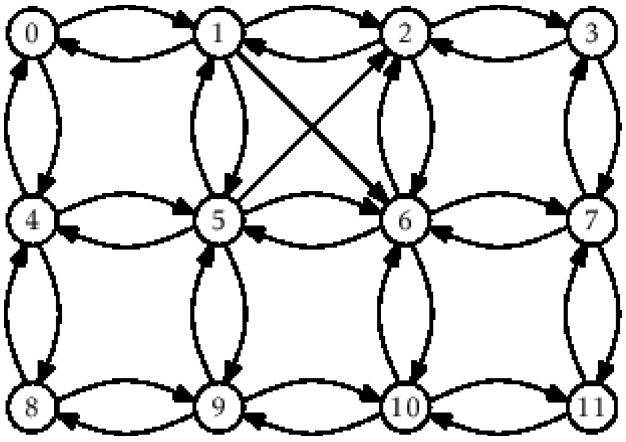
12.2 `AdjacencyLists`: A Graph as a Collection of Lists

Adjacency list representations of graphs take a more vertex-centric approach. There are many possible implementations of adjacency lists. In this section, we present a simple one. At the end of the section, we discuss different possibilities. In an adjacency list representation, the graph $G = (V, E)$ is represented as an array, `adj`, of lists. The list `adj[i]` contains a list of all the vertices adjacent to vertex `i`. That is, it contains every index `j` such that $(i, j) \in E$.

```
int n;
List<Integer>[] adj;
AdjacencyLists(int n0) {
    n = n0;
    adj = (List<Integer>[])new List[n];
    for (int i = 0; i < n; i++)
        adj[i] = new ArrayStack<Integer>(Integer.class);
}
```

(An example is shown in Figure 12.3.) In this particular implementation, we represent each list in `adj` as an `ArrayStack`, because we would like constant time access by position. Other options are also possible. Specifically, we could have implemented `adj` as a `DLList`.



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

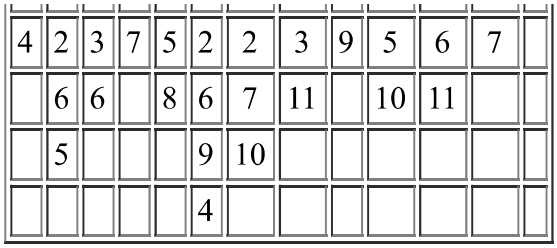


Figure 12.3: A graph and its adjacency lists

The `addEdge(i, j)` operation just appends the value `j` to the list `adj[i]`:

```
void addEdge(int i, int j) {
    adj[i].add(j);
}
```

This takes constant time.

The `removeEdge(i, j)` operation searches through the list `adj[i]` until it finds `j` and then removes it:

```
void removeEdge(int i, int j) {
    Iterator<Integer> it = adj[i].iterator();
    while (it.hasNext()) {
        if (it.next() == j) {
            it.remove();
            return;
        }
    }
}
```

This takes $O(\deg(i))$ time, where $\deg(i)$ (the degree of `i`) counts the number of edges in E that have `i` as their source.

The `hasEdge(i, j)` operation is similar; it searches through the list `adj[i]` until it finds `j` (and returns true), or reaches the end of the list (and returns false):

```
boolean hasEdge(int i, int j) {
    return adj[i].contains(j);
}
```

This also takes $O(\deg(i))$ time.

The `outEdges(i)` operation is very simple; it returns the list `adj[i]` :

```
List<Integer> outEdges(int i) {
    return adj[i];
}
```