This project uses Python 3

I used Spyder IDE to code.



These are the only imports used for Q1.

Q3 has no imports.

A brief description of the methods is in the comments in the code.

Q1. Informed Search (10pts)

I hardcoded the vertex list representation of the undirected weighted graph at the top of the Q1.py script.

1. Greedy: -

* States Expanded: S,e,r,f,G,
* Path Returned: S-e-r-f-G
* Cost: 20
* Writeup:

I used priority queue to implement this search. I insert the starting node, its heuristic, and the path to it (itself). I loop until the priority queue is empty.

Since all greedy search does is look at the lowest heuristic neighbor node, and go to it, I added all the neighboring nodes to the priority queue, using their heuristic value as the priority, along with the path to the neighbor.

If the node being visited currently is the target, then I print the cost and the path to the target.

1. A\* Search: -

* States Expanded: S,d,e,r,f,G,
* Path Returned: S-d-e-r-f-G
* Cost: 38
* Writeup:

I used a priority queue to implement this search. I insert the starting node, its heuristic, and the path to it (itself). I loop until the priority queue is empty.

A\* uses the basis of greedy search, but simply adds the weight of the edge being traversed to the heuristic and chooses the lowest sum. The implementation is the same as Greedy, however when the neighbor is being inserted into the priority queue, the priority is the sum of the heuristic of that neighbor, and the edge weight from the current node to the neighbor. This gives us the optimal outcome.

Q2. Constraint Satisfaction Problems (5pt)

1. [1 pt] After a value is assigned to A, which domains might be changed after **forward checking** for A?

Answer: B-D-E.

Forward checking entails checking the neighboring nodes after a value is assigned to a particular node. Hence A’s neighbors affected are B, D and E

1. [1 pt] After a value is assigned to A, then **forward checking** is run for A. Then a value is assigned to D. Which domains might be changed as a result of running **forward checking** for D?

Answer: E-F.

After A is assigned a value, and forward checking is run, B, D, E have their domains changed. Out of the available values, D is assigned one, so D’s non visited neighbors must be forward checked. D’s neighbors are E and F

1. [1 pt] After a value is assigned to A, which domains might be changed as a result of enforcing **arc consistency** after this assignment?

Answer: B, C, D, E, F, H, G

When A is assigned a color, enforcing arc consistency would mean deleting the unavailable color from the domain of neighboring nodes, and their neighbors as well. As such, we must step through all connected node arcs and update their domains. Say A is assigned White, then B, D, E would have White removed from their domains, leaving only Black as an option. Now, C, F, G, H would all have Black removed from their domains since their parent node’s domain has been changed.

1. [1 pt] After a value is assigned to A, and then **arc consistency** is enforced. Then a value is assigned to D. Which domains might be changed after enforcing **arc consistency** after the assignment to D?

Answer: E, F, G, H

When A is assigned a value, and arc consistency is enforced, and then D is assigned a value, all non-assigned nodes connected to D will have their arc consistency enforced. As such, E, F will have their domains updated. Since E and F have their domains updated, G and H will also have a limited domain to choose from, hence their domains are also updated.

1. [1 pt] Is there a valid solution for assigning colors to all graph nodes? Why?

Answer: No.

The existence of cycle A-D-E in the graph means that its not possible for a Binary system to correctly assign colors to the graph. If A is assigned White, then both D and E must be assigned Black, but D and E are neighbors, so they cannot be assigned the same color.

Q3. Adversarial Search (5pts)

I modified the Binary Tree class given in the Homework assignment to correctly include the node values. I then created the tree given in the question by instantiating the root and inserting nodes accordingly.

The leaf nodes were numbered from left to right. The subtree nodes were labeled by Height, and path. EG: H\_3\_L\_L\_L = From root, Left, Left, Left at height 3. Leaf\_1 = 1st Leaf from the left

1. Minimax: -

Return Value: Leaf\_1.obj – The terminal state node Leaf\_1 in this instance.

Output Value: 3

Output Path: ['Leaf\_1', 'H\_3\_L\_L\_L', 'H\_2\_L\_L', 'H\_1\_L', 'Root’] (Reverse for top to bottom)

The solution to this problem was broken down into 3 functions.

A value function which starts the recursive calls to min\_value and max\_value depending on the state of the root. It calls min\_value if root state is Max, and max\_value if root state is Min. This function prints the final path to the selected terminal node and returns the terminal node.

A min\_value function. min\_value gets the minimum of its children’s values. If the node passed in is a leaf, then it starts the return path from the leaf, and returns the value, the node object, and the retPath of that leaf.

A max\_value function. max\_value gets the maximum of its children’s values. If the node passed in is a leaf, then it starts the return path from the leaf, and returns the value, the node object, and the retPath.

A diagram of a diagram

Description automatically generated

Output Value: 3

Output Path: Root –> H\_1\_L – > H\_2\_L\_L –> H\_3\_L\_L\_L –> Leaf\_1

1. Alpha Beta Pruning: -

Return Value: 3, Leaf\_1

Output: 3

This problem was solved using one function. This function ab\_pruning runs recursively, starting from the root node, the depth, the state Min or Max, alpha and beta.

If the current node is a leaf node, then return its value and id.

If the current state is Max, then initialize the current best value to -infinity. Loop through the current node’s children, starting with its left child first, and recursively call ab\_pruning on the child, with “Min” as the state. Update the best value and alpha with the returned value. If the returned value is >= beta, then it doesn’t matter what the other children have, since the minimizer will never go down that path. So prune those children.

If the current state is Min, then initialize best value to +infinity. Loop through the current node’s children, starting with its left child first, and recursively call ab\_pruning on the child, with “Max” as the state. Update the best value and beta with the returned value. If the returned value is <= alpha, then it doesn’t matter what the other children have, since the maximizer will never go down that path. So prune those children.

1. Terminal States Pruned: Leaf\_4, subtree H\_3\_L\_R\_R , Leaf\_10 and Leaf\_14

Leaf\_4 : alpha = 3, beta = 2

Follow ab\_pruning till we get to H\_3\_L\_L\_R. The algorithm checks Leaf\_3, and updates H\_3\_L\_L\_R’s value to 2, its alpha is already 3 from H\_3\_L\_L\_L, its beta is infinity. The algorithm then updates beta for H\_3\_L\_L\_R to its value, which is 2. It then checks if beta <= alpha. In this case 2<=3, which means that since we’ve found a value <=3, the minimizer H\_3\_L\_L\_R should not care what the other children might be, since the maximizer H\_2\_L\_L will always pick 3 over any value <=2 in its right subtree. Hence, prune Leaf\_4

Subtree H\_3\_L\_R\_R: alpha = 7, beta = 3

Follow ab\_pruning till we get to H\_2\_L\_R. By following ab\_pruning on its left subtree, we get the updated alpha value to be 7, the beta value 3, is inherited from the minimizer H\_1\_L. Before exploring subtree H\_2\_L\_R, the algorithm checks if beta<=alpha. 3<=7 which means no matter what value is on the right subtree for H\_2\_L\_R, it will have to be >=7 to replace 7. And since the minimizer H\_1\_L will never choose that option, we can prune that subtree.

Leaf\_10: