

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)

- $S = (<[\mathbf{Fa\ Fo\ Go\ Gr\ |}]\mathbf{>})$
 - Paths to Children:
 - $R_3: <_{[\mathbf{Fa\ Fo\ Go\ Gr\ |}]}[\mathbf{Fo\ Gr\ |}\ \mathbf{Fa\ Go}]\mathbf{>}$
- $Q_1 = (<_{[\mathbf{Fa\ Fo\ Go\ Gr\ |}]}[\mathbf{Fo\ Gr\ |}\ \mathbf{Fa\ Go}]\mathbf{>})$

Depth-first search (queues)

- $S = (<[Fa\ Fo\ Go\ Gr\ |]\>)$
- $Q_1 = (<_{[Fa\ Fo\ Go\ Gr\ |]}\mathbf{[Fo\ Gr\ |Fa\ Go]}\>)$
 - Paths to Children:
 - $R_2: <_{[Fa\ Fo\ Go\ Gr\ |]}\mathbf{[Fa\ Fo\ Gr\ |Go]}\>$
 - $R_4: <_{[Fa\ Fo\ Go\ Gr\ |]}\mathbf{[Fa\ Fo\ Go\ Gr\ |]}\>$
- $Q_2 = (<_{[Fa\ Fo\ Go\ Gr\ |]}\mathbf{[Fa\ Fo\ Gr\ |Go]}\>)$

Depth-first search (queues)

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

Depth-first search (queues)

- $S = (<[Fa\ Fo\ Go\ Gr\ |\]>)$
- $Q_1 = (<_{[Fa\ Fo\ Go\ Gr\ |]}[Fo\ Gr\ |\ Fa\ Go]>)$
- $Q_2 = (<_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |\ Fa\ Go]}[Fa\ Fo\ Gr\ |\ Go]>)$
- $Q_3 = (<_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |\ Fa\ Go][Fa\ Fo\ Gr\ |\ Go]}[Gr\ |\ Fa\ Fo\ Go]>, <_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |\ Fa\ Go][Fa\ Fo\ Gr\ |\ Go]}[Fo\ |\ Fa\ Go\ Gr]>)$

– Paths to Children:

- $R_4: <_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |\ Fa\ Go][Fa\ Fo\ Gr\ |\ Go]}[Fa\ Fo\ Gr\ |\ Go]>$
- $R_4: <_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |\ Fa\ Go][Fa\ Fo\ Gr\ |\ Go]}[Fa\ Go\ Gr\ |\ Fo]>$
- $Q_4 = (<_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |\ Fa\ Go][Fa\ Fo\ Gr\ |\ Go]}[Fa\ Go\ Gr\ |\ Fo]>, <_{[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |\ Fa\ Go][Fa\ Fo\ Gr\ |\ Go]}[Fo\ |\ Fa\ Go\ Gr]>)$

Depth-first search (queues)

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

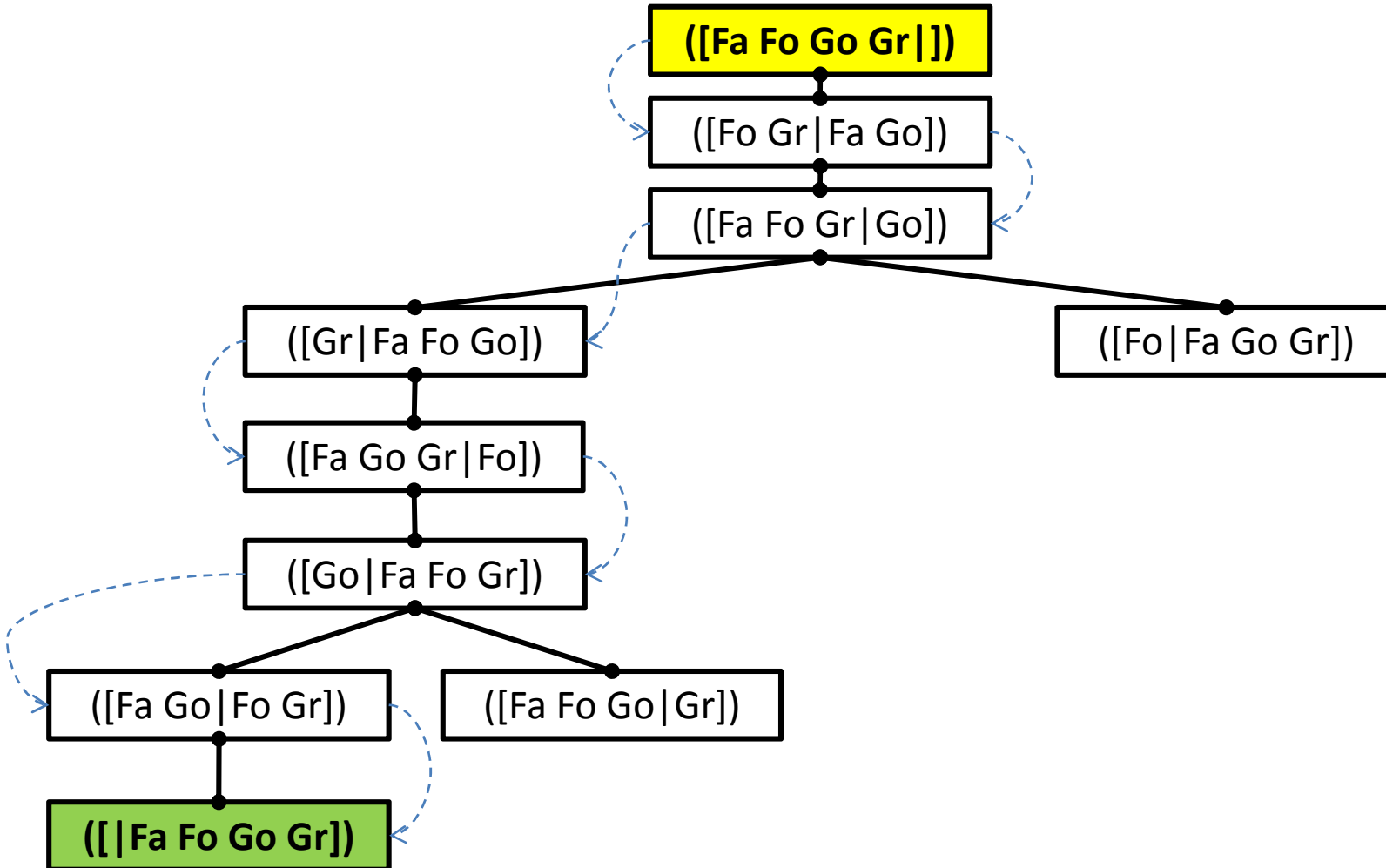
Depth-first search (queues)

- $S = (<[Fa\ Fo\ Go\ Gr\ |]>)$
- $Q_1 = (<[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go]>)$
- $Q_2 = (<[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go]>)$
- $Q_3 = (<[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go]>, <[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Fo\ |Fa\ Go\ Gr]>)$
- $Q_4 = (<[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo]>, <[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Fo\ |Fa\ Go\ Gr]>)$
- $Q_5 = (<[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr]>, <[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Fo\ |Fa\ Go\ Gr]>)$
 - Paths to Children:
 - $R_2: <[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Go\ |Fo\ Gr]>$
 - $R_4: <[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Fo\ Go\ |Gr]>$
 - $R_4: <[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Go\ Gr\ |Fo]>$
- $Q_6 = (<[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Go\ |Fo\ Gr]>, <[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Gr\ |Fa\ Fo\ Go][Fa\ Go\ Gr\ |Fo][Go\ |Fa\ Fo\ Gr][Fa\ Fo\ Go\ |Gr]>, <[Fa\ Fo\ Go\ Gr\ |][Fo\ Gr\ |Fa\ Go][Fa\ Fo\ Gr\ |Go][Fo\ |Fa\ Go\ Gr]>)$

Depth-first search (queues)

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

Depth-first search (search tree)



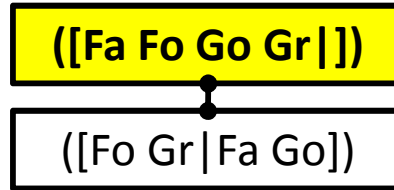
Farmer, Fox, Goose and Grain

BREADTH-FIRST SEARCH

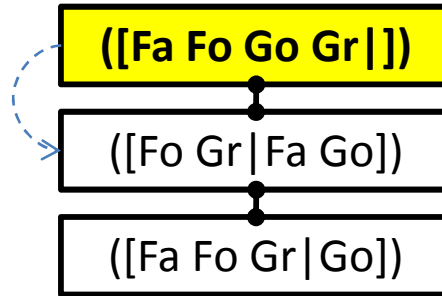
Breadth-first search (queues)

- ***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

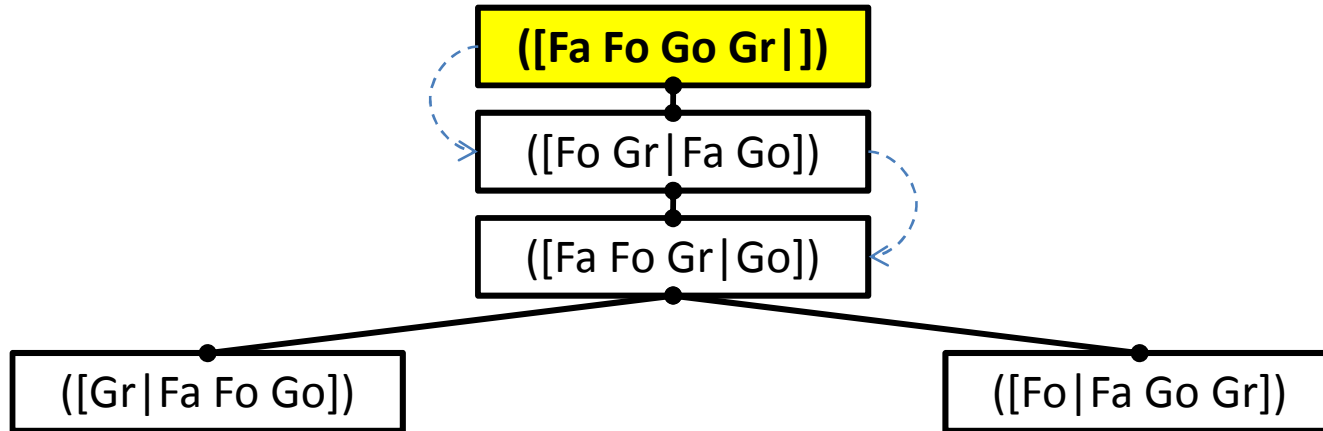
Breadth-first search (search tree)



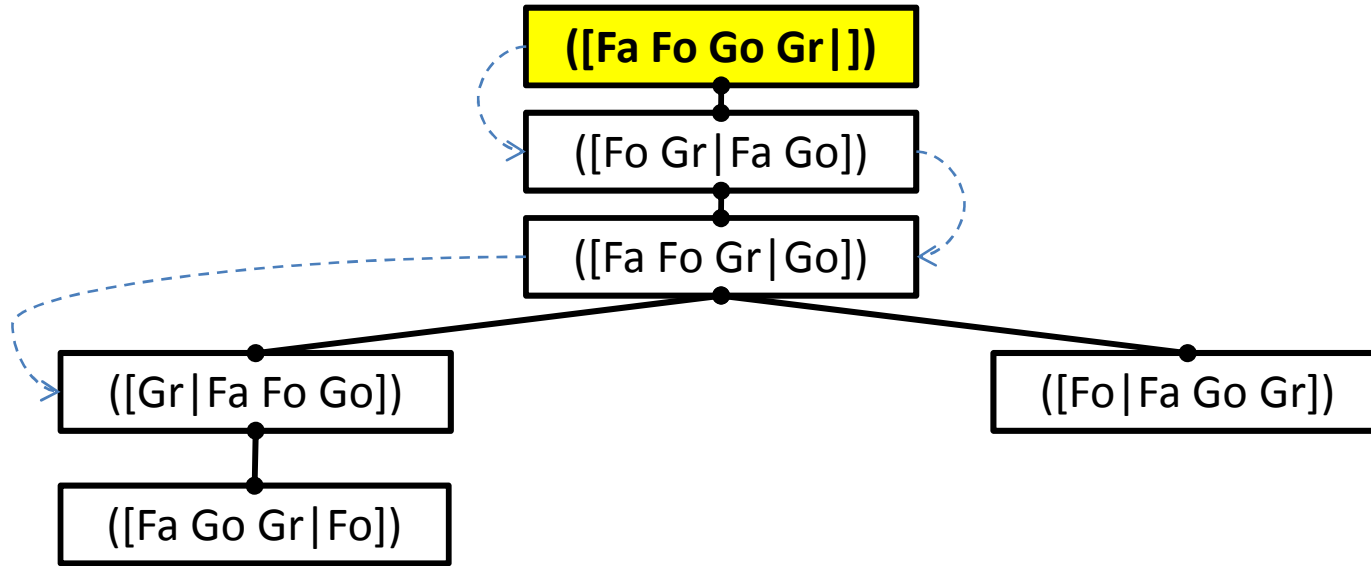
Breadth-first search (search tree)



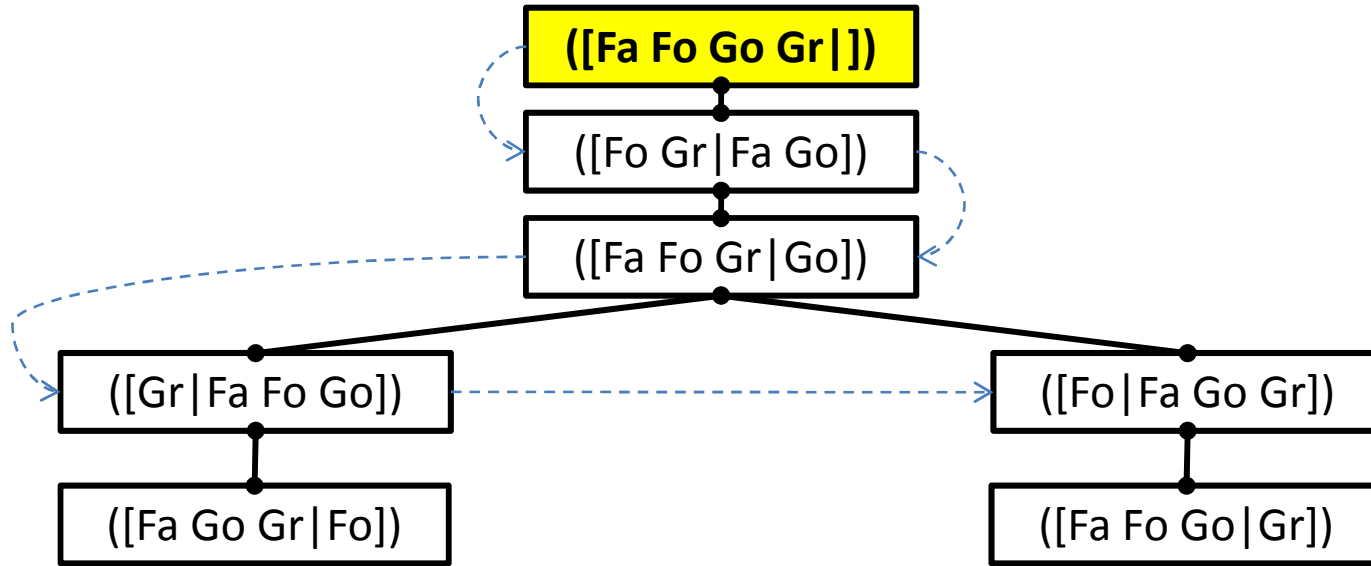
Breadth-first search (search tree)



