BLG435E
Artificial Intelligence

Practice Session 1: Problem-Solving and Search

Contents

- Water Jugs Problem
  - Formulizing the problem in a well-defined form
  - Solving the problem using DFS and BFS (uninformed search)
- Path Planning in a Maze
  - Formulizing the problem in a well-defined form
  - Solving the problem using Greedy Search and A* (informed search)
- A* Search Example
  - Necessity of using an admissible heuristic

Water Jugs Problem

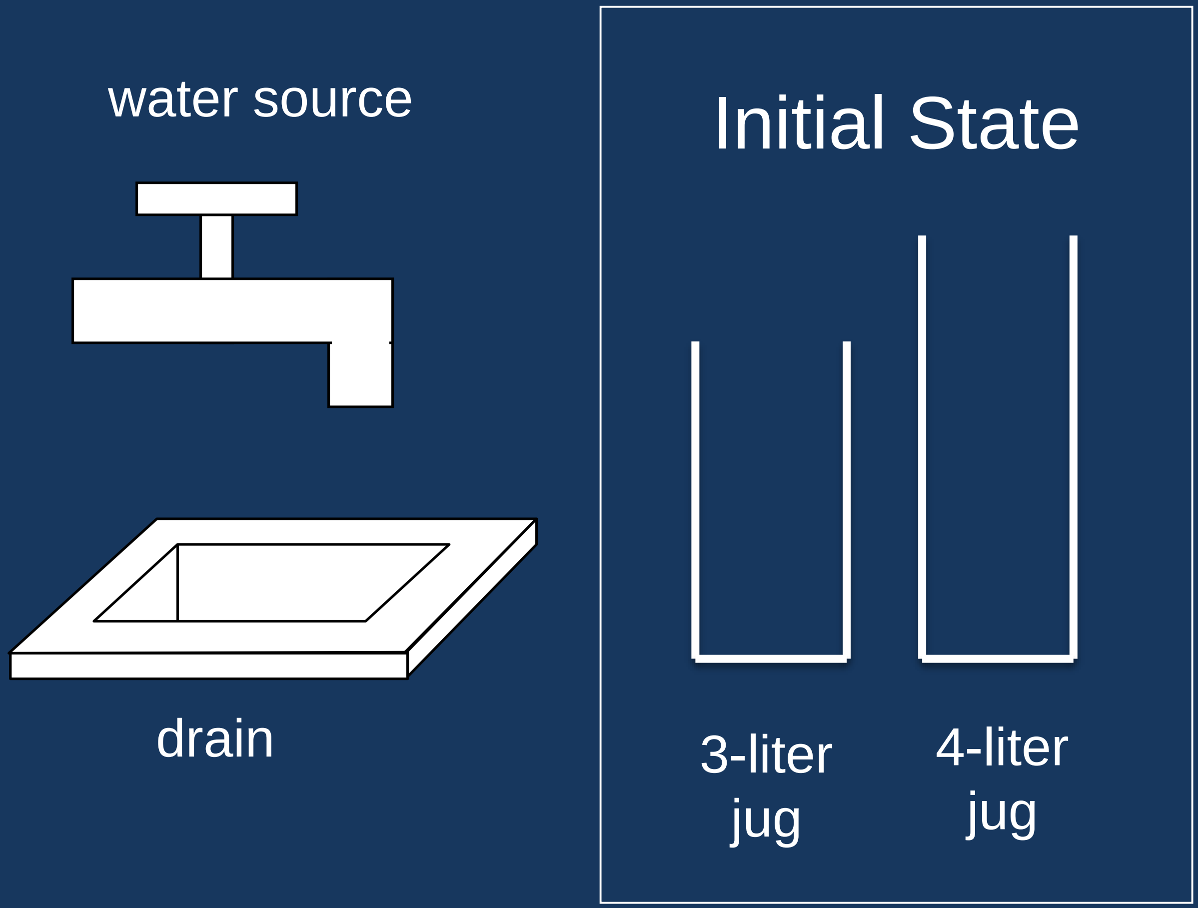
- You are given two jugs, a 3-liter jug, and a 4-liter jug.
- Initially both jugs are empty.
- Neither jugs has any measuring markers on it.

- There is a water source and a drain.

- Either jug can be filled from water source, emptied to drain, or poured into the other. (Filling and emptying operations must be in complete.)

- You are allowed to drain (i.e. waste) water if necessary.

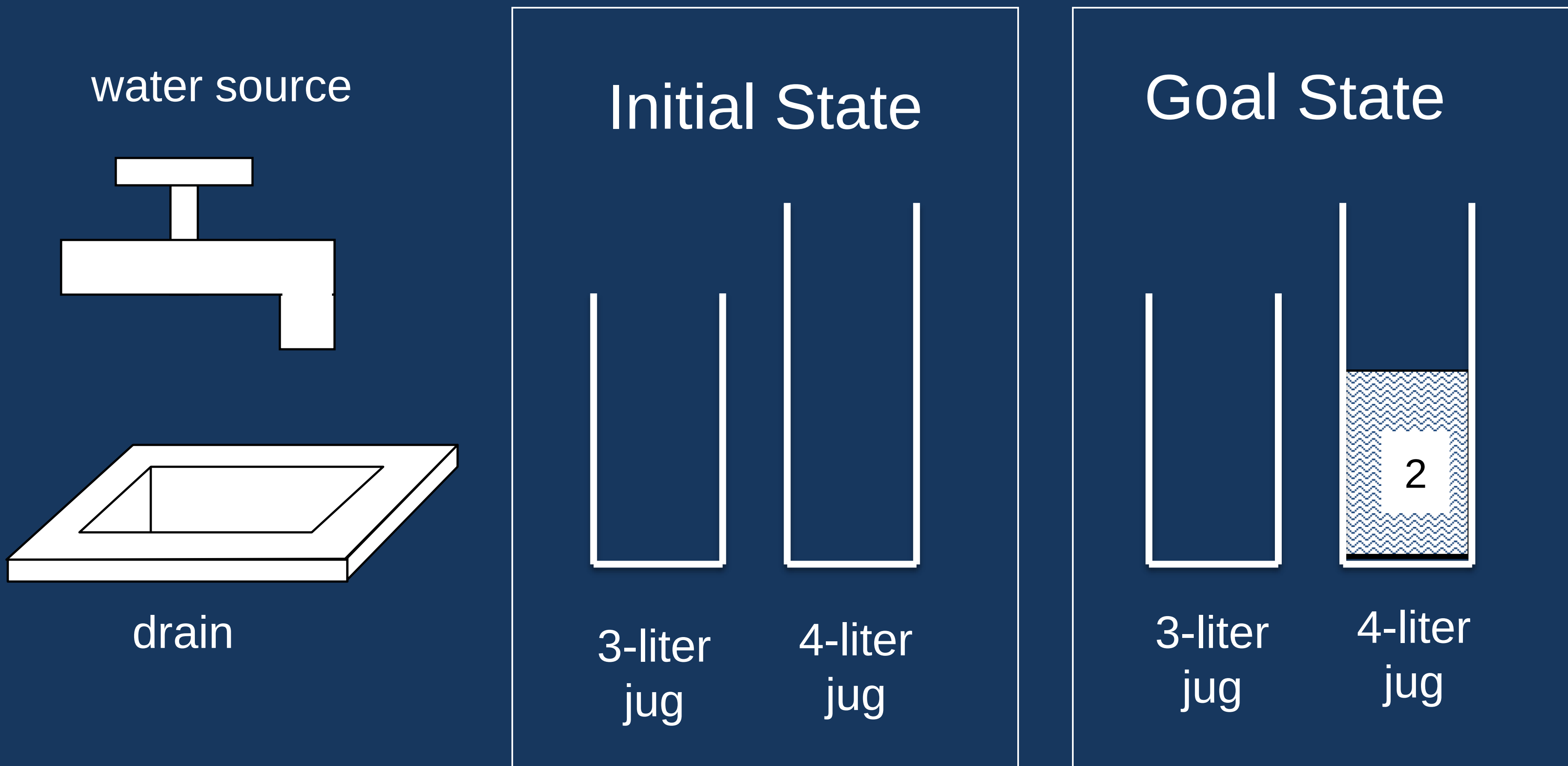
Water Jugs Problem



Water Jugs Problem

- **Problem:** How can you get exactly 2 liters of water into the 4-liter jug? Find the sequence of actions (i.e. solution path) from initial state to the goal state.

Water Jugs Problem



Water Jugs Problem

- **States**: Amounts of water in each jug. We can represent states with **(J_3 , J_4)** variable pairs.

  J_3 is for the 3-liter jug.

  J_4 is for the 4-liter jug.

- **Initial State**: $J_3 = 0$ and $J_4 = 0$
- **Goal Test**: $J_3 = 0$ and $J_4 = 2$
- **Actions**: 6 possible actions
- **Path cost**: uniform step costs for each action
- Size of the state space: 14 possible states

Actions in Water Jugs Problem

Action 1: Fill the 4-liter jug to capacity from a water source.

Action 2: Fill the 3-liter jug to capacity from a water source.

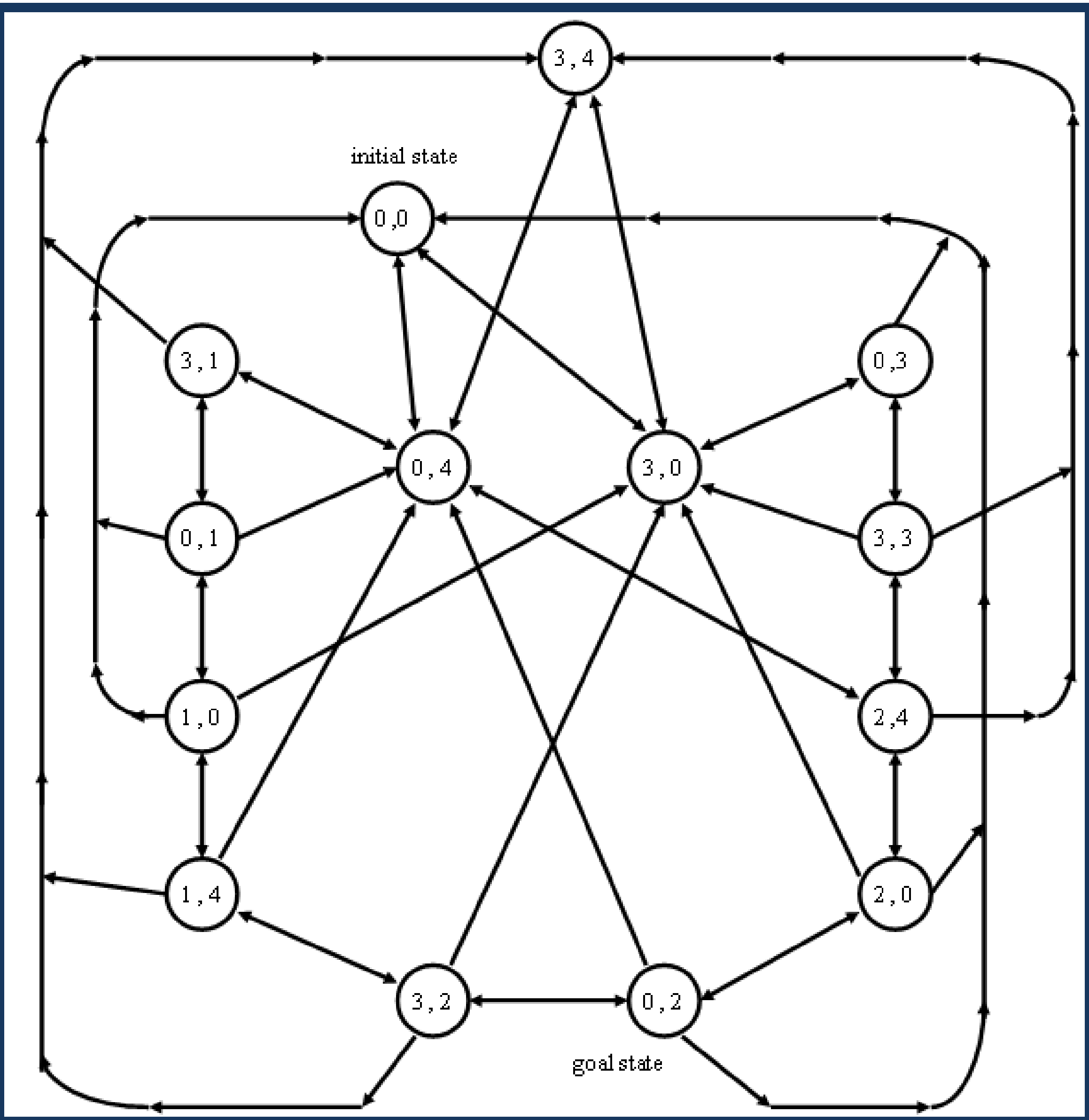
Action 3: Empty the 4-liter jug into the drain.

Action 4: Empty the 3-liter jug into the drain.

Action 5: Pour from the 3-liter jug into 4-liter jug until capacity reached.

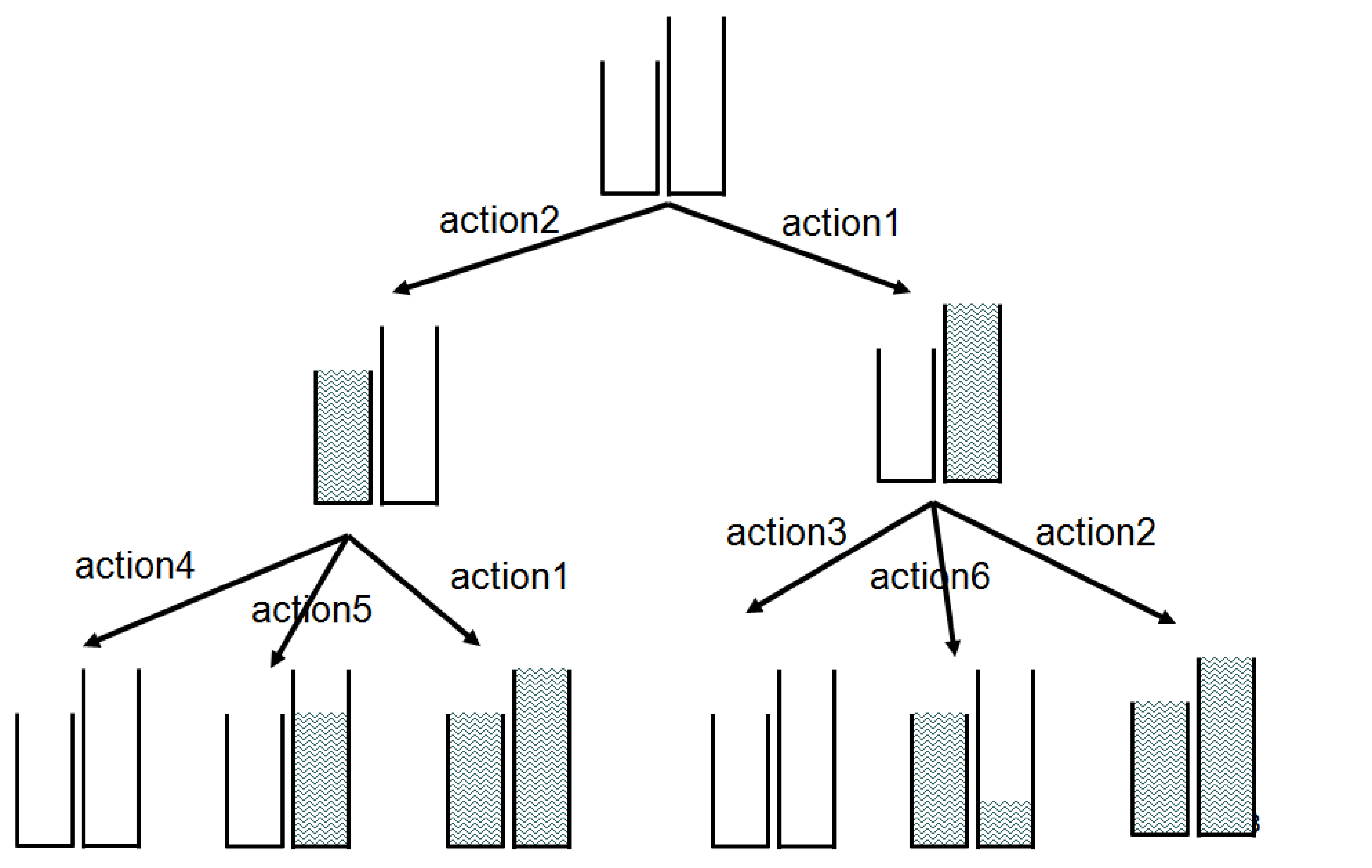
Action 6: Pour from the 4-liter jug into 3-liter jug until capacity reached.

Search Space-Water Jugs Problem



(J_3, J_4)
14 nodes
46 arcs

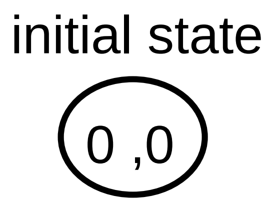
Search Tree–Water Jugs Problem



Water Jugs Problem Solution

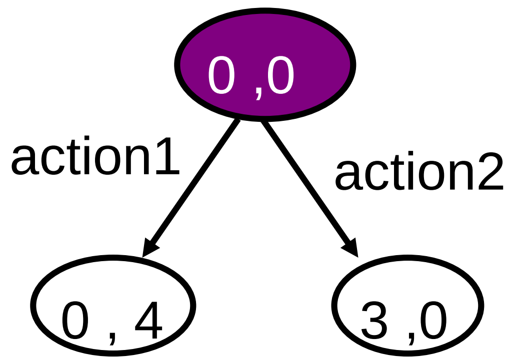
STEP	ACTION	3-liter Jug	4-liter Jug
0	Initial state: Both jugs are empty	0	0
1	Action2) Fill the 3-liter jug	3	0
2	Action5) Pour from the 3-liter jug into 4-liter jug	0	3
3	Action2) Fill the 3-liter jug	3	3
4	Action5) Pour from the 3-liter jug into 4-liter jug	2	4
5	Action3) Empty the 4-liter jug into a drain	2	0
6	Action5) Pour from the 3-liter jug into 4-liter jug	0	2

Water Jugs Problem DFS Solution



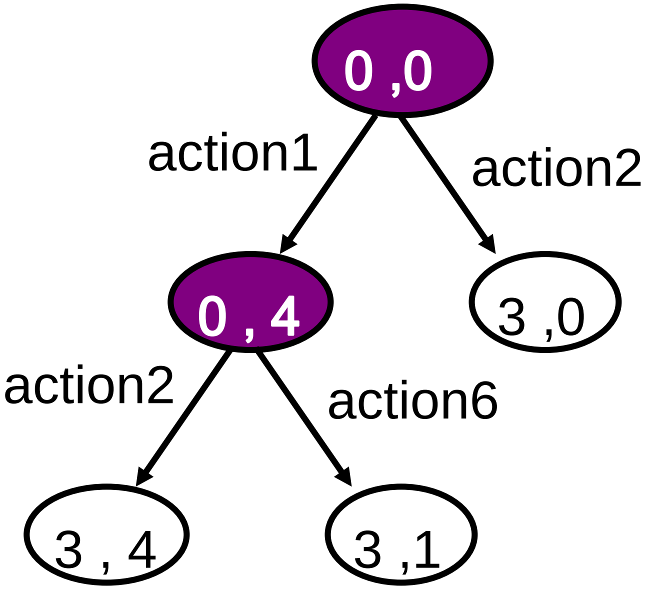
OPEN = (0,0)
CLOSED = empty

Water Jugs Problem DFS Solution



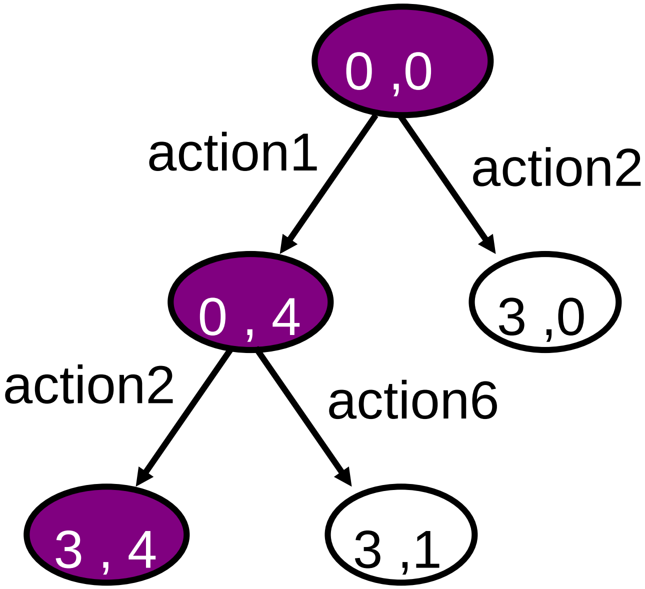
OPEN = (0,4) (3,0)
CLOSED = (0,0)

Water Jugs Problem DFS Solution



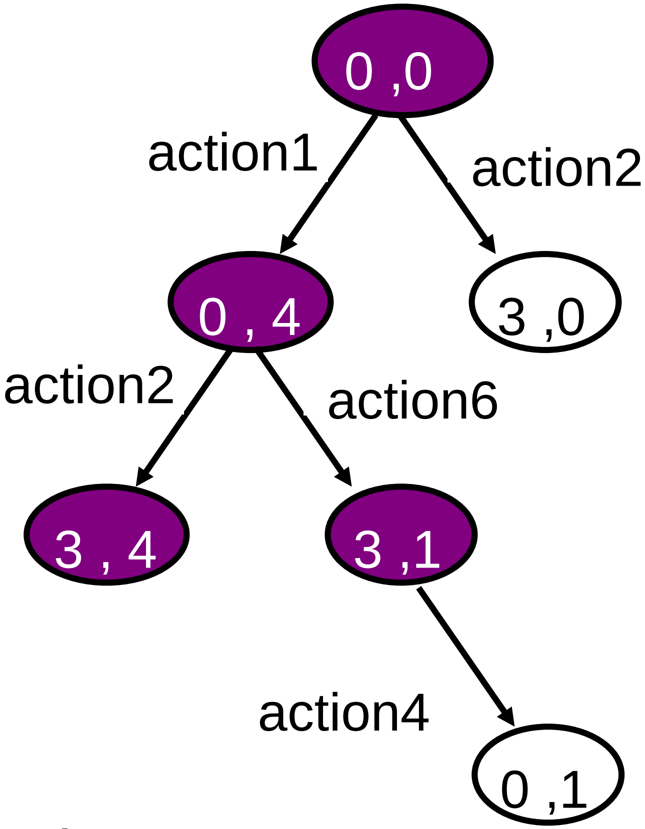
OPEN = (3,4) (3,1) (3,0)
CLOSED = (0,0) (0,4)

Water Jugs Problem DFS Solution



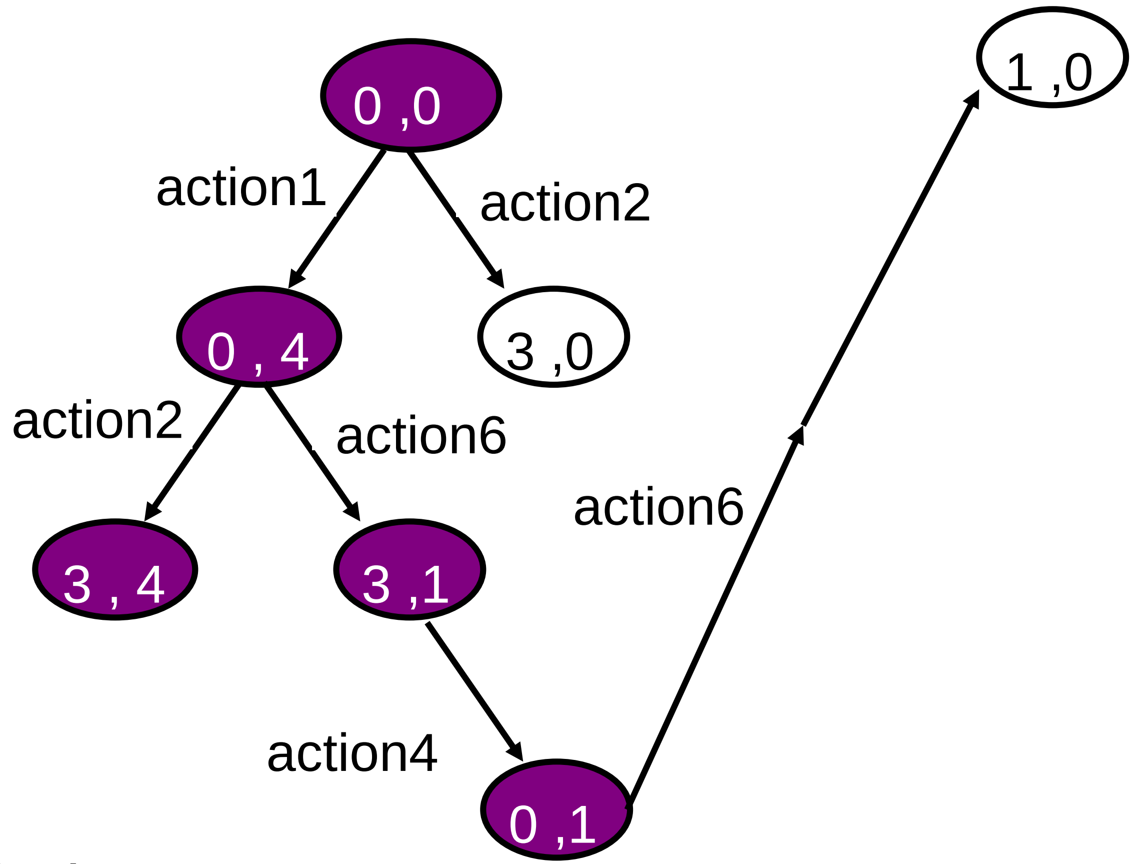
OPEN = (3,1) (3,0)
CLOSED = (0,0) (0,4) (3,4)

Water Jugs Problem DFS Solution



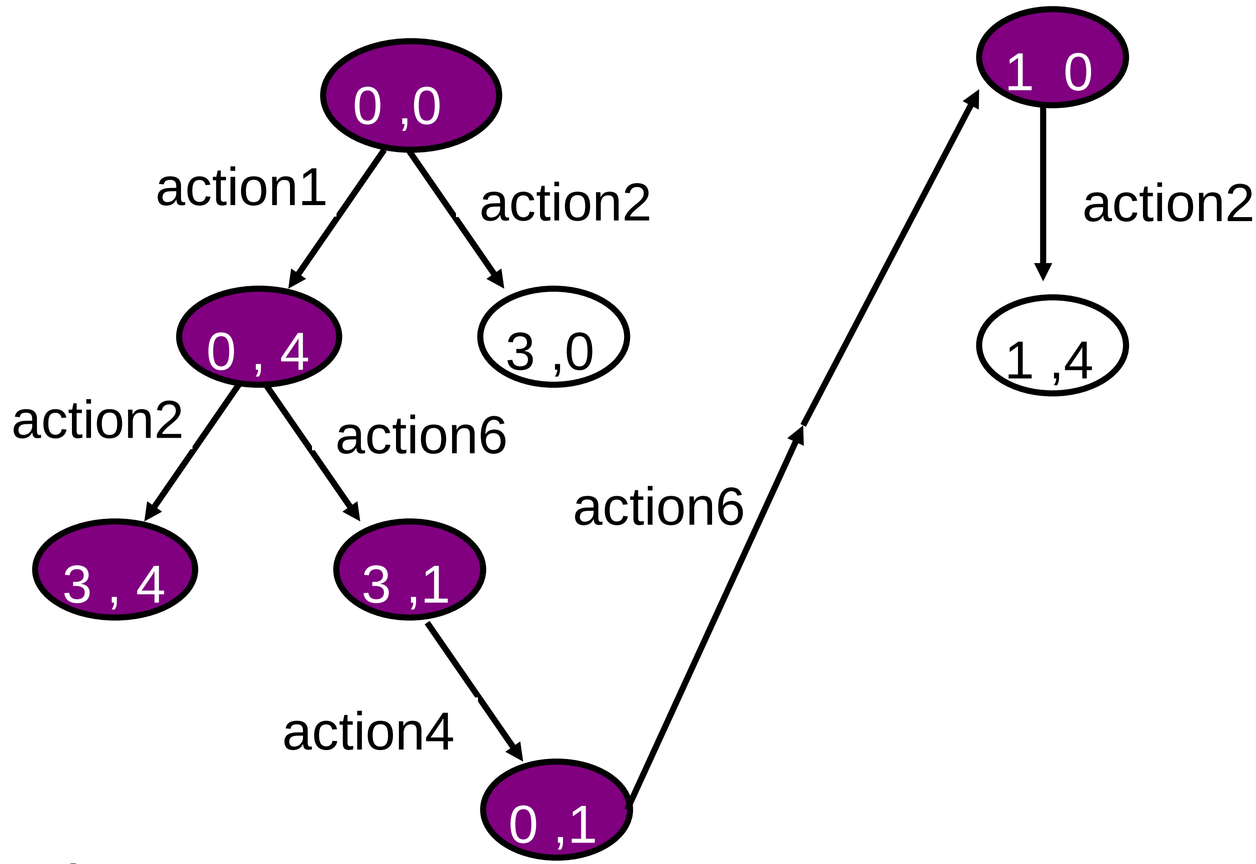
OPEN = (0,1) (3,0)
CLOSED = (0,0) (0,4) (3,4) (3,1)

Water Jugs Problem DFS Solution



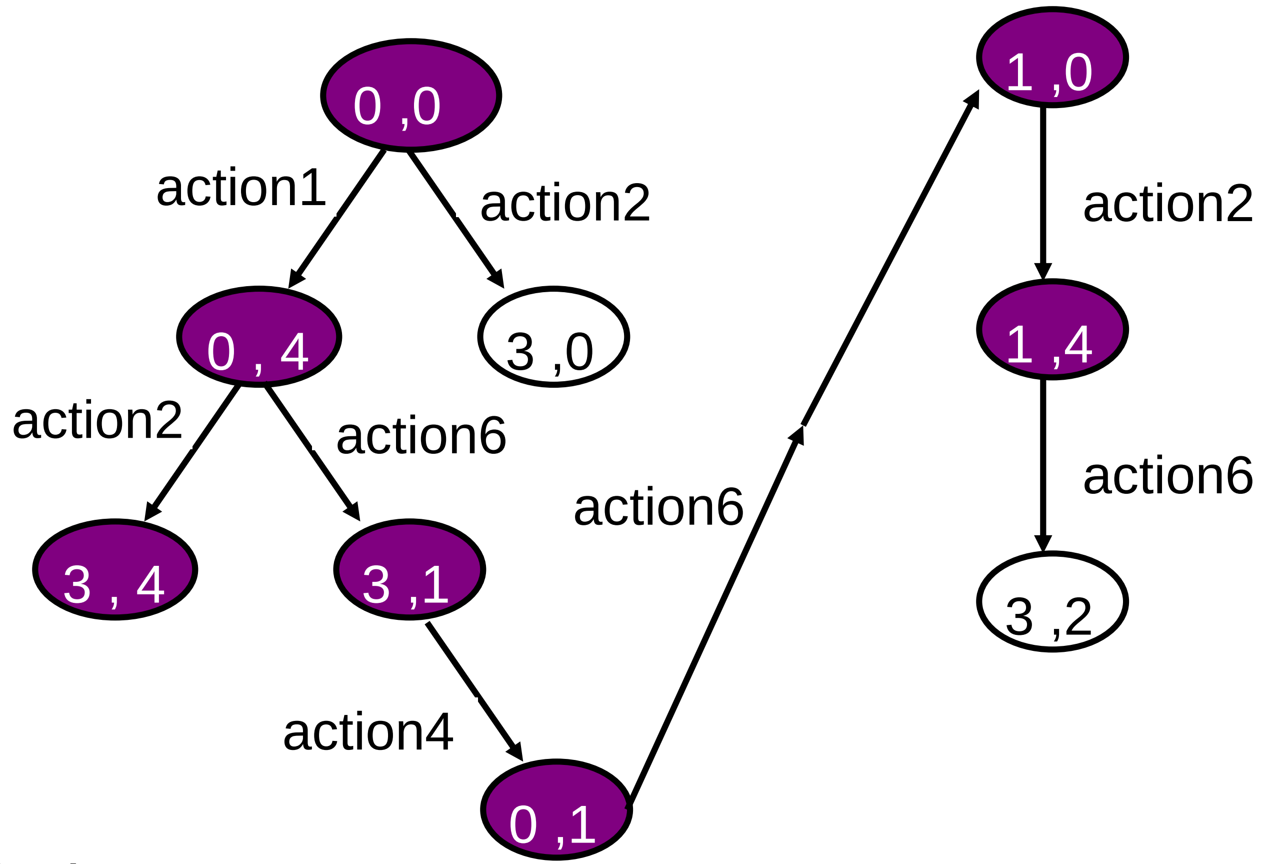
OPEN = (1,0) (3,0)
CLOSED = (0,0) (0,4) (3,4) (3,1) (0,1)

Water Jugs Problem DFS Solution



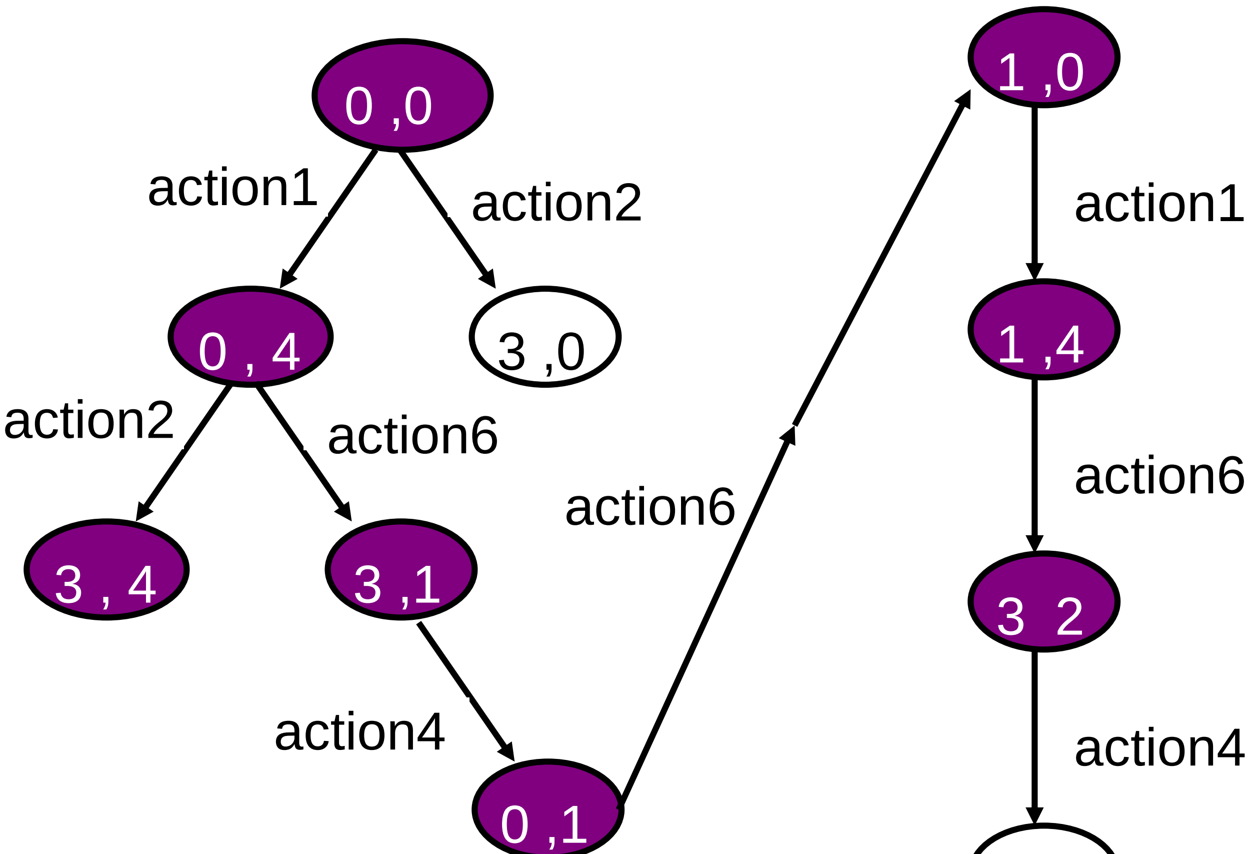
OPEN = (1,4) (3,0)
CLOSED = (0,0) (0,4) (3,4) (3,1) (0,1) (1,0)

Water Jugs Problem DFS Solution



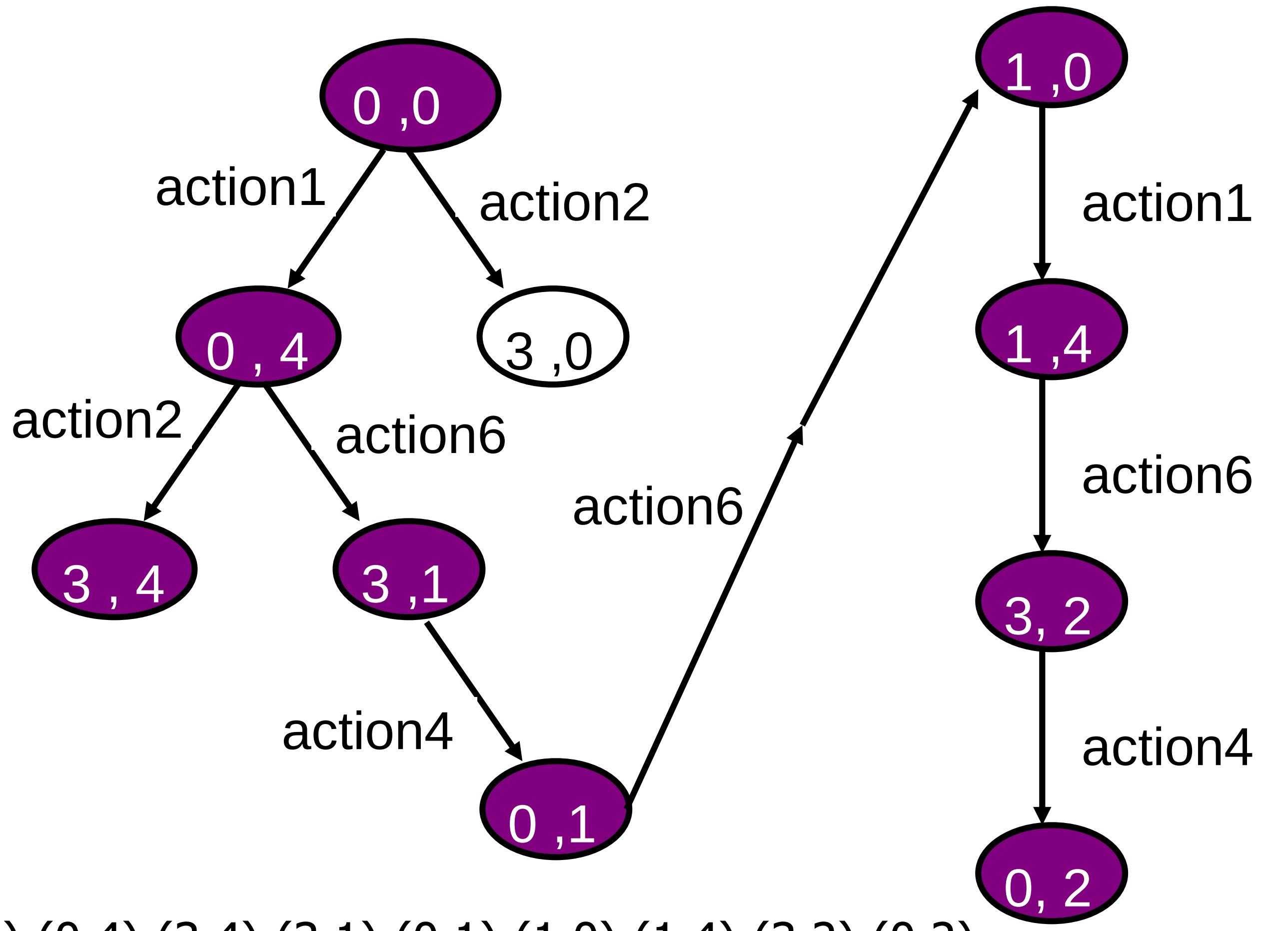
OPEN = (3,2) (3,0)
CLOSED = (0,0) (0,4) (3,4) (3,1) (0,1) (1,0) (1,4)

Water Jugs Problem DFS Solution



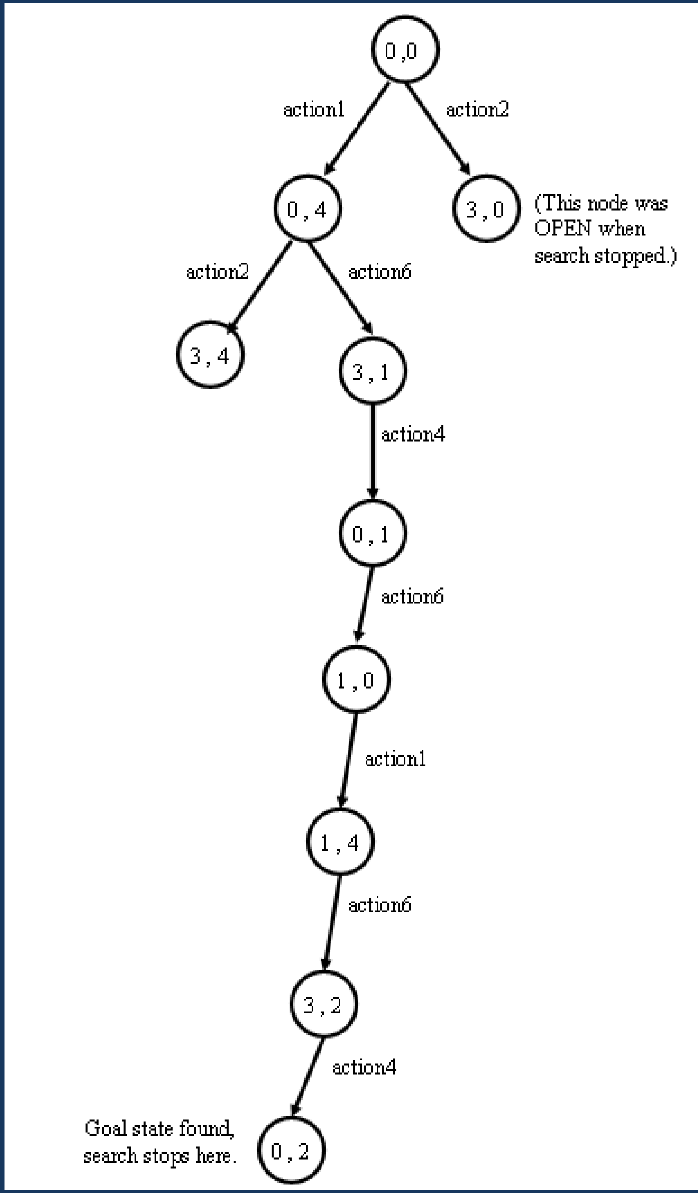
OPEN = (0,2) (3,0)
CLOSED = (0,0) (0,4) (3,4) (3,1) (0,1) (1,0) (1,4) (3,2)

Water Jugs Problem DFS Solution



OPEN = (3,0)
CLOSED = (0,0) (0,4) (3,4) (3,1) (0,1) (1,0) (1,4) (3,2) (0,2)

Water Jugs Problem DFS Solution



Water Jugs Problem DFS Solution

- <u>Solution Path:</u>

  initial state: (0,0)

  action1: (0,4)

  action6: (3,1)

  action4: (0,1)

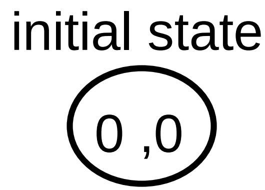
  action6: (1,0)

  action1: (1,4)

  action6: (3,2)

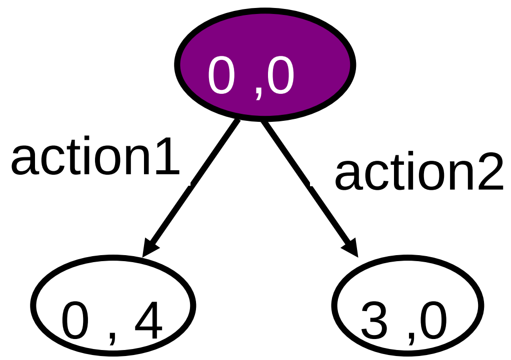
  action4: (0,2) goal state

Water Jugs Problem BFS Solution



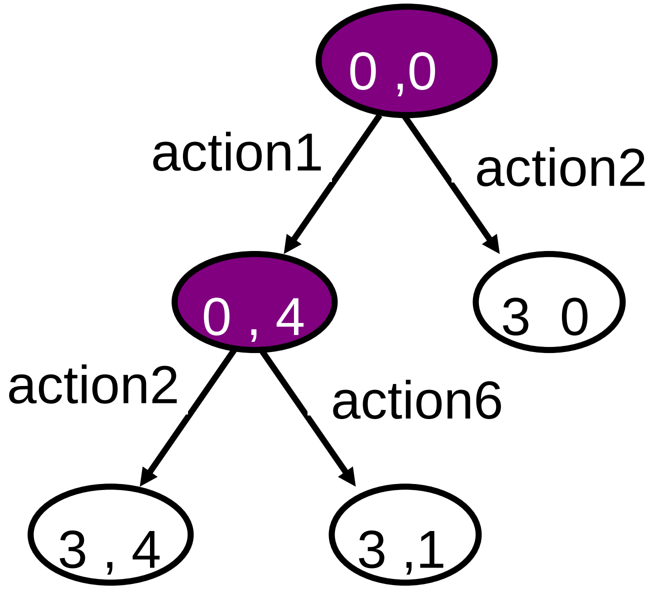
OPEN = (0,0)
CLOSED = empty

Water Jugs Problem BFS Solution



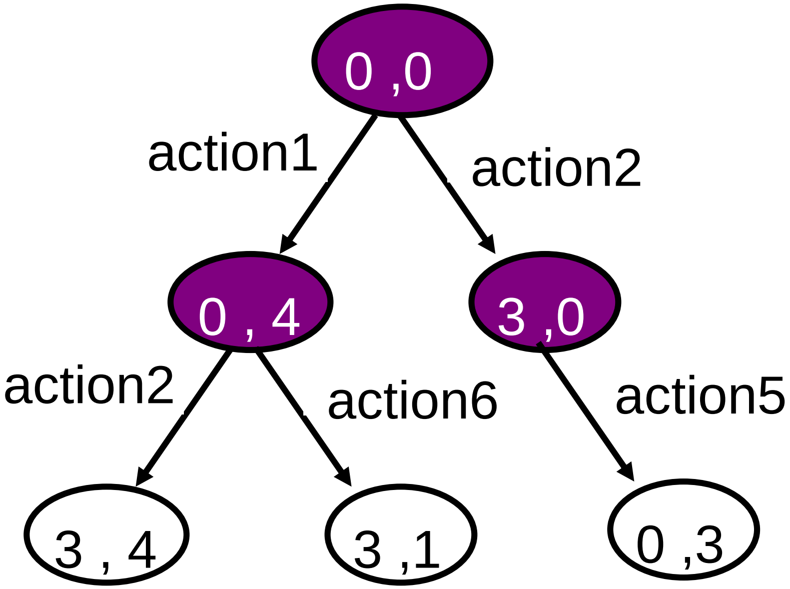
OPEN = (0,4) (3,0)
CLOSED = (0,0)

Water Jugs Problem BFS Solution



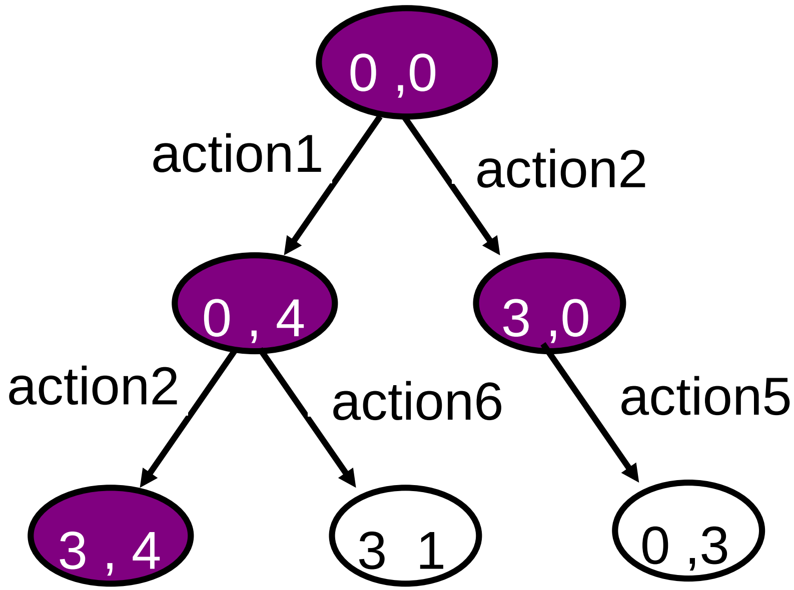
OPEN = (3,0) (3,4) (3,1)
CLOSED = (0,0) (0,4)

