Puzzle Solving and State Space Search

A state refers to the status of a puzzle or game at a given moment. It could be a the positions of the pieces on a chessboard. In solving a puzzle or a problem one starts from some initial state and tries to reach a goal state by passing through a series of intermediate states. In game playing, each move on the game board is a transition from one state to another.

If you think of each state being connected to those states which can follow from it, you have a graph. Such a collection of interconnected states is called a state space graph.

In state based search, a computer program may start from an initial state, then look at one of its successor or children states to see if it is the goal state. If it is not a successor state of the successor is examined and so on until a goal state is reached. This approach is known as *depth first search.* The program may reach a dead end state from where it cannot proceed further. In such a situation the program may “backtrack”, i.e. undo its last move and try an alternative successor to its previous state. A path from the initial state to the goal state constitutes a solution.

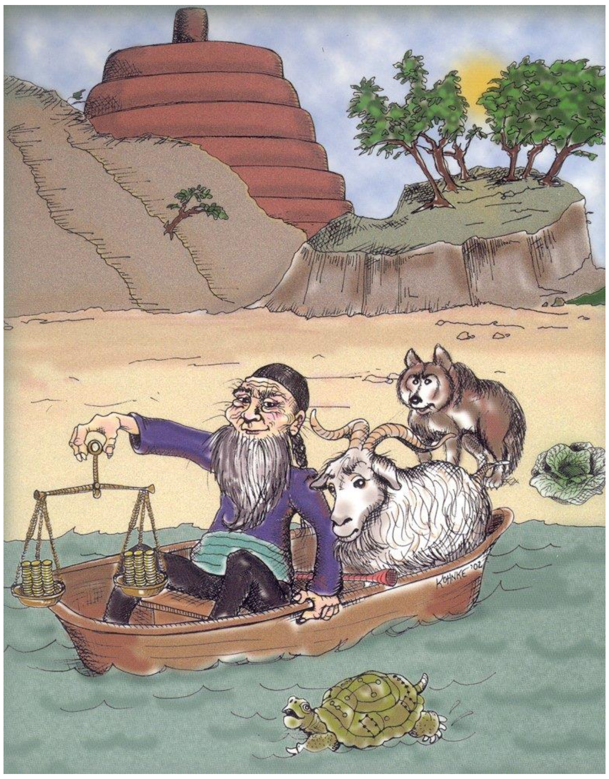
There are a number of ways of searching through the state space to find a solution. In depth first search, the program will examine a state, then proceed from it to one of its successor states and then repeats this process with the successor state, going a level deeper each time.

Breadth first search, BFS, is an alternative approach is where one examines all the successor states of a given state before proceeding to a new level.

Best first search chooses the next state based on some measure of how promising it is. It does a calculation for each possible next state and then chooses the best one. Best first search uses a priority queue.

Example: Farmer, Wolf, Goat and Cabbage Puzzle

A farmer is on the east bank of a river with a wolf, goat and cabbage in his care. Without his presence the wolf would eat the goat or the goat would eat the cabbage. The wolf is not interested in the cabbage. The farmer wishes to bring his three charges across the river. However the boat available to him can only carry one of the wolf, goat and cabbage besides himself. The puzzle is: is there a sequence of river crossings so that the farmer can transfer himself and the other three all intact to the west bank?



State Space Search

We envisage a puzzle or game as a network of possible states in any one of which the puzzle can be at a given time. Two states are linked if there is a valid move that allows the puzzle to go from the first state to the second.

