SOLUTIONS AND MARKING SCHEME

Queen Mary
University of London

ECS629U-759P: ARTIFICIAL INTELLIGENCE

2019/20 – Semester 2

Brice Denoun
Dr. Arman Khouzani, Dr. Georgios Tzimiropoulos

Coursework 1 (12%): Search Problems

Winter 2020

DUE DATE: PLEASE REFER TO THE EXACT DATE FROM QMPLUS
NO LATE submission is allowed: submission link will disappear upon deadline!

Instructions: The coursework is composed of three questions. The first two involve coding exercises. The last one is a writing exercise about adversarial search. You are expected to submit two files ONLY through the QM+ submission link:

1. A single **PDF** file that must be **at most 5 pages long**. Any file submitted under the wrong format (e.g. `doc`, `docx`, `odt`, etc) or exceeding the length limit may be penalised. The PDF can include scans of handwritten material as long as it is legible but a typewritten version is encouraged (even better if you use LaTeX). **ATTENTION: Do not include the text of the question in your answers. Failure to do so may disqualify your submission!**
2. A *single compressed folder* containing all your (well documented and sufficiently commented) codes. The codes can be in any language, although `Python/java/C/julia/JS/Lisp` are preferred in that order. Your codes should run (no bugs!). There must be an easy to follow `readme.txt` file inside the folder.

It is allowed to use codes from online resources. However, this has to be clearly cited with reference in your report. Collaboration is NOT permitted when attempting the answers. High level discussion when not attempting the questions may be fine, but discouraged. A better means is using our discussion forum on QM+. There is zero tolerance policy for cases of plagiarism. If in doubt, ask! Please be aware that systems can be busy and slow to respond shortly before deadlines. So you should aim to submit at least one hour before the announced deadline.

Question 1: Let's Travel! (Uniform Cost Search)

A friend of yours gave you a map of UK, but encoded as a `json` file: `UK_cities.json`. The file encodes all the roads between major cities of the UK with their length of the road (in km). After numerous arguments, you both decided to go from London to Aberdeen. Although you would like to go there while visiting as many cities as possible, neither you nor your friend can afford the time! You are then trying to reach Aberdeen from London taking the shortest path (the total driving distance).

(a) (15 points) Following the UCS algorithm, manually execute it for three iterations for this problem. As a reminder, the pseudo-code of the UCS is provided in Figure 1, which matches Figure 3.13 in the textbook (3rd edition).

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

Figure 1: Pseudo-code for the USC (Uniform-Cost Search) algorithm on a graph. This is borrowed from [1, Fig. 3.13].

- By iteration, we are referring to the outer "loop". So, in your trace, at least three nodes should be selected for expansion.
- In particular, you should provide (only) the content of the following variables, with the following specific data structure:
  (i) the "frontier": a queue of "nodes" ordered by their path-cost
  (ii) the "explored": a set of "states"
- You should represent each "node" in the frontier as a tuple of the following information: ("state", "path-cost", "best-path-to-reach-that-state"). Note: the "state" in our problem is just the name of the city you are at! The "path-cost" is just a positive real number; and the "best-path-to-reach-that-state" is a list of states starting from the initial state to that node's state.
- For the trace, you should provide only one value per each line. Write down only the new value each time any of the above two variables change. The order definitely matters.
- If you encounter any tie-breaking situation in the executing, e.g. in adding the nodes to the frontier, go with the alphabetical (lexicographical) order based on the name of the city.
- No explanation or comments are necessary. For instance, you don't even need to tell which line/part of the pseudo-code was responsible for an update. We only check whether the trace is correct (including the order of updates).

- An example few first steps of the trace with the above description would be as follows:

```
1 frontier = [('london', 0, ['london'])]
2 explored = {}
3 frontier = []
4 explored = {'london'}
5 frontier = [('birmingham', 110,['london','birmingham'])]
6 ...
7 ...
8 ...
```

