TECHNISCHE UNIVERSITÄT BERLIN

Fakultät IV – Elektrotechnik und Informatik
Fachgebiet Neurotechnologie (MAR 4-3)
Prof. Dr. Benjamin Blankertz
Röhr / Stahl

Algorithmen und Datenstrukturen, SoSe 18
Klausur am 08.08.2018

Bitte füllen Sie alle folgenden Felder aus:

Klausur-ID:	**666A**
tubIT-login:	
Vorname:	
Nachname:	
Matrikelnummer:	
Studiengang:	
Hochschule:	

Durch meine Unterschrift bestätige ich die Korrektheit obiger Angaben.

Ort, Datum — Unterschrift

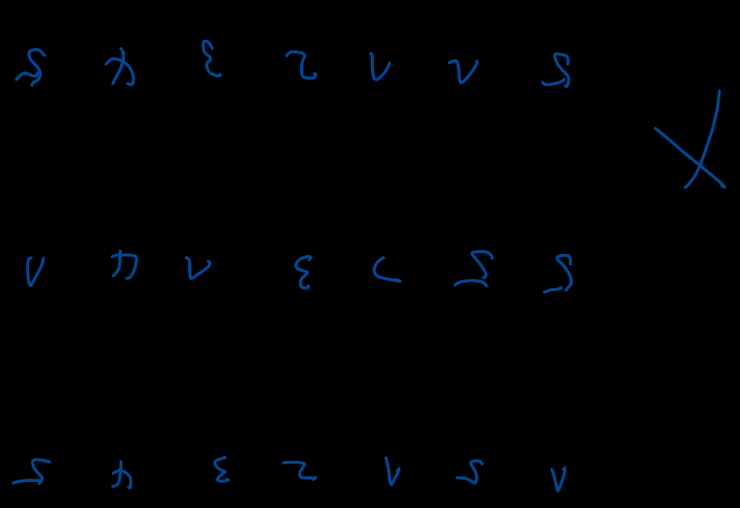
Beachten Sie die folgenden Hinweise!

- Die Klausuren sind nummeriert und werden anonymisiert korrigiert. Bitte schreiben Sie Ihren Namen **nur** auf das Deckblatt.
- Insgesamt können in der Klausur **100 Punkte** erreicht werden.
- Diese Klausur besteht mit diesem Deckblatt aus den (nummerierten) Seiten **1-14**.
- Schreiben Sie **nicht** mit roter Farbe, grüner Farbe oder Bleistift. Diese Lösungen werden nicht bewertet!
- Notieren Sie Ihre Antworten nur auf dem Blatt (oder dessen Rückseite), auf dem auch die zugehörige Aufgabe steht, da die Aufgaben getrennt korrigiert werden.
- Geben Sie nur eine Lösung pro Aufgabe ab, streichen Sie alle alternativen Lösungsansätze auf Schmier-/Notitzblättern durch.
- Bitte den Barcode am Ende der Seiten nicht beschädigen oder überschreiben.

Zusatzblätter No.:	

找边数最短

找路径最短



Aufgabe 3: Laufzeit ($4 + 6 = 10$ Punkte)

Eine Variante zum Finden minimaler Spannbäume (die nicht in der Vorlesung besprochen wurde) beruht auf der folgenden Eigenschaft:

> Sei ein beliebiger Zyklus in einem Graphen gegeben, und sei e eine der darin enthaltenen Kanten mit dem höchsten Gewicht. Dann gibt es einen minimalen Spannbaum, der die Kante e nicht enthält.

Der Pseudocode für diesen MST-Algorithmus sieht wie folgt aus:

```
1  // gegeben ein gewichteter Graph G = (V, E)
2   sortiere  die Kanten absteigend nach ihrem Gewicht
3  for jede Kante e ∈ E in dieser Reihenfolge:
4    if e ist Teil eines Zyklus von G:
5      entferne e aus G
6    end
7  end
8  return G   // G ist der MST
```

(a) Als Teil des Algorithmus muss für eine gegebene Kante e geprüft werden, ob e Teil eines Zyklus in $\boldsymbol{G}$ ist. Beschreiben Sie einen Algorithmus, der dies in linearer Zeit erledigt. Dabei können Sie Verfahren benutzen, die in der Vorlesung besprochen wurden.

Sei $G = (V, E)$, $e = (v, w)$

e ist eine Teil eines Graph, genau wenn in $G' = (V, E \setminus \{e\})$ ein Weg von v nach w existiert.

Dann wir nutzen dfs in G' mit startknote v, zu finden, ob w erreichbar ist

Laufzeit $O(V + E)$

(b) Wie ist die Laufzeit des gegebenen Algorithmus? Die Laufzeit muss für jeden Teil des Pseudocodes angegeben und daraus die Gesamtlaufzeit folgerichtig abgeleitet werden.

Z3: $O(E)$

Z4: $O(V + E)$

Z5: $O(1)$

insgesamt: $O(E(V + E + 1)) = O(|E| \cdot |V| + |E|^2)$

Aufgabe 4: Objektorientierte Programmmierung (4+4+2+1 = 11 Punkte)

Im Folgenden ist unvollständiger Code vorgegeben.

(a) Vervollständigen Sie den Code (an den unterstrichenen Stellen) so, dass er kompiliert und dass die Klassen `Car` und `Bicycle` von der Klasse `Vehicle` erben.

```
1  abstract class Vehicle {
2     protected int numberOfWheels;
3
4     abstract void refillAir(int num);
5
6     public Vehicle(int numberOfWheels) {
7        this.numberOfWheels = numberOfWheels;
8     }
9
10    public void repair() {
11       for (int i = 0; i < numberOfWheels; i++) {
12          refillAir(i);
13       }
14    };
15 }
```

```
1  class Car extends Vehicle {
2     public Car() {
3        super(4);
4     }
5
6     public void refillAir(int num)
7     {
8        System.out.print("c"+num);
9     }
10 }
```

```
1  class Bicycle extends Vihicle {
2     public Bicycle() {
3        super(2);
4     }
5
6     public void refillAir(int num)
7     {
8        System.out.print("b"+num);
9     }
10 }
```


(b) Betrachten Sie nun die folgende Methode:

```
1  public class Garage {
2     public static void main(String[] args) {
3        Vehicle [] allVehicles = new Vehicle[3];
4        Vehicle [] vehicles = new Vehicle[3];
5        Car c1 = new Car();
6        allVehicles[0]=c1;
7        Bicycle b1 = new Bicycle();
8        allVehicles[1]=b1;
9        Bicycle b2 = new Bicycle();
10       allVehicles[2]=b2;
11
12       try{
13          for (Vehicle v : allVehicles)
14             v.repair();
15          }
16       catch(Exception e){
17          System.out.println(e);
18          }
19    }
20 }
```

Was wird bei der Ausführung auf der Konsole ausgegeben?

C0C1C2C3b0b1b0b1

(c) Betrachten Sie weiterhin die Klasse `Garage`. Wenn Sie den try-catch-Block (Zeile 12-18) mit dem folgenden ersetzen, was gibt das Programm aus?

```
1        try{
2           for (int i=0; i<=3; i++)
3              allVehicles[i].repair();
4           }
5        catch(Exception e){
6            System.out.println(e);
7           }
```

java.lang.ArrayIndexOutOfBoundsException: 3

(d) Betrachten Sie weiterhin die Klasse `Garage`. Wenn Sie den try-catch-Block (Zeile 12-18) mit dem folgenden ersetzen, was gibt das Programm nun aus?

```
1        try{
2           for (int i=0; i<3; i++)
3              vehicles[i].repair();
4           }
5        catch(Exception e){
6             System.out.println(e);
7           }
```

java.lang.NullPointerException

10, 20, 30, 40, 50, 100

$X < 110$

$Y < 60$

$Z < 60$

EJICFGBA

GCHIDJEF

ABGCH
ABGCID
ABGCI
ABG CIJE
ABG F

36 34
A C ✓

40
E ✓

1
A ✓

32 30 28
C D H
✓

55 53
H G

25
F ✓

15 12
✓ A C

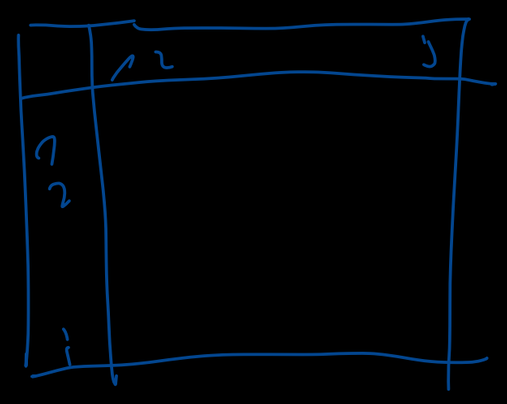
20
D ✓

I – G – E – H – B F – D – C – A

C	D	F	H	E	B	G	I
1	12	20	25	28	34	40	53

$O(2^n \cdot n)$

abc 有 2^3 个子序列，a 在/不在，b 在/不在



$$\text{OPT}(i,j) = \begin{cases} 1 & \text{falls } i = j \\ 2 & \text{falls } str[i] = str[j] \text{ und } i+1 = j \\ \text{OPT}(i+1, j-1) + 2 & \text{falls } str[i] = str[j] \\ \max(\text{OPT}(i, j-1), \text{OPT}(i+1, j)) & \text{sonst} \end{cases}$$