CS 3600 – Introduction to Intelligent Systems

1. Missionaries and Cannibals.

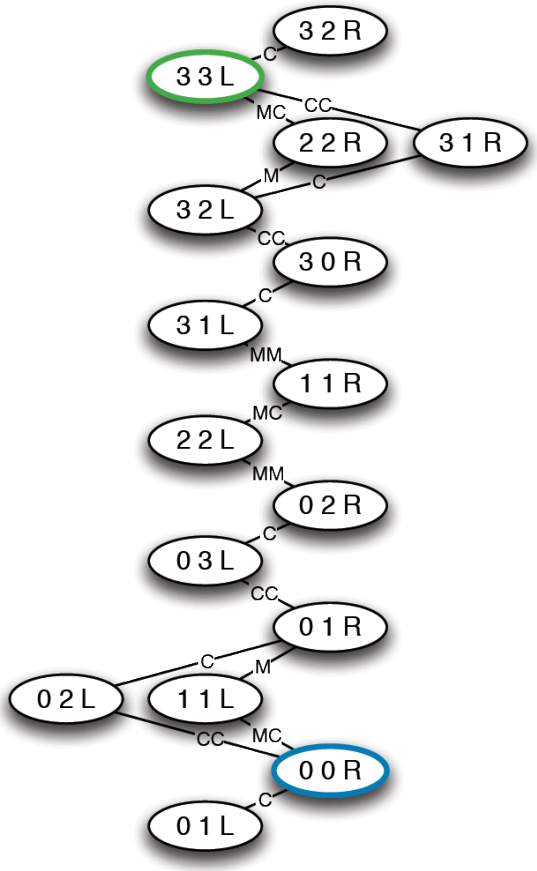
Missionaries and Cannibals is a problem in which 3 missionaries and 3 cannibals want to cross from the left bank of a river to the right bank of the river. There is a boat on the left bank, but it only carries at most two people at a time (and can never cross with zero people). If cannibals ever outnumber missionaries on either bank, the cannibals will eat the missionaries.

A state can be represented by a triple, (*m c b*), where *m* is the number of missionaries on the left, *c* is the number of cannibals on the left, and *b* indicates whether the boat is on the left bank or right bank. For example, the initial state is (3 3 L) and the goal state is (0 0 R).

Operators are:

- MM: 2 missionaries cross the river
- CC: 2 cannibals cross the river
- MC: 1 missionary and 1 cannibal cross the river
- M: 1 missionary crosses the river
- C: 1 cannibal crosses the river

Draw a diagram showing all the legal states and transitions from states corresponding to all **legal** operations. See Figure 3.3 in Russell & Norvig (p. 65) for an example of what your diagram should look like.



2. Answer the following question.

Hint: think about branching factor of states near the initial state and goal state, as compared to the branching factor of states in the "middle" of the state space.

3. Breadth-first search

Solve the Missionaries and Cannibals problem by implementing the BFS algorithm given in class. Implement the BFS algorithm to show the open list and closed list at every iteration of the algorithm until the goal is visited. Use the format below. I have given the first two iterations.

open: 33L
closed: nil

open: 31R, 22R, 32R
closed: 33L

open: 22R, 32R, 32L
closed: 31R, 33L

open: 32R, 32L, 32L
closed: 22R, 31R, 33L

open: 32L, 32L
closed: 32R, 22R, 31R, 33L

open: 32L, 30R
closed: 32L, 32R, 22R, 31R, 33L

open: 30R
closed: 32L, 32R, 22R, 31R, 33L

open: 31L
closed: 30R, 32L, 32R, 22R, 31R, 33L

open: 11R
closed: 31L, 30R, 32L, 32R, 22R, 31R, 33L

open: 22L
closed: 11R, 31L, 30R, 32L, 32R, 22R, 31R, 33L

open: 02R
closed: 22L, 30R, 11R, 31L, 30R, 32L, 32R, 22R, 31R, 33L

open: 03L
closed: 02R, 22L, 30R, 11R, 31L, 30R, 32L, 32R, 22R, 31R, 33L

open: 03L
closed: 11R, 02R, 22L, 30R, 11R, 31L, 30R, 32L, 32R, 22R, 31R, 33L

open: 01R
closed: 03L, 11R, 02R, 22L, 30R, 11R, 31L, 30R, 32L, 32R, 22R, 31R, 33L

open: 11L, 02L
closed: 01R, 03L, 11R, 02R, 22L, 30R, 11R, 31L, 30R, 32L, 32R, 22R, 31R, 33L

open: 02L, 00R, 01R
closed: 11L, 01R, 03L, 11R, 02R, 22L, 30R, 11R, 31L, 30R, 32L, 32R, 22R, 31R, 33L

open: 00R, 01R, 00R,
closed: 02L, 11L, 01R, 03L, 11R, 02R, 22L, 30R, 11R, 31L, 30R, 32L, 32R, 22R, 31R, 33L

