CMPUT 275: Winter 2018

Some Definitions and Concepts Regarding Sets and Graphs

1 Set Notation

Here is some common mathematical notation regarding sets. In the following, uppercase letters like A, B refer to sets and lowercase letters like x, y refer to items.

- $|A|$ is the size of the set A, the number of items it contains.
  Example: $|\{x, y\}| = 2$ and $|\{x, y, z\}| = 3$.
  In mathematics, this may be infinite (i.e. the set of all integers is infinite). However, in Python all `set()` instances have finite size.
- $\emptyset$ denotes the empty set, the set containing no items. Note $|\emptyset| = 0$.
- $x \in A$ means x is an element in A, $x \notin A$ means x is not an element in A
  Example: if $A = \{w, x, z\}$ then $x \in A$ and $y \notin A$
- We say $A \subseteq B$ if every element of A also lies in B. **Read**: "A is a subset of B".

  We also say $A \not\subseteq B$ if A is not a subset of B. This means *some* element of A is not an element of B. That is, there is some $x \in A$ with $x \notin B$. A set is always a subset of itself.
  Example: $\{x, y\} \subseteq \{w, x, y, z\}$ but $\{x, y\} \not\subseteq \{w, x, z\}$.
- $A \cap B$ denotes the set containing precisely the items appearing in both A and B.
  Example: $\{x, y, z\} \cap \{w, x, y\} = \{x, y\}$.
- $A \cup B$ denotes the set precisely the items appearing in at least one of A or B.
  Example: $\{x, y, z\} \cup \{w, x, y\} = \{w, x, y, z\}$.

2 Graph Notation

A **directed graph** is a pair $(V; E)$ where V is a set of vertices and E is a set of directed edges. An edge is an ordered pair (u, v) (so $(u, v) \neq (v, u)$ for distinct vertices u, v) where

$u \in V$ and $v \in E$. Note this definition does allow for **loops** (edges to itself) like (v, v) but cannot model instances where there may be more than one edge connecting two vertices[1].

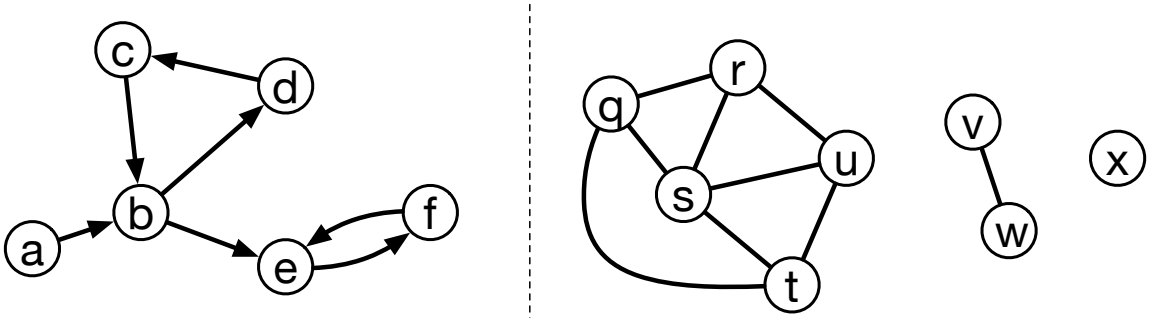
For an edge (u, v), we say vertices u and v are **endpoints** of the edge. Sometimes the edges in a directed graph are called **arcs**.

An **undirected graph** is a pair $(V; E)$ where V is a set of vertices and E is a set of *undirected* edges. The only difference with directed graphs is that the edges are unordered pairs. Sometimes an undirected edge between vertices u and v is denotes uv. So saying $uv \in E$ is equivalent to saying $vu \in E$ as both uv and vu refer to the same edge. We can *model/simulate* an undirected graph as a directed graph by including both arcs (u, v) and (v, u) in the directed graph for every undirected edge uv.

Note: Different notation, and sometimes even slightly different definitions, are present throughout the literature on graph theory. The point is to know *how* it is being applied. When you read an application or algorithm involving graphs, hopefully the author has supplied their particular definition. If not (which happens often), try thinking about what definition would make sense.

