TU Ilmenau, Fakultät für Informatik und Automatisierung
FG Komplexitätstheorie und Effiziente Algorithmen
Univ.-Prof. Dr. M. Dietzfelbinger, Philipp Schlag, Stefan Walzer, Alexander Jeuk
https://www.tu-ilmenau.de/iti/lehre/lehre-ss-2020/aud/

TECHNISCHE UNIVERSITÄT
ILMENAU

Algorithmen und Datenstrukturen SS 2020 – C++-Einführung

Zur Bearbeitung zwischen 2. Juni - 5. Juni (KW 23)

Vorraussetzung: Diese Aufgabenbeschreibung geht davon aus, dass Sie bereits eine **C++ Entwicklungsumgebung** bei sich eingerichtet haben. Wir empfehlen *Visual Studio Code* und beschreiben die Installation in einem anderem Dokument.
Ziel folgender Aufgaben ist, dass Sie sich mit den **Grundlagen von C++**, der Nutzung von **Compiler** und **Debugger** sowie dem **DOMjudge** Interface vertraut machen. Wir empfehlen, dass Sie neben den vorgeschlagenen Aufgaben auch selbst ein wenig herumprobieren und sich in der Sprachreferenz umschauen:

`https://en.cppreference.com` (Englisch) `https://de.cppreference.com` (Deutsch)

Für alle Aufgaben ist ein C++ Programm zu schreiben, das eine einfache Problemstellung löst. Das Programm soll Eingaben von der Standardeingabe (`stdin`) einlesen und die zugehörigen Ausgaben auf die Standardausgabe (`stdout`) schreiben. Für jede Aufgabe wird eine Datei `sample.in` mit mehreren Testfällen sowie eine Datei `sample.ans` mit den zugehörigen Musterausgaben bereitgestellt. Testen Sie Ihr Programm mit den **bereitgestellten Testfällen**. Anschließend sollen Sie über das DOMjudge Interface ihren **Programmtext einreichen** (*nicht* das kompilierte Programm):

DOMjudge: `http://kt30.theoinf.tu-ilmenau.de`

Auf dem Server wird Ihr Programm automatisch kompiliert und ausgeführt. In der Regel werden neben den Beispieleingaben auch andere „geheime“ Eingaben zum Testen verwendet. Nach wenigen Sekunden bekommen Sie eine **Rückmeldung**. Mögliche sind:

CORRECT	Ihr Programm funktioniert auf allen (unseren) Eingaben korrekt.
TIME LIMIT EXCEEDED	Ihr Programm braucht *erheblich* länger als es sollte. Der Grund dafür ist fast immer eine Endlosschleife oder ein völlig ungeeigneter Algorithmus.
RUNTIME ERROR	Ihr Programm stürzt ab. Gründe können sein (unter anderem): Sie lesen jenseits einer Arraygrenze, Sie dereferenzieren einen Null-Zeiger, Sie teilen durch Null oder Sie reservieren viel zu viel Speicher.
WRONG ANSWER	Ihr Programm produziert die falsche Ausgabe. Beachten Sie, dass auch Formatierungsfehler (z.B. „`(1, 2)`“ statt „`(1,2)`“) zu diesem Ergebnis führen können. Führen Sie daher unbedingt auf Ihrem eigenen Computer einen maschinellen Vergleich („diff“) zwischen erwarteter und tatsächlicher Ausgabe bei den bereitgestellten Testfällen durch.

Sie dürfen Ihre Einreichung **beliebig oft wiederholen**. Probieren Sie aber bitte nicht blind herum, unsere Systeme sind zum Debuggen weder gedacht noch geeignet!

Bitte schauen Sie sich parallel zu diesem Aufgabenblatt die entsprechenden **Erklärvideos** auf Moodle an, die Sie durch die Aufgaben führen werden. Sie benötigen außerdem das **Praktikumsmaterial**.

Aufgabe 0 (Hello [*your name here*]!)

Entpacken Sie das Praktikumsmaterial und öffnen Sie den entpackten Ordner in VSCode. Wählen Sie anschließend links im Explorer die Datei `0-hello/hello.cpp` aus.
Bearbeiten Sie das Programm, sodass es einen Namen aus der Eingabedatei liest und "Hello *[name]*!" in die Ausgabedatei schreibt.

Listing 1: `sample.in`

```
1 Dave
```

Listing 2: `sample.ans`

```
1 Hello Dave!
```

Führen Sie Ihr Programm mit allen drei bereitgestellten Konfigurationen aus:

`INTERACTIVE`	Das Programm wird ausgeführt und *SIE* stellen den Input manuell über das Terminal zur Verfügung.
`RUN sample.in`	Das Programm wird mit der Beispieleingabe ausgeführt. Sie sehen die Ausgabe im Terminal.
`DIFF sample.in`	Das Programm wird mit der Beispieleingabe aus der Datei `sample.in` ausgeführt. Die Ausgabe Ihres Programms wird anschließend automatisch mit der erwarteten Ausgabe aus der Datei `sample.ans` verglichen. Alle Abweichungen werden angezeigt. Keine Ausgabe bedeutet keine Abweichung!

