Graph Traversals

1 Graphs

1.1 Graphs in Problem Solving

Often, a problem can be represented as a graph. The solution to the problem is obtained by solving a problem on the corresponding graph.

1.1.1 Example: Undirected State Graph

A simple game using three coins. At the start of the game, the middle coin is "tails" and the other two are "heads."

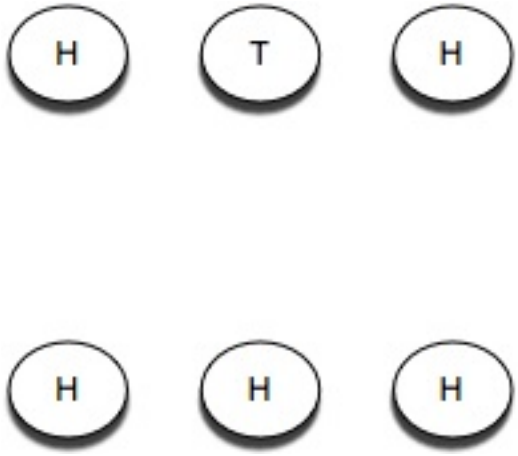
The goal of the game is to change the configuration of the coins so that the middle coin is heads and the other two are tails, like this:

Rules of the game:

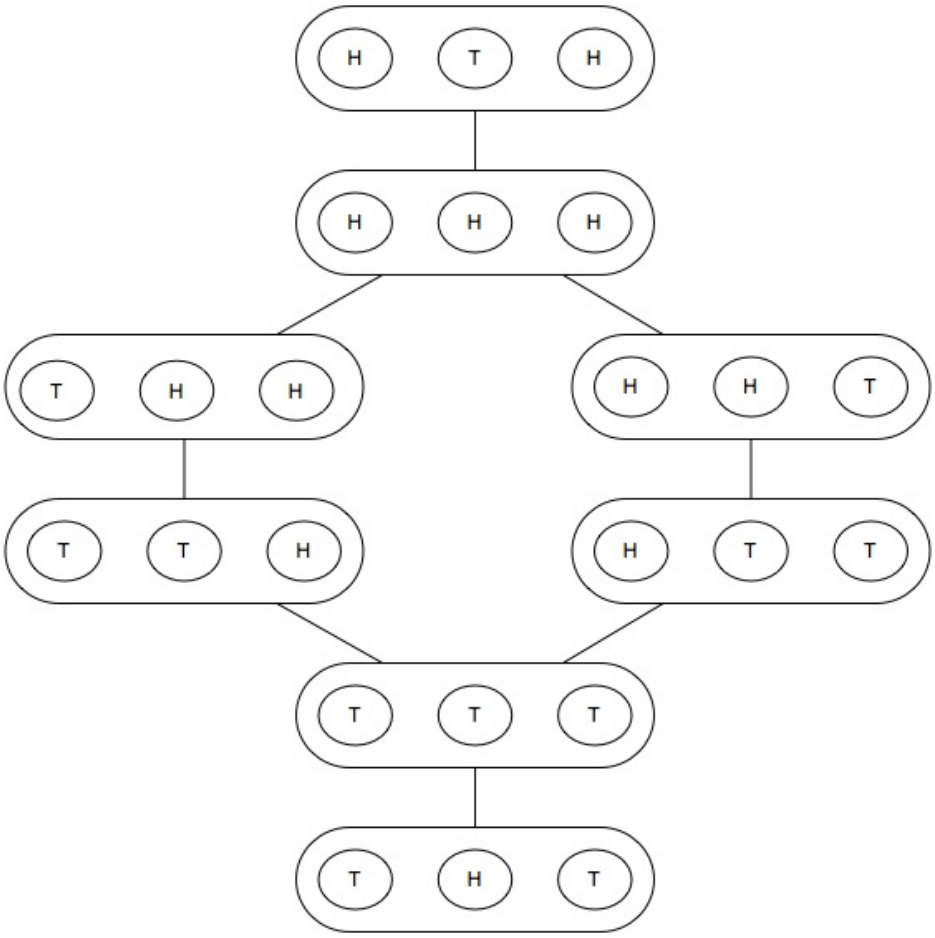
1. You may flip the middle coin (from heads to tail or vice versa) whenever you want.
2. You may flip one of the end coins (from heads to tail or vice versa) only if the other two coins are the same as each other (both heads or both tails).

