CS533 — Midterm Exam — Wednesday, November 1, 2000

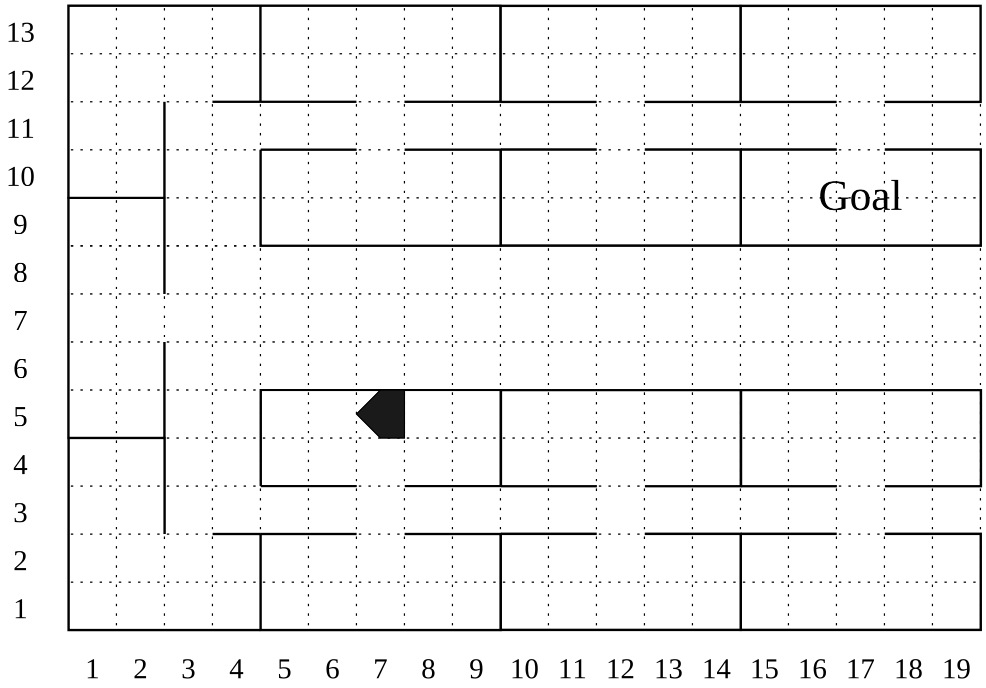
This exam is open book, open notes. You have 50 minutes to work the exam. The number of points for each problem (or part of a problem) is shown in square brackets [50] next to each problem. One point = one minute. **Please write your name on each page!**

1. [6] What member functions of the `rational` class are executed by the following code? (Write your answers where indicated by the answer: comments. Each answer: may be zero, one, or two member functions.)

In this answer, I have ignored calls to `rational::normalize`. Also note that the `rational` class does not have a destructor.

```
rational f(rational a, rational & b)  // answer: none
{
    return a - b;         // answer: rational::operator-
                                     rational::rational(rational&)
}
int main()
{
  rational c(1,4);        // answer: rational::rational(int,int)
  rational d(2,5);        // answer: rational::rational(int,int)
  d = f(c, d);            // answer: rational::rational(rational&)
                                     rational::operator=(rational&)
  return 0;               // answer: none
}
```

2. [12] Consider the following floor plan for a building.



Suppose we wish to program a mobile robot to navigate through buildings such as this one. The robot will *not* have a map of the building, so it must perform a problem-space search to find its destination. Here is the problem space:

- **State**. We will assume that space is divided into a grid of cells. The robot can be facing one of the four directions North, South, East, or West. A state gives the grid cell and orientation of the robot (e.g., the robot is currently at cell (7,5) pointed West).
- **Operators**. The robot has three operators:
  - **forward:** move one step forward.
  - **left:** turn left (rotate 90 degrees counter-clockwise) but stay in the same cell.
  - **right:** turn right (rotate 90 degrees clockwise) but stay in the same cell.
- **Starting State**. The starting state is robot in cell (7,5) pointing West.
- **Feasible Solution**. The goal is to get into the room marked "Goal". Any cell in that room (and any orientation) is a feasible solution.

(a) [2] What is the branching factor for this space?

3 (trivial)

(b) [4] What is the minimum solution depth to a feasible solution? Give a sequence of actions that will achieve that depth.

