Uni Osnabrück / Theoretische Informatik — Prof. Dr. M. Chimani

Algorithm Engineering — Übungsprojekt

Aufgabenstellung & Übersicht (Preview)

Wintersemester 2020/21

Aufgabenstellung: Punktbeschriftung

Gegeben eine Menge von Punkten in der Ebene, und zu jedem Punkt die Breite und Höhe seines Beschriftungsfeldes. Einen Punkt zu beschriften heißt, sein Beschriftungsfeld „an den Punkt“ anzulegen. Beschriftungsfelder dürfen andere Punkte überdecken, es ist jedoch nicht zulässig, dass sich zwei Beschriftungsfelder überlappen (berühren ist erlaubt!). Ziel ist es, möglichst viele Punkte zu beschriften.

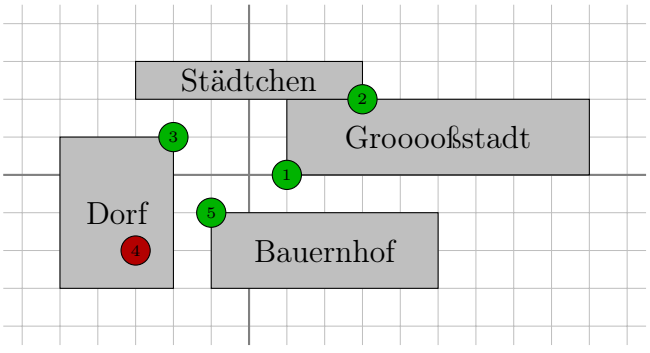
Wir beschränken uns auf „4-Positionen-Beschriftungen“, d.h. einer der Eckpunkte des Beschriftungsfeldes muss mit dem zu beschriftenden Punkt übereinstimmen. Auch gehen wir davon aus, dass alle vorkommenden Werte ganzzahlig sind.

Ein- und Ausgabeformat. Die Eingabedatei ist eine Textdatei. Die erste Zeile besteht ausschließlich aus einer positiven ganzen Zahl, die die Anzahl der in der Datei folgenden Punkte angibt. Jede weitere Zeile spezifiziert einen Punkt (und seine Lösung) durch acht Werte $x, y, \ell, h, t, b, x', y'$ (in dieser Reihenfolge; durch ein oder mehrere Whitespaces (Leerzeichen, Tab, etc.) getrennt: Die ganzen Zahlen x und y (auch negativ!) bezeichnen die horizontale x- bzw. vertikale y-Koordinate des Punktes. Die positiven ganzen Zahlen ℓ und h geben die Länge bzw. Höhe des zugehörigen Beschriftungsfelds an. Der Leerzeichen-freie String t gibt die Beschriftung selbst an (diese ist für die Berechnung selbst irrelevant).

Die letzten drei Werte geben die Lösung für diesen Punkt an—wenn eine Datei als Eingabedatei geladen wird, werden diese Werte i.A. ignoriert (solange sie richtig formatiert sind). Der Wert b gibt an, ob der Punkt beschriftet wird ($b = 1$) oder nicht ($b = 0$). Die beiden weiteren Werte x', y' geben die x und y Koordinate der linken oberen Ecke des Beschriftungsfelds an (beliebige ganze Zahlen falls $b = 0$).

Beispiel einer Ein-/Ausgabedatei.
Die kursiv gedruckten (roten) Werte werden beim Einlesen der Datei i.A. ignoriert.

```
5
 1  0 8 2 Groooooßstadt 1  1  2
 3  2 6 1 Städtchen    1 -3  3
-2  1 3 4 Dorf         1 -5  1
-3 -2 5 2 Kuhdorf      0  0  0
-1 -1 6 2 Bauernhof    1 -1 -1
```



Programmiersprache. Der Code wird in C++ geschrieben, und muss mittels eines Makefiles (nur durch Eingabe von „`make`“) unter gcc 9.3 kompilieren. Um einen einfachen Austausch und Vergleich zu gestatten sind nicht-standard Zusatzbibliotheken zu vermeiden. Falls Sie meinen, unbedingt Bibliotheken zu benötigen, sprechen Sie mit mir.

Ablaufplan

Phase 1. Benchmark-Instanzen & Framework **Deadline: Woche vom 9. Nov.**

(A) Sammeln bzw. generieren Sie Testinstanzen unterschiedlicher Größe, die in der Realität interessant sind. (Dies sollte mehr Zeit und Aufmerksamkeit in Anspruch nehmen als (B)!)

und

(B) Erstellen Sie einen funktionierenden Prototypen ihres Programms. Hier werden die Spezifikationen nun (im Gegensatz zu den meisten anderen Spezifikationen des Projekts) sehr genau angegeben, damit wir in weiterer Folge die Algorithmen der verschiedenen Gruppen leicht untereinander austauschen werden können.

Sei `progname` ein beliebiger Name für ihr Programm:

- Der Aufruf „`progname`“ (ohne Parameter) gibt auf der Konsole eine Meldung über die möglichen Aufrufparameter aus.
- Der Aufruf „`progname -in dat1 -out dat2`“ liest die Eingabedatei `dat1` ein, führt den Algorithmus aus, und schreibt die Lösung in die Datei `dat2`. Auf der Konsole gibt der Algorithmus zwei Zahlen—durch Tabulator (`'\t'`) getrennt—aus: die Anzahl der beschrifteten Punkte und die benötigte Laufzeit.
- Der Aufruf „`progname -eval dat1`“ liest die Datei `dat1` ein. Falls die darin beschriebene Lösung zulässig ist, wird die Anzahl der beschrifteten Punkte auf die Konsole geschrieben; sonst der Text „`ERROR: `“, gefolgt von einer (hilfreichen) Fehlermeldung, warum die Lösung nicht korrekt ist.

Der Algorithmus soll zu diesem Zeitpunkt noch nicht „clever“ sein; eine beliebige zulässige (=überlappungsfreie) Lösung zu generieren reicht aus. Anmerkung zur Laufzeit: Messen Sie nur die Zeit zwischen nach-dem-Laden und vor-dem-Schreiben!
Zur Deadline: Sammeln der Benchmark-Instanzen aller Gruppen

Phase 2. Mindestens zwei Heuristiken **Deadline: Woche vom 7. Dez.**

Erstellen Sie mindestens zwei grundverschiedene Algorithmen, die das Problem heuristisch möglichst gut lösen. Wir betrachten an dieser Stelle nur „traditionelle“ Programmierungen, d.h. ohne Ausnutzen von Parallelität (mehrere Threads, o.ä.). Ihr Programm darf beliebige weitere Parameter zur Algorithmenkonfiguration akzeptieren (sollte aber auch mit Defaultwerten, ohne Extra-Parameter, funktioneren).

Zur Deadline: Vorstellung der Resultate inkl. Ausblick was sich verbessern ließe; Austausch der Algorithmen

Phase 3. Exakte Verfahren **Deadline: Woche vom 11. Jan.**

Details folgen.

Zur Deadline: Vorstellung der Resultate und Vergleiche ggü. den Heuristiken.

Phase 4. Verbesserte Algorithmen **Deadline: Woche vom 1. Feb.**

Details folgen.

Zur Deadline: Vorstellung der Resultate