

Exercises: Artificial Intelligence

The farmer, fox, goose and grain

Problem

- A farmer has to cross a river with his fox, goose and grain. Each trip, his boat can only carry himself and one of his possessions. How can he cross the river if an unguarded fox eats the goose and an unguarded goose the grain.
 - Find a good representation.
 - Perform Depth-first search (queues)
 - Perform Breadth-first search (search tree)

Farmer, Fox, Goose and Grain

PROBLEM REPRESENTATION

Representation

- States of the form $[\mathcal{L}|\mathcal{R}]$, where:
 - \mathcal{L} : *Items on left bank*
 - \mathcal{R} : *Items on right bank*
- \mathcal{L} and \mathcal{R} contain:
 - Fa: *Farmer*
 - Fo: *Fox*
 - Go: *Goose*
 - Gr: *Grain*

Representation

- Start: [Fa Fo Go Gr |]
- Goal: [| Fa Fo Go Gr]
- Rules:
 - $R_1: [Fa \mathcal{X} | \mathcal{Y}] \longrightarrow [\mathcal{X} | Fa \mathcal{Y}]$
 - $R_2: [\mathcal{X} | Fa \mathcal{Y}] \longrightarrow [Fa \mathcal{X} | \mathcal{Y}]$
 - $R_3: [Fa z \mathcal{X} | \mathcal{Y}] \longrightarrow [\mathcal{X} | Fa z \mathcal{Y}]$
 - $R_4: [\mathcal{X} | Fa z \mathcal{Y}] \longrightarrow [Fa z \mathcal{X} | \mathcal{Y}]$
 - No combination (Fo,Go) or (Go,Gr) on either bank, without the farmer.

Farmer, Fox, Goose and Grain

DEPTH-FIRST SEARCH

Depth-first search (queues)

- ***Input:***
 - **QUEUE:** Path only containing root
- ***Algorithm:***
 - **WHILE** (QUEUE not empty && goal not reached) **DO**
 - Remove first path from QUEUE
 - Create paths to all children
 - Reject paths with loops
 - Add paths to ***front*** of QUEUE
 - **IF** goal reached
 - **THEN** success
 - **ELSE** failure

Depth-first search (queues)

- Start = (\langle [Fa Fo Go Gr|] \rangle)

Depth-first search (queues)

- $S = (\langle [\mathbf{Fa\ Fo\ Go\ Gr\ |}] \rangle)$
 - Paths to Children:
 - $R_3: \langle_{[\mathbf{Fa\ Fo\ Go\ Gr\ |}]} [\mathbf{Fo\ Gr\ |}\ \mathbf{Fa\ Go}] \rangle$
- $Q_1 = (\langle_{[\mathbf{Fa\ Fo\ Go\ Gr\ |}]} [\mathbf{Fo\ Gr\ |}\ \mathbf{Fa\ Go}] \rangle)$

Depth-first search (queues)

- $S = (\langle [Fa\ Fo\ Go\ Gr\ |] \rangle)$
- $Q_1 = (\langle_{[Fa\ Fo\ Go\ Gr]} [Fa\ Go] [Fo\ Gr] [Fa\ Go] \rangle)$
 - Paths to Children:
 - $R_2: \langle_{[Fa\ Fo\ Go\ Gr]} [Fo\ Gr] [Fa\ Go] [Fa\ Fo\ Gr] [Go] \rangle$
 - $R_4: \langle_{[Fa\ Fo\ Go\ Gr]} [Fo\ Gr] [Fa\ Go] [Fa\ Fo\ Go\ Gr] \rangle$
- $Q_2 = (\langle_{[Fa\ Fo\ Go\ Gr]} [Fa\ Go] [Fo\ Gr] [Fa\ Go] [Fa\ Fo\ Gr] [Go] \rangle)$

Depth-first search (queues)

- $S = (\langle [Fa\ Fo\ Go\ Gr\ |] \rangle)$
- $Q_1 = (\langle_{[Fa\ Fo\ Go\ Gr\ |]} [Fo\ Gr\ | Fa\ Go] \rangle)$
- $Q_2 = (\langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go]} [Fa\ Fo\ Gr\ | Go] \rangle)$
 - Paths to Children:
 - $R_1: \langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ |Go]} [Fo\ Gr\ | Fa\ Go] \rangle$
 - $R_3: \langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ |Go]} [Gr\ | Fa\ Fo\ Go] \rangle$
 - $R_3: \langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ |Go]} [Fo\ | Fa\ Go\ Gr] \rangle$
- $Q_3 = (\langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ |Go]} [Gr\ | Fa\ Fo\ Go] \rangle, \langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ |Go]} [Fo\ | Fa\ Go\ Gr] \rangle)$

Depth-first search (queues)

- $S = (\langle [Fa\ Fo\ Go\ Gr\ |] \rangle)$
- $Q_1 = (\langle_{[Fa\ Fo\ Go\ Gr]} [Fo\ Gr\ | Fa\ Go] \rangle)$
- $Q_2 = (\langle_{[Fa\ Fo\ Go\ Gr][Fo\ Gr\ | Fa\ Go]} [Fa\ Fo\ Gr\ | Go] \rangle)$
- $Q_3 = (\langle_{[Fa\ Fo\ Go\ Gr][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go]} [Gr\ | Fa\ Fo\ Go] \rangle, \langle_{[Fa\ Fo\ Go\ Gr][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go]} [Fo\ | Fa\ Go\ Gr] \rangle)$
 - Paths to Children:
 - $R_4: \langle_{[Fa\ Fo\ Go\ Gr][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go][Gr\ | Fa\ Fo\ Go]} [Fa\ Fo\ Gr\ | Go] \rangle$
 - $R_4: \langle_{[Fa\ Fo\ Go\ Gr][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go][Gr\ | Fa\ Fo\ Go]} [Fa\ Go\ Gr\ | Fo] \rangle$
- $Q_4 = (\langle_{[Fa\ Fo\ Go\ Gr][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go][Gr\ | Fa\ Fo\ Go]} [Fa\ Go\ Gr\ | Fo] \rangle, \langle_{[Fa\ Fo\ Go\ Gr][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go]} [Fo\ | Fa\ Go\ Gr] \rangle)$

Depth-first search (queues)

- $S = (\langle [Fa\ Fo\ Go\ Gr\ |] \rangle)$
- $Q_1 = (\langle_{[Fa\ Fo\ Go\ Gr\ |]} [Fo\ Gr\ | Fa\ Go] \rangle)$
- $Q_2 = (\langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go]} [Fa\ Fo\ Gr\ | Go] \rangle)$
- $Q_3 = (\langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go]} [Gr\ | Fa\ Fo\ Go] \rangle, \langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go]} [Fo\ | Fa\ Go\ Gr] \rangle)$
- $Q_4 = (\langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go][Gr\ | Fa\ Fo\ Go]} [Fa\ Go\ Gr\ | Fo] \rangle, \langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go]} [Fo\ | Fa\ Go\ Gr] \rangle)$
 - Paths to Children:
 - $R_3: \langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go]} [Gr\ | Fa\ Fo\ Go] \rangle$
 - $R_3: \langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go]} [Go\ | Fa\ Fo\ Gr] \rangle$
- $Q_5 = (\langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go][Gr\ | Fa\ Fo\ Go][Fa\ Go\ Gr\ | Fo]} [Go\ | Fa\ Fo\ Gr] \rangle, \langle_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ | Fa\ Go][Fa\ Fo\ Gr\ | Go]} [Fo\ | Fa\ Go\ Gr] \rangle)$

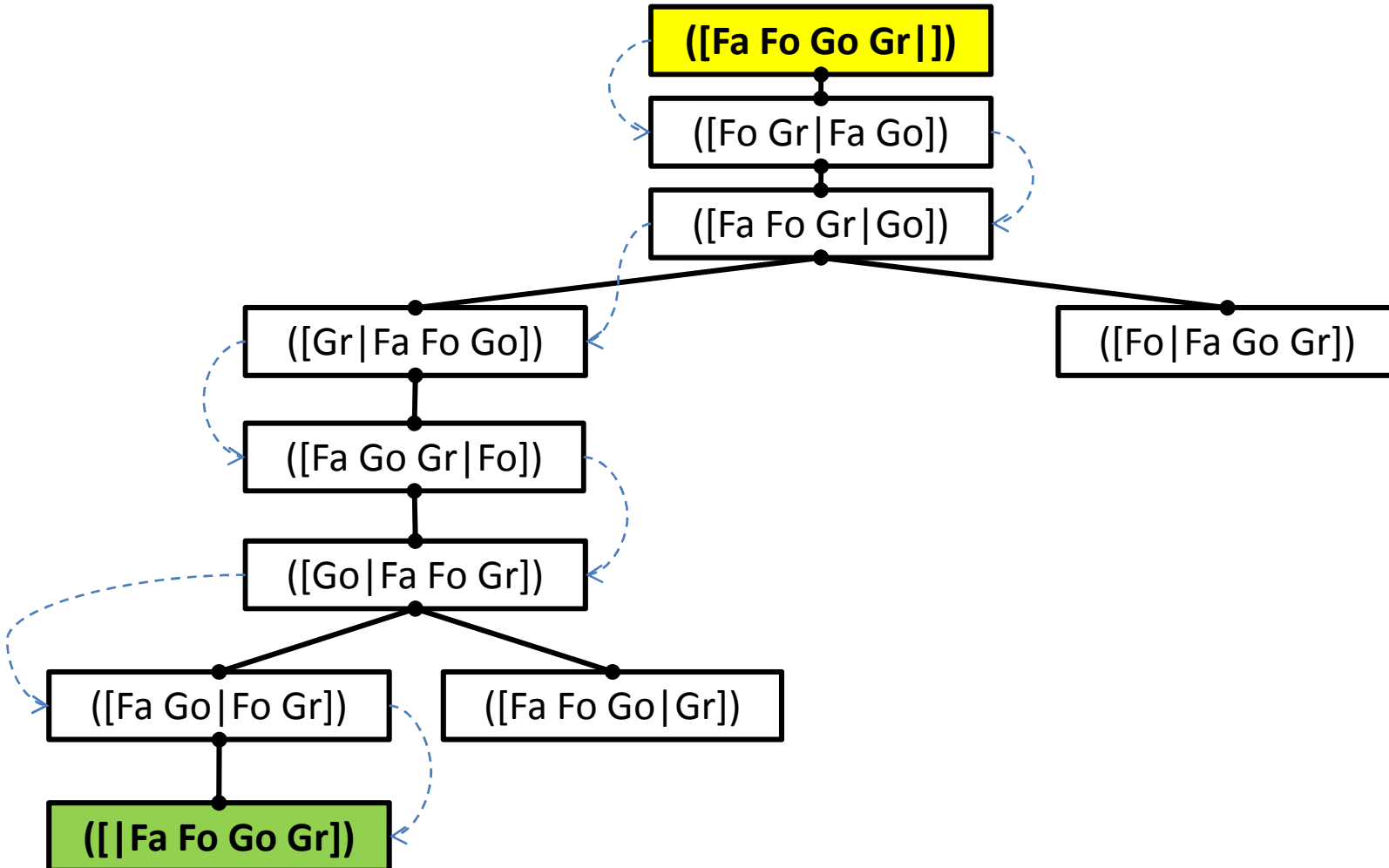
Depth-first search (queues)

- $S = (\langle [Fa\ Fo\ Go\ Gr\] \rangle)$
- $Q_1 = (\langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] \rangle)$
- $Q_2 = (\langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] \rangle)$
- $Q_3 = (\langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Gr\ | Fa\ Fo\ Go] \rangle, \langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Fo\ | Fa\ Go\ Gr] \rangle)$
- $Q_4 = (\langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Gr\ | Fa\ Fo\ Go] [Fa\ Go\ Gr\ | Fo] \rangle, \langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Fo\ | Fa\ Go\ Gr] \rangle)$
- $Q_5 = (\langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Gr\ | Fa\ Fo\ Go] [Fa\ Go\ Gr\ | Fo] [\mathbf{Go\ | Fa\ Fo\ Gr}] \rangle, \langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Fo\ | Fa\ Go\ Gr] \rangle)$
 - Paths to Children:
 - $R_2: \langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Gr\ | Fa\ Fo\ Go] [Fa\ Go\ Gr\ | Fo] [Go\ | Fa\ Fo\ Gr] [Fa\ Go\ | Fo\ Gr] \rangle$
 - $R_4: \langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Gr\ | Fa\ Fo\ Go] [Fa\ Go\ Gr\ | Fo] [Go\ | Fa\ Fo\ Gr] [Fa\ Fo\ Go\ | Gr] \rangle$
 - $R_4: \langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Gr\ | Fa\ Fo\ Go] [Fa\ Go\ Gr\ | Fo] [Go\ | Fa\ Fo\ Gr] [\mathbf{Fa\ Go\ Gr\ | Fo}] \rangle$
- $Q_6 = (\langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Gr\ | Fa\ Fo\ Go] [Fa\ Go\ Gr\ | Fo] [Go\ | Fa\ Fo\ Gr] [\mathbf{Fa\ Go\ | Fo\ Gr}] \rangle, \langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Gr\ | Fa\ Fo\ Go] [Fa\ Go\ Gr\ | Fo] [Go\ | Fa\ Fo\ Gr] [\mathbf{Fa\ Fo\ Go\ | Gr}] \rangle, \langle [Fa\ Fo\ Go\ Gr\] [Fo\ Gr\ | Fa\ Go] [Fa\ Fo\ Gr\ | Go] [Fo\ | Fa\ Go\ Gr] \rangle)$

Depth-first search (queues)

- $S = (\langle [Fa\ Fo\ Go\ Gr\] \rangle)$
- $Q_1 = (\langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go] \rangle)$
- $Q_2 = (\langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go] \rangle)$
- $Q_3 = (\langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go] \rangle, \langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Fo\ |Fa\ Go\ Gr] \rangle)$
- $Q_4 = (\langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo] \rangle, \langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Fo\ |Fa\ Go\ Gr] \rangle)$
- $Q_5 = (\langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr] \rangle, \langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Fo\ |Fa\ Go\ Gr] \rangle)$
- $Q_6 = (\langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Go\ |Fo\ Gr] \rangle, \langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Fo\ Go\ |Gr] \rangle, \langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Fo\ |Fa\ Go\ Gr] \rangle)$
 - Paths to Children:
 - $R_1: \langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Go\ |Fo\ Gr][Go\ |Fa\ Fo\ Gr] \rangle$
 - $R_3: \langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Go\ |Fo\ Gr][Fa\ Fo\ Go\ |Gr] \rangle$
- $G = (\langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Go\ |Fo\ Gr][Fa\ Fo\ Go\ |Gr] \rangle, \langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Fo\ Go\ |Gr] \rangle, \langle [Fa\ Fo\ Go\ Gr][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Fo\ |Fa\ Go\ Gr] \rangle)$

Depth-first search (search tree)



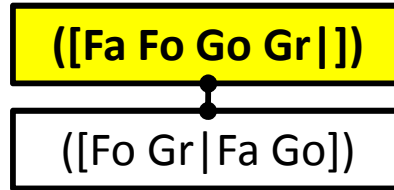
Farmer, Fox, Goose and Grain

BREADTH-FIRST SEARCH

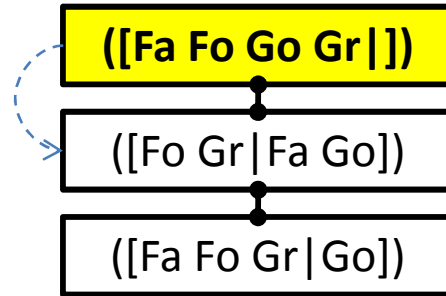
Breadth-first search (queues)

- ***Input:***
 - **QUEUE:** Path only containing root
- ***Algorithm:***
 - **WHILE** (QUEUE not empty && goal not reached) **DO**
 - Remove first path from QUEUE
 - Create paths to all children
 - Reject paths with loops
 - Add paths to ***end*** of QUEUE
 - **IF** goal reached
 - **THEN** success
 - **ELSE** failure

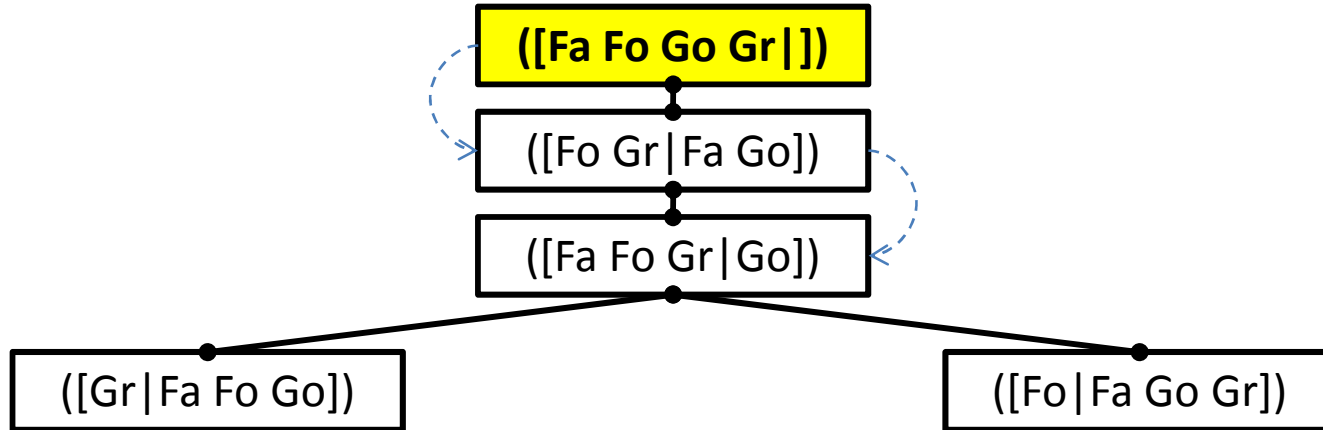
Breadth-first search (search tree)



