Artificial Intelligence: Representation and Problem Solving
15-381, Fall 2009

Homework 2

DUE: Wednesday, September 30th, 2009, 3:00 P.M.

REMINDER: Due at the beginning of the class on the day it is due.

[**Maximum points: 100**]

Handin instructions

- Put the following in `/afs/andrew/course/15/381-f09/handin/`*your andrew id*`/hw2/`
  - The output of your code in the 4 files `output1.txt` - `output4.txt`
  - The andrew ID's of your team members in `team.txt`
- Bring the typeset or handwritten answers to the rest of the questions to class. Answers *MUST* be legible for credit.
- Late homework is due by 3:00 P.M. on the day they are due. Please bring late homeworks to GHC 7027 (slide it under the door if Heather Carney is not there).

Guidelines

- You can work in a group of two people. This group does not need to be the same group as for the other homeworks. You only need to turn in one writeup per group, but you need to include the andrew ID's of the group members. You only need to turn in one program output per group.
- If you have any questions about this assignment, contact the instructors at 15381-instructors@cs.cmu.edu

Problem 1: More Search (30 points)

Answer these questions in one or two concise paragraphs.

1. In lecture, we presented a *hill climbing* algorithm for a discrete state space. Discuss how you might extend *hill climbing* for a continuous state space.

2. Consider searching for an answer to the N-Queens problem using *hill climbing*. Let us define the states as a configuration of all the queens (possibly conflicting) and let us define the neighbors as the states you can reach by moving one queen. What is the eval function for this problem? In this case what do plateaux and local maxima represent in terms of the eval function?

3. *Simulated annealing* will find a global optimum with probability approaching 1 if you lower T slowly enough. Please explain what *simulated annealing* will do in each of the following cases

   (a) What happens if you lower T very quickly?

   (b) What happens if you never lower T?

4. Explain other ways that *hill climbing* can add randomness to avoid getting stuck. (By other we mean other than *simulated annealing*)

5. *Genetic algorithms* are difficult to use well. Please explain three reasons why it is difficult to solve a problem with a *genetic algorithm*.

6. Here are three populations from a genetic algorithm:

   (a) 01010000 10000001

   (b) 10010000 01000001

   (c) 01010000 10100001

   Population b was created by population a reproducing (the crossover is after the second digit). Population c was created by population a mutating (the second string was mutated). Let the fitness function be the number of 1's in a string. What is the maximum fitness of each population? Is there a better crossover point than the one used for creating population b? If so, where is it and why is it better?

Problem 2: Multiplayer Minimax and Alpha Beta Pruning (35 points)

Abe, Mary and Sue Ann are playing each other in a game of Calvinball. (Calvinball is a game for which all the rules are made up. You do not need to know anything about Calvinball for this problem.) Today, the rules dictate that at the end of the game each player gets some points, and the winner is the player with the most points. Each TA would rather win than lose. If a TA is going to win, he or she would like to maximize his or her points. If a TA is going to lose, he or she would also like to maximize his or her points.

2.1: Defining an evaluation function (5 Points)

Define a utility function UTILITY(P_A,P_M,P_{SA}) that outputs the vector (U_A,U_M,U_{SA}), where P_A,P_M,P_{SA} are Abe, Mary, and Sue Ann's respective points and U_A,U_M,U_{SA} are Abe, Mary, and Sue Ann's respective utilities.

2.2: Run Minimax (15 Points)

In reality, if a TA is going to lose, he or she will try to minimize the winner's score. Let us define a new utility function to represent this.

- If $P_A \geq P_M$, P_{SA} then UTILITY'(P_A,P_M,P_{SA}) = (P_A, -P_A, -P_A)
- If $P_M > P_A$, P_{SA} then UTILITY'(P_A,P_M,P_{SA}) = (-P_M, P_M, -P_M)
- If $P_{SA} > P_A$, P_M then UTILITY'(P_A,P_M,P_{SA}) = (-P_{SA}, -P_{SA}, P_{SA})

Apply the Minimax algorithm using this utility function and generate a vector of length 3 for each node of the game tree.

You can do your work on the provided worksheet and just fill in the blanks. The numbers below each leaf are (P_A,P_M,P_{SA}).

2.3: Alpha Beta Pruning (15 Points)

Abe and Sue Ann have decided to team up, to get revenge on Mary. Now Abe and Sue Ann have the goal to get one of them to win with the most points possible. Modify the utility function from 2.2 and run Minimax again, but this time with Alpha Beta Pruning. Draw a line cutting an edge to indicate a branch being pruned. If more pruning would occur with another ordering of the leaves, provide that ordering (for example, if going down the right branch initially would allow you to look at fewer nodes).

You can do your work on the provided worksheet.

