ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Autumn Semester 2021 Prof. R. Wattenhofer

# Computational Thinking

Friday, February 11, 2022, 9:00-11:00

**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 or English, or mix German and English.

Unless explicitly stated, you do **not** have to justify your answers. However, writing down your thoughts (math, text, or annotated sketches) might help us understand your approach. This may then result in points being awarded even if your answer is not correct. Please ensure that your handwriting is readable.

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 | | 16 |
| 2 - Complexity | | 18 |
| 3 - Cryptography | | 17 |
| 4 - Data and Storage | | 18 |
| 5 - Machine Learning | | 18 |
| 6 - Neural Networks | | 17 |
| 7 - Computability | | 16 |
| **Total** | | **120** |

# 1 Algorithms (16 points)

### Longest Common Subsequence

Given two strings, the longest common subsequence (LCS) is the longest subsequence present in both. A subsequence of a string is a new string generated from the original string with some characters (can be none or all) deleted without changing the relative order of the remaining characters. For example, an LCS of "ABCDEFG" and "AEDFHR" is "AEF" of length 3. The code below attempts to compute the length of the LCS of two strings.

```
def lcs(X, Y):
    if len(X) == 0 or len(Y) == 0:
        return 1
    elif X[-1] == Y[-1]:
        return 1 + lcs(X[:-1], Y[:-1])
    else:
        return min(lcs(X, Y[:-1]), lcs(X[:-1], Y))

# Define the two strings and call the function
X = "AGGTAB"
Y = "GXTXAYB"
print("Length of LCS is ", lcs(X, Y))
```

**a)** [4 pt] Algorithm $lcs(X, Y)$ does not correctly solve the LCS problem, since it has two small mistakes. Mark **and correct** both mistakes in the code above.

**b)** [3 pt] It can sometimes be difficult to calculate the exact runtime of an algorithm. This is why we usually work with asymptotic bounds that tell us enough about how the runtime scales when the input is big. Choose the tightest asymptotic upper bound for the runtime of the displayed $lcs(X, Y)$ algorithm. Assume that $m$ and $n$ denote the lengths of the two input strings, respectively.

- ○ $\mathcal{O}(m^2n^2)$
- ○ $\mathcal{O}(2^{\min(n,m)})$
- ○ $\mathcal{O}(2^{n+m})$
- ○ $\mathcal{O}(2^{nm})$

**c)** [9 pt] The above code is inefficient with respect to run time. Write an algorithm that uses bottom-up dynamic programming to solve the LCS problem in $\mathcal{O}(mn)$ time.

```
def fastLCS(X , Y):
    # Find the length of the strings
    m = len(X)
    n = len(Y)
    # Fill in the rest of the lcs function









# Define the two sequences and call the functions
X = "AGGTAB"
Y = "GXTXAYB"
print("Length of LCS is ", fastLCS(X, Y))
```

## Solution

### Longest Common Subsequence

**a)** Lines 3 and 7 all contain a mistake. The correct algorithm is displayed below.

```
1  def lcs(X, Y):
2      if len(X) == 0 or len(Y) == 0:
3          return 0
4      elif X[-1] == Y[-1]:
5          return 1 + lcs(X[:-1], Y[:-1])
6      else:
7          return max(lcs(X, Y[:-1]), lcs(X[:-1], Y))
8
9  # Define the two strings and call the function
10 X = "AGGTAB"
11 Y = "GXTXAYB"
12 print("Length of LCS is ", lcs(X , Y))
```

**b)** $\mathcal{O}(2^{n+m})$ where $m$ and $n$ are the lengths of $X$ and $Y$.

**c)**
```
def fastLCS(X , Y):
    # Find the length of the strings
    m = len(X)
    n = len(Y)
    memo = [[0]*(n+1) for i in range(m+1)]

    for i in range(1,m+1):
        for j in range(1,n+1):
            if X[i-1]==Y[j-1]:
                memo[i][j] = memo[i-1][j-1] + 1
            else:
                memo[i][j] = max(memo[i-1][j] , memo[i][j-1])
    return memo[m][n]

# Define the two sequences and call the functions
X = "AGGTAB"
Y = "GXTXAYB"
print("Length of LCS is ", fastLCS(X, Y))
```

## 2 Complexity (18 points)

Let $G = (V, E)$ be an undirected and unweighted graph and $s, t \in V$. In this question, paths can have no vertex twice. An $s, t$-path is a path between $s$ and $t$. Denote

