

1. Aufgabenblatt zum Softwarepraktikum “Muster- und Bilderkennung” im Grundstudium

Bemerkungen:

Besondere Ankündigungen erfolgen normalerweise per E-Mail oder während der zweiwöchentlichen Vorbesprechungen. Ansonsten sind auf der Webseite

<http://www-i6.informatik.rwth-aachen.de/web/Teaching/LabCourses/SS06/Softwarepraktikum>

aktuelle Informationen abgelegt.

Die Aufgaben sind in Zweiergruppen zu lösen. Die Bearbeitungszeit für einen Aufgabenkomplex beträgt zwei Wochen. Die Programme müssen im CIP-Pool unter Nutzung der GNU-Compiler gcc bzw. g++ laufen. Für das Praktikum ist der rote Raum mittwochs 17-21 Uhr und donnerstags 14-18 Uhr reserviert. Anwesenheitspflicht im CIP-Pool besteht jedoch nur während der Abgabe der Aufgaben.

Aufgabe:

Zur Erinnerung: Beim *Hashing* wird ein Datensatz bezüglich seines Schlüssels $k \in \mathcal{K}$ in einem linearen Feld mit Indizes $0, \dots, m - 1$ gespeichert. Dieses Feld bezeichnet man als *Hashtabelle* der Größe m . Die *Hashfunktion* $h : \mathcal{K} \rightarrow \{0, \dots, m - 1\}$ ordnet jedem Schlüssel k einen Index $h(k)$ mit $0 \leq h(k) \leq m - 1$ zu (sog. *Hashadresse*). Im Allgemeinen ist h nicht injektiv, so daß Adresskollisionen auftreten können.

1. Schreiben Sie ein Programm, welches einen beliebigen Text einliest und dabei ein Wörterbuch erstellt. Verwenden Sie hierzu eine Hashtabelle, welche auf Zeichenketten arbeitet. Die Größe der Hashtabelle sei $m = 1999$. Überlegen Sie sich hierzu eine geeignete Hashfunktion. Verwenden Sie in Ihrