Angewandte Mathematik

Dennis Wilfert, Johann Studt

Gelöste Probleme

Schönstes Programm

Schönstes Problem

Probleme:

- Magic Numbers (471)
- Division (725)
- Formula1 (11056)
- Power et. al (10515)
- Password Search (902)

Gelöste Probleme

Teilbarkeit: Perfection (382), Goldbach's Conjecture (543), Goldbach Conjecture II (686), Divison (725), You can say 11 (10929)

Zahlen: Magic Numbers (471), Ecological Bin Packing (102), Finding Rectangles (638)

Zeichenketten: TEX Quotes (272), Palindromes (401), Rotating Sentences (490), Password Search (902), Find the Telephone (10921), Formula 1 (11056), Group Reverse (11192), Decoding the Message (11220)

Formeln: Polynomial Shutdown (392), Power Et. Al (10515)

Basisumwandlung: Roman Numerals (185), The Return of the Roman Empire (759), Kibbles "n" Bits "n" Bits (446), All You Need Is Love! Bits "n" (10193)

Große Zahlen: Overflow (465), Simplifying Fractions (10814), Connect the Cable Wires (10862), Krakovia (10925), The Ghost of Programmers (10992), Product (10106)

Schönstes Programm

Perfection (382): Lermer Florian, Sayli Hidir, Taskin Umut

Password Search (902): Wilfert Dennis, Studt Johann

Rational Spiral (493):Miesel Christoph, Seilbeck Robert, Wolfram Andre

Schönstes Problem

Power Et. Al (10515)

You can say 11 (10929)

Band Width (140)

Magic Numbers 471

Für eine beliebige Zahl "N" sollen sämtliche Brüche gefunden werden, dessen Zähler und Nenner jeweils keine sich wiederholenden Ziffern beinhalten dürfen

Magic Numbers 471

Beispiel

Eingabe:

1 (Anzahl der Testfälle)

1234567890

Ausgabe:

1234567890 / 1 = 1234567890

2469135780 / 2 = 1234567890

4938271560 / 4 = 1234567890

6172839450 / 5 = 1234567890

8641975230 / 7 = 1234567890

9876543120 / 8 = 1234567890

Eingabe:

1

47298

Ausgabe:

47298 / 1 = 47298

236490 / 5 = 47298

...

8209135476 / 173562 = 47298

8431057692 / 178254 = 47298

...

9702853614 / 205143 = 47298

9742016358 / 205971 = 47298

Magic Numbers 471

Lösungsansatz

Schleife die von 1-9876543210 hochzählt

Überprüfung der Zählervariable auf wiederholende Ziffern

Multiplizieren der Zählervariable mit N

Ergebnis auf Gültigkeit überprüfen

Bei Gültigkeit beider Werte formatiert ausgeben

Magic Numbers 471

Prüfung auf wiederholende Ziffern

10-stelliges Array deklarieren

Zahl durch modulo 10 teilen

Position des Wertes im Array um 1 erhöht

Übersteigt ein Wert im Array den Wert 1 wird abgebrochen

Ansonsten wird die Zahl durch 10 geteilt

```
s2 von 1 bis 9876543210 Schritt 1
    wenn s2 gültig dann
        s1 = N * s2
    wenn s1 < 9876543210 und s1 gültig
        Ausgeben
```

```
Funktion gültig
    Solange s >= 1
        array[(s % 10)] +=1
        wenn array[(s %10)] > 1  dann return false
        sonst s / 10
return true
```

Division 725

Zu einer Zahl N soll ein Bruch gefunden werden bei dem Zähler und Nenner unterschiedliche Ziffern vorweisen.

Dabei ist darauf zu achten das Zähler und Nenner keine wiederholenden Ziffern beinhalten

$$N = \frac{abcde}{fghij}$$

Division 725

Unterschied zu Magic Numbers:

Zähler und Nenner müssen immer 5 Stellen besitzen.

Die 0 kann auch vorne anstehen(01234)

N liegt nur zwischen 2 und 79

$$N = \frac{abcde}{fghij}$$

Division 725

Beispiel

Eingabe:

61

62

0 (beendet die Eingabe)

Ausgabe:

There are no solutions for 61.

79546 / 01283 = 62

94736 / 01528 = 62

Eingabe:

44

0

Ausgabe:

58476 / 01329 = 44

59268 / 01347 = 44

67892 / 01543 = 44

69432 / 01578 = 44

95348 / 02167 = 44

Division 725

Lösungsansatz

Schleife von 1234 bis 98765

Überprüfung auf wiederholende Ziffern

N wird mit Zählervariable multipliziert

Überprüfung auf Gültigkeit des Ergebnisses

Sind die Zahl gültig werden sie formatiert ausgegeben

Gibt es keinen gültigen Bruch für N

wird „There are no solutions for N“ ausgegeben

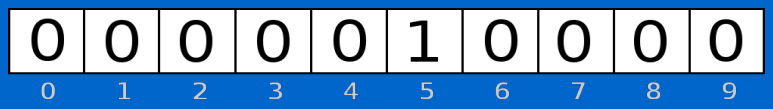
Division 725

Prüfung auf wiederholende Ziffern

Ziffern werden durch modulo 10 Rechnung bestimmt

$$12345 \% 10 = 5$$

Position im Array wird um 1 hochgezählt



Zahl wird durch 10 geteilt

$$12345 / 10 = 1234$$

Nach erfolgreichem durchlaufen wird das Array zurückgegeben oder „null“ bei sich wiederholenden Ziffern

Bei nächsten Aufruf kann das Array wieder übergeben werden

```
s2 von 1234 bis 98765 Schritt 1
    wenn s2 gültig dann
            s1 = N * s2
    wenn s1 > 98765 abbrechen
    sonst  wenn s1 gültig
       wenn s1 > 10000 und s2 > 10000 dann ausgeben

       wenn s1 > 10000 und s2 < 10000 und array[0] == 0
               dann 0 vor s2 setzen und ausgeben
```

```
Funktion gültig
    Solange s >= 1
        array[(s % 10)] +=1
        wenn array[(s %10)] > 1  dann return null
        sonst s / 10
    return array[]
```

Formula 1 11056

- Fahrer sollen nach ihrer Fahrzeit chronologisch geordnet werden
- Anzahl der zu vergleichenden Fahrer gefolgt von je einer Zeile mit Namen und Fahrzeit.
- Bei gleicher Fahrzeit sollen die Namen „case-insensitive“ sortiert ausgegeben werden
- Es können gleiche Namen mit verschiedenen Zeiten auftreten

Formula 1 11056

Beispiel

Eingabe:

3

Schumacher : 1 min 23 sec 172 ms

Barrichello : 2 min 12 sec 999 ms

Senna : 0 min 55 sec 582 ms

Ausgabe:

Row 1

Senna

Schumacher

Row 2

Barrichello

Eingabe:

4

z : 1 min 23 sec 000 ms

A : 1 min 23 sec 000 ms

Schumacher: 1 min 10 sec 000 ms

Schumacher: 1 min 30 sec 000 ms

Ausgabe:

Row 1

Schumacher

A

Row 2

z

Schumacher

Formula 1 11056

Lösungsansatz

- Auslesen einer Zeile als String
- Namen in ein StringArray speichern
- Zeiten in ms umgerechnet in ein LongArray speichern
- Array durchlaufen um die kürzeste Zeit zu finden
- Kommt diese Zeit nur 1 mal vor wird der dazugehörige Fahrer ausgegeben
- Die Zeit wird auf das Maximum gesetzt damit sie beim nächsten Durchgang nicht noch einmal gefunden wird
- Die Ausgabe der Startreihe wird durch einen extra Zähler bestimmt

Formula 1 11056

Sonderfall bei gleichen Zeiten

- Anzahl der gleichen Zeiten wird bestimmt
- Dazugehörige Namen werden in einem neuen Array gespeichert
- Mit der Funktion Arrays.sort werden die Namen in die richtige Reihenfolge gebracht

```
Von 0 bis Fahreranzahl Schritt 1
    Wenn Zeit < Zeitminimum dann Zeitminimum = Zeit
                                        und same = false
    Wenn Zeit == Zeitminimum dann same = true
Wenn same == false dann Fahrer ausgeben
Sonst
    Von 0 bis Fahreranzahl Schritt 1
     Wenn Zeit == Zeitminimum dann array[] = Fahrername
   Arrays.sort(same,
  String.CASE_INSENSITIVE_ORDER);

    Von 0 bis same Schritt 1
       Fahrernamen ausgeben
```

Power et. al 10515

2 positive Zahlen m & n sind gegeben

Sie repräsentieren eine exponentielle Zahl m^n

Es soll nun die letzte Ziffer des Ergebnisses von m^n ermittelt werden

m & n können jeweils bis zu 10^{101} Ziffern besitzen

Power et. al 10515

Beispiele

Eingabe:

2 2

2 5

0 0

Ausgabe:

4

2

Eingabe:

89654785412589577454545451212 78965455431024520245

0 0

Ausgabe:

2

Power et. al 10515

Lösungsansatz

m = 54784629465928749656894264385342892359567298235697

n = 98356398523598652846395836492835635893652839652958692380 4

m = 7

n = 04

7 * 7 = 49 n = 3

49 * 7 = 343 n = 2

3 * 7 = 21 n = 1

21 * 7 = 147 n = 0

7

Power et. al 10515

Lösungsansatz

Auslesen der beiden Werte mit StringTokenizer und speichern als 2 Strings

Auslesen der letzten Stelle der Basis-m und der letzten beiden Stellen des Exponenten-n

m wird n mal mit sich selbst multipliziert

Übersteigt das Ergebnis 99 wird nur mit der letzten Stelle weitergerechnet

Sonderfall:

Wenn die letzten 2 Stellen der Zahl n 0 sind dann wird n gleich 4 gesetzt werden

```
Wenn Exponent == 00 dann Exponent = 4
Produkt = Basis
Solange Exponent > 0
        Wenn Produkt > 99 dann Produkt = Produkt % 10
        Produkt = Produkt * Basis
        Exponent -1
Produkt % 10 ausgeben
```

Password Search 902

Gegeben ist String aus Buchstaben und eine vorangestellten Zahl N zwischen 1 und 10.

Es soll nun der am häufigsten vorkommende Substring mit der Länge N gesucht werden.

Kommen mehrere Substring gleich oft vor muss der erste von allen ausgegeben werden

Password Search 902

Beispiele

Eingabe:

3 baababacb

(baa, aab, aba, bab, aba, bac, acb)

Ausgabe:

aba

Eingabe:

5 fdbacdertdbacd

Ausgabe:

dbacd

Password Search 902

Lösungsansatz

- Den String in einer Schleife durchlaufen und die Substrings in einer Hashtable eintragen
- Zusätzlich wirdzu jedem Substring ein Integerwert gespeichert der die Anzahl seines Vorkommens darstellt
- Die Position des am häufigsten vorkommende Substring wird zusätzlich in einer Variable gespeichert, genauso wie die Anzahl seines vorkommens

2 abcdababcd

Hashtable:

String	Integer
ab	3
bc	2
cd	2
da	1
ba	1

int substrmax = 6
int max = 3

```
max = 1
Von 0 bis StringLänge+ 1 – SubStringLänge Schritt 1
        Wenn Hashtable beinhaltet SubString
            Dann Anzahl auslesen
                Wenn Anzahl < max dann Anzahl + 1 in Hashtable
                Sonst Anzahl + 1 in Hashtable und max = Anzahl + 1
                    Und subStringMax = i
      Sonst SubString und 1 in Hashtable
subStringmax ausgeben
```

Vielen Dank für eure Aufmerksamkeit!