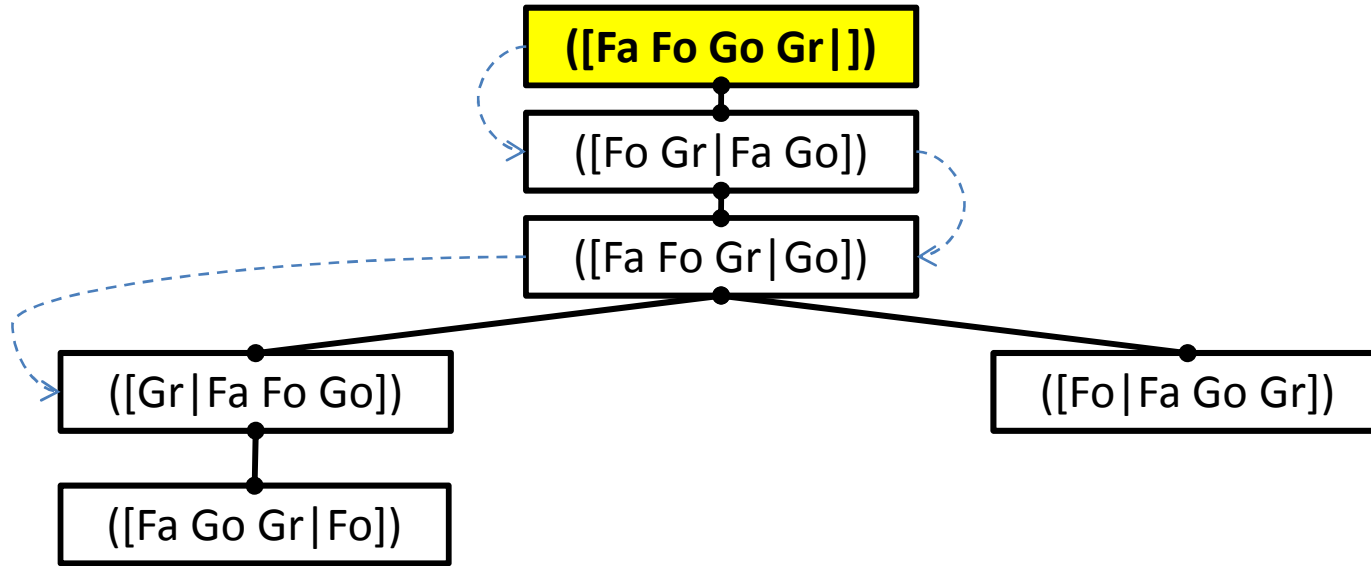
Breadth-first search (search tree)



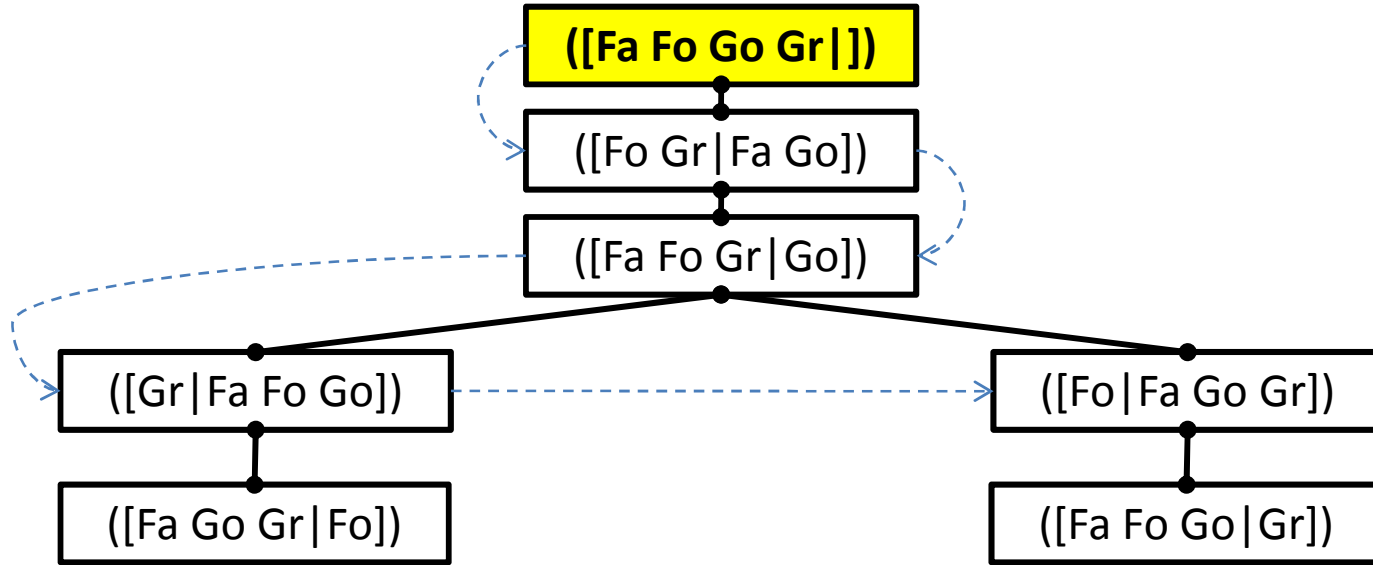
Breadth-first search (search tree)



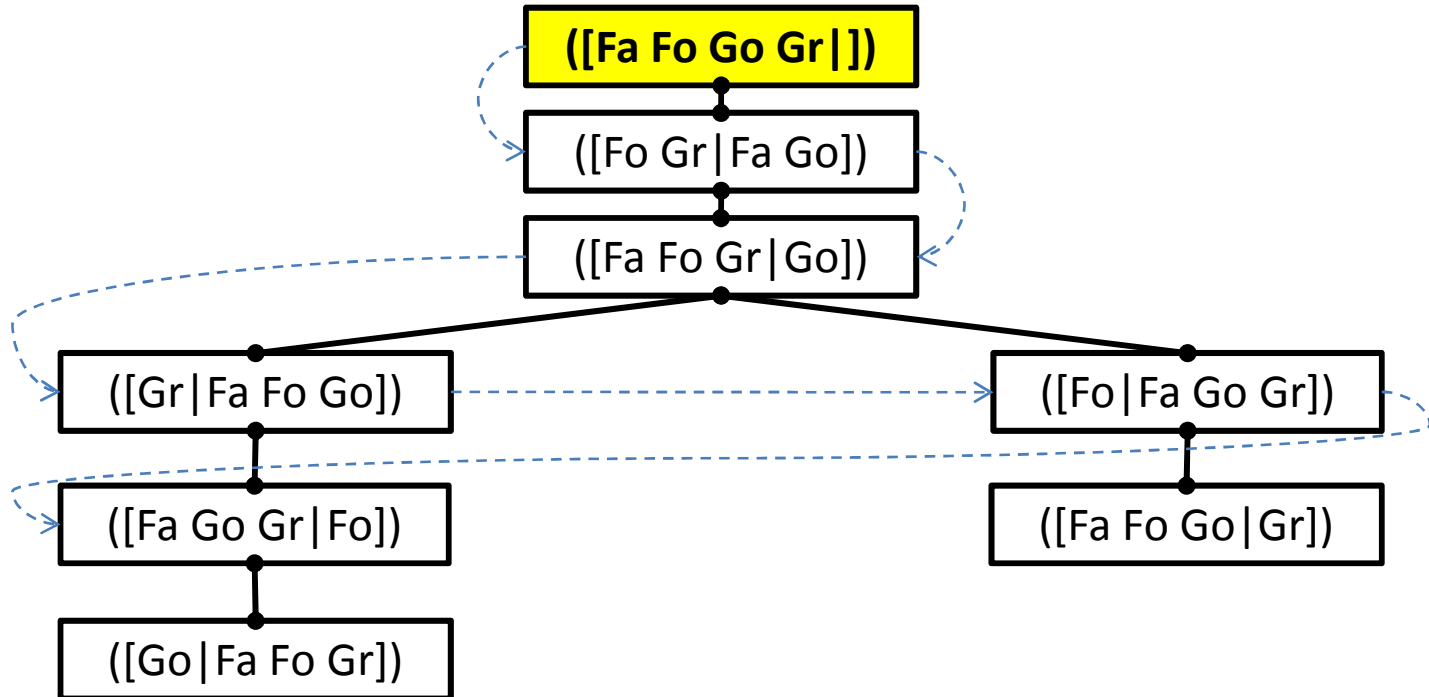
Breadth-first search (search tree)



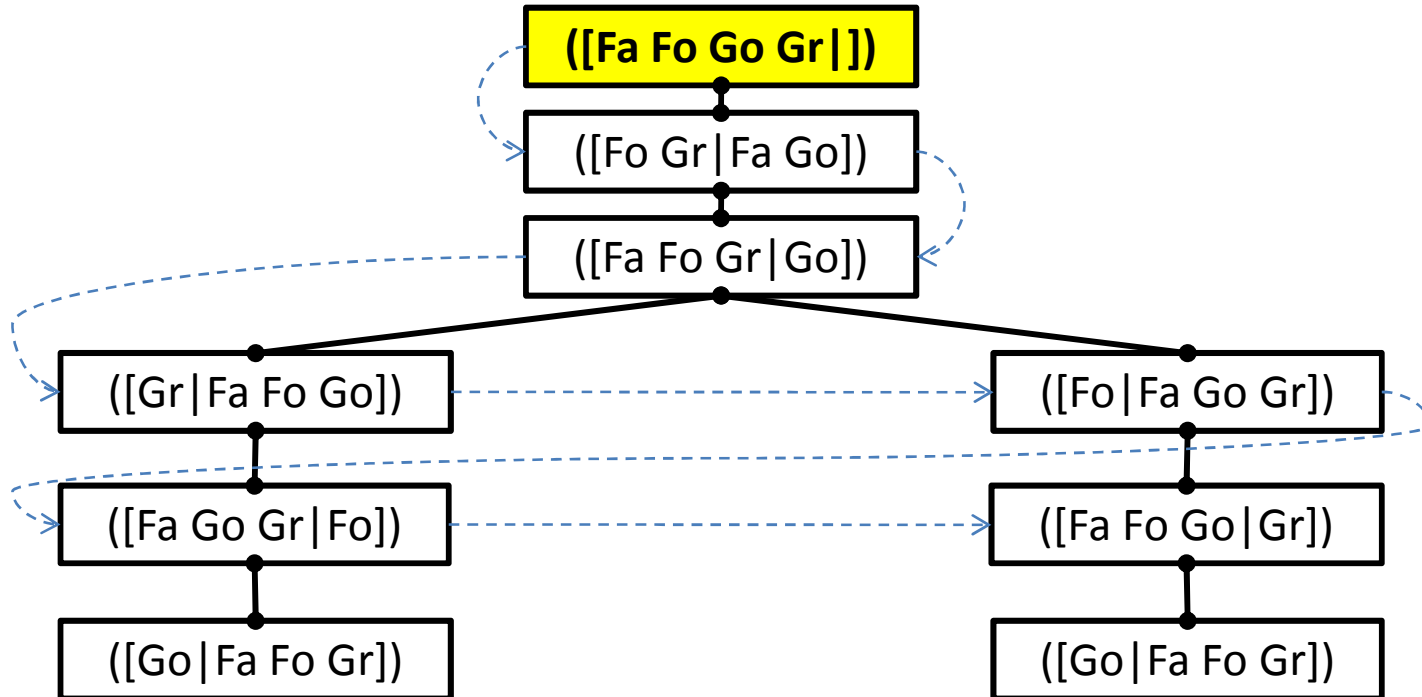
Breadth-first search (search tree)



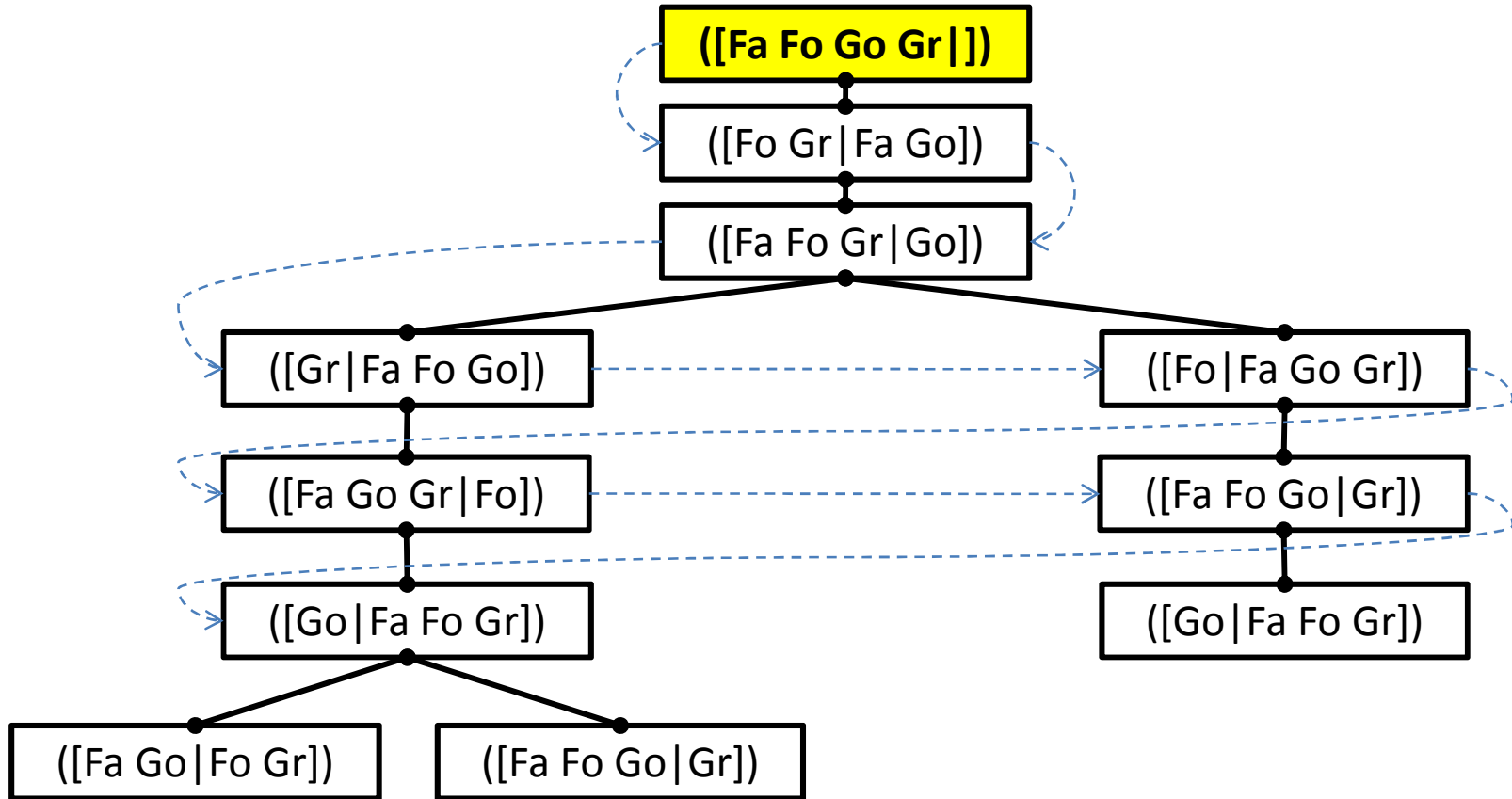
Breadth-first search (search tree)



