Programming for Engineers

Workshop on intelligent agents and search

Dr Ronald Grau School of Engineering and Informatics Autumn term 2019

Previous lab class

Problem-solving

- Symbol processing is a means to transform information & knowledge
- How to formalise a problem for search
- Two uninformed search strategies
  - BFS – Breadth-First Search
  - DFS – Depth-First Search

Problem-solving

Problem-solving with computer programs

- We use data structures (variables, arrays, etc.) to represent our problem-solving knowledge and to store partial solutions.
- We use language syntax to implement a formal mechanism that
  - Identifies ways in which possible actions can be arranged into sequences.
  - Finds a particular sequence of actions which achieves a desired result.

⬇

This is called **search**

Problem-solving

Problem-solving with computer programs

Basic components of a search algorithm:

- An ***initial state*** to start from and a ***goal state*** to look for (state representation).
- We keep and update ***a record of those states we still need to explore*** (*tree search*).
- In *graph search*, we also keep ***a list of those states already explored***.
- A ***successor function*** that encodes all valid rules for going from one state to the next.
- The successor function manipulates the symbols in the state representation to "compute" (i.e. generate) new states automatically whenever it is called.

Problem-solving

Problem-solving with computer programs

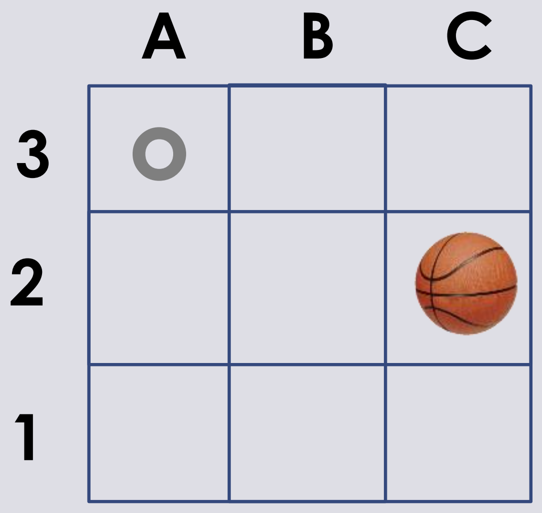
Basic flow of control in a search algorithm :

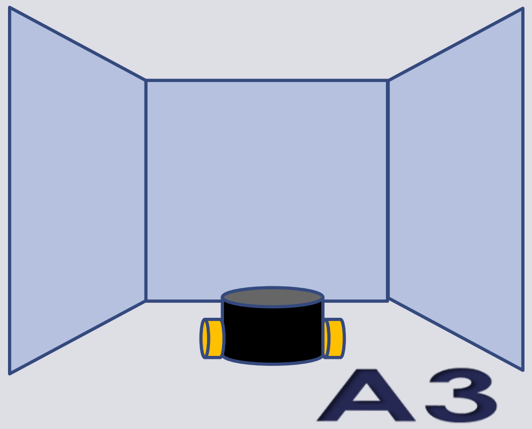
- Starting with the initial state, we run the successor function to find successive states
- We explore the successors by 1) performing a ***goal test*** and 2) running the successor function (if the *current node* is not a goal) in order to reveal *its* successors.
- Repeat the above until a goal has been found or until we have exhausted all options.
- If we found a state that is the goal, we backtrack to the initial state to establish the path that presents a solution.
  For this to work, every state needs to remember its origin, i.e. *parent* state.

Recap: BFS

Breadth-First Search (BFS)

- FIFO queue ADT





BFS: First-In First-Out

Waiting (queue)

FRONT REAR

Visited (list)

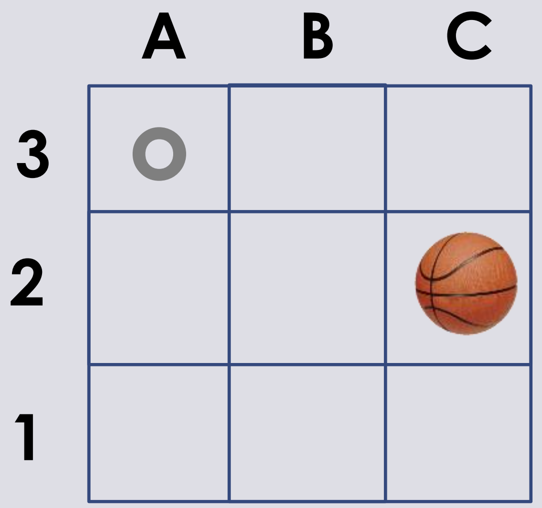
Search tree:

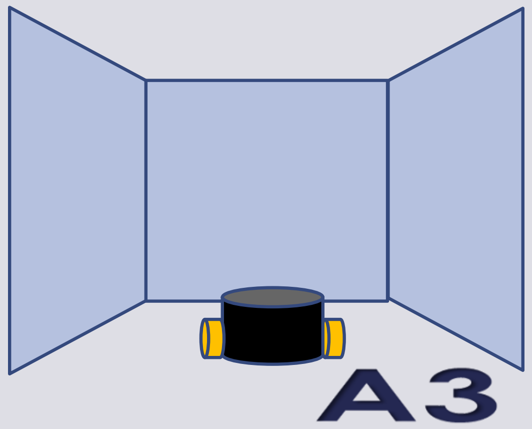
A3

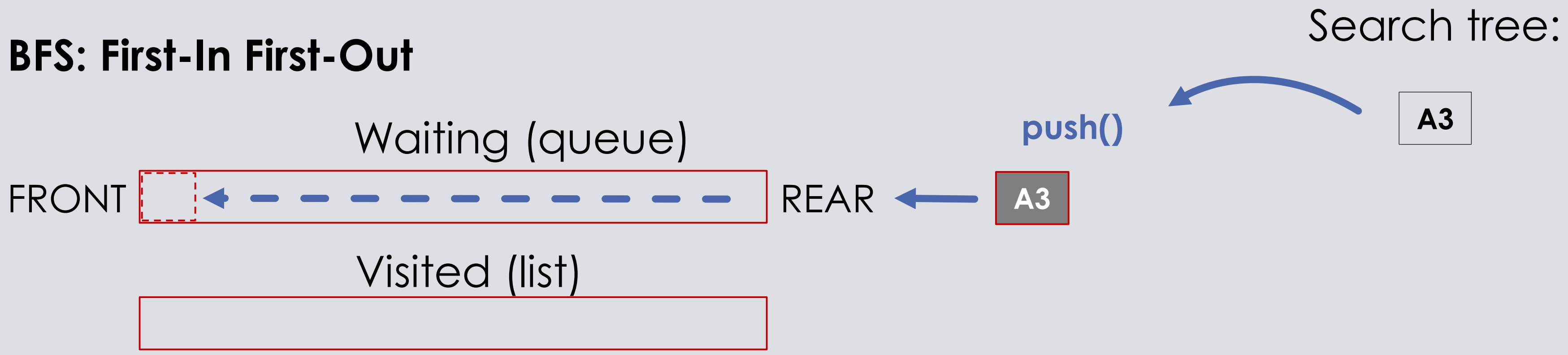
Recap: BFS

Breadth-First Search (BFS)

- FIFO queue ADT



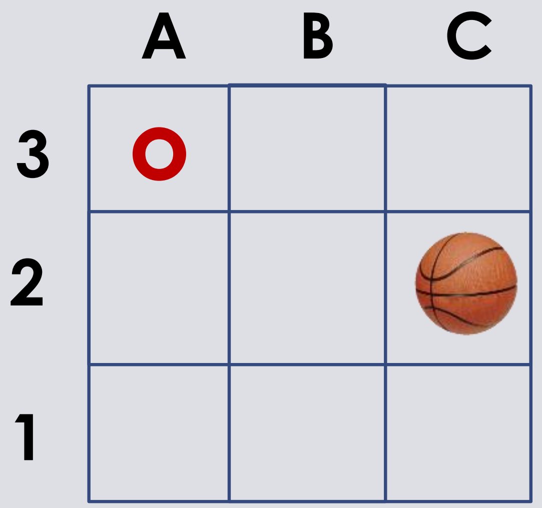


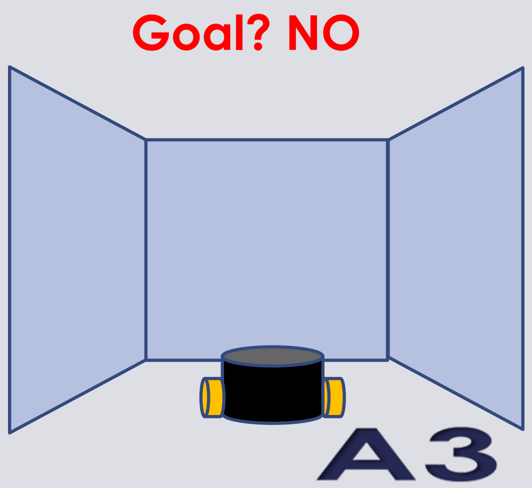


Recap: BFS

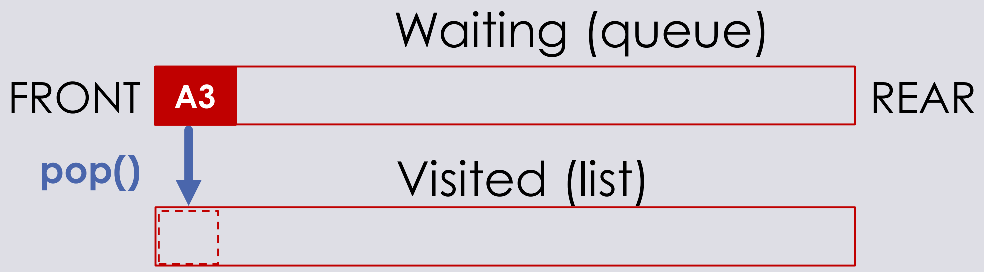
Breadth-First Search (BFS)

- FIFO queue ADT

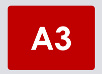




BFS: First-In First-Out



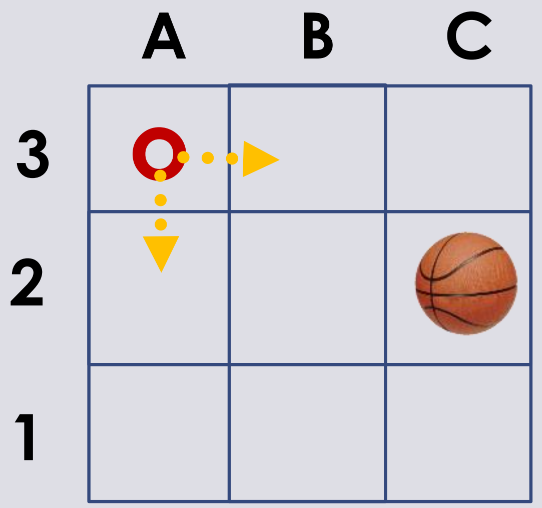
Search tree:

