Graph ADT

```
void insertVertex(V vertex);
vector<V> getVertices();
bool containsVertex(V vertex);
void removeVertex(V vertex);
```

List<T> T: type of elements in list

Type parameters:
- V — vertices
- W — weights
- L — labels

```
void insertEdge(V src, V dest, W weight, L label);
Edge<V,W,L> getEdge(V src, V dest);
bool containsEdge(V src, V dest);
void removeEdge(V src, V dest);
vector<V> getNeighbor(V src);
vector<Edge<V,W,L>> getOutgoingEdges(V src);
vector<Edge<V,W,L>> getIncomingEdges(V dest);
```

```
template <typename V, typename W, typename L>
class Edge {
    public:
        V source;
        V destination;
        W weight;
        L label;
};
```

How to implement?

1. LinkedGraph → node class, edge class
   difficult to summarize

2. AdjacencyList Graph:

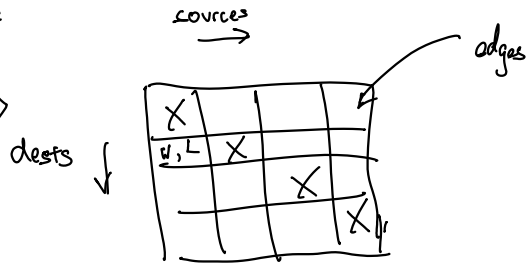
   Dictionary<V, List<Edge<V, W, L>>*>*
   (source vertex; getOutgoingEdges)

   Dictionary<V, Dictionary<V, Edge<V,W,L>>*>*
   (source vertex; destination vertex)

   sparse
   dynamic

3. Adjacency Matrix Graph:

   Dictionary<V, int>



   dense
   static