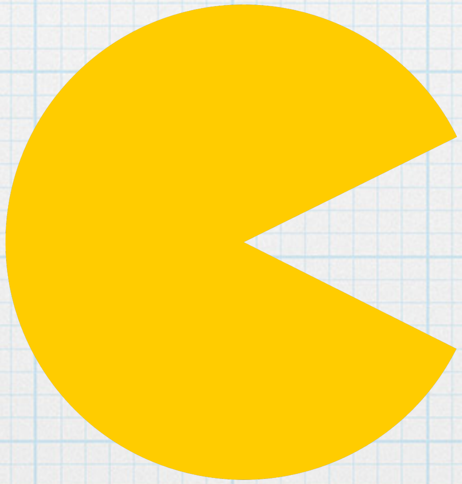
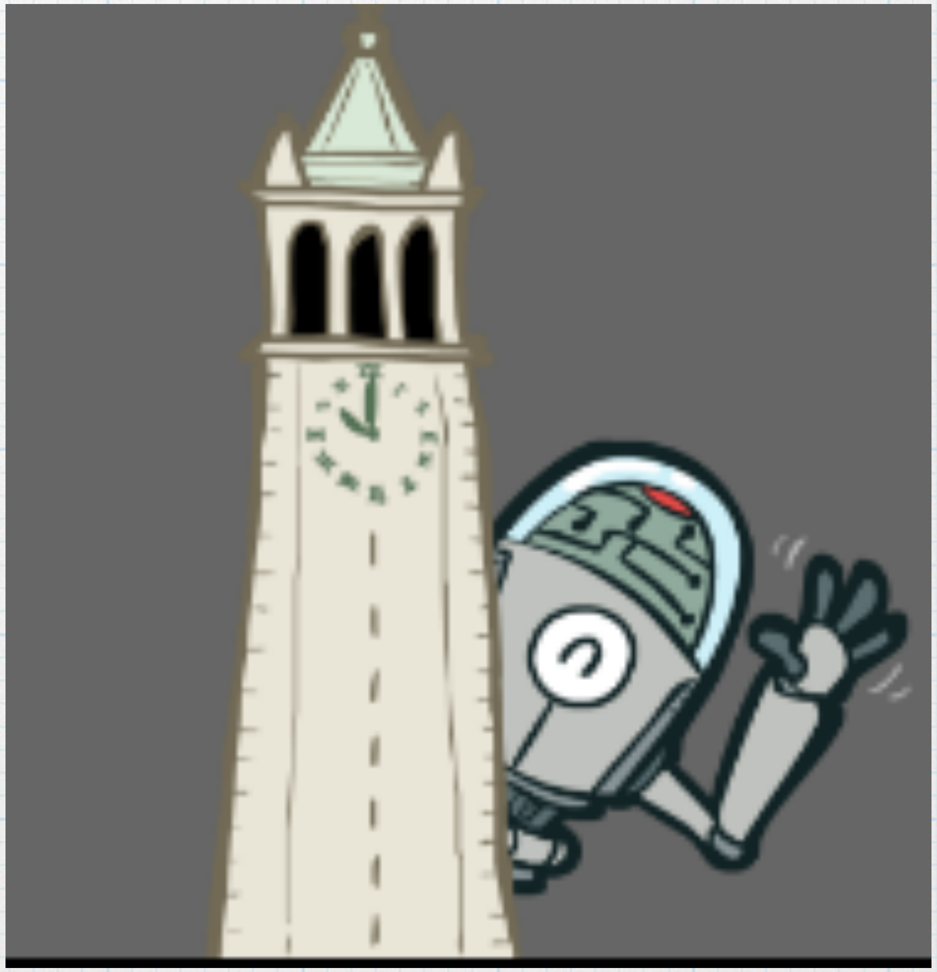
CS 188 Discussion 0: Welcome!

TA: Sherdil Niyaz



Welcome to Discussion!



Who am I?

- Sherdil Niyaz, Senior in EECS
- Discussion: 1-2 W in 3113 Etch
- Hopefully another section if we expand the course
- Office hours: 1-3pm on Friday in 341A Soda (In Upper Division Lounge)
- Interests: Teaching, Robotics, CS Theory, AI

Email

- sniyaz@berkeley.edu
- Feeling lost in the class? Falling behind? Just want to talk about the course (or anything?) Don't be afraid to email!
- Also, please bug me if I don't respond. I don't mind.

Section Site

- http://sniyaz.weebly.com/cs188.html
- Don't feel pressured to take notes. I will put up anything I use on this site.
- Instead, I want you listening in section and not rushing to write things down :)

Rules of Section

- **Be respectful.** Don't be condescending to people who take longer to really master a topic.
- **Don't be afraid to ask questions**. The only stupid question is the one you don't ask.
- If I don't address you using your **name**, call me out on it!
- If I talk too fast, give me a signal to slow down.

Who are you?

- Turn to somebody next to you and introduce yourself!
- Name! Year! Major! Social Security Number!
- Share an interesting thing you've done, about you, etc. Just something interesting.
- You may be asked to share.....make sure you pay attention.

Anything interesting?