2.1 Some Definitions

Let $G = (V; E)$ denote a graph. It can be directed or undirected. The examples below refer to the following graphs.



- A **walk** is a sequence of vertices of G (perhaps with repeated vertices) $v_0, v_1, \ldots, v_k$ where $v_i v_{i+1}$ is an edge for each $0 \leq i < k$. An $s - t$ **walk** is a walk starting at s and ending at t (i.e. $s = v_0$ and $t = v_k$). The length of the walk is k: the number of

[1] We got around this issue in our graph class implementation by using Python `lists`. So our `Graph` class in Python can have multiple copies of edges between vertices. Such graphs are sometimes called **multigraphs** in literature.

edges traversed. We even allow walks of length 0 (i.e. one vertex).
Example: $[a, b, d, c, b, e, f, e, f]$ or $[q, r, q, r, s, u, s, t, q, r]$

- A **circuit** is a walk with $v_0 = v_k$. It ends where it starts.
Example: $[a]$ or $[b, d, c, b, d, c, b, d, c, b]$ or $[v, w, v]$.

- A **path** is a walk with no repeated vertices: $v_i \neq v_j$ for any $0 \leq i < j \leq k$.
Example: $[a, b, e, f]$ or $[q, r, s, u, t]$. The two walks above are not paths as they have repeated nodes.

- A **cycle** is a walk where $v_0, v_1, \ldots, v_{k-1}$ is a path and $v_0 = v_k$. **Additionally**: in directed graphs we require $k \geq 2$ and in undirected graphs we require $k \geq 3$.
Example: $[e, f, e]$ or $[q, s, t, u, r, q]$. None of the circuits above are cycles.

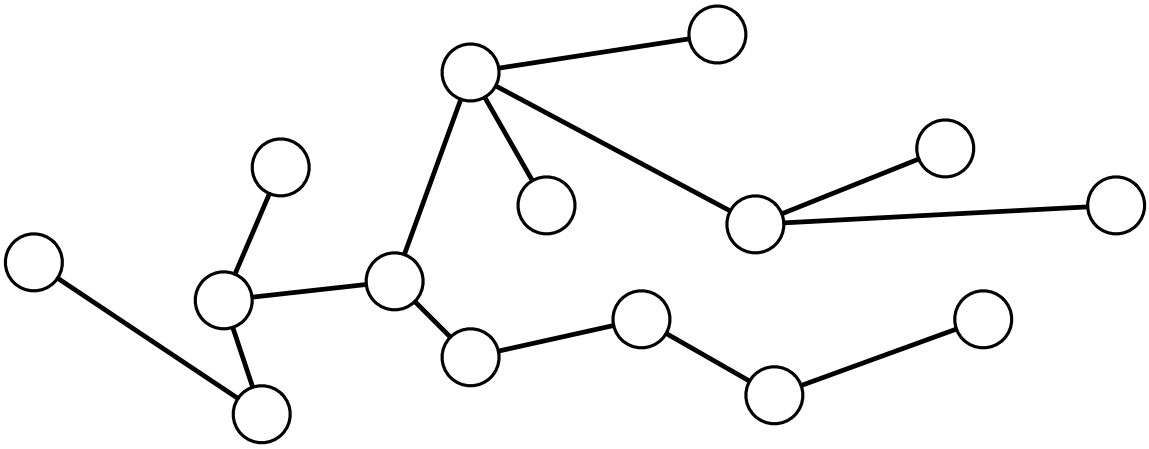
- **Undirected Graphs Only**: A **connected component** is a subset of nodes $C \subseteq V$ such that there is a path between any two $u, v \in C$, yet there is no path between any $u \in C$ and $v \notin C$.
Example: There are 3 connected components in the undirected graph. Namely, $\{q, r, s, t, u\}, \{v, w\}$ and $\{x\}$.

- **Undirected Graphs Only**: A vertex is **isolated** if it is not the endpoint of any edge.
Example: x is the only isolated vertex in the undirected graph above.

