CSE 574
HW 2
Due Friday September 14 at 5pm

Please submit on Blackboard. Show work for full credit. See the class policies for late HW assignments as we will strictly enforce.

Name:

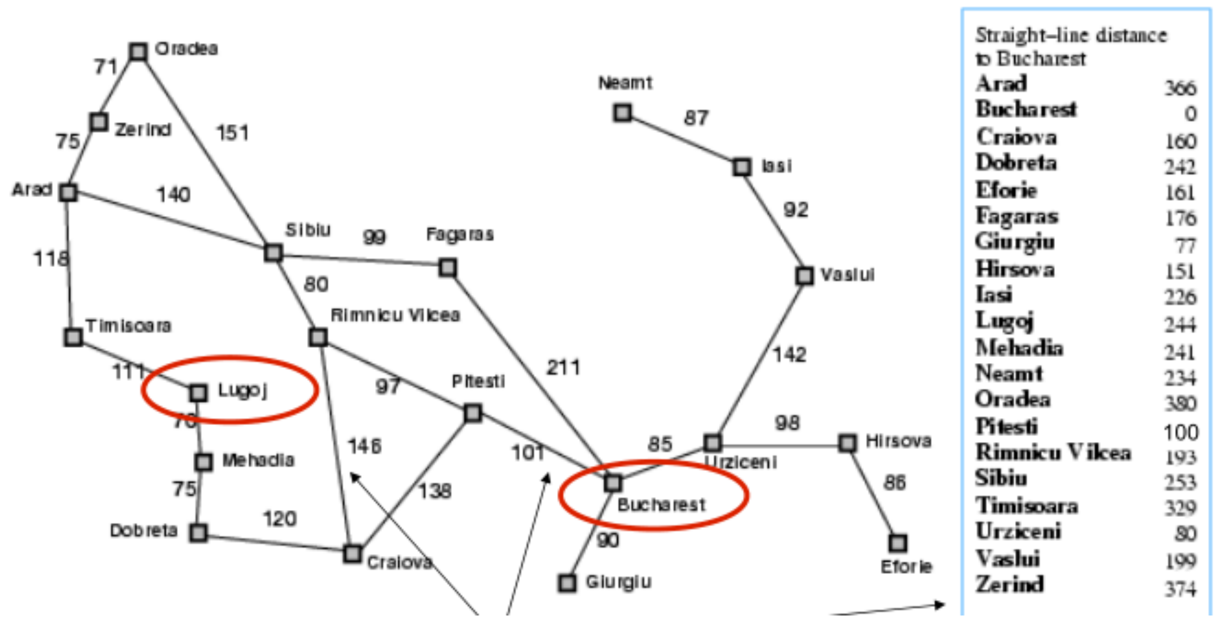
ASU ID:

1. Uninformed Search. Iterative lengthening search is an iterative analog of uniform cost search. The idea is to use increasing limits on path cost. If a node is generated whose path cost exceeds the current limit, it is immediately discarded. For each new iteration, the limit is set to the lowest path cost of any node discarded in the previous iteration.
a. Show that this algorithm is optimal for general path costs.
b. Consider a uniform tree with branching factor b, solution depth d, and unit step costs. How many iterations will iterative lengthening require? (please show your work)
c. Now consider step costs drawn from the continuous range [ε,1], where 0 < ε < 1. How many iterations are required in the worst case?

Solution:

a. Algorithm expands nodes in order of increasing path cost: therefore the first goal it encounters will be the goal with cheapest cost(enough to give credit). Uniform cost just iteratively inspects the unexplored node nearest to the start node. Iterative lengthening search is a uniform cost search that runs multiple times with a growing bound on the maximum cost from start to end before quitting.
b. For each iteration, the number of nodes would be $b+b^2+..+b^n$. The final iteration would be $b+b^2+…+b^d$. Then, all runswould be summed together for d times, the final count is $db+(d-1)b^2+…+b^d$(enough for full credit).
So, the number of iterationsis $O(b^d)$.
c. In worst case, the limit would be expanded from 0 to d with step ε. Thus the number of iterations in the worst case is d/ ε .