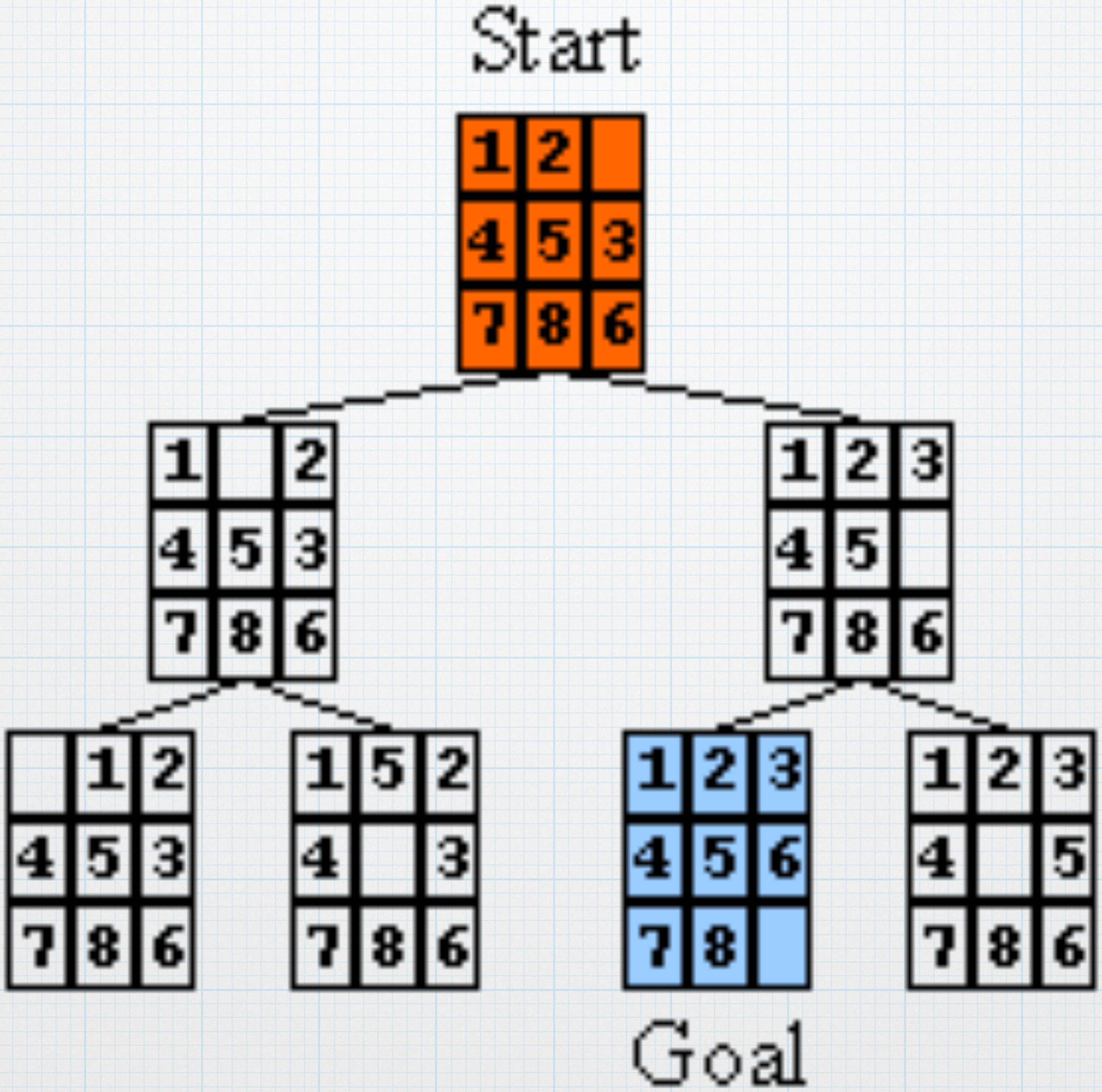
(Keep it PG-13 and legal please)

Things you should remember for this class

- CS61B: Graphs, Asymptotic Analysis
- CS70: General Probability, Expectation, Bayes' Rule
- Not comprehensive list
- Now is a good time to review these if you've forgotten!

