12.1 `AdjacencyMatrix`: Representing a Graph by a Matrix

An adjacency matrix is a way of representing an `n` vertex graph $G = (V, E)$ by an `n`×`n` matrix, `a`, whose entries are boolean values.

```
int n;
boolean[][] a;
AdjacencyMatrix(int n0) {
    n = n0;
    a = new boolean[n][n];
}
```

The matrix entry `a[i][j]` is defined as

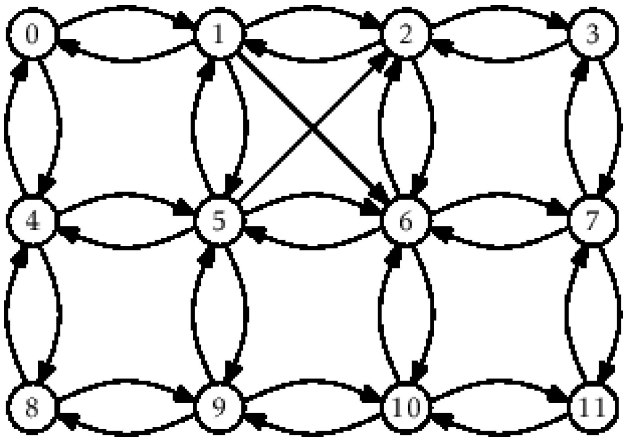
$$\texttt{a[i][j]} = \begin{cases} \texttt{true} & \text{if } (\texttt{i}, \texttt{j}) \in E \\ \texttt{false} & \text{otherwise} \end{cases}$$

The adjacency matrix for the graph in Figure 12.1 is shown in Figure 12.2.

In this representation, the operations `addEdge(i,j)`, `removeEdge(i,j)`, and `hasEdge(i,j)` just involve setting or reading the matrix entry `a[i][j]`:

```
void addEdge(int i, int j) {
    a[i][j] = true;
}
void removeEdge(int i, int j) {
    a[i][j] = false;
}
boolean hasEdge(int i, int j) {
    return a[i][j];
}
```

These operations clearly take constant time per operation.



	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	0	0	1	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1	0	0	0	0	0
2	1	0	0	1	0	0	1	0	0	0	0	0
3	0	0	1	0	0	0	0	1	0	0	0	0
4	1	0	0	0	0	1	0	0	1	0	0	0
5	0	1	1	0	1	0	1	0	0	1	0	0
6	0	0	1	0	0	1	0	1	0	0	1	0
7	0	0	0	1	0	0	1	0	0	0	0	1
8	0	0	0	0	1	0	0	0	0	1	0	0
9	0	0	0	0	0	1	0	0	1	0	1	0
10	0	0	0	0	0	0	1	0	0	1	0	1
11	0	0	0	0	0	0	0	1	0	0	1	0

Figure 12.2: A graph and its adjacency matrix.

Where the adjacency matrix performs poorly is with the `outEdges(i)` and `inEdges(i)` operations. To implement these, we must scan all `n` entries in the corresponding row or column of `a` and gather up all the indices, `j`, where `a[i][j]`, respectively `a[j][i]`, is true.

```
    List<Integer> outEdges(int i) {
        List<Integer> edges = new ArrayList<Integer>();
        for (int j = 0; j < n; j++)
            if (a[i][j]) edges.add(j);
        return edges;
    }
    List<Integer> inEdges(int i) {
        List<Integer> edges = new ArrayList<Integer>();
        for (int j = 0; j < n; j++)
            if (a[j][i]) edges.add(j);
```

```
        return edges;
    }
```

These operations clearly take $O(n)$ time per operation.

Another drawback of the adjacency matrix representation is that it is large. It stores an $n \times n$ boolean matrix, so it requires at least n^2 bits of memory. The implementation here uses a matrix of `boolean` values so it actually uses on the order of n^2 bytes of memory. A more careful implementation, which packs w boolean values into each word of memory, could reduce this space usage to $O(n^2/w)$ words of memory.

Theorem 12..1 *The `AdjacencyMatrix` data structure implements the `Graph` interface. An `AdjacencyMatrix` supports the operations*

- `addEdge(i,j)`, `removeEdge(i,j)`, and `hasEdge(i,j)` in constant time per operation; and
- `inEdges(i)`, and `outEdges(i)` in $O(n)$ time per operation.

The space used by an `AdjacencyMatrix` is $O(n^2)$.

Despite its high memory requirements and poor performance of the `inEdges(i)` and `outEdges(i)` operations, an `AdjacencyMatrix` can still be useful for some applications. In particular, when the graph G is dense, i.e., it has close to n^2 edges, then a memory usage of n^2 may be acceptable.

The `AdjacencyMatrix` data structure is also commonly used because algebraic operations on the matrix `a` can be used to efficiently compute properties of the graph G. This is a topic for a course on algorithms, but we point out one such property here: If we treat the entries of `a` as integers (1 for `true` and 0 for `false`) and multiply `a` by itself using matrix multiplication then we get the matrix a^2. Recall, from the definition of matrix multiplication, that

$$a^2[i][j] = \sum_{k=0}^{n-1} a[i][k] \cdot a[k][j] \ .$$

Interpreting this sum in terms of the graph G, this formula counts the number of vertices, `k`, such that G contains both edges `(i,k)` and `(k,j)`. That is, it counts the number of paths from `i` to `j` (through intermediate vertices, `k`) whose length is exactly two. This observation is the foundation of an algorithm that computes the shortest paths between all pairs of vertices in G using only $O(\log n)$ matrix multiplications.