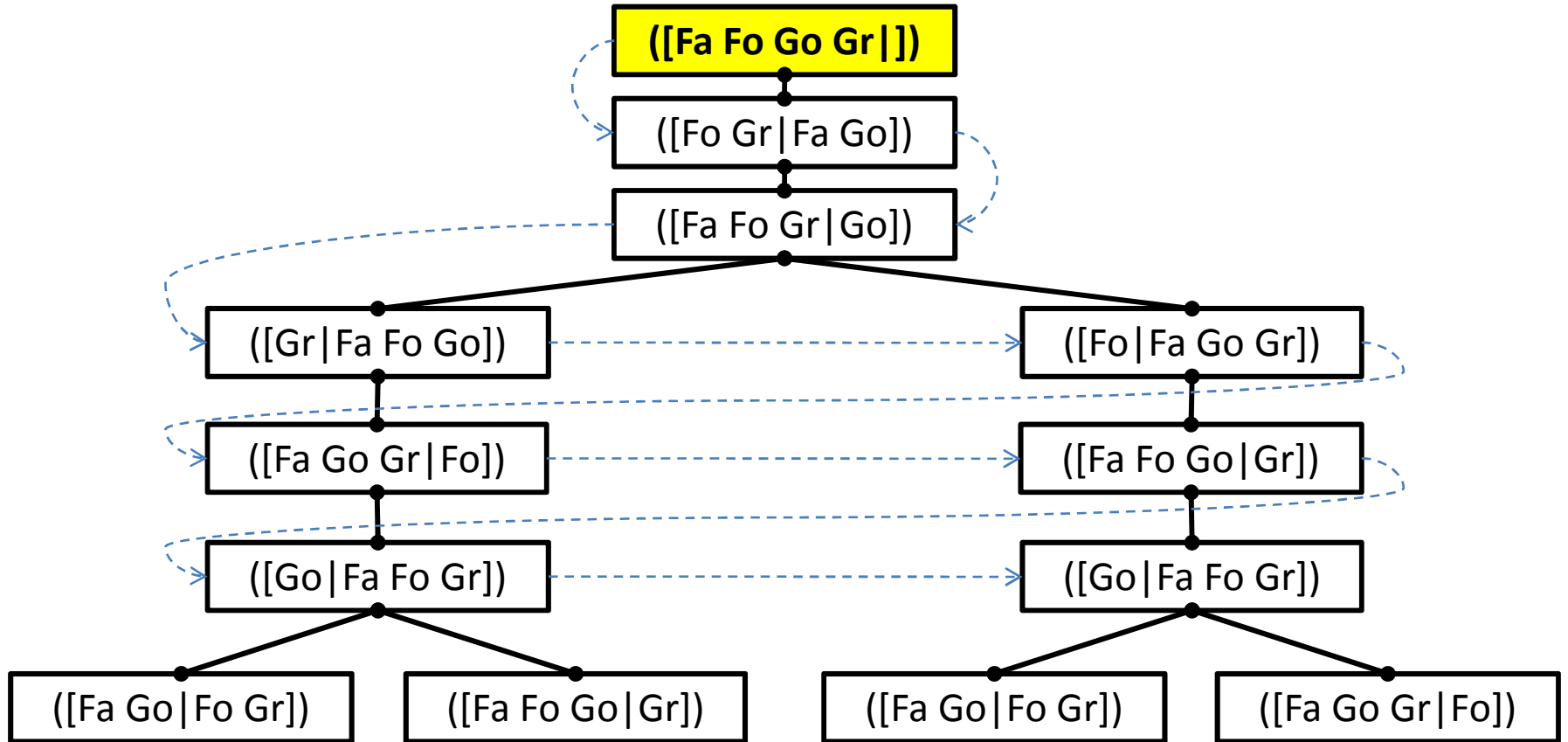
Breadth-first search (search tree)



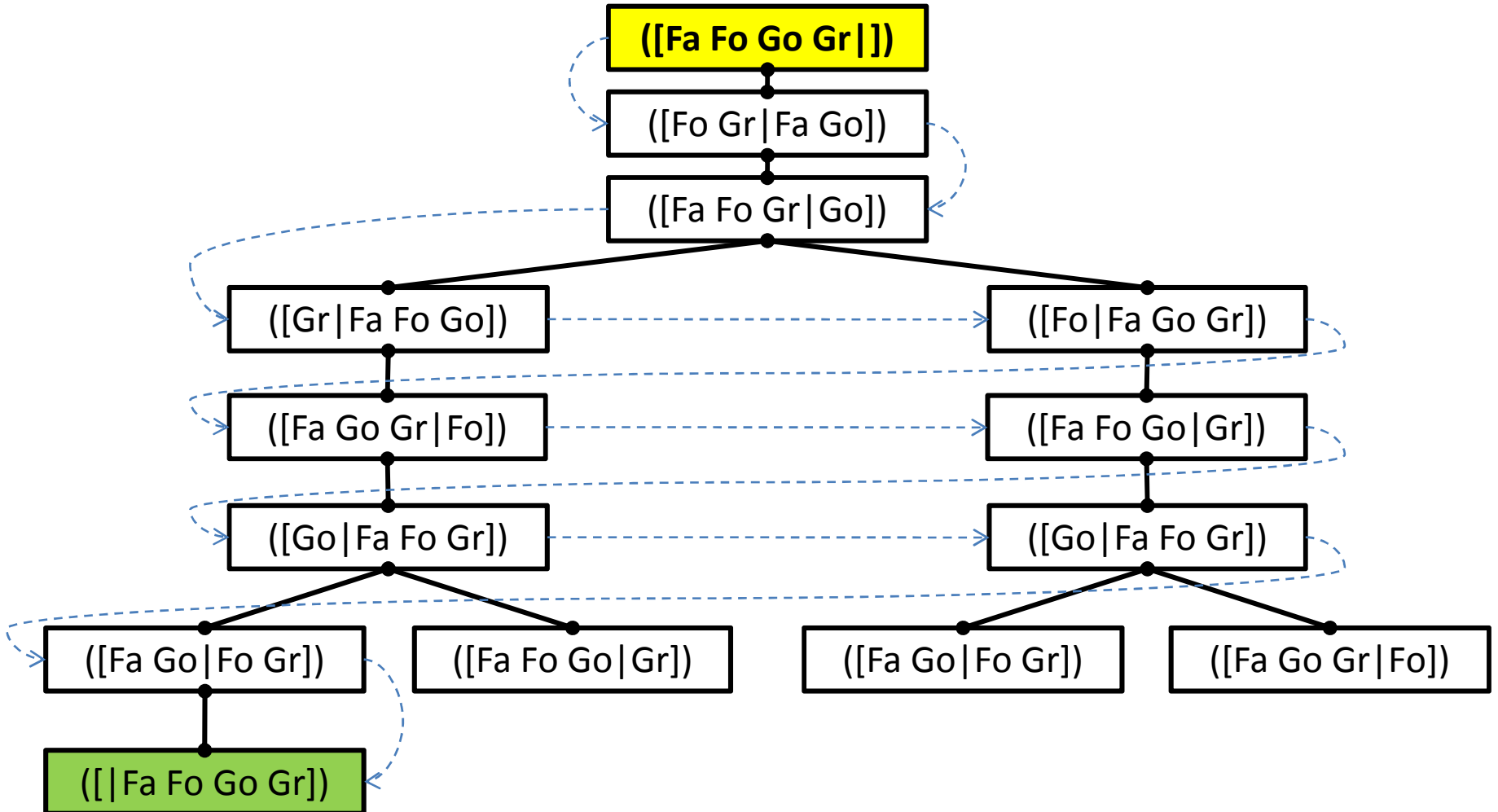
Breadth-first search (search tree)



Breadth-first search (search tree)



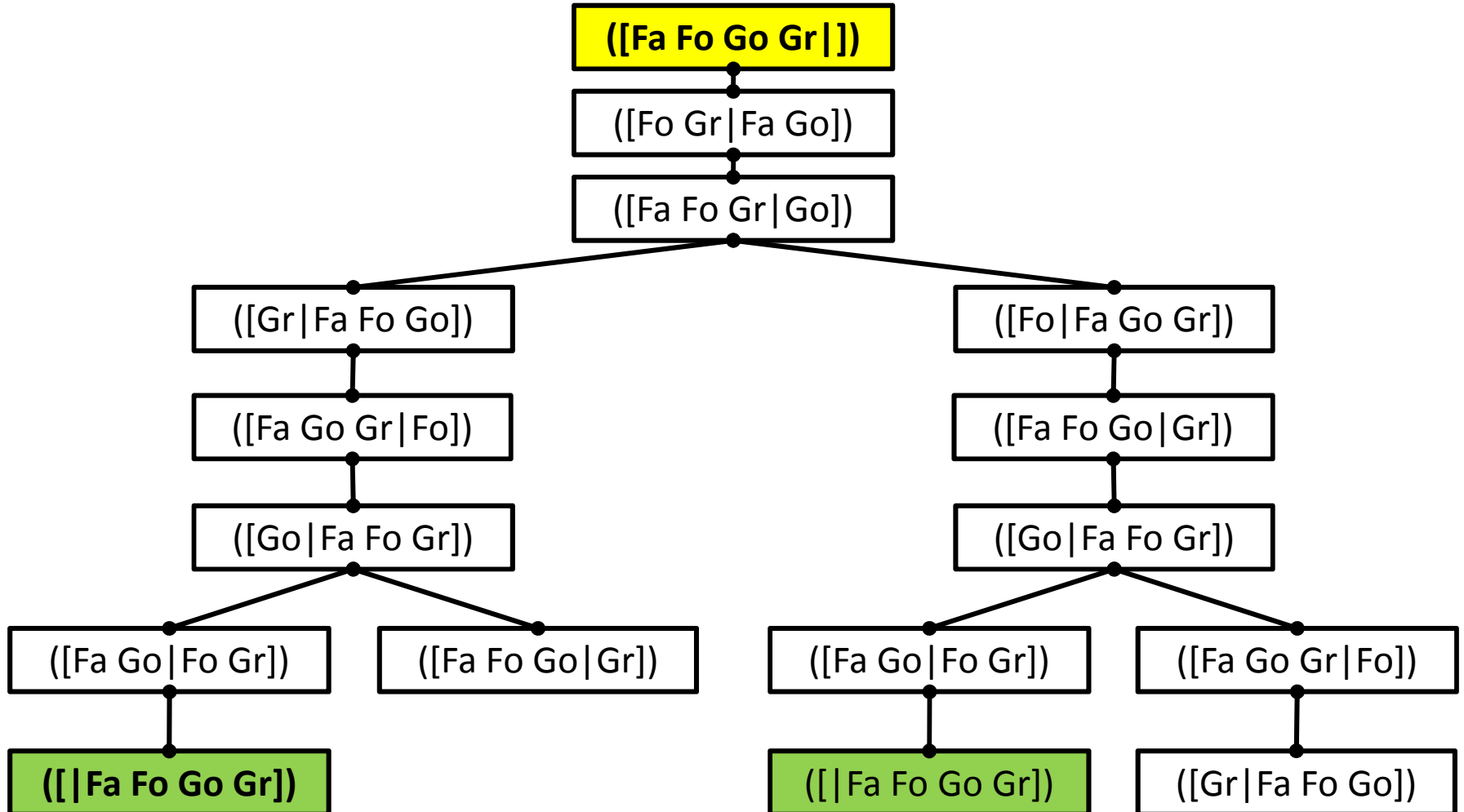
Breadth-first search (search tree)



Farmer, Fox, Goose and Grain

ENTIRE SEARCH TREE

Entire search tree



Exercises: Artificial Intelligence

Bidirectional Search

Problem

- Which methods other than breadth-first can be used in bidirectional search?
 - Is it possible to replace breadth-first for either or both of the forward and backward direction?
- Does the method still work if the check for the shared state is replaced by a check for identical end nodes?

Bidirectional Search

PROBLEM 1: BREADTH-FIRST?

Other methods than 2 x breadth-first

- Bidirectional search is complete for each combination with at least one complete search-strategy.
 - 2 x Breadth-first
 - 2 x Depth-first
 - Breadth-first and Depth-first
- Not each combination benefits from searching at both ends.

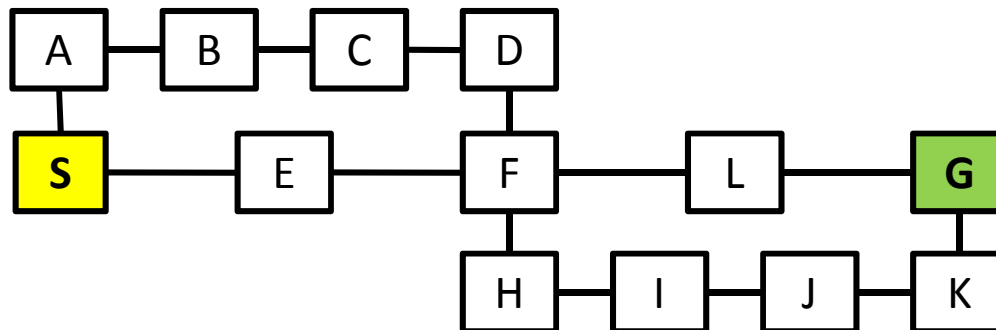
2 x Depth-first

- Forward:

– ($\langle S \rangle$) \rightarrow ($\langle SA \rangle, \langle SE \rangle$) \rightarrow ($\langle SAB \rangle, \langle SE \rangle$) \rightarrow ($\langle SABCD \rangle, \langle SE \rangle$)
 \rightarrow ($\langle \mathbf{SABCDF} \rangle, \langle \mathbf{SE} \rangle$)

- Backward:

– ($\langle G \rangle$) \rightarrow ($\langle GK \rangle, \langle GL \rangle$) \rightarrow ($\langle GKJ \rangle, \langle GL \rangle$) \rightarrow ($\langle GKJI \rangle, \langle GL \rangle$)
 \rightarrow ($\langle \mathbf{GKJIHF} \rangle, \langle \mathbf{GL} \rangle$)



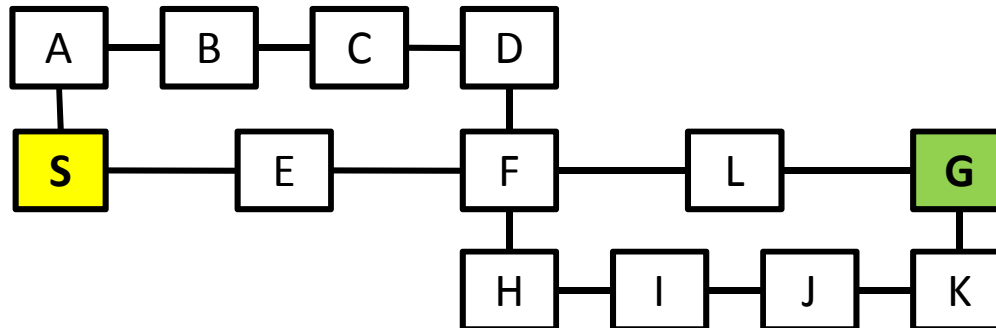
2 x Breadth-first

- Forward:

- ($\langle S \rangle$) \rightarrow ($\langle SA \rangle, \langle SE \rangle$) \rightarrow ($\langle SE \rangle, \langle SAB \rangle$) \rightarrow ($\langle \mathbf{SAB} \rangle, \langle \mathbf{SEF} \rangle$)

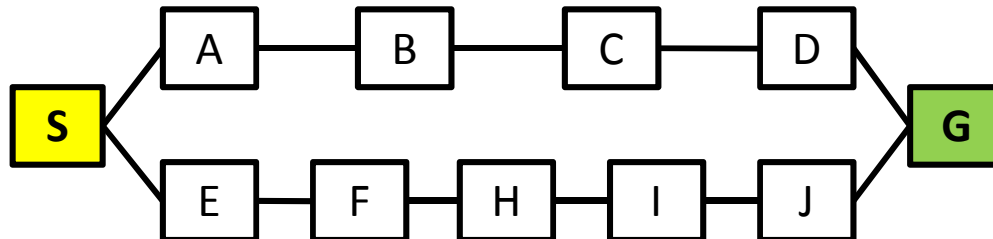
- Backward:

- ($\langle G \rangle$) \rightarrow ($\langle GK \rangle, \langle GL \rangle$) \rightarrow ($\langle GL \rangle, \langle GKJ \rangle$) \rightarrow ($\langle \mathbf{GKJ} \rangle, \langle \mathbf{GLF} \rangle$)



Breadth-first and Depth-first

- Forward (Breadth-first):
 - ($\langle S \rangle$) \rightarrow ($\langle SA \rangle, \langle SE \rangle$) \rightarrow ($\langle SE \rangle, \langle SAB \rangle$) \rightarrow ($\langle SAB \rangle, \langle SEF \rangle$)
 \rightarrow (**$\langle SEF \rangle$** , $\langle SABC \rangle$)
- Backward (Depth-first):
 - ($\langle G \rangle$) \rightarrow ($\langle GJ \rangle, \langle GD \rangle$) \rightarrow ($\langle GJI \rangle, \langle GD \rangle$) \rightarrow ($\langle GJIH \rangle, \langle GD \rangle$)
 \rightarrow (**$\langle GJIHF \rangle$** , $\langle GD \rangle$)

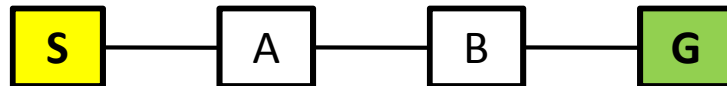


Bidirectional Search

PROBLEM 2: SHARED-STATE CHECK?

