Name: Matriculation number:

↑ Please write **clearly** and use capitalized BLOCK LETTERS ↑

Chair of Algorithms
and Data Structures
Prof. Dr. Hannah Bast
Sebastian Walter

Information Retrieval / Databases and Information Systems WS 2023/2024

https://ad-wiki.informatik.uni-freiburg.de/teaching

Exam

Monday, 18th of March 2024, 14:00 - 16:30 h, in building 101

There are *five* tasks. Each task is worth 20 points, that is, you can reach a maximum of 100 points. You need 50 points to pass the exam.

You have 2 hours 30 minutes of time overall. This is 20 minutes plus 10 minutes buffer per task.

You are allowed to use *one* DIN A4 sheet of paper with any contents of your choice related to the lectures and the exercises. You can write on the front and on the back and it can be handwritten or a printout, but it must be from yourself, not from someone else. You are not allowed to use any computing devices or mobile phones, in particular nothing with which you can communicate with others or connect to the Internet or parallel universes.

You may write down your solutions in either English or German.

Important: For the programming tasks you should use Python. Each programming task specifies a maximum number of lines you are allowed to write. For all other tasks, do not simply write down the final result, but it should also be clear how you derived it.

Multiple-choice tasks: Some of the tasks consist of a set of questions, to each of which the answer is either *true* or *false*. A correct answer gives plus 2 points, a wrong answer gives minus 2 points, and no answer gives 0 points. So if you don't know the answer, do not guess but write *no answer* or simply nothing. No explanations are needed. Altogether, you cannot get less than 0 points for such tasks.

How and what to hand in: Please write, in the blue box at the top, your name and your matriculation number. Please write your solutions on the exam: use the front side of the sheet on which the question is printed and then the back side of the *previous* sheet. If you wish to submit additional paper (which should be the exception), write your name and matriculation number on every additional sheet.

Good luck!

Task 1 (Inverted Index and Ranking, 20 points)

1.1 (6 points) Consider the following collection of four documents $D_1, \ldots, D_4$. Write down the inverted index with *tf* scores. For which C and α does Zipf's Law ($F_n = C \cdot n^{-\alpha}$) hold exactly for this document collection?

D_1 : b c a c d c
D_2 : b c a c d c b c
D_3 : a d a c c b
D_4 : c a a c c

a: (D_1, 1), (D_2, 1), (D_3, 2), (D_4, 2)
b: (D_1, 1), (D_2, 2), (D_3, 1)
c: (D_1, 3), (D_2, 4), (D_3, 2), (D_4, 3)
d: (D_1, 1), (D_2, 1), (D_3, 1)

Word frequencies sorted: $F_1 = 12$, $F_2 = 6$, $F_3 = 4$, $F_4 = 3$, with $C = 12$ and $\alpha = 1$, this fits $F_n = C \cdot n^{-\alpha} = 12/n$.

1.2 (10 points) Write a function *merge(A: list[int], B: list[int])* $\rightarrow$ *list[int]* that merges two sorted lists of integers into a single sorted list of integers, but without removing duplicates. For example, *merge([1, 3, 4], [2, 3, 5]) = [1, 2, 3, 3, 4, 5]*. The function should run in time $O(|A| + |B|)$ and should have at most 15 lines.

```
def merge(A, B):
    i = j = 0
    res = []
    while i < len(A) or j < len(B):
        if j == len(B) or (i < len(A) and A[i] <= B[j]):
            res.append(A[i])
            i += 1
        else:
            res.append(B[j])
            j += 1
    return res
```

1.3 (4 points) Given the list of result IDs $[6, 7, 2, 4, 5]$ and the set of relevant IDs $\{1, 2, 3, 4\}$. Write down the following values: $P@2$, $P@R$, AP. For $\text{rel}_1 = 0, \text{rel}_2 = 0$, and $\text{rel}_3 = 4$, write down $DCG@3$.

1. $P@2 = 0/2 = 0$
2. $P@R = 2/4 = 1/2$
3. $AP = (1/3 + 2/4)/4 = (5/6)/4 = 5/24$
4. For $\text{rel}_1 = 0, \text{rel}_2 = 0$, and $\text{rel}_3 = 4$, $DCG@3 = \sum_{i=1}^{3} \text{rel}_i / \log_2(i + 1) = 4/\log_2(4) = 2$

Task 2 (Database Basics, 20 points)

2.1 (12 points) You are given the following database schema, where columns marked PK are part of a table's primary key and columns marked FK are foreign keys to other tables.

movies(id INT PK, title TEXT, release_year INT)
persons(id INT PK, name TEXT)
actors(movie_id INT FK(movies.id) PK, person_id INT FK(persons.id) PK)
streaming(movie_id INT FK(movies.id) PK, platform TEXT PK, views INT)

Write a corresponding SQL query for each of the following queries:

1. Titles of all movies released in the 20th century

SELECT title FROM movies WHERE release_year > 1900 AND release_year <= 2000;

2. Titles of the top 10 movies ordered by total number of views on "Netflix" and "Amazon Prime"

SELECT movies.title FROM movies, streaming WHERE movies.id = streaming.movie_id
AND (streaming.platform = "Netflix" OR streaming.platform = "Amazon Prime")
GROUP BY movies.id ORDER BY SUM(streaming.views) DESC LIMIT 10;

3. Title and comma-separated list of actors for all movies with at least 10 actors

SELECT movies.title, GROUP_CONCAT(persons.name, ',') FROM movies, actors, persons
WHERE movies.id = actors.movie_id AND actors.person_id = persons.id
GROUP BY movies.id HAVING COUNT(actors.person_id) >= 10;

2.2 (8 points) The two query plans below each execute the same query over two tables A and B. Compute the cost estimate for each plan if A has 10 rows and 4 columns and B has 20 rows and 2 columns. Give an example of a shape for A and B such that plan 2 is cheaper than plan 1.

Assume the following cost estimates: *select* $\rightarrow n \cdot k$, *project* $\rightarrow n \cdot k'$, and *cartesian-product* $\rightarrow n_1 \cdot n_2 \cdot (k_1 + k_2)$, where n, n_1, n_2 and k, k_1, k_2 refer to the number of rows and columns of the respective input tables and k' refers to the number of output columns. Assume that *select* selects half of the rows. It doesn't matter what exactly ϕ_1 and ϕ_2 are.

Query plan 1:

1. $A' = \text{select}(A, \phi_1)$
2. $B' = \text{select}(B, \phi_2)$
3. $C = \text{cartesian-product}(A', B')$
4. $R = \text{project}(C, \text{all cols of } C, \text{true})$

1. $\text{cost} = 10 \cdot 4, \text{rows} = 5, \text{cols} = 4$
2. $\text{cost} = 20 \cdot 2, \text{rows} = 10, \text{cols} = 2$
3. $\text{cost} = 5 \cdot 10 \cdot (4+2), \text{rows} = 50, \text{cols} = 6$
4. $\text{cost} = 50 \cdot 6$

Total cost = 40 + 40 + 300 + 300 = 680

Query plan 2:

1. $C = \text{cartesian-product}(A, B)$
2. $C' = \text{select}(C, \phi_1)$
3. $C'' = \text{select}(C', \phi_2)$
4. $R = \text{project}(C'', \text{all cols of } C'', \text{true})$

1. $\text{cost} = 10 \cdot 20 \cdot (4+2), \text{rows} = 200, \text{cols} = 6$
2. $\text{cost} = 200 \cdot 6, \text{rows} = 100, \text{cols} = 6$
3. $\text{cost} = 100 \cdot 6, \text{rows} = 50, \text{cols} = 6$
4. $\text{cost} = 50 \cdot 6$

Total cost = 1200 + 1200 + 600 + 300 = 3300

If one of A or B is empty and the other is not, plan 2 has estimated cost 0, and plan 1 has not.

