1 CandyGrab Continued

Today we get to bring the CandyGrab agents from lecture to life! Download and unzip:

- https://www.cs.cmu.edu/~15281/recitations/rec1/candygrab.zip

This code is meant for you to mess around with during recitation. Don't worry about breaking things, as you can always redownload the original code. Inside the CandyGrab directory you'll find:

- `candygrab.py`: The game environment
- `agent.py`: The Agent base class. All agents inherit from this simple base class.
- `agent*.py`: Implementations of various agents. We saw some of these in lecture.
- `play_one.py`: Simple script to have two agents play one round of the game
- `play_many.py`: Simple script to have two agents play many rounds of the game

Feel free to edit any of these files or make copies of `play_*.py` or `agent*.py` to create your own battles or agents. But first, we'll have you just run and inspect the code to get an idea of how things are working in this simple game. You can start by running the following:

- `python36 play_one.py`
- `python36 play_many.py`

Take a look through the source code and answer the following questions:

(a) How is the state represented in the code?

(b) What is Agent005's strategy?

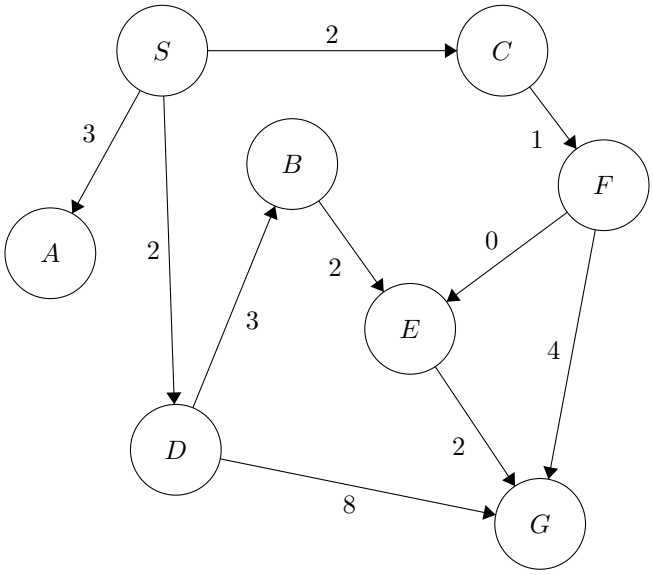
(c) What is Agent003's strategy?

(d) How do the AgentRL stats change when playing against an Agent008 rather than an Agent007? What difference in the two agents do you think causes this? (You should modify `play_many.py`)

(e) When we run the games 1000 and 10000 times, how do the AgentRL stats change when playing against another AgentRL? Which states seem to be good states to take? (You should modify `play_many.py`)

(Bonus) AgentRL has an `explore_mode` setting that defaults to `True`. What happens when you play 500 games, turn off explore mode (`bb8.explore_mode = False`), and then play 500 more games?

2 Search Algorithms



Using each of the following graph search algorithms presented in lecture, write out the order in which nodes are added to the explored set, with start state S and goal state G. Break ties in alphabetical order. Additionally, what is the path returned by each algorithm? What is the total cost of each path?

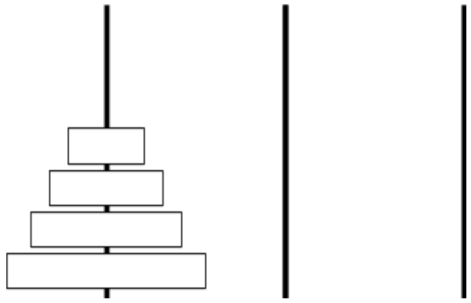
(a) Breadth-first

(b) Depth-first

(c) Uniform cost

(d) Iterative deepening

3 Tower of Hanoi



The Tower of Hanoi is a canonical puzzle studying problem solving and formulation. The puzzle starts with n disks of different sizes stacked in order of size (see picture above) on a peg, along with two empty pegs. We can move disks freely between the pegs, but larger disks cannot be stacked on top of smaller ones. The goal is to move all disks to the third peg.

We will attempt to formulate Tower of Hanoi as a search problem.

(a) How could we represent this puzzle as a problem, ie. what would the states be?

For the following questions, assume that you have used your state representation in your answer above.

(b) What is the size of the state space in terms of n?

(c) What is the starting state?

(d) From some given state, what legal actions are there?

(e) What is the goal test? Remember that this determines whether a given state is a goal state, `goalTest(state)`.