- LPath$(G, s, t, k)$ the problem of finding an $s, t$-path of length strictly less than $k$,
- EPath$(G, s, t, k)$ the problem of finding an $s, t$-path of exactly length $k$,
- EPath$^k(G, s, t)$ the problem of finding an $s, t$-path of exactly length *constant* $k$,
- GPath$(G, s, t, k)$ the problem of finding an $s, t$-path of length strictly greater than $k$,
- HamPath$(G, s, t)$ the problem of finding a Hamiltonian path from $s$ to $t$.
  *[Recall that a Hamiltonian path is a path that visits each vertex in the graph exactly once.]*

**a)** `[3 points]` Prove that LPath is in **P**.

**b)** `[3 points]` Explain, without using $\mathbf{P} \subseteq \mathbf{NP}$, why LPath is in **NP**.

**c)** [4 points] Assume HAMPATH$(G, s, t)$ is **NP-hard**. Prove that GPATH is **NP-complete**.

**d)** [4 points] Is EPATH$^k$ in **P** for all $k \geq 0$? Is EPATH in **P**? Justify your answers.

**e)** [4 points] Show that if **P** = **NP** then EPATH$^{2022}$ is **NP-complete**.

## Solution

**a)** Multiple ways of doing this. A straightforward one is to do a breadth-first search while marking all the already-visited vertices and not visiting them when encountered again. If a path is found, we can verify that its length is $< k$ in linear time. This shows **P**-ness.

**b)** The path is the certificate verifiable in polynomial time. This is because we can check whether any two consecutive vertices listed in the certificate are neighbours in linear time (search the edge set for the pair), if no vertex appears twice in quadratic time (search for duplicates), if the path is of length $< k$ in linear time (count), and if the vertices at (beginning, end) are $(s, t)$ in constant time.

**c)** HamPath, GPath are **NP** because the path serves as a certificate. An example polynomial reduction from HamPath to GPath reads $\langle G, s, t\rangle$ (this is a linear-time operation), computes $n = |V| - 2$ (linear-time), and outputs $\langle G, s, t, n\rangle$ (linear-time).

Since a polynomial reduction from HamPath to GPath exists and HamPath is **NPC**, GPath is **NP-hard**, and since it is **NP**, it is **NPC**.

**d)** Yes, EPath$^k$ is in **P**. One can just change the requirement on $k$ in the proof of the first point. No, EPath is not in **P**. The reduction from point **c)** works here as well.

**e)** If **P** = **NP** then all **NPC** problems are in **P**, and recall that all* problems in **P** reduce polynomially to each other. Take SAT which is **NPC** (Cook-Levin in lecture notes). Then it reduces polynomially to EPath$^{2022}$, and hence EPath$^{2022}$ is **NPC**.

* Unimportant detail: almost all, save from Decide($\emptyset$) and Decide($\Sigma^*$).

# 3 Cryptography (17 points)

## Reductions

**a)** [3 points] Show that $DDH \leq CDH$.

## Commitment Schemes

Is the following commitment scheme correct, (computationally/perfectly) hiding and (computationally/perfectly) binding? Explain!

```
# p,g,m as defined in the lecture

def commit_Alice(m):
    Pick a random r ∈ {1,2,...,p−1}
    c = g^m · g^r  mod p
    Send c to Bob

def reveal_Bob(m,c,r): # Bob receives m,c,r from Alice
    return c == g^m · g^r  mod p
```

**b)** [2 points] Correctness:

**c)** [3 points] Hiding:

**d)** [3 points] Binding:

### Perfect vs Computational Security

**e)** `[6 points]` Complete the following table by specifying whether the algorithm is perfectly or computationally secure, as well as mentioning the assumption that it relies on.

| **Algorithm** | **Security** | | **Assumption** | | | | |
|---|---|---|---|---|---|---|---|
| | *Perfect* | *Computational* | *DDH* | *CDH* | *DL* | *Collision* | *No Assumption* |
| One-time-pad | ✗ | | | | | | ✗ |
| El-Gamal-Encryption | | | | | | | |
| El-Gamal Digital Signatures | | | | | | | |
| Pedersen-Commitment (Hiding) | | | | | | | |
| Pedersen-Commitment (Binding) | | | | | | | |
| Shamir's Secret Sharing | | | | | | | |

## Solution

### Reductions

**a)** Given $g^a, g^b, g^c$, we use CDH to compute $g^{ab}$ and decide whether $g^{ab} = g^c$.

### Commitment Schemes

**b)** Correctness: Given $m, c, r$, Bob can verify $c = g^m \cdot g^r \mod p$. Thus, the given commitment scheme is correct.

**c)** Commitment scheme is perfectly hiding: Given a commitment $c$ every message $m$ is equally likely to be the committed message to $c$. Given $m, r$ and any $m'$, there exists (a unique) $r'$ such that $g^m \cdot g^r = g^{m'} \cdot g^{r'} \mod p$. We just have to solve the linear equation $m+r = m'+r' \mod p-1$.

