UNIVERSITY OF BIRMINGHAM

School of Computer Science

Artificial Intelligence 1

Main Summer Examinations 2023

Time allowed: 2 hours

[Answer all questions]

Question 1 Clustering

Suppose we have five observations, $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}$ and $\mathbf{x}^{(5)}$, for which we compute the following dissimilarity matrix

$$D = \begin{array}{c} \\ \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \mathbf{x}^{(3)} \\ \mathbf{x}^{(4)} \\ \mathbf{x}^{(5)} \end{array} \begin{array}{c} \begin{array}{ccccc} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \mathbf{x}^{(3)} & \mathbf{x}^{(4)} & \mathbf{x}^{(5)} \end{array} \\ \left(\begin{array}{ccccc} 0 & 0.3 & 0.6 & 0.7 & 0.8 \\ & 0 & 0.1 & 0.2 & 0.4 \\ & & 0 & 0.45 & 0.25 \\ & & & 0 & 0.9 \\ & & & & 0 \end{array}\right) \end{array},$$

(a) Based on the given dissimilarity matrix, hierarchically cluster the observations using **complete linkage**. Sketch the dendrogram, clearly illustrating the height at which each cluster fusion occurs. **[10 marks]**

Shay

1) a)

	x_1	x_2	x_3	x_4	x_5
x_1	0	0.3	0.6	0.7	0.8
x_2		0	0.1	0.2	0.4
x_3			0	0.45	0.25
x_4				0	0.9
x_5					0

