A. Building graphs

time limit per test: 10.0 s
memory limit per test: 512 MB
input: standard input
output: standard output

Define graph ADT and its implementation based on adjacency matrix. In particular, you need to implement the following:

1. `Graph` ADT (as a Java interface or an abstract class in C++), supporting the following methods:
   a. `addVertex(value)` — add a vertex with value `value` to the graph and return reference to the created vertex object;
   b. `removeVertex(v)` — remove a vertex, provided a reference to a vertex object;
   c. `addEdge(from, to, weight)` — add a (directed) edge between `from` and `to` vertices with weight `weight`, return reference to the created edge object;
   d. `removeEdge(e)` — remove an edge, given a reference to an edge object;
   e. `edgesFrom(v)` — return a collection or edge objects that are going from vertex `v`;
   f. `edgesTo(v)` — return a collection or edge objects that are going into vertex `v`;
   g. `findVertex(value)` — find any vertex with the specified value;
   h. `findEdge(from_value, to_value)` — find any edge with specified values in the source and target vertices.
   i. `hasEdge(v, u)` — determine whether there exists a directed edge from `v` to `u`;
2. `Vertex` class of vertex objects;
3. `Edge` class of edge objects;
4. `AdjacencyMatrixGraph` class implementing `Graph` ADT using adjacency matrix;
5. All of the above should be generic in the type of values stored at vertices and weights of edges.

After implementing all of the above, write a program that inputs instructions from the standard input to create, modify and query a graph. Here is a list of instructions:

1. `ADD_VERTEX <name>` — add a vertex with a given name (a name may consist of alphanumeric symbols (such as `Example1` or `test123`));
2. `REMOVE_VERTEX <name>` — remove a vertex with a given name;
3. `ADD_EDGE <from_name> <to_name> <weight>` — add an edge from `from_name` to `to_name` with an integer weight `weight`;
4. `REMOVE_EDGE <from_name> <to_name>` — remove an edge from `from_name` to `to_name`;
5. `HAS_EDGE <from_name> <to_name>` — output `TRUE` if there is an edge from `from_name` to `to_name` and `FALSE` otherwise;

Examples

input Copy

```
ADD_VERTEX CF
ADD_VERTEX DE
HAS_EDGE CF DE
ADD_EDGE CF DE -18
HAS_EDGE CF DE
```

output Copy

```
FALSE
TRUE
```