Proseminar
Effiziente Algorithmen

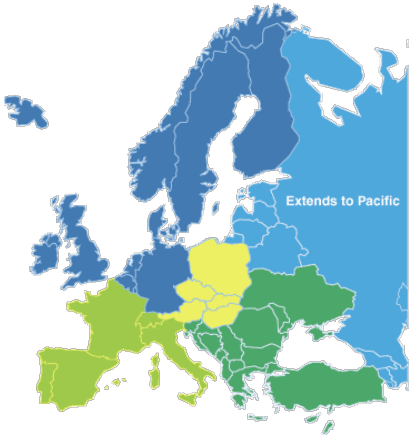
Kapitel 1: Einführung

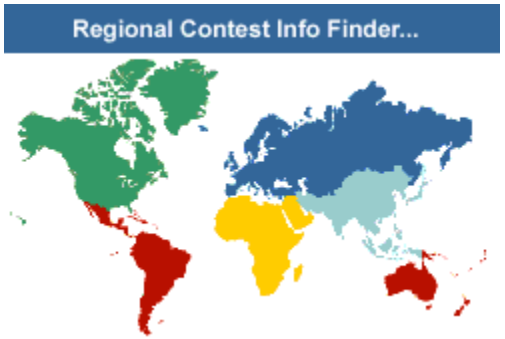
Prof. Dr. Christian Scheideler

WS 2020

ACM International Collegiate Programming Contest

- Hochschulinterner Wettbewerb
- Regionale Wettbewerbe
- Weltfinale






acm

Übersicht

- Einführung
- Datenstrukturen
- Zeichenketten
- Suchen und Sortieren
- Arithmetik und Kombinatorik
- Zahlentheorie
- Graphdurchlauf
- Graphalgorithmen
- Dynamische Programmierung
- Backtracking und Branch-and-Bound
- Geometrie

Buch:

S. Skiena und M. Revilla

Programming Challenges

Springer Verlag, 2003

Anforderungen

- Anwesenheit im Kurs
- Korrekte Lösung von mind. 60% der wöchentlichen Programmieraufgaben (in Teams bis zu 3 Personen)
- Aufarbeitung und Präsentation eines ICPC Wettbewerbs (in Teams bis zu 3 Personen)

Einreichung der Lösungen:

- http://uva.onlinejudge.org/index.php
- Email mit Programm und UVa-Bestätigung an scheideler@upb.de

ICPC Problemarchiv

- http://uva.onlinejudge.org/index.php

1. Register
2. Login

- Unter “Online Judge” finden sich alle relevanten Informationen zu euren Einreichungen

- Problemklassifizierung: http://www.cs.uleth.ca/%7Echeng/contest/hints.html

Feedback

- **Akzeptiert (AC):** Herzlichen Glückwunsch!
- **Präsentationsfehler (PE):** Ausgabe richtig, aber nicht im richtigen Format
- **Akzeptiert (PE):** Ein geringfügiger Präsentationsfehler, aber das Problem wird trotzdem noch als gelöst betrachtet.
- **Falsche Antwort (WA):** Einer oder mehrere Fälle sind falsch gelöst worden.
- **Compilerfehler (CE):** Die Fehlermeldungen werden zurückgeschickt.
- **Laufzeitfehler (RE):** Das Programm brach während der Abarbeitung mit einem Fehler ab. Die Fehlermeldung wird zurückgeschickt.

Feedback

- **Zeitlimit überschritten (TL):** Das Programm hat zu lange gebraucht.
- **Speicherlimit überschritten (ML):** Das Programm hat zuviel Speicher gebraucht.
- **Ausgabelimit überschritten (OL):** Zuviel Ausgabetext (deutet auf Endlosschleife)
- **Beschränkte Funktion (RF):** Benimm Dich!
- **Einreichungsfehler (SE):** User ID oder Problem ID inkorrekt.

Programmiersprachen

Erlaubte Programmiersprachen:

- C, C++, Pascal, Java

Ein- und Ausgabe über Standard I/O

- C:
  while (scanf(“%ld”, &x)!=EOF) … printf(“%ld\n”,y);
- C++:
  while (std::cin >> x) … std::cout << y << endl;
- Pascal:
  while not eof do … readln(x); … writeln(y);

Tipps zum Programmieren

Wichtige Kriterien:

- Zeitaufwand und Platzaufwand, aber
- vor allem Programmieraufwand!

- Schreib zunächst Kommentare!
- Dokumentiere jede Variable!
- Verwende symbolische Konstanten!
- Vermeide komplexe Datentypen (Objekte, Pointer)!
- Verwende Subroutinen!
- Verwende Debugging-Anweisungen klug!
  (bedingte Ausgabe bei Problemen, gib Erläuterungen zu Ausgabewerten)

Elementare Datentypen

- char, int, long, real, double,...
- Einfache Arrays (A[i]): oftmals vorzuziehen gegenüber Pointerstrukturen
  - mit Bereichsüberprüfung
  - etwas größer als notwendig
- Mehrdimensionale Arrays (A[i][j]): gut für Gitter und Matrizen
- Structs: Nur sinnvoll, wenn Arrays nicht anwendbar (besser typedef int point[2]; als struct point { int x, y; };)

Das 3n+1 Problem

Betrachte den folgenden Algorithmus zur Berechnung einer Folge von Zahlen:

Starte mit einer natürlichen Zahl n. Falls n gerade ist, teile durch 2. Falls n ungerade ist, multipliziere sie mit 3 und addiere 1 hinzu. Wiederhole diese Regel für die neue Zahl, bis die Zahl 1 erreicht ist.