x_2, x_3 @ 0.1

$\downarrow$

x_2x_3

	x_1	x_2x_3	x_4	x_5
x_1	0	0.6	0.7	0.8
x_2x_3		0	0.45	0.4
x_4			0	0.9
x_5				0

x_2x_3, x_5

$\downarrow$

$x_2x_3x_5$ @ 0.4

	x_1	$x_2x_3x_5$	x_4
x_1	0	0.8	0.7
$x_2x_3x_5$		0	0.9
x_4		0.9	0

x_1, x_4

$\downarrow$

x_1x_4 @ 0.7

	x_1x_4	$x_2x_3x_5$
x_1x_4	0	0.9
$x_2x_3x_5$		0

$x_1x_4, x_2x_3x_5$

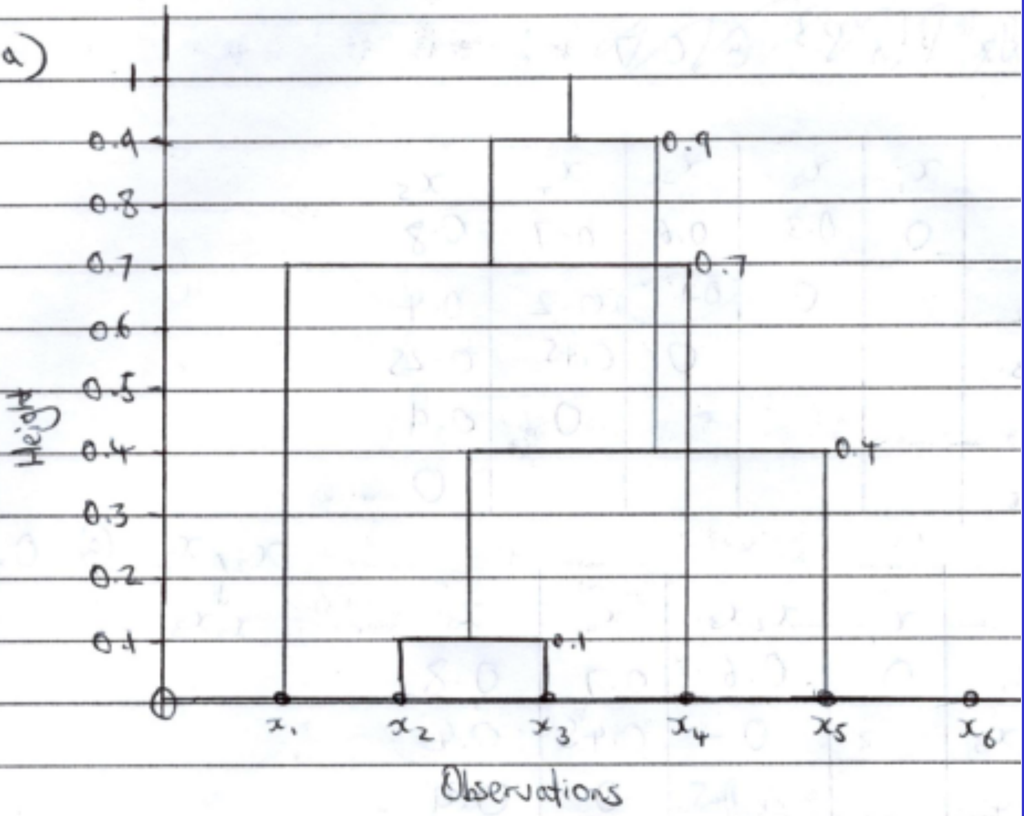
$\downarrow$

$x_1x_2x_3x_4x_5$ @ 0.9

Shay

1) a)

contined



(b) Assume that a clustering algorithm produces the following two clusters: $C_1 = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)})$ and $C_2 = (\mathbf{x}^{(3)}, \mathbf{x}^{(4)}, \mathbf{x}^{(5)})$. For each of the observations in cluster C_1, compute the Silhouette coefficient using the information from the dissimilarity matrix. Comment on the suitability of their assignment to cluster C_1.

Note: The Silhouette coefficient of an observation $\mathbf{x}$ is defined as:

$$SC(\mathbf{x}) = \frac{b(\mathbf{x}) - a(\mathbf{x})}{\max\{b(\mathbf{x}), a(\mathbf{x})\}},$$

where $a(\mathbf{x})$ is the average dissimilarity of $\mathbf{x}$ with respect to other observations in its cluster, and $b(\mathbf{x})$ is the minimum average dissimilarity of $\mathbf{x}$ with respect to all clusters to which it does not belong. **[10 marks]**

Phoebe

$C_1 = (\vec{x}^{(1)}, \vec{x}^{(2)})$

$a(\vec{x}) = \dfrac{\text{sum of all distances from ex in cluster}}{|C_i| - 1}$

$b(\vec{x}) = \min \dfrac{\text{sum of all distances in cluster } C_k}{|C_k|}$

$SC(\vec{x}) = \dfrac{b(\vec{x}) - a(\vec{x})}{\max\{a(\vec{x}), b(\vec{x})\}}$

for 1 example

$\underline{x^{(1)}}$:

$a(x^{(1)}) = \frac{0.3}{1} = 0.3$

$b(x^{(1)}) = \frac{0.6+0.7+0.8}{3} = 0.7$

$SC(x^{(1)}) = \frac{0.4}{0.7} = 0.5714$

$\underline{x^{(2)}}$:

$a(x^{(2)}) = 0.3$

$b(x^{(2)}) = \frac{0.1+0.2+0.4}{3} = 0.2333$

$SC(x^{(2)}) = \frac{-0.0667}{0.3} = -0.\cancel{0}222$

Shay

b) $C_1 = (x_1, x_2)$ $\qquad$ $C_2 = (x_3, x_4, x_5)$

$$SC(x) = \frac{b(x) - a(x)}{\max\{b(x), a(x)\}}$$

~~$a(x_1) =$~~ ~~$b(x_1)$~~

~~$a(x_2) =$~~ ~~$b(x_2)$~~

$a(x_1) = 0.3$ $\qquad$ $a(x_2) = 0.3$

$b(x_1) = (0.6 + 0.7 + 0.8)/3$
$= 0.7$

$b(x_2) = (0.1 + 0.2 + 0.4)/3$
$= 0.2\dot{3}$

$SC(x_1) = \dfrac{0.7 - 0.3}{0.7} = 0.571$ ✓ well suited to it's cluster.

$SC(x_2) = \dfrac{0.23 - 0.3}{0.3} = -0.233$ ✗ would be more optimally clustered in a different cluster.

Question 2 Supervised Learning

(a) The following pseudo-code represents one iteration through the training set for gradient descent applied to univariate linear regression.

```
cost = 0;
w0 = 0;
w1 = 0;
for j=1 to size(trainingSet) do
    f = w0 + w1*x(j);
    cost = cost + (y(j) − f)^2;
    w0 = w0 - α * (f − y(j));
    w1 = w1 - α * (f − y(j)) * x(j);
```

Assume that the value of the learning rate, α is 1.

