CS121
Introduction to Artificial Intelligence
Winter 2004

Homework #1
(Search Problems)
Out: 1/14/03 — Due: 1/21/03

How to complete this HW: First copy this file; then insert your answers in the file immediately below each question; finally email the completed file to cs121-win0304-staff@lists.stanford.edu before 1/21 at midnight.
When you e-mail your homework, your subject line should be your leland username and then the homework you are submitting (e.g. "mykel: hw1"). If you must resubmit, make this clear in the subject line (e.g. "mykel: hw1 RESUBMIT").

Note on Honor Code: You must NOT look at previously published solutions of any of these problems in preparing your answers. You may discuss these problems with other students in the class (in fact, you are encouraged to do so) and/or look into other documents (books, web sites), with the exception of published solutions, without taking any written or electronic notes. If you have discussed any of the problems with other students, indicate their name(s) here:
……………………………………………………………………………………………
……
Any intentional transgression of these rules will be considered an honor code violation.

General information: In the following, whenever an explanation is required ("Why?"), there will not be full-credit without an explanation. Keep explanations short and to the point. Excessive verbosity will be penalized. If you have any doubt on how to interpret a question, either tell us in advance, so that we can help you understand the question, or tell us how you understand it in your returned solution.

Grading:

Problem#	Max. grade	Your grade
I	5	
II	5	
III	5	
IV	5	
V	5	
VI	5	

Total	10	

I. (5 points) Express the following problem as a search state problem:

"You have to color the various regions of a planar map (for example, the map of the 48 contiguous states of the United States) using only four colors (Red, Blue, Yellow, and Green), in such a way that no two adjacent regions have the same color."

Give the general description of a state, the initial state, the goal test, and the successor function. For the successor function it is not necessary that you write every possible "state → state" combination, but you should make it clear how one would derive the successor states of any given state. Note that we do not ask you to describe a solution of the problem!

```
Solution:
State Description: Map with each state assigned a color
or none
Initial State: No regions colored
Goal Test: All regions colored and no two adjacent
regions with the same color
Successor Function: Assign a color to an uncolored region
```

```
Note that there are other solutions, e.g.:

State description: Map with each state assigned a color
or none, and no two adjacent states colored with the same
color

Initial state: No regions colored

Goal Test: All regions colored

Successor Function: Assign a color to an uncolored
region; if this region is adjacent to an already colored
region, pick a different color.
```

II. (5 points) We ask you the same question for the following problem:

"Three jugs are filled with water. Neither has any measuring marker on it. Each can be fully or partially emptied in a drain or another jug. The jugs respectively measure 15, 7, and 3 gallons. One needs to measure out exactly 2 gallons."

Note that we do not ask you to describe a solution of the problem!

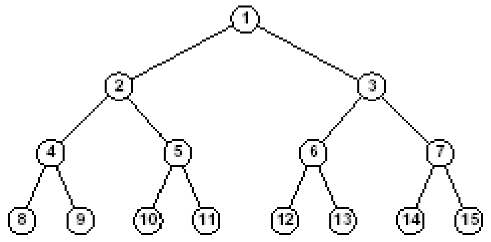
```
Solution:
State Description: [x, y, z]
Initial State: [15, 7, 3]
```

```
Goal Test: [x, y, z], such that x=2, or y=2, or z=2
Successor Function: Given [x, y, z], generate:
[0, y, z], [x, 0, z], [x, y, 0] (empty into drain)
[x-min(x+y,7)+y, min(x+y,7), z] (empty x into y)
[x, y-min(y+z,3)+z, min(y+z,3)] (empty y into z)
[min(x+z,15), y, z-min(x+z,15)+x] (empty z into x)
[x-min(x+z,3)+z, y, min(x+z,3)] (empty x into z)
[min(x+y,15), y-min(x+y,15)+x, z] (empty y into x)
[x, min(y+z,7), z-min(y+z,7)+y] (empty z into y)
```

III. (5 points) Consider a state space where states are describes by integers. The initial state is 1 and the successor function for state *n* returns two states 2*n* and 2*n*+1.

Draw the portion of the state space that contains the initial state and all states that can be reached from the initial state down to depth 3.
Let the goal state be 11. List the order in which nodes are visited by breadth-first search, depth-first search limited to depth 3, and iterative deepening search.
Would backward search be appropriate for this problem? Describe how it would work

Solution:



BFS: 1 2 3 4 5 6 7 8 9 10 11

[Note: There are other solutions because the successors of a node can be taken in any order, e.g., 1 3 2 7 6 4 5 14 15 12 13 10 11 is another BFS solution]

Depth Limited: 1 2 4 8 9 5 10 11

[Note: For the same reason as above, there are other possible solutions]

Iterative Deepening: 1; 1 2 3; 1 2 4 5 3 6 7; 1 2 4 8 9 5 10 11

[Same remark again]

Backwards searching would be helpful because the branching factor going up form the goal is 1.