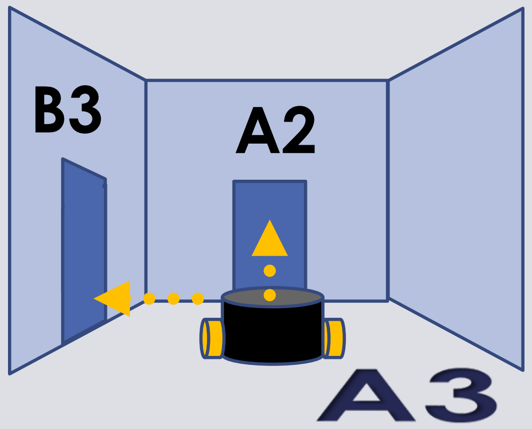


Recap: BFS

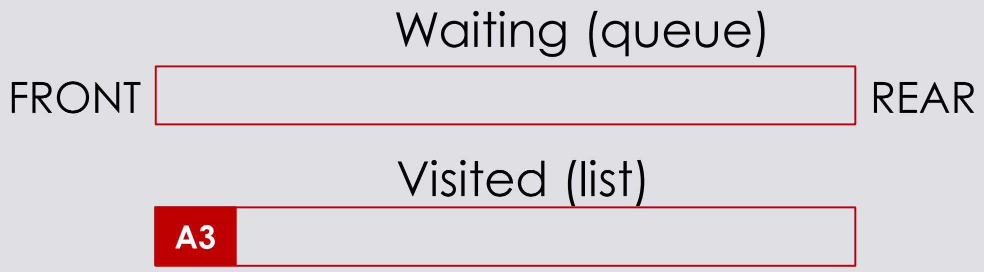
Breadth-First Search (BFS)

- FIFO queue ADT

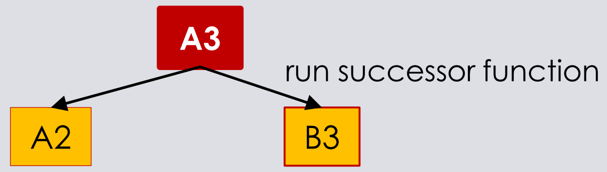




BFS: First-In First-Out



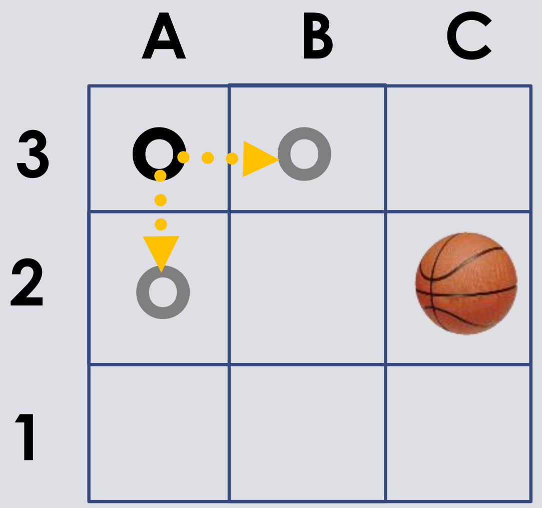
Search tree:

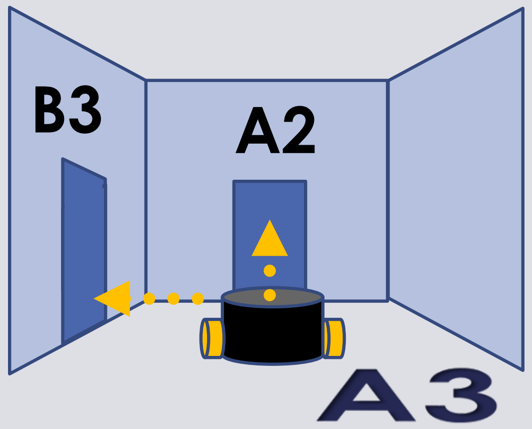


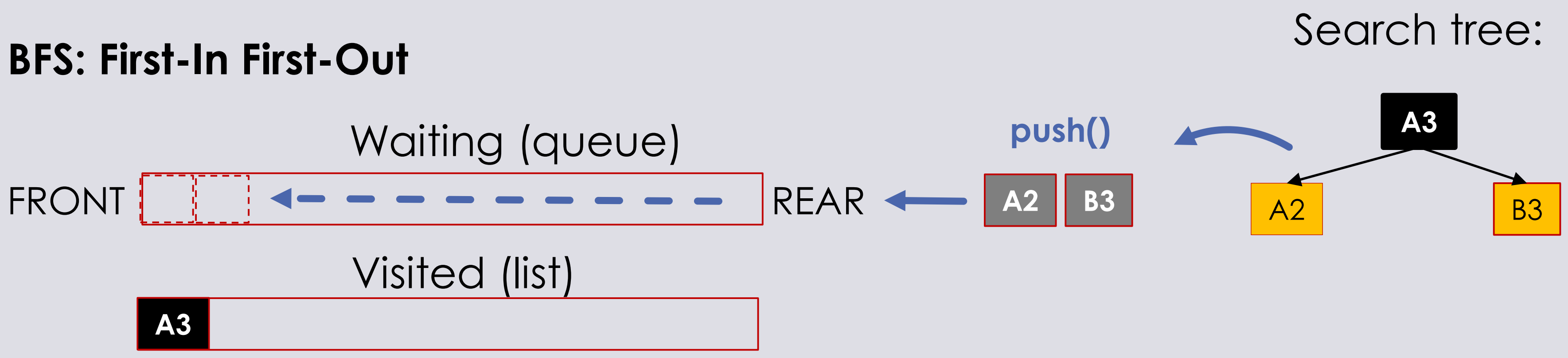
Recap: BFS

Breadth-First Search (BFS)

- FIFO queue ADT



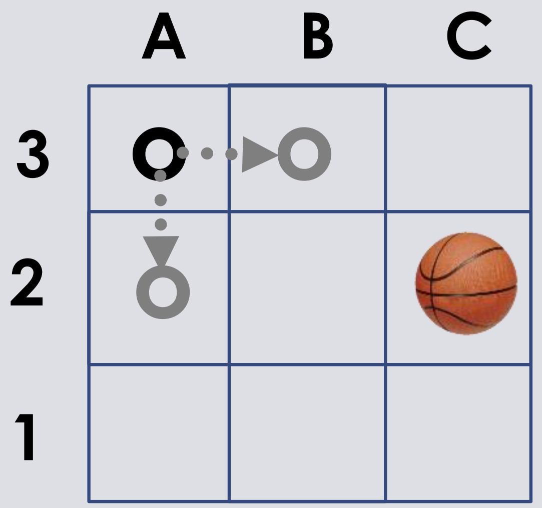


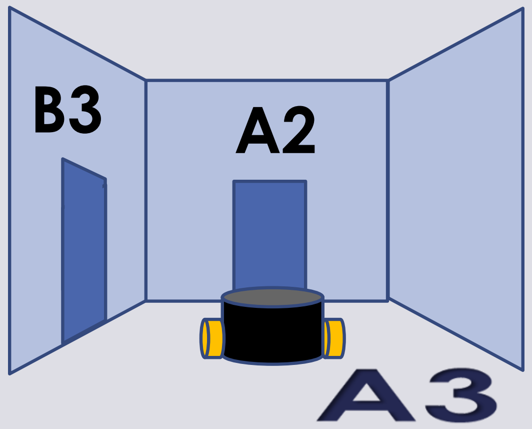


Recap: BFS

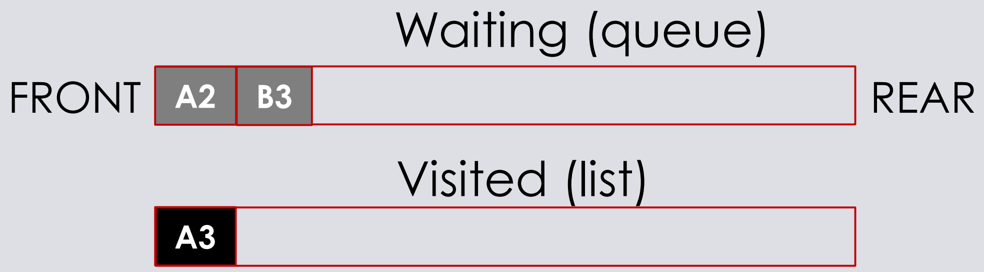
Breadth-First Search (BFS)

- FIFO queue ADT

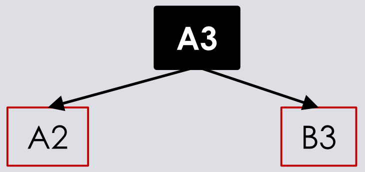




BFS: First-In First-Out



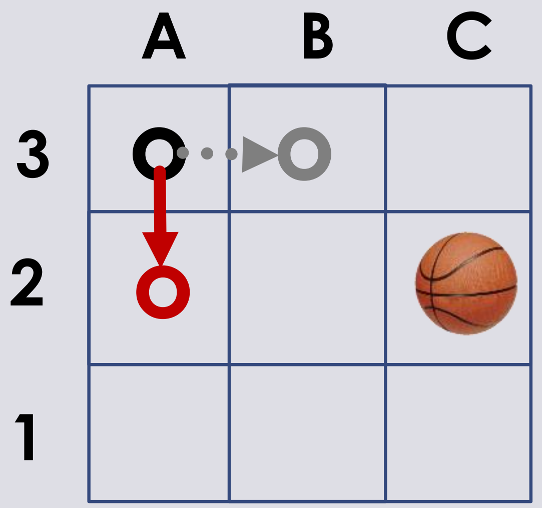
Search tree:

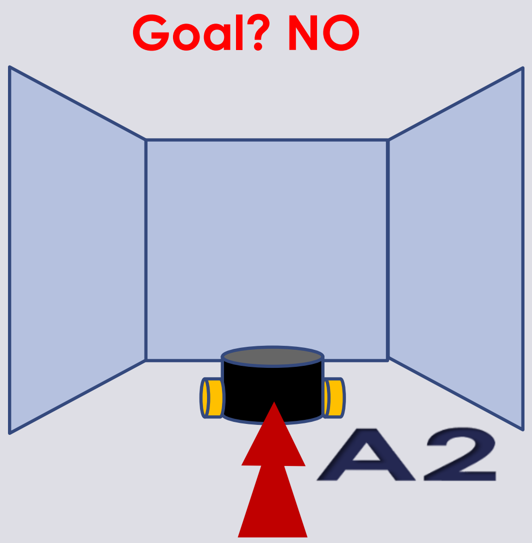


Recap: BFS

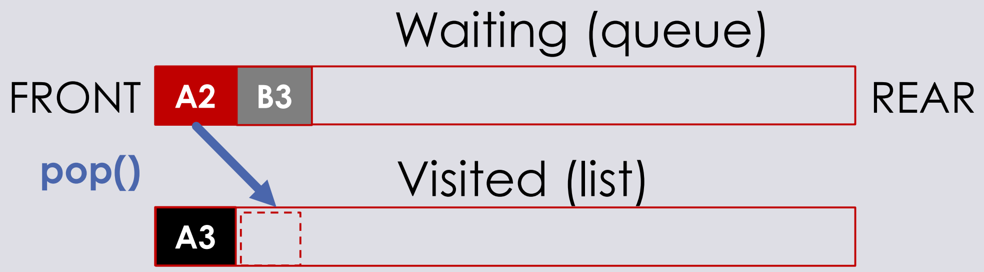
Breadth-First Search (BFS)

- FIFO queue ADT

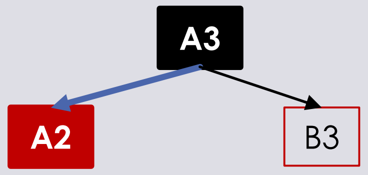




BFS: First-In First-Out



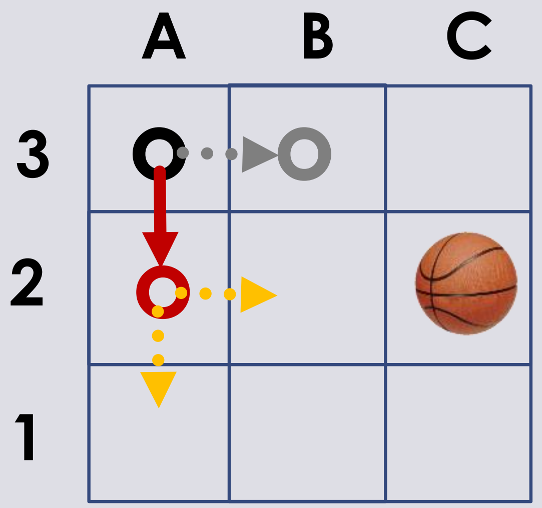
Search tree:

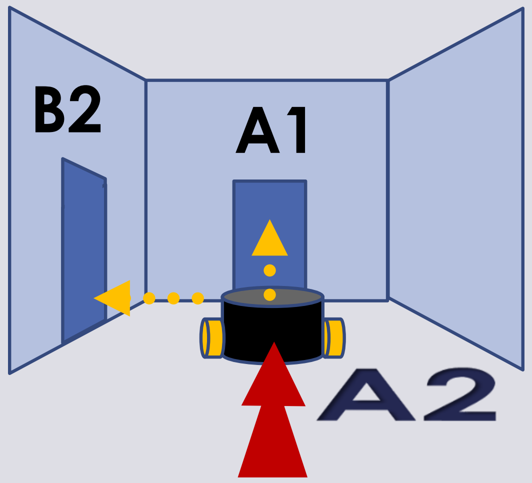


Recap: BFS

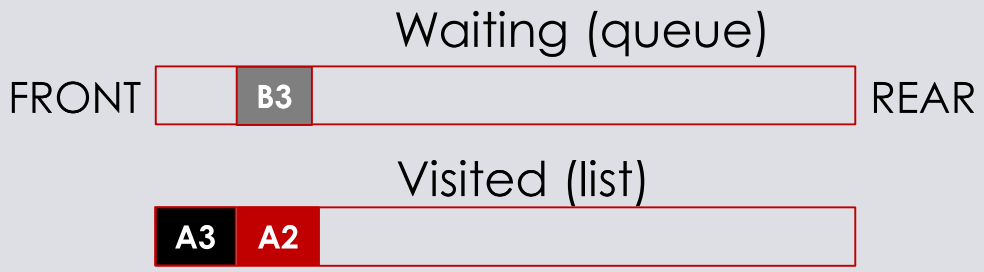
Breadth-First Search (BFS)

- FIFO queue ADT

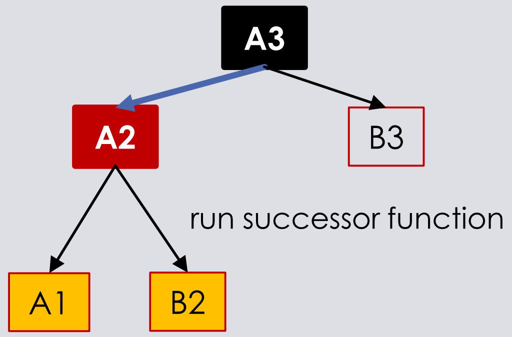




BFS: First-In First-Out



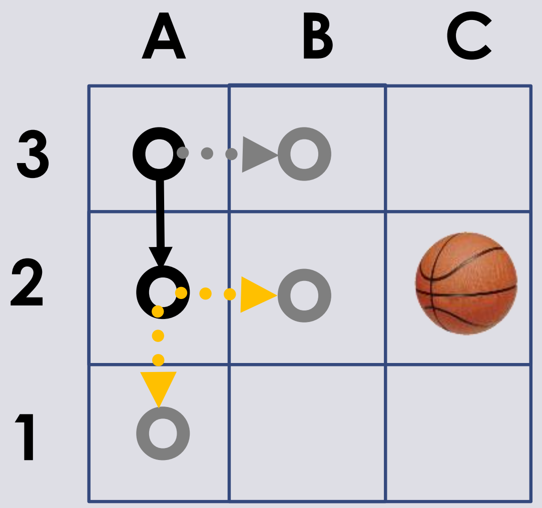
Search tree:

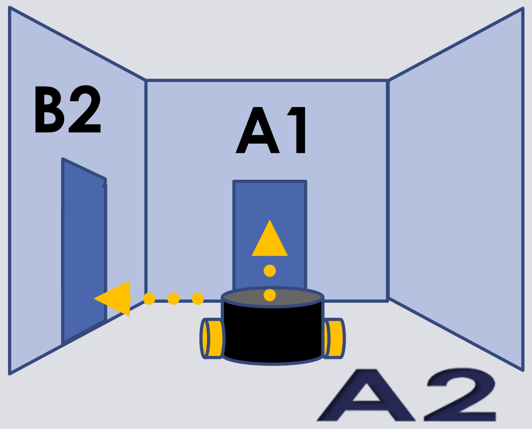


Recap: BFS

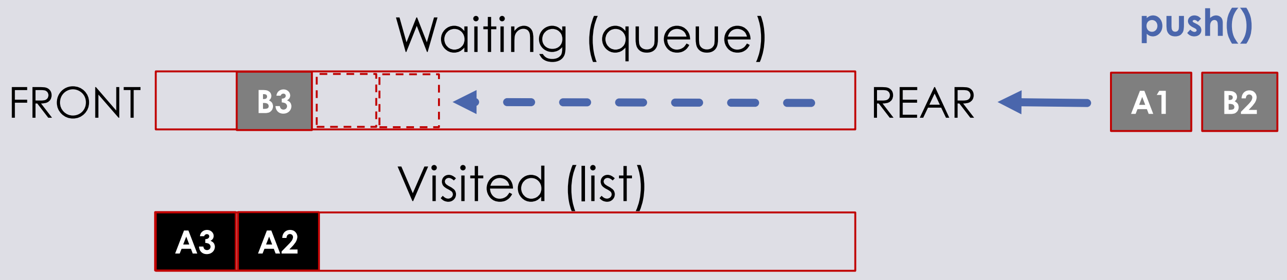
Breadth-First Search (BFS)

- FIFO queue ADT

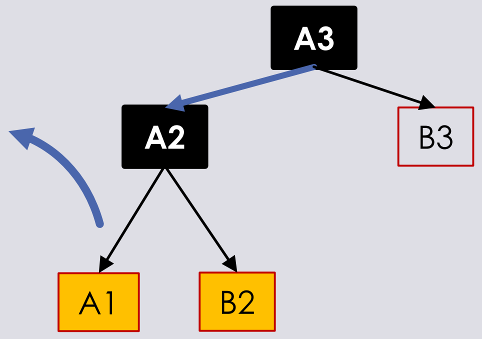




BFS: First-In First-Out



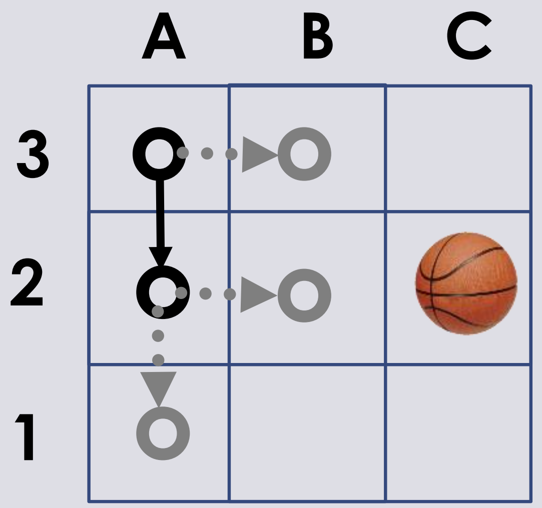
Search tree:

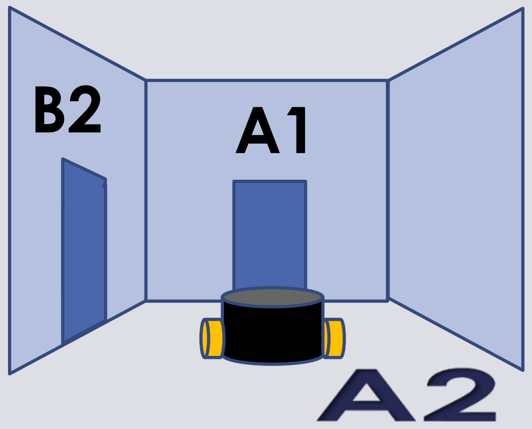


Recap: BFS

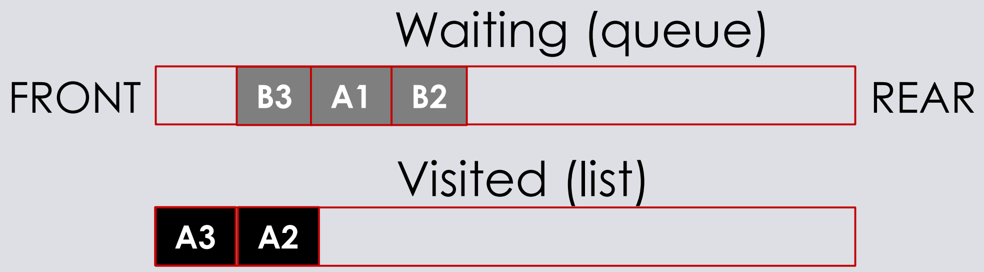
Breadth-First Search (BFS)