You are not allowed to change the coins in any other way, such as shuffling them around.

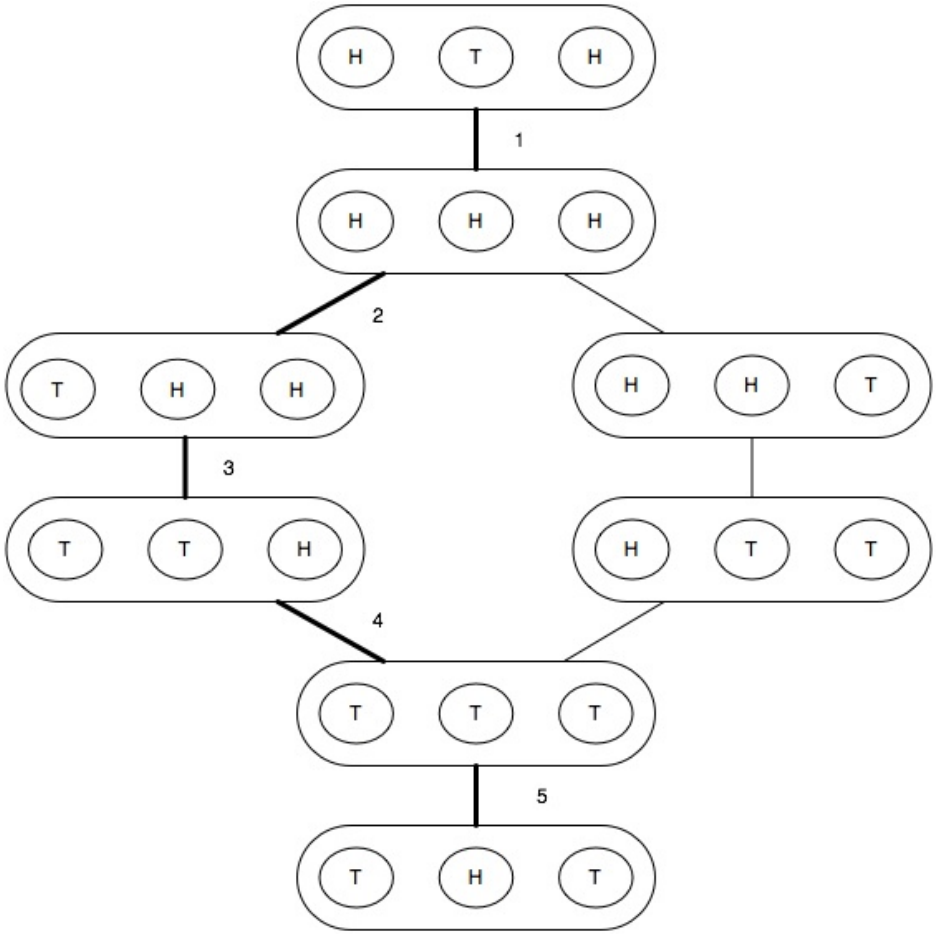
From the starting position *head-tail-head*, the first rule allows us to flip the middle coin:



Undirected State Graph for the Coin Game



One Solution for the Coin Game



1.2 Graph Traversals

Graph vertices don't have children like tree nodes, so typical tree traversal algorithms are not generally applicable to graphs.

There are two common ways to traverse graphs:

1. **Breadth-First Search** uses a queue to keep track of vertices that still need to be visited.
2. **Depth-First Search** uses a stack to keep track of vertices. It can also be implemented recursively so that it does not explicitly use a stack of vertices.

