Graph Representation – Adjacency Matrix

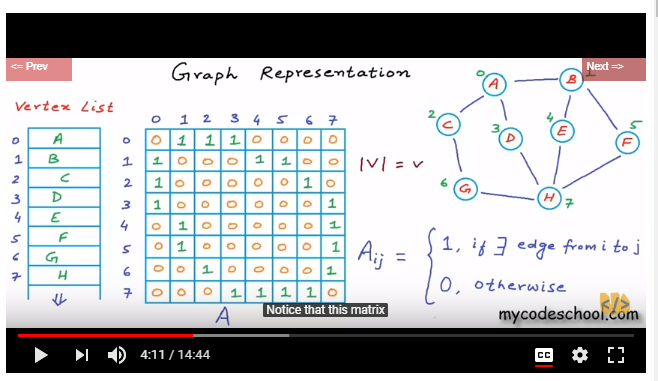
\*We have seen graph representation as Vertex-Edge List which has time complexity of O(|E|) which can be square of number of vertex in worst case. In graph anything running in the order of edges is very costly. We want to keep the cost in order of number of vertices.

\*One more possible design is we can store the edges in a two dimensional array or matrix. We can have a two-dimensional matrix or array of size V\*V where V is number of vertices.

\*If the graph is unweighted and each vertex has an index between 0 and V-1 then in this 2d array we can set A[i][j] as 1 if there is an edge from i to j.

\*Vertex 0 is connected to vertex 1,2 and 3. Vertex 1 is connected to 0,4 and 5. This is undirected graph so if we have an edge from 0 to 1, we also have an edge from 1 to 0. Node 2 is connected to 0 and 6. 3 is connected to 0 and 7 and 4 is connected to 1 and 7. 5 once gain is connected to 1 and 7. 6 is connected to 2 and 7 and 7 is connected to 3,4,5,6. All remaining position should be set to 0.

Look at the image here



\*Notice that this matrix is symmetric. For an undirected graph, this matrix would be symmetric because A[i][j]=A[j][i]. We have two positions filled for each edge. We need to just go through one of the symmetric half of the matrix. This won’t be true for an undirected graph. Only one position will be filled for each edge and we will have to go through the entire matrix to see all the edges.

\*Now this kind of representation where edges or connections are stores in a 2D array is called adjacency matrix.

\*Now we want to find all nodes adjacent to node named F.

If we are given name of a node than we first need to know its index. And to know the index we need to scan the vertex list. Once we find the position we can go that row in the adjacency matrix and we can scan the complete row to find all the adjacent nodes.

Finding the index of the node in vertex list in worst case will cost us time proportional to the number of vertices. And scanning the row in the adjacency matrix will again cost us time proportional to the number of vertices.

So overall time complexity is O (v).

\*Now most of the time while performing operations, we must pass indices to avoid scanning the vertex list all the time. If we know an index, we can figure out the name in constant time because in an array we can access an element in constant time but if we know a name and want to figure out index then it will cost us O(v).

\* Now what will be cost of finding if 2 nodes are connected or not.

We can get either index or the name of the two nodes. If we have indices of the two nodes we simply need to look at value in a particular row and particular column. We simply need to look at A[i][j] for some values of I and J which will happen in constant time.

But if names are given it will cost O(v).

\*The cost of scanning vertex list can be avoided. We can create a hash table with names and indices as key value pairs and then the time cost of finding index from name would also be constant i.e. O(1).

Operation Time-Cost

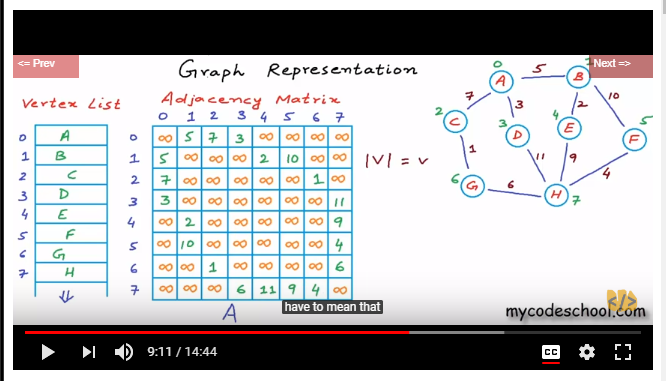
Finding adjacent nodes O(v)

Finding if two nodes are connected O(1)

\*Now we want to store a weighted graph in adjacency matrix representation.

A[i][j] can be set as weight of the edge. For non—existence edges we can have a very large value set may be infinity.

Look at the image here



\*With adjacency matrix representation we have improved upon time but we have gone really high on memory usage. There is a big space-time tradeoff here. We are using V square space. We are storing if two nodes are connected but at the same time we are storing if 2 nodes are not connected.

\*If graph is sparse we waste lot of memory storing 0. Most of the practical scenario graph won’t have number edges close to V square. Let’s take example of facebook.

Suppose we have billion of users.

If|V|=10^9

Will we ever have 10^18 connections in the graph. That would mean everyone in the network is friend of everyone in the network. A user on an average will have 10^3 friends.

So number of edges will be

|E|=(10^9 \* 10^3)/2= 5\*10^11 (divide by 2 as it’s an undirected graph)

5\*10^11 << 10^18

10^18 bytes = 1000 PB

5\*10^11 bytes = 0.5 TB

\*Adjacency matrix is good when a graph is dense.

\* Webpages on world wide web can be considered as a directed graph with web pages as node and link to other pages as an edge.