- FIFO queue ADT

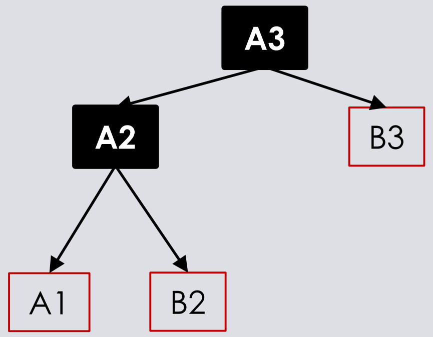




BFS: First-In First-Out



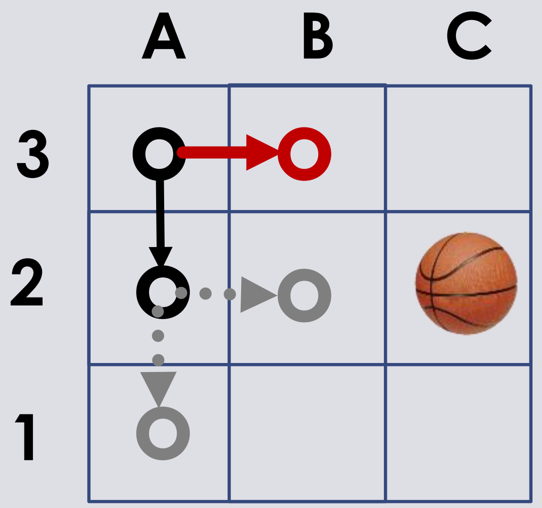
Search tree:

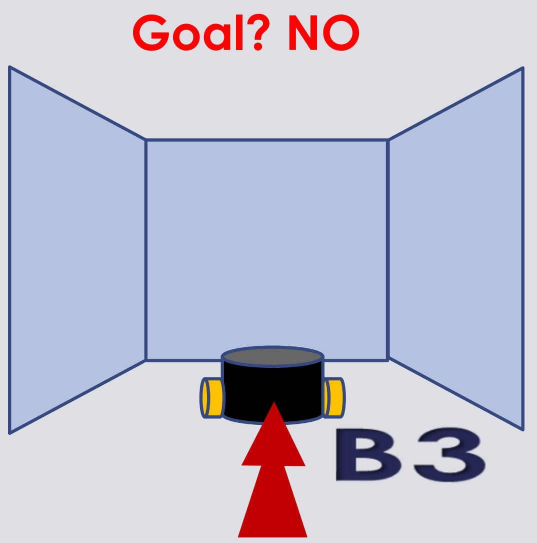


Recap: BFS

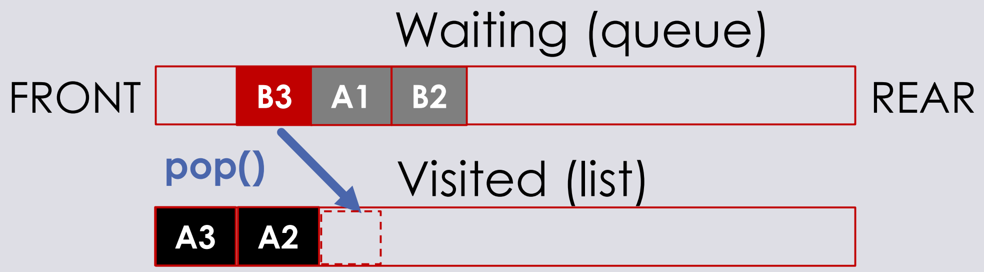
Breadth-First Search (BFS)

- FIFO queue ADT

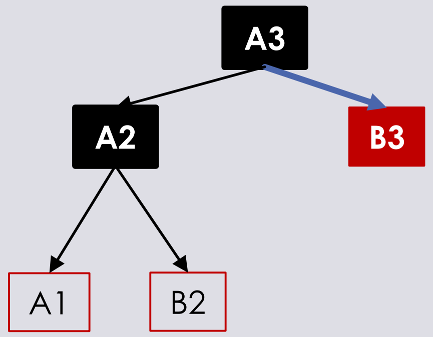




BFS: First-In First-Out



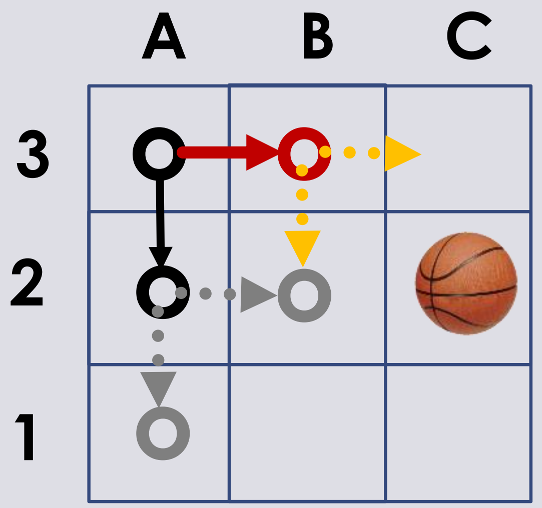
Search tree:

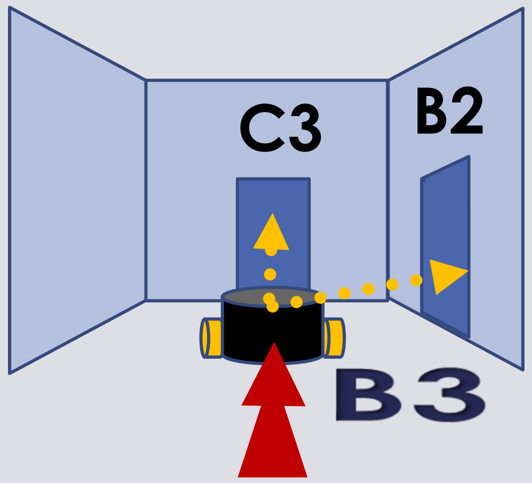


Recap: BFS

Breadth-First Search (BFS)

- FIFO queue ADT





BFS: First-In First-Out

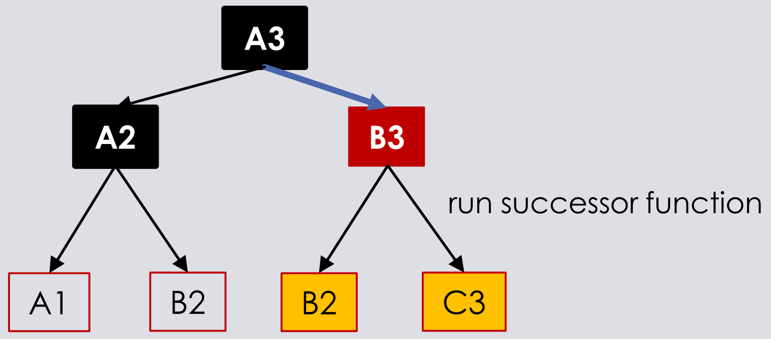
Waiting (queue)

FRONT | A1 | B2 | REAR

Visited (list)

A3 | A2 | B3

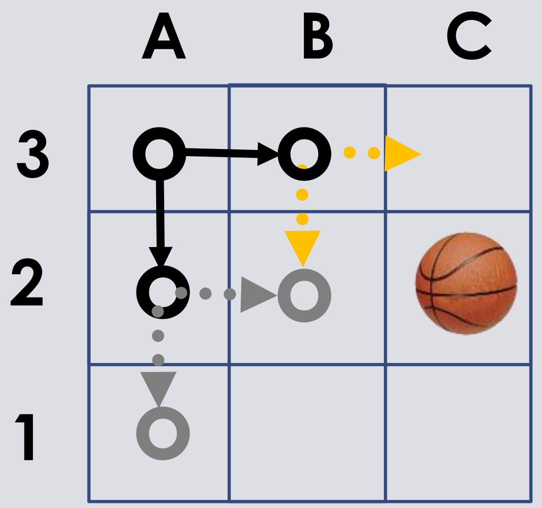
Search tree:

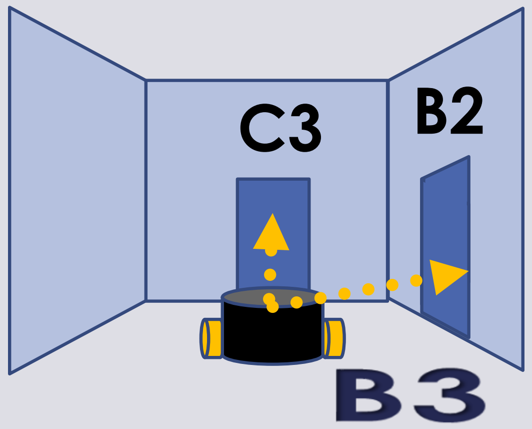


Recap: BFS

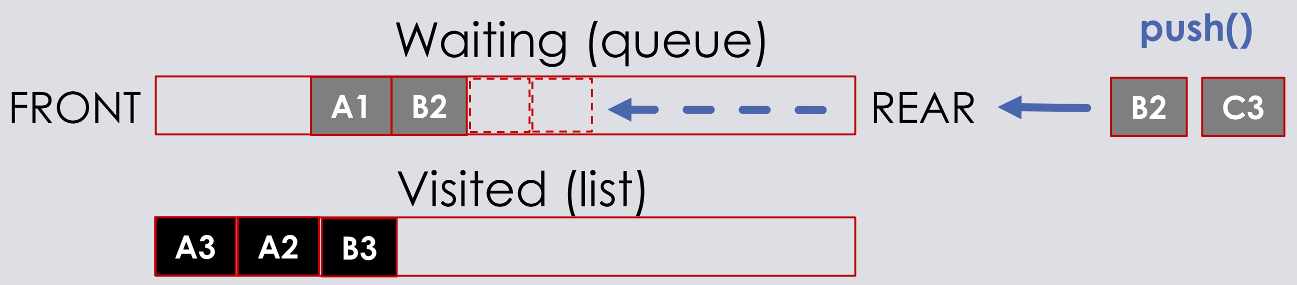
Breadth-First Search (BFS)

- FIFO queue ADT

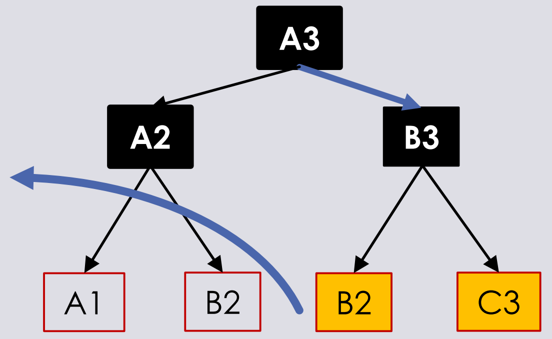




BFS: First-In First-Out



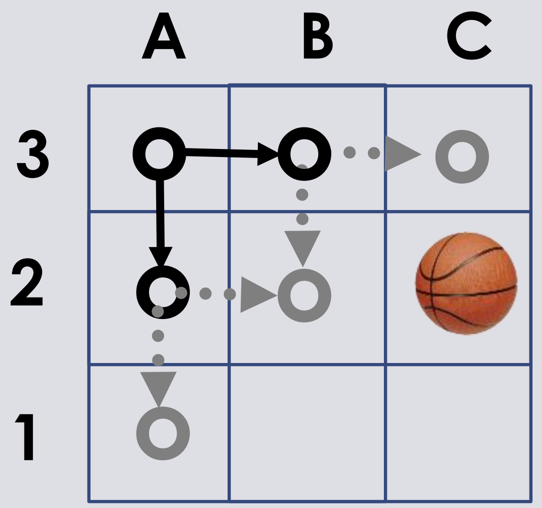
Search tree:

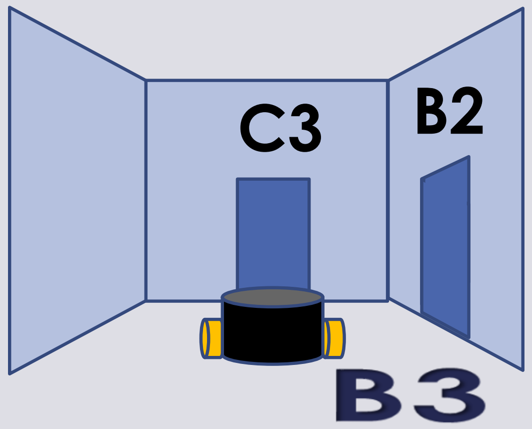


Recap: BFS

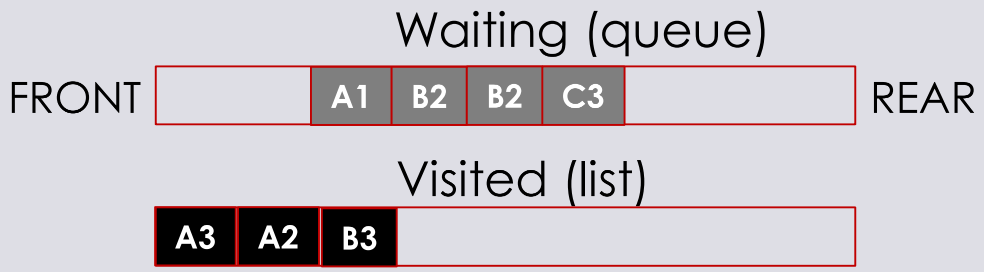
Breadth-First Search (BFS)

- FIFO queue ADT

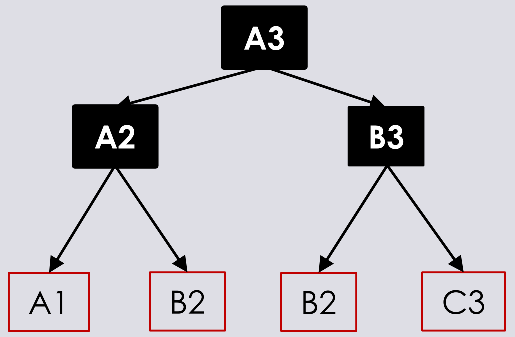




BFS: First-In First-Out



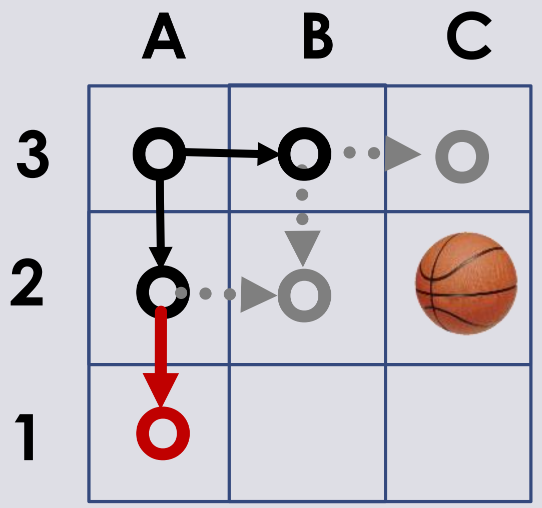
Search tree:

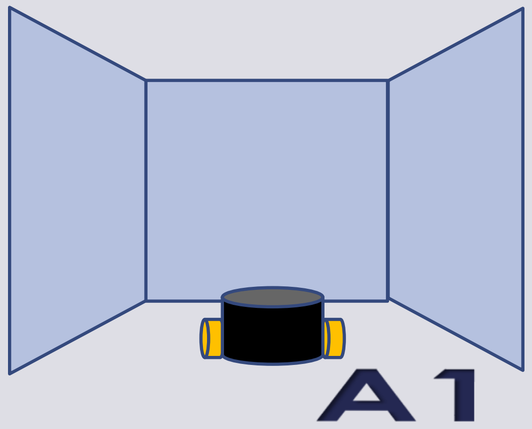


Recap: BFS

Breadth-First Search (BFS)

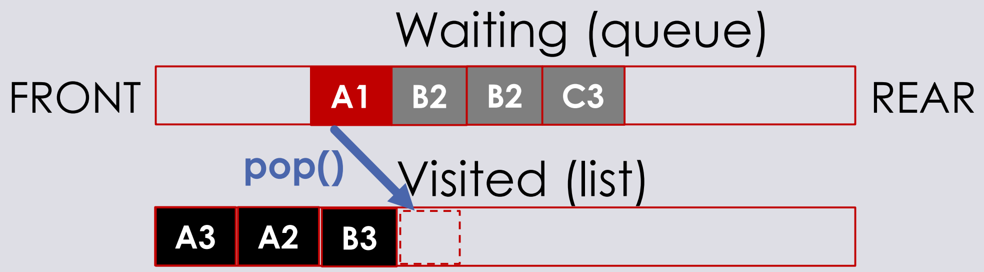
- FIFO queue ADT



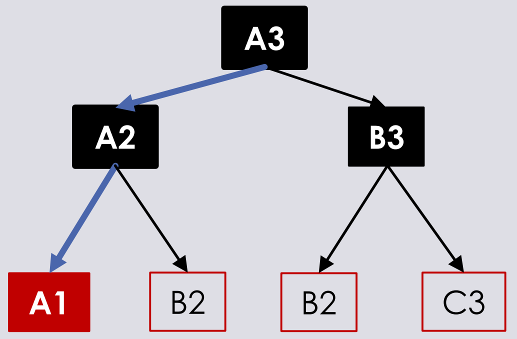


BFS: First-In First-Out

and so on....



Search tree:



Implementing BFS

```
main () {
 // search happens here
 Define initial state -> WAITING queue
 while(states in WAITING) {
  get first state from WAITING
  state -> VISITED
  do a goal test
  if state is not a goal, call successor function
  if goal, print solution
 }
}
```

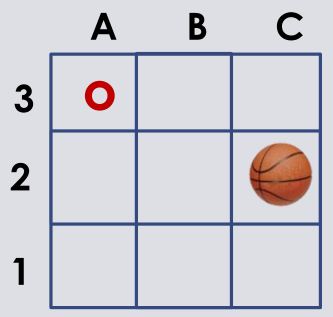
```
successor_function(state) {
 // transition model here
 create a successor of state
 if state not in VISITED
   successor -> WAITING
 ...
}
```

```
goal_test(state) {
 return 1 if state is a goal
 return 0 if not
}
```

```
print_solution(state) {
 // recursive call to backtrack to the top
 if state != initial state
   print_solution(parent of state)
 print(state);
}
```

Workshop exercise

Create a program called "RoboSearch" that determines the shortest path from A3 to C2, using BFS search.



FIRST, DOWNLOAD THE BFS CODE TEMPLATE FROM CANVAS
THEN, ADDRESS ALL ITEMS THAT HAVE A "TO DO" COMMENT

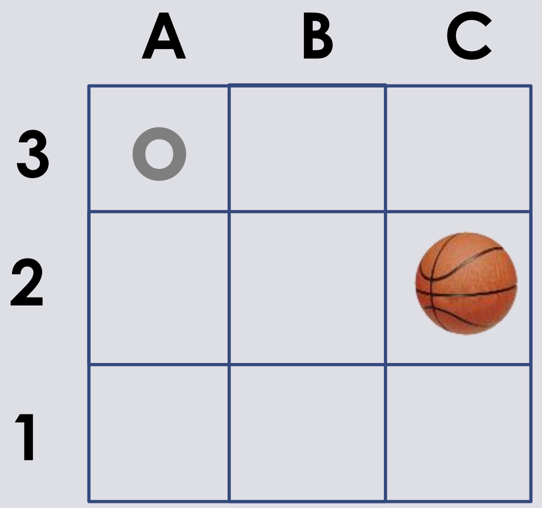
- Find an appropriate state representation, and initialise variables with the ***initial state*** and the ***goal state***.
  (*Tip*: Each room has a label that consists of a letter and an integer)
- Design a ***successor function*** that encodes how the robot can travel between rooms.
  (*Tip*: You can do basic arithmetic not just on integers but also on char data types!)
- Run the search and then print out the first shortest path found, starting from the root node. Optionally, let the search continue to print any further shortest paths found.

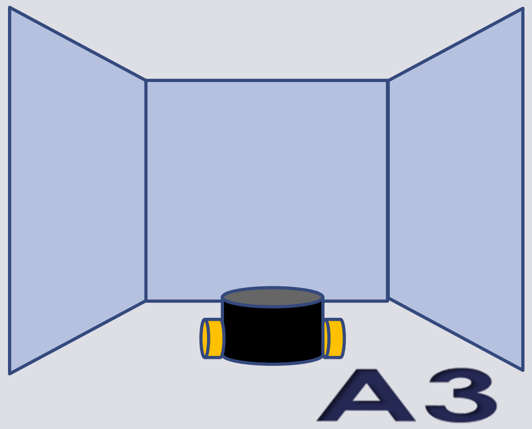
Tip: If stuck, inspect the water-gauging example from our last lab class.

Recap: DFS

Depth-First Search (DFS)

- LIFO queue ADT (Stack)





DFS: Last-In First-Out

Waiting (stack)

FRONT | | REAR

Visited (list)

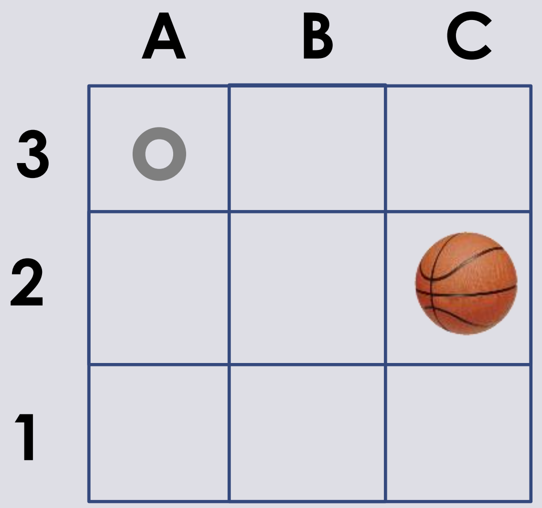
Search tree:

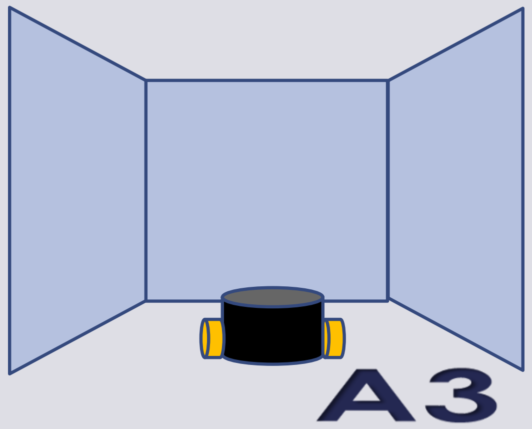
A3

Recap: DFS

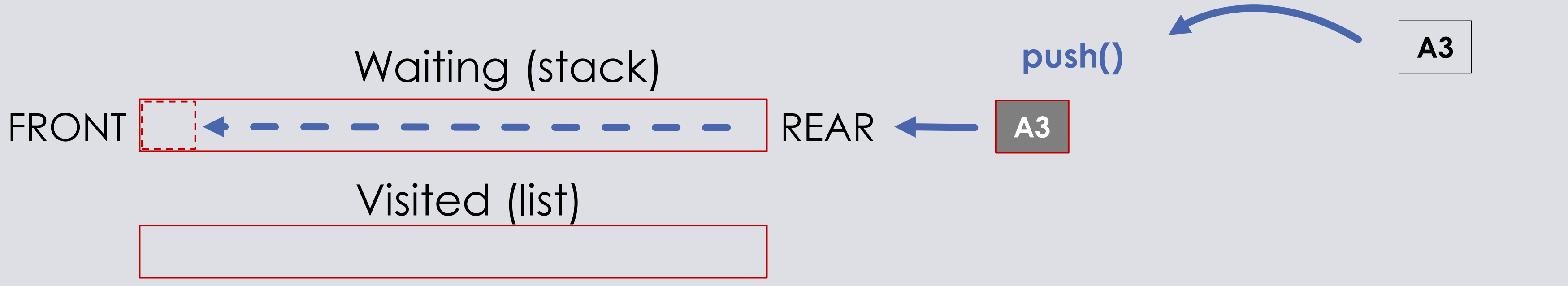
Depth-First Search (DFS)

- LIFO queue ADT (Stack)





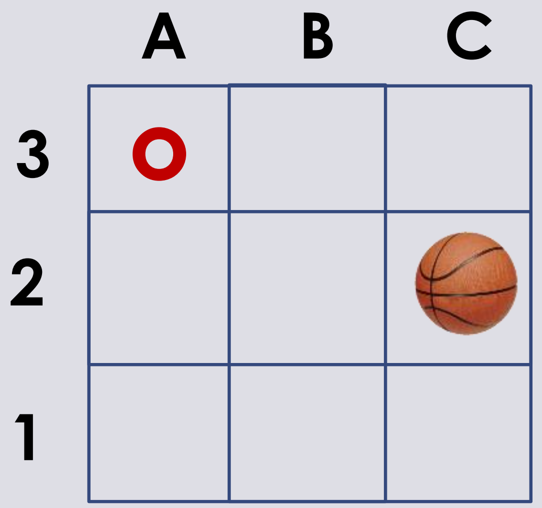
DFS: Last-In First-Out



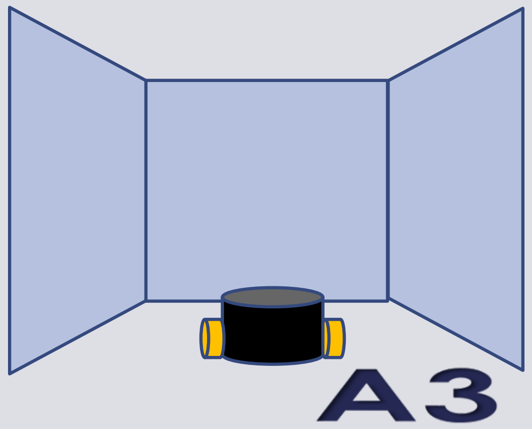
Recap: DFS

Depth-First Search (DFS)

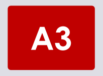
- LIFO queue ADT (Stack)



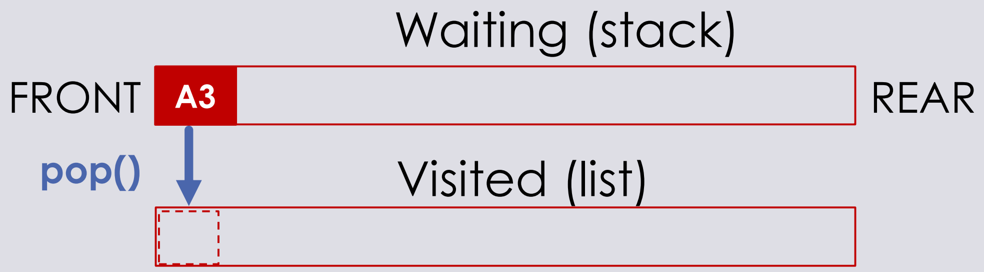
Goal? NO



Search tree:



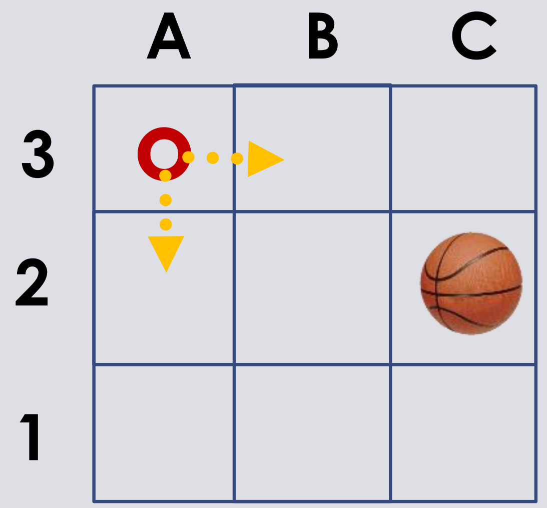
DFS: Last-In First-Out

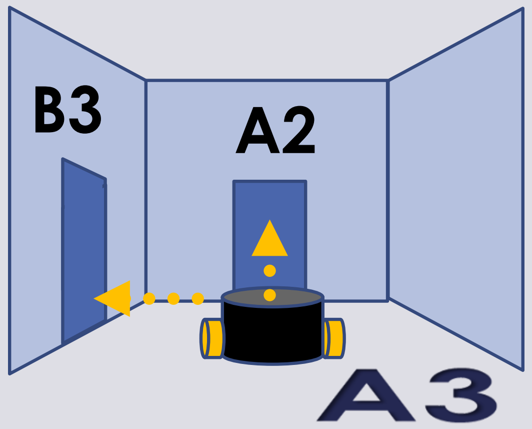


Recap: DFS

Depth-First Search (DFS)

- LIFO queue ADT (Stack)





DFS: Last-In First-Out

Waiting (stack)

FRONT		REAR

Visited (list)

A3	

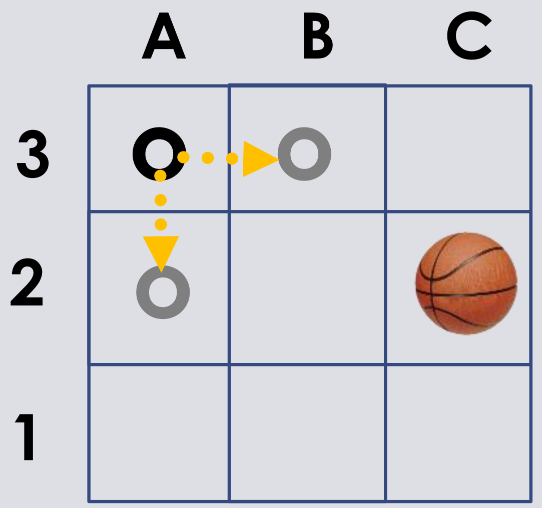
Search tree:

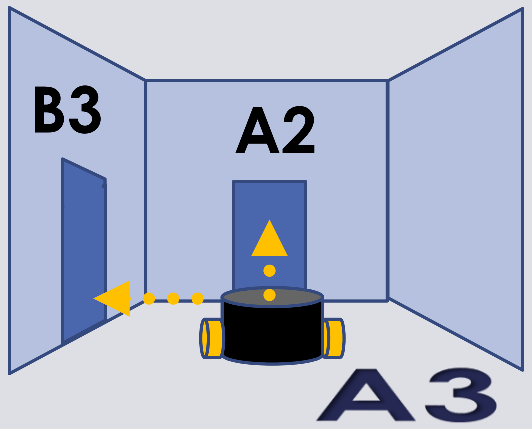
$$\begin{tikzcd} & A3 \arrow[dl] \arrow[dr, "\text{run successor function}"] & \\ A2 & & B3 \end{tikzcd}$$

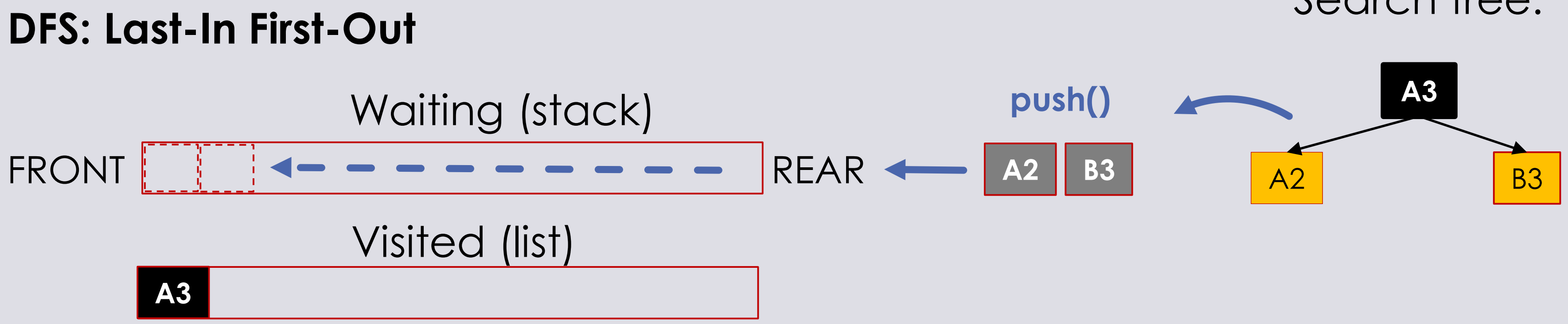
Recap: DFS

Depth-First Search (DFS)

- LIFO queue ADT (Stack)



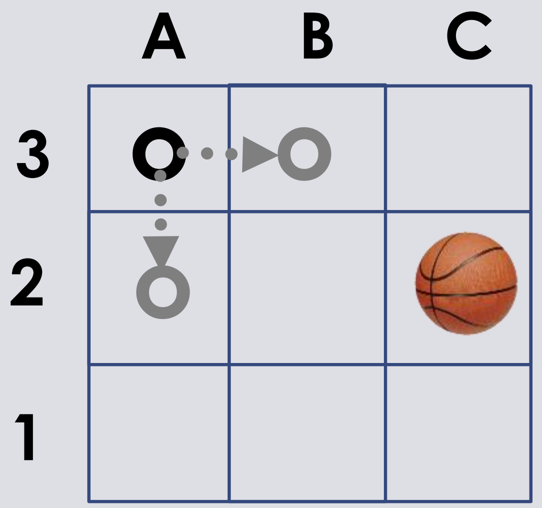


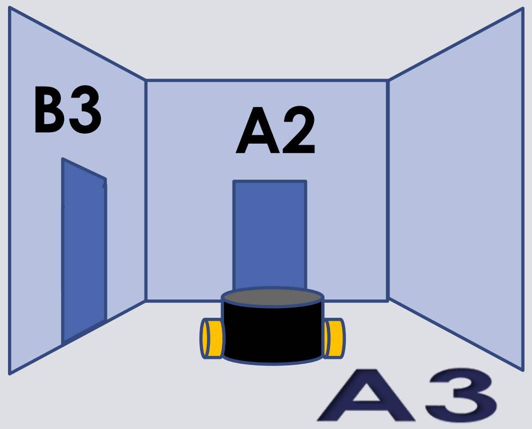


Recap: DFS

Depth-First Search (DFS)

- LIFO queue ADT (Stack)





DFS: Last-In First-Out

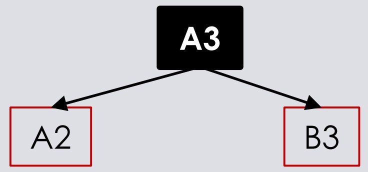
Waiting (stack)

FRONT | A2 | B3 | | REAR

Visited (list)

| A3 | |

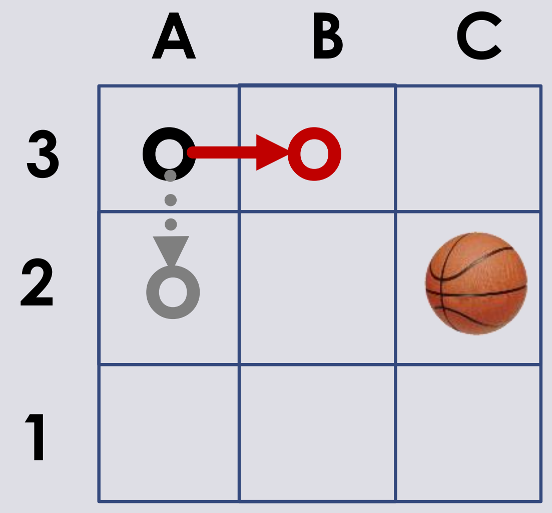
Search tree:



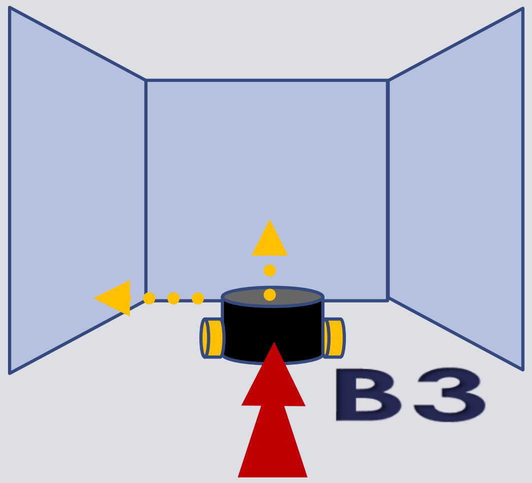
Recap: DFS

Depth-First Search (DFS)

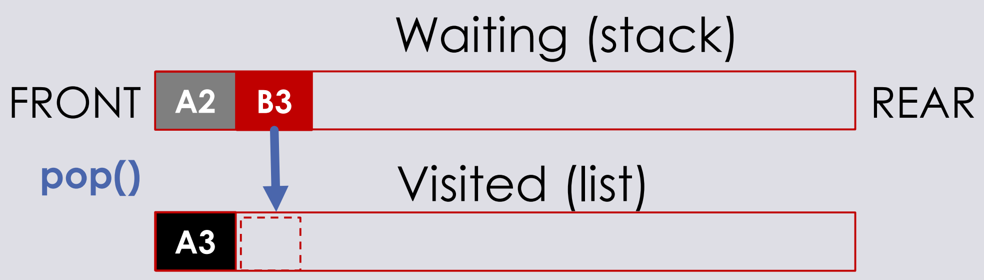
- LIFO queue ADT (Stack)



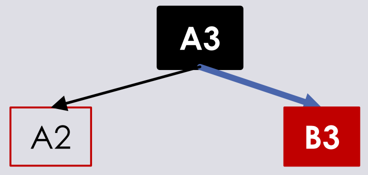
Goal? NO



DFS: Last-In First-Out



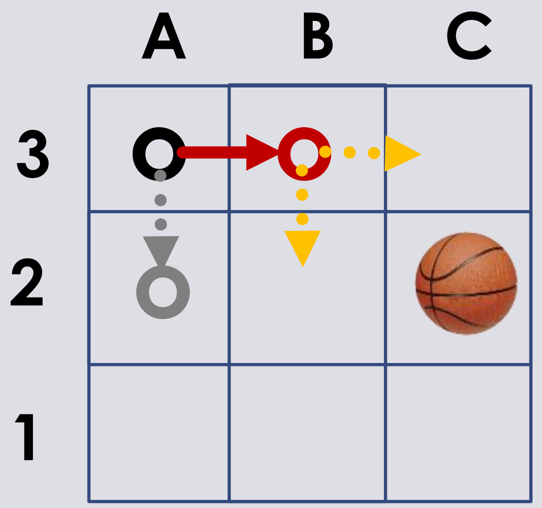
Search tree:

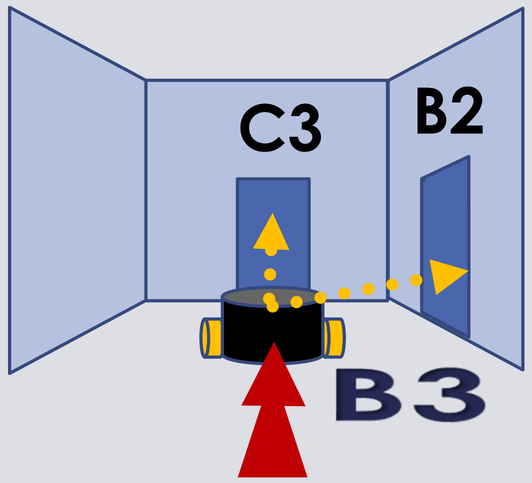


Recap: DFS

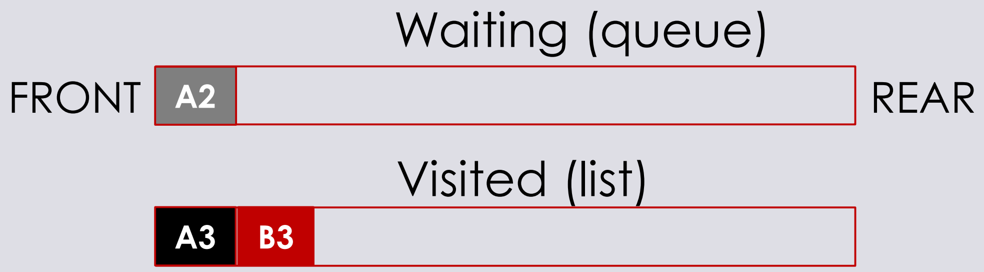
Depth-First Search (DFS)

- LIFO queue ADT (Stack)

