Sergio E. Garcia Tapia
Algorithms by Sedgewick and Wayne (4th edition) [SW11]
January 13, 2025

4.2: Directed Graphs

Exercise 1. What is the maximum number of edges in a digraph with V vertices and no parallel edges? What is the minimum number of edges in a digraph with V vertices, none of which are isolated?

Solution. Suppose parallel edges are not allowed, but self-loops are. If there are V vertices, and v_i is a given vertex, then there are $|V|$ possible edges incident from v_i, including the self-loop `v`$_i$`->v`$_i$. If we do this for $i = 1, \ldots, |V|$, then we see that there are $|V|^2$ possible edges.

Now suppose that parallel edges and self-loops are allowed, but we care for the case where no vertices are isolated. This means that the outdegree of the vertex cannot be 0 (what about indegree)? If there are $v_1, \ldots, v_n$ vertices, where $n = |V|$, then we can arrange for exactly one edge to leave each vertex, namely v$_i$-¿v$_{i+1}$, for $1 \leq i < n$. Then, we add one more edge v$_n$-¿v$_i$, for example. Thus we have a minimum of $|V|$ edges.

Exercise 2. Draw, in the style of the figure in the text (page 524), the adjacency lists built by `Digraph`'s input stream constructor for the file `tinyDGex2.txt` (see Figure 1).

```
12
16
 8  4
 2  3
 0  5
 0  6
 3  6
10  3
 7  11
 7  8
11  8
 2  0
 6  2
 5  2
 5  10
 3  10
 8  1
 4  1
```

Solution. See Figure 2.

Exercise 3. Create a copy constructor for `Digraph` that takes as input a digraph `G` and creates and initializes a copy of the digraph. Any changes a client makes to `G` should not affect the newly created digraph.

Solution. See `com.segarciat.algs4.ch4.sec2.ex03`.

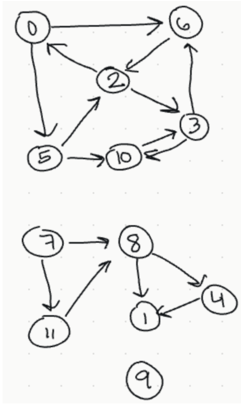

Figure 1: Digraph formed by using the input stream constructor to `Digraph` with `tinyGex2.txt`.

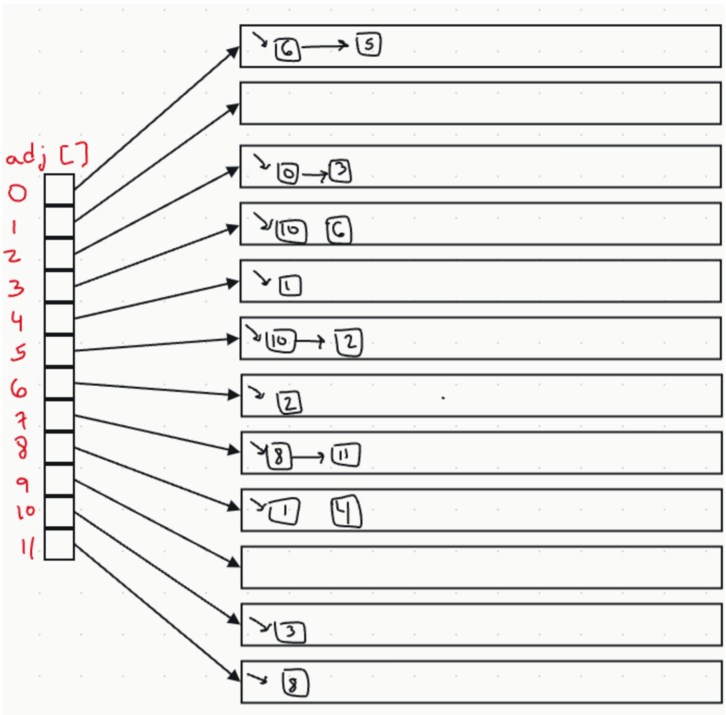

Figure 2: Adjacency list for `Digraph` built from `tinyGex2.txt`.

Exercise 4. Add a method `hasEdge()` to `Digraph` which takes two `int` arguments `v` and `w` and returns `true` if the graph has an edge `v->w`, `false` otherwise.

Solution. See `com.segarciat.algs4.ch4.sec2.ex04`.

Exercise 5. Modify `Digraph` to disallow parallel edges and self-loops.

Solution. See `com.segarciat.algs4.ch4.sec2.ex05`.

Exercise 6. Develop a test client for `Digraph`.

Solution. See `com.segarciat.algs4.ch4.sec2.ex06`.