Water Jugs Problem BFS Solution



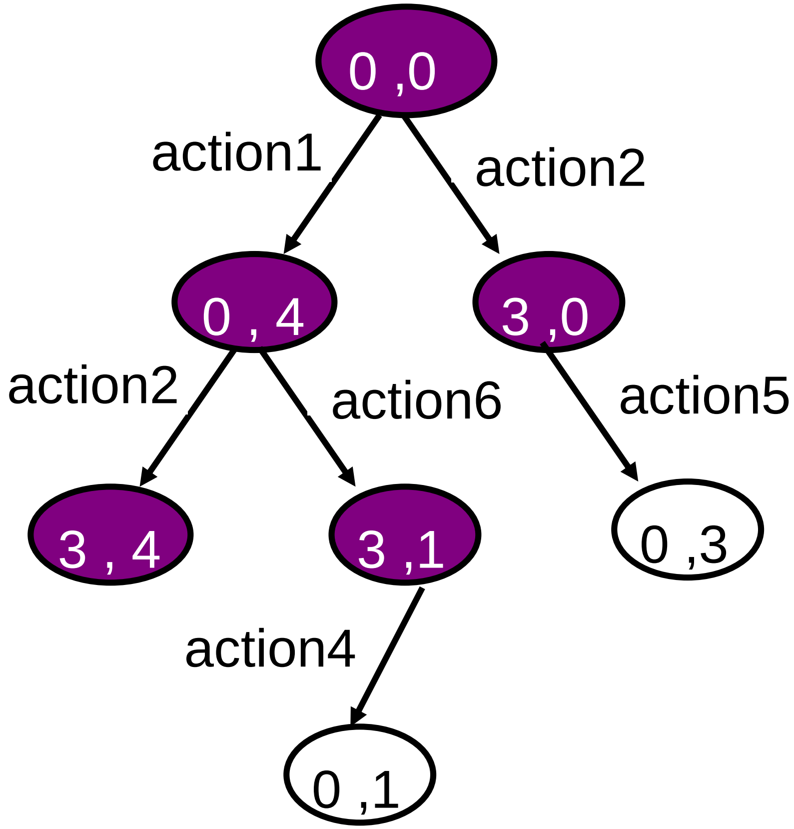
OPEN = (3,4) (3,1) (0,3)
CLOSED = (0,0) (0,4) (3,0)

Water Jugs Problem BFS Solution



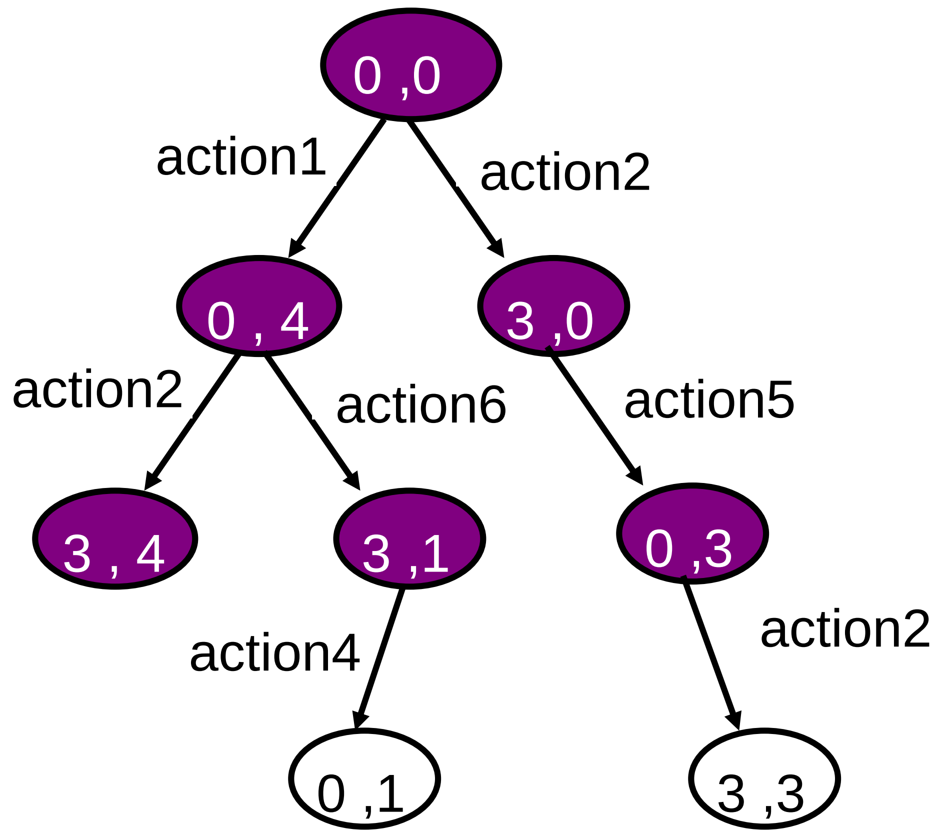
OPEN = (3,1) (0,3)
CLOSED = (0,0) (0,4) (3,0) (3,4)

Water Jugs Problem BFS Solution



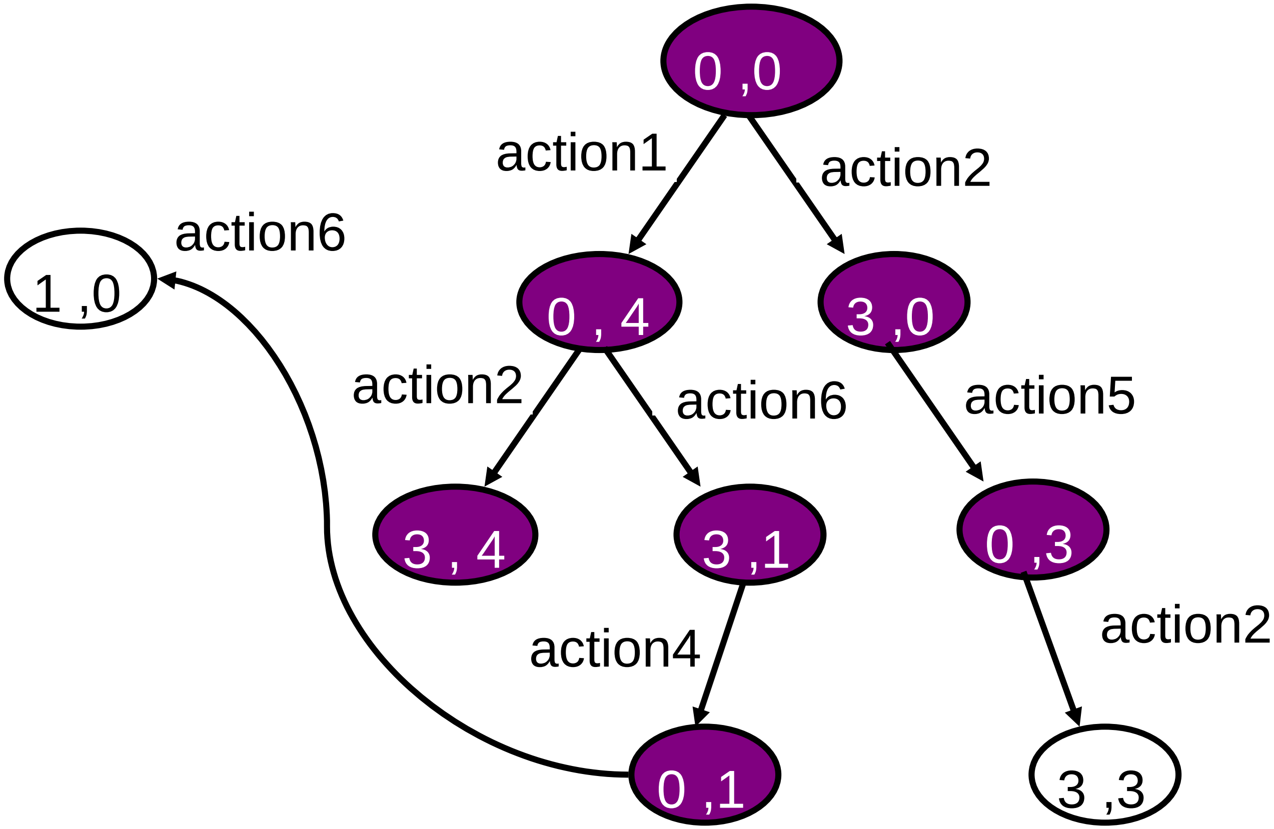
OPEN = (0,3) (0,1)
CLOSED = (0,0) (0,4) (3,0) (3,4) (3,1)

Water Jugs Problem BFS Solution



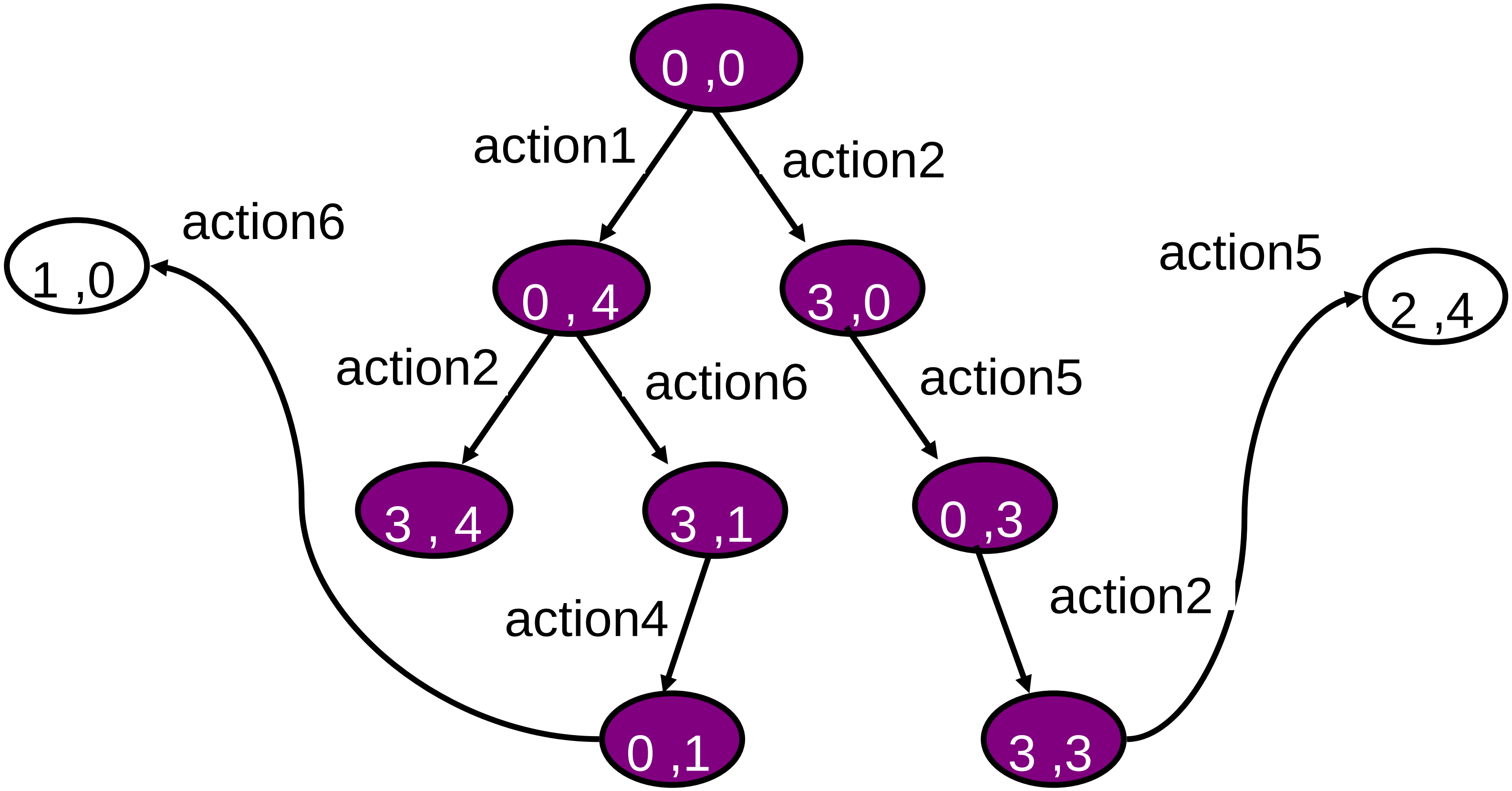
OPEN = (0,1) (3,3)
CLOSED = (0,0) (0,4) (3,0) (3,4) (3,1) (0,3)

Water Jugs Problem BFS Solution



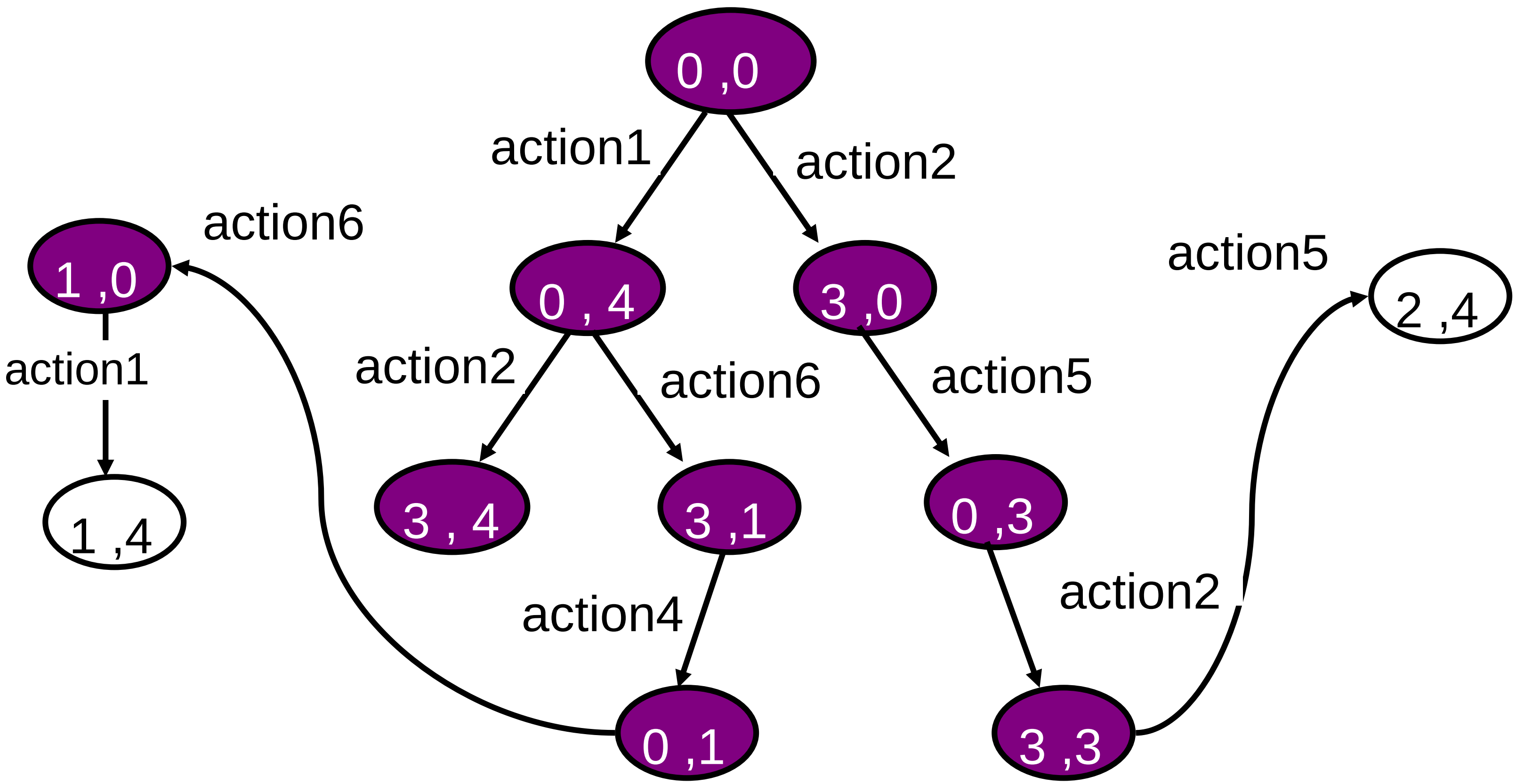
OPEN = (3,3) (1,0)
CLOSED = (0,0) (0,4) (3,0) (3,4) (3,1) (0,3) (0,1)

Water Jugs Problem BFS Solution



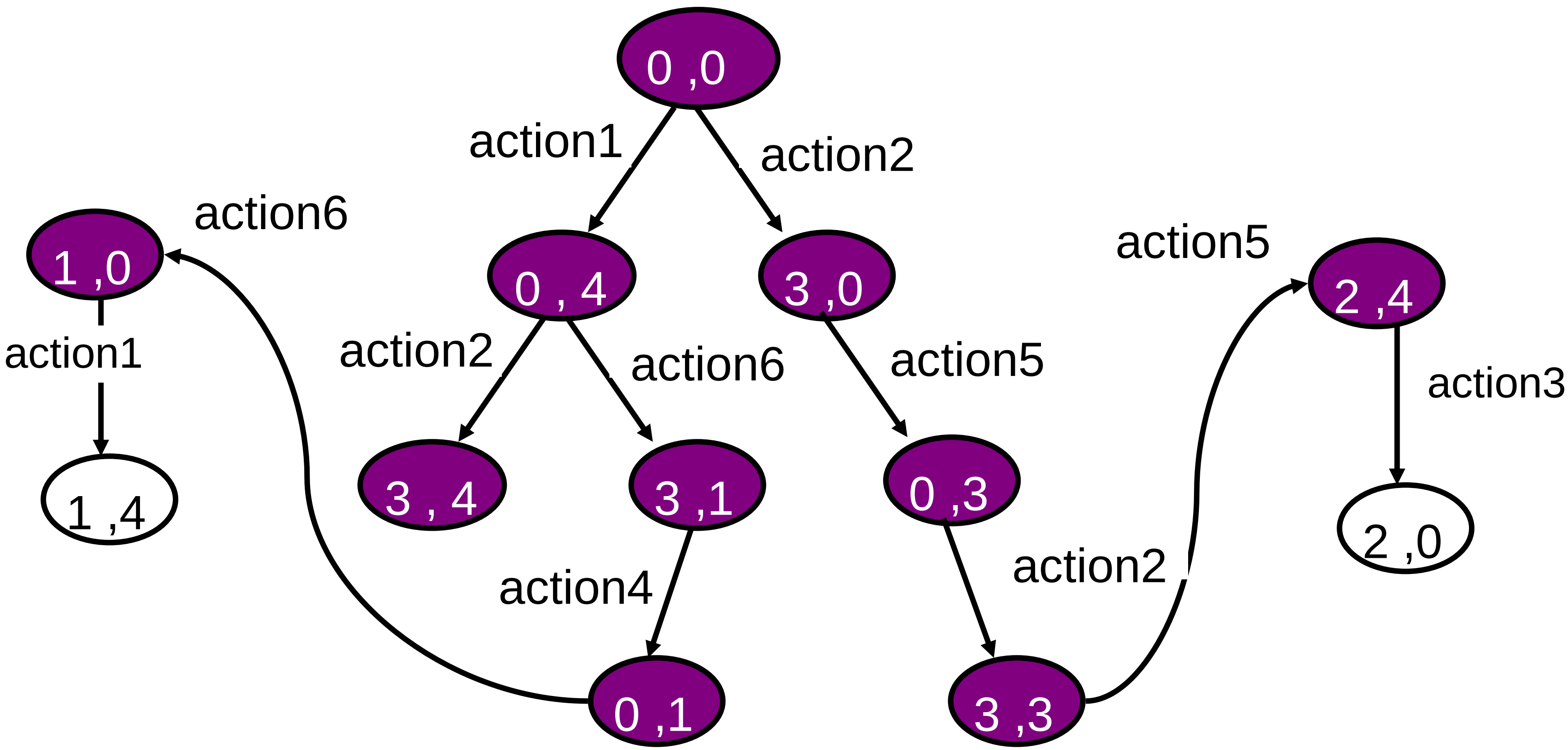
OPEN = (1,0) (2,4)
CLOSED = (0,0) (0,4) (3,0) (3,4) (3,1) (0,3) (0,1) (3,3)

Water Jugs Problem BFS Solution



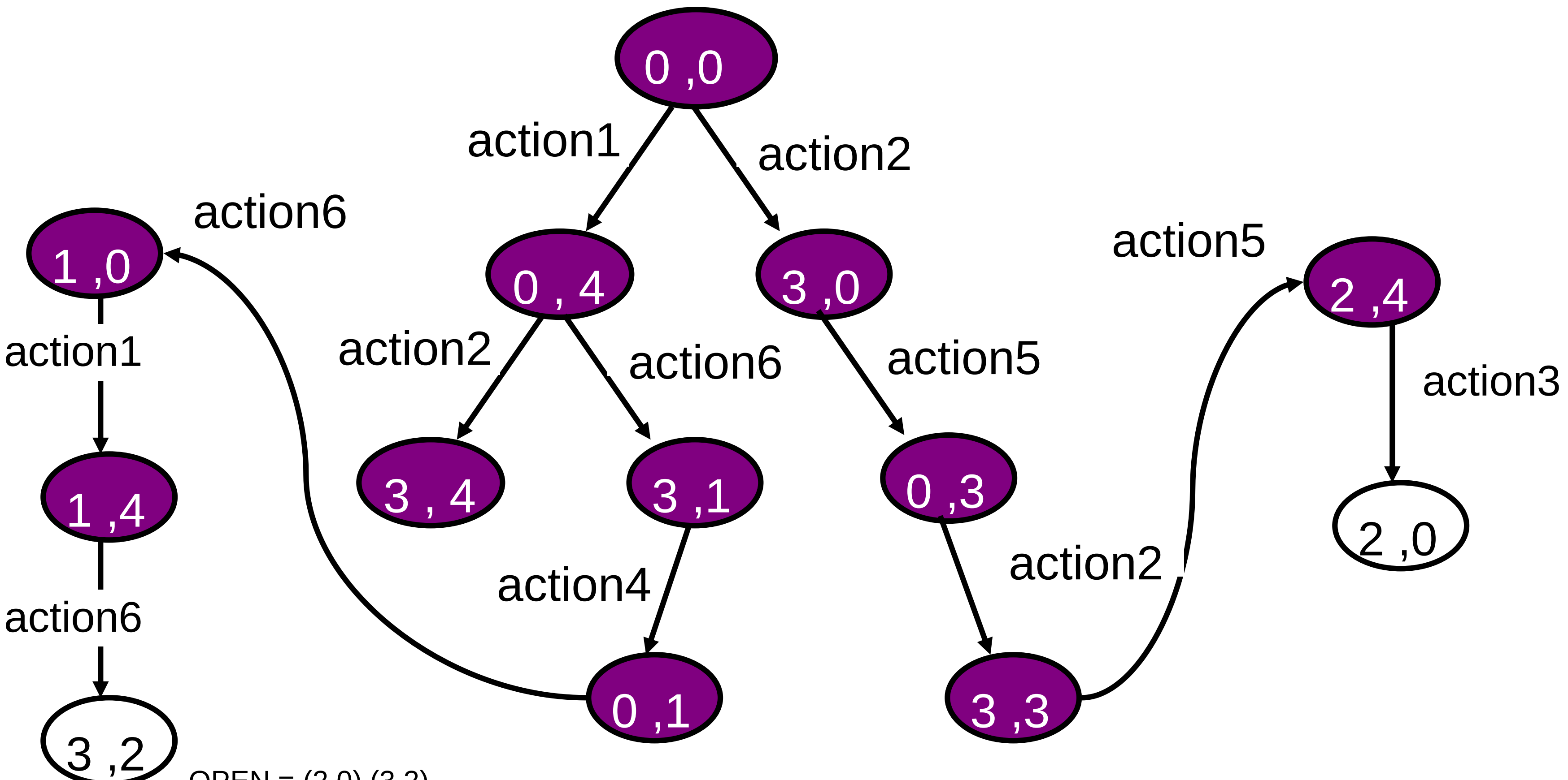
OPEN = (2,4) (1,4)
CLOSED = (0,0) (0,4) (3,0) (3,4) (3,1) (0,3) (0,1) (3,3) (1,0)

