Problem 3

There are three kinds of implementation. Here I have chosen adjancent list. The pros and cons are listed below.

Collection of edges

Advantages

- Less complexity
  The ADT of the graph will be a Set. No need for more complex data structure
- Adding Node/edge cost less (only need to check edges)
- The rep of this implementation will be true
  As long as the edge rep kept, this ADT keeps rep when all the time (with TreeSet implementation, kept graph identical edge free)

Disadvantages

- Unable to efficiently listed all nodes in the graph
  Have to go through all edges each time and create a temperory set to avoid dumplicates
- Unable to efficiently locate all edges of a parent node: looks up requireds to loop through each edge
- Unable to efficiently loop through the gragh
  When doing DFS over the graph, since this ADT does not support locate edges of a node, each time we need to loop all edges to find the next path

Adjacency List

Advantages

- Efficiently locate all edges of a parent nodes
- Efficiently locate all nodes by looking at keys of the maps
- Automcatically kept all the order of the graph (with TreeMap/TreeSet implementations)

Disadvantages

- Complex ADT e.g. `TreeMap<String, TreeSet<Edge>>`
  There is some mental burden when coding the graph
- Cost of adding edge/node is higher (check map and then check list)
- The rep invariant need additional checks
  When adding new edge, we need to ensure node already exists in the map, while collection of edge does not required

Adjacency matrix

Advantages

- Efficiently loop up a specific edges/edges of parent nodes
- Efficient add edge function e.g. `O(1)`

- Efficiently look up nodes of the graph

Disadvantages

- High Memory consuming when haveing holes in the graph
- Unable to support multi-edge graph natively
  Workaround maybe use list instead of number in the graph, which cause more memory consumption
- Complex ADT
  Required additional list to store nodes, and then provision matrix based on node list

My Choice

I have chosen **Adjancent List**. Since I has fast loop up of nodes as well as edges from a parent node, which fulfilled all requirements of the graph interface we need to have.

It also save memeory comparing to adjancent martix and can automatically order the node and edges in alphabatical order without further coding (by using TreeSet/TreeMap) Implementation.

Based on the time/space complexity, I think adgancent list is the most suitable one for current requirements. It can also easily convert to other ADT so it has a good extensibility as well.