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 = (<[Fa Fo Go Gr||]>)

Depth-first search (queues)

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

Depth-first search (queues)

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

Depth-first search (queues)

- S = (<[Fa Fo Go Gr|]>)
- Q$_1$ = (<$_{\text{[Fa Fo Go Gr|]}}$[Fo Gr|Fa Go]>)
- Q$_2$ = (**<**$_{\textbf{[Fa Fo Go Gr|][Fo Gr|Fa Go]}}$**[Fa Fo Gr|Go]>**)
  - Paths to Children:
    - R$_1$: <$_{\text{[Fa Fo Go Gr|][Fo Gr|Fa Go][Fa Fo Gr|Go]}}$[Fo Gr|Fa Go]>
    - R$_3$: <$_{\text{[Fa Fo Go Gr|][Fo Gr|Fa Go][Fa Fo Gr|Go]}}$[Gr|Fa Fo Go]>
    - R$_3$: <$_{\text{[Fa Fo Go Gr|][Fo Gr|Fa Go][Fa Fo Gr|Go]}}$[Fo|Fa Go Gr]>
- Q$_3$ = (**<**$_{\textbf{[Fa Fo Go Gr|][Fo Gr|Fa Go][Fa Fo Gr|Go]}}$**[Gr|Fa Fo Go]>,<**$_{\textbf{[Fa Fo Go Gr|][Fo Gr|Fa Go][Fa Fo Gr|Go]}}$**[Fo|Fa Go Gr]>**)

Depth-first search (queues)

- $S = (<\text{[Fa Fo Go Gr|]}>)$
- $Q_1 = (<_{\text{[Fa Fo Go Gr|]}}\text{[Fo Gr|Fa Go]}>)$
- $Q_2 = (<_{\text{[Fa Fo Go Gr|][Fo Gr|Fa Go]}}\text{[Fa Fo Gr|Go]}>)$
- $Q_3 = (\mathbf{<}_{\textbf{[Fa Fo Go Gr|][Fo Gr|Fa Go][Fa Fo Gr|Go]}}\textbf{[Gr|Fa Fo Go]}\mathbf{>},<_{\text{[Fa Fo Go Gr|][Fo Gr|Fa Go][Fa Fo Gr|Go]}}\text{[Fo|Fa Go Gr]}>)$
  - Paths to Children:
    - R_4: $<_{\text{[Fa Fo Go Gr|][Fo Gr|Fa Go]}\underline{\text{[Fa Fo Gr|Go]}}\text{[Gr|Fa Fo Go]}}\underline{\text{[Fa Fo Gr|Go]}}>$
    - R_4: $<_{\text{[Fa Fo Go Gr|][Fo Gr|Fa Go][Fa Fo Gr|Go][Gr|Fa Fo Go]}}\text{[Fa Go Gr|Fo]}>$
- $Q_4 = (\mathbf{<}_{\textbf{[Fa Fo Go Gr|][Fo Gr|Fa Go][Fa Fo Gr|Go][Gr|Fa Fo Go]}}\textbf{[Fa Go Gr|Fo]}\mathbf{>},<_{\text{[Fa Fo Go Gr|][Fo Gr|Fa Go][Fa Fo Gr|Go]}}\text{[Fo|Fa Go Gr]}>)$

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]>)

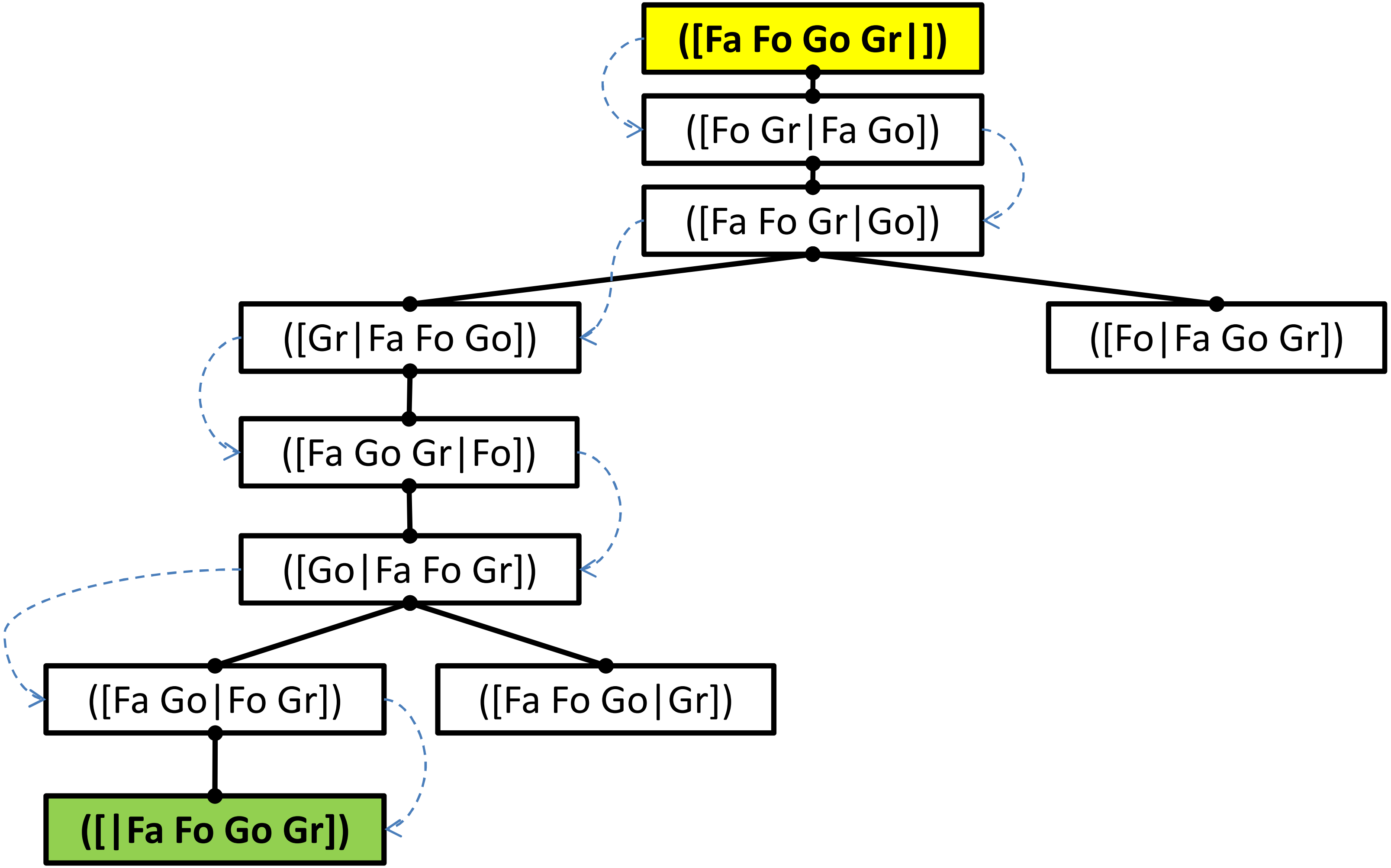
Depth-first search (queues)

- 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 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 Go|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)



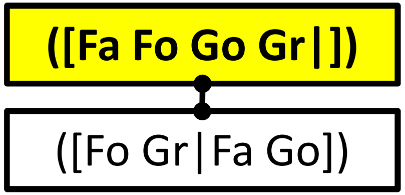
Farmer, Fox, Goose and Grain

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

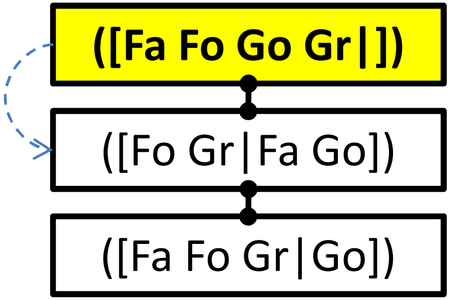
Breadth-first search (search tree)



([Fa Fo Go Gr|])

([Fo Gr|Fa Go])

Breadth-first search (search tree)

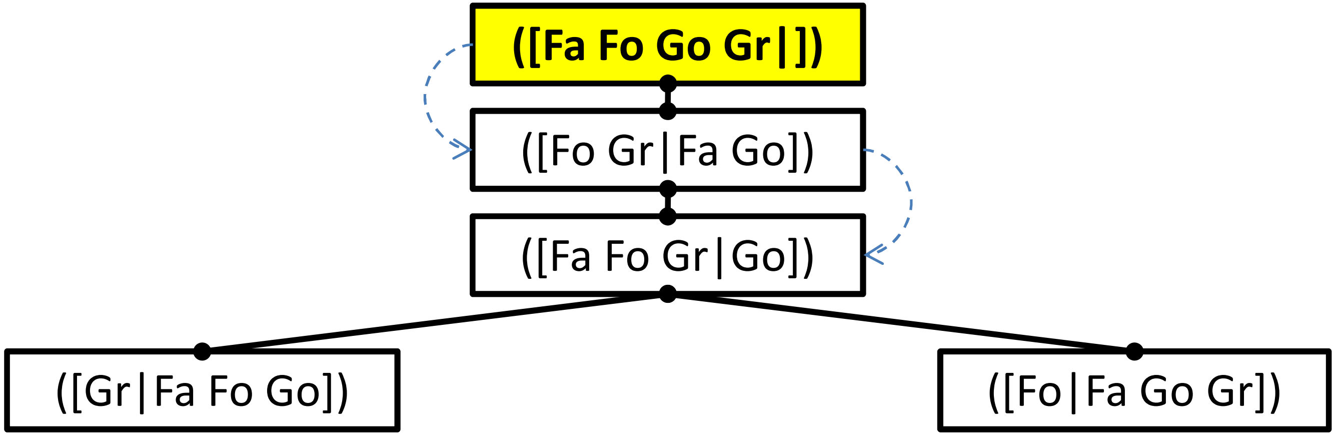


([Fa Fo Go Gr|])

([Fo Gr|Fa Go])

([Fa Fo Gr|Go])

Breadth-first search (search tree)



([Fa Fo Go Gr|])

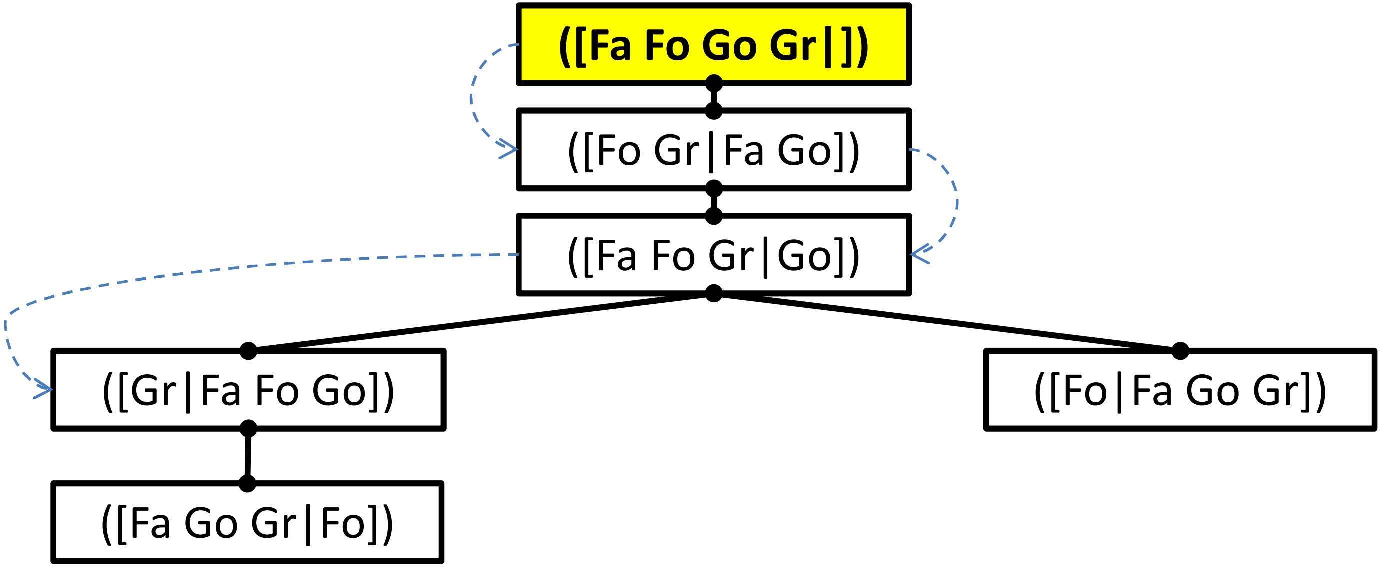
([Fo Gr|Fa Go])

([Fa Fo Gr|Go])

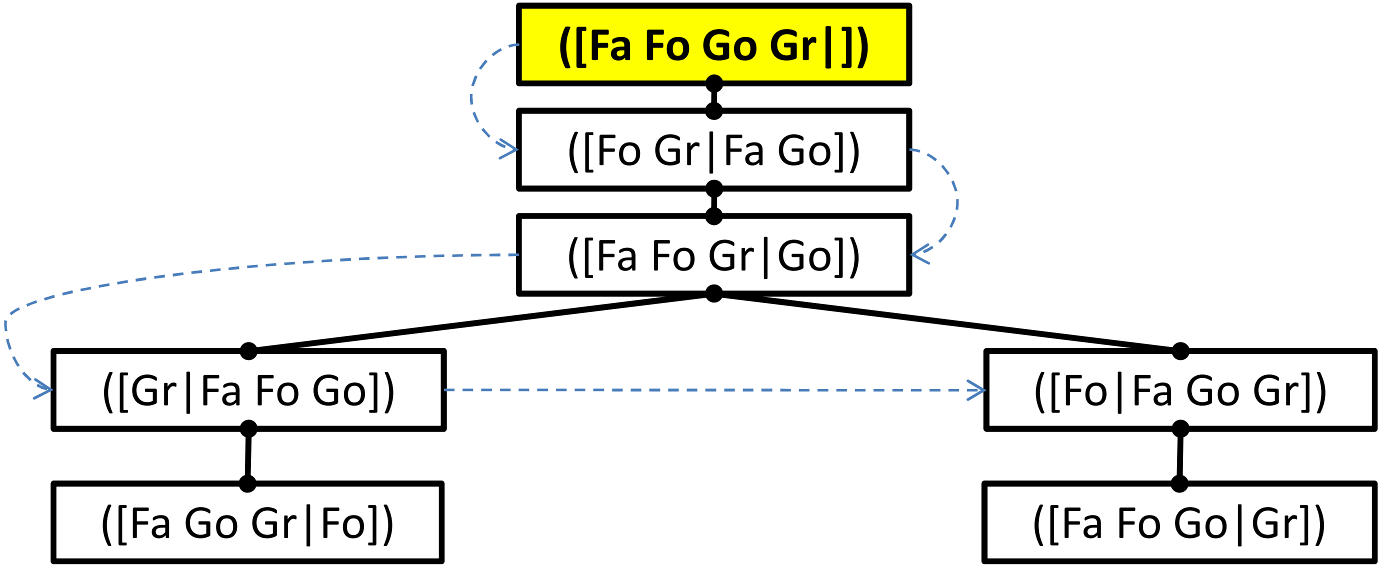
([Gr|Fa Fo Go])

([Fo|Fa Go Gr])

Breadth-first search (search tree)



Breadth-first search (search tree)



([Fa Fo Go Gr|])

([Fo Gr|Fa Go])

([Fa Fo Gr|Go])

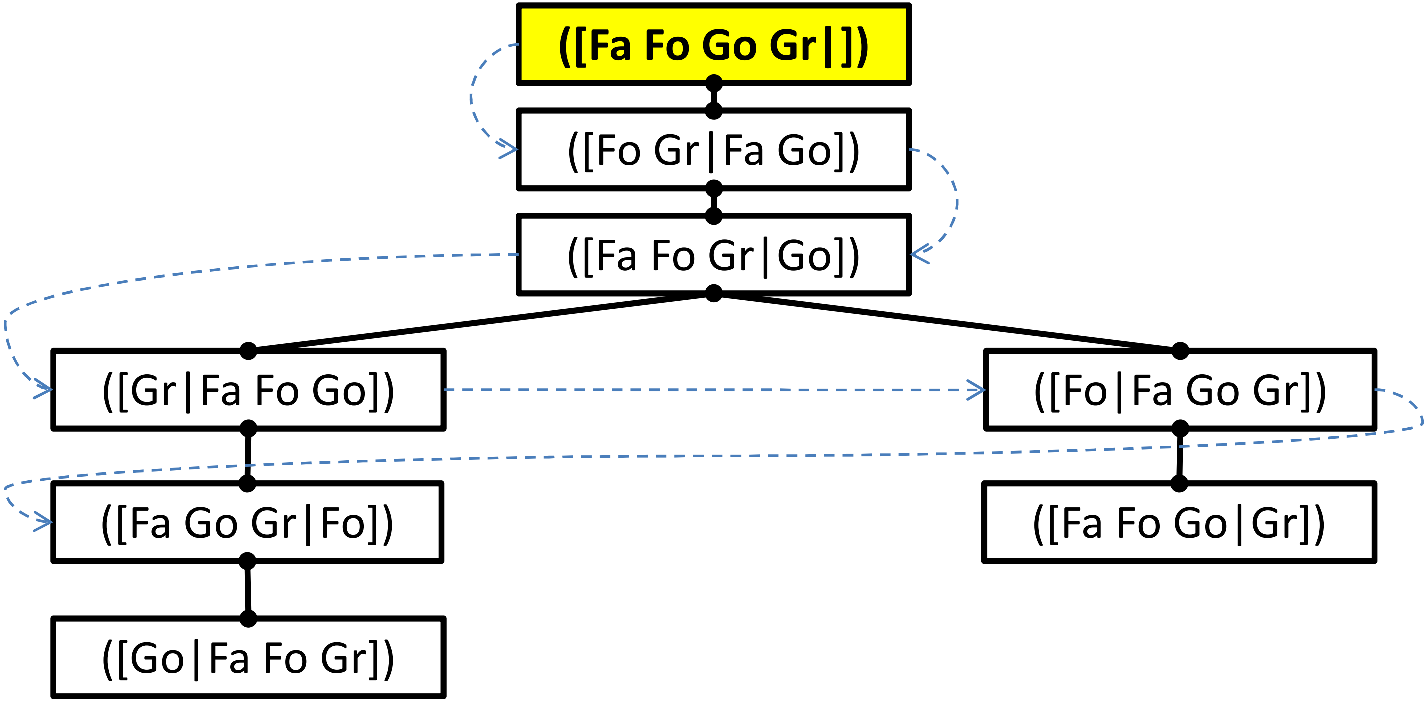
([Gr|Fa Fo Go])

([Fo|Fa Go Gr])

([Fa Go Gr|Fo])

([Fa Fo Go|Gr])

Breadth-first search (search tree)



([Fa Fo Go Gr|])

([Fo Gr|Fa Go])

([Fa Fo Gr|Go])

([Gr|Fa Fo Go])

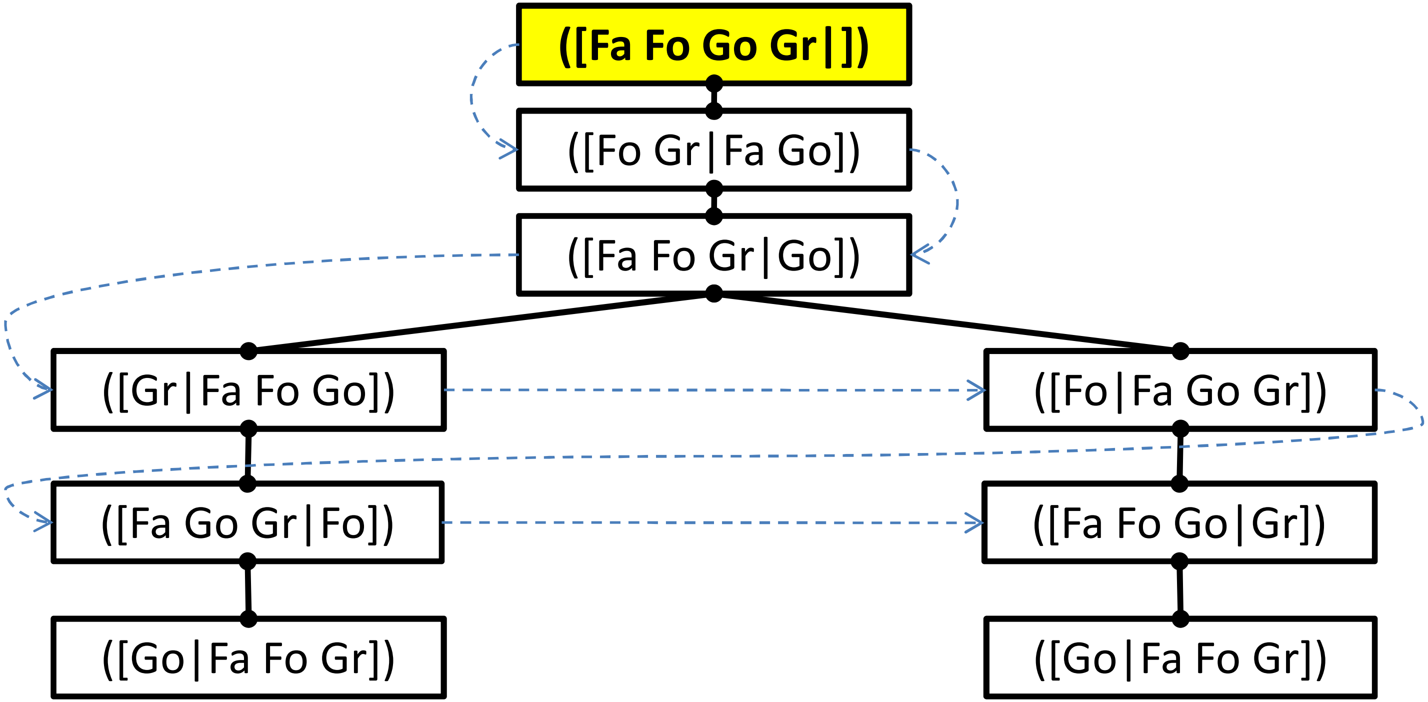
([Fo|Fa Go Gr])

([Fa Go Gr|Fo])

([Fa Fo Go|Gr])

([Go|Fa Fo Gr])

Breadth-first search (search tree)



([Fa Fo Go Gr|])

([Fo Gr|Fa Go])

([Fa Fo Gr|Go])

([Gr|Fa Fo Go])

([Fo|Fa Go Gr])

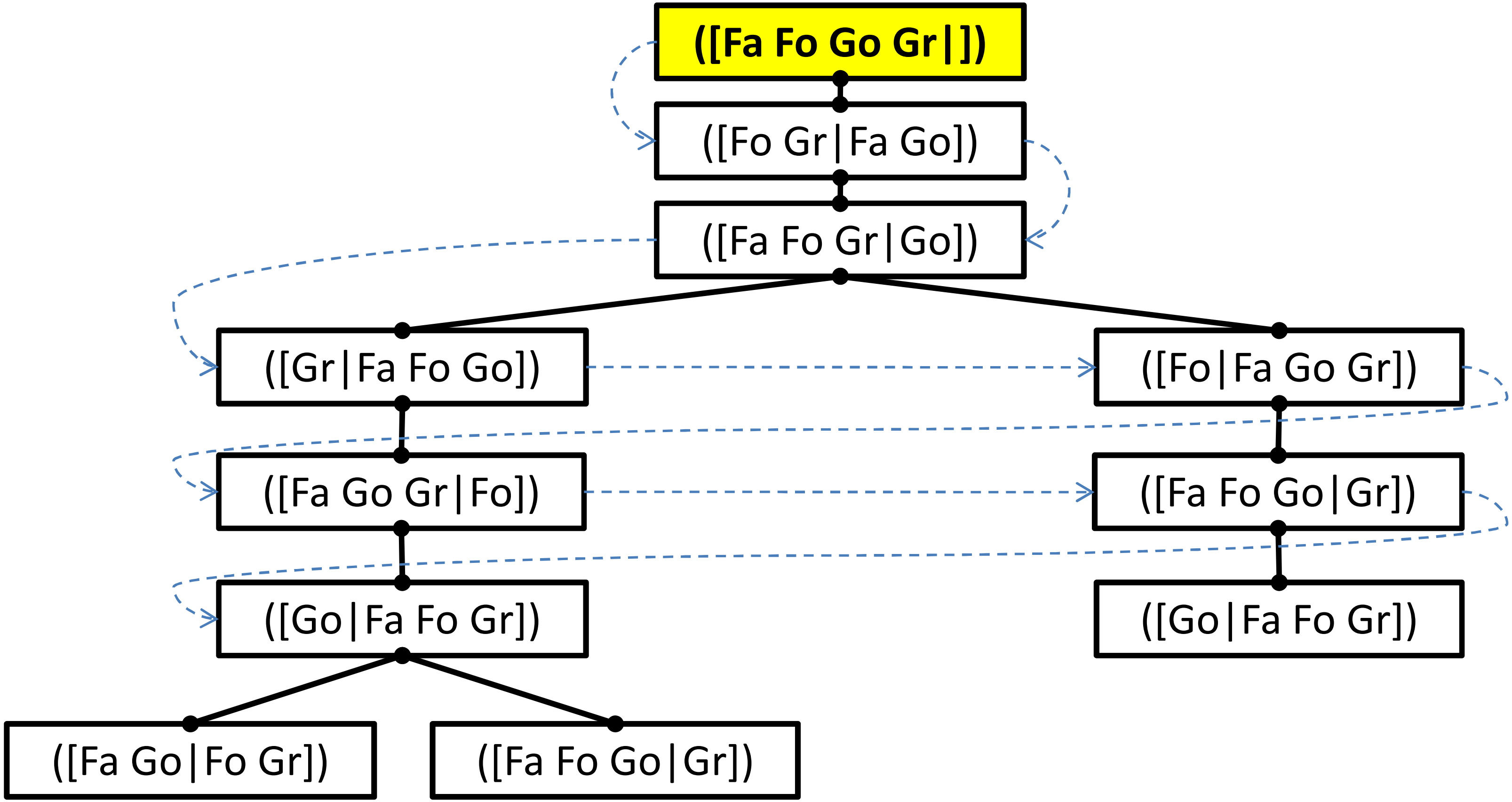
([Fa Go Gr|Fo])

([Fa Fo Go|Gr])

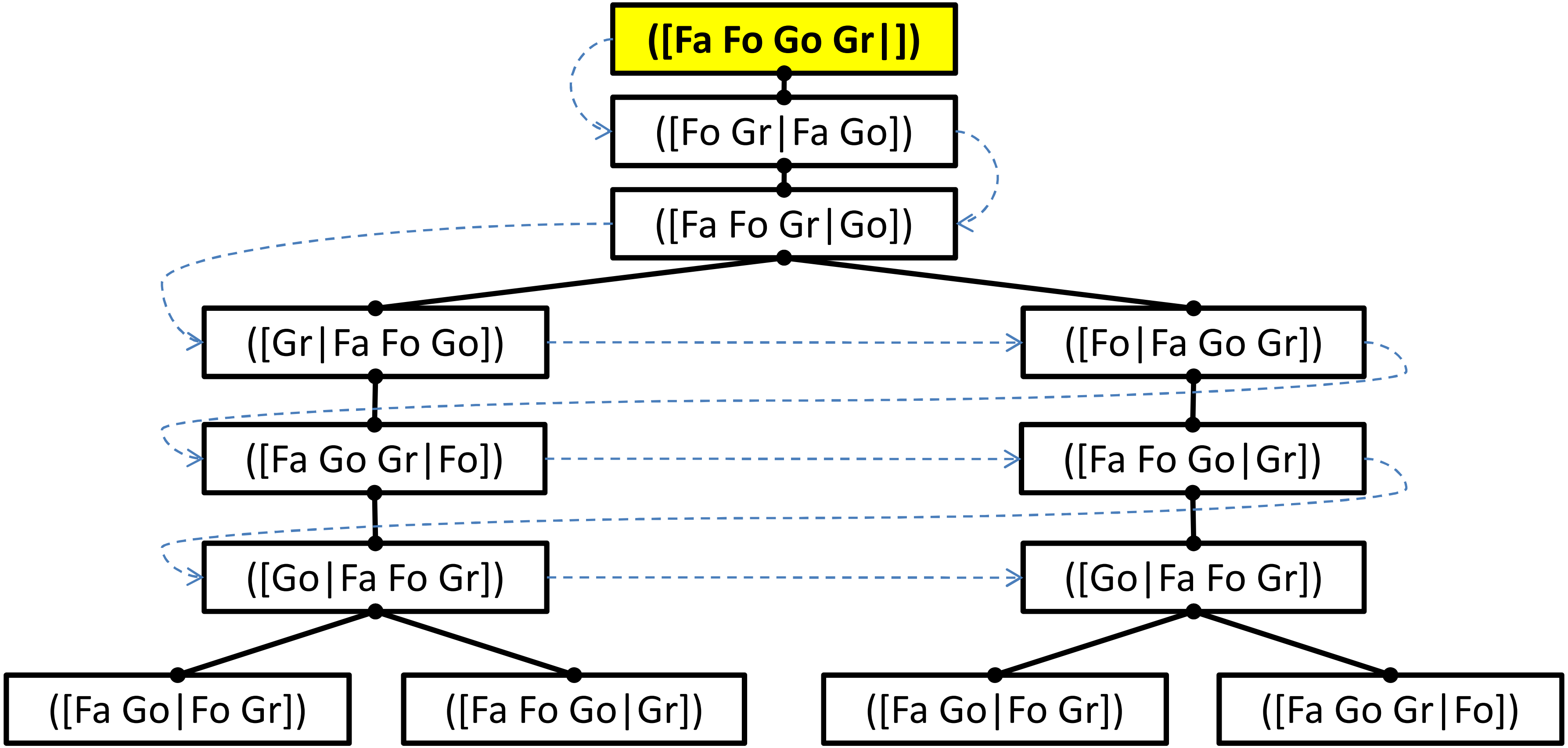
([Go|Fa Fo Gr])

([Go|Fa Fo Gr])

Breadth-first search (search tree)



Breadth-first search (search tree)



([Fa Fo Go Gr|])

([Fo Gr|Fa Go])

([Fa Fo Gr|Go])

([Gr|Fa Fo Go])

([Fo|Fa Go Gr])

([Fa Go Gr|Fo])

([Fa Fo Go|Gr])

([Go|Fa Fo Gr])

([Go|Fa Fo Gr])

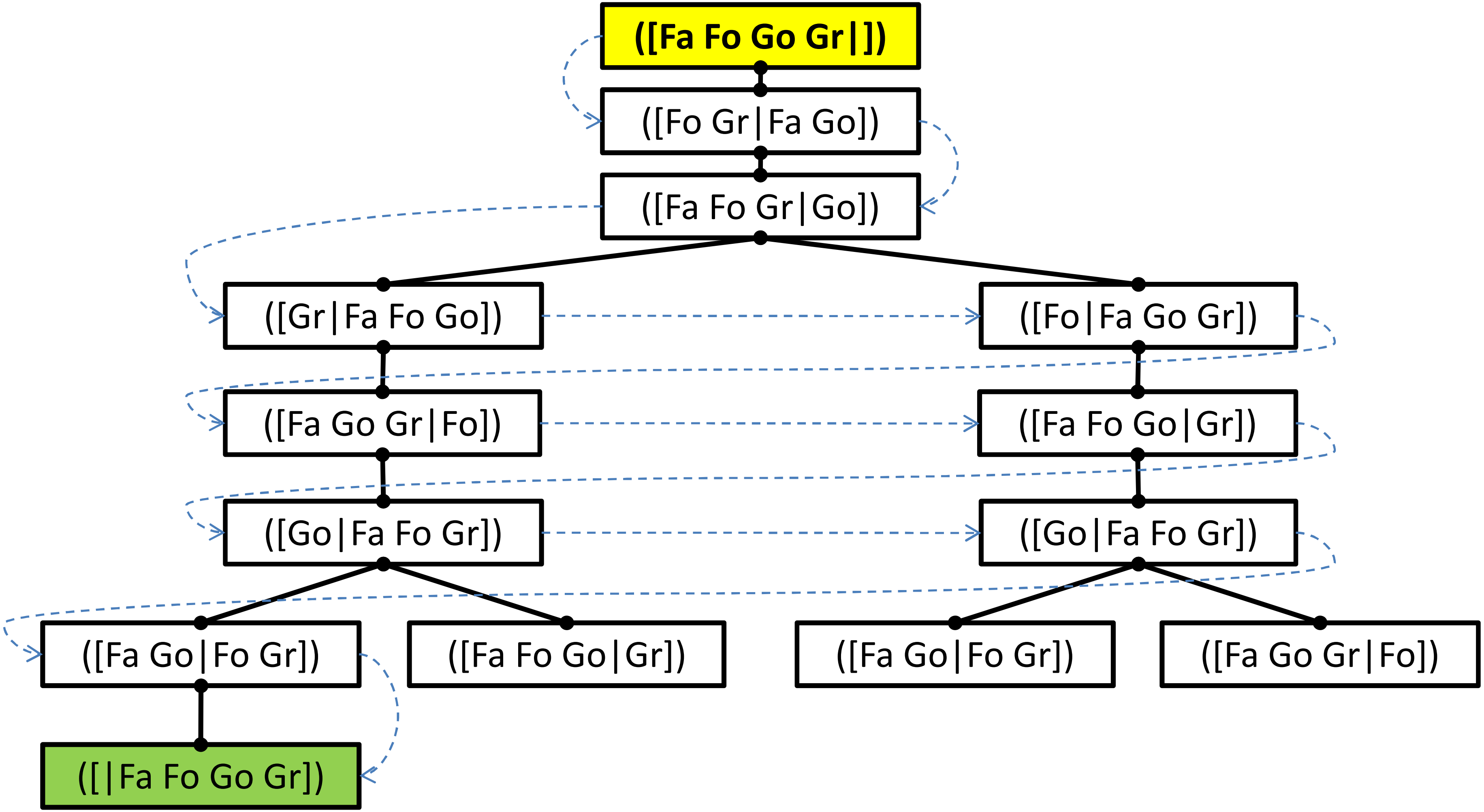
([Fa Go|Fo Gr])

([Fa Fo Go|Gr])

([Fa Go|Fo Gr])

([Fa Go Gr|Fo])

Breadth-first search (search tree)



([Fa Fo Go Gr|])

([Fo Gr|Fa Go])

([Fa Fo Gr|Go])

([Gr|Fa Fo Go])

([Fo|Fa Go Gr])

([Fa Go Gr|Fo])

([Fa Fo Go|Gr])

([Go|Fa Fo Gr])

([Go|Fa Fo Gr])

([Fa Go|Fo Gr])

([Fa Fo Go|Gr])

([Fa Go|Fo Gr])

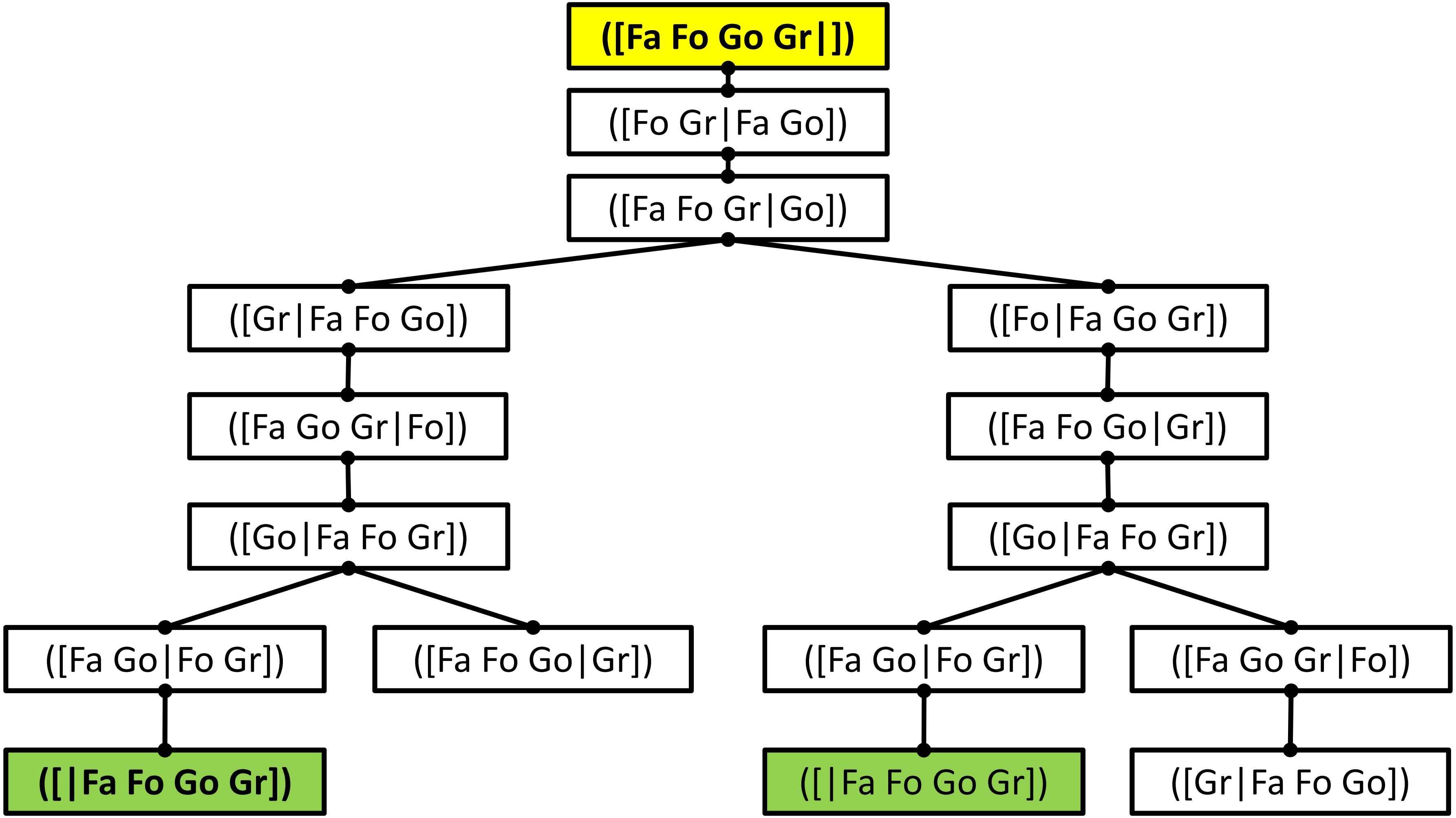
([Fa Go Gr|Fo])

([|Fa Fo Go Gr])

Farmer, Fox, Goose and Grain

ENTIRE SEARCH TREE

Entire search tree



Exercises: Artificial Intelligence

Bidiretional 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

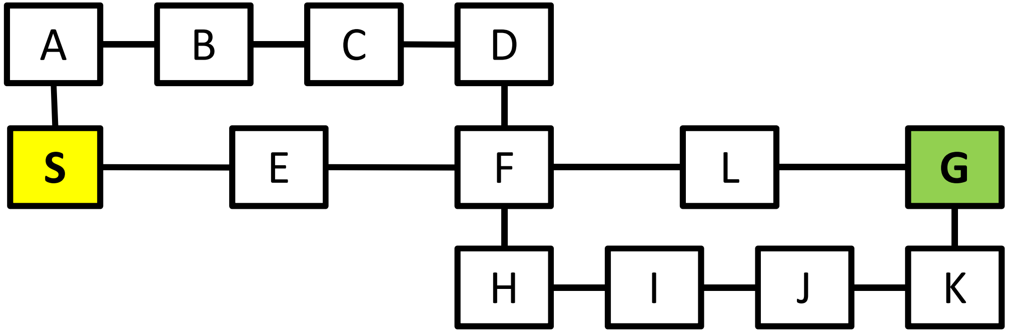
PROBLEM 1: BREADTH-FIRST?

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.

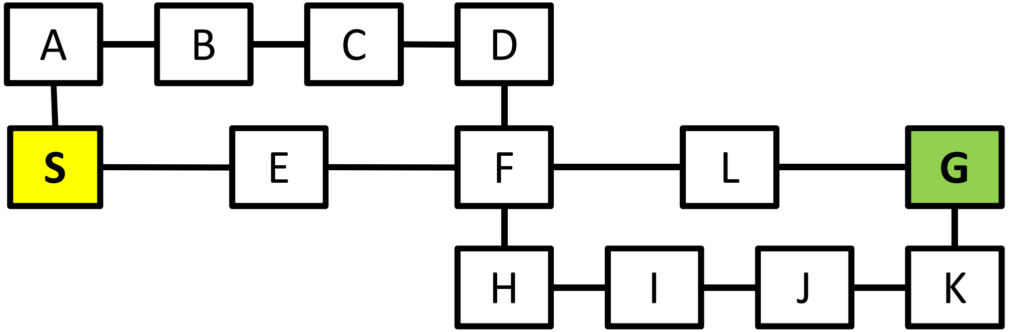
2 x Depth-first

- Forward:
  - (<S>) →(<SA>,<SE>) →(<SAB>,<SE>) →(<SABC>,<SE>) →(<SABCD>,<SE>) →**(<SABCDF>,<SE>)**
- Backward:
  - (<G>) →(<GK>,<GL>) →(<GKJ>,<GL>) →(<GKJI>,<GL>) →(<GKJIH>,<GL>) →**(<GKJIHF>,<GL>)**



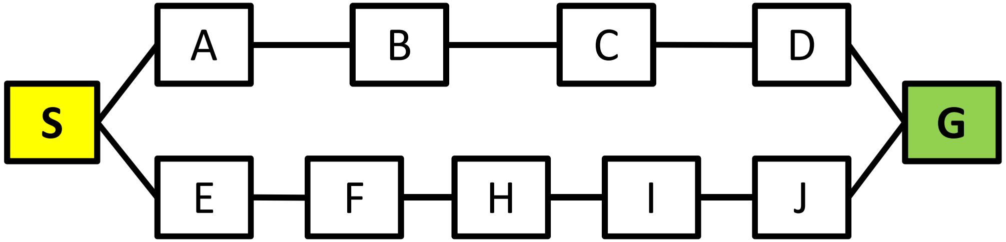
2 x Breadth-first

- Forward:
  - (<S>)→(<SA>,<SE>)→(<SE>,<SAB>)→**(<SAB>,<SEF>)**
- Backward:
  - (<G>)→(<GK>,<GL>)→(<GL>,<GKJ>)→**(<GKJ>,<GLF>)**



Breadth-first and Depth-first

- Forward (Breadth-first):
  - (<S>)→(<SA>,<SE>)→(<SE>,<SAB>)→(<SAB>,<SEF>) →**(<SEF>,<SABC>)**
- Backward (Depth-first):
  - (<G>)→(<GJ>,<GD>)→(<GJI>,<GD>)→(<GJIH>,<GD>) →**(<GJIHF>,<GD>)**

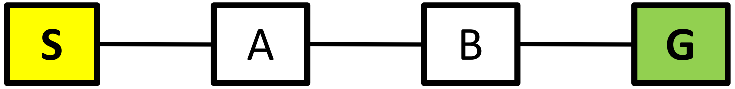


Bidirectional Search

PROBLEM 2: SHARED-STATE CHECK?

Replace shared-state check

- When only checking identical end-states, paths can cross each other unnoticed.
- Forward:
  - (<S>)→(<SA>)→(<SAB>)→(<SABG>)
- Backward:
  - (<G>)→(<GB>)→(<GBA>)→(<GBAS>)



Exercises: Artificial Intelligence

Beam Search

Beam Search

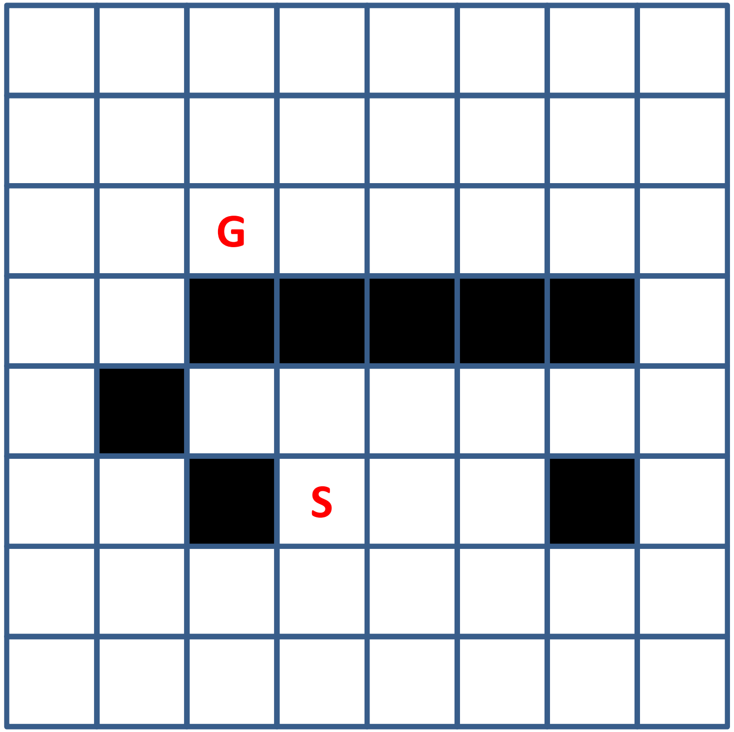
- ***Input:***
  - **QUEUE**: Path only containing root
  - **WIDTH**: Number
- ***Algorithm:***
  - **WHILE** (QUEUE not empty && goal not reached) **DO**
    - Remove ***all paths*** from QUEUE
    - Create paths to all children (of all paths)
    - Reject paths with loops
    - ***Sort new paths (according 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

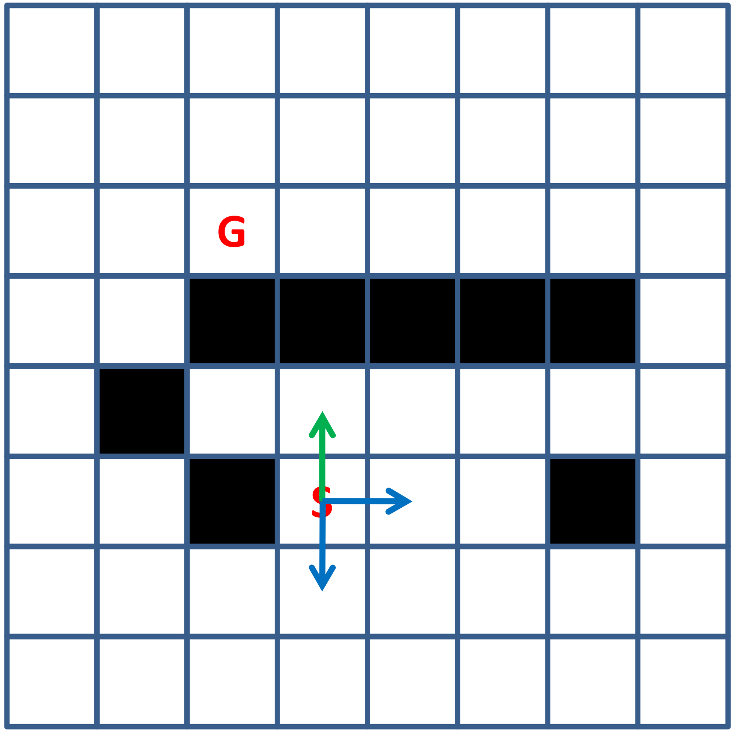
- Find a path from ‘S’ to ‘G’, without passing through black squares.
  - Legal steps (order):
    - up, left, right, down
- Perform:
  - Depth-first search
  - Hill-climbing I Search
    - With suitable heuristic
  - Greedy Search
    - With same heuristic

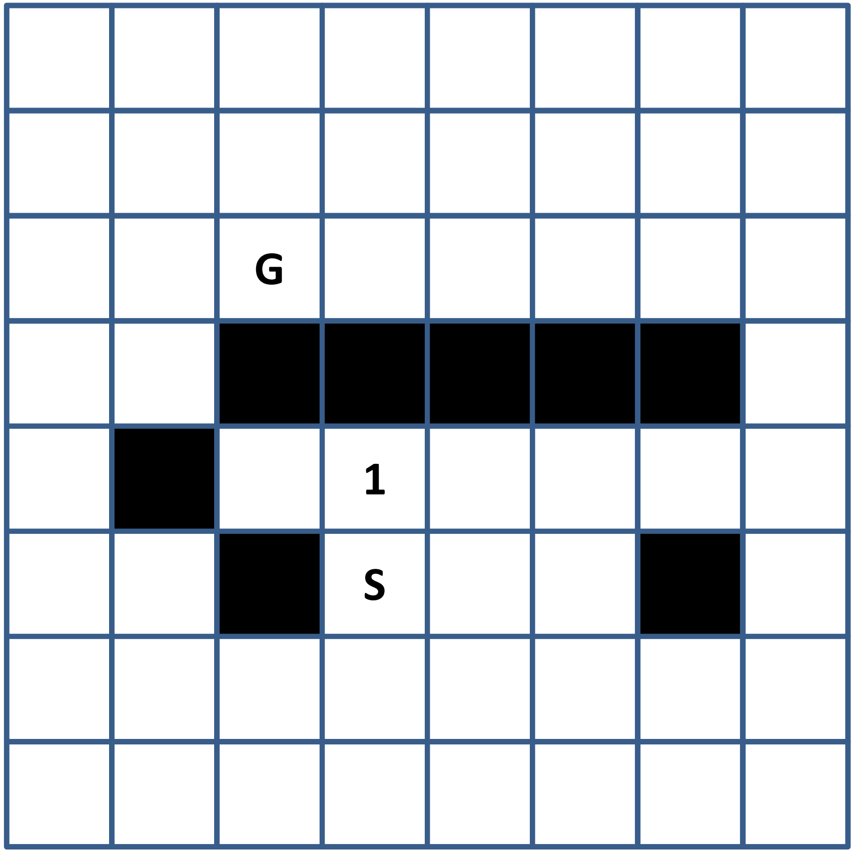


Path Search

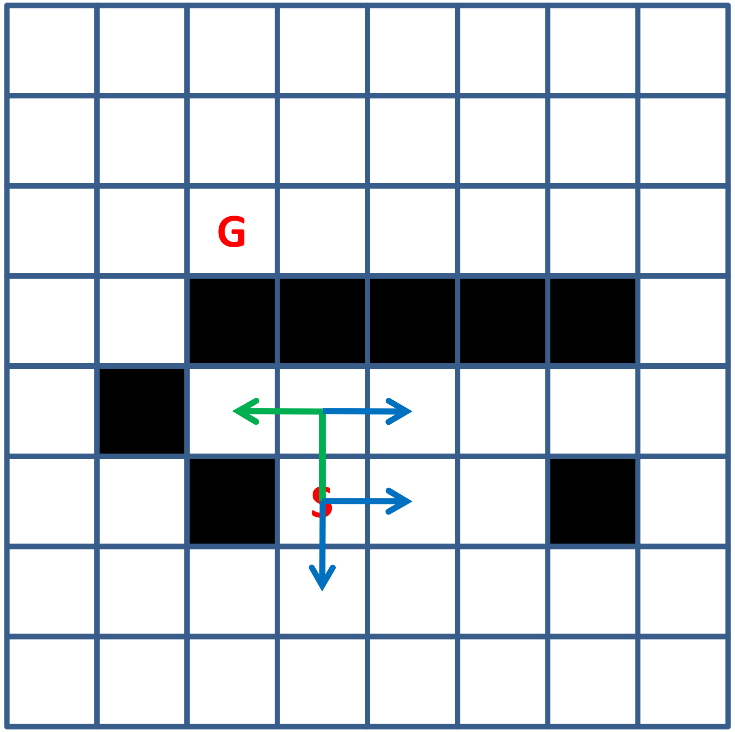
DEPTH-FIRST SEARCH

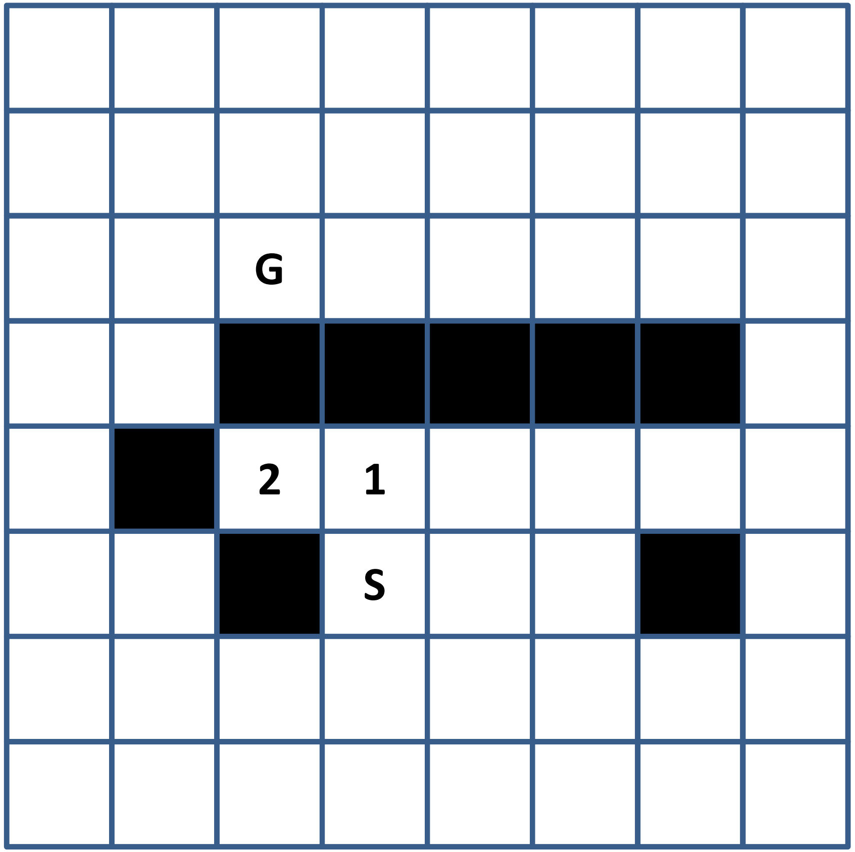
Depth-first Search



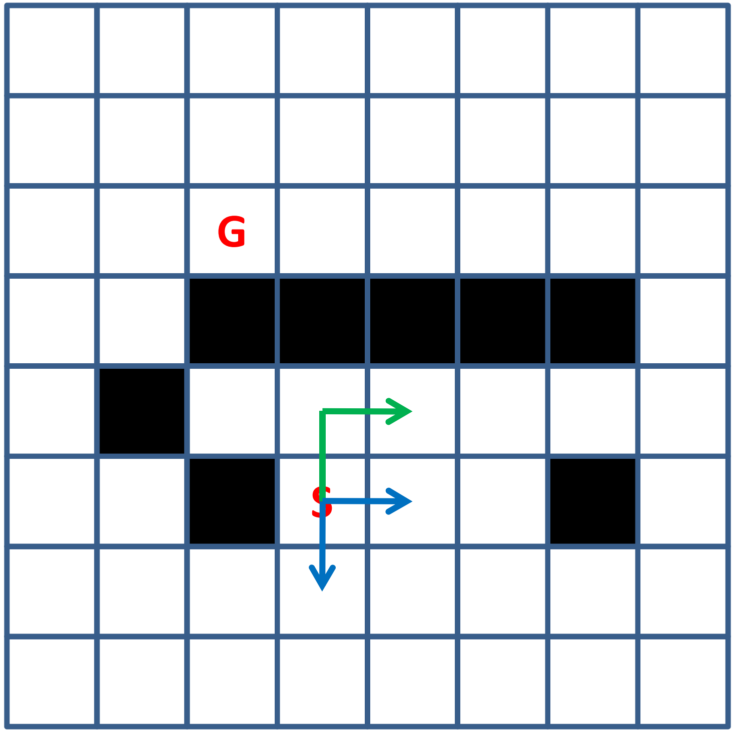


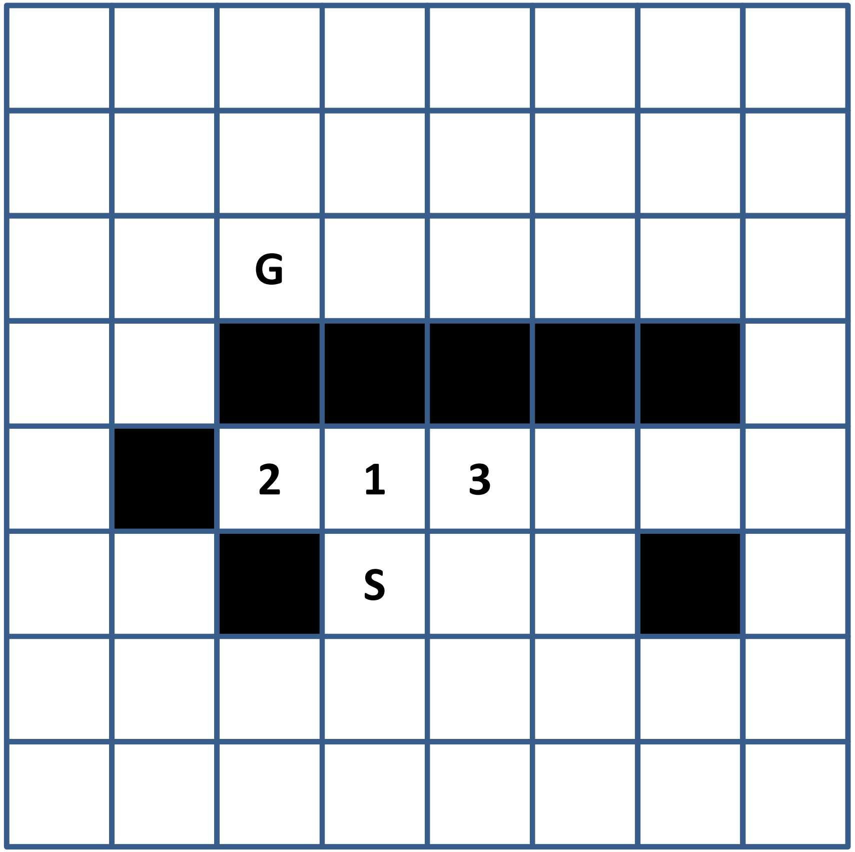
Depth-first Search



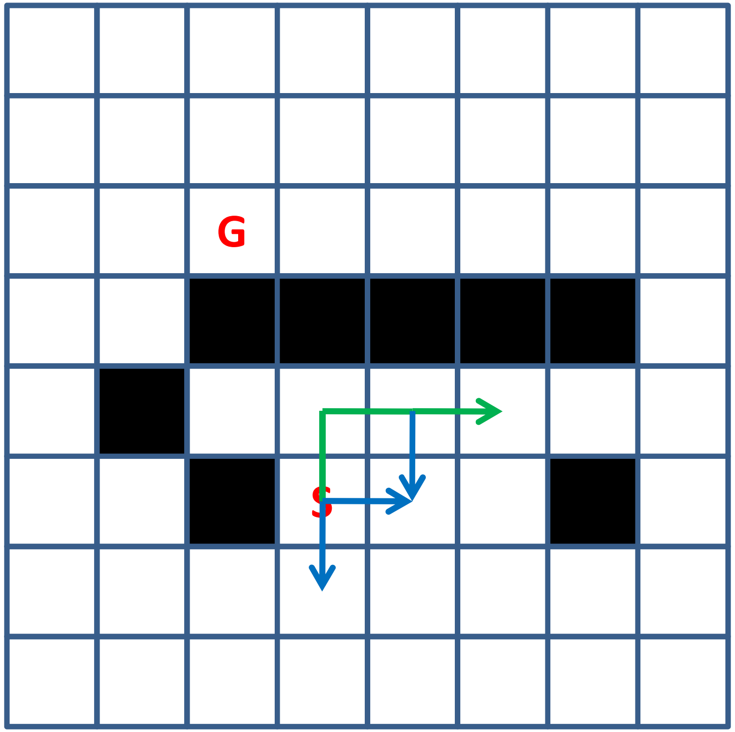


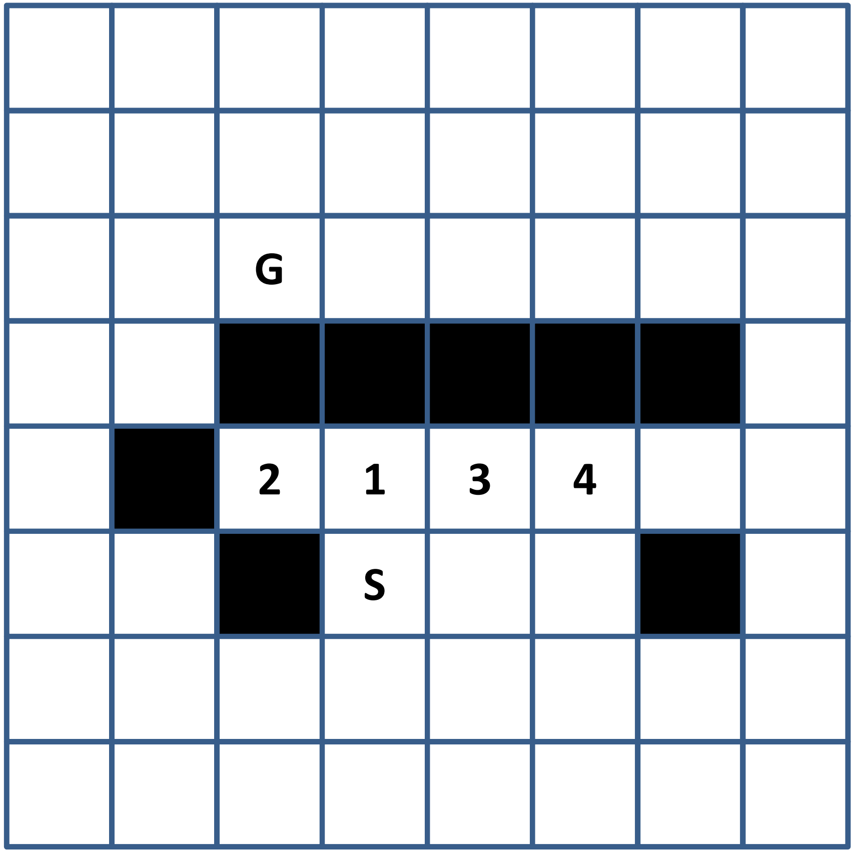
Depth-first Search



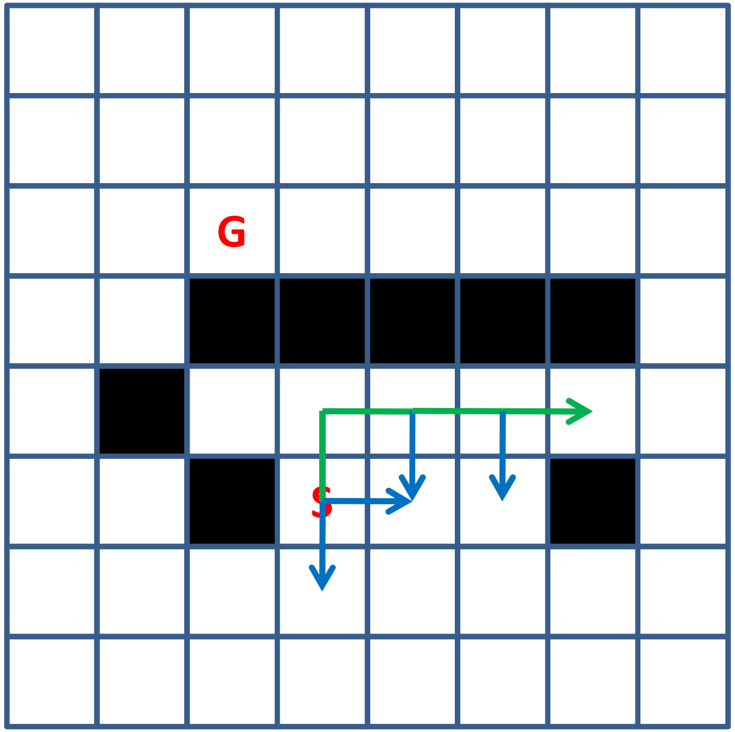


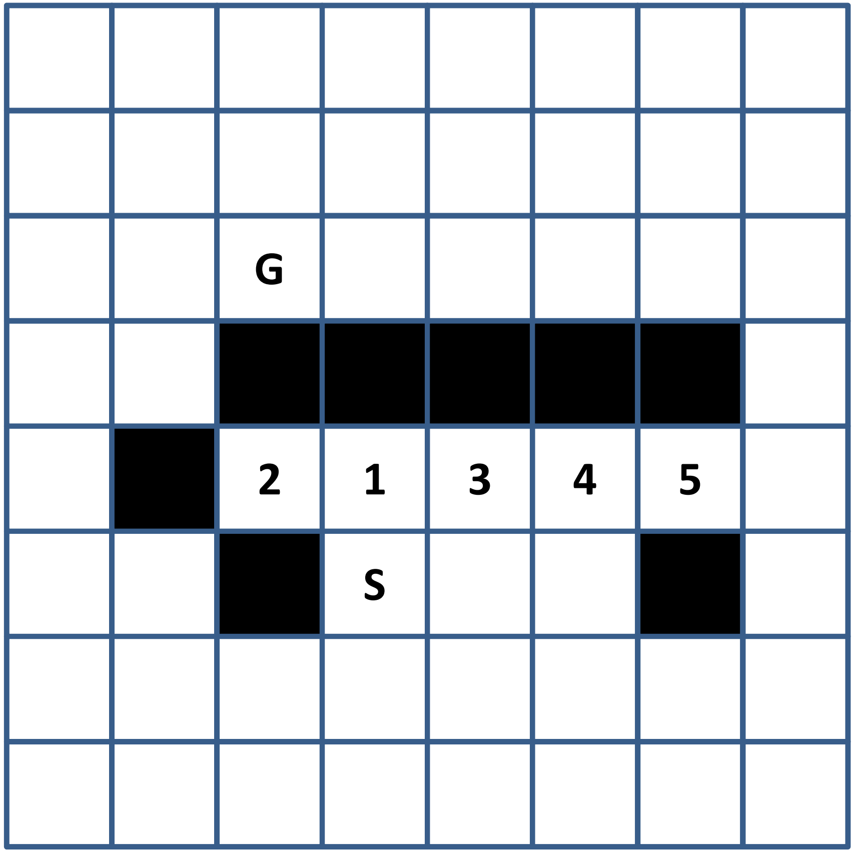
Depth-first Search



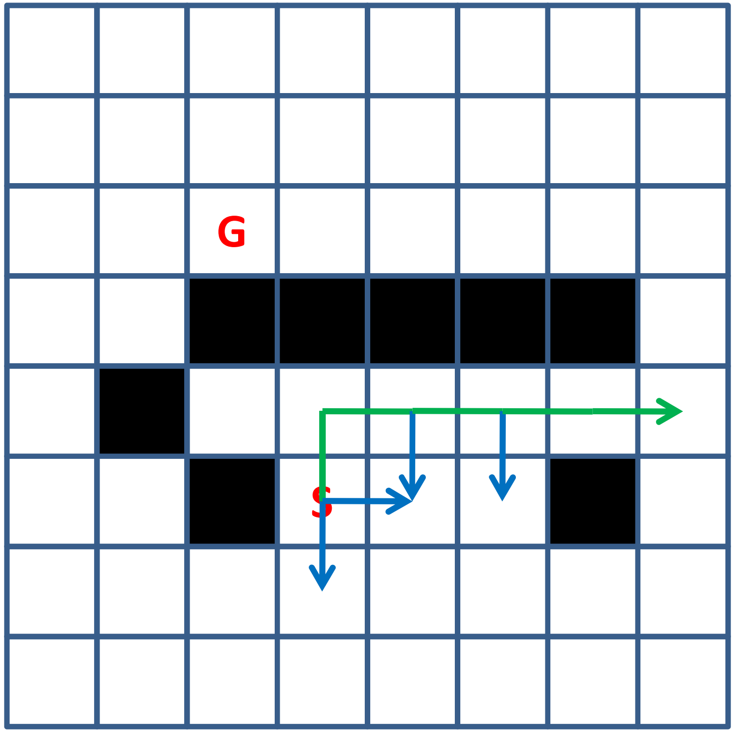


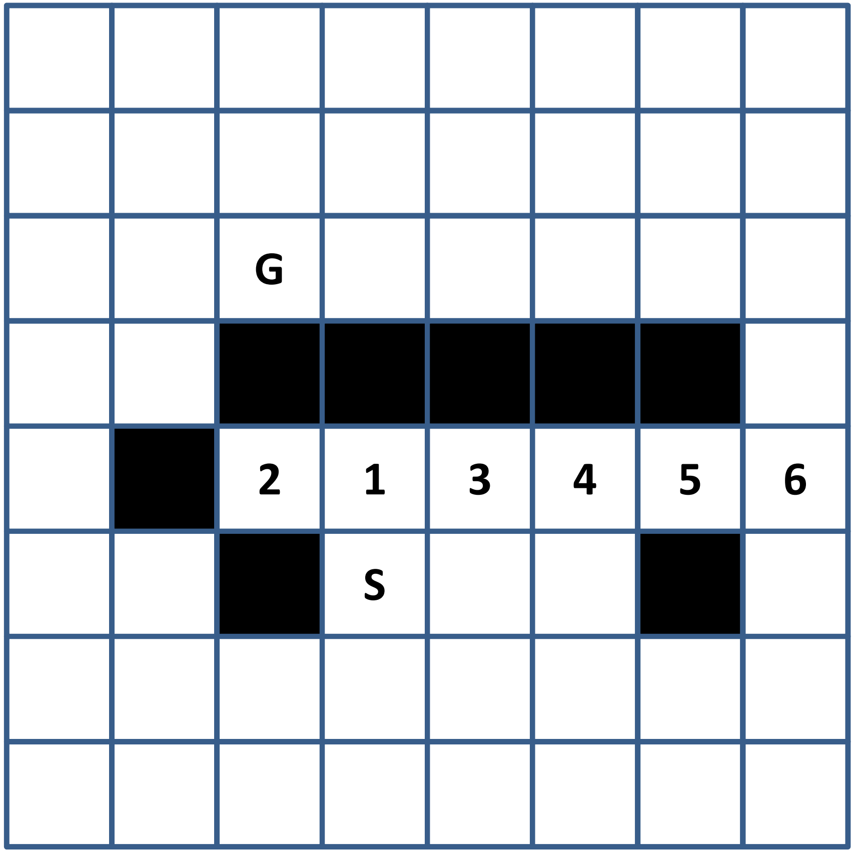
Depth-first Search



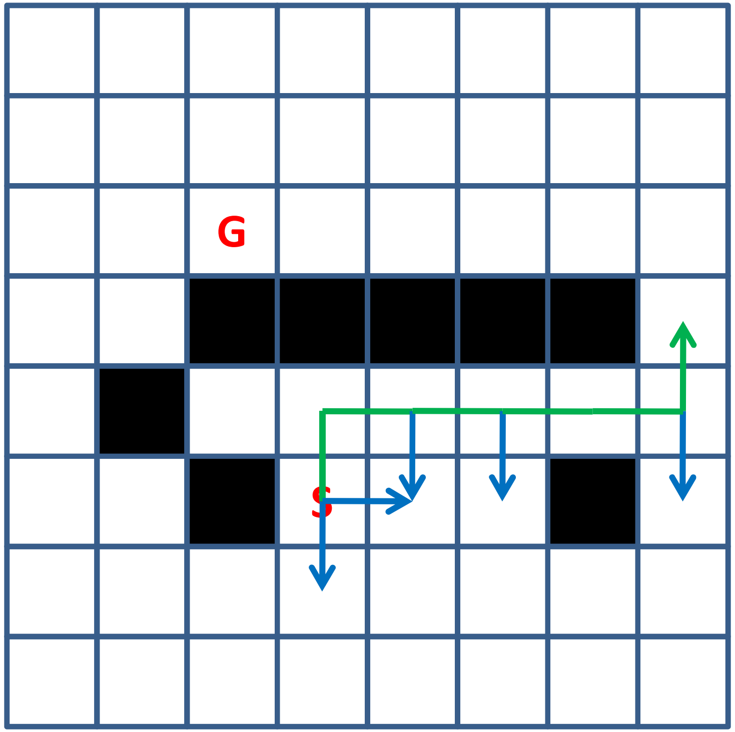


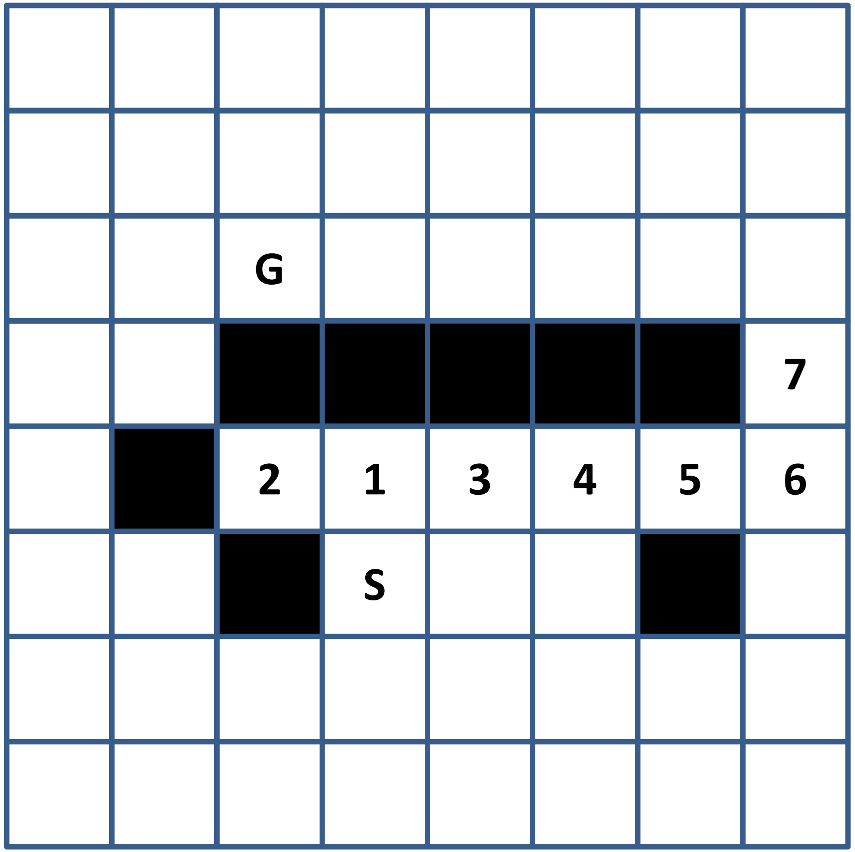
Depth-first Search



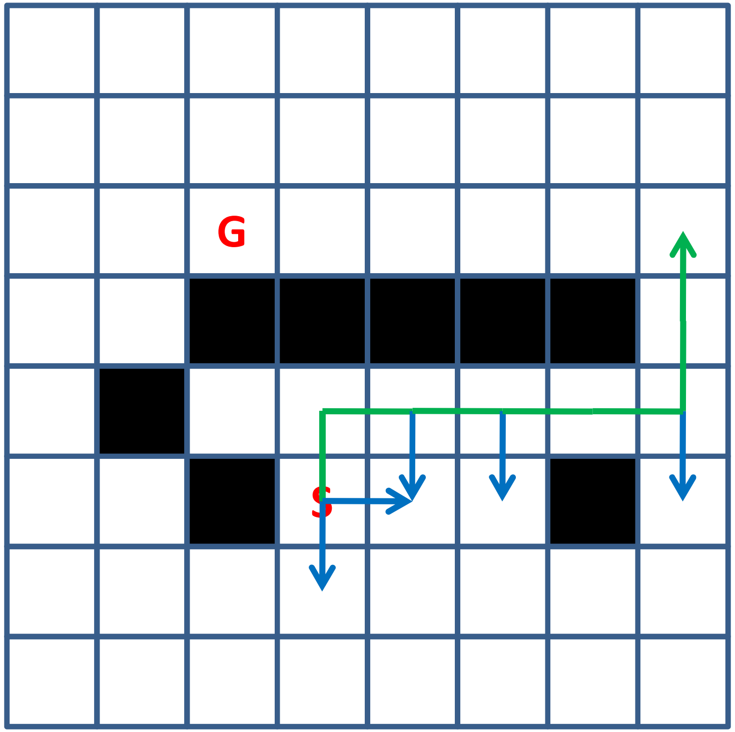


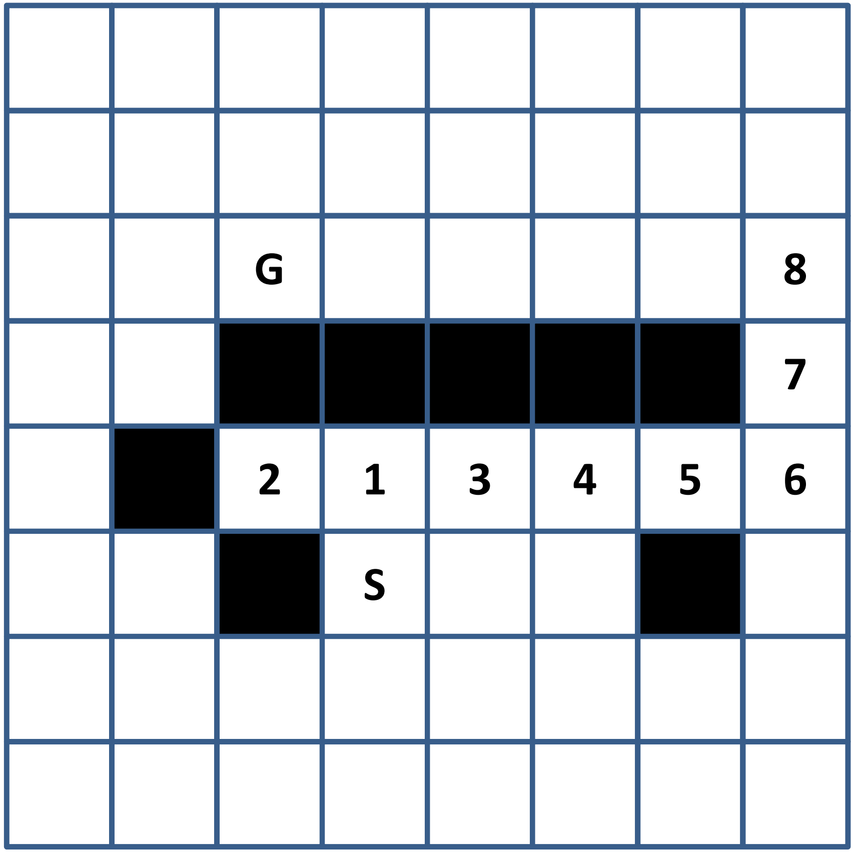
Depth-first Search



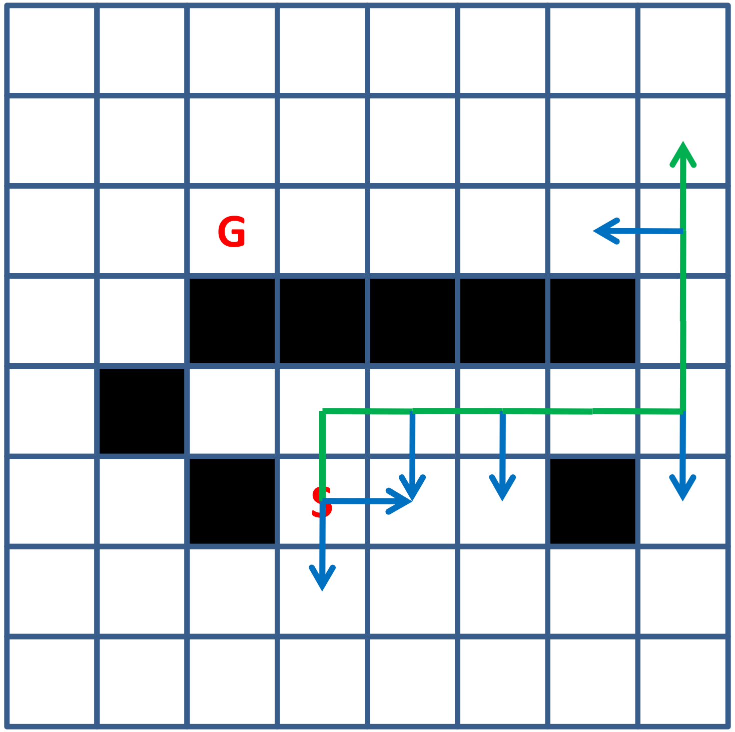


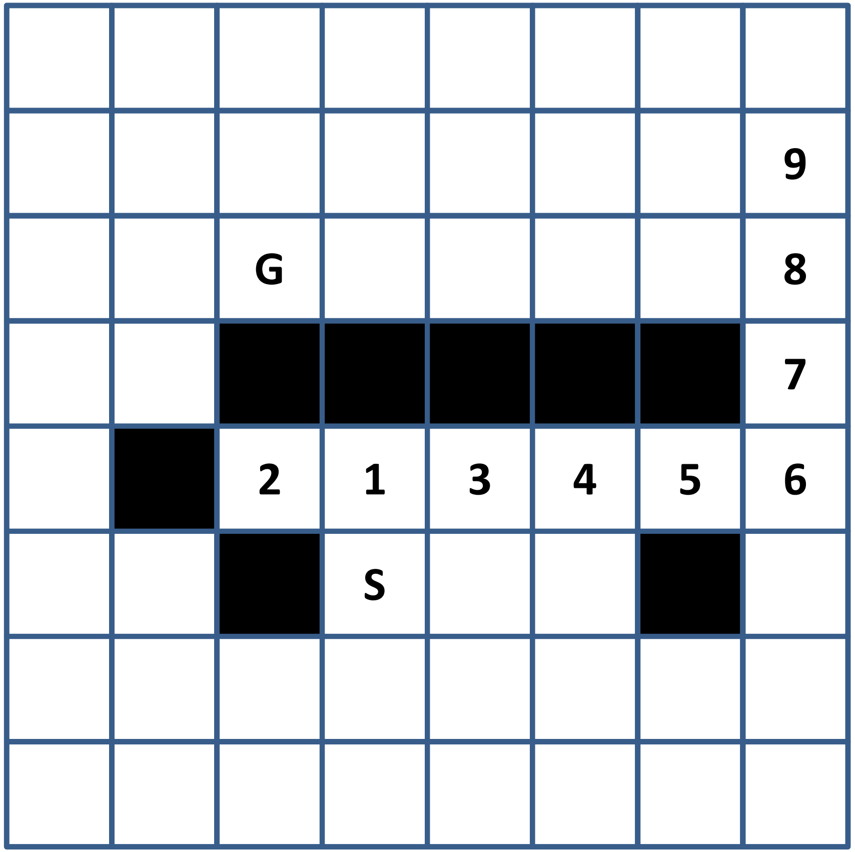
Depth-first Search



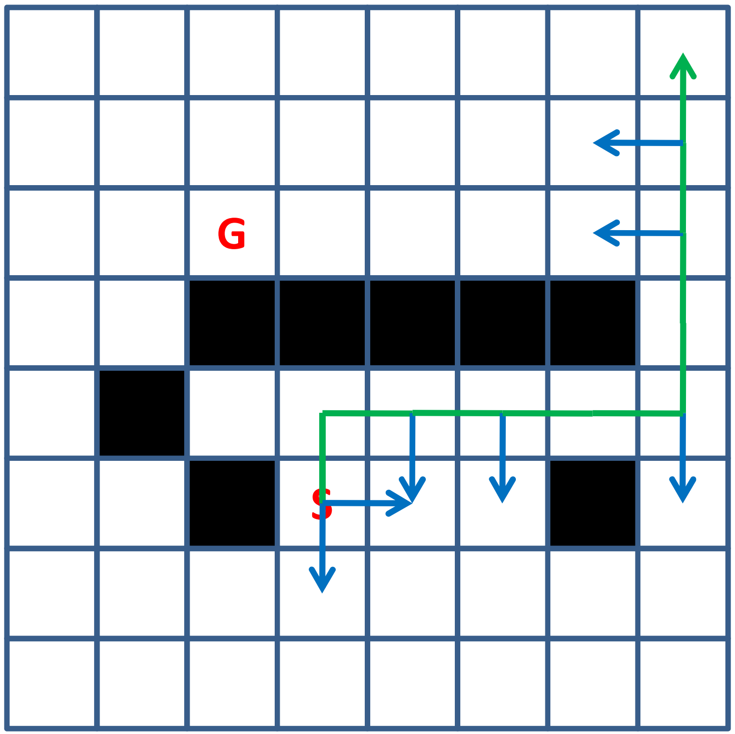


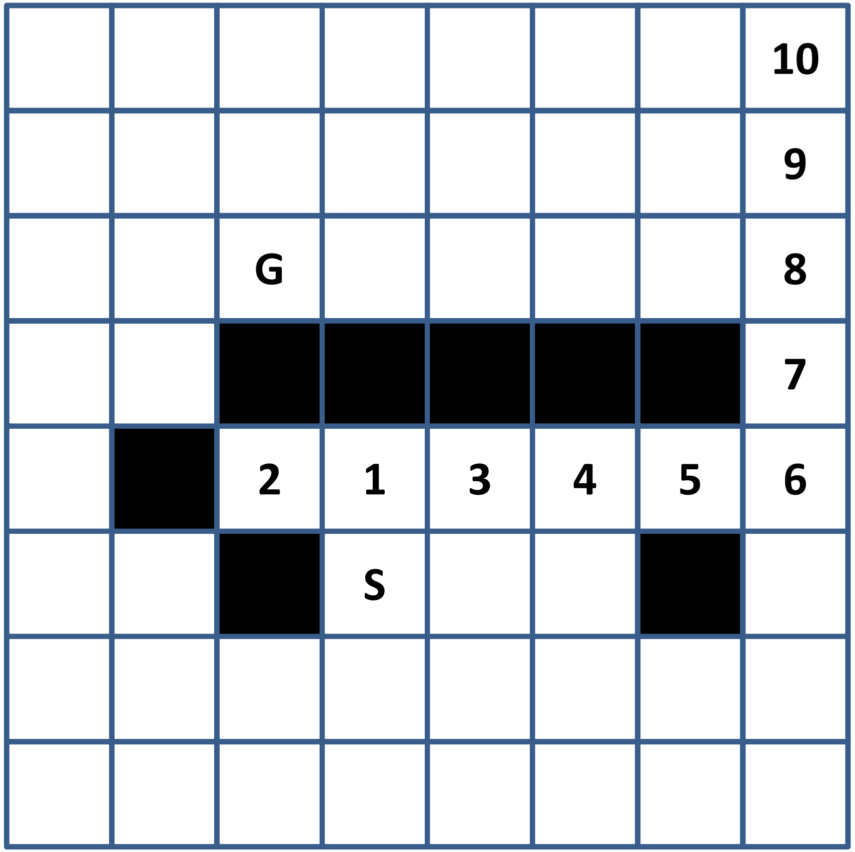
Depth-first Search





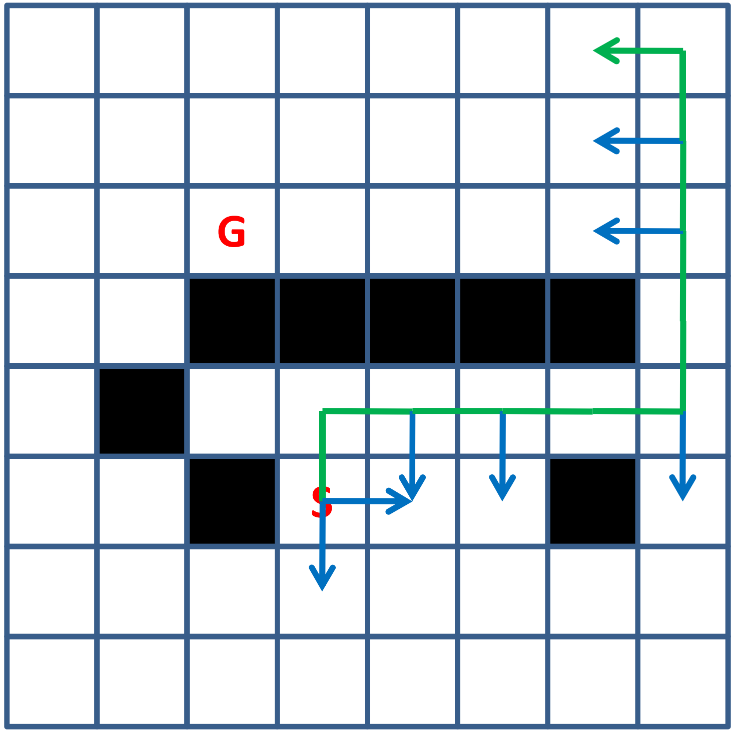
Depth-first Search

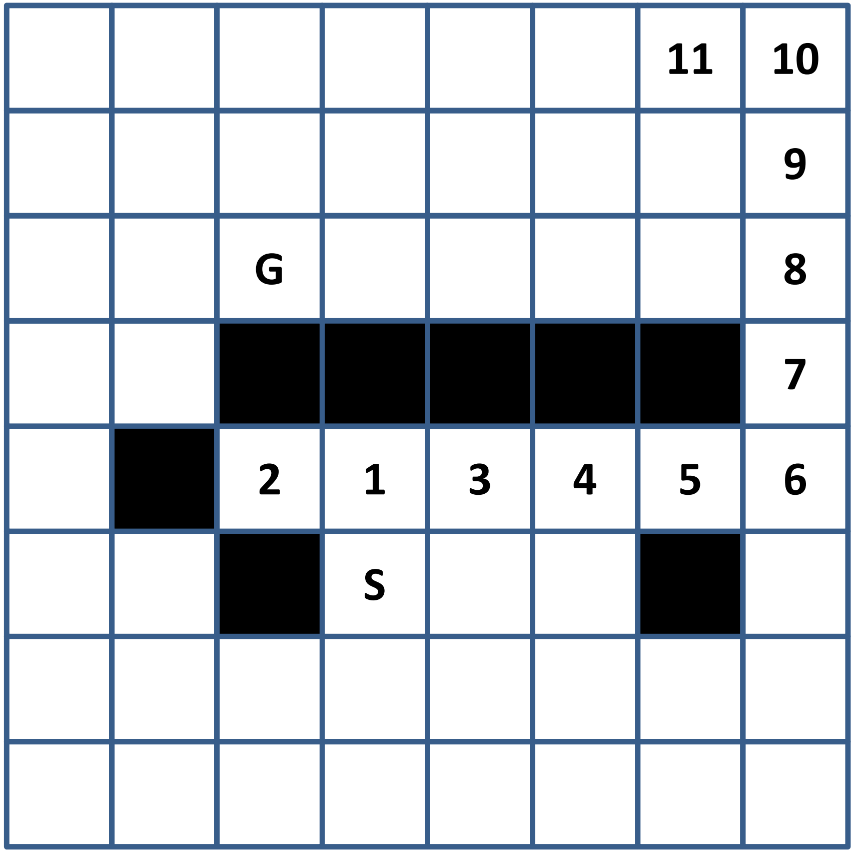




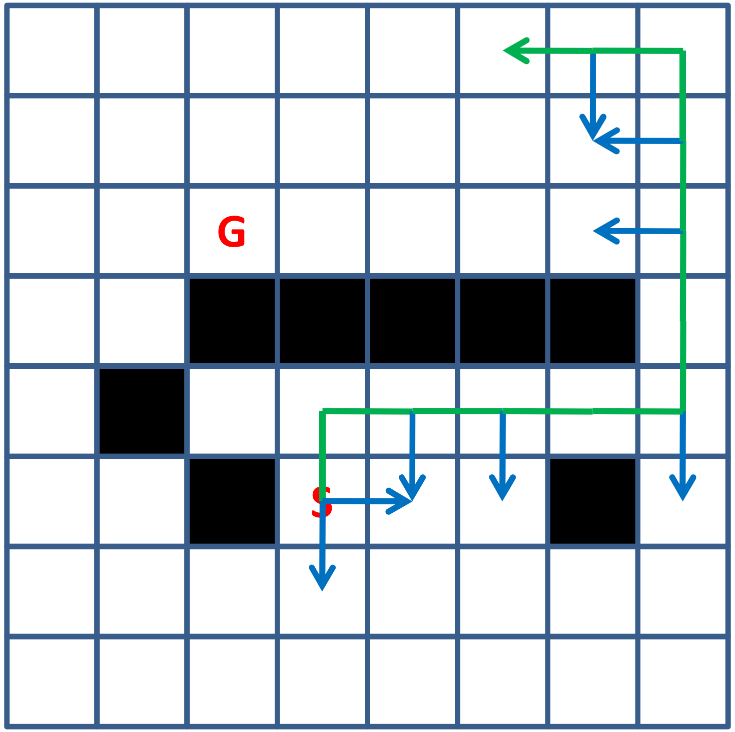
							10
							9
		G					8
		■	■	■	■	■	7
	■	2	1	3	4	5	6
		■	S			■	