**d)** Commitment scheme is not binding (neither computationally nor perfectly): After Alice commits $c = g^m \cdot g^r$, she can easily reveal any message $m'$ by simply solving $m+r = m'+r' \mod p-1$.

### Perfect vs Computational Security

**e)**

| **Algorithm** | **Security** | | **Assumption** | | | | |
|---|---|---|---|---|---|---|---|
| | *Perfect* | *Computational* | *DDH* | *CDH* | *DL* | *Collision* | *No Assumption* |
| One-time-pad | ✗ | | | | | | ✗ |
| El-Gamal-Encryption | | ✗ | ✗ | | | | |
| El-Gamal Digital Signatures | | ✗ | | | ✗ | ✗ | |
| Pedersen-Commitment (Hiding) | ✗ | | | | | | ✗ |
| Pedersen-Commitment (Binding) | | ✗ | | | ✗ | | |
| Shamir's Secret Sharing | ✗ | | | | | | ✗ |

# 4 Data and Storage (18 points)

## Hash Functions

**a)** [3 points] For $u, m \geq 1$, consider the universe $U = \{0, \ldots, u-1\}$ of keys, the hash table $M = \{0, \ldots, m-1\}$ and hash functions $h_j(i) = i + j \mod m$ for $j = 0, \ldots, m-1$. Then $\mathcal{H} = \{h_j : j \in \{1, \ldots, m-1\}\}$ is a universal family of hash functions if and only if

- ○ $m$ is prime.
- ○ $u \leq m$.
- ○ $m$ is prime and $u \leq m$.
- ○ none of the above.

**b)** [3 points] Briefly justify your answer to **a)** (in 2-3 sentences).

## Covid Test Database

We consider the SQL database in which the health authorities of a country store all information about Covid-19 tests. The database contains the following tables (table keys are underlined):

person(<u>passport_nr</u>, name, postcode)
test(<u>id</u>, passport_nr, result, test_date)
area(<u>postcode</u>, name, population)

**c)** [4 points] Describe what the following SQL query returns:

```
SELECT COUNT(*)
FROM (
      SELECT DISTINCT passport_nr
      FROM test
      WHERE test.result = 'positive'
) AS pos
JOIN (
      SELECT DISTINCT passport_nr
      FROM test
      WHERE test.result = 'negative'
) AS neg ON pos.passport_nr = neg.passport_nr
```

**d)** `[2 points]` What does the query in **c)** return if the JOIN is replaced by a LEFT OUTER JOIN?

**e)** `[6 points]` Write a SQL query that outputs the names of the 10 areas with the highest number of tests.

## Solution

**a)** $u \leq m$

**b)** If $u \geq m + 1$, the keys 0 and $m$ will collide for all hash functions in $H$, i.e. have collision probability 1. If $u \leq m$, any pair of two keys will not collide for any hash function in $H$, i.e. have collision probability 0.

The condition of $m$ being prime is irrelevant to the problem.

**c)** The number of people who have been tested positive as well as negative.

**d)** The number of people who have been tested positive.

**e)**
```
SELECT area.name, count(*) as nr_tests
FROM area
JOIN person ON person.postcode = area.postcode
JOIN test ON test.passport_nr = person.passport_nr
GROUP BY area.postcode, area.name
ORDER BY nr_tests DESC
LIMIT 10;
```

# 5 Machine Learning (18 points)

## Decision Trees

Take a look at the data in Figure 1. You want to fit the given data using a decision tree.

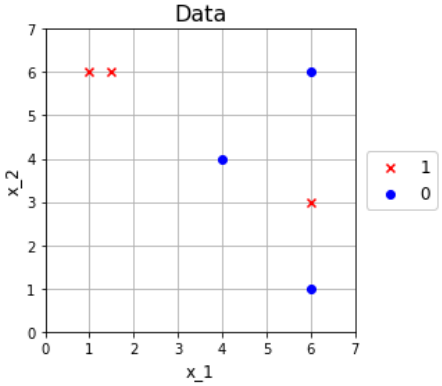



Figure 1: Your data.

In the lecture, we mentioned Gini Impurity. Here is the formal definition:

**Definition** (Classification splitting criterion: Gini Impurity)**.** *For node $v$ containing samples $D_v$ from $k$ classes, the gini measure of impurity is defined as:*

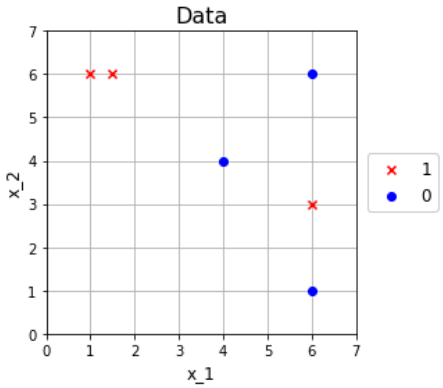
$$G = 1 - \sum_{i=1}^{k} p_i^2$$

*where*

$$p_i = \frac{\{\boldsymbol{x} \in D_v \mid f(\boldsymbol{x}) = i\}}{|D_v|}$$

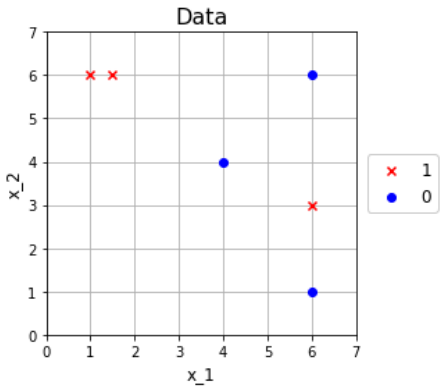
*is the fraction of samples within $D_v$ that belongs to class $i$.*

**a)** `[2 points]` Draw the decision boundary for the first split you would get using the CART loss with Gini impurity.

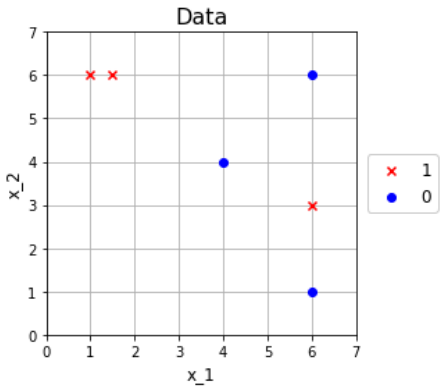


**b)** `[2 points]` Show the loss calculation for your split in part **a)**.

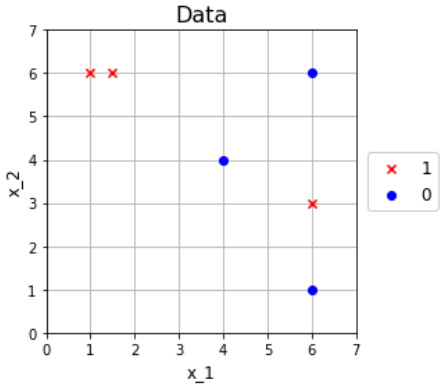
**c)** `[2 points]` Finish the CART-Gini decision tree from part **a)** by drawing the remaining decision boundaries (until each leaf is pure).



**d)** `[3 points]` What is the minimum depth of a decision tree that can fit the data perfectly? Draw an example decision tree. You may draw the tree or the decision boundaries.




**e)** `[2 points]` Add a point to the data such that the minimal depth required increases by 1. (Explain, or mark directly in the plot)



**f)** [5 points] Suppose you add some custom features to the dataset. Which of the following custom features allow for a single decision to separate all the data points? Select all features (if any) that apply and provide example decisions.

- ○ $x_3 = x_1 x_2$
- ○ $x_4 = (x_1 - 7)^2 + (x_2 - 3)^2$
- ○ $x_5 = x_1 + x_2$
- ○ $x_6 = \sin(\frac{\pi}{2}(x_2 - x_1))$

**g)** [2 points] Why might it be a bad idea to add any custom features?

## Solution

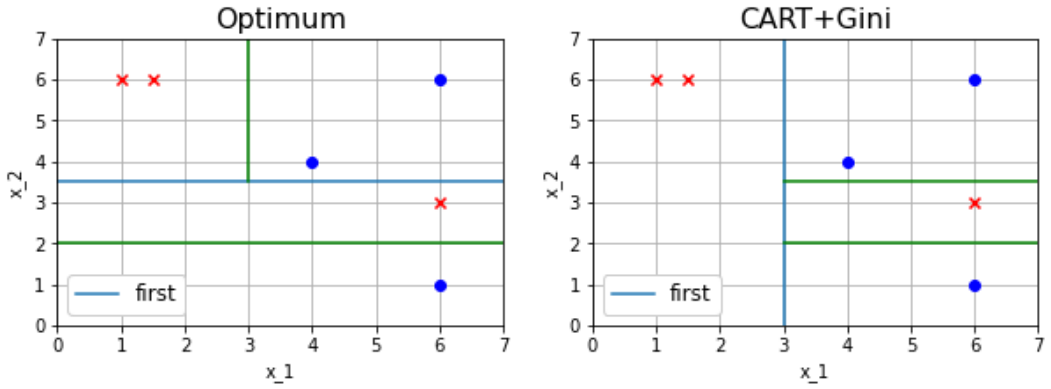



Figure 2: Minimum depth decision tree vs. Gini decision tree.