Search

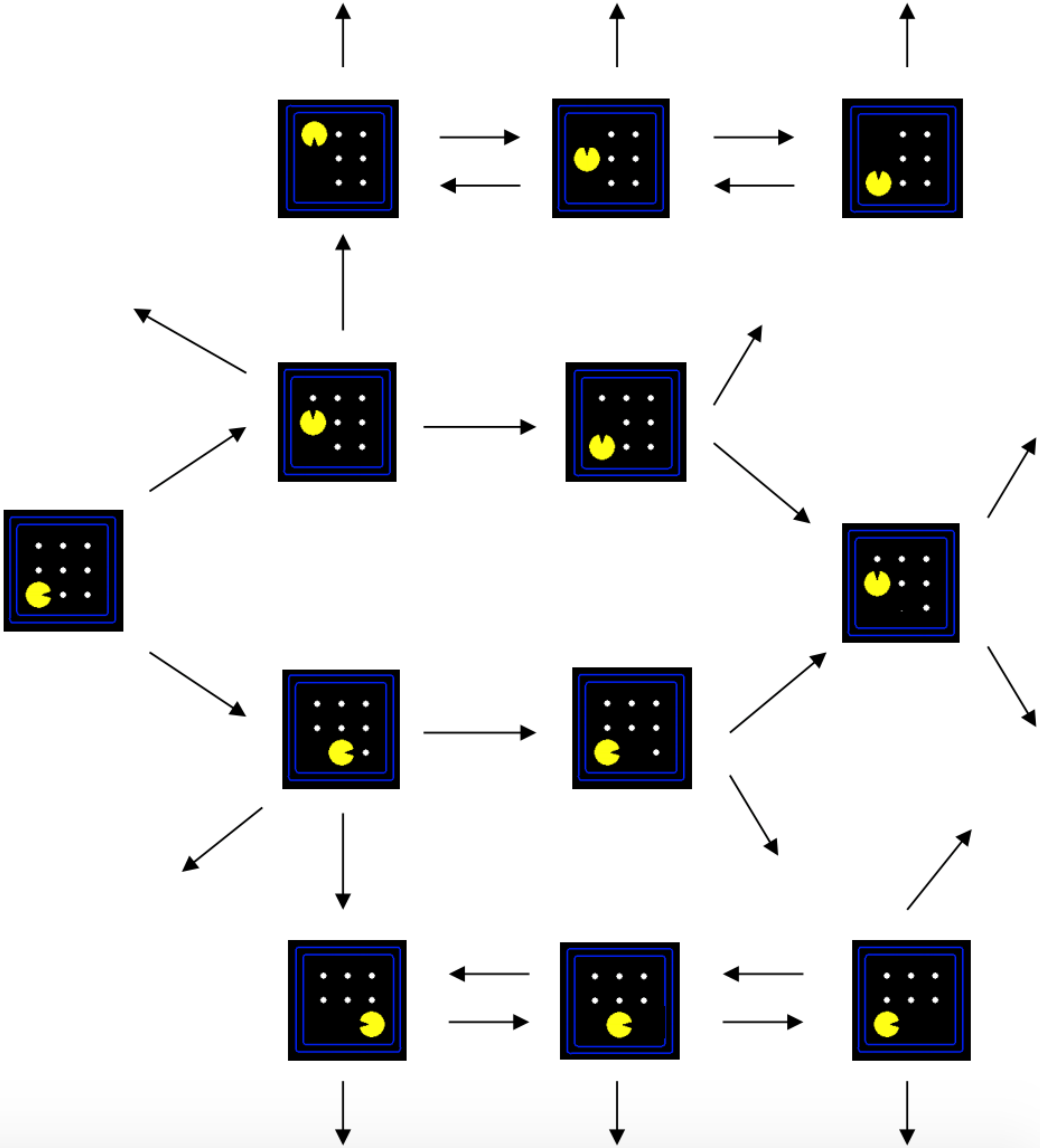
- I have a problem with a bunch of states I can progress through as I love the problem.
- I'm at a START state. I want to reach an END state. How should I get there?
- There are actually multiple paradigms to solving problems like this. The first is a graph based approach.
- There are others! (Game Trees, CSPs, Logic)



Start

Goal

1 2 _
4 5 3
7 8 6
1 _ 2
4 5 3
7 8 6
1 2 3
4 5 _
7 8 6
_ 1 2
4 5 3
7 8 6
1 5 2
4 _ 3
7 8 6
1 2 3
4 5 6
7 8 _
1 2 3
4 _ 5
7 8 6



State representation

- Information needed to encode what your progress though the problem is.

- Another way to think about it: what information do you store store to know which node in the graph you're at?

- **Minimal State Representation**: what is the **smallest** amount of information you can store to know which node in the graph you're at?

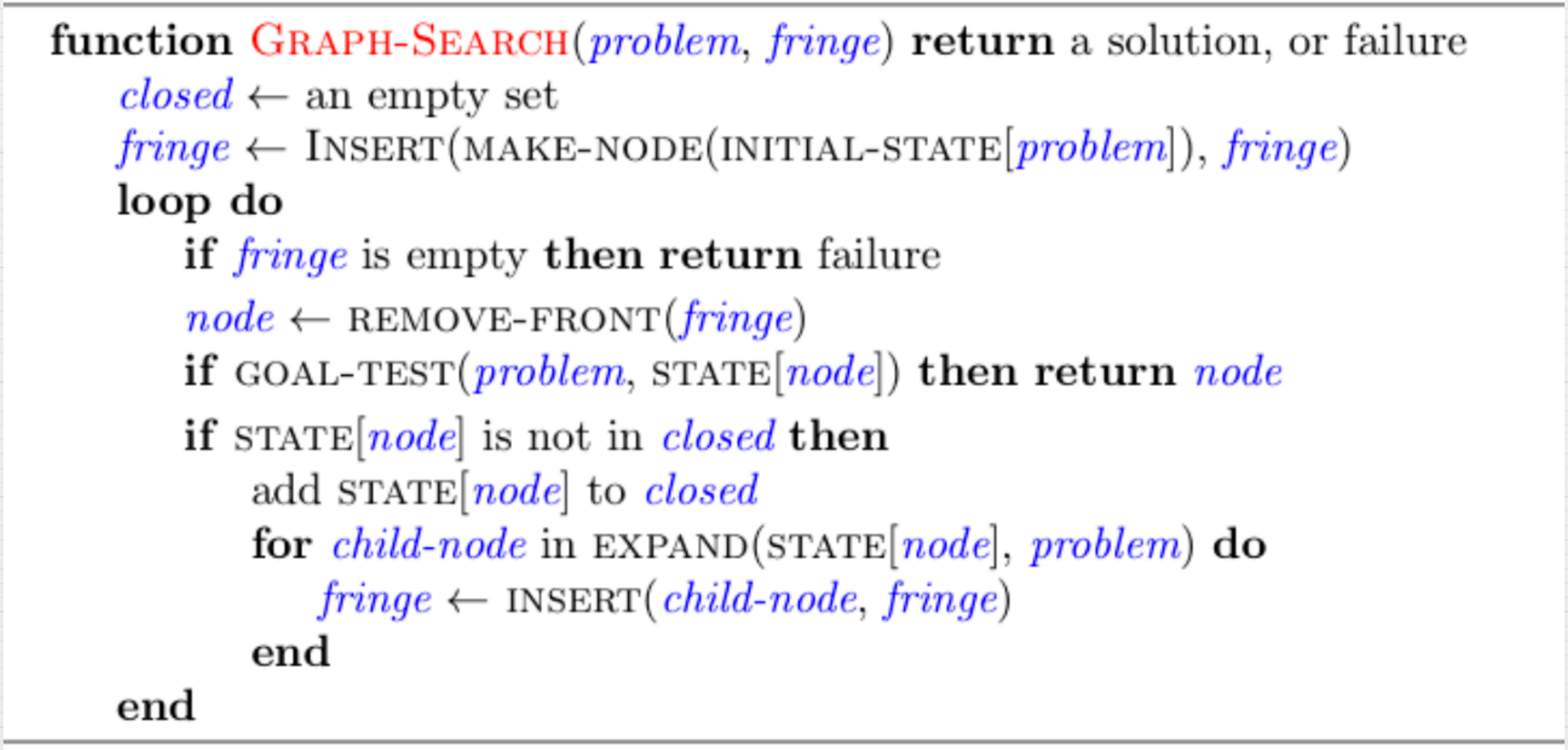
Transition function

- Which actions can I take at each state?
- Where do those action take me?
- **Graph analogy**: each edge out of a state/node represents an action. Which edges should exist between states and which shouldn't?

Graph Approach

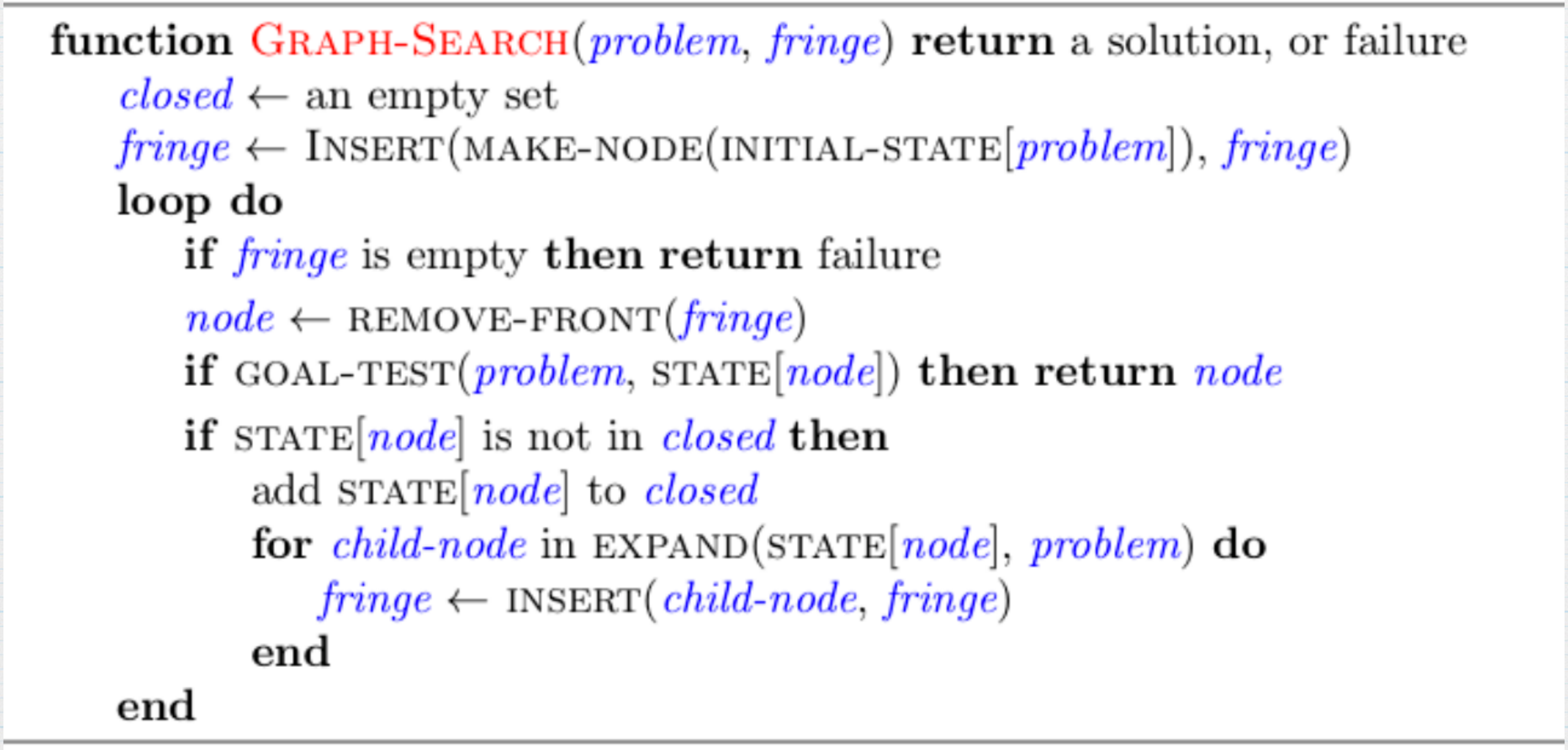
- Nodes = states
- Edges = actions. Called transition function.
- Action can have costs.
- How do we solve? Just apply **graph search algorithms** from CS61B! (DFS, BFS...)