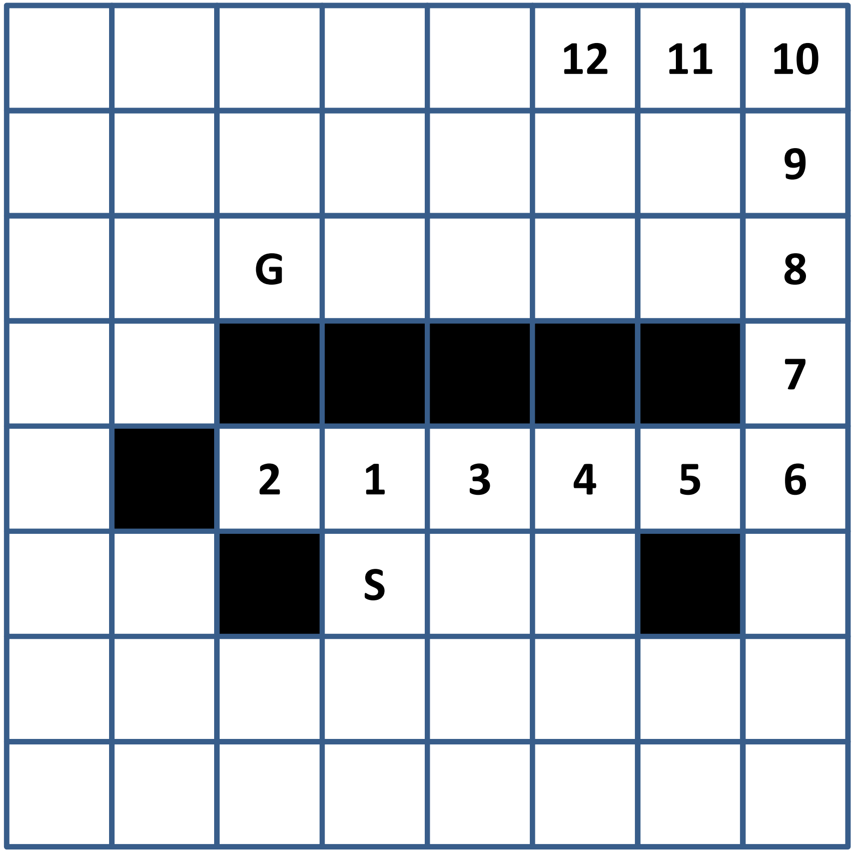
Depth-first Search



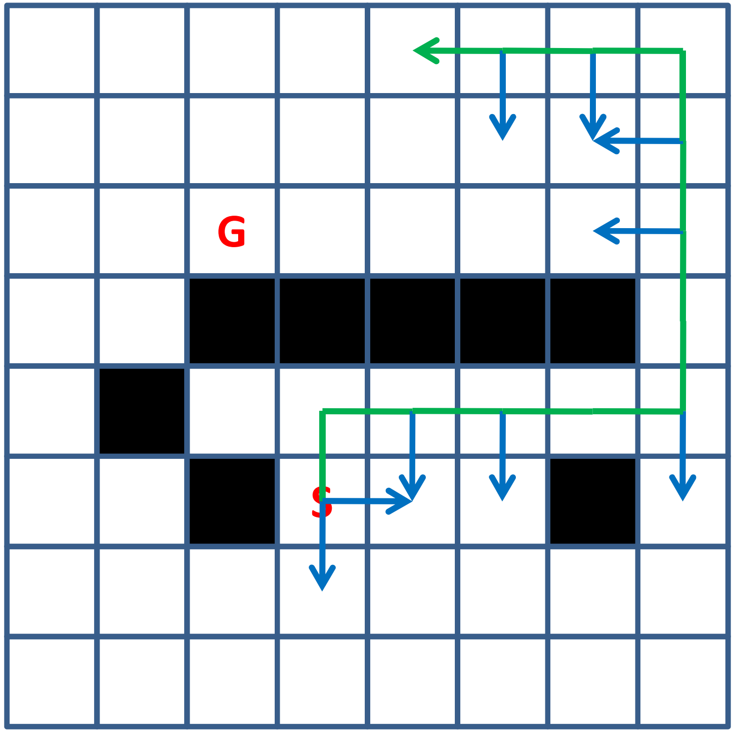


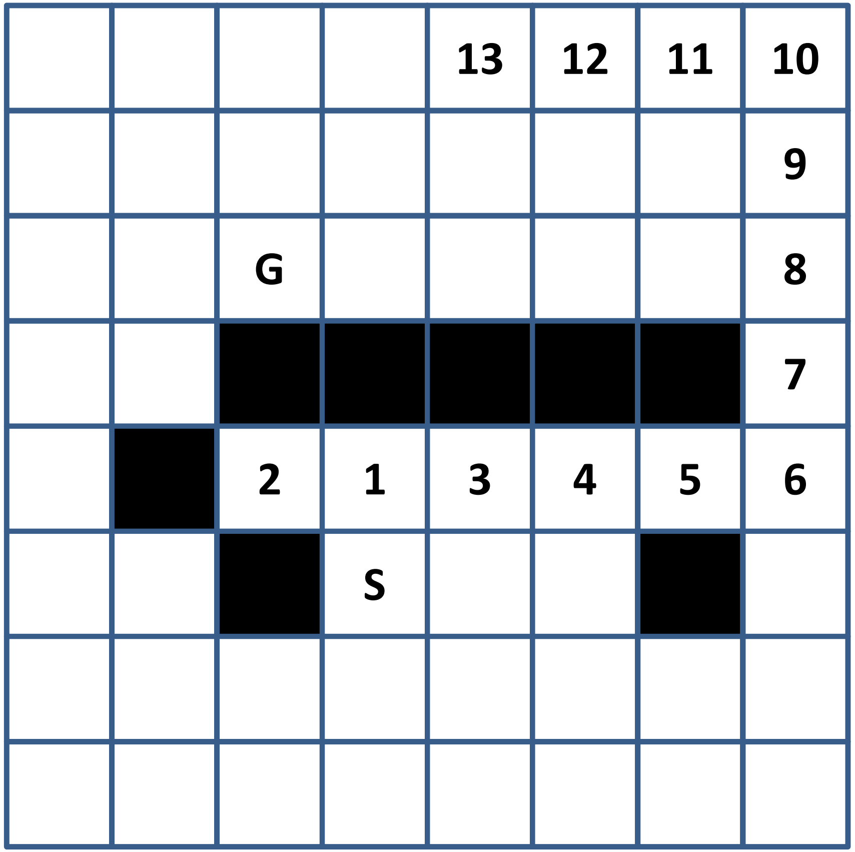
Depth-first Search





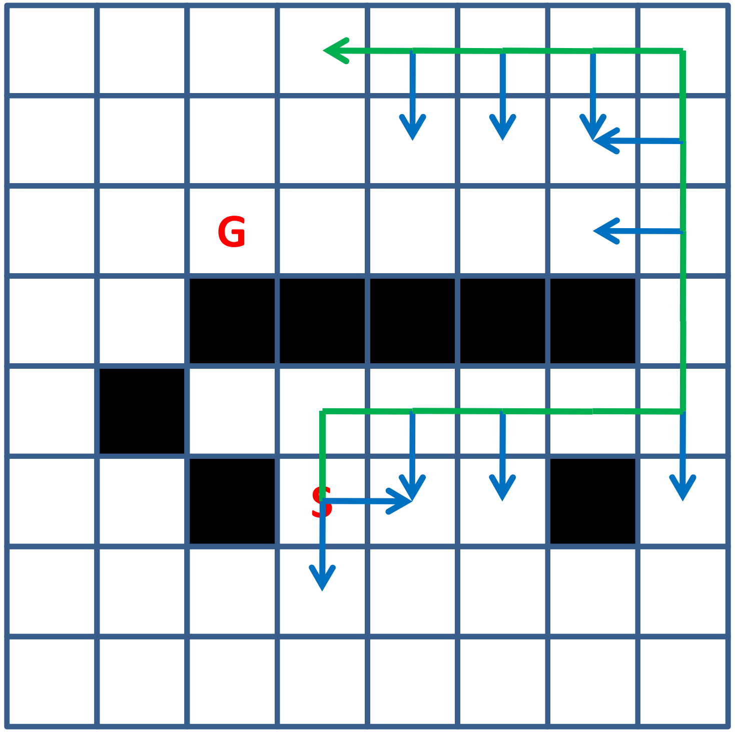
Depth-first Search

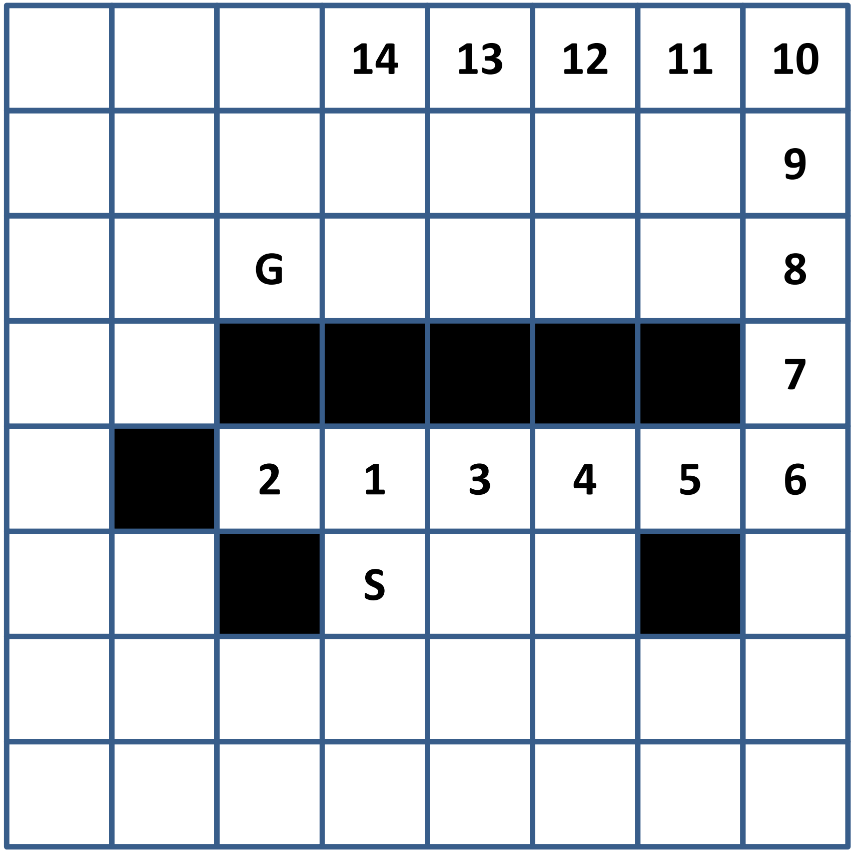




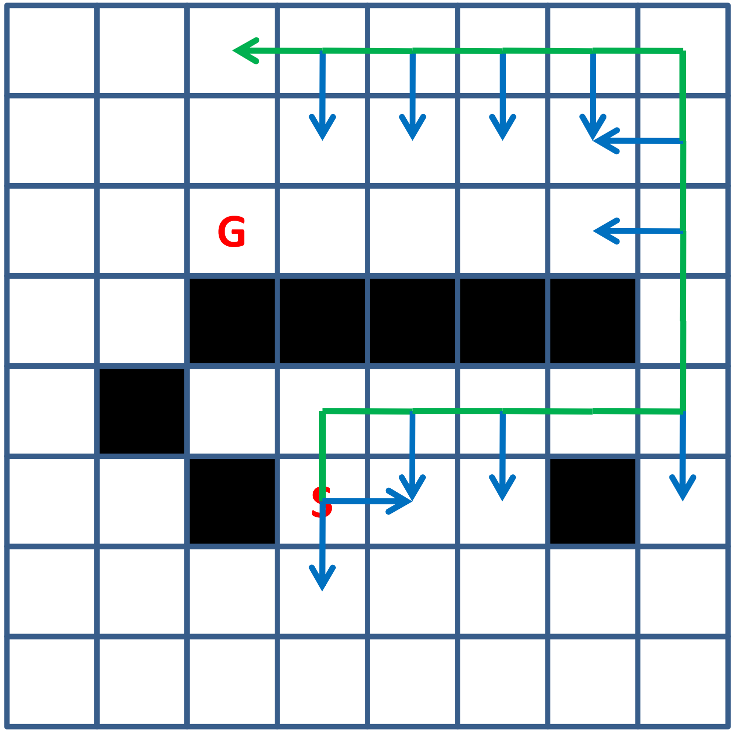
				13	12	11	10
							9
		G					8
		■	■	■	■	■	7
	■	2	1	3	4	5	6
		■	S			■	

Depth-first Search



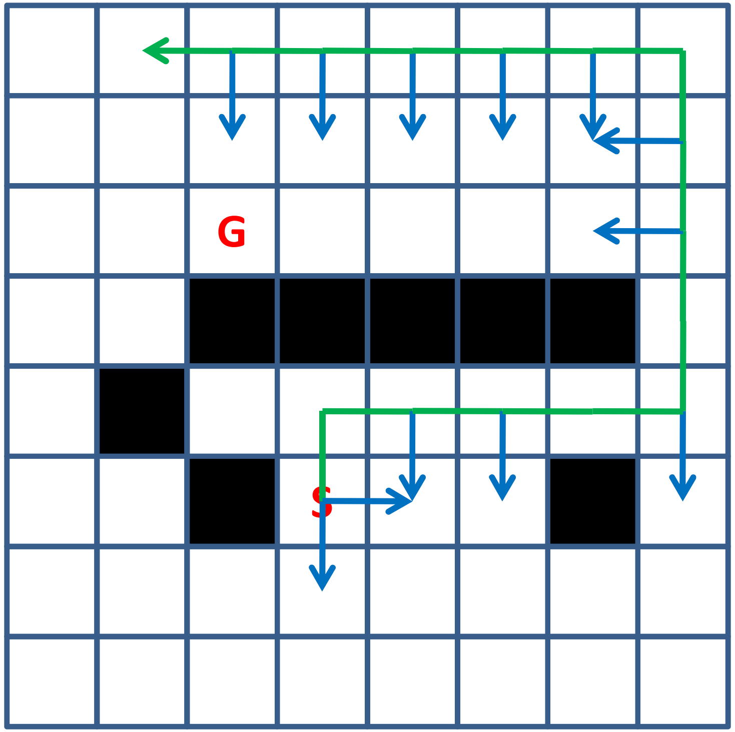


Depth-first Search



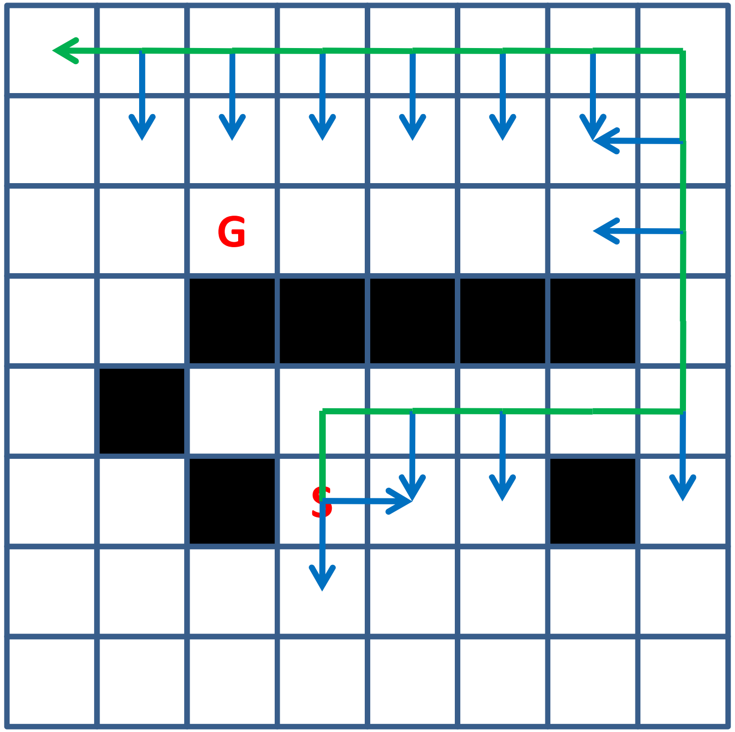
		15	14	13	12	11	10
							9
		G					8
		■	■	■	■	■	7
	■	2	1	3	4	5	6
		■	S			■	

Depth-first Search



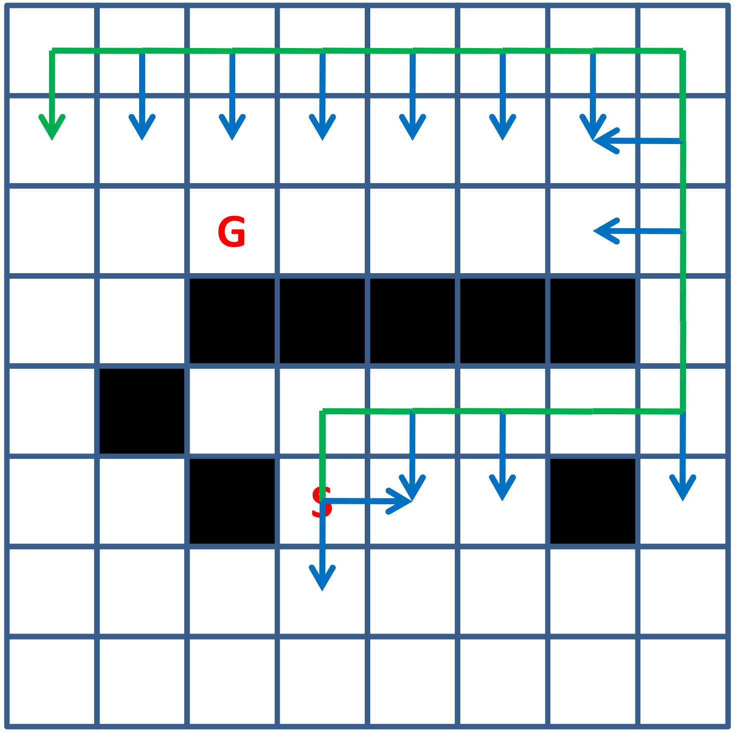
	16	15	14	13	12	11	10
							9
		G					8
		■	■	■	■	■	7
	■	2	1	3	4	5	6
		■	S			■	

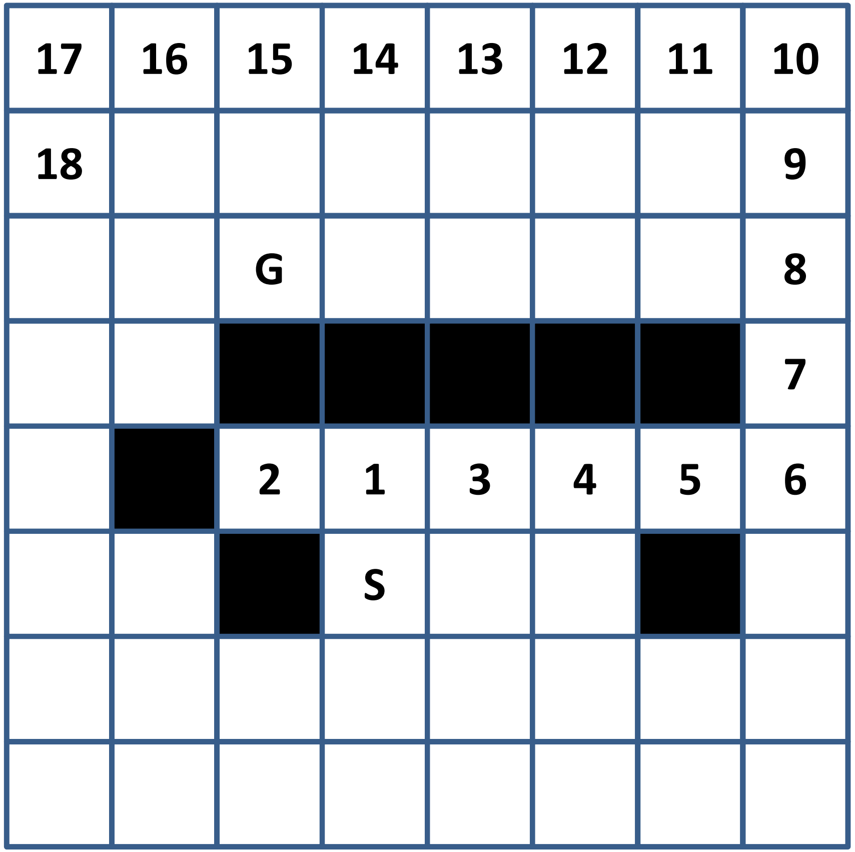
Depth-first Search



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

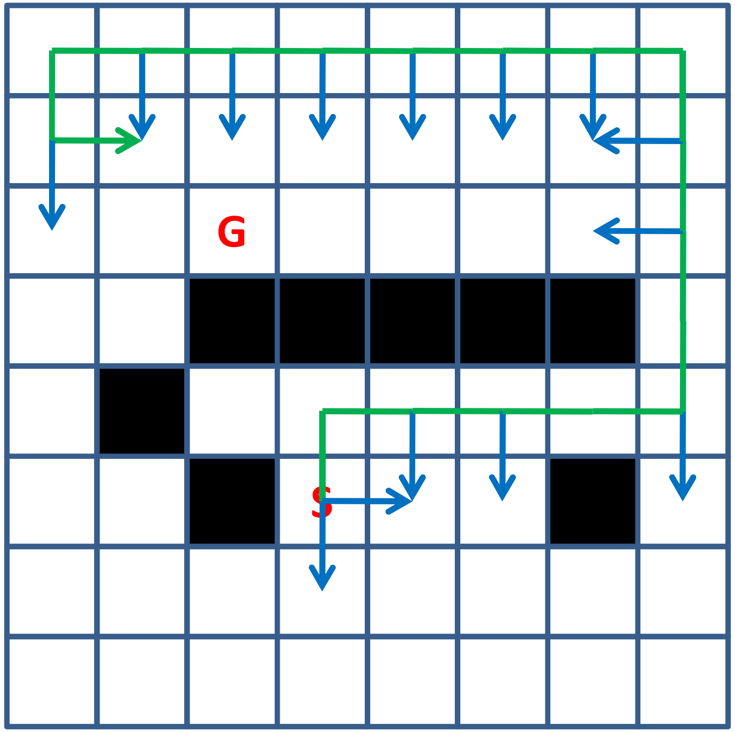
Depth-first Search





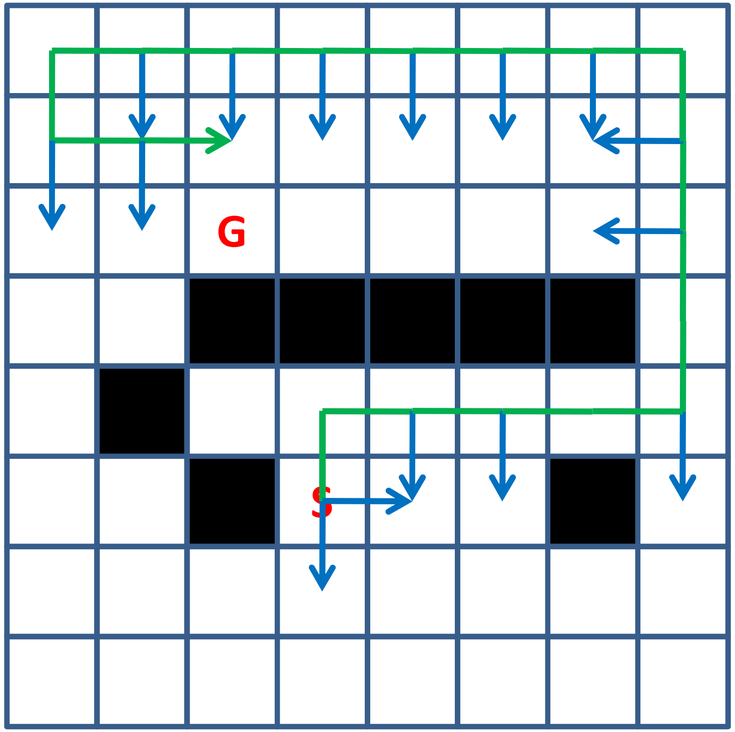
17	16	15	14	13	12	11	10
18							9
		G					8
		■	■	■	■	■	7
	■	2	1	3	4	5	6
		■	S			■	

Depth-first Search



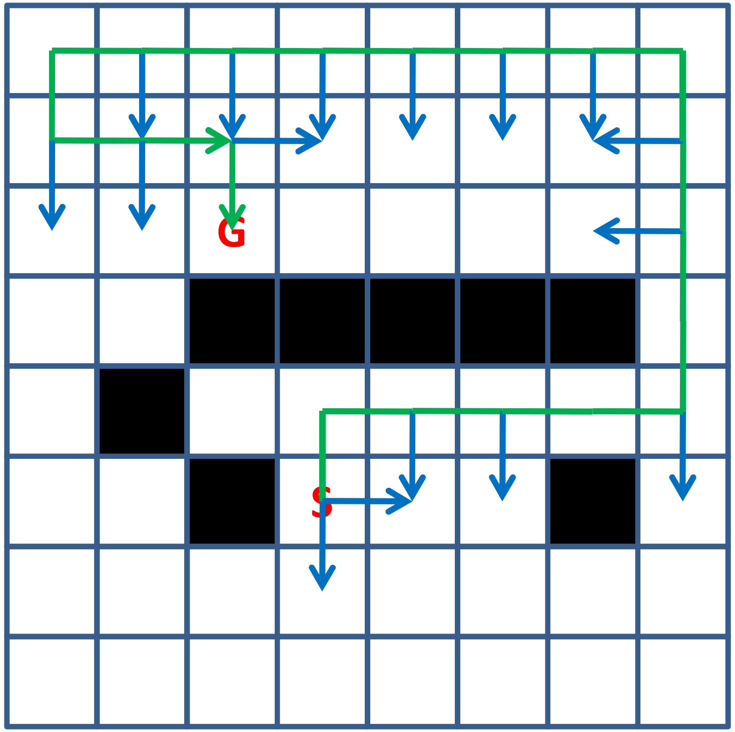
17	16	15	14	13	12	11	10
18	19						9
		G					8
		■	■	■	■	■	7
	■	2	1	3	4	5	6
		■	S			■	

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			■	

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

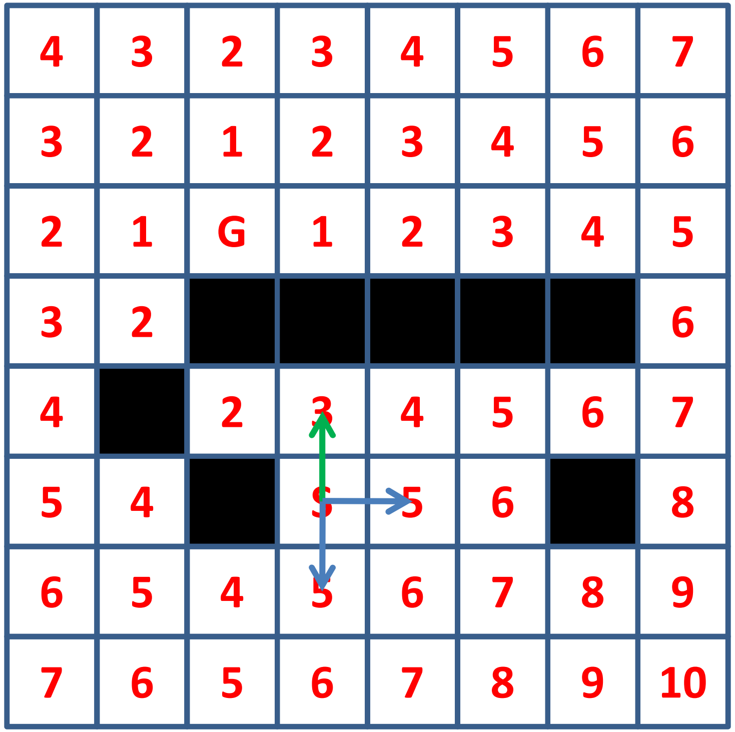
Heuristic: Manhattan Distance

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

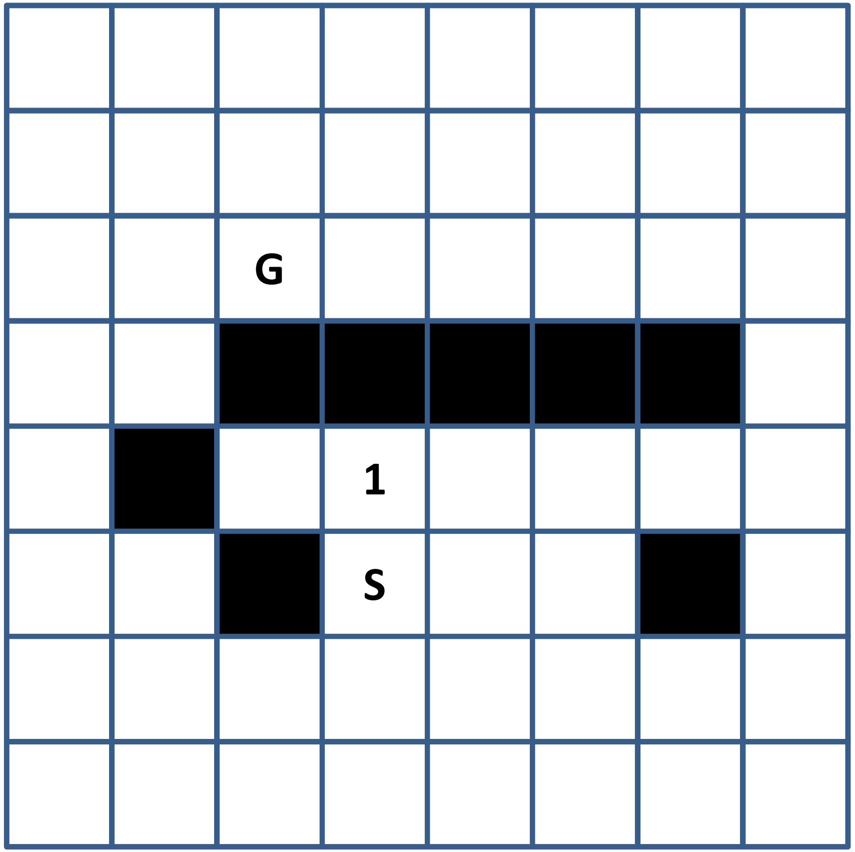
Hill-climbing I Search

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

Hill-climbing I Search

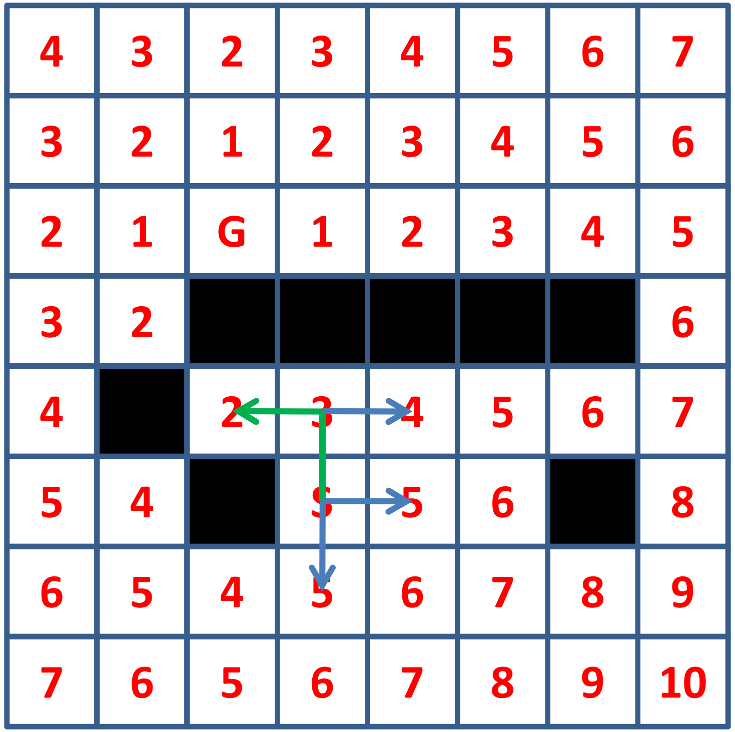


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

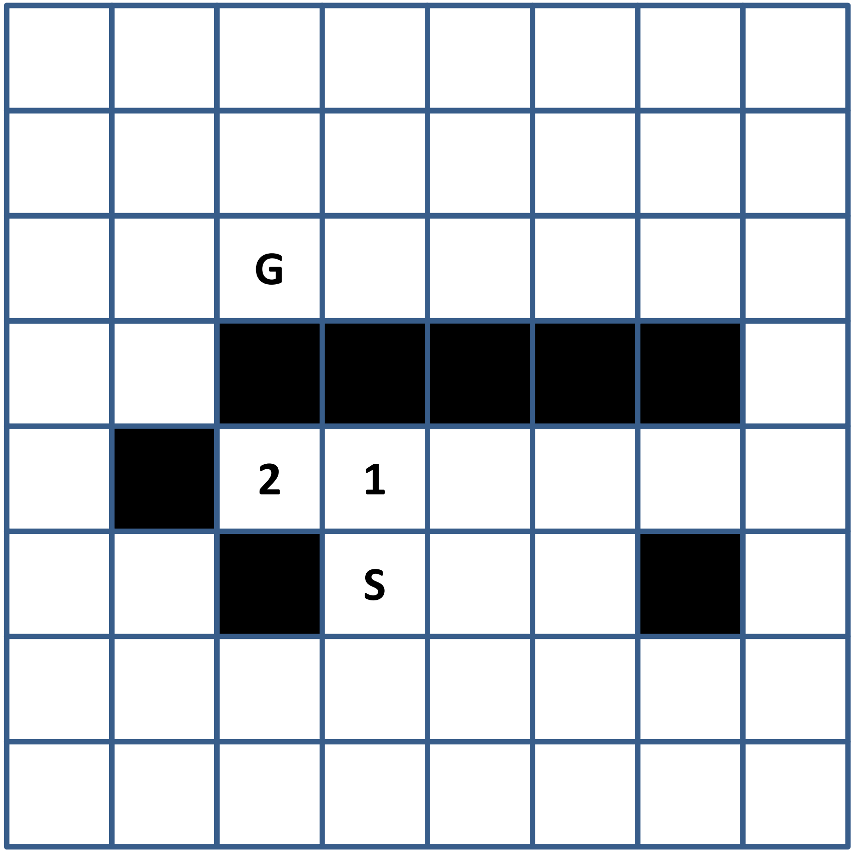


		G					
		■	■	■	■	■	
	■		1				
		■	S			■	

Hill-climbing I Search

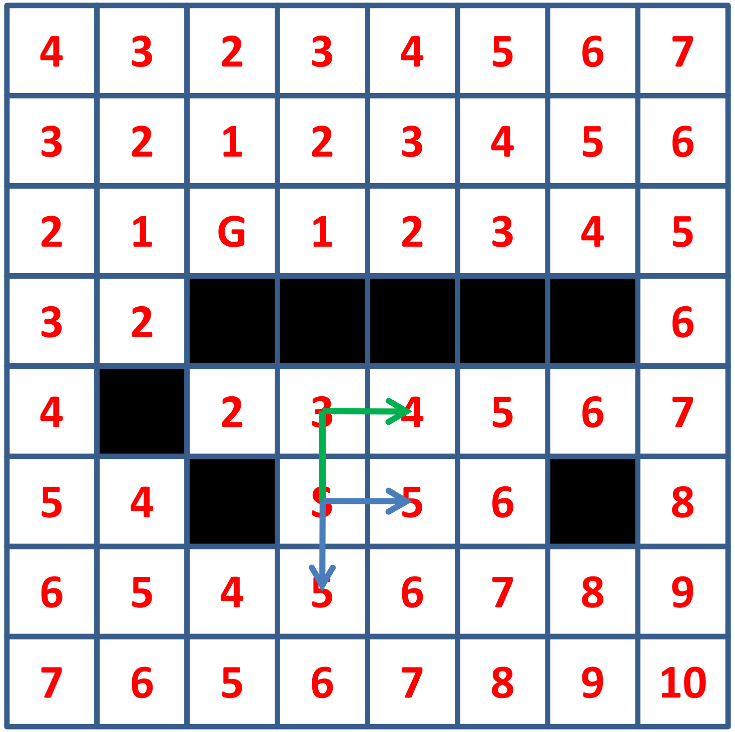


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

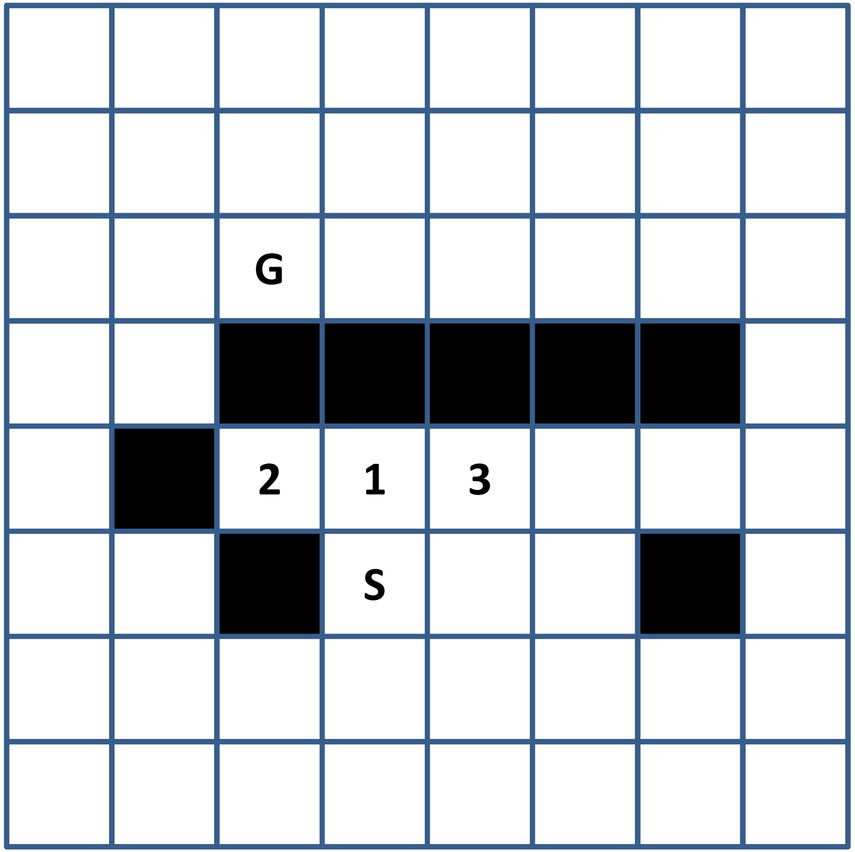


		G					
		■	■	■	■	■	
	■	2	1				
		■	S			■	

Hill-climbing I Search

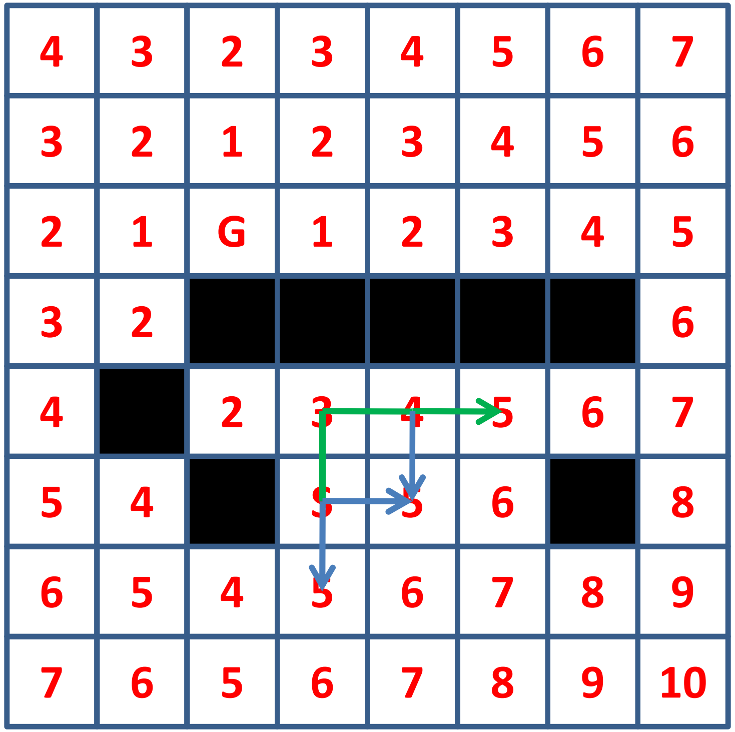


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

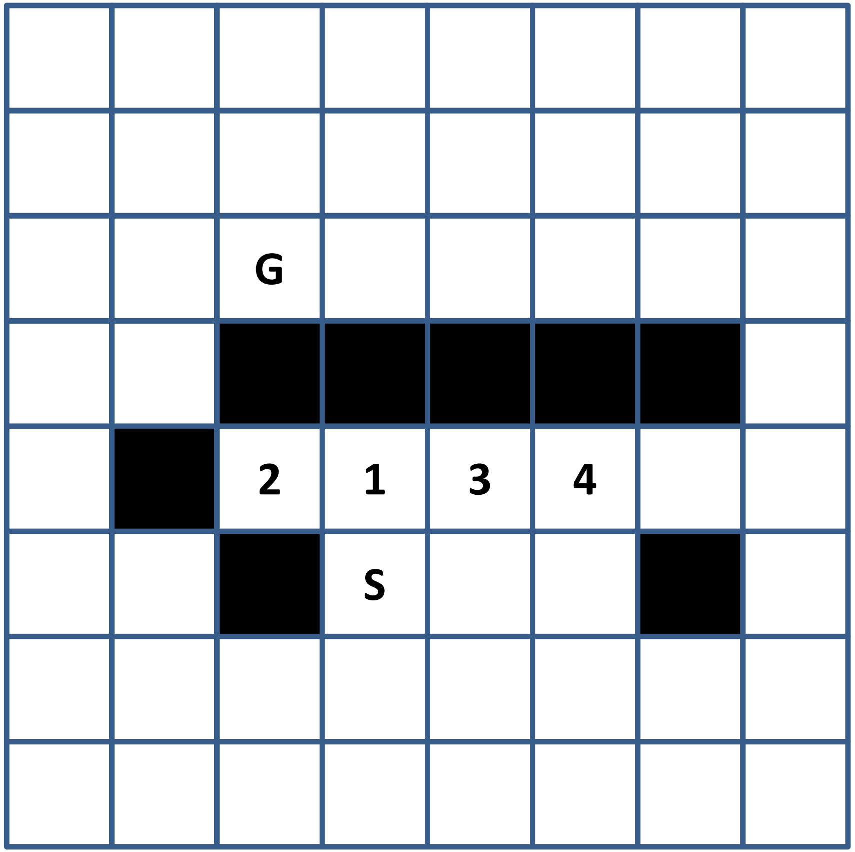


		G					
		■	■	■	■	■	
	■	2	1	3			
		■	S			■	

Hill-climbing I Search

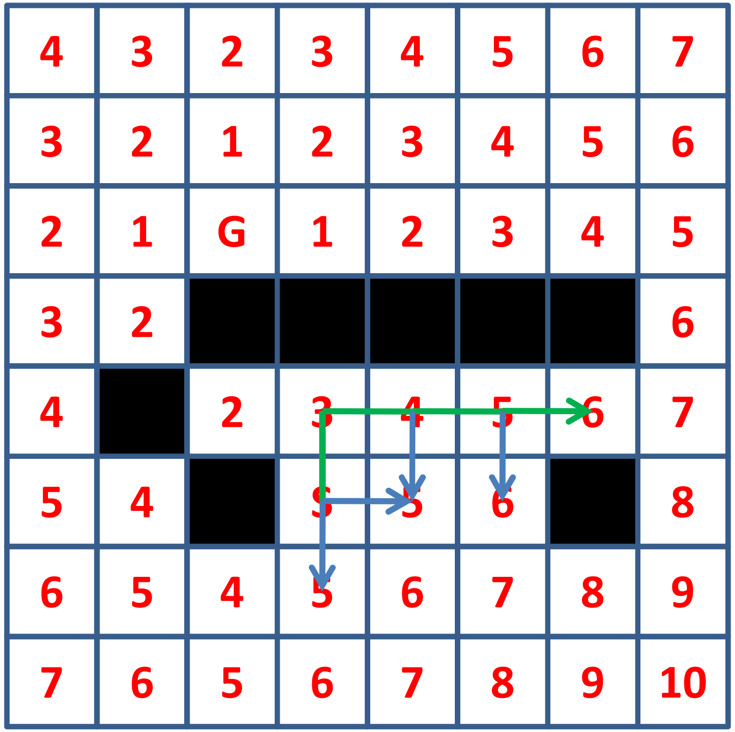


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

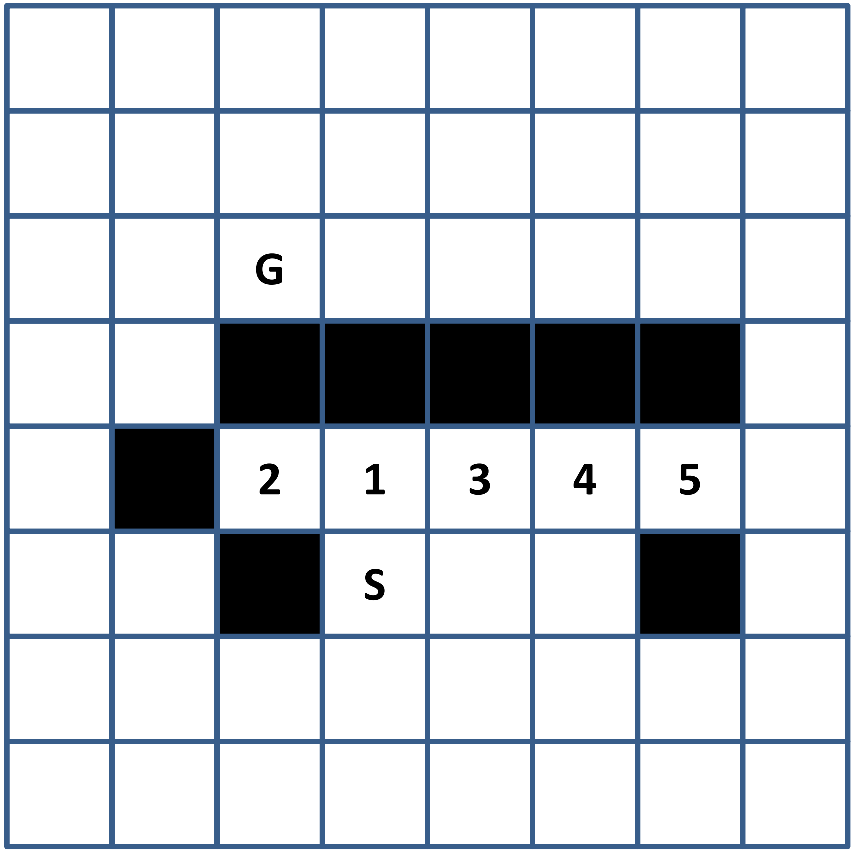


		G					
		■	■	■	■	■	
	■	2	1	3	4		
		■	S			■	

Hill-climbing I Search

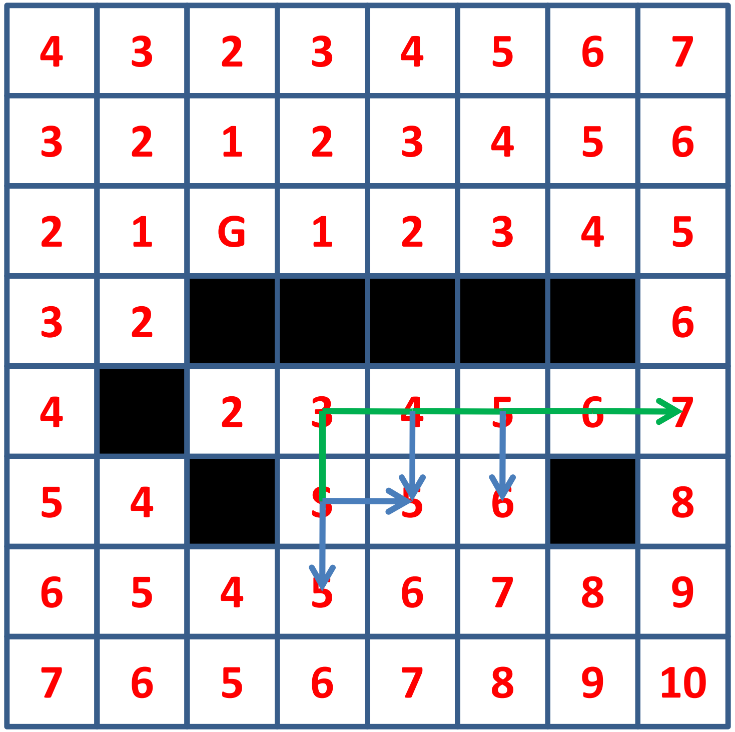


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10



		G					
		■	■	■	■	■	
	■	2	1	3	4	5	
		■	S			■	

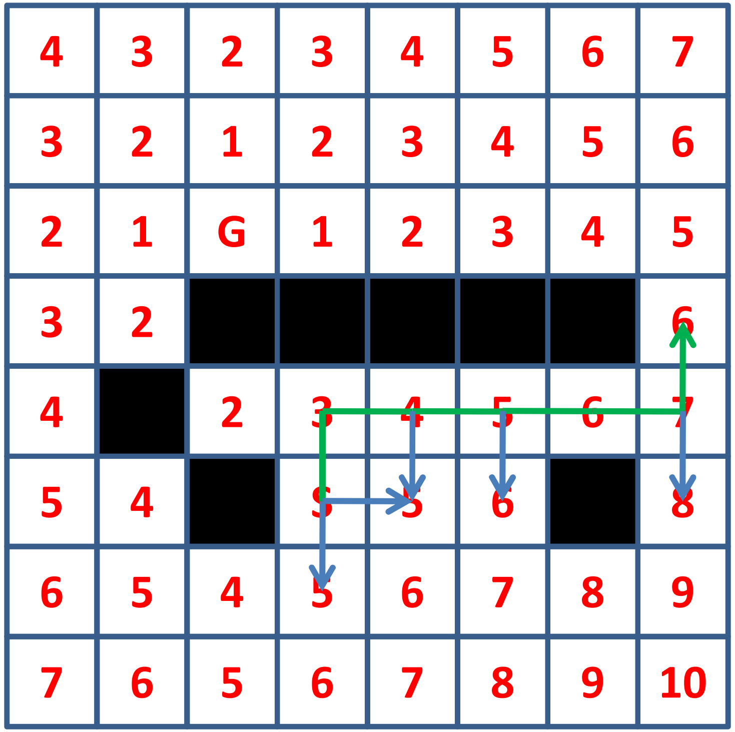
Hill-climbing I Search



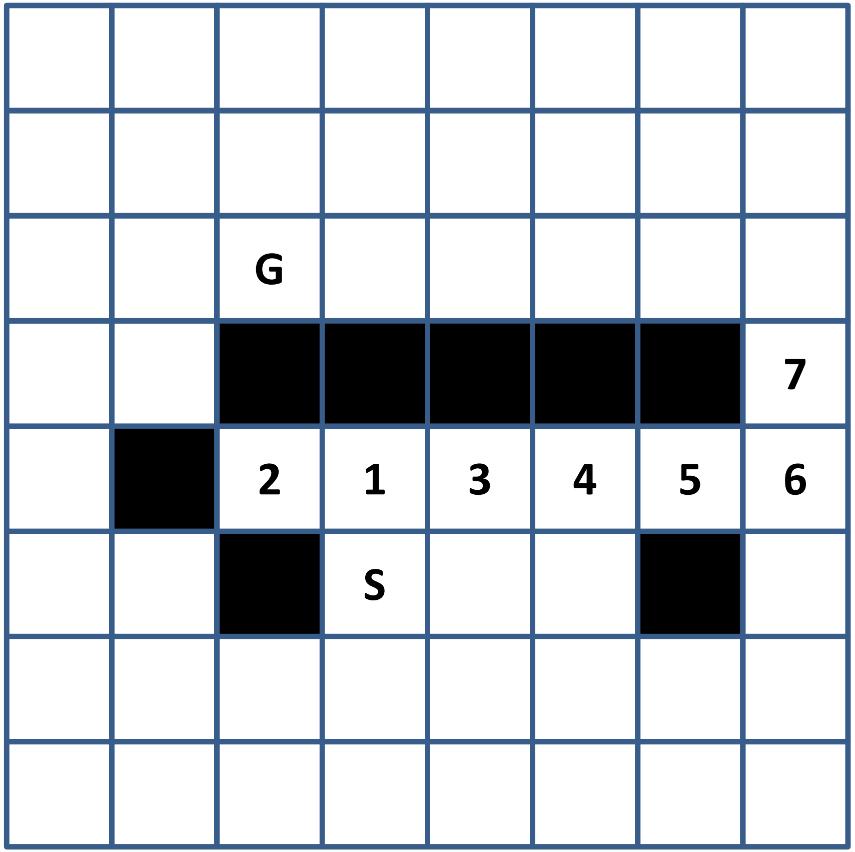
4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					
		■	■	■	■	■	
	■	2	1	3	4	5	6
		■	S			■	