Replace shared-state check

- When only checking identical end-states, paths can cross each other unnoticed.
- Forward:
 - $(\langle S \rangle) \rightarrow (\langle SA \rangle) \rightarrow (\langle SAB \rangle) \rightarrow (\langle SABG \rangle)$
- Backward:
 - $(\langle G \rangle) \rightarrow (\langle GB \rangle) \rightarrow (\langle GBA \rangle) \rightarrow (\langle GBAS \rangle)$



Exercises: Artificial Intelligence

Beam Search

Beam Search

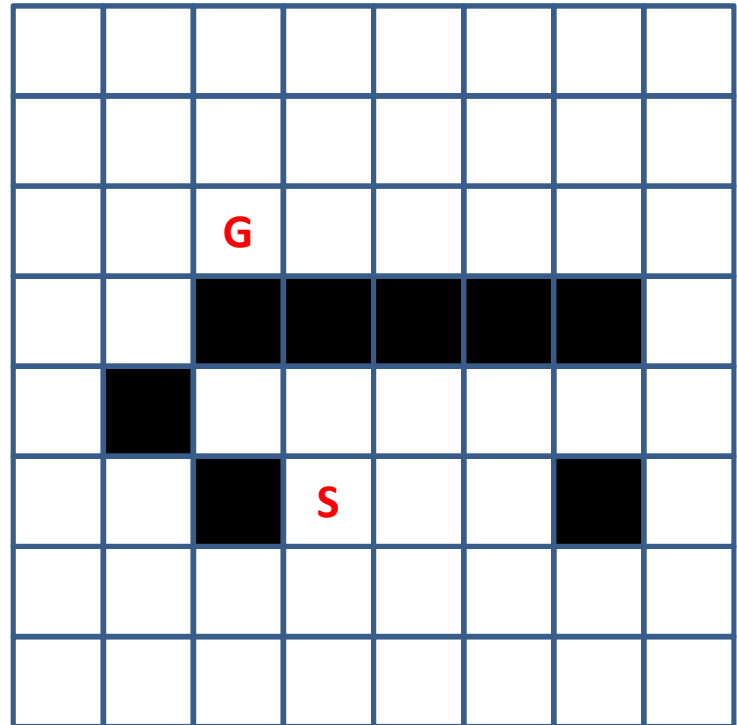
- **Input:**
 - **QUEUE:** Path only containing root
 - **WIDTH:** Number
- **Algorithm:**
 - **WHILE** (QUEUE not empty && goal not reached) **DO**
 - Remove **all paths** from QUEUE
 - Create paths to all children (of all paths)
 - Reject paths with loops
 - **Sort new paths (according to heuristic)**
 - **(Optimization: Remove paths without successor)**
 - Add WIDTH **best paths** to QUEUE
 - **IF** goal reached
 - **THEN** success
 - **ELSE** failure

Exercises: Artificial Intelligence

Path Search

Problem

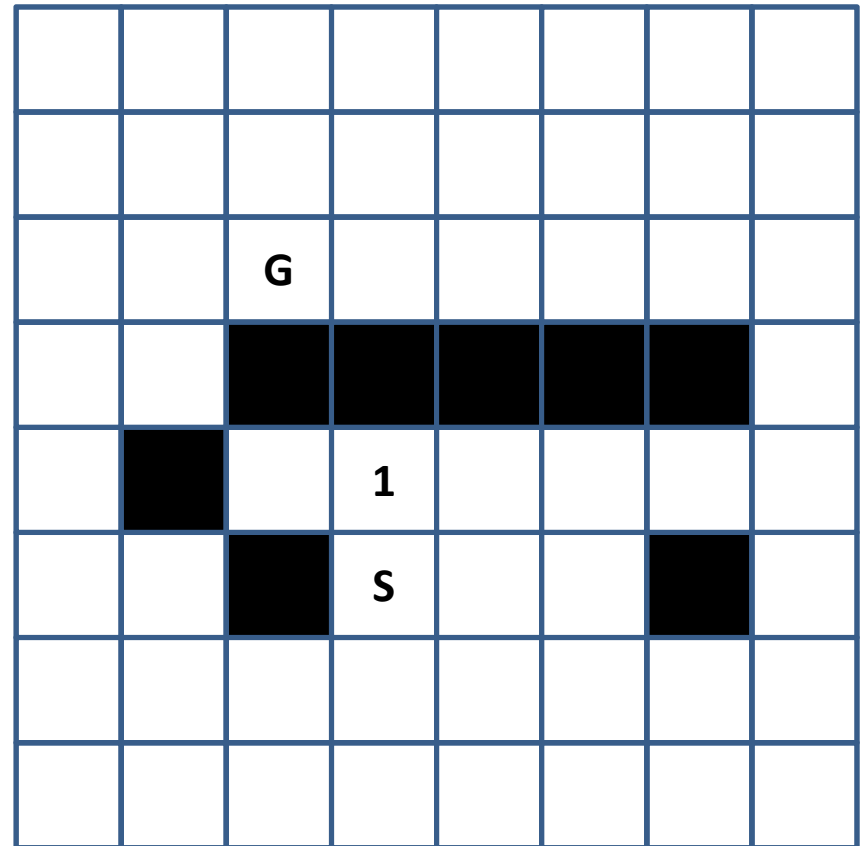
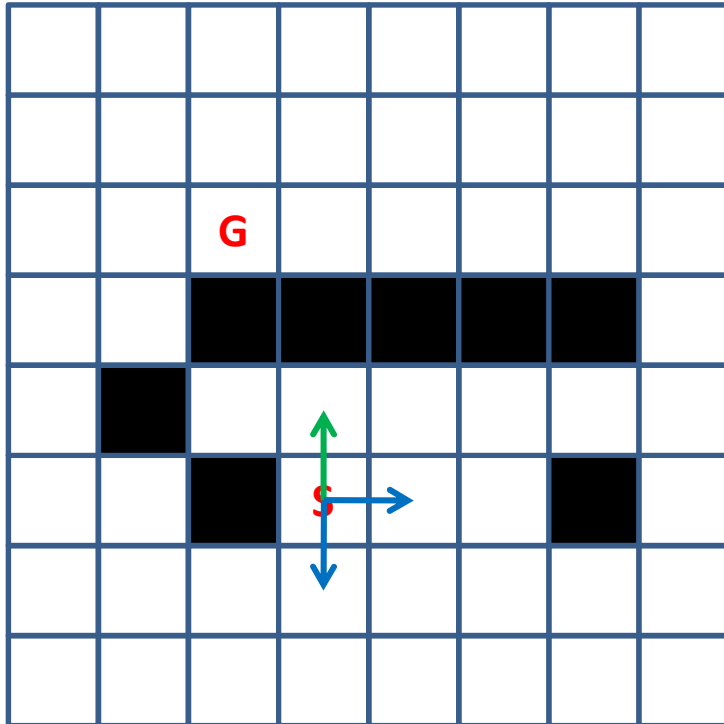
- Find a path from 'S' to 'G', without passing through black squares.
 - Legal steps (order):
 - up, left, right, down
- Perform:
 - Depth-first search
 - Hill-climbing | Search
 - With suitable heuristic
 - Greedy Search
 - With same heuristic



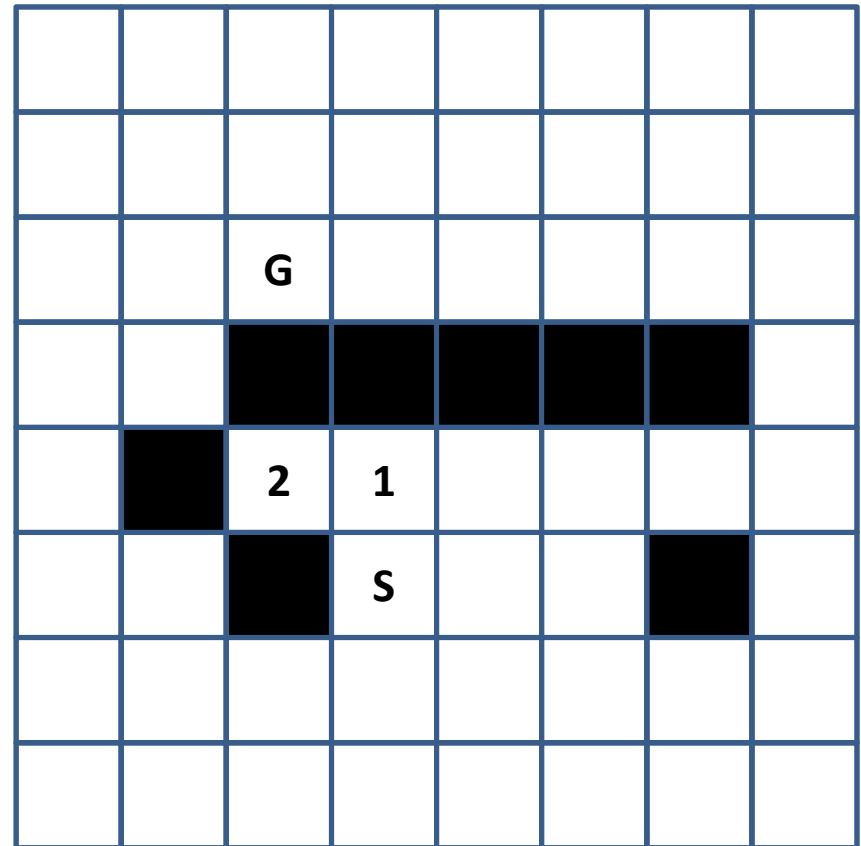
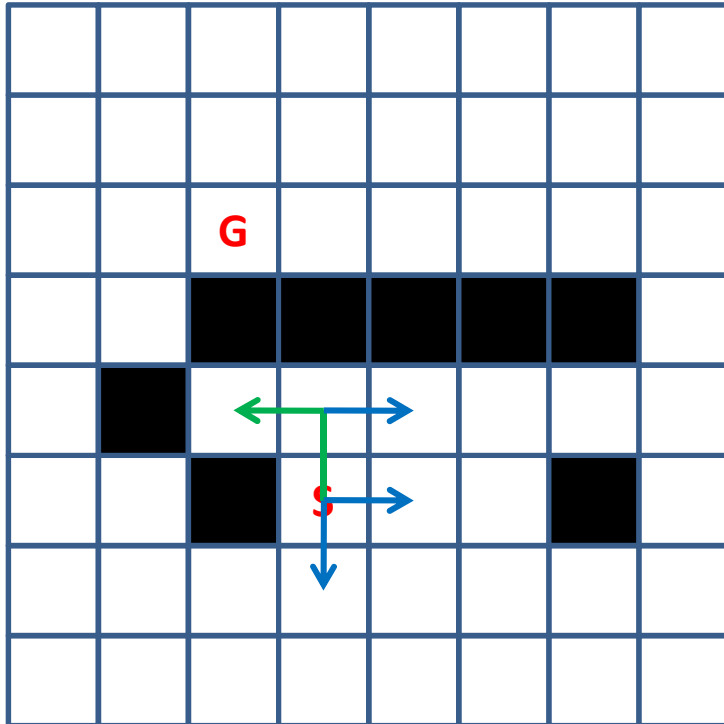
Path Search

DEPTH-FIRST SEARCH

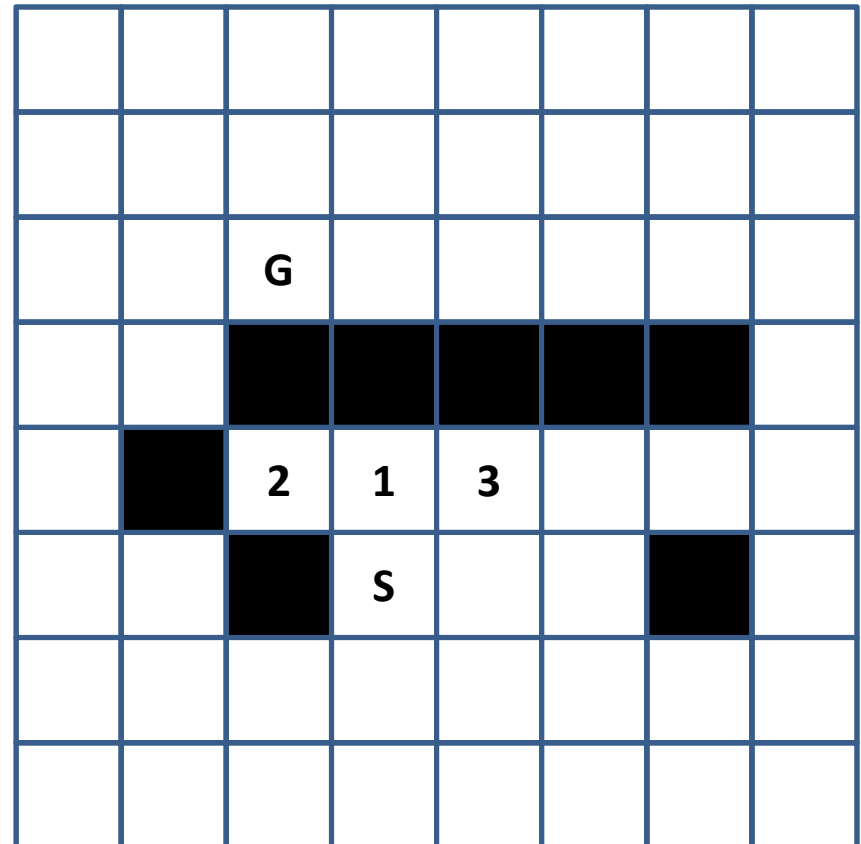
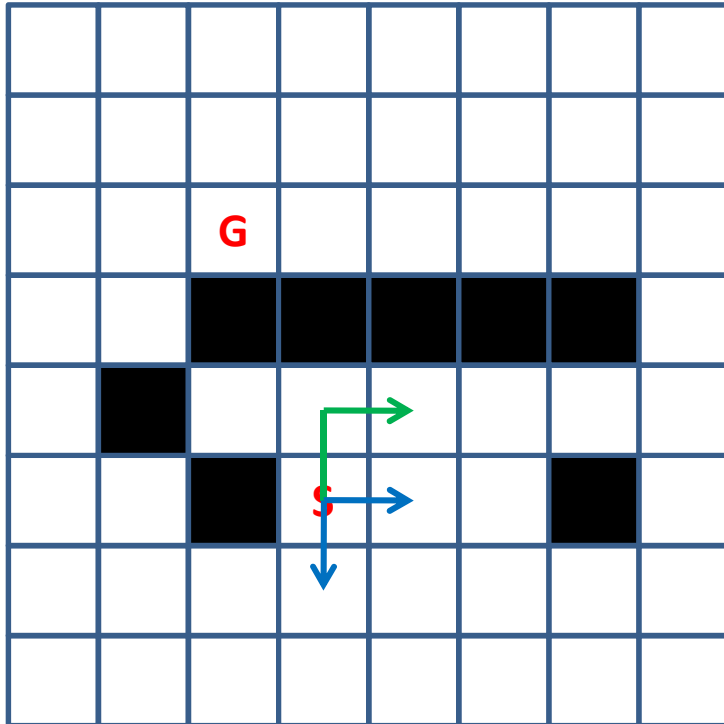
Depth-first Search



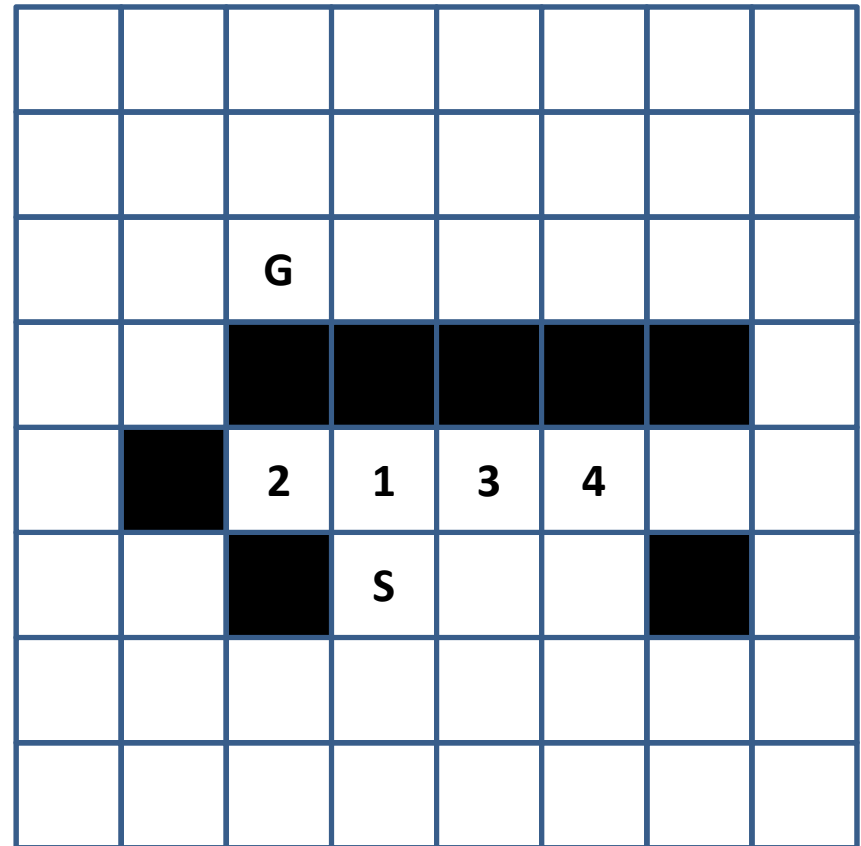
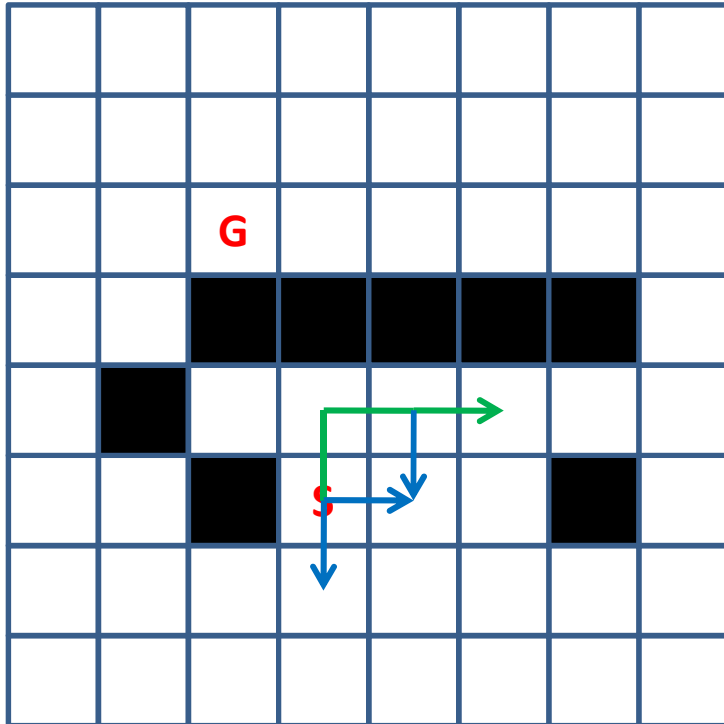
Depth-first Search



