Q1. [10 pts] All Searches Lead to the Same Destination

For all the questions below assume :

- All search algorithms are *graph* search (as opposed to tree search).
- $c_{ij} > 0$ is the cost to go from node i to node j.
- There is only one goal node (as opposed to a set of goal nodes).
- All ties are broken alphabetically.
- Assume heuristics are consistent.

Definition: Two search algorithms are defined to be ***equivalent*** if and only if they expand the same nodes in the same order and return the same path.

In this question we study what happens if we run uniform cost search with action costs d_{ij} that are potentially different from the search problem's actual action costs c_{ij}. Concretely, we will study how this might, or might not, result in running uniform cost search (with these new choices of action costs) being equivalent to another search algorithm.

(a) [2 pts] Mark *all* choices for costs d_{ij} that make running **Uniform Cost Search** algorithm with these costs d_{ij} *equivalent* to running **Breadth-First Search**.

- ○ $d_{ij} = 0$
- ● $d_{ij} = \alpha$, $\alpha > 0$
- ○ $d_{ij} = \alpha$, $\alpha < 0$
- ● $d_{ij} = 1$
- ○ $d_{ij} = -1$
- ○ None of the above

Beadth First search expands the node at the shallowest depth first. Assigning a constant positive weight to all edges allows to weigh the nodes by their depth in the search tree.

(b) [2 pts] Mark *all* choices for costs d_{ij} that make running **Uniform Cost Search** algorithm with these costs d_{ij} *equivalent* to running **Depth-First Search**.

- ○ $d_{ij} = 0$
- ○ $d_{ij} = \alpha$, $\alpha > 0$
- ● $d_{ij} = \alpha$, $\alpha < 0$
- ○ $d_{ij} = 1$
- ● $d_{ij} = -1$
- ○ None of the above

Depth First search expands the nodes which were most recently added to the fringe first. Asssigning a constant negative weight to all edges essentially allows to reduce the value of the most recently nodes by that constant, making them the nodes with the minimum value in the fringe when using uniform cost search.

(c) [2 pts] Mark *all* choices for costs d_{ij} that make running **Uniform Cost Search** algorithm with these costs d_{ij} *equivalent* to running **Uniform Cost Search** with the original costs c_{ij}.

- ○ $d_{ij} = c_{ij}^2$
- ○ $d_{ij} = 1/c_{ij}$
- ● $d_{ij} = \alpha\ c_{ij}, \qquad \alpha > 0$
- ○ $d_{ij} = c_{ij} + \alpha, \qquad \alpha > 0$
- ○ $d_{ij} = \alpha\ c_{ij} + \beta, \quad \alpha > 0,\ \beta > 0$
- ○ None of the above

Uniform cost search expands the node with the lowest cost-so-far $= \sum_{ij} c_{ij}$ on the fringe. Hence, the relative ordering between two nodes is determined by the value of $\sum_{ij} c_{ij}$ for a given node. Amongst the above given choices, only for $d_{ij} = \alpha\ c_{ij},\ \alpha > 0$, can we conclude,
$\sum_{ij \in \text{ path}(n)} d_{ij} \geq \sum_{ij \in \text{ path}(m)} d_{ij} \iff \sum_{ij \in \text{ path}(n)} c_{ij} \geq \sum_{ij \in \text{ path}(m)} c_{ij}$, for some nodes n and m.

(d) Let $h(n)$ be the value of the heuristic function at node n.

(i) [2 pts] Mark *all* choices for costs d_{ij} that make running **Uniform Cost Search** algorithm with these costs d_{ij} *equivalent* to running **Greedy Search** with the original costs c_{ij} and heuristic function h.

- ○ $d_{ij} = h(i) - h(j)$
- ● $d_{ij} = h(j) - h(i)$
- ○ $d_{ij} = \alpha\ h(i), \qquad \alpha > 0$
- ○ $d_{ij} = \alpha\ h(j), \qquad \alpha > 0$
- ○ $d_{ij} = c_{ij} + h(j) + h(i)$
- ○ None of the above

Greedy search expands the node with the lowest heuristic function value $h(n)$. If $d_{ij} = h(j) - h(i)$, then the cost of a node n on the fringe when running uniform-cost search will be $\sum_{ij} d_{ij} = h(n) - h(start)$. As $h(start)$ is a common constant subtracted from the cost of all nodes on the fringe, the relative ordering of the nodes on the fringe is still determined by $h(n)$, i.e. their heuristic values.

(ii) [2 pts] Mark *all* choices for costs d_{ij} that make running **Uniform Cost Search** algorithm with these costs d_{ij} *equivalent* to running **A* Search** with the original costs c_{ij} and heuristic function h.

- ○ $d_{ij} = \alpha\ h(i), \qquad \alpha > 0$
- ○ $d_{ij} = \alpha\ h(j), \qquad \alpha > 0$
- ○ $d_{ij} = c_{ij} + h(i)$
- ○ $d_{ij} = c_{ij} + h(j)$
- ○ $d_{ij} = c_{ij} + h(i) - h(j)$
- ● $d_{ij} = c_{ij} + h(j) - h(i)$
- ○ None of the above

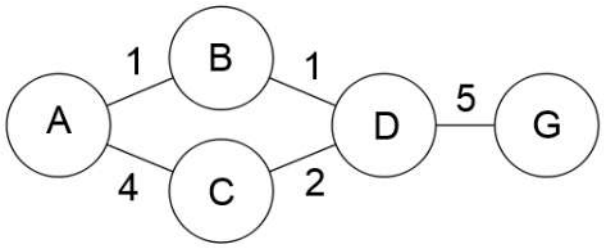
A^* search expands the node with the lowest $f(n) + h(n)$ value, where $f(n) = \sum_{ij} c_{ij}$ is the cost-so-far and h is the heuristic value. If $d_{ij} = c_{ij} + h(j) - h(i)$, then the cost of a node n on the fringe when running uniform-cost search will be $\sum_{ij} d_{ij} = \sum_{ij} c_{ij} + h(n) - h(start) = f(n) + h(n) - h(start)$. As $h(start)$ is a common constant subtracted from the cost of all nodes on the fringe, the relative ordering of the nodes on the fringe is still determined by $f(n) + h(n)$.

Q2. [18 pts] Dynamic A* Search

After running A* graph search and finding an optimal path from start to goal, the cost of one of the edges, $X \to Y$, in the graph changes. Rather than re-running the entire search, you want to find a more efficient way of finding the optimal path for this new search problem.

You have access to the fringe, the closed set and the search tree as they were at the completion of the initial search. In addition, you have a *closed node map* that maps a state, s from the closed set to a list of nodes in the search tree ending in s which were not expanded because s was already in the closed set.

For example, after running A* search with the null heuristic on the following graph, the data structures would be as follows:



Fringe: {}

Closed Set: {A, B, C, D}

Closed Node Map: {A:[ ], B:[ ], C:[ ], D:[(A-C-D, 6)]}

Search Tree:
{A : [(A-B, 1), (A-C, 4)],
A-B : [(A-B-D, 2)],
A-C : [ ],
A-B-D : [(A-B-D-G, 7)],
A-B-D-E : [ ]}

For a general graph, for each of the following scenarios, select the choice that finds the correct optimal path and cost *while expanding the fewest nodes*. Note that if you select the 4^{th} choice, you must fill in the change, and if you select the last choice, you must describe the set of nodes to add to the fringe.

In the answer choices below, if an option states some nodes will be added to the fringe, this also implies that the final state of each node gets cleared out of the closed set (indeed, otherwise it'd be rather useless to add something back into the fringe). You may assume that there are no ties in terms of path costs.

Following is a set of eight choices you should use to answer the questions on the following page.

i. The optimal path does not change, and the cost remains the same.
ii. The optimal path does not change, but the cost increases by n
iii. The optimal path does not change, but the cost decreases by n
iv. The optimal path does not change, but the cost changes by _____
v. The optimal path for the new search problem can be found by adding the subtree rooted at X that was expanded in the original search back onto the fringe and re-starting the search.
vi. The optimal path for the new search problem can be found by adding the subtree rooted at Y that was expanded in the original search back onto the fringe and re-starting the search.
vii. The optimal path for the new search problem can be found by adding all nodes for each state in the *closed node map* back onto the fringe and re-starting the search.
viii. The optimal path for the new search problem can be found by adding some other set of nodes back onto the fringe and re-starting the search. Describe the set below.

(a) [3 pts] Cost of $X \to Y$ is increased by $n, n > 0$, the edge is on the optimal path, and was explored by the first search.

○ i ○ ii ○ iii ○ iv, Change:
○ v ○ vi ○ vii ● viii, Describe the set below:

The combination of all the nodes from the *closed node map* for the final state of each node in the subtree rooted at Y plus the node ending at Y that was expanded in the initial search. This means that you are re-exploring every path that was originally closed off by a path that included the edge $X \to Y$.

(b) [3 pts] Cost of $X \to Y$ is decreased by $n, n > 0$, the edge is on the optimal path, and was explored by the first search.

○ i ○ ii ● iii ○ iv, Change:
○ v ○ vi ○ vii ○ viii, Describe the set below:

The original optimal path's cost decreases by n because $X \to Y$ is on the original optimal path. The cost of any other path in the graph will decrease by at most n (either n or 0 depending on whether or not it includes $X \to Y$). Because the optimal path was already cheaper than any other path, and decreased by at least as much as any other path, it must still be cheaper than any other path.

(c) [3 pts] Cost of $X \to Y$ is increased by $n, n > 0$, the edge is not on the optimal path, and was explored by the first search.

● i ○ ii ○ iii ○ iv, Change:
○ v ○ vi ○ vii ○ viii, Describe the set below:

The cost of the original optimal path, which is lower than the cost of any other path, stays the same, while the cost of any other path either stays the same or increases. Thus, the original optimal path is still optimal.

(d) [3 pts] Cost of $X \to Y$ is decreased by $n, n > 0$, the edge is not on the optimal path, and was explored by the first search.

○ i ○ ii ○ iii ○ iv, Change:
○ v ○ vi ○ vii ○ viii, Describe the set below:

The combination of the previous goal node and the node ending at X that was expanded in the initial search.

There are two possible paths in this case. The first is the original optimal path, which is considered by adding the previous goal node back onto the fringe. The other option is the cheapest path that includes $X \to Y$, because that is the only cost that has changed. There is no guarantee that the node ending at Y, and thus the subtree rooted at Y contains $X \to Y$, so the subtree rooted at X must be added in order to find the cheapest path through $X \to Y$.

(e) [3 pts] Cost of $X \to Y$ is increased by $n, n > 0$, the edge is not on the optimal path, and was not explored by the first search.

● i ○ ii ○ iii ○ iv, Change:
○ v ○ vi ○ vii ○ viii, Describe the set below:

This is the same as part (c).

(f) [3 pts] Cost of $X \to Y$ is decreased by $n, n > 0$, the edge is not on the optimal path, and was not explored by the first search.

● i ○ ii ○ iii ○ iv, Change:
○ v ○ vi ○ vii ○ viii, Describe the set below:

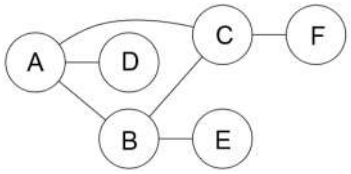
Assuming that the cost of $X \to Y$ remains positive, because the edge was never explored, the cost of the path to X is already higher than the cost of the optimal path. Thus, the cost of the path to Y through X can only be higher, so the optimal path remains the same.

If you allow edge weights to be negative, it is necessary to find the optimal path to Y through X separately. Because the edge was not explored, a node ending at X was never expanded, so the negative edge would still never be seen unless the path was found separately and added onto the fringe. In this case, adding this path and the original goal path, similar to (d), would find the optimal path with the updated edge cost.

Q3. [16 pts] CSPs

(a) The graph below is a constraint graph for a CSP that has only binary constraints. Initially, no variables have been assigned.

For each of the following scenarios, mark all variables for which the specified filtering might result in their domain being changed.



(i) [1 pt] A value is assigned to A. Which domains might be changed as a result of running forward checking for A?

○ A ● B ● C ● D ○ E ○ F

Forward checking for A only considers arcs where A is the head. This includes $B \to A$, $C \to A$, $D \to A$. Enforcing these arcs can change the domains of the tails.

(ii) [1 pt] A value is assigned to A, and then forward checking is run for A. Then a value is assigned to B. Which domains might be changed as a result of running forward checking for B?

○ A ○ B ● C ○ D ● E ○ F

Similar to the previous part, forward checking for B enforces the arcs $A \to B$, $C \to B$, and $E \to B$. However, because A has been assigned, and a value is assigned to B, which is consistent with A or else no value would have been assigned, the domain of A will not change.

(iii) [1 pt] A value is assigned to A. Which domains might be changed as a result of enforcing arc consistency after this assignment?

○ A ● B ● C ● D ● E ● F

Enforcing arc consistency can affect any unassigned variable in the graph that has a path to the assigned variable. This is because a change to the domain of X results in enforcing all arcs where X is the head, so changes propagate through the graph. Note that the only time in which the domain for A changes is if any domain becomes empty, in which case the arc consistency algorithm usually returns immediately and backtracking is required, so it does not really make sense to consider new domains in this case.