Water Jugs Problem BFS Solution



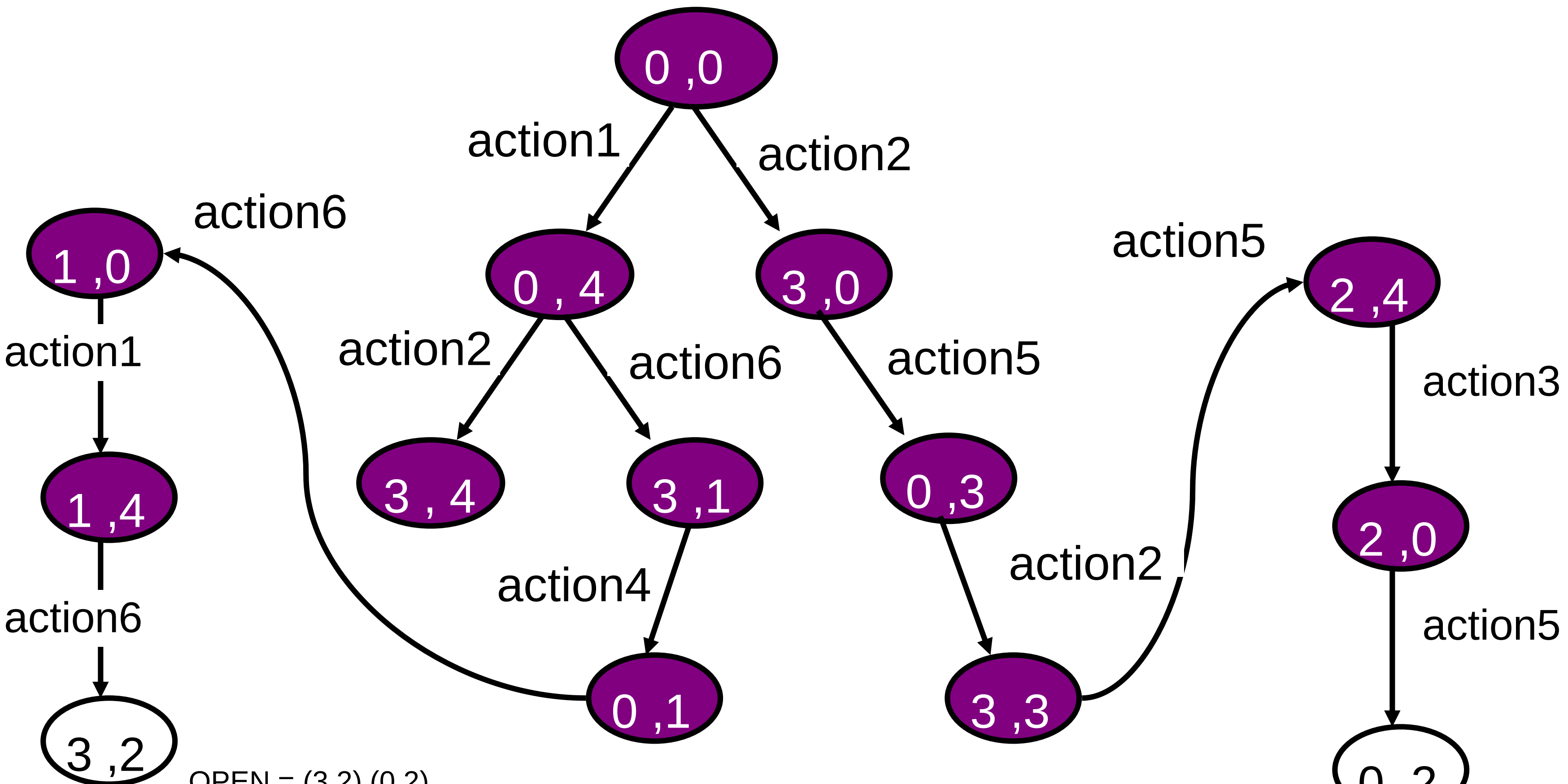
OPEN = (1,4) (2,0)
CLOSED = (0,0) (0,4) (3,0) (3,4) (3,1) (0,3) (0,1) (3,3) (1,0) (2,4)

Water Jugs Problem BFS Solution



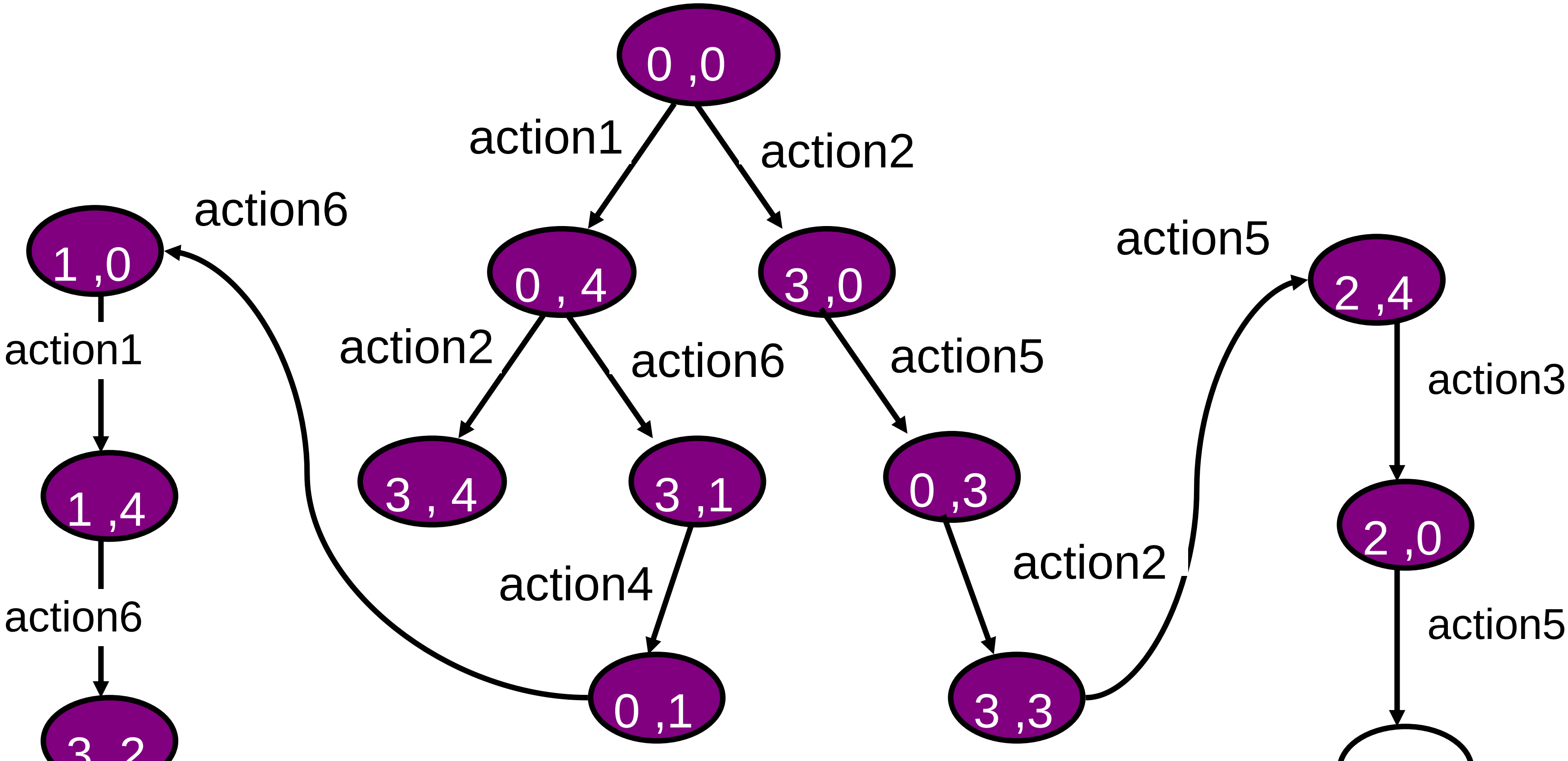
OPEN = (2,0) (3,2)
CLOSED = (0,0) (0,4) (3,0) (3,4) (3,1) (0,3) (0,1) (3,3) (1,0) (2,4) (1,4)

Water Jugs Problem BFS Solution



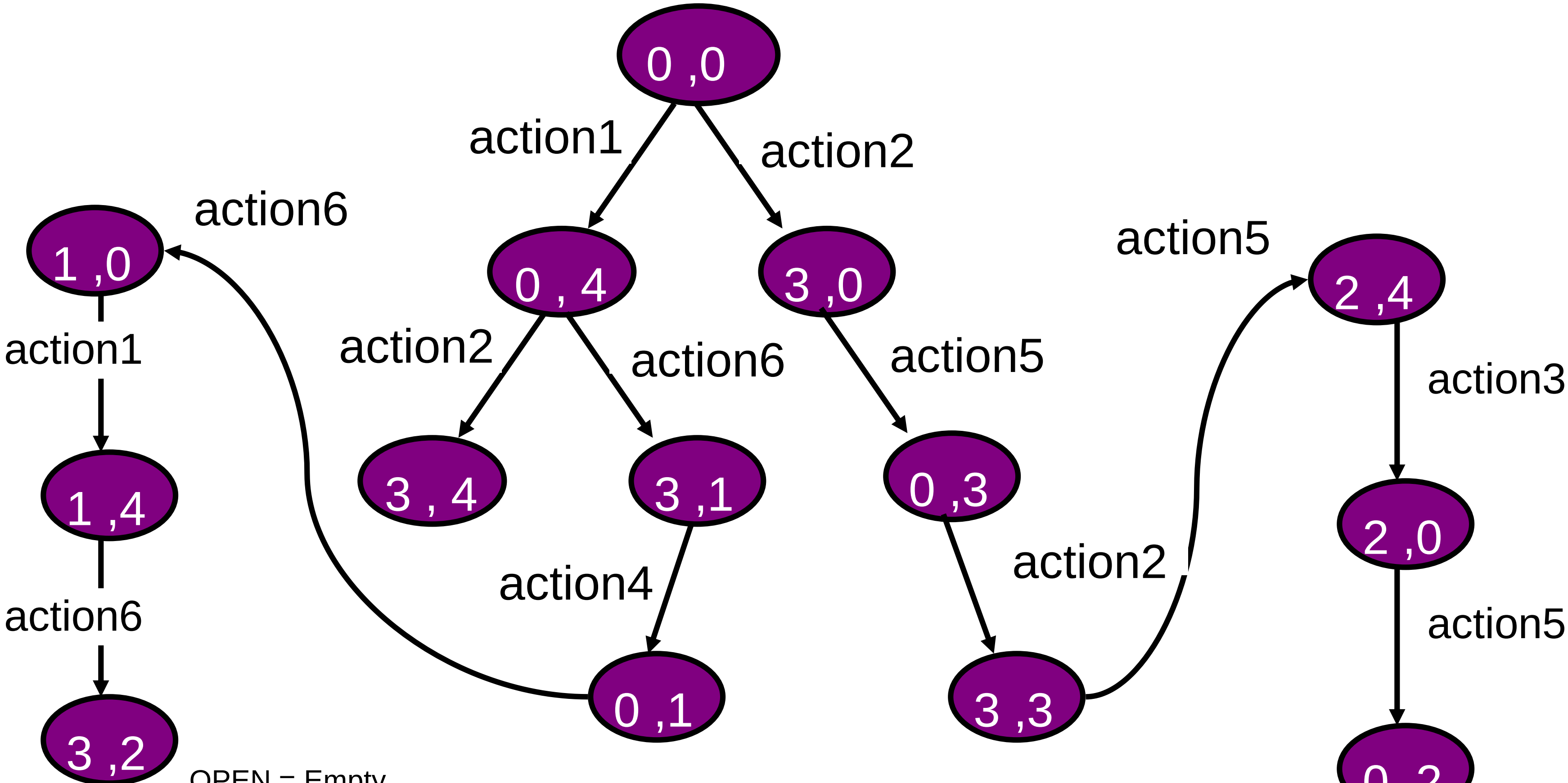
OPEN = (3,2) (0,2)
CLOSED = (0,0) (0,4) (3,0) (3,4) (3,1) (0,3) (0,1) (3,3) (1,0) (2,4) (1,4) (2,0)

Water Jugs Problem BFS Solution



OPEN = (0,2)
CLOSED = (0,0) (0,4) (3,0) (3,4) (3,1) (0,3) (0,1) (3,3) (1,0) (2,4) (1,4) (2,0) (3,2)

Water Jugs Problem BFS Solution



OPEN = Empty
CLOSED = (0,0) (0,4) (3,0) (3,4) (3,1) (0,3) (0,1) (3,3) (1,0) (2,4) (1,4) (2,0) (3,2) (0,2)

Water Jugs Problem BFS Solution

- Solution Path:

  initial state: (0,0)

  action2: (3,0)

  action5: (0,3)

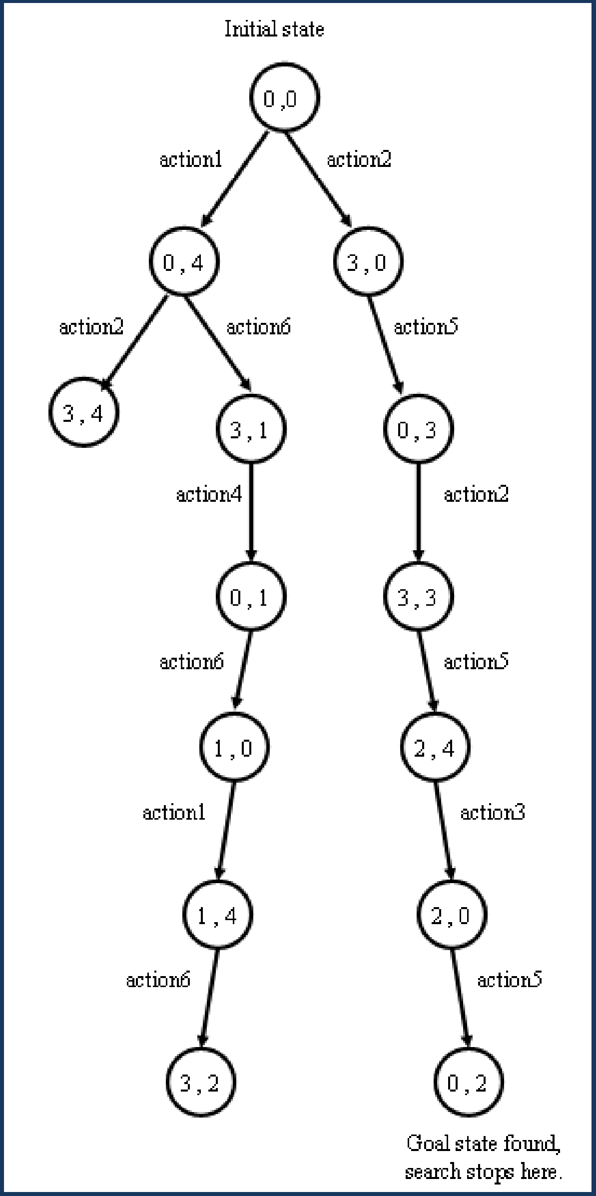
  action2: (3,3)

  action5: (2,4)

  action3: (2,0)

  action5: (0,2) goal state

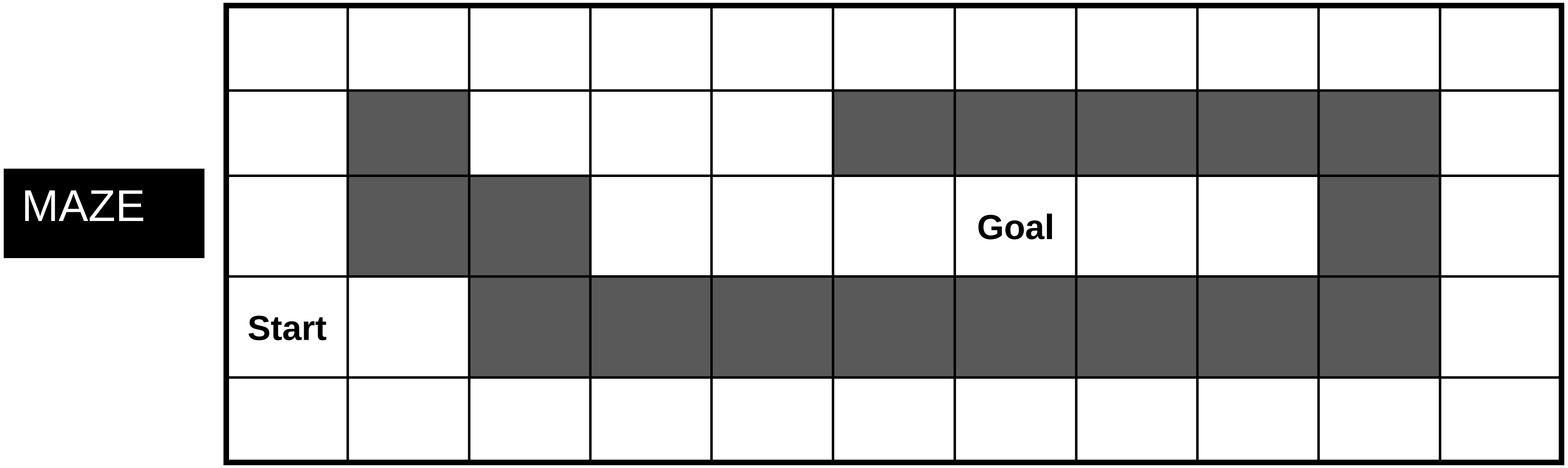
Water Jugs Problem BFS Solution



Path Planning in a Maze

- A robot will go from a start location to a goal location in a maze (labyrinth).
- At each step, there are four directions to move: North, East, South, West; if there is no obstacle.
- <u>Problem:</u> Find the shortest path for the robot, so that the robot can follow this path plan.
- The map is known, the plan is constructed offline.

Path Planning in a Maze



Path Planning–Problem Formulation

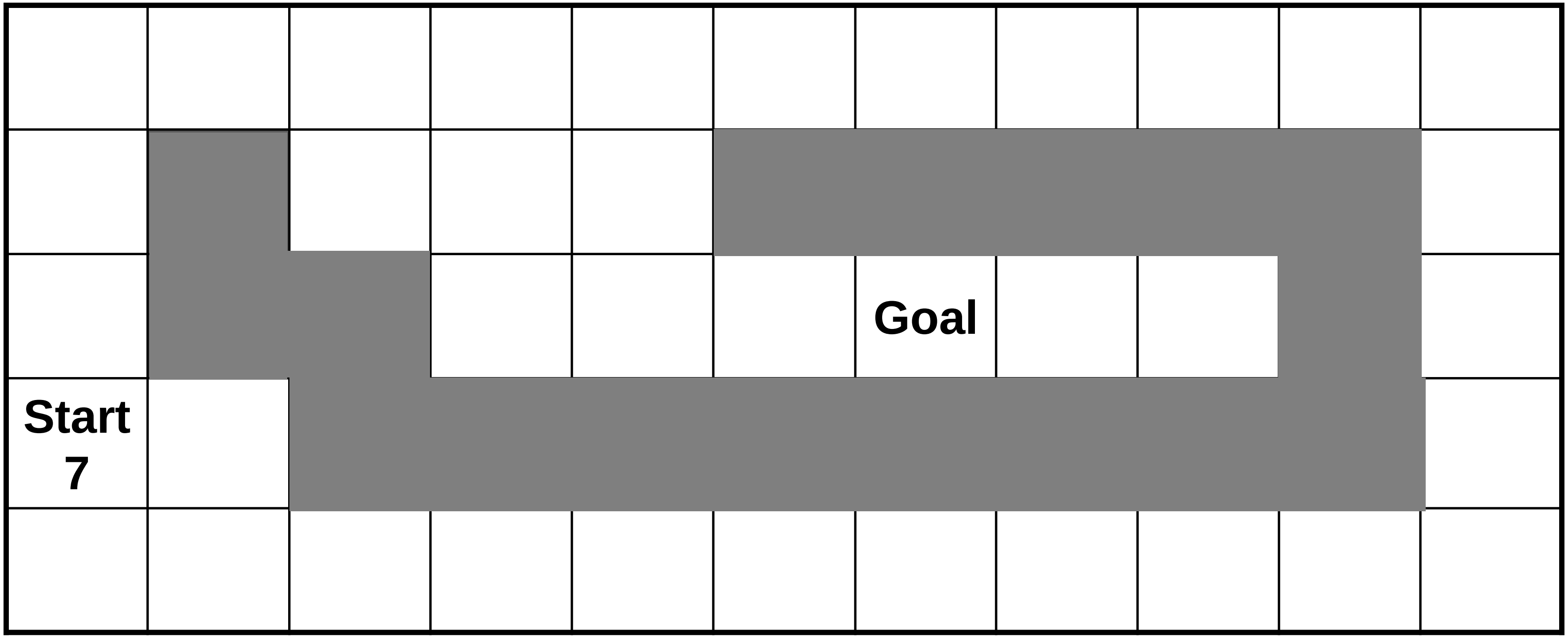
- **States**: Maze locations
- **Initial State**: Start location of the robot
- **Actions**: Move in four directions
 {North, South, West, East} if possible
- **Goal Test**: Stop location of robot
- **Step Cost**: 1
  Path Cost: Length of the path from start to goal.

Path Planning- Greedy Solution

- h(n) = Manhattan distance from node n to goal
- f(n) = h(n)
- If there are nodes with equal h(n) values, Greedy search will select the **newest** node in OPEN.