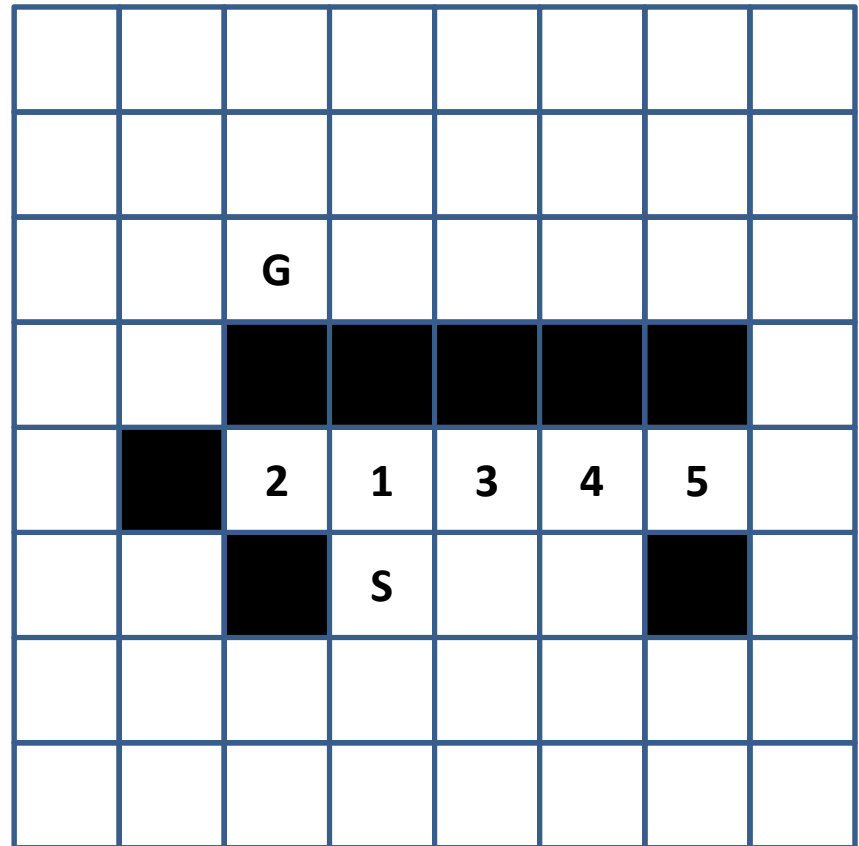
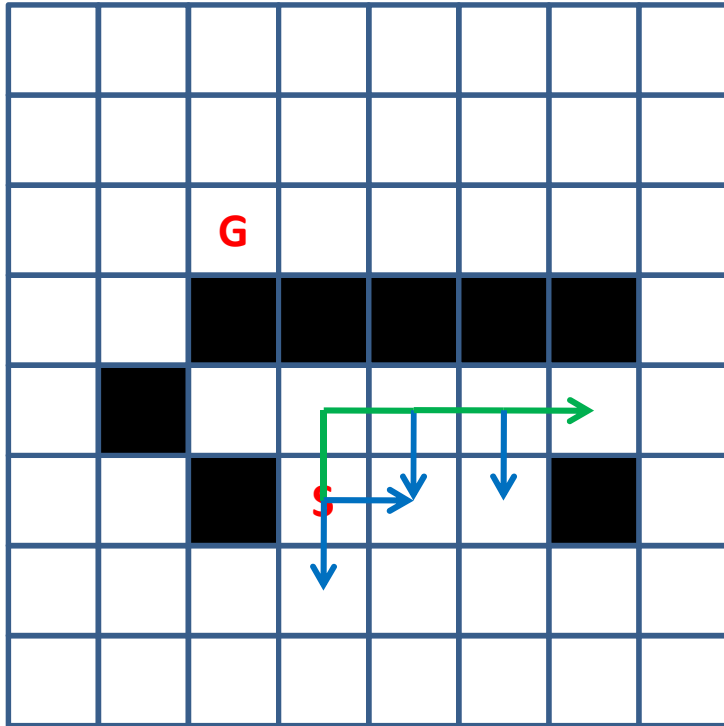
Depth-first Search



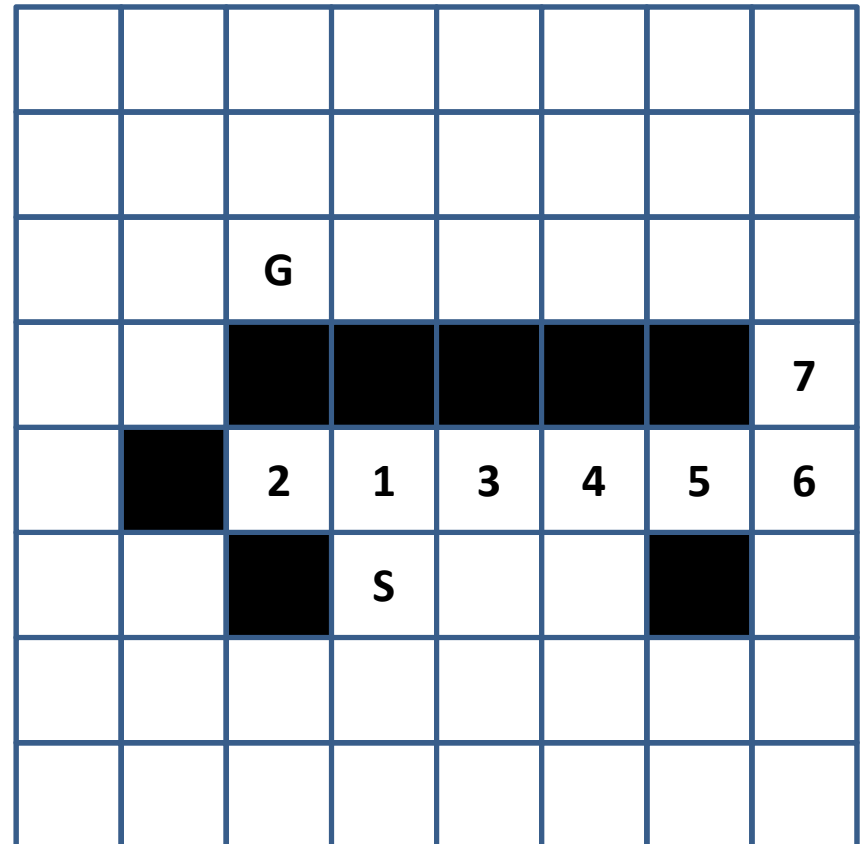
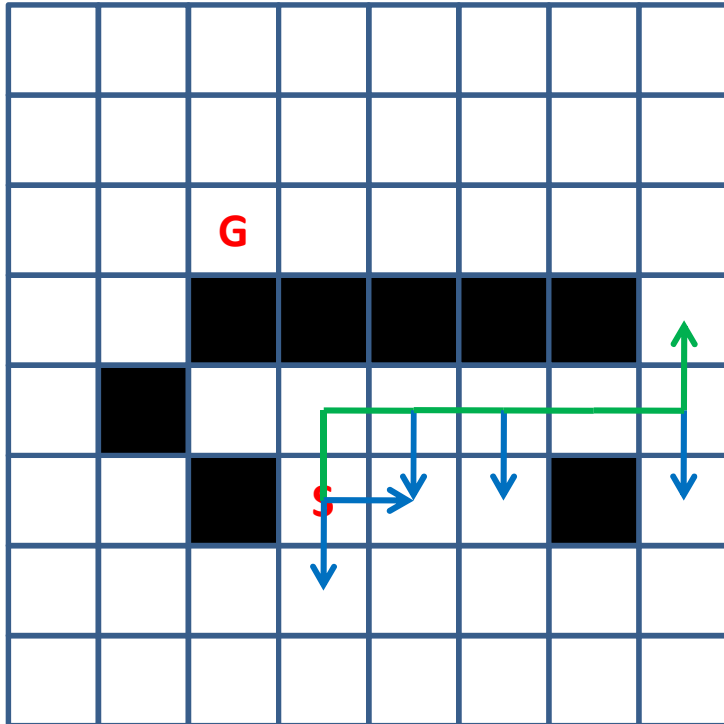
Depth-first Search



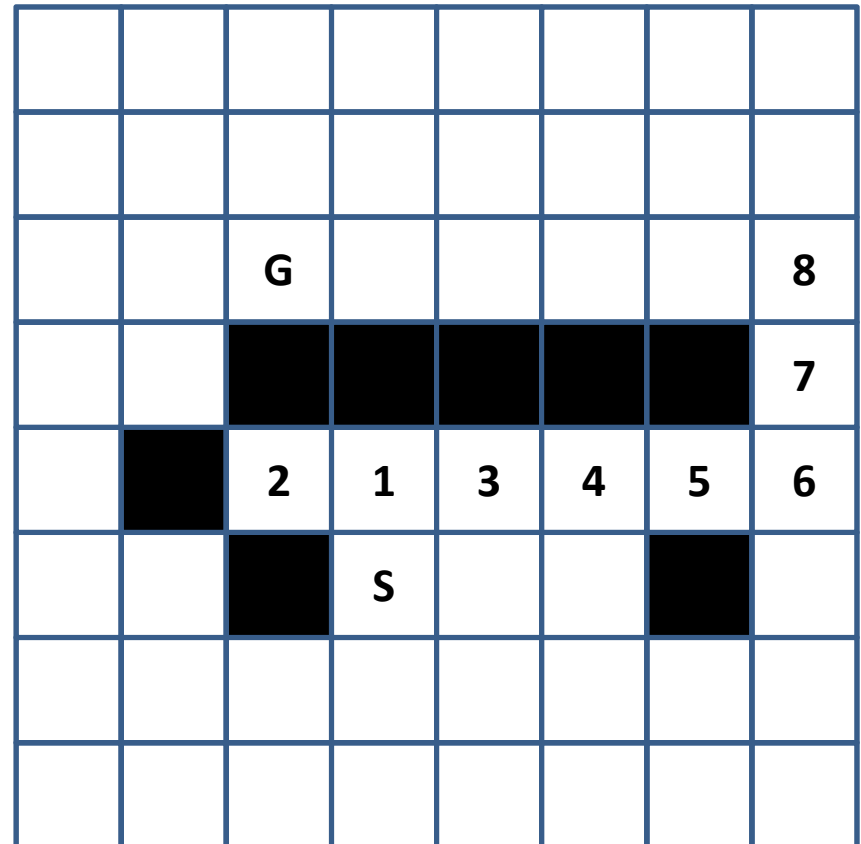
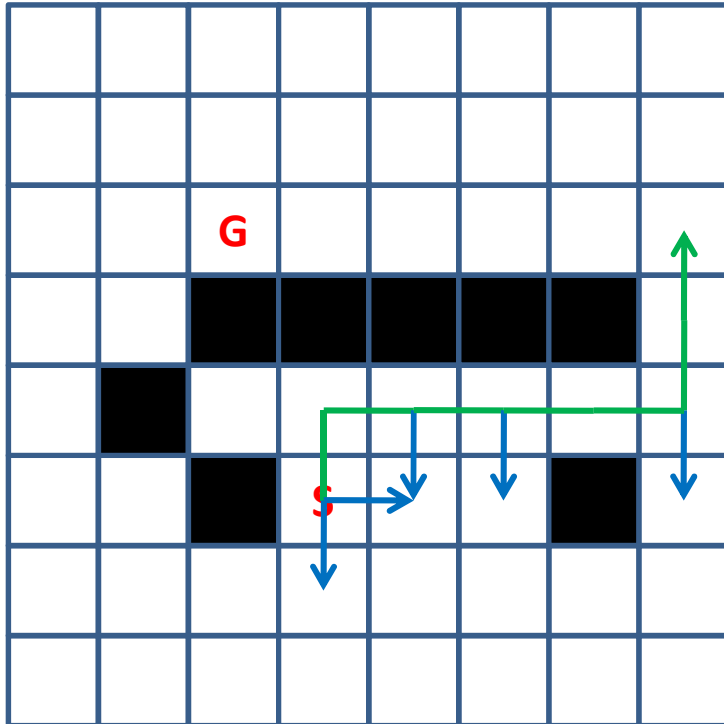
Depth-first Search



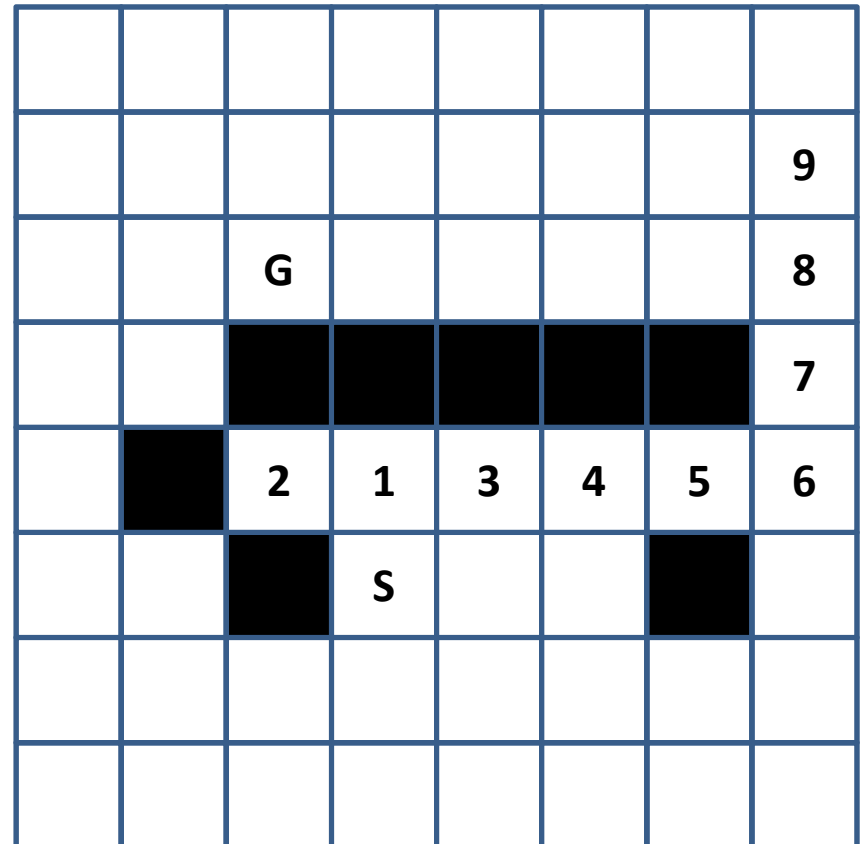
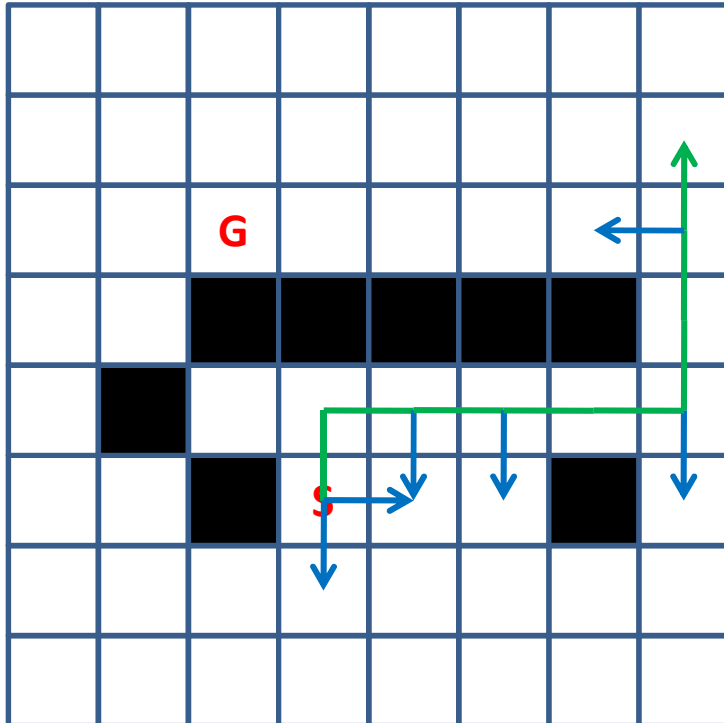
Depth-first Search