(iv) [1 pt] A value is assigned to A, and then arc consistency is enforced. Then a value is assigned to B. Which domains might be changed as a result of enforcing arc consistency after the assignment to B?

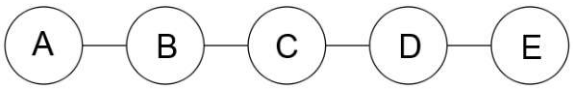
○ A ○ B ● C ○ D ● E ● F

After assigning a value to A, and enforcing arc consistency, future assignments and enforcing arc consistency will not result in a change to As domain. This means that Ds domain won't change because the only arc that might cause a change, $D \to A$ will never be enforced.

(b) You decide to try a new approach to using arc consistency in which you initially enforce arc consistency, and then enforce arc consistency every time you have assigned an even number of variables.

You have to backtrack if, after a value has been assigned to a variable, X, the recursion returns at X without a solution. Concretely, this means that for a single variable with d values remaining, it is possible to backtrack up to d times. For each of the following constraint graphs, if each variable has a domain of size d, how many times would you have to backtrack in the worst case for each of the specified orderings?

(i) [6 pts]



A-B-C-D-E: 0

A-E-B-D-C: $2d$

C-B-D-E-A: 0

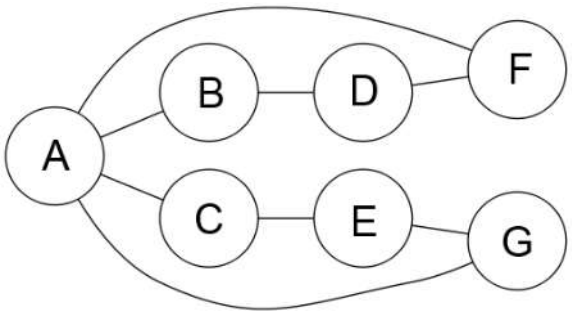
If no solution containing the current assignment exists on a tree structured CSP, then enforcing arc consistency will always result in an empty domain. This means that running arc consistency on a tree structured

CSP will immediately tell you whether or not the current assignment is part of a valid solution, so you can immediately start backtracking without further assignments.

$A - B - C - D - E$ and $C - B - D - E - A$ are both linear orderings of the variables in the tree, which is essentially the same as running the two pass algorithm, which will solve a tree structured CSP with no backtracking.

$A - E - B - D - C$ is not a linear ordering, so while the odd assignments are guaranteed to be part of a valid solution, the even assignments are not (because arc consistency was not enforced after assigning the odd variables). This means that you may have to backtrack on every even assignment, specifically E and D. Note that because you know whether or not the assignment to E is valid immediately after assigning it, the backtracking behavior is not nested (meaning you backtrack on E up to d times without assigning further variables). The same is true for D, so the overall behavior is backtracking $2d$ times.

(ii) [6 pts]



A-B-C-D-E-F-G: d^2

F-D-B-A-C-G-E: $d^4 + d$

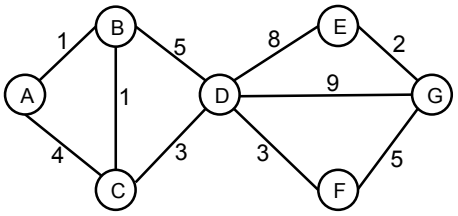
C-A-F-E-B-G-D: d^2

$A - B - C - D - E - F - G$: The initial assignment of A, B might require backtracking on both variables, because there is no guarantee that the initial assignment to A is a valid solution. Because A is a cutset for this graph, the resulting graph consists of two trees, so enforcing arc consistency immediately returns whether the assignments to A and B are part of a solution, and you can begin backtracking without further assignments.

$F - D - B - A - C - G - E$: Until A is assigned, there is no guarantee that any of the previous values assigned are part of a valid solution. This means that you may be required to backtrack on all of them, resulting in d^4 times. Furthermore, the remaining tree is not assigned in a linear order, so further backtracking may be required on G (similar to the second ordering above) resulting in a total of $d^4 + d$.

$C - A - F - E - B - G - D$: This ordering is similar to the first one. except that the resulting trees are not being assigned in linear order. However, because each tree only has a single value assigned in between each run of arc consistency, no backtracking will be required (you can think of each variable as being the root of the tree, and the assignment creating a new tree or two where arc consistency has been enforced), resulting in a total of d^2 times.

Q1. [9 pts] Search



Node	h_1	h_2
A	9.5	10
B	9	12
C	8	10
D	7	8
E	1.5	1
F	4	4.5
G	0	0

Consider the state space graph shown above. A is the start state and G is the goal state. The costs for each edge are shown on the graph. Each edge can be traversed in both directions. Note that the heuristic h_1 is consistent but the heuristic h_2 is not consistent.

(a) [4 pts] **Possible paths returned**

For each of the following graph search strategies (*do not answer for tree search*), mark which, if any, of the listed paths it could return. Note that for some search strategies the specific path returned might depend on tie-breaking behavior. In any such cases, make sure to mark *all* paths that could be returned under some tie-breaking scheme.

Search Algorithm	**A-B-D-G**	**A-C-D-G**	**A-B-C-D-F-G**
Depth first search	x	x	x
Breadth first search	x	x	
Uniform cost search			x
A* search with heuristic h_1			x
A* search with heuristic h_2			x

The return paths depend on tie-breaking behaviors so any possible path has to be marked. DFS can return any path. BFS will return all the shallowest paths, i.e. A-B-D-G and A-C-D-G. A-B-C-D-F-G is the optimal path for this problem, so that UCS and A* using consistent heuristic h_1 will return that path. Although, h_2 is not consistent, it will also return this path.

(b) Heuristic function properties

Suppose you are completing the new heuristic function h_3 shown below. All the values are fixed except $h_3(B)$.

Node	A	B	C	D	E	F	G
h_3	10	?	9	7	1.5	4.5	0

For each of the following conditions, write the set of values that are possible for $h_3(B)$. For example, to denote all non-negative numbers, write $[0, \infty]$, to denote the empty set, write $\varnothing$, and so on.

(i) [1 pt] **What values of $h_3(B)$ make h_3 admissible?**

To make h_3 admissible, $h_3(B)$ has to be less than or equal to the actual optimal cost from B to goal G, which is the cost of path B-C-D-F-G, i.e. 12. The answer is $0 \leq h_3(B) \leq 12$

(ii) [2 pts] **What values of $h_3(B)$ make h_3 consistent?**

All the other nodes except node B satisfy the consistency conditions. The consistency conditions that do involve the state B are:

$$h(A) \leq c(A,B) + h(B) \qquad h(B) \leq c(B,A) + h(A)$$
$$h(C) \leq c(C,B) + h(B) \qquad h(B) \leq c(B,C) + h(C)$$
$$h(D) \leq c(D,B) + h(B) \qquad h(B) \leq c(B,D) + h(D)$$

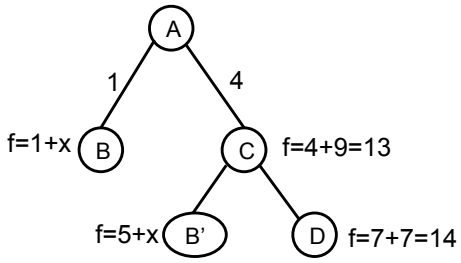
Filling in the numbers shows this results in the condition: $9 \leq h_3(B) \leq 10$

(iii) [2 pts] **What values of $h_3(B)$ will cause A* graph search to expand node A, then node C, then node B, then node D in order?**

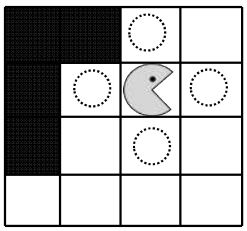
The A* search tree using heuristic h_3 is on the right. In order to make A* graph search expand node A, then node C, then node B, suppose $h_3(B) = x$, we need

$$1 + x > 13$$
$$5 + x < 14 \quad (expand\ B') \quad or \quad 1 + x < 14 \quad (expand\ B)$$

so we can get $12 < h_3(B) < 13$



Q2. [9 pts] InvisiPac



Pacman finds himself to have an invisible "friend", InvisiPac. Whenever InvisiPac visits a square with a food pellet, InvisiPac will eat that food pellet—giving away its location at that time. Suppose the maze's size is MxN and there are F food pellets at the beginning.

Pacman and InvisiPac alternate moves. Pacman can move to any adjacent square (including the one where InvisiPac is) that are not walls, just as in the regular game. After Pacman moves, InvisiPac can teleport into any of the four squares that are adjacent to Pacman, as marked with the dashed circle in the graph. InvisiPac can occupy wall squares.

(a) For this subquestion, whenever InvisiPac moves, it chooses randomly from the squares adjacent to Pacman. The dots eaten by InvisiPac don't count as Pacman's score. Pacman's task is to eat as many food pellets as possible.

(i) [1 pt] Which of the following is best suited to model this problem from Pacman's perspective?

state space search **CSP** **minimax game** [MDP] **RL**

InvisiPac moves to each adjacent square randomly with probably $\frac{1}{4}$. From pacman's point of view, it is a MDP problem with the transition function reflecting this uncertainty.

(ii) [2 pts] What is the size of a minimal state space for this problem? Give your answer as a product of factors that reference problem quantities such as M, N, F, etc. as appropriate. Below each factor, state the information it encodes. For example, you might write $4 \times MN$ and write *number of directions* underneath the first term and *Pacman's position* under the second.

$2^F \times MN$ (boolean vector for whether each food has been eaten, pacman's position)

(b) For this subquestion, whenever InvisiPac moves, it always moves into the same square relative to Pacman. For example, if InvisiPac starts one square North of Pacman, InvisiPac will always move into the square North of Pacman. Pacman knows that InvisiPac is stuck this way, but doesn't know which of the four relative locations he is stuck in. As before, if InvisiPac ends up being in a square with a food pellet, it will eat it and Pacman will thereby find out InvisiPac's location. Pacman's task is to find a strategy that minimizes the worst-case number of moves it could take before Pacman knows InvisiPac's location.

(i) [1 pt] Which of the following is best suited to model this problem from Pacman's perspective?

[**state space search**] **CSP** **minimax game** **MDP** **RL**

The invisiPac will be stuck in one of the four squares relative to Pacman. It is a search problem and state space include the boolean vector for which each of the four locations invisiPac might be. The goal is to reach a state only one possible location the invisiPac can be.

(ii) [2 pts] What is the size of a minimal state space for this problem? Give your answer as a product of factors that reference problem quantities such as M, N, F, etc. as appropriate. Below each factor, state the information it encodes. For example, you might write $4 \times MN$ and write *number of directions* underneath the first term and *Pacman's position* under the second.

$2^F \times MN \times 2^4$ (boolean vector for whether each food has been eaten, pacman's position, boolean vector for which each of the four locations invisiPac might be)

(c) For this subquestion, whenever InvisiPac moves, it can choose freely between any of the four squares adjacent to Pacman. InvisiPac tries to eat as many food pellets as possible. Pacman's task is to eat as many food pellets as possible.

(i) [1 pt] Which of the following is best suited to model this problem from Pacman's perspective?

state space search **CSP** [**minimax game**] **MDP** **RL**

InvisiPac tries to eat as many food pellets as possible, thus plays adversially. It is a minimax game problem.

(ii) [2 pts] What is the size of a minimal state space for this problem? Give your answer as a product of factors that reference problem quantities such as M, N, F, etc. as appropriate. Below each factor, state the information it encodes. For example, you might write $4 \times MN$ and write *number of directions* underneath the first term and *Pacman's position* under the second.
$2^F \times MN$ (boolean vector for whether each food has been eaten, pacman's position)

Q3. [27 pts] CSPs

(a) Pacman's new house

After years of struggling through mazes, Pacman has finally made peace with the ghosts, Blinky, Pinky, Inky, and Clyde, and invited them to live with him and Ms. Pacman. The move has forced Pacman to change the rooming assignments in his house, which has 6 rooms. He has decided to figure out the new assignments with a CSP in which the variables are Pacman **(P)**, Ms. Pacman **(M)**, Blinky **(B)**, Pinky **(K)**, Inky **(I)**, and Clyde **(C)**, the values are which room they will stay in, from 1-6, and the constraints are:

i) No two agents can stay in the same room
ii) **P** > 3
iii) **K** is less than **P**
iv) **M** is either 5 or 6
v) **P** > **M**
vi) **B** is even
vii) **I** is not 1 or 6
viii) |**I-C**| = 1
ix) |**P-B**| = 2

(i) [1 pt] **Unary constraints** On the grid below cross out the values from each domain that are eliminated by enforcing unary constraints.

P	~~1~~	~~2~~	~~3~~	4	5	6
B	~~1~~	2	~~3~~	4	~~5~~	6
C	1	2	3	4	5	6
K	1	2	3	4	5	6
I	~~1~~	2	3	4	5	~~6~~
M	~~1~~	~~2~~	~~3~~	~~4~~	5	6

The unary constraints are ii, iv, vi, and vii. ii crosses out 1,2, and 3 for P. iv crosses out 1,2,3,4 for M. vi crosses out 1,3, and 5 for B. vii crosses out 1 and 6 for I. K and C have no unary constraints, so their domains remain the same.

(ii) [1 pt] **MRV** According to the Minimum Remaining Value (MRV) heuristic, which variable should be assigned to first?

○ P ○ B ○ C ○ K ○ I ● M

M has the fewest value remaining in its domain (2), so it should be selected first for assignment.

(iii) [2 pts] **Forward Checking** For the purposes of decoupling this problem from your solution to the previous problem, assume we choose to assign P first, and assign it the value 6. What are the resulting domains after enforcing unary constraints (from part i) and running forward checking for this assignment?

P						6
B	~~1~~	~~2~~	~~3~~	4	~~5~~	~~6~~
C	1	2	3	4	5	~~6~~
K	1	2	3	4	5	~~6~~
I	~~1~~	2	3	4	5	~~6~~
M	~~1~~	~~2~~	~~3~~	~~4~~	5	~~6~~

In addition to enforcing the unary constraints from part i, the domains are further constrained by all constraints involving P. This includes constraints i, iii, v, and ix. i removes 6 from the domains of all variables. iii removes 6 from the domain of K (already removed by constraint i). v removes 6 from the domain of M (also already removed by i). ix removes 2 and 6 from the domain of B.

(iv) [3 pts] **Iterative Improvement** Instead of running backtracking search, you decide to start over and run iterative improvement with the min-conflicts heuristic for value selection. Starting with the following assignment:

P:6, B:4, C:3, K:2, I:1, M:5

First, for each variable write down how many constraints it violates in the table below.
Then, in the table on the right, for all variables that could be selected for assignment, put an x in any box

that corresponds to a possible value that could be assigned to that variable according to min-conflicts. When marking next values a variable could take on, only mark values different from the current one.

Variable	# violated
P	0
B	0
C	1
K	0
I	2
M	0

	1	2	3	4	5	6
P						
B						
C		x				
K						
I		x		x		
M						

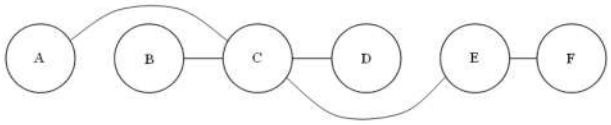
Both I and C violate constraint viii, because |I-C|=2. I also violates constraint vii. No other variables violate any constraints. According to iterative improvement, any conflicted variable could be selected for assignment, in this case I and C. According to min-conflicts, the values that those variables can take on are the values that minimize the number of constraints violated by the variable. Assigning 2 or 4 to I causes it to violate constraint i, because other variables already have the values 2 and 4. Assigning 2 to C also only causes C to violate 1 constraint.

(b) Variable ordering

We say that a variable X is backtracked if, after a value has been assigned to X, the recursion returns at X without a solution, and a different value must be assigned to X.
For this problem, consider the following three algorithms:

1. Run backtracking search with no filtering
2. Initially enforce arc consistency, then run backtracking search with no filtering
3. Initially enforce arc consistency, then run backtracking search while enforcing arc consistency after each assignment

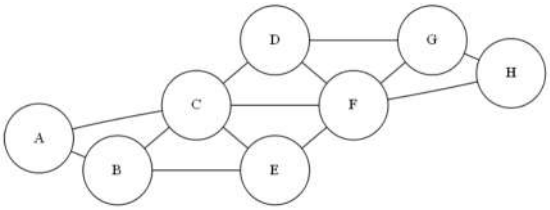
(i) [5 pts]
For each algorithm, circle all orderings of variable assignments that guarantee that no backtracking will be necessary when finding a solution to the CSP represented by the following constraint graph.



Algorithm 1	Algorithm 2	Algorithm 3
A-B-C-D-E-F	A-B-C-D-E-F	A-B-C-D-E-F
F-E-D-C-B-A	F-E-D-C-B-A	F-E-D-C-B-A
C-A-B-D-E-F	C-A-B-D-E-F	C-A-B-D-E-F
B-D-A-F-E-C	B-D-A-F-E-C	B-D-A-F-E-C
D-E-F-C-B-A	D-E-F-C-B-A	D-E-F-C-B-A
B-C-D-A-E-F	B-C-D-A-E-F	B-C-D-A-E-F

Algorithm 1:
No filtering means that there are no guarantees that an assignment to one variable has consistent assignments in any other variable, so backtracking may be necessary.
Algorithm 2:
This algorithm is very similar to the tree-structured CSP algorithm presented in class, in which arcs are enforced from one right to left, and then variables are assigned from left to right. The arcs enforced in that algorithm are a subset of all arcs enforced when enforcing arc consistency. Thus, any linear ordering of variables in which each variable is assigned before all of its children in the tree will guarantee no backtracking.
Algorithm 3:
Any first assignment can be the root of a tree, which, from class, we know is consistent and will not require backtracking. This assignment can then be viewed as conditioning the graph on that variable, and after re-running arc consistency, it can be removed from the graph. This results in either one or two tree-structured graphs that are also arc consistent, and the process can be repeated.

(ii) [5 pts]
For each algorithm, circle all orderings of variable assignments that guarantee that no more than two variables will be backtracked when finding a solution to the CSP represented by the following constraint graph.



Algorithm 1	Algorithm 2	Algorithm 3
C-F-A-B-E-D-G-H	C-F-A-B-E-D-G-H	C-F-A-B-E-D-G-H
F-C-A-H-E-B-D-G	F-C-A-H-E-B-D-G	F-C-A-H-E-B-D-G
A-B-C-E-D-F-G-H	A-B-C-E-D-F-G-H	A-B-C-E-D-F-G-H
G-C-H-F-B-D-E-A	G-C-H-F-B-D-E-A	G-C-H-F-B-D-E-A
A-B-E-D-G-H-C-F	A-B-E-D-G-H-C-F	A-B-E-D-G-H-C-F
A-D-B-G-E-H-C-F	A-D-B-G-E-H-C-F	A-D-B-G-E-H-C-F

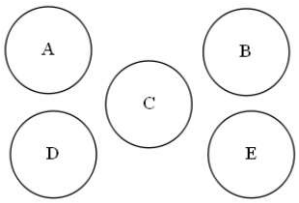
Algorithm 1:
This might backtrack for the same reason as algorithm 1 for the previous problem.
Algorithm 2:
If the first two assignments are not a cutset (C-F, C-G, or F-B), the graph will still contain cycles, for which there is no guarantee that backtracking will not be necessary. If the first two assignments are a cutset, the remaining graph will be a tree. However, because arc consistency was not enforced after the assignment, there is no guarantee against further backtracking. To see this, consider the sub-graph A,B,C,E, with domains {1,2,3}, and constraints A=C, B≥A, E=C+2, B≥C, E=B. If this is assigned in the order C-A-B-E, then by assigning 1 to C and A, assigning either 1 or 2 to B would result in an empty domain for E and cause B to backtrack.
Algorithm 3:
After assigning the cutset, the remaining graph is a tree, which guarantees no further backtracking with algorithm 3 as seen in the previous problem.

(c) All Satisfying Assignments Now consider a modified CSP in which we wish to find every possible satisfying assignment, rather than just one such assignment as in normal CSPs. In order to solve this new problem, consider a new algorithm which is the same as the normal backtracking search algorithm, except that when it sees a solution, instead of returning it, the solution gets added to a list, and the algorithm backtracks. Once there are no variables remaining to backtrack on, the algorithm returns the list of solutions it has found.

For each graph below, select whether or not using the MRV and/or LCV heuristics could affect the number of nodes expanded in the search tree in this new situation.

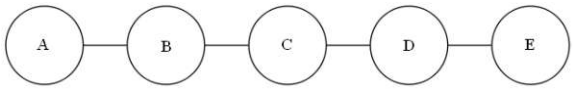
The remaining parts all have a similar reasoning. Since every value has to be checked regardless of the outcome of previous assignments, the order in which the values are checked does not matter, so LCV has no effect.
In the general case, in which there are constraints between variables, the size of each domain can vary based on the order in which variables are assigned, so MRV can still have an effect on the number of nodes expanded for the new "find all solutions" task.
The one time that MRV is guaranteed to not have any effect is when the constraint graph is completely disconnected, as is the case for part i. In this case, the domains of each variable do not depend on any other variable's assignment. Thus, the ordering of variables does not matter, and MRV cannot have any effect on the number of nodes expanded.

(i) [2 pts]



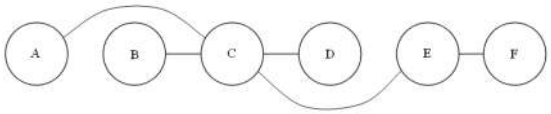
- ● Neither MRV nor LCV can have an effect.
- ○ Only MRV can have an effect.
- ○ Only LCV can have an effect .
- ○ Both MRV and LCV can have an effect.

(ii) [2 pts]



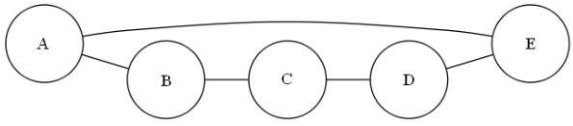
- ○ Neither MRV nor LCV can have an effect.
- ● Only MRV can have an effect.
- ○ Only LCV can have an effect .
- ○ Both MRV and LCV can have an effect.

(iii) [2 pts]



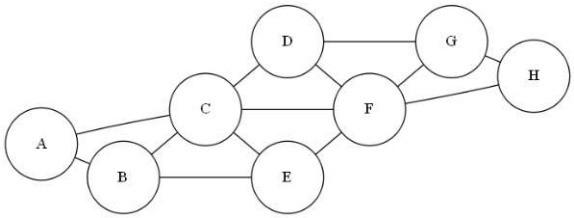
- ○ Neither MRV nor LCV can have an effect.
- ● Only MRV can have an effect.
- ○ Only LCV can have an effect .
- ○ Both MRV and LCV can have an effect.

(iv) [2 pts]



- ○ Neither MRV nor LCV can have an effect.
- ● Only MRV can have an effect.
- ○ Only LCV can have an effect .
- ○ Both MRV and LCV can have an effect.

(v) [2 pts]



- ○ Neither MRV nor LCV can have an effect.
- ● Only MRV can have an effect.
- ○ Only LCV can have an effect .
- ○ Both MRV and LCV can have an effect.

Q2. [17 pts] CSPs: Midterm 1 Staff Assignments

CS188 Midterm I is coming up, and the CS188 staff has yet to write the test. There are a total of 6 questions on the exam and each question will cover a topic. Here is the format of the exam:

- q1. Search
- q2. Games
- q3. CSPs
- q4. MDPs
- q5. True/False
- q6. Short Answer

There are 7 people on the course staff: Brad, Donahue, Ferguson, Judy, Kyle, Michael, and Nick. Each of them is responsible to work with Prof. Abbeel on one question. (But a question could end up having more than one staff person, or potentially zero staff assigned to it.) However, the staff are pretty quirky and want the following constraints to be satisfied:

(i) Donahue (D) will not work on a question together with Judy (J).

(ii) Kyle (K) must work on either Search, Games or CSPs

(iii) Michael (M) is very odd, so he can only contribute to an odd-numbered question.

(iv) Nick (N) must work on a question that's before Michael (M)'s question.

(v) Kyle (K) must work on a question that's before Donahue (D)'s question

(vi) Brad (B) does not like grading exams, so he must work on True/False.

(vii) Judy (J) must work on a question that's after Nick (N)'s question.

(viii) If Brad (B) is to work with someone, it cannot be with Nick (N).

(ix) Nick (N) cannot work on question 6.

(x) Ferguson (F) cannot work on questions 4, 5, or 6

(xi) Donahue (D) cannot work on question 5.

(xii) Donahue (D) must work on a question before Ferguson (F)'s question.

(a) [2 pts] We will model this problem as a constraint satisfaction problem (CSP). Our variables correspond to each of the staff members, J, F, N, D, M, B, K, and the domains are the questions 1, 2, 3, 4, 5, 6. After applying the unary constraints, what are the resulting domains of each variable? (The second grid with variables and domains is provided as a back-up in case you mess up on the first one.)

B					5	
D	1	2	3	4		6
F	1	2	3			
J	1	2	3	4	5	6
K	1	2	3			
N	1	2	3	4	5	
M	1		3		5	

(b) [2 pts] If we apply the Minimum Remaining Value (MRV) heuristic, which variable should be assigned first?

Brad – because he has the least values left in his domain.

(c) [3 pts] Normally we would now proceed with the variable you found in (b), but to decouple this question from the previous one (and prevent potential errors from propagating), let's proceed with assigning Michael first. For value ordering we use the Least Constraining Value (LCV) heuristic, where we use *Forward Checking* to compute the number of remaining values in other variables domains. What ordering of values is prescribed by the LCV heuristic? Include your work—i.e., include the resulting filtered domains that are different for the different values.

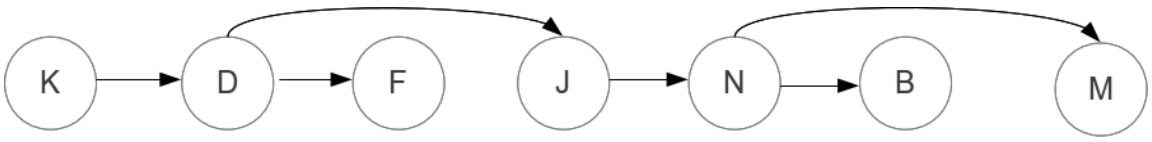
Michael's value will be assigned as 5, 3, 1, in that order.

Why these variables? They are the only feasible variables for Michael. Why this order? This is the increasing order of the number of constraints on each variable.

The only binary constraint incolving Michael is "Nick (N) must work on a question that's before Michael (M)'s question." So, only Nick's domain is affected by forward checking on these assignments, and it will change from {1, 2, 3, 4, 5} to {1, 2, 3, 4}, {1, 2}, and { } for the assignments 5, 3, 1, respectively.

(d) Realizing this is a tree-structured CSP, we decide not to run backtracking search, and instead use the efficient two-pass algorithm to solve tree-structured CSPs. We will run this two-pass algorithm <u>after</u> applying the unary constraints from part (a). Below is the linearized version of the tree-structured CSP graph for you to work with.

(i) [6 pts] **First Pass: Domain Pruning.** Pass from *right to left* to perform Domain Pruning. Write the values that remain in each domain below each node in the figure above.



K	D	F	J	N	B	M
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	3	3	3	3	3
4	4	4	4	4	4	4
5	5	5	5	5	5	5
6	6	6	6	6	6	6

Remaining values in each domain after the domain pruning right-to-left pass:
Kyle: 1
Donahue: 1,2
Ferguson: 1,2,3
Judy: 2,3,4,5,6
Nick: 1,2,3,4
Brad: 5
Michael: 1,3,5

(ii) [4 pts] **Second Pass: Find Solution.** Pass from *left to right*, assigning values for the solution. If there is more than one possible assignment, choose the highest value.

Assigned Values after the left-to-right pass:

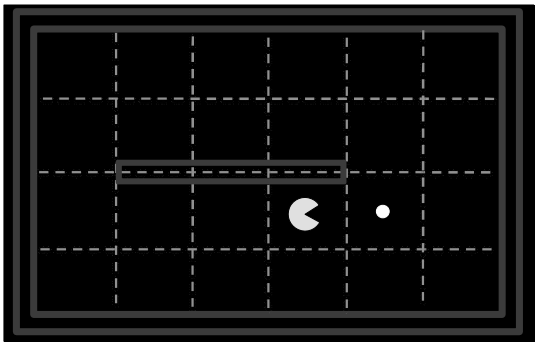
Kyle: 1
Donahue: 2
Ferguson: 3
Judy: 6
Nick: 4
Brad: 5
Michael: 5

Q3. [11 pts] Solving Search Problems with MDPs

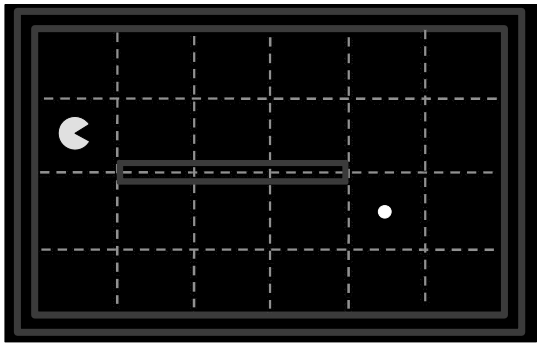
The following parts consider a Pacman agent in a deterministic environment. A goal state is reached when there are no remaining food pellets on the board. Pacman's available actions are $\{N, S, E, W\}$, but Pacman **can not** move into a wall. Whenever Pacman eats a food pellet he receives a reward of +1.

Assume that pacman eats a food pellet as soon as he occupies the location of the food pellet—i.e., the reward is received for the transition into the square with the food pellet.

Consider the particular Pacman board states shown below. Throughout this problem assume that $V_0(s) = 0$ for all states, s. Let the discount factor, $\gamma = 1$.



State A



State B

(a) [2 pts] What is the optimal value of state A, $V^*(A)$?

1

(b) [2 pts] What is the optimal value of state B, $V^*(B)$?

1

The reason the answers are the same for both (b) and (a) is that there is no penalty for existing. With a discount factor of 1, eating the food at any future step is just as valuable as eating it on the next step. An optimal policy will definitely find the food, so the optimal value of any state is always 1.

(c) [2 pts] At what iteration, k, will $V_k(B)$ first be non-zero?

5

The value function at iteration k is equivalent to the maximum reward possible within k steps of the state in question, B. Since the food pellet is exactly 5 steps away from Pacman in state B, $V_5(B) = 1$ and $V_{K<5}(B) = 0$.

(d) [2 pts] How do the optimal q-state values of moving W and E from state A compare? (*choose one*)

○ $Q^*(A, W) > Q^*(A, E)$ ○ $Q^*(A, W) < Q^*(A, E)$ ● $Q^*(A, W) = Q^*(A, E)$

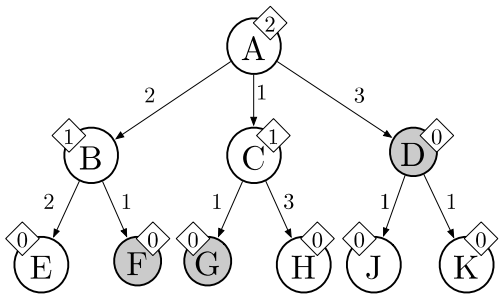
Once again, since $\gamma = 1$, the optimal value of every state is the same, since the optimal policy will eventually eat the food.

(e) [3 pts] If we use this MDP formulation, is the policy found guaranteed to produce the shortest path from pacman's starting position to the food pellet? If not, how could you modify the MDP formulation to guarantee that the optimal policy found will produce the shortest path from pacman's starting position to the food pellet?

No. The Q-values for going $West$ and $East$ from state A are equal so there is no preference given to the shortest path to the goal state. Adding a negative living reward (example: -1 for every time step) will help differentiate between two paths of different lengths. Setting $\gamma < 1$ will make rewards seen in the future worth less than those seen right now, incentivizing Pacman to arrive at the goal as early as possible.

Q1. [10 pts] Search: Algorithms

Consider the state space search problem shown to the right. A is the start state and the shaded states are goals. Arrows encode possible state transitions, and numbers by the arrows represent action costs. Note that state transitions are directed; for example, $A \rightarrow B$ is a valid transition, but $B \rightarrow A$ is not. Numbers shown in diamonds are heuristic values that estimate the optimal (minimal) cost from that node to a goal.



For each of the following search algorithms, write down the nodes that are removed from fringe in the course of the search, as well as the final path returned. Because the original problem graph is a tree, the tree and graph versions of these algorithms will do the same thing, and you can use either version of the algorithms to compute your answer.

Assume that the data structure implementations and successor state orderings are all such that *ties are broken alphabetically*. For example, a partial plan $S \rightarrow X \rightarrow A$ would be expanded before $S \rightarrow X \rightarrow B$; similarly, $S \rightarrow A \rightarrow Z$ would be expanded before $S \rightarrow B \rightarrow A$.

(a) [2 pts] **Depth-First Search (ignores costs)**

Nodes removed from fringe: A, B, E, F

Path returned: A, B, F

(b) [2 pts] **Breadth-First Search (ignores costs)**

Nodes removed from fringe: A, B, C, D

Path returned: A, D

(c) [2 pts] **Uniform-Cost Search**

Nodes removed from fringe: A, C, B, G

Path returned: A, C, G

(d) [2 pts] **Greedy Search**

Nodes removed from fringe: A, D

Path returned: A, D

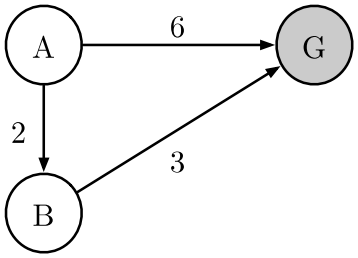
(e) [2 pts] **A* Search**

Nodes removed from fringe: A, C, G

Path returned: A, C, G

Q2. [6 pts] Search: Heuristic Function Properties

For the following questions, consider the search problem shown on the left. It has only three states, and three directed edges. A is the start node and G is the goal node. To the right, four different heuristic functions are defined, numbered I through IV.



	h(A)	h(B)	h(G)
I	4	1	0
II	5	4	0
III	4	3	0
IV	5	2	0

(a) [4 pts] **Admissibility and Consistency**

For each heuristic function, circle whether it is admissible and whether it is consistent with respect to the search problem given above.

	Admissible?		Consistent?	
I	[Yes]	No	Yes	[No]
II	Yes	[No]	Yes	[No]
III	[Yes]	No	[Yes]	No
IV	[Yes]	No	Yes	[No]

II is the only inadmissible heuristic, as it overestimates the cost from B: $h(B) = 4$, when the actual cost to G is 3.

To check whether a heuristic is consistent, ensure that for all paths, $h(N) - h(L) \leq \text{path}(N \to L)$, where N and L stand in for the actual nodes. In this problem, $h(G)$ is always 0, so making sure that the direct paths to the goal ($A \to G$ and $B \to G$) are consistent is the same as making sure that the heuristic is admissible. The path from A to B is a different story.

Heuristic I is not consistent: $h(A) - h(B) = 4 - 1 = 3 \not\leq \text{path}(A \to B) = 2$.
Heuristic III is consistent: $h(A) - h(B) = 4 - 3 = 1 \leq 2$
Heuristic IV is not consistent: $h(A) - h(B) = 5 - 2 = 3 \not\leq 2$

(b) [2 pts] **Function Domination**

Recall that *domination* has a specific meaning when talking about heuristic functions.

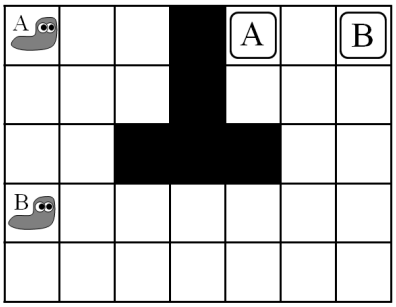
Circle all true statements among the following.

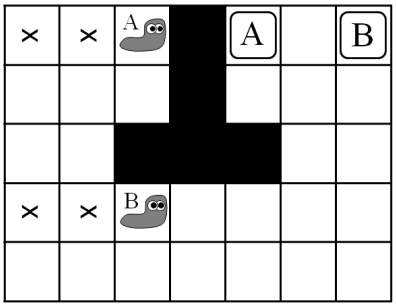
1. Heuristic function III dominates IV.
2. Heuristic function IV dominates III.
3. [Heuristic functions III and IV have no dominance relationship.]
4. Heuristic function I dominates IV.
5. [Heuristic function IV dominates I.]
6. Heuristic functions I and IV have no dominance relationship.

For one heuristic to dominate another, *all* of its values must be greater than or equal to the corresponding values of the other heuristic. Simply make sure that this is the case. If it is not, the two heuristics have no dominance relationship.

Q3. [8 pts] Search: Slugs

You are once again tasked with planning ways to get various insects out of a maze. This time, it's slugs! As shown in the diagram below to the left, two slugs A and B want to exit a maze via their own personal exits. In each time step, both slugs move, though each can choose to either stay in place or move into an adjacent free square. The slugs cannot move into a square that the other slug is moving into. In addition, the slugs leave behind a sticky, poisonous substance and so they cannot move into any square that *either* slug has ever been in. For example, if both slugs move right twice, the maze is as shown in the diagram below to right, with the x squares unpassable to either slug.





You must pose a search problem that will get them to their exits in as few time steps as possible. You may assume that the board is of size N by M; all answers should hold for a general instance, not simply the instance shown above. (You do not need to generalize beyond two slugs.)

(a) [3 pts] How many states are there in a minimal representation of the space? Justify with a brief description of the components of your state space.

$2^{MN}(MN)^2$

The state includes a bit for each of the MN squares, indicating whether the square has been visited (2^{MN} possibilities). It also includes the locations of each slug (MN possibilities for each of the two slugs).

(b) [2 pts] What is the branching factor? Justify with a brief description of the successor function.

$5 \times 5 = 25$ for the first time step, $4 \times 4 = 16$ afterwards.

At the start state each slug has at most five possible next locations (North, South, East, West, Stay). At all future time steps one of those options will certainly be blocked off by the snail's own trail left at the previous time step. Only 4 possible next locations remain.

We accepted both 25 and 16 as correct answers.

(c) [3 pts] Give a non-trivial admissible heuristic for this problem.

max(maze distance of bug A to its exit, maze distance of bug B to its exit)

Many other correct answers are possible.

Q4. [10 pts] Value Functions

Consider a general search problem defined by:

- A set of states, S.
- A start state s_0.
- A set of goal states G, with $G \subset S$.
- A successor function $Succ(s)$ that gives the set of states s' that you can go to from the current state s.
- For each successor s' of s, the cost (weight) $W(s, s')$ of that action.

As usual, the search problem is to find a lowest-cost path from the state state s_0 to a goal $g \in G$. You may assume that each non-goal state has at least one successor, that the weights are all positive, and that all states can reach a goal.

Define $C(s)$ to be the *optimal cost* of the state s; that is, the lowest-cost path from s to any goal. For $g \in G$, clearly $C(g) = 0$.

(a) [4 pts] Write a Bellman-style (one-step lookahead) equation that expresses $C(s)$ for a non-goal s in terms of the optimal costs of other states.

$$C(s) \quad = \quad \min_{s' \in Succ(s)} [W(s, s') + C(s')]$$

(b) [2 pts] Consider a heuristic function $h(s)$ with $h(s) \geq 0$. What relation must hold between $h(s)$ and $C(s)$ for $h(s)$ to be an admissible heuristic? (Your answer should be a mathematical expression.)

$h(s) \leq C(s), \forall s \in S$

(c) [4 pts] By analogy to value iteration, define $C_k(s)$ to be the minimum cost of any plan starting from s that is either length k or reaches a goal in at most k actions. Imagine we use C_k as a heuristic function.

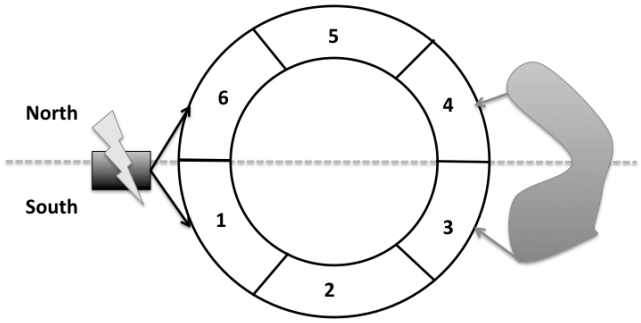
Circle all true statement(s) among the following:

1. $C_k(s)$ might be inadmissible for any given value of k.
2. $\boxed{C_k(s) \text{ is admissible for all } k.}$ If there is a goal reachable within k actions, then $C_k(s)$ gives the exact cost to the nearest such goal. If all goals require plans of longer than k to reach, then the cheapest plan of length k underestimates the true cost.
3. $C_k(s)$ is only guaranteed to be admissible if k exceeds the length of the shortest (in steps) optimal path from a state to a goal.
4. $C_k(s)$ is only guaranteed to be admissible if k exceeds the length of the longest (in steps) optimal path from a state to a goal.
5. $\boxed{C(s) \text{ (the optimal costs) are admissible.}}$
6. $C_k(s)$ might be inconsistent for any given value of k.
7. $\boxed{C_k(s) \text{ is consistent for all } k.}$ Moving from s to a successor s' decreases C_k by at most $W(s, s')$. Since the heuristic value decreases by at most the cost of the transition, the heuristic is consistent.
8. $C_k(s)$ is only guaranteed to be consistent if k exceeds the length of the shortest (in steps) optimal path from a state to a goal.
9. $C_k(s)$ is only guaranteed to be consistent if k exceeds the length of the longest (in steps) optimal path from a state to a goal.
10. $\boxed{C(s) \text{ (the optimal costs) are consistent.}}$

Q5. [9 pts] CSPs: Apple's New Campus

Apple's new circular campus is nearing completion. Unfortunately, the chief architect on the project was using Google Maps to store the location of each individual department, and after upgrading to iOS 6, all the plans for the new campus were lost!

The following is an approximate map of the campus:



The campus has six offices, labeled 1 through 6, and six departments:

- Legal (L)
- Maps Team (M)
- Prototyping (P)
- Engineering (E)
- Tim Cook's office (T)
- Secret Storage (S)

Offices can be *next to* one another, if they share a wall (for an instance, Offices 1-6). Offices can also be *across* from one another (specifically, Offices 1-4, 2-5, 3-6).

The Electrical Grid is connected to offices 1 and 6. The Lake is visible from offices 3 and 4. There are two "halves" of the campus – South (Offices 1-3) and North (Offices 4-6).

The constraints are as follows:

i. (L)egal wants a view of the lake to look for prior art examples.

ii. (T)im Cook's office must not be across from (M)aps.

iii. (P)rototyping must have an electrical connection.

iv. (S)ecret Storage must be next to (E)ngineering.

v. (E)ngineering must be across from (T)im Cook's office.

vi. (P)rototyping and (L)egal cannot be next to one another.

vii. (P)rototyping and (E)ngineering must be on opposite sides of the campus (if one is on the North side, the other must be on the South side).

viii. No two departments may occupy the same office.

This page is repeated as the second-to-last page of this midterm for you to rip out and use for reference as you work through the problem.

(a) [3 pts] **Constraints**. Note: There are multiple ways to model constraint *viii*. In your answers below, assume constraint *viii* is modeled as multiple pairwise constraints, not a large n-ary constraint.

(i) [1 pt] Circle your answers below. Which constraints are unary?

$\boxed{\text{i}}$ ii $\boxed{\text{iii}}$ iv v vi vii $viii$

(ii) [1 pt] In the constraint graph for this CSP, how many edges are there?
Constraint *vii* connects each pair of variables; there are $\binom{6}{2} = 15$ such pairs.

(iii) [1 pt] Write out the explicit form of constraint *iii*.
$P \in \{1, 6\}$

(b) [6 pts] **Domain Filtering.** *We strongly recommend that you use a pencil for the following problems.*

(i) [2 pts] The table below shows the variable domains after unary constraints have been enforced and the value 1 has been assigned to the variable P.
Cross out all values that are eliminated by running Forward Checking after this assignment.

L			3	4		
M	1	2	3	4	5	6
P	1					
E	1	2	3	4	5	6
T	1	2	3	4	5	6
S	1	2	3	4	5	6

(ii) [4 pts] The table below shows the variable domains after unary constraints have been enforced, the value 1 has been assigned to the variable P, and now the value 3 has been assigned to variable T.
Cross out all values that are eliminated if arc consistency is enforced after this assignment. (Note that enforcing arc consistency will subsume all previous pruning.)

L			3	4		
M	1	2	3	4	5	6
P	1					
E	1	2	3	4	5	6
T			3			
S	1	2	3	4	5	6

Q6. [7 pts] CSPs: Properties

(a) [1 pt] When enforcing arc consistency in a CSP, the set of values which remain when the algorithm terminates does not depend on the order in which arcs are processed from the queue.

True **False**

(b) [1 pt] In a general CSP with n variables, each taking d possible values, what is the maximum number of times a backtracking search algorithm might have to backtrack (i.e. the number of the times it generates an assignment, partial or complete, that violates the constraints) before finding a solution or concluding that none exists? (circle one)

0 $O(1)$ $O(nd^2)$ $O(n^2d^3)$ $O(d^n)$ ∞

In general, the search might have to examine all possible assignments.

(c) [1 pt] What is the maximum number of times a backtracking search algorithm might have to backtrack in a general CSP, if it is running arc consistency and applying the MRV and LCV heuristics? (circle one)

0 $O(1)$ $O(nd^2)$ $O(n^2d^3)$ $O(d^n)$ ∞

The MRV and LCV heuristics are often helpful to guide the search, but are not guaranteed to reduce backtracking in the worst case.

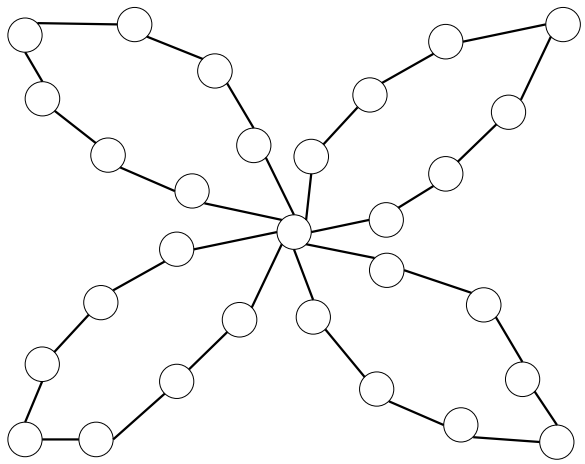
In fact, CSP solving is NP-complete, so any polynomial-time method for solving general CSPs would consititute a proof of $P = NP$ (worth a million dollars from the Clay Mathematics Institute!).

(d) [1 pt] What is the maximum number of times a backtracking search algorithm might have to backtrack in a *tree-structured* CSP, if it is running arc consistency and using an optimal variable ordering? (circle one)

0 $O(1)$ $O(nd^2)$ $O(n^2d^3)$ $O(d^n)$ ∞

Applying arc consistency to a tree-structured CSP guarantees that no backtracking is required, if variables are assigned starting at the root and moving down towards the leaves.

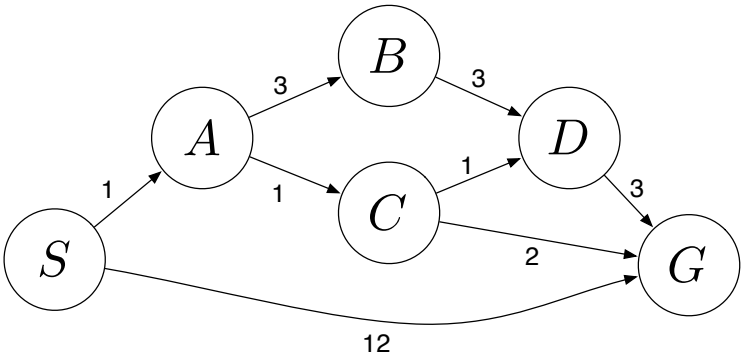
(e) [3 pts] **Constraint Graph** Consider the following constraint graph:



In two sentences or less, describe a strategy for efficiently solving a CSP with this constraint structure.

Loop over assignments to the variable in the middle of the constraint graph. Treating this node as a cutset, the graph becomes four independent tree-structured CSPs, each of which can be solved efficiently.

1. (12 points) Search



Answer the following questions about the search problem shown above. Break any ties alphabetically. For the questions that ask for a path, please give your answers in the form '$S - A - D - G$.'

(a) (2 pt) What path would breadth-first graph search return for this search problem?

$S - G$

(b) (2 pt) What path would uniform cost graph search return for this search problem?

$S - A - C - G$

(c) (2 pt) What path would depth-first graph search return for this search problem?

$S - A - B - D - G$

(d) (2 pt) What path would A* graph search, using a consistent heuristic, return for this search problem?

$S - A - C - G$

(e) (4 pt) Consider the heuristics for this problem shown in the table below.

State	h_1	h_2
S	5	4
A	3	2
B	6	6
C	2	1
D	3	3
G	0	0

i. (1 pt) Is h_1 admissible? **Yes** No

ii. (1 pt) Is h_1 consistent? **Yes** No

iii. (1 pt) Is h_2 admissible? Yes **No**

iv. (1 pt) Is h_2 consistent? **Yes** No

An admissible heuristic must underestimate or be equal to the true cost.
A consistent heuristic must satisfy $h(N) - h(L) \leq \text{path}(N \to L)$ for all paths and nodes N and L.

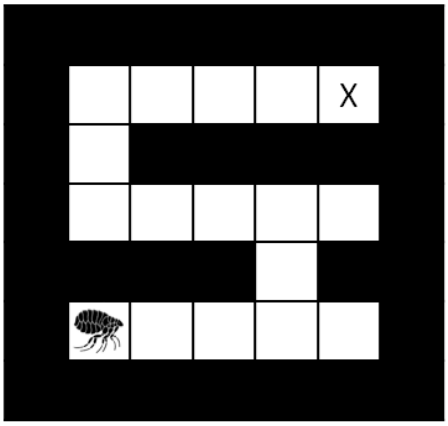
$h1$ overestimates the cost $S \to G$ as 5 when it is 4, so it is inadmissible.
$h1$ is not consistent because $h(S) - h(A) \leq \text{path}(S \to A)$ is violated as $5 - 3 \leq 1$.
$h2$ does not overestimate costs and is admissible.
$h2$ is not consistent because $h(S) - h(A) \leq \text{path}(S \to A)$ is violated as $4 - 2 \leq 1$.

2. (12 points) Hive Minds: Redux

Let's revisit our bug friends from assignment 2. To recap, you control one or more insects in a rectangular maze-like environment with dimensions $M \times N$, as shown in the figures below. At each time step, an insect can move North, East, South, or West (but not diagonally) into an adjacent square if that square is currently free, or the insect may stay in its current location. Squares may be blocked by walls (as denoted by the black squares), but the map is known.

For the following questions, you should answer for a general instance of the problem, not simply for the example maps shown.

(a) (6 pt) The Flea



You now control a single flea as shown in the maze above, which must reach a designated target location X. However, in addition to moving along the maze as usual, your flea can jump on top of the walls. When on a wall, the flea can walk along the top of the wall as it would when in the maze. It can also jump off of the wall, back into the maze. Jumping onto the wall has a cost of 2, while all other actions (including jumping back into the maze) have a cost of 1. Note that the flea can only jump onto walls that are in adjacent squares (either north, south, west, or east of the flea).

i. (2 pt) Give a *minimal* state representation for the above search problem.

The state is the location of the flea as an (x, y) coordinate. The map is known, including walls and the goal, and the actions of the flea depend only on its location.

ii. (2 pt) Give the size of the state space for this search problem.

The state space is $M \times N$. The flea can occupy any free location in a given maze, and any square might be free or a wall in a maze, so any of the $M \times N$ locations are possible.

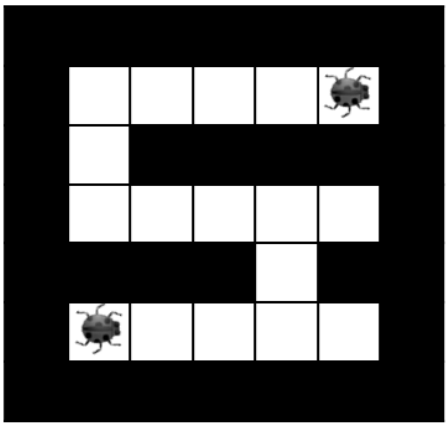
iii. (2 pt) Is the following heuristic admissible? **Yes** **No**

$$h_{\text{flea}} = \text{the Manhattan distance from the flea to the goal.}$$

It is yielded by the relaxed problem where the flea passes through walls. It never overestimates because 1. a wall can never decrease the length of a path to the goal and 2. the cost of the flea jumping up a wall (2) is higher than the cost of moving.

If it is *not* admissible, provide a nontrivial admissible heuristic in the space below.

(b) (6 pt) Long Lost Bug Friends



You now control a pair of long lost bug friends. You know the maze, but you do not have any information about which square each bug starts in. You want to help the bugs reunite. You must pose a search problem whose solution is an all-purpose sequence of actions such that, after executing those actions, both bugs will be on the same square, regardless of their initial positions. Any square will do, as the bugs have no goal in mind other than to see each other once again. Both bugs execute the actions mindlessly and do not know whether their moves succeed; if they use an action which would move them in a blocked direction, they will stay where they are. Unlike the flea in the previous question, bugs *cannot* jump onto walls. Both bugs can move in each time step. Every time step that passes has a cost of one.

i. (2 pt) Give a *minimal* state representation for the above search problem.

The state is a list of boolean variables, one for each position in the maze, which marks whether the position could contain a bug. There is no need to separately keep track of the bugs since their starting positions are not known; to ensure they meet only a single square must be possible for both.

ii. (2 pt) Give the size of the state space for this search problem.

The size is 2^{MN} since every of the $M \times N$ possible maze positions must be considered and every position has a boolean variable. A full state is the product of the individual position states, which are binary valued for the base of 2.

iii. (2 pt) Give a nontrivial admissible heuristic for this search problem.

h_{friends} = the maximum Manhattan distance of all possible pairs of points the bugs can be in.
This is never an overestimate because the number of steps to join the insects with certainty is at least the shortest path (with no obstacles) between their farthest possible locations from one another. Remember that the starting locations are unknown so the bugs cannot simply be controlled to move toward each other.

3. (12 points) A* Graph Search

```
function A* Graph Search(problem)
    fringe ← an empty priority queue
    fringe ← Insert(Make-Node(Initial-State[problem]), fringe)
    closed ← an empty set
    Add Initial-State[problem] to closed
    loop
        if fringe is empty then
            return failure
        end if
        node ← Remove-Front(fringe)
        if Goal-Test(problem, State[node]) then
            return node
        end if
        for successor in GetSuccessors(problem, State[node]) do
            if successor not in closed then
                Add successor to closed
                fringe ← Insert(Make-Successor-Node(successor, node), fringe)
            end if
        end for
    end loop
end function
```

The above implementation of A graph search may be incorrect!* In the list below circle all of the problems that the bugs may cause when executing graph search and justify your answer. Note that the fringe is a priority queue. Nodes are inserted into the fringe using the standard key for A*, namely $f = g + h$. h is a consistent heuristic.

(a) The GetSuccessors function could be called multiple times on the same state.

(b) The algorithm is no longer complete.

(c) The algorithm could return a suboptimal solution.

(d) The implementation is incorrect, but none of the above problems will be caused.

(e) The implementation is correct.

To receive any credit you must briefly justify your answer in space below.

The bug is the insertion of *successor* into *closed* at time of insertion of a node into the fringe, rather than at the time that node gets popped from the fringe. As a consequence of this bug, the first path encountered to a state will put that state in the closed list. This can cause suboptimality as we only have the guarantee that a state has been reached optimally once a node reaching it gets popped off the fringe.

(a) is False as when a node that reaches a state s is placed in the fringe, that state s is also put on the closed list. This means never in the future can a node be placed on the fringe that ends in that same state s, and hence the same state s can be the argument to GetSuccessors at most once.

(b) is False. A* tree search is complete. The difference is that the above algorithm will cut off parts of the tree search whenever it has placed a node on the fringe in the past that ends in the same state. So compared to tree search we only lose copies of subtrees that we are covering. Hence the above algorithm is complete.

(c) is True. See explanation at beginning of solution.

(d) is False.

(e) is False.

4. (21 points) Time Management

Two of our GSIs, Arjun and Woody, are making their schedules for a busy morning. There are five tasks to be carried out:

(F) Pick up food for the group's research seminar, which, sadly, takes one precious hour.
(H) Prepare homework questions, which takes 2 consecutive hours.
(P) Prepare the PR2 robot for a group of preschoolers' visit, which takes one hour.
(S) Lead the research seminar, which takes one hour.
(T) Teach the preschoolers about the PR2 robot, which takes 2 consecutive hours.

The schedule consists of one-hour slots: 8am-9am, 9am-10am, 10am-11am, 11am-12pm. The requirements for the schedule are as follows:

(a) In any given time slot each GSI can do at most one task (F, H, P, S, T).

(b) The PR2 preparation (P) should happen before teaching the preschoolers (T).

(c) The food should be picked up (F) before the seminar (S).

(d) The seminar (S) should be finished by 10am.

(e) Arjun is going to deal with food pick up (F) since he has a car.

(f) The GSI not leading the seminar (S) should still attend, and hence cannot perform another task (F, T, P, H) during the seminar.

(g) The seminar (S) leader does not teach the preschoolers (T).

(h) The GSI who teaches the preschoolers (T) must also prepare the PR2 robot (P).

(i) Preparing homework questions (H) takes 2 consecutive hours, and hence should start at or before 10am.

(j) Teaching the preschoolers (T) takes 2 consecutive hours, and hence should start at or before 10am.

To formalize this problem as a CSP, use the variables F, H, P, S and T. The values they take on indicate the GSI responsible for it, and the starting time slot during which the task is carried out (for a task that spans 2 hours, the variable represents the starting time, but keep in mind that the GSI will be occupied for the next hour also - make sure you enforce constraint (a)!). Hence there are eight possible values for each variable, which we will denote by A8, A9, A10, A11, W8, W9, W10, W11, where the letter corresponds to the GSI and the number corresponds to the time slot. For example, assigning the value of A8 to a variables means that this task is carried about by Arjun from 8am to 9am.

(a) (2 pt) What is the size of the state space for this CSP?

8^5 since every task variable has 8 values, 4 time slots $\times$ 2 GSIs to carry them out, and there are 5 such tasks.

(b) (2 pt) Which of the statements above include unary constraints?

A unary constraint constrains the domain of a single variable. (d), (e), (i), (j) are unary constraints. (i) and (j) express both unary constraints (on the time of the tasks) and binary constraints (the length of tasks excludes other assignments during their time).

(c) (4 pt) In the table below, enforce all unary constraints by crossing out values in the table on the left below. If you made a mistake, cross out the whole table and use the right one.

F	A8	A9	A10	A11	~~W8~~	~~W9~~	~~W10~~	~~W11~~
H	A8	A9	A10	~~A11~~	W8	W9	W10	~~W11~~
P	A8	A9	A10	A11	W8	W9	W10	W11
S	A8	A9	~~A10~~	~~A11~~	W8	W9	~~W10~~	~~W11~~
T	A8	A9	A10	~~A11~~	W8	W9	W10	~~W11~~

(d) (3 pt) Start from the table above, select the variable S and assign the value A9 to it. Perform forward checking by crossing out values in the table below. Again the table on the right is for you to use in case you believe you made a mistake.

F	A8	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
H	~~A8~~	~~A9~~	A10	~~A11~~	~~W8~~	~~W9~~	W10	~~W11~~
P	A8	~~A9~~	A10	A11	W8	~~W9~~	W10	W11
S	~~A8~~	A9	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
T	~~A8~~	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	W10	~~W11~~

Forward checking prunes the variables' domains of values inconsistent with S = A9, including: other choices for S, conflicting time slots for A9 (a), the choices of F that do not precede 9 (c), W from working during 9 (f), and A teaching the preschoolers (g).

(e) **(3 pt)** Based on the result of (d), what variable will we choose to assign next based on the MRV heuristic (breaking ties alphabetically)? Assign the first possible value to this variable, and perform forward checking by crossing out values in the table below. Again the table on the right is for you to use in case you believe you made a mistake.

Variable _F_ is selected and gets assigned value _A8_.

F	A8	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
H	~~A8~~	~~A9~~	A10	~~A11~~	~~W8~~	~~W9~~	W10	~~W11~~
P	~~A8~~	~~A9~~	A10	A11	W8	~~W9~~	W10	W11
S	~~A8~~	A9	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
T	~~A8~~	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	W10	~~W11~~

Have we arrived at a dead end (i.e., has any of the domains become empty)?

No.

(f) **(4 pt)** We return to the result from enforcing just the unary constraints, which we did in (c). Select the variable S and assign the value A9. Enforce arc consistency by crossing out values in the table below.

F	A8	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
H	~~A8~~	~~A9~~	A10	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
P	~~A8~~	~~A9~~	~~A10~~	~~A11~~	W8	~~W9~~	~~W10~~	~~W11~~
S	~~A8~~	A9	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
T	~~A8~~	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	W10	~~W11~~

(g) **(2 pt)** Compare your answers to (d) and to (f). Does arc consistency remove more values or less values than forward checking does? Explain why.

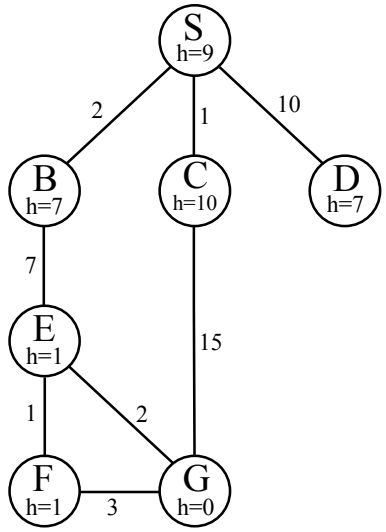
Arc consistency removes more values by checking more relationships between variables: AC checks consistency between every pair of variables, and re-checks after domain pruning, while FC only checks between assigned and unassigned variables.

(h) **(1 pt)** Check your answer to (f). Without backtracking, does any solution exist along this path? Provide the solution(s) or state that there is none.

AC along this path gives 1 solution: F: _A8_ H: _A10_ P: _W8_ S: _A9_ T: _W10_
Backtracking is unnecessary since the constraints have been enforced by arc consistency and only single values remained in each domain.

1. (12 points) Search

Consider the search graph shown below. S is the start state and G is the goal state. All edges are bidirectional.



For each of the following search strategies, give the path that would be returned, or write *none* if no path will be returned. If there are any ties, assume alphabetical tiebreaking (i.e., nodes for states earlier in the alphabet are expanded first in the case of ties).

(a) (1 pt) Depth-first graph search

S-B-E-F-G

(b) (1 pt) Breadth-first graph search

S-C-G

(c) (1 pt) Uniform cost graph search

S-B-E-G

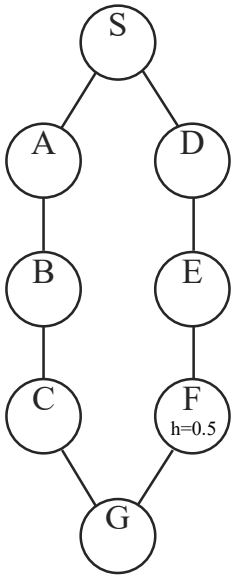
(d) (1 pt) Greedy graph search

S-B-E-G

(e) (2 pt) A* graph search

S-B-E-G

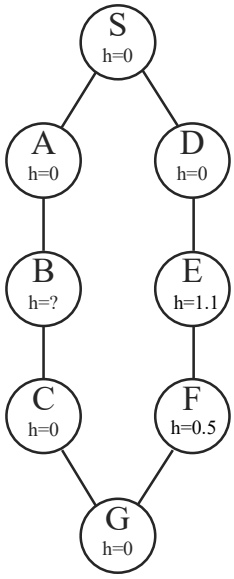
For the following question parts, all edges in the graphs discussed have cost 1.



