Probeklausur
Informatische Werkzeuge in den Geistes- und Sozialwissenschaften II

16. Juli 2020

The „Lösungen“ der Aufgaben in diesem Dokument sollen den Studierenden als Anfangspunkt für die Beantwortung der Aufgaben dienen. Trotz aller Bemühungen kann es zu unvollständigen oder sogar Fehlern in den „Lösungen“ kommen. Da die Korrektur und Benotung der gestellten Aufgaben niemals lediglich auf einen „Vergleich mit der Musterlösung“ hinausläuft, ist dies auch nicht so schlimm. In jedem Fall sollten die Studierenden die Lösungen nachvollziehen und im Prinzip mittels des Lehrstoffs selbst verifizieren können.

Sollten Sie „Lösungen“ finden, die Sie nicht verstehen oder sogar für fehlerhaft halten diskutieren Sie diese am besten mit den Tutoren oder auf dem Kursforum und benachrichtigen Sie die Lehrenden.

1 Versionskontrolle

Aufgabe 1.1 (Dezentrale Versionskontrolle)

Erklären Sie kurz (!) das Konzept von dezentralen Versionskontrollsystemen. Können `git`-Repositorien auch ohne einen Server wie `GitHub` oder `GitLab` betrieben werden? 4 min

Lösung:In dezentraler Versionskontrolle gibt es nicht eine zentrale Version, auf der alle BenutzerInnen arbeiten. Stattdessen verwalten alle BenutzerInnen ein eigenes, lokales Repositorium, welches dann bei Bedarf mit einem globalen System synchronisiert werden kann.

Deshalb ist das Verwenden von `git` auch ohne `GitHub` o.ä. möglich. Man kann problemlos nur den lokalen Teil verwenden und niemals pushen oder pullen.

Aufgabe 1.2 (git Beispielablauf)

Betrachten Sie folgende beispielhafte Zeitlinie. Markieren Sie jede Zeile jeweils mit “Kein Problem” oder “Problem”, falls bei der Ausführung eine Fehlermeldung auftreten sollte. 4 min

Falls ein Problem auftritt, beschreiben Sie es und geben Sie den Lösungsweg an.

1. Person A klont das Repositorium R,
2. Person B klont auch R,

3. Person A verändert eine Datei `foo.txt` in Zeile 10,
4. Person A führt einen `git add foo.txt` und `git commit` aus,
5. Person B verändert die Datei `foo.txt` in Zeile 10,
6. Person B führt `git add foo.txt`, `git commit` und `git push` aus,
7. Person A führt `git push` aus,
8. Person A verändert die Datei `foo.txt` in Zeile 23,
9. Person A führt `git push` aus.

Lösung:Probleme treten auf in Schritt 7. Person As Stand ist hier nicht mehr aktuell, verglichen mit der globalen Kopie. Deshalb muss Person A hier erst `pull`en. Dabei wird ein Konflikt auftreten, weil Person B an die selben Zeile geändert hat. Nachdem der Konflikt behoben ist, kann Person A einen Merge-Commit vornehmen und dann `push`en.

In Schritt 9 tritt ein weiteres Problem auf. Person A muss hier erst `add`en und `commit`ten, bevor ein `push` möglich ist.

Aufgabe 1.3 (Definition von pull)

Was ist gemeint, wenn im Kontext von Versionskontrolle von "`pull = fetch + merge`" gesprochen wird? 4 min

---

Lösung:Kurz gesagt: `git pull` führt ein `git fetch` und anschließend ein `git merge` aus.

Ein `git fetch` kann man zu jeder Zeit machen, um die lokalen Äste (Branches) zu aktualisieren und mit den entfernten Ästen (z.B. `origin`, `remote`) abzugleichen. Diese Operation ändert dabei keine lokalen Äste, sie zeigt lediglich an, wenn Änderungen verfügbar sind. Man kann `git fetch` also bedenkenlos ausführen, ohne die lokale Arbeitskopie zu ändern.

Ein `git pull` führt ein `git fetch` und anschließend ein `git merge` aus. `git pull` wird dabei immer in den aktuellen Ast der Arbeitskopie mergen. Man wählt sich also den Ast aus, von dem man pullen möchte und pullt diesen (überführt dessen Änderungen) in den aktuellen Ast.

Ein `git pull` ist das, was man tun muss, um einen lokalen Ast auf den Stand des dazugehörigen entfernten Ast zu bringen.