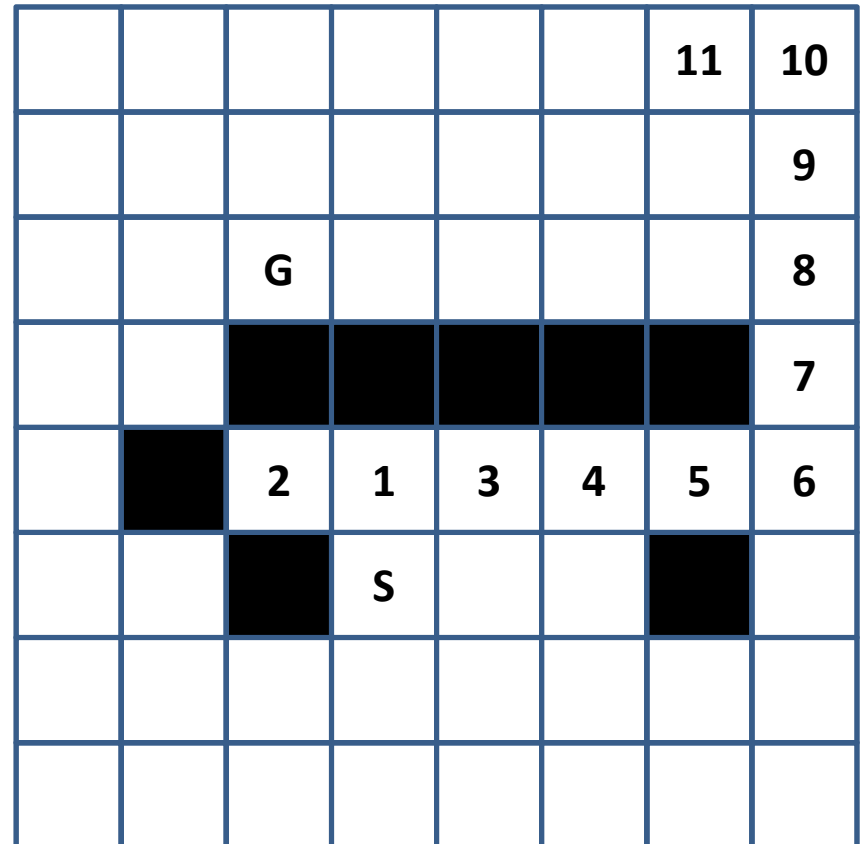
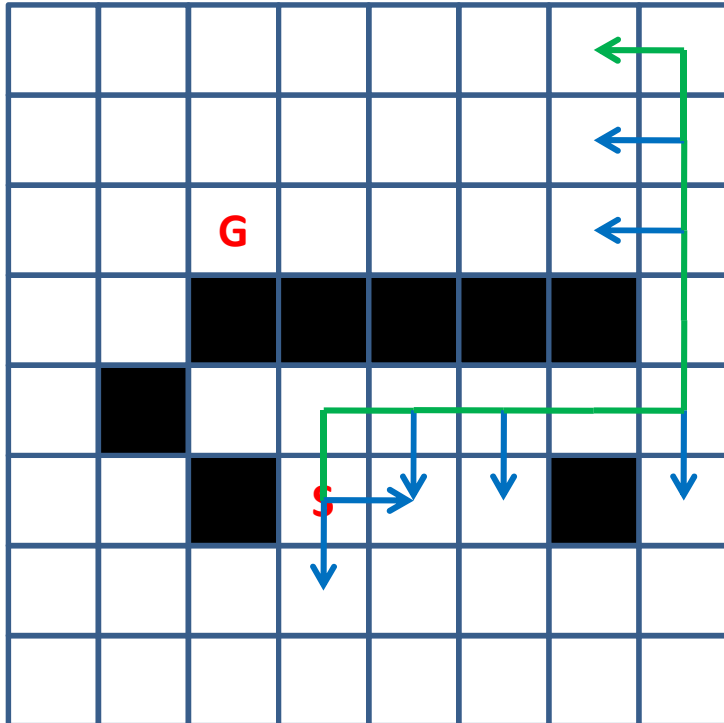
Depth-first Search



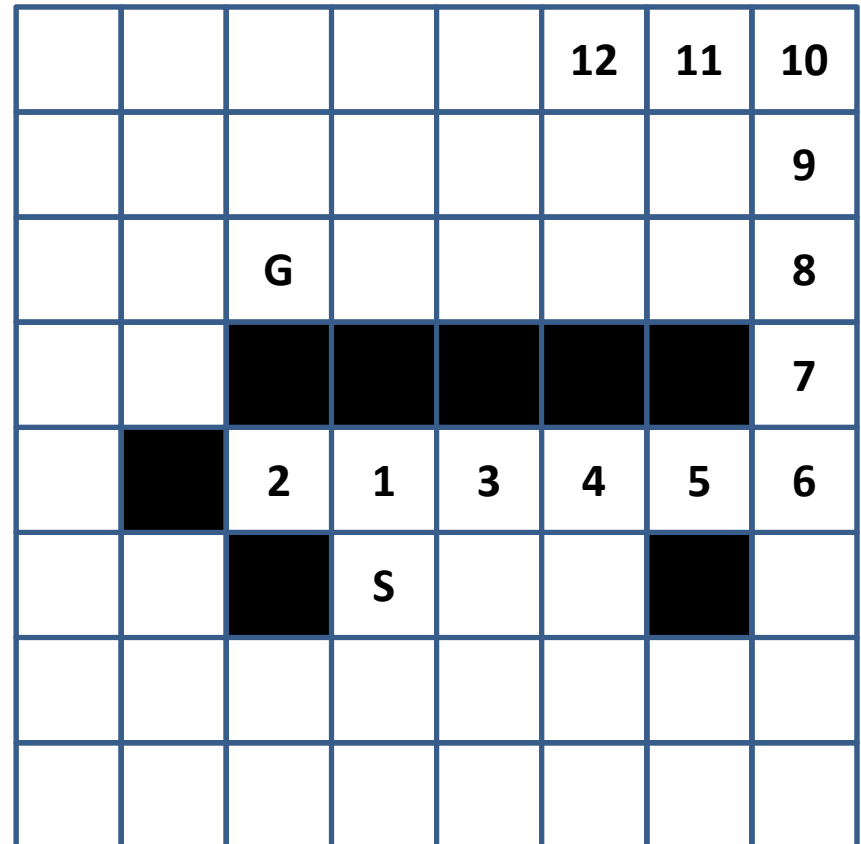
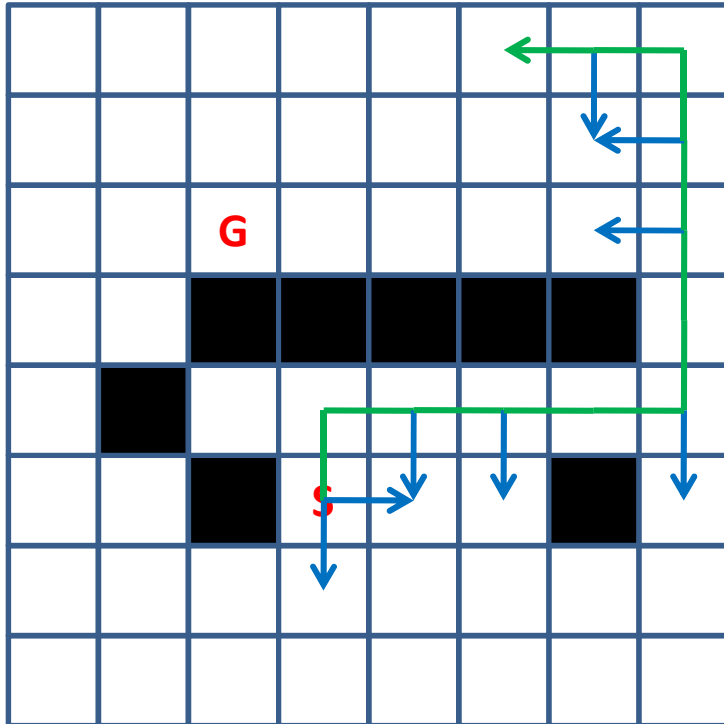
Depth-first Search



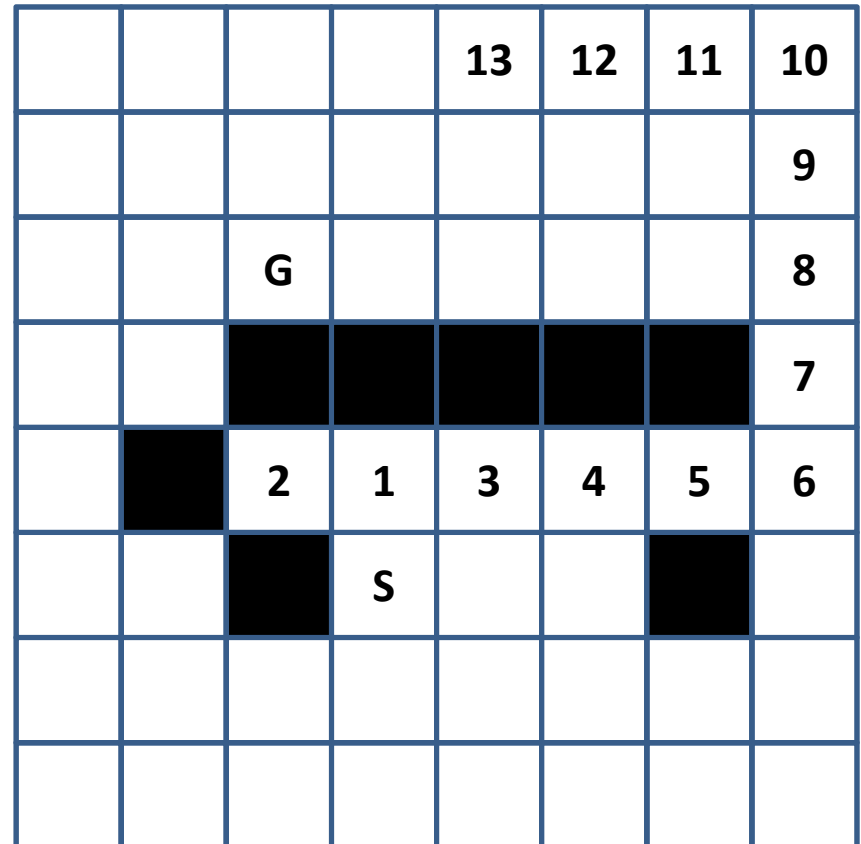
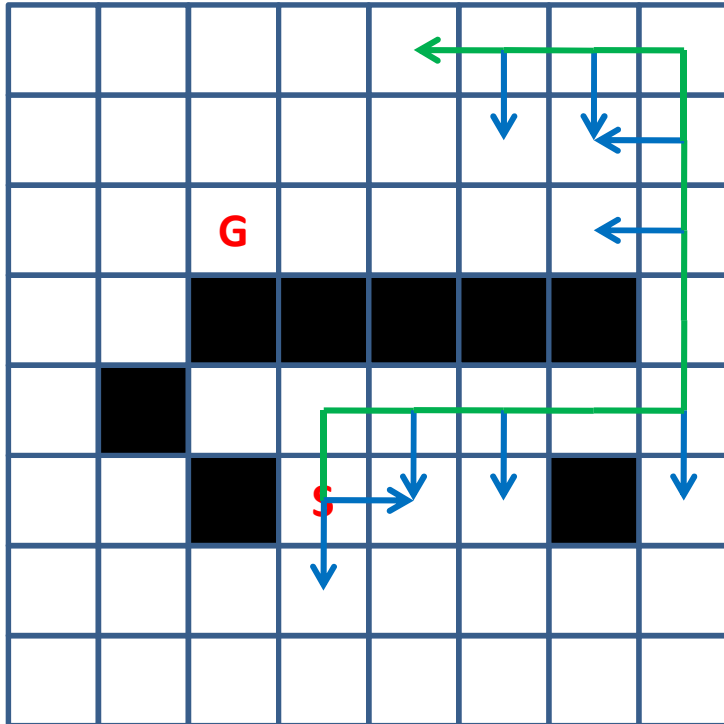
Depth-first Search



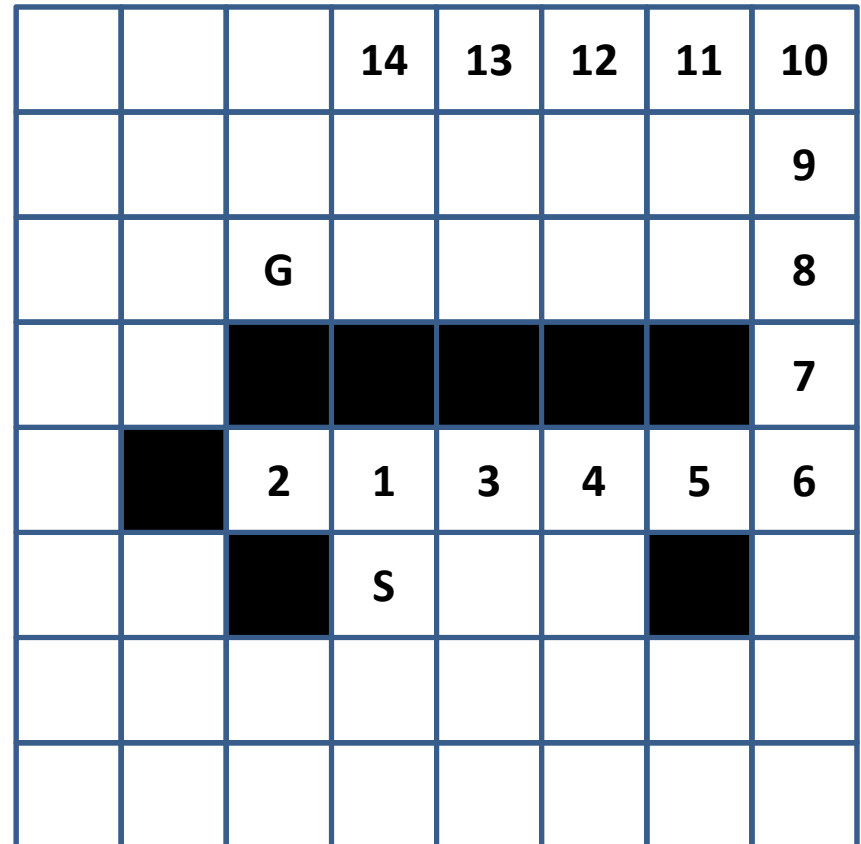
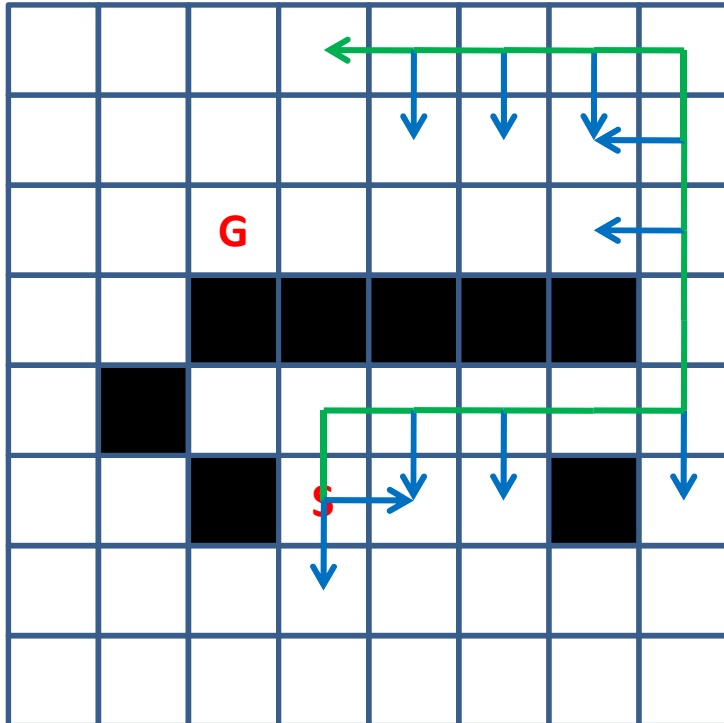
Depth-first Search



