A2

Part I must be done individually, remember to show your work for all quantitative questions. Part II may be done with a pair programmer or individually.

Part I – Questions (20 points each unless noted otherwise)

1. Prove that the Manhattan distance is consistent with any constant transition cost $k \geq 1$.

   Consistency states: $h(n) \leq cost(n, action, n') + h(n')$.

   By assumption, cost of any legal move is k, therefore, we can rewrite the consistency equation as: $h(n) \leq k + h(n')$.

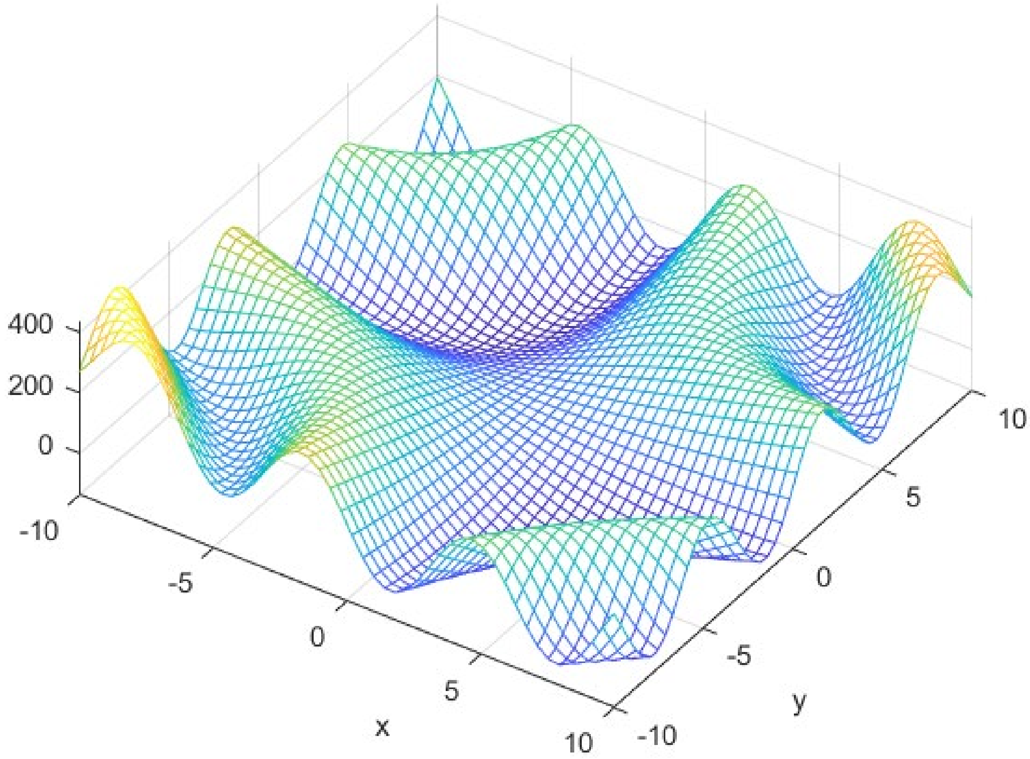
   When we make a move, and are using the Manhattan distance as a heuristic, exactly one tile will have moved by either one row or column position. If it is closer to the goal in n', then h(n') =h(n) -1, otherwise it is farther from the goal and h(n') = h(n)+1.

   Thus $h(n') = h(n) \pm 1$, and substituting for h(n'), we have:
   $$h(n) \leq k + h(n) \pm 1.$$

   Subtracting h(n) from both sides, we have:
   $$0 \leq k \pm 1.$$

   Both $0 \leq k - 1$ and $0 \leq k + 1$ must be true for consistency to hold. As $k \geq 1$, both cases will hold for all k regardless of the node n and its successor n'. (We could show this formally by induction on k, but that is not necessary for this question.) Consequently, the heuristic is admissible.

2. An agent is navigating the following fictional landscape on planet Kepler 22-b that corresponds to the function: $f(x, y) = 150sin(xy/10) - 2y - 3x + xy + x^2 + y^2$:



The agent is looking for the lowest point in the landscape (goal: find potential surface water). a) What is the gradient function? b) Given position (-.6, 7), what would the new position be after one update if the update step size is 0.01? c) Verify that this decreases the objective function by computing the objective function at the original and new locations.