Give the numerical values of 'cost', 'w0', and 'w1' at the end of the execution of this pseudo-code for the following training set: $\{(-3, -1), (1, 1), (2, 5)\}$. **Show all your working.** **[10 marks]**

Shay

2) a) cost = 0 w0 = 0 w1 = 0

(-3, -1)

$f = 0 + 0(-3) = 0$

$cost = 0 + ((-1) - 0)^2 = 1$

$w0 = 0 - 1(0 - (-1)) = -1$

$w1 = 0 - 1(0 - (-1))(-3) = 3$

cost = 1 w0 = -1 w1 = 3

(1, 1)

$f = -1 + 3(1) = 2$

$cost = 1 + (1 - 2)^2 = 2$

$w0 = -1 - 1(2 - 1) = -2$

$w1 = 3 - 1(2 - 1)(1) = 2$

cost = 2 w0 = -2 w1 = 2

(2, 5)

$f = -2 + 2(2) = 2$

$cost = 2 + (5 - 2)^2 = 11$

$w0 = -2 - 1(2 - 5) = 1$

$w1 = 2 - 1(2 - 5)(2) = 8$

cost = 11 w0 = 1 w1 = 8


```
cost = 0;
w0 = 0;
w1 = 0;
for j=1 to size(trainingSet) do
    f = w0 + w1*x(j);
    cost = cost + (y(j) − f)^2;
    w0 = w0 - α * (f − y(j));
    w1 = w1 - α * (f − y(j)) * x(j);
```

(b) Suppose that you want to use a k-Nearest Neighbours classification, but the distance metric is not explicitly specified to you. Instead, you are given a "black box" where you input a set of points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots \mathbf{x}^{(N)}$ and a new point $\mathbf{x}$, and the black box outputs the nearest neighbour of $\mathbf{x}$ and its corresponding class label.

Can you construct a k-Nearest Neighbour classification algorithm based on this black box alone? The suggested algorithm should work for any given value of $k \in \{1, 2, \ldots, N\}$. If so, describe your algorithm. If not, explain why not? **[10 marks]**

Phoebe

kNN (trainingSet , $\vec{x}$, k):
  labels = [ ] ← tempSet = trainingSet
  if k > 0:
    for (int i = 0; i < k ; i++): tempSet
    labels.append(blackbox.label (~~trainingSet~~, $\vec{x}$))
    tempSet = trainingset.remove(blackbox.closestneighbour(trainingset, $\vec{x}$))
  for x in range(0, len(labels)):
    $\vec{x}$.label = findmostpopulouslabel(labels)

Shay

2) b) The k-Nearest Neighbour algorithm ~~looks~~ is ~~predet~~ initialised with a k value. This k value determines the number of neighbours to consider when classifying a data point. It will classify a data point based on what the majority of ~~the~~ its k neighbours are classified as. ~~With the black box, we won't be able to construct~~

With a set of points, ~~let's images we can pass in~~ and a new point x, we can pass them all into the black box to get the closest point to x. Say our set of initial points is {(1), (2), (5), (7)} and our new point x is (8), we would recieve (7) from the black box output if the black box uses simple ~~Euclidean~~ Manhattan distance. Then, pass in the set of points excluding the (7) we know of, ~~and~~ so {(1), (2), (5)} and the new point (8), to get the second nearest neighbour. Repeat k times to get the k closest neighbours, and then run the rest of the algorithm as normal, ~~taken~~ classifying the new point (8) based on a majority vote of its k nearest neighbours.

Question 3 Optimisation

(a) Give one strength and one weakness of Hill Climbing. **Justify** your answer. **[10 marks]**

+ve	-ve
- no heuristics needed	- gets stuck in local min/max or plateaus
- greedy algorithm	- not complete (e.g. asymptote)
- less computing power	- can get stuck in infinite/loopy paths
-easier to implement in code	- only looks at immediate neighbours
- lower time complexity since greedy compared to smth like simulated annealing	
- low space complexity	

Shay

3) a) One strength of Hill Climbing is that it isn't very computationally heavy. We just need to generate direct neighbours and then do a single comparison for each neighbour to see if it improves the solution. Furthermore, this simplicity will lead us to an optimal solution in the event that there are no suboptimal local minima/maxima.

One weakness of Hill Climbing is that most of the time it will not ~~to~~ reach an optimal solution. Because it only checks direct neighbouring states, the algorithm can easily get stuck in a local minimum or maximum, ~~and~~ returning a suboptimal solution.

