Homework 1: Graph Theory and Graph ADTs

Unity ID: avshirod

Problem 1 :

A customer asks you to provide the most efficient implementation of the following operations over a large graph (about 10,000 vertices): add/delete edges/vertices.

1. What questions would you ask the customer that will help you to make the right decision about the customer's request? [Hints: Check the types of graphs]

Ans: The efficiency of the graph implementation would depend on how we represent it (Space) and that in turn will decide how much time the frequent operations will take (Time).
So the things we would like to know about the graph from the customer are:
The directionality of the graph (Edges are directed vs undirected), are the edges labelled, are the edges weighted, does the graph have any loops or self-loops, is the graph dense vs sparse.
After we know these basics about the graph, and which operations will the customer usually perform on the graph (plus if the graph is static vs dynamic), we can decide what will be an efficient implementation.

2. Provide three scenarios depending on the answers to those questions.

Ans:
Scenario 1: (e.g. User relationships on Spotify)
Graph will be Directed, Unlabeled, Unweighted, Single, No loops, Sparse
Users won't be added that frequently (lower number of update operations). Majority of operations would be look-up (let's say to suggest friends or songs)

Scenario 2: (e.g. User-Song relationships on Spotify)
Graph will be Undirected, Labelled, Weighted, Multiple edges, No loops, Dense
Graph will be dynamic, and would need to updated frequently

Scenario 3: (e.g. Artist-Song relationships on Spotify)
Graph will be Undirected, Unlabeled, Unweighted, Multiple edges, No loops, Sparse
Graph would be static, updates still less than look-ups

3. For each scenario, propose the most efficient data structure and supply the time complexity of the requested operations in terms of big-O. What is the space requirement for the selected data structure?

Ans:

Scenario	Data structure	Time complexity	Space requirement
1	Adjacency matrix	O(n^2)	O(n^2) (n is nodes)
2	Adjacency list	O(1)	O(|v| + |e|) (e >> v)
3	Adjacency matrix	O(n^2)	O(n^2) (n is nodes)

Problem 2 :

Let G be a simple directed graph with N nodes. The path matrix of G is the n-square matrix P defined as follows: Each (i,j) element of the matrix is 1 if there is a path from node i to node j in G.

1. Using only an adjacency matrix A of graph G defined as a boolean (0,1) matrix, design an algorithm that constructs the matrix P in the most efficient manner. Write any mathematical relationship for P you will use for your algorithm and justify this relationship (formal or informal proof is up to you). [Hint: What is A times A? What is A times A times A?]

2. Provide a pseudo-code of the algorithm.

3. Analyse the time complexity of your algorithm in terms of big-O.

4. Define the values of N, for which this algorithm becomes impractical (e.g., will take more than a day to get an answer) on my laptop running at 2.5 GHz (i.e., 2.4 * 10^9 floating point operations per second).

5. If my laptop has 512 MB, 1GB, or 16 GB of RAM, will your answer to (4) change given my memory constraints (i.e., in terms of big-O for space requirements) and how many bytes do you require per your selected data structure; crude estimation with the justification is enough?

Ans:

1. When we multiply adjacency matrix with itself (A*A), we will get a Path matrix with path length of two.
Let's say, initially, A -> B, and B -> C. A is not connected to C directly.
But we can reach to C from A, by using B as an intermediate node. (A -> B -> C)

```
2. # G - Simple directed graph with N nodes
# P - Path matrix
# A - Adjacency matrix

def matrix_mult(m1, m2):
      return m1*m2

P = copy(A)
for _ in range(n-1):
      P *= A
```

3. Time complexity of above algorithm would depend on the number of nodes in the matrix (N). Considering that we will be performing a N*N matrix multiplication N times, it would be O(N^3).

4. 2.4 * 10^9 floating operations per second means around 2.0736 * 10^14 floating operations in a day's limit.

For our algorithm to fail this limit, N^3 needs to be greater than this.

Using calculator, the cube root comes out to be a little less than 59190. So if the matrix has 59190 or more nodes, the instructor's laptop would take more than a day to compute P from A using above algorithm.

5. As we are representing both P and A as adjacency matrix with Boolean data, we can represent N*N values in N*N bits (i.e. N*N/8 bytes, or N*N/(8*10^6) MBs, N*N/(8*10^9) GBs).

If the laptop has RAM restrictions, then that would restrict the number of bytes available to us to represent the graph.

GB	MB	Bytes	Bits	Nodes	Rounding off
0.5	500	500000000	4E+09	63245.55	63246
1	1000	1E+09	8E+09	89442.72	89443
16	16000	1.6E+10	1.28E+11	357770.9	357771