Alle Konfigurationen verwenden stets die gerade angezeigte `.cpp` Datei. Das Programm wird kompiliert und anschließend im Debugger gestartet (sofern keine Compilerfehler auftreten).
Wenn Ihr Code zu funktionieren scheint, reichen Sie ihn über das DOMjudge Interface ein.

Einschub (Debugging)

Sehen Sie sich das Video zum Thema Debugger an. Sie sollten danach folgende Fragen beantworten können.

(a) Zunächst widmen wir uns GDB und dem Nutzen von GDB in der VSCode Umgebung.
Versuchen Sie folgende Fragen zu beantworten:

- **(i)** Wie startet man den GNU Debugger (GDB) in VSCode?
- **(ii)** Wie setzt man Breakpoints und was machen diese?
- **(iii)** Wo sieht man lokale Variablen?
- **(iv)** Wie kann man Schritt für Schritt durch den Code gehen?

(b) Öffnen Sie die Datei `debug/buggy-code.cpp` in VSCode. Die Datei enthält zahlreiche Fehler die ein Compiler nicht erkennen kann und die zu Ausnahmen/Exceptions zur Laufzeit führen. Ein Beispiel wäre eine Zugriffsverletzung (access violation) bei ungültigem Speicherzugriff.

- **(i)** Welche Art von Ausnahme wird in der zuerst aufgerufenen Funktion erzeugt?
- **(ii)** Setzen Sie mit Hilfe der Variablenanzeige und einem geeigneten Breakpoint die lokale Variable `test` auf 2. Gehen Sie dann mit Single-Stepping durch die Schleifen der `Simple_Arithmetic` Funktion. Welchen Wert hat die Variable `var`, wenn die Exception ausgelöst wird?

Aufgabe 1 (ADD)

Jede Zeile der Eingabe enthält zwei Zahlen x, y. Auszugeben ist jeweils die Summe $x + y$.

Listing 3: `sample.in`

```
3 7
10 12
-10 0
32 19
-54 -11
```

Listing 4: `sample.ans`

```
10
22
-10
51
-65
```

Aufgabe 2 (RMQ)

Die erste Zeile der Eingabe enthält eine Zahl $n \geq 0$. Die zweite Zeile enthält n Zahlen $x_0, \ldots, x_{n-1}$.
Es folgen weitere Zeilen, die jeweils zwei Zahlen i und j enthalten ($0 \leq i < j \leq n$). Für jede solche *Anfrage* (i, j) ist die Zahl $\min_{i \leq k < j} x_k$ auszugeben, das heißt, die kleinste Zahl unter allen Zahlen $x_i, \ldots, x_{j-1}$.

Listing 5: `sample.in`

```
13
1 3 23 1 -5 8 4 7 9 -2 3 -4 1
0 1
0 2
1 3
2 4
3 5
4 6
8 13
0 13
```

Listing 6: `sample.ans`

```
1
1
3
1
-5
-5
-4
-5
```

Bemerkung: Anfragen dieser Art nennt man *Range Minimum Queries* (RMQ).
Ein Array A mit n Einträgen vom Typ `int` können Sie anlegen mit

```
vector<int> A(n);
```

wenn Sie am Anfang der `.cpp` Datei `#include<vector>` hinzufügen.

Aufgabe 3 (REVERSE)

Jede Zeile der Eingabe enthält eine Zahl $n \geq 0$ sowie n weitere Zahlen $x_0, \ldots, x_{n-1}$. Auszugeben sind die Zahlen in umgekehrter Reihenfolge, also als $x_{n-1}, \ldots, x_0$.

Listing 7: `sample.in`

```
 9   -34 54 1 -39 87 0 8 -9 1
 7   345 -28 -31 89 75 -8 971
 1   10
 5   54 3 22 3 -10
```

Listing 8: `sample.ans`

```
1 -9 8 0 87 -39 1 54 -34
971 -8 75 89 -31 -28 345
10
-10 3 22 3 54
```

(a) Benutzen Sie **einfach-verkettete Listen**, um diese Aufgabe zu lösen!

(b) Lösen Sie die Aufgabe nochmal unter Verwendung eines `stack<int>`. Lesen Sie dazu in der Referenz unter `https://en.cppreference.com/w/cpp/container/stack` nach, wie die Funktionen `push`, `pop` und `top` funktionieren. Vergessen Sie nicht `#include<stack>`!

Aufgabe 4 (POINTADD)

Jede Zeile enthält die Koordinatenvektoren zweier Punkte $p_1, p_2 \in \mathbb{Z}^2$. Auszugeben ist die Summe der beiden Vektoren.