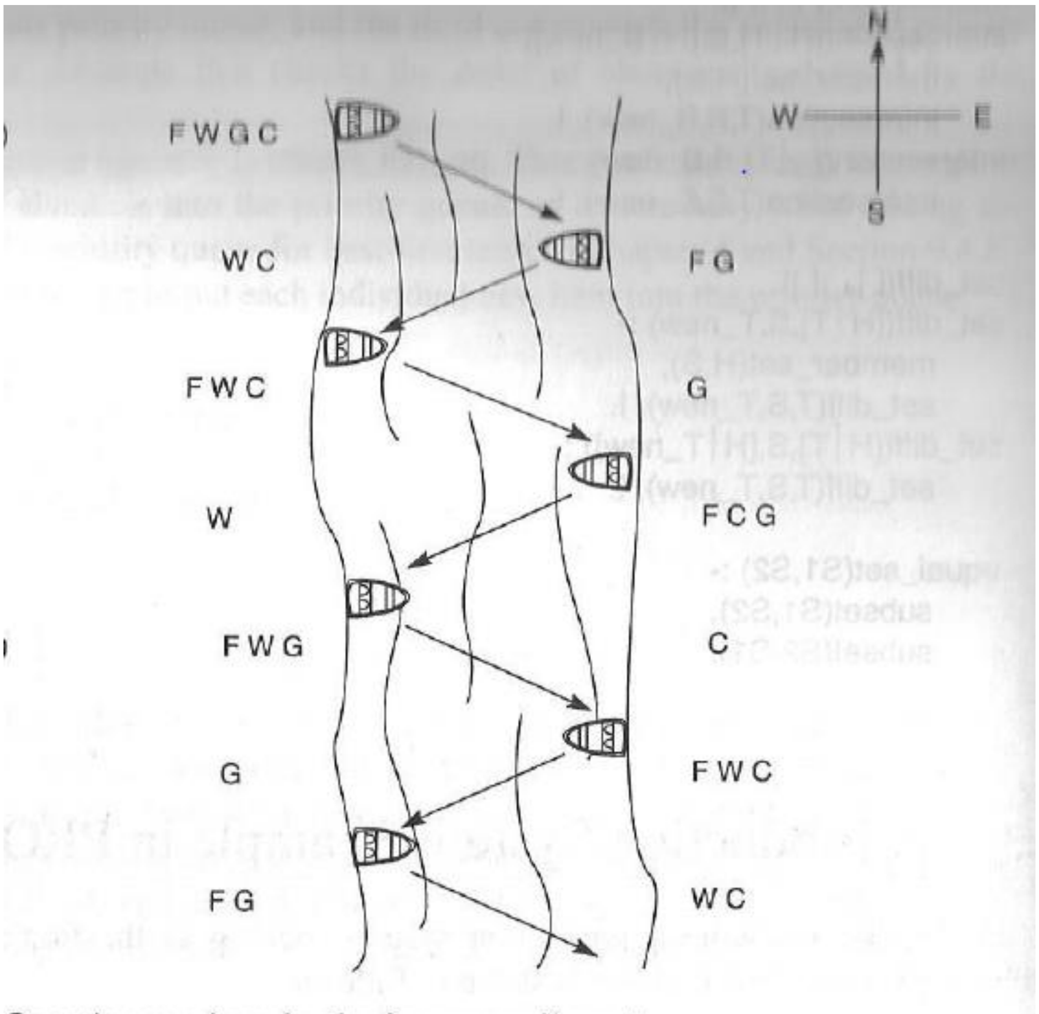
In solving a puzzle or a problem (such as the farmer, wolf, goat and cabbage one – fwgc from hereon) one starts from some initial state and tries to reach a goal state (e.g. all on west bank) by passing through a series of intermediate states. In game playing, each move transfers the system from one state to the next. For example in the fwgc problem, crossing the river in a boat either on his own or with one of the others changes the state of the system. There is usually a set of rules which tell one how to go from one state to another. Each state may be succeeded by one of a number of others.

For example a valid state change in the fwgc problem is:

$$\text{f w g c} \;\Big|\; \text{river} \;\Big|\; \xrightarrow{\text{farmer crosses with goat}} \; \text{w c} \;\Big|\; \text{river} \;\Big|\; \text{f g}$$

Such a collection of interconnected states is called a **state space graph**.

In state based search, a computer program may start from an initial state, then look at one of its successor states and so on until it reaches a goal state. It may reach a dead end state from where it cannot proceed further. In such a case the program may “backtrack”, i.e. undo its last move and try an alternative successor to its previous state.



Sample crossings for the farmer, wolf, goat, and cabbage problem.

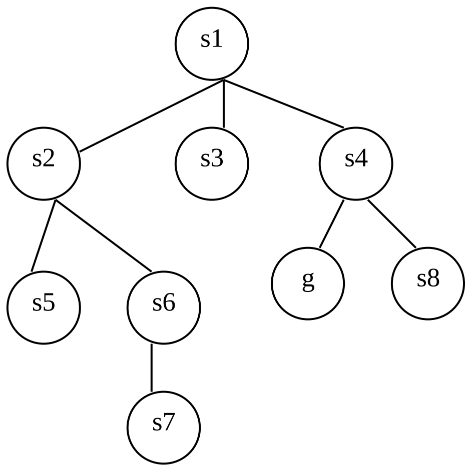
Depth First Search

In **depth first search** (DFS), the program will examine a state, then proceed from it to one of its successor states and then repeats this process with the successor state. Going a level deeper each time in the state space is referred to as depth first search.

In the state space example, starting with state s1, DFS with backtracking and would examine the states in the following order: s1, s2, s5, s6, s3, s4. DFS uses a stack to store states for examination. On examining a state, it pushes all its successor states onto a stack. The stack is then popped for next state to examine.

For example, on examining s1, stack is [s2, s3, s4], next s2 is removed and examined to give [s5, s6, s3, s4], then s5 removed and examined and so on. The stack contents will vary as follows:

	[s1]	
s1	[s2, s3, s4]	
s2	[s5, s6, s3, s4]	
s5	[s6, s3, s4]	
s6	[s7, s3, s4]	
s7	[s3, s4]	
s3	[s4]	
s4	[g, s8]	
g	[s8]	-- goal found



Once the goal state is found, the puzzle is solved. However, it is important that the path from the initial state (s1) to the goal state (g) is stored and maybe displayed so that whoever run the program can find out the set of moves that lead to the solution, otherwise the person just finds out that a computer program can solve the puzzle.

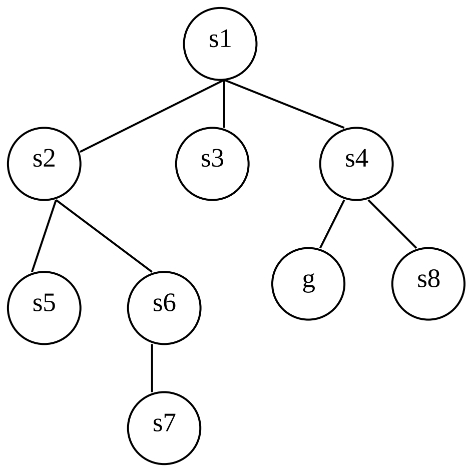
Breadth First Search

Breadth first search, BFS, is an alternative approach is where one examines all the successor states of a given state before proceeding to a new level. BFS would search graph fragment in order of: s1, s2, s3, s4, s5, s6, g. BFS uses a queue to store states for examination. On examining a state, its successor states are added to the back of the queue. The next state for examining is taken from the front of the queue.

For example, on examining s1, queue is [s2, s3, s4], next s2 is removed and examined to give [s3, s4, s5, s6], then s3 removed and examined and so on.

The queue contents will be:

	[s1]	
s1	[s2, s3, s4]	
s2	[s3, s4, s5, s6]	
s3	[s4, s5, s6]	
s4	[s5, s6, g, s8]	
s5	[s6, g, s8]	
s6	[g, s8, s7]	
g	[s8, s7]	-- goal found