Die folgende Folge ergibt sich zum Beispiel für n=22:

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Man nimmt an, dass der Algorithmus für alle natürliche Zahlen terminiert, aber bisher konnte noch niemand einen Beweis dafür finden. Zumindest ist bekannt, dass der Algorithmus für alle Zahlen bis 1.000.000 terminiert.

Für eine Zahl n sei die Folgenlänge von n die Anzahl der Zahlen, die erzeugt werden bis und inklusive 1. D.h. im obigen Beispiel ist die Folgenlänge 16. Gib für beliebige zwei Zahlen i und j die maximale Folgenlänge für alle Zahlen zwischen i und j (inklusive i und j) aus.

Das 3n+1 Problem

Eingabe: Eine Folge von Zahlenpaaren, ein Paar pro Zeile. Alle Zahlen sind kleiner als 1.000.000 und größer als 0.

Ausgabe: Für jedes Zahlenpaar i und j, gib i und j und die maximale Folgenlänge für i und j aus. Diese drei Zahlen sollen durch ein Leerzeichen getrennt sein und in einer Zeile stehen, mit einer Zeile pro Zahlenpaar.

Beispiel:

1 10	1 10 20
100 200	100 200 125

Erster Versuch

```
1.  #include <iostream>

2.  int main() {
3.      unsigned long a, b, c, d, max, i;
4.
5.      while (std::cin >> a >> b) {
6.          max = 0;
7.          for (c=a; c<=b; c++) {
8.              i = 1;
9.              d = c;
10.             while (d!=1) {
11.                 if (d%2) d = 3*d+1;
12.                     else d = d/2;
13.                 i = i+1;
14.             }
15.             if (i>max) max = i;
16.         }
17.         std::cout << a << " " << b << " " << max << "\n";
18.     }
19. }
```

Erster Versuch

Hi,

This is an automated response from UVa Online Judge.

Your submission with number **14354035** for the problem **100 - The 3n + 1 problem** has failed with verdict ***Wrong answer***.

Although your program was successful at compilation and execution stages, it was not able of solving the proposed problem.

Best regards,

The UVa Online Judge team

Das 3n+1 Problem

- Das Problem ist kniffeliger als es scheint!
- Lies die Spezifikation genau!!!

Fallstricke:

- Es ist nicht vorgegeben, dass i<j sein muss!
- Die Zahlen können sehr groß werden!

Bignum

```
1. #include <stdio>

2. #define   MAXDIGITS        100    /* max length */

3. typedef struct {
4.         char digit[MAXDIGITS];      /* number */
5.         int lastdigit;                 /* max digit */
6. } bignum;
```

Bignum

```
1. print_bignum(bignum *n)
2. {
3.     int i;

4.     for (i=n->lastdigit; i>=0; i--)
5.             printf("%c",'0'+ n->digits[i]);
6. }
```

Bignum

```
1.  int_to_bignum(int s, bignum *n)
2.  {
3.          int i;                                  /* counter */
4.          int t;                                  /* int to work with */

5.          for (i=0; i<MAXDIGITS; i++) n->digits[i] = (char) 0;

6.          n->lastdigit = -1;
7.          t = abs(s);

8.          while (t > 0) {
9.                  n->lastdigit ++;
10.                 n->digits[ n->lastdigit ] = (t % 10);
11.                 t = t / 10;
12.         }

13.         if (s == 0) n->lastdigit = 0;
14. }
```

Zweiter Versuch

Hi,

This is an automated response from UVa Online Judge.

Your submission with number **14354890** for the problem **100 - The 3n + 1 problem** has failed with verdict ***Time limit exceeded***.

Your program used more CPU time than what is allowed for this problem. That means that your algorithm is not fast enough or that it entered into an infinite loop.

Best regards,

The UVa Online Judge team

Häufig verwendete Techniken

- Diverse Datenstrukturen (Priority Queue, Union-Find,…)
- Binäre Suche
- Breiten- und Tiefensuche
- Kürzeste Wege
- Minimale Spannbäume
- Divide-and-Conquer
- Dynamische Programmierung
- Zahlentheoretische Algorithmen (Primzahlsieb, GGT, Matrixmultiplikation,…)

} DuA

- Bipartites Matching
- Netzwerkfluss

} GA

- Geometrische Probleme
- Heuristiken für harte Probleme (A*)

} Master

Weitere Elementare Probleme

- 105: The Skyline Problem
- 119: Greedy Gift Givers
- 10137: The Trip
- 10267: Graphical Editor
- 10033: Interpreter
- 10142: Australian Voting
- 156: Ananagrams

Hausaufgabe:

- 201: Squares

Fragen?