CSC520 Fall 2020 Homework 2
Due September 16^{th} at 11:59pm EST

This assignment includes both conceptual and code questions. It must be completed individually. You may not collaborate with other students, share code, or exchange partial answers. Questions involving answers or code must be emailed to the instructor or TAs directly or discussed during office hours. Your answers to the conceptual questions must be uploaded to Moodle as a pdf file titled `Assign2-<unityid>.pdf`. Your source code must be submitted as a self-contained zip called `Assign2-<unityid>.zip` All code must be clear, readable, and well-commented. You may only use libraries to provide standard data structures or file parsing. All third party library use must be checked with Dr. Lynch or the TA *before* submission.

Note: The code will be tested on the NCSU VCL system. You are advised to test your code there before submission.

Question 1 (15 points)

Your goal is to create an automatic scheduling system for your 500 & 600 level grad courses. You can have at minimum 9 and at most 12 credit hours each semester. Consider the grad course information listed in the online course catalogue and frame this as a constraint satisfaction problem:

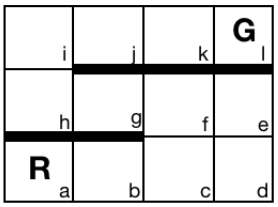
a. What are the variables in your problem?

b. What is the domain of each variable?

c. Give a specification for the constraints that would be imposed using mathematical and logical operators.

Question 2 (5 points)

Robot R is in position a in the given maze and its' goal is to reach G in position l. The possible set of actions for this agent is Up, Down, Left, and Right. The thick lines indicate walls and every time that an action is against a wall, nothing happens and the robot stays in the same state. For each action, there is a *slip* chance for the agent to move in the opposite direction. Assuming the following set of consecutive actions $A = \{\rightarrow, \rightarrow, \uparrow, \leftarrow, \leftarrow\}$, indicate the belief state after executing each action $a \in A$.



Question 3 (15 pts)

A set if game trees are shown below. Assume the root node represents the current game state, and that the system must now select the child of the root with the maximum value. Squares and circles represent minimizing and maximizing functions with static position values in the leaf nodes. Perform both minimax search and α-β pruning as called for below.

a. (5 pts) Inside each node of below game tree, indicate the minmax values for all nodes, including the root, assuming no pruning. What is the best first move for max?

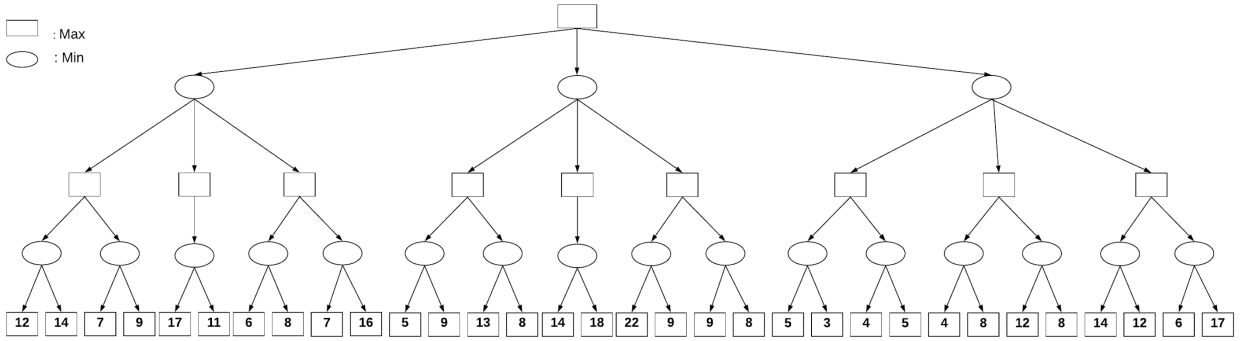


Figure 1: Game Tree for Q4.a

b. (10 pts) Using the game tree below, to the left of each node, indicate the $\alpha - \beta$ intervals for all interior nodes and show how the intervals change dynamically during search, assuming the move generator proceeds left to right. Draw a short line through the edges that would be pruned by α-pruning, and draw an α next to these lines. Draw a short line through the edges that would be pruned by β-pruning, and draw a β next to these lines.