(Erklärung von `https://mixable.blog/git-unterschied-zwischen-git-pull-und-git-fetch/`)

---

2 Datenbanken

Aufgabe 2.1 (Datenbanktabelle)

Sie möchten eine Datenbanktabelle für die Mitarbeitenden der Capitalist Corporation anlegen, die `PersonalID`, `Name`, `Vorname`, `Jobtitel` und `Jahresgehalt` erfasst. Der Name der Tabelle soll "Angestellte" sein. Die `PersonalID` ist eine ganze Zahl und ein Primärschlüssel. 4 min

Geben Sie den entsprechenden `SQL`-Befehl an.

Lösung:

```
CREATE TABLE Angestellte (PersonalID INTEGER PRIMARY KEY,
                          Name TEXT,
  Vorname TEXT,
  Jobtitel TEXT,
  Jahresgehalt INTEGER);
```

Aufgabe 2.2 (Gleiches Gehalt)

Das Jahresgehalt ergibt sich in der Capitalist Corporation ausschließlich durch den Jobtitel, das heißt alle Mitarbeitenden mit dem selben Titel verdienen auch das selbe Gehalt. 4 min

1. Was ist das Problem bei der Tabelle aus Aufgabe 2.1?
2. Was wäre in diesem Fall die effizientere Weise, die Daten der Mitarbeitenden zu speichern?
3. Welche Vorteile hätte die Alternative?

Sie müssen die Alternative nicht implementieren, nur beschreiben.

Lösung:

1. Die Tabelle aus dem Problem oben führt zu redundant gespeicherten Daten.
2. Eine Alternative ist es, Jobtitel und Jahresgehalt in einer separaten Tabelle zu speichern und dann in der Mitarbeiter-Tabelle auf den entsprechenden Eintrag dort zu verweisen – mit einem Fremdschlüssel (Foreign Key).
3. Das spart Speicherplatz und macht die Wartung einfacher. Wenn eine Gehaltserhöhung ansteht, muß nur ein Eintrag angepasst werden.

Aufgabe 2.3 (Nils Nichtig Heiratet Nelly Null)

Der Mitarbeiter Nils Nichtig (PersonalID 1729) heiratet Nelly Null und nimmt ihren Nachnamen an. Wie könnte dies zu Problemen führen, wenn der entsprechende Eintrag in der Datenbank geändert wird? 3 min

Lösung:Null ist ein Schlüsselwort in `SQL`. Hier muss aufgepasst werden, dass wir dem Nachnamen nicht Null zuweisen (was bedeuten würde, dass kein Wert vorhanden ist), sondern den String "`Null`".

Aufgabe 2.4 (Capitalist Corporation Ändert den Namen)

Geben Sie den *korrekten* `SQL`-Befehl an, der den Nachnamen des Mitarbeiters Nils Nichtig aus Aufgabe 2.3 ändert, nachdem er Nelly Null geheiratet hat. 2 min

Lösung:

```
UPDATE Angestellte SET Nachname = ‘‘Null’’  WHERE PersonalID = 1729;
```

Aufgabe 2.5 (Noch gleicheres Gehalt)

Capitalist Corporation wird umstrukturiert zu Community Collective. Alle Mitarbeitenden sollen ab sofort das gleiche Gehalt bekommen. Schreiben Sie ein python-Programm was alle Gehälter (und nur die Gehälter) abfragt, aufsummiert, durch die Anzahl der Mitarbeitenden teilt und allen dieses Gehalt gibt. 10 min

Lösung:Hier ist eine mögliche Implementation:

```
import sqlite3

db = sqlite3.connect("database.db")

cursor = db.cursor()
cursor.execute("SELECT Jahresgehalt FROM Angestellte")

gehaelter = cursor.fetchall()
anzahl = len(gehaelter)
summe = 0

for g in gehaelter :
    summe += int(g[0])

durchschnitt = summe / anzahl

cursor.execute("UPDATE Angestellte SET Jahresgehalt=?", (durchschnitt,))

db.commit()
db.close()
```

3 Bilder und Bildmanipulation

Aufgabe 3.1 (Beatrices Beispiel-App)

Beatrice Beispiel hat eine App programmiert, mit der BenutzerInnen Bilder zeichnen können und diese mit ihren Freundinnen und Freunden teilen können. Diese wiederum können die Bilder wieder verändern und zurückschicken und so weiter. Die Bilder werden im JPEG-Format verschickt. 4 min

Nach kurzer Zeit bemerken BenutzerInnen seltsame Artefakte in den Bildern. Was ist das Problem?

Lösung:JPEG ist keine verlustfreie Komprimierung. Wiederholtes Öffnen und Speichern der Datei führt deshalb zu Fehlern, die sich jedes mal aufaddieren und damit immer schlimmer werden.

Aufgabe 3.2 (Raster- und Vektorgrafiken)

Erklären Sie kurz den Unterschied zwischen Vektor- und Rastergrafiken. Warum macht das Konzept von "Auflösung" nur bei einem der beiden Sinn? 4 min

Lösung:Eine *Raster*grafik speichert Informationen in einem großen Gitter (oder Raster) von einzelnen Pixeln. Die Dimensionen dieses Gitters ergeben die Auflösung.

Eine Vektorgrafik hingegen speichert keine Pixel, sondern Informationen über geometrische Formen (z.B. Zentrum, Radius und Strichdicke eines Kreises, Start und Ziel und Breite einer Linie, ...), die zusammen das Bild ergeben. Hier gibt es keine eindeutige Auflösung, weil wir diese Formen beliebig skalieren (zoomen) können.

Aufgabe 3.3 (Kantenerkennung)

Beschreiben Sie kurz in eigenen Worten, wie Kantenerkennung auf Bildern funktioniert. 5 min

Lösung:In einer Rastergrafik können Kanten erkannt werden, indem der Farbwert eines Pixels mit den Farbwerten der Pixel in seiner *Nachbarschaft* (also die angrenzenden Pixel in alle Richtungen) verglichen wird.

Ist der Wert gleich oder unter einem gewissen (arbiträren) Schwellenwert, so liegt keine Kante vor. Ist der Wert stark unterschiedlich, wurde möglicherweise eine Kante entdeckt.

Anfang und Ende der Kante können gefunden werden, wenn man ganze Nachbarschaften von Pixeln vergleicht und sieht, wo Unterschiede anfangen und enden.

Aufgabe 3.4 (Kantenerkennung mit Sobel-Filter)

Schreiben Sie eine python-Funktion edge_detection, die ein Bild `img` und eine Zahl `threshold` als Parameter nimmt. 12 min

Ihre Funktion soll einen Sobel-Filter (wie in der Vorlesung) implementieren und ein neues Bild zurückgeben, in welchem für jeden Pixel gespeichert ist, ob es Teil einer Kante ist oder nicht. Somit sollte in jedem Pixel, durch den eine Kante geht, der Wert 255 gespeichert sein und in jedem anderen Pixel der Wert 0.

Verwenden Sie *nicht* die `Pillow`-Funktion img.filter. Folgendes Codegerüst ist gegeben:

```
from PIL import Image

def edge_detection(img, threshold) :
    # Neues Bild erstellen. "L" bedeutet grayscale.
    result = Image.new("L", (img.width, img.height), 0)

    # Ihr Code hier...

    return result
```

Hinweis: Ein paar nützliche Informationen finden Sie im Folgenden:

- Sowohl der Parameter img als auch die Rückgabe Ihrer Funktion sind Pillow-Bilder
- Beide Bilder sind Graustufen-Bilder, haben also nur einen Kanal
- Sie können den Wert eines Pixels wie folgt auslesen:
  value = img.getpixel((x, y)) # Nur ein Wert pro Pixel, denn das Bild ist in Graustufen
- Sie können den Wert eines Pixels wie folgt setzen:
  img.putpixel((x, y), value)
- Der Sobel-Filter hat folgende Gewichte:

für horizontale Kanten	für vertikale Kanten
$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

- Da der Filter auf Nachbar-Pixel zugreift, müssen Sie beachten, dass Sie nicht auf Pixel außerhalb des Bildes zugreifen. Führen Sie dazu den Filter und damit die Kanten-Entscheidung nur auf Pixeln durch, die nicht am Rand des Bildes liegen.

Lösung:Hier ist eine mögliche Implementation:

```
from PIL import Image, ImageOps

def edge_detection(img, threshold) :
    # Neues Bild erstellen. "L" bedeutet grayscale.
    result = Image.new("L", (img.width, img.height), 0)

    # Keine Kanten am Bildrand
    for y in range(1, img.height - 1) :
        for x in range(1, img.width - 1) :
            tl = img.getpixel((x-1, y-1))
            t = img.getpixel((x, y-1))
            tr = img.getpixel((x+1, y-1))

            l = img.getpixel((x-1, y))
            r = img.getpixel((x+1, y))

            bl = img.getpixel((x-1, y+1))
            b = img.getpixel((x, y+1))
            br = img.getpixel((x+1, y+1))

            horizontal = abs((bl - tl) + (br - tr) + 2 * (b - t))
            vertical = abs((tr - tl) + (br - bl) + 2 * (r - l))

            if horizontal >= threshold or vertical >= threshold :
                result.putpixel((x, y), 255)
            else :
                result.putpixel((x, y), 0)

    return result

img = Image.open("test.jpg")
sobeled = edge_detection(ImageOps.grayscale(img), 200)
sobeled.save("sobel.jpg")
```