- **Undirected Graphs Only**: A graph is **connected** if V is a connected component. That is, any two vertices have a path between each other.
Example: The undirected graph above is not connected. It has 3 connected components.

- **Undirected Graphs Only**: A **tree** is a connected graph with no cycles. An example of a tree is below. Neat properties you can prove about trees:
  - $|E| = |V| - 1$.
  - There is exactly one path connecting any two vertices.
  - In every connected graph $G = (V; E)$ there is some $F \subseteq E$ such that $(V; F)$ is a tree called a **spanning tree** of G.
  - A graph that is not connected does not have a spanning tree.

  Example: The undirected graph above does not have a spanning tree because it is not connected. However, if we consider only the graph depicted with vertices $\{q, r, s, t, u\}$ then one spanning tree of this graph is $\{qr, qs, su, st\}$. There are many different spanning trees of this graph.

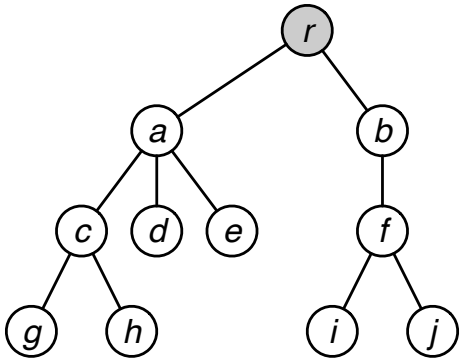


A tree (no vertex labels)

2.2 Rooted Trees

We will see rooted trees a few times: notably in heaps, the union-find data structure, and in binary search trees. They are defined in both directed and undirected settings.

An undirected graph $G = (V; E)$ is a **rooted tree** if it is a tree and one particular vertex is designated as the **root vertex**. In the picture below, the root vertex is r.



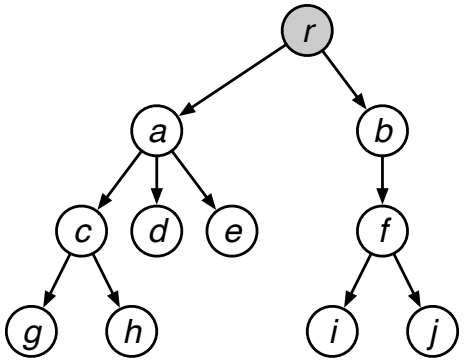
- For an edge uv, we say u is a **child** of v and that v is a **parent** of u if v lies on the path between u and the root. Intuitively, when we draw trees hanging "down" from the root (pictured above) the children of a vertex v are the vertices drawn below v and the parent is drawn above. The root is the only vertex with no parent. Every other vertex has exactly one parent.
  Example: In the picture above, c is a child of a and a is the parent of c.

- We say that u is a **descendent** of v and v is an **ancestor** of u if v lies on the path between u and the root. **Special Cases**: a child of v is a descendent of v, the parent of v is an ancestor of v. A vertex is always a descendent and an ancestor of itself. The root is an ancestor of every other vertex. The only vertex the root is a descendent of is the root itself. **Example**: The descendents of a are c, d, e, f, g, h.

- A **leaf** is a vertex with no children.
  Example: The leaves are f, g, h.

In some sense you have already seen rooted trees. The sequence of recursive calls made by `quicksort` or `mergesort` can be viewed as a rooted tree where edges indicate one recursive call was made while the other was executing. The root "vertex" in this picture was the initial recursive call.

A directed graph $G = (V; E)$ is a **rooted tree** if the undirected version of it (i.e. obtained by ignoring directions of edges) is a rooted tree. So there is still a distinguished root vertex. Furthermore, for each non-root vertex the only edge pointing in to that vertex is from its parent. See the picture below. Sometimes this is called a **rooted out-tree** as the edges point out and away from the root.



Alternatively, for each non-root vertex the only edge pointing out of that vertex is to its parent. This is a different definition, but still common. Sometimes this is called a **rooted in-tree**.

Usually the directions of the edges do not matter: they may be suggestive of some concept used in the application but in reality any definition would likely suffice as long as the presenter is clear about how it is being used.