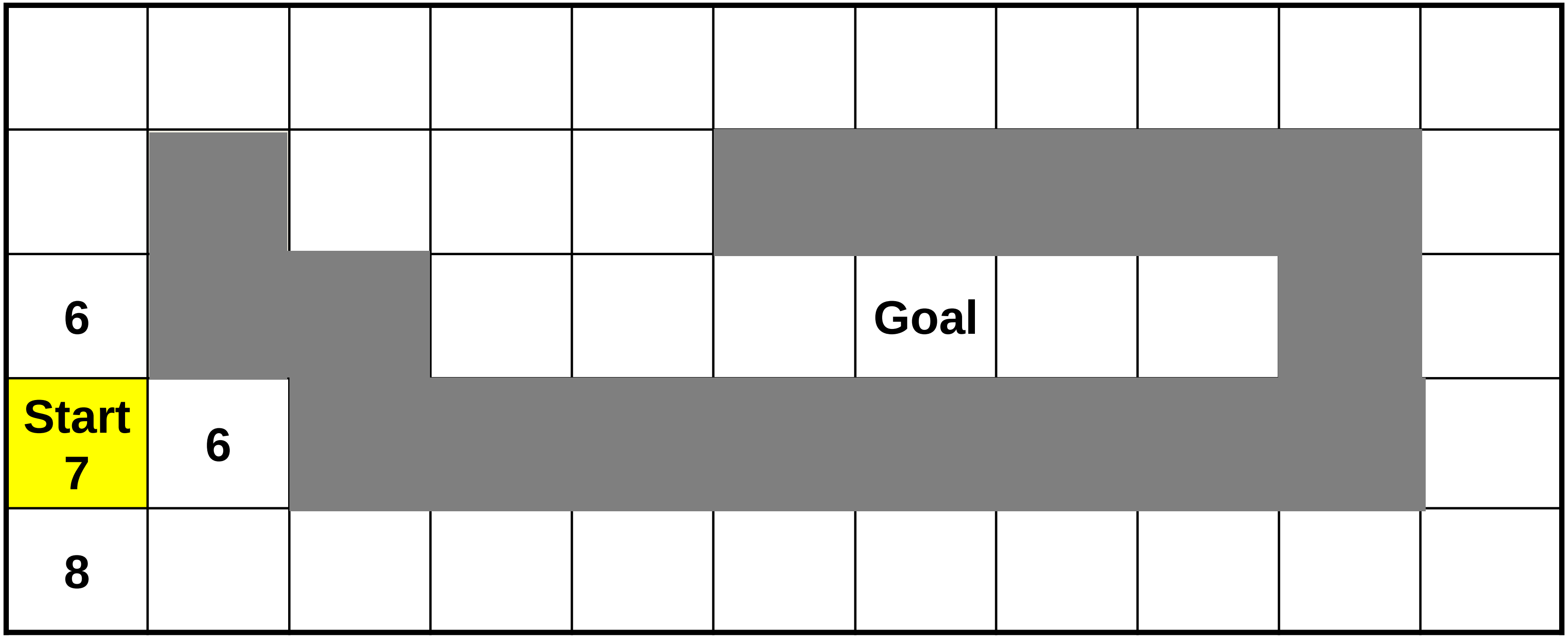
Path Planning- Greedy Solution

- Initial step: The start location is in the OPEN list.
- Manhattan distance from start location to the goal location:
  f (start) = h (start) = 7



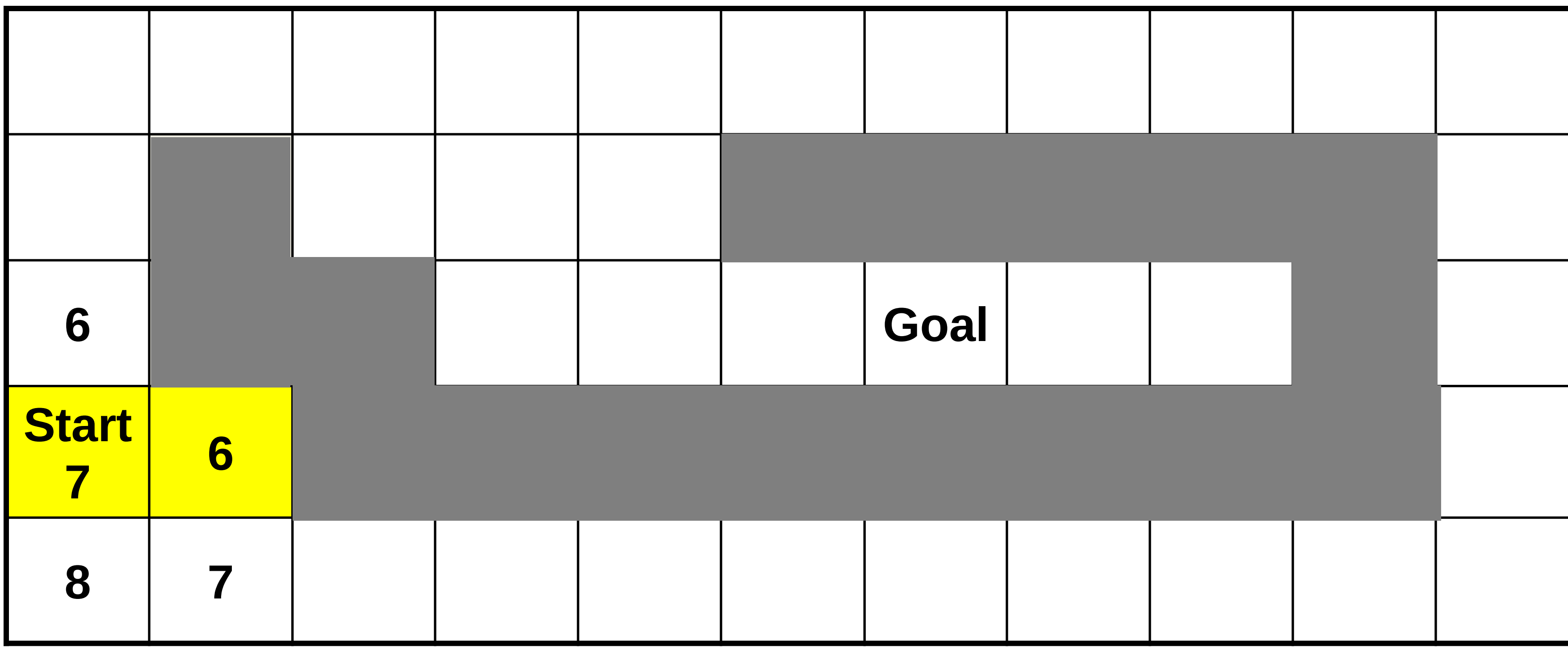
Path Planning- Greedy Solution

- Start location (7) is selected and closed.
- Its successors are generated and put on OPEN.
- Manhattan distances of successors are (6), (6), and (8).



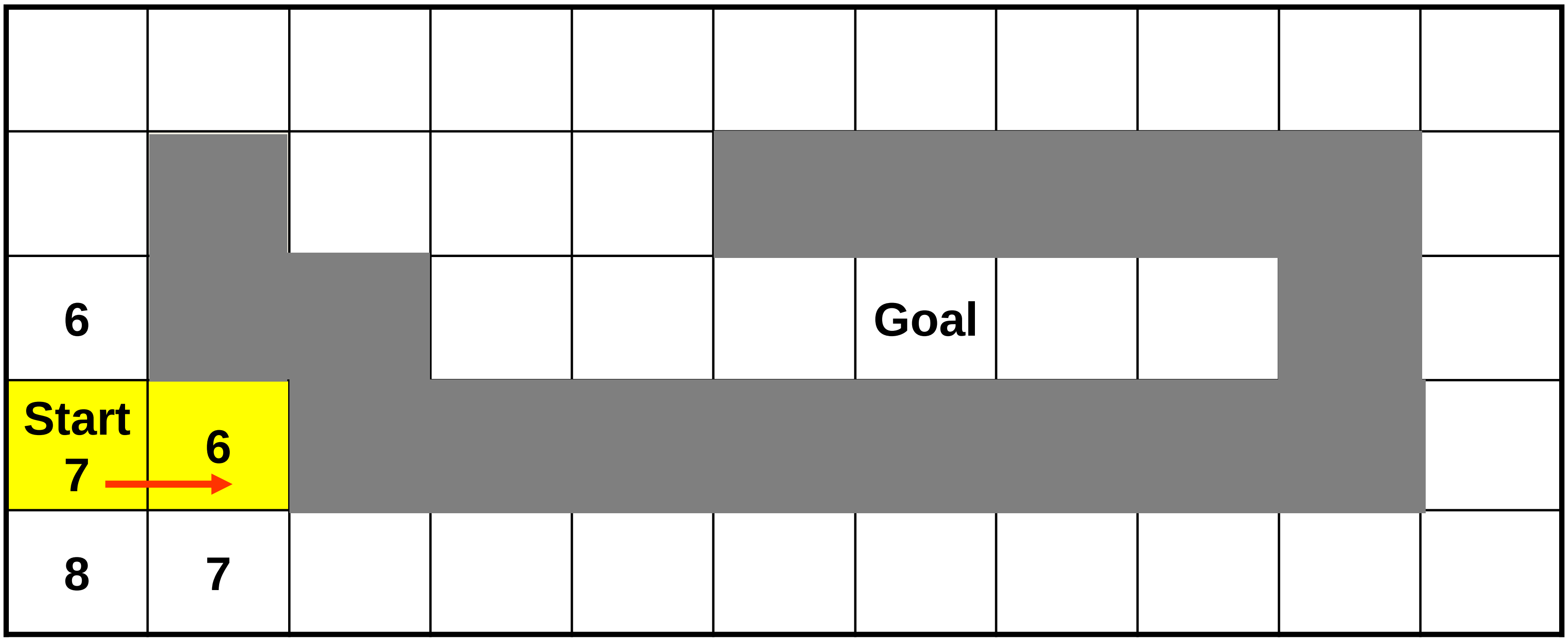
Path Planning- Greedy Solution

- Search moves to east.
- The lower location (6) is selected and closed.
- Its successor (7) is generated and put on OPEN.



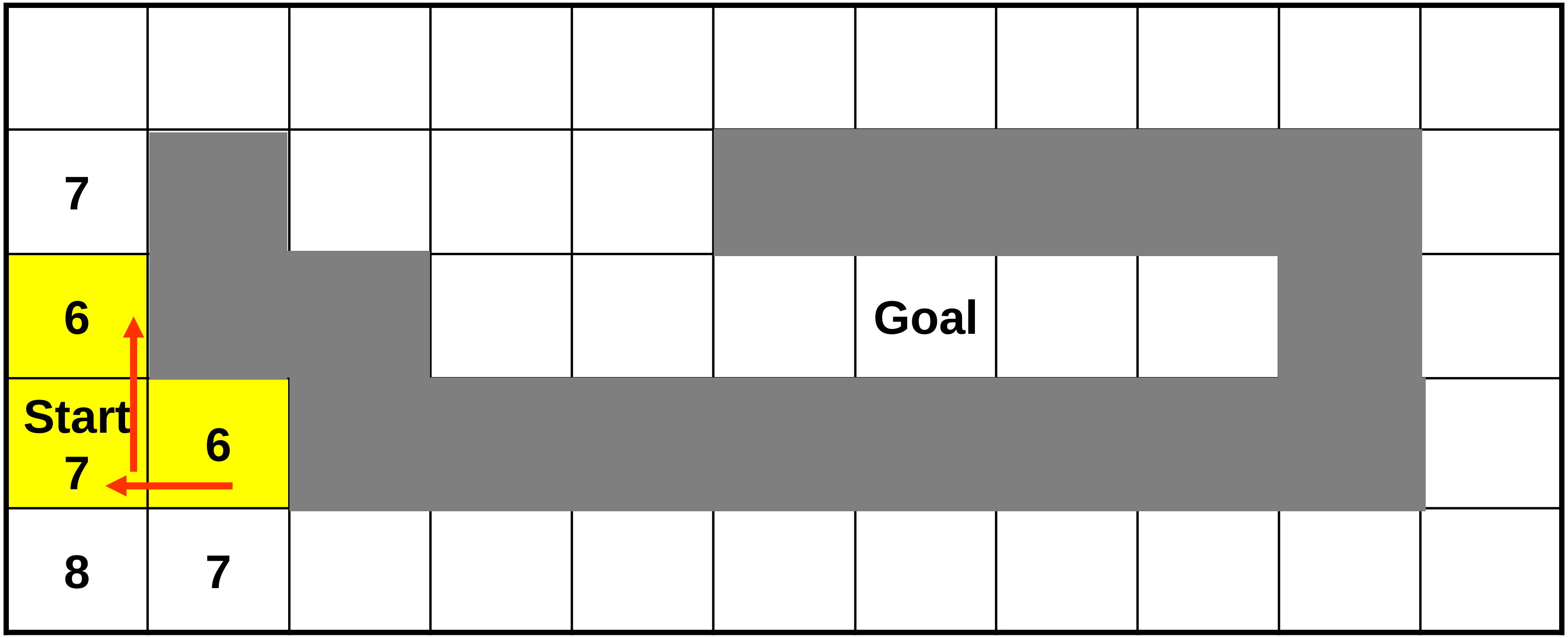
Path Planning- Greedy Solution

- Search moves to east.
- The lower location (6) is selected and closed.
- Its successor (7) is generated and put on OPEN.



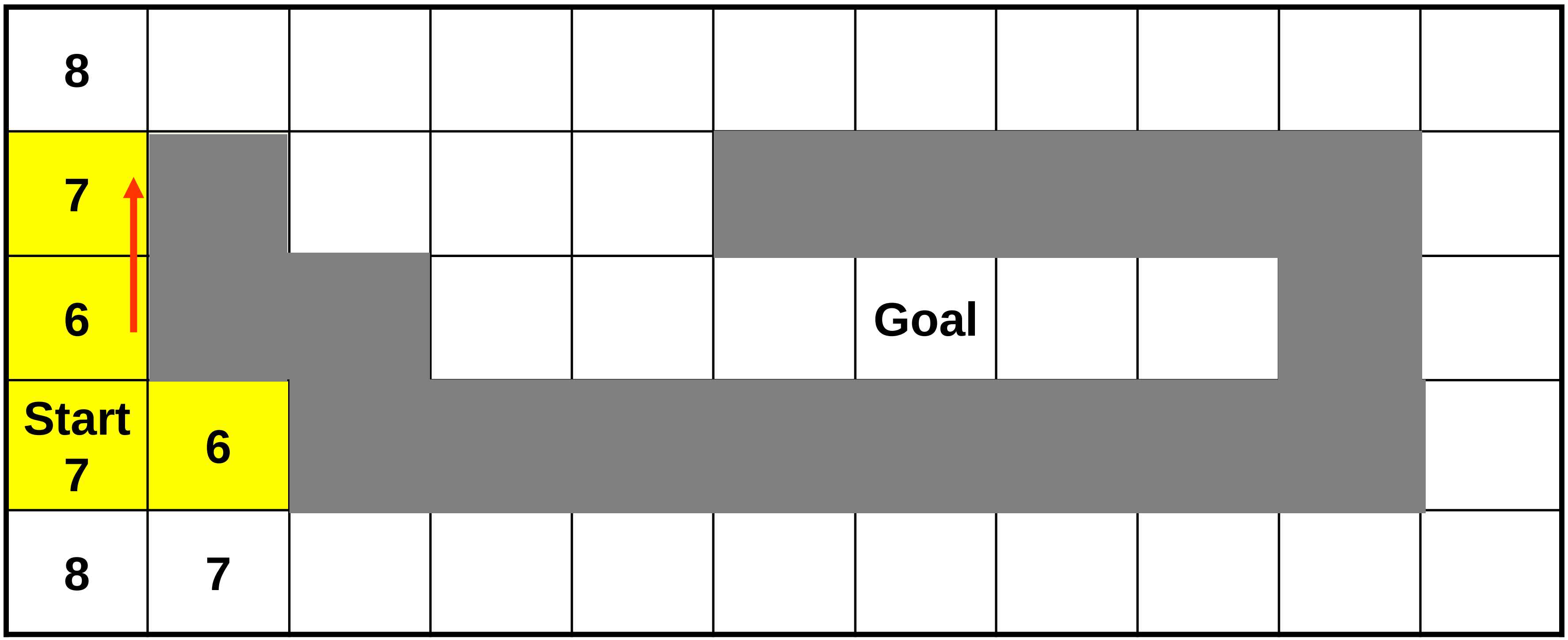
Path Planning- Greedy Solution

- Search comes back and moves to north.
- The upper location (6) is selected and closed.
- Its successor (7) is generated and put on OPEN.



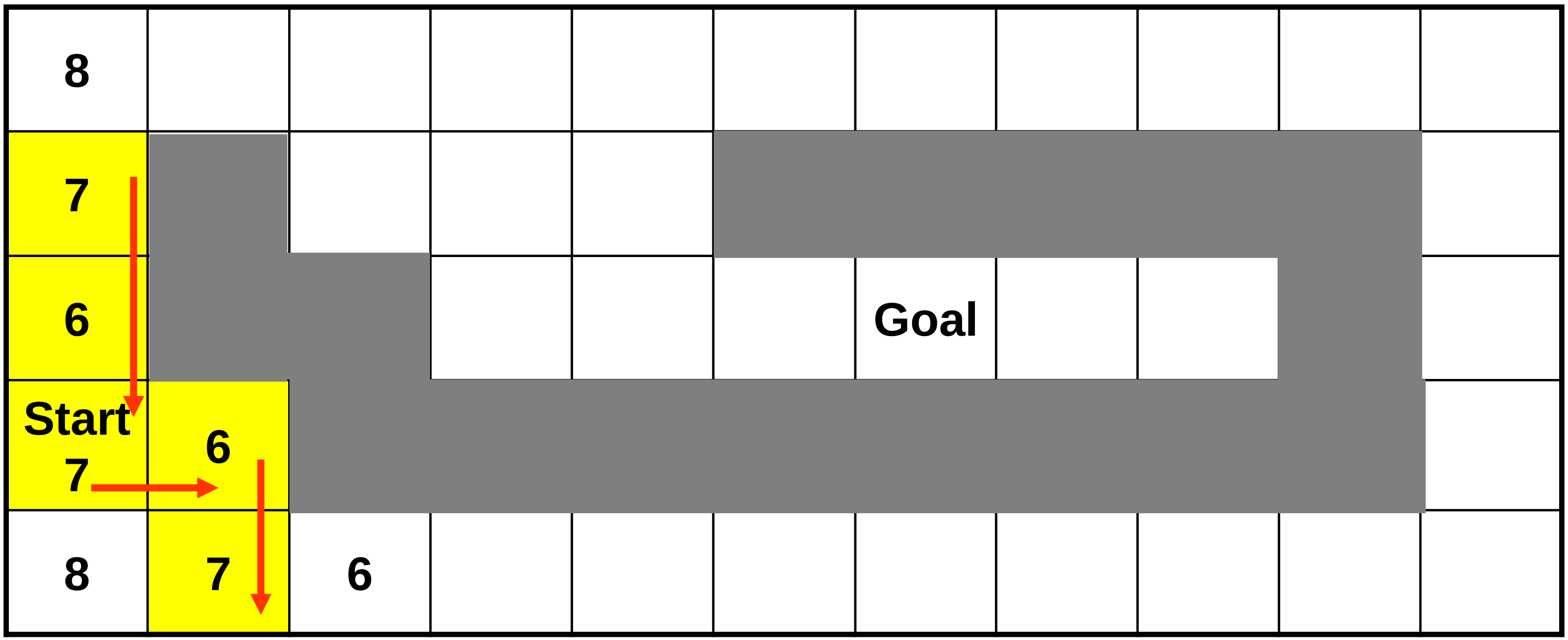
Path Planning- Greedy Solution

- Search moves to north.
- The upper location (7) is selected and closed.
- Its successor (8) is generated and put on OPEN.



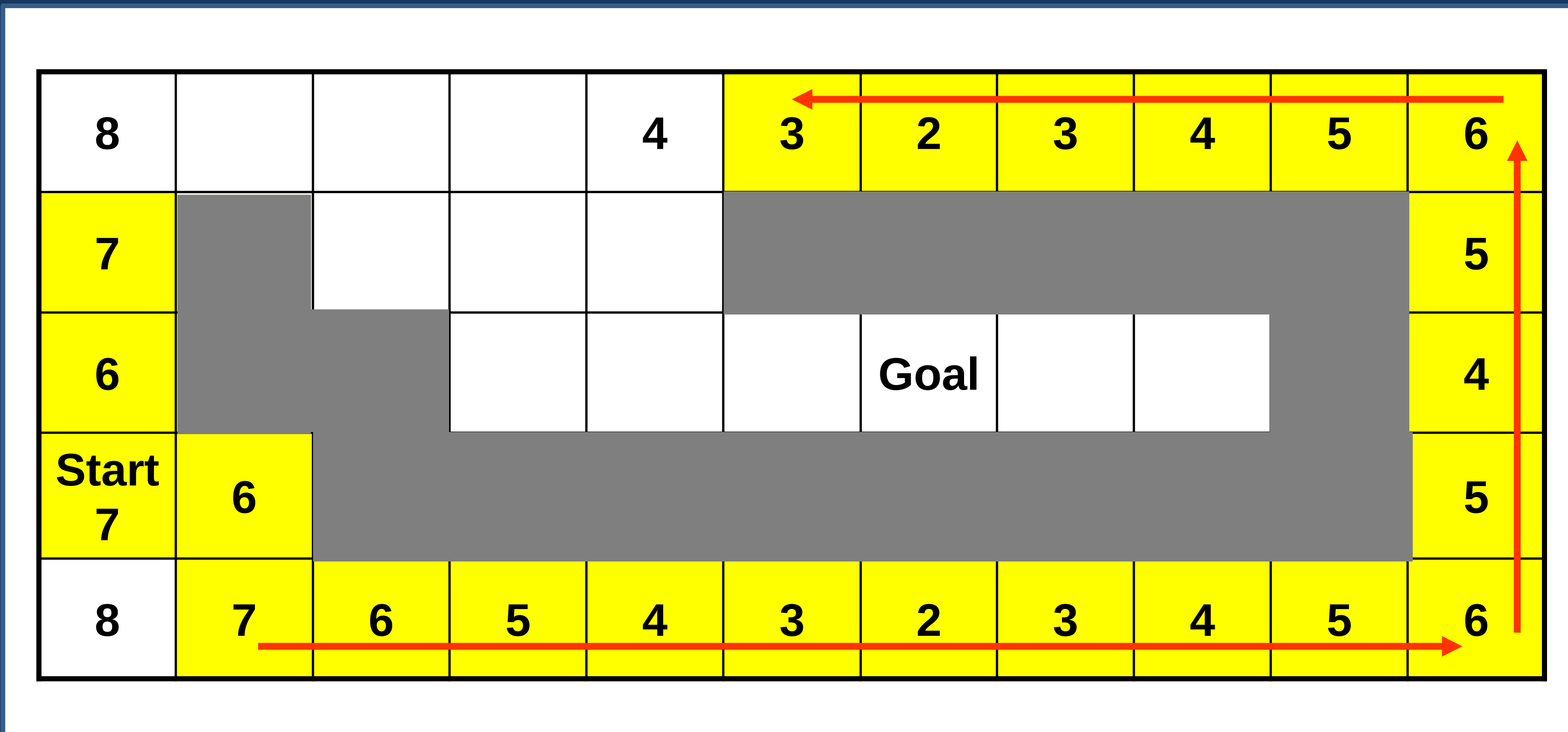
Path Planning- Greedy Solution

- Search comes back and moves to south.
- The lower location (7) is selected and closed.
- Its successor (6) is generated and put on OPEN.



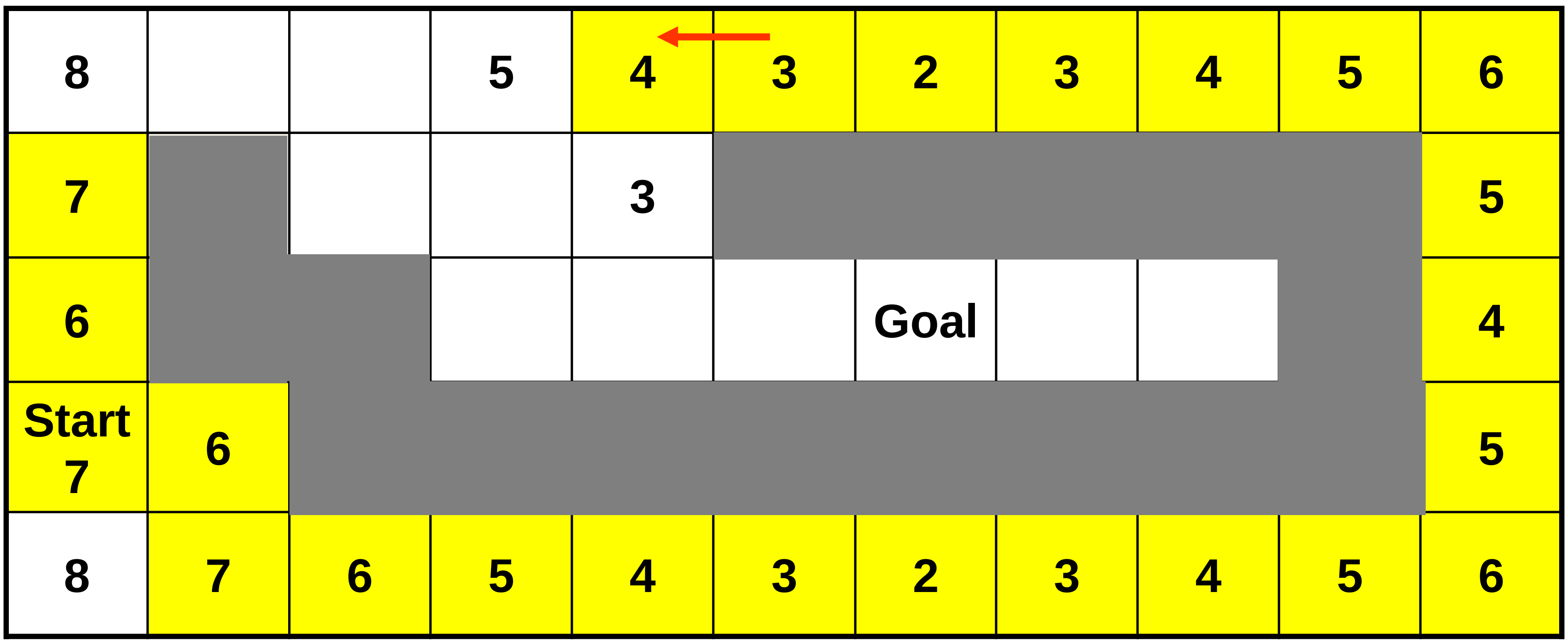
Path Planning- Greedy Solution

- Search goes all the way up through the corridor.
- The location (3) is selected and closed.
- Its successor (4) is generated and put on OPEN.



Path Planning- Greedy Solution

- Search moves to west.
- The location (4) is selected and closed.
- Its succesors (5) and (3) are generated and put on OPEN.