Figure 1: Worksheet for 2.2

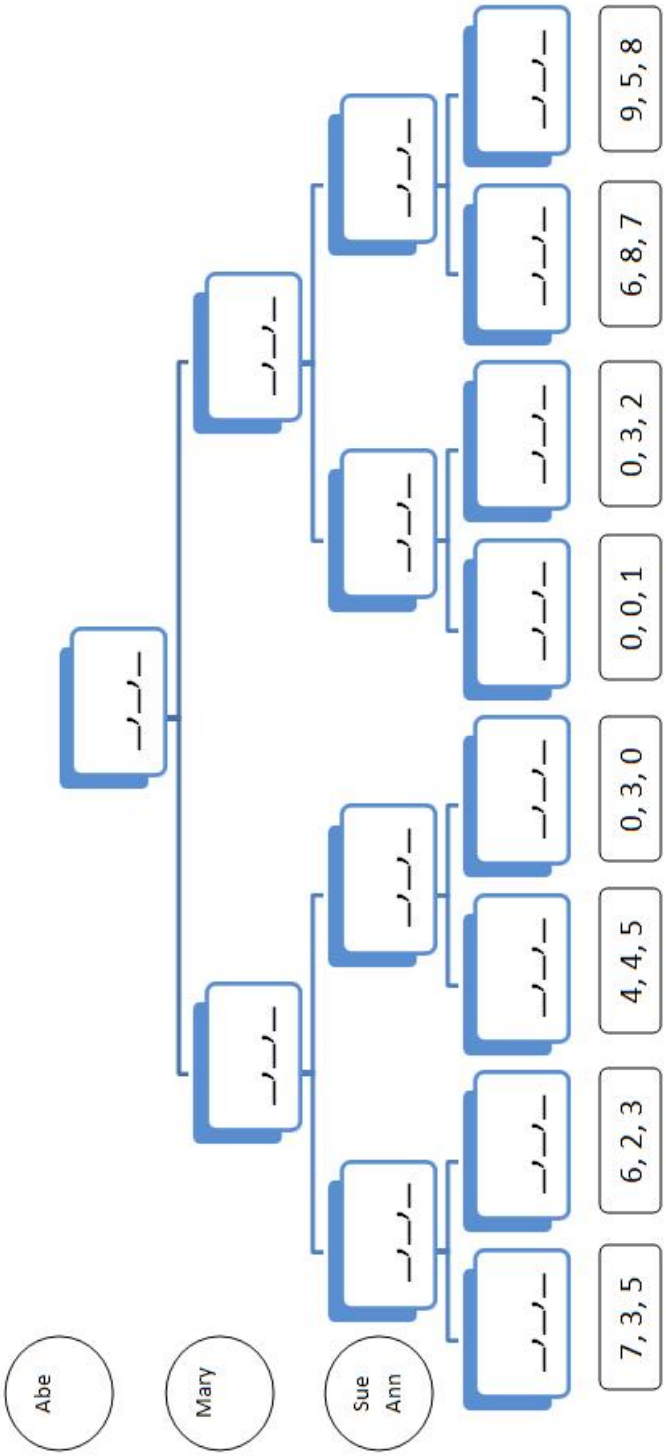
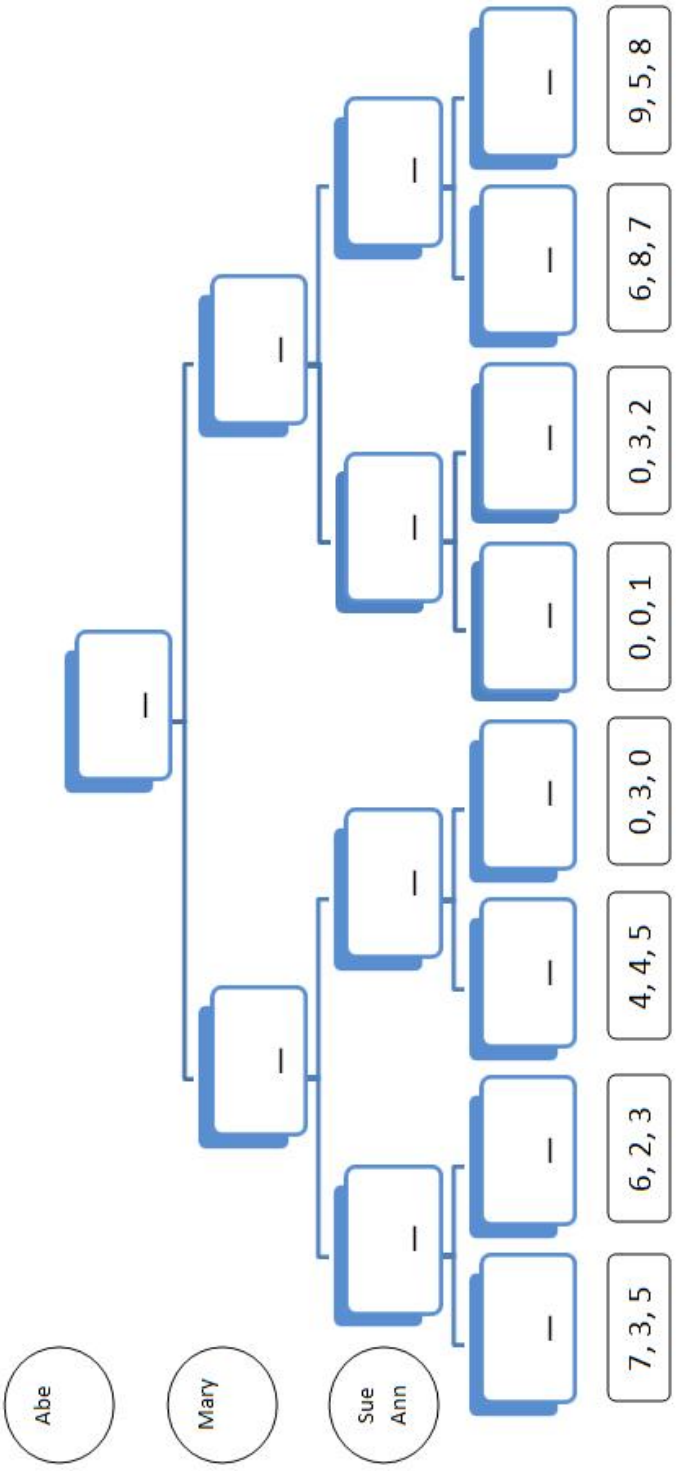