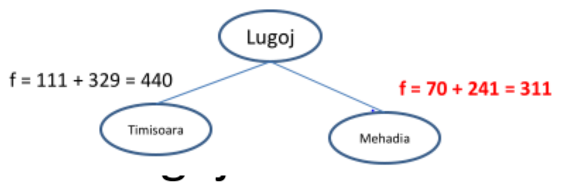
2. **Informed Search.**

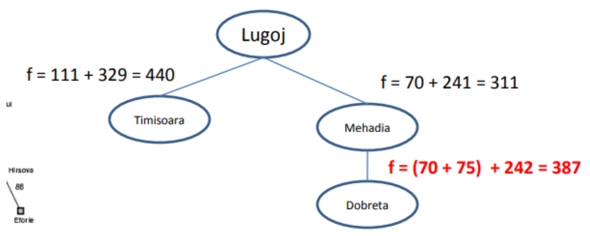


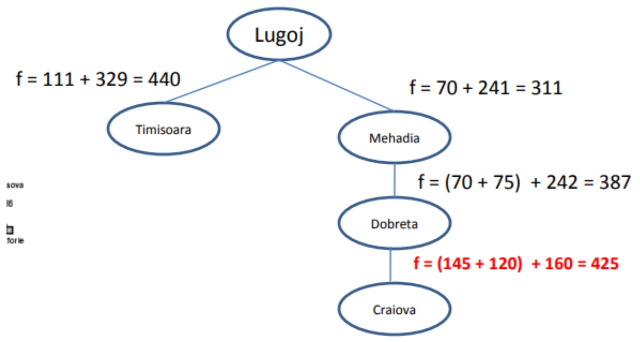
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

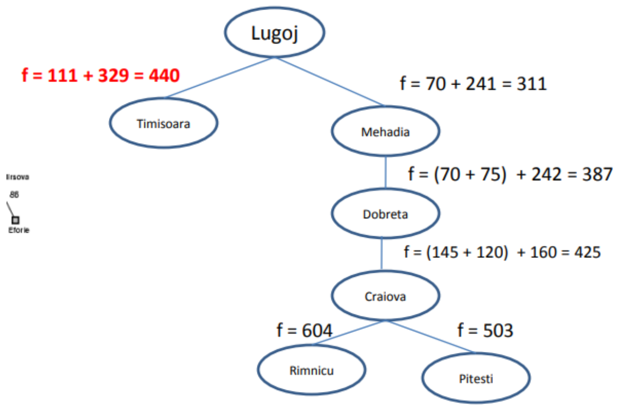
Applying the A* search on this graph to reach Bucharest from Lugoj using the straight-line distance heuristic function. Show the sequences of nodes which are considered by algorithm. Show the f,g,and h for each node

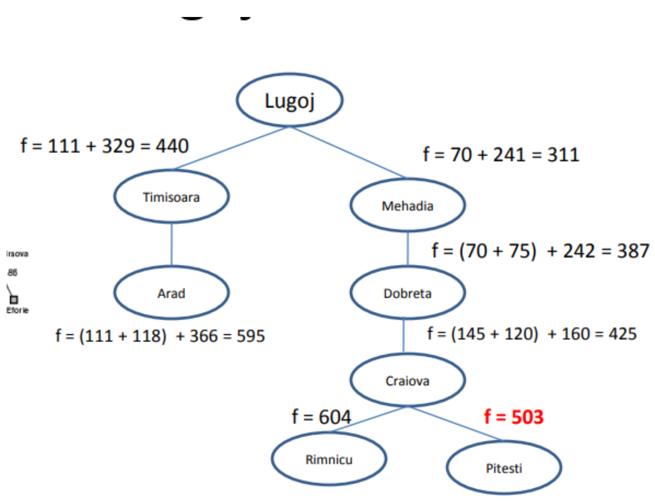
Solution:

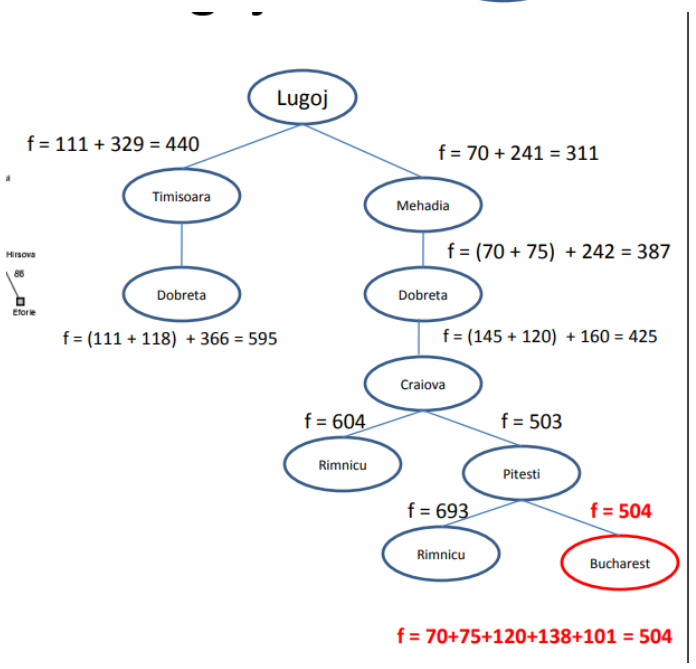




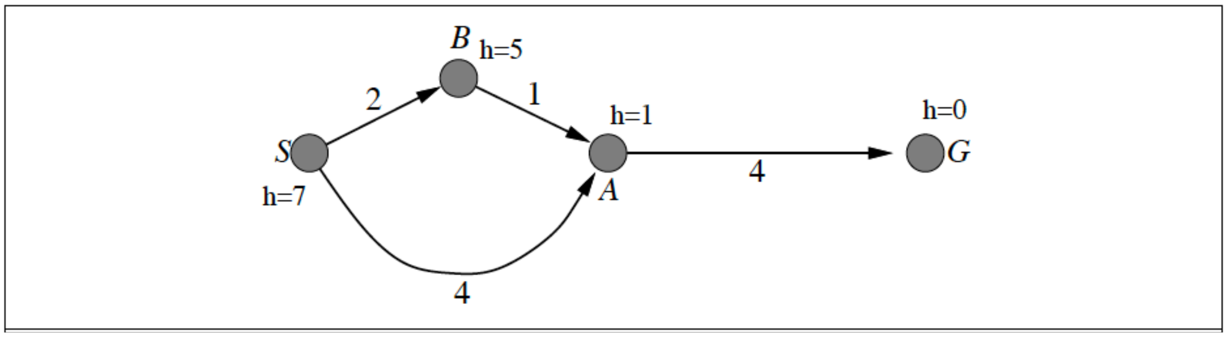

Lugoj
f = 111 + 329 = 440
f = 70 + 241 = 311
Timisoara
Mehadia
f = (70 + 75) + 242 = 387
Dobreta
f = (145 + 120) + 160 = 425
Craiova
sova
Eforie


Lugoj
f = 111 + 329 = 440
f = 70 + 241 = 311
Timisoara
Mehadia
f = (70 + 75) + 242 = 387
Dobreta
f = (145 + 120) + 160 = 425
Craiova
f = 604
f = 503
Rimnicu
Pitesti
Hirsova
86
Eforie





3. **Properties of a heuristic.** With the numbers on the arcs as ground truth costs and the h values as heuristics in the below diagram, is h(n) admissible? Is h(n) consistent? (Explicitly use the definitions of admissible and consistency in your argument. Show that A* using GRAPH-SEARCH on the state space shown below returns a suboptimal solution.



Solution:

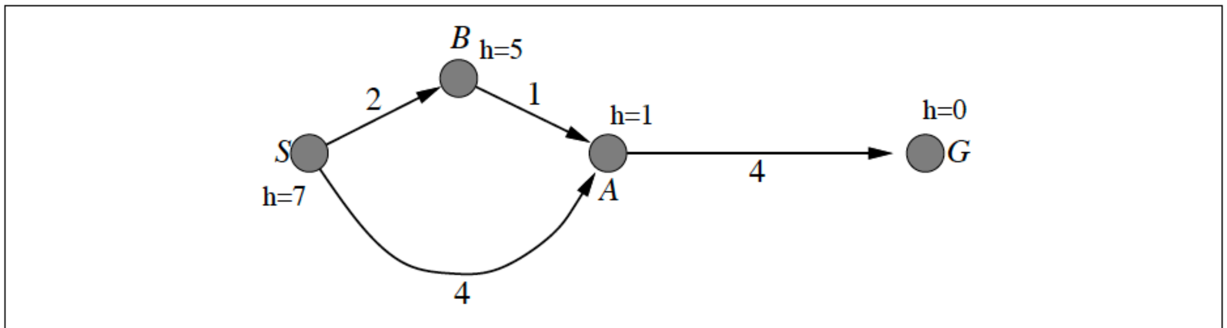
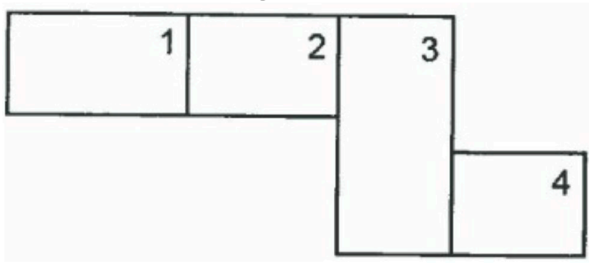


