HW1
CS5100 Fall 2014
Matt Roy

1A.
A -> B -> C -> D -> E -> G

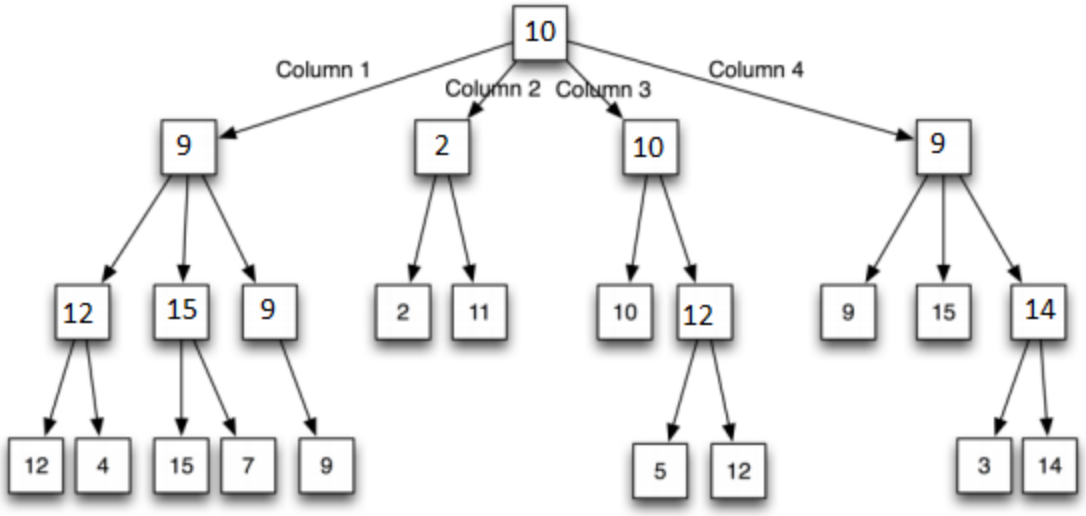
1B.
A -> B -> C -> D ->E ->G

1C.
A -> C -> E -> G

1D.
A -> C -> D -> G

1E.
A -> B -> C -> D -> G

2A.



2B.
Max should select Column 3.

3A.
States - Specifies the location of the each of the two friends.
Initial State - Both friends in their starting cities.
Action - An action is friend1 moving to a neighbor city and friend2 moving to a neighbor city.
Transition Model - The two friends arrive in the cities they chose to move to.
Goal Test - Are the friends in the same city.

Path Cost - The path cost is max(d(i1,j1), d(i2,j2)). We have to wait for whichever friend takes the longest to get their city.

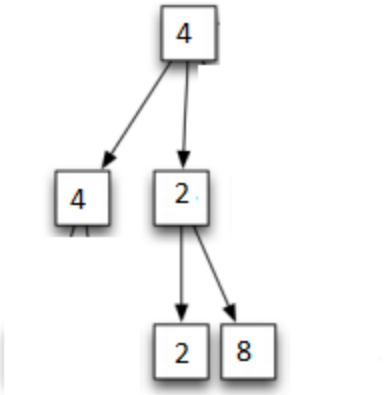
3B.
i. Since the cost is the driving distance, the road between two cities must be at least as long as the straight line distance. So D(i,j) is admissible.
ii. The road could be less than twice the straight line distance, so 2D(i,j) would not be admissible. Since an admissible heuristic must always be less than the actual cost, this is not guaranteed to be admissible.
iii. The road could not be any less than the straight line distance, so 1/2D(i,j) would still be admissible even though it is not as ideal as D(i,j).

3C.
If we assume that both friends must move on each turn, they could miss each other. In a very simple case, imagine a map that has only two cities, each friend starts in a different one. The only option is for each friend to drive to the other city, bypassing each other on the road.

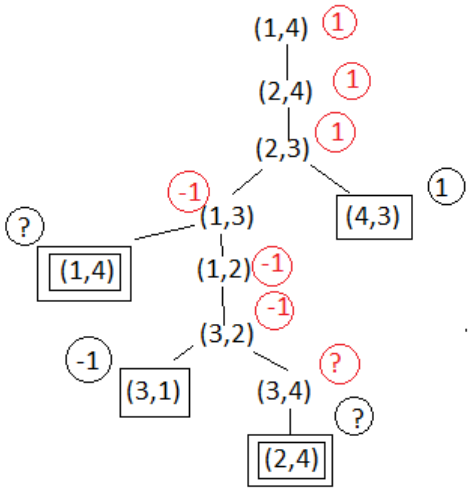
3D.
If one of the friends comes back to the same city twice, he could have made more progress by moving towards the other friend rather than making a loop at the revisited city. So he should have made a different choice rather than looping back.

4.
If MIN makes a suboptimal choice, we are guaranteed that there exists a greater utility value in that branch of the tree. Since we already took the minimum choice, the other choices must be greater than the optimal path.

If we have a tree that has a mix of large utility values and small utilities near each other, MIN could choose a path that leads it down a larger utility value path. In the case below, MAX would choose 4, if MAX had taken 2, a suboptimal MIN would have mistakenly chosen path 8, and it would have led to a higher utility value.



5A.



5B.
For the loop states, I ignored them. Since they lead to a state we have already seen, the agent could have chosen a different move at that point and avoided the loop.

5C.
The min max algorithm would never stop going down a loop path. My modification would be to track states that we have seen and if we get to a state that we have seen before, prune that entire branch out of the tree. It's important to track not only the positions of the markers but who's turn it is. The same state with a different player moving will lead to a different set of min/max outcomes. As long as the game is deterministic, any move made at a subsequent visit to a state, could have been made the first time. Therefore this algorithm should solve any problem with loops.

5D.
When there are an even amount of spaces, when it's A move, there will always be an even amount of spaces between A and B or A and B will be adjacent. If A jumps B, it will end up close to the goal than B and will get there in less turns. As long as A always moves toward B, A will always get the chance to jump B and will end up closer to the it's goal than B to it's goal.

The same is true of B in an odd numbered game. B will end up with an even amount of spaces or be adjacent to A on it's turn.