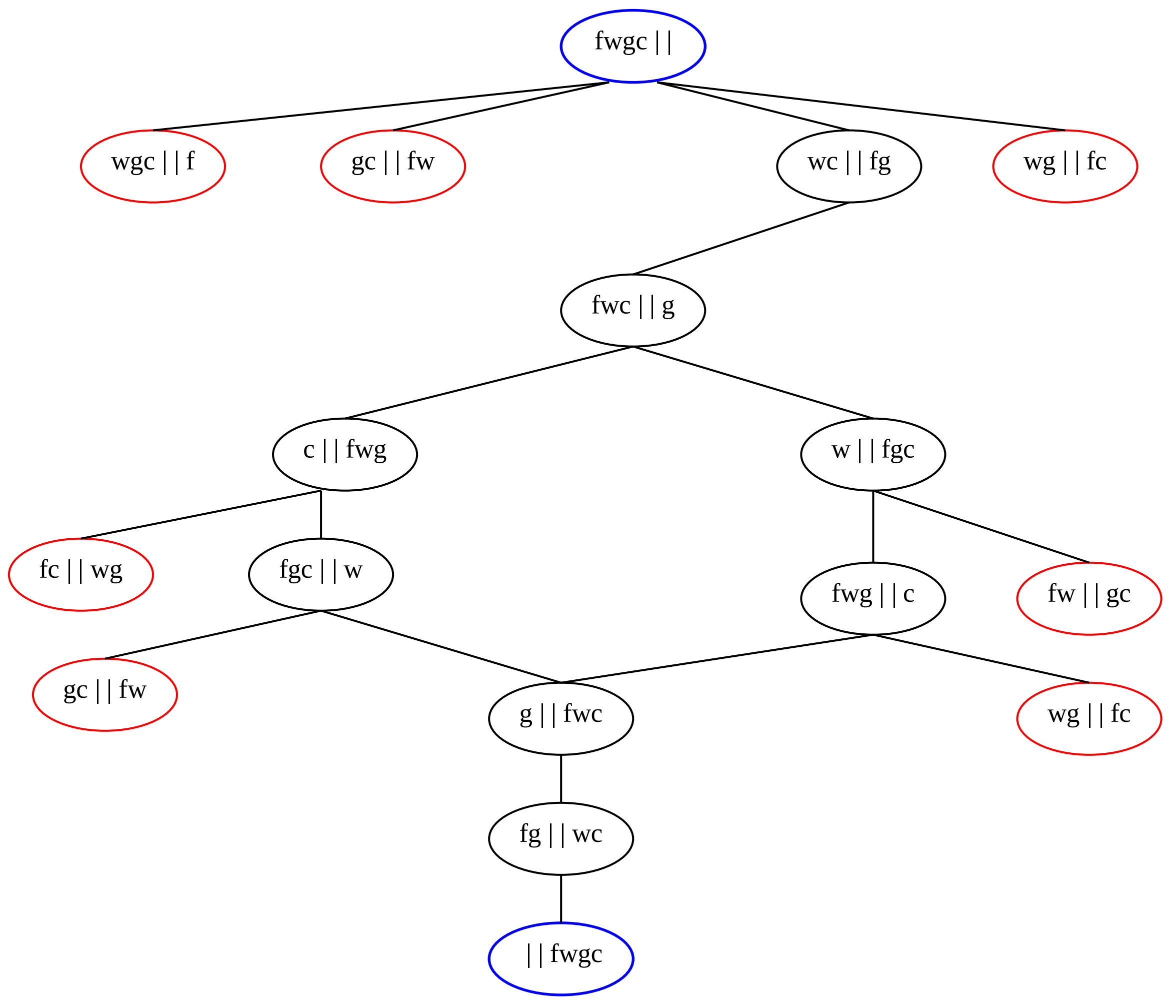
Best First Search

This approach chooses the next state based on some measure of how promising it is. It does a calculation for each possible next state and then chooses the best one. Best first search uses a priority queue – possibly a heap and is outside of our scope in first year.

In searching through a state space graph, care must be taken so that the program does not go round in cycles. This can be done by keeping a list of states visited so far in getting to current state and checking that a possible next state is not in the list.