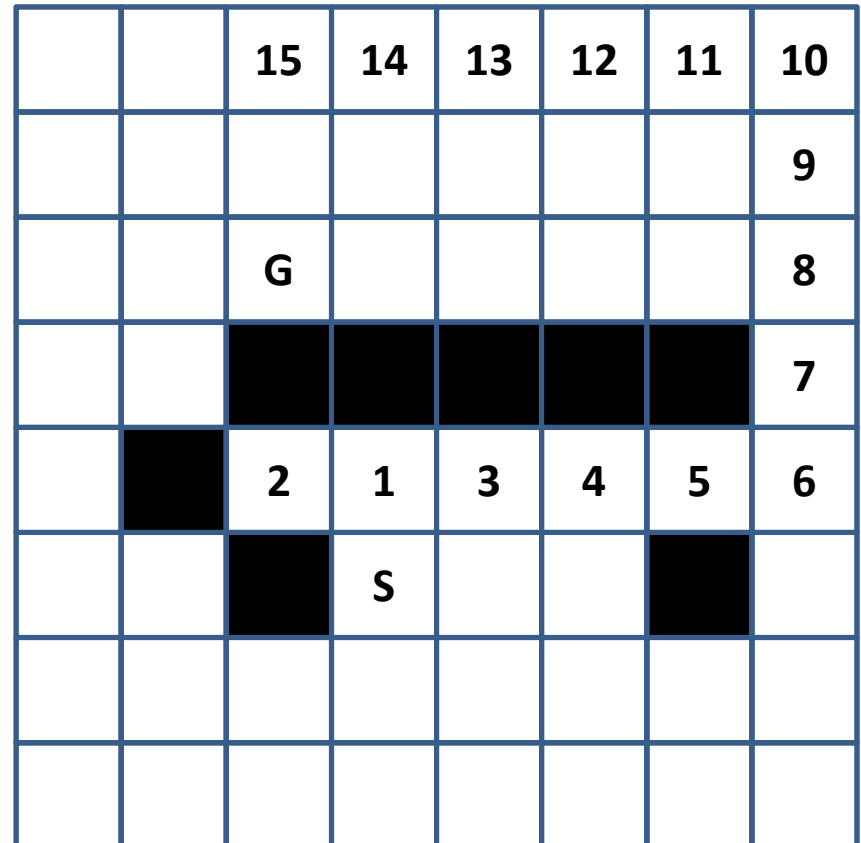
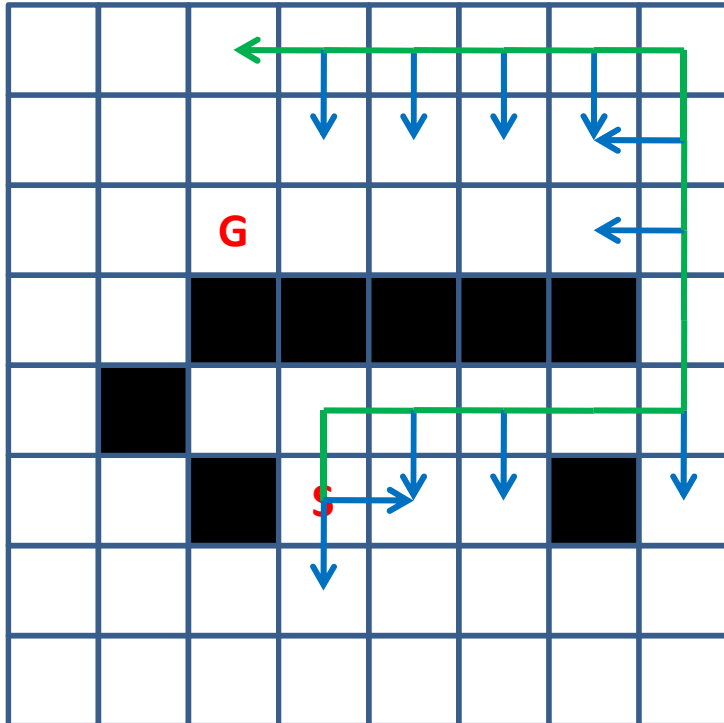
Depth-first Search



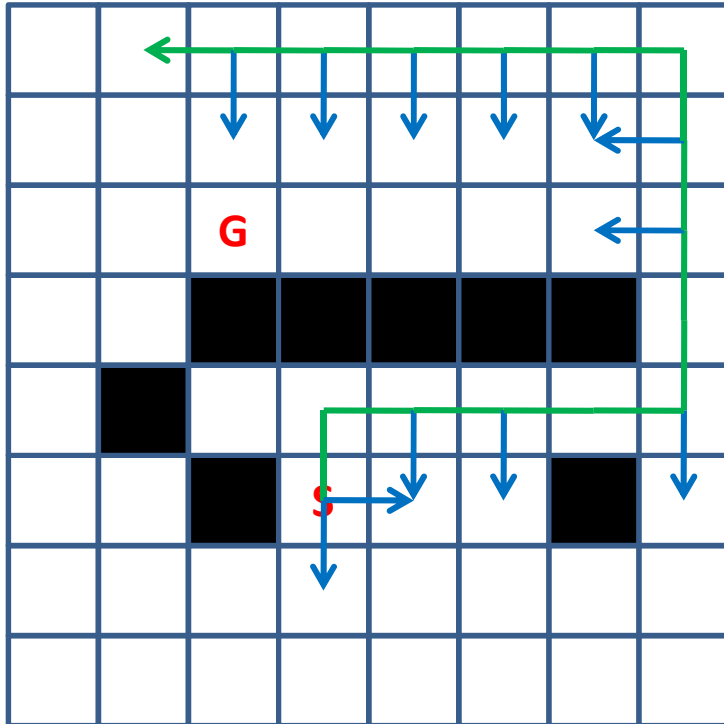
Depth-first Search



Depth-first Search

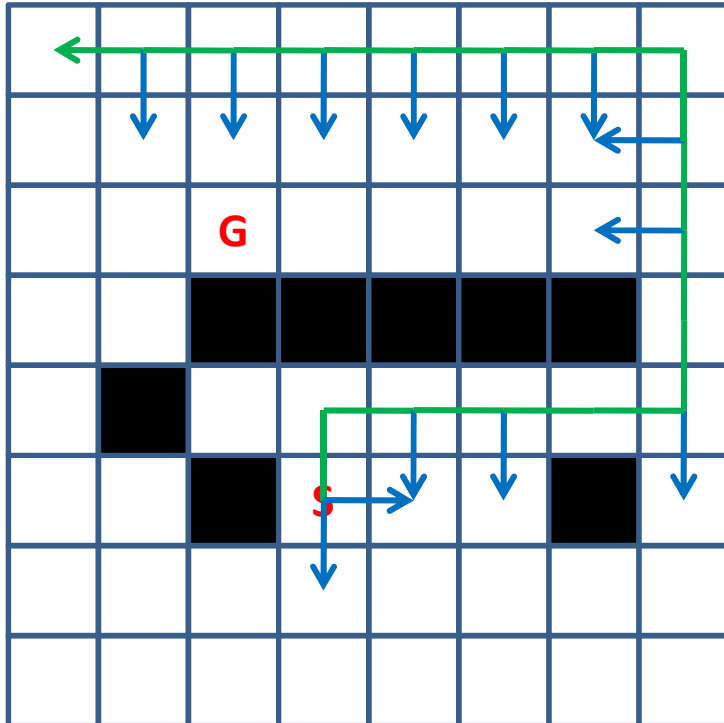


Depth-first Search



	16	15	14	13	12	11	10
							9
		G					8
							7
		2	1	3	4	5	6
			S				

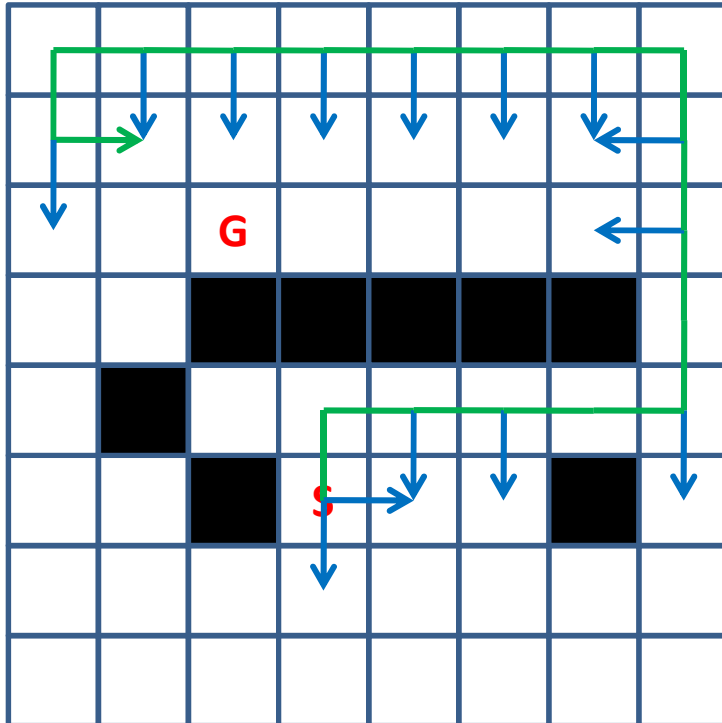
Depth-first Search



17	16	15	14	13	12	11	10
							9
		G					8
							7
							6
							5
							4
							3
							2
							1
							0

A 10x10 grid with columns numbered 10 to 17 from right to left. The start cell 'S' is at (14, 4) and the goal cell 'G' is at (15, 8). Black cells are located at (15, 7), (16, 7), (17, 7), (15, 6), (16, 6), (17, 6), (15, 5), (16, 5), (17, 5), (15, 4), (16, 4), (17, 4), (15, 3), (16, 3), (17, 3), (15, 2), (16, 2), (17, 2), (15, 1), (16, 1), (17, 1), (15, 0), (16, 0), (17, 0). The path from 'S' to 'G' is shown with blue arrows: S → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8.

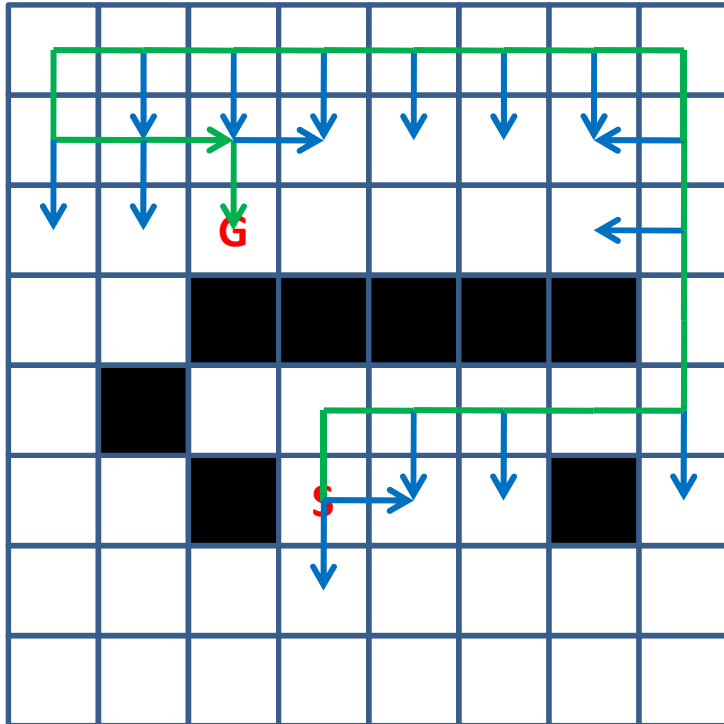
Depth-first Search



17	16	15	14	13	12	11	10
18	19						9
		G					8
							7
							6
							5
							4
							3
							2
							1
							0
							0

A 10x10 grid illustrating Depth-First Search. The start cell 'S' is at (4,4) and the goal cell 'G' is at (3,3). Blue arrows show the search path starting from 'S', moving down to (5,4), then right to (5,5), then up to (4,5), then left to (4,6), then up to (3,6), then left to (3,5), then up to (2,5), then left to (2,4), then up to (1,4), then left to (1,3), then down to (2,3), and finally right to (3,3). A green line traces this path.

Depth-first Search



17	16	15	14	13	12	11	10
18	19	20					9
		G					8
							7
							6
							5
							4
							3
							2
							1
							0
							0

Path Search

HILL-CLIMBING | SEARCH

Heuristic: Manhattan Distance

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	0	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		4	5	6		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

Hill-climbing I Search

- ***Input:***
 - **QUEUE:** Path only containing root
- ***Algorithm:***
 - **WHILE** (QUEUE not empty && goal not reached) **DO**
 - Remove first path from QUEUE
 - Create paths to all children
 - Reject paths with loops
 - **Sort new paths using heuristic**
 - Add **sorted** paths to **front** of QUEUE
 - **IF** goal reached
 - **THEN** success
 - **ELSE** failure

Hill-climbing I Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		5	6			8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

A grid representing a search space. The goal 'G' is located at row 3, column 3. The start 'S' is at row 5, column 4. A path is shown with arrows: a green arrow points left from S to 2, a blue arrow points right from 2 to 3, a blue arrow points right from 3 to 4, and a blue arrow points down from 4 to 5.

		G					

A grid representing a search space with obstacles. The goal 'G' is at row 3, column 3. The start 'S' is at row 6, column 4. Obstacles are represented by black cells at (3,4), (3,5), (3,6), (3,7), (4,2), (5,3), and (6,7). The numbers 2 and 1 are placed at (5,3) and (6,4) respectively.

Hill-climbing I Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		S	5	6		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

A grid representing a search space with values from 1 to 10. The goal 'G' is at (2,3) and the start 'S' is at (5,4). Black cells are at (3,2), (3,3), (3,4), (3,5), (3,6), (4,1), (5,2), (5,3), (5,7), and (6,7). A green path shows a move from S to (4,3) and then to (4,4). A blue path shows a move from S to (5,5) and then to (6,4).

		G					

A grid representing a search space with values from 1 to 4. The goal 'G' is at (2,3) and the start 'S' is at (5,3). Black cells are at (3,2), (3,3), (3,4), (3,5), (3,6), (4,1), (5,2), (5,7), and (6,7). A path shows a move from S to (4,2), (4,3), (4,4), and (4,5).

Hill-climbing I Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		S	5	6		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G						
							7	
			2	1	3	4	5	6
				S				

Hill-climbing | Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		S	5	6		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					8	
							7	
			2	1	3	4	5	6
				S				

Hill-climbing I Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		5	6			8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

A grid of numbers from 1 to 10. A path is highlighted with green arrows: (2,3) to (2,4) to (2,5) to (2,6) to (2,7) to (2,8) to (3,8) to (4,8) to (5,8) to (6,8) to (7,8) to (8,8) to (9,8) to (10,8). Blue arrows point from (2,8) to (3,8), (3,8) to (4,8), (4,8) to (5,8), (5,8) to (6,8), (6,8) to (7,8), (7,8) to (8,8), (8,8) to (9,8), and (9,8) to (10,8). A green arrow points from (2,7) to (2,8). A black obstacle is present at (2,2).

		G				9	8	
							7	
			2	1	3	4	5	6
				S				

A grid with obstacles (black squares) and a path. The path consists of the sequence of numbers: 2, 1, 3, 4, 5, 6. The start node 'S' is at (5,4) and the goal node 'G' is at (2,3). Obstacles are located at (2,2), (3,2), (3,3), (3,4), (3,5), (3,6), (4,1), (5,2), (5,7), (6,2), (6,7), (6,8).

Hill-climbing I Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		5	6	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

A grid representing a search space with values from 1 to 10. The goal 'G' is at (2,3). A path is shown with green arrows: (2,3) to (2,4) to (2,5) to (2,6) to (2,7) to (2,8) to (3,8) to (4,8) to (5,8) to (6,8) to (7,8) to (8,8) to (9,8) to (10,8). Blue arrows show a path from (2,7) to (3,7) to (4,7) to (5,7) to (6,7) to (7,7) to (8,7) to (9,7) to (10,7).

		G			10	9	8	
							7	
			2	1	3	4	5	6
				S				

A grid representing a search space with values from 1 to 10. The goal 'G' is at (2,3) and the start 'S' is at (6,4). Black cells are at (3,2), (3,3), (3,4), (3,5), (3,6), (3,7), (4,1), (5,2), (5,3), (5,4), (5,5), (5,6), (5,7), (6,1), (6,2), (6,3), (6,5), (6,6), (6,7), (6,8), (6,9), (6,10).

Hill-climbing I Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		5	6	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

A grid representing a search space with values from 1 to 10. The goal 'G' is at (2,3). A path is shown with blue arrows: (2,3) → (2,4) → (2,5) → (2,6) → (2,7) → (2,8) → (3,8) → (4,8) → (5,8) → (6,8) → (7,8) → (8,8) → (9,8) → (10,8). A green path is shown: (2,3) → (3,3) → (4,3) → (5,3) → (6,3) → (7,3) → (8,3) → (9,3) → (10,3).

		G		11	10	9	8	
							7	
			2	1	3	4	5	6
				S				

A grid representing a search space with values from 1 to 11. The goal 'G' is at (2,3) and the start 'S' is at (6,4). Black cells are at (3,2), (3,3), (3,4), (3,5), (3,6), (4,1), (5,2), (6,3), (6,7), (6,8).

Hill-climbing I Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		5	6			8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

Diagram illustrating a grid with values and search paths. The grid contains numbers 1-10 and a goal 'G'. A green path starts at (row 3, col 5) and moves left to (row 3, col 1). Blue arrows indicate a search path starting from (row 3, col 5) and moving up, down, left, and right to adjacent cells.

