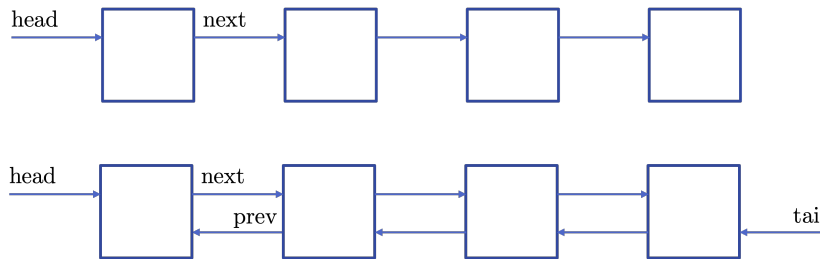


Abstract Data Types and Depth-First Search

Linked Lists

Consider a list $L = [x_1, x_2, \dots, x_n]$ where each x_i is an element in the list. We keep a pointer to the head (and the tail) of the list. Each element x_i has a pointer “next” (and “previous”).

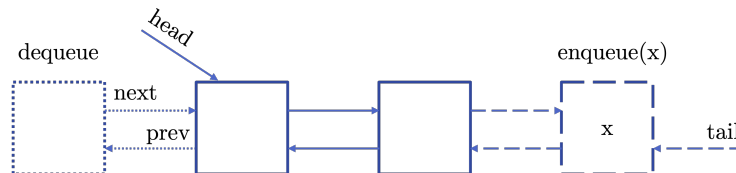


- What is the (worst-case) runtime to find an element?
- What is the (worst-case) runtime to insert or delete an element (once it's found)?

Queues

Queues are First-In, First-Out (FIFO) linked lists. They support the operations:

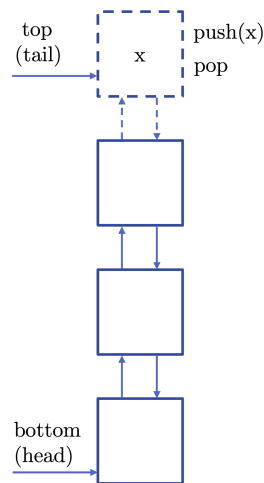
- $\text{enqueue}(q, x)$: insert element x to the back of the queue q . Formally, $q = q \circ x$.
- $\text{dequeue}(q)$: delete the element at the front of the queue q and return it. Formally, $q = [x_2, \dots, x_n]$, return x_1 .



Stacks

Stacks are what's known as Last-In, First-Out (LIFO) linked lists. They support the operations:

- $\text{push}(s, x)$: insert element x to the top (back) of the stack s . Formally, $s = s \circ x$.
- $\text{pop}(s)$: delete the element at the top (back) of the stack s and return it. Formally, $s = [x_1, \dots, x_{n-1}]$, return x_n .



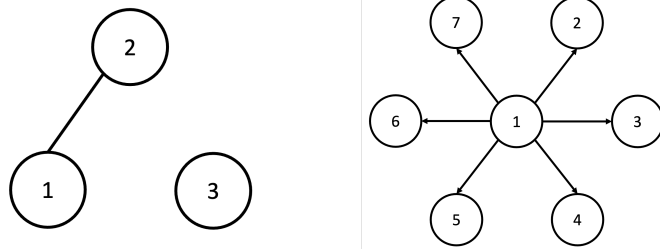
Graphs

Definition 1. A (directed) *graph* $G = (V, E)$ is defined by a set of vertices V and a set of (ordered) edges $E \subseteq V \times V$.

Definition 2. A *directed edge* is an ordered pair of vertices (u, v) and is usually indicated by drawing a line between u and v , with an arrow pointing towards v .

Definition 3. An *undirected edge* is an unordered pair of vertices $\{u, v\}$ and is usually indicated by drawing a line between u and v . It indicates the existence of ordered edges (u, v) and (v, u) .

Typically undirected edges will also be notated (u, v) out of sloppiness.

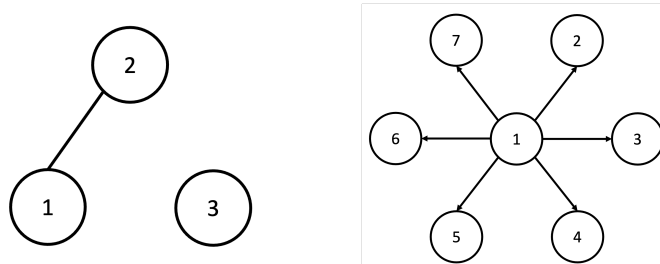


Some conventions:

- We will refer to the number of vertices (or the *size* of the vertex set $|V|$) as n .
- We will refer to the number of edges (or the *size* of the edge set $|E|$) as m .
- Often we will simply name the vertices $V = \{1, \dots, n\}$ so an edge (i, j) is an edge from the i^{th} vertex to the j^{th} vertex.
- You may also hear vertices referred to as “nodes” or edges referred to as “arcs.”

Definition 4. We call vertices i and j *adjacent* or *neighbors* if there is an edge $(i, j) \in E$. In directed graphs, we may explicitly refer to *out-neighbors* ($\{j : (i, j) \in E\}$) or *in-neighbors* ($\{j : (j, i) \in E\}$).

Definition 5. The *degree* of a vertex v is the number of neighbors it has. That is, $d_v = |\{u : (v, u) \in E\}|$. For directed graphs, we may refer to a vertex’s *in-degree* or *out-degree*, and its *degree* is the sum of these.



Definition 6. A *path* from u to w is a sequence of edges e_1, e_2, \dots, e_k such that $e_1 = (u, v_1)$, $e_i = (v_{i-1}, v_{i+1})$, and $e_k = (v_{k-1}, w)$. That is, the first edge starts at u , the last edge ends at w , and each preceding edge ends where the previous edge starts.

Definition 7. We say that a pair of vertices are *connected* if there exists a path between them.

We see graphs all over; networks are an entire field of study! What can you represent with graphs?

-
-
-
-
-

What graph problems do you know?

-
-
-

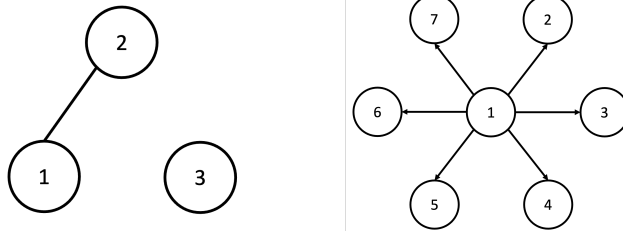
Abstract Data Types for Graphs

There are two primary ways that we represent graphs in the computer.

Definition 8. An *adjacency matrix* for $G = (V, E)$ is an $n \times n$ binary matrix A where $A_{ij} = 1$ if and only if $(i, j) \in E$.

Pros of using an adjacency matrix:

Cons of using an adjacency matrix:



Definition 9. An *adjacency list* for $G = (V, E)$ is an array A of length n where the i^{th} entry contains a linked list of i 's neighbors. That is, j is in the list $A[i]$ if and only if $(i, j) \in E$.

Pros of using an adjacency list:

Cons of using an adjacency list:

Exercise: Ask yourself the following questions for both adjacency matrices and adjacency lists to fill out the pros and cons (above) for each graph ADT above:

- What is the worst-case runtime to look up a specific edge (i, j) ?
- What is the worst-case space needed to store the graph?
- What is the runtime to list all edges adjacent to i ? On average, per edge adjacent to i ?