Figure S3.2 A graph with an inconsistent heuristic on which GRAPH-SEARCH fails to return the optimal solution. The successors of S are A with $f=5$ and B with $f=7$. A is expanded first, so the path via B will be discarded because A will already be in the closed list.

4. **Constraint Satisfaction Problem.**

In a zoo several animals were brought and four enclosures were built. Because there are more animals than enclosures, some animals have to be in the same enclosures as others. However, the animals are very picky about who they live with. Can you plan where each animal goes?

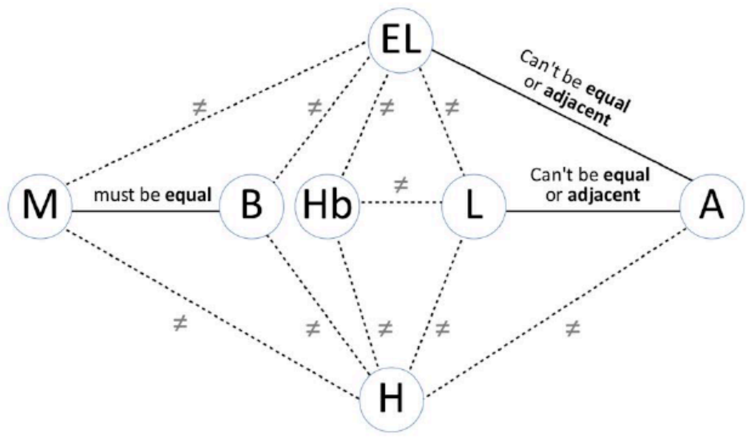


The animals chosen are a LION, ANTELOPE, HYENA, EVIL LION, HORNBILL, MEERKAT, and BOAR. They have given you the plans of the zoo layout.
Each numbered area is a zoo enclosure. Multiple animals can go into the same enclosure, and not all enclosures have to be filled.
Each animal has restrictions about where it can be placed.
1. The LION and the EVIL LION hate each other, and do not want to be in the same enclosure.
2. The MEERKAT and BOAR are best friends, and have to be in the same enclosure.
3. The HYENA smells bad. Only the EVIL LION will share his enclosure.
4. The EVIL LION wants to eat the MEERKAT, BOAR, and HORNBILL.
5. The LION and the EVIL LION want to eat the ANTELOPE so badly that the ANTELOPE cannot be in either the same enclosure or in an enclosure adjacent to the LION or EVIL LION.
7. The LION annoys the HORNBILL, so the HORNBILL doesn't want to be in the

LION's enclosure.
8. The LION is king, so he wants to be in enclosure 1.

Using the reduced domains provided below, find one solution using depth first search with forward checking and propagation through domains reduced by any number of values (propagation through reduced domains). Show your work by filling out the domain worksheet on this page and drawing the search tree on the next page. Break ties in numerical order (1,2,3,4).

Constraint graph for this problem

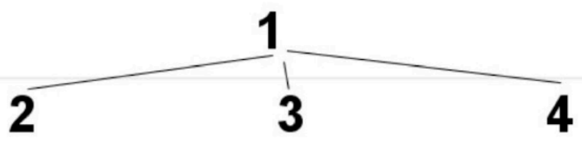


Domains for this problem

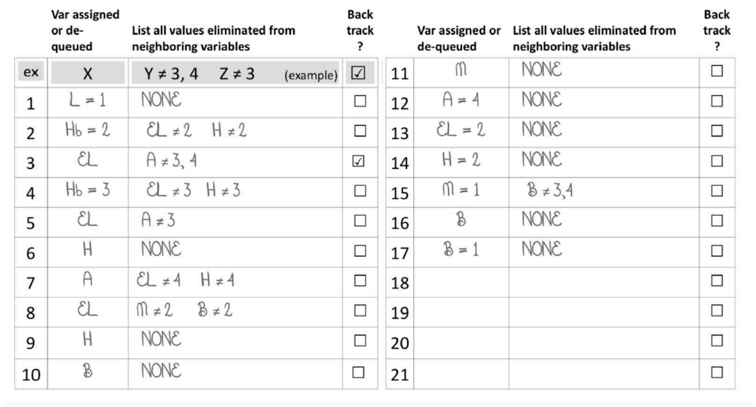
L	1
Hb	2 3 4
A	3 4
EL	2 3 4
H	2 3 4
M	1 2 3 4
B	1 2 3 4