Hill-climbing I Search

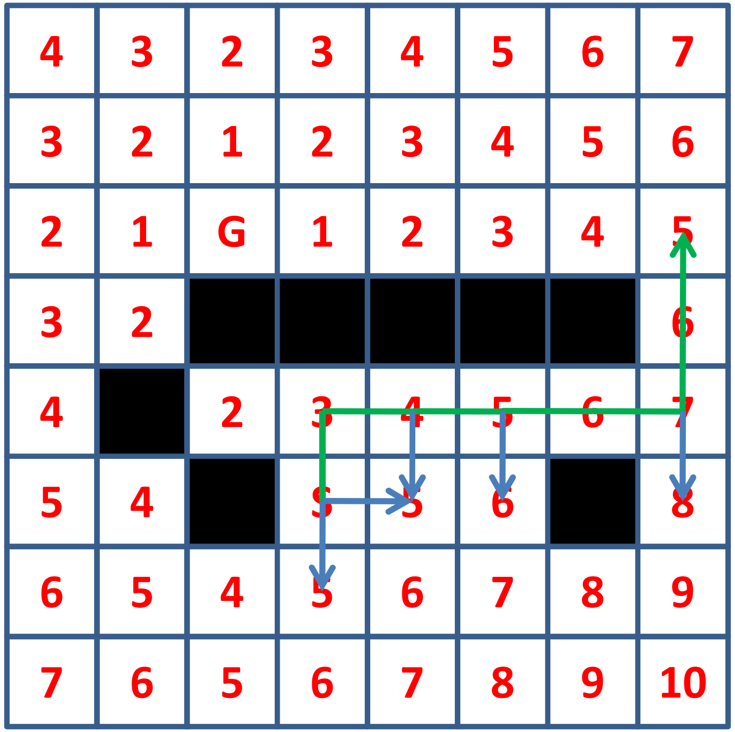


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

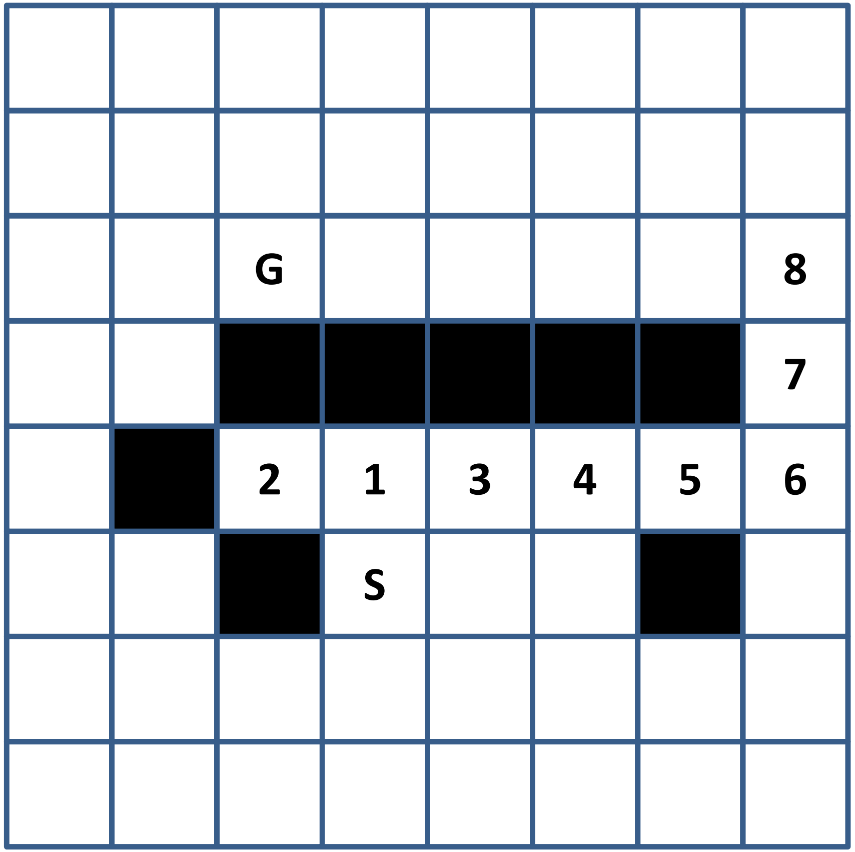


		G					
		■	■	■	■	■	7
	■	2	1	3	4	5	6
		■	S			■	

Hill-climbing I Search

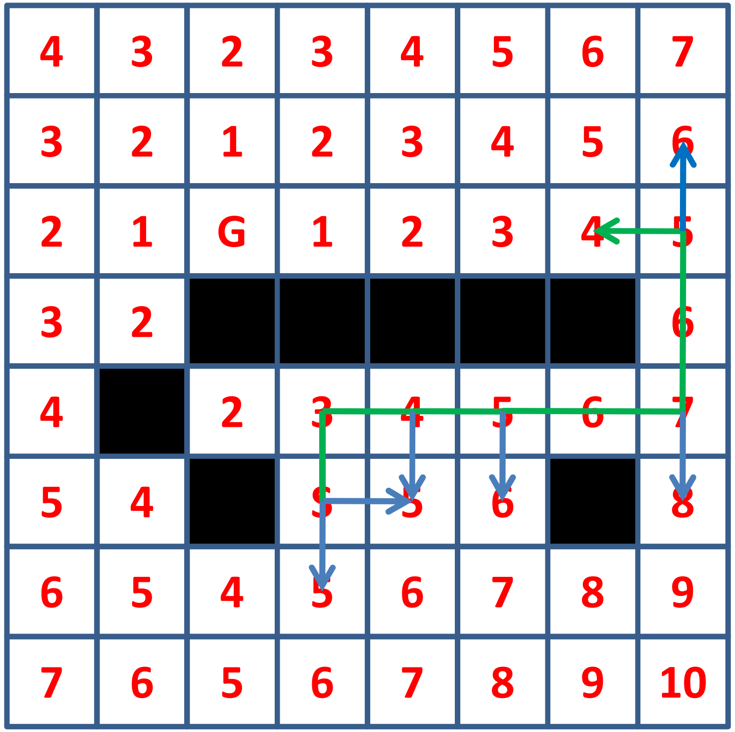


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

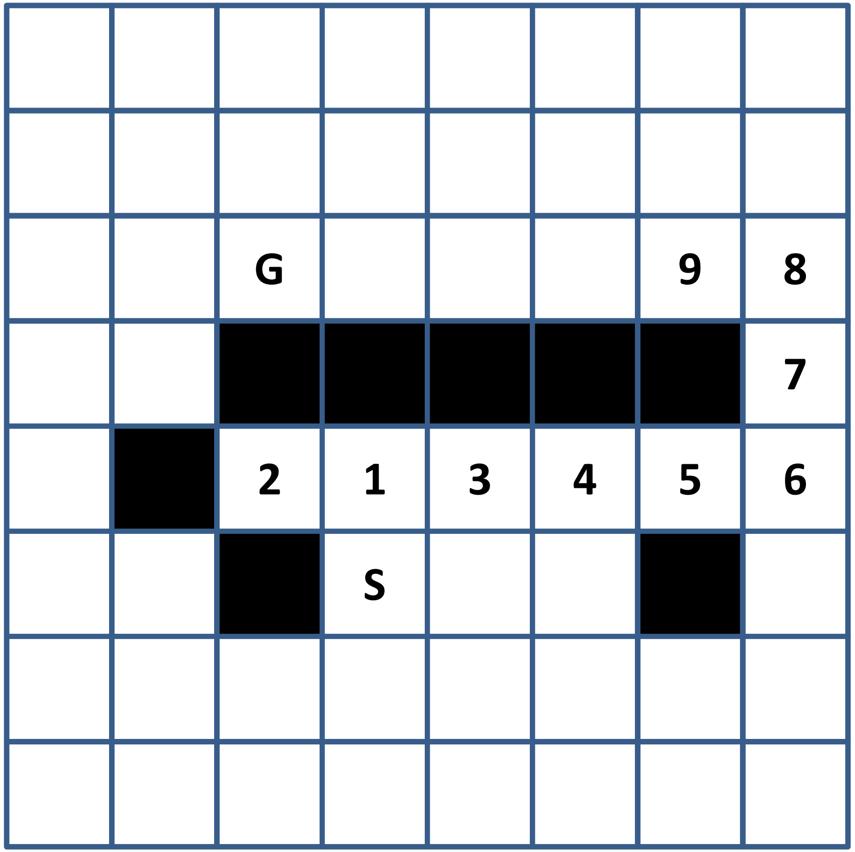


		G					8
		■	■	■	■	■	7
	■	2	1	3	4	5	6
		■	S			■	

Hill-climbing I Search

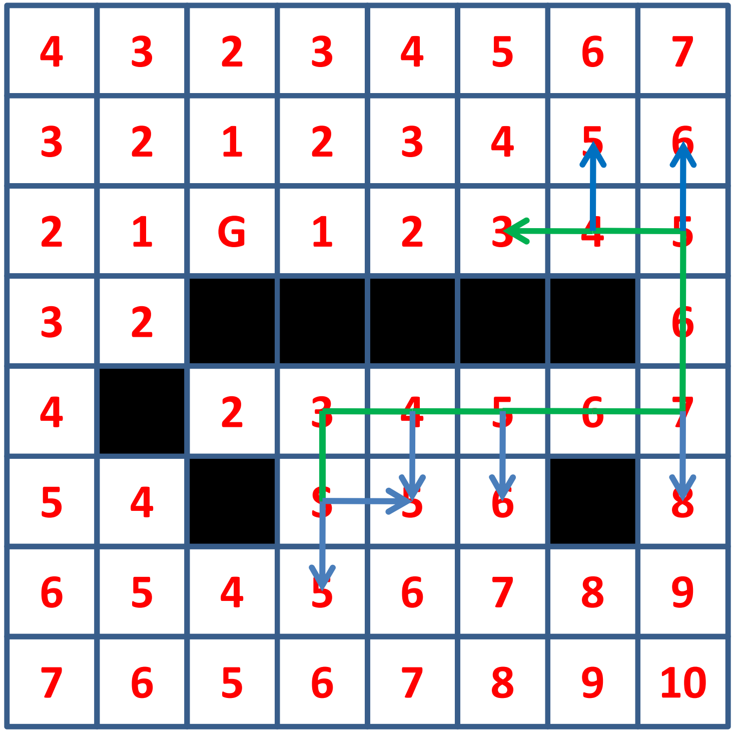


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

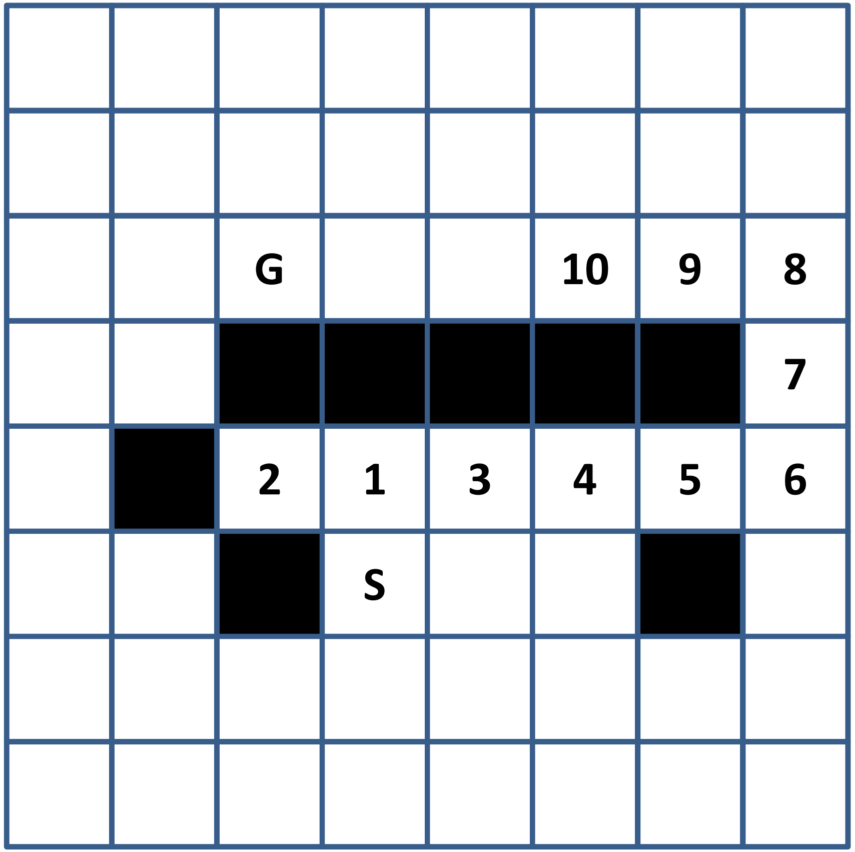


		G				9	8
		■	■	■	■	■	7
	■	2	1	3	4	5	6
		■	S			■	

Hill-climbing I Search

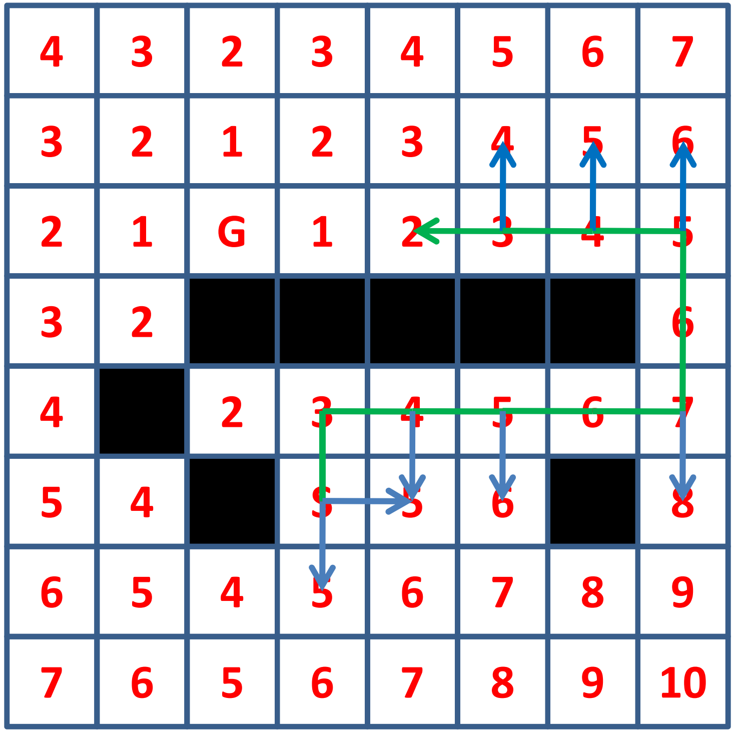


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

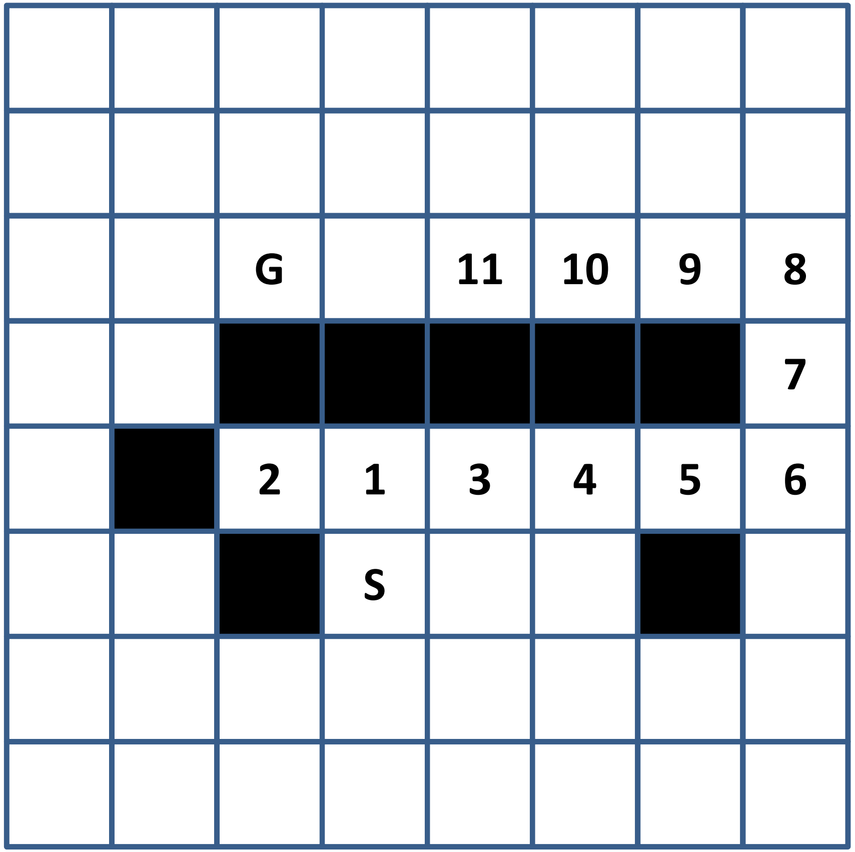


		G			10	9	8
		■	■	■	■	■	7
	■	2	1	3	4	5	6
		■	S			■	

Hill-climbing I Search

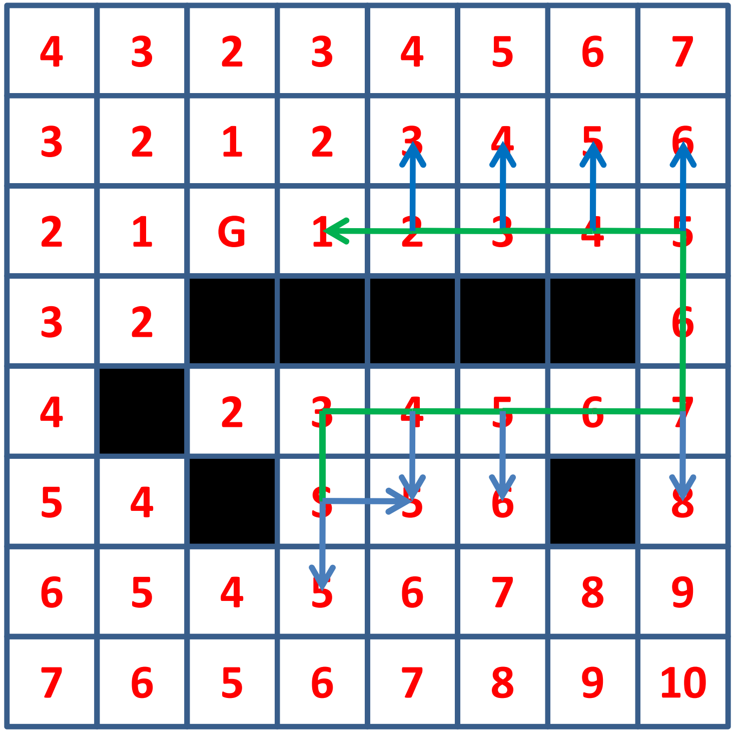


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10



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

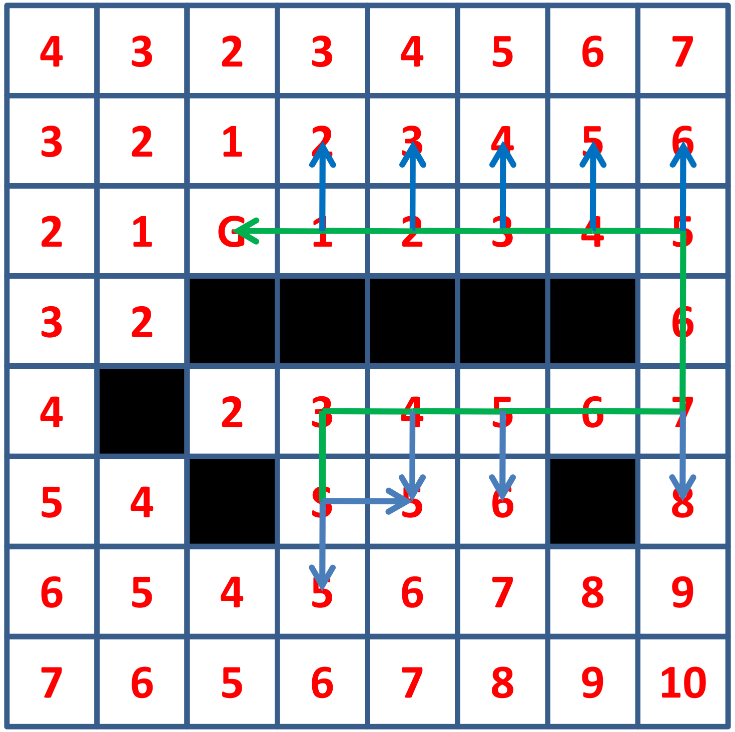
Hill-climbing I Search



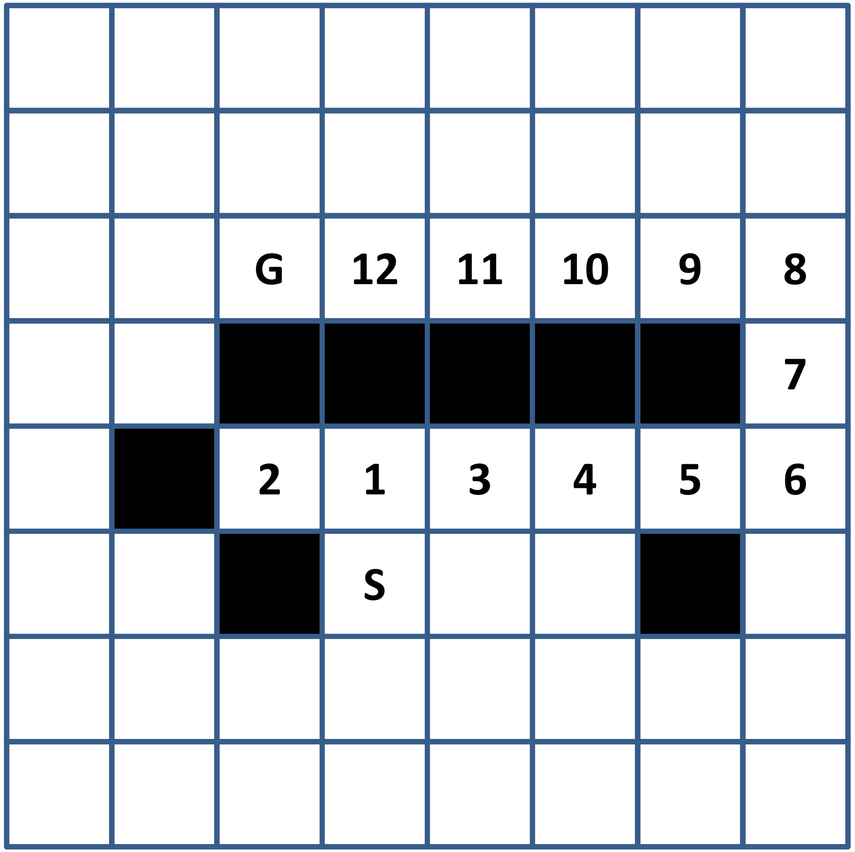
4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		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	■	S	5	6	■	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			■	