Solution: CC, C, CC, C, MM, MC, MM, C, CC, M, MC

3. Depth-first search

Same as problem 2, but use the **DFS** algorithm given in class.

open: 33L
closed: nil

open: 31R, 22R, 32R
closed: 33L

open: 32L, 22R, 32R
closed: 31R, 33L

open: 30R, 22R, 32R
closed: 32L, 31R, 33L

open: 31L, 22R, 32R
closed: 30R, 32L, 31R, 33L

open: 11R, 30R, 22R, 32R
closed: 31L, 30R, 32L, 31R, 33L

open: 22L, 30R, 22R, 32R
closed: 11R, 31L, 30R, 32L, 31R, 33L

open: 02R, 30R, 22R, 32R
closed: 22L, 11R, 31L, 30R, 32L, 31R, 33L

open: 03L, 30R, 22R, 32R
closed: 02R, 22L, 11R, 31L, 30R, 32L, 31R, 33L

open: 01R, 30R, 22R, 32R
closed: 03L, 02R, 22L, 11R, 31L, 30R, 32L, 31R, 33L

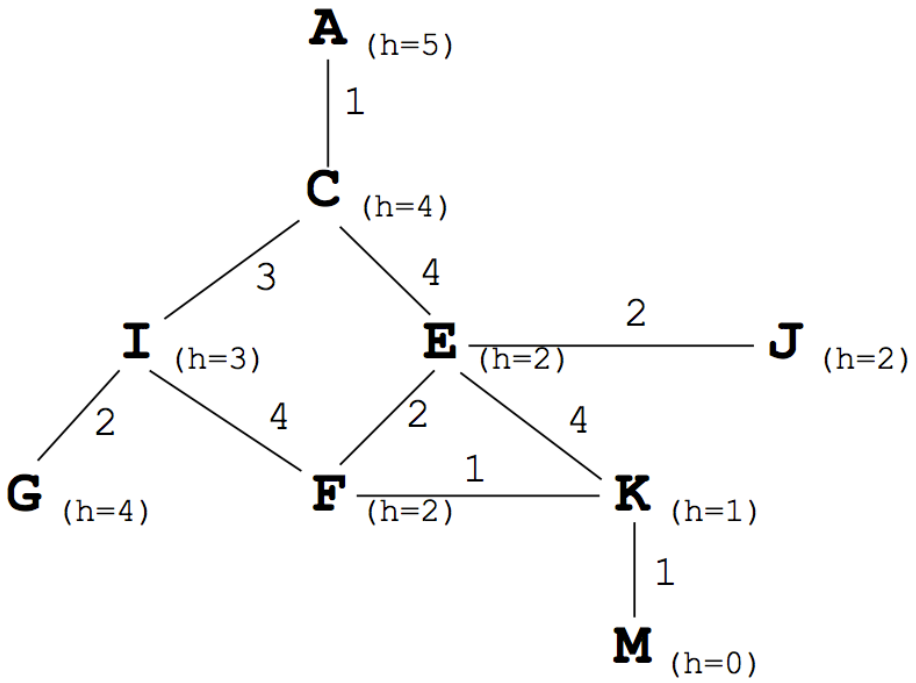
open: 11L, 02L, 30R, 22R, 32R
closed: 01R, 03L, 02R, 22L, 11R, 31L, 30R, 32L, 31R, 33L

open: 00R, 01R, 02L, 30R, 22R, 32R
closed: 11L, 01R, 03L, 02R, 22L, 11R, 31L, 30R, 32L, 31R, 33L

