A*

A path finding algorithm.

A path finding algorithm.

- Given a state space, such as a 2-dimensional map, find a path from point A to point B in that space, if such a path exists.
- If such a path exists, return a path within certain criteria - ie: shortest path, most straight path, avoiding certain areas, etc.

The A* Algorithm

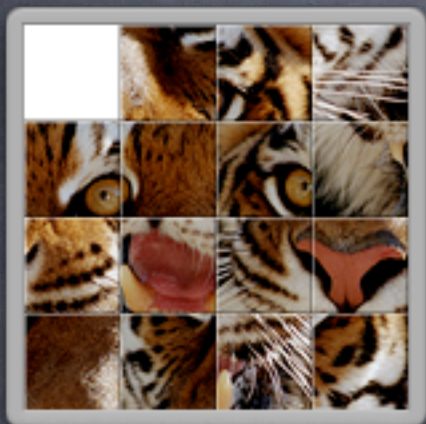
- Developed in 1968 for solving different kind of problems such as the ‘15-puzzle’.
- Searches for the least costly path from a starting state to a goal state by examining adjacent states of a particular state.

The A* Algorithm

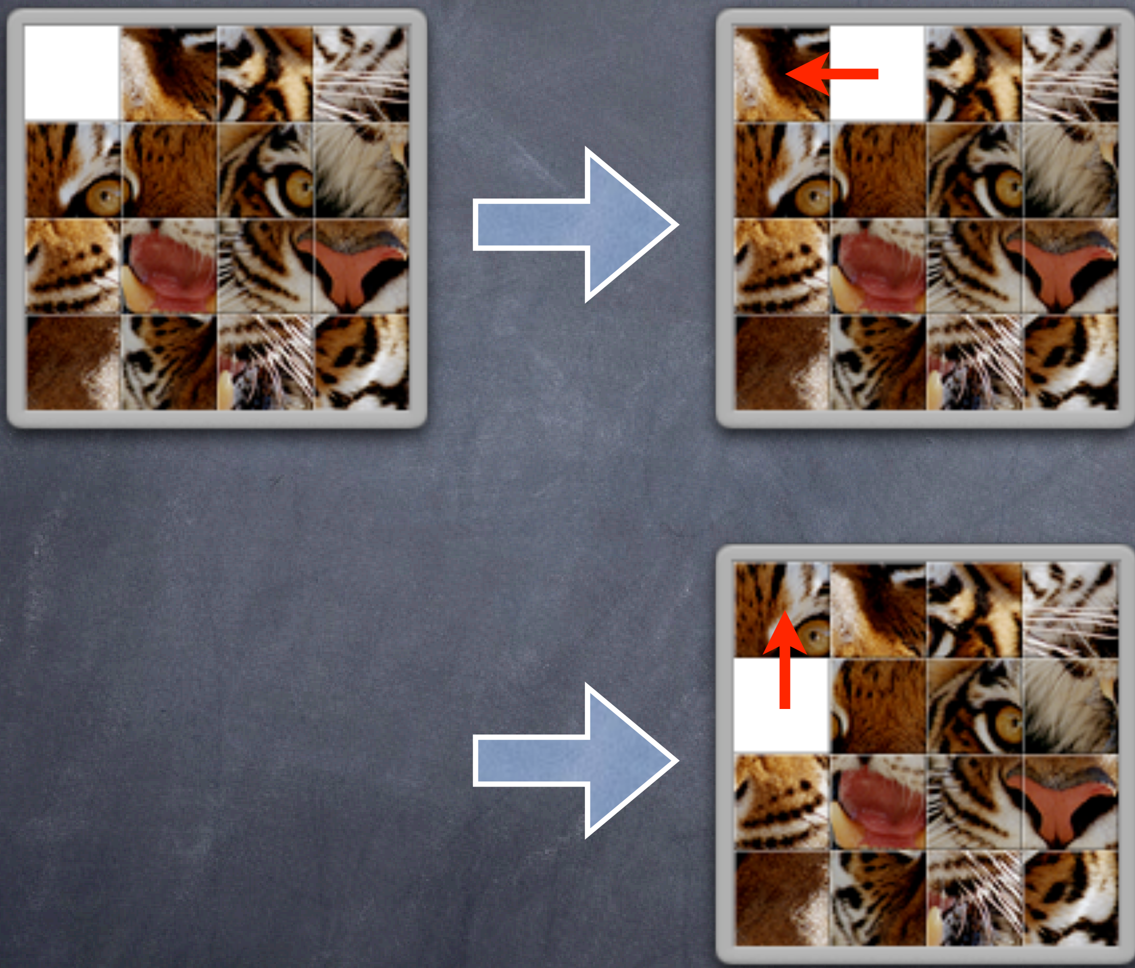
- Developed in 1968 for solving different kind of problems such as the '15-puzzle'.

- Searches for the least costly path from a starting state to a goal state by examining adjacent states of a particular state.

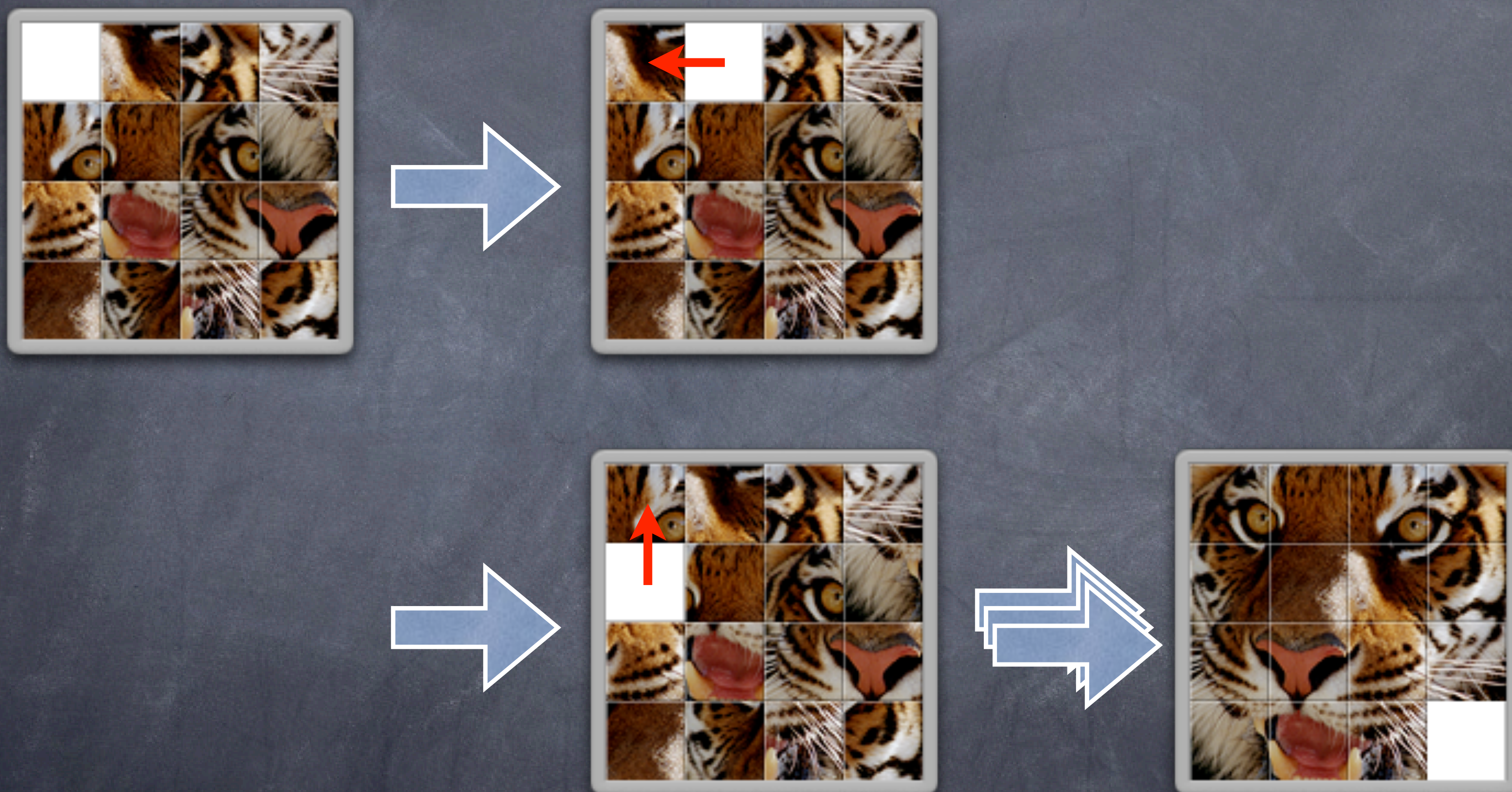
Starting State Adjacent States Goal State



Starting State | Adjacent States | Goal State

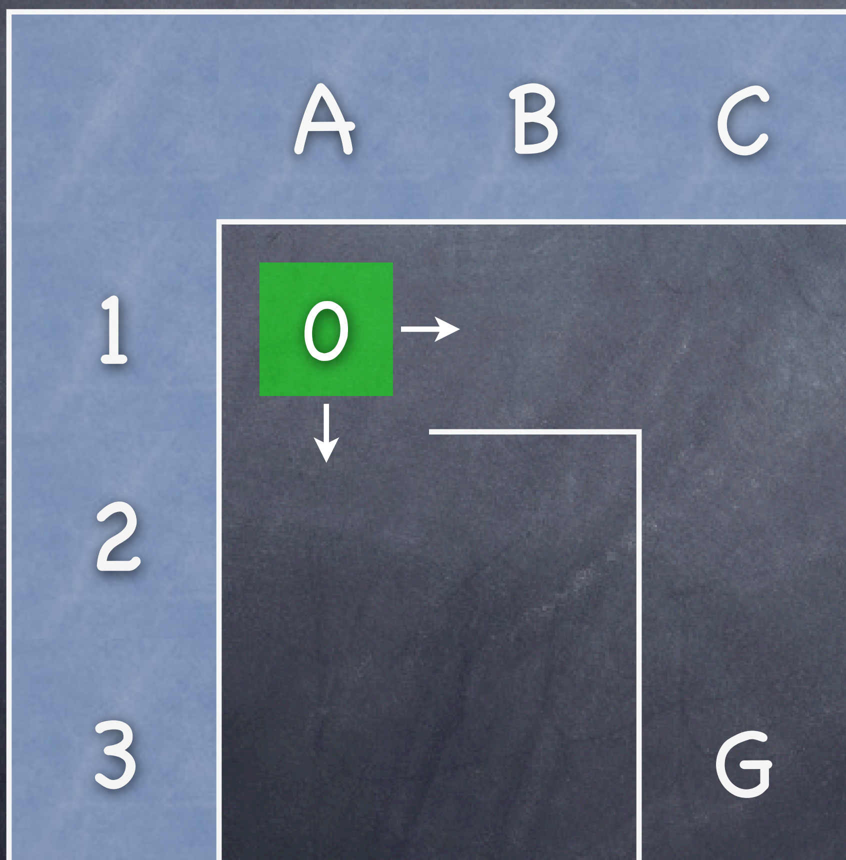


Starting State Adjacent States Goal State



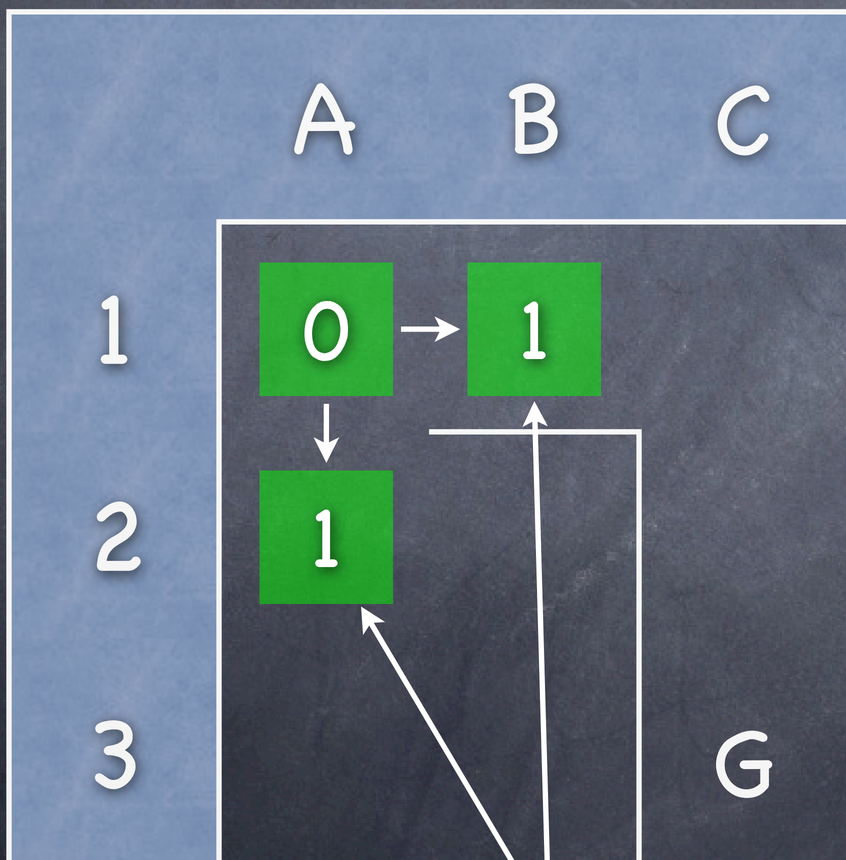
- The algorithm works by repeatedly examining the most promising unexplored adjacent state.
- A priority queue '**Open**' contains all adjacent unexamined states, sorted in order of lowest cost.
- A list '**Closed**' contains all examined states.
- Initially, the **Closed** list is empty, while the **Open** list contains a single starting state.