Reminder: $\frac{d}{dx} sin(kxy) = ky\ cos(kyx)$

Let us compute the gradient: $\begin{bmatrix} \frac{d}{dx} f(x,y) \\ \frac{d}{dy} f(x,y) \end{bmatrix}$

$$= \begin{bmatrix} \frac{d}{dx} 150sin(xy/10) - 2y - 3x + xy + x^2 + y^2 \\ \frac{d}{dy} 150sin(xy/10) - 2y - 3x + xy + x^2 + y^2 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{150}{10} y\ cos(xy/10) - 3 + y + 2x^1 \\ \frac{150}{10} x\ cos(xy/10) - 2 + x + 2y^1 \end{bmatrix}$$

$$= \begin{bmatrix} 15y\ cos(xy/10) - 3 + y + 2x \\ 15x\ cos(xy/10) - 2 + x + 2y \end{bmatrix}$$

If step = 0.1, then

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}x\\y\end{bmatrix} - \mathit{step}\begin{bmatrix}\frac{d}{dx}f(x,y)\\ \frac{d}{dy}f(x,y)\end{bmatrix} = \begin{bmatrix}x\\y\end{bmatrix} - \mathit{step}\begin{bmatrix}15y\cos(xy/10)-3+y+2x\\ 15x\cos(xy/10)-2+x+2y\end{bmatrix}$$

note that we subtracted the gradient as we want to minimize the objective function.

With concrete numbers, this becomes:

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}-.6\\7\end{bmatrix} - 0.01\begin{bmatrix}15\cdot 7\cos(-0.6\cdot 7/10)-3+7+2\cdot -0.6\\ 15\cdot(-0.6)\cos(-0.6\cdot 7/10)-2-0.6+2\cdot 7\end{bmatrix}$$

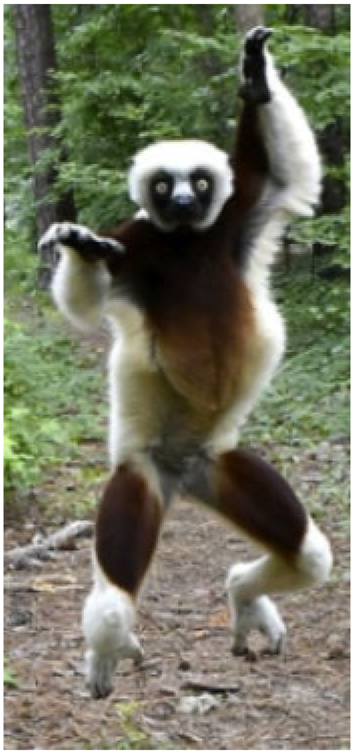
$$=\begin{bmatrix}-.6\\7\end{bmatrix} - 0.01\begin{bmatrix}98.6743\\3.1822\end{bmatrix}=\begin{bmatrix}-1.59\\6.97\end{bmatrix}$$

Objective function: $f\begin{pmatrix}-.6\\7\end{pmatrix} = -31.8041$

After one step, $f\begin{pmatrix}-1.59\\6.97\end{pmatrix} = -112.74$, indicating that we have moved to a better state.

3. A zoologist is managing three populations of critically endangered Coquerel's sifaka (*Propithecus coquereli*, image credit Duke Lemur Center). The sifaka have a natural diet of: leaves, flowers, fruit, bark, and dead wood. The zoologist is currently feeding the populations accordingly:

	Units of food provided				
population	leaves	flowers	fruit	bark	dead wood
A	4	4	5	7	5
B	5	6	5	3	4
C	5	5	5	5	4