Exercise 7. The *indegree* of a vertex in a digraph is the number of directed edges that point to that vertex. The *outdegree* of vertex in a digraph is the number of directed edges that emanate from that vertex. No vertex is reachable from a vertex of outdegree 0, which is called a *sink*; a vertex of indegree 0, which is called a *source*, is not reachable from any other vertex. A digraph where self-loops are allowed *and* every vertex has outdegree 1 is called a *map* (a function from the set of integers from 0 to $V - 1$ onto itself). Write a program `Degrees.java`. that implements the following API:

```
      public class Degrees
                   int Degrees(Digraph G)  // constructor
                   int indegree(int v)     // indegree of v
                   int outdegree(int v)    // outdegree of v
  Iterable<Integer> sources()              // sources
  Iterable<Integer> sinks()                // sinks
               boolean isMap()             // is G a map?
```

Solution. See `com.segarciat.algs4.ch4.sec2.ex07`.

Exercise 9. Write a method that checks whether a given permutation of a DAG's vertices is a topological order of that DAG.

Solution. See `com.segarciat.algs4.ch4.sec2.ex09`.

Exercise 10. Given a DAG, does there exist a topological order that cannot result from applying a DFS-based algorithm, no matter in what order the vertices adjacent to each vertex are chosen? Prove your answer.

Solution. No, such a topological order does not exist. It is possible to obtain any topological order by using a DFS-based algorithm.

Proof. Let G be a DAG of V vertices, $n = V$, and σ be a topological order on G. If σ_k is the kth vertex in the order, then σ_n must be a sink. Otherwise, a vertex would follow it, and we would not have a topological order. If we apply DFS to σ_n, it would return immediately. When considering σ_i, where $i < n$, all vertices that come after σ_i have been marked. Once again, either σ_i is a sink (and DFS immediately returns), or it points to a vertex that has already been marked. In either case, the result is that the vertex is done being processed. We continue this way until reaching $i = 1$, at which point our DFS-based algorithm ends. Along the way, vertices were done in reverse order of σ, and hence the algorithm computes the topological order to be the reverse order of the reverse order of σ, which of course is σ itself. $\square$

Exercise 11. Describe a family of sparse digraphs whose number of directed cycles grows exponentially in the number of vertices.

Solution. Digraphs with a center vertex. For example, consider a graph G of $n+1$ vertices, where for each `k` there is an edge `k->0` and an edge `0->k`, where $1 \leq k \leq n$. Also, for k and $k+1$, there is a pair of edges `k->(k+1)` and `(k+1)->k`, for $k > 1$, and for $k = n$, we have `k->1` and `1->k`. Then 0 is the center vertex of such a graph. Such a graph is sparse because there are $6 \cdot n$ edges when there are $n+1$ vertices.

Such a graph is strongly connected. Given subset of the vertices that contains 0, we can create a directed cycle containing 0. If S_0 is the set of vertices without 0, and $\mathcal{P}(S_0)$ is the power set of S_0, $|\mathcal{P}(S_0)| = 2^n$. By appending 0 to each set, we have at least one unique cycle for each set in $\mathcal{P}(S_0)$, meaning at least 2^n cycles.

Exercise 12. Prove that the strong components in G^R are the same as in G.

Solution.

Proof. Suppose that C is a strong component of G, and let $u, v \in C$. Then there is a path p_{uv} from u to v and there is a path p_{vu} from v to u. In G^R, edges change direction, so the edges in the path p_{uv} and reverse to become path p_{uv}^R, which is now a path from v to u. Similarly, p_{vu}^R is a path from u to v. Hence, u and v belong to the same strong component in G^R. Thus, if C^R is the strong component in G^R that u and v belong to, we see that $C \subseteq C^R$.

Now suppose that $w \notin C$, but w belongs to the same component as u and v in G^R (that is, $u, v, w \in C^R$). Then, without loss of generality, there is a path p_{vw}^R from v to w and a path p_{wv}^R from w to v in G^R. If we reverse the edges of G^R to obtain $(G^R)^R = G$, then the edges in both paths are reversed, and we obtain a path $(p_{vw}^R)^R$, from w to v and a path $(p_{wv}^R)^R$ from v to w in G. This implies that w and v are in the same strong component, which contradicts the definition of w. Hence, $w \notin C^R$.

We've just argued that if $w \notin C$, then $w \notin C^R$, which implies that $C^R \subseteq C$. We conclude $C = C^R$. $\square$

Exercise 13. Prove that two vertices in a digraph G are in the same strong component if and only if there is a directed cycle (not necessarily simple) containing them both.

Solution.

Proof. Let $v, w \in G$.

Suppose that v and w belong to the same strong component. Then w is reachable from v through a directed path p_{vw} and v is reachable from w through a directed path p_{wv}. By concatenating the paths, we create a cycle that contains both v and w.

Now suppose that there is a cycle $u_1, u_2, \ldots, u_n, u_1$ containing v an w. Suppose $v = u_i$ and $w = u_j$, where $j > i$. Then there is a path $v_i v_{i+1} \cdots v_{j-1} v_j$ from v_i to v_j and a path $v_j v_{j+1} \cdots v_n v_1 \cdot v_i$ from v_j to v_i. Hence, v and w are strongly connected. $\square$

References

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. 4th ed. Addison-Wesley, 2011. ISBN: 9780321573513.