**a)** See Figure 2.

**b)** First split has loss

$$L = \frac{2}{6}\left(1 - 1^2\right) + \frac{4}{6}\left(1 - \left(\frac{3}{4}^2 + \frac{1}{4}^2\right)\right) = \frac{1}{4}.$$

**c)** See Figure 2.

**d)** Optimum depth = 2. See Figure 2.

**e)** See Figure 4 for an example solution. See Figure 3 for all possible solutions. Areas indicate possible solution areas by colour. Grey indicates no point in this area is a solution. Blue/red mean blue/red point respectively in this area is correct.

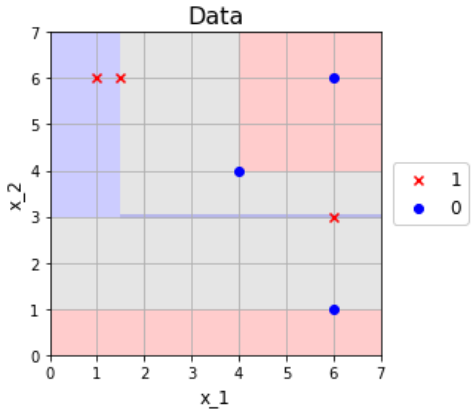



Figure 3: All possible solutions.

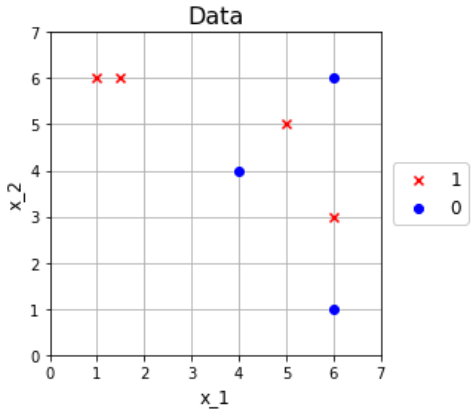



Figure 4: Possible Solution.

**f)** By observing the data, we can see that $x_4$ and $x_5$ will not work. The decision boundaries of $x_5$ are lines and we cannot separate the two classes with a single line; for $x_4$ they are circles and a single circle won't do it either. Similarly, for $x_3$ the decision boundaries are of the form $x_2 = \frac{c}{x_1}$, which will also not work here. You can only achieve a single split with $x_6$, e.g., $x_6 > 0.3$.

**g)** The decision tree you get will most likely not generalize to unseen data (unless there is a good underlying reason to add the given custom feature).

# 6 Neural Networks (17 points)

## Multiple Choice

**a)** [3 points] Which layers or architectures take advantage of weight sharing?

- ○ Convolutional Neural Network (CNN)
- ○ Long Short Term Memory (LSTM)
- ○ Multi-layer Perceptron (MLP)
- ○ Gated Recurrent Unit (GRU)
- ○ Attention
- ○ Linear Regressor

## A Simple Neural Network

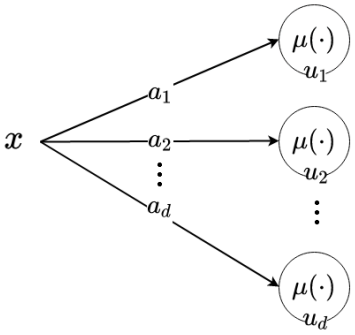



Figure 5: A simple feature extractor.

Consider the feature extractor displayed in Figure 5 with weights $a_1$ to $a_d$, an activation function $\mu$ on the nodes $u_i$ and a single-dimensional input $x$. More formally, the output at node $u_i$ is defined as $\hat{f}_i = \mu(a_i \cdot x)$. Note that there is no bias/intercept term ("$w_0$") in the feature extractor. Assume that each input $x \in X$ maps to a binary classification output $y \in \{0, 1\}$.

**b)** [6 points] Consider the following function on the outputs of our feature extractor:

$\hat{y} = \sigma(w_0 \cdot 1 + w_1 \cdot \hat{f}_1 + \cdots + w_n \cdot \hat{f}_n)$, with $\sigma(z) = \begin{cases} 1 & \text{for z} > 0 \\ 0 & \text{for z} \leq 0 \end{cases}$ (linear classifier).

Assume $\mu(z) = z$. What is the VC dimension of this model? Justify your answer.

We use the outputs $\hat{f}_i$ of our feature extractor as inputs for a wide and deep MLP. We denote $\omega(z) = ReLU(z) = max(z, 0)$ as the activation function for all nodes of the MLP.