Solution: Here is a trace at the outer loop:

```
Frontier is: [[0, 'london', None]]
Frontier len is: 1
Explored nodes are: {'london'}
Explored nodes len is: 1
Current node is: london
Frontier is: [[52, 'brighton', ['london', None]],
[54, 'cambridge', ['london', None]],
[57, 'oxford', ['london', None]],
[71, 'dover', ['london', None]],
[110, 'birmingham', ['london', None]],
[116, 'bristol', ['london', None]],
[161, 'cardiff', ['london', None]],
[172, 'exeter', ['london', None]],
[172, 'hull', ['london', None]],
[198, 'leeds', ['london', None]],
[198, 'liverpool', ['london', None]],
[302, 'carlisle', ['london', None]],
[396, 'glasgow', ['london', None]]]
Frontier len is: 13
Explored nodes are: {'brighton', 'london'}
Explored nodes len is: 2
Current node is: brighton
Frontier is: [[54, 'cambridge', ['london', None]],
[57, 'oxford', ['london', None]],
[71, 'dover', ['london', None]],
[101, 'portsmouth', ['brighton', ['london', None]]],
[110, 'birmingham', ['london', None]],
[116, 'bristol', ['london', None]],
[161, 'cardiff', ['london', None]],
[172, 'exeter', ['london', None]],
[172, 'hull', ['london', None]],
[198, 'leeds', ['london', None]],
[198, 'liverpool', ['london', None]],
[230, 'nottingham', ['brighton', ['london', None]]],
[268, 'sheffield', ['brighton', ['london', None]]],
[288, 'swansea', ['brighton', ['london', None]]],
[301, 'aberystwyth', ['brighton', ['london', None]]],
[302, 'carlisle', ['london', None]],
[302, 'york', ['brighton', ['london', None]]],
```

```
[329, 'penzance', ['brighton', ['london', None]]],
[396, 'glasgow', ['london', None]]]
Frontier len is: 19
Explored nodes are: {'cambridge', 'brighton', 'london'}
Explored nodes len is: 3
Current node is: cambridge
```

Marking scheme: 15 points. You could instead provide the trace with the inner loop. Both were accepted.

(b) (10 points) Implement the UCS for this problem. Run it with the provided data. You don't need to include the code in your report (you submit it separately). Instead:

(b-1) Provide the optimal path found by the algorithm (both the path and its length);

(b-2) Make the code print the trace (in the same format as requested in part (a) but only provide the *last two* iterations (of the outer loop) in your report.

Note: you don't have to follow the provided pseudo-code in your implementation necessarily. However, if the logic of your code deviates significantly from this version, then you need to provide your pseudo-code as well.

Solution: An example implementation of the UCS is provided in `ucs.py` (find it in the same place you found this solution sheet on QM+).

It turns out that you have multiple optimal paths: `[london, cambridge, york, aberdeen]`, `[london, cambridge, york, edinburgh, aberdeen]` and `[london, cambridge, york, newcastle, edinburgh, aberdeen]`, all of which share the same total cost of 502.

The trace of the last two epochs would look something like the following:

```
Frontier is: [[387, 'edinburgh',
['york', ['cambridge', ['london', None]]]],
[396, 'glasgow', ['london', None]]]
Frontier len is: 2
Explored nodes are: {'exeter', 'manchester', 'newcastle', 'penzance',
'cambridge', 'bristol', 'swansea', 'aberystwyth', 'liverpool', 'leeds',
'brighton', 'birmingham', 'carlisle', 'cardiff', 'hull', 'sheffield',
'london', 'dover', 'york', 'edinburgh', 'portsmouth', 'nottingham',
'oxford'}
Explored nodes len is: 23
Current node is: edinburgh
Found solution for [502, 'aberdeen',
['edinburgh', ['york', ['cambridge', ['london', None]]]]]
Frontier is: [[396, 'glasgow', ['london', None]]]
Frontier len is: 1
Explored nodes are: {'exeter', 'manchester', 'glasgow', 'newcastle',
'penzance', 'cambridge', 'bristol', 'swansea', 'aberystwyth',
'liverpool', 'leeds', 'brighton', 'birmingham', 'carlisle',
'cardiff', 'hull', 'sheffield', 'london', 'dover', 'york',
'edinburgh', 'portsmouth', 'nottingham', 'oxford'}
Explored nodes len is: 24
Current node is: glasgow
```

Marking scheme: 10 points. Providing one of the optimal path was enough to get full mark for the first part of the question.

(c) (10 points) Suppose both you and your friend are environmentally aware. So you would like to find the path that has the lowest overall cost, which is *the sum of the overall driving time plus the overall cost of the air pollution due to your driving*.
Here are the details:

- You are free to choose your driving speed on each road of the path (but assume that you don't change your speed on a given road). So e.g. if you choose to drive at 40 km/h on the road between London to Birmingham, you maintain that speed on this road. But you are free to choose a different speed on the next road, etc.).
- If you drive at speed v km/h (on any road), you are responsible for an air pollution cost of $(0.00001v^2)$ units *per hour*.

Using your answer to the previous part, provide the best path in this scenario (both the optimal cost and the path). Again, you don't need to provide the code in your report, but you need to briefly explain your approach.