(f) **(3 pt)** Suppose that you are designing a heuristic h for the graph on the right. You are told that $h(F) = 0.5$, but given no other information. What ranges of values are possible for $h(D)$ if the following conditions must hold? Your answer should be a range, e.g. $2 \leq h(D) < 10$. You may assume that h is nonnegative.

i. h must be admissible
$0 \leq h(D) \leq 3$
The path to goal from D is 3.

ii. h must be admissible and consistent
$0 \leq h(D) \leq 2.5$
In order for $h(E)$ to be consistent, it must hold that $h(E) - h(F) \leq 1$, since the path from E to F is of cost 1. Similarly, it must hold that $h(D) - h(F) = h(D) - 0.5 \leq 2$, or $h(D) \leq 2.5$.



(g) **(3 pt)** Now suppose that $h(\text{F}) = 0.5$, $h(\text{E}) = 1.1$, and all other heuristic values except $h(\text{B})$ are fixed to zero (as shown on the right). For each of the following parts, indicate the range of values for $h(\text{B})$ that yield an admissible heuristic AND result in the given expansion ordering when using A* graph search. If the given ordering is impossible with an admissible heuristic, write *none*. Break ties alphabetically. Again, you may assume that h is nonnegative.

i. B expanded before E expanded before F
$0.0 \leq h(\text{B}) \leq 1.1$

ii. E expanded before B expanded before F
$1.1 < h(\text{B}) \leq 1.5$

2. (12 points) Formulation: Holiday Shopping

You are programming a holiday shopping robot that will drive from store to store in order to buy all the gifts on your shopping list. You have a set of N gifts $G = \{g_1, g_2, \ldots g_N\}$ that must be purchased. There are M stores, $S = \{s_1, s_2, \ldots s_M\}$ each of which stocks a known inventory of items: we write $g_k \in s_i$ if store s_i stocks gift g_k. Shops may cover more than one gift on your list and will never be out of the items they stock. Your home is the store s_1, which stocks no items.

The actions you will consider are travel-and-buy actions in which the robot travels from its current location s_i to another store s_j in the fastest possible way and buys whatever items remaining on the shopping list that are sold at s_j. The time to travel-and-buy from s_i to s_j is $t(s_i, s_j)$. You may assume all travel-and-buy actions represent shortest paths, so there is no faster way to get between s_i and s_j via some other store. The robot begins at your home with no gifts purchased. You want it to buy all the items in as short a time as possible and return home.

For this planning problem, you use a state space where each state is a pair (s, u) where s is the current location and u is the set of unpurchased gifts on your list (so $g \in u$ indicates that gift g has not yet been purchased).

(a) (1 pt) How large is the state space in terms of the quantities defined above?

$M \times 2^N$. You are in one of M places (simple index from 1 to M), and have not purchased some subset of N items (binary vector of size N).

(b) (4 pt) For each of the following heuristics, which apply to states (s, u), circle whether it is admissible, consistent, neither, or both. Assume that the minimum of an empty set is zero.

([neither] / admissible / consistent / both) The shortest time from the current location to any other store: $\min_{s' \neq s} t(s, s')$

(neither / admissible / consistent / [both]) The time to get home from the current location: $t(s, s_1)$

(neither / admissible / consistent / [both]) The shortest time to get to any store selling any unpurchased gift: $\min_{g \in u}(\min_{s': g \in s'} t(s, s'))$

(neither / [admissible] / consistent / both) The shortest time to get home from any store selling any unpurchased gift: $\min_{g \in u}(\min_{s': g \in s'} t(s', s_1))$

([neither] / admissible / consistent / both) The total time to get each unpurchased gift individually: $\sum_{g \in u}(\min_{s': g \in s'} t(s, s'))$

([neither] / admissible / consistent / both) The number of unpurchased gifts times the shortest store-to-store time: $|u|(\min_{s_i, s_j \neq s_i} t(s_i, s_j))$

Remember, a consistent heuristic doesn't decrease from state to state by more than it actually costs to get from state to state. And of course, a heuristic is admissible if it is consistent. If you're confused, remember: the problem defines the minimum of an empty set as 0.

i. This heuristic does not return 0 in the goal state (s_1, $\emptyset$), since it gives the minimum distance to any store *other than the current one*.

ii. We'll always need to get home from any state; the distance to home from home is 0; and this heuristic does not decrease by more than it costs to get from state to state.

iii. We'll always need to get that last unpurchased item, and taking the min distance store guarantees that we underestimate how much distance we actually have to travel. It is consistent because the heuristic never diminishes by more than what is travelled.

iv. We'll always need to get home from getting the last unpurchased item, and taking the min underestimates the actual requirement. What makes this heuristic inconsistent is that when we visit the last store to pick up the last unfinished item, the value of the heuristic goes to 0. Let's say the graph looks like this: s_3 `--1-->` s_2 `----5---->` s_1, with s_2 containing the last item. From s_3, the heuristic is 5, but from s_2, the heuristic is now 0, meaning that traveling from s_3 to s_2 decreases the heuristic by 5 but the actual cost is only 1.

v. This can overestimate the actual amount of work required.

vi. Same.

You have waited until very late to do your shopping, so you decide to send an swarm of R robot minions to shop in parallel. Each robot moves at the same speed, so the same store-to-store times apply. The problem is now to have all robots start at home, end at home, and for each item to have been bought by at least one robot (you don't have to worry about whether duplicates get bought). Hint: consider that robots may not all arrive at stores in sync.

(c) (4 pt) Give a minimal state space for this search problem (be formal and precise!)

We need the location of each robot at each time. At a given time, a robot can either be at one of M stores, or in any of $(T-1)M$ transition locations, where T is the maximum travel distance between two stores. Thus, the location of each robot takes $(MT)^R$. We also need the set of items purchased (2^N). Therefore, the size of each state is: $(MT)^R \times 2^N$.

One final task remains: you still must find your younger brother a stuffed Woozle, the hot new children's toy. Unfortunately, no store is guaranteed to stock one. Instead, each store s_i has an initial probability p_i of still having a Woozle available. Moreover, that probability drops exponentially as other buyers scoop them up, so after t time has passed, s_i's probability has dropped to $\beta^t p_i$. You cannot simply try a store repeatedly; once it is out of stock, that store will stay out of stock. Worse, you only have a single robot that can handle this kind of uncertainty! Phrase the problem as a single-agent MDP for planning a search policy for just this one gift (no shopping lists). You receive a single reward of +1 upon successfully buying a Woozle, at which point the MDP ends (don't worry about getting home); all other rewards are zeros. You may assume a discount of 1.

(d) (3 pt) Give a minimal state space for this MDP (be formal and precise!)

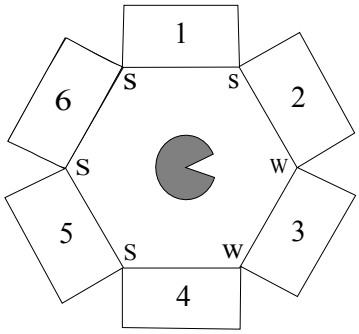
Which stores have been checked: 2^M
Whether Woozle has been bought: 2
Current time: T.
We may also want to keep track of the current location (M), but since there is no reward for traveling, we don't have to model that aspect of the problem.

3. (13 points) CSPs: Trapped Pacman

Pacman is trapped! He is surrounded by mysterious corridors, each of which leads to either a pit (P), a ghost (G), or an exit (E). In order to escape, he needs to figure out which corridors, if any, lead to an exit and freedom, rather than the certain doom of a pit or a ghost.

The one sign of what lies behind the corridors is the wind: a pit produces a strong breeze (S) and an exit produces a weak breeze (W), while a ghost doesn't produce any breeze at all. Unfortuantely, Pacman cannot measure the the strength of the breeze at a specific corridor. Instead, he can stand *between* two adjacent corridors and feel the max of the two breezes. For example, if he stands between a pit and an exit he will sense a strong (S) breeze, while if he stands between an exit and a ghost, he will sense a weak (W) breeze. The measurements for all intersections are shown in the figure below.

Also, while the total number of exits might be zero, one, or more, Pacman knows that two neighboring squares will *not* both be exits.



Pacman models this problem using variables X_i for each corridor i and domains P, G, and E.

(a) (3 pt) State the binary and/or unary constraints for this CSP (either implicitly or explicitly).

From the breezes, we get the following constraints:

Binary		Unary
$X_1 = P$ or $X_2 = P$,	$X_2 = E$ or $X_3 = E$,	$X_2 \neq P$
$X_3 = E$ or $X_4 = E$,	$X_4 = P$ or $X_5 = P$,	$X_3 \neq P$
$X_5 = P$ or $X_6 = P$,	$X_1 = P$ or $X_6 = P$,	$X_4 \neq P$

And there is another binary constraint: If adjacent(i, j), then $\neg(X_i = E \wedge X_j = E)$.

(b) (4 pt) Cross out the values from the domains of the variables that will be deleted in enforcing arc consistency.

X_1	P		
X_2		G	E
X_3		G	E
X_4		G	E
X_5	P		
X_6	P	G	E

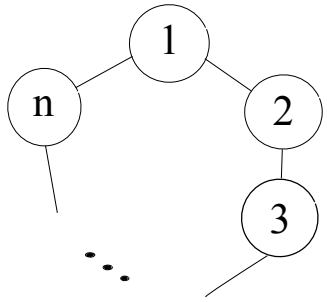
(c) (1 pt) According to MRV, which variable or variables could the solver assign first?

X_1 or X_5 (tie breaking)

(d) (1 pt) Assume that Pacman knows that $X_6 = G$. List all the solutions of this CSP or write *none* if no solutions exist.

(P,E,G,E,P,G)
(P,G,E,G,P,G)
Don't forget that exits cannot be adjacent to each other, and that it takes at least one exit to generate a weak breeze.

The CSP described above has a circular structure with 6 variables. Now consider a CSP forming a circular structure that has n variables ($n > 2$), as shown below. Also assume that the domain of each variable has cardinality d.



(e) (2 pt) Explain precisely how to solve this general class of circle-structured CSPs efficiently (i.e. in time linear in the number of variables), using methods covered in class. Your answer should be at most two sentences.

We fix X_j for some j and assign it a value from its domain (i.e. use cutset conditioning on one variable). The rest of the CSP now forms a tree structure, which can be efficiently solved without backtracking by one backward arc-enforcing and one forward value-setting pass. We try all possible values for our selected variable X_j until we find a solution.

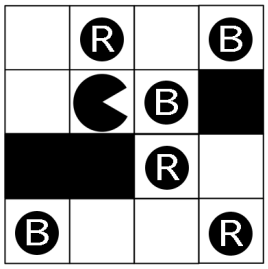
(f) (2 pt) If standard backtracking search were run on a circle-structured graph, enforcing arc consistency at every step, what, if anything, can be said about the worst-case backtracking behavior (e.g. number of times the search could backtrack)?

A tree structured CSP can be solved without any backtracking. Thus, the above circle-structured CSP can be solved after backtracking at most d times, since we might have to try up to d values for X_j before finding a solution.

CS 188 Spring 2011 | Introduction to Artificial Intelligence | Midterm Exam Solutions

Q1. [11 pts] Foodie Pacman

There are two kinds of food pellets, each with a different color (red and blue). Pacman is only interested in tasting the two different kinds of food: the game ends when he has eaten 1 red pellet and 1 blue pellet (though Pacman may eat more than one of each pellet). Pacman has four actions: moving up, down, left, or right, and does not have a "stay" action. There are K red pellets and K blue pellets, and the dimensions of the board are N by M.



$K = 3$, $N = 4$, $M = 4$

(a) [1 pt] Give an efficient state space formulation of this problem. Specify the domain of each variable in your state space.

We need two variables to describe the location of pacman, one boolen variable showing whether pacmac already ate a red pellet, and another boolean variable for the blue pellets. Formally:

$(x \in [1:N], y \in [1:M], eaten_R \in \{T, F\}, eaten_B \in \{T, F\})$

(b) [2 pts] Give a tight upper bound on the size of the state space.

There are at most $N \times M$ possible locations for pacman and 4 possible assignments to the boolean variables so the size of the state space is upper bounded by $4 \times N \times M$

(c) [2 pts] Give a tight upper bound on the branching factor of the search problem.

Each state has at most four distinct successors corresponding to the four possible actions. The branching factor is at most 4.

(d) [1 pt] Assuming Pacman starts the game in position (x,y), what is the initial state?

(x, y, F, F). The two boolean state variables are both $false$.

(e) [1 pt] Define a goal test for the problem.

$(eaten_R == T)\&\&(eaten_B == T)$

(f) [4 pts] For each of the following heuristics, indicate (yes/no) whether or not it is admissible (a correct answer is worth 1 point, leaving it blank is worth 0 points, and an incorrect answer is worth -1 points).

Q4. [8 pts] Search

(a) [4 pts] The following implementation of graph search may be incorrect. Circle all the problems with the code.

```
function GRAPH-SEARCH(problem, fringe)
    closed ← an empty set,
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop
        if fringe is empty then
            return failure
        end if
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem,STATE[node]) then
            return node
        end if
        ADD STATE[node] TO closed
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end loop
end function
```

1. Nodes may be expanded twice.
2. The algorithm is no longer complete.
3. The algorithm could return an incorrect solution.
4. None of the above.