**c)** `[5 points]` We fix the parameters of the feature extractor to some $a_i < 0$, for all $i$ in $\{1..d\}$ and assume $\mu(z) = ReLU(z) = max(z, 0)$. We are given six inputs with corresponding labels $(x, y)$: $(-6, 0), (-2, 1), (2, 0), (3, 1), (4, 1), (7, 1)$. What is the best accuracy achievable by training the MLP while not training the feature extractor? Justify your answer. (**Hint:** You do not need to find any explicit weights.)

**d)** `[3 points]` We decide to normalize the inputs $x_i$ to the range $[0, 1]$ and to train the weights $a_i$ of our feature extractor, initializing them with our previously used weights $a_i < 0$. What issue do you see arise during training?

# Solution

## Multiple Choice

**a)** CNN, LSTM, GRU, Attention

## A Simple Neural Network

**b)** The VC dimension is 2. Without non-linearities the whole model behaves like a single multiplicative factor plus bias, independent of $n$. $\hat{y} = \sigma(w_0 \cdot 1 + w_1 \cdot \mu(a_1 \cdot x) + \cdots + w_n \cdot \mu(a_n \cdot x) = \sigma(w_0 + x \cdot (a_1 \cdot w_1 + \cdots + a_n \cdot w_n))$ Given two datapoints $(x_1, y_1)$ and $(x_2, y_2)$, without loss of generality assume $x_1 < x_2$. We can always choose $(a_1 \cdot w_1 + \cdots + a_n \cdot w_n) = (y_2 - y_1)$ and $w_0 = (y_1 - 1) \cdot x_1 + (y_2 - 1) \cdot x_2$. However, for 3 points $x_1$, $x_2$ and $x_3$ with $x_1 < x_2 < x_3$ and $y_1 = y_3 = 1, y_2 = 0$ we cannot define weights that include $x_1$ and $x_3$ but not $x_2$.

**c)** $\frac{5}{6}$. If we look at the features generated by our feature extractor, we realize that all positive inputs get mapped to all zeros, as they are multiplied by negative weights and $ReLU(z) = 0$ for $z \leq 0$. The model therefore has no way to differentiate between positive inputs and its best bet is to assign the most common label to them. It can still correctly classify the negative inputs, leading to 5 out of the 6 points classified correctly.

**d)** As we now only have positive inputs, leading to negative pre-activations on the feature extractor neurons, all gradients acting on the weights $a_1$ to $a_n$ are 0 (vanishing gradients) ($\frac{dReLU(z)}{dz} = 0$ for $z \leq 0$). This in turn means the network is never able to recover and therefore can only correctly classify the most common label at best.

# 7 Computability (16 points)

## Turing Machines (TMs)

**a)** `[2 points]` A randomized TM can compute more than a Turing machine.

- ○ True.
- ○ False.

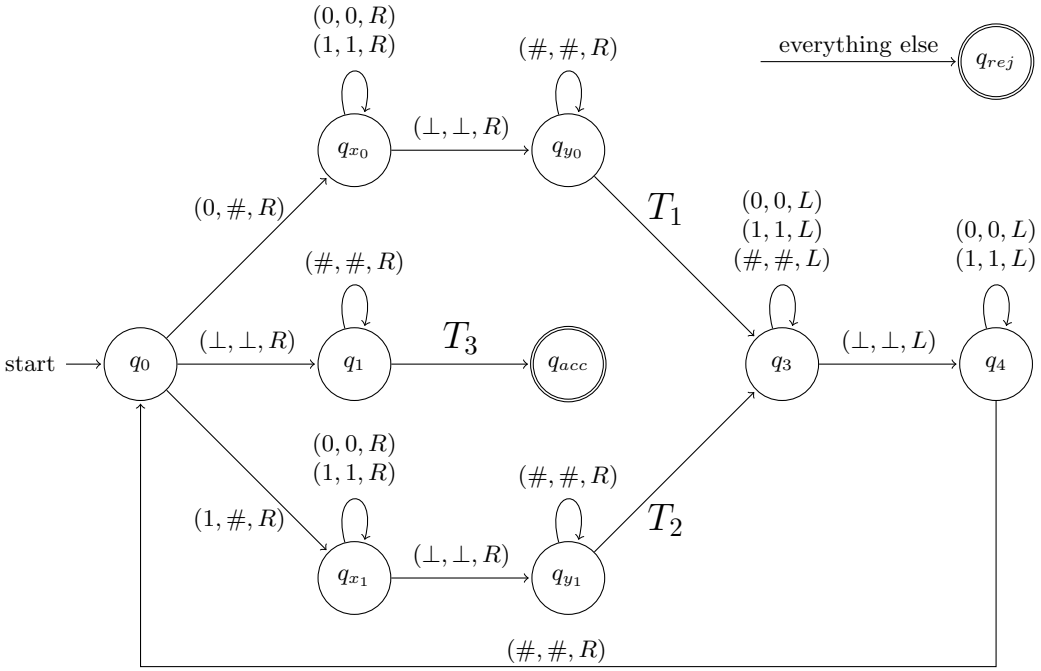
**b)** `[6 points]` Consider the TM with input $x \perp y \perp$ on its tape. We interpret the input as two integers $x, y \geq 1$ in binary format (most significant bit first and no leading zeros). Our goal is to design a TM that checks $x == y$ and enters the accept state $q_{acc}$ if $x$ and $y$ are equal, and the reject state $q_{rej}$ otherwise.

A transition tuple $T = (1, \perp, R)$ on an arrow from state $q_A$ to state $q_B$ means that, if the TM is in state $q_A$ and reads a 1 on the tape, it will write $\perp$ on the tape, move its pointer to the right ($R$), and go into state $q_B$. In case the TM reads a symbol for which no transition is defined, it automatically enters state $q_{rej}$.

The starting state is $q_0$ and the TM's pointer starts on the most significant bit of $x$. Further note that while $x, y$'s alphabet is restricted to $\{0, 1\}$, the tape's alphabet is $\{0, 1, \perp, \#\}$.

Complete the TM to check $x == y$ by filling in the missing transition tuples:

$T_1 =$ $T_2 =$ $T_3 =$



## Post Correspondence Problem (PCP)

**c)** [4 points] Consider the following domino set:

$$\left[\frac{bbca}{c}\right], \quad \left[\frac{d}{adc}\right], \quad \left[\frac{aca}{ac}\right], \quad \left[\frac{c}{acbb}\right], \quad \left[\frac{ab}{b}\right]$$

Is the corresponding PCP instance solvable? Explain your answer.

**d)** [4 points] Consider the following domino set:

$$\left[\frac{bbca}{c}\right], \quad \left[\frac{d}{adc}\right], \quad \left[\frac{aca}{ac}\right], \quad \left[\frac{c}{acbb}\right], \quad \left[\frac{bb}{b}\right]$$

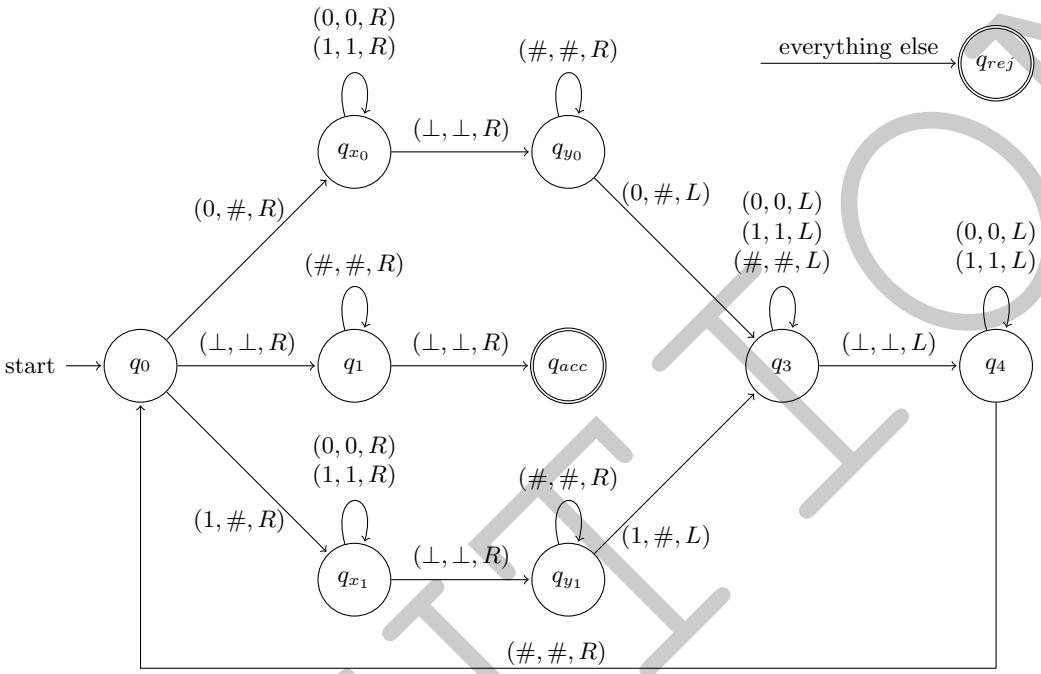
Is the corresponding PCP instance solvable? Explain your answer.