Solution: The overall cost of a path here is the total time plus the total air pollution cost. But this is equal to the driving times plus the air pollution cost summed over each edge of the path independently. Mathematically, if we consider a path from an initial node to a goal node to be $(e_1, e_2, \ldots, e_n)$, were e_i is the i'th node, then $\text{cost}(path)$ can be broken down to $\text{cost}(e_1) + \text{cost}(e_2) + \ldots + \text{cost}(e_n)$. Therefore, instead of having to search in the domain of the entire paths, we can firs find the optimal edge weights for each edge in the network (more accurately, for each edge that the algorithm needs), and then find the optimal path using those optimal weights.
So now consider a general edge e, that is a road between two cities. Let the original weight of that edge be denoted by l, representing the length of that road. Suppose we choose to drive on the road with velocity v. The new cost of that edge would be:

$$\text{cost}(e) = \frac{l}{v} + \alpha v^2 \times \frac{l}{v} = (\frac{1}{v} + \alpha v)l \tag{1}$$

Some of you overlooked the fact that αv^2 (where $\alpha = 0.00001 = 10^{-5}$) was given as the *per hour* cost of air pollution; so to get the air pollution cost of that edge, we need to multiply that by driving time l/v. Note that l here is not a variable, that is just the distance between those two cities at the both ends of that edge, what is a variable here is v, the driving speed, which the problems says we can choose per each edge.
There are no limits set for us (except that of course the speed cannot be negative, and here, it cannot be zero either!, so $v > 0$). So in order to find the optimal speed, we can take the derivative of the function $\frac{1}{v} + \alpha v$ with respect to v and set it equal to zero:

$$\frac{d}{dv}\left(v^{-1} + \alpha v\right) = -v^{-2} + \alpha = 0$$

The valid solution to this is $v^* = \sqrt{1/\alpha}$, which if you replace for $\alpha = 10^{-5}$, gives you $\sqrt{10^5} \approx 316.2$ km/h. Technically, to be sure that what we have found is indeed a minimiser value (and not a maximiser), we should also check the second derivative to make sure it is positive (so the function around the zero-derivative point looks

like a cup, not an upside-down cup!). The second derivative is v^{-3}, which is always positive for any $v > 0$, including our v^*. So, what is the optimal weight of that edge? All we need to do is to plug in v^* in (1):

$$\text{optimal cost}(e) = (\frac{1}{v^*} + \alpha v^*)l = (2\sqrt{\alpha})l \approx (0.006325)l$$

At this point, you can just re-run the UCS with the difference that instead of the previous weights of each edge, you scale it with 0.006325. However, if you are observant, you can notice that an interesting thing has just happened(!): the optimal weight of each edge has became just a constant scaling of l, which was the previous weight of that edge. So, we in fact don't even need to re-run the UCS as we already know what the optimal solution is! The optimal path will be just as before! In fact, we also know what the overall cost is going to be: it will be just a re-scaling of the previous overall cost, it will be $0.006325 \times 502 \approx 3.175$ units.

Fun-fact/disclaimer(!): you should of course just follow the assumptions of the question, which is no speed limit implied on the roads. That said, here the optimal speed of 316.2 km/h on each road is obviously impractical. This is because of a typo (mi scuzi!): the α was supposed to be 0.0001 as opposed to 0.00001, which gives an optimal speed of 100 km/h, a more realistic/practicable value!

Marking scheme: 10 points. If you don't argue why the optimal path does not change from before, if everything else is perfect, you will get 7/10. (Note: it is not trivial that this would be the case: since the weight of each edge is changing, then the optimal path could have been different! It just happened (fortuitously!) in this specific problem that the new weights were just a constant scaling of the previous weights, with the same scaling for all edges.) If you don't make the observation that you don't have to re-run the UCS but everything else is perfect, you get 9/10. If you omit the multiplication by time in computation of the air pollution, but everything else is perfect, you get 6/10. If you don't do any analysis and just use a brute-force to find the best speed per each link, but everything else is perfect, you get 7/10. If you don't find the best speed per each link, but rather choose an arbitrary or random value, then mark is capped at 4.

(d) (10 points (bonus)) Now, for a bonus 10 points!, suppose you have a super-car and you are environmentally ignorant! Suppose:

- As in the previous part, you can choose your driving speed per each road.
- However: suppose each road now has a speed-limit. For simplicity, assume that the value of the speed limit is the same as the value provided for the length of that road in the `json` file.
- On each road, you can even choose to go over the speed limit! However, there is a likelihood that you get caught on a speed-camera and get a fine of 1000 units. In particular, if you drive on a road with speed limit of v_{lim} at speed v, then the likelihood (i.e., probability) of getting the fine is as follows:

$$\begin{cases} 1 - e^{-(v - v_{\text{lim}})}, & \text{if } v > v_{\text{lim}} \\ 0, & \text{if } v \leq v_{\text{lim}} \end{cases}$$

- You may only be fined at most once on a given road, but fines on different roads accumulate.

Again, using your UCS code, find the best path in this problem, assuming the overall cost is the total travel time in hours plus your total fines (if any).

Solution: As in part c, the overall cost of a path can be broken down as the sum of costs per each edge that constitutes that path. So again, we can find the optimal speed per each edge, and then apply our UCS algorithm with the new weights to find the optimal path (or get lucky again, and somehow find the optimal path without having to run UCS again).

The cost of an edge with length l when we drive at speed v on that road (where $0 < v \leq 300$ km/h), would be:

$$\text{optimal cost}(e) = 100 \times \frac{1}{v} + 1000 \times \begin{cases} 1 - e^{-(v - v_{\text{lim}})}, & \text{if } v > v_{\text{lim}} \\ 0, & \text{if } v \leq v_{\text{lim}} \end{cases}$$

This is because $100 \times (l/v)$ is the rental fee of the car pertaining to that road, and the second part is the *expected value* of the fine we would receive on that road. Note that whether or not we receive a fine on a road (if we drive above speed limit) is a random variable. In order to be able to meaningfully compare different options, we have to convert it to a non-random value. The most obvious candidate for this is to use the expected value (the average value if we would have to repeat this scenario many times). The expected value of the fine, is just 1000 (the ticket value) times the probability that we get a ticket.

This is also a function of just v, and we should just find a speed that minimises this per each road. First, we can make the observation that there is no point in driving below the speed limit on the road (unless of course the speed limit is above 300 km/h, which is the maximum speed of our super-car). This is because up to the speed limit, we will be reducing the first part (the rental fee of the car) without adding any expected fine (which is zero). So the only question is, will it be worth it to go above the speed limit, as it will reduce the first part, but starts to increase the second part.

Now either analytically, or through brute-force search, or by using any numerical optimisation library like `scipy.optimize`, you can run the USC with the weight of each edge being passed as the result of this single-variable optimisation (and that would have scored you a perfect mark if you indeed get the correct answer, which is 300 units).

However, with a little bit of analysis, you can also show that with the given parameters, it is never worth going over the speed limit (the reduction in the rental fee never outweighs the expected fine for any road). So for the given parameters of the problem, it is always optimal to just stick with the speed limit. Now, because we are told to assume the speed limit is the same as the distance, this means the cost per each edge is just 100×1, i.e., 100. So the optimal solution becomes any path that can get us from London to Aberdeen with taking as few roads as possible. The lowest number of roads is three, which gets you the overall cost of 300.

Marking scheme: 10 points. The perfect score is reserved for a combination of the correct answer and correct rich explanation and/or code. Partial scores possible depending on the quality of the answer and/or code. If instead of using the expected value, you are generating the fines randomly (according to the probabilities), then your score is capped at 4 (because such analyses would not make much sense!).

Question 2: It's on the tip of my tongue! (Genetic Algorithm)

Becoming old is certainly difficult. And one of the signs of growing older is some form of amnesia. As a matter of fact, I cannot exactly remember my two favourite passwords, but still, I have a hazy idea about what it looked like!

Your goal is to help me remember these two passwords using a genetic algorithm: It turns our even though I am amnesiac, I can still tell how close to my password a candidate is. Let me give you the few details I remember with certainty: the password was exactly **10 characters long** and could only contain upper-case English letters, numbers between 0 and 9, and the underscore symbol _. For instance, `1_PASSWORD` and `MYPA55W0RD` are legitimate candidates.

So here is the deal: For each password you show me, I am going to give a score between 0.0 and 1.0 indicating how close to my forgotten password I feel it is (where 1.0 means that the password is exactly correct) and despite my memory, I have a trustworthy intuition!

To be more precise, you have access to a function called `blurry_memory` as follows:

input: a list of candidate passwords, e.g. ["`____TEST____`", "`IS_I7_L0V3`"], along with your own student number (e.g 190123123), and the index of the password you are trying to find: 0 or 1 (recall: there are two passwords to recover).

output an ordered dictionary containing the passwords as keys and the associated scores as values (e.g "`____TEST____`": 0.045, "`IS_I7_L0V3`": 0.247)

To access this function, you can add the following line to the header of your python3 code:

```
from amnesiac import blurry_memory
```

Implement a Genetic Algorithm to recover the two forgotten passwords.

What should you provide in your report:

(a) (10 points) What did you get for the two passwords? (recall: you should use your own student ID number whenever invoking `blurry_memory`).

Solution: A vanilla version of what has been asked can be found in `algo_gen.py`. You can run it by changing the student number.
Here is the table containing the passwords of each student:

Student	Password1	Password2
190592769	M1N_I_M4X_	_Y0UM4DBRO
170448130	B3L1K3B1LL	F1ND_NE55Y
170479363	WE1L_D0N3_	F1ND_NE55Y
170928388	6RUMPY_C47	_C0NGR47S5
170368517	Y0UDON754Y	9_FR33D0M8
150049542	B3S7_M0DUL	1_H4T3_DF5
181122056	1_L1K3TH4T	J0HN__5NOW
190084873	_4_3VER4I_	9_FR33D0M8
150047755	F1ND_NE55Y	6RUMPY_C47
170343949	M1N_I_M4X_	F1ND_NE55Y
190583055	1_H4T3_DF5	F1ND_NE55Y
190028305	1_L1K3TH4T	WE1L_D0N3_
190517522	CHUKN088IS	1_H4T3_DF5
160205075	J0HN__5NOW	D0GW0OWFUN
190734613	_4_3VER4I_	CHUKN088IS
160572438	B3L1K3B1LL	_C0NGR47S5

160486423	9_FR33D0M8	CHUKN088IS
170434328	_C0NGR47S5	6RUMPY_C47
170247197	CHUKN088IS	J0HN__5NOW
190262046	1_H4T3_DF5	WE1L_D0N3_
190778145	1_L1K3TH4T	9_FR33D0M8
161133092	M1N_I_M4X_	J0HN__5NOW
170049926	K4NG_0UF__	WE1L_D0N3_
160580134	M1N_I_M4X_	B3L1K3B1LL
170709031	_Y0UM4DBRO	F1ND_NE55Y
170980700	M1N_I_M4X_	Y0UDON754Y
170172202	_C0NGR47S5	1_H4T3_DF5
190572587	1_H4T3_DF5	7ROLL_F4CE
170916396	7ROLL_F4CE	1_H4T3_DF5
170074157	9_FR33D0M8	B3L1K3B1LL
170205533	1_H4T3_DF5	7ROLL_F4CE
161125684	B3L1K3B1LL	9_FR33D0M8
170639668	_Y0UM4DBRO	9_FR33D0M8
190536501	6RUMPY_C47	M1N_I_M4X_
160365449	7ROLL_F4CE	Y0UDON754Y
190347321	Y0UDON754Y	J0HN__5NOW
190203706	6RUMPY_C47	F1ND_NE55Y
160794683	9_FR33D0M8	1_H4T3_DF5
170407562	K4NG_0UF__	F1ND_NE55Y
170219839	_9CE_TW1CK	1_L1K3TH4T
190759775	D0GW0OWFUN	M1N_I_M4X_
170488387	F1ND_NE55Y	B3L1K3B1LL
170954310	7ROLL_F4CE	_Y0UM4DBRO
130785761	_C0NGR47S5	9_FR33D0M8
170180425	_Y0UM4DBRO	M1N_I_M4X_
170506315	_4_3VER4I_	B3L1K3B1LL
130052685	9_FR33D0M8	_4_3VER4I_
170732110	K4NG_0UF__	_9CE_TW1CK
170984786	_4_3VER4I_	M1N_I_M4X_
190766164	_4_3VER4I_	WE1L_D0N3_
170496854	R1DD7E4U__	B3S7_M0DUL
190766681	7ROLL_F4CE	B3L1K3B1LL
170238298	D0GW0OWFUN	_9CE_TW1CK
170213468	1_H4T3_DF5	_4_3VER4I_
170434362	6RUMPY_C47	9_FR33D0M8
190649694	D0GW0OWFUN	B3S7_M0DUL
160554847	CHUKN088IS	F1ND_NE55Y
190789473	D0GW0OWFUN	_9CE_TW1CK
160557413	R1DD7E4U__	1_H4T3_DF5
160392294	_C0NGR47S5	_4_3VER4I_
161107303	7ROLL_F4CE	1_L1K3TH4T
160602218	WE1L_D0N3_	_C0NGR47S5
190677099	CHUKN088IS	Y0UDON754Y
170946670	1_L1K3TH4T	D0GW0OWFUN
150073455	B3L1K3B1LL	M1N_I_M4X_
190172273	D0GW0OWFUN	K4NG_0UF__
160436082	CHUKN088IS	_Y0UM4DBRO