		G	12	11	10	9	8	
							7	
			2	1	3	4	5	6
				S				

Diagram illustrating a grid with values and search paths. The grid contains numbers 1-12 and a goal 'G'. A search path starts at (row 6, col 4) and moves up, down, left, and right to adjacent cells.

Hill-climbing I Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		5	6			8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

A grid illustrating a search space for Hill-climbing I Search. The grid is 8x8. The values in the grid are: Row 1: 4, 3, 2, 3, 4, 5, 6, 7; Row 2: 3, 2, 1, 2, 3, 4, 5, 6; Row 3: 2, 1, G, 1, 2, 3, 4, 5; Row 4: 3, 2, black, black, black, black, black, 6; Row 5: 4, black, 2, 3, 4, 5, 6, 7; Row 6: 5, 4, black, 5, 6, black, black, 8; Row 7: 6, 5, 4, 5, 6, 7, 8, 9; Row 8: 7, 6, 5, 6, 7, 8, 9, 10. A green path starts at G (row 3, col 3) and moves left to 1 (row 3, col 4), then right to 2 (row 3, col 5), 3 (row 3, col 6), 4 (row 3, col 7), and 5 (row 3, col 8). Blue arrows indicate the search path: from 5 (row 3, col 8) to 6 (row 4, col 8), 7 (row 5, col 8), 8 (row 6, col 8), 9 (row 7, col 8), 10 (row 8, col 8); from 5 (row 3, col 8) to 4 (row 3, col 7), 3 (row 3, col 6), 2 (row 3, col 5), 1 (row 3, col 4); from 1 (row 3, col 4) to 2 (row 4, col 4), 3 (row 5, col 4), 4 (row 6, col 4), 5 (row 7, col 4), 6 (row 8, col 4); from 2 (row 3, col 5) to 3 (row 4, col 5), 4 (row 5, col 5), 5 (row 6, col 5), 6 (row 7, col 5), 7 (row 8, col 5); from 3 (row 3, col 6) to 4 (row 4, col 6), 5 (row 5, col 6), 6 (row 6, col 6), 7 (row 7, col 6), 8 (row 8, col 6); from 4 (row 3, col 7) to 5 (row 4, col 7), 6 (row 5, col 7), 7 (row 6, col 7), 8 (row 7, col 7), 9 (row 8, col 7).

		G	12	11	10	9	8	
							7	
			2	1	3	4	5	6
				S				

A grid illustrating a search space for Hill-climbing I Search. The grid is 8x8. The values in the grid are: Row 1: empty, empty, empty, empty, empty, empty, empty, empty; Row 2: empty, empty, empty, empty, empty, empty, empty, empty; Row 3: empty, empty, G, 12, 11, 10, 9, 8; Row 4: empty, empty, black, black, black, black, black, 7; Row 5: empty, black, 2, 1, 3, 4, 5, 6; Row 6: empty, empty, black, S, empty, empty, black, empty; Row 7: empty, empty, empty, empty, empty, empty, empty, empty; Row 8: empty, empty, empty, empty, empty, empty, empty, empty.

Path Search

GREEDY SEARCH

Greedy Search

- ***Input:***
 - **QUEUE:** Path only containing root
- ***Algorithm:***
 - **WHILE** (QUEUE not empty && goal not reached) **DO**
 - Remove first path from QUEUE
 - Create paths to all children
 - Reject paths with loops
 - Add paths to QUEUE and ***sort the entire QUEUE (heuristic)***
 - **IF** goal reached
 - **THEN** success
 - **ELSE** failure

Greedy Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		5	5	6		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					
16		2/9	1/8	3/7	4/10		
15	14		S	5/6			
	13	12	11				

Greedy Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		5	5	6		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					
17							
16		2/9	1/8	3/7	4/10		
15	14		S	5/6			
	13	12	11				

Greedy Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		5	5	6		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

18		G					
17							
16		2/9	1/8	3/7	4/10		
15	14		S	5/6			
	13	12	11				

Greedy Search

4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		5	5	6		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

Diagram illustrating a Greedy Search path on a grid. The grid contains numerical values and obstacles (black squares). The path starts at 'G' (row 3, column 3) and ends at 'S' (row 6, column 4). The path is marked with arrows: green arrows for the initial path and blue arrows for a subsequent path. Obstacles are located at (row, column): (4,2), (4,3), (4,4), (4,5), (4,6), (5,1), (6,2), (6,3), (6,7), (7,8).

18	19	G					
17							
16		2/9	1/8	3/7	4/10		
15	14		S	5/6			
	13	12	11				

Diagram illustrating a Greedy Search path on a grid. The grid contains numerical values, fractions, and obstacles (black squares). The path starts at 'G' (row 3, column 3) and ends at 'S' (row 5, column 4). The path is marked with arrows: green arrows for the initial path and blue arrows for a subsequent path. Obstacles are located at (row, column): (4,2), (4,3), (4,4), (4,5), (4,6), (5,1), (6,2), (6,3), (6,7), (7,8).

Exercises: Artificial Intelligence

Water Jugs

Problem

- Solve the water jugs problem
 - Given two jugs of 4 liter and 3 liter respectively, fill the 4 liter jug with 2 liter of water.
 - Find a good heuristic.
 - Perform Hill-climbing II Search.

Water jugs

PROBLEM REPRESENTATION

Representation

- States of the form $[x,y]$, where:
 - x : *contents of 4 liter jug*
 - y : *contents of 3 liter jug*
- Start: $[0,0]$
- Goal: $[2,0]$

Representation

- Rules:

- Fill x: $[x,y] \wedge x < 4 \longrightarrow [4,y]$

- Fill y: $[x,y] \wedge y < 3 \longrightarrow [x,3]$

- Empty x: $[x,y] \wedge x > 0 \longrightarrow [0,y]$

- Empty y: $[x,y] \wedge y > 0 \longrightarrow [x,0]$

- Fill x with y: $[x,y] \wedge x+y > 4 \wedge y > 0 \longrightarrow [4,(x+y-4)]$

- Fill x with y: $[x,y] \wedge x+y \leq 4 \wedge y > 0 \longrightarrow [(x+y),0]$

- Fill y with x: $[x,y] \wedge x+y > 3 \wedge x > 0 \longrightarrow [(x+y-3),3]$

- Fill y with x: $[x,y] \wedge x+y \leq 3 \wedge x > 0 \longrightarrow [0,(x+y)]$

Water jugs

HEURISTIC

Heuristic

- $H([x,y]) = f(x) + f(y)$
- $f(x)$ is defined as follows:

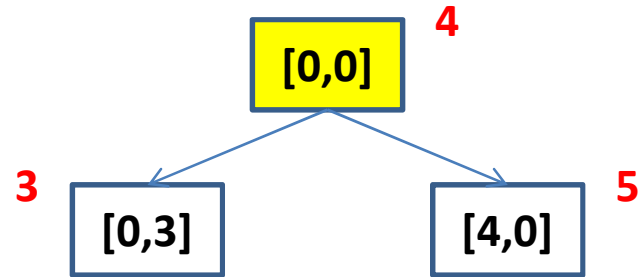
x	0	1	2	3	4
f(x)	2	1	0	1	3

- We need a jug filled with 2 liter.
- To obtain a jug filled with 2 liter we need a jug filled with either 1 or 3 liter.
- We consider an empty jug better than a jug filled with 4 liter.

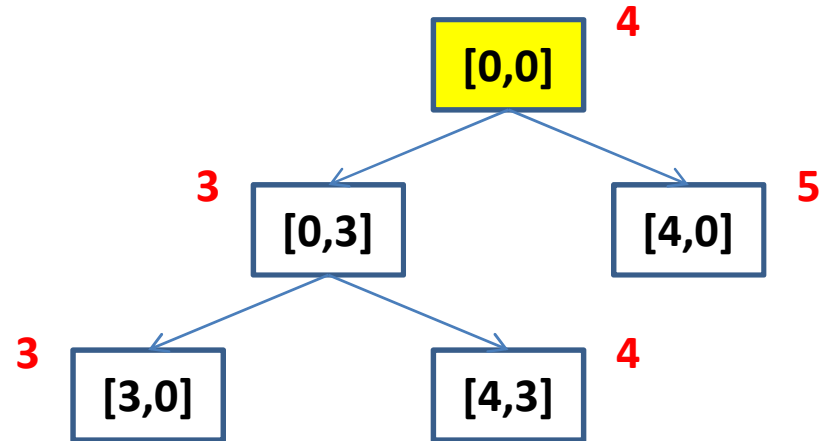
Water jugs

HILL-CLIMBING II SEARCH

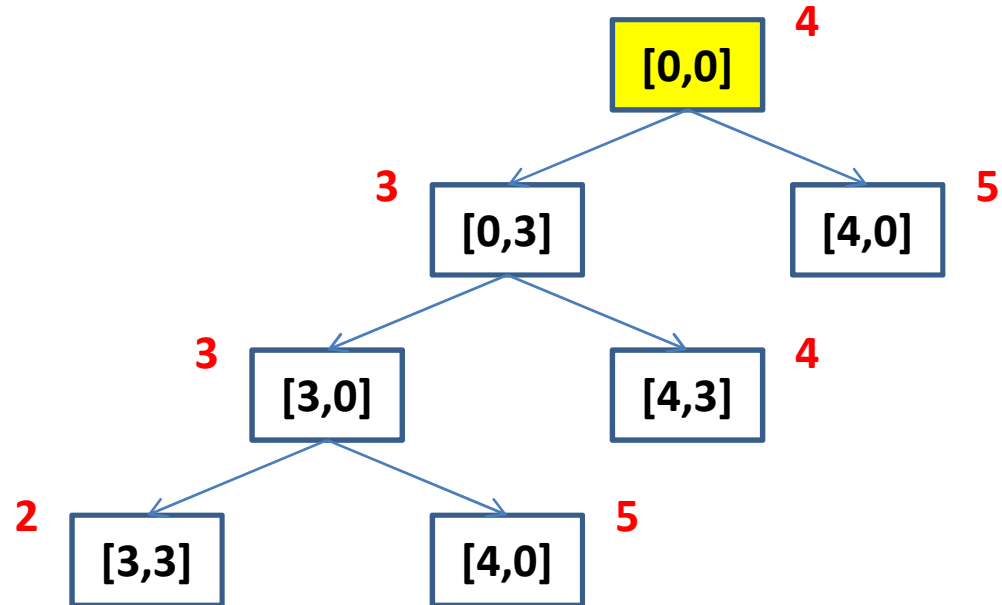
Hill-climbing II Search



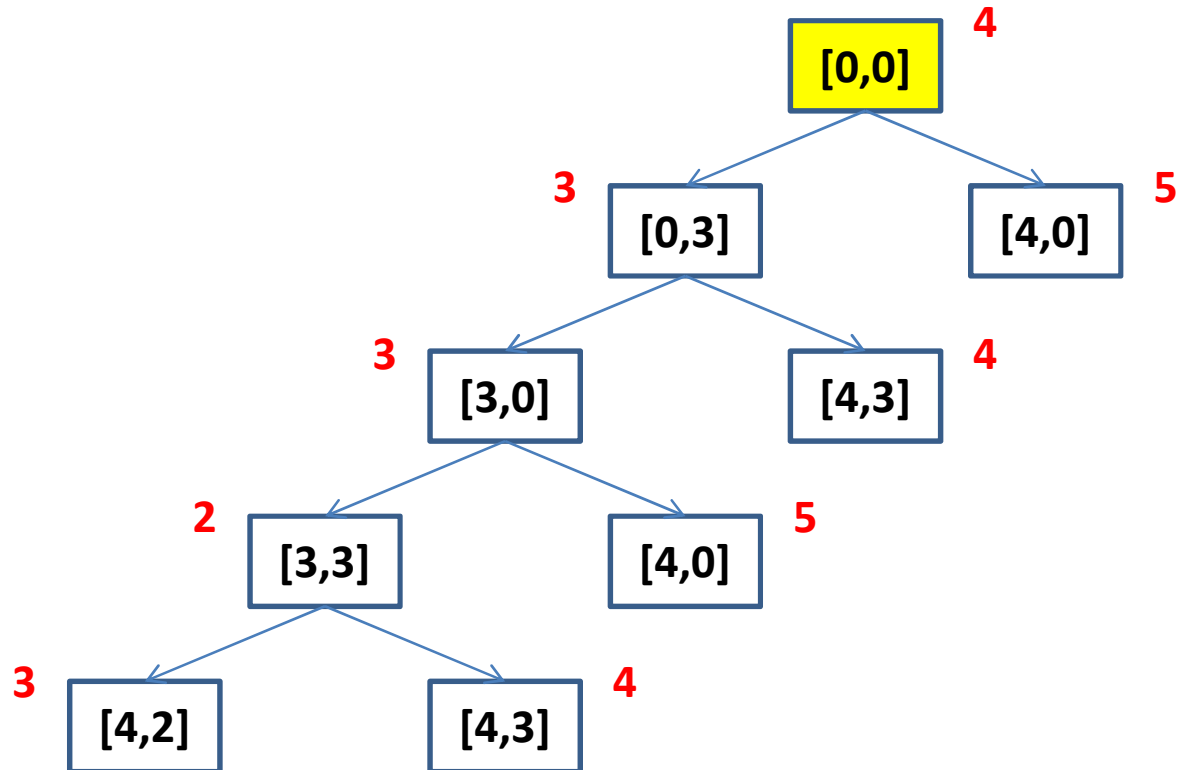
Hill-climbing II Search



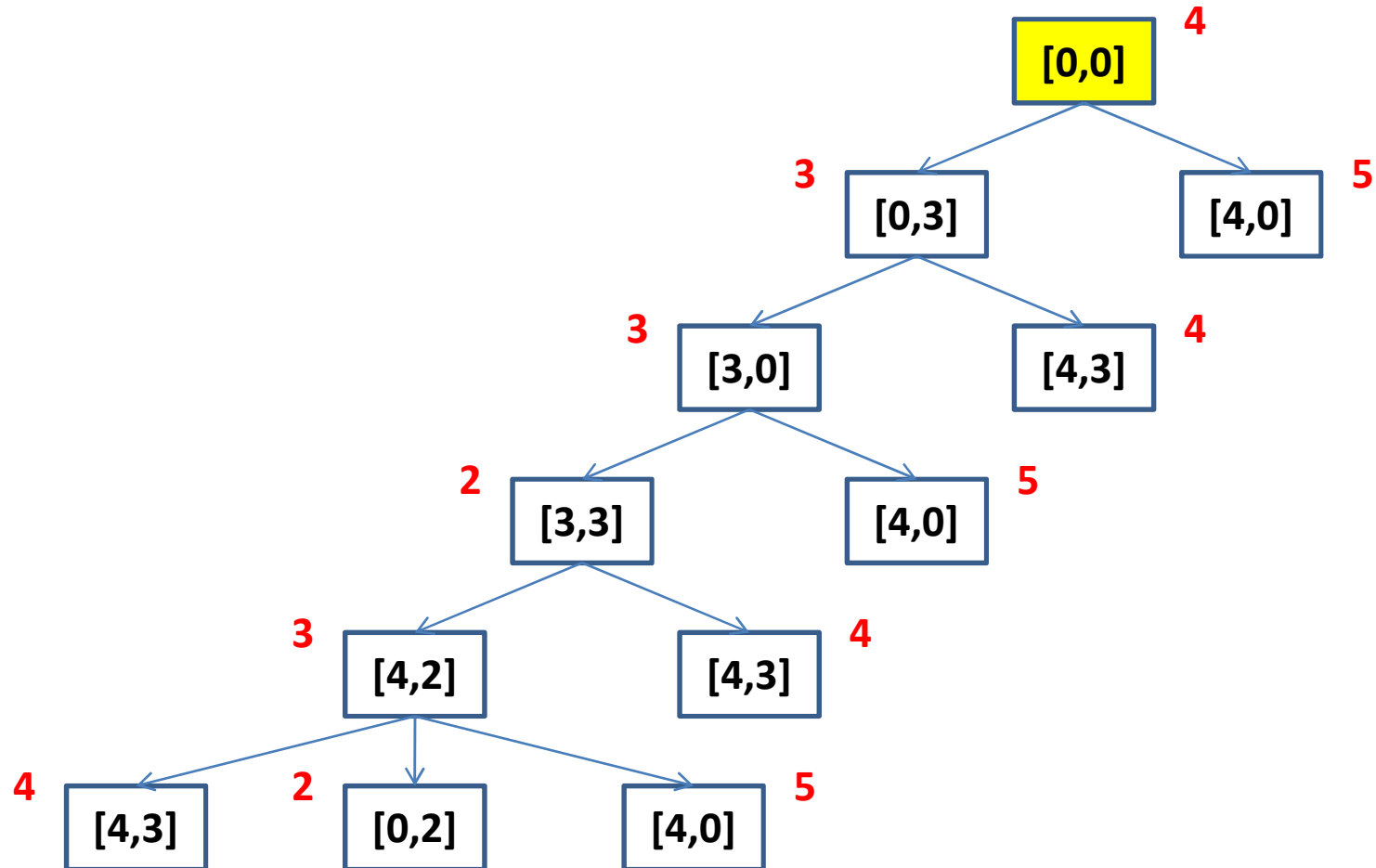
Hill-climbing II Search



Hill-climbing II Search



Hill-climbing II Search



Hill-climbing II Search