Task 3 (Advanced SQL, SPARQL, and Knowledge Graphs, 20 points)

3.1 (7 points) The following SQL query is the result of transforming a SPARQL query to SQL for a database that stores all the triples in one table *triples*. Write down a SPARQL query that leads to this SQL query. Write down in natural language what this query computes.

```
SELECT t3.subject, t3.object
FROM triples AS t1, triples AS t2, triples AS t3
WHERE t1.predicate = "has_mother" AND
      t2.predicate = "has_father" AND
      t1.subject = t2.subject AND
      t3.predicate = "married_to" AND
      t3.subject = t1.object AND
      t3.object = t2.object;
```

```
SELECT ?mother ?father WHERE {
  ?child has_mother ?mother .   # t1
  ?child has_father ?father .   # t2
  ?mother married_to ?father .  # t3
}
```

All pairs of mother and father
who have a child together
and who are married to each other

3.2 (6 points) For each of the following queries, write down whether it computes the three Oscars that Meryl Streep has won, together with the corresponding movie. For each query write either *true* or *false*. No explanation is needed. A correct answer gives plus 2 points, a wrong answer gives minus 2 points. If you don't know the answer, write *no answer* or nothing. You cannot get less than 0 points for this task. Hints: *SELECT ** selects all variables from the query, *P166* is *won-award*, *Q873* is *Meryl Streep*, *P1686* is *for which film*, and *P31 Q19020* means *is an Oscar*.

```
SELECT * WHERE {
  wd:Q873 wdt:P166 ?a .
  ?a wdt:P1686 ?b .
  ?a wdt:P31 wd:Q19020 .
}
```

False

```
SELECT * WHERE {
  wd:Q873 p:P166 ?a .
  ?a ps:P166 ?b .
  ?a pq:P1686 ?b .
  ?a wdt:P31 wd:Q19020 .
}
```

False

```
SELECT * WHERE {
  wd:Q873 p:P166 ?a .
  ?a ps:P166 ?b .
  ?a pq:P1686 ?c .
  ?b wdt:P31 wd:Q19020 .
}
```

True

3.3 (7 points) Manually compute the result of the SQL query given on the left, using the table *movies* given on the right. Write down in natural language what this query computes.

```
SELECT year, title, score
FROM movies NATURAL JOIN (
  SELECT year, (MAX(score) AS score)
  FROM movies
  GROUP BY year
)
ORDER BY score DESC;
```

title	score	year
Spider-Man	8.3	2021
Everything Everywhere	7.8	2022
Oppenheimer	8.4	2023
Don't Look Up	7.2	2021
Dune	8.0	2021
Top Gun	8.2	2022

year	title	score
2023	Oppenheimer	8.4
2021	Spider-Man	8.3
2022	Top Gun	8.2

For each year, the movie with the highest score; order descending by scores.

Task 4 (Fuzzy Search, Q-Gram Index and Web Applications, 20 points)

4.1 (5 points) For the query prefix $x = bree$ and the words $y_1 = free$, $y_2 = dreisam$ and $y_3 = spree$, write down the left-padded 3-grams for the 3-gram index. For each word y_i with $i \in \{1, 2, 3\}$, determine how many 3-grams it has in common with the query prefix. Then for each y_i, say whether $\text{PED}(x, y_i)$ needs to be computed explicitly in order to find out if $\text{PED}(x, y_i) \leq 1$ or whether it is enough to look at the number of common 3-grams.

$|Q_3(x) \cap Q_3(y_i)| \geq |Q_3(x)| - 3 \cdot 1 = 1$

bree: $$b $br bre ree

free: $$f $fr fre **ree** $\rightarrow |Q_3(x) \cap Q_3(y_1)| = 1 \geq 1 \rightarrow$ need to compute PED

dreisam: $$d $dr dre rei eis isa sam $\rightarrow |Q_3(x) \cap Q_3(y_2)| = 0 \ngeq 1 \rightarrow$ no need to compute PED

spree: $$s $sp spr pre **ree** $\rightarrow |Q_3(x) \cap Q_3(y_3)| = 1 \geq 1 \rightarrow$ need to compute PED

4.2 (8 points) For each of the following statements, say whether it is *true* or *false*. No explanation is needed. A correct answer gives plus 2 points, a wrong answer gives minus 2 points. If you don't know the answer, write *no answer* or nothing. You cannot get less than 0 points for this task. The "algorithm for fuzzy prefix search" refers to the algorithm from the respective lecture.