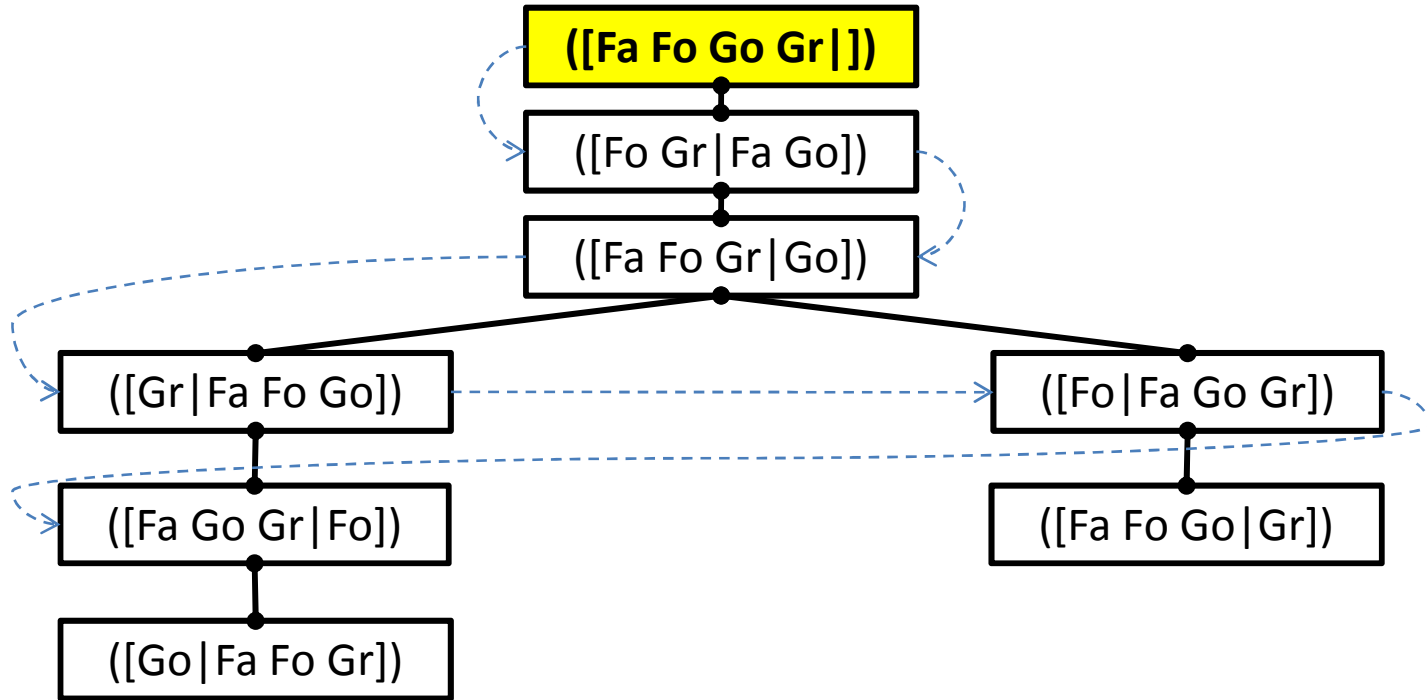
Breadth-first search (search tree)



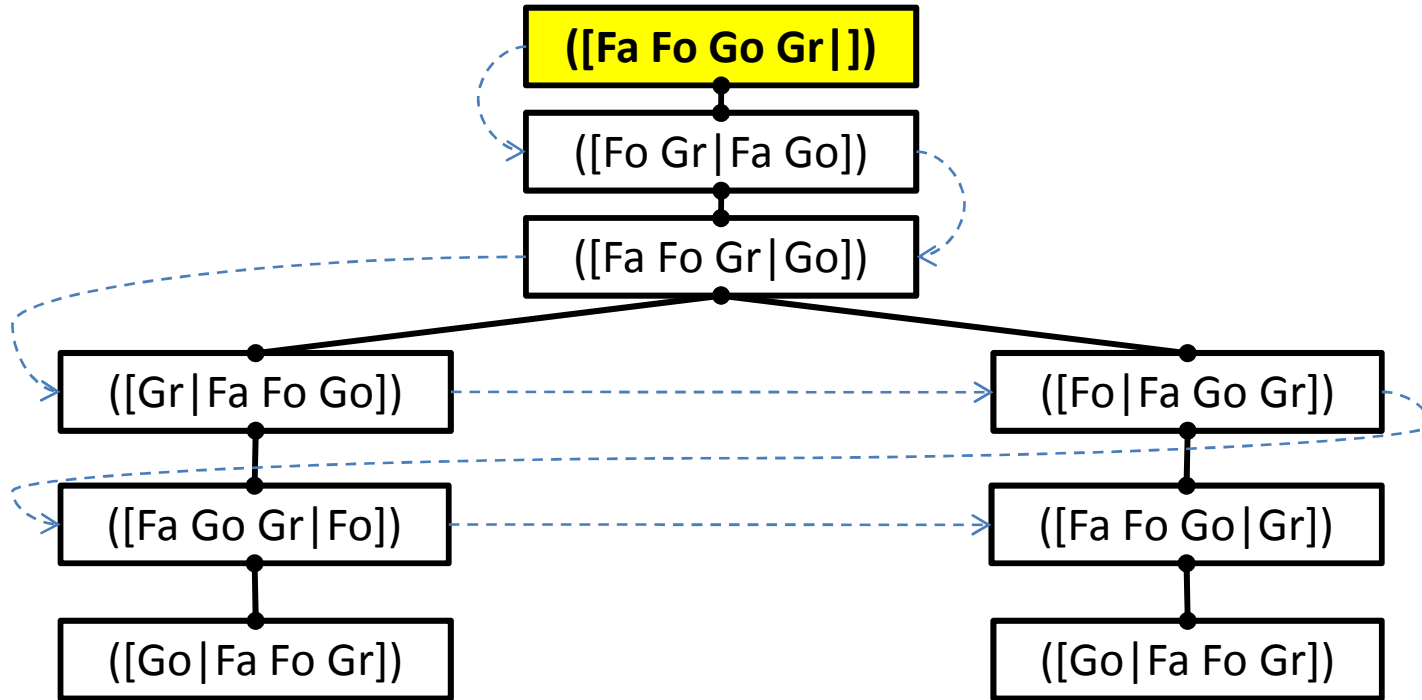
Breadth-first search (search tree)



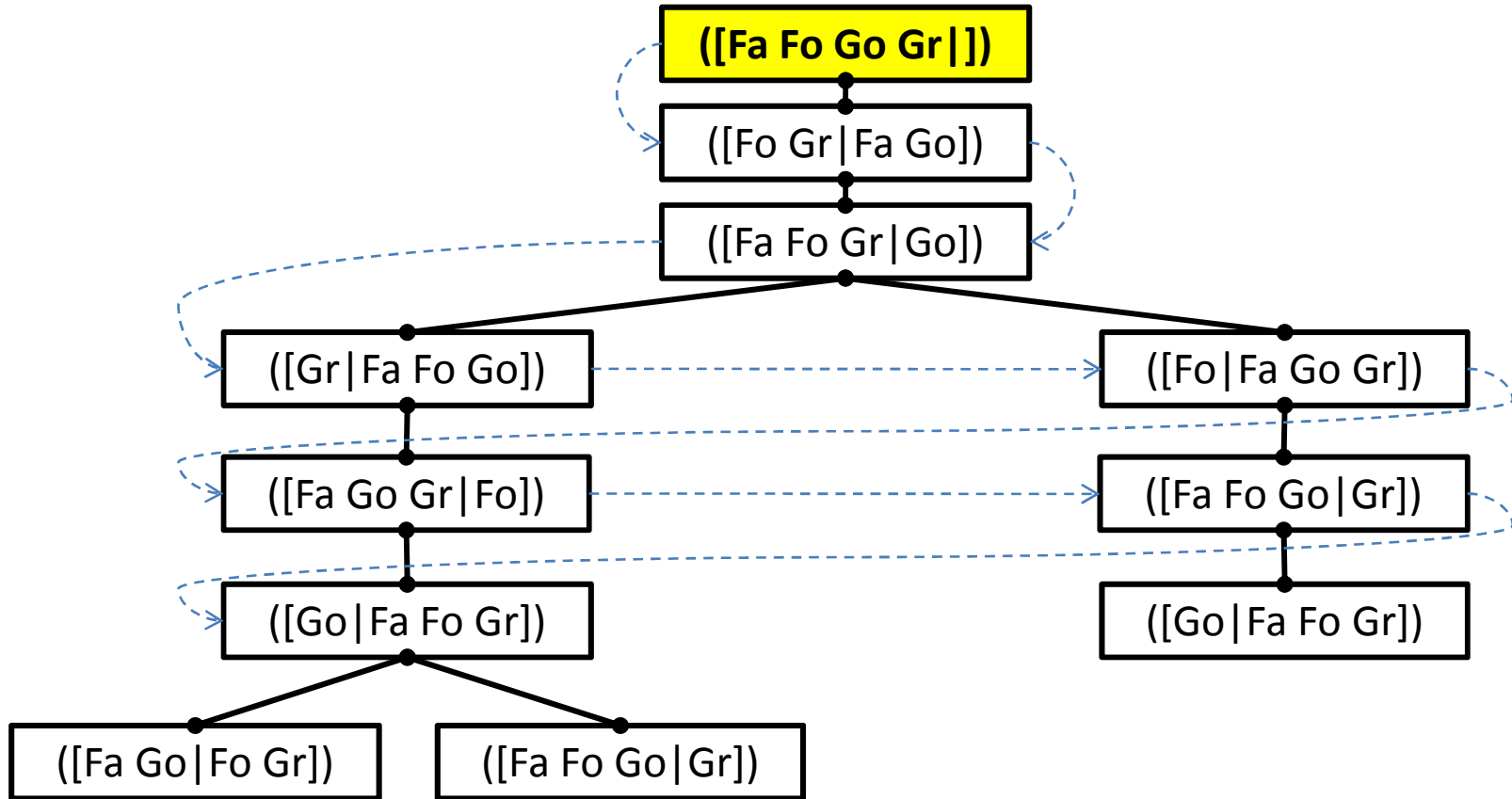
Breadth-first search (search tree)



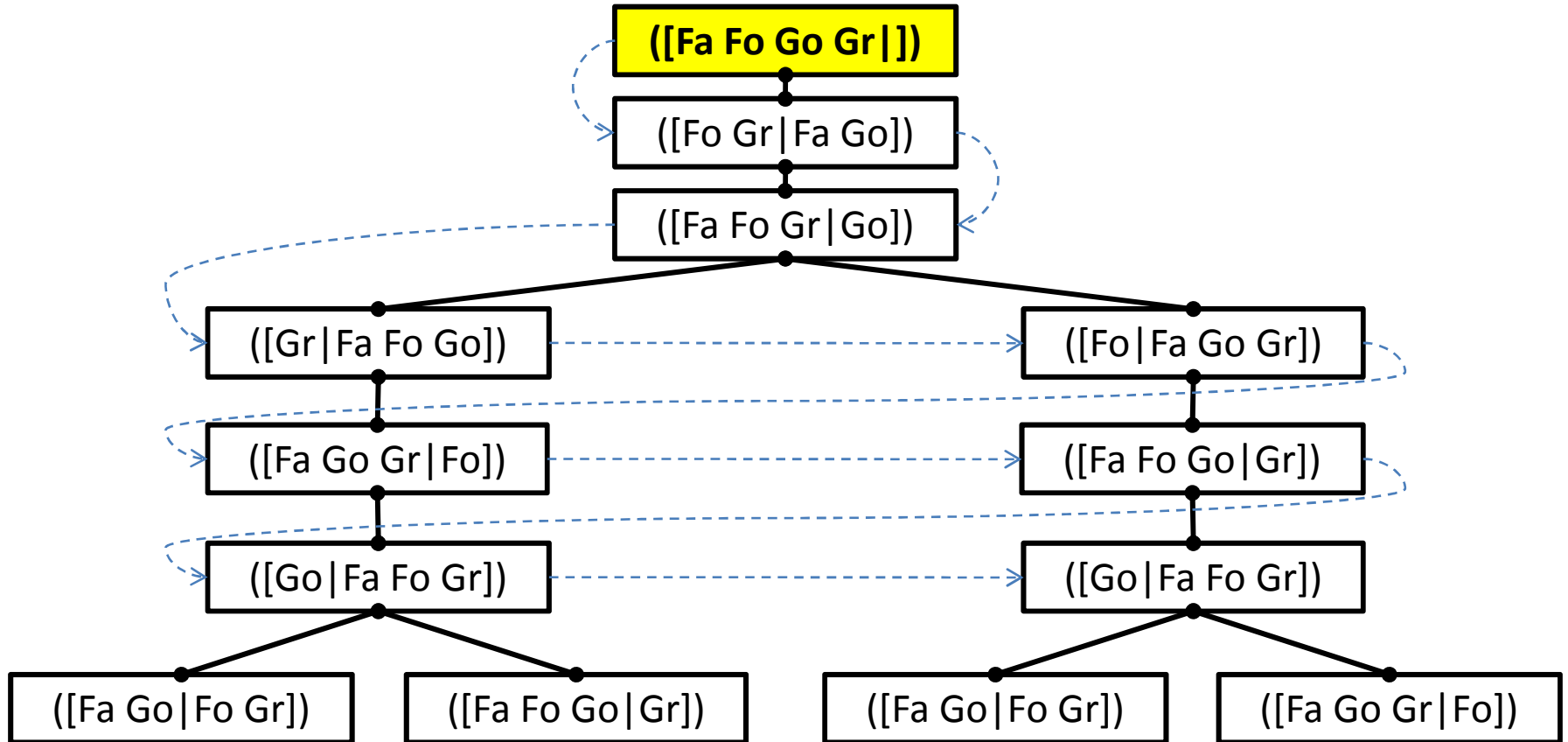
Breadth-first search (search tree)



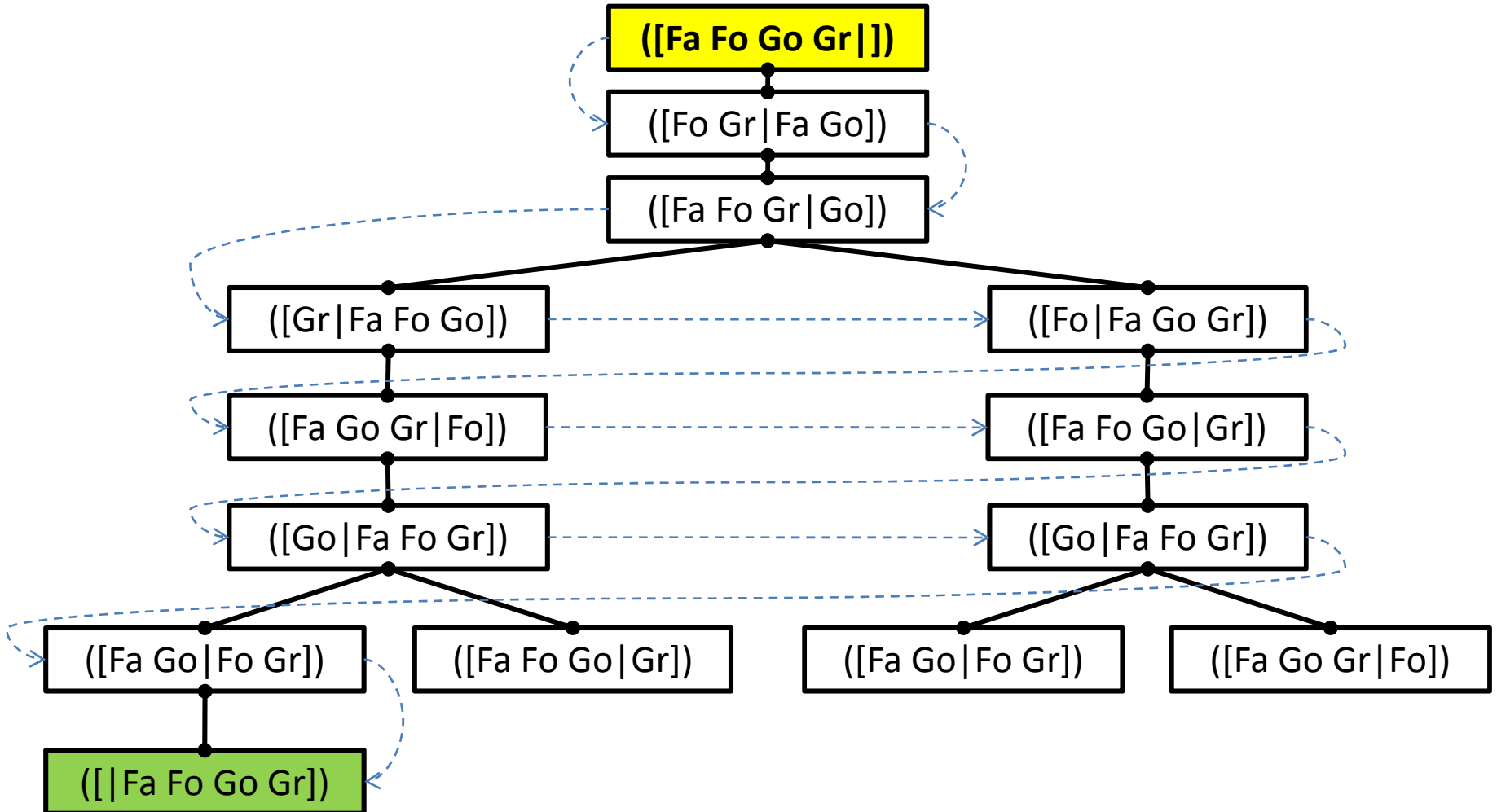
Breadth-first search (search tree)



Breadth-first search (search tree)