Solution: CC, C, CC, C, MM, MC, MM, C, CC, M, MC

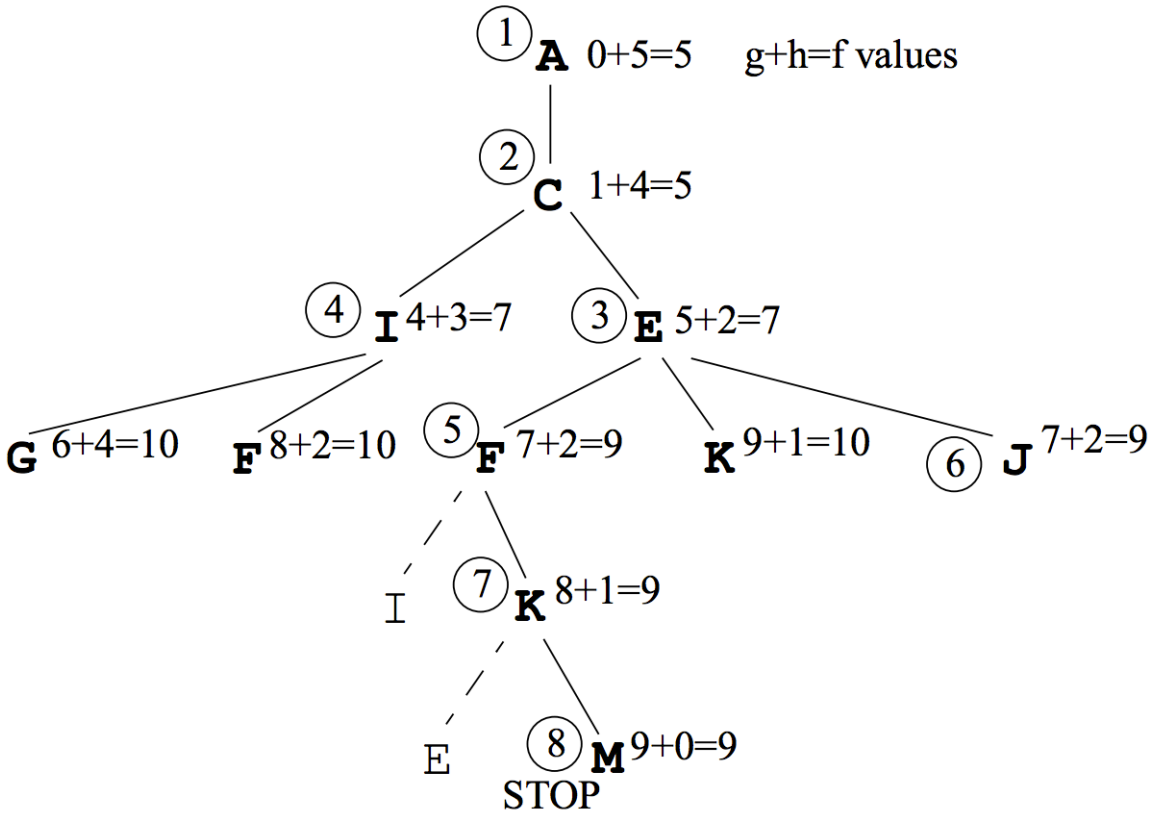
CS 3600 – Introduction to Artificial Intelligence

Practice Search problem



Consider the graph given above. It shows the actual distances between cities on a map. Each city is labeled with an estimate of the distance of the city from city M. Each path from a city to its neighbor is labeled with the length of the path. Consider an informed search for a shortest path from city A to city M using these estimates as heuristic function. Show how a version of A* would find this path. Assume that A* never expands the same state more than once and that it breaks ties toward cities with names earlier in the alphabet (A before B, B before C, and so on). Show the resulting search tree, including the f-value, g-value, and h-value of each node. Also clearly indicate the order in which the nodes are expanded (= label the node expanded first with 1, the node expanded next with 2, and so on - do not forget to label those nodes whose expansion only generates states that have already been expanded).

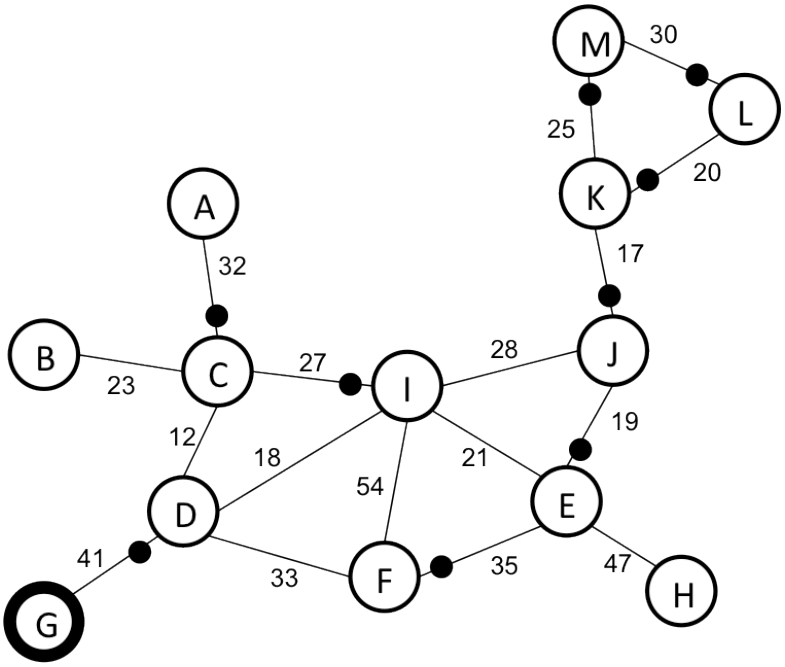
Search tree: (circled numbers indicate the order of expansion)