1. For all strings x and y, it holds that $\text{PED}(x, y) \leq ED(x, y)$. True, because the PED is defined as the minimum over the ED of x and all prefixes of y (and y itself is a prefix of y).

2. For all strings x and y, it holds that $\text{PED}(x, y) \geq abs(|x| - |y|)$. False, because we're looking at prefixes of y. E.g. for $x = br$ and $y = breisgau$: $\text{PED}(x, y) = 0$

3. Without any padding, the algorithm for fuzzy prefix search is still correct but less efficient. True. With strings x and y and their padded versions x', y': $\text{PED}(x', y') = \text{PED}(x, y)$ but due to more q-grams, the PED needs to be computed less often.

4. With padding on both sides, the algorithm for fuzzy prefix search is still correct. False. With $q = 3$, consider double-padded strings $x' = \$\$br\$\$$ and $y' = \$\$breisgau\$\$$ and their left-padded versions x'', y'': $\text{PED}(x', y') = 2 \neq 0 = \text{PED}(x'', y'')$

4.3 (7 points) Write a simple HTML page with heading *Search*, an input field and a button, which upon a click of the button, sends the contents of the input field back to the server. Use forms, not JavaScript. Given your HTML, the input text *example*, and assuming that the page is accessed via *localhost:8000/search.html* how does the first line of the sent request look like when clicking the button?

```
<html><body><h1>Search</h1>
  <form>
    <input type="text" value="xyz" name="query">
    <input type="submit" value="Search">
  </form>
</body></html>
```

First line of request: GET /search.html?query=example HTTP/1.1

Task 5 (Word Embeddings, Logistic Regression, Language Models, 20 points)

5.1 (7 points) Write a function *cos_sim(emb1: list[float], emb2: list[float])* $\rightarrow$ *float* that computes the cosine similarity between two given word embeddings *emb1* and *emb2*. You can assume that the embeddings have the same dimension. You can use basic arithmetic operations and functions from the *math* module, like *math.sqrt*, but you should not rely on the module *torch*. Your function should have at most 10 lines.

```
def cos_sim(emb1: list[float], emb2: list[float]) -> float:
    sum_squares_emb1 = sum_squares_emb2 = sum_products = 0
    for i in range(len(emb1)):
        sum_squares_emb1 += emb1[i] * emb1[i]
        sum_squares_emb2 += emb2[i] * emb2[i]
        sum_products += emb1[i] * emb2[i]
    return sum_products / math.sqrt(sum_squares_emb1 * sum_squares_emb2)
```

5.2 (8 points) Write a function *epoch(X: tensor, y: tensor, w: tensor)* $\rightarrow$ *tensor* that does one epoch of logistic regression, where *tensor* is an abbreviation for *torch.Tensor*. The arguments are as follows: X is a 2D tensor with input vectors as rows, y is a 1D tensor with the input labels, w is a 1D tensor with the current weights, and the function should return a 1D tensor with the new weights after the epoch. You can assume that all values of all tensors involved are of type *float* and that the bias is already part of X and w. You should take a batch size of 1 and a learning rate of 0.1. Your function should have at most 10 lines.

```
epoch(X: tensor, y: tensor, w: tensor) -> tensor:
    for i in range(X.shape(0)):
        xi = X[i]  # The i-th input vector, a 1D tensor
        yi = y[i]  # The i-th label, a single float
        factor = 0.1 * (yi - torch.sigmoid(torch.dot(xi, w))  # a float
        w = w + factor * xi
    return w
```

5.3 (5 points) Express softmax$(x, 0)$ in terms of $x \in \mathbb{R}$ and the sigmoid function σ. Do not just write down the result, but also how you got there.

1. $\text{softmax}(x, 0) = (e^x, e^0)/(e^x + e^0)$
2. The first component is $e^x/(e^x + 1) = 1/(1 + e^{-x}) = \sigma(x)$
3. The second component must then be $1 - \sigma(x) = \sigma(-x)$
4. Hence $\text{softmax}(x, 0) = (\sigma(x), \sigma(-x))$