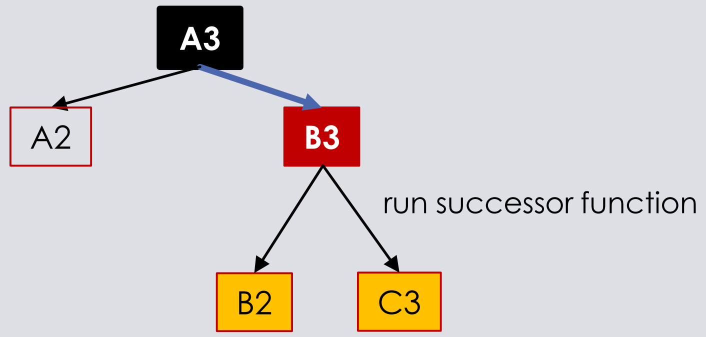




DFS: Last-In First-Out



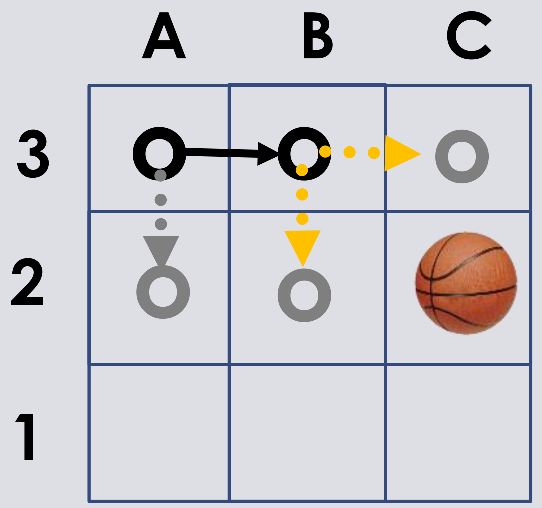
Search tree:

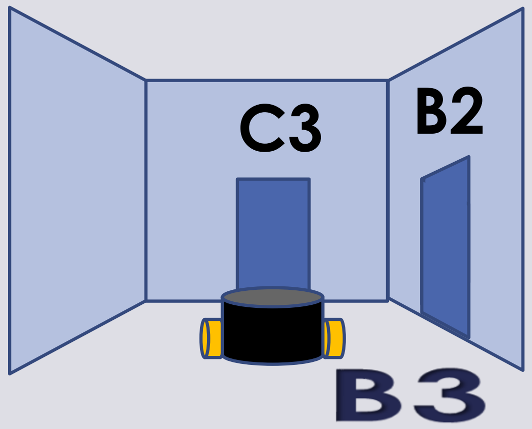


Recap: DFS

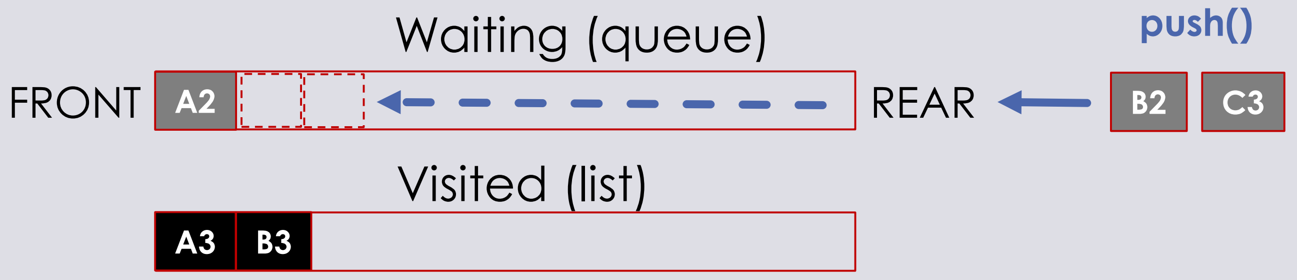
Depth-First Search (DFS)

- LIFO queue ADT (Stack)

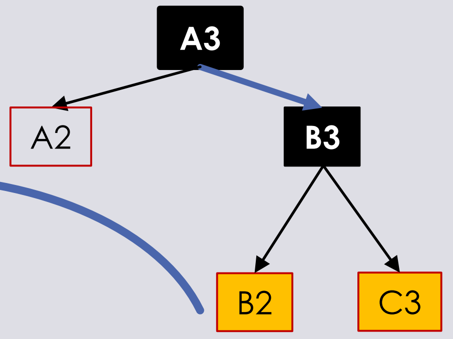




DFS: Last-In First-Out



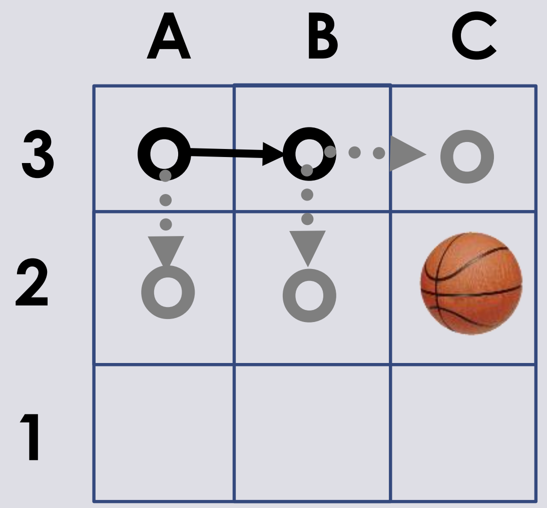
Search tree:

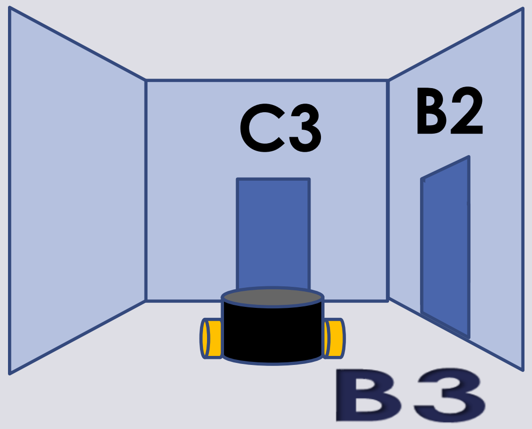


Recap: DFS

Depth-First Search (DFS)

- LIFO queue ADT (Stack)





DFS: Last-In First-Out

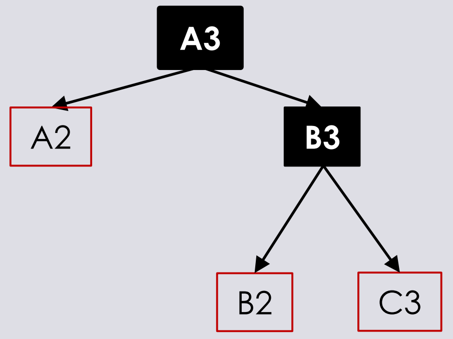
Waiting (queue)

FRONT | A2 | B2 | C3 | | REAR

Visited (list)

| A3 | B3 | |

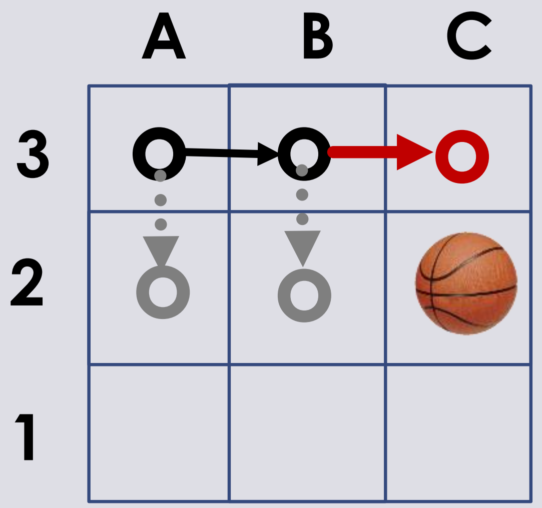
Search tree:

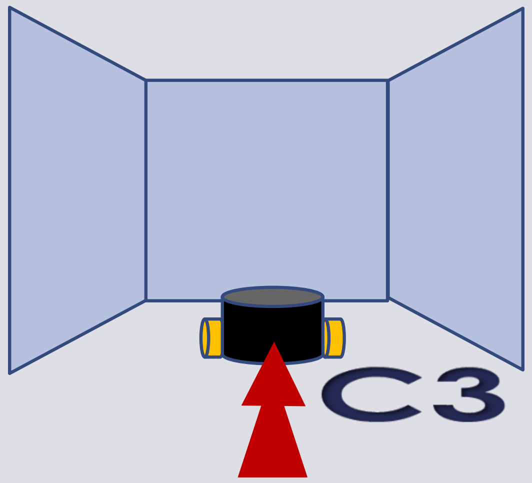


Recap: DFS

Depth-First Search (DFS)

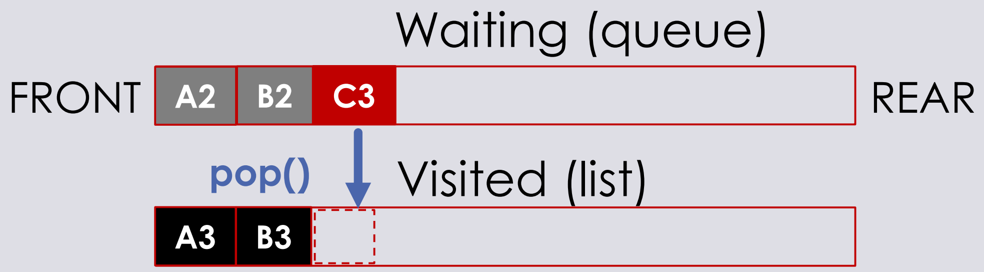
- LIFO queue ADT (Stack)



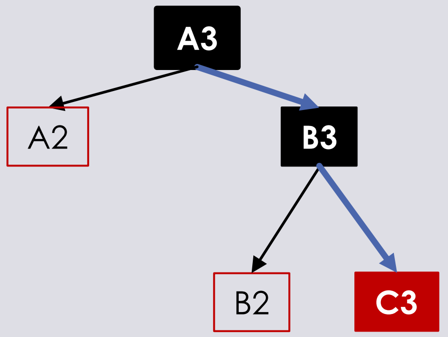


and so on....

DFS: Last-In First-Out



Search tree:



Implementing DFS – one small change

```
main () {
 // search happens here
 Define initial state -> WAITING queue
 while(states in WAITING) {
  get ~~first~~ last state from WAITING
  state -> VISITED
  do a goal test
  if state is not a goal, call successor function
  if goal, print solution
 }
}
```

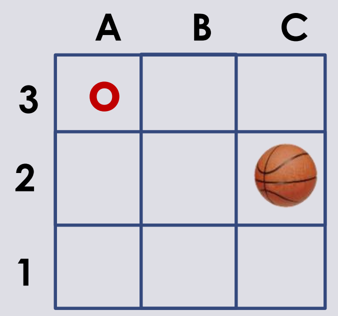
```
successor_function(state) {
 // transition model here
 create a successor of state
 if state not in VISITED
   successor -> WAITING
 ...
}
```

```
goal_test(state) {
 return 1 if state is a goal
 return 0 if not
}
```

```
print_solution(state) {
 // recursive call to backtrack to the top
 if state != initial state
   print_solution(parent of state)
 print(state);
}
```

Workshop exercise

Amend your existing BFS program to use DFS instead.



- Doing this should only require a minor change (two lines of code)
- Does DFS find a shortest path for this particular problem?