

Digitale Systeme 2018

C-Programmierpraktikum

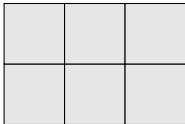
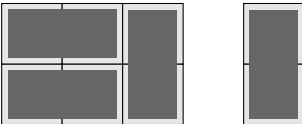
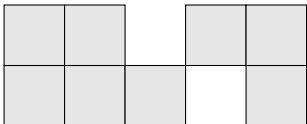

1 Aufgabenstellung

Fliesenlegerin Frauke soll einige Räume mit rechteckigen Fliesen der Größe 1×2 fliesen. Leider ist ihr Fliesenschneider kaputt. Die zu fliesende Fläche F ist die Vereinigung von Kacheln der Kantenlänge 1, deren Eckpunkte allesamt auf dem Einheitsgitter in 2 Dimensionen liegen:

$$K_{x,y} := [x, x+1] \times [y, y+1] \quad (1)$$

$$F = \bigcup_{i=0}^{N-1} K_{x_i, y_i} \quad x_i, y_i \in \mathbb{N} \quad (2)$$

Kann Frauke das bewerkstelligen? Ihre Aufgabe ist es, ein C-Programm zu schreiben, welches für eine gegebene Fläche F entscheidet, ob es möglich ist, sie mit nicht-überlappenden Fliesen der Größe 1×2 auszulegen. Falls dies möglich ist, soll auch eine konkrete Überdeckung ausgegeben werden.

Problem	eine maximale Überdeckung	lösbar
		ja
		nein

2 Ein- und Ausgabedaten

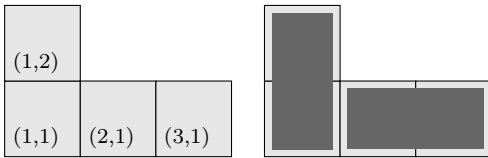
Ihr Programm soll die Fläche F von der Standardeingabe (`stdin`) lesen. Die Eingabe ist ein ASCII-kodierter Bytestrom mit Linefeed (`'\n'`) als Zeilenumbrüche. Jede Zeile der Eingabe besteht aus zwei nicht-negativen Ganzzahlen x_i und y_i in Dezimalkodierung, getrennt durch mindestens ein Leerzeichen. Diese Zahlen beschreiben eine Kachel K_{x_i, y_i} . Die Zahlen *können* beliebig viele redundante führende Nullen enthalten. Vor, zwischen und hinter den Zahlen können beliebig viele zusätzliche unnötige Leerzeichen stehen. Die letzte Zeile *kann* mit einem Zeilenumbruch abgeschlossen werden, muss aber nicht. Zwei Zeilen beschreiben niemals dieselbe

Kachel. Alle in der Eingabe auftretenden Zahlen sind echt kleiner 2^{32} , sonst ist die Eingabe ungültig.


Ihr Programm soll das Ergebnis als ASCII-kodierten Bytestrom auf die Standardausgabe (`stdout`) schreiben. Wenn ein Befliesen gemäß der Problemstellung nicht möglich ist, soll ihr Programm `"None\n"` ausgeben. Wenn ein Befliesen möglich ist, soll für *eine mögliche gültige Befliesung* jede 1×2 -Fliese durch genau eine Zeile beschrieben werden, und zwar im Format `"x_i y_i;x_j y_j"`, wobei (x_i, y_i) und (x_j, y_j) zwei benachbarte Kacheln beschreiben, d.h. $|x_i - x_j| + |y_i - y_j| = 1$.

Beispiel

Eingabe	Ausgabe
1_1	
_ _ 0 1_2	2_1; 3_1
2_1	1_2; 1_1
3_0 0 1	



1_1	None
2_2	
2_1	
3_1	



Wenn Ihr Programm eine Eingabe erhält, die nicht der obigen Spezifikation genügt, soll Ihr Programm gar nichts auf die Standardausgabe schreiben, sondern stattdessen eine sinnvolle Fehlermeldung auf die Standardfehlerausgabe (`stderr`) schreiben und beenden. Insbesondere sollten folgende Fälle als falsche Eingabe gewertet werden:

- Ein anderes Zeichen als ein Leerzeichen, ein Linefeed oder eine Dezimalziffer tritt auf.
- Eine Zeile mit null, ein oder mehr als zwei Ganzzahlen tritt auf.
- Zwei Zeilen in der Eingabe beschreiben die selbe Kachel.

Der Einfachheit halber *dürfen* Sie Tabulatoren genau so behandeln wie Leerzeichen und statt eines einzelnen Linefeeds (LF) auch ein carriage return (CR) oder ein „CR LF“ akzeptieren. Wir verzichten darauf, diesbezüglich zu testen.