CS61B Fun Times



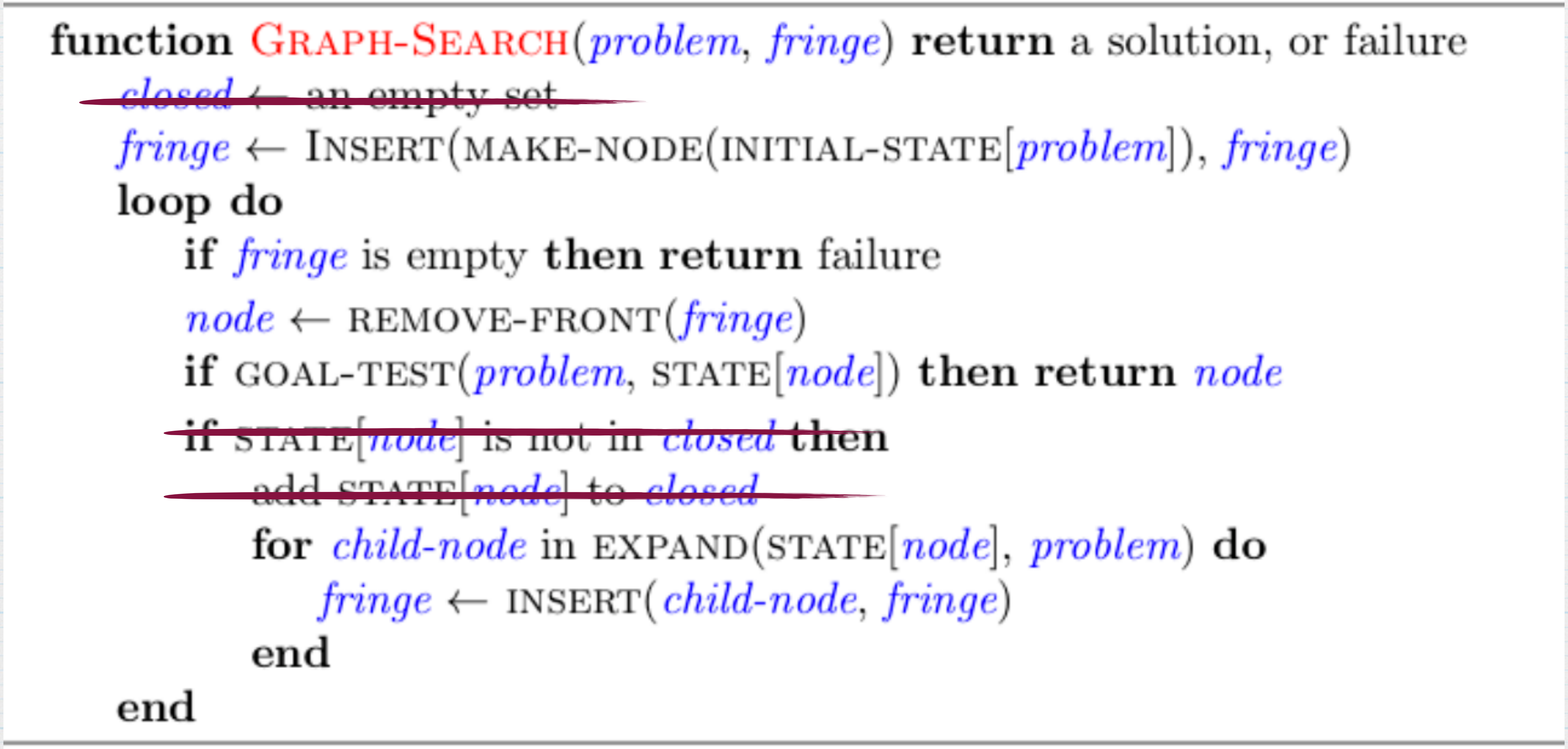
```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

How is Tree Search Different?