161051763	_4_3VER4I_	_Y0UM4DBRO
170367093	9_FR33D0M8	J0HN__5NOW
190838391	WE1L_D0N3_	7ROLL_F4CE
190188920	9_FR33D0M8	B3L1K3B1LL
140368985	CHUKN088IS	M1N_I_M4X_
190635387	M1N_I_M4X_	_Y0UM4DBRO
170346364	K4NG_0UF__	1_H4T3_DF5
170412669	_4_3VER4I_	7ROLL_F4CE
170465045	CHUKN088IS	R1DD7E4U__
170788737	B3S7_M0DUL	WE1L_D0N3_
170459778	J0HN__5NOW	Y0UDON754Y
190649476	WE1L_D0N3_	F1ND_NE55Y
180273942	CHUKN088IS	_9CE_TW1CK
190760070	_Y0UM4DBRO	J0HN__5NOW
170279361	9_FR33D0M8	R1DD7E4U__
170216584	D0GW0OWFUN	9_FR33D0M8
190561673	F1ND_NE55Y	1_L1K3TH4T
140869770	J0HN__5NOW	_9CE_TW1CK
170467212	7ROLL_F4CE	B3S7_M0DUL
170276290	_C0NGR47S5	M1N_I_M4X_
170111887	J0HN__5NOW	CHUKN088IS
150442640	R1DD7E4U__	F1ND_NE55Y
160497430	D0GW0OWFUN	6RUMPY_C47
190735252	K4NG_0UF__	B3S7_M0DUL
170146926	WE1L_D0N3_	6RUMPY_C47
170224279	K4NG_0UF__	1_L1K3TH4T
170404505	B3S7_M0DUL	9_FR33D0M8
160446874	1_L1K3TH4T	6RUMPY_C47
170294685	B3L1K3B1LL	F1ND_NE55Y
150411934	9_FR33D0M8	B3L1K3B1LL
190583712	7ROLL_F4CE	_C0NGR47S5
170281456	B3L1K3B1LL	B3S7_M0DUL
160951714	6RUMPY_C47	CHUKN088IS
190758309	J0HN__5NOW	D0GW0OWFUN
190573735	CHUKN088IS	1_L1K3TH4T
160166824	M1N_I_M4X_	7ROLL_F4CE
150444208	F1ND_NE55Y	WE1L_D0N3_
190189237	R1DD7E4U__	7ROLL_F4CE
170434889	_C0NGR47S5	_9CE_TW1CK
170285753	7ROLL_F4CE	D0GW0OWFUN
170992828	CHUKN088IS	B3S7_M0DUL
190620349	6RUMPY_C47	9_FR33D0M8
170906559	D0GW0OWFUN	B3S7_M0DUL
170366144	M1N_I_M4X_	_Y0UM4DBRO
170182337	1_H4T3_DF5	D0GW0OWFUN
170226527	9_FR33D0M8	J0HN__5NOW
170747723	J0HN__5NOW	_4_3VER4I_
170442529	K4NG_0UF__	CHUKN088IS
170219976	9_FR33D0M8	M1N_I_M4X_
170253006	Y0UDON754Y	CHUKN088IS
181004877	9_FR33D0M8	WE1L_D0N3_

```
| 190602194  | R1DD7E4U__ | B3L1K3B1LL |
| 190705107  | 6RUMPY_C47 | K4NG_0UF__ |
| 170364117  | _Y0UM4DBRO | 1_L1K3TH4T |
| 170205658  | K4NG_0UF__ | _Y0UM4DBRO |
| 190038751  | _C0NGR47S5 | WE1L_D0N3_ |
| 190385633  | CHUKN088IS | _9CE_TW1CK |
| 181044334  | Y0UDON754Y | WE1L_D0N3_ |
| 190620132  | 9_FR33D0M8 | F1ND_NE55Y |
| 170252238  | 6RUMPY_C47 | J0HN__5NOW |
| 151006445  | 7ROLL_F4CE | 1_H4T3_DF5 |
| 190875374  | 1_H4T3_DF5 | 9_FR33D0M8 |
| 170793199  | R1DD7E4U__ | _4_3VER4I_ |
| 170904304  | CHUKN088IS | 6RUMPY_C47 |
| 190227953  | 1_L1K3TH4T | D0GW0OWFUN |
| 161137953  | _4_3VER4I_ | K4NG_0UF__ |
| 190707330  | J0HN__5NOW | F1ND_NE55Y |
| 170469881  | _4_3VER4I_ | _Y0UM4DBRO |
| 160426234  | Y0UDON754Y | R1DD7E4U__ |
| 190005502  | K4NG_0UF__ | F1ND_NE55Y |
```

Marking scheme: 5pt if password is correct. 4pt if student only submitted 1 correct password, (5 for correct password 4 for the other one).âĂć2-3pt if password is incorrect (non converging algorithm) depending on closeness of password. âĂć1pt if password does not converge and looks random (less than 0.50 fitness). 0pt if None

---

(b) (5 points) Explain *briefly* how specifically you chose to do the state encoding, selection, crossover, and mutation.

Solution: You don't have a unique solution since it depends on how you have implemented it. Here is what was expected:

- Initialisation/encoding: Explain clearly how the population was created and which format was used to create population. Some possible answers for this stage are: Representing population as a list of strings, list of tuples (string + fitness), custom class, etc. Strings are created randomly or from a sample of pre-defined passwords.
- Selection: Given an existing population, explain how the blurry_memory function has been run for every single specimen to determine individual fitness. Explain the method to select a number of specimens that will create a new, superior population. Some possible answers for this stage are: Roulette wheel method (probabilistic chance of passing genes over, with more chance assigned to fitter specimens), elitism method (the n-best specimens pass their genes and the other specimens are forgotten), tournament method (create tournaments of n specimens where the fittest wins and passes their genes).
- Crossover: Explain how Crossover has been performed after selecting fittest individuals. Possible answers for crossover methods include: Single-point crossover, n-point crossover, uniform crossover

- Mutation: Explain how the generation have been mutated. Answer for mutation stage: Given a x% mutation chance, each specimen may be mutated. Mutation method may include changing a single character in the specimen, randomising whole specimen, change a few characters, etc.