32.
L,F,F,R,F,F,F,R,F,F,F,F,F,F,F,F,F,R,F,F,F,F,F,F,F,F,F,F,F,F,F,F,F,R,F

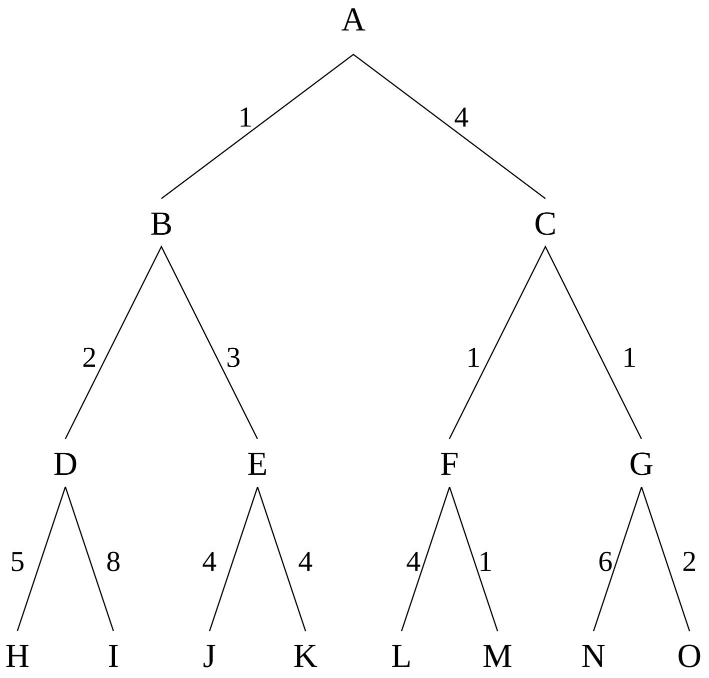
(c) [4] What is the total number of states in the state space?

There are 13*19 = 247 cells, and we can be in one of 4 orientations in each cell, so we get 13*19*4 = 988.

(d) [2] Is the search space redundant? If so, give an example.

Yes, extremely redundant. For example, the sequence L,L and R,R both get us to the same state (as does L,L,L,L,L,L and L,R,L,L, and R,R,R,L,R,L).

3. [17] Consider the following search tree.



The starting state is A, and all of the states at the bottom of the tree (H through O) are terminal states. Each branch in the tree is labeled with the "cost" of taking that branch. The goal is to find the terminal state that has the lowest cost path from A. For example, the cost of the path from A to O is $4 + 1 + 2 = 7$.

(a) [6] Suppose we run a greedy search algorithm that follows the lowest-cost branch at each state. What states will it visit, and what is the cost of the terminal state that it reaches? It will visit A, B, D, H. The cost will be $1 + 2 + 5 = 8$.

(b) [11] Suppose we are given the following h function that estimates the remaining cost-to-go for each state:

State	h
A	0
B	2
C	2
D	4
E	3
F	0
G	1

What states will be expanded during an A* search?

The following table shows the A* search. Each node s is followed by the value of $h(s)+g(s)$.

Frontier	Chosen Node
A(0+0)	A
B(2+1) C(2+4)	B
C(2+4) D(4+3) E(3+4)	C
F(0+5) G(1+5) D(4+3) E(3+4)	F
M(0+6) G(1+5) D(4+3) E(3+4) L(0+9)	M (solution)

4. [5] Briefly explain why the repair problem space for scheduling has a smaller branching factor than the constructive problem space.

The constructive problem space creates a branch for each possible start time of the next task in the topological order. The potential start times of each task are bounded by (a) the latest finish time of the prerequisites and (b) the latest possible start time based on the position of this task in a (latest) critical path schedule. That is usually a large number of possible start times.

In the repair space, there is only one branch for each task involved in a resource violation. Tasks not involved in any resource violations do not require any branching at all. In short, we are focused on only the tasks that participate in violations rather than on all the tasks. And for each such task, there is only one possible new start time rather than several. Once a task has been moved, it is unlikely to be moved again.

5. [4] List two advantages of propagating constraints during the design process.

Reduces the size of the search space by incorporating constraints into the generation of the remainder of the design.

Makes it *possible* to handle problems with continuous parameter choices, which cannot be solved via generate-and-test.

It increases the solution density by reducing or eliminating the possibility of generating infeasible states.

6. [6] Given each of the following constraints, indicate all of the different ways in which it could be propagated given that we know one or more of the variables involved.

(a)[3] $a = b \times c$

Given any two of the three variables, we can solve for the other.
Given any one of the variables, we can establish a relationship between the other two. For example, given that $a = 4$, we can infer that $b = \frac{4}{c}$. One tricky case: If $a = 0$, then we know that either $b = 0$ or $c = 0$ or both.

(b) [3] $a = \max(b, c)$

Given b and c, we can compute a.
Given b, we can derive the constraint $a \geq b$
Given c, we can derive the constraint $a \geq c$
Given a and b, if $a > b$, we can conclude $c = a$
if $a = b$, we can conclude $c \leq b$