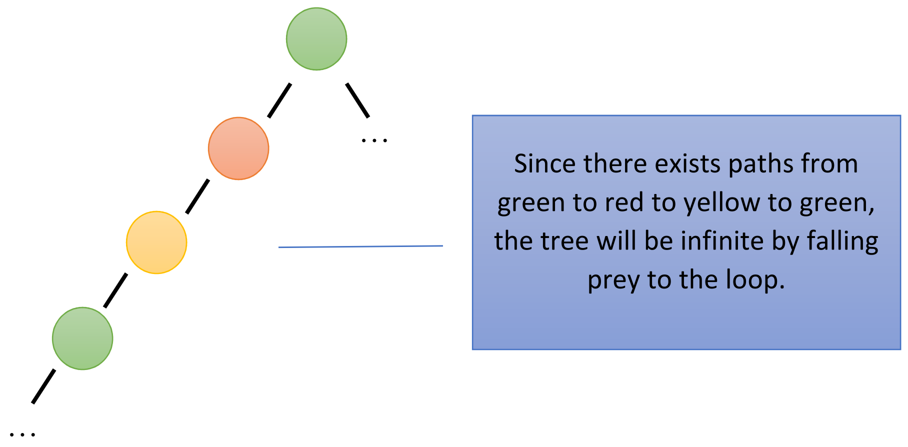
Illinois Institute of Technology
Department of Computer Science

Homework Assignment 1

CS 480 Artificial Intelligence: Planning and Control
Fall Semester, 2018

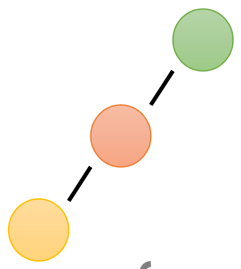
1. **Does a finite state space always lead to a finite search tree? How about a finite state space that is a tree? Can you specify precisely what types of state spaces always lead to finite search trees?**

No, a finite state space does not always lead to a finite search tree because there is a chance of loops. Taking the example explained in class, given a finite state space of three locations in Romania, there can exists a loop between three cities. There can be a path from city A, another to city B and another to city C; thus, the tree will have leaves to each city creating an infinite loop. The diagram below describes this problem:


Since there exists paths from green to red to yellow to green, the tree will be infinite by falling prey to the loop.

Yes, a finite state space can be a finite tree since there exists no loops in the tree.

The previous case arose due to loops or cycles existing between the finite states in the three. If there are not cycles between the finite state space, (i.e. there does not exist paths that can start at a state or node that will come back to that node or state) then the finite state space creates a finite state tree. If the finite state space leads to a simple graph (by definition does not have cycles), then the graph will create a finite search tree. The diagram below visualizes the tree:



2. (a) Color a planar map using only four colors, in such a way that no two adjacent regions have the same color.

Initial State:	The planar map has no colors
Goal Test:	Is all the map colored? Does he adjacent regions not have the same color as the initial region?
Successor Function:	A color is selected for a region. Repeating the selecting process for other regions
Cost Function:	The number of assigned regions

(b) A 3-foot-tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. He would like to get the bananas. The room contains two stackable, movable, climbable 3-foot-high crates.

Initial State:	The monkey has no bananas. The crates are displaced in the room.
Goal Test:	Does the monkey have bananas?
Successor Function:	A combination of the following: 1. The crates are moved, stacked and/or unstacked. 2. The monkey is moving (x direction) or climbing (y direction) or displacing the bananas.
Cost Function:	The number of steps taken to acquire the bananas. (moving, stacking, unstacking crates, moving and climbing done by the monkey and/or displacing the bananas)

(c) You have three jugs, measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You can fill the jugs up or empty them out from one to another or on to the ground. You need to measure out exactly one gallon.

Initial State:	The jugs are empty (no water in the jugs).
Goal Test:	Does any of the jugs contain one gallon of water?
Successor Function:	Filling the jugs with water. Emptying the water to the ground from 12-gallon jug, 8-gallon jug and/or 3-gallon jug. Emptying the water from partially/fully filled jugs to another partially filled/empty jug.
Cost Function:	The number of steps of emptying or filling the jugs.

3. **(a) Suppose actions can have arbitrarily large negative costs. Why would this force any optimal algorithm to explore the whole state space?**

   Since an optimal algorithm would be trying to minimize the total action cost, negative costs associated to an action would greatly minimize any chosen path. Thus, a path that would not be optimal can contain an action with negative cost that can make it optimal. If this path is ruled out in the beginning of the search algorithm for an optimal route, this path will be ignored and not an optimal path would be returned. Since this can occur for any given path, the whole tree must be explored to find an optimal path.

   (b) Does it help if we insist that step costs must be greater than or equal to some negative constant c? Consider both state spaces that are trees and those that are full graphs.

   Given a cutoff negative constant greatly helps in the calculation of the best path and not needing to search the entire tree. First the tree can be searched for the most optimal path. Since it is known how many nodes exists in the tree and how many nodes remain in the tree by excluding the nodes that make up the optimal path, all the remaining nodes times the constant c (-c * nodes remaining) can make up a rival path in worst case. By comparing the "optimal path" to this number, it can be concluded if this path is indeed optimal (-c*nodes remain > "optimal path"?).

   When considering full graphs, the same rule applies if it is a simple graph (no cycles or loops). Otherwise, the constant does not help since a looped path that produces a total negative cost can be reused infinitely to cut the cost to any "optimal path" deduced beforehand.

   (c) Suppose there is a set of operators that form a loop, so that executing them in some order results in no net change to the state. If all of these operators have negative cost, what does this imply about the optimal behavior for an agent in such an environment?

   As mentioned before, if the graph is not a simple graph, there cannot be an optimal path. This is because a negative loop makes the path more optimal as it is iterated on repeatedly. Thus, an optimal path is never reached and will forever be looping in the cycle.

   (d) One can easily imagine operators with high negative cost, even in domains such as route finding. For example, some stretches of road might have such beautiful scenery as to far outweigh the normal costs in terms of time and fuel. Explain, within the context of state-space search, why humans do not drive around scenic loops indefinitely. How could such an idea be formalized in operators for route finding so that artificial agents can also avoid looping?

Within the state-space search, humans do not drive around scenic loops indefinitely since time and fuel will have a higher weight as the path is looped; even though the weight of a scenic route action is negative, the weight is relative to fuel and time. Thus, the loop's cost increases as time goes on. The graph becomes dynamic and this idea can be applied to operators to avoid looping. Dynamically changing the cost of future negative paths if a negative path was already taken or deleting negative paths as a set amount of negative cost in a path are accrued can stop the agent from looping.