Novel ideas for any of these stages are allowed and not penalised, as long as you have explained it in the report.

Marking scheme: 5 points: 1pt for correctly explaining representation/encoding, 1pt for correctly explaining selection stage, 1pt for correctly explaining crossover, 1pt for correctly explaining mutation, 1pt for correctly explaining when mutation is triggered (chance of mutation) and/or if you show a good understanding of election/mutation/crossover.

(c) (10 points) Report on the number of reproductions you had to do to converge to the password? Ideally, run your code multiple times and report on the average and variance instead of a single run (since Genetic Algorithm involves randomisation).

Solution: Of course this question can be answered in a lot of different ways. What was expected it for you to explain how many times you ran it, give the average number of iterations to converge to each password. A possible answer for this stage may be:After running the genetic algorithm code N times, I found that the correct password is found after an average of X iterations with a standard deviation of Y. The results can be visualised on graph or table Y

Marking scheme: Result (5pt) - Explanation/times (3pt) - avg/var given (1.5pt) -> avg (1pt) + var (0.5pt) - Code or/and table (0.5pt)

(d) (10 points) Explore the effect of at least one hyper-parameter on the performance of your algorithm. This could for instance be the choice of the mutation rate, or the selection/pairing scheme, the population number, etc. Ideally, provide a table or a figure with a very concise discussion.

Solution:

Once again, it exists a lot of different answers. The point is for you to experiment by changing the value of some hyper parameter and showcase your findings. Findings don't need to be positive, just showcase that the workings/dynamics of the GA change with the hyper-parameters. Possible parameters to change include mutation chance, cross chance, amount of characters mutated, population number, number of iterations, crossover/mutation/selection method, etc. A possible answer for this stage may be: I decided to experiment with varying the mutation chance of my algorithm. I found out that if the mutation chance was too low, password would not always converge to a correct password, but if it was too high, the amount of iterations needed to create valid passwords is higher. My optimal mutation chance is X%. You must show understanding of the importance of hyper-parameter changed and correctly represent results using a graph or table.

Marking scheme: Explanation quality (8pt) - (4pt if password not found) - Table/-Fig (2pt)

Question 3: Let's play a game! (Adversarial search) .....

You are now engaged in a thrilling two-player adversarial game which is represented by the following tree. Suppose you are the MAX player (who goes first).

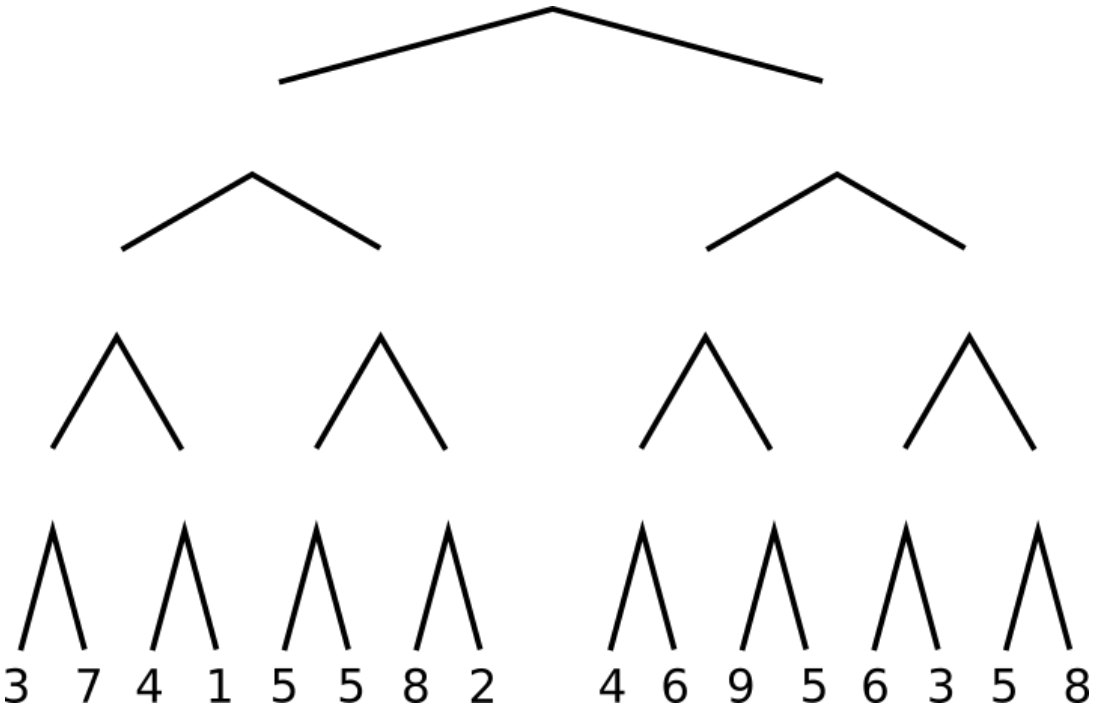

Figure 2: Tree of the game

(a) (10 points) **Play optimally (MINIMAX algorithm)**

Knowing that you start (top of the tree is you), complete the blank nodes in the provided tree using the MINIMAX approach.