CS 3600 – Introduction to AI

Practice Search Problem

Consider the search space diagrammed below. Action costs are given next to each arc. Use "E" as the initial state and "G" as the goal state. The successor function for a node starts with the arc with the black dot next to it first, and the generates successors in a counter-clockwise fashion. For example, Successors(E) = {J, I, F, H} and Successors(J) = {K, I, E}.



1. What order will nodes be visited using the **breadth-first** algorithm? Ignore action costs. List the visit order and the final solution.

2. What order will nodes be visited using the **depth-first** algorithm? Ignore action costs. List the visit order and the final solution.

3. What order will nodes be visited using the **uniform cost search** algorithm? List the visit order and the final solution.

4. What order will nodes be visited using the best-first algorithm? Use the heuristic function given below. List the visit order and the final solution.

h(A) = 65	h(D) = 41	h(H) = 100	h(K) = 50
h(B) = 25	h(E) = 85	h(I) = 55	h(L) = 100
h(C) = 50	h(F) = 30	h(J) = 110	h(M) = 95

1. What order will nodes be visited using the **breadth-first** algorithm? Ignore action costs. List the visit order and the final solution.

Open: E
Closed: nil

Visit E

Open: J I F H
Closed: E

Visit J

Open: I F H K I E * Note: I am placing duplicates on open, but ignore later
Closed: E J

Visit I

Open: F H K I E C D F E J
Closed: E J I

Visit F

Open: H K I E C D F E J D
Closed: E J I F

Visit H

Open: K I E C D F E J D
Closed: E J I F H

Visit K

Open: I E C D F E J D L M
Closed: E J I F H K

Visit C * Note: Skipping I and E, which are already visited

Open: D F E J D L M A B D
Closed: E J I F H K C

Visit D

Open: F E J D L M A B D G F * Can jump straight to G if cost of goal() isn't great.
Closed: E J I F H K C D
Visit L * Skip F, E, J, D, all of which are already visited

Open: M A B D G F M
Closed: E J I F H K C D L

Visit M

Open: A B D G F M
Closed: E J I F H K C D L M

Visit A

Open: B D G F M
Closed: E J I F H K C D L M A

Visit B

Open: D G F M
Closed: E J I F H K C D L M A B

Visit G * Skip D ,which is already visited

Open: F M
Closed: E J I F H K C D L M A B G

Visit order: E J I F H K C D L M A B G

Solution: E->I, I->D, G->D

2. What order will nodes be visited using the **depth-first** algorithm? Ignore action costs. List the visit order and the final solution.

Open: E
Closed: nil

Visit E

Open: J I F H
Closed: E

Visit J

Open: K I I F H
Closed: E J

Visit K

Open: L M I I F H
Closed: E J K

Visit L

Open: M M I I F H
Closed: E J K L

Visit M

Open: M I I F H
Closed: E J K L M

Visit I * Skip M, already visited

Open: C D F I F H
Closed: E J K L M I

Visit C

Open: A B D D F I F H
Closed: E J K L M I C

Visit A

Open: B D D F I F H
Closed: E J K L M I C A

Visit B

Open: D D F I F H
Closed: E J K L M I C A B

Visit D

Open: G F D F I F H
Closed: E J K L M I C A B D

Visit G

Open: F D F I F H
Closed: E J K L M I C A B D G

Visit order: E J K L M I C A B D G

Solution: E->I, I->C, C->D, D->G

3. What order will nodes be visited using the **uniform cost search** algorithm? List the visit order and the final solution.

Open: E(0)
Closed: nil

Visit E

Open: J(19) I(21) F(35) H(47)
Closed: E

Visit J

Open: I(21) F(35) K(36) H(47)
Closed: E J

Visit I

Open: F(35) K(36) D(39) H(47) C(48)
Closed: E J I

Visit F

Open: K(36) D(39) H(47) C(48)
Closed: E J I F

Visit K

Open: D(39) H(47) C(48) L(56) M(61)
Closed: E J I F K

Visit D

Open: H(47) C(48) L(56) M(61) G(80)
Closed: E J I F K D

Visit H

Open: C(48) L(56) M(61) G(80)
Closed: E J I F K D H

Visit C

Open: L(56) M(61) B(71) G(80) A(80)
Closed: E J I F K D H C

Visit L

Open: M(61) B(71) G(80) A(80)
Closed: E J I F K D H C L

Visit M

Open: B(71) G(80) A(80)
Closed: E J I F K D H C L M

Visit B

Open: G(80) A(80)
Closed: E J I F K D H C L M B

Visit G

Open: A(80)
Closed: E J I F K D H C L M B G

Visit order: E J I F K D H C L M B G

Solution: E->I, I->D, D->G

4. What order will nodes be visited using the best-first algorithm? Use the heuristic function given below. List the visit order and the final solution.

Open: E(85)
Closed: nil

Visit E

Open: F(65) I(76) J(129) H(147)
Closed: E

Visit F

Open: I(76) D(109) J(129) H(147)
Closed: E F

Visit I

Open: D(80) C(98) J(129) H(147)
Closed: E F I

Visit D
Open: G(80) C(98) J(129) H(147) G(80)
Closed: E F I D

Visit G

Open: C(98) J(129) H(147) G(80)
Closed: E F I D G

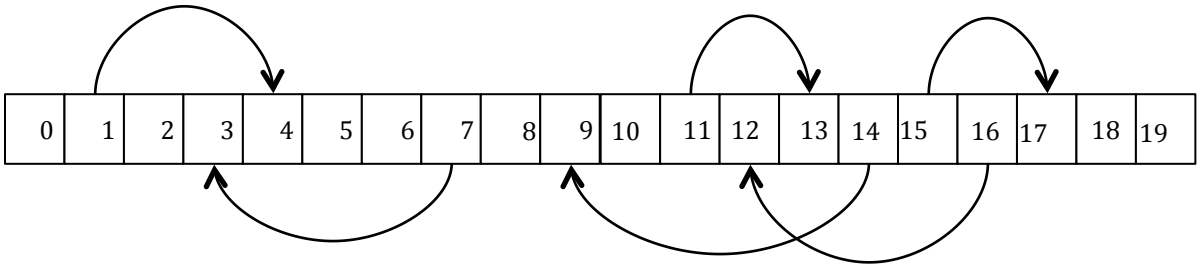
Visit order: E F I D G

Solution: E->I, I->D, D->G

CS 3600 – Introduction to Artificial Intelligence

Adversarial Games with Dice

Consider the following board game, based loosely on chutes and ladders. Two players race from position 0 to position 19. The first person to land exactly on position 19 wins. If you land on any position with an arc emanating, you automatically leap to the position at which the arc terminates. For example, if you stop on position 1, you are teleported to position 4.



On each turn, a player may perform one of the following actions:

- Move one position forward or backward.
- Roll one 4-sided die (1D4) and move the resultant number of positions forward or backward.
- Roll two 4-sided dice (2D4) and move the resultant summed number of positions forward or backward.

Problems:

1. Design a state representation. What is the initial state?

2. Write the pseudocode for a successor function. What kinds of states are there? What does each ply in an adversarial search tree represent?

3. Write the pseudocode for a terminal function.

4. Write the pseudocode for a utility function.

5. Draw out a portion of the adversarial search tree. Show enough ply for at least two turns.

6. Suppose the board is very long and it is impractical to reach terminal states each time. Design a cut off function and evaluation function that returns the value of intermediate, non-terminal states.

1. Design a state representation. What is the initial state?

State is a tuple (p1-location, p2-location, die-roll, whose-turn, phase). The first two are board positions. Die-roll is the sum of the numbers on the dice at the moment. Phase determines which part of a player's turn a player is in. Phase can be {start, 1D4, 2D4, move} as if these were the states of a small FSM with possible implicit transitions {start→1D4, start→2D4, 1D4→move, 2D4→move}.

2. Write the pseudocode for a successor function. What kinds of states are there? What does each ply in an adversarial search tree represent?

States can be min, max, or chance. However, each player has two max nodes per turn: one for picking how to roll the dice, and one for moving forward or backward.