(b) Consider the following optimisation problem and algorithm design to solve it:

Optimisation Problem

A builder has N possible clients. Each client $i \in \{1, \ldots, N\}$ would like the builder to undertake a job that takes h_i hours to complete. The builder wishes to decide which clients to accept over the next 30 days so as to maximise their income while ensuring that they work no more than 7.5 hours per day. This problem can be mathematically formulated as follows:

$$\sum_{i=1}^{N} x_i h_i \leq 30 \times 7.5$$

$$\begin{aligned} &\text{maximise} && f(\mathbf{x}) \\ &\text{subject to} && g(\mathbf{x}) = \tfrac{1}{30}\left(\sum_{i=1}^{N} x_i h_i\right) - 7.5 \leq 0 \end{aligned}$$

where $\mathbf{x}$ is a vector of size N, and $\forall i \in \{1, \ldots, N\}$, $x_i = 0$ if the builder does not accept job i and $x_i = 1$ otherwise; and $f(\mathbf{x})$ is a function that calculates the income.

Simulated Annealing Algorithm Design

Representation: a direct representation of the design variable $\mathbf{x}$. In other words, a vector $\mathbf{x}$ of size N, where $\forall i \in \{1, \ldots, N\}$, $x_i = 0$ if the builder does not accept job i and $x_i = 1$ otherwise.

Algorithm 1: Initialisation Procedure.

Input: Number of possible clients N.
Output: Candidate solution $\mathbf{x}$.

```
x = new vector of size N;
for i=1 to N do
    x_i = value picked uniformly at random from {0, 1};
return x
```

Algorithm 2: Neighbourhood Operator.

Input: Current solution $\mathbf{x}$; number of possible clients N.
Output: Neighbour $\mathbf{x}'$.

```
x' = copy of x;
i = value picked uniformly at random from {1, ..., N};
x'_i = 1 - x_i;
if g(x') > 0 then
    x'_i = 0;
return x'
```

infeasible → infeasible → infeasible
→ feasible → feasible
→ feasible → ...

Are the representation, initialisation and neighbourhood operators correctly designed (i.e., suitable) for this problem? Assume that we wish to deal with the constraints of this problem based on the design of the algorithm's operators. **Justify** your answer by explaining either why all three operators are suitable, or what is wrong.

Note:

- The problem formulation correctly reflects the intended problem, i.e., you do not need to check whether the problem formulation itself is correct.
- You do not need to consider how efficient the design of the operators is, just whether or not it is a correct design for the problem.
- It is acceptable for the neighbourhood operator to sometimes generate a neighbour that is the same as the current solution.

[10 marks]

Phoebe

b) Does the **representation** reflect the intended problem?

yes. design variable is variable, where the builder either accepts or doesn't accept the job.

Does the **initialisatation** reflect the intended problem?

yes. it will seem "random" how builders decide to accept their jobs, and we must check each N builders

Do the **neighbourhood operators** reflect the intended problem?

yes.

- if the neighbour's $g(x) > 0$ (broken constraint) we make it so they're "not accepted" the job (i.e. $x_i' = 0$)

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

current ← *problem*.INITIAL

for $t = 1$ **to** ∞ **do**

$T \leftarrow schedule(t)$

if $T = 0$ **then return** *current*

next ← a randomly selected successor of *current*

$\Delta E \leftarrow$ VALUE(*current*) − VALUE(*next*)

if $\Delta E > 0$ **then** *current* ← *next*

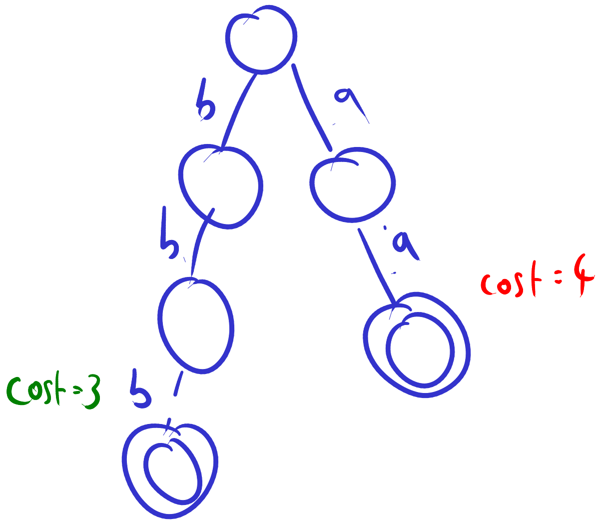
else *current* ← *next* only with probability $e^{\Delta E/T}$