Solution: Using the MINIMAX algorithm, the obtained tree is represented in Figure 3.

Marking scheme: The more correct nodes the more points

(b) (10 points) **Play optimally, quicker thinking! (α-β pruning)**

Suppose there is a mean arbiter that doesn't like too much hesitation at the beginning. Perform α-β pruning on the tree you filled, searching from left to right. Which branches are pruned and why?

Solution: The red branches of Figure 4 correspond to the pruned one. You also have the values of alpha and beta for each node.

Marking scheme: 7 points on pruning and 3 points on explanation. Each branch wrongly pruned results in 1 point subtracted.

(c) (10 points) Suppose the entry fee to play the game is x units (for either players).

c-(1) What are the ranges of values for x that will be still worth playing the game? (to enter the game at all).

c-(2) For a fixed value of x, do you prefer to be the first player (MAXimiser) or the second player (MINimiser)? Very briefly explain your answer.

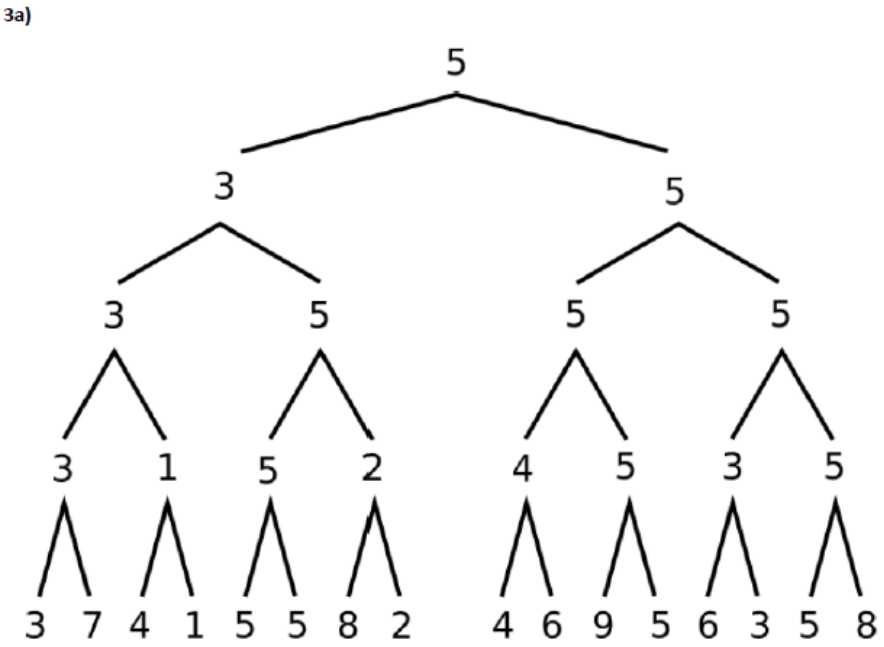

Figure 3: Filled tree of the game

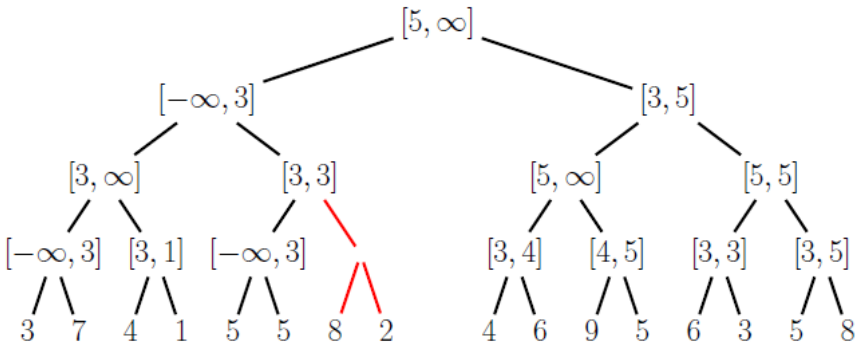

Figure 4: Result of α-β pruning

Solution: 1) The output of MINIMAX allows to know the sequence of actions that are going to be performed if the other player is playing optimally. We also know that this will end up by the MAX player winning 5 units. So we are looking for all values x such as 5 - x >= 0 which means the acceptable range of values is if the entry fee is between 0 and 5.
2) We just showed that if playing optimally it would be better to be MAX player for all entry fee between 0 and 5 since the gain is positive. In a similar way if you play as the MIN player your gain will be x - 5, which means that if the gain is bigger than 5, it's better to play as MIN.

References

[1] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2009.