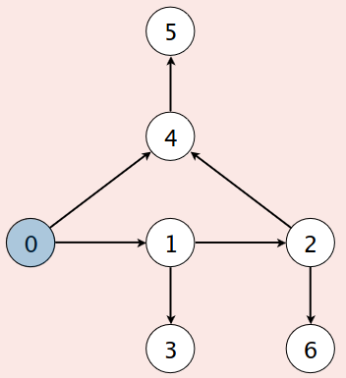
Precept Outline
- Review of Lectures 15 and 16:
  - Graphs and Digraphs
  - Graph search: DFS and BFS

Relevant Book Sections
- Book chapters: 4.1 and 4.2

A. Review: Graphs, Digraphs and Graph Search

Your preceptor will briefly review key points of this week's lectures. They may use the following graph to trace examples:



B. Rooted DAGs

Suppose G is a digraph (i.e. a directed graph). G is *acyclic* if it doesn't contain any directed cycles (i.e. a directed path whose first and last vertices are the same). Describe an algorithm that runs in $O(V + E)$-time to detect if G is acyclic.

Note. A directed acyclic graph is often known by its acronym *DAG*.

In a DAG, a vertex v is an ancestor of a vertex w if there is a directed path from w to v.

A **rooted DAG** is a DAG that has one vertex (the root) that is an ancestor of every other vertex. Describe an algorithm that runs in $O(V + E)$-time to detect if a given DAG G is rooted.

C. Tag Game on a Graph

Two players play tag on an undirected graph G. A *runner* starts at s_1 and wants to reach a target vertex t. A *tagger* starts at s_2 and wants to catch the runner. Each round, both players simultaneously move to an adjacent vertex or stay in place. The tagger catches the runner if they ever share a vertex (including t). The tagger may never move more than k steps from s_2.

The tagger is omniscient. You can assume the game is played as follows:

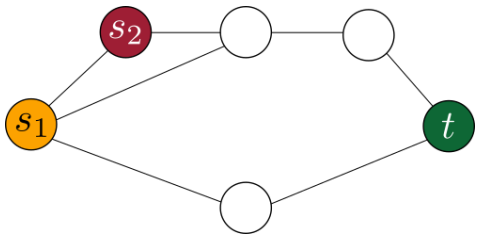
- The runner picks a walk from s_1 to t, i.e., a sequence of vertices where consecutive vertices are equal or adjacent, and the first vertex is s_1 and the last vertex is t.
- The tagger sees it and picks a walk starting at s_2 in response.
- The tagger wins if they share a vertex with the runner at any round; otherwise the runner wins.

Given G, s_1, s_2, t, and k, design an algorithm to decide whether the runner can guarantee a win.

Performance requirements. For full credit, your algorithm must take $\Theta(E + V)$ time, where V and E are the number of vertices and edges in G, respectively.

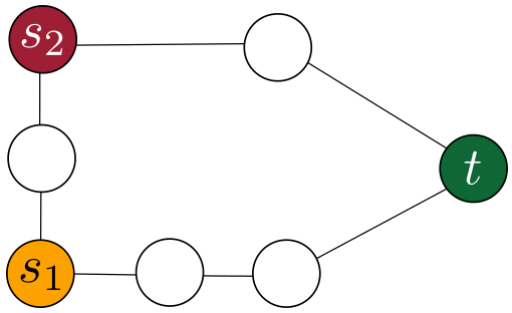
Example:

Suppose the following graph is the one the players are using, labeled accordingly.



Then, the first player can win by moving to the bottom vertex and then to t, regardless of what k is. It's easy to see that the second player can never catch the first if they follow this path.

Now, suppose the following graph is the one the players are using, labeled accordingly.



If $k \geq 2$, then the first player cannot win. If the first player decides to take the bottom path, then the second player can use the top path to go around and tag it just before they reach t. If the first player decides to take the top path, then the second player can just wait in s_2.

However, if k is 1, then the first player can win by taking the bottom path.

To help you think about this problem, let's start by considering a simpler version of it. Suppose G is a *cycle graph*. A cycle graph with V vertices has V edges and follows the pattern: vertex 1 is adjacent to V and 2, vertex 2 is adjacent to 3, ..., vertex $V-1$ is adjacent to V, and vertex V is adjacent to 1. The image above is an example of a cycle graph with 6 vertices.

Describe a solution to the tag game on a graph when the graph is a cycle graph.

Now, suppose G is a *path graph*. A path graph with V vertices has $V-1$ edges and follows the pattern: vertex 1 is adjacent to 2, vertex 2 is adjacent to 3, ..., vertex $V-1$ is adjacent to V. So it looks like a path.

Describe a solution to the tag game on a graph when the graph is a path graph.

With the intuition developed before, describe a solution to the tag game on an arbitrary graph.

D. The Water Pitcher

Part 1:

You have a single pitcher with capacity Y liters, initially empty, and an infinite water supply. You want to measure exactly X liters of water in the pitcher (where $0 \leq X \leq Y$). You are allowed to perform the following operations:

- Fill the pitcher completely to its capacity.
- Empty the pitcher completely down the drain.
- Add exactly A liters to the pitcher (only if the pitcher has room for A more liters).
- Remove exactly A liters from the pitcher (only if the pitcher contains at least A liters).
- Add exactly B liters to the pitcher (only if the pitcher has room for B more liters).
- Remove exactly B liters from the pitcher (only if the pitcher contains at least B liters).

All of X, Y, A, and B are nonnegative integers.

Design an algorithm to find the minimum number of operations required to measure exactly X liters in the pitcher. If it is impossible, return -1.

Performance requirements. Your algorithm must run in $\Theta(Y)$ time in the worst case.

Hint: model the problem as a graph problem. You can associate a vertex with each possible volume of water in the pitcher, e.g., vertex 0 corresponds to the pitcher being empty, vertex i corresponds to the pitcher containing i liters of water. Think about how to model the operations as edges between vertices. Once you do so, can you see how solving the problem reduces to a certain graph algorithm?



Part 2:

Now suppose you have *two* pitchers with integer capacities X and Y liters, both initially empty, and an infinite water supply. You want to measure exactly Z liters of water in one of the pitchers. You are allowed to perform the following operations:

- Fill a pitcher completely to its capacity.
- Empty a pitcher completely down the drain.
- Pour water from one pitcher to the other until the receiving pitcher is full or the pouring pitcher is empty.

Design an algorithm to find the minimum number of operations required to measure exactly Z liters. If it is impossible, return -1.

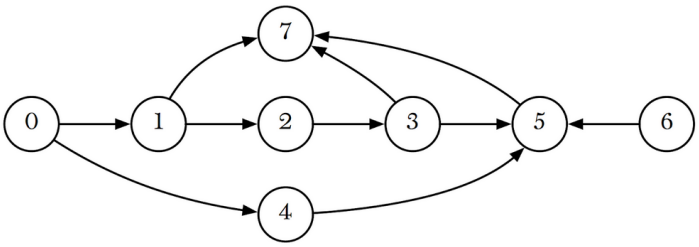
Performance requirements. Your algorithm must run in $\Theta(XY)$ time in the worst case.

Note: we are not going to go over the solution to this problem in precept. Instead, you should use it to practice solving design problems, like we encouraged you before the midterm. We recommend you take 30 minutes over the next week to sit down and think about this problem on your own. It's ok if you don't solve it, the important thing is that you try to solve it for a while. We will publish the solution next week.

E. Optional Bonus Problems

Part 1: Shortest Common Ancestor

In the following digraph, compute the (smallest) sum of the path lengths from vertices 1 and 4 to all common ancestors. Find the shortest common ancestor and the shortest ancestral path (definitions are in the previous section).



Describe an algorithm for calculating the shortest common ancestor of two vertices v and w. Your algorithm should run in linear time (i.e., proportional to $V + E$).