) complexity. Edge insertion also requires searching the linked list, meaning the complexity becomes O(n + c) for average and worst case, where c is the vertex number. Best case however would be O(n) if the edge list had an index for the end of the list.

Search

Vertex search is O(n) for best, worst and average case.Similarly for edge search, which however involves finding the origin vertex making it's worst and average and best case O(n + c) where c is the vertex number.

Deletion

Like search, vertex deletion also has O(n+c) for best, worst and average case since it includes edge deletion. Again, this means that since deleting an edge in the edge linked-list first requires searching for the vertex it originates from, making it's best, worst and average case O(n + c) where c is the vertex number. Best case could be improved to O(1+1) if both lists were indexed.

Binary Tree - Linked List

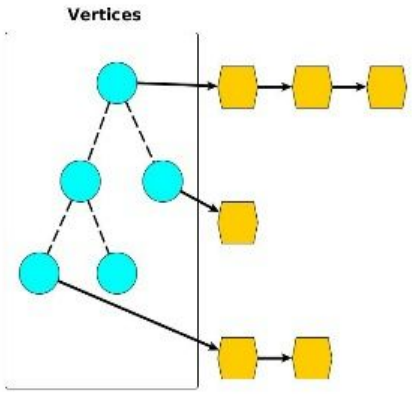


Figure 6: BST-Linked List diagram

For this implementation, vertex information was stored in a binary search tree, and the edge information was in a linked list. This meant far better lookup times for vertices.

Insertion

For Vertex insertion, a Binary search tree insertion has O(log(n)) complexity for average and best case, but however a worst case of O(n) if the tree is sorted as it would make it unbalanced. For edge insertion, this means a log(c) lookup of the vertex first where c is the vertex number, giving a complexity of O(log(c) + n) for average case. In the worst case, the binary tree is sorted meaning O(c + n). In the best case, there is an index for the end the edge list, meaning O(log(c) + 1) insertion.

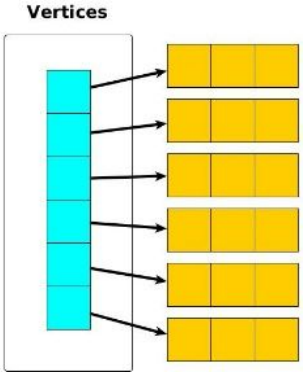
Searching

For Vertex searching, the complexity is again O(log(n)) for average and best case. The worst case still remains a sorted tree, where a search would be of O(n) complexity. For edge insertion, average and best case are of O(log(c) + n) complexity, and in the worst case of a sorted binary tree, O(c + n) where c is the vertex number.

Deletion

Much like searching, deletion retains a complexity of O(log(n)+c) for average and best case vertex deletion, and a worst case of O(n+c) for a sorted tree,where c are the edges to be deleted. For edge deletion, this also retains O(log(c) + n) complexity for average case. Best case could have an indexed edge list giving O(log(c) + 1) , and a worst case of O(c + n) if the binary tree were sorted.

Array-Array



This implementation used a 2D array in order to represent an adjacency matrix (see figure 2). For this implementation, vertex information was chosen to be stored in a normal array, whereas the edges used a single 2D array to represent the data. This meant that there was more space in the 2D array, but at the cost of having extra space needed for the vertices.

Insertion

Vertex insertion had O(n) complexity for best, worst and average case given a value-indexed array. This also holds for edge insertion, which no longer requires source vertex lookup, giving it O(n) for all cases

Search

Vertex searching still maintains O(n) complexity for worst and average cases, with O(1) lookups for best case. However, the edge searching now has O(m * n) complexity for worst and average case, where m and n are the different 2D array dimensions for all cases, making it the worst time complexity for searching of all the data structures. However, it can have O(1) efficiency for index lookups.

Deletion

Vertex deletion still has O(n+(m*n)) average and worst case complexity given edge deletion, although with indexing it has O(2) best case. Edge deletion has O(m*n) worst and average case, but can also have O(1) lookups for best case with indexing.