ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Autumn Semester 2022 Prof. R. Wattenhofer

Computational Thinking

Wednesday, January 25, 2023, 15:00-17:00

Do not turn over until instructed to do so

This examination lasts for 120 minutes and comprises 120 points. There is one block of questions for each lecture chapter.

You may answer in German, English, or combine German and English.

Unless explicitly stated, you do **not** have to justify your answers. Writing down your thoughts (math, text, or annotated sketches), however, might help with the understanding of your approach. This may then result in points being awarded even if your answer is not correct. Please write legibly. Unreadable answers will not be graded.

Some questions will ask you to fill in answers in a template. If you decide to start over you will find fill-in replacements at the end of the examination booklet.

Please write your name and student number on every additional sheet. Please write your name and student number in the following fields on this cover sheet.

Family Name	**First Name**	**Student Number**

Task	**Achieved Points**	**Maximum Points**
1 - Algorithms		19
2 - Complexity		14
3 - Cryptography		17
4 - Databases		17
5 - Machine Learning		20
6 - Neural Networks		16
7 - Computability		17
Total		**120**

1 Algorithms (19 points)

Trampoline World

You are visiting *Trampoline World*, a theme park with lots of trampolines. Travelling from one part of the park to another is only possible by jumping along a corridor of trampolines. These corridors are one-way, you are not allowed to jump backwards. You are given an array of trampoline `strengths` (all values strictly positive integers). If you jump on trampoline i with strength k you can maximally jump to the $(i+k)^{\text{th}}$ trampoline. You may also decide to jump less far and land on a closer trampoline. You start out on trampoline 0 and want to end up on the landing mat behind the last trampoline. You want to use the least number of trampolines to get from trampoline 0 to the landing mat.

```
1 def minTrampolineJumpsNaive(strengths, i):
2     if i >= len(strengths):
3         return 0
4     temp = len(strengths)
5     for k in range(1, strengths[i]):
6         temp = min(temp, 1+minTrampolineJumpsNaive(strengths,
          ↪  k))
7     return temp
```

a) `[4 points]` `minTrampolineJumpsNaive` does not yet run correctly but you want to define two test cases. What should a correct implementation of `minTrampolineJumps` return for the following inputs?

`minTrampolineJumps([4, 5, 2, 4, 3, 2, 1], 0) =`

`minTrampolineJumps([2, 4, 1, 2, 5, 1, 3, 2, 3, 1, 1], 0) =`

b) `[2 points]` The variable `temp` is not very descriptive. Can you help make the code more readable by giving `temp` a better name?

c) `[4 points]` `minTrampolineJumpsNaive` does not return the correct result, it has **two mistakes**. Find and correct them directly in the above code snippet.

d) [2 points] Even when corrected, `minTrampolineJumpsNaive` turns out to be unsuitable for a very long corridor of trampolines. Explain why.

e) [7 points] It turns out that this problem can be solved optimally with a greedy approach. Find and correct the **two mistakes** in the code and write down the time complexity of the algorithm in the big-O notation.

```
1  def minTrampolineJumpsGreedy(strengths):
2      result = 0
3      left = 0
4      right = 0
5      while right < len(strengths):
6          rightmost = 0
7          for i in range(left, right+1):
8              rightmost = min(rightmost, i + strengths[i])
9          left = right+1
10         right = rightmost+1
11         result += 1
12     return result
```

Time complexity:

Solution

Trampoline World

a) `minTrampolineJumps([4, 5, 2, 4, 3, 2, 1], 0) == 2`
`minTrampolineJumps([2, 4, 1, 2, 5, 1, 3, 2, 3, 1, 1], 0) == 4`

b) Multiple possibilities, e.g. `leastAmountOfJumps`, `currentMinimumJumps`.

c) Following snippet has two changes: strength[i]+1, $i + k$

```
def minTrampolineJumpsNaive(strengths, i):
    if i >= len(strengths):
        return 0
    temp = len(strengths)
    for k in range(1, strengths[i]+1):
        temp = min(temp, 1+minTrampolineJumps(strengths, i+k))
    return temp
```

d) Because the recursive algorithm computes every jump possibility from every trampoline, a lot of redundant computation is happening. Exponential runtime.

e) Following snippet has two changes: max(...), right = rightmost

```
def minTrampolineJumpsGreedy(strengths):
    result = 0
    left = 0
    right = 0
    while right < len(strengths):
        rightmost = 0
        for i in range(left, right+1):
            rightmost = max(rightmost, i + strengths[i])
        left = right+1
        right = rightmost
        result += 1
    return result
```

Time complexity: $O(n)$ where $n = len(strengths)$ because `i` visits every element in `strengths` at most once.

2 Complexity (14 points)

Polynomial Time Reductions

Let A and B be two decision problems. What can the existence of a polynomial time reduction from A to B ("$A \leq B$")...

a) `[3 points]` ...prove about A given knowledge about the complexity class of B?

A is in **P**.	○ True	○ False
A is **NP**-hard.	○ True	○ False
A is **NP**-complete.	○ True	○ False

b) `[3 points]` ...prove about B given knowledge about the complexity class of A?

B is in **P**.	○ True	○ False
B is **NP**-hard.	○ True	○ False
B is **NP**-complete.	○ True	○ False

Easier TSP

From the lecture we know that it is NP-hard to approximate the traveling salesperson problem (TSP) up to any constant factor $\alpha > 1$ for general graphs. For metric graphs however, a 2-approximation was given.
Decide whether TSP can be approximated in polynomial time up to a constant factor in the following two cases. In each case, state your answer (Yes/No) and justify it in 1-3 sentences.

c) `[4 points]` Graphs where each edge length d is a real number with $1 \leq d \leq 2$.

d) `[4 points]` Graphs where each edge length is the multiple of a positive integer k.

Solution

a) A is in P.

b) B is **NP**-hard.

c) Yes. In such graphs all round trips have lengths between n and $2n$ (inclusive). So choosing any round trip is already a 2-approximation. (Alternative: All such graphs are metric since the triangle inequality always holds with these edge lengths ($1 + 1 \geq 2$), and hence there exists a 2-approximation.)

d) No. The inapproximability proof in Chapter 2 of the script only uses graphs with two different edge lengths: 1 and $\alpha \cdot n + 1$. The proof works works exactly the same using edge lengths k and $k(\lceil \alpha \cdot n + 1 \rceil)$ instead.

3 Cryptography (17 points)

Public-Key Cryptography

a) `[2 points]` Let $g = 2$, $p = 19$. Compute the public key k_p, if the secret key is $k_s = 6$.

b) `[4 points]` Let $g = 2$, $p = 19$ and $k_s = 6$. Decrypt the ElGamal encrypted message $C = (2, 7)$.

Zero Knowledge Proofs

Consider a setup with combination bike locks. A bike lock can only be opened by somebody who knows the combination, and it is impossible to open a bike lock just by guessing the combination. Watson (on the left) knows the combinations of *all* bike locks.

Figure 1: Illustration of the setup with two combination bike locks.

c) `[3 points]` Let us first consider the case with only *two* bike locks. Describe a protocol that allows Sherlock (on the right) to prove to Watson (on the left) that he knows how to open at least *one* of the bike locks without revealing to Watson *which* bike lock he (Sherlock) can open.

d) [3 points] Consider now *ten* bike locks. Again, Sherlock wants to prove to Watson that he knows the combination of at least one of the bike locks (without revealing which one). Show that the following protocol is not zero-knowledge.

```
def Ten_Bike_Locks():
    1. Watson connects the locks to form a chain ring
    2. Watson gives the chain ring to Sherlock
    3. Sherlock opens the chain ring by opening one of the locks
    4. Sherlock gives the opened ring (as a chain) to Watson
```

Figure 2: Illustration of Ten Bike Locks protocol.

e) [5 points] Design a protocol that fixes the problem in the previous exercise. That is, describe a protocol that allows Sherlock to prove to Watson that he knows how to open at least *one* of the ten bike locks without revealing to Watson *which* bike lock Sherlock can open.

Solution

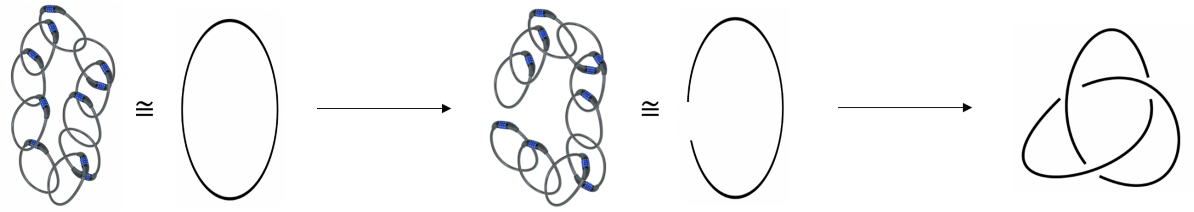
a) $k_p = g^{k_s} = 2^6 \mod 19 = 64 \mod 19 = 7$.

b) $m' = c_2 \cdot c_1^{k_s \cdot (p-2)} \mod p = 7 \cdot 2^{6 \cdot 17} \mod 19 = 7 \cdot (2^6)^{17} \mod 19 = 7 \cdot 7^{17} \mod 19 = 7^{18} \mod 19 = 1$. In the last step, we use Fermat's Little theorem.

c) Watson gives to Sherlock two combination bike locks (already locked). Sherlock opens one of the bike locks and locks it with the other (forming a chain). Sherlock shows the chain to Watson. Sherlock cannot form the chain without knowing how to open one of the locks, and Watson cannot know which one Sherlock locked (since it is a chain). Below, the protocol is illustrated as a figure.

d) Watson can tell that either the first or the last combination bike lock in the chain is the one that Sherlock opened.

e) One of the possible solutions is the following. In Step 3, Sherlock first opens the chain ring (by opening on of the locks). Then, he creates a knot in the chain and the closes the ring again. In Step 4, Sherlock gives the knotted chain ring back to Watson. The illustration below explains how to create a knotted ring.



Another possible solution: Watson creates two chain rings, each with 5 combination bike locks and gives them to Sherlock. Sherlock opens one of the rings (by being able to open at least one of the bike locks) and interlocks the rings together. Sherlock shows the interlocked rings to Watson. This is illustrated in the given figure below.

4 Databases (17 points)

Join Types

a) `[7 points]` Given the following two tables:

a	b	c
1	1	1
2	2	1
3	3	1
4	0	1

Figure 3: Table `left`.

d	e
1	1
2	1
3	1
4	1

Figure 4: Table `right`.

For each pair of queries below, list all valid pairs of column names (to replace <x> and <y>), such that both queries output the same number of rows:

```
SELECT * FROM left INNER JOIN right ON left.<x> = right.<y>;
SELECT * FROM left LEFT OUTER JOIN right ON left.<x> = right.<y>;
```

```
SELECT * FROM left FULL OUTER JOIN right ON left.<x> = right.<y>;
SELECT * FROM left RIGHT OUTER JOIN right ON left.<x> = right.<y>;
```

```
SELECT * FROM left FULL OUTER JOIN right ON left.<x> = right.<y>;
SELECT * FROM left INNER JOIN right ON left.<x> = right.<y>;
```

Course Registration DB Queries

We consider a database of a university course registration system. It has tables for students, courses, and enrollments of students in courses (table keys are underlined):

```
students(id, name)
courses(id, name, max_students)
enrollments(student_id, course_id, priority)
```

The column `priority` in the `enrollments` table indicates the student's preferences, a lower number indicates higher priority. The column `max_students` in the `courses` table indicates the maximum number of students allowed for admission of the course.

b) `[4 points]` What does the following SQL query return?

```
SELECT c.id, COUNT(e.student_id) - c.max_students AS x
FROM courses c
LEFT OUTER JOIN enrollments e ON e.course_id = c.id
GROUP BY c.id
HAVING x > 0;
```

c) `[2 points]` What would change in the output of the query from b) if we changed COUNT(e.student_id) to COUNT(DISTINCT e.student_id) and why?

d) `[4 points]` Write a SQL query which returns for all students their names and the names of all courses they are enrolled for. Also consider the case that the students might not be enrolled for any courses. The student should still appear in that case. Courses which no one enrolled for on the other hand should not appear. The output should be ordered for each student (not over all students) by their priority for the respective course (high priority to low priority).

Solution

a) INNER JOIN = LEFT OUTER JOIN: (a,d), (c,d), (c,e)
RIGHT OUTER JOIN = FULL OUTER JOIN: (a,d), (c,d), (c,e)
FULL OUTER JOIN = INNER JOIN: (a,d), (c,e)

b) The query returns for each course that is overbooked, i.e. has more students enrolled for it than its `max_students` field allows, the ID of the course and how many students over the maximum are enrolled.

c) Nothing, since `student_id` and `course_id` together form a key on the `enrollments` table and are therefore unique.

d)
```
SELECT s.name, c.name
FROM students s
LEFT OUTER JOIN enrollments e ON e.student_id = s.id
LEFT OUTER JOIN courses c ON e.course_id = c.id
ORDER BY s.id, e.priority ASC;
```

5 Machine Learning (20 points)

You are given a set of samples $x_0, x_1, \ldots, x_{n-1} \in \mathbb{R}$ and want to fit them to the ground truth labels $y_0, y_1, \ldots, y_{n-1} \in \mathbb{R}$ using the linear regression model $\hat{f}(x_i) = w_0 + w_1 x_i$. We can pair the input values and labels to get the dataset $D := \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$.
The first thing you need to decide is what loss function to use. Two possibilities are the Mean Squared Error loss (MSE) and the Mean Absolute Error (MAE), defined as follows:

$$L_{MSE}(\hat{f}, D) = \frac{1}{n} \sum_{(x,y)\in D} (y - \hat{f}(x))^2 \qquad L_{MAE}(\hat{f}, D) = \frac{1}{n} \sum_{(x,y)\in D} |y - \hat{f}(x)|$$

a) `[3 points]` We know that our dataset may contain so-called *outliers*, data points that result in a big error even if the model fits the underlying distribution we sampled from perfectly. Which of the two losses is impacted less by these samples and results in a model that fits closer to the real distribution? Explain why.

b) `[3 points]` In the following we fit the data with Gradient Descent using MAE. The dataset and the weights (after some learning steps) are as follows:

$$D = \{(x_0 = 1, y_0 = 1), (x_1 = 2, y_1 = 4)\} \qquad w_0 = w_1 = 1$$

What are the current predictions and MAE loss for this model?

$\hat{f}(x_0) =$ $\square$ $\hat{f}(x_1) =$ $\square$ $L_{MAE}(\hat{f}, D) =$ $\square$

c) `[4 points]` The MAE Loss is not differentiable for $y = \hat{f}(x)$. We therefore set the gradient at this point to 0. What are the partial derivatives for w_0 and w_1 on the rest of the loss function for just one datapoint (x, y)?

$$\frac{\partial L}{\partial w_0} L_{MAE}(\hat{f}, \{(x,y)\}) = \begin{cases} \square & \hat{f}(x) > y \\ \square & \hat{f}(x) < y \\ 0 & \hat{f}(x) = y \end{cases} \qquad \frac{\partial L}{\partial w_1} L_{MAE}(\hat{f}, \{(x,y)\}) = \begin{cases} \square & \hat{f}(x) > y \\ \square & \hat{f}(x) < y \\ 0 & \hat{f}(x) = y \end{cases}$$

d) [4 points] Run one step of Gradient Descent using $\alpha = 2$ and report the new values.

$w_0^{new} =$ $\square$ $w_1^{new} =$ $\square$

e) [2 points] What are the optimal weights w_0^* and w_1^* for MAE and what is the resulting loss?

$w_0^* =$ $\square$ $w_1^* =$ $\square$ $L_{MAE} =$ $\square$

f) [4 points] Could we have chosen a different α in **d)** such that we achieve the optimal loss in a single step? Explain why.

Solution

a) The MSE punishes big errors more than MAE and is thus impacted more by them. The MAE would result in a model that fits the real distribution more closely in this case.

b) $\hat{f}(1) = 2, \hat{f}(2) = 3, L_{MAE}(\hat{f}, D) = 1$

c) $\frac{\partial L}{\partial w_0} L_{MAE}(\hat{f}, \{(x,y)\}) = \begin{cases} +1 & \hat{f}(x) > y \\ -1 & \hat{f}(x) < y \\ 0 & \hat{f}(x) = y \end{cases}$ $\frac{\partial L}{\partial w_1} L_{MAE}(\hat{f}, \{(x,y)\}) = \begin{cases} +x & \hat{f}(x) > y \\ -x & \hat{f}(x) < y \\ 0 & \hat{f}(x) = y \end{cases}$

d) $w_0^{new} = 1 - \alpha \frac{1}{n}(+1 - 1) = 1,$
$w_1^{new} = 1 - \alpha \frac{1}{n}(x_0 - x_1) = 1 - 0.5\alpha \cdot (-1) = 1 + 0.5\alpha = 2$

e) $w_0^* = -2, w_1^* = 3, L = 0$

f) In question **d)**, we saw that $w_0^{new} = w_0$, no matter what α is. We also know that it has to become -2 to minimize the loss (without $w_0 = -2$, it is not possible for the loss terms on the two datapoints to become both 0). This means that we cannot reach the optimal weights in just one step.

6 Neural Networks (16 points)

Consider the alphabet $\mathcal{A} = \{A, B, C\}$. We are given a sequence of letters from this alphabet and our goal is to find out if there is an occurrence of two *Bs* directly after each other in that sequence, for example ABC**BB**A. We decide to tackle the problem with an RNN, use one-hot encoding for all letters in $\mathcal{A}$ and a custom activation function $\sigma(z) = \begin{cases} 0 & z \leq 0 \\ z & 0 < z < 2 \\ 2 & 2 \leq z \end{cases}$.

We are given the one-dimensional state $\boldsymbol{s}_t$ and the input $\boldsymbol{x}_t$. Like in the lecture both are prepended with the bias term; instead of $\boldsymbol{s}_t = \kappa$, we really have $\boldsymbol{s}_t = (1, \kappa)$. We define the following state update function:

$$\boldsymbol{s}_{t+1} = \hat{h}(\boldsymbol{x}_t, \boldsymbol{s}_t) = \sigma\left(\boldsymbol{w}_x \cdot \boldsymbol{x}_t + \boldsymbol{w}_s \cdot \boldsymbol{s}_t\right)$$

During the training of the RNN, it slowly learns to represent all relevant information in the state. More specifically, $\boldsymbol{s}_t = 1$ if the RNN has *just* seen a B, and $\boldsymbol{s}_t = 2$ if the RNN already has encountered two consecutive *Bs*, given initial state $\boldsymbol{s}_0 = 0$.

a) `[10 points]` Here are ten $\boldsymbol{w}_x$ and $\boldsymbol{w}_s$ combinations that could appear during training. Mark all $\boldsymbol{w}_x$ and $\boldsymbol{w}_s$ with True that produce a state $\boldsymbol{s}_t$ as described above.

- $\boldsymbol{w}_x = \begin{pmatrix} -1 & 0 & 1 & 0 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} -1 & 1 \end{pmatrix}$ ○ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 1 & -3 \end{pmatrix}$ ○ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} -4 & 0 & 5 & 0 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & 3 \end{pmatrix}$ ○ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} -3 & 0 & 4 & 0 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & 2 \end{pmatrix}$ ○ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} 0 & -1 & 1 & -1 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & 2 \end{pmatrix}$ ○ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} -1 & -2 & 1 & -2 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 1 & 2 \end{pmatrix}$ ○ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} 1 & -1 & 1 & -1 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & 1 \end{pmatrix}$ ○ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} -1 & 2 & -1 & 2 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & -1 \end{pmatrix}$ ○ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} 2 & -2 & 1 & -2 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} -2 & 2 \end{pmatrix}$ ○ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} -1 & -1 & 2 & -1 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & 2 \end{pmatrix}$ ○ True ○ False

b) [4 points] Assume the sequence ends with a STOP token, encoded as $\begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix}^T$ (prepended with the bias term at the beginning). Fill in the correct values for the output function $y_T = \hat{g}(\boldsymbol{x}_T, \boldsymbol{s}_T)$, that outputs a scalar y_T that is 1 if and only if the RNN has encountered two consecutive Bs in the sequence and $\boldsymbol{x}_T$ is the STOP token.

$$y_T = \sigma\left(\begin{pmatrix} \square & \square & \square & \square \end{pmatrix} \cdot \boldsymbol{x}_T + \begin{pmatrix} \square & \square \end{pmatrix} \cdot \boldsymbol{s}_T\right)$$

c) [2 points] Is it possible to solve this task with the attention mechanism applied to all one-hot encoded elements of the sequence (and no other information)? Justify your answer.

Solution

a) Lets name the elements of our weight matrices in the following way: $\boldsymbol{w}_x = \begin{pmatrix} x1 & a & b & c \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} x2 & s \end{pmatrix}$. We combine the bias term $bias = x1 + x2$. Our activation function σ only allows the state $\boldsymbol{s}_t$ to be 0, 1 or 2. As described in the exercise, our state needs to be $\boldsymbol{s}_T = 2$ as soon as there have been two consecutive Bs in the sequence. We have $\boldsymbol{s}_{t+1} = 1$ if $\boldsymbol{x}_t = B$ but $\boldsymbol{x}_{t-1} \neq B$ and previously there have not been two consecutive Bs in the sequence. We have $\boldsymbol{s}_{t+1} = 0$ if $\boldsymbol{x}_t \neq B$ and previously there have not been two consecutive Bs in the sequence. Given these premises, we can define a set of rules that need to hold in order for the RNN to achieve the desired states (all rules defined for a equally need to hold for c as well):

(1) $bias + a < 0$ (So that state 1 can be reset to 0)

(2) $bias + b = 1$ (So that state 0 can be set to 1 and state 1 can be set to 2)

(3) $bias + a + s \leq 0$ (So that state 1 can be reset to 0 if A or C come after a single B)

(4) $bias + b + s \geq 2$ (So that state 1 can be set to 2 after two Bs in a row)

(5) $bias + a + 2s \geq 2$ (so that state 2 stays at 2 even after other letters appear after the two Bs)

Of the following solutions, all incorrect ones violate at least one of the above rules and therefore lead to incorrect states, violating the $\boldsymbol{s}_T = 2$ iff there have been two consecutive Bs in the sequence constraint.

- $\boldsymbol{w}_x = \begin{pmatrix} -1 & 0 & 1 & 0 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} -1 & 1 \end{pmatrix}$ ○ True ⊗ False (2)
- $\boldsymbol{w}_x = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 1 & -3 \end{pmatrix}$ ○ True ⊗ False (1,2,4,5)
- $\boldsymbol{w}_x = \begin{pmatrix} -4 & 0 & 5 & 0 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & 3 \end{pmatrix}$ ⊗ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} -3 & 0 & 4 & 0 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & 2 \end{pmatrix}$ ○ True ⊗ False (5)
- $\boldsymbol{w}_x = \begin{pmatrix} 0 & -1 & 1 & -1 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & 2 \end{pmatrix}$ ○ True ⊗ False (3)
- $\boldsymbol{w}_x = \begin{pmatrix} -1 & -2 & 1 & -2 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 1 & 2 \end{pmatrix}$ ⊗ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} 1 & -1 & 1 & -1 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & 1 \end{pmatrix}$ ○ True ⊗ False (1,3)
- $\boldsymbol{w}_x = \begin{pmatrix} -1 & 2 & -1 & 2 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & -1 \end{pmatrix}$ ○ True ⊗ False (1,2,4)
- $\boldsymbol{w}_x = \begin{pmatrix} 2 & -2 & 1 & -2 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} -2 & 2 \end{pmatrix}$ ⊗ True ○ False
- $\boldsymbol{w}_x = \begin{pmatrix} -1 & -1 & 2 & -1 \end{pmatrix}, \boldsymbol{w}_s = \begin{pmatrix} 0 & 2 \end{pmatrix}$ ⊗ True ○ False

$$\boldsymbol{w}_x := \begin{pmatrix} 0 & 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \end{pmatrix}, \boldsymbol{w}_s := \begin{pmatrix} -1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

b) There are two options, 1 iff condition met, else 0:

$$y_T = \sigma\left(\begin{pmatrix} 0 & -1 & -1 & -1 \end{pmatrix} \cdot \boldsymbol{x}_T + \begin{pmatrix} -1 & 1 \end{pmatrix} \cdot \boldsymbol{s}_T\right)$$

The other option is 1 iff condition met, else 2 (due to 0 not being specified in question)

$$y_T = \sigma\left(\begin{pmatrix} 0 & 1 & 1 & 1 \end{pmatrix} \cdot \boldsymbol{x}_T + \begin{pmatrix} 3 & -1 \end{pmatrix} \cdot \boldsymbol{s}_T\right)$$

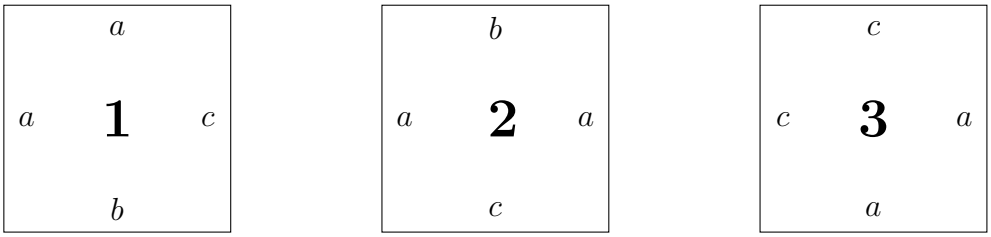
c) No in the standard way it is not possible, because the attention mechanism is invariant to the position of an element in the sequence. Only using positional encoding would allow for that to be possible.

7 Computability (17 points)

Tiling

a) [3 points] Consider a $m \times n$ grid that we want to fill with d given tiles, where $m, n, d \in \mathbb{N}$. Is the problem decidable? Explain your answer.

b) [5 points] Does a valid tiling of the infinite plane exist for the following three tiles? Explain your answer.



Turing Machines

Consider the Turing Machine (TM) described by the transition tables below. The initial tape stores a sequence of a's and b's (bounded by $\perp$ in the beginning and end). The reading head starts on the first non $\perp$ character. The machine consists of four states s_0, s_1, s_2, and s_3 plus a halting state s_h.

Starting state: s_0

Transitions from state s_0

Read		Write	Pointer	Next state
a	$\longrightarrow$	a	right	s_0
b	$\longrightarrow$	b	right	s_1
$\perp$	$\longrightarrow$	$\perp$	stay	s_h

Transitions from state s_1

Read		Write	Pointer	Next state
a	$\longrightarrow$	b	left	s_2
b	$\longrightarrow$	b	right	s_1
$\perp$	$\longrightarrow$	$\perp$	stay	s_h

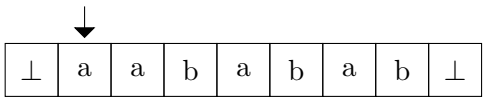
Transitions from state s_2

Read		Write	Pointer	Next state
a	$\longrightarrow$	$\perp$	stay	s_h
b	$\longrightarrow$	a	left	s_3
$\perp$	$\longrightarrow$	$\perp$	stay	s_h

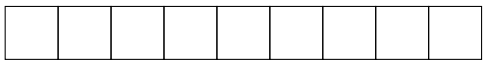
Transitions from state s_3

Read		Write	Pointer	Next state
a	$\longrightarrow$	a	left	s_3
b	$\longrightarrow$	b	left	s_3
$\perp$	$\longrightarrow$	$\perp$	right	s_0

c) `[5 points]` Consider the initial tape



Draw the contents of the tape and the position of the reading head after transitioning from s_3 to s_0 for the first time.



d) `[4 points]` What's the purpose of this TM? (In one high level sentence, how does the TM change the tape?)

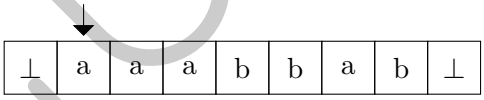
Solution

a) Because the grid is finite, we can brute force and try every of the at most $(mn)^d$ combinations. The problem is always decidable.

b) Yes, the following 3×3 square can be repeated to form a valid periodic tiling.

c / c **3** a / a	b / a **2** a / c	a / a **1** c / b
a / a **1** c / b	c / c **3** a / a	b / a **2** a / c
b / a **2** a / c	a / a **1** c / b	c / c **3** a / a

c) The tape contents and the position of the head after transitioning from s_3 to s_0 are as follows:



d) The Turing Machine sorts the input (first all a's then all b's).

Extra Pages

Use this page if you need extra space.