# Uninformed Search

Lab 2

## Agenda

1. Lab 1 questions (homework)
2. Exercise 1 (tree search)
3. Exercise 2 (vacuum search)
4. Homework

# Questions?

**Regarding Lab 1**

## Exercise 1

- Refer to search.py
- Implement the functions INSERT, INSERT_ALL, and REMOVE_FIRST

- Successor nodes are inserted at front of the fringe (successor list) as a node is expanded. Is this a breadth (FIFO) or depth-first search (LIFO)?
- For goal J, give the fringe (successor list) after expanding each node
- What is the effect of inserting successor nodes at the end of the fringe as a node is expanded? A depth or breadth-first search?
- For goal J, give the fringe (successor list) after expanding each node.

## Exercise 2

Use your search program to solve the vacuum world problem using breadth-first search.

Hint: one way to represent the state space in Python is by a dictionary where the current state is a tuple:
(location, A status, B status)
and a list holds successor states for each action.
[(location, A status, B status), (location, A status, B status), (location, A status, B status)]

For example:
('A', 'Dirty', 'Dirty’): [('A', 'Clean', 'Dirty'), ('A', 'Dirty', 'Dirty'), ('B', 'Dirty', 'Dirty’)]

## Homework

Modify your search-program to solve the following problem:
A farmer has a goat, a cabbage and a wolf to move across a river with a boat that can only hold himself and one other passenger. If the goat and wolf are alone, the wolf will eat the goat. If the goat and cabbage are alone, the goat will eat the cabbage.
Define the state space for the problem. Hint: Use a tuple to represent the side of the river each is located; for example ('W', 'E', 'W', 'W') can represent the (farmer, wolf, goat, cabbage) locations. Use a list of tuples for the successor states. Include successor states that violate the problem constraints, that is ('W', 'W', 'E', 'E') which is the goat is alone with the cabbage; don't violate requirement that the boat can hold only two passengers (e.g all four passengers cannot move from one side to another at once, that is ('W', 'W', 'W', 'W') cannot become ('E', 'E', 'E', 'E')).
Define successor_fn to return a list of states that do not violate the problem constraints