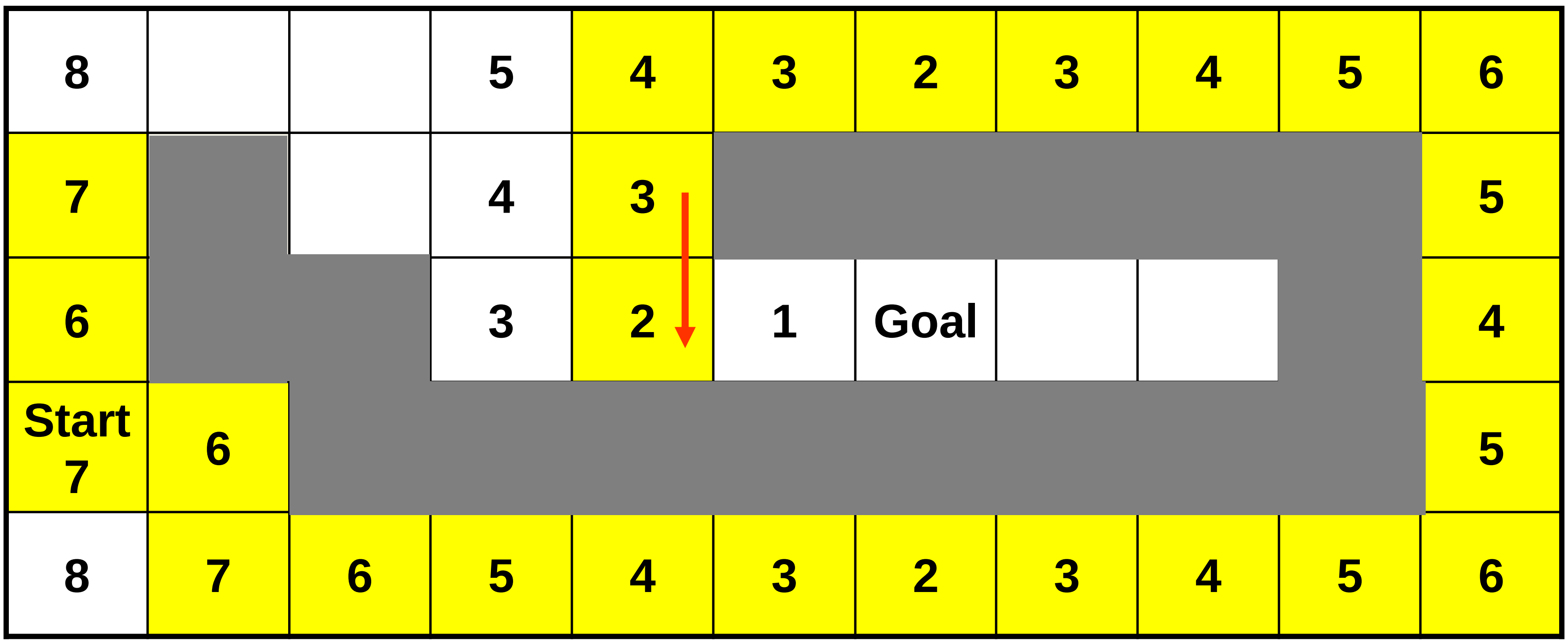
Path Planning- Greedy Solution

- Search moves to south.
- The location (3) is selected and closed.
- Its succesors (4) and (2) are generated and put on OPEN.

8			5	4	3	2	3	4	5	6
7			4	3						5
6				2		Goal				4
Start 7	6									5
8	7	6	5	4	3	2	3	4	5	6

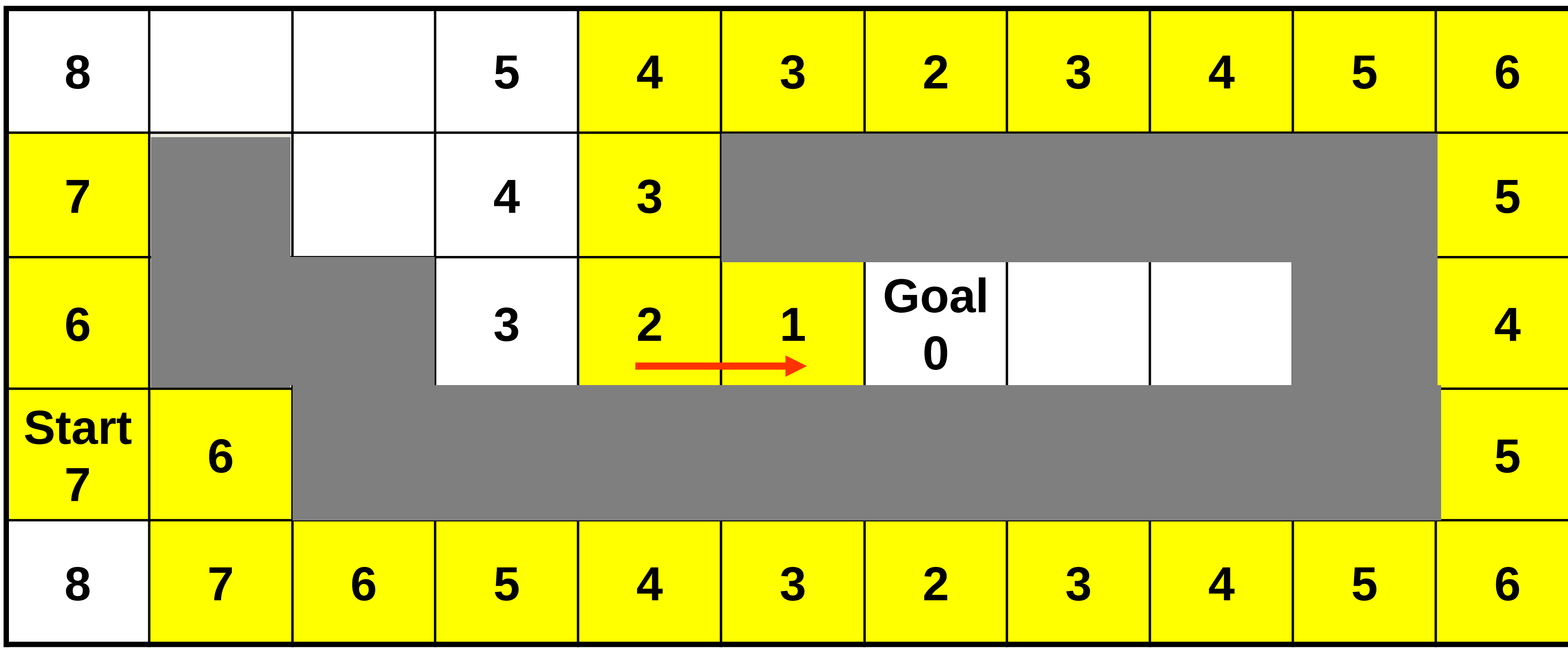
Path Planning- Greedy Solution

- Search moves to south.
- The location (2) is selected and closed.
- Its succesors (3) and (1) are generated and put on OPEN.



Path Planning- Greedy Solution

- Search moves to east.
- The location (1) is selected and closed.
- Its successor (0) is generated and put on OPEN.

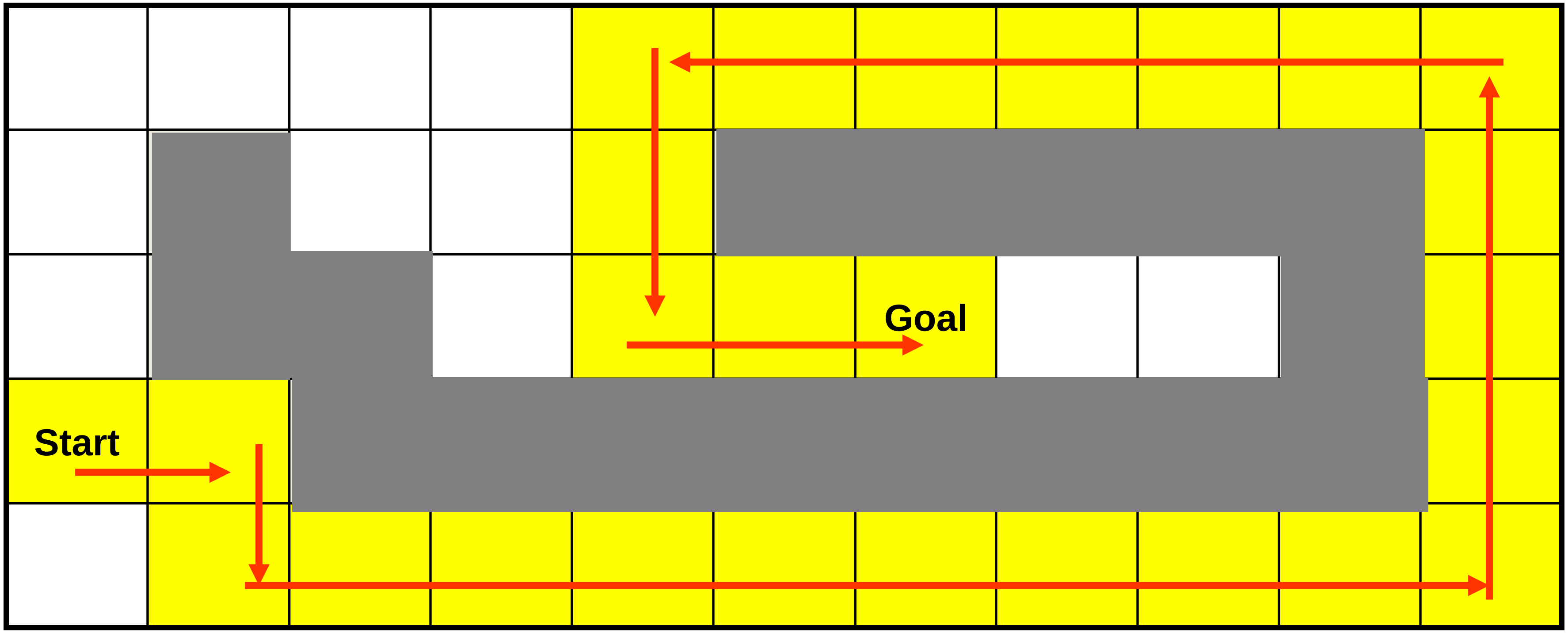


Path Planning- Greedy Solution

- Search moves to east.
- Search ends here, because the goal location is found.

Path Planning- Greedy Solution

- Path found by Greedy search
- Path length=25



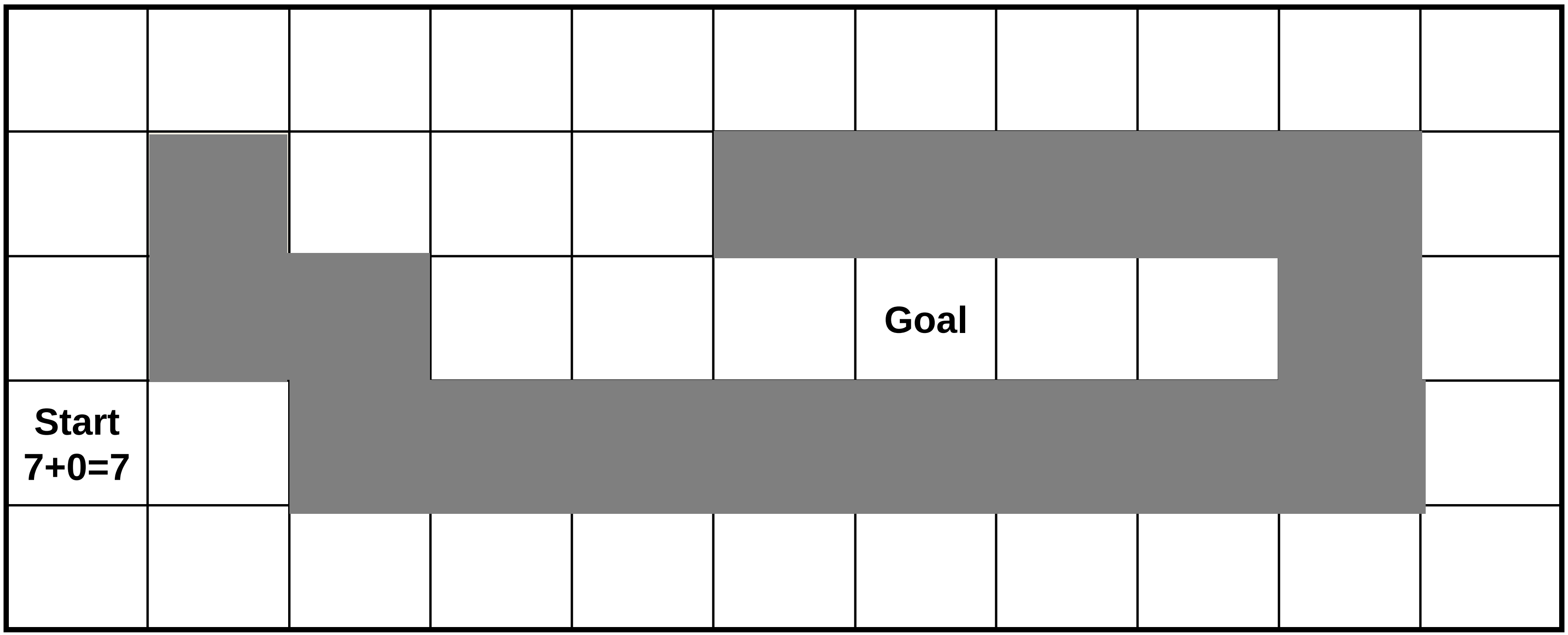
Path Planning- A* Solution

- h(n) = Manhattan distance from node n to goal
- g(n) = Actual cost so far from start to node n
- f(n) = h(n) + g(n)
- If there are nodes with equal f(n) values, A* will select the **newest** node in OPEN.

Path Planning- A* Solution

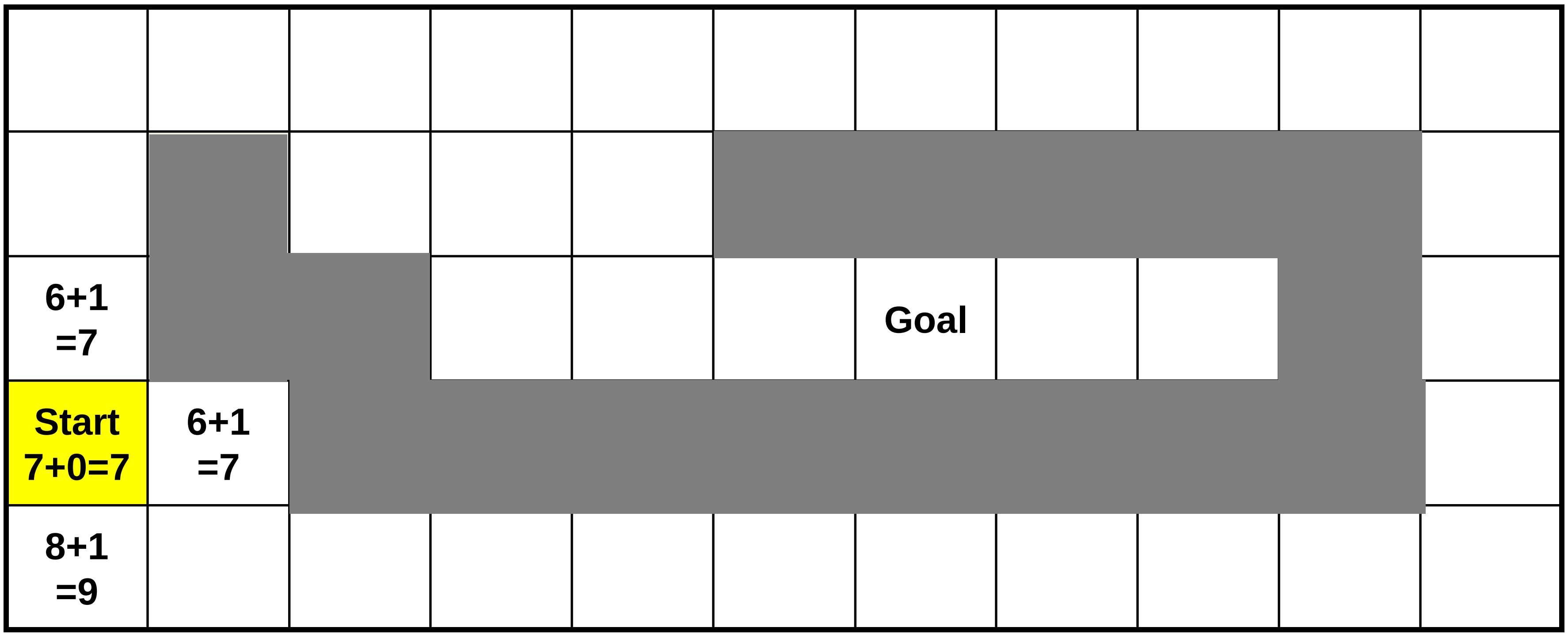
- Initial step: The start location is in the OPEN list.
- f (start) = h (start) + g (start)

  = 7 + 0 = 7



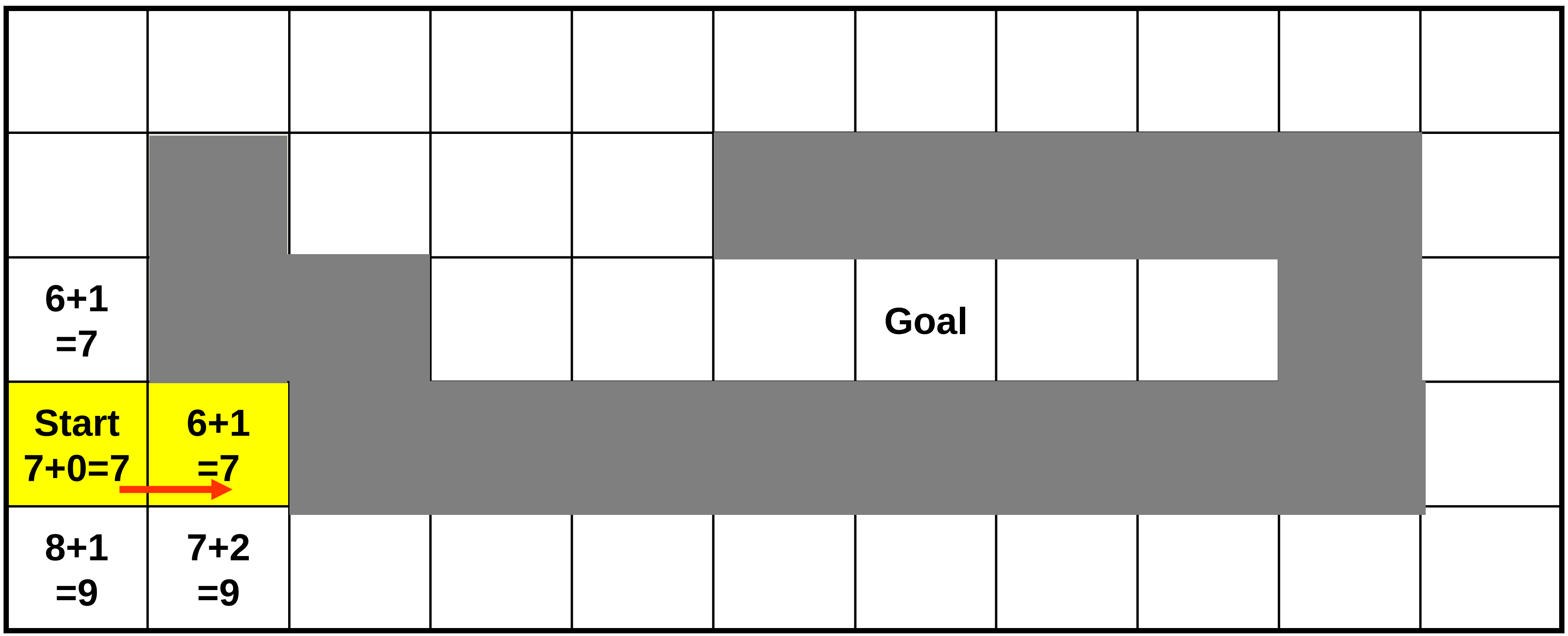
Path Planning- A* Solution

- Start location (7+0=7) is selected and closed.
- Its successors are generated and put on OPEN.
- f values of the successor locations are (6+1=7), (6+1=7), and (8+1=9).



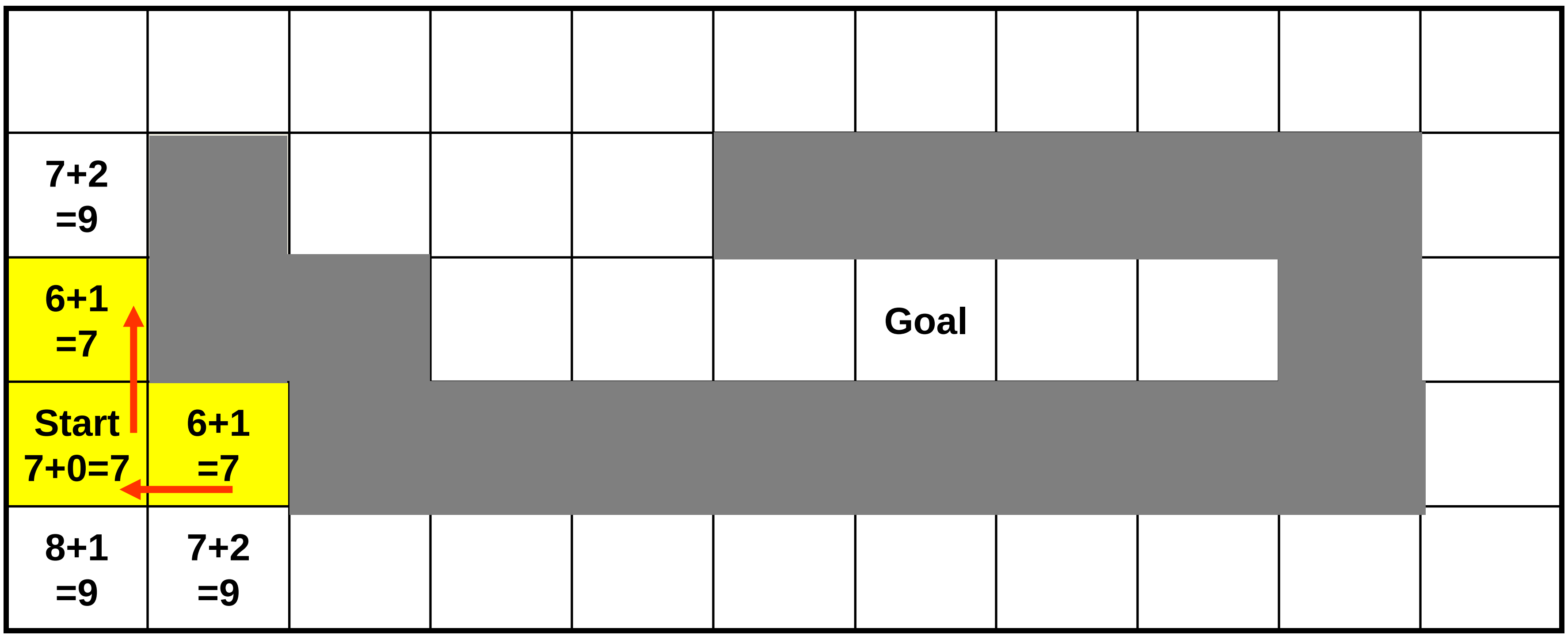
Path Planning- A* Solution

- Search moves to east.
- The lower location (6+1=7) is selected and closed.
- Its successor (7+2=9) is generated and put on OPEN.



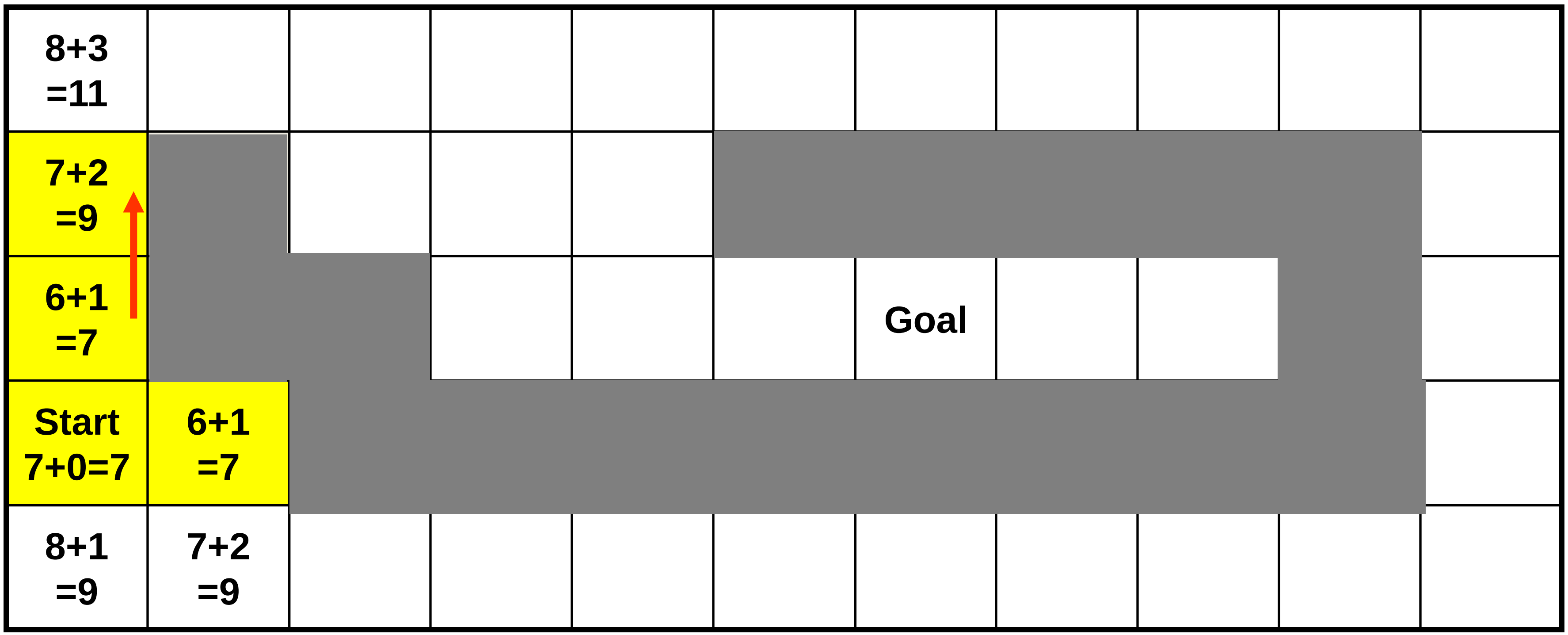
Path Planning- A* Solution

- Search comes back and moves to north.
- The upper location (6+1=7) is selected and closed.
- Its successor (7+2=9) is generated and put on OPEN.