Start at the goal node (11). The successor of node N is floor(N/2).

IV. (5 points) Does a finite state space always lead to a finite search tree? If not, give an example where a finite state space leads to an infinite search tree. How could you avoid an infinite tree?

Solution:
A finite search space can lead to an infinite search tree if states are repeated. For example, in the 8 puzzle, you can always return to the state of a node's parent by moving the empty tile back. By doing this you can generate an infinitely long chain of search nodes. This can be prevented by avoiding repeated states.

V. (5 points) Consider the *n*-queens problem using the following formulation:
A state is an arrangement of k ($0 \le k \le n$) queens, one per column, in the leftmost k columns, with no two queens attacking one another.
The successor function adds a queen to a square in the leftmost empty column such that it is not attacked by any other queen.
The goal state is one with n queens.

Show that $(n!)^{1/3}$ is a lower bound on the size of the state space.

> [Hint: Consider the maximum number of squares in each column that can be attacked by queens from previous columns.]
> Consider the case where $n = 8$. The actual number of states is 2,057. How does this compare to the above lower bound? What are the causes for the difference?

Solution:

The number of state with 0 queens is 1.

The number of states with 1 queen is n

The number of states with 2 queens is lower bounded by n(n−3), since the queen in the 1st column eliminates at most three cells in column 2.

The number of states with 3 queens is lower bounded by n(n−3)(n−6), since each of the 2 queens in columns 1 and 2 eliminates at most 3 cells in column 3.

Etc…

Let k be the largest integer such that (n−3k) is equal to 1, 2, or 3. Continuing the above reasoning, the number of states with more than (k+1) queens is trivially lower bounded by 0.

So, in total, the number of states is lower bounded by:

$1 + n + n(n-3) + \ldots + n(n-3)(n-6)\ldots(n-3k)$,

which itself is lower bounded by:

$n(n-3)(n-6)\ldots(n-3k)$.

We can write:

$$\begin{aligned} n! &= n(n-1)(n-2)\ldots1 \\ &= n(n-3)(n-6)\ldots \times (n-1)(n-4)(n-7)\ldots \times (n-2)(n-5)\ldots \\ &= A \times B \times C \end{aligned}$$

Since $A > B > C$ (every factor of A is greater than a factor of B, which in turn is greater than a factor of C, and A has at least as many factors as B, which has as at least as many factors as C), $n(n-3)(n-6)\ldots$ is greater than $(n!)^{1/3}$.

Alternative explanation:

$$\begin{aligned} (n(n-3)(n-6)\ldots)^3 &= n*(n\ \ )*(n\ \ )*(n-3)*(n-3)*(n-3)\ldots \\ n! &= n*(n-1)*(n-2)*(n-3)*(n-4)*(n-5) \end{aligned}$$

The factors in the first line are greater or equal to the factors in the second line. Therefore, $(n(n-3)(n-6)\ldots)^3 > n!$ and $(n(n-3)(n-6)\ldots) > (n!)^{1/3}$.

Hence, $(n!)^{1/3}$ is a lower bound on the total number of states.

$(8!)^{1/3}=34.29$. This less than 2057, which is expected since it is a lower bound. The reason the actual number of states is greater because it counts eliminated spaces multiple times (where ever a diagonal and row intersect for example).

VI. (5 points) Construct a search tree of uniform branching factor *b* and uniform depth *d* for which it is *possible* that depth-first search uses more memory than breadth-first search. (The depth of a node in a search tree is the length of the path from the root of the tree to N. In particular, the depth of the root is 0.)

Is there any tree and distribution of goal nodes for which depth-first search *always* requires more memory than breadth-first? Explain briefly. To answer this question, recall that depth-first search assumes no intrinsic ordering of the successors of a node; it may pick any ordering.

Solution:
In the worst case, depth-first search requires memory proportional to the depth of the tree, while breadth-first search requires $O(b^g)$, where g is the depth of the shallowest goal node.
In a deep tree (large d) with a sufficiently small branching factor (small b) and a shallow goal node (small g), depth-first search requires more memory than breadth-first search.

There is no case where depth-first search always requires more memory than breadth-first search, since depth-first search can always go straight to the shallowest goal node.