# Solution

## Turing Machines (TMs)

**a)** *False* – There is no known computation model that can compute more than a Turing machine.

**b)** A filled in Turing Machine is displayed below.




## Post Correspondence Problem (PCP)

**c)** *Not solvable* – The first domino is domino 3. This can be followed by domino 2 or 4.

If the second domino is domino 2, only domino 4 can follow. This can not be continued.

If the second domino is domino 4, then the lower sequence has an extra *bb*. This can only be followed by domino 1, which brings us back to the previous phase after the first domino. Thus, the PCP is not solvable.

**d)** *Solvable* – The following sequence is valid:

$$\left[\frac{aca}{ac}\right], \left[\frac{c}{acbb}\right], \left[\frac{bb}{b}\right], \left[\frac{bb}{b}\right]$$

Thus, the PCP is solvable.

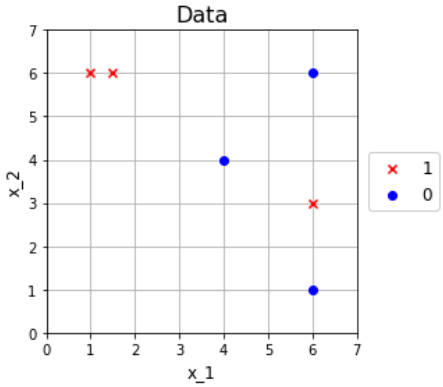
## Replacements

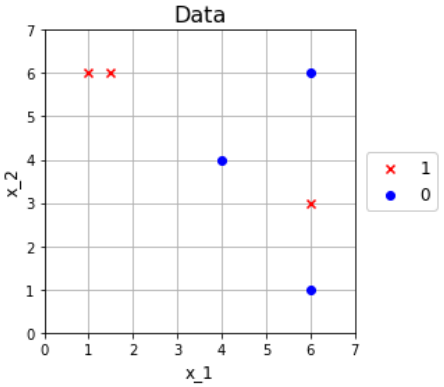
Here, you can find replacement templates for the fill in tasks, in case your original solution becomes too messy. If you use these, please **clearly** indicate which one we should consider.

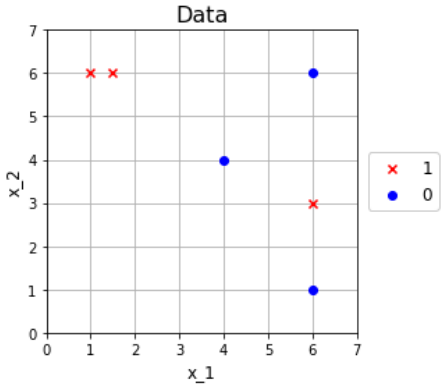
```python
def lcs(X, Y):
    if len(X) == 0 or len(Y) == 0:
        return 1
    elif X[-1] == Y[-1]:
        return 1 + lcs(X[:-1], Y[:-1])
    else:
        return min(lcs(X, Y[:-1]), lcs(X[:-1], Y))

# Define the two strings and call the function
X = "AGGTAB"
Y = "GXTXAYB"
print("Length of LCS is ", lcs(X, Y))
```

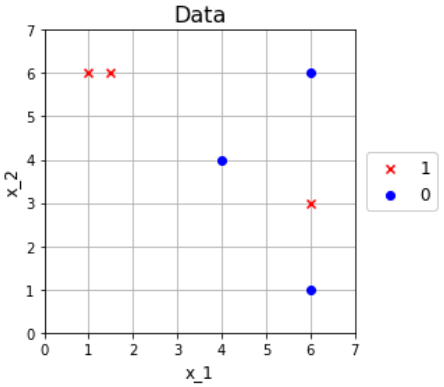
```python
def fastLCS(X , Y):
    # Find the length of the strings
    m = len(X)
    n = len(Y)
    # Fill in the rest of the lcs function

# Define the two sequences and call the functions
X = "AGGTAB"
Y = "GXTXAYB"
print("Length of LCS is ", fastLCS(X, Y))
```

| Algorithm | Security | | Assumption | | | | |
|---|---|---|---|---|---|---|---|
| | *Perfect* | *Computational* | *DDH* | *CDH* | *DL* | *Collision* | *No Assumption* |
| One-time-pad | ✗ | | | | | | ✗ |
| El-Gamal-Encryption | | | | | | | |
| El-Gamal Digital Signatures | | | | | | | |
| Pedersen-Commitment (Hiding) | | | | | | | |
| Pedersen-Commitment (Binding) | | | | | | | |
| Shamir's Secret Sharing | | | | | | | |



Use this page if you need extra space.