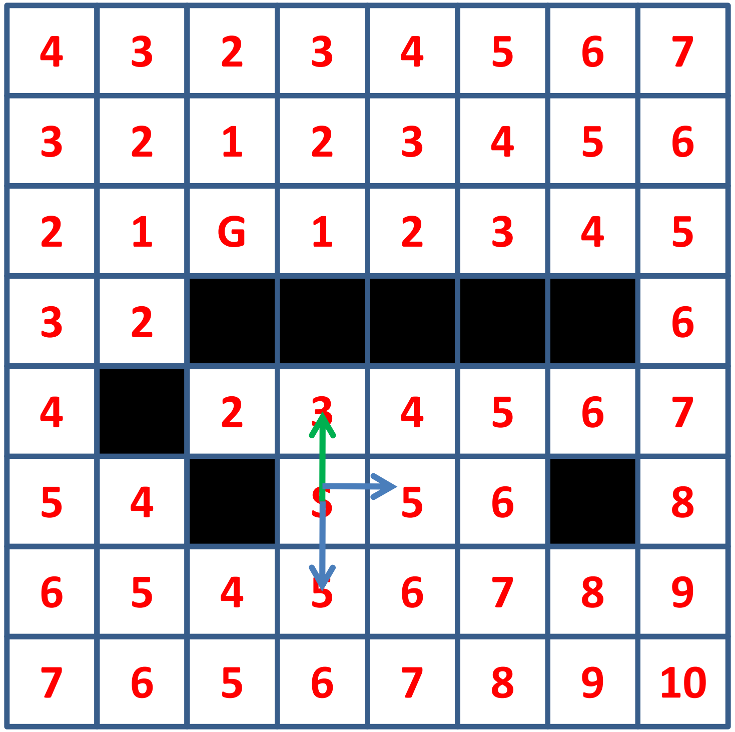
Path Search

GREEDY SEARCH

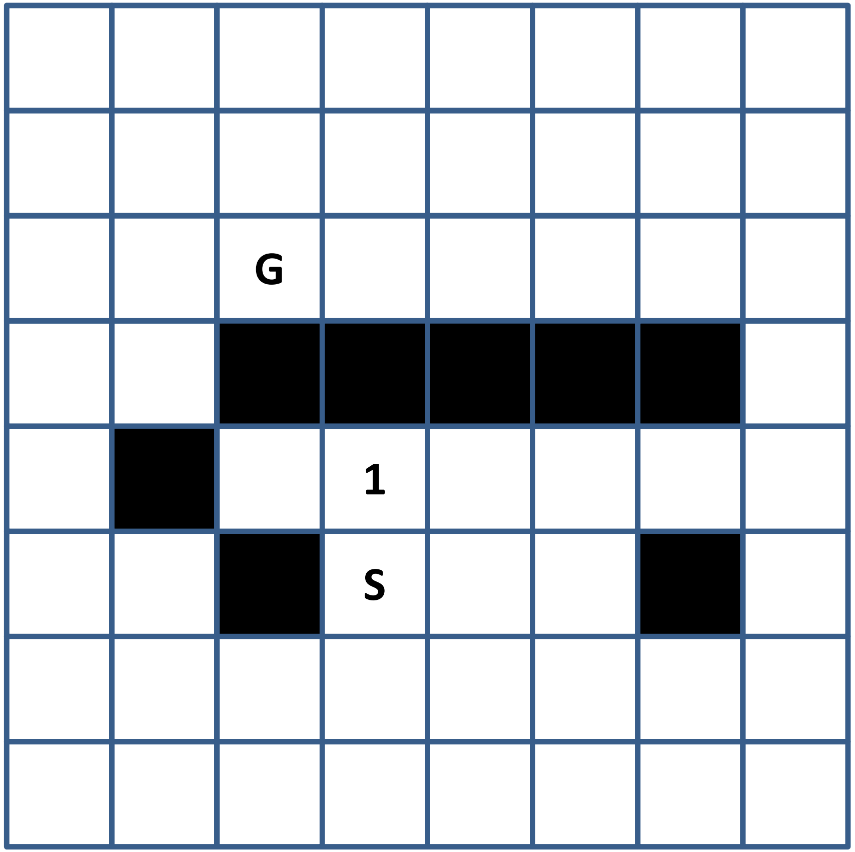
Greedy Search

- ***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

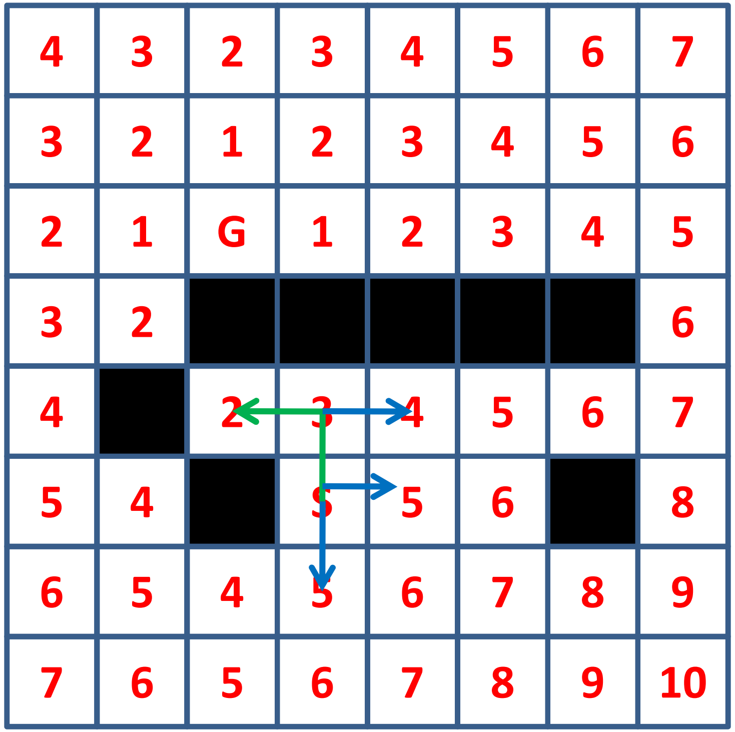


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

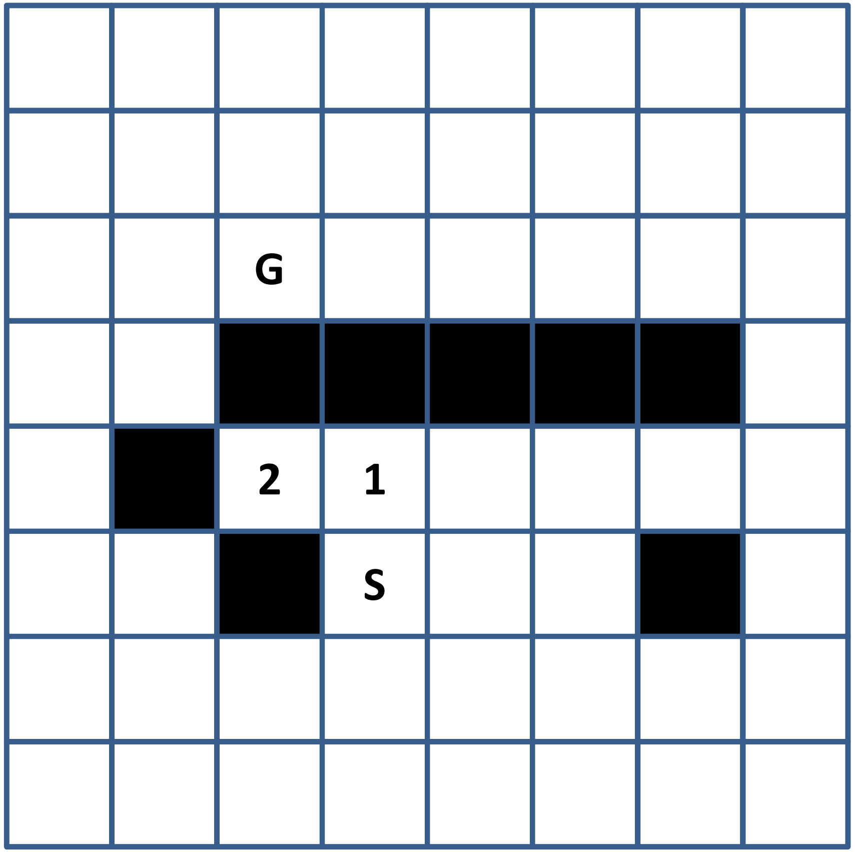


		G					
		■	■	■	■	■	
	■		1				
		■	S			■	

Greedy Search

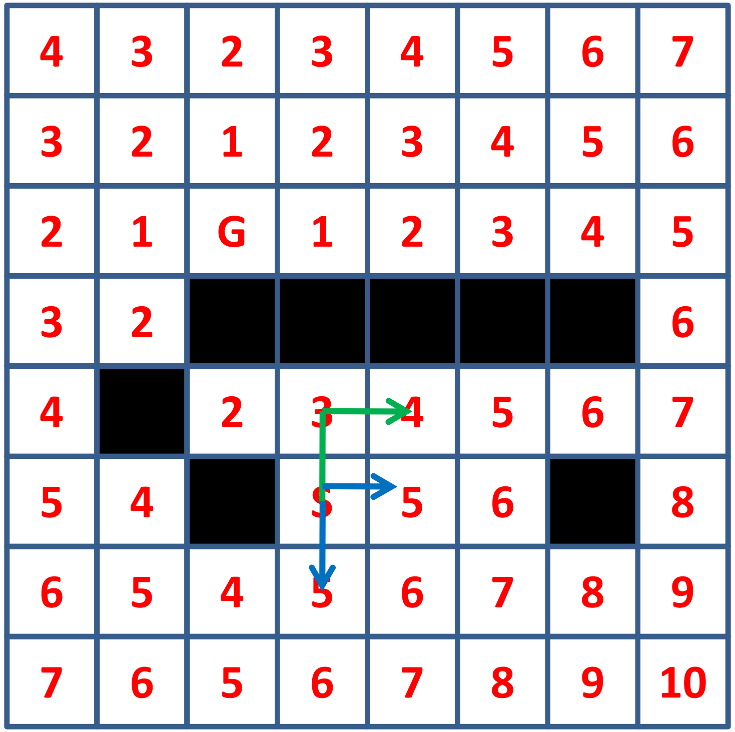


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

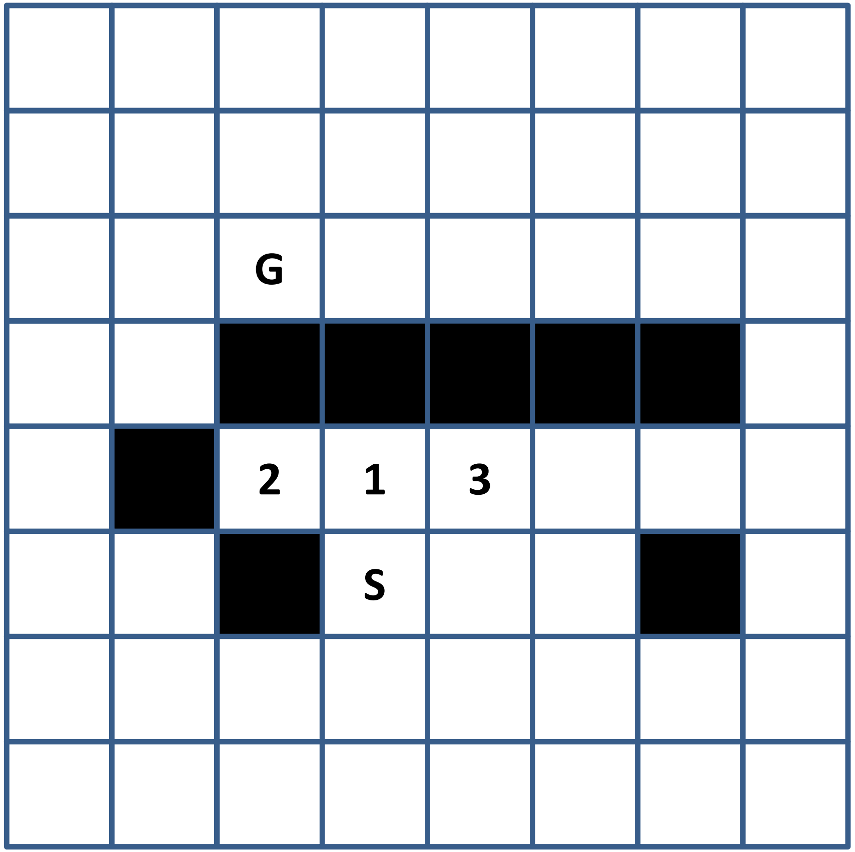


		G					
		■	■	■	■	■	
	■	2	1				
		■	S			■	

Greedy Search

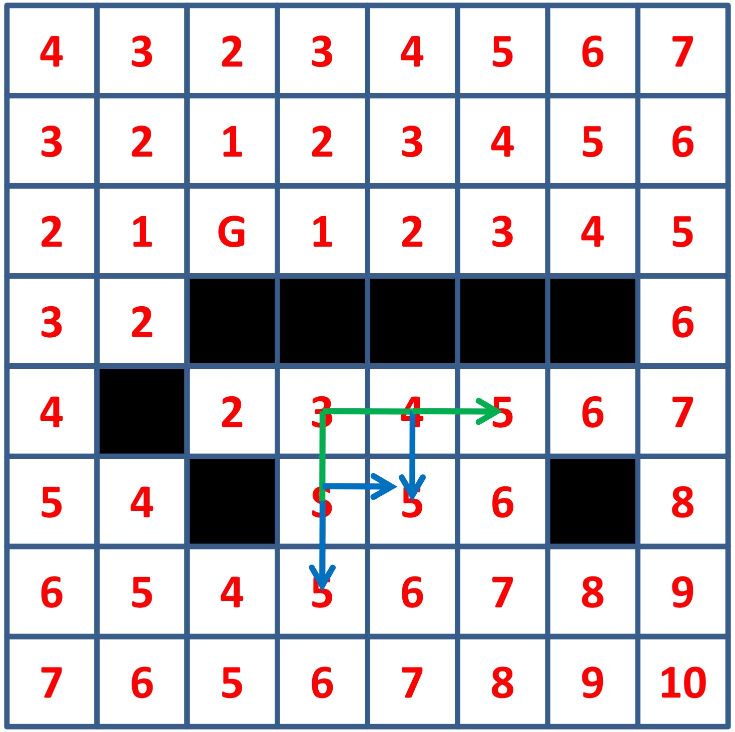


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

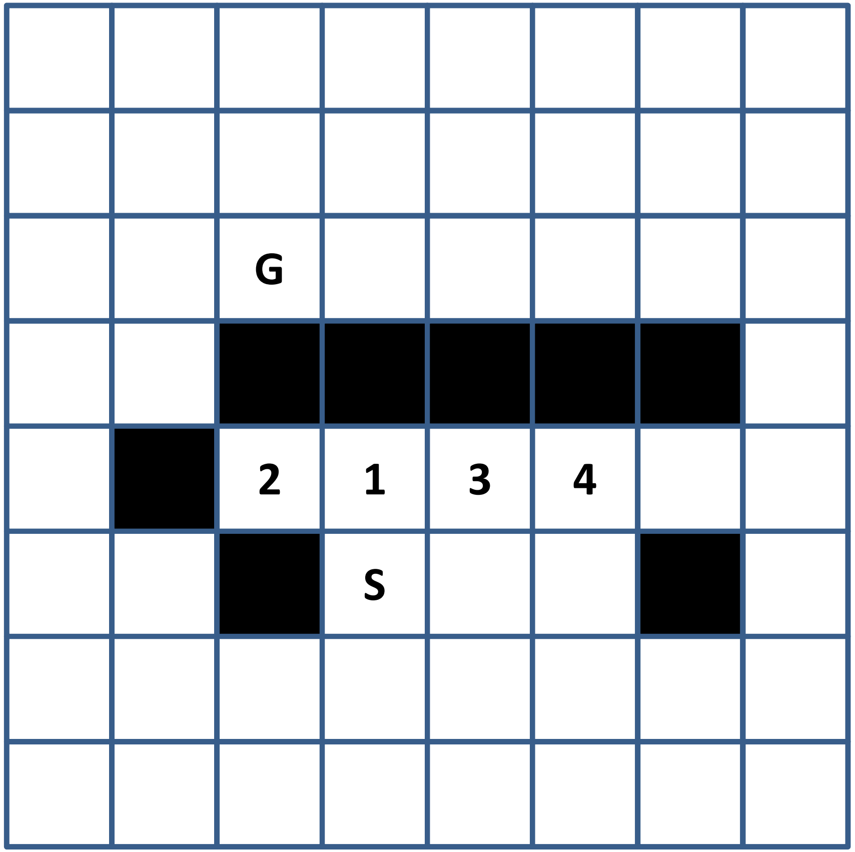


		G					
		■	■	■	■	■	
	■	2	1	3			
		■	S			■	

Greedy Search

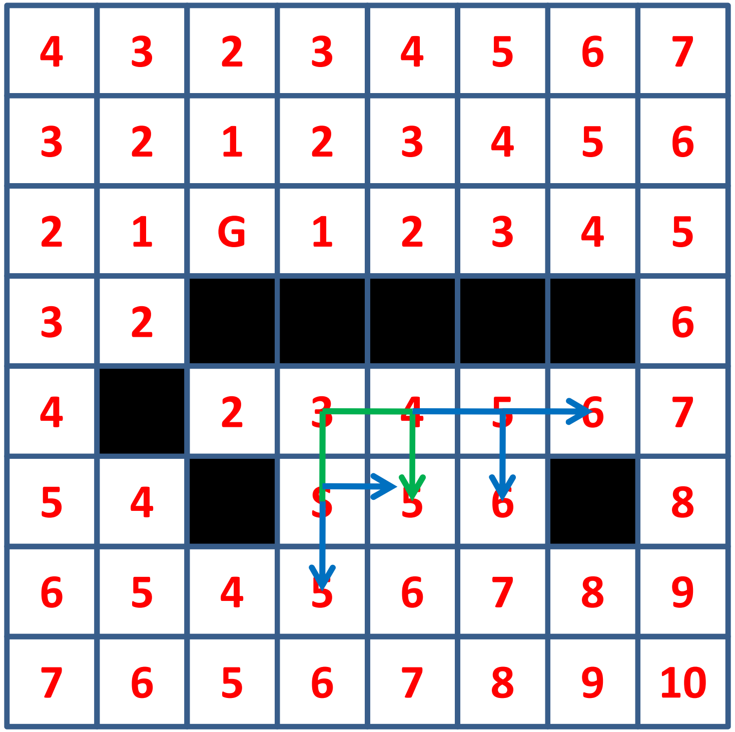


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

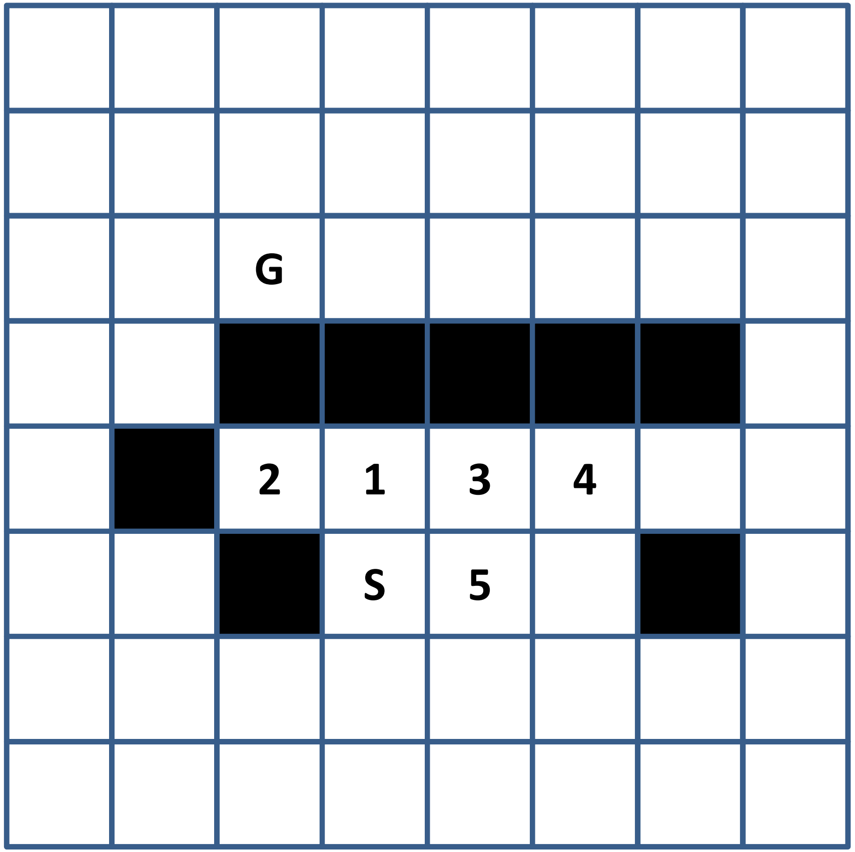


		G					
		■	■	■	■	■	
	■	2	1	3	4		
		■	S			■	

Greedy Search

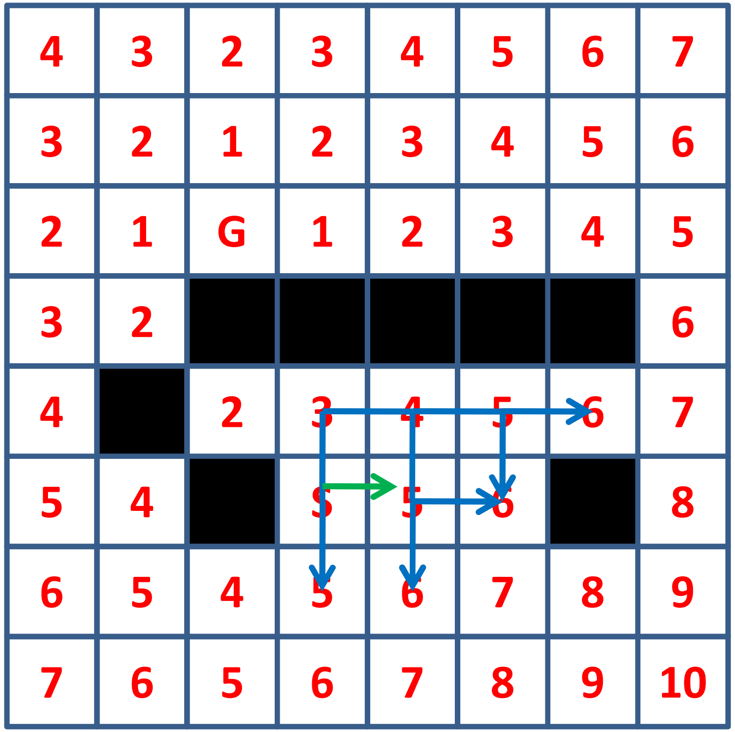


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

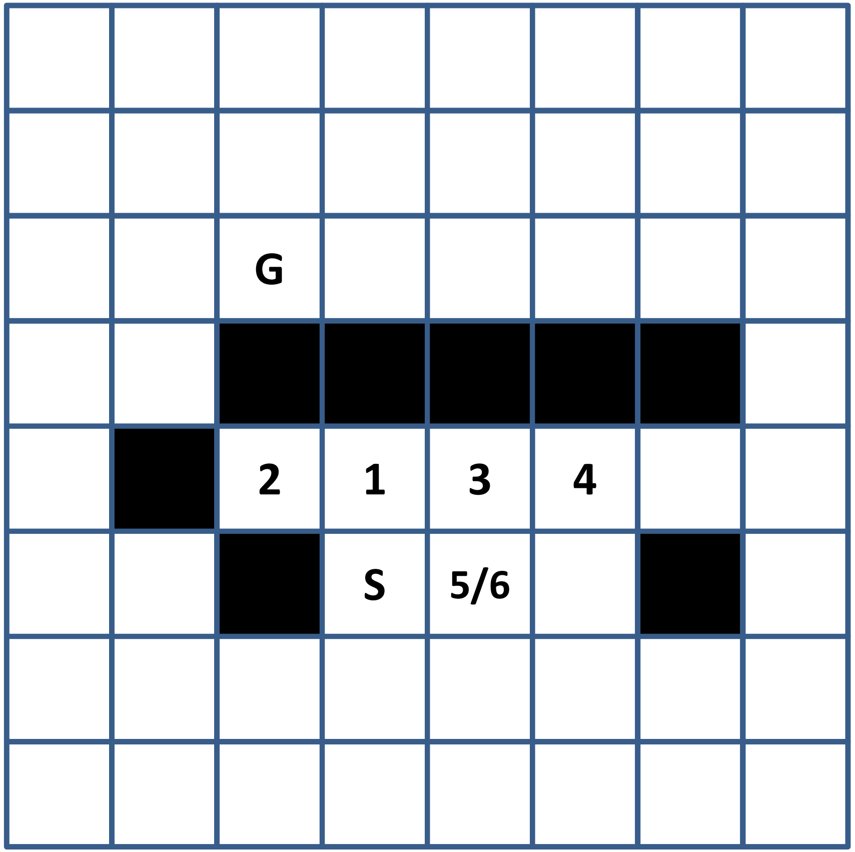


		G					
		■	■	■	■	■	
	■	2	1	3	4		
		■	S	5		■	

Greedy Search

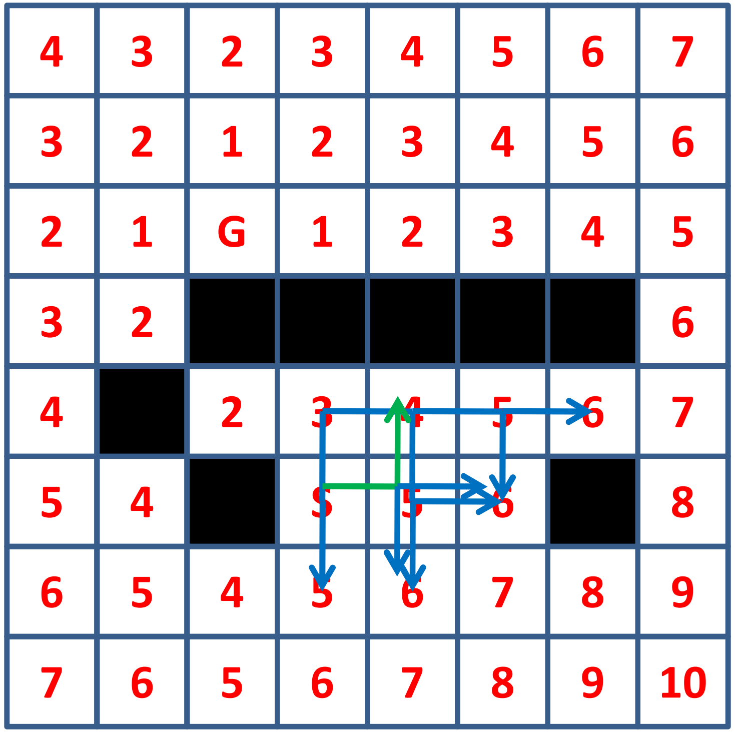


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

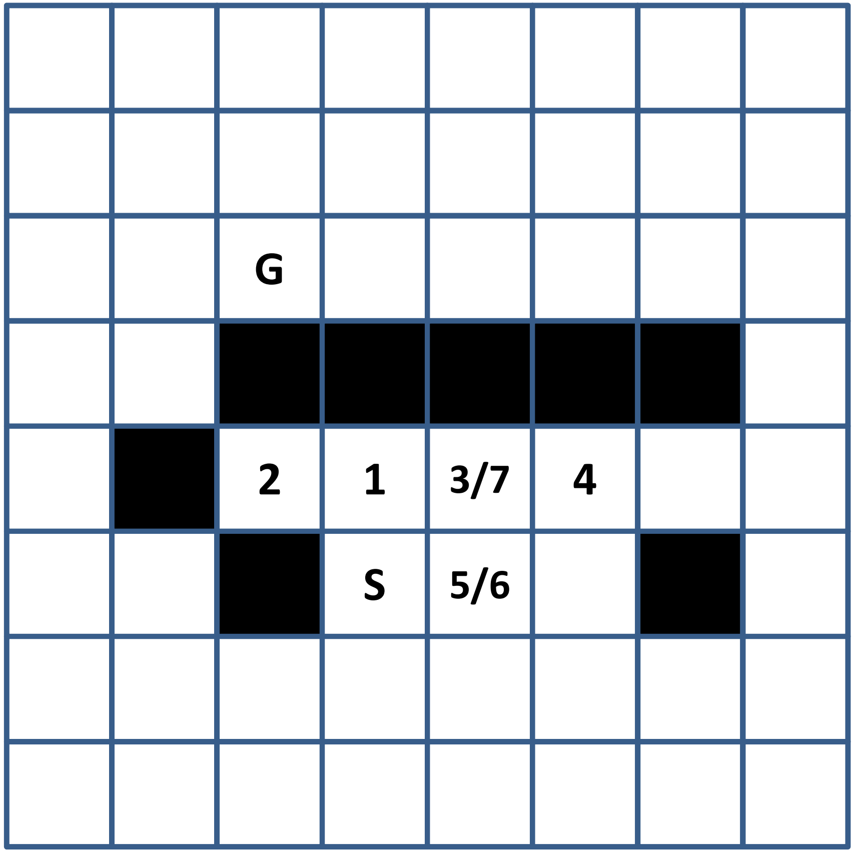


		G					
		■	■	■	■	■	
	■	2	1	3	4		
		■	S	5/6		■	

Greedy Search

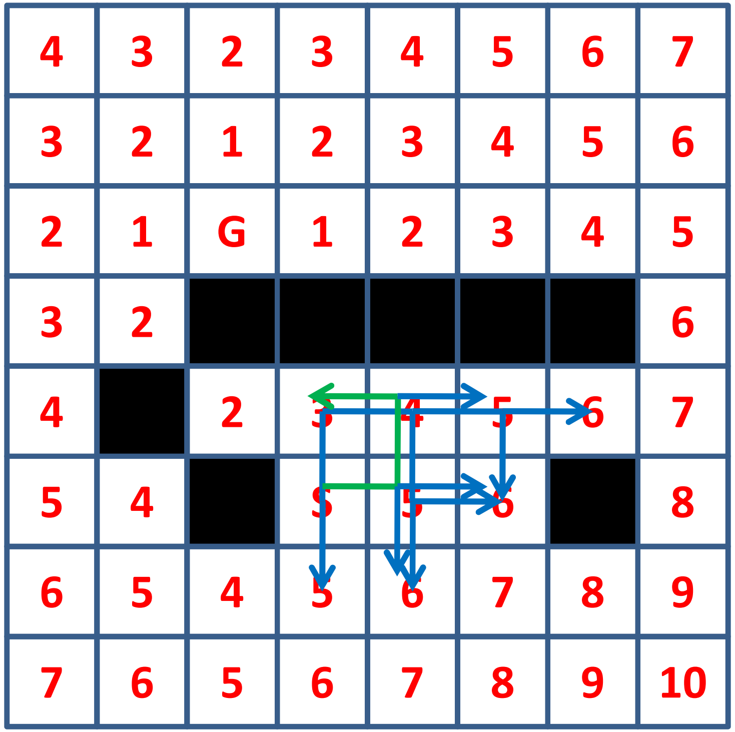


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10



		G					
		■	■	■	■	■	
	■	2	1	3/7	4		
		■	S	5/6		■	

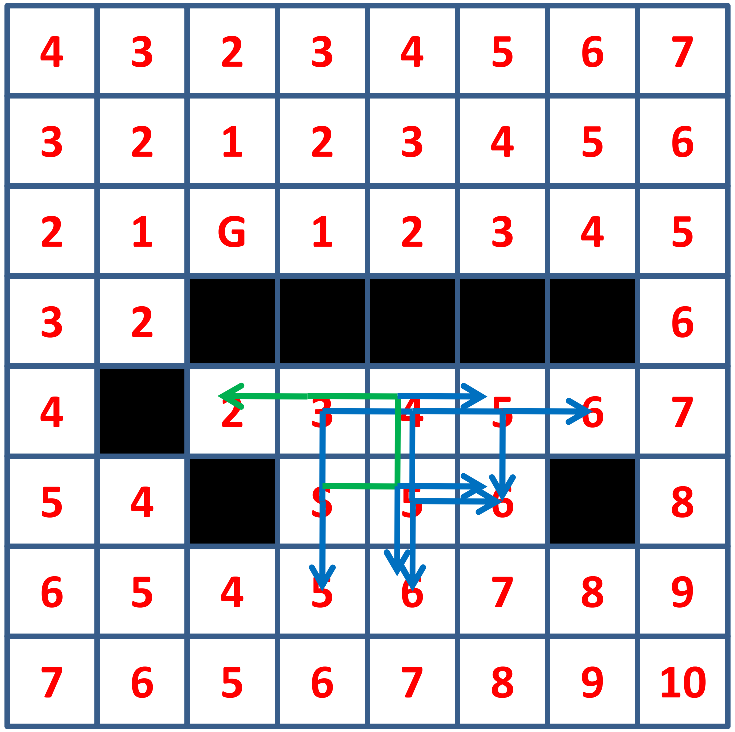
Greedy Search



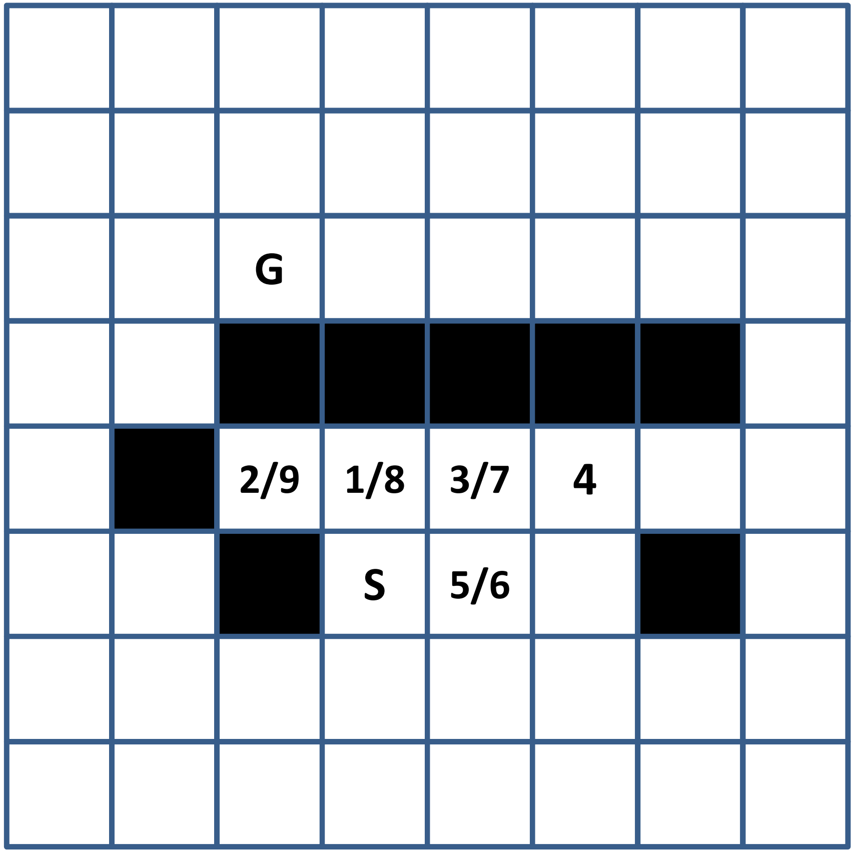
4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

		G					
		■	■	■	■	■	
	■	2	1/8	3/7	4		
		■	S	5/6		■	

Greedy Search

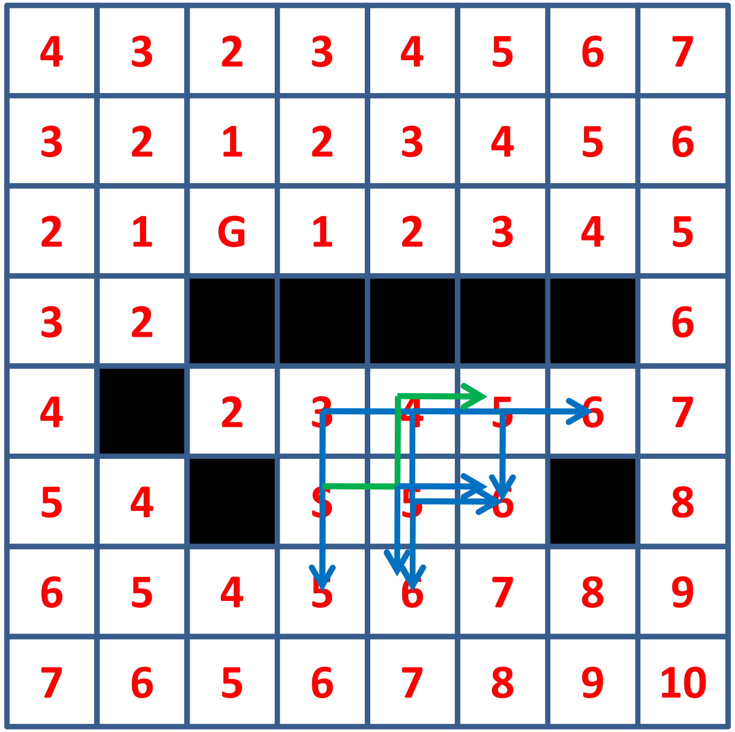


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

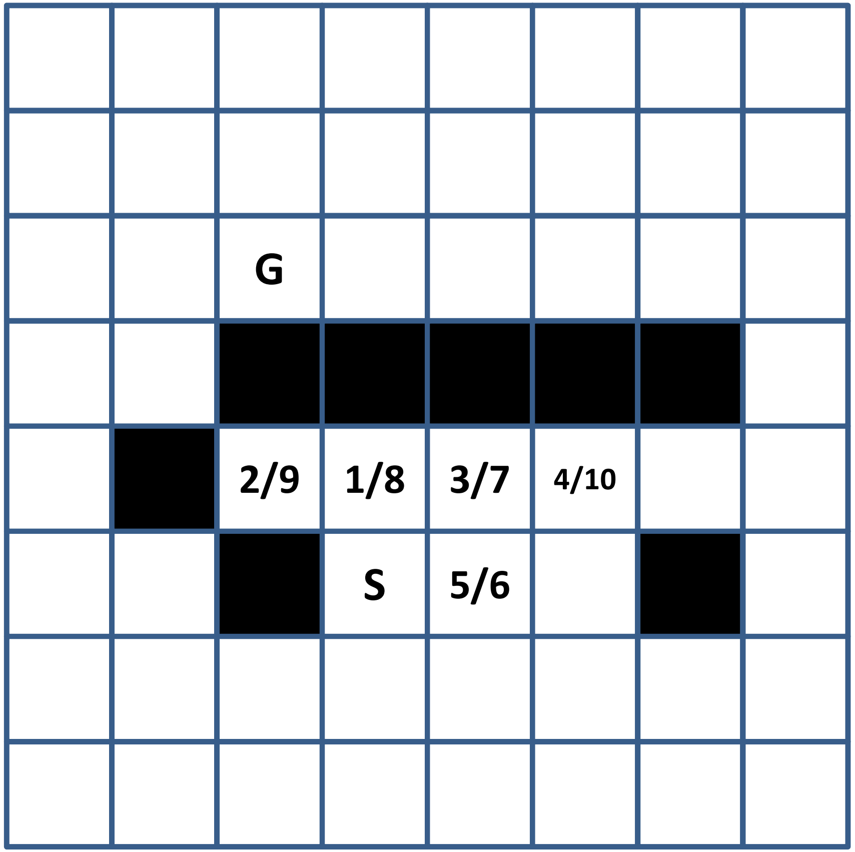


		G					
		■	■	■	■	■	
	■	2/9	1/8	3/7	4		
		■	S	5/6		■	

Greedy Search

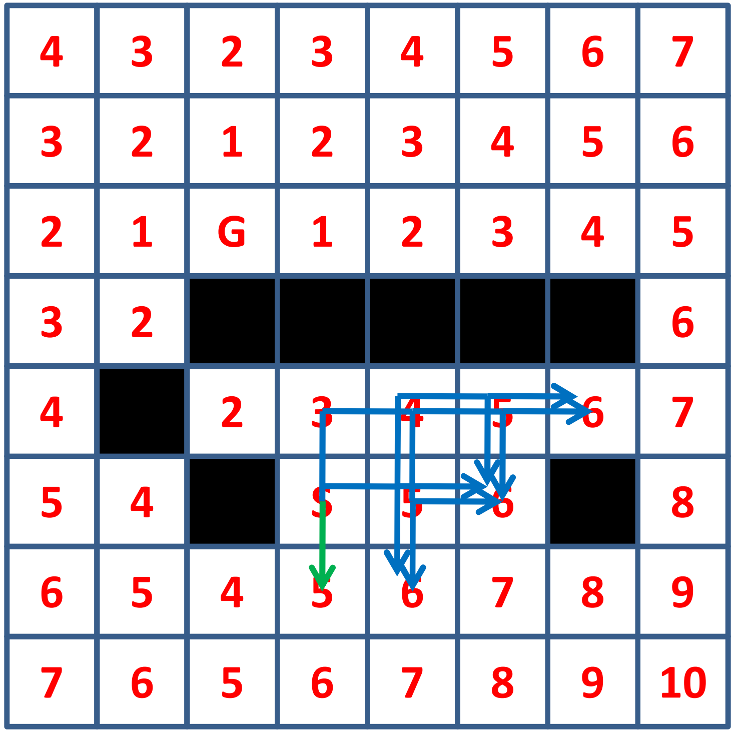


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

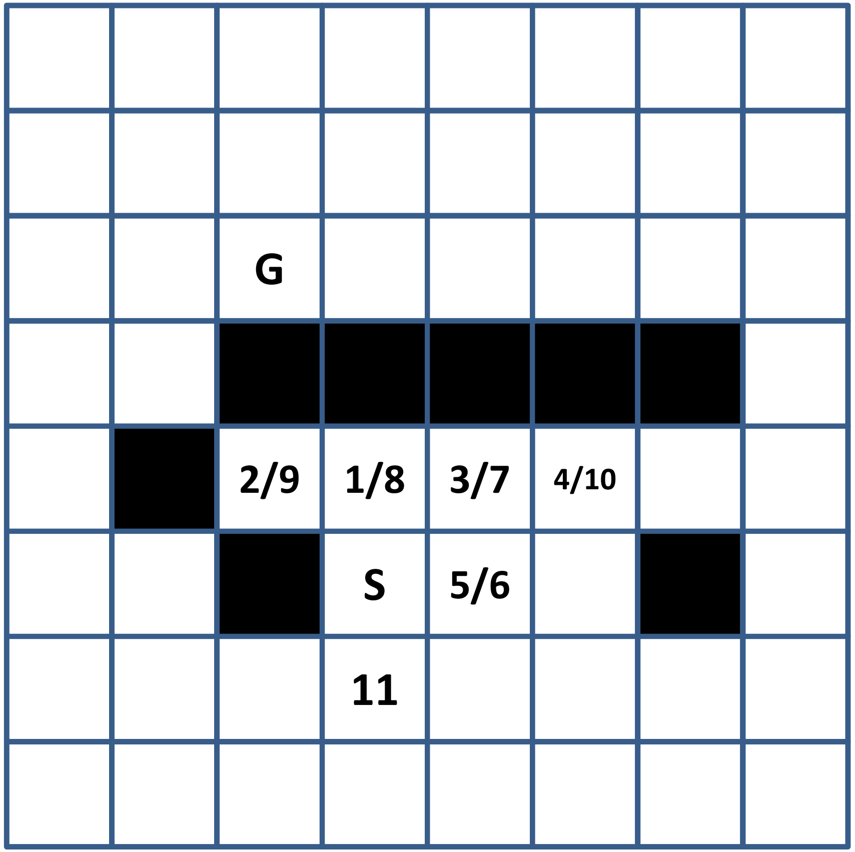


		G					
		■	■	■	■	■	
	■	2/9	1/8	3/7	4/10		
		■	S	5/6		■	

Greedy Search

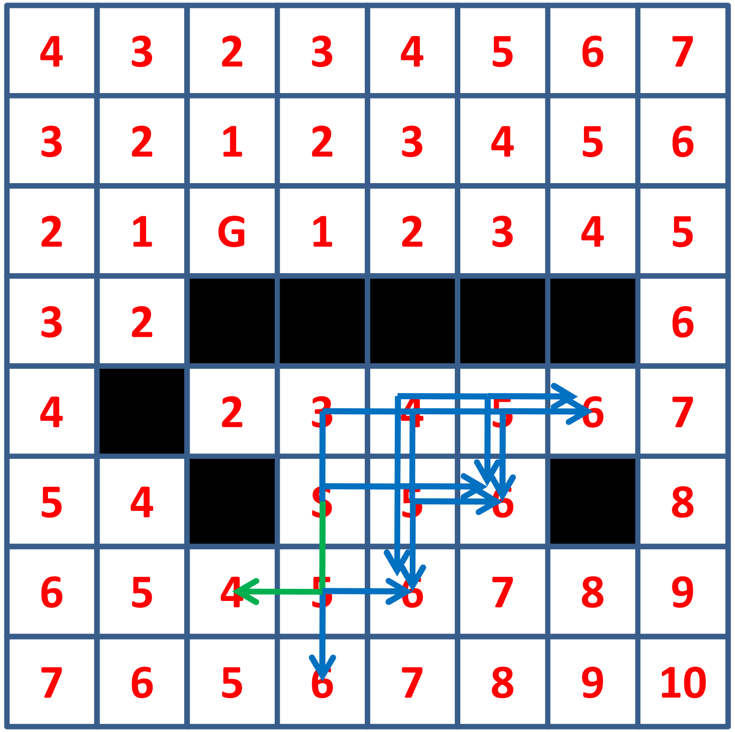


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

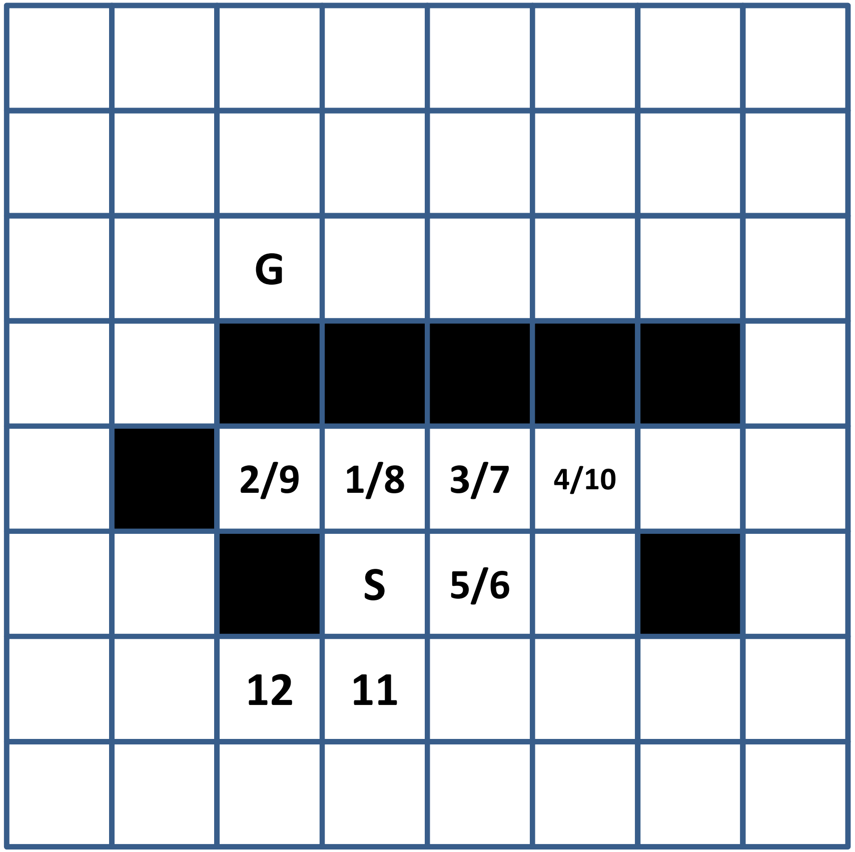


		G					
		■	■	■	■	■	
	■	2/9	1/8	3/7	4/10		
		■	S	5/6		■	
			11				

Greedy Search

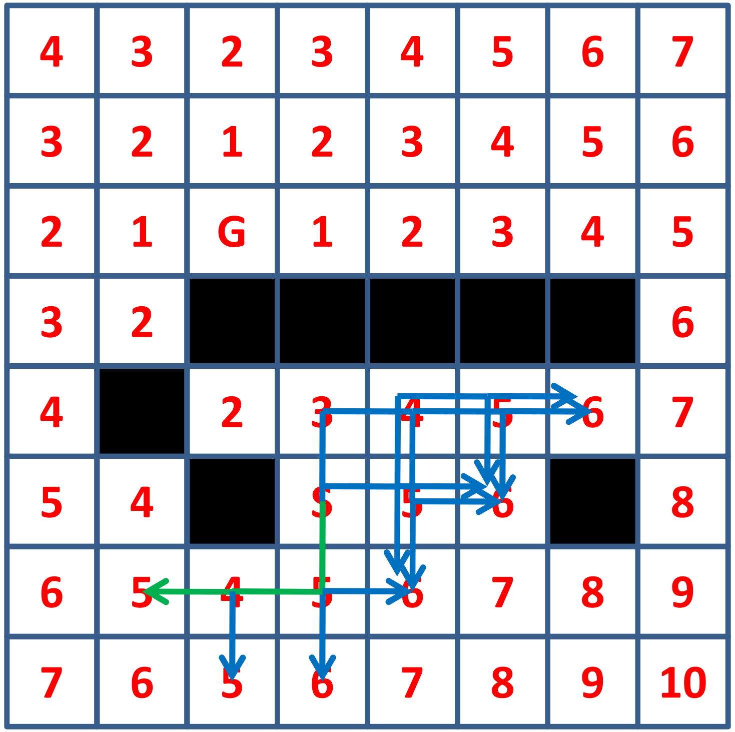


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

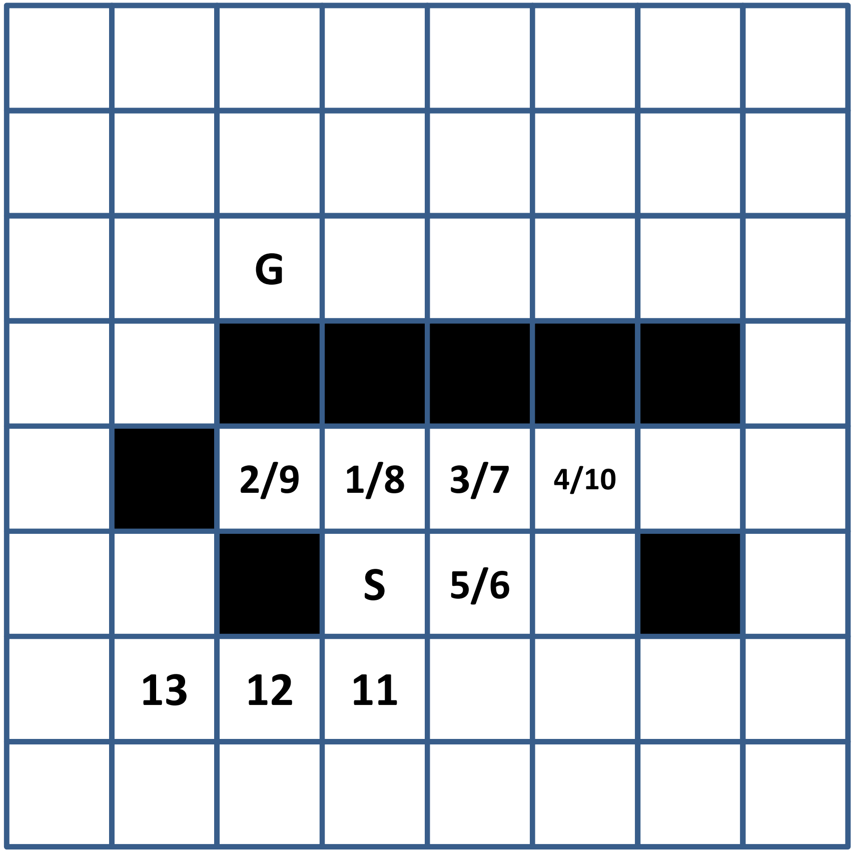


		G					
		■	■	■	■	■	
	■	2/9	1/8	3/7	4/10		
		■	S	5/6		■	
		12	11				

Greedy Search

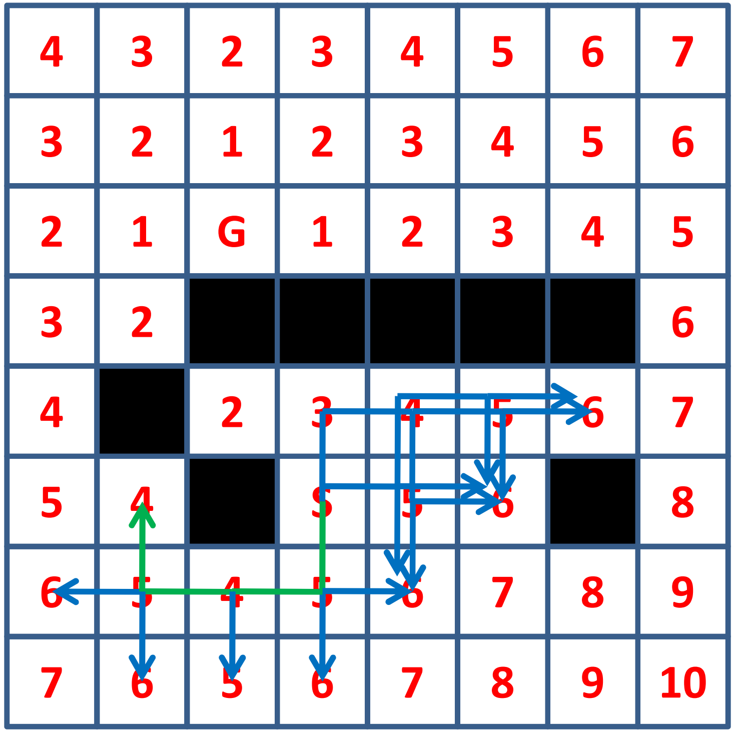


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10



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

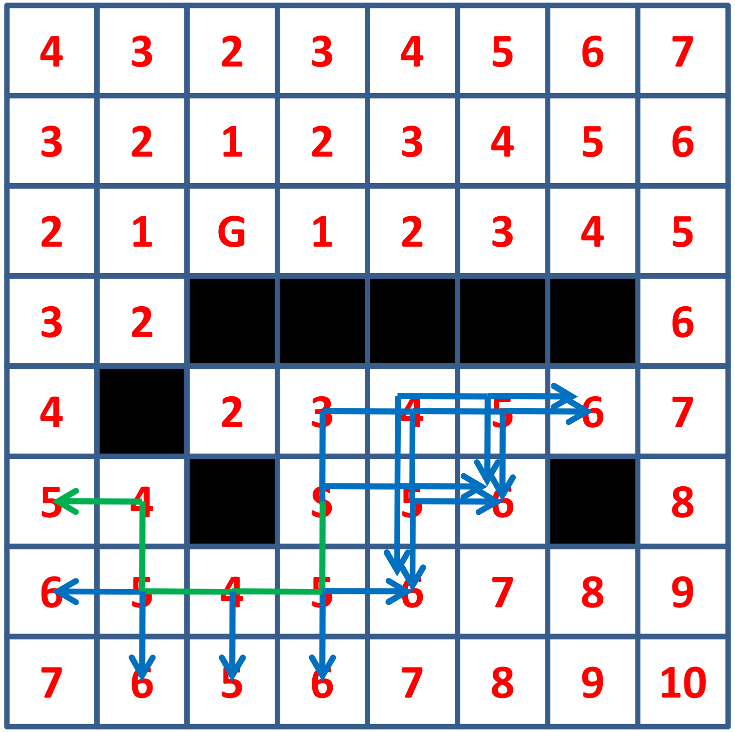
Greedy Search



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	■	S	5	6	■	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		
	14	■	S	5/6		■	
	13	12	11				