```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

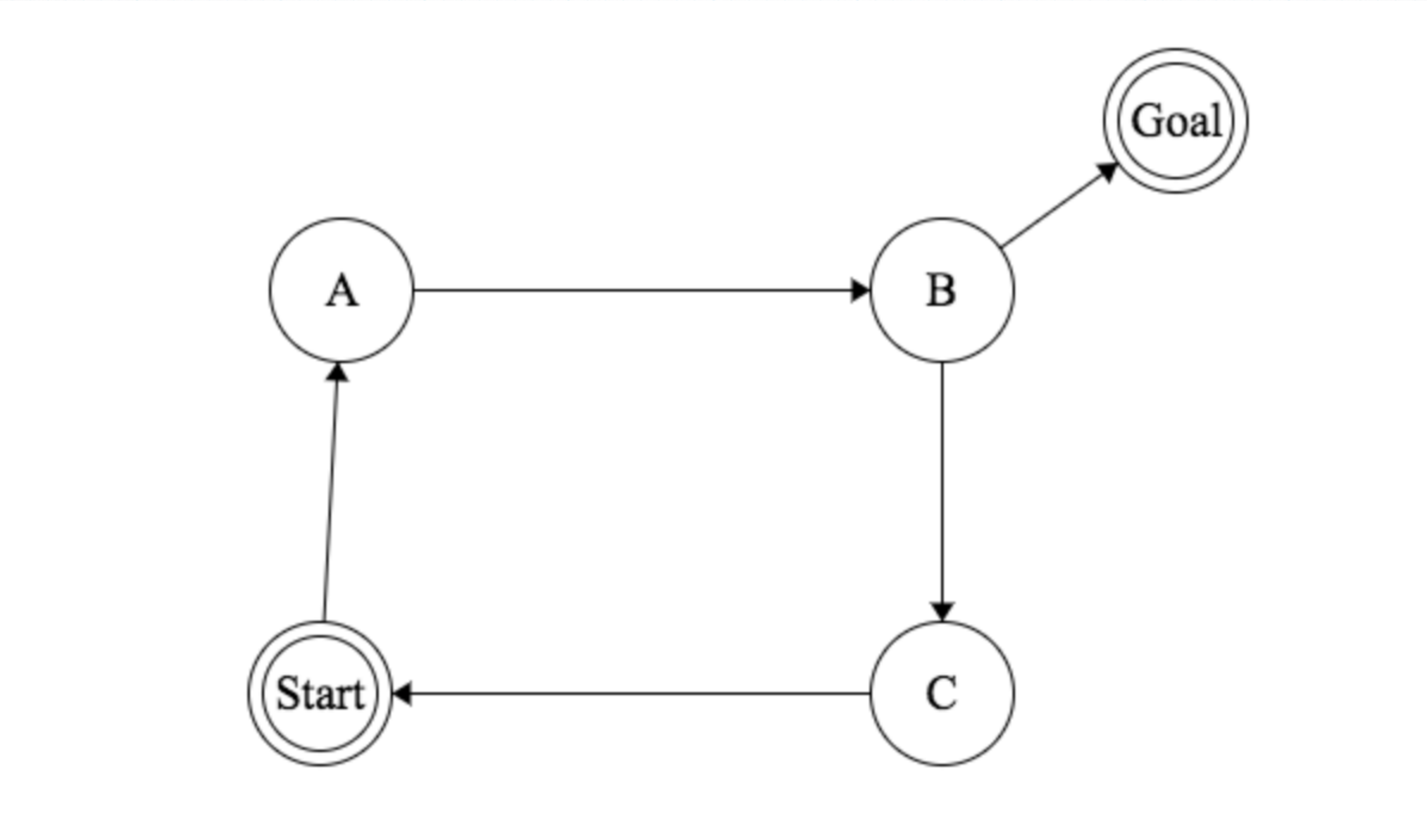
How is Tree Search Different?



```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    ~~closed ← an empty set~~
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        ~~if STATE[node] is not in closed then~~
            ~~add STATE[node] to closed~~
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

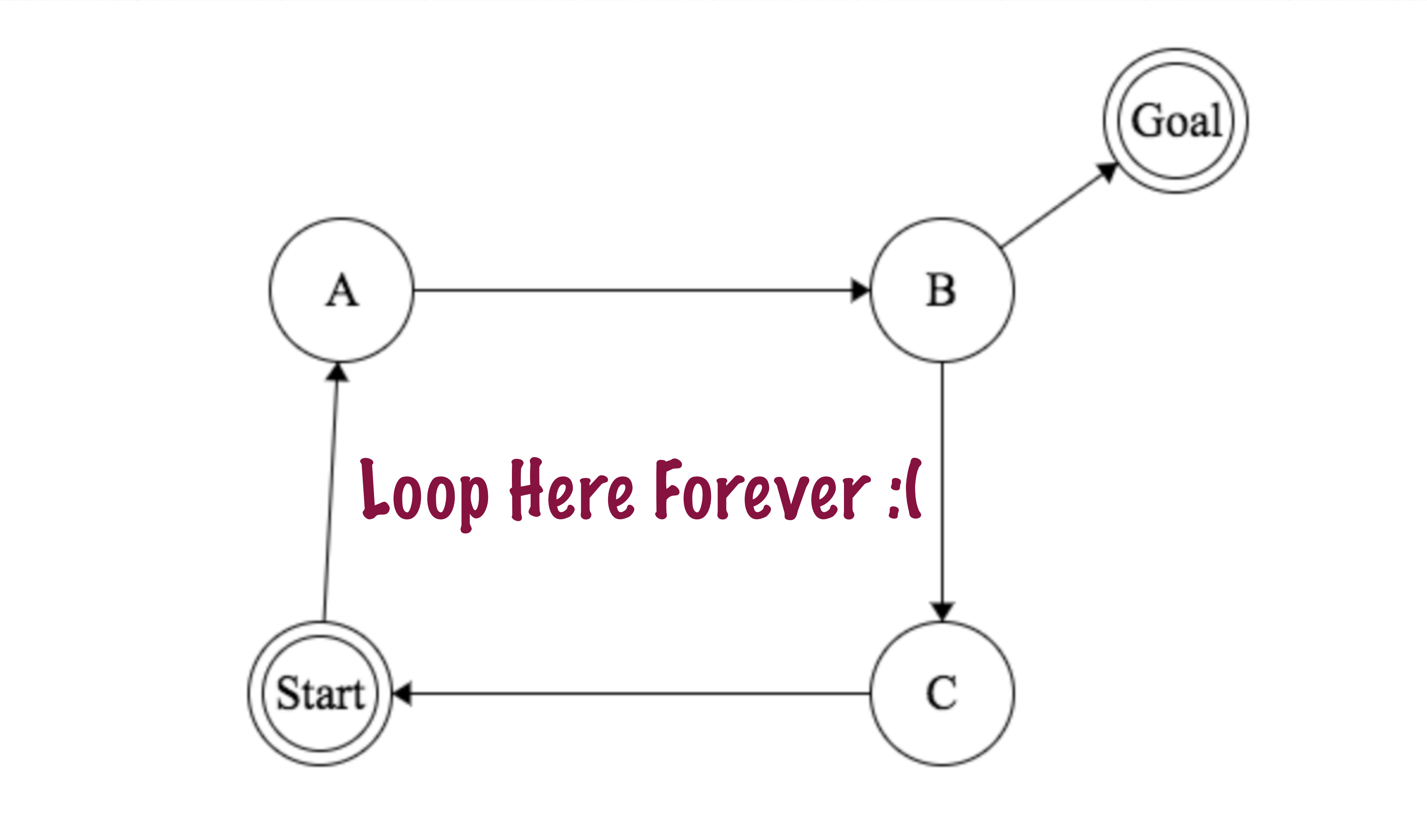
Ignore the Closed/Explored Set!

Tree Search Failure

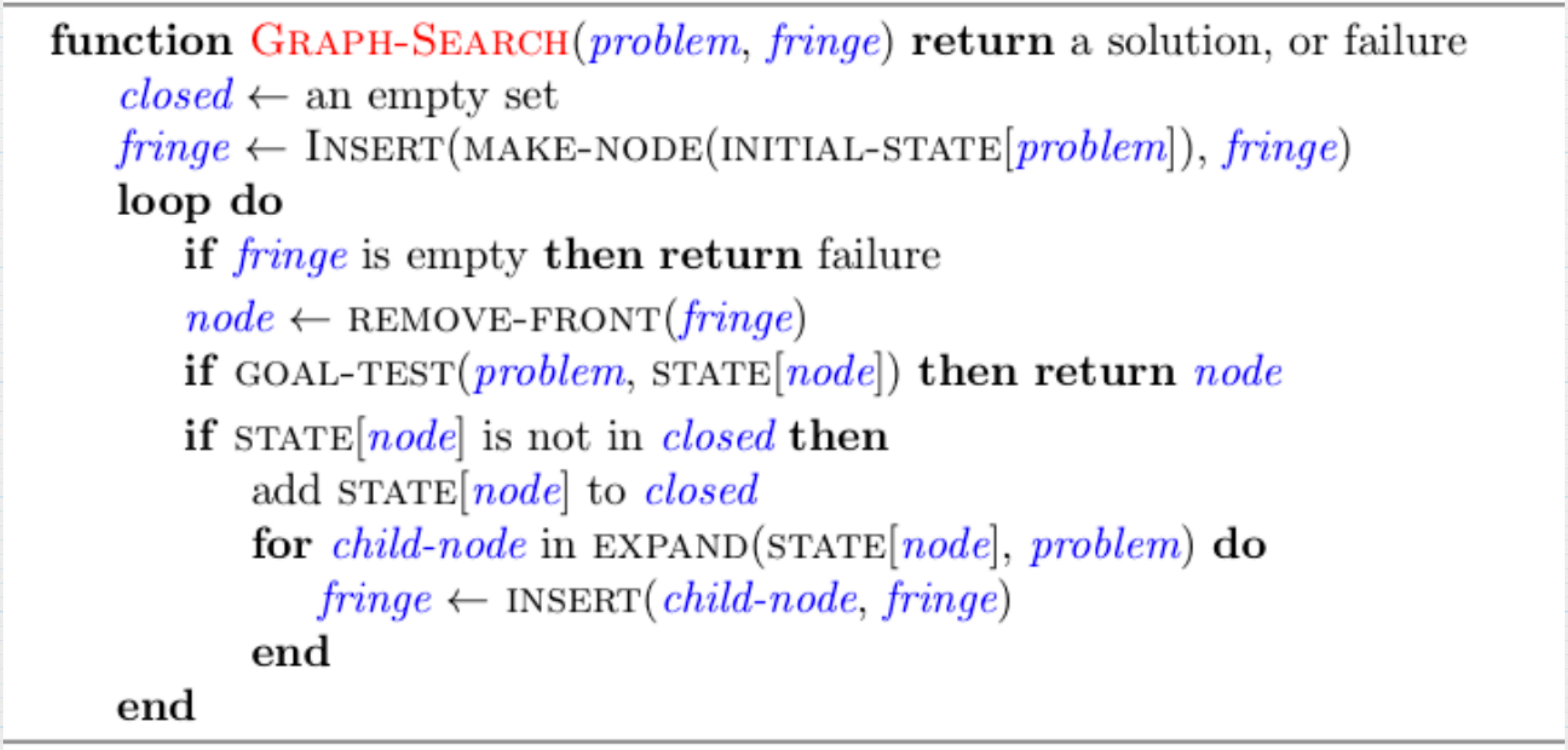


- What can happen if we get unlucky?