A Simple Example



Open	Closed
A1 (0)	

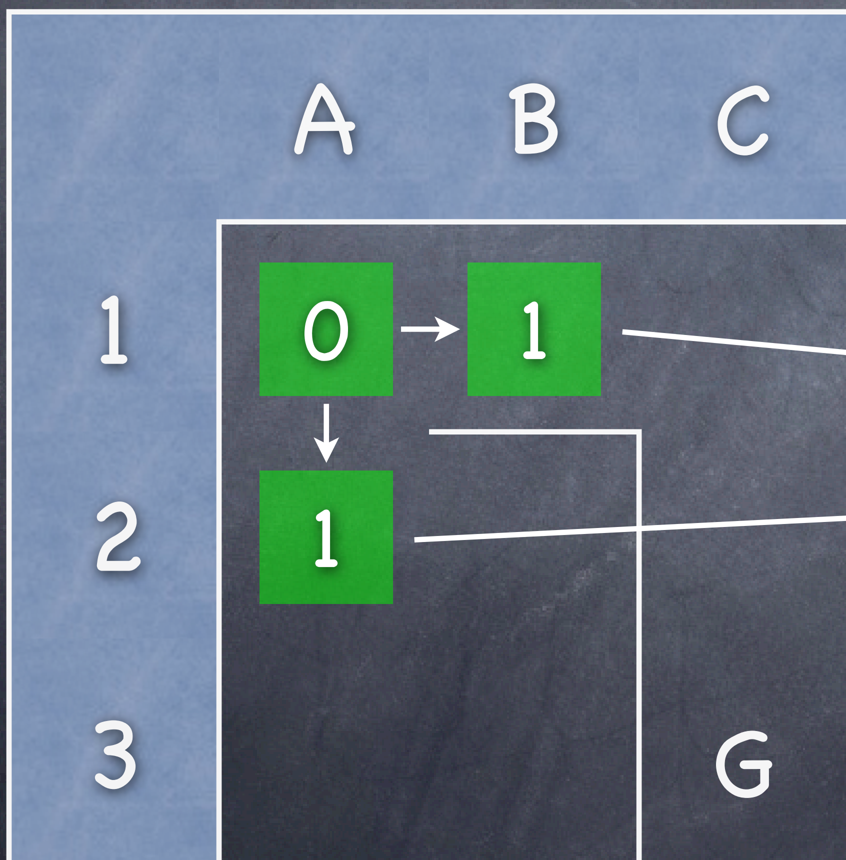
A Simple Example



Open	Closed
A1 (0)	

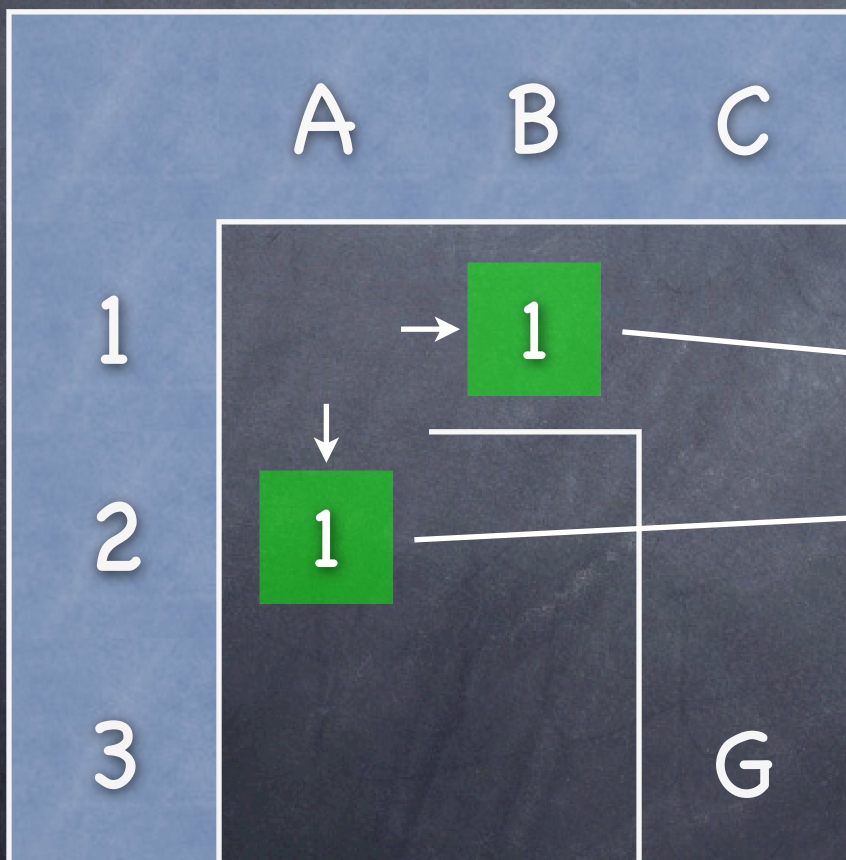
Adjacent States (1 move from start)

A Simple Example



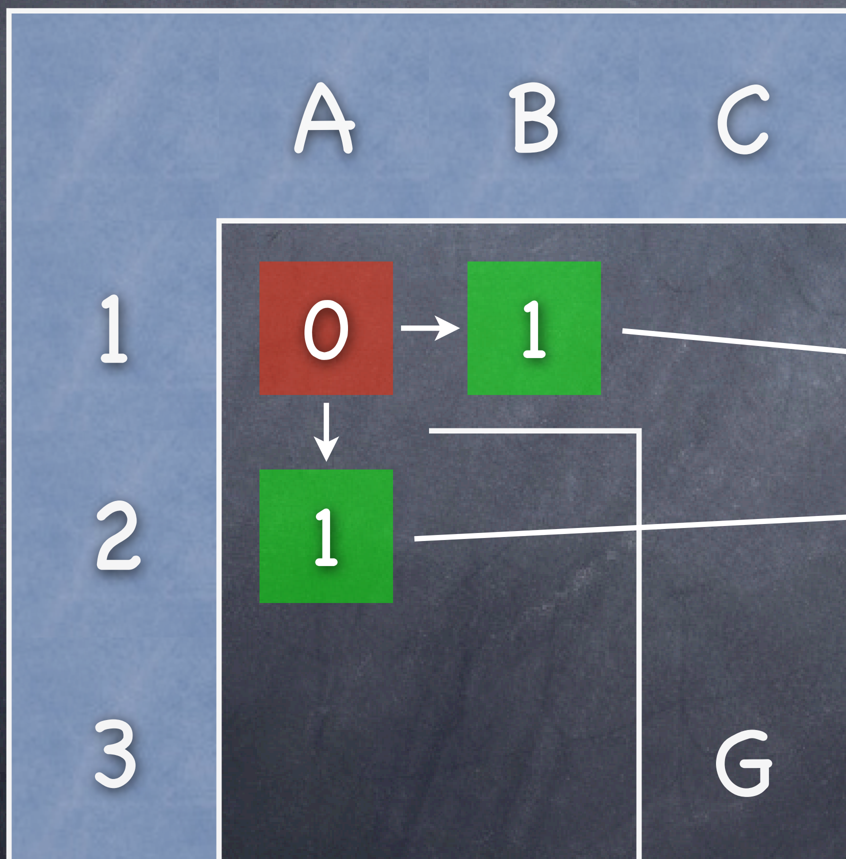
Open	Closed
A1 (0)	
B1 (1)	
A2 (1)	

A Simple Example



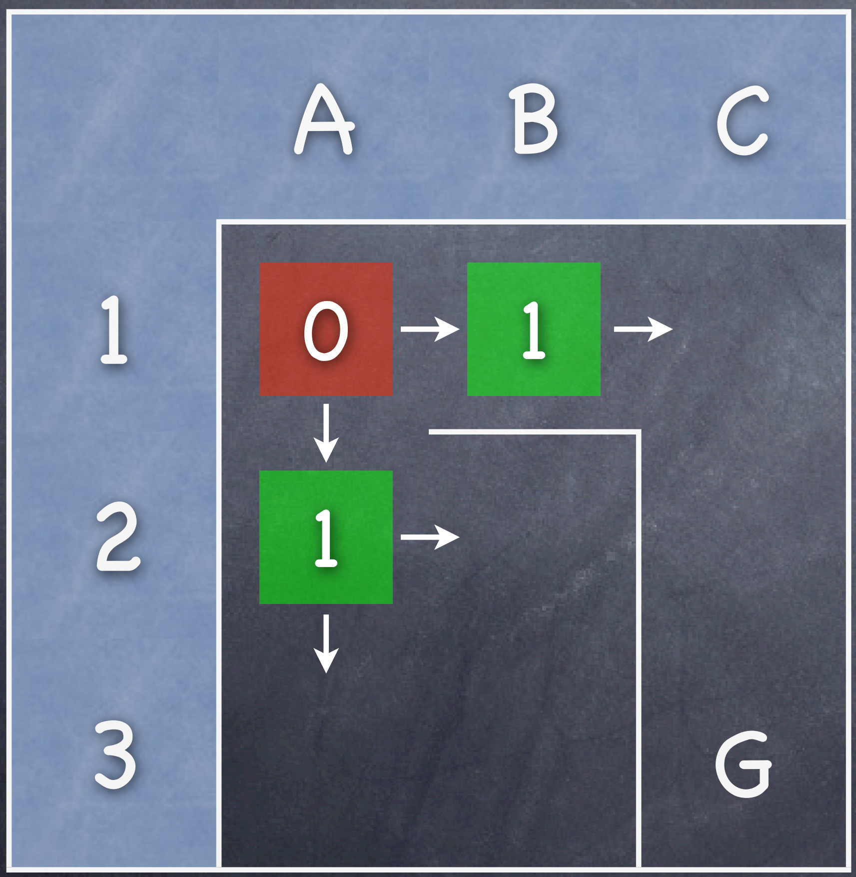
Open	Closed
B1 (1)	
A2 (1)	

A Simple Example



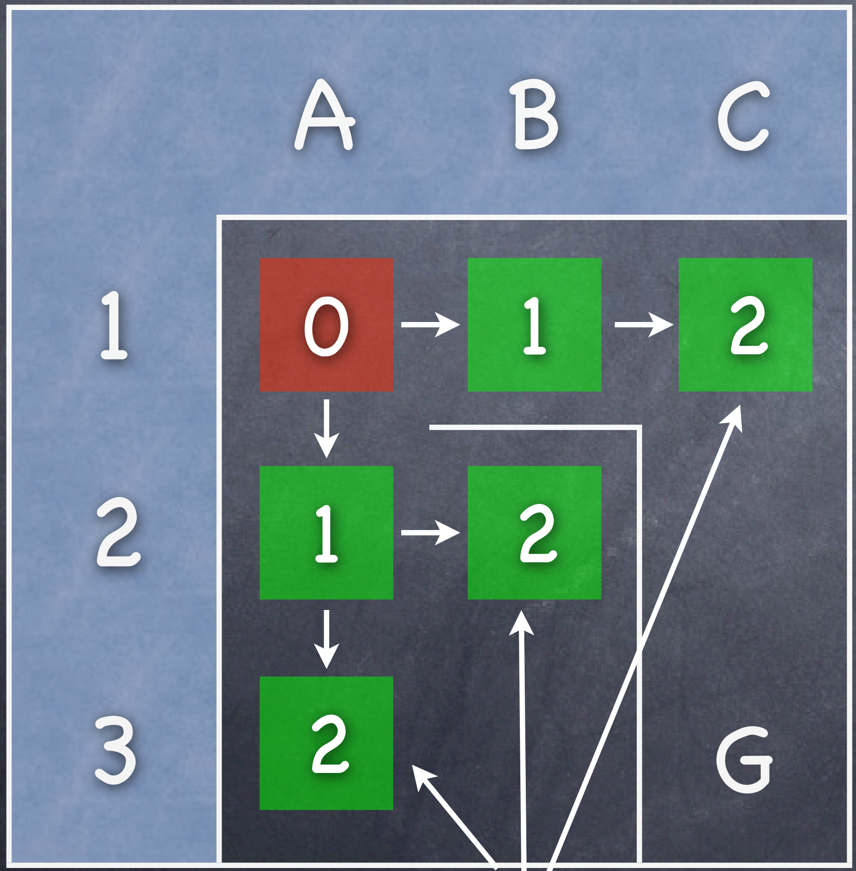
Open	Closed
	A1 (0)
B1 (1)	
A2 (1)	

A Simple Example



Open	Closed
B1 (1)	A1 (0)
A2 (1)	

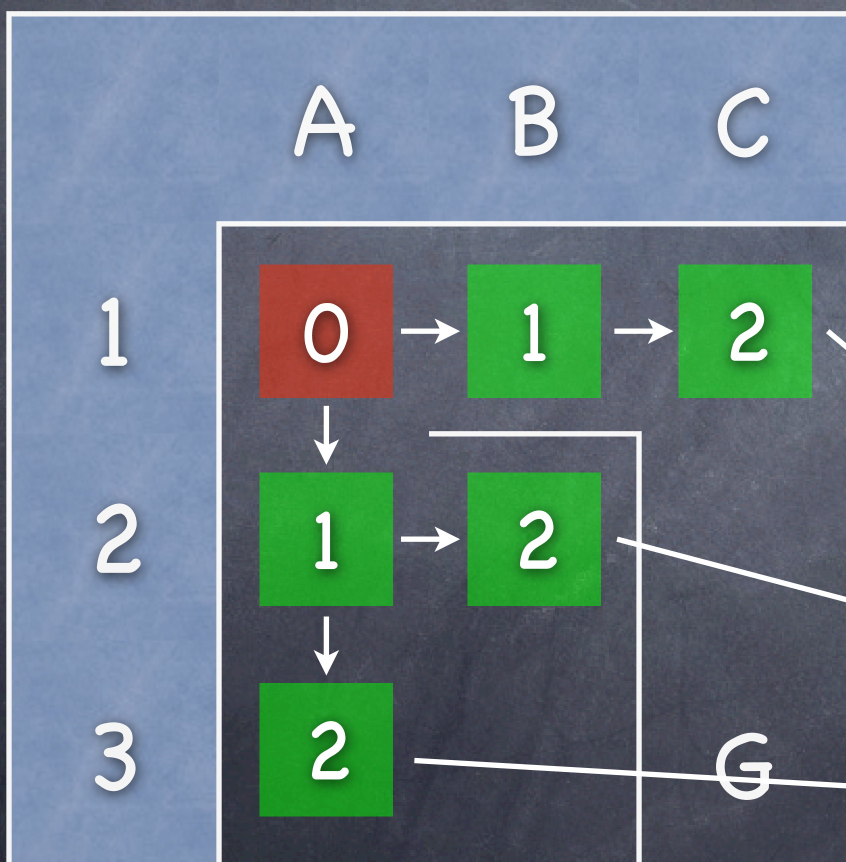
A Simple Example



Open	Closed
B1 (1)	A1 (0)
A2 (1)	

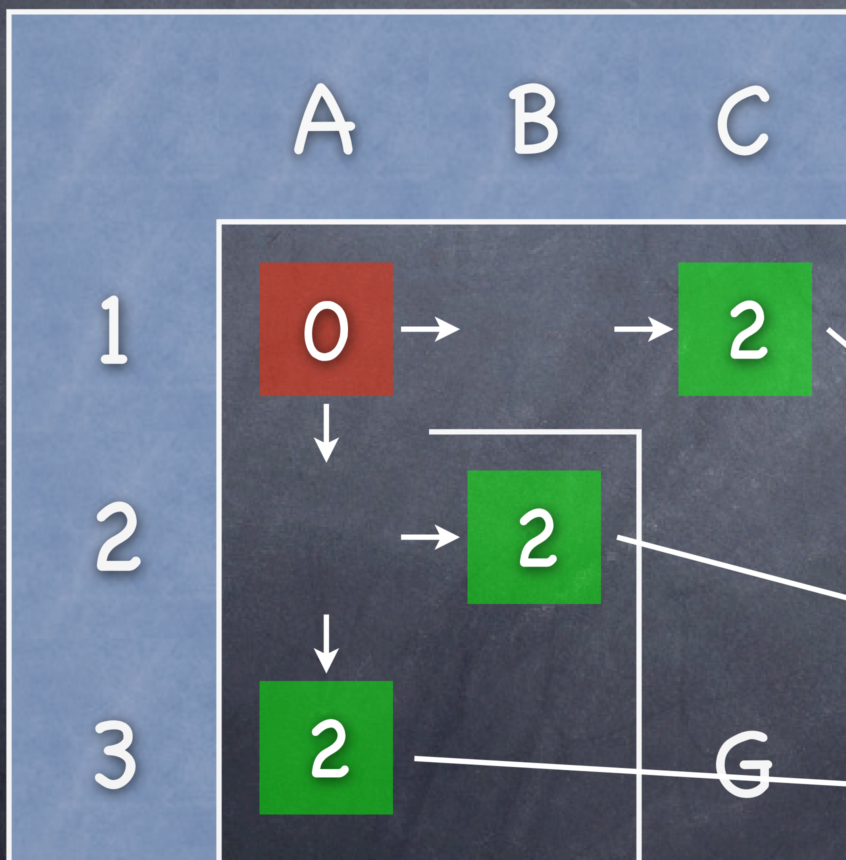
Adjacent States (2 moves from start)

A Simple Example



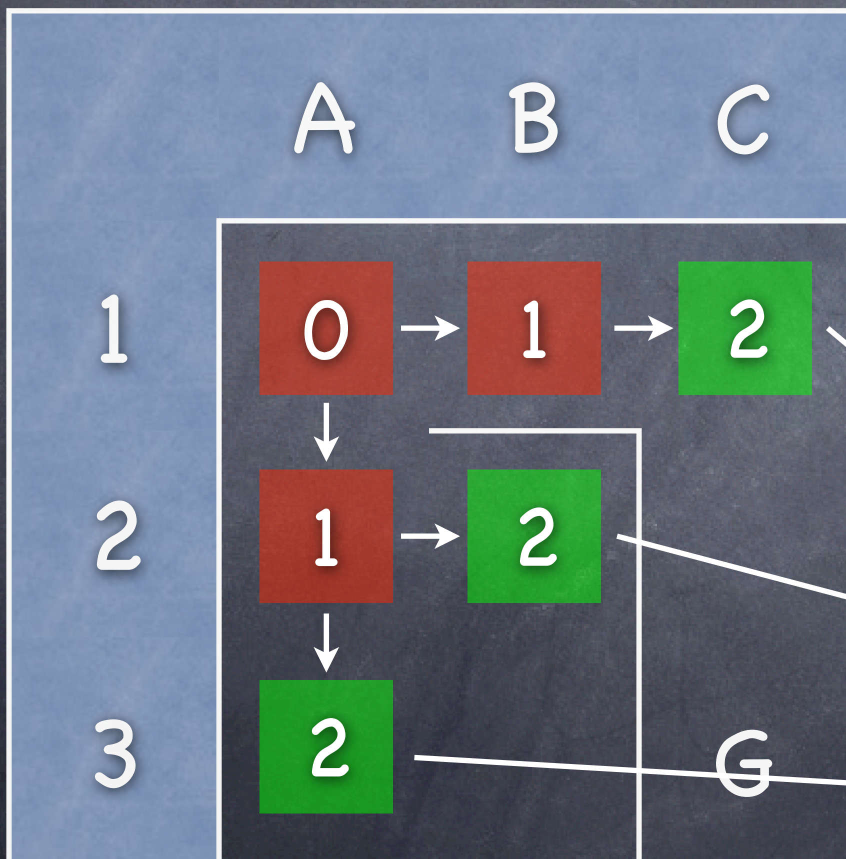
Open	Closed
B1 (1)	A1 (0)
A2 (1)	
C1 (2)	
B2 (2)	
A3 (2)	

A Simple Example



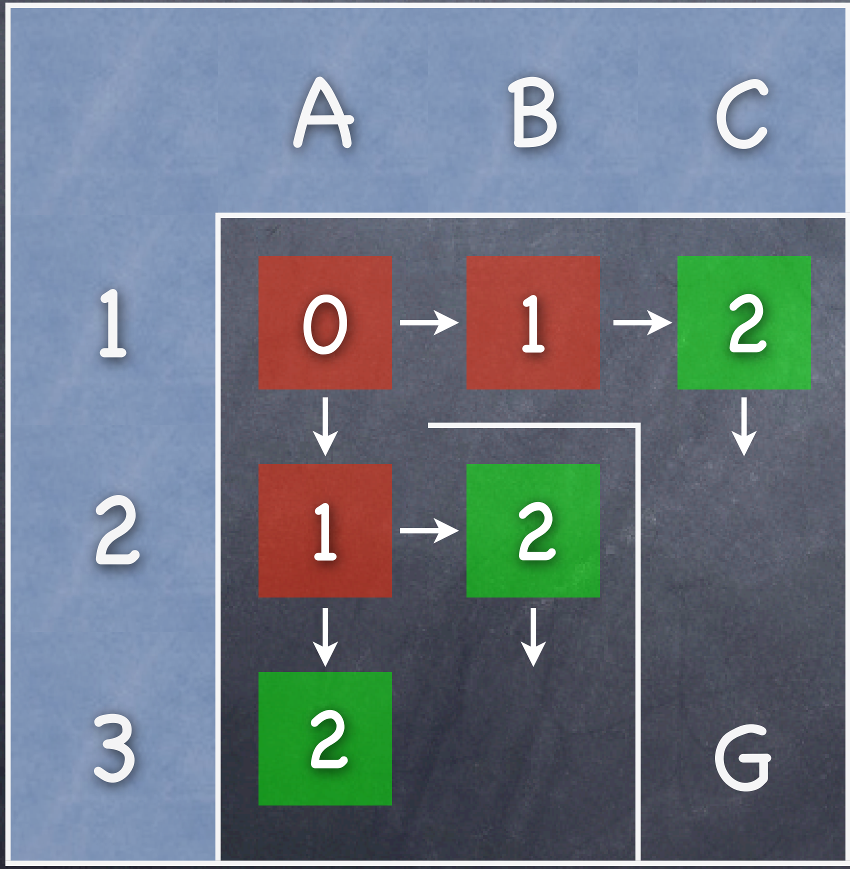
Open	Closed
	A1 (0)
C1 (2)	
B2 (2)	
A3 (2)	