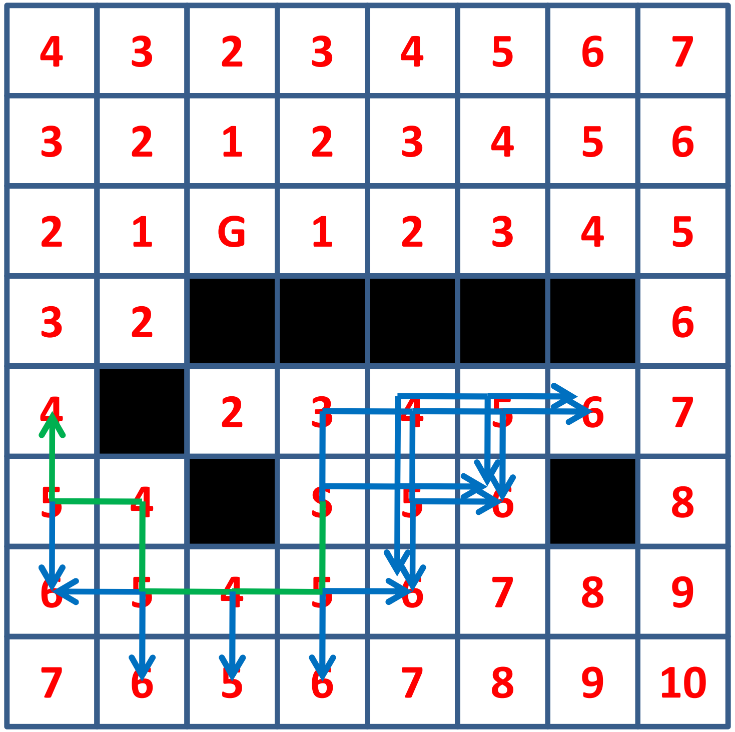
Greedy Search



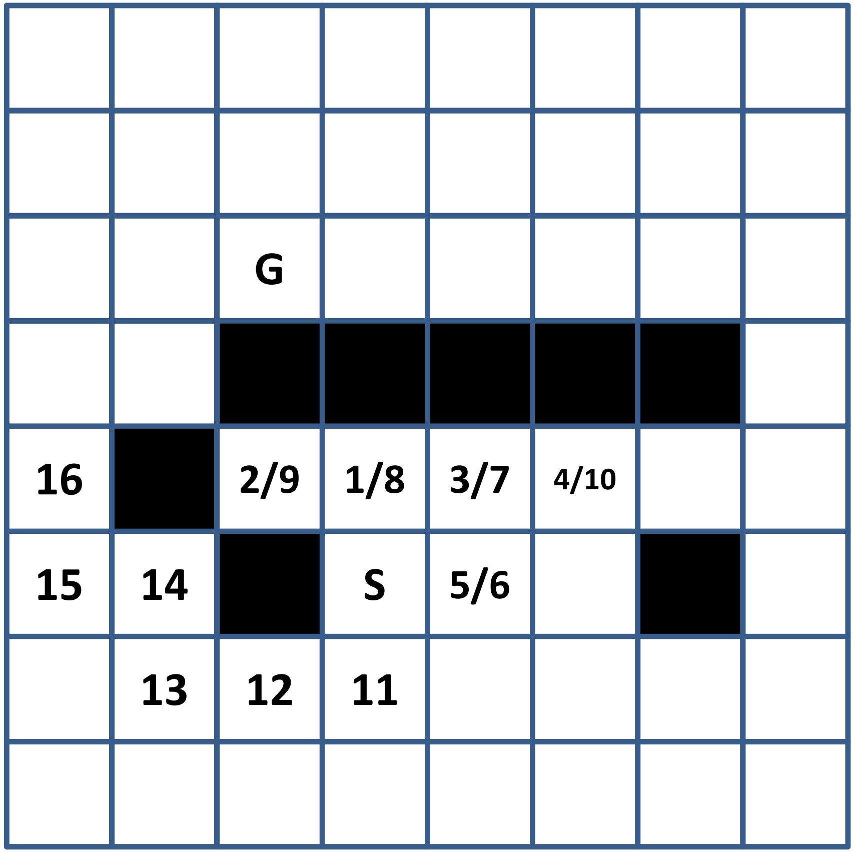
4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

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

Greedy Search

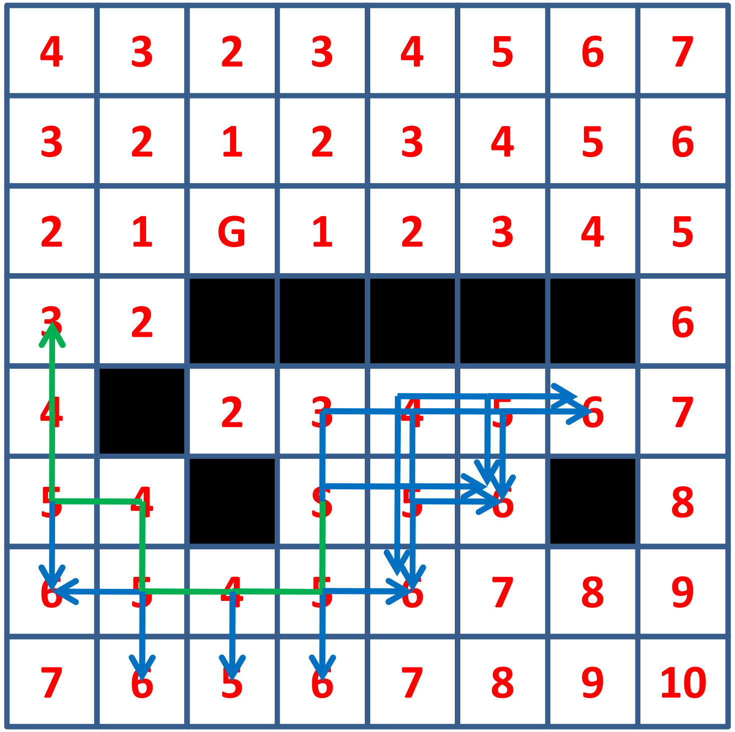


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

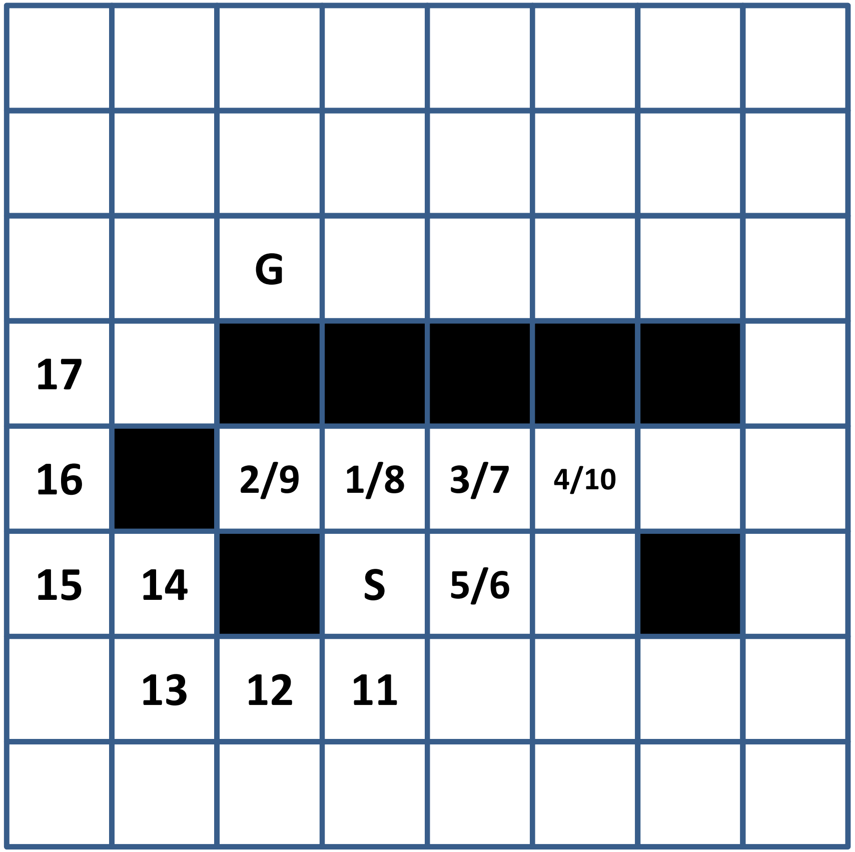


		G					
		■	■	■	■	■	
16	■	2/9	1/8	3/7	4/10		
15	14	■	S	5/6		■	
	13	12	11				

Greedy Search

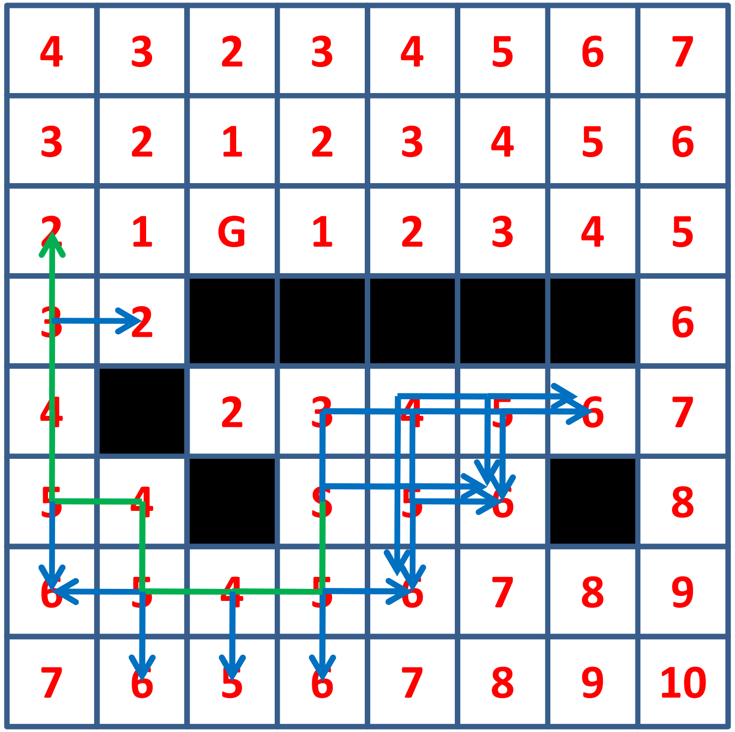


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

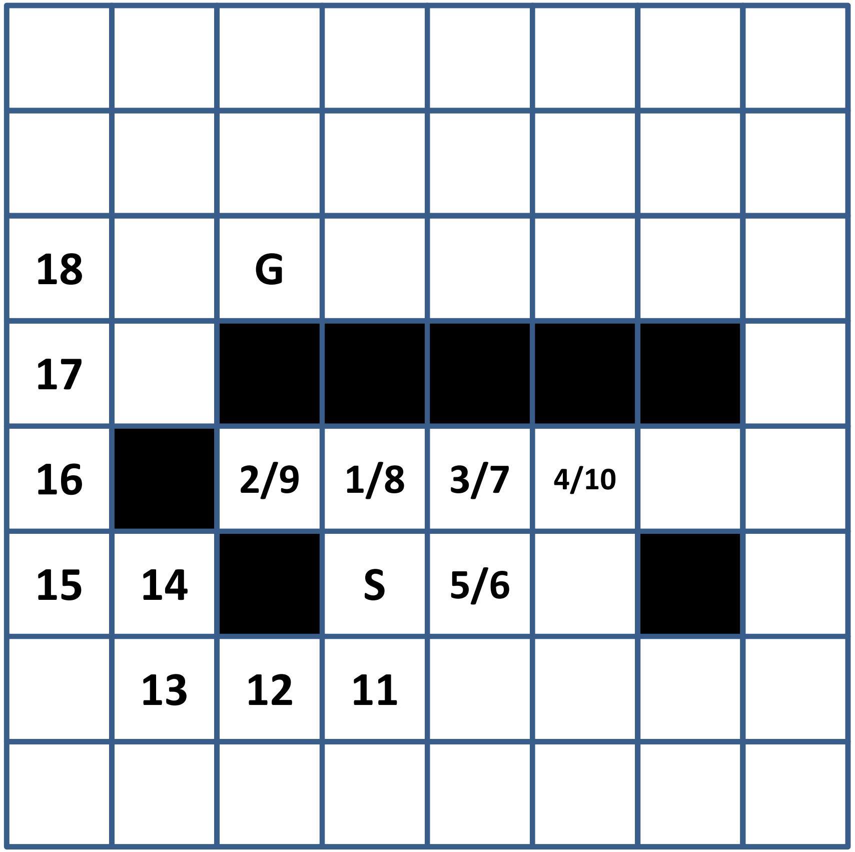


		G					
17		■	■	■	■	■	
16	■	2/9	1/8	3/7	4/10		
15	14	■	S	5/6		■	
	13	12	11				

Greedy Search

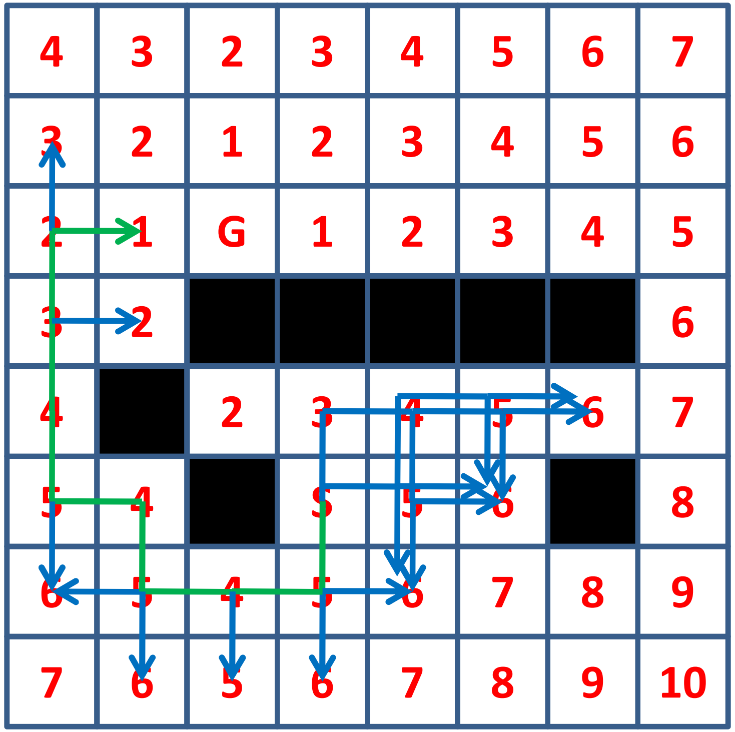


4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10



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

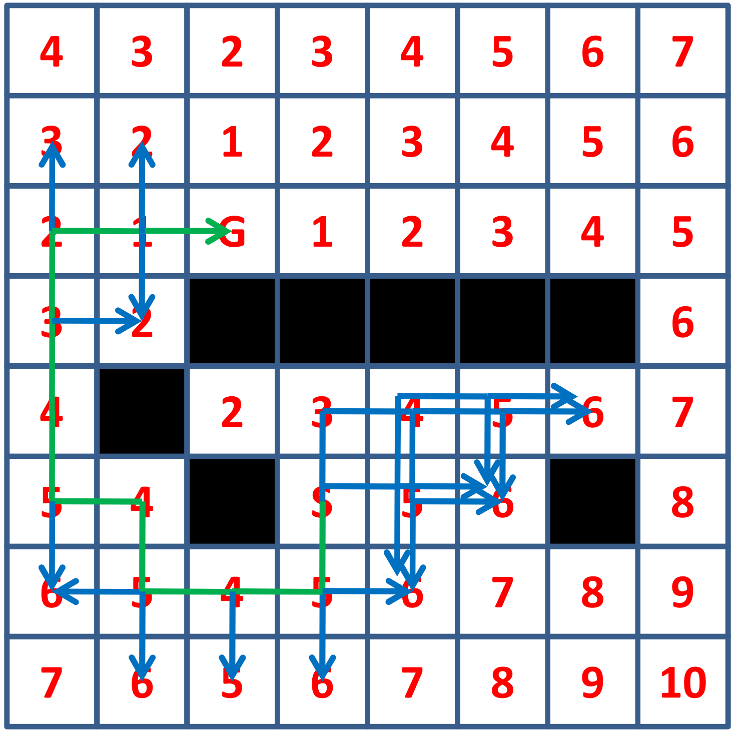
Greedy Search



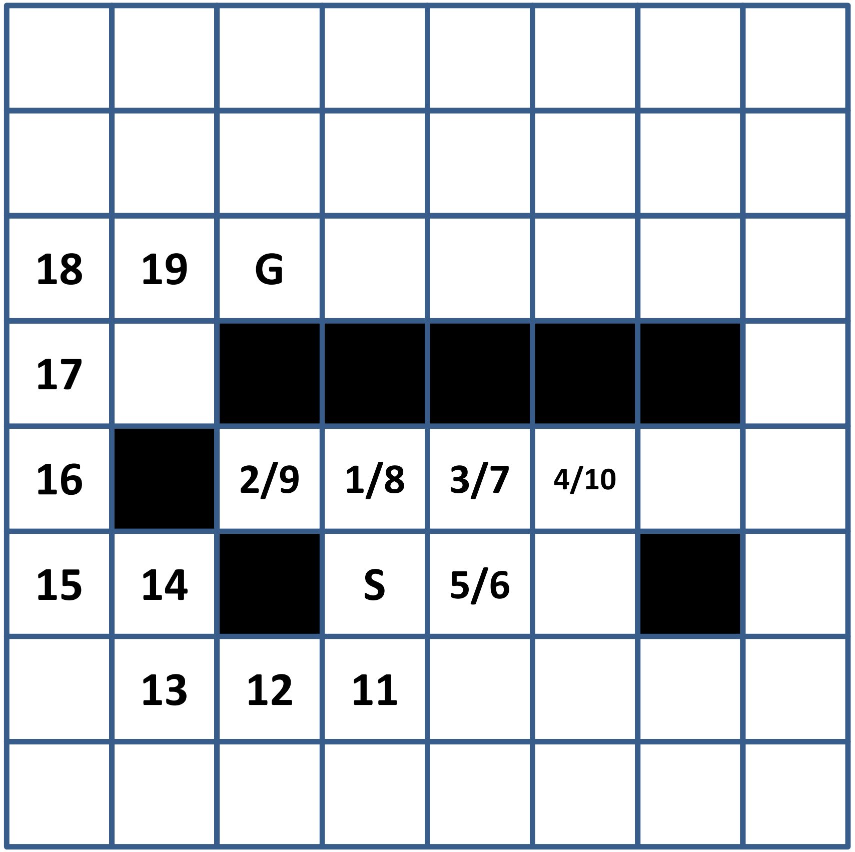
4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
2	1	G	1	2	3	4	5
3	2	■	■	■	■	■	6
4	■	2	3	4	5	6	7
5	4	■	S	5	6	■	8
6	5	4	5	6	7	8	9
7	6	5	6	7	8	9	10

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	■	S	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:

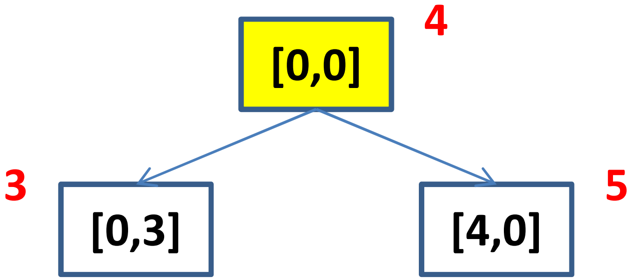
x	**0**	**1**	**2**	**3**	**4**
f(x)	2	1	0	1	3

  - 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.

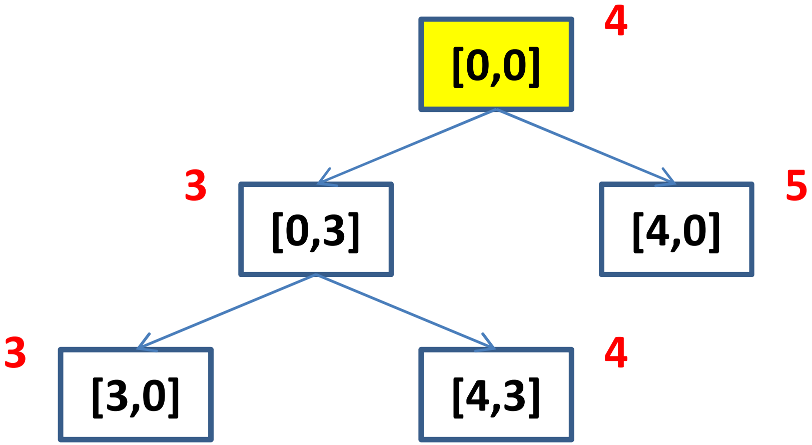
Water jugs

HILL-CLIMBING II SEARCH

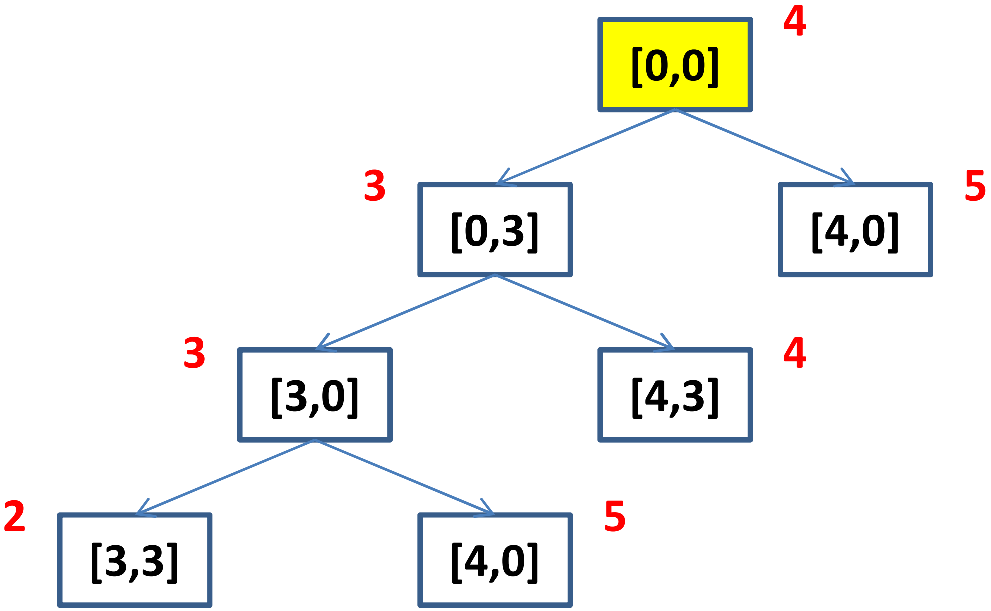
Hill-climbing II Search



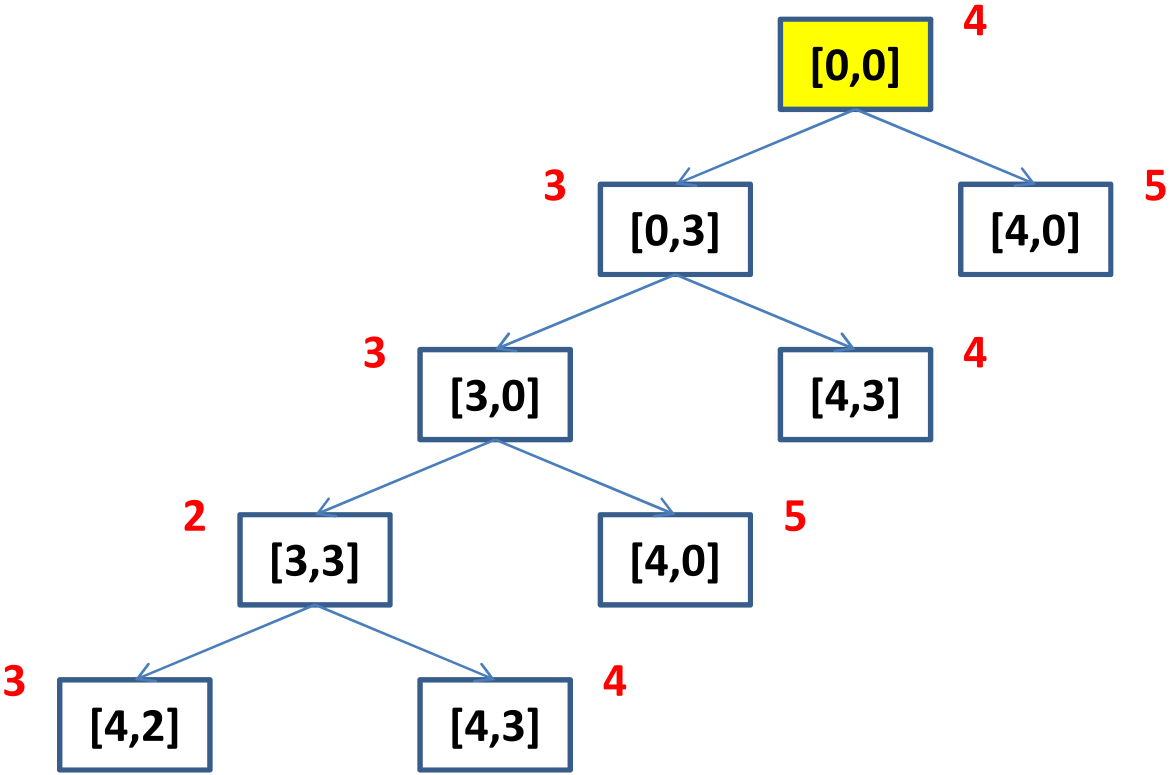
Hill-climbing II Search



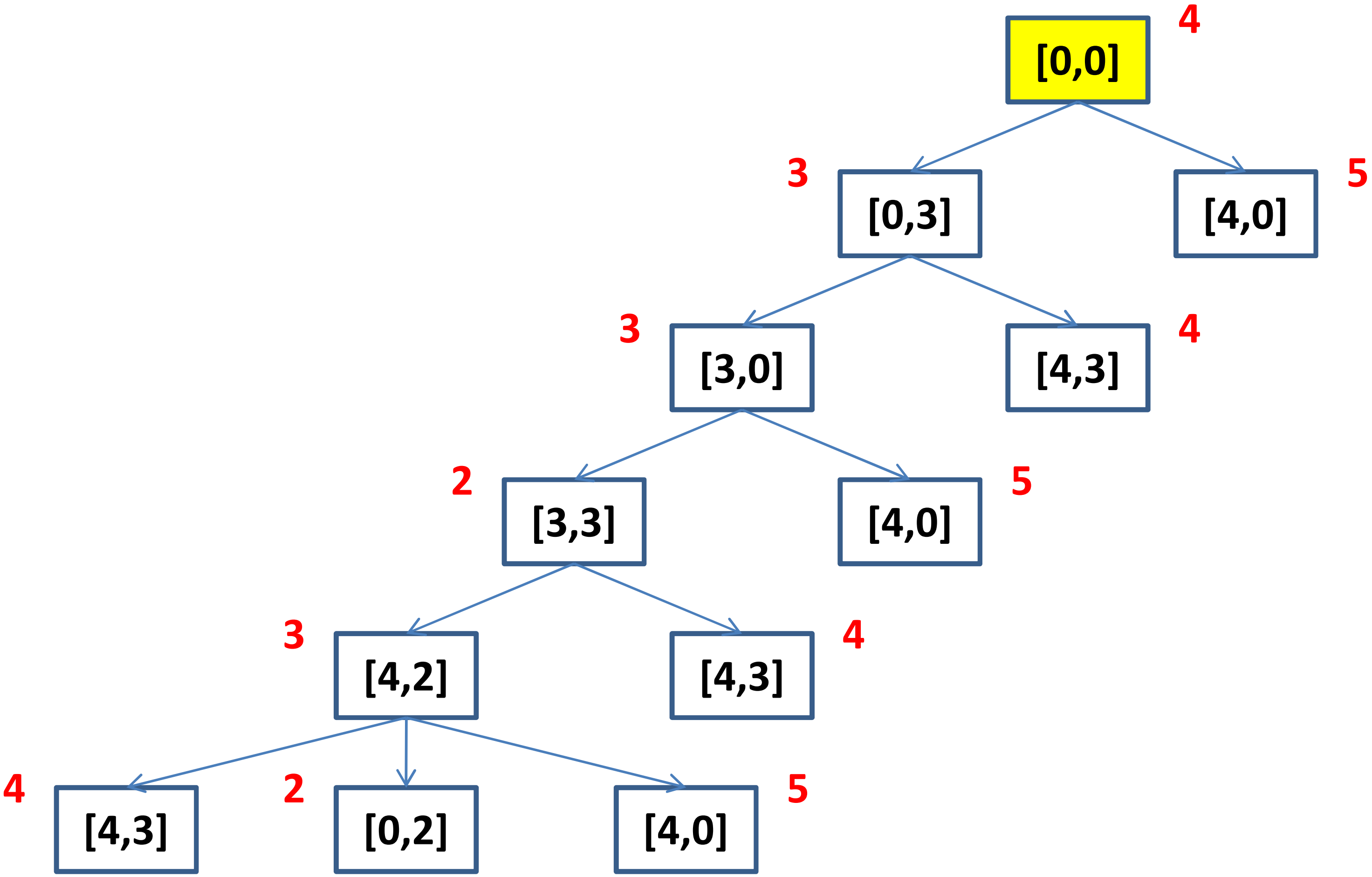
Hill-climbing II Search



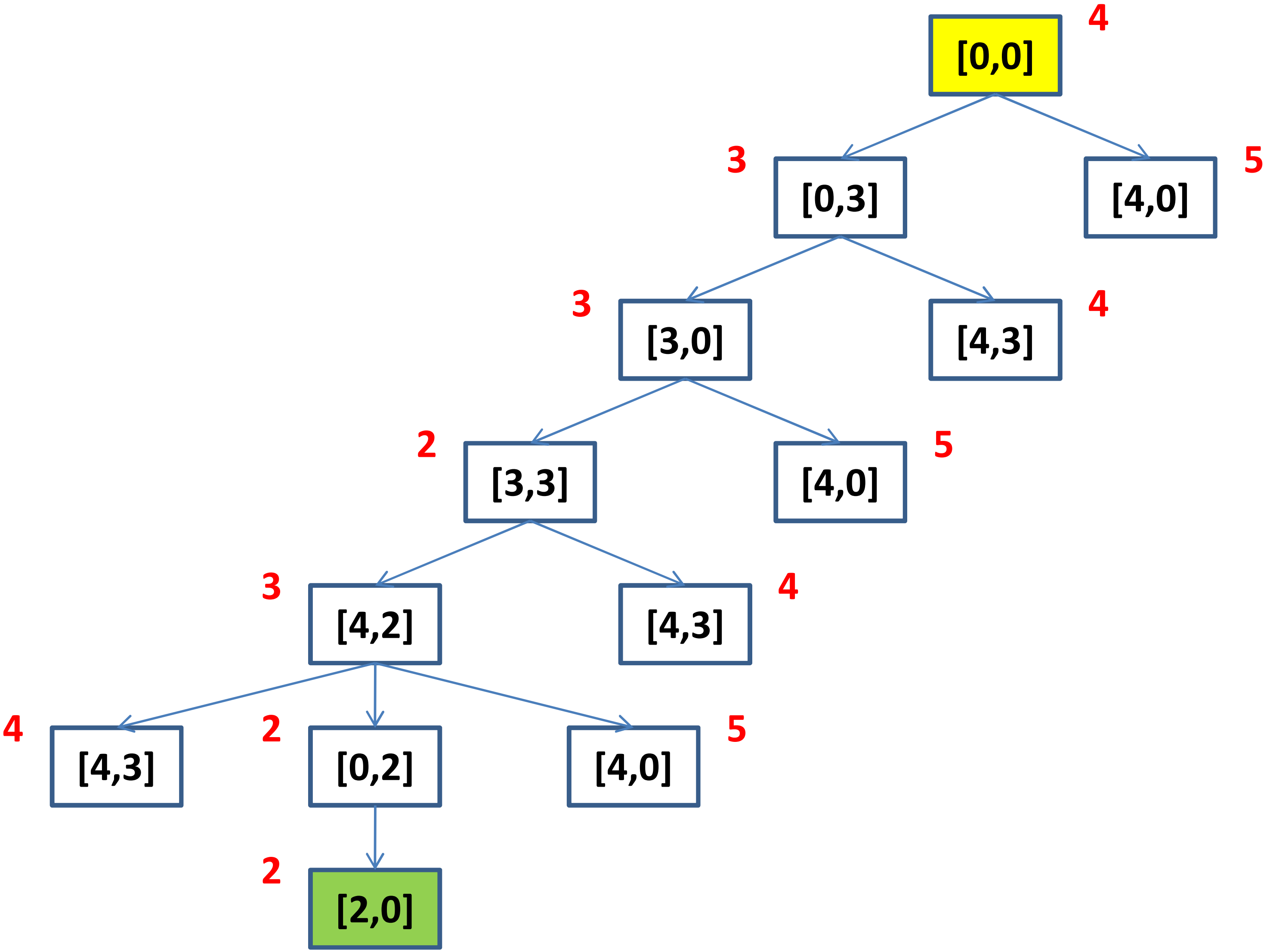
Hill-climbing II Search



Hill-climbing II Search



Hill-climbing II Search



[0,0]
4
3
[0,3]
[4,0]
5
3
[3,0]
[4,3]
4
2
[3,3]
[4,0]
5
3
[4,2]
[4,3]
4
4
[4,3]
2
[0,2]
[4,0]
5
2
[2,0]