Ply 1: Agent picks to move-1, roll 1D4, or roll 2D4
Ply 2: Chance nodes for die rolls
Ply 3: Agent picks to move forward or backward the number of spaces determined during ply 1 or ply 2.
Ply 4: Opponent picks to move a single space, move 1D4, or move 2D4
Ply 5: Chance nodes for die rolls
Ply 6: Opponent picks to move forward or backward
…

```
Function successors (state = (p1, p2, die, turn, phase))
    IF (phase == 'start')
        RETURN ( [action: 'move-1', state: (p1, p2, 1, turn, 'move')],
                 [action: '1D4', state: (p1, p2, 0, turn, '1D4')],
                 [action: '2D4', state: (p1, p2, 0, turn, '2D4')] )
    IF (phase == '1D4')
        RETURN  ([action: 1, state: (p1, p2, 1, turn, 'move')],
                 [action: 2, state: (p1, p2, 2, turn, 'move')],
                 [action: 3, state: (p1, p2, 3, turn, 'move')],
                 [action: 4, state: (p1, p2, 4, turn, 'move')] )
    IF (phase == '2D4')
        RETURN ( [action: 2, state: (p1, p2, 2, turn, 'move')],
                 [action: 3, state: (p1, p2, 3, turn, 'move')],
                 …
                 [action: 8, state: (p1, p2, 8, turn, 'move')] )
    IF (phase == 'move' and turn == 'p1')
        RETURN ( [action: 'left', state: (p1-die, p2, 0, 'p2', 'start')],
                 [action: 'right', state: (p1+die, p2, start, 'p2', 'start')] )
    IF (phase == 'move' and turn == 'p2')
        RETURN ( [action: 'left', state: (p1, p2-die, 0, 'p1', 'start')],
                 [action: 'right', state: (p1, p2+die, 0, 'p1', 'start')] )
```

(Note: I am assuming the expectiminimax-value recursive function handles computation of probability)

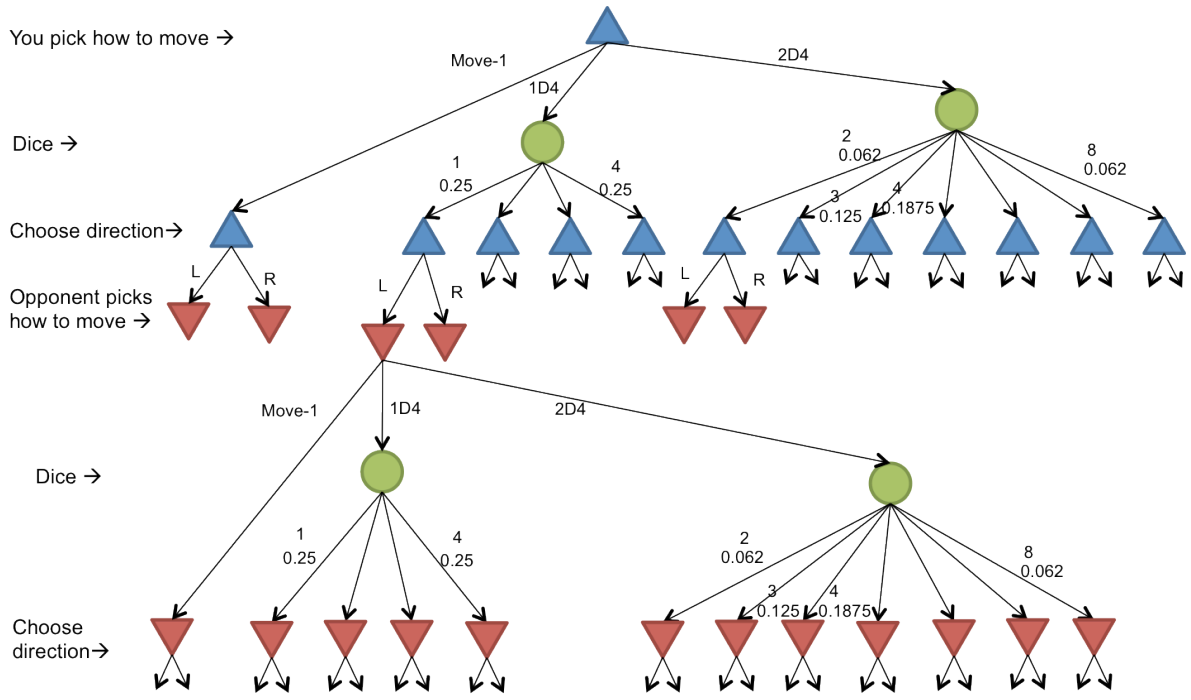
3. Write the pseudocode for a terminal function.

Function terminal (state)
    RETURN (p1 == 19 OR p2 == 19)

4. Write the pseudocode for a utility function.

Function utility (state)
    IF (p1 == 19) return +1
    ELSE return -1

5. Draw out a portion of the adversarial search tree. Show enough ply for at least two turns.



6. Suppose the board is very long and it is impractical to reach terminal states each time. Design a cut off function and evaluation function that returns the value of intermediate, non-terminal states.

A standard fixed depth cut-off can be used. However, it may also be able to compute a cut-off depth based on stability. For example, if one player has a large enough lead (and there are no ladders that will teleport the losing player close to or ahead of the winning player), one could assume that the utility of descendants is unlikely to change.

One way to compute the material evaluation value is the distance between p1-location and p2-location. A note of warning: If the utility function returns +1 or -1 and the evaluation function for non-terminal states returns numbers outside the [-1, +1] range, you can get unexpected results. You may want to find a way to *normalize* the evaluation function so that the range of results is comparable to that of the utility function.

CS 3600 Introduction to AI

Constraint Satisfaction Problem

Solve the cryptarithmetic problem in Figure 6.2 of the Russell & Norvig textbook by hand, using the strategy of back-tracking with forward checking and the MRV and least-constraining value heuristics.

The exact steps depend on certain choices you are free to make; here are the ones I made:

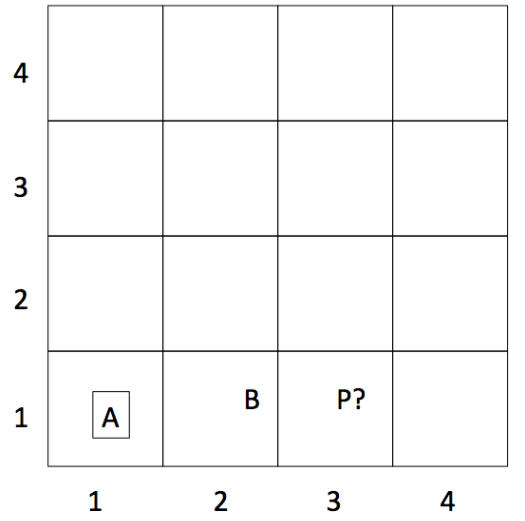
- **a.** Choose the $\mathbf{X}_3$ variable. Its domain is **{0, 1}**.
- **b.** Choose the value 1 for $\mathbf{X}_3$. (We can't choose 0; it wouldn't survive forward checking, because it would force **F** to be 0, and the leading digit of the sum must be non-zero.)
- **c.** Choose **F**, because it has only one remaining value.
- **d.** Choose the value 1 for **F**.
- **e.** Now $\mathbf{X}_2$ and $\mathbf{X}_1$ are tied for minimum remaining values at 2; let's choose $\mathbf{X}_2$.
- **f.** Either value survives forward checking, let's choose 0 for $\mathbf{X}_2$.
- **g.** Now $\mathbf{X}_1$ has the minimum remaining values.
- **h.** Again, arbitrarily choose 0 for the value of $\mathbf{X}_1$.
- **i.** The variable **O** must be an even number (because it is the sum of **T** + **T** less than 5 (because **O** + **O** = **R** + **10 Å~ 0**). That makes it most constrained.
- **j.** Arbitrarily choose 4 as the value of **O**.
- **k.** **R** now has only 1 remaining value.
- **l.** Choose the value 8 for **R**.
- **m.** **T** now has only 1 remaining value.
- **n.** Choose the value 7 for **T**.
- **o.** **U** must be an even number less than 9; choose **U**.
- **p.** The only value for **U** that survives forward checking is 6.
- **q.** The only variable left is **W**.
- **r.** The only value left for **W** is 3.
- **s.** This is a solution.

This is a rather easy (under-constrained) puzzle, so it is not surprising that we arrive at a solution with no backtracking (given that we are allowed to use forward checking).

CS 3600 – Introduction to AI

Resolution

Consider the Wumpus world problem below. The world is a 4 x 4 grid with the agent in (1, 1). The Agent's knowledge base is given, where Bij means breeze in location (i, j), and Pij means pit in location (i, j).



¬P11
¬P21
¬P12
¬B11
¬B12
B21
B11 ↔ (P12 ∨ P21)
B21 ↔ (P11 ∨ P22 ∨ P31)
B12 ↔ (P11 ∨ P22 ∨ P13)

The Agent wants to know if there is a Pit in location (3, 1), i.e., P31.

Use Resolution to determine whether P31 is entailed by the knowledge base.

1. Convert the formulae in the knowledge base to conjunctive normal form.

2. Negate the conclusion and convert the conclusion to conjunctive normal form.

3. Draw out the resolution search space. Use the following heuristic:

Prefer clauses derived from the conclusion from smallest to largest (# of literals).
Next prefer clauses not derived from the conclusion, from smallest to largest.
Break ties in alpha-numeric order, e.g., B11, B12, B13, …, B21, B22, …, P11, P12, …

1. Convert the formulae in the knowledge base to conjunctive normal form.

¬P11

¬P21

¬P12

¬B11

¬B12

B21

B11 ↔ (P12 ∨ P21) becomes:
¬B11 ∨ P12 ∨ P21
¬P12 ∨ B11
¬P21 ∨ B11

B21 ↔ (P11 ∨ P22 ∨ P31) becomes:
¬P11 ∨ B21
¬P22 ∨ B21
¬P31 ∨ B21
¬B21 ∨ P11 ∨ P22 ∨ P31

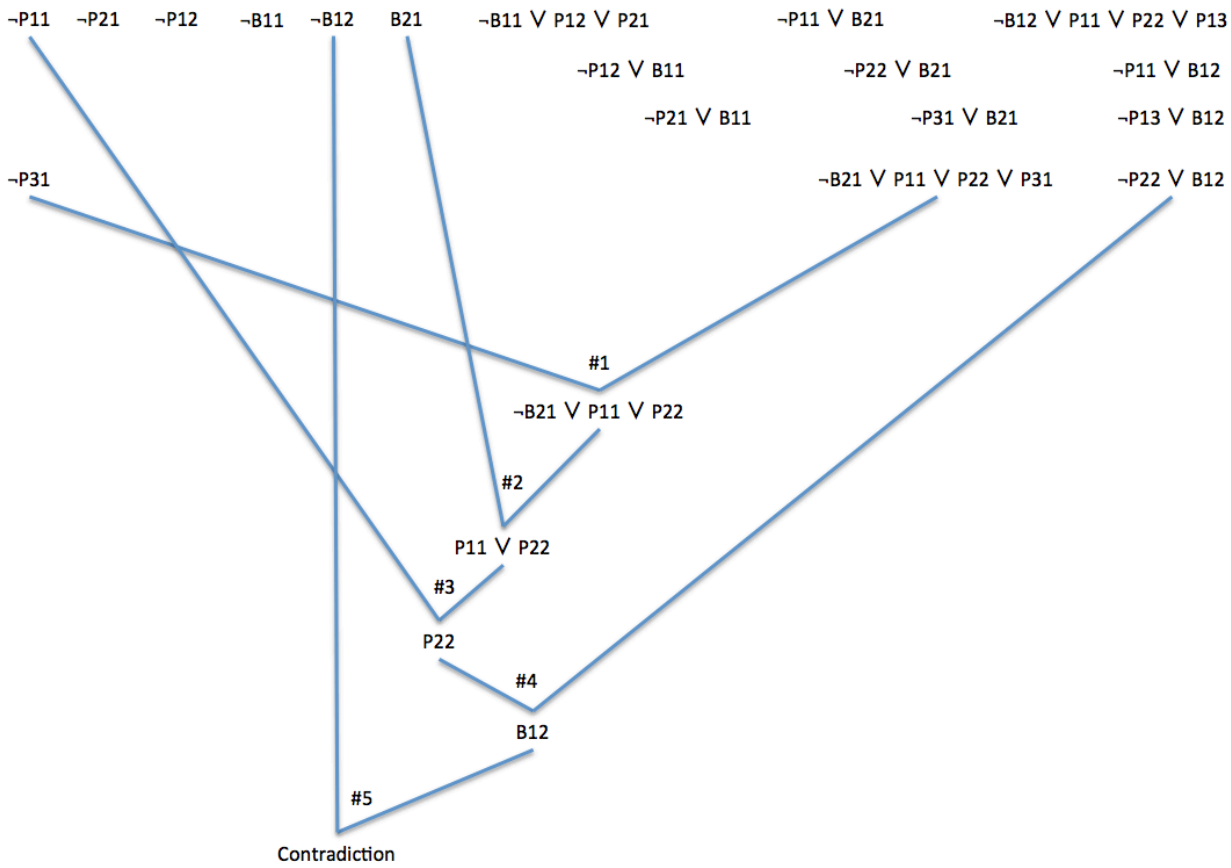
B12 ↔ (P11 ∨ P22 ∨ P13) becomes:
¬B12 ∨ P11 ∨ P22 ∨ P13
¬P11 ∨ B12
¬P13 ∨ B12
¬P22 ∨ B12

2. Negate the conclusion and convert the conclusion to conjunctive normal form.

¬P31

3. Draw out the resolution search space. Use the following heuristic:

Prefer clauses derived from the conclusion from smallest to largest (# of literals).
Next prefer clauses not derived from the conclusion, from smallest to largest.
Break ties in alpha-numeric order, e.g., B11, B12, B13, …, B21, B22, …, P11, P12, …



(I give the ordering of resolutions I used based on my interpretation of the heuristic. Other paths to contradictions exist as well.)