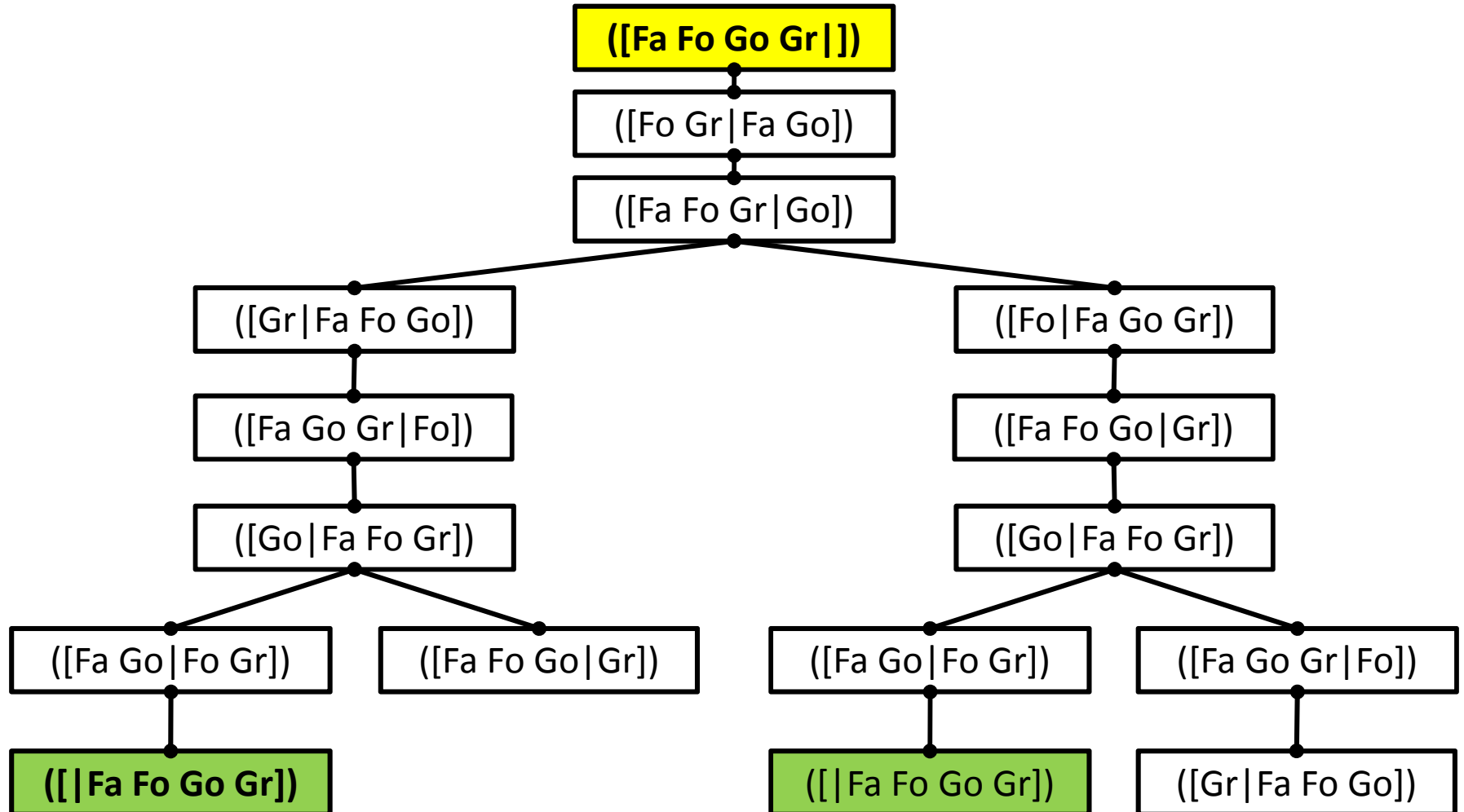
Breadth-first search (search tree)



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

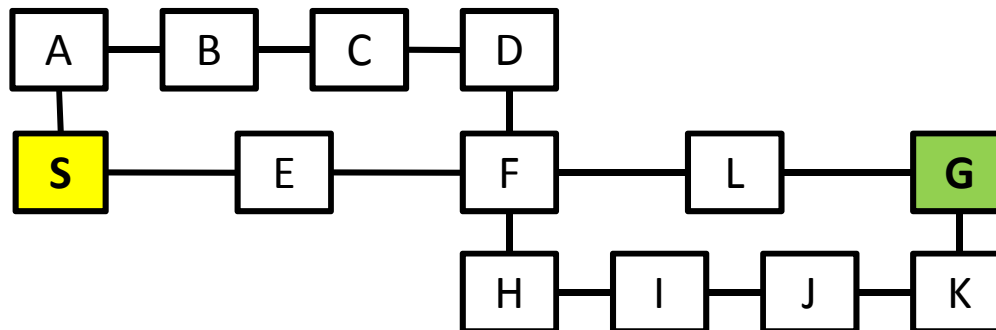
PROBLEM 1: BREADTH-FIRST?

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.

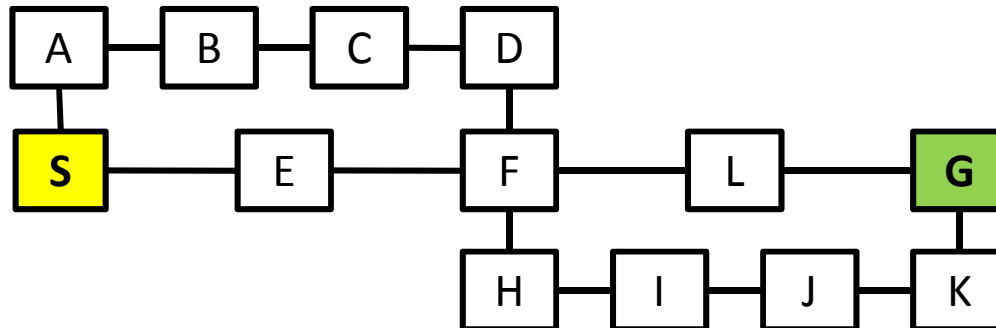
2 x Depth-first

- 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 (**$\langle SABCD\underline{F} \rangle, \langle SE \rangle$**)
- 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 ($\langle GKJIH \rangle, \langle GL \rangle$) \rightarrow (**$\langle GKJIH\underline{F} \rangle, \langle GL \rangle$**)



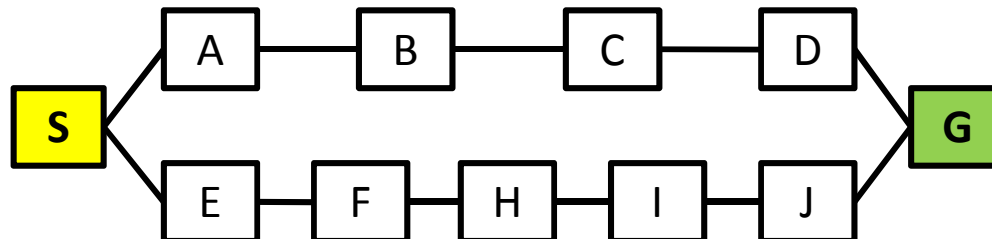
2 x Breadth-first

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

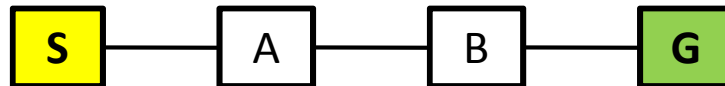


Bidirectional Search

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

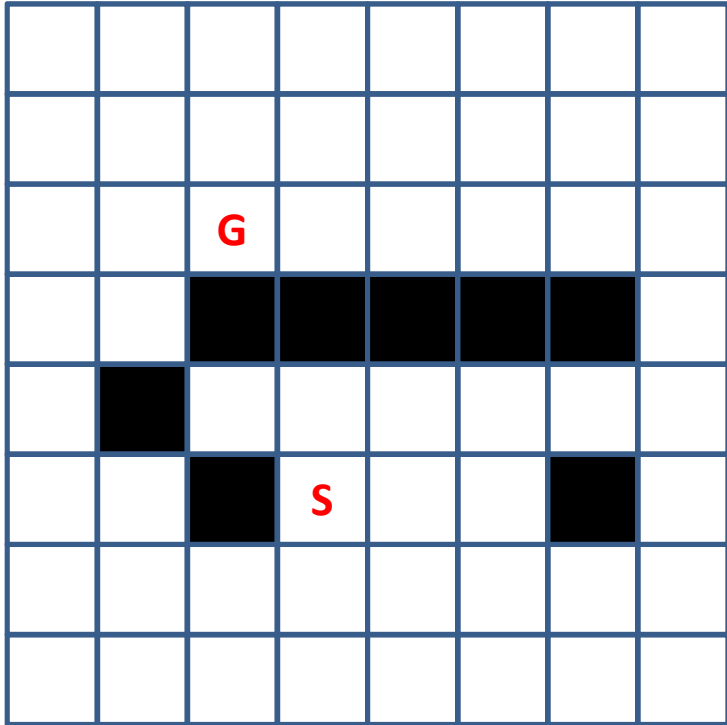
Beam Search

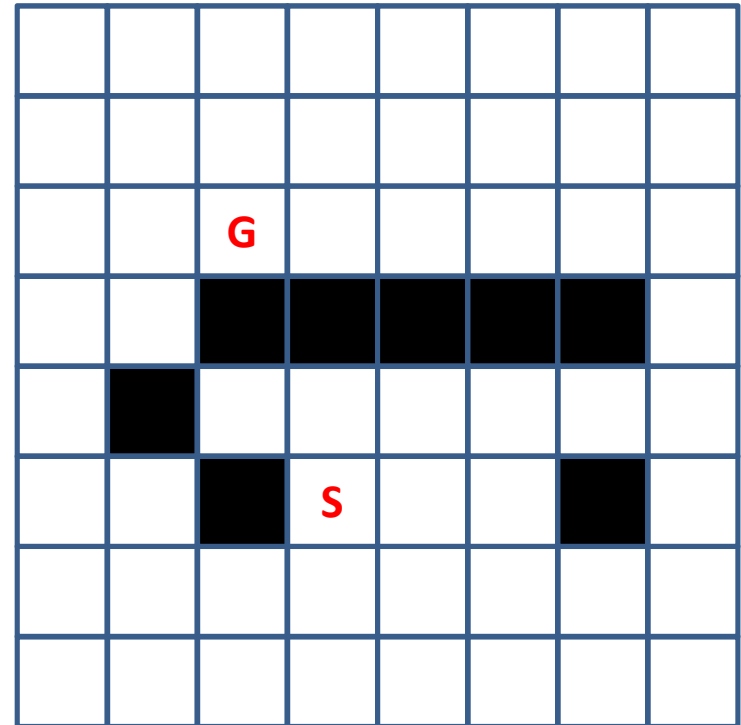
- ***Input:***
 - **QUEUE:** Path only containing root
 - **WIDTH:** Number
- ***Algorithm:***
 - **WHILE** (QUEUE not empty && goal not reached) **DO**
 - Remove **all paths** from QUEUE
 - Create paths to all children (of all paths)
 - Reject paths with loops
 - **Sort new paths (according to heuristic)**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

Exercises: Artificial Intelligence

Path Search

Problem

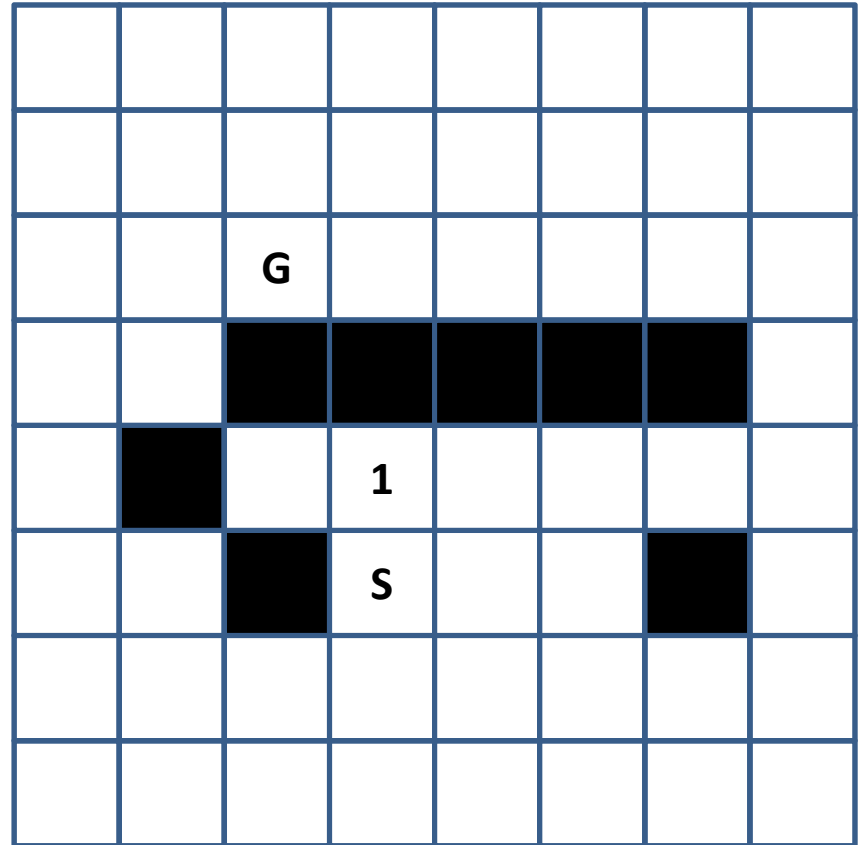
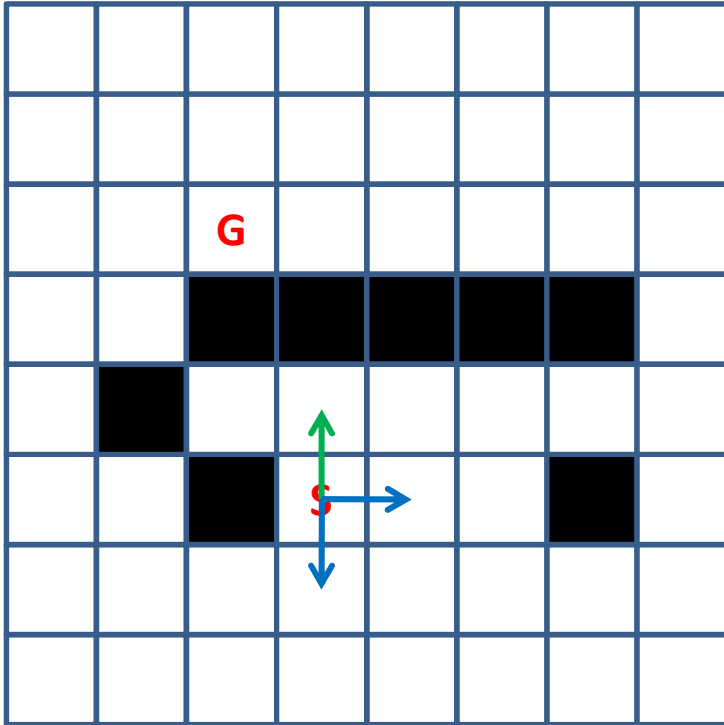
- Find a path from 'S' to 'G', without passing through black squares.
 - Legal steps (order):
 - up, left, right, down
 - Perform:
 - Depth-first search
 - Hill-climbing I Search
 - With suitable heuristic
 - Greedy Search
 - With same heuristic
- 
- | | | | | | | | |
|--|---|---|---|---|---|---|--|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