3 Beispieldaten

Wir stellen Ihnen im Moodle einige Beispiel-Eingabedateien zum Testen Ihres Programms zur Verfügung, sowie dazugehörige Ausgaben. Man beachte, dass es zu einer Eingabe sehr viele gültige Ausgaben geben kann, von denen wir natürlich nur eine mögliche stellen (siehe auch Abschnitt 6.2). Die Beispiel-Eingaben decken viele — aber nicht alle — Grenzfälle ab, mit denen Ihr Programm zurecht kommen muss.

Für die Überprüfung Ihrer Abgabe werden andere Datensätze zum Einsatz kommen.

4 Hinweise zum Algorithmus

Die Aufgabenstellung lässt sich auf folgende Weise lösen:

1. Lesen Sie zunächst die Eingabedaten in eine geeignete Datenstruktur und überprüfen Sie deren Korrektheit.
2. Partitionieren Sie die zu fliesende Fläche in „Räume“, also in größte zusammenhängende Flächen. Hierbei kann die Funktion `qsort` nützlich sein. Beachten Sie, dass das Gesamtproblem hierdurch in voneinander unabhängige Teilprobleme zerfällt.
3. Legen Sie Fliesen greedy, wo die Lage eindeutig ist: Ist eine noch freie Kachel nur zu einer ihrer vier Seiten von einer anderen freien Kachel benachbart, ist, im Fall dass eine Lösung existiert, die Lage der Fliese auf diesen Kacheln eindeutig.
4. Verwenden Sie für alle übrigen Situationen Tiefensuche mit Backtracking um die gesamte noch zur Verfügung stehende Fläche auszulegen.

Es gibt allerdings auch mindestens eine sehr viel elegantere Lösung für das Problem, die außerdem auch etwas einfacher umzusetzen ist. Es könnte hilfreich sein, sich den Algorithmus von Hopcroft und Karp anzusehen. Sie dürfen allerdings jedes (korrekte) Lösungsverfahren einsetzen, das zwei grundlegende Bedingungen erfüllt:

- Es soll nicht noch ineffizienter sein als das oben skizzierte Verfahren.
- Sie müssen es uns bei der Vorstellung Ihres Programms erläutern und begründen können.

Backtracking kann eine sehr schlechte Laufzeitkomplexität aufweisen. Sorgen Sie dafür, dass Ihr Programm wenigstens Teilprobleme mit sehr einfacher Geometrie, wie rechteckige Räume, in quadratischer Laufzeit oder schneller lösen kann.

5 Zeitplan

Datum	Beschreibung
25.06.2018	Empfohlene Deadline für eine erste Abgabe; bei Abgabe bis zu diesem Termin können wir im Falle von Problemen mit der Abgabe gewährleisten, dass nach einer ersten Rückmeldung noch Zeit zur Nachbesserung bis zur endgültigen Deadline bleibt
14.08.2018	Spätest-mögliche Deadline für die Abgabe der Endversion, die alle Kriterien vollständig erfüllen muss
27.08.2018 – 07.09.2018	In diesem Zeitraum finden die mündlichen Testate in Einzelsitzungen statt. Eine frühere Abnahme ist nach individueller Absprache möglich.

6 Wichtige Hinweise und weitere Anforderungen

Beachten Sie bei der Erstellung Ihres Programms **unbedingt** folgende Punkte:

- Halten Sie sich strikt an das oben definierte Ausgabeformat und die Aufrufkonventionen. Wir überprüfen Ihr Programm automatisiert auf die Korrektheit der ausgegebenen Lösungen. *Eine falsch formatierte Ausgabe wird nicht als korrekt anerkannt!* Im Zweifelsfall fragen Sie *rechtzeitig vor Abgabe* nach!
- Verwenden Sie keine Bibliotheken außer der C-Standardbibliothek wie sie in der 2011er Ausgabe des ISO C Standard beschrieben ist. Insbesondere Nutzer von Softwareprodukten aus Redmond weisen wir darauf hin, dass Funktionen die einen CamelCase-Namen tragen wie `StrToInt`, sehr wahrscheinlich nicht Teil der C-Standardbibliothek sind.
- Ihr Programm soll als eine einzige C-Quelldatei mit dem Namen `loesung.c` in gültigem ISO C11 geschrieben sein. Ihr Programm muss sich mit folgendem einzelnen gcc-6.2.1-Compileraufruf¹ fehlerfrei übersetzen und linken lassen:

```
gcc -o loesung -O3 -std=c11 -Wall -Werror loesung.c
```