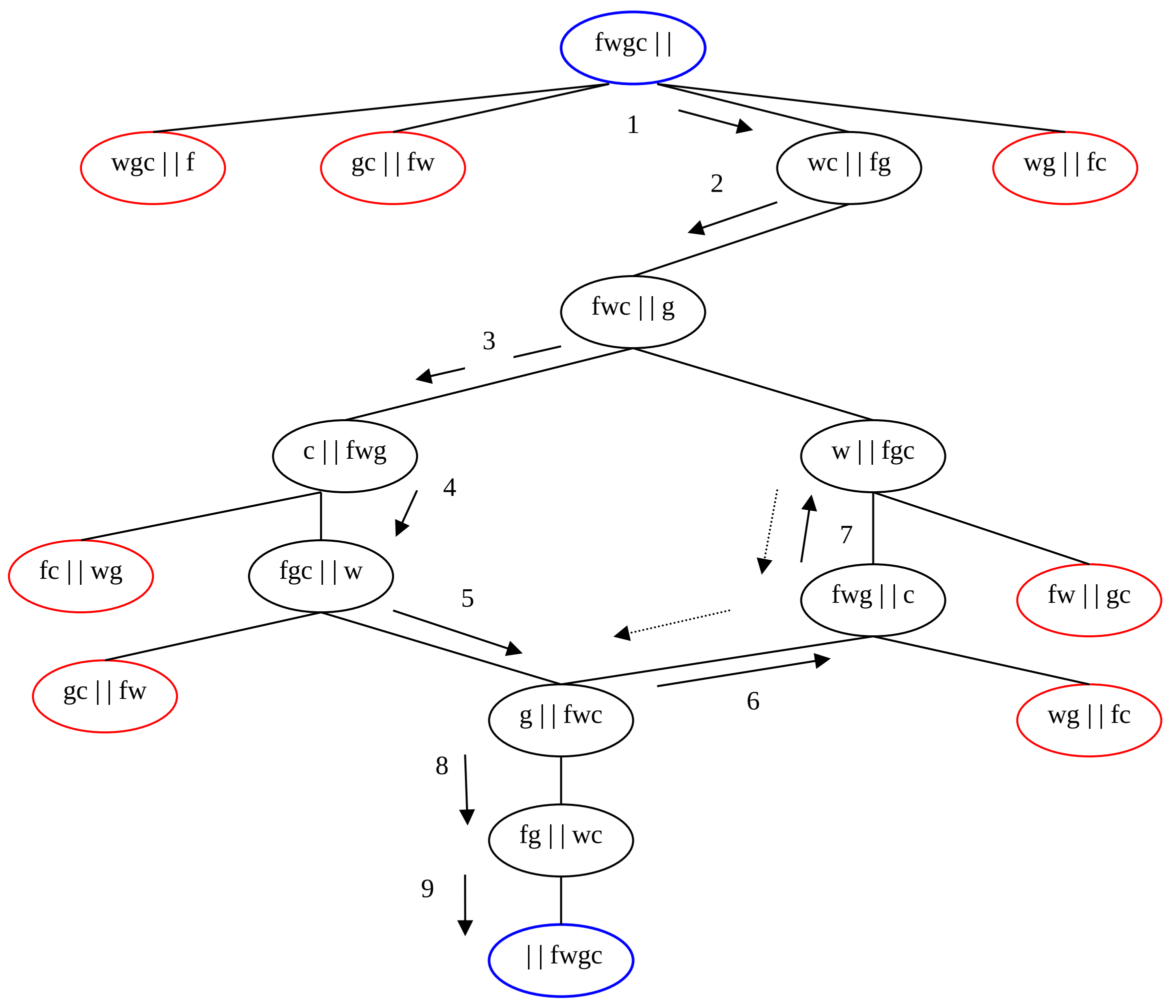
Farmer Wolf Goat and Cabbage Sate Space

How to represent the state, i.e. where the farmer, wolf, goat and cabbage are at a given time? Any one of them can have 2 possible positions, on the east or west bank of the river. One way to do this is to store their positions in a 4-tuple or record with 4 values or fields. (f, w, g, c) where the possible values for f, w, g and c are 'E' or 'W'. For example ('w', 'w', 'e', 'w') means farmer, wolf and cabbage on west bank and goat on east bank.