A Simple Example



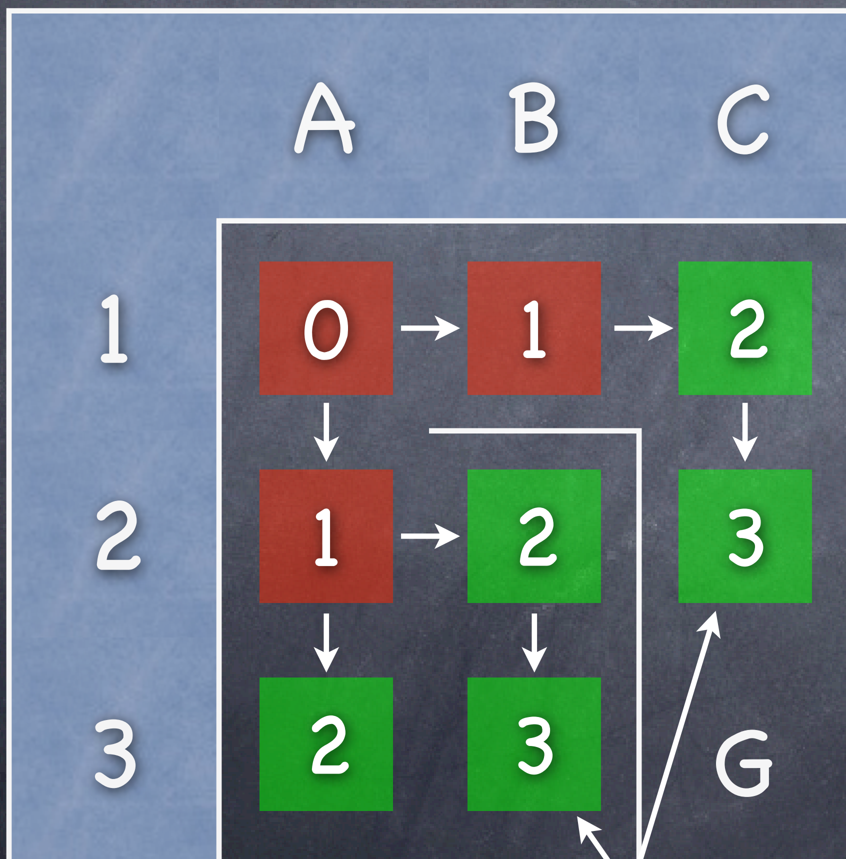
Open	Closed
	A1 (0)
	B1 (1)
C1 (2)	A2 (1)
B2 (2)	
A3 (2)	

A Simple Example



Open	Closed
C1 (2)	A1 (0)
B2 (2)	B1 (1)
A3 (2)	A2 (1)

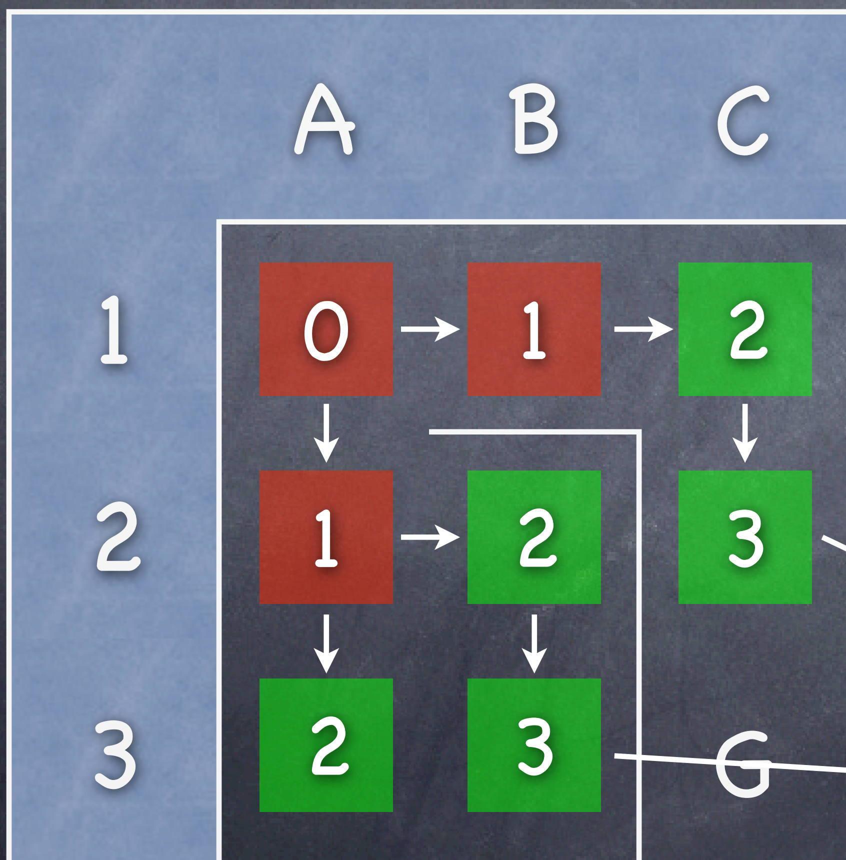
A Simple Example



Open	Closed
C1 (2)	A1 (0)
B2 (2)	B1 (1)
A3 (2)	A2 (1)

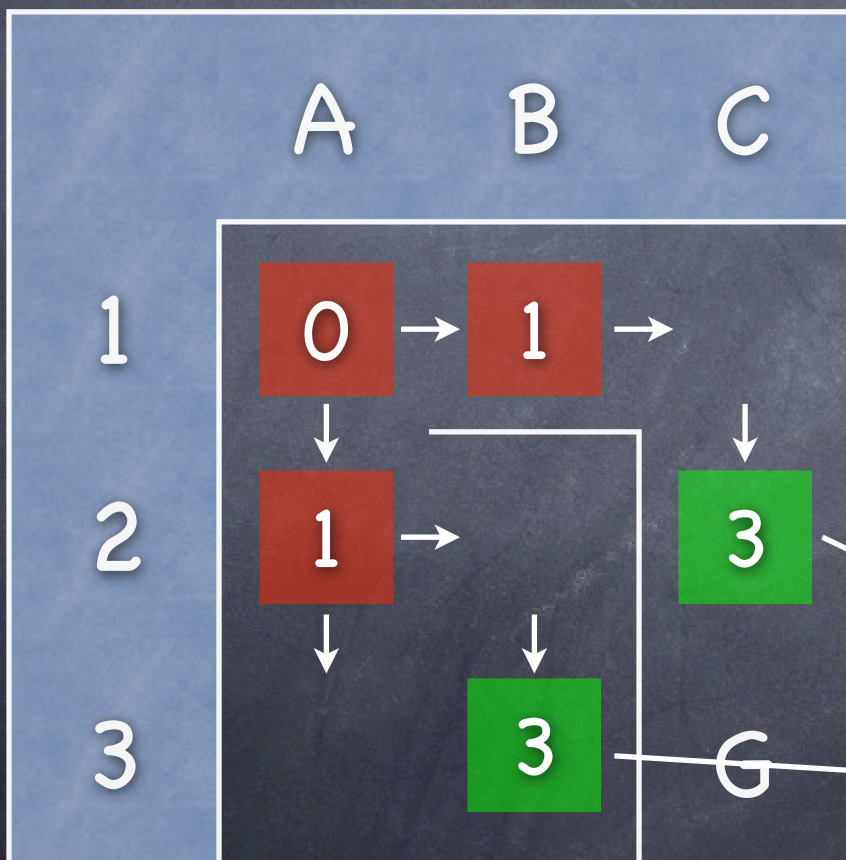
Adjacent States (3 move from start)