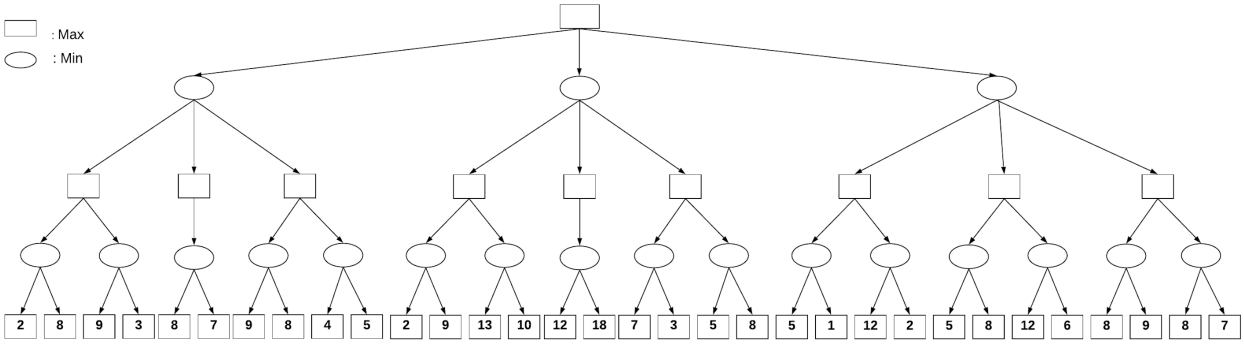


Figure 2: Game Tree for Q4.b

Question 4 (65 pts)

You are conducting a series of randomized controlled trials for blood plasma. But doing so required correctly matching donated plasma to targeted patients. You have been given two datasets that correspond to simple donor selection problem. Your job is to pick the *right* sample to give to each individual. The two datafiles reflect different experimental runs. In `BloodDataA.csv` the samples are moderatly effective, or else benign. Each correct selection therefore is scored as a 1 and each incorrect selection is scored as a 0. In `BloodDataB.csv` the doses are either very effective (score 100) or provoke a mild reaction (score -1).

Your task in this assignment is to implement a simple multi-armed bandit algorithm that selects plasma based on their normalized probability and then analyze its performance on the datasets with different parameter settings. Your code should be called as follows:

```
java -jar Bandit2.jar <exp> <dist> <decay> <rwt> <w0> <infile> <outfile>
```

```
python Bandit2/BanditAlg.py <exp> <dist> <decay> <rwt> <w0> <infile> <outfile>
```

Where:

- `exp` is the exploration rate.
- `dist` is the uniform distribution parameter, use the same distribution for all values.
- `decay` is the decay rate.
- `rwt` is the reward weight function.
- `w0` is the initial weight value for the ads.
- `infile` is the appropriate ad data file.
- `outfile` is the output log described below.

As your code is run it should write out a log to the outfile in the following format:

```
Step,Choice,CurrReward,TotReward,p1,w1,...p10,w10
```

Where:

- `Step` is an incremental step count.
- `Choice` is the sample chosen on this step.
- `CurrReward` is the score for the current action.
- `TotReward` is your running total score.
- `pi` Is the probability of selection assigned to each item after the decision.
- `wi` is the weight assigned to each item after the decision.

Once you have implemented the code you should *analyze* its performance on the files for different parameter values and use it to answer the following questions. For each question report the changes you observe, the parameters you test, and back up your analysis with appropriate values or visualizations.

a. How is the performance affected by changing the decay parameter from a *slow* decay rate (e.g. 0.05) to a fast decay rate (e.g. 0.3) and why?

b. Are more conservative exploration rates better or worse for the contract models and why?

c. How does varying the reward weight affect the performance?

As always your code should be clear, readable, and well documented, and it should include a README file that explains how to build and execute it on the servers.

Question 5 (*15 pts extra credit*)

For this extra credit, build upon your implementation of Q4 to select parameters via grid search. This code should iterate over possible parameter values to identify the highest payoff values. The code should be called as follows:

```
java -jar Bandit2Grid.jar <maxiter> <step> <infile> <outfile>

python Bandit2/BanditGrid.py <maxiter> <step> <infile> <outfile>
```

Where `maxiter` represents the maximum number of iterations that the code will take before stopping and `step` represents the increment unit for each of the parameter values. Your code should store a running log to the output file in the format described above with the following additions. At the start of each execution you must report the current parameter values as follows:

```
** Start round,exp,dist,decay,rwt,w0 **
```

Where `round` represents the number of times the grid search has run the experiment. Finally at the end of the experiment you must report the best scoring parameter values as follows:

```
** BestScore, exp,dist,decau,rwt,w0,TotReward **
```