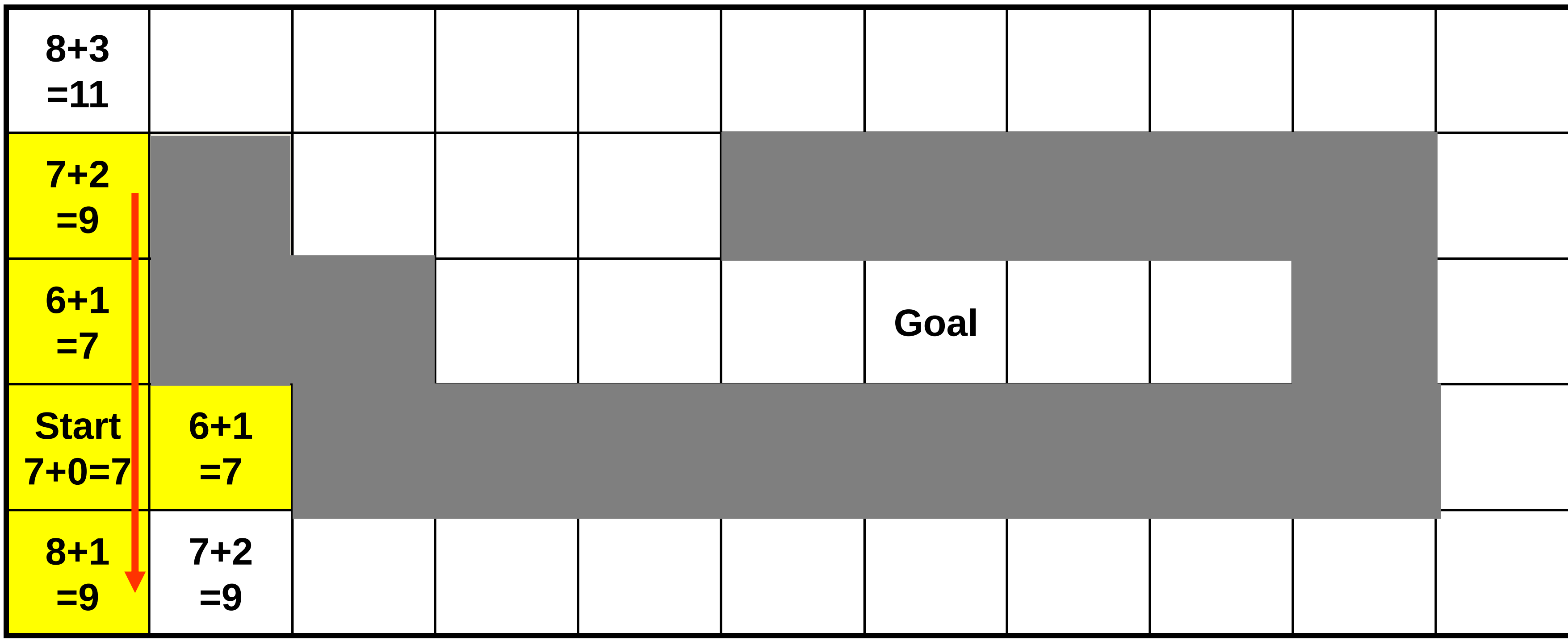
Path Planning- A* Solution

- Search moves to north.
- The upper location (7+2=9) is selected and closed.
- Its successor (8+3=11) is generated and put on OPEN.



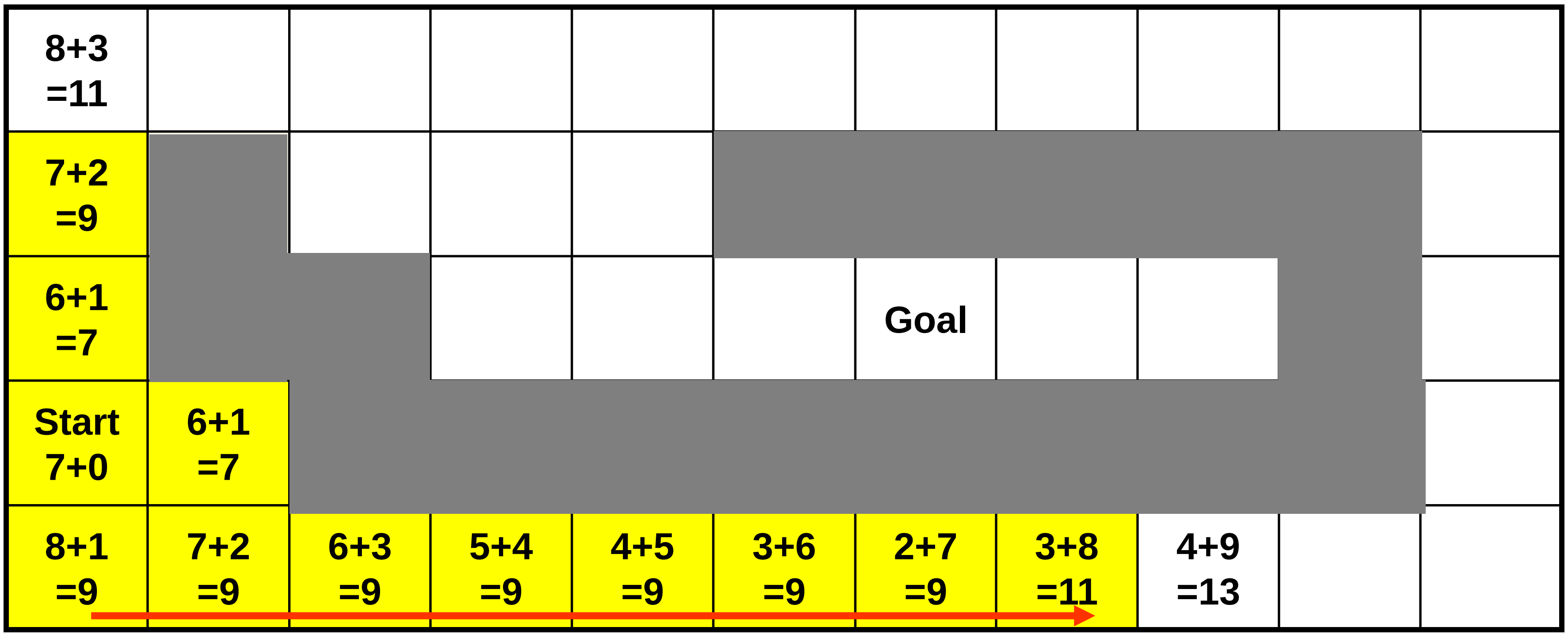
Path Planning- A* Solution

- Search comes back and moves to south.
- The lower location (8+1=9) is selected and closed.
- It has no unexplored successors.



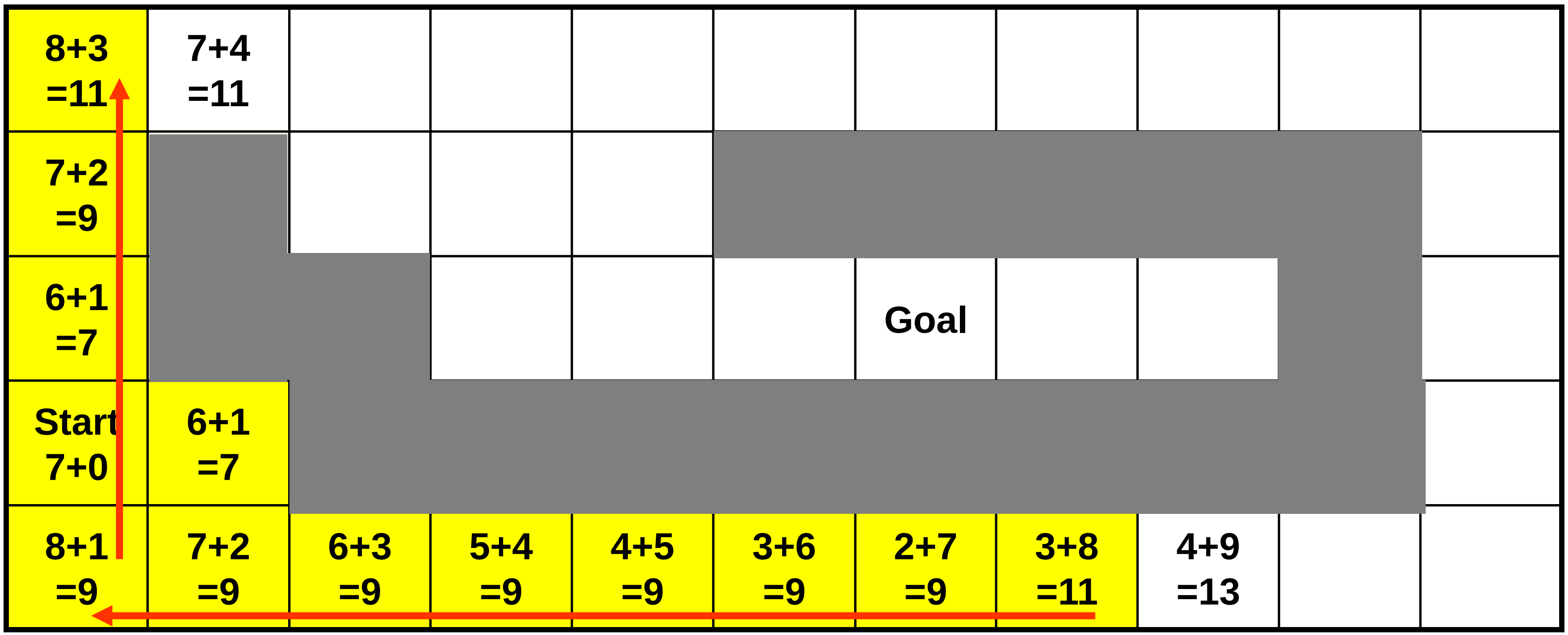
Path Planning- A* Solution

- Search goes to east a few times.
- The lower location (3+8=11) is selected and closed.
- Its successor (4+9=13) is generated and put on OPEN.



Path Planning- A* Solution

- Search comes back and goes to north.
- The upper location (8+3=11) is selected and closed.
- Its successor (7+4=11) is generated and put on OPEN.



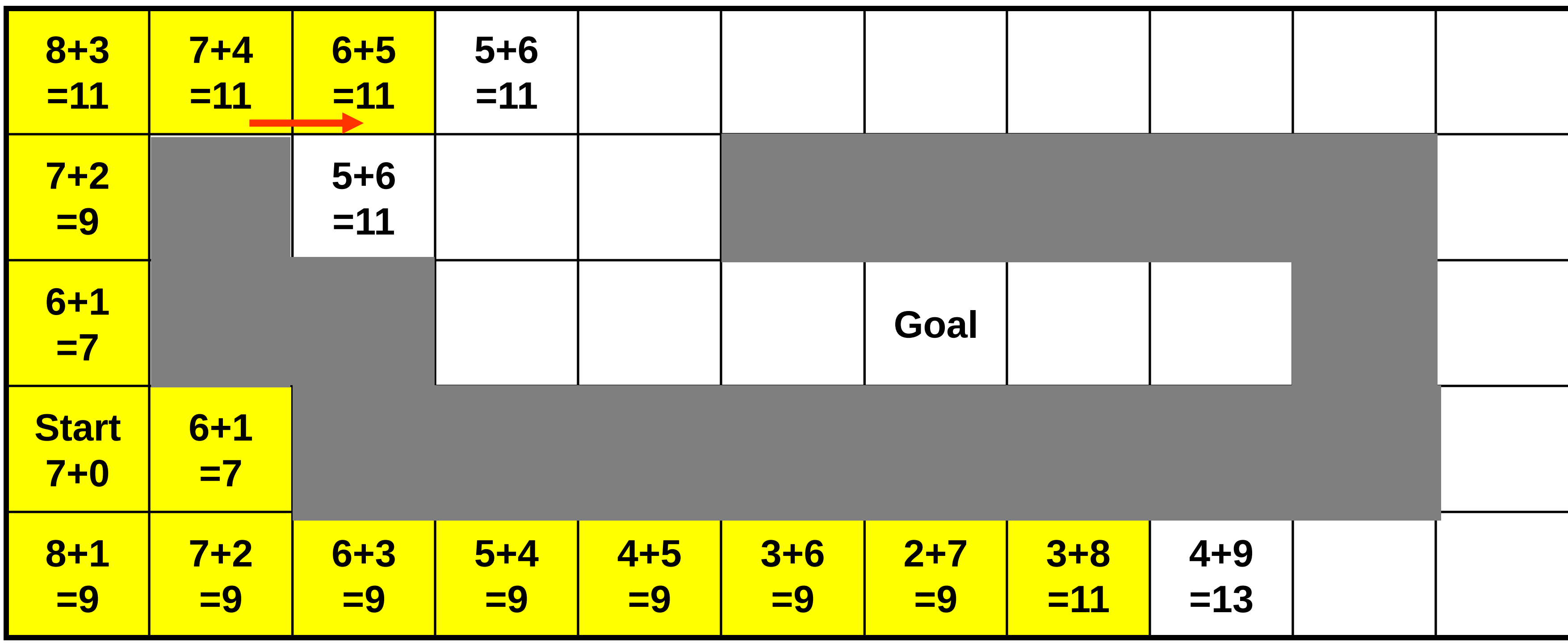
Path Planning- A* Solution

- Search goes to east.
- The location (7+4=11) is selected and closed.
- Its successor (6+5=11) is generated and put on OPEN.

8+3 =11	7+4 =11	6+5 =11								
7+2 =9										
6+1 =7						Goal				
Start 7+0	6+1 =7									
8+1 =9	7+2 =9	6+3 =9	5+4 =9	4+5 =9	3+6 =9	2+7 =9	3+8 =11	4+9 =13		

Path Planning- A* Solution

- Search goes to east.
- The location (6+5=11) is selected and closed.
- Its successors (5+6=11) and (5+6=11) are generated and put on OPEN.



Path Planning- A* Solution

- Search goes to east.
- The upper location (5+6=11) is selected and closed.
- Its successors (4+7=11) and (4+7=11) are generated and put on OPEN.

8+3 =11	7+4 =11	6+5 =11	5+6 =11	4+7 =11						
7+2 =9		5+6 =11	4+7 =11							
6+1 =7						Goal				
Start 7+0	6+1 =7									
8+1 =9	7+2 =9	6+3 =9	5+4 =9	4+5 =9	3+6 =9	2+7 =9	3+8 =11	4+9 =13		

Path Planning- A* Solution

- Search goes to east.
- The upper location (4+7=11) is selected and closed.
- Its successors (3+8=11) and (3+8=11) are generated and put on OPEN.

8+3 =11	7+4 =11	6+5 =11	5+6 =11	4+7 =11	3+8 =11					
7+2 =9		5+6 =11	4+7 =11	3+8 =11						
6+1 =7						Goal				
Start 7+0	6+1 =7									
8+1 =9	7+2 =9	6+3 =9	5+4 =9	4+5 =9	3+6 =9	2+7 =9	3+8 =11	4+9 =13		

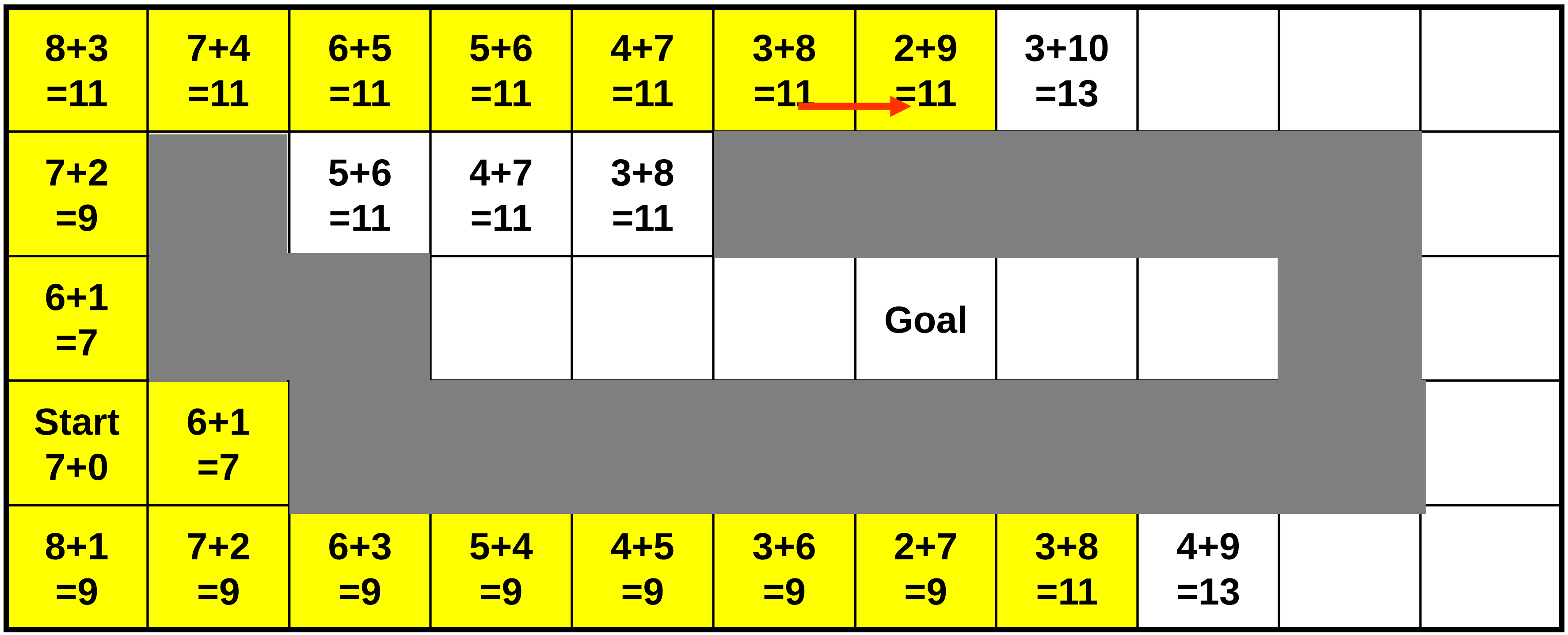
Path Planning- A* Solution

- Search goes to east.
- The upper location (3+8=11) is selected and closed.
- Its successor (2+9=11) is generated and put on OPEN.

8+3 =11	7+4 =11	6+5 =11	5+6 =11	4+7 =11	3+8 =11	2+9 =11				
7+2 =9		5+6 =11	4+7 =11	3+8 =11						
6+1 =7						Goal				
Start 7+0	6+1 =7									
8+1 =9	7+2 =9	6+3 =9	5+4 =9	4+5 =9	3+6 =9	2+7 =9	3+8 =11	4+9 =13		

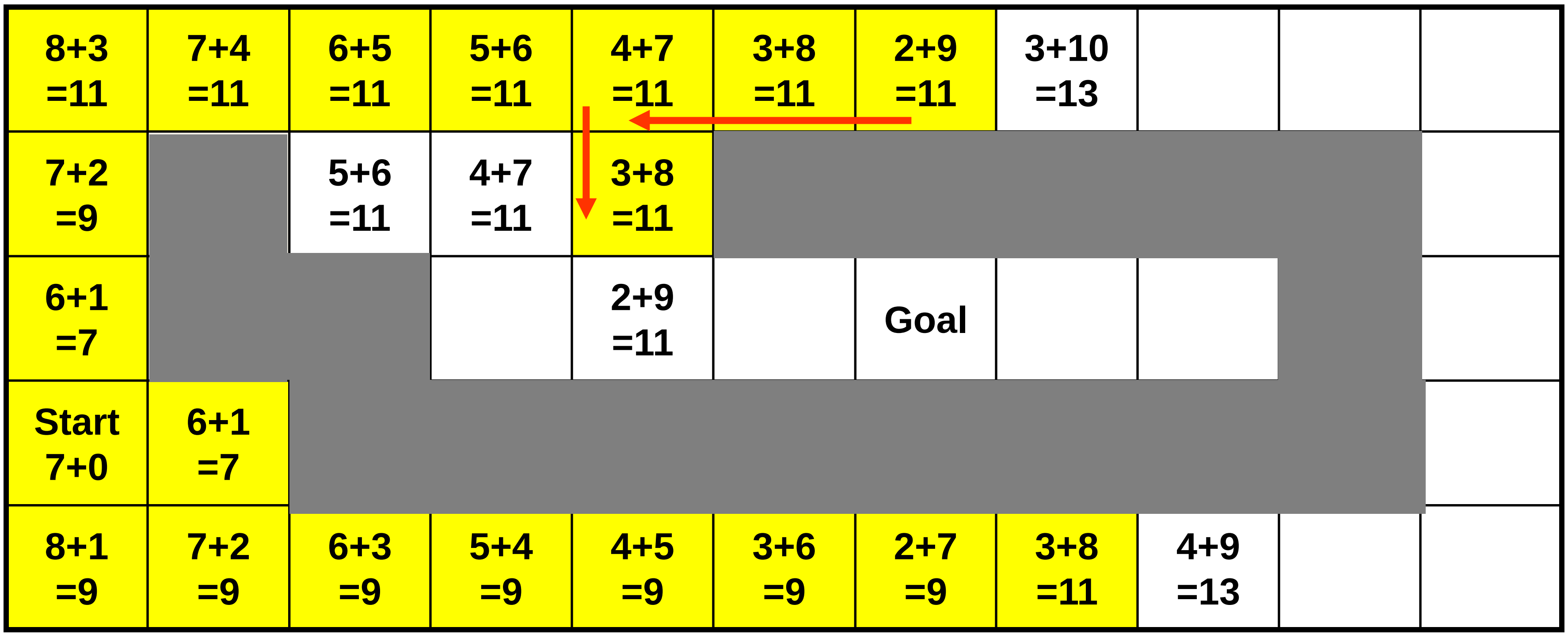
Path Planning- A* Solution

- Search goes to east.
- The location (2+9=11) is selected and closed.
- Its successor (3+10=13) is generated and put on OPEN.



Path Planning- A* Solution

- Search comes back and goes to south.
- The lower location (3+8=11) is selected and closed.
- Its successor (2+9=11) is generated and put on OPEN.



Path Planning- A* Solution

- Search goes to south.
- The lower location (2+9=11) is selected and closed.
- Its successors (3+10=13) and (1+10=11) are generated and put on OPEN.

8+3 =11	7+4 =11	6+5 =11	5+6 =11	4+7 =11	3+8 =11	2+9 =11	3+10 =13			
7+2 =9		5+6 =11	4+7 =11	3+8 =11						
6+1 =7			3+10 =13	2+9 =11	1+10 =11	Goal				
Start 7+0	6+1 =7									
8+1 =9	7+2 =9	6+3 =9	5+4 =9	4+5 =9	3+6 =9	2+7 =9	3+8 =11	4+9 =13		

Path Planning- A* Solution

- Search goes to east.
- The lower location (1+10=11) is selected and closed.
- Its successor (0+11=11) isgenerated and put on OPEN.

8+3 =11	7+4 =11	6+5 =11	5+6 =11	4+7 =11	3+8 =11	2+9 =11	3+10 =13			
7+2 =9		5+6 =11	4+7 =11	3+8 =11						
6+1 =7			3+10 =13	2+9 =11	1+10 =11	Goal 0+11				
Start 7+0	6+1 =7									
8+1 =9	7+2 =9	6+3 =9	5+4 =9	4+5 =9	3+6 =9	2+7 =9	3+8 =11	4+9 =13		

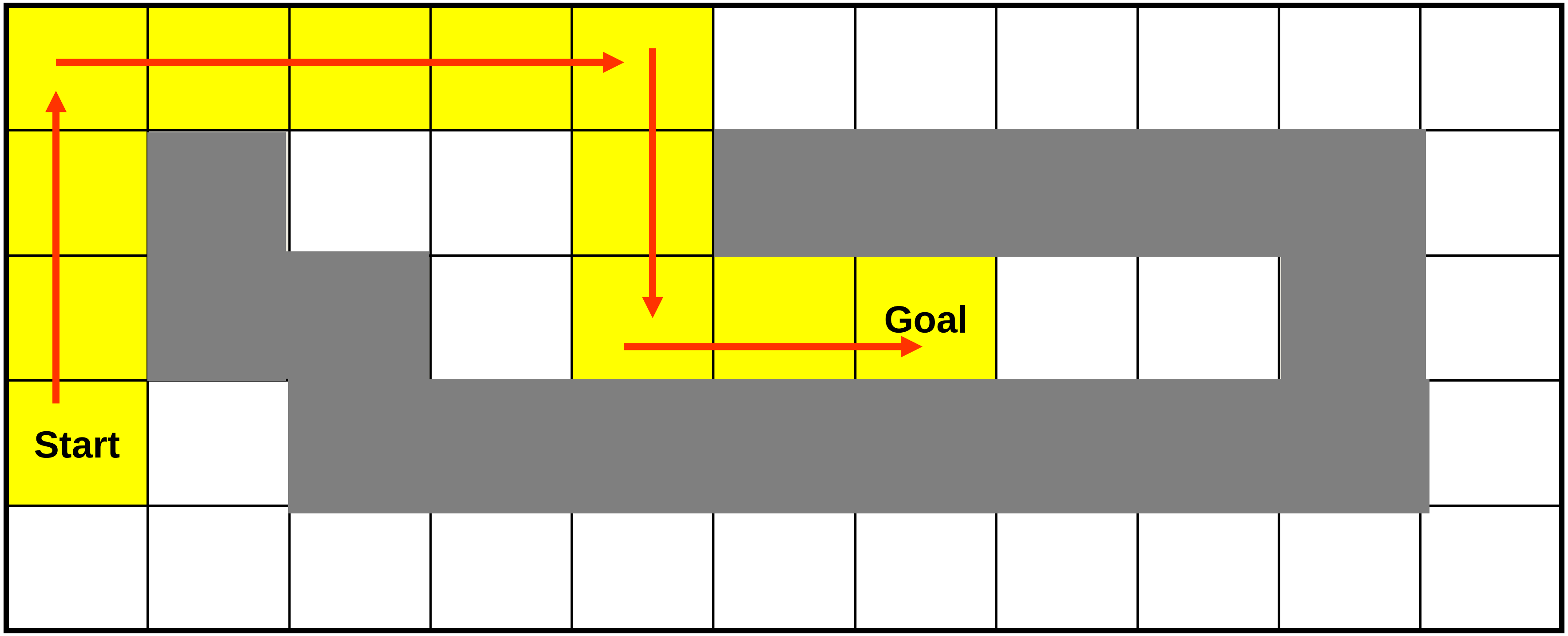
Path Planning- A* Solution

- Search goes to east.
- Search ends here, because the goal location is found and there is not any other node with a lower f(n) value.

8+3 =11	7+4 =11	6+5 =11	5+6 =11	4+7 =11	3+8 =11	2+9 =11	3+10 =13			
7+2 =9		5+6 =11	4+7 =11	3+8 =11						
6+1 =7			3+10 =13	2+9 =11	1+10 =11	Goal 0+11				
Start 7+0	6+1 =7									
8+1 =9	7+2 =9	6+3 =9	5+4 =9	4+5 =9	3+6 =9	2+7 =9	3+8 =11	4+9 =13		

Path Planning- A* Solution

- Path found by A* search
- Path length=11 (shortest path)



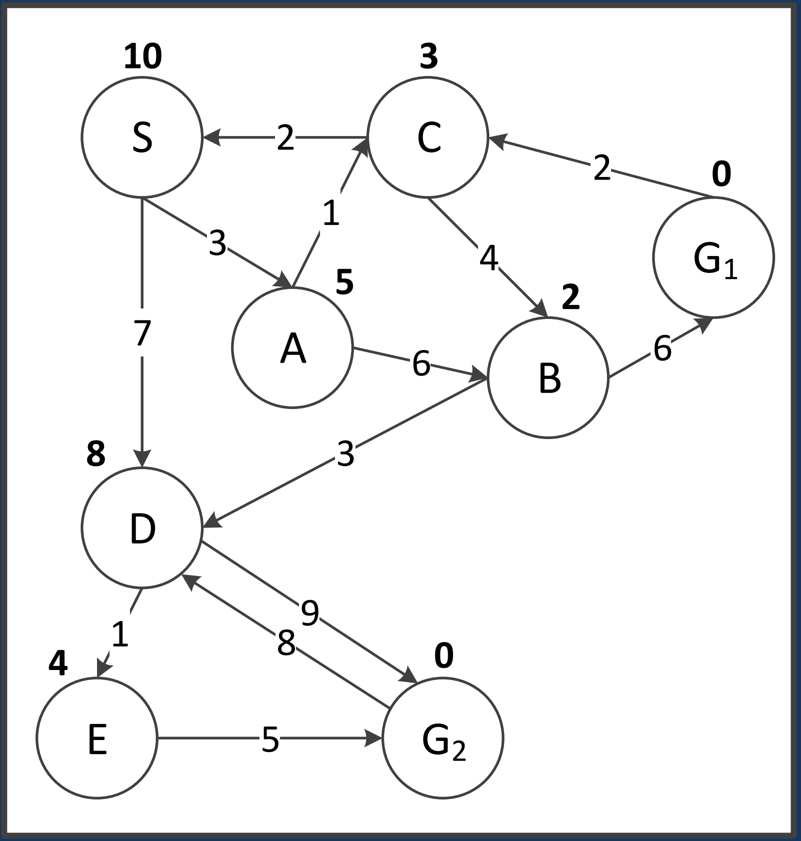
Exercise: Missionaries and Cannibals

- The missionaries and cannibals problem is stated as follows: Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side, without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.

- This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint (Amarel, 1968).

A* Search Example

- A state space is given on the right where the initial state is **S** and the goal states are **G$_1$** and **G$_2$**.
- The possible actions between states are indicated by arrows where the number labeling each arrow is the actual cost of the action.
- The number in **bold** near each state is the value of the heuristic function **h** at that state.



A* Search Example

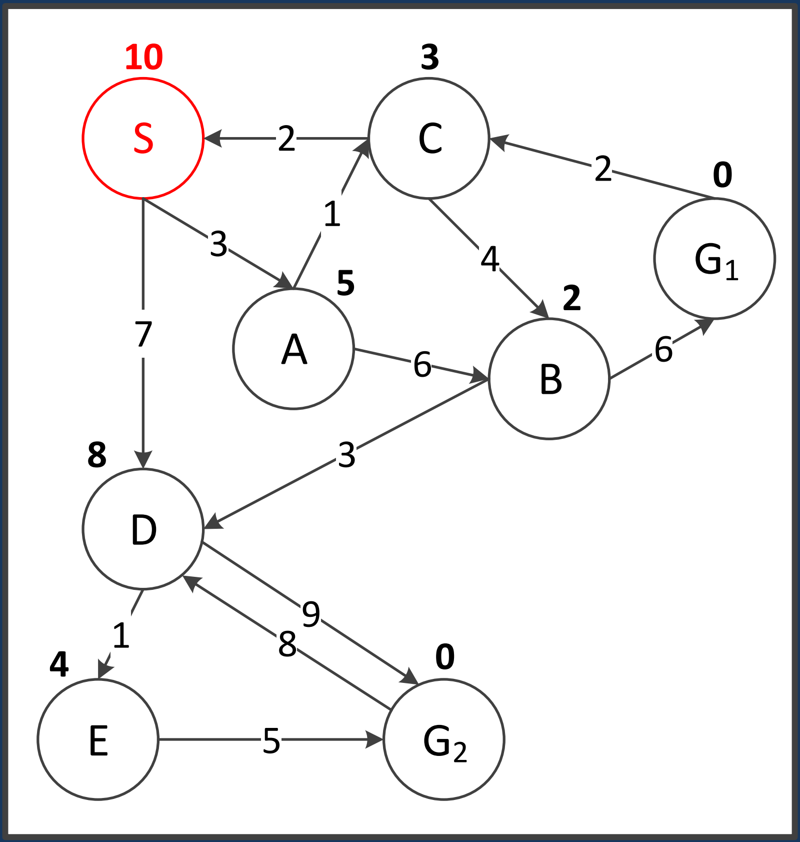
- Fill in the given table by using A* algorithm.
- Assume that the algorithm does not check if a state is revisited (hence there may be several nodes with the same state in the search tree).

n	***state***	***g(n)***	***h(n)***	***f(n)***	***#exp***
1	S				1
2					
…					

- The states produced by the successor function are always ordered in alphabetic order.
- In the rightmost column (***#exp***), indicate the order in which nodes are expanded (i.e., are removed from the fringe). If a node is not expanded, leave the corresponding cell empty.

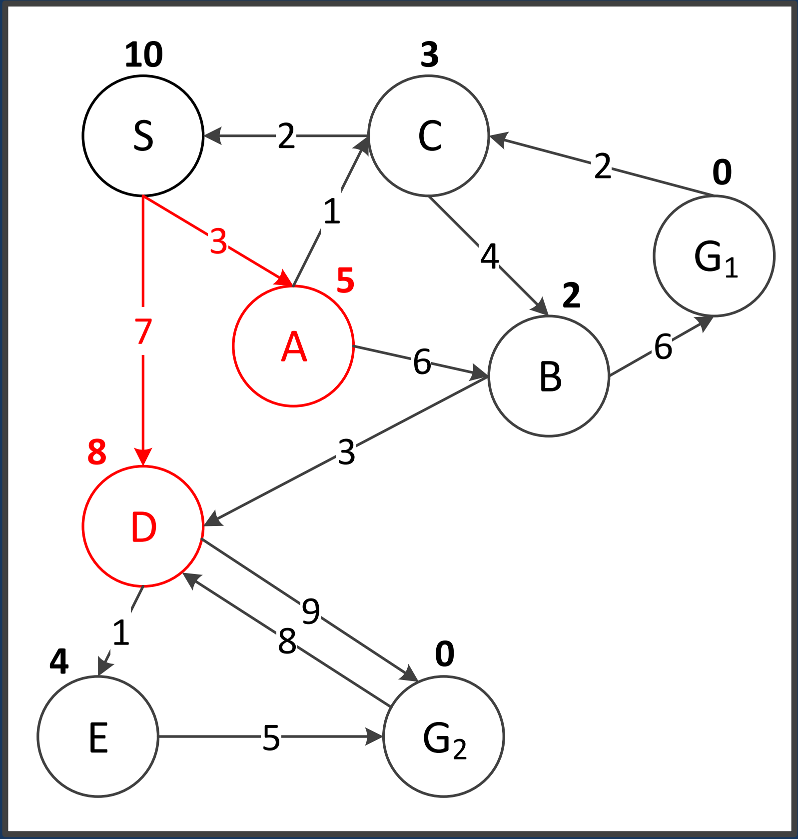
A* Search Example

n	state	g(n)	h(n)	f(n)	#exp
1	S	0	10	10	1
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					



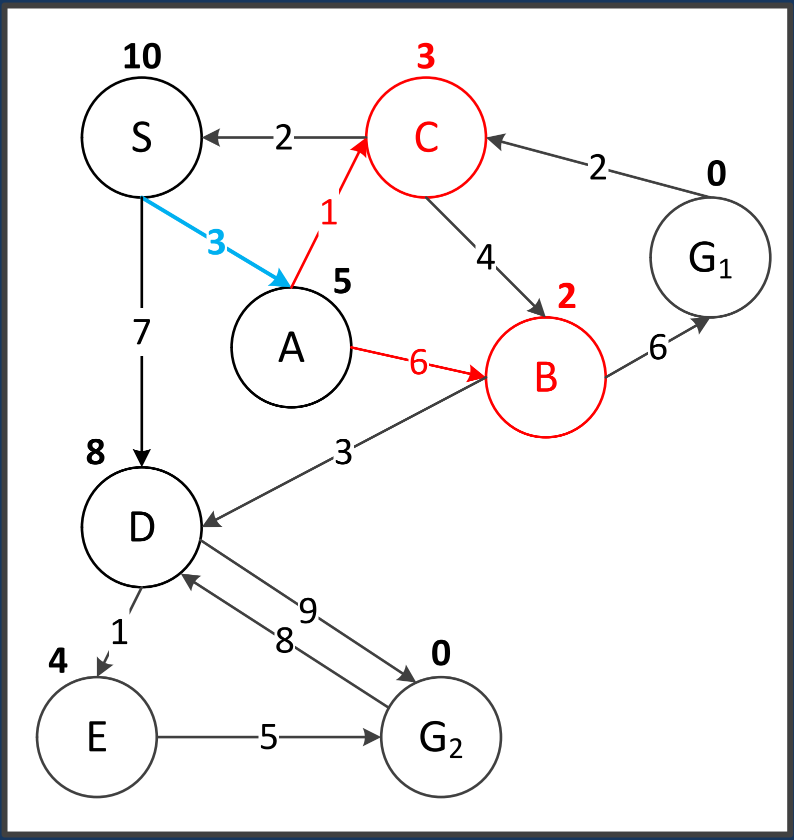
A* Search Example

n	state	g(n)	h(n)	f(n)	#exp
1	S	0	10	10	1
2	A(←S)	3	5	8	2
3	D(←S)	7	8	15	
4					
5					
6					
7					
8					
9					
10					
11					



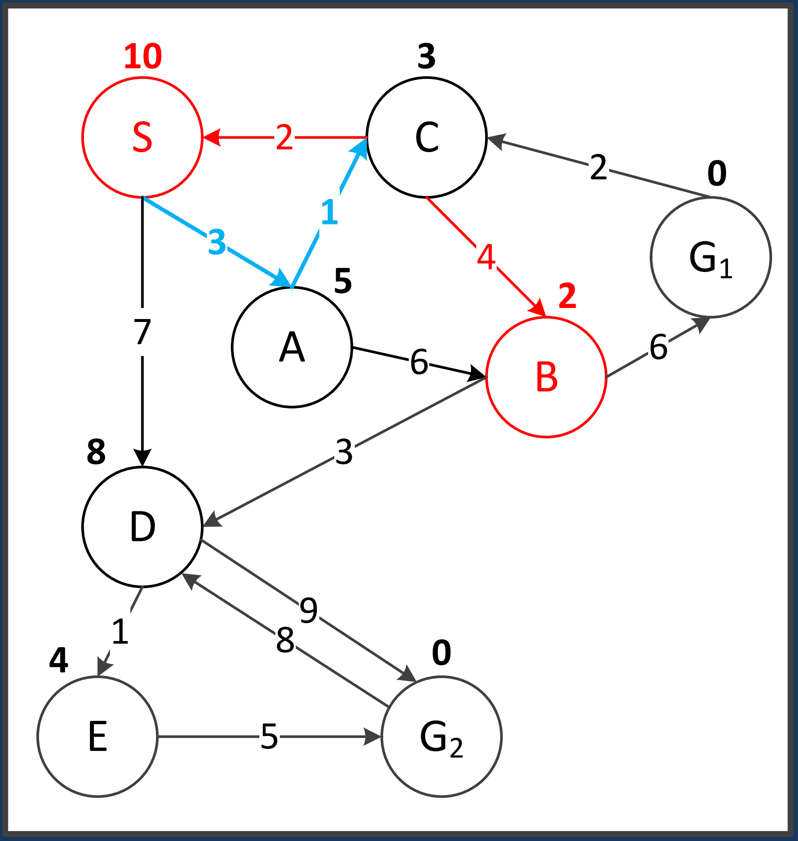
A* Search Example

n	state	g(n)	h(n)	f(n)	#exp
1	S	0	10	10	1
2	A(←S)	3	5	8	2
3	D(←S)	7	8	15	
4	B(←A)	9	2	11	
5	C(←A)	4	3	7	3
6					
7					
8					
9					
10					
11					



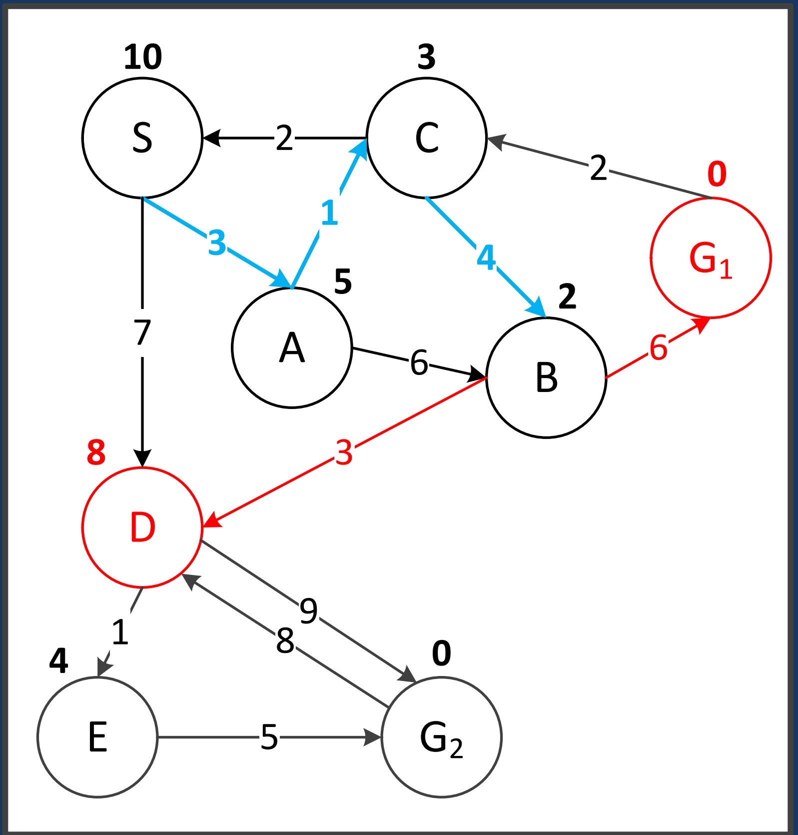
A* Search Example

n	state	g(n)	h(n)	f(n)	#exp
1	S	0	10	10	1
2	A(←S)	3	5	8	2
3	D(←S)	7	8	15	
4	B(←A)	9	2	11	
5	C(←A)	4	3	7	3
6	B(←C)	8	2	10	4
7	S(←C)	6	10	16	
8					
9					
10					
11					



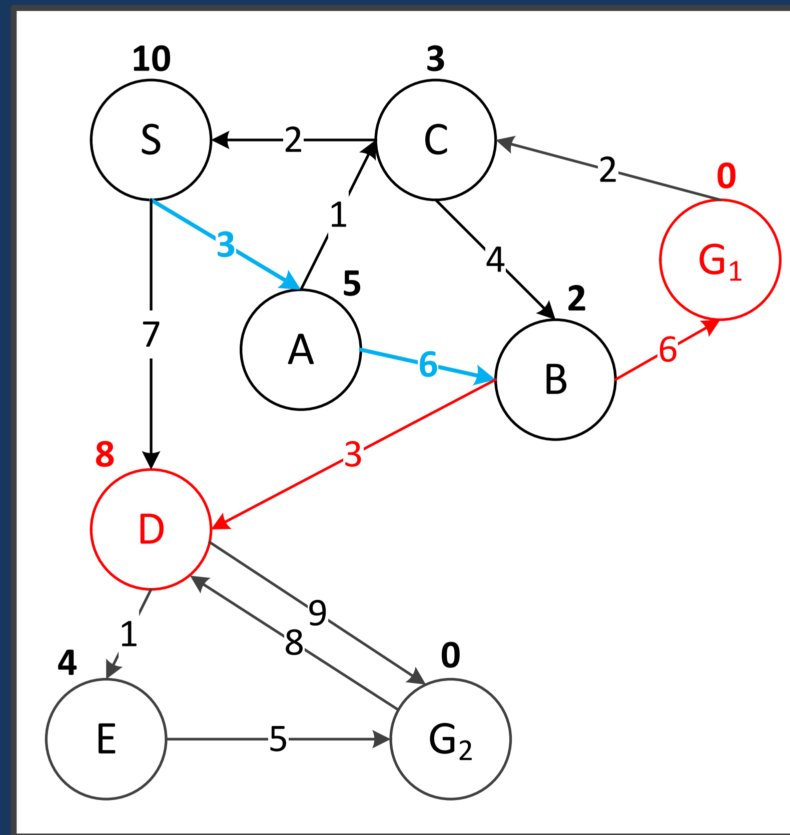
A* Search Example

n	state	g(n)	h(n)	f(n)	#exp
1	S	0	10	10	1
2	A(←S)	3	5	8	2
3	D(←S)	7	8	15	
4	B(←A)	9	2	11	5
5	C(←A)	4	3	7	3
6	B(←C)	8	2	10	4
7	S(←C)	6	10	16	
8	D(←B)	11	8	19	
9	G_1(←B)	14	0	14	
10					
11					



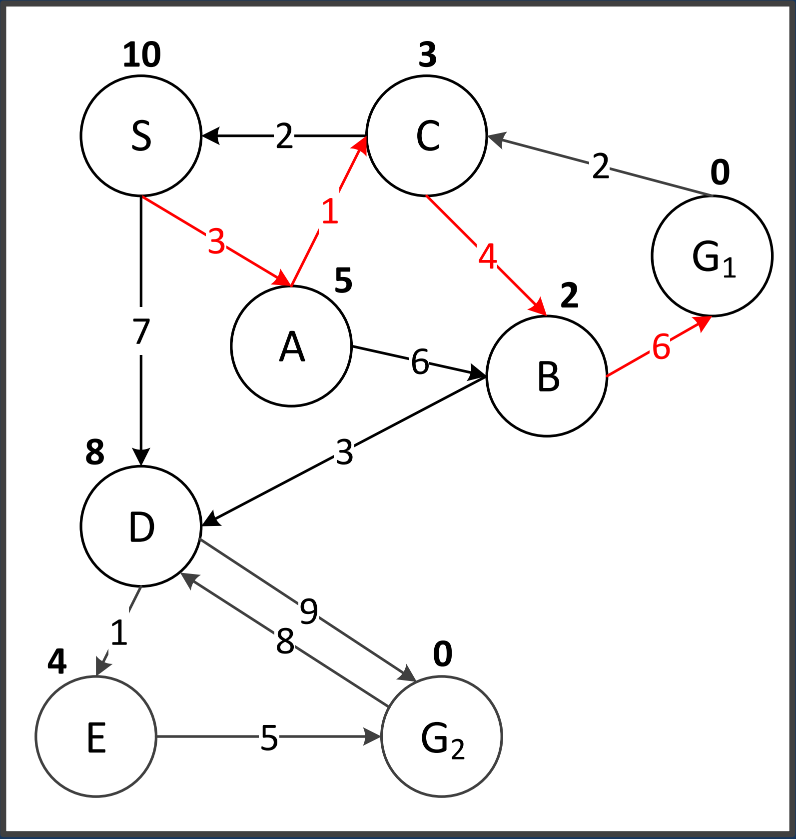
A* Search Example

n	state	g(n)	h(n)	f(n)	#exp
1	S	0	10	10	1
2	A(←S)	3	5	8	2
3	D(←S)	7	8	15	
4	B(←A)	9	2	11	5
5	C(←A)	4	3	7	3
6	B(←C)	8	2	10	4
7	S(←C)	6	10	16	
8	D(←B)	11	8	19	
9	G_1(←B)	14	0	14	6
10	D(←B)	12	8	20	
11	G_1(←B)	15	0	15	



A* Search Example

n	state	g(n)	h(n)	f(n)	#exp
1	S	0	10	10	1
2	A(←S)	3	5	8	2
3	D(←S)	7	8	15	
4	B(←A)	9	2	11	5
5	C(←A)	4	3	7	3
6	B(←C)	8	2	10	4
7	S(←C)	6	10	16	
8	D(←B)	11	8	19	
9	G_1(←B)	14	0	14	6
10	D(←B)	12	8	20	
11	G_1(←B)	15	0	15	



A* Search Example

- **Q:** Is the route you have found optimal? Please discuss briefly.
- **A:** It's not optimal (cost of the path found=14 > cost of the optimal path=13). The reason is that the used heuristic function is not admissible (overestimation for D).