Listing 9: `sample.in`

```
(3,4) (-10,20)
(214,213) (-9,4)
(12,-12) (-54,-78)
(-2,74) (-9,2)
```

Listing 10: `sample.ans`

```
(-7,24)
(205,217)
(-42,-90)
(-11,76)
```

Bemerkung: Um einen Punkt aus der Eingabe einzulesen, deklarieren Sie zwei Variablen `x`, `y` vom Typ `int` und eine Variable `c` vom Typ `char`. Letztere dient nur dazu, die Kommata und Klammern aufzufangen. Nutzen Sie dann `cin >> c >> x >> c >> y >> c;`. Bei obiger Eingabedatei würde dann entsprechend `x` den Wert 3 und `y` den Wert 4 enthalten.
Bemerkung (ii): Trotz des Namens bietet sich die Verwendung der C++ Datenstruktur `std::vector` zur Lösung dieser Aufgabe *nicht* an.

Aufgabe 5 (LISTINSERT)

Für jede Zeile der Eingabe soll ein Element in eine anfangs leere Liste eingefügt werden. Eine Zeile enthält zwei Zahlen x und y und verlangt, dass die Zahl y hinter dem ersten Vorkommen der Zahl x in die Liste eingehängt wird. Falls x nicht in der Liste enthalten ist, soll y an das Ende der Liste angehängt werden (das ist insbesondere bei der ersten Einfügung der Fall).
Geben Sie die Liste nach jeder Einfügung aus.

Listing 11: `sample.in`

```
0 1
1 3
1 2
1 5
2 4
4 6
5 7
1 8
8 9
```

Listing 12: `sample.ans`

```
1
1 3
1 2 3
1 5 2 3
1 5 2 4 3
1 5 2 4 6 3
1 5 7 2 4 6 3
1 8 5 7 2 4 6 3
1 8 9 5 7 2 4 6 3
```

Aufgabe 6 (SORT)

Jede Zeile der Eingabe enthält eine Zahl $n \geq 0$ sowie n weitere Zahlen $x_0, \ldots, x_{n-1}$. Auszugeben sind die Zahlen $x_0, \ldots, x_{n-1}$ in aufsteigend sortierter Reihenfolge. Implementieren Sie dazu den Sortieralgorithmus **InsertionSort**.

Listing 13: `sample.in`

```
 9   -34 54 1 -39 87 0 8 -9 1
 7   345 -28 -31 89 75 -8 971
 1   10
 5   54 3 22 3 -10
```

Listing 14: `sample.ans`

```
-39 -34 -9 0 1 1 8 54 87
-31 -28 -8 75 89 345 971
10
-10 3 3 22 54
```

Als Vorbild können Sie den Code aus Kapitel 1 der Vorlesung verwenden.
Alternativ können Sie Ihre Lösung für Aufgabe **5** anpassen.

Aufgabe 7 (MATRIX-RMQ)

Die Eingabe beginnt mit einer Zeile, die zwei Zahlen $n > 0$ und $m > 0$ enthält. Anschließend folgen n Zeilen mit jeweils m Zahlen, die zusammen eine Matrix $A = (a_{ij})_{0 \le i < n, 0 \le j < m}$ beschreiben. Es folgen weitere Zeilen mit jeweils vier Zahlen i_1, j_1, i_2, j_2 mit $0 \le i_1 < i_2 \le n$ und $0 \le j_1 < j_2 \le m$. Diese weiteren Zeilen sind *Anfragen*. Für jede Anfrage ist $\min_{i_1 \le i < i_2, j_1 \le j < j_2} a_{ij}$ auszugeben, das heißt die kleinste Zahl in der entsprechenden Teilmatrix von A.

Listing 15: `sample.in`

```
5 3
1 2 3
6 5 4
7 8 9
6 5 3
3 2 0
0 0 1 1
1 1 3 3
1 0 2 3
2 1 4 3
0 0 5 3
```

Listing 16: `sample.ans`

```
1
4
4
3
0
```

Bemerkung: Eine Matrix der Größe $n \times m$ können Sie anlegen mit:

```
vector<vector<int>> A(n, vector<int>(m,0));
```

Hierbei ist `vector<int>(m,0)` ein Array mit m Elementen vom Typ `int`, die mit 0 vorinitialisiert sind, also eine Null-Zeile. `A` selbst ist ein Array, das n solche Null-Zeilen enthält. Auf den Eintrag a_{ij} können Sie dann mit `A[i][j]` lesend und schreibend zugreifen.

Aufgabe 8 (FIBONACCI)

Jede Zeile der Eingabe enthält eine Zahl $0 \le n \le 80$. Auszugeben ist jeweils die n-te Fibonacci-Zahl.

Listing 17: `sample.in`

```
0
1
2
3
4
5
6
7
10
20
30
```

Listing 18: `sample.ans`

```
0
1
1
2
3
5
8
13
55
6765
832040
```

Hinweis: Beachten Sie, dass in der Beispieleingabe nur Zahlen $n \le 30$ vorkommen, das DOMjudge System allerdings auch die größte erlaubte Eingabe $n = 80$ testet. Eine naive Implementierung wird daher zum Ergebnis TIME LIMIT EXCEEDED führen. Außerdem ist die korrekte Ausgabe für $n = 80$ eine sehr große Zahl die nicht durch den Datentyp `int` (32-bit Integer) dargestellt werden kann, wohl aber durch den Datentyp `uint64_t` (64-bit Integer).