Um Fehler bei der Implementierung zu vermeiden, empfiehlt es sich, zusätzlich die Compiler-Flags `-Wextra` und `-Wpedantic` zu verwenden. Verlangt wird dies aber nicht.
- Ihr Programm muss auf der Kommandozeile ohne grafische Oberfläche lauffähig sein und darf keine Kommandozeilenargumente erwarten, d.h. es soll mit folgendem Aufruf (in `sh`, `bash` oder `zsh`) ausführbar sein:

```
cat example01.dat | ./loesung
```

(Was semantisch nicht gleichwertig ist zu `./loesung < example01.dat`.)
- Achten Sie darauf, dass Ihr Programm auf unterschiedlichen Architekturen lauffähig ist (z. B. mit unterschiedlich langen Integer-Typen).
- Legen Sie Ihr Programm so aus, dass es mit beliebig großen Eingabedaten umgehen kann. Hierfür ist es unbedingt notwendig, dass Sie dynamische Speicherverwaltung einsetzen. Wir werden Ihr Programm mit *großen* Datensätzen testen!

¹Den gcc 6.2.1 finden Sie z.B. auf den Rechnern des Berlin-Pools ([alex|britz|buch|buckow|...].informatik.hu-berlin.de) installiert unter dem Namen `gcc-6`.

- Ihr abgegebenes Programm wird automatisch auf Speicherlecks überprüft. Stellen Sie sicher, dass es keine Speicherlecks aufweist. Dies können Sie beispielsweise mit dem Werkzeug `valgrind`² bewerkstelligen. Geben Sie jeglichen dynamisch zugewiesenen Speicher vor Programmende korrekt wieder frei.
- Stellen Sie sicher, dass Ihr Programm *niemals* eine Schutzverletzung (segmentation fault, segfault) auslöst, egal welche gültige oder ungültige Eingabe es erhält.
- Wir werden alle abgegebenen Lösungen benchmarken. Die ressourcensparsamsten Lösungen werden am Ende des Semesters mit einem kleinen Preis ausgezeichnet. Wir werden hierfür die Lösungen in Ausführungszeit sowie Speicherverbrauch untersuchen.
- Kommentieren Sie Ihren Quelltext in angemessenem Umfang.
- Wir werden Feedback zu Ihren Lösungsversuchen ausschließlich als persönliche Moodle-Nachrichten versenden. Sorgen Sie dafür, dass Sie diese Nachrichten rechtzeitig lesen.

6.1 Valgrind

Valgrind ist ein sehr vielseitiges Werkzeug. Im einfachsten Fall testen Sie Ihr Programm mittels `cat example01.dat | valgrind ./loesung 1> /dev/null`

Wenn Ihr Programm keine Fehler in der Speicherverwaltung aufweist, sollten in der Valgrind-Ausgabe folgende zwei Zeilen vorkommen:

```
==1234== All heap blocks were freed - no leaks are possible
==1234== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

6.2 Ihre Resultate überprüfen

Für eine gültige Eingabe kann es viele verschiedene gültige Ausgaben geben. Dementsprechend können Sie nicht erwarten, dass Ihr eigenes Programm die selbe Ausgabe generiert, wie die von uns zur Verfügung gestellten Beispielausgaben. Um zwei Ausgaben auf Kompatibilität hin zu überprüfen, können Sie das Python-Script `check_result` verwenden. Beispiel:

```
cat example01.dat | ./loesung | ./check_result example01.out
bzw. falls Sie zeitgleich valgrind verwenden möchten:
cat example01.dat | valgrind ./loesung 1| ./check_result example01.out
```

6.3 Randfälle

- Die leere Eingabe ist eine gültige Eingabe. Die korrekte Ausgabe ist in diesem Fall die leere Ausgabe.
- Bei einer fehlerhaften Eingabe *muss* Ihr Programm mit einer Fehlermeldung abbrechen und trotzdem allen dynamisch allozierten Speicher wieder freigeben. Wenn in der Eingabe eine oder mehrere Zahlen größer oder gleich 2^{32} auftreten, dürfen Sie das als fehlerhaft werten und abbrechen oder stattdessen eine *korrekte Ausgabe* erzeugen.
- Wenn die Eingabe formal richtig ist, Ihr Programm aber nicht genug Speicher allozieren kann, soll es mit einer entsprechenden Fehlermeldung abbrechen. Beachten Sie aber,

²<http://valgrind.org/>

dass Ihr Programm alle von uns zur Verfügung gestellten Beispieleingaben mit 256 MiB Speicher bearbeiten können muss. Sie können den Speicher, den Ihr Programm erhalten kann, künstlich verknappen mit `ulimit`. Um den Speicher z.B. auf 1 MiB zu reduzieren, verwenden Sie

```
ulimit -d 1024
```

Beachten Sie in diesem Zusammenhang auch, dass eventuell ein Limit für die Stackgröße (`ulimit -s`) voreingestellt ist (möglicher Weise 8 MiB).

Viel Erfolg!