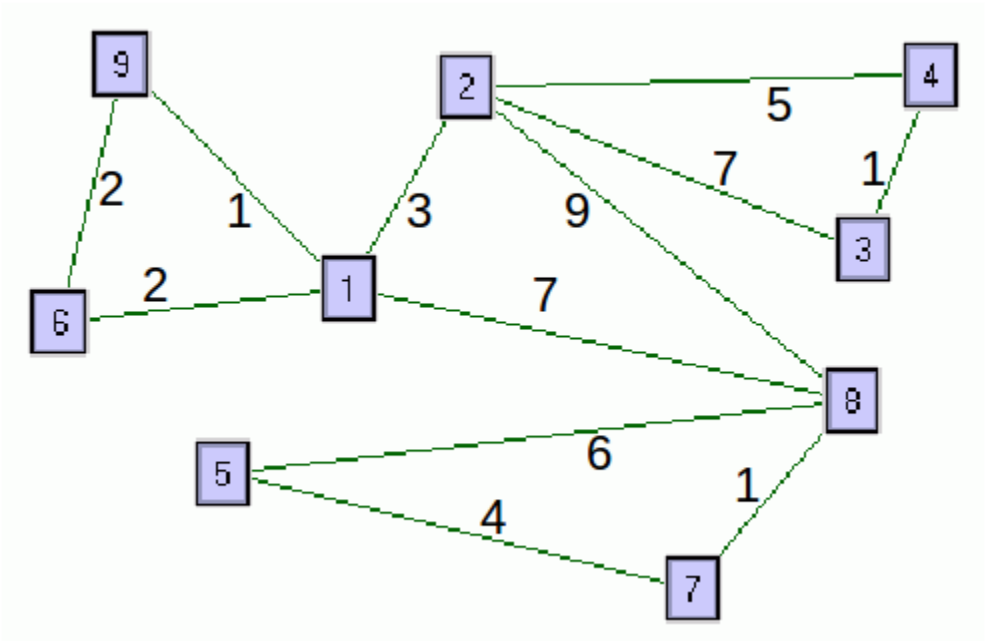
4511W, Spring-2018
ASSIGNMENT 1 :
Assigned: 02/11/18 Due: 02/18/18 at 11:55 PM (submit via moodle, you may scan or take a picture of your paper answers in a zip if you have multiple files)

Problem 1. (15 points)
Perform A* search on the following graph. At each step show (1) the fringe with the corresponding f-values and (2) the explored set. The initial state is vertex "5" and the goal is vertex "4".



Fringe:	Explored:
{N5=14}	{}
{N7=19, N8=21}	{N5}
{N8=20}	{N5, N7}
{N1=18, N2=18}	{N5, N7, N8}
(Note: I will pick N1, but you could skip this and pick N2)	
{N2=18, N9=23, N6=24}	{N5, N7, N8, N1}
{N3=22, N4=19, N9=23, N6=24}	{N5, N7, N8, N1, N2}
Pick N4, which is goal so done.	

Problem 2. (20 points)
(1) Is the heuristic in problem 1 admissible? Is it consistent?

No, on both accounts.

Node 9 has heuristic cost of 10. The shortest path from Node 9 to goal (node 4) is: 9 -> 1 -> 2 -> 4. This total distance is: 1+3+5 = 9. So this node has h-value > optimal, so the whole heuristic is not admissible.

Consistency is a stronger requirement, so it is impossible if not admissible. Even if you don't use your

answer to the first part, between nodes 9 and 2 this doesn't hold (along with many others). $h(9) \leq d(9,2) + h(2)$ ... yields ... $10 \leq 9 + 4$, which is false.

(2) Suppose you had two heuristics. The first heuristic, h_1, is admissible and consistent. The second, h_2, is admissible but **not** consistent. Prove or disprove whether a new heuristic defined as: $h_3 = \min(h_1,h_2)$, is admissible and/or consistent.

This "h_3" heuristic is admissible. We know that for all nodes "n", $h_1(n) \leq$ optimal(n). Since $h_3 = \min(h_1, h_2)$, we can conclude $h_3 \leq h_1$. Thus, $h_3(n) < h_1(n) <$ optimal(n), so $h_3(n) <$ optimal(n) for all nodes. This proves admissibility.

There are many examples of when this is inconsistent. Let us consider a very simple graph: A -> B -> C. The edge from A to B is 5, so d(A,B) = 5 and d(B,C) = 99999. Since the d(B,C) is so large, any small-ish heuristic for A and B will be trivially admissible. Then just let h_2 be smaller than h_1 for A and B, then h_3 will be just h_2, which is inconsistent. In the example, let $h_1(A) = 10$, $h_2(A) = 9$, $h_1(B) = 7$, $h_2(B) = 1$. Then $h_3(A) \leq d(A,B) + h_3(B)$ ... yields ... $9 \leq 5 + 1$, which is false.

Problem 3. (30 points)
For each of the following problems, give: (1) the relaxation, (2) a general description of how to solve it optimally and (3) how to compute the heuristic for a given state (i.e. how to get a value from a state that is not computationally intensive.)

(1) Suppose you have three kids that need to get to school and your car is broken so you have to bike. Unfortunately, you can only carry one child on the bike safely so you will have to make multiple trips. Your children are also not the best behaved. When you are not watching the children, if the oldest and middle child are together they fight. Also if you are not watching, the middle child will sit on the youngest child. The goal is to get all three children to school without them fighting/sitting on each other.

(1) This is very similar to the cannibal and missionaries problem we discussed in class. We can relax the problem by removing the conditions of the children fighting (i.e. there are no problems with leaving the children alone).
(2) The optimal solution is just to bring any child to school then bike home without a child.
(3) If there are "n" children still at home in a given state, the heuristic is then "2n-1"

(2) You are playing puzzle-loop: https://www.puzzle-loop.com/faq.php . Suppose you know some vertex that is guaranteed to have an edge leaving it.

(1) We can relax the rule that you need to have no "loose ends", or in other words, you can have disjoin line parts.
(2) The optimal solution would then be to just simply fill in a single edge around a number, prioritizing all numbers adjacent to each other (so the edge double counts).
(3) The worst case is if every edge fills up two number values (i.e. a 2x2 sqaure with all 2's could be filled with just 4 lines to fill up a summed value of 8 numeric values). So the heuristic would be the number of unfulfilled number-values divided by 2.

Problem 4. (20 points)
For each of the problems, (1) state what algorithm you think is best search from uniformed, informed and local searches. Provide a specific search algorithm and not just one of these categories. Then (2) justify your answer with a **short** paragraph (one to three sentences) why you think this is the most appropriate choice.

(1) Suppose a company hired you to make a better worms AI: https://www.youtube.com/watch?v=aczSWJ8-PK0 (you don't need to watch the whole thing, but just to get an idea of the game). The goal is to kill all enemy worms before you lose all of yours. For simplicity sake, assume you are playing 2-player only and you have some supporting code already worked out for a bot that can figure out how to position yourself. All you need to do now is figure out which weapon to use on which character. How would you find an efficient way to win against the current AI on one specific level?

A genetic algorithm would probably work best here. Your "genome" would be "1st round weapon, 1st around target, 2nd round weapon, 2nd around target, ..." for a significantly large number of genes (can overshoot without much penalty). You could choose a different try to argue a different local search algorithm, but you would have to be clear how you would frame the problem (as I think it is more difficult in the other cases, as you would need a heuristic).

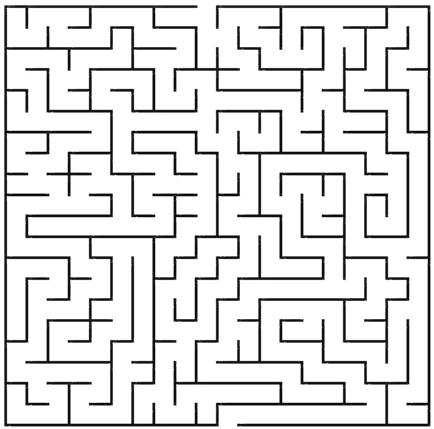
Justification: We do not need an optimal solution, just something better than the current AI. So we can use a faster approximate technique from local searches, rather than the complete & optimal searches from informed/uninformed. As mentioned above, the genetic algorithm is specifically appealing as the game will provide a "fitness" value at the end of the game (how much life and how many worms you or your opponent had at the end). Whereas coming up with a heuristic for the game would be more difficult. (Or at least a good one.)

(2) An airplane carrying plutonium crashed into the desert. You landed nearby with a bio-hazard suit and a Geiger counter. You need to find and retrieve the plutonium. (You can assume all the plutonium is together at the same spot.)