Question 4 Search

(a) Consider a problem with two possible actions: *a* and *b*. The cost of *a* is 2 and the cost of *b* is 1. The shallowest goal node can be at any level of the tree and the problem solution may involve a sequence of these actions.
Is breadth-first search optimal to solve this problem, given this cost function? **Justify** your answer.

[10 marks]

No - it can find a short path (in terms of number of actions taken) but will not necessarily find one with the lowest cost.



Shay

4) a) A breadth-first search ~~explo~~ ~~to~~ explores the nodes in a graph on a level by level basis. Since the cost of expanding and adding an "a" action is double that of expanding a "b" action, a breadth-first search may not be as efficient or optimal as ~~expanding the~~ iteratively expanding from the node with the smallest path cost that we haven't expanded from yet. e.g. instead of expanding a, b, aa, ab, ba, bb

cost: 2 1 4 3 3 2

we expand b, a, bb, ab, ba, aa ✓ use a path-finding algo like A*

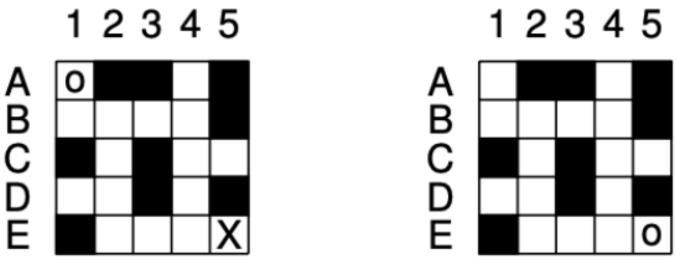
cost: 1 2 2 3 3 4

since then we may find the shallowest goal node ~~goal~~ ~~satisfy~~ with lower cost.

(b) An agent must traverse a maze in order to find and collect a treasure. This task can be formulated as a search problem as in the following:

- Initial state: the agent is in $A1$.
- Goal test: the agent is in $E5$ (automatically collects the treasure).
- Actions: *move(n,n')* moves the agent from state n to a state n' that is to the left of, or the right of, or above or below state n.
- Transition model: see figure below. States depicted in black, e.g., $A2$, are considered obstacles and cannot be traversed by the agent.
- Path cost: each action has cost 1.

Generate the A* tree until the goal node is found.



Initial State **Goal State**

The heuristic to be used is the following:

$$h(n) = d_M(n, n_{goal}) = \sum_{i=1}^{2} |p_i - q_i|,$$

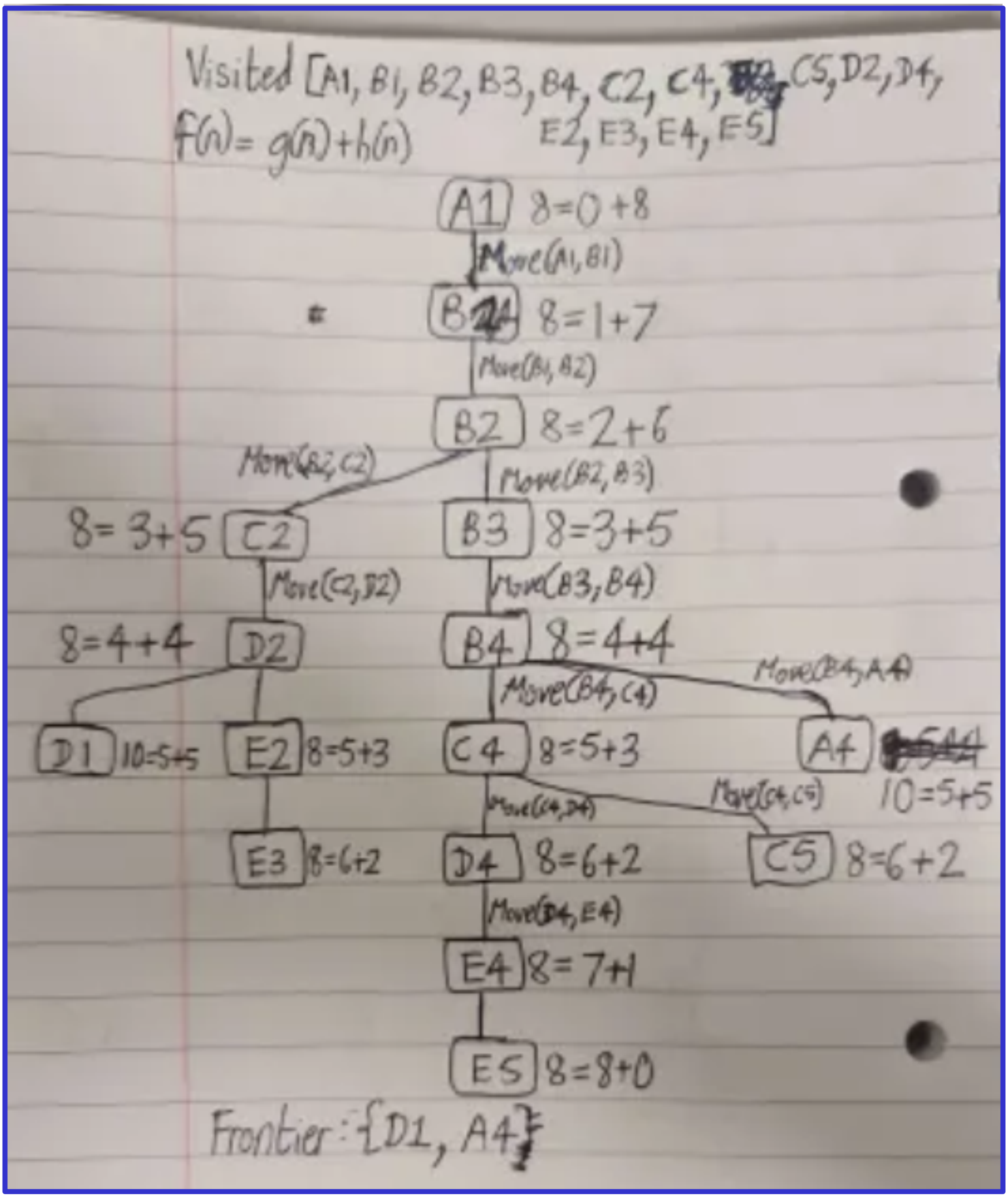
where $d_M(n, n_{goal})$ represents the Manhattan distance between node n and the goal node n_{goal}. When comparing the first coordinate of each state, assume the following values for each of the possible letters in the grid: $A = 1$, $B = 2$, $C = 3$, $D = 4$ and $E = 5$. Order of expansion: if $f(n)$ produces the same lowest value for two nodes, expand the nodes in alphanumerical order (e.g., $A2$ expanded before $B1$). Additionally, if $g(n)$ also produces the same lowest value for two nodes, keep the the node that has been added last into the frontier. Recall that the evaluation function $f(n) = g(n) + h(n)$ is the sum of the cost to reach node n and the heuristic calculated at node n.

Write down the following:

- Search tree produced by A*, indicating the $f(n)$, $g(n)$ and $h(n)$ values of each node n; which nodes are in the frontier when the algorithm terminates; and which nodes have been pruned as a result of the above instructions.
- The solution retrieved by A* and its cost.
- The sequence of nodes visited by A* in the order they are visited. Note: you can identify a node through its state, e.g., $A1$ or $B1$.

[10 marks]

Adwit


Visited [A1, B1, B2, B3, B4, C2, C4, C5, D2, D4, E2, E3, E4, E5]
f(n) = g(n) + h(n)
A1 8=0+8
Move(A1, B1)
B1 8=1+7
Move(B1, B2)
B2 8=2+6
Move(B2, C2)
Move(B2, B3)
8= 3+5 C2
B3 8=3+5
Move(C2, D2)
Move(B3, B4)
8=4+4 D2
B4 8=4+4
Move(B4, A4)
Move(B4, C4)
D1 10=5+5
E2 8=5+3
C4 8=5+3
A4
10=5+5
Move(C4, C5)
Move(C4, D4)
E3 8=6+2
D4 8=6+2
C5 8=6+2
Move(D4, E4)
E4 8=7+1
E5 8=8+0
Frontier: {D1, A4}