Reminder:
Fill out this worksheet as you draw your search tree. There may be more rows than you need. Every time you assign a variable or remove a variable from the propagation queue, fill out a new row in the table. (The same variable might appear in more than one row, especially if you have to backtrack.) In that row, indicate which variable you assigned or dequeued; write its assigned value if it has one (e.g. X=x), otherwise just write its name (X). In the second column, list the values that were just eliminated from neighboring variables as a result. If no values were just eliminated, write NONE instead. If your search has to backtrack after assigning or de-queuing a variable: first, finish listing all values eliminated from neighboring variables in the current row. Next, check the backtrack box in that row. Then, continue with the next assignment in the following row as usual. At some point, you might add several variables to your propagation queue at once. Break ties by adding variables to your propagation queue in alphabetical order.

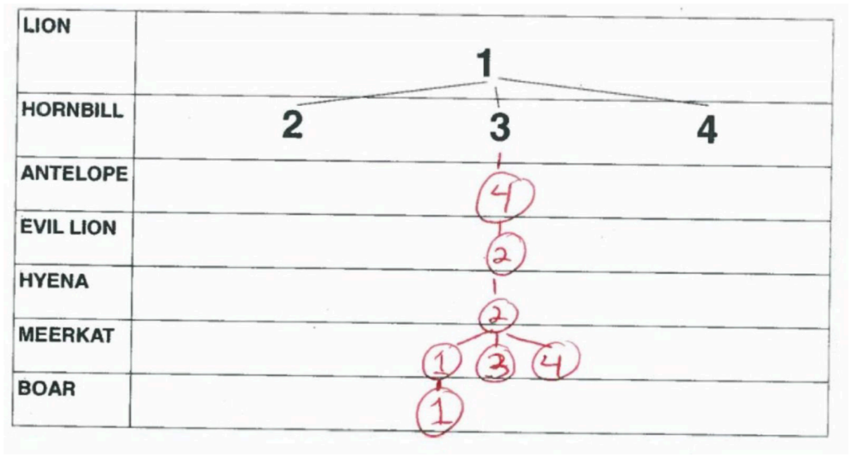
	Var assigned or de-queued	List all values eliminated from neighboring variables	Back track ?
ex	X	Y ≠ 3, 4 Z ≠ 3 (example)	☑
1			☐
2			☐
3			☐
4			☐
5			☐
6			☐
7			☐
8			☐
9			☐
10			☐
11			☐
12			☐
13			☐
14			☐
15			☐
16			☐
17			☐
18			☐
19			☐
20			☐
21			☐

LION	
HORNBILL	
ANTELOPE	
EVIL LION	
HYENA	
MEERKAT	
BOAR	

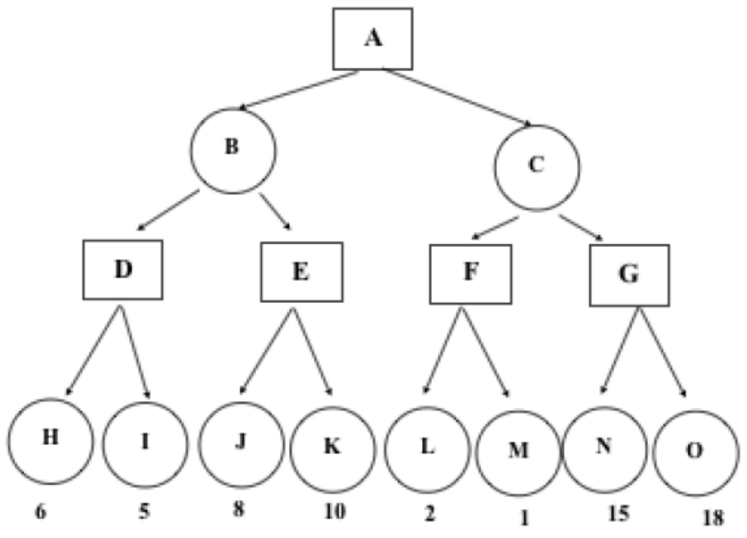
Solution:



	Var assigned or de-queued	List all values eliminated from neighboring variables	Back track ?
ex	X	Y ≠ 3, 4 Z ≠ 3 (example)	☑
1	L = 1	NONE	☐
2	Hb = 2	EL ≠ 2 H ≠ 2	☐
3	EL	A ≠ 3, 4	☑
4	Hb = 3	EL ≠ 3 H ≠ 3	☐
5	EL	A ≠ 3	☐
6	H	NONE	☐
7	A	EL ≠ 4 H ≠ 4	☐
8	EL	M ≠ 2 B ≠ 2	☐
9	H	NONE	☐
10	B	NONE	☐
11	M	NONE	☐
12	A = 4	NONE	☐
13	EL = 2	NONE	☐
14	H = 2	NONE	☐
15	M = 1	B ≠ 3,4	☐
16	B	NONE	☐
17	B = 1	NONE	☐
18			☐
19			☐
20			☐
21			☐



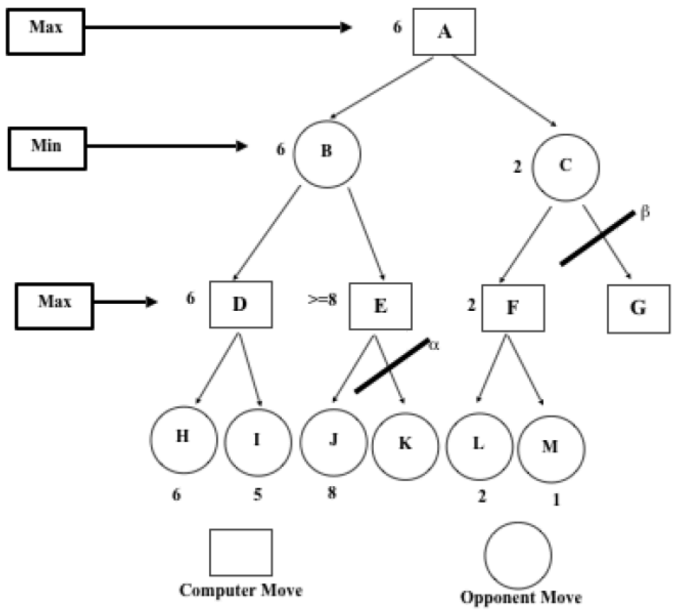
5. **Alpha-Beta Pruning.** Given the following search tree, apply the alpha-beta pruning algorithm to it and show the search tree that would be built by this algorithm. Make sure that you show where the alpha and beta cuts are applied and which parts of the search tree are pruned as a result.



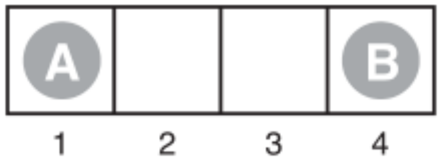
Solution:
Assume we have evaluated, using depth first search down to node I. This means we can maximise node D to a value of 6. If we now continue the search we will eventually evaluate node J and assign it a value of 8. At this point we do not need to evaluate K (or any of its siblings); for the following reasons.

- Node E already has a value of at least 8 (as MAX is trying to maximise this value so it cannot take on a smaller value).
- Node E is already greater than node D and, as MIN will be trying to minimise these two values, it is always going to choose D over E.
- Therefore, we can cut off the search at this point (shown an the diagram).

Similarly, we can cut of the search from node G downwards.
- Having evaluated F to 2, C cannot be more than this value (as C is trying to minimise).
- B is already greater than 2.
- Therefore, MAX is always going to prefer B over C, so we cannot cut off the rest of the search below C.



6. Adversarial Search. A two-player is described below.



The player A will move firstly. Each player could only move to empty slot in either direction. If the opponent is the neighbor of yours, you could jump over him if there is any space area. For example, if B is on 3, A is on 1, A could jump over B to 4. The game would end when one player reach the opposite of the board. If A reaches 4 firstly, the value of the game to A +1, if B reaches 1 firstly, the value of the game to A -1.

a. Draw the complete game tree, using the following conventions:

- Write each state as (sA, sB), where sA and sB denote the token locations.
- Put each terminal state in a square box and write its game value in a circle.
- Put loop states (states that already appear on the path to the root) in double square

boxes. Since their value is unclear, annotate each with a “?” in a circle.