Hill-climbing search.

Justification: The Geiger counter is basically like a "heuristic" that informs you how far you are from the goal. With a physical search like this, using A* is difficult as you don't want to walk a bit from the start, record the Geiger counter rate, then go back to the start and walk a bit different direction to record another rate, etc. You will probably do more walking than just meandering while trying to keep the Geiger counter going up (this search is hill-climbing).

(3) Suppose you want to find the exit to a simple maze, such as shown below. However, the environment is only partially observable, so you can't see what are in the squares until your algorithm "explores" them. Your goal is to find the shortest path to the exit.



Breadth-first-search

Justification: You can't use many informed or local searches since you don't have a heuristic (you don't know the distance to the end since you don't know exactly where the end is). Depth-first-search is not guaranteed to find an optimal solution, so breadth-first-search is the only algorithm really left.

A side note: you might argue depth-first-search for efficiency reasons. However, mazes are special cases where there is not a very large time or memory difference between DFS and BFS. This is as both algorithms increase the fringe size when they reach an intersection and reduce the fringe size when they reach a dead-end. So both actually end up using about the same amount of memory, rather than linear vs. exponential.

(4) You are going on the road trip and are doing some very meticulous planning. You already know the route you are taking, but you want to optimize fuel costs. You have recorded where all the gas stations are located and the price at each one. You also know on a full tank, which other gas stations you can reach (assume gas use is not variable like in real life).

A* search

Justification: It is fairly easy to come up with a heuristic (just use the cheapest gas price to estimate the cost for the remaining distance). As such the informed/local search algorithms are going to perform better than the uninformed searches. We want the optimal solution, so we should probably not do a local search. A* is optimal and fast (given that our heuristic is not terrible).

Programming (python/lisp):

The book provides code for the algorithms presented. For this class, we will use the python version of the code. Download the python code here:
https://github.com/aimacode/aima-python

The code requires python3 to run (but some of the tests are written for python 2). For this assignment, you will need to know how to use the implemented genetic algorithm. Of note are:
/root/search.py
/root/tests/test_search.py

I was having trouble running test_search.py directly, but it will be useful to reference.

Problem 5. (20 points)
For this problem, compare the actual run-time of depth-first search, iterative-deepening depth-first search and genetic algorithms. Do this for the n-queens problem (this is already built into the code).

To get the run-time of code on a cse-labs machine put “time” before the program, so for example:

```
time python3 myFile.py
```

... then look for this in the output:

```
8.736u 0.000s 0:11.75 91.3%	0+0k 0+0io 0pf+0w
```

This corresponds to an 8.736 second run-time(of computation).

Note: For depth-first search, use the “depth_first_tree_search” function, not one of the other variants.

Note2: We mentioned in class that there are many variations of on how you can implement genetic algorithms. Please make sure that you are using the book's version correctly.

Answer the following:
(1) For n=11 run all three tests and report the runtime.

Solution (these numbers will vary on the machine, but the proportion between them should be about the same):
DFS = 0.032 sec
ID-DFS = 10.752 sec
Genetic = 3.252 sec (side note: fitness is about 95% of the max (52/55))

(2) For n=20 find the runtime for depth-first search and genetic algorithms (ID-DFS takes a long time...)

Solution:
DFS = 12.852 sec
Genetic = 9.129 sec (side note: fitness is about 96% of the max (182/190))

(3) For n=40 find the runtime for the genetic algorithm. (Both DFS and ID-DFS take a long time.)

Solution:
Genetic = 34.976 sec (side note: fitness is about 96% of the max (750/780))

(4) Write a short paragraph of analysis weighing the pros/cons of the algorithms in this setting.

ID-DFS uses less memory, but significantly more computation time than DFS. Although this is stll the same in big-O notation, the constant factor is significantly bigger. (Note: go run ID-DFS for n=20 for kicks if you want.) So you should really only use ID-DFS if there is not enough memory for a BFS or A* (or whatever). DFS is much faster (though it “lucks out” a bit on n=11), but is not guaranteed optimality (though for this problem, this is not an issue). The genetic algorithm by far scales the best, but it almost never reaches the exact optimal solution with the default population size and number of generations. (If you incorrectly picked the wrong f_thres value, it will be extremely fast, but very poor results). You should only really use the genetic algorithm in this case if you only need “something close” to an n-queens solution not an exact one.