ReadMe

Structure of my implementation

<1> HeapItem

I created an element data structure "HeapItem" to hold the handle, key and value of type T. Since the min-first priority queue is based array, the heap object takes an integer as the reference, which will be updated according to the index of the queue.

<2> Priority queue

My implementation of the priority queue is based on ArrayList, and the instance variable, "pqSize", is used for the "isEmpty()" method and acts as index indicator. A method call "Heapify(int i)" is used to bubble down to fix up the queue, and a while loop is used to bubble up to fix the queue. Those are mainly used in the method of "insert" and "decreaseKey". In addition, the "swap" method is the key to the implementation of this part.

<3> Shortest path

I used an array of Vertex to record every vertex's parent and an array of integer to record the edge's id by which a particular vertex can be arrived. For the non-layover-aware part, it is just the implementation of Dijkstra's algorithm, and the distance of a particular vertex is the total time cost from the start vertex to it, which is just derived by adding up the weight of relevant edges. For the extension part, the distance of a particular vertex is the layover time plus the distance for non-layover-aware part.

Big Mistakes I made

In the shortest path part, when I implemented the "returnPath", my original thought was making an array of child of each vertex, then starting from the start vertex to find then end vertex indicated by the endId. The original code looks like:

```
Vertex start = this.G.get(startId);
ArrayList<Integer> edge = new ArrayList<Integer>();
 while(start != null && start.id() != endId  ){
    Vertex.EdgeIterator ir = start.adj();
    while(ir.hasNext()){
       Edge e = ir.next();
       if(e.to() == this.child[start.id()]){
            edge.add(e.id());
        }
    }
    start = this.child[start.id()];
 }
```

This kind of implementation caused an error of "`ERROR: path is not connected in G`". This comes from the fact that there may be different flights from vertex u to vertex v. Also, implementing "this.child[u.id()] = v" in the dequeue part was a big mistake, because v would be dequeued after u, and it can be any adjacent vertex. So when u is dequeued, v cannot be determined.

This may be a typical bug, at least for me, so I include it in ReadMe to document.