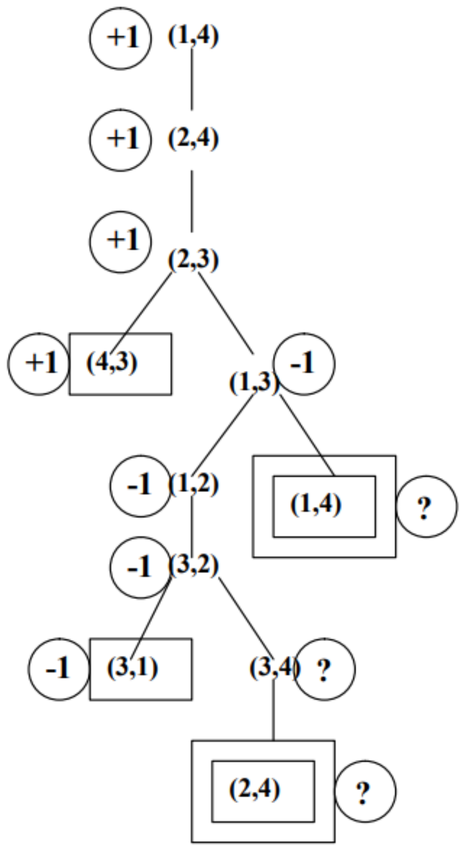
b. Now mark each node with its backed-up minimax value (also in a circle). Explain how you handled the “?” values and why.

c. Explain why the standard minimax algorithm would fail on this game tree and briefly sketch how you might fix it, drawing on your answer to (b). Does your modified algorithm give optimal decisions for all games with loops?

d. This 4-square game can be generalized to n squares for any n > 2. Prove that A wins if n is even and loses if n is odd.

Solution:

a.



b. The “?” values are handled by assuming that an agent with a choice between winning the game and entering a “?” state will always choose the win. That is, min(–1,?) is –1 and max(+1,?) is +1. If all successors are “?”, the backed-up value is “?”.

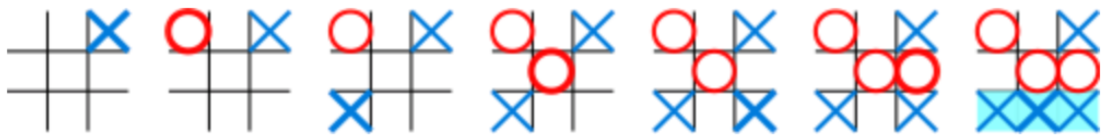
c. A standard minimax is depth-first and would go into an infinite loop. It can be fixed by comparing the current state against the stack, and if it occurs on the stack, then by returning a "?” value propagation of states with differing values. For example, with draws or wins of different degrees, it is not clear how to handle these in the propagation process. There are more complex methods that determine whether the node has a well-determined value, as in this example, or is really an infinite loop. Both player would prefer to stay in the loop or have no choice. That case could be regarded as draw.

d. Suppose the size of boxes is larger than 4. The only moves made by two player is to move to the middle of the game in order to get closer to each size obviously the heuristic function. If the number of boxes is odd, A would stand on the middle box faster than B. Than B could jump over A and get closer to another size than A being. Afterall, the only available option for them is to continue move to each side. Since B is closer to another side one step more, B will always reach goal firstly.
If the number of the boxes is even, A and B will meet at the middle of squares both of whom are same far away from another side. Since A is the first turn, A could move now and jump over B. This step will permanently give A advantage to reach another side and win the game.

7. **Programming assignment.**

Tic-tac-toe
Tic-tac-toe is a pencil-and-paper game for two players, X (ascii value 88) and O (ascii value 79), who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game. Empty space is represented by _ (ascii value 95), and the X player goes first.

Here is an example game won by the first player, X:



The function alpha_beta_minimax() takes in a state of the board. ***You should complete this function*** to print 2 space separated integers *r* and *c* which denote the row and column that will be marked in your next move. The top left position is denoted by (0,0).

Note: the state is a class that has the board (2d array which captures values in 3X3 grid), current player, whether its max-player or not, next move.

Please implement the alpha_beta_minimax() using the included helper functions in alpha_beta.py included as a file with this assignment.

How does it work?

Your code is run alternately with the opponent bot for every move.

Example input:

X

___

___

_XO

Example output:

1 0

Explanation

Next move is

___

X__

_XO