```
// Class that represents a directed graph.
class Graph {
  // Representation of the graph:
  // * The graph is represented by 3 ordered dictionaries (std::map),
  //   one that stores the inbound edges for each vertex, one that stores
  //   the outbound edges for each vertex, and one that stores the cost
  //   associated to each edge.
  // * The existence of a vertex can be checked by simply checking if it
  //   exists as a key in either the `_inbound` or `_outbound` dictionaries.
  // * The existence of an edge can be checked by simply checking if it exists
  //   as a key in the `_cost` dictionary.
  // * The Graph class contains a default constructor, copy constructor and
  //   move constructor, so copying the data from one graph to another is as
  //   easy as writing `Graph g2 = g;`.
  std::map<int, std::vector<int>> _inbound, _outbound;
  std::map<std::pair<int, int>, int> _cost;

public:
  // Creates a graph with `vertexCount` vertices numbered from 0
  // to `vertexCount` - 1, and `edgeCount` randomly generated edges.
  // Throws: std::invalid_argument if `edgeCount` > `vertexCount` ^ 2.
  static Graph randomGraph(int vertexCount, int edgeCount);

  // Reads a graph from the given stream, which is expected
  // to be in the "old" format (that assumes the graph contains all
  // vertices from 0 to n - 1).
  // Throws: std::invalid_argument if edge count > vertex count ^ 2.
  static Graph fromStreamOld(std::istream& is);

  // Returns the number of vertices in the graph.
  int vertexCount() const;

  // Checks whether the graph contains the specified vertex.
  // Returns: `true` if the vertex exists in the graph, otherwise `false`.
  bool isVertex(int vertex) const;

  // Returns a vector containing all vertices in the graph, in ascending order.
  std::vector<int> vertices() const;

  // Checks whether there exists an edge between the 2 vertices.
  // Returns: `true` if there exists an edge between the 2 vertices,
  // otherwise `false`.
  // Throws: std::out_of_range if either of the vertices is not in the graph.
  bool existsEdge(int vertex1, int vertex2) const;

  // Returns the in degree of the specified vertex.
```

```
// Throws: std::out_of_range if the specified vertex is not in the graph.
int inDegree(int vertex) const;

// Returns the out degree of the specified vertex.
// Throws: std::out_of_range if the specified vertex is not in the graph.
int outDegree(int vertex) const;

// Returns a vector containing all the inbound edges of the specified vertex.
// Throws: std::out_of_range if the specified vertex is not in the graph.
// Law: forall v. forall v2 in inbound(v). existsEdge(v2, v)
const std::vector<int>& inbound(int vertex) const;

// Returns a vector containing all the outbound edges of the specified vertex.
// Throws: std::out_of_range if the specified vertex is not in the graph.
// Law: forall v. forall v2 in outbound(v). existsEdge(v, v2)
const std::vector<int>& outbound(int vertex) const;

// Returns the cost associated to the edge between `vertex1` and `vertex2`.
// Throws:
//   * std::out_of_range if either vertex is not in the graph.
//   * std::invalid_argument if there is no edge between vertex1 and vertex2.
int getCost(int vertex1, int vertex2) const;

// Sets the cost associated to the edge between `vertex1` and `vertex2` to
// be equal to `cost`.
// Throws:
//   * std::out_of_range if either vertex is not in the graph.
//   * std::invalid_argument if there is no edge between vertex1 and vertex2.
void setCost(int vertex1, int vertex2, int cost);

// Adds a new edge between `vertex1` and `vertex2`, with the cost equal
// to `cost`.
// Throws:
//   * std::out_of_range if either vertex is not in the graph.
//   * std::invalid_argument if there already exists an edge between
//     vertex1 and vertex2.
void addEdge(int vertex1, int vertex2, int cost);

// Removes the edge between `vertex1` and `vertex2`.
// Throws:
//   * std::out_of_range if either vertex is not in the graph.
//   * std::invalid_argument if there is no edge between vertex1 and vertex2.
void removeEdge(int vertex1, int vertex2);

// Adds the specified vertex to the graph.
// Throws: std::invalid_argument if the vertex already exists.
```

```
  void addVertex(int vertex);

  // Removes the specified vertex from the graph.
  // Throws: std::invalid_argument if the vertex is not in the graph.
  void removeVertex(int vertex);
};

// Reads a graph from the given stream.
inline std::istream& operator >>(std::istream& is, Graph& g);

// Writes the graph to the stream.
inline std::ostream& operator <<(std::ostream& os, const Graph& g);
```