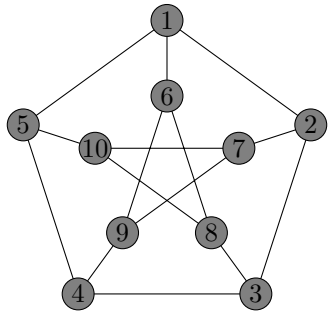

Figure 1: The Petersen graph.

Administrivia

- At class start: turn in solutions to this problem set and pick up the next problem set.
- At class end: pick up graded solutions to last problem set.
- For next class: read sections 11.1-3 in the book.

Graphs

Review

A *graph* is a pair $G = (V, E)$, where

- V is a set of *vertices* (or nodes), and
- E is a set of *edges*, with $E \subset V \times V$ if G is directed and $E \subset \{e \subset V \mid 1 \leq |e| \leq 2\}$ if G is undirected.

The Petersen graph, shown above, is a counter-example to many famous conjectures in graph theory. This evil graph, consisting of two interlocking pentegrams, has brought despair to many mathematicians.

Graphs are ubiquitous in computer science because they model so many situations that occur in practice. Here are some examples:

- transportation: nodes are cities, edges are rail lines (directed or undirected, weighted by distance)
- dependency graph: nodes are processes and edges represent one process waiting for the output of another (directed)
- conflict graph: nodes are tasks and edges between them indicate that both would require the same resource (undirected)
- person–task graph: nodes are people and tasks (bipartite) and edges represent that a person can perform a given task (undirected, weighted)
- social graph: nodes are people and edges are some relationship such as "friends" (undirected) or "likes" (directed)
- web graph: nodes are pages and edges represent hyperlinks from one page to another (directed)

A **path** from v to w is a sequence $v = v_0, v_1, \ldots, v_k = w$ such that, for $0 \leq i < k$, we have $(v_i, v_{i+1}) \in E$ if G is directed or $\{v_i, v_{i+1}\} \in E$ if G is undirected. The number k is the **length** of the path. For example, two paths from 5 to 2 in the above graph are $5, 1, 2$ and $5, 10, 7, 2$.

Finally, we will need one new definition. A **cycle** is a path of positive length from a node to itself. For example, if we connect a path from 5 to 2 with a path from 2 to 5, then we get a cycle through 5: $5, 1, 2, 7, 10, 5$.

DAGs

A **DAG** is a directed, acyclic graph. There are many practical situations which demand that the graph have no cycles.

Can the the dependency graph between processes have cycles? No. A cycle would represent a set of deadlocked processes that could never finish. Operating systems have to detect when a process attempts to create a cycle in the dependency graph and issue an error.

How about the graph of the "eats" relation between animas? Yes! For example, a person eats bird eats fish eats mosquito eats person.

Any graph G (directed or undirected) gives rise to a binary relation R_G where $v \, R_G \, w$ if there is a path from v to w. For example, since edges in a dependency graph G represent direct dependencies, paths represent direct or indirect dependencies, so we have $v \, R_G \, w$ if task w depends on v directly or indirectly.

Proposition 0.1. *If G is a DAG, then R_G is a partial order.*

Proof. We need to show the three properties of a partial order:

- R_G is reflexive, $v \, R_G \, v$, since there is always a length 0 path from v to itself.
- R_G is transitive. If $u \, R_G \, v$ and $v \, R_G \, w$, then there is a path from u to v and a path from v to w. Concatenating these two paths gives us a path from u to w; hence, $u \, R_G \, v$.

  Note that, so far, we have not used the fact that G is a DAG.
- R_G is antisymmetric. Suppose we have $v \, R_G \, w$ and $w \, R_G \, v$. By the argument above, this means we have a path from v to itself. G does not have cycles, so what do we know about this path? It has length zero, so $v = w$.

□

Trees

A **tree** is an undirected graph that (1) is connected and (2) has no simple cycles.

You have already seen trees before in the form of the binary search trees. However, like graphs, trees are ubiquitous in computer science. Here are some examples:

- directories in the file system
- organizational charts
- a game tree: the root represents the starting position of the game. There is an edge for each possible move. Those nodes represent the position reached. They have edges to each subsequent move. (In a two player game, each row of nodes would be for the same player, with adjacent rows alternating between players.)

Theorem 0.2. *Let $G = (V, E)$ be an undirected graph. Then G is a tree iff, for every $u, v \in V$, with $u \neq v$, there exists a unique simple path from u to v.*

Proof. $\Rightarrow$ There exists at least one path because G is connected. And if there were two paths, then connecting them end to end would give a cycle, which we know does not exist.

