```java
import java.util.LinkedList;

/**
 * DO NOT CHANGE THIS FILE.
 *
 * A basic Graph interface
 */
public interface Graph<T> {
  /** Returns true if this graph is empty, false otherwise. */
  public boolean isEmpty();

  /** Returns the number of vertices in this graph. */
  public int n();

  /** Returns the number of arcs in this graph. */
  public int m();

  /** Returns true iff a directed edge exists from v1 to v2. */
  public boolean isArc (T vertex1, T vertex2);

  /** Returns true iff an edge exists between two given vertices
    * which means that two corresponding arcs exist in the graph */
  public boolean isEdge (T vertex1, T vertex2);

  /** Returns true IFF the graph is undirected, that is, for every
    * pair of nodes i,j for which there is an arc, the opposite arc
    * is also present in the graph.  */
  public boolean isUndirected();

  /** Adds a vertex to this graph, associating object with vertex.
    * If the vertex already exists, nothing is inserted. */
  public void addVertex (T vertex);

  /** Removes a single vertex with the given value from this graph.
    * If the vertex does not exist, it does not change the graph. */
  public void removeVertex (T vertex);

  /** Inserts an arc from vertex1 to vertex2.
    If the vertices exist. Else it does not change the graph. */
  public void addArc (T vertex1, T vertex2, int weight);

  /** Removes an arc from vertex v1 to vertex v2,
    * if the vertices exist. Else it does not change the graph. */
  public void removeArc (T vertex1, T vertex2);

  /** Inserts an edge between two vertices of this graph,
    * if the vertices exist. Else does not change the graph. */
  public void addEdge (T vertex1, T vertex2, int weight);

  /** Removes an edge between two vertices of this graph,
    if the vertices exist, else does not change the graph. */
  public void removeEdge (T vertex1, T vertex2);

  /** Retrieve from a graph the vertices x following vertex v (v->x)
    and returns them onto a linked list */
  public LinkedList<T> getSuccessors(T vertex);

  /** Retrieve from a graph the vertices x pointing to vertex v (x->v)
    and returns them onto a linked list */
  public LinkedList<T> getPredecessors(T vertex);

  /** Returns a string representation of the adjacency matrix. */
  public String toString();

  /** Saves the current graph into a .tgf file.
    If it cannot save the file, a message is printed. */
  public void saveTGF(String tgf_file_name);

}
```