The stated algorithm is equivalent to tree search. In graph search, nodes added to the "closed" list should not be expanded again. Since this algorithm does not do that, it can get stuck in a loop and that is why it is not complete.

(b) [4 pts] The following implementation of A* graph search may be incorrect. You may assume that the algorithm is being run with a consistent heuristic. Circle all the problems with the code.

```
function A*-SEARCH(problem, fringe)
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop
        if fringe is empty then
            return failure
        end if
        node ← REMOVE-FRONT(fringe)
        if STATE[node] IS NOT IN closed then
            ADD STATE[node] TO closed
            for successor IN GETSUCCESSORS(problem, STATE[node]) do
                fringe ← INSERT(MAKE-NODE(successor), fringe)
                if GOAL-TEST(problem,successor) then
                    return successor
                end if
            end for
        end if
    end loop
end function
```

1. Nodes may be expanded twice.
2. The algorithm is no longer complete.
3. The algorithm could return an incorrect solution.
4. None of the above.

The stated algorithm expands fewer nodes to find a goal, but it does not always find the optimal goal in terms of cost. Note that "incorrect" means that it "not optimal" here.

Q6. [13 pts] Crossword Puzzles as CSPs

You are developing a program to automatically solve crossword puzzles, because you think a good income source for you might be to submit them to the New York Times ($200 for a weekday puzzle, $1000 for a Sunday).[1] For those unfamiliar with crossword puzzles, a crossword puzzle is a game in which one is given a grid of squares that must be filled in with intersecting words going from left to right and top to bottom. There are a given set of starting positions for words (in the grid below, the positions 1, 2, 3, 4, and 5), where words must be placed going across (left to right) or down (top to bottom). At any position where words intersect, the letters in the intersecting words must match. Further, no two words in the puzzle can be identical. An example is the grid below, in which the down words (1, 2, and 3) are `DEN`, `ARE`, and `MAT`, while the across words (1, 4, and 5) are `DAM`, `ERA`, and `NET`.

Example Crossword Grid and Solution

[1]D	[2]A	[3]M
[4]E	R	A
[5]N	E	T

A part of your plan to make crosswords, you decide you will create a program that uses the CSP solving techniques you have learned in CS 188, since you want to make yourself obsolete at your own job from the get-go. Your first task is to choose the representation of your problem. You start with a dictionary of all the words you could put in the crossword puzzle, where the dictionary is of size K and consists of the words $\{d_1, d_2, \ldots, d_K\}$. Assume that you are given a grid with N empty squares and M different entries for words (and there are 26 letters in the English language). In the example above, $N = 9$ and $M = 6$ (three words across and three words down).

You initially decide to use words as the variables in your CSP. Let D_1 denote the first down word, D_2 the second, D_3 the third, etc., and similarly let A_k denote the kth across word. For example, in the crossword above, $A_1 =$ `DAM`, $D_1 =$ `DEN`, $D_2 =$ `ARE`, and so on. Let $D_1[i]$ denote the letter in the ith position of the word D_1.

(a) [1 pt] What is the size of the state space for this CSP?

Several answers are acceptable for this problem. The simplest is that the dictionary has size K and there are M words, giving state space size K^M. A slightly tighter bound is achieved by noting that once one word is placed, the next words must all be different, giving $K(K-1)(K-2)\cdots(K-M+1) = \frac{K!}{(K-M)!}$. Noticing that we are choosing M distinct words out of a possible K gives the state space bound $\binom{K}{M}$.

Several students tried to include N in their answers; since the letters have nothing to do with this formulation of the problem, this was incorrect. Many students also incorrectly had M^K.

(b) [3 pts] Precisely (i.e. use mathematical notation to) describe the constraints of the CSP when we use words as variables.

For every pair of across and down words D_k and A_l that intersect, we have the constraint that their letters are equal. Specifically, if they intersect in positions i and j, we have $D_k[i] = A_l[j]$.

We also have the pairwise constraints that none of the words are the same: for $k \neq k'$, $D_k \neq D_{k'}$ and $A_k \neq A_{k'}$, and for all k, k', we have $A_k \neq D_{k'}$.

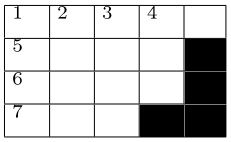
In addition, each word must have the correct length. One possible formulation is that for all $L \in \mathbb{N}$, for all words D_k and A_l with length L in the puzzle, we have $\text{length}(D_k) = L$ and $\text{length}(A_l) = L$.

The biggest problem that students had was assuming that all crossword puzzles were contiguous squares (or rectangles) like the example. While that works for the above example, it will not work generally. Several students missed one or two of the above constraints, and all three were necessary for full credit. Minor mistakes included missing a few of the inequality constraints.

[1] `http://www.nytimes.com/2009/07/19/business/media/19askthetimes.html`

After defining your CSP, you decide to go ahead and make a small crossword using the grid below. Assume that you use the words on the right as your dictionary.

Crossword Grid



Dictionary Words

ARCS, BLAM, BEAR, BLOGS, LARD, LARP,
GAME, GAMUT, GRAMS, GPS, MDS, ORCS, WARBLER

(c) [1 pt] Enforce all *unary* constraints by crossing out values in the table below.

D_1	ARCS	BLAM	BEAR	~~BLOGS~~	LARD	LARP	~~GPS~~	~~MDS~~	GAME	~~GAMUT~~	~~GRAMS~~	ORCS	~~WARBLER~~
D_2	ARCS	BLAM	BEAR	~~BLOGS~~	LARD	LARP	~~GPS~~	~~MDS~~	GAME	~~GAMUT~~	~~GRAMS~~	ORCS	~~WARBLER~~
D_3	ARCS	BLAM	BEAR	~~BLOGS~~	LARD	LARP	~~GPS~~	~~MDS~~	GAME	~~GAMUT~~	~~GRAMS~~	ORCS	~~WARBLER~~
D_4	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	GPS	MDS	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
A_1	~~ARCS~~	~~BLAM~~	~~BEAR~~	BLOGS	~~LARD~~	~~LARP~~	~~GPS~~	~~MDS~~	~~GAME~~	GAMUT	GRAMS	~~ORCS~~	~~WARBLER~~
A_5	ARCS	BLAM	BEAR	~~BLOGS~~	LARD	LARP	~~GPS~~	~~MDS~~	GAME	~~GAMUT~~	~~GRAMS~~	ORCS	~~WARBLER~~
A_6	ARCS	BLAM	BEAR	~~BLOGS~~	LARD	LARP	~~GPS~~	~~MDS~~	GAME	~~GAMUT~~	~~GRAMS~~	ORCS	~~WARBLER~~
A_7	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	GPS	MDS	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~

Here's an extra table in case you make a mistake:

D_1	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
D_2	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
D_3	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
D_4	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
A_1	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
A_5	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
A_6	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
A_7	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER

(d) [1 pt] Assume that in backtracking search, we assign A_1 to be GRAMS. Enforce unary constraints, and in addition, cross out all the values eliminated by forward checking against A_1 as a result of this assignment.

D_1	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	~~GPS~~	~~MDS~~	GAME	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
D_2	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	~~GPS~~	~~MDS~~	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
D_3	ARCS	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	~~GPS~~	~~MDS~~	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
D_4	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	~~GPS~~	MDS	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
A_1	~~ARCS~~	~~BLAM~~	~~BEAR~~	BLOGS	~~LARD~~	~~LARP~~	~~GPS~~	~~MDS~~	~~GAME~~	GAMUT	GRAMS	~~ORCS~~	~~WARBLER~~
A_5	ARCS	BLAM	BEAR	~~BLOGS~~	LARD	LARP	~~GPS~~	~~MDS~~	GAME	~~GAMUT~~	~~GRAMS~~	ORCS	~~WARBLER~~
A_6	ARCS	BLAM	BEAR	~~BLOGS~~	LARD	LARP	~~GPS~~	~~MDS~~	GAME	~~GAMUT~~	~~GRAMS~~	ORCS	~~WARBLER~~
A_7	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	GPS	MDS	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~

Here's an extra table in case you make a mistake:

D_1	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
D_2	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
D_3	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
D_4	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
A_1	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
A_5	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
A_6	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER
A_7	ARCS	BLAM	BEAR	BLOGS	LARD	LARP	GPS	MDS	GAME	GAMUT	GRAMS	ORCS	WARBLER

(e) [3 pts] Now let's consider how much arc consistency can prune the domains for this problem, even when no assignments have been made yet. I.e., assume no variables have been assigned yet, enforce unary constraints first, and then enforce arc consistency by crossing out values in the table below.

D_1	~~ARCS~~	BLAM	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	~~GPS~~	~~MDS~~	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
D_2	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	LARD	~~LARP~~	~~GPS~~	~~MDS~~	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
D_3	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	~~GPS~~	~~MDS~~	~~GAME~~	~~GAMUT~~	~~GRAMS~~	ORCS	~~WARBLER~~
D_4	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	GPS	~~MDS~~	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
A_1	~~ARCS~~	~~BLAM~~	~~BEAR~~	BLOGS	~~LARD~~	~~LARP~~	~~GPS~~	~~MDS~~	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
A_5	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	LARP	~~GPS~~	~~MDS~~	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
A_6	ARCS	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	~~GPS~~	~~MDS~~	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~
A_7	~~ARCS~~	~~BLAM~~	~~BEAR~~	~~BLOGS~~	~~LARD~~	~~LARP~~	~~GPS~~	MDS	~~GAME~~	~~GAMUT~~	~~GRAMS~~	~~ORCS~~	~~WARBLER~~

The common mistake in this question was to leave a few blocks of words that students thought could not be eliminated. Probably the most common was to allow both LARD and LARP for D_2 and A_5. This is incorrect; for D_2, no assignment of A_7 is consistent with LARP, and for A_5, no assignment of D_4 is consistent with LARD.

(f) [1 pt] How many solutions to the crossword puzzle are there? Fill them (or the single solution if there is only one) in below.

1B	2L	3O	4G	S
5L	A	R	P	■
6A	R	C	S	■
7M	D	S	■	■

1	2	3	4	
5				■
6				■
7			■	■

1	2	3	4	
5				■
6				■
7			■	■

There is one solution (above)

Your friend suggests using letters as variables instead of words, thinking that sabotaging you will be funny. Starting from the top-left corner and going left-to-right then top-to-bottom, let X_1 be the first letter, X_2 be the second, X_3 the third, etc. In the very first example, $X_1 = \texttt{D}$, $X_2 = \texttt{A}$, and so on.

(g) [1 pt] What is the size of the state space for this formulation of the CSP?

26^N. There are 26 letters and N possible positions.

(h) [2 pts] Assume that in your implementation of backtracking search, you use the least constraining value heuristic. Assume that X_1 is the first variable you choose to instantiate. For the crossword puzzle used in parts (c)-(f), what letter(s) might your search assign to X_1?

We realized that this question was too vague to be answered correctly, so we gave everyone 2 points for the problem. The least constraining value heuristic, once a variable has been chosen, assigns the value that according to some metric (chosen by the implementer of the heuristic) leaves the domains of the remaining variables most open. How one eliminates values from the domains of other variables upon an assignment can impact the choice of the value as well (whether one uses arc consistency or forward checking).

We now sketch a solution to the problem assuming we use forward checking. Let $X_1, X_2, \ldots, X_5$ be the letters in the top row of the crossword and X_1, X_6, X_7, X_8 be the first column down. Upon assigning $X_1 = \texttt{G}$, the possible domains for the remaining letters are

$$X_2 \in \{\texttt{A}, \texttt{R}\}, X_3 \in \{\texttt{M}, \texttt{A}\}, X_4 \in \{\texttt{U}, \texttt{M}\}, X_5 \in \{\texttt{T}, \texttt{S}\}, X_6 \in \{\texttt{A}\}, X_7 \in \{\texttt{M}\}, X_8 \in \{\texttt{E}\}.$$

Upon assigning $X_1 = \texttt{B}$, the possible domains remaining are

$$X_2 \in \{\texttt{L}\}, X_3 \in \{\texttt{O}\}, X_4 \in \{\texttt{G}\}, X_5 \in \{\texttt{S}\}, X_6 \in \{\texttt{L}, \texttt{E}\}, X_7 \in \{\texttt{A}\}, X_8 \in \{\texttt{M}, \texttt{R}\}.$$

The remaining variables are unaffected since we are using only forward checking. Now, we see that with the assignment $X_1 = \texttt{G}$, the minimum size remaining for any domain is 1, while the sum of the sizes remaining domains is 11; for $X_1 = \texttt{B}$, the minimum size is 1, while the sum of the sizes remaining is 9. So depending on whether we use minimum domain or the sum of the sizes of the remaining domains, the correct solutions are G and B or only G, respectively.

Any choice but $X_1 = \texttt{B}$ or $X_1 = \texttt{G}$ will eliminate all values for one of the other variables after forward checking.