$\Leftarrow$ We will prove the contrapositive. If G is not a tree, then either it is disconnected or it has a cycle. In the former case, choosing u and v in different connected components, we see that there is no path from u to v. In the latter case, we have a simple cycle $u, v, w, \ldots, u$. Then we have two different paths from u to v: one is u, v, and the other is the reverse of $v, w, \ldots, u$. □

Interestingly, all of the examples above have a **root**. However, our definition said nothing about a root. Indeed, in a generic tree, any node can be a root: just "pick up" the tree by that node and let the others dangle below.

We can see, though, that the notion of a root is important. So we should give this a name.

A **rooted tree** is a pair (G, v) where G is a tree and $r \in V$ is the root node. Any node in the tree has a unique path to the root, by the theorem above. The adjacent node on the path from v to r is called its **parent**. The set of nodes on the path from v to r excuding v itself are called **ancestors**.

If v is the parent of w, then w is called the **child** of v. And the nodes for which v is an ancestor are called the **descendants** of v.

Theorem 0.3. *Let $G = (V, E)$ be an undirected graph. Then G is a tree iff G is connected and $|E| = |V| - 1$.*

Proof. We will prove this by induction on $n = |V|$.

Base case, $n = 1$: We have $V = \{v\}$. The only possible edge is $\{v\}$. If this is an edge, then $|E| > 0$ $(= 1-1)$ and G is not a tree since v, v is a non-simple cycle. If this is not an edge, then $|E| = 0$, and G is a tree.

Inductive case, $n > 1$: $\Rightarrow$ Suppose we pick an edge $\{v, w\} \in E$ and remove it from the graph. This gives us a graph with two connected components. Why? (If not, we have a cycle.) Let the sizes of these components be n_1 and n_2. Each component is a tree since it is connected and acyclic, so by the inductive hypothesis, the number of edges remaining is $n_1 - 1 + n_2 - 1 = n_1 + n_2 - 2 = n - 2$. Thus, the number of edges in G was $n - 1$.

$\Leftarrow$ Suppose that $|E| = n - 1$. Then, G has a leaf node. Why? If not, then summing the degrees of the nodes, we have $2|E| \geq 2n \Rightarrow |E| \geq n$. Removing this node and the single adjacent edge, gives a graph G' with $n - 1$ nodes and $n - 2$ edges. By the inductive hypothesis, this is a tree. Adding a node and a single edge to it gives a graph that is connected, and the new edge cannot create a simple cycle since this node has only one edge. □

Optimization*

An optimization problem seeks to find the $x \in S$, for some set S (the "feasible set"), that minimizes or maximizes the objective function $f(x)$.

Optimization problems are also ubiquitous in computer science (as well as mathematics and operations research). In fact, many of the first problems that computers were used to solve were optimization problems.

Perhaps surprisingly, nearly all of the well-known optimization problems are graph problems. This gives yet more evidence of how often graphs model real-life situations.

Here are some of the most important optimization problems:

- min cost flow (a.k.a. transportation or transshipment problem): find the cheapest way to get products from sources to destinations. This was analyzed during WWII to minimize shipping costs.

- max flow: in this case, edge weights are not costs but capacities. The objective is now to find the maximum amount of flow between sources and destination that does not exceed any edge capacity. This problem arose during WWII when trying to determine how many resources the enemy could deliver to the front lines.

  A famous theorem states that the max flow is equal to the min cut. A cut is the set of edges between $S \subset V$ and $T \subset V$, where $S \cup T = V$. The min cut problem also arose in WWII when trying to determine the fewest number of rail lines to bomb in order to stop resources from reaching the front.

- shortest path: find the path from one city to some other with the smallest total cost.

  A related problem is longest path. Here, the weights do not represent cost but rather goods. For example, it might be a measure of the amount of nice scenery to look at while traveling. Interestingly, while we can efficiently find shortest paths, there is no known efficient algorithm for finding longest paths, and it is widely believed that none exists.

- min cost hamiltonian cycle (a.k.a. traveling salesman problem): a cycle is Hamiltonian if it visits every vertex. This is another example of an optimization problem for which no efficient algorithm is known nor expected to exist.

- min cost spanning tree: choose a set of edges that connect all of the vertices together but with minimum total cost. From what we learned above, we know that these edges will always form a tree: if any cycle exists, we can always remove an edge without destroying connectivity.

- min cost matching (a.k.a. assignment problem): find the way of assigning each person to a task with minimum total cost.

- min coloring: find the way of assigning one of k colors to each vertex so that no connected vertices share the same color and such that k is as small as possible. This is yet another problem with no efficient algorithm.

  One situation where this problem arises is in compilers. The compiler needs to assign variables to registers in order to use them and it cannot assign two variables used at the same time to the same register.