Figure 2: Worksheet for 2.3



Problem 3: Do Gabe's Homework! (CSP) (35 points)

Gabe recently found out that, for his Compilers (15-411) homework, he has to solve a CSP. Gabe realized he could make his AI students solve the problem for him. (Well a simplified version).

The problem is register allocation. When a computer runs a program, the processor has registers in which it stores numbers for quick access. This is similar to keeping numbers in your head when you do mental math. Gabe can keep 2 numbers in his head at once; any more he gets confused. The x86 processor has 8 registers, so it can keep 8 numbers at once.

When Gabe calculates the expression 1+2+4+5, he breaks it down into steps so that he doesn't need to keep all the numbers in his head at the same time. First he calculates 1+2, then 3+4, then 7+5. For the first operation he only needs to remember 1 and 2, for the second he only needs 3 and 4, and for the third 7 and 5. This way, by reusing space in his limited memory, he can calculate with more than two numbers! Similarly, if a program has 10 variables, the processor doesn't need a register for each variable. It can reuse registers!

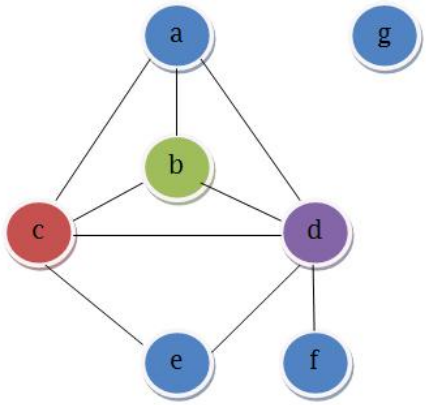
If the program needs to use two variables at the same time, however, then it does need to have the variables in separate registers. In this case we say the two variables interfere with each other. Furthermore, a variable will interfere with other variables between its declaration and its last use. For example, in Figure 3, `a` is defined on line 1 and used for the last time on line 5. Since we can't overwrite `a` until we are done with it, `a` interferes with `b`,`c`, and d.

We can define this problem as a graph coloring problem. First let each variable be a node. Then create an edge between v1 and v2 if v1 and v2 interfere. If we define registers to be colors, we can then color the graph, so no adjacent nodes have the same color. This way no two variables v1 and v2 that interfere with each other will be assigned to the same register!

Figure 3: An example program

1. `a = 1`
2. `b = 2`
3. `c = 4`
4. `d = 7`
5. `e = a + b`
6. `f = e + c`
7. `g = f + d`
8. `return g`

Figure 4: Corresponding graph for the program in Figure 3



We can arbitrarily assign registers to colors. Here we could say blue is r1, green is r2, red is r3, and d is r4, where r1,r2,r3, and r4 are registers.

So now time to do Gabe's homework!

Question 3.1 (35 points)

In `/afs/andrew/course/15/381-f09/hw2` there are 4 files: input1.txt, input2.txt, input3.txt and input4.txt. Each file describes the variables from a program that need to be allocated registers. There are two types of lines in these files.

1. `V` n means that there exists the variable n in this program.
2. `I` m n means that variable m and variable n interfere with each other.

For each file inputX.txt, you need to turn in a file named outputX.txt. If you successfully were able to allocate registers to all the variables, then outputX.txt should contain a line for every variable. m *r1* means variable m was assigned the register *r1*. Remember: there are only 8 registers. Let us call them r0,r1,r2,r3,r4,r5,r6, and r7. Please sort the output by the variable name in ascending order (lowest to highest).

If there are not enough registers to solve the problem, then outputX.txt should contain a single line with the minimum number of registers you would need.

Please turn in each of the 4 output files as described in the handin instructions on page 1.