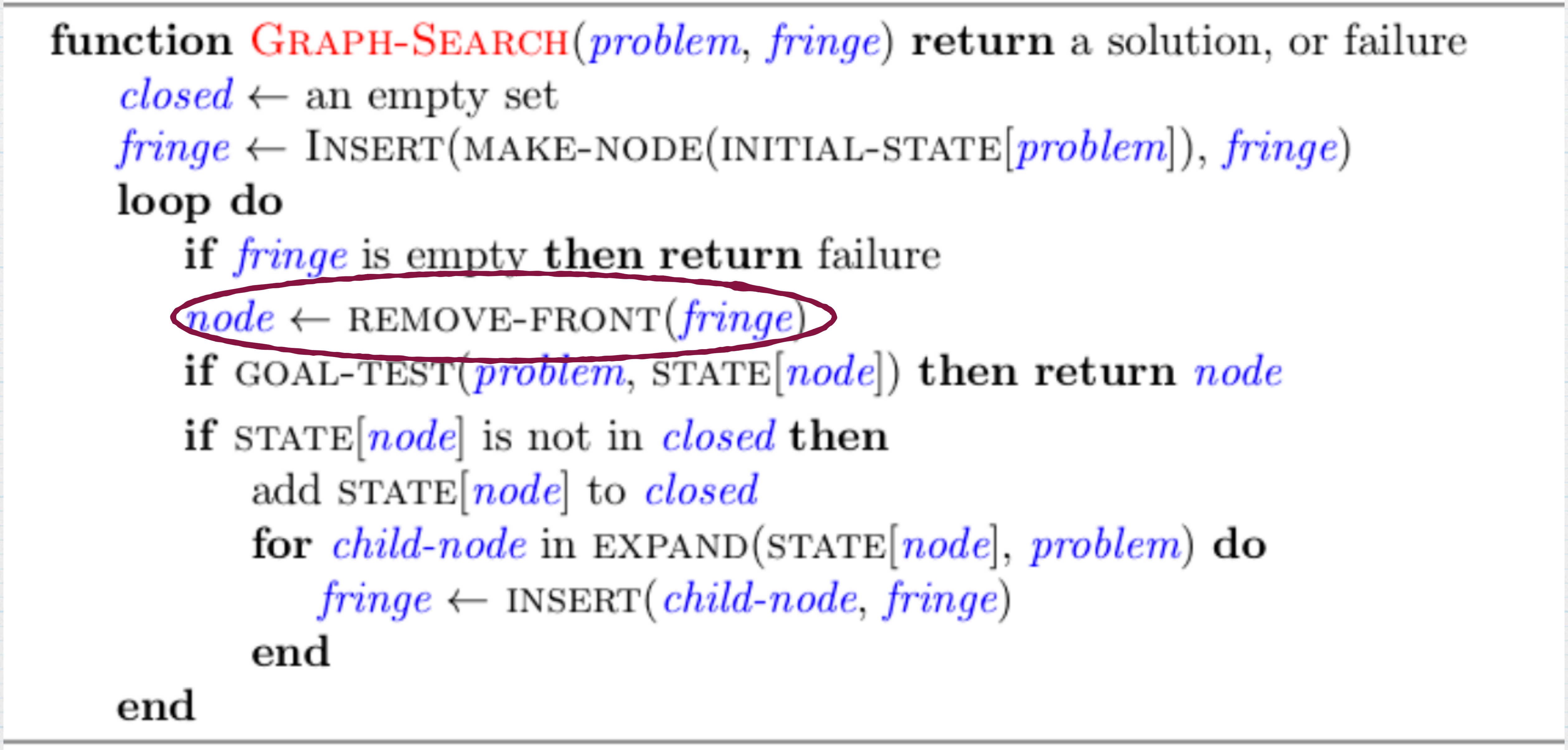
Tree Search Failure



- What can happen if we get unlucky?



```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```



```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

This is the ONLY decision that changes the type of search!

PDB tutorial (if time allows)