**Task #4**

**2. Compare between adjacency matrix and adjacency list.**

A graph can be represented in two ways:

* **Adjacency List:** An Adjacency list is an array consisting of the address of all the linked lists. The first node of the linked list represents the vertex and the remaining lists connected to this node represents the vertices to which this node is connected. This representation can also be used to represent a weighted graph. The linked list can slightly be changed to even store the weight of the edge.
* **Adjacency Matrix:** Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

**Time and Space Complexity of Adjacency Matrix and List**

* The space complexity of an adjacency list is O ( V + E ) O(V + E) O(V+E), and in the worst case, it is O ( V 2 ) O(V^{2}) O(V2) when every node is connected to all the other nodes.
* The space complexity of the adjacency matrix is O ( V 2 ) O(V^{2}) O(V2).
* An adjacency list is more efficient, in terms of storage requirements, for representing a graph. By choosing an adjacency list as a way to store the graph in memory, this may save space.
* Assuming the graph has *n* vertices, the time complexity to build such a matrix is *O (n^2)*.
* If *m* is the number of edges in a graph, then the time complexity for building such a list is *O(m).*

**Which is better adjacency matrix or adjacency list?**

Adjacency Matrix may be used for representing Dense Graphs and Adjacency List for representing Sparse Graphs. Note: Dense Graph are those which has large number of edges and sparse graphs are those which has small number of edges.

Adjacency list is much more efficient for the storage of the graph, especially sparse graphs, when there is a lot less edges than nodes. In terms of the accessing time, adjacency matrix is much more efficient when finding the relationships in a graph.

**3. how does DFS and BFS work?**

**Breadth-First Search:**

BFS, Breadth-First Search, is a vertex-based technique for finding the shortest path in the graph. It uses a Queue data structure that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. It is slower than DFS.

**Depth First Search:**

DFS, Depth First Search, is an edge-based technique. It uses the Stack data structure and performs two stages, first visited vertices are pushed into the stack, and second if there are no vertices then visited vertices are popped.

**Differences between BFS and DFS**.

|  |  |  |
| --- | --- | --- |
|  | BFS | DFS |
| **Data structure** | BFS uses Queue to find the shortest path | DFS uses Stack to find the shortest path |
| **Source** | BFS is better when target is closer to Source | DFS is better when target is far from source |
| **Speed** | BFS is slower than DFS | DFS is faster than BFS |
| **Time Complexity** | Time Complexity of BFS = O(V+E) where V is vertices and E is edges | Time Complexity of DFS is also O(V+E) where V is vertices and E is edges |

**6. Implement Unordered/Multiset in python**

Allows elements to occur multiple times. It supports the same methods as Python set. It supports set operations like Union, intersection, and (symmetric), order does not matter in discriminating multisets. The essential difference between the set and the multiset is that in a set the keys must be unique, while a multiset permits duplicate keys.

To install use: pip install multiset

.