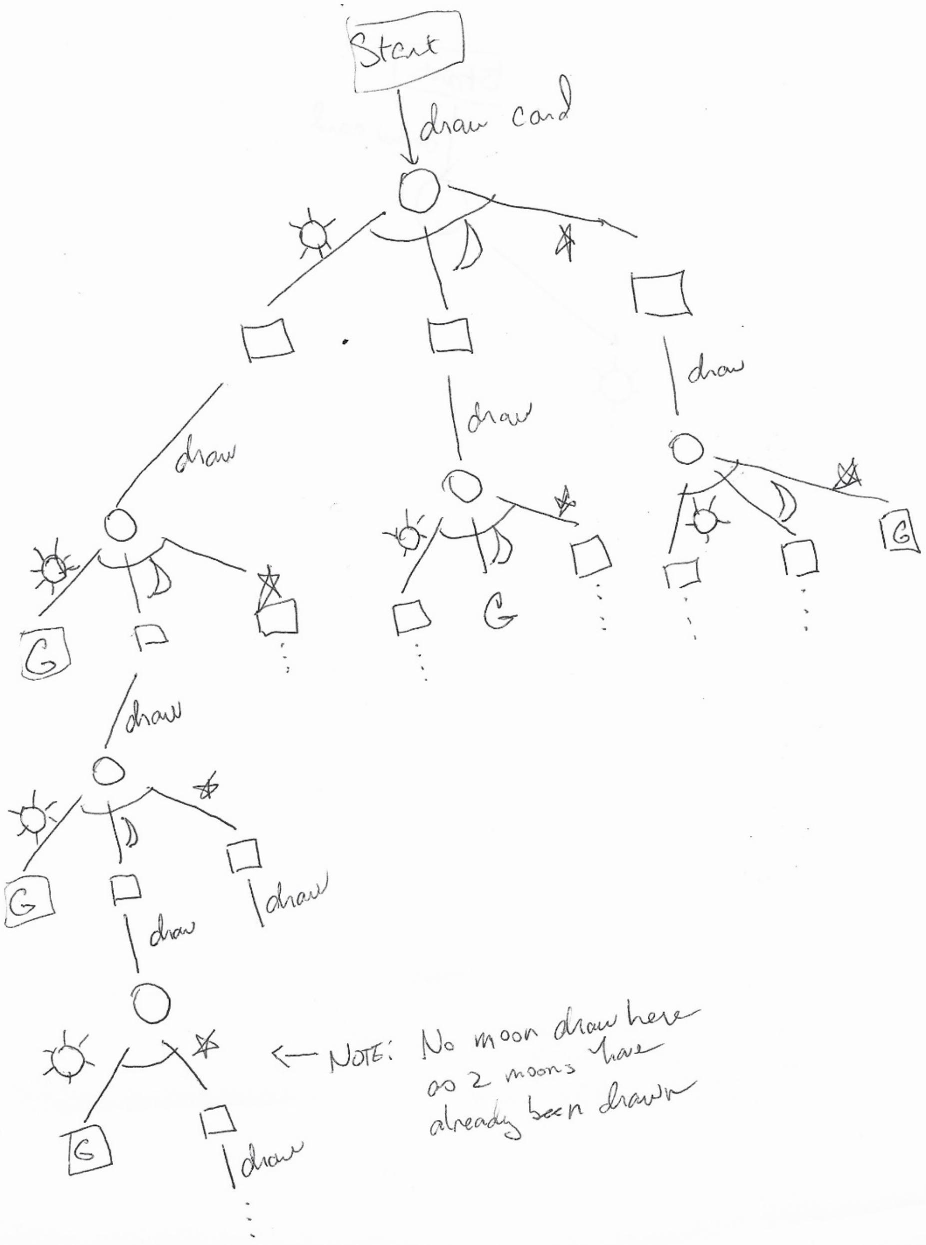
The lemurs' fitness is defined as the squared deviation from their optimal weight. Treating the diet as a list, e.g. [4,4,5,7,5] for population A, write Python-like pseudocode functions for *mutation(diet)* and *crossover(diet1, diet2)*.

(Note that while sifaka really are endangered and this is what they eat during part of the year, this simplified mockup does not take into account things that a real world modeling problem would need to consider such as nutritional value, etc. Otherwise said, if you find yourself managing a population of lemurs 😉, do not do this.)

Solutions may vary. Sample solution:

```
import random

def crossover(diet1, diet2, debug=True):
    """
    crossover - Evolutionary programming crossover between two states
    :param diet1:  List of what is being fed to population i
    :param diet2: List of what is being fed to population j
    :return: New list with diet comprised of components of i and j

    Example:  crossover([0,1,2,3,4], [5,6,7,8,9])
                  Will use diet1 [2, 4], diet2 [0, 1, 3]
              returns [5, 6, 2, 8, 4]
    """

    # In class, we suggested picking a random split point and taking
    # some diet1 and some from diet2:
    # e.g. diet1 = [7, 8, 5, 2, 3]
    #      diet2 = [6, 4, 2, 1, 2]
    # We could have something along the lines of diet1[0:split] followed by
    # diet2[split:].  This is fine if you implemented this.

    # Here we show a different type of mutation where we select positions
    # randomly to crossover

    # Create random order for indices, e.g. 3, 0, 1, 2, 4
    N = len(diet1)
    order = [i for i in range(N)]  # 0, 1, 2, ..., N-1
    random.shuffle(order)  # randomize, e.g. 2, 7, N-1, ... 3

    # Pick number to crossover
    crossN = random.randint(1, N-1)
    cross = order[0:crossN]  # Cross these indices
    if debug:
        cross.sort()  # easier for humans
        rest = order[crossN:]  # Not needed,
        print(f'Will use diet1 {rest}, diet2 {cross}')

    # To make this easy, we start with a copy of diet1 and bring in diet2
    newdiet = diet1.copy()
    for d2_idx in cross:
        newdiet[d2_idx] = diet2[d2_idx]  # copy in selected diet2

    return newdiet

def mutate(diet):
    """
    mutate - Mutate an attribute randomly, returns mutated atttribute list
    """
    # Randomly select an attribute to mutate
    attridx = generate random index between 0 and len(diet)-1
    # Determine how much to mutate
    delta = pick a random number between [-1 and 1]

    newdiet = diet.copy()
    # Mutate, but don't ever let a diet element go below 0
    newdiet[attridx] = max(newdiet[attridx] = delta, 0)

    return newdiet
```

4. Consider the fairly simple problem of selecting 2 matching cards from a deck of cards with pictures on them. The pictures consist of moon, sun, and stars, and there are two instances of each card (6 cards in total). Legal moves are pick a card that has not been picked, and the goal state is matching the first card drawn. Sketch an and-or search tree for this problem. You do not need to draw the full tree, just enough to make it clear that you understand what the full tree would look like.



← NOTE: No moon draw here as 2 moons have already been drawn