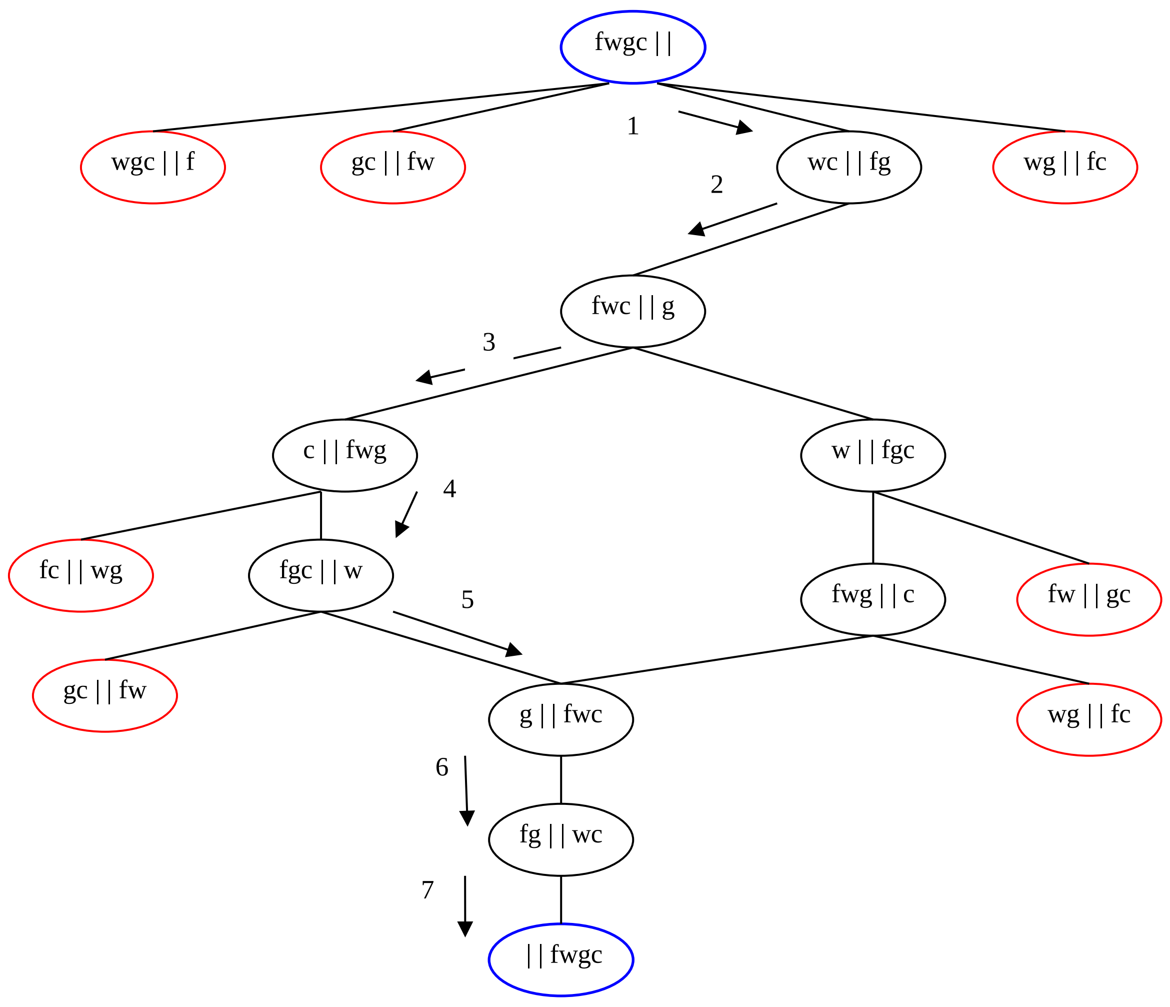


Note: states in red are not valid states or are unsafe and strictly speaking should not be part of the graph.