A Simple Example



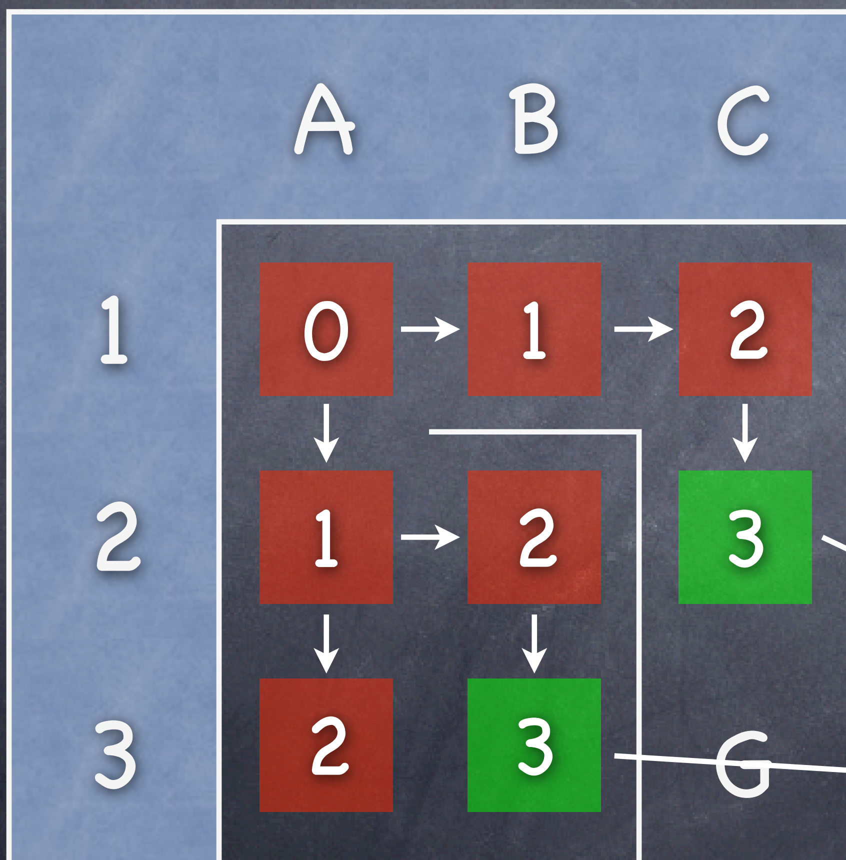
Open	Closed
C1 (2)	A1 (0)
B2 (2)	B1 (1)
A3 (2)	A2 (1)
C2 (3)	
B3 (3)	

A Simple Example



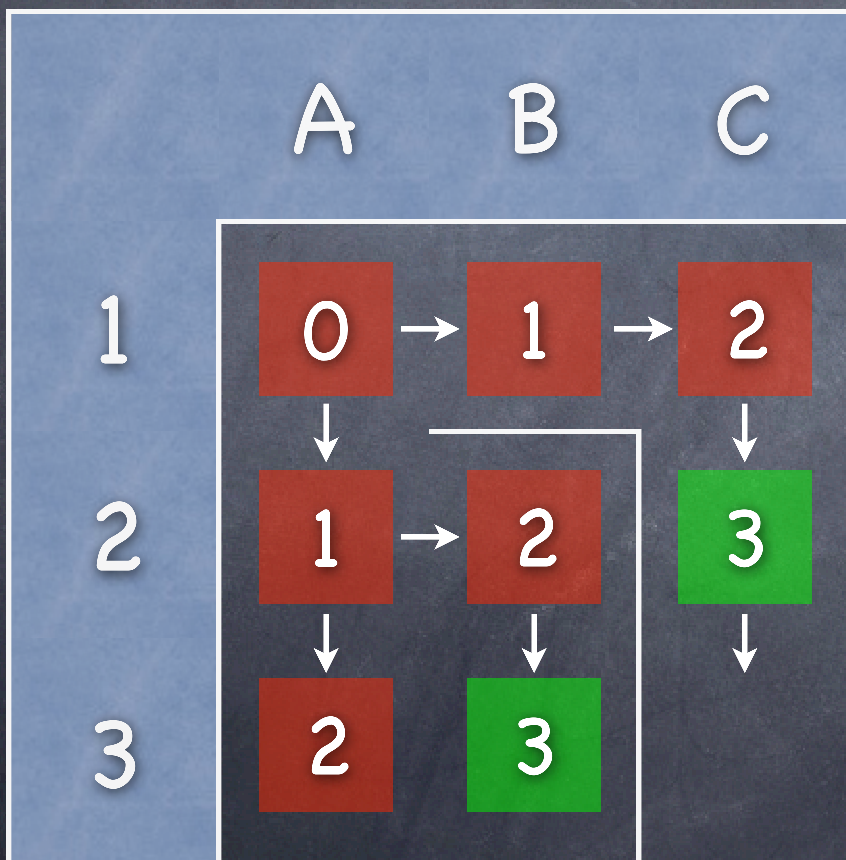
Open	Closed
	A1 (0)
	B1 (1)
	A2 (1)
C2 (3)	
B3 (3)	

A Simple Example



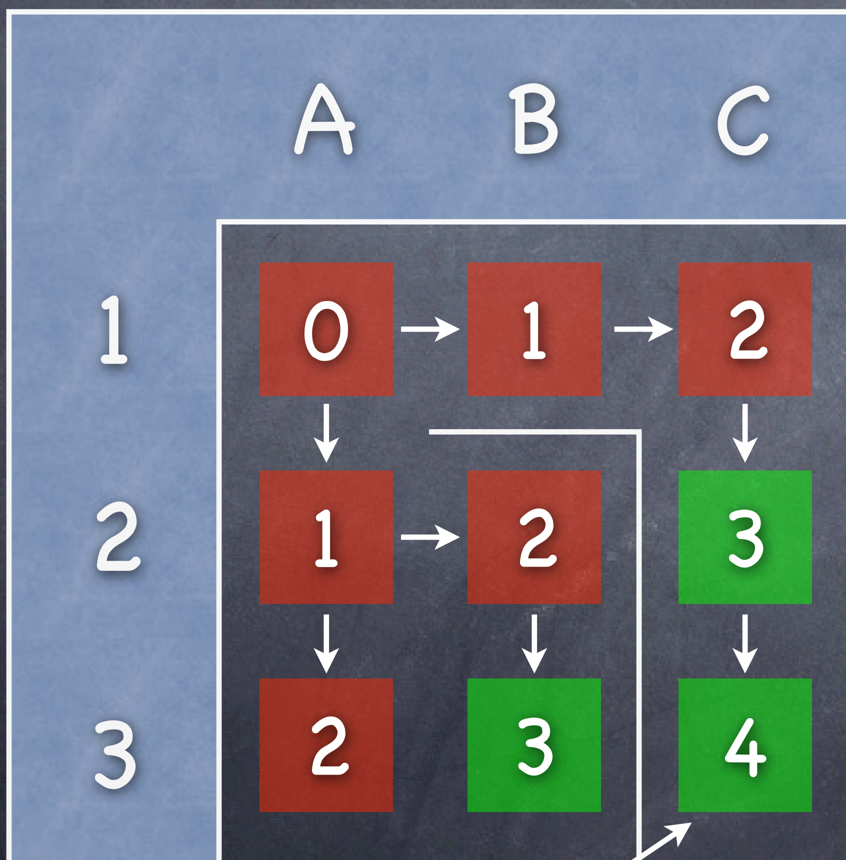
Open	Closed
	A1 (0)
	B1 (1)
	A2 (1)
C2 (3)	C1 (2)
B3 (3)	...

A Simple Example



Open	Closed
C2 (3)	A1 (0)
B3 (3)	B1 (1)
	A2 (1)
	C1 (2)
	...

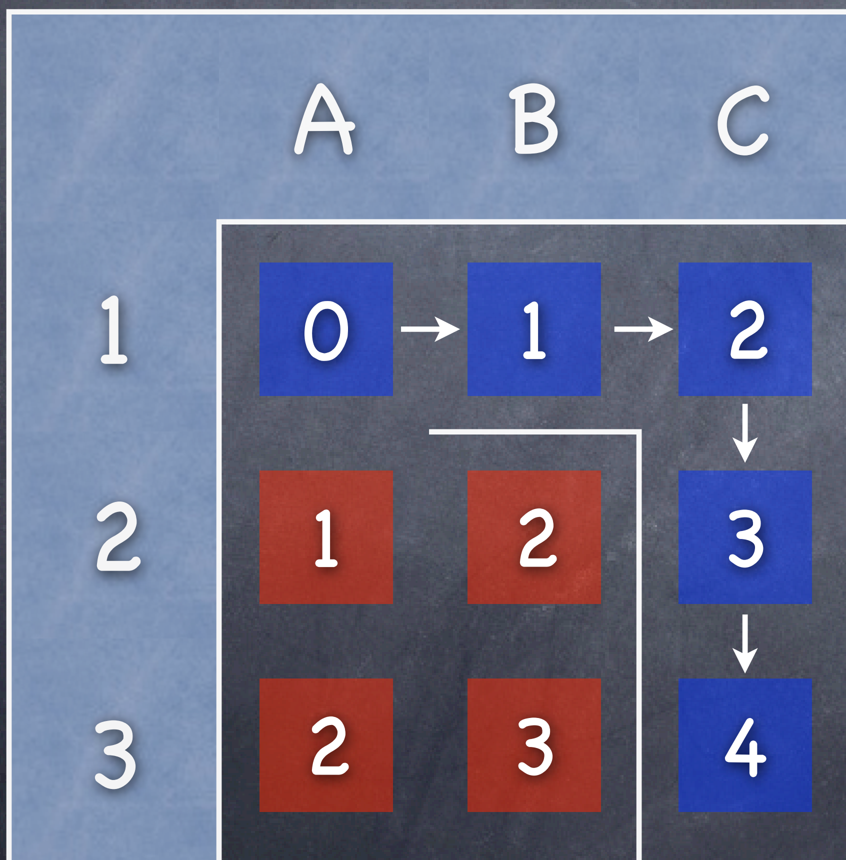
A Simple Example



Adjacent State is Goal!

Open	Closed
C2 (3)	A1 (0)
B3 (3)	B1 (1)
	A2 (1)
	C1 (2)
	...

A Simple Example



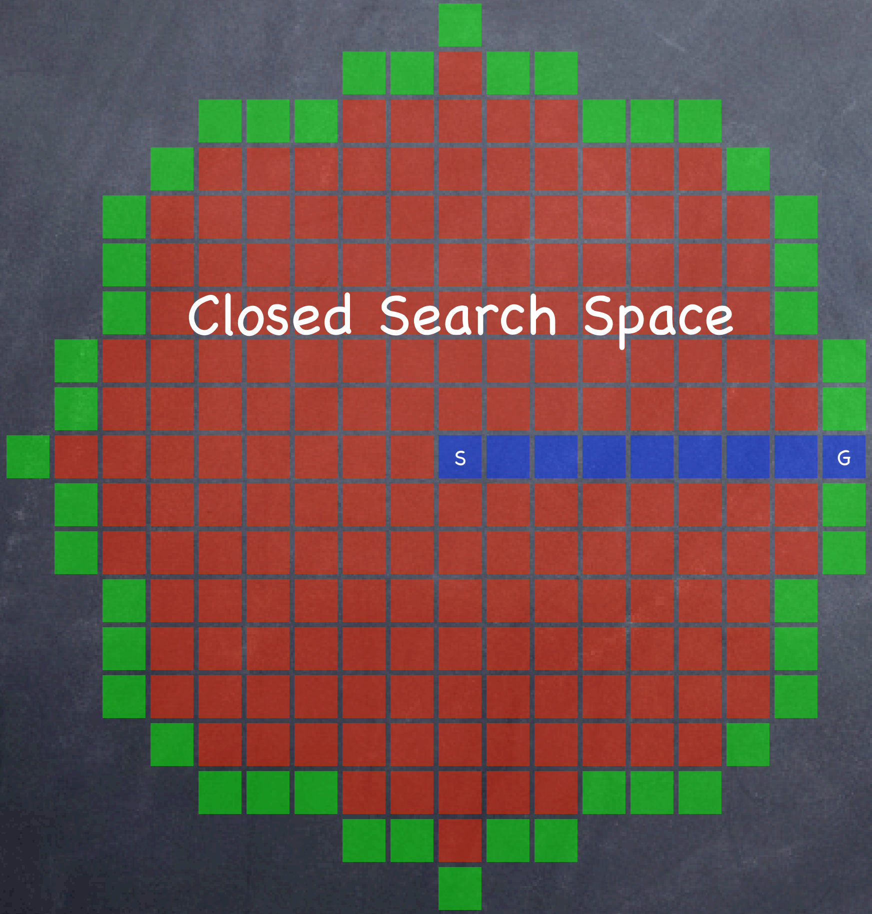
- It is now possible to construct a path back to the starting point.

A Simple Example

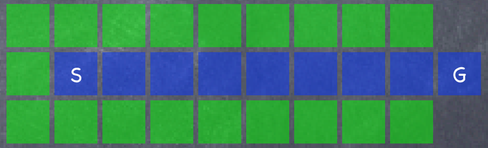
- This example is very simple, and every state within the 3x3 grid was explored. However, many optimizations can be made to the A* algorithm to increase efficiency.
- One such optimization is to add a **Heuristic** to the cost of each state evaluated. A good heuristic will increase efficiency - up to 100% in best cases.

Heuristic

Bad Heuristic



Good Heuristic



Summary

- The A* algorithm will always return the most efficient path, if one exists.
- If there is no path, however, then the A* algorithm becomes inefficient, as all state space will be explored.
- The A* algorithm can be extended to support multiple goals and multiple start locations, and is generally very adaptable to many different problem domains, ranging from music to computer graphics.