1. Given a Vertex object x, give Java code to enumerate its outgoing edges.

   To enumerate through vertex object x's outgoing edges you would use a for each loop: for(edge e: x.vertex.edgesFrom())

2. Given a VertexAndDist object x and a new distance d, give Java code to create an updated object with the same vertex as x but with distance d.

   In the Vertex and Dist class, it has a comment saying that the pair cannot be modified so you have to create a new pair. To make a new pair you need to make a decreaser to store the next vertexAndDist then make a new vertexAndDist as seen below:

   Decreaser ¡VertexAndDist¿ distOfW = handles.get(e.to);
   distOfW.decrease(new VertexAndDist(e.to, dist));

3. For simplicity of implementation, we use HashMaps to map vertices to handles/parents and edges to weights. How could you modify the Vertex and/or Edge classes, as well as the maps themselves, to implement the maps using ordinary arrays, with no hashing? Be sure to address both the vertex and edge maps in your answer. (Hint: consider the Vertex's "id" field for inspiration.)

   Each vertex has its own id value. These id values can be stored in an array so that you know which vertex your accessing. So the id can be used as an index in an array. The Edge class doesn't have an id to go along with it but it can tell you the vertex before and after it. So maybe knowing the order to the vertex's from before, you can map out the order of the edges based on which vertex their from is and which vertex is going to. Or you could put the edges in an array based on weight, similar idea to the Kruskal's algorithm. It is important to note here though that the id is private while the to and from is public for the Edge class. So the id can not be used on side of the class which results in the id being accessed only from the public vertex method in the class. The to and from are public though and can be used in other classes.

   For questions 4-7, please be sure to justify the running times you claim using what you know about the cost of Dijkstra's algorithm and the meaning of dense and sparse graphs.

4. Suppose we know that our input graph G = (V, E) is dense. What is the asymptotic running time of Dijkstra's algorithm on G in terms of the number of vertices $|V|$?

   If G is dense then it has the maximum number of edges for its vertices. With this Dijkstra's algorithm has more paths to search. The time complexity is O($(|V|+|E|)log|V|$) in the original algorithm. Since the graph is dense the edges $|E|$ is proportionate to $|V|^2$. So this results in O($|V|^2$log$|V|$).

5. Now suppose we know that our input graph G = (V, E) is sparse. What is the asymptotic running time in terms of $|V|$?

   If G is spares then the number of vertices and edges are about the same. So the run time of the original Dijkstra's algorithm would be O($|V| + |E|log|V|$). Since the graph is sparse $|V|$ and $|E|$ are proportionate and the time complexity is now O($2|V|log|V|$) and the 2 can be taken away to be O($|V|log|V|$).

A Fibonacci heap is a fancy priority queue data structure. For a heap of size n, it takes (log n) time to do an extractMin() operation but only O(1) time to do an insert or decrease operation. Suppose we replace the binary heap used in Dijkstra's algorithm by a Fibonacci heap.

6. If the graph is dense, what is the asymptotic complexity of Dijkstra's algorithm using a Fibonacci heap, in terms of $|V|$?

   When using the Fibonacci heap in Dijkstra's algorithm it has a run time of $O(|E| + |V|log|V|)$. So when the graph is dense $|E|$ is proportional to $|V|^2$. Substituting this in gives a run time of $O(|V|^2 + |V|log|V|)$. This can be reduced to $O(|V|^2)$ from big O notation rules.

7. If the graph is sparse, what is the asymptotic complexity of Dijkstra's algorithm using a Fibonacci heap, in terms of $|V|$?

   When using the Fibonacci heap in Dijkstra's algorithm it has a run time of $O(|E|+|V|log|V|)$. So when the graph is dense $|E|$ is proportional to $|V|$.Substituting this in gives a run time of $O(|V|+|V|log|V|)$. This can be reduced to $O(|V|log|V|)$ from big O notation rules.