15-381 - Fall 2000
MIDTERM EXAM - Sample Solutions
Prepared by L. Hiyakumoto and R. Jensen

Problem 1: Uninformed Search [*10 points*]

Assume you have an infinite state space with one state identified as the initial state and one as the goal state. All operators are reversible with forwards and backwards branching factor of 4, and the depth to the unique solution is 16. Express each answer as an exact calculation (leaving exponents, summations, etc without expansion if you wish) for full credit. Use big-O or an approximate calculation for partial credit.

A. How many states would be visited in a bi-directional breadth-first search in the worst case?

$$2\sum_{i=0}^{8} 4^i$$

B. How many states in a forward-only breadth-first search in the worst case, with an island at d=4 and an island at d=8 (but no island at d=12)?

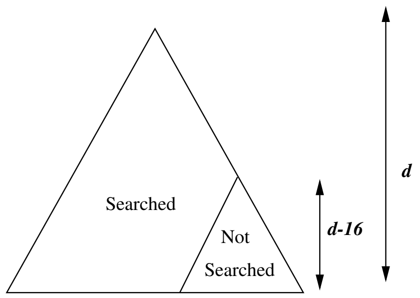
$$2\sum_{i=0}^{4} 4^i + \sum_{i=0}^{8} 4^i$$

C. How many states would be visited in the worst-case for depth-first search without depth bounds?

$$\infty$$

D. How many states in the worst case for depth-first search with depth bound $d > 16$, if the branching factor is 2 and the depth of the unique solution is still 16?

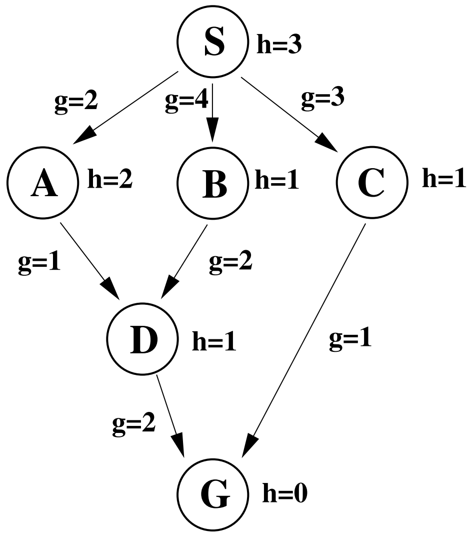
$$\sum_{i=0}^{d} 2^i - \sum_{i=1}^{d-16} 2^i$$



Problem 2: Heuristic Search [*15 points*]

A. Consider the search problem below with start state S and goal state G. We know that the given heuristic is admissible. Furhtermore, we know the content of the priority queue of A^* after each node expansion. Use this knowledge to fill out the unknown values for g and h.

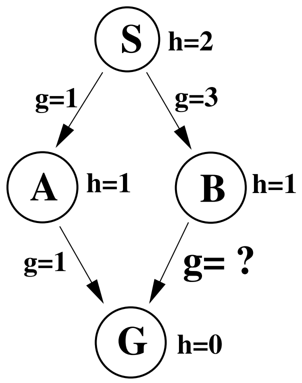
1. { (S, f=3) }
2. { (A, f=4), (C, f=4), (B, f=5) }
3. { (D, f=4), (C, f=4), (B, f=5) }
4. { (C, f=4), (G, f=5), (B, f=5) }
5. { (G, f=4), (B, f=5) }



B. Chris claims that: In general, if we are given an arbitrary search problem, for which we know: (i) the heuristic values, h, of *all* the nodes; (ii) that the heuristic is admissible; and (iii) all the values of the priority queue of A^*'s search episode, then the costs (g) of *all* the edges in this arbitrary search problem can be uniquely determined. Is Chris correct? If yes, then prove it informally. Otherwise, give an example of a state graph (similar to part A) that shows that Chris is incorrect.

Chris is not correct. In a search episode we are not guaranteed to visit all edges that are necessary in order to compute all the g-values. A counter-example is:

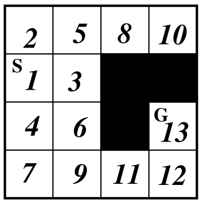
1. { (S, f=2) }
2. { (A, f=2), (B, f=4) }
3. { (G, f=2), (B, f=4) }



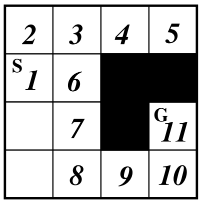
Problem 3: More Search [*15 points*]

Given the maze with start and goal states as shown, write in the order in which the states are visited for each of the search techniques listed below. Mark the start state with a '1', the next state visited with a '2' etc.. Assume that loops are detected and you consider moves in the order Up, Right, Down, Left.

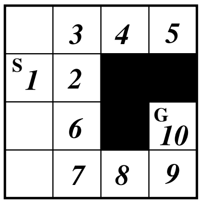
A. Breath-first search.



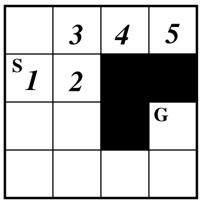
B. Depth-first search.



C. Best-first search - using horizontal distance between the current and goal positions, e.g., the horizontal distance between S and G is 3.



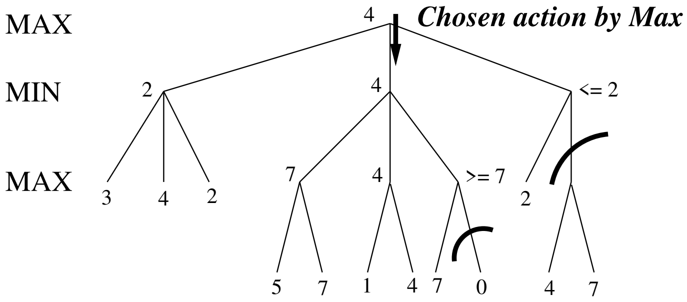
D. Hill-climbing - using horizontal distance between the current and goal positions, e.g., the horizontal distance between S and G is 3.



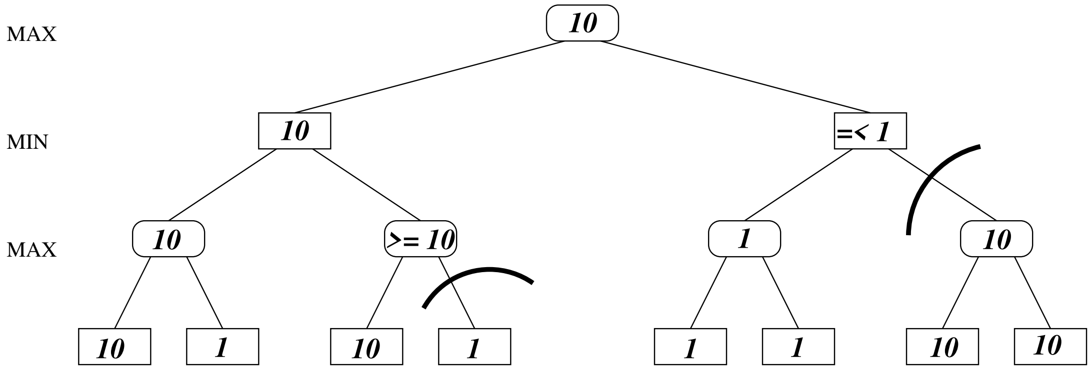
Stuck !!

Problem 4: Games [*10 points*]

A. Show the move chosen and which branches (if any) would be cut off during an alpha-beta search for the following game tree (nodes are searched from left-to-right).



B. Fill in terminal values for the following game tree that will result in alpha-beta doing the maximum number of cut-offs possible. Assume that nodes are always considered left-to-right.



The two cuts shown removes the largest number of nodes from the game tree
The terminal value configuration is one of many configurations causing these cuts

Problem 5: Logic [*15 points*]

A. Using three-valued logic (T,F,?), state whether the following clause is valid, satisfiable, or unsatisfiable.

$((A \Rightarrow B) \wedge A) \Leftrightarrow B$

The clause is satisfiable.

A	B	$A \Rightarrow B$	$(A \Rightarrow B) \wedge A$	$((A \Rightarrow B) \wedge A) \Leftrightarrow B$
T	T	T	T	T
T	F	F	F	T
T	?	?	?	T
F	T	T	F	F
F	F	T	F	T
F	?	T	F	?
?	T	T	?	?
?	F	?	F	T
?	?	?	?	?

(Note: you could also simplify the given clause to $(B \wedge A) \Leftrightarrow B$ first.)

B. Use resolution to prove $W \Rightarrow \neg T$ given the set of facts:

1. $(P \wedge Q) \Rightarrow R$
2. $T \Rightarrow Q$
3. $W \Rightarrow P$
4. $\neg R$

Direct proof

1. $\neg P \vee \neg Q \vee R$
2. $\neg T \vee Q$
3. $\neg W \vee P$
4. $\neg R$
5. $\neg P \vee \neg Q$ (res. 1,4)
6. $\neg P \vee \neg T$ (res. 2,5)
7. $\neg W \vee \neg T$ (res. 3,6)

Proof by contradiction
First notice:

$$\neg(W \Rightarrow \neg T) = \neg(\neg W \vee \neg T) = W \wedge T$$

1. $\neg P \vee \neg Q \vee R$
2. $\neg T \vee Q$
3. $\neg W \vee P$
4. $\neg R$
5. W
6. T
7. $\neg P \vee \neg Q$ (res. 1,4)
8. Q (res. 2,6)
9. $\neg P$ (res. 7,8)
10. P (res. 5,3)
11. false (res. 9,10)

Problem 6: Constraint Satisfaction [*10 points*]

Consider the following expression:

$((a \wedge b) \vee \neg a) \wedge (((a \vee \neg b \vee c) \wedge \neg c) \vee d)$

A. Re-write this expression in conjunctive normal form.

$(a \vee \neg a) \wedge (b \vee \neg a) \wedge (a \vee \neg b \vee c \vee d) \wedge (\neg c \vee d)$

B. Use the DPLL algorithm to determine if the expression is satisfiable. If so, what is the assignment that DPLL produces? Assume that, in choosing between unassigned literals, DPLL selects them in alphabetical order, and tries the value True before False.

DPLL : $a \leftarrow \mathsf{True}$
$(\mathsf{True} \vee \mathsf{False}) \wedge (\mathsf{b} \vee \mathsf{False}) \wedge (\mathsf{True} \vee \neg\mathsf{b} \vee \mathsf{c} \vee \mathsf{d}) \wedge (\neg\mathsf{c} \vee \mathsf{d})$

Propagate : $b \leftarrow \mathsf{True}$
$(\mathsf{True} \vee \mathsf{False}) \wedge (\mathsf{True} \vee \mathsf{False}) \wedge (\mathsf{True} \vee \mathsf{False} \vee \mathsf{c} \vee \mathsf{d}) \wedge (\neg\mathsf{c} \vee \mathsf{d})$

DPLL : $c \leftarrow \mathsf{True}$
$(\mathsf{True} \vee \mathsf{False}) \wedge (\mathsf{True} \vee \mathsf{False}) \wedge (\mathsf{True} \vee \mathsf{False} \vee \mathsf{True} \vee \mathsf{d}) \wedge (\mathsf{False} \vee \mathsf{d})$

Propagate: $d \leftarrow \mathsf{True}$
$(\mathsf{True} \vee \mathsf{False}) \wedge (\mathsf{True} \vee \mathsf{False}) \wedge (\mathsf{True} \vee \mathsf{False} \vee \mathsf{True} \vee \mathsf{True}) \wedge (\mathsf{False} \vee \mathsf{True})$

Thus, the expression is satsifiable with the assignment
$a = \mathsf{True}$, $b = \mathsf{True}$, $c = \mathsf{True}$, $d = \mathsf{True}$.

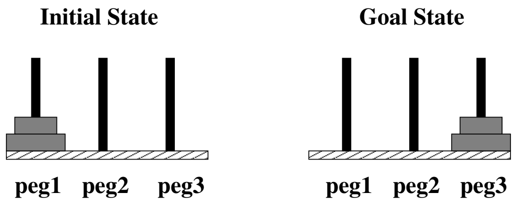
Problem 7: Planning [*15 points*]

Consider the Tower of Hanoi task, with 2 disks and any number of pegs. One disk is LARGE and another disk is SMALL. The two operators below capture the actions legal in this task. As usual, variables are represented with a "?" prefix, e.g., ?peg-x.

```
operator MOVE-SMALL ?peg-x ?peg-y
 preconds: (on SMALL ?peg-x)


 add: (on SMALL ?peg-y)
 del: (on SMALL ?peg-x)
```

```
operator MOVE-LARGE ?peg-x ?peg-y
 preconds: (on LARGE ?peg-x)
           (not (on SMALL ?peg-x))
           (not (on SMALL ?peg-y))
 add: (on LARGE ?peg-y)
 del: (on LARGE ?peg-x)
```

A. What is an optimal plan to solve the 3-peg problem shown in the figure below?




```
(MOVE-SMALL peg1 peg2)
(MOVE-LARGE peg1 peg3)
(MOVE-SMALL peg2 peg3)
```

B. Consider two different goal specifications for the problem above:

```
g1 = (and (on SMALL peg3) (on LARGE peg3))
g2 = (and (on LARGE peg3) (on SMALL peg3)).
```

1. Would the GPS planner (means-ends analysis, linear planning with immediate state changes) find the same plan for the two goal specifications? Why or why not?

   Suppose we choose differences from left to right, choose operators in the order `MOVE-SMALL` and `MOVE-LARGE`, choose operator bindings for pegs in the order `peg1`, `peg2`, `peg3`, and detect repeated states. With these assumptions we can illustrate how the goal stack that GPS uses to solve problems can result in different plans for the two goal specifications (tracing the algorithm provided in lecture notes):

GPS(`((on SMALL peg1) (on LARGE peg1))`,
`((on SMALL peg3) (on LARGE peg3))`):
$d \leftarrow$ **(on SMALL peg3)**
$o \leftarrow$ `(MOVE-SMALL peg1 peg3)`
$s \leftarrow$ GPS(`((on SMALL peg1) (on LARGE peg1))`,
`((on SMALL peg1))`)
$s' \leftarrow$ Apply(`(MOVE-SMALL peg1 peg3)`,
`((on SMALL peg1) (on LARGE peg1))`)

GPS(`((on SMALL peg3) (on LARGE peg1))`,
`((on SMALL peg3) (on LARGE peg3))`):
$d \leftarrow$ `(on LARGE peg3)`
$o \leftarrow$ `(MOVE-LARGE peg1 peg3)`
$s \leftarrow$ GPS(`((on SMALL peg3) (on LARGE peg1))`,
`((not (on SMALL peg1)) (not (on SMALL peg3)))`)
$d \leftarrow$ `(not (on SMALL peg3))`
$o \leftarrow$ `(MOVE-SMALL peg3 peg2)`
$s \leftarrow$ GPS(`((on SMALL peg3) (on LARGE peg1))`,
`((on SMALL peg3))`)
$s' \leftarrow$ Apply(`(MOVE-SMALL peg3 peg2)`,
`((on SMALL peg3) (on LARGE peg1))`)
$s' \leftarrow$ Apply(`(MOVE-LARGE peg1 peg3)`,
`((on SMALL peg2) (on LARGE peg1))`)

GPS(`((on SMALL peg2) (on LARGE peg3))`,
`((on SMALL peg3) (on LARGE peg3))`)
$d \leftarrow$ `(on SMALL peg3)`
$o \leftarrow$ `(MOVE-SMALL peg2 peg3)`
$s \leftarrow$ GPS(`((on SMALL peg2) (on LARGE peg3))`,
`((on SMALL peg2))`)
$s' \leftarrow$ Apply(`(MOVE-SMALL peg2 peg3)`,
`((on SMALL peg2) (on LARGE peg3))`)

Final plan for `g1`:

```
(MOVE-SMALL peg1 peg3)
(MOVE-SMALL peg3 peg2)
(MOVE-LARGE peg1 peg3)
(MOVE-SMALL peg2 peg3)
```

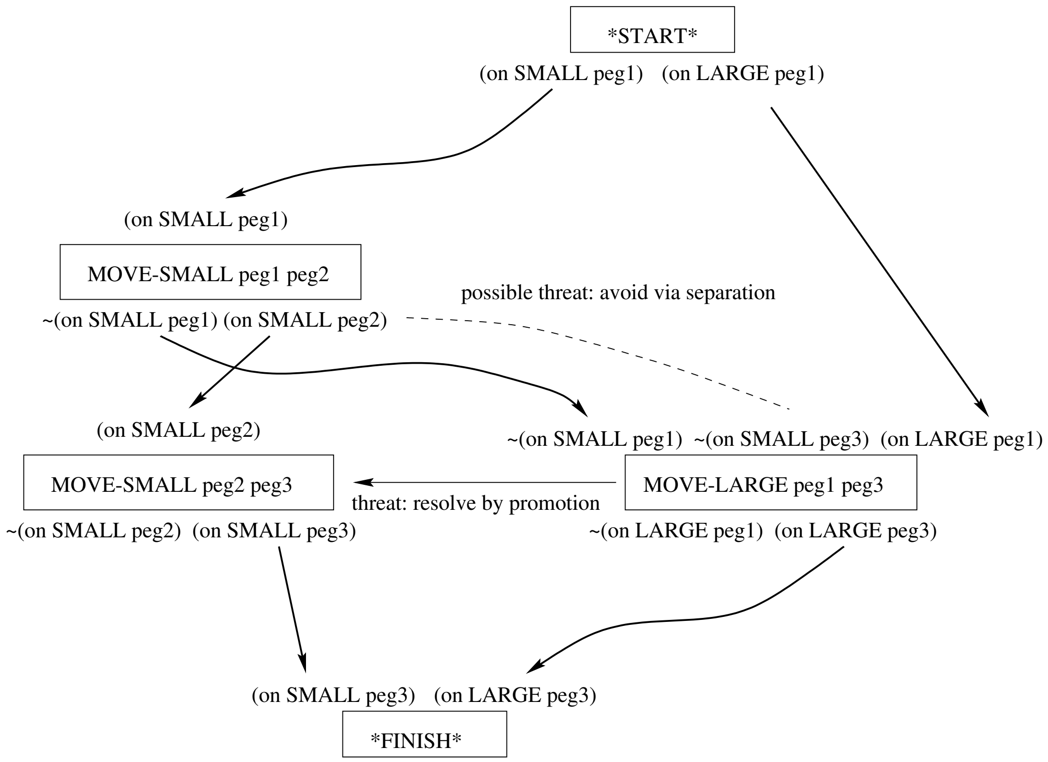
```
GPS(((on SMALL peg1) (on LARGE peg1)),
    ((on LARGE peg3) (on SMALL peg3))):
    d ← (on LARGE peg3)
    o ← (MOVE-LARGE peg1 peg3)
    s ← GPS(((on SMALL peg1) (on LARGE peg1)),
            ((not (on SMALL peg1)) (not (on SMALL peg3))))
        d ← (not (on SMALL peg1))
        o ← (MOVE-SMALL peg1 peg2)
        s ← GPS(((on SMALL peg1) (on LARGE peg1)),
                ((on SMALL peg1)))
        s' ← Apply((MOVE-SMALL peg1 peg2),
                   ((on SMALL peg1) (on LARGE peg1)))
    s' ← Apply((MOVE-LARGE peg1 peg3),
               ((on SMALL peg2) (on LARGE peg1)))

GPS(((on SMALL peg2) (on LARGE peg3)),
    ((on LARGE peg3) (on SMALL peg3)))
    d ← (on SMALL peg3)
    o ← (MOVE-SMALL peg2 peg3)
    s ← GPS(((on SMALL peg2) (on LARGE peg3)),
            ((on SMALL peg2)))
    s' ← Apply((MOVE-SMALL peg2 peg3),
               ((on SMALL peg2) (on LARGE peg3)))
```

Final plan for g2:

```
(MOVE-SMALL peg1 peg2)
(MOVE-LARGE peg1 peg3)
(MOVE-SMALL peg2 peg3)
```

Note that the plan in the second case is not guaranteed to be optimal: it depends on the operator and binding ordering (i.e. if we had considered peg3 before peg2 for g2, we could have found the same plan as for the first specification.)

2. What would be the plan found by a partial order planner?


START
(on SMALL peg1) (on LARGE peg1)
(on SMALL peg1)
MOVE-SMALL peg1 peg2
possible threat: avoid via separation
~(on SMALL peg1) (on SMALL peg2)
(on SMALL peg2)
~(on SMALL peg1) ~(on SMALL peg3) (on LARGE peg1)
MOVE-SMALL peg2 peg3
threat: resolve by promotion
MOVE-LARGE peg1 peg3
~(on SMALL peg2) (on SMALL peg3)
~(on LARGE peg1) (on LARGE peg3)
(on SMALL peg3) (on LARGE peg3)
FINISH

Problem 8: NLP [*10 points*]

Consider the context-free grammar:

S $\rightarrow$ NP VP
S $\rightarrow$ NP VP CONJ S
VP $\rightarrow$ V NP
VP $\rightarrow$ V NP PP
NP $\rightarrow$ N
NP $\rightarrow$ DET N
NP $\rightarrow$ DET ADJL N
ADJL $\rightarrow$ ADJ
ADJL $\rightarrow$ ADJ ADJL
PP $\rightarrow$ PRP NP

Convert this small context-free grammar into a set of simple ATNs, one ATN per true non-terminal. You should forget about the augmentation table for this problem. (This is equivalent to assuming all augmentation tests are true, and no registers are ever set, which also means the ATN is being used only as a recognizer.) You may use the notation from class (same as in the handout reading), or the slight variant from the homework problem. Note that for the purpose of this exercise you do not need to know what the symbols represent, but if you are curious: S = sentence, VP = verb phrase, NP = noun phrase, ADJL = adjective list, PP = prepositional phrase, N = noun, V = verb, CONJ = conjunction (e.g. ”and”, ”or”, ”but”), PRP = preposition, DET = determiner, and ADJ = adjective.