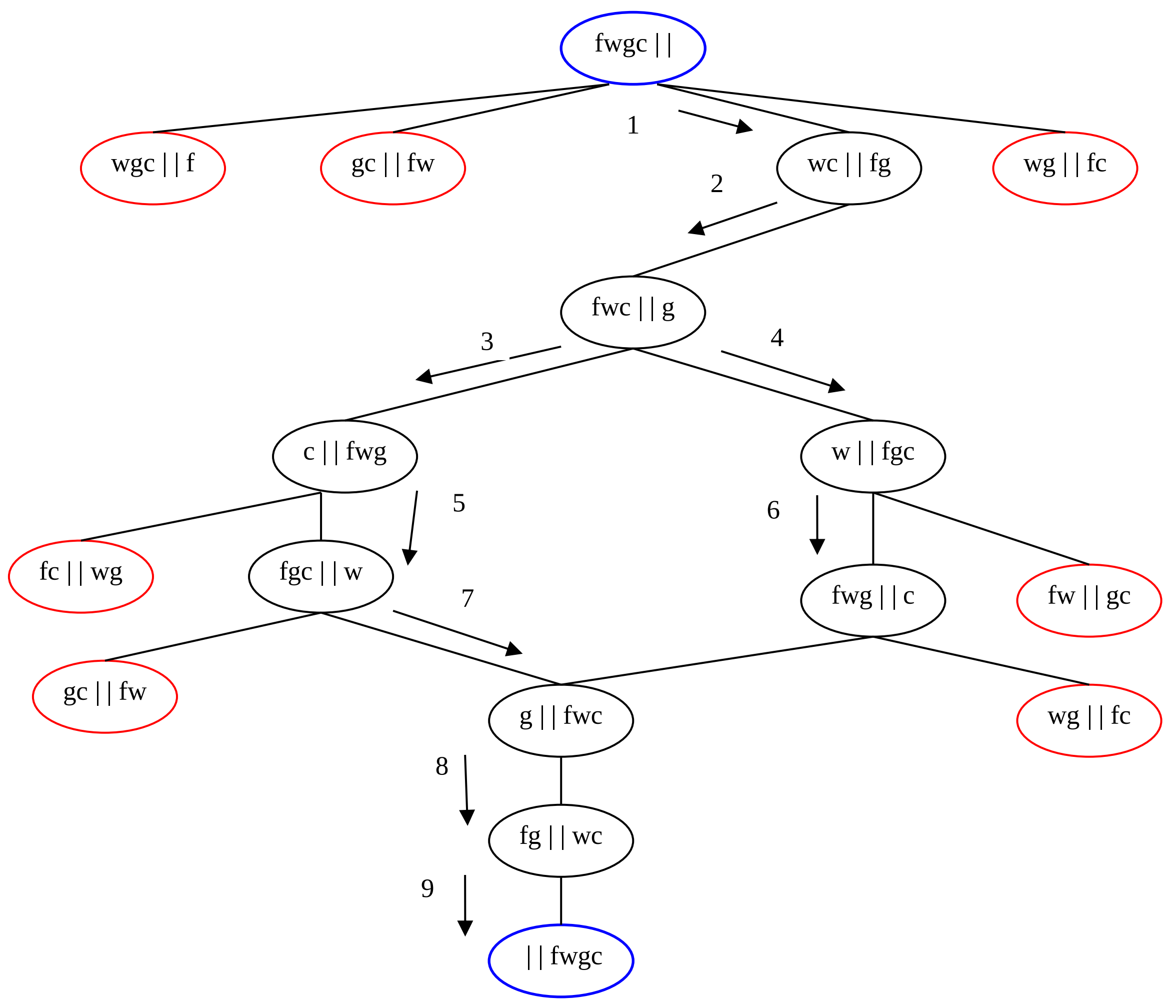
Depth First Search of State Space with Backtracking



Another Depth First Search of State Space



Breadth First Search of State Space



Exercise

Consider how a given state of this puzzle can be represented in Prolog. Then define a predicate **unsafe(State)** which is true or false depending on whether a state is unsafe or not. Unsafe means the wolf can eat the goat or the goat can eat the cabbage when the farmer is on the opposite bank.

Exercise

Do you think you could adapt the Prolog code that solves the 8-puzzle to this one? Have a try!