3805ICT Advanced Algorithms – Assignment 1

Chris Vuong – s5193954

**Question 3:** Most graph computing algorithms assume that the adjacency matrix and

adjacency lists can be stored in computer memory so the following 2 operations will

be fast:

* Is vertex v connected to vertex u?
* Produce a list of all vertices connected to v.

However, the advent of very large graphs (e.g., 100,000 vertices and > 1,000,000

edges) prevents the memory storage of the adjacency matrix and standard adjacency

lists for these graphs. Design and implement in C++ a data structure for storing such

graphs that can effectively perform the 2 operations listed above. Demonstrate

the efficiency of your data structure.

Answer:

Idea:

**Adjacency matrix:**

This method is using an NxN table with n as the node of the graph. Inside the table is the true or false value, which can be replaced with binary integers 0 and 1. So for the 100,000 vertices, we need a 100000x100000 table, which is 10000000000 units of storage to store everything. Assuming we are storing an edge as a bit, we may need up to 10000000000 Bs or 1.25 GBs which is too large for a practice.

Although we do not make this adj matrix into practice because of the disadvantage of the storage, let’s assume how this method performs.

*Is vertex v connected to vertex u?*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 1 | 0 |

Assuming we have the 4x4 table above and checking for whether 3 is connected to 1, we must check graph[i][j] = 1 or 0, where i is the columns and j is the row. In the case of the bi-directed graph, graph[i][j] may be different to graph[j][i]. O(1)

*Produce a list of all vertices connected to v.*

To produce a list of all vertices connected to v, this method must loop through all the n elements of v list to check which vertices are connected to v. O(n)

**Adjacency list:**

Compared to the adjacency matrix, the adjacency list is using a lesser number of storage. In the adjacency list, we create an array of a list that will store all the vertices that connect to the vertex index. The storage of this method is not strictly O(n2), but it will be depending on the graph. As the definition, an undirected graph of n vertices can have a maximum of n(n-1) edges and directed vertices can have a maximum of n(n-1)/2 edges. Therefore, in the worst-case scenario, the storage of this method can be O(n2).

*Pseudo-code :*

|  |
| --- |
| Int \*\*graph = new int[N][];  // to adding an edge v to vertex u  Function Adding\_u\_to\_v (u,v, graph) => void  graph[u].push\_back(v); // assuming push\_back is a function that add v to the array u in graph  graph[v].push\_back(u);// incase this graph is undirected. This line can be omitted if it is directed graph  // Is vertex v connected to vertex u?  Function Is\_v\_connected\_u (u,v,graph) => bool  Loop to all graph[u] as edge : O(e)  If edge == v:  Return true;  Return false;  // Produce a list of all vertices connected to v.  Function vetices\_connected\_u (u, graph) => list  Return make\_list(graph[u] ); O(1) |

*Efficiency :*

|  |  |  |
| --- | --- | --- |
| Vertices | V |  |
| Edges | E |  |
| Storages | O(V+E) | The storage will be the total number of vertices and edges |
| Adding | O(1) | The adding is just allocating a new number into the list |
| Checking if u is connected in v | <= O(E) | To check if u is connected to v, it is depended on the number of edges u has. In the worst case, u have all the edges of the graph then it takes O(E). |
| Produce a list of all vertices connected to v | O(1) | Because this is a 2D array, the list is produced at the time the graph is created. |

OUTPUT:

|  |
| --- |
| ./a  Generating the Adjacency list :  Nodes that connect to node 100:  10079 -> 76924 -> 93250 -> 75720 -> 32030 -> 44678 -> 10178 -> 10070 -> 10326 -> 87721 -> 31442 -> 5678 -> 81913 -> 63721 -> 21894 -> 16832 -> 66495 -> 33746 -> 64349 -> 74468 -> 81583 -> 14640 -> 94921 -> 40646 -> 14405 -> 48480 -> 96028 -> 86211 -> 40386 -> 8534 -> 21566 -> 24542 -> 92155 -> 53150 -> 77777 -> 64383 -> 90247 -> 65330 -> 82349 -> 99538 -> 28133 -> 67231 -> 51058 -> 27459 -> 90641 -> 49798 -> 40171 -> 69626 -> 20062 ->  Is 123 connected to 100 :0  Is 10079 connected to 100: 1  Generating the Adjacency matrix : Fail from compiling |