Traversal algorithms must be careful that they don't enter a repetitive cycle (e.g., to a neighbor, then back, etc.). It is necessary to *mark* each vertex as it is processed.

The progress of a traversal after the start vertex depends on the traversal method that is used.

To design and implement search (traversal) algorithms, a programmer must be able to analyze and predict their behavior. Questions that need to be answered:

- Is the algorithm guaranteed to find a solution?
- Will the algorithm terminate, or can it get stuck in an infinite loop?
- When a solution is found, is it guaranteed to be optimal?
- What is the complexity of the search process in terms of time usage? Memory usage?

1.2.1 Depth-First Search (DFS)

A traversal that only processes those vertices that can be reached from the start vertex.

The traversal proceeds as far as possible before it backs up.

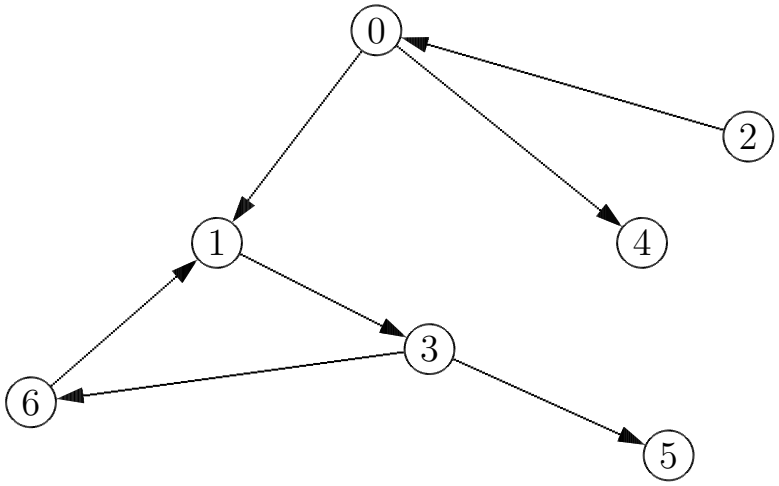
1.2.2 Breadth-First Search (BFS)

A breadth-first search uses a queue to keep track of which vertices might still have unprocessed neighbors. The search begins with a starting vertex, which is processed, marked, and placed in the queue. After this, the queue is processed repeatedly using the following algorithm:

1. Remove a vertex, v, from the front of the queue.
2. For each unmarked neighbor, u of v:
   Process u, mark u, and then place u in the queue (since u may have further unprocessed neighbors).

Example: Depth-First Search

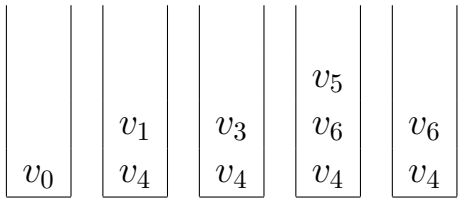
Start: v_0
Goal: v_6



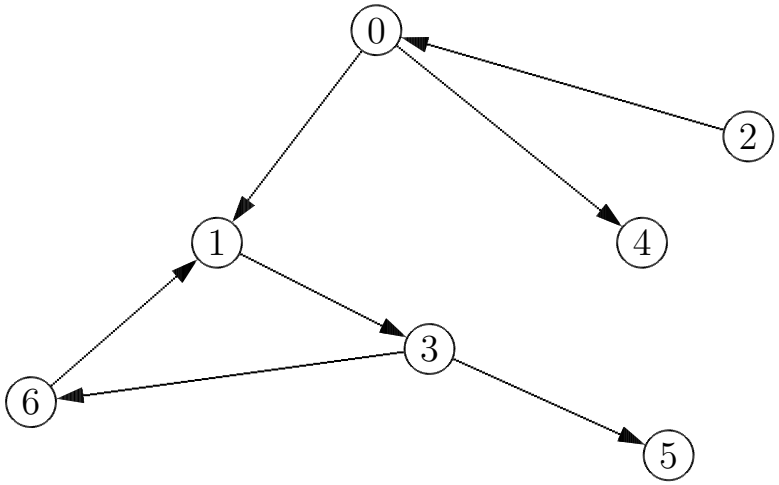
Directed Graph

Depth First Search

Stack Contents:

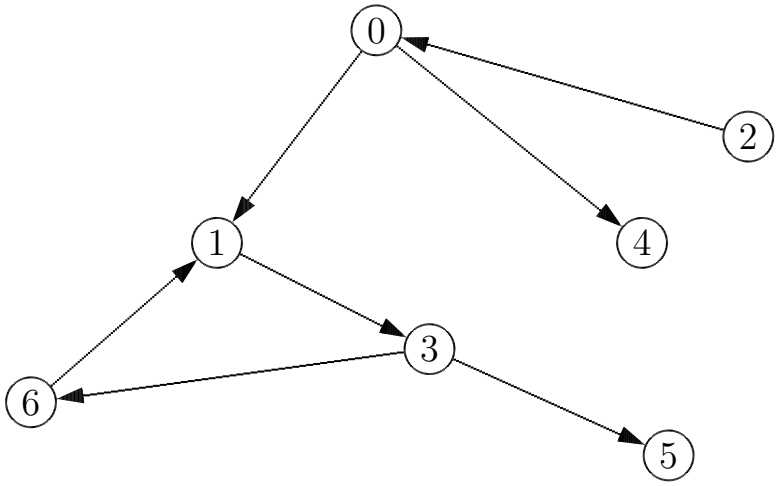


Start: v_0
Goal: v_6



Example: Breadth-First Search

Start: v_0
Goal: v_6



Directed Graph

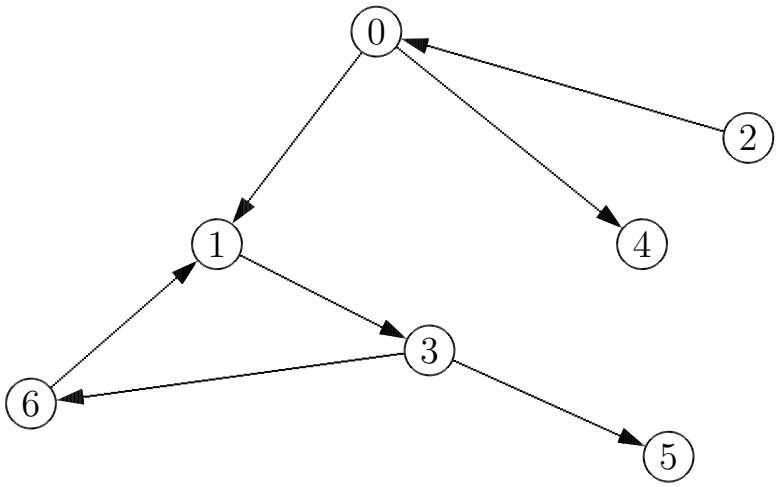
Breadth First Search

Queue Contents:

v_0		
v_1	v_4	
v_4	v_3	
v_3		
v_5	v_6	
v_6		

Front Back

Start: v_0
Goal: v_6



1.2.3 Differences between Breadth-first and Depth-First Searching

- Since all paths of length one are investigated before examining paths of length two, and all paths of length two before examining paths of length three, a breadth-first search is guaranteed to always discover a path from start to goal containing the fewest steps.
- Since one path is investigated before any alternatives are examined, a depth-first search *may*, if it is lucky, discover a solution more quickly than the equivalent breadth- first algorithm.
- Suppose for a particular problem that some but not all paths are infinite, and there exists at least one path from start to goal that is finite. Breadth-first search is guaranteed to find a finite solution. Depth-first search may have the unfortunate luck to pursue a never-ending path, and it can fail to find a solution.