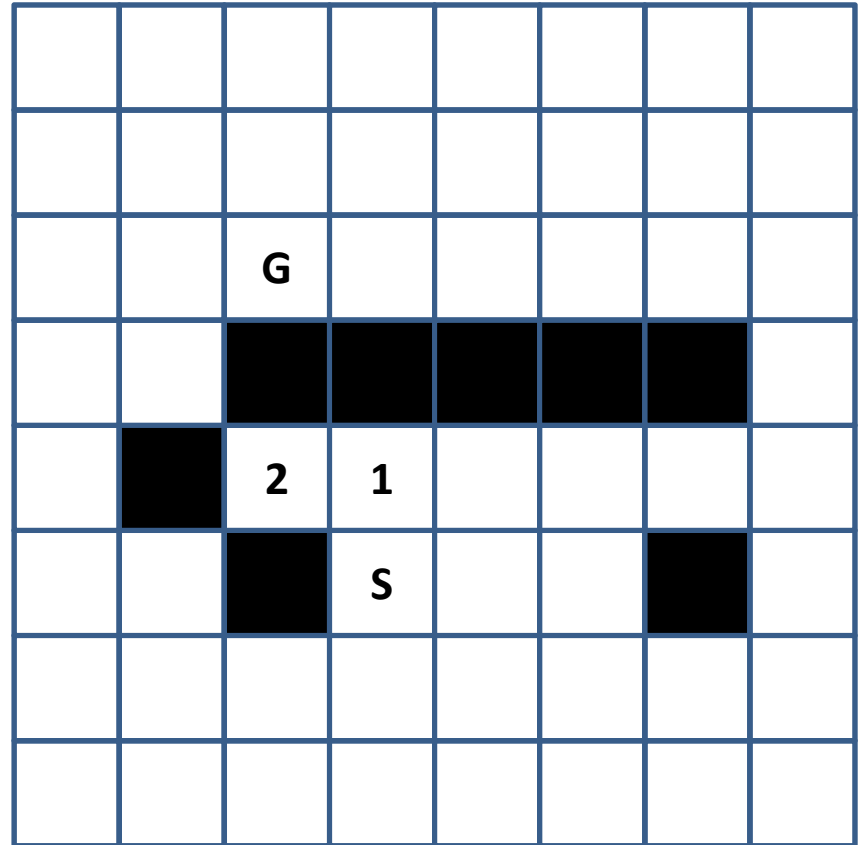
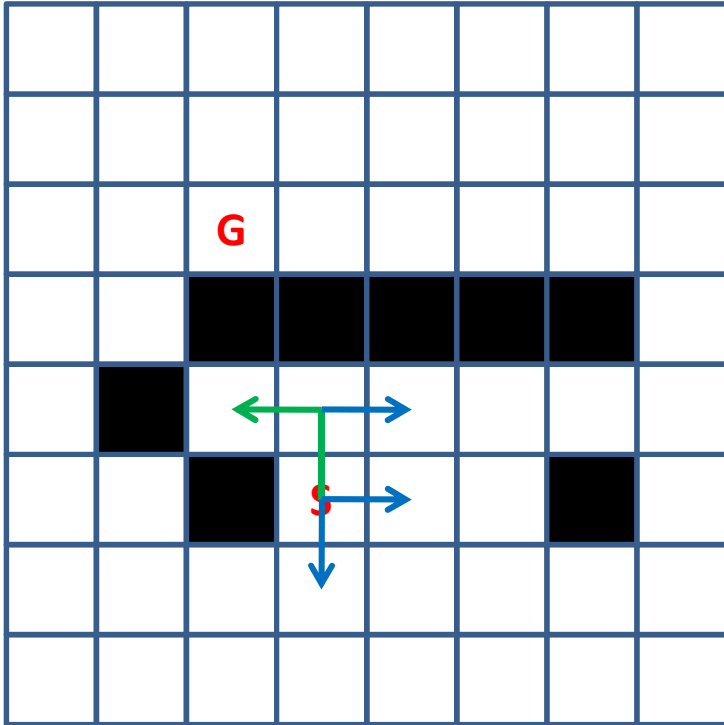
Path Search

DEPTH-FIRST SEARCH

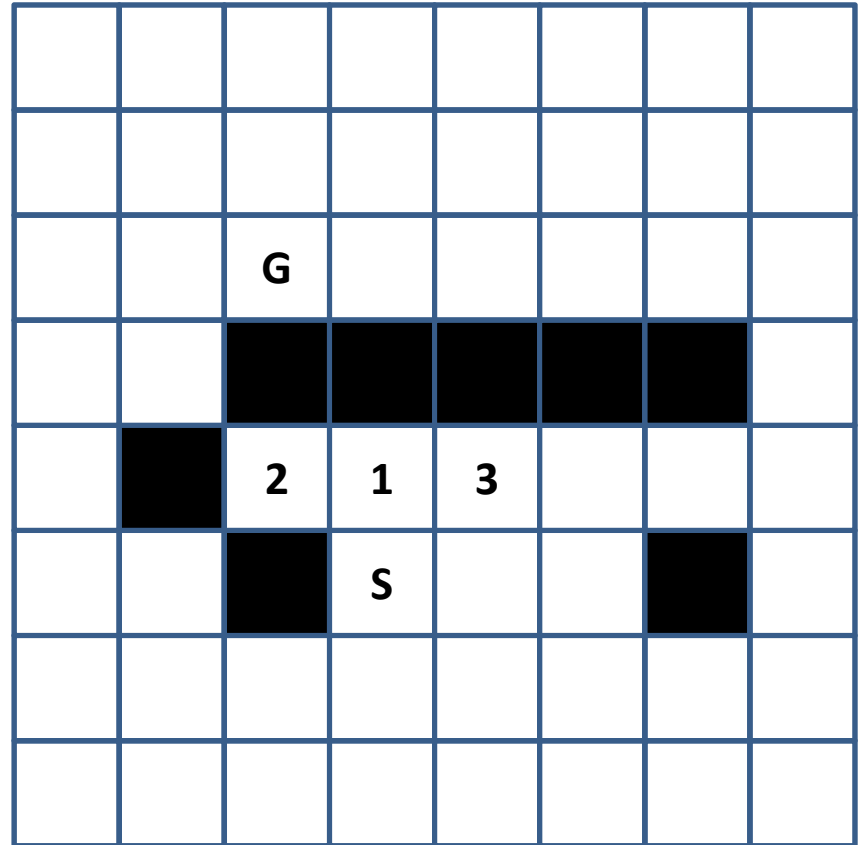
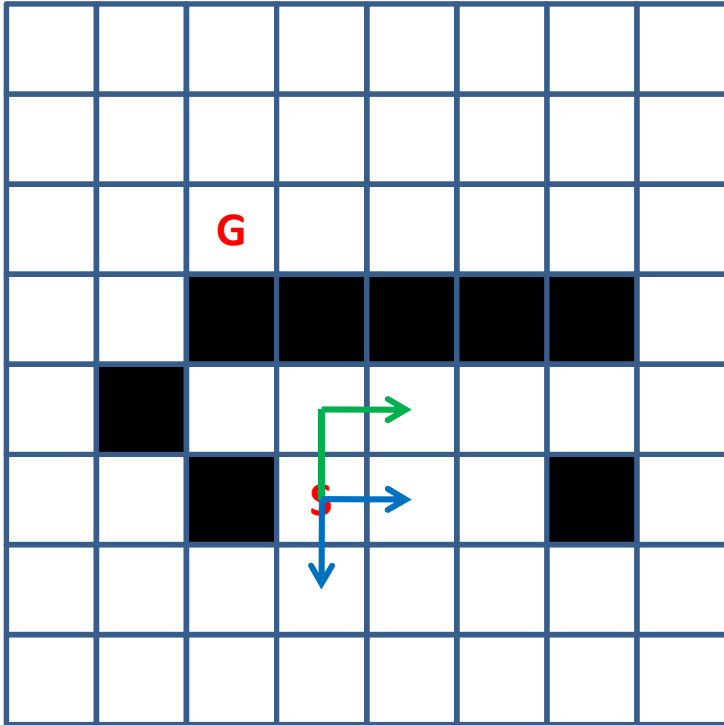
Depth-first Search



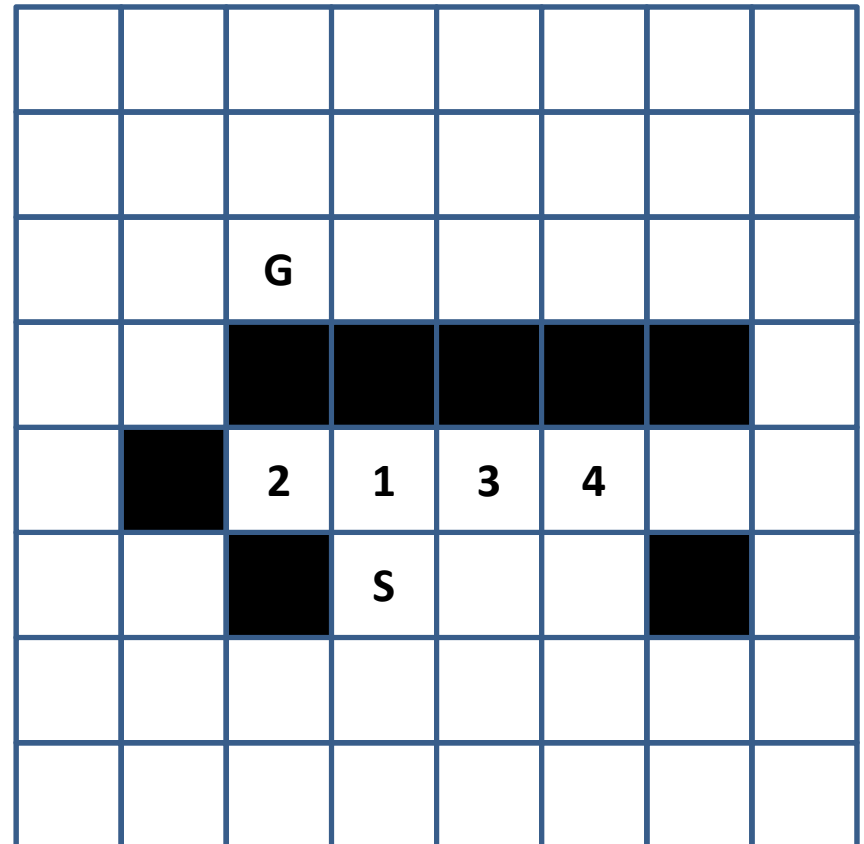
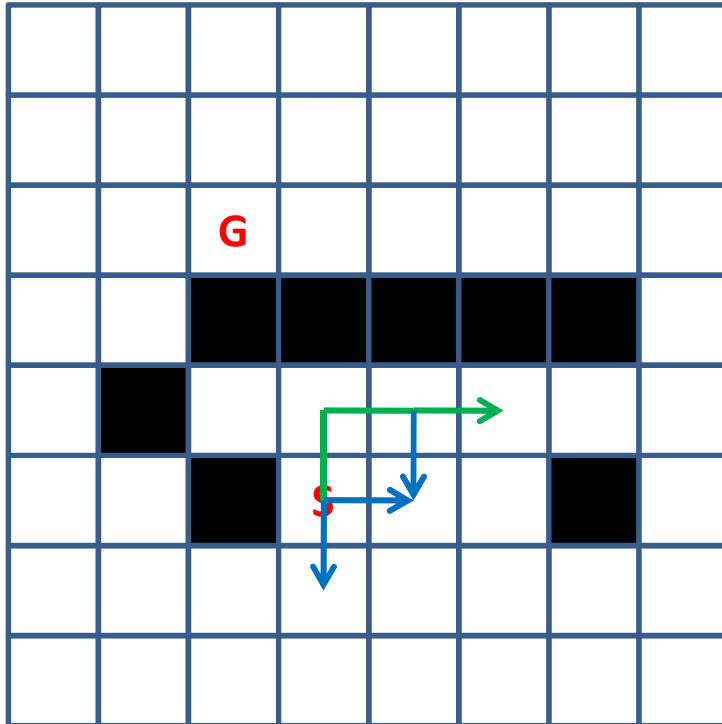
Depth-first Search



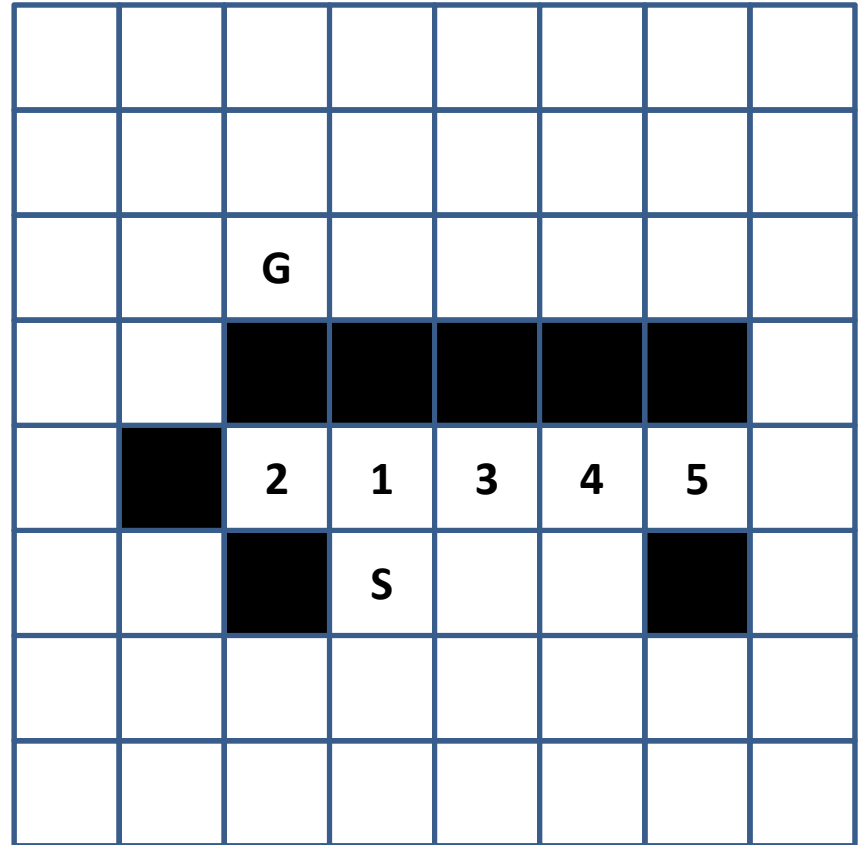
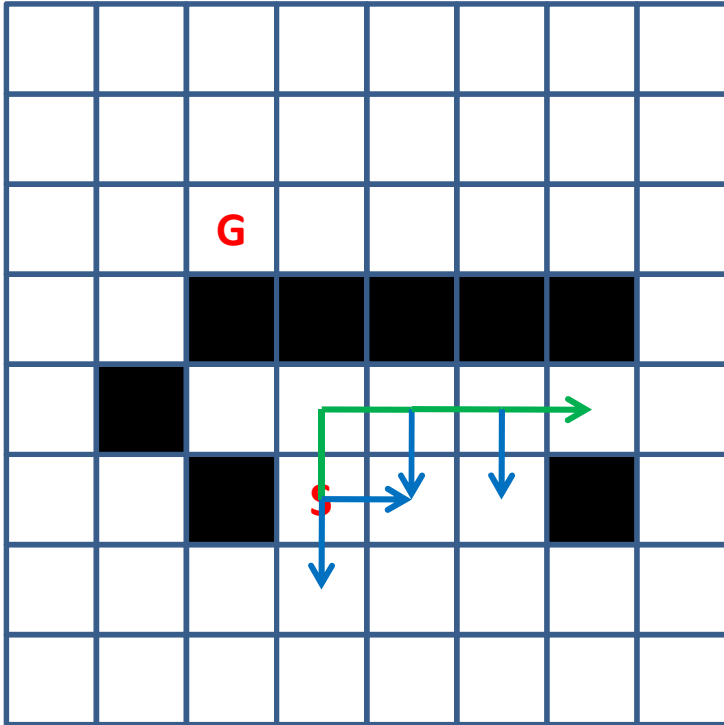
Depth-first Search



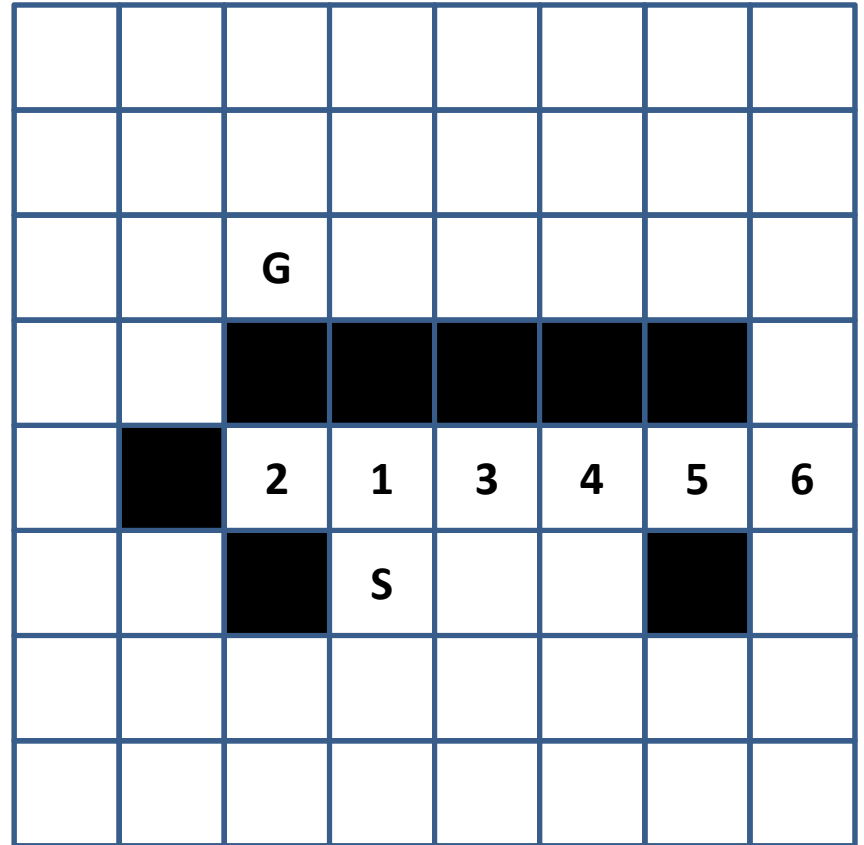
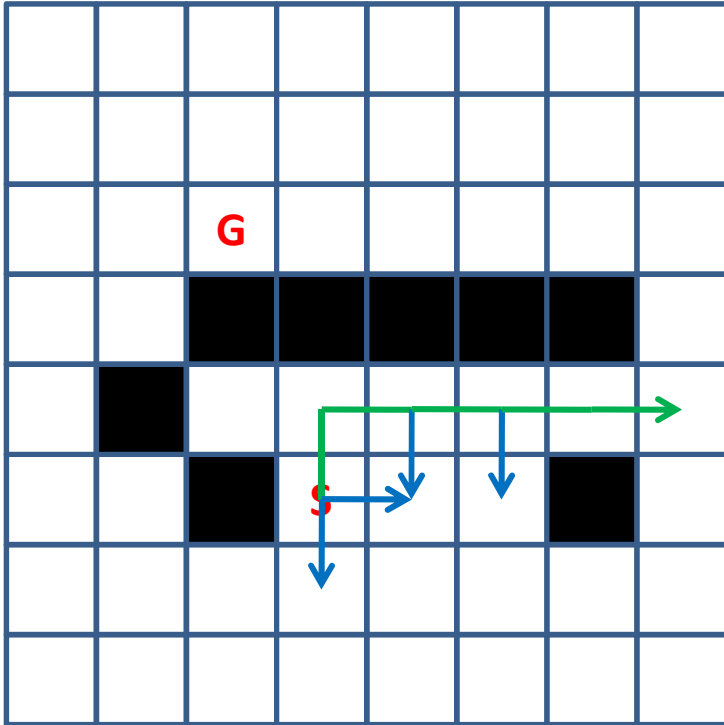
Depth-first Search



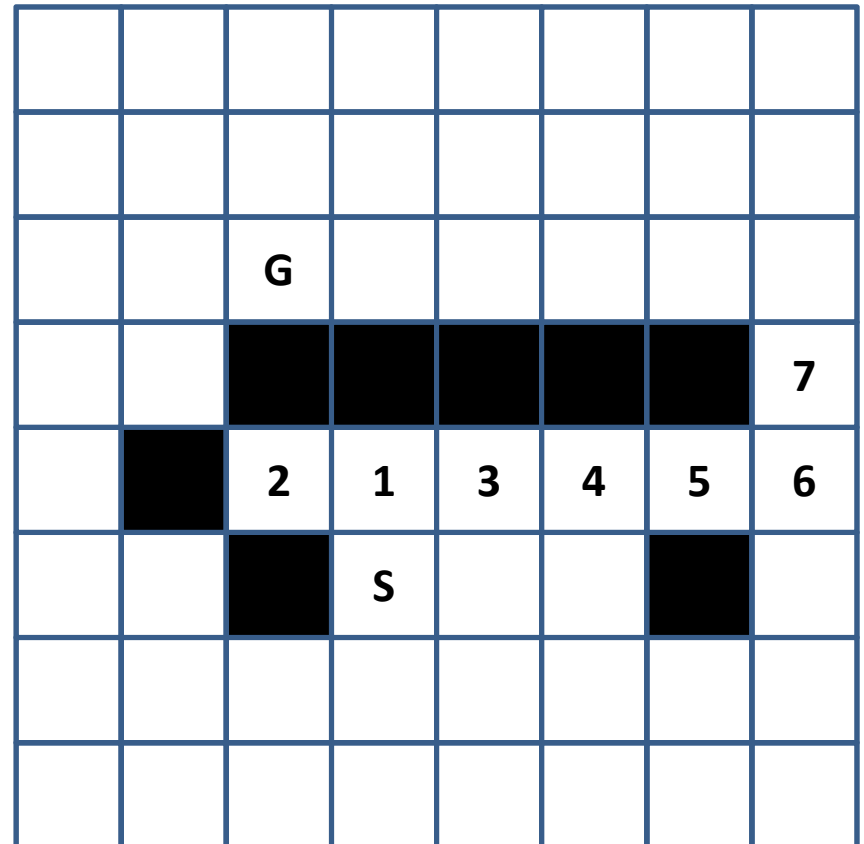
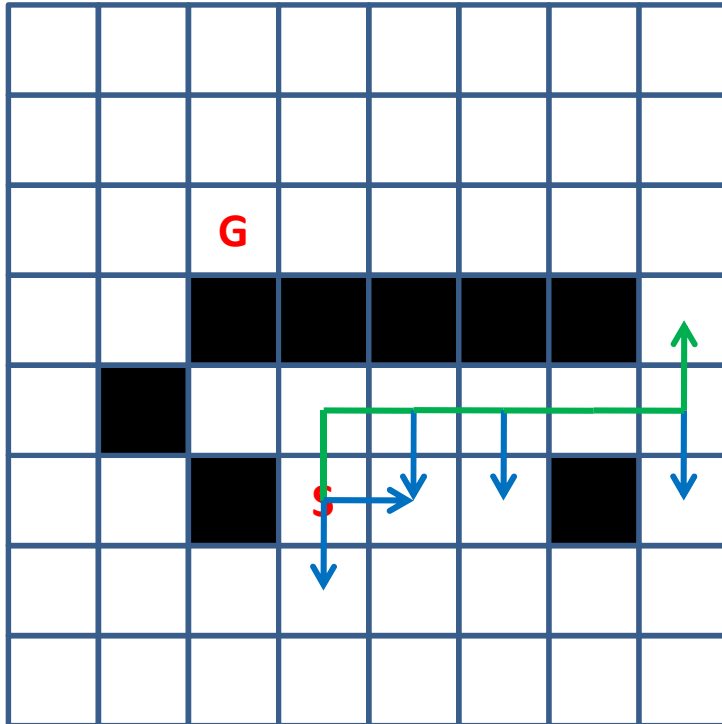
Depth-first Search



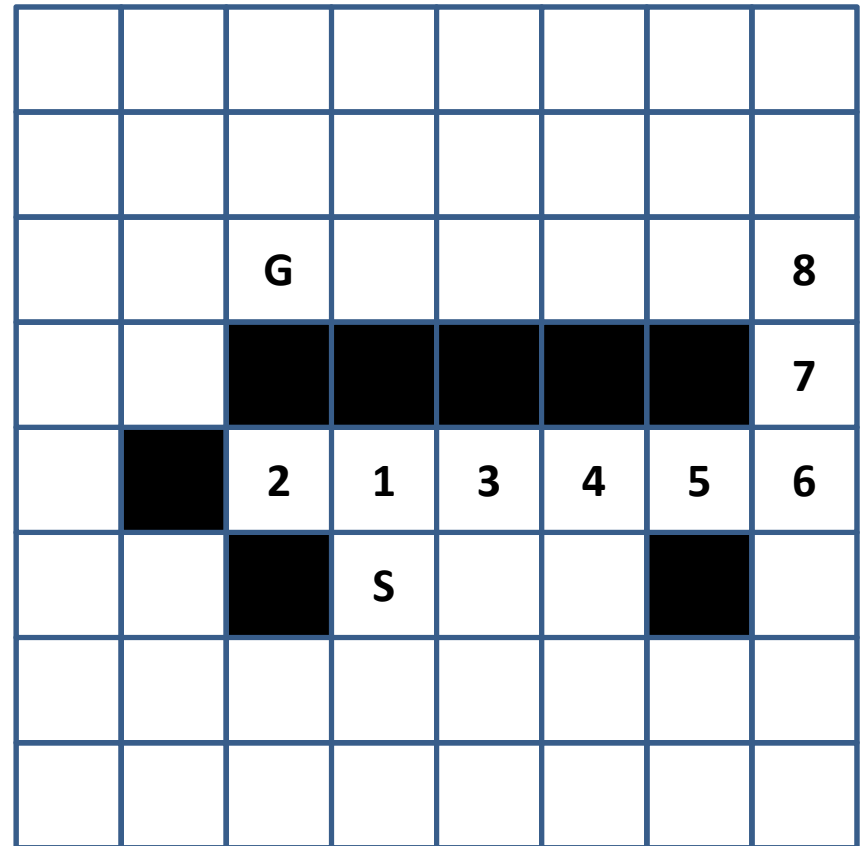
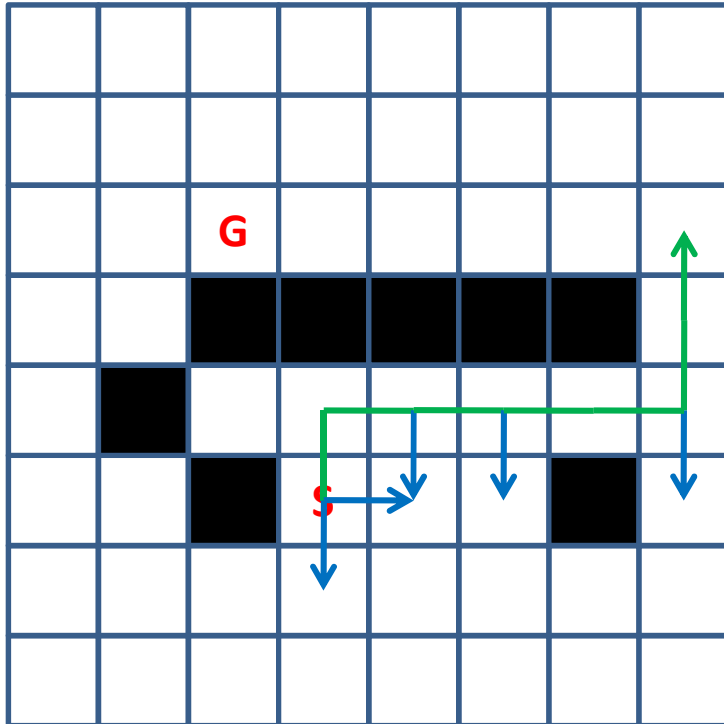
Depth-first Search



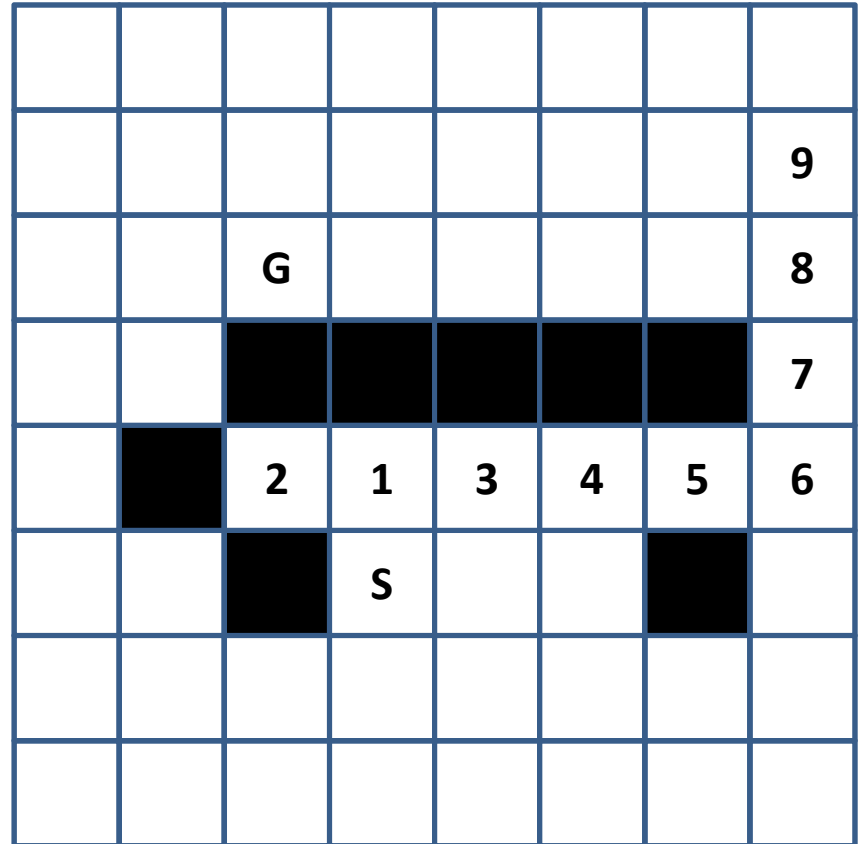
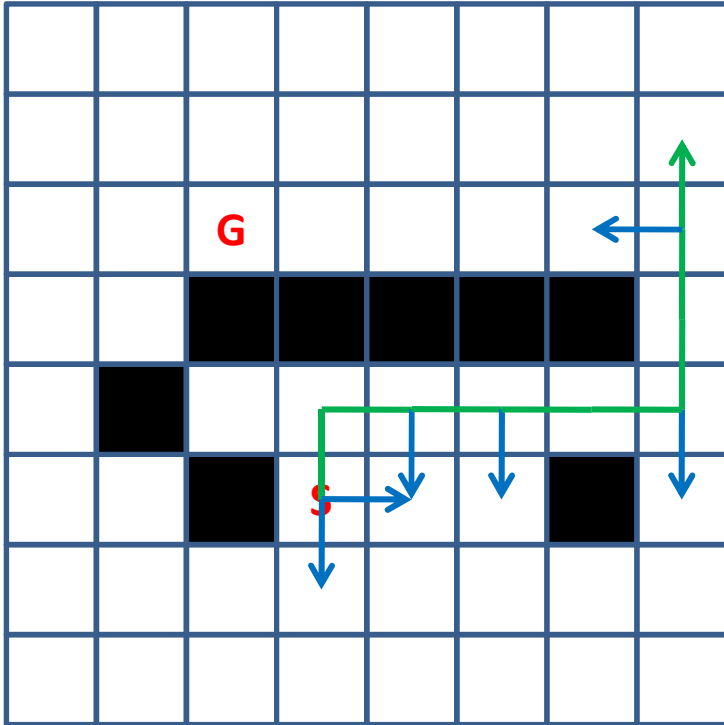
Depth-first Search



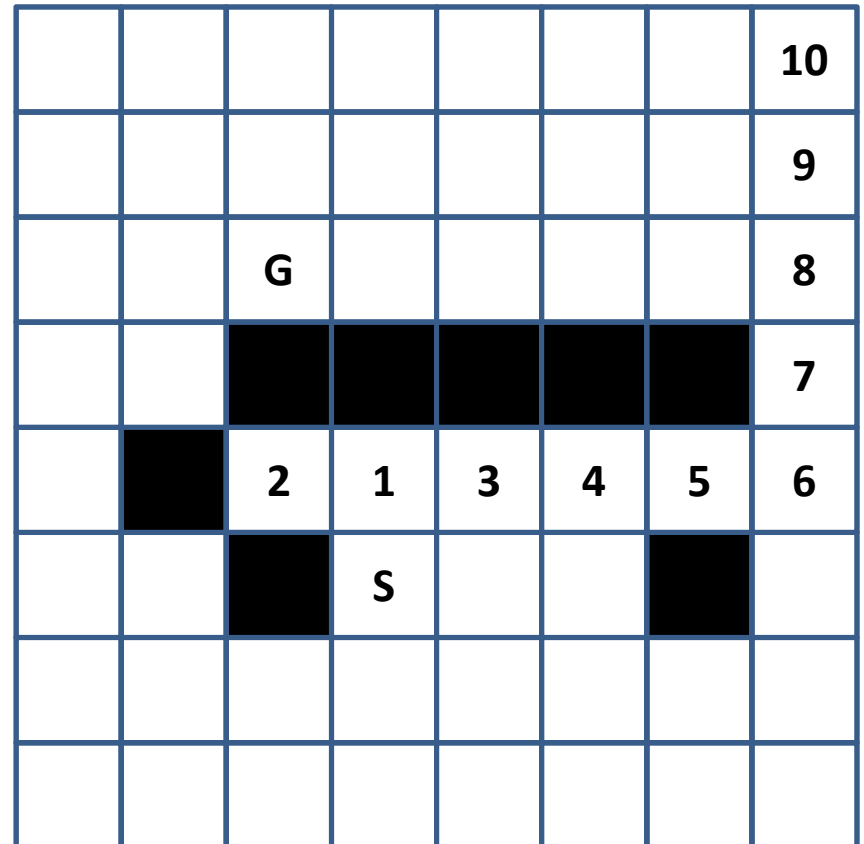
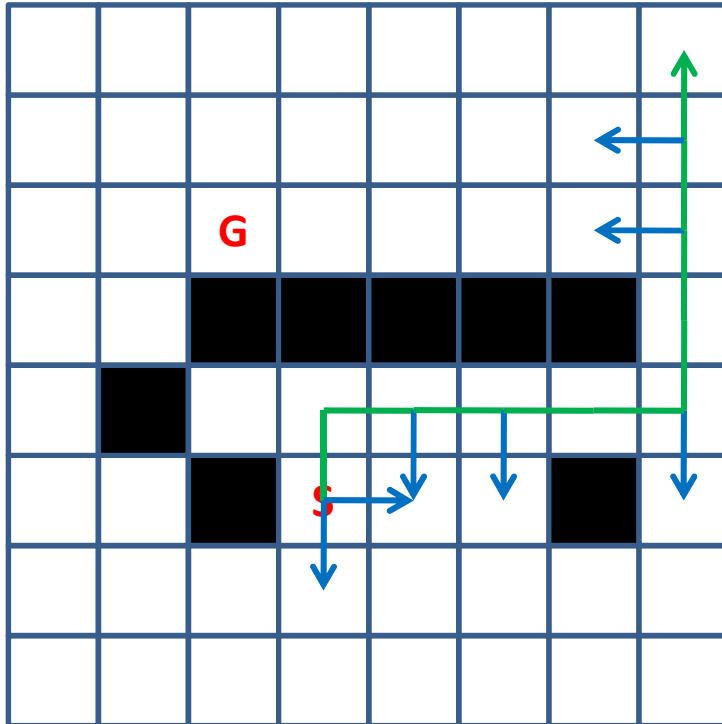
Depth-first Search



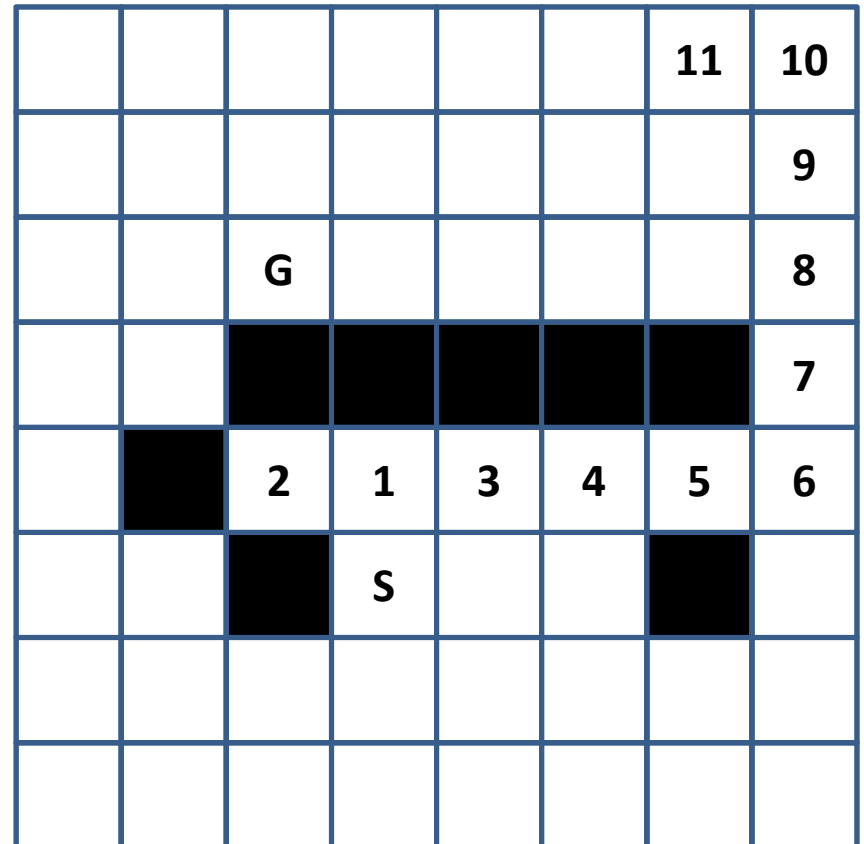
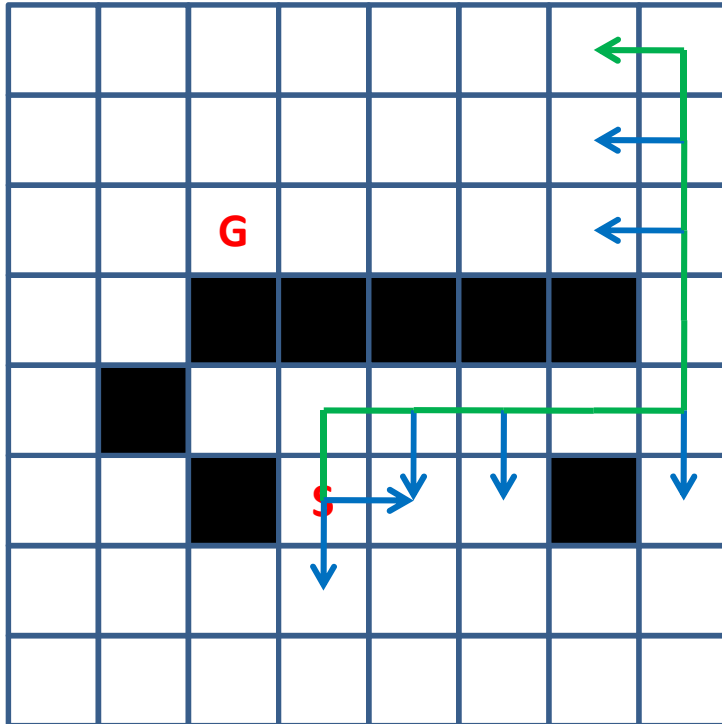
Depth-first Search



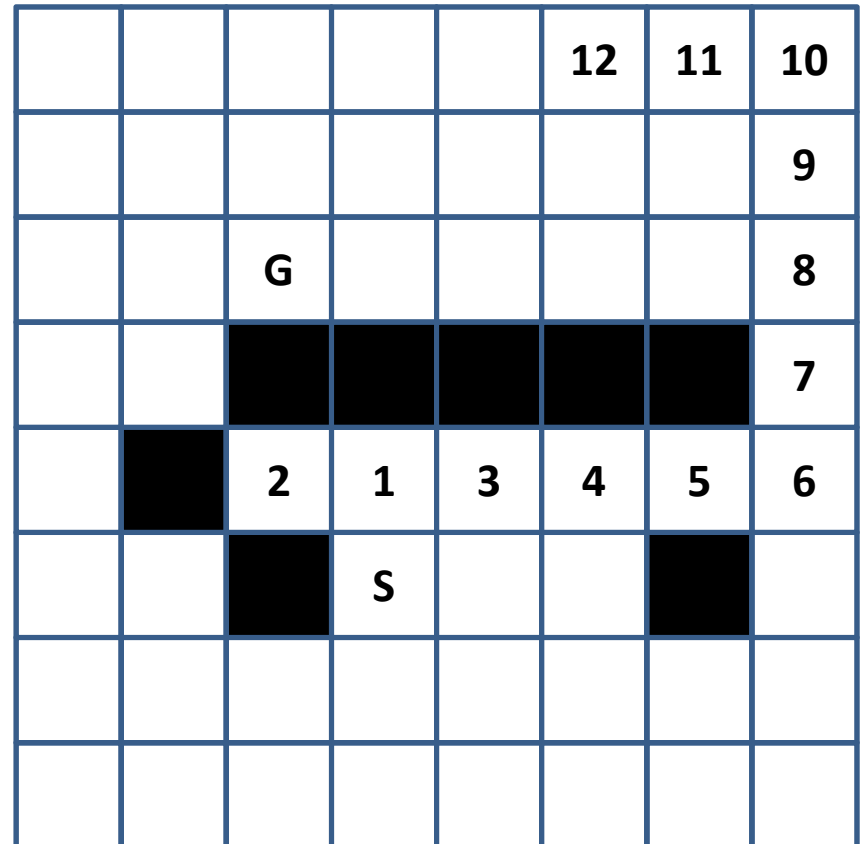
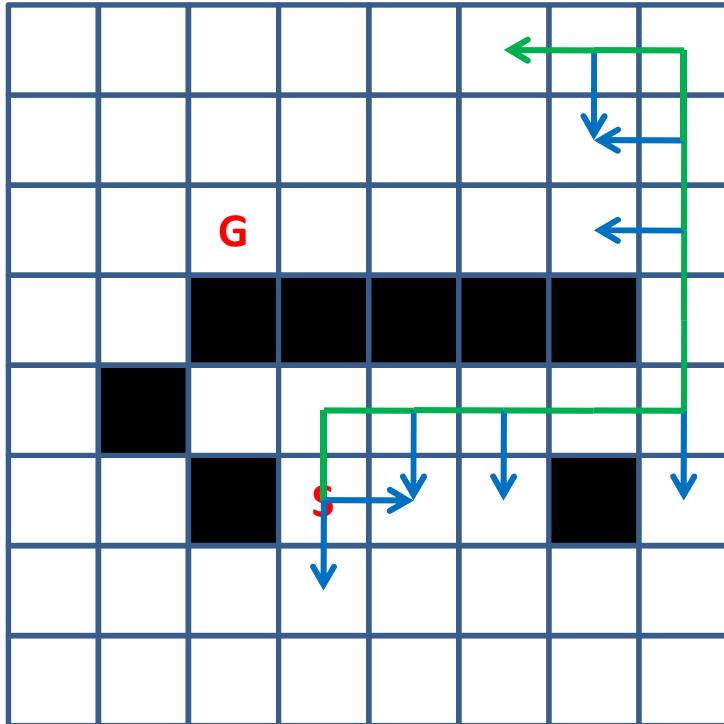
Depth-first Search



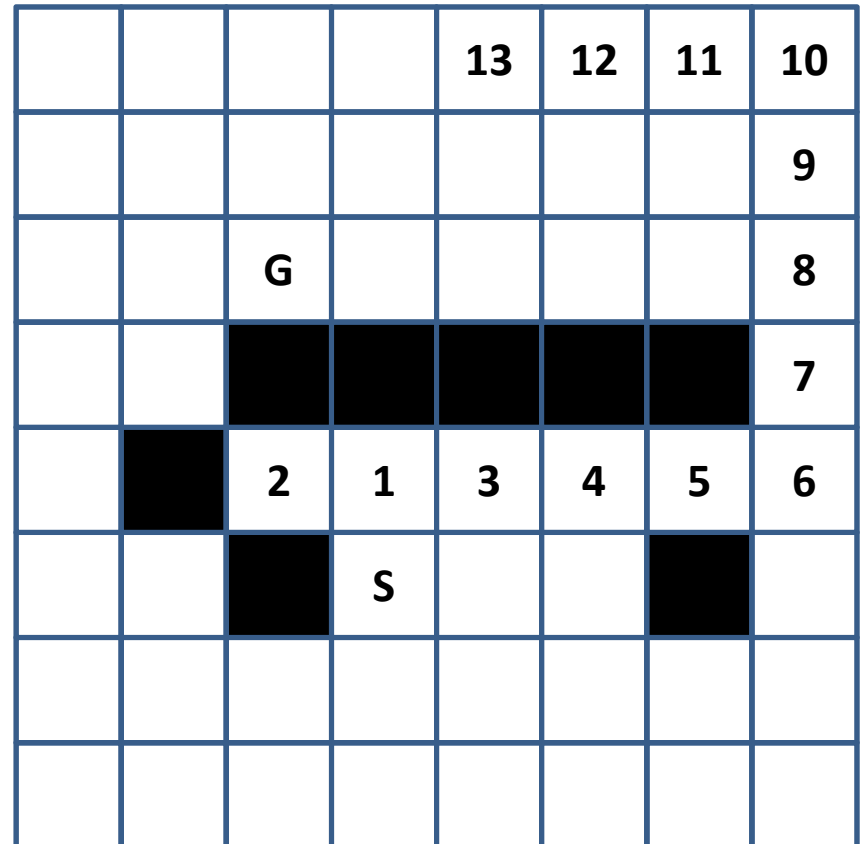
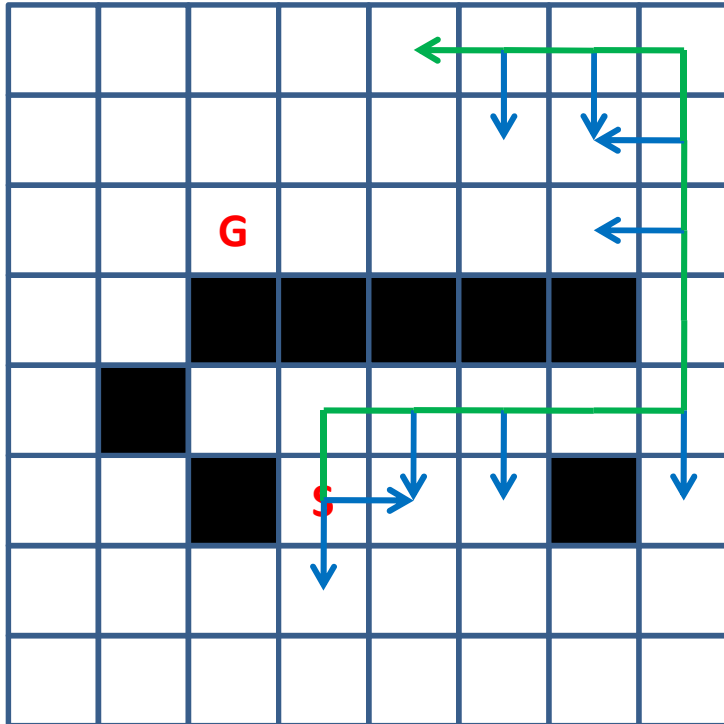
Depth-first Search



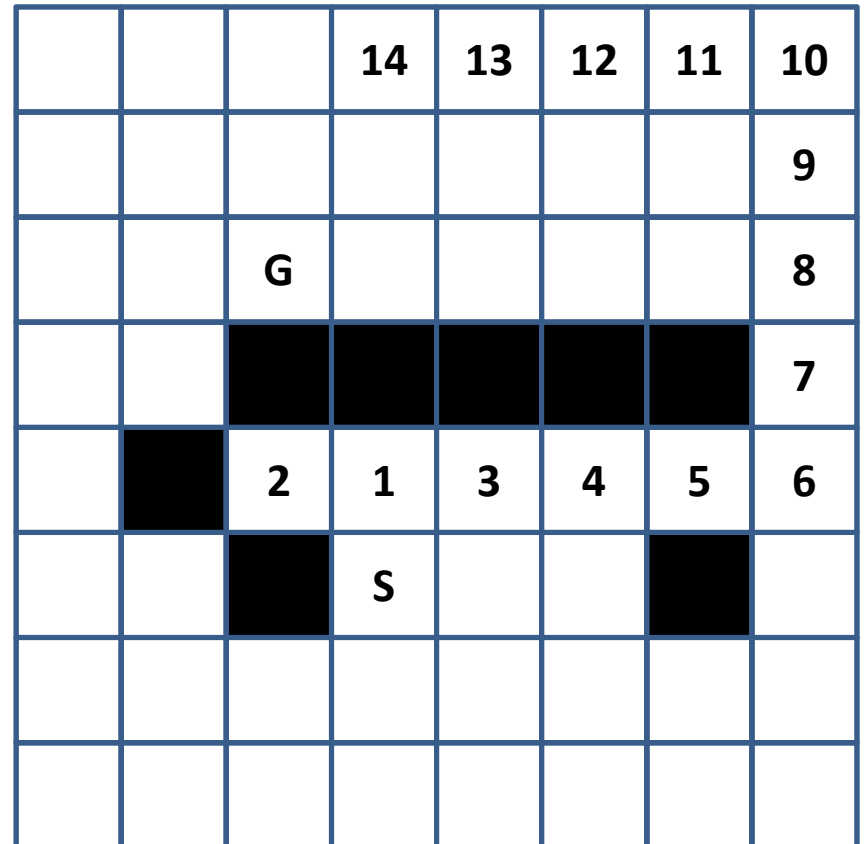
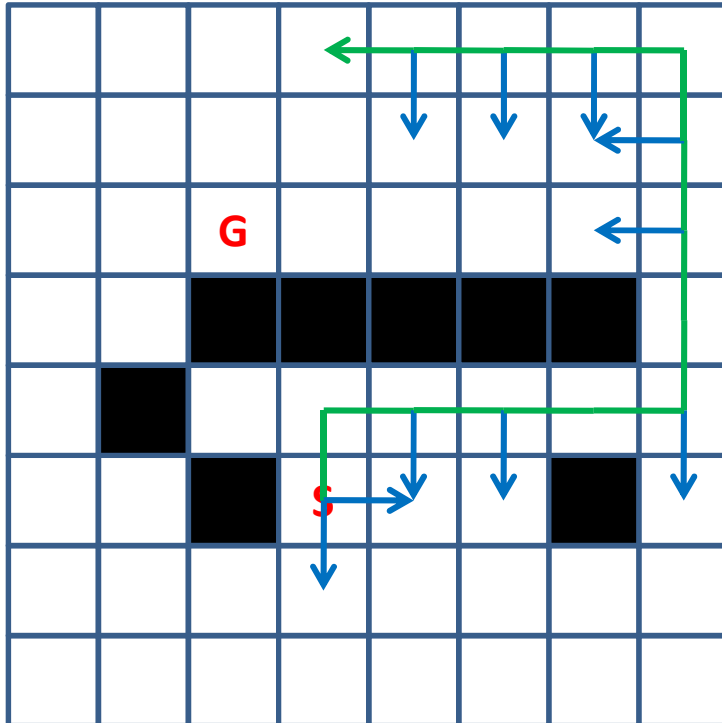
Depth-first Search



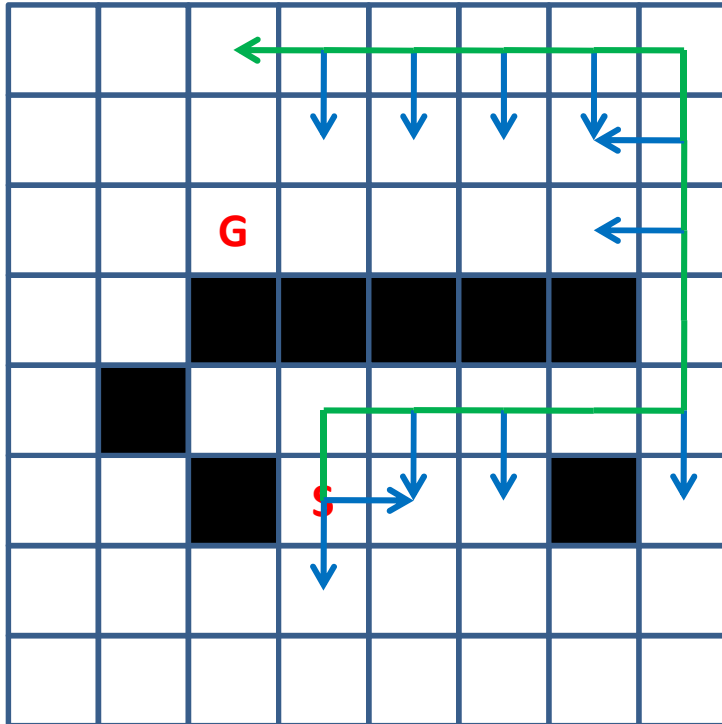
Depth-first Search



Depth-first Search

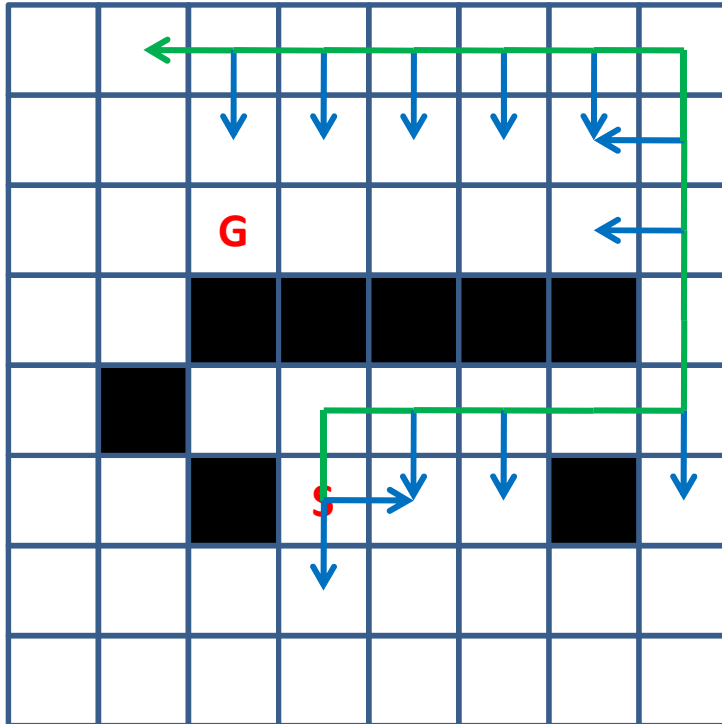


Depth-first Search



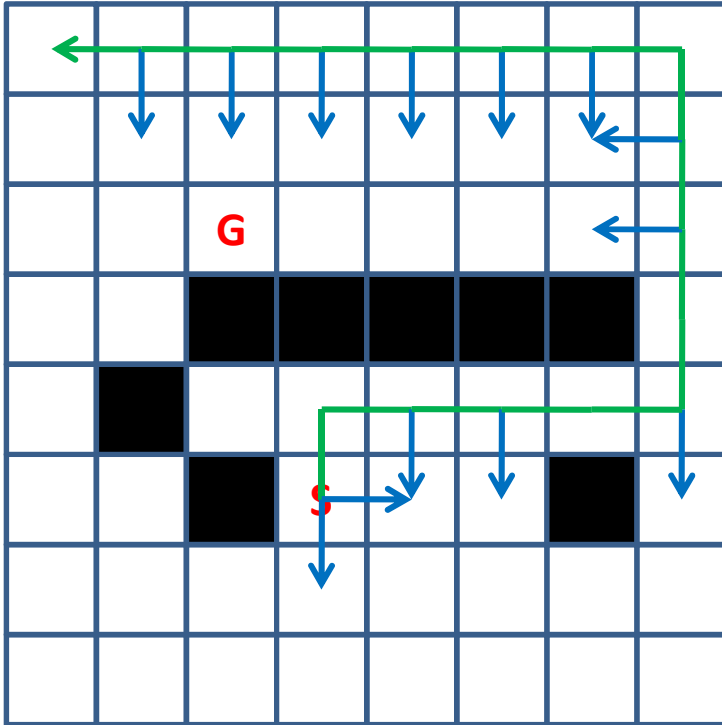
		15	14	13	12	11	10
							9
		G					8
							7
		2	1	3	4	5	6
			S				

Depth-first Search



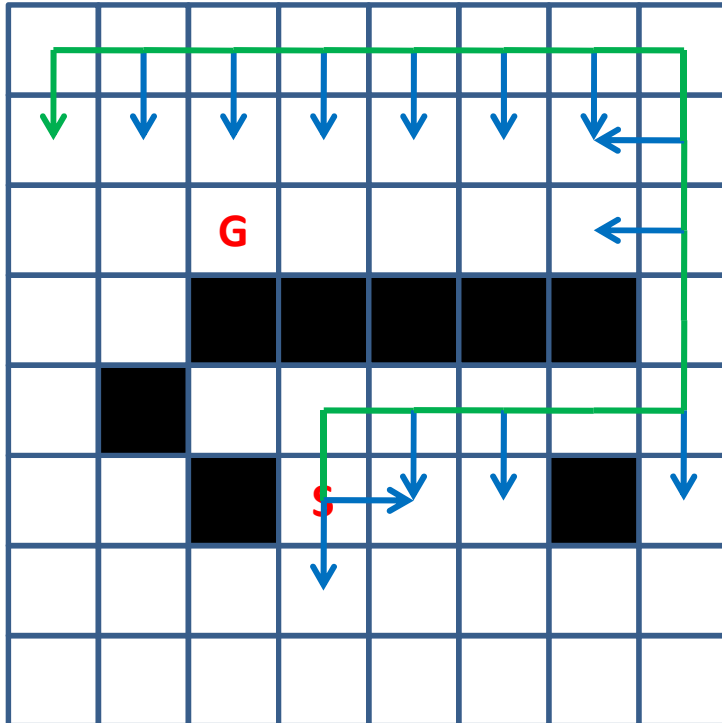
	16	15	14	13	12	11	10
							9
		G					8
							7
		2	1	3	4	5	6
			S				

Depth-first Search



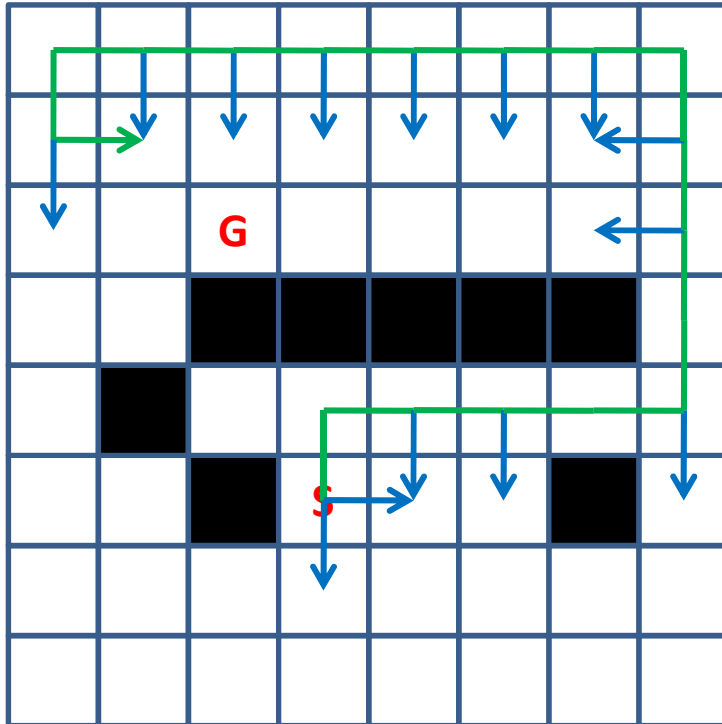
17	16	15	14	13	12	11	10
							9
		G					8
							7
							6
							5
							4
							3
							2
							1

Depth-first Search



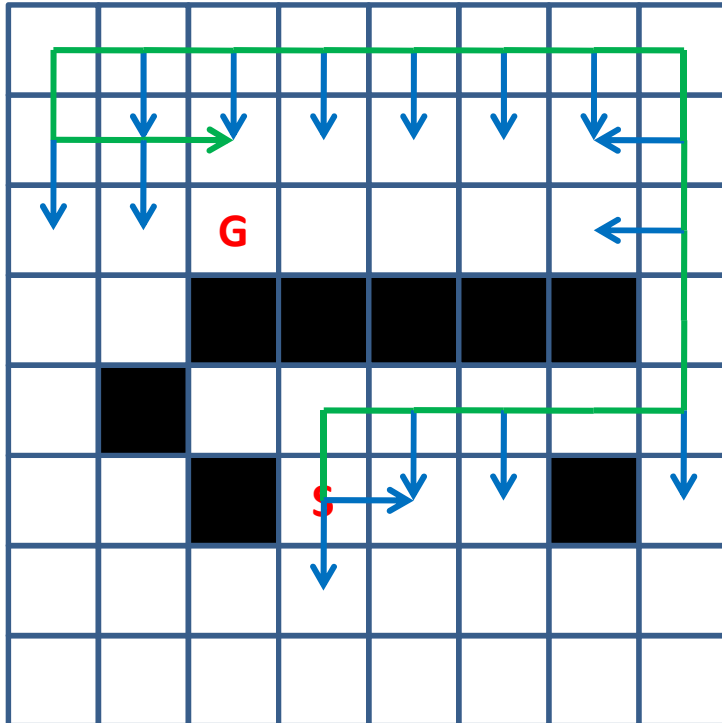
17	16	15	14	13	12	11	10
18							9
		G					8
							7
							6
							5
							4
							3
							2
							1
							S

Depth-first Search



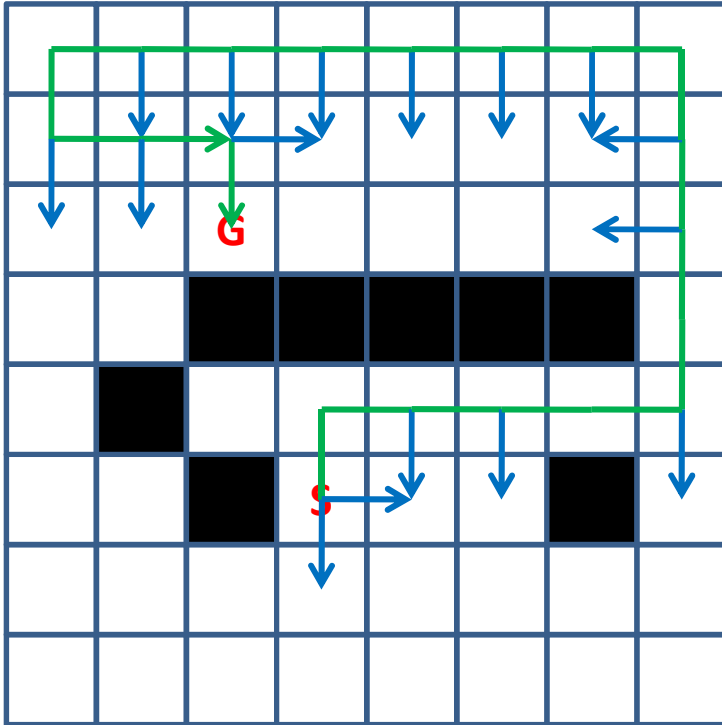
17	16	15	14	13	12	11	10
18	19						9
		G					8
							7
							6
							5
							4
							3
							2
							1
							0
							-1
							-2
							-3
							-4
							-5
							-6
							-7
							-8
							-9
							-10
							-11
							-12
							-13
							-14
							-15
							-16
							-17
							-18
							-19
							-20
							-21
							-22
							-23
							-24
							-25
							-26
							-27
							-28
							-29
							-30
							-31
							-32
							-33
							-34
							-35
							-36
							-37
							-38
							-39
							-40
							-41
							-42
							-43
							-44
							-45
							-46
							-47
							-48
							-49
							-50
							-51
							-52
							-53
							-54
							-55
							-56
							-57
							-58
							-59
							-60
							-61
							-62
							-63
							-64
							-65
							-66
							-67
							-68
							-69
							-70
							-71
							-72
							-73
							-74
							-75
							-76
							-77
							-78
							-79
							-80
							-81
							-82
							-83
							-84
							-85
							-86
							-87
							-88
							-89
							-90
							-91
							-92
							-93
							-94
							-95
							-96
							-97
							-98
							-99
							-100

Depth-first Search



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

Depth-first Search



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

Path Search

HILL-CLIMBING I SEARCH

Heuristic: Manhattan Distance

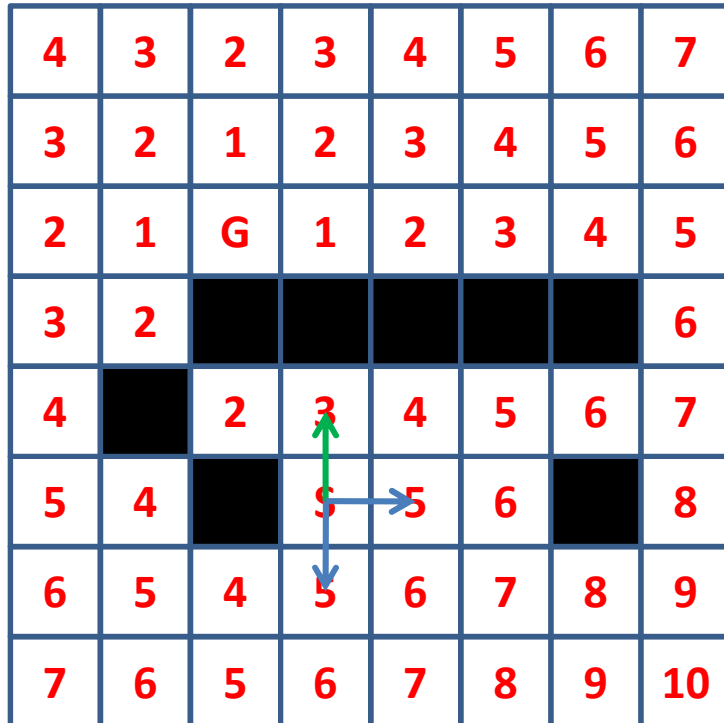
4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	0	1	2	3	4	5
3	2						6
4		2	3	4	5	6	7
5	4		4	5	6		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

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

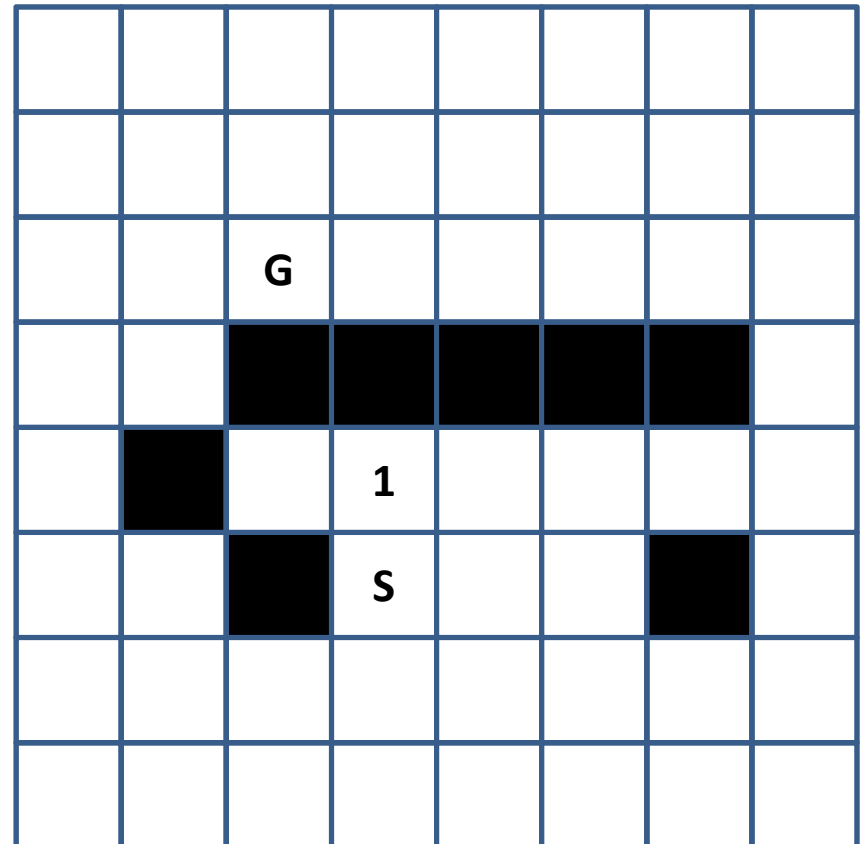
Hill-climbing I Search

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		S	5	6		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10



A diagram of an 8x8 grid representing a search space. The grid contains numerical values in red text. A path is highlighted with arrows: a green arrow points up from the cell (row 5, column 4) labeled 'S' to the cell (row 4, column 4) labeled '3', and a blue arrow points right from the cell (row 5, column 4) labeled 'S' to the cell (row 5, column 5) labeled '5'. The grid also contains several blacked-out cells: (row 3, column 2), (row 3, column 3), (row 3, column 4), (row 3, column 5), (row 3, column 6), (row 3, column 7), (row 4, column 1), (row 5, column 2), (row 5, column 7), (row 6, column 7).

		G					



A diagram of an 8x8 grid representing a search space. The grid contains numerical values in black text. A path is highlighted with arrows: a green arrow points up from the cell (row 5, column 4) labeled 'S' to the cell (row 4, column 4) labeled '1', and a blue arrow points right from the cell (row 5, column 4) labeled 'S' to the cell (row 5, column 5) labeled '5'. The grid also contains several blacked-out cells: (row 3, column 2), (row 3, column 3), (row 3, column 4), (row 3, column 5), (row 3, column 6), (row 3, column 7), (row 4, column 1), (row 5, column 2), (row 5, column 7), (row 6, column 7).

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		5	6			8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

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

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

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

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

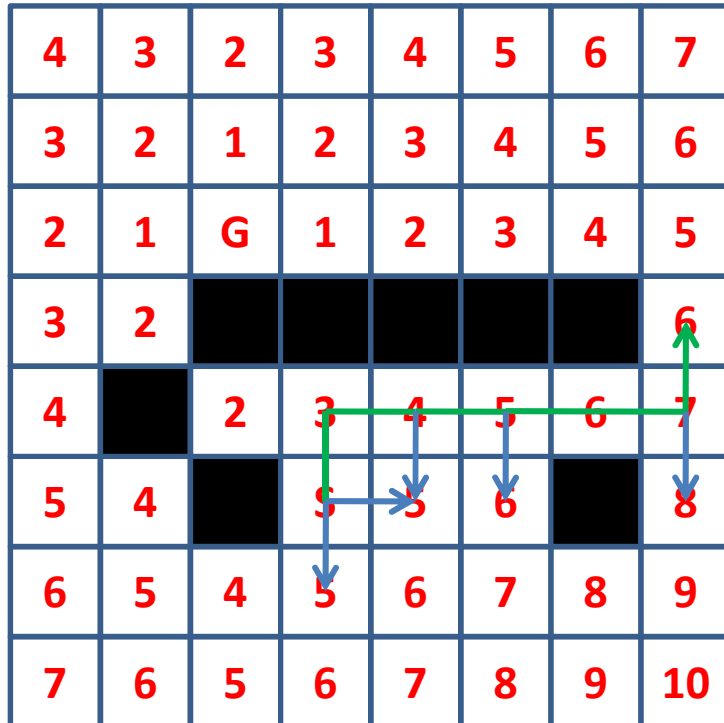
The diagram shows an 8x8 grid with numbers 1-10 and obstacles (black squares). The goal 'G' is at (3,3) and the start 'S' is at (5,4). A green arrow points from S to (5,6) and a blue arrow points from S to (4,5).

		G					
		2	1	3	4	5	6
			S				

The diagram shows an 8x8 grid with numbers 1-6 and obstacles (black squares). The goal 'G' is at (3,3) and the start 'S' is at (6,3). A green arrow points from S to (6,6) and a blue arrow points from S to (5,4).

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



		G					
							7

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

The diagram shows an 8x8 grid with numerical values. A path is highlighted from the goal node 'G' (row 3, column 3) to the node '5' (row 3, column 8) using a green line with arrows. From node '5', a path is highlighted to the node '10' (row 8, column 8) using a blue line with arrows. The path from 5 to 10 moves down to 6, then right to 7, then down to 8, then right to 9, and finally down to 10.

		G					8
							7
		2	1	3	4	5	6
			S				

The diagram shows an 8x8 grid with numerical values. A path is highlighted from the goal node 'G' (row 3, column 3) to the node 'S' (row 6, column 3) using a green line with arrows. From node 'S', a path is highlighted to the node '10' (row 8, column 8) using a blue line with arrows. The path from S to 10 moves down to 6, then right to 7, then down to 8, then right to 9, and finally down to 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

		G				9	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	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G			10	9	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	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G		11	10	9	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	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G	12	11	10	9	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	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

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

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	6			8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					
			1				
			S				

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	6			8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					
		2	1				
			S				

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	6			8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					

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	6	7	8	9
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					
		2	1	3	4		
			S				

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					
		2	1	3	4		
			S	5			

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	6	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					
		2	1	3	4		
			S	5/6			

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	6	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					
		2	1	3/7	4		
			S	5/6			

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	6	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

[illegible]

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	6	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

[illegible]

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	6	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

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

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	6	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					

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	6	7		8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

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

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					

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

[illegible]

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

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

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:

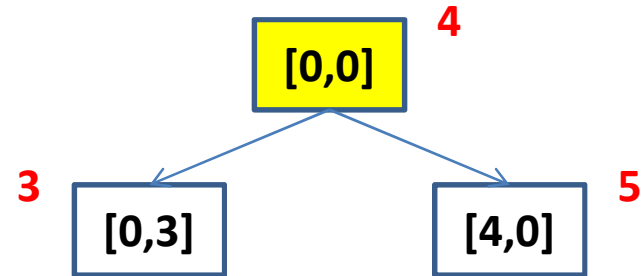
x	0	1	2	3	4
f(x)	2	1	0	1	3

- 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.

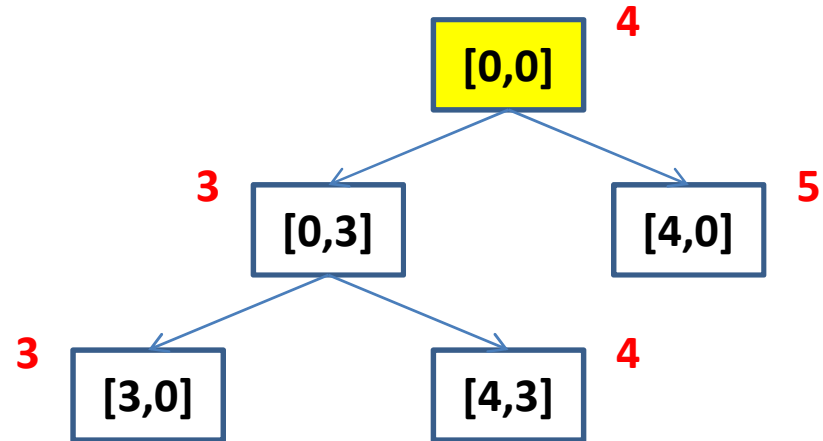
Water jugs

HILL-CLIMBING II SEARCH

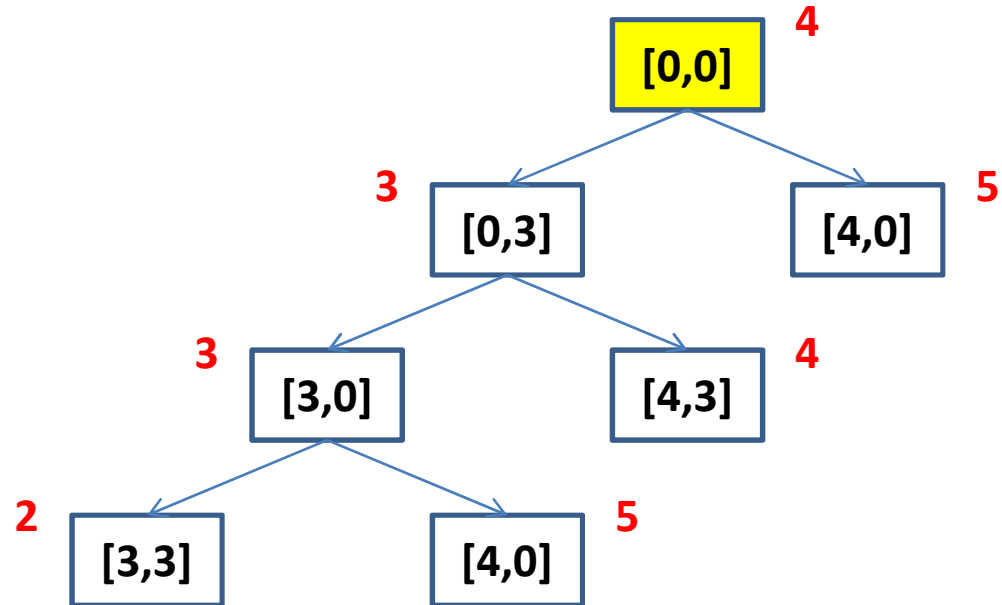
Hill-climbing II Search



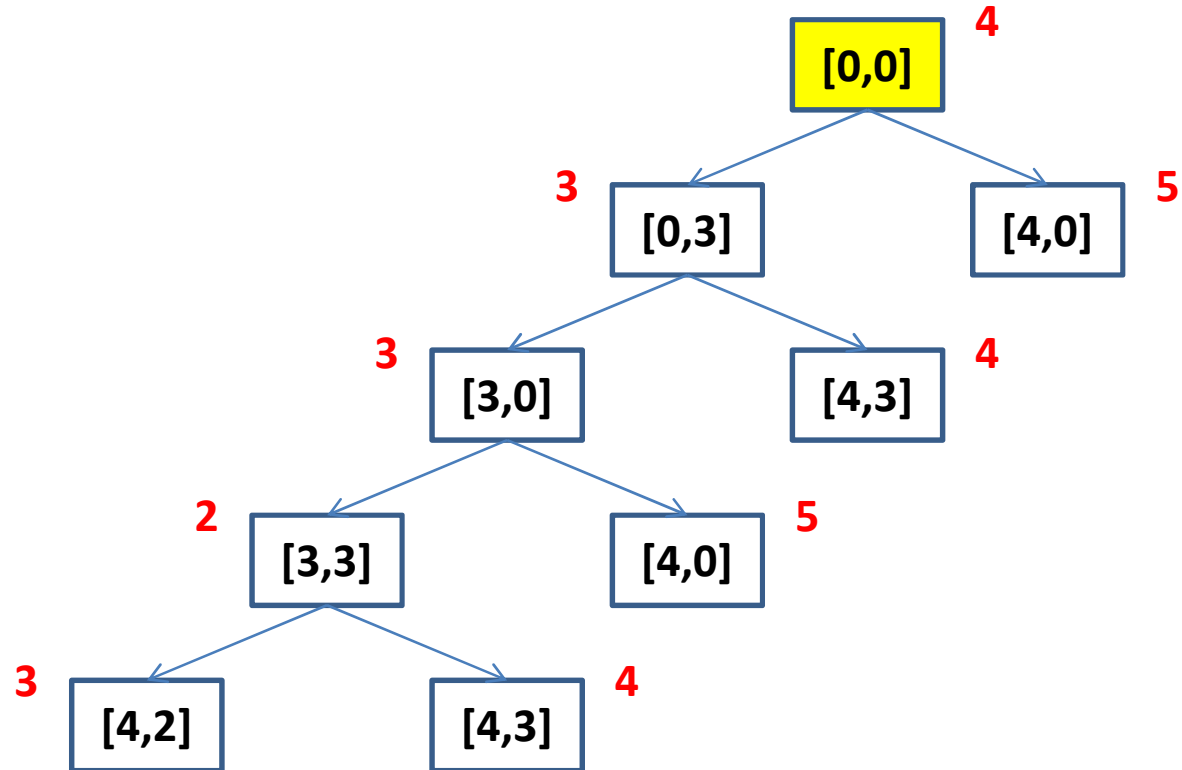
Hill-climbing II Search



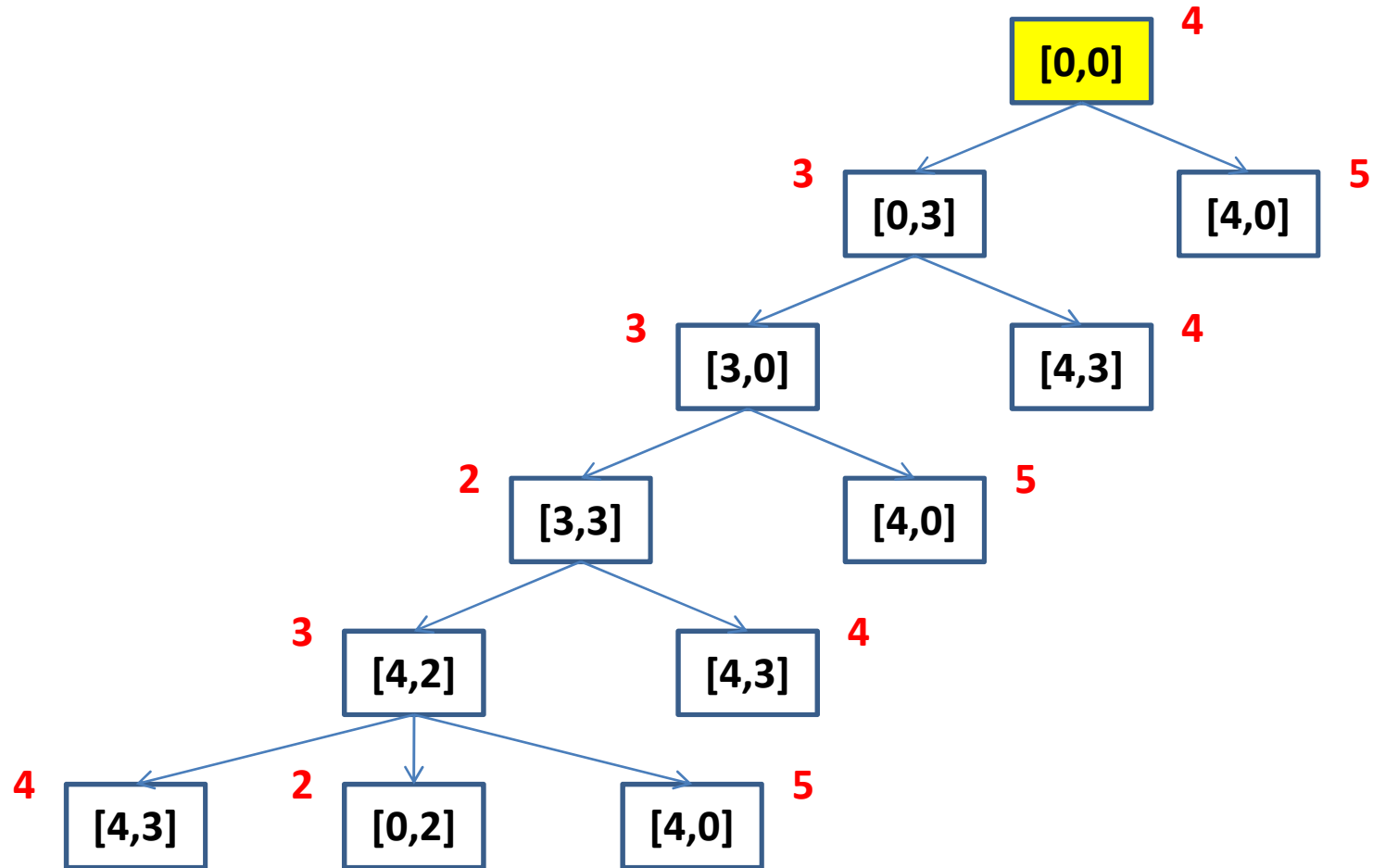
Hill-climbing II Search



Hill-climbing II Search



Hill-climbing II Search



Hill-climbing II Search

