Sergio E. Garcia Tapia
Algorithms by Sedgewick and Wayne (4th edition) [SW11]
January 09, 2025

4.1: Undirected Graphs

Exercise 1. What is the maximum number of edges in a graph with V vertices and no parallel edges? What is the minimum number of edges in a graph with V vertices, none of which are isolated (have degree 0)?

Solution. No parallel edges means that at most one edge connects any two given nodes. For any vertex v_i, there are V possible edge candidates, including v_0 itself (because loops are not disallowed, we can assume they are allowed). Then, for v_1, there are $V-1$ edges allowed: one for v_1, and one for each other vertex, except v_0. Continuing this way, we find that there is a maximum of $V!$ (V factorial) edges.

If V is even, then the minimum is $V/2$, since we can pair all vertices. If V is odd, it is $\lfloor V/2 \rfloor + 1$.

Exercise 2. Draw, in the style of the figure in the text (page 524), the adjacency lists built by `Graph`'s input stream constructor for the file `tinyGex2.txt` depicted at left (input from `tinyGex2.txt`).

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

Solution. See Figure 1.

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

Solution. See `com.segarciat.algs4.ch4.sec1.ex03`.

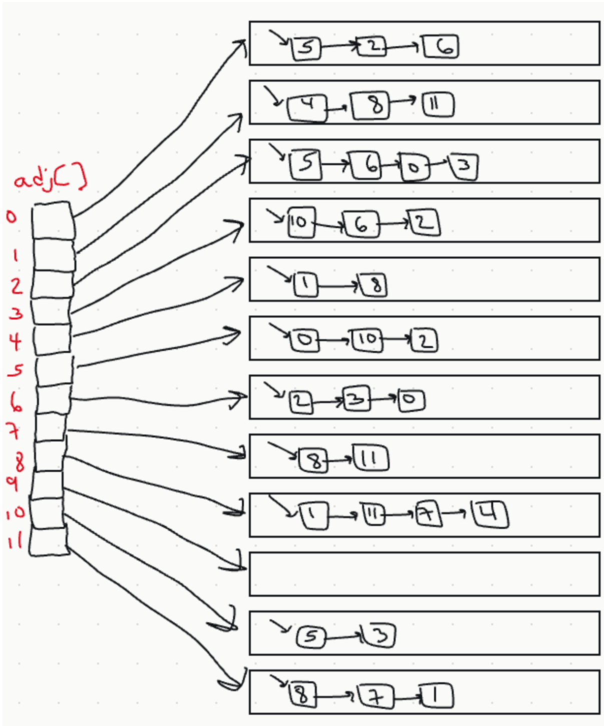

Figure 1: Adjacency list representation for undirected graph from `tinyGex2.txt`

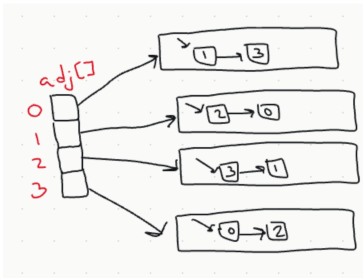

Figure 2: Impossible adjacency-lists for a four-vertex graph with edges `0-1`, `1-2`, `2-3`, and `3-0`.

Exercise 4. Add a method `hasEdge()` to `Graph` 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.sec1.ex04`.

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

Solution. See `com.segarciat.algs4.ch4.sec1.ex05`.

Exercise 6. Consider the four-vertex graph with edges `0-1`, `1-2`, `2-3`, and `3-0`. Draw an array of adjacency-lists that could *not* have been built calling `addEdge()` for these edges *no matter what order*.

Solution. See Figure 2. The contents suggest that

1. According to 0's adjacency list, `0-3` comes before `0-1`.
2. According to 3's adjacency list, `2-3` comes before `0-3`.

3. According to 2's adjacency list, `1-2` comes before `2-3`.
4. According to 1's adjacency list, `0-1` comes before `1-2`.

According to the first three, the implied order is

```
1-2
2-3
0-3
0-1
```

but then 1's adjacency list says that `0-1` comes before `1-2`, which contradicts that `0-1` comes last in the list above. This can be seen from the adjacency lists because there must be first pair, which means that there is a pair of vertices `v` and `w` that are last in each other's adjacency lists. That would imply that `v-w` (or `w-v`) was the first edge inserted. That doesn't happen in the figure, however.

Exercise 7. Develop a test client for `Graph` that reads a graph from the input stream named as command-line argument and then prints it, relying on `toString()`.

References

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