Problem 1: Informed Search

Part (1): Greedy Search

Thought Process

For my greedy search algorithm, I worked off of the idea of it being a depth first search that rather then using a stack, pops off the node with the most promising heuristic value. Doing the calculations by hand yeilded the same results as my algorithem.

Code and Results

The code can be found in `/src/InformedSearch.py` as the function named `greedySearch`. The program results can be found in `/output/InformedSearch.txt` under the `Greedy Search:` section. The results read:

```
Greedy Search:
Expanded to: ['S', 'e', 'r', 'f', 'G']
Path returned: ['S', 'e', 'r', 'f', 'G']
```

Part (2): A* Search

Thought Process

For my A* search algorithm, I used the idea of a queue from Uniform Cost Search and the heuristics from greedy search togeather along with the appropriate math to calculate which node to expand to next.

Code and Results

The code can be found in `/src/InformedSearch.py` as the function named `AStarSearch`. The program results can be found in `/output/InformedSearch.txt` under the `A* Search:` section. The results read:

```
A* Search:
Expanded to: ['S', 'd', 'e', 'r', 'b', 'a', 'f', 'G']
Path returned: ['S', 'd', 'e', 'r', 'f', 'G']
```

Part (3): Admissability

Yes this graph with hueristic h is admissable. This is because for each node, its heuristic is opimistic, satisfying the inequality `0 <= h(n) <= h*(n)`. In otherwords, for every single node, the true shortest distance to the goal counted along the edges of the graph, is greater then the heueristics value for that node. To compute this, I went node by node with a sheet of paper to verify that this held for all nodes and their heuristics.

Part (4): Consistency

Yes this graph with hueristic h is consistent. This is because we can show there does not exist an "arc" from a node `A` to a node `C` for which the inequality `h(A) - h(C) <= cost(A to C)` is violated. In the case for our graph, I have again gone through and verified that every single arc fails to violate this inequality, the diffrence of the heuristic between the nodes is always smaller or equal to the true cost between nodes.

Problem 2: Constraint satisfaction problems

Problem 3: Adversarial search

Part (1): Minimax Search

Thought Process

The minimax search is a straitforward process to implment recursivly based on the node type.

Code and Results

The code can be found in `/src/AdversarialSearch.py` as the function named `minimaxSearch`. The program results can be found in `/output/AdversarialSearch.txt` under the `Minimax Search:` section. The results read:

```
Preforming Minimax Search:
The chosen terminal state: 3
```

Part (2): Manual Minimax Computation

image

Part (3): alpha-beta pruning

Thought Process

The minimax search with alpha beta pruning utilizes an alpha and beta passed along the recursive minimax search to help discover if subtrees can be pruned.

Code and Results

The code can be found in `/src/AdversarialSearch.py` as the function named `alphaBetaPrune`. The program results can be found in `/output/AdversarialSearch.txt` under the `Minimax Search with alphaBeta Pruneing:` section. The results read:

```
Preforming Minimax Search with alpha-beta pruning:
Pruned llrl alpha: 3 beta: inf
Pruned lrl alpha: -inf beta: 3
Pruned rlll alpha: 3 beta: inf
Pruned rrll alpha: 3 beta: 4
Pruned rrrr alpha: 3 beta: 4
Pruned rr alpha: 3 beta: 4
The chosen terminal state: 3
```

Part (4): alpha-beta pruning results

Results

The following branches are cut off at some point while searching, almost all of them are cut off due to the alpha of 3 being the best available option to the maximizer and the beta of 4 being the best option for the minimizer.

```
Branch llrl     alpha: 3       beta: +inf
Branch lrl      alpha: -inf    beta: 3
Branch rlll     alpha: 3       beta: +inf
Branch rrll     alpha: 3       beta: 4
Branch rrrr     alpha: 3       beta: 4
Branch rr       alpha: 3       beta: 4
```

image

Since this is just a more efficient Minimax search, we do not obtain the he output path, but rather just obtain the chosen terminal state.