Graphs

A **graph** contains **nodes** (also called **vertices**) and **edges**. An edge connects two nodes. We can use graphs to represent relationships: there is a relationship between node A and node B if there is an edge between A and B.

In an **undirected graph**, relationships are mutual.

In a **directed graph**, relationships are one-way, from the **source** to the **destination**.

In a **weighted graph**, relationships have associated information (e.g., cost or “weight”).

An edge is **adjacent** to A if that edge emanates from A.

Node B is **adjacent** to node A if there is an edge from A to B.

The **neighbors** of A are all the nodes adjacent to A.

A **path** is a sequence of edges between adjacent nodes.

Node B is **reachable** from node A if there is a path from A to B.

In a **complete graph**, there exists an edge between each pair of nodes.

In a **connected graph**, there exists a path between each pair of nodes.

A **sparse** graph has few edges, relative to the maximum amount it can have.

A **dense** graph has many edges, relative to the maximum amount it can have.

In an **acyclic** graph, there are a finite number of paths between any two nodes.

In a **cyclic** graph, there may be an infinite number of paths between two nodes.

### Designing and implementing a new data structure

The **interface** describes *what* a data structure can do (e.g., its operations). The interface is a promise from the provider of the data structure to the user of the data structure.

The **implementation** describes *how* the data structure works (e.g., how the data are stored / organized and which algorithms are used to provide the operations). The implementation makes good on the promise of the interface.

It should be possible to replace the implementation without modifying the interface.

### Representing graphs

**Edge list**: store a list of edges (where an edge is stored as a triple of source, destination, weight).

**Adjacency list**: for each source node, store a list of adjacencies (where an adjacency is stored as a pair of destination, weight).

**Adjacency matrix**: a table that, for each pair of nodes, stores the cost of an edge between those nodes.
Racket reference (for when you’re working on the exercises)

Common syntax

(op arg₁ arg₂ ... argₙ)
(let* ([var₁ expr₁]
    ...
    [varₙ exprₙ])
        body-expr)
(if conditional-expr
    true-expr
    false-expr)
(cond [condition₁ expr₁]
    ...
    [conditionₙ exprₙ]
    [else else-expr])
(define (function-name param₁ ... paramₙ)
    body-expr)

Common functions

(equal? expr expr)
(modoulo expr expr)
(quotient expr expr)
(or expr expr)
(and expr expr)
(not expr)

List functions

(empty? list)
(list? list)
(cons elem list)
(list elem elem ...)
(first list)
(rest list)
(append list list)
(remove-duplicates list)
(member elem list)

Higher-order functions

(map function list)
(filter function list)
(foldl op seed list)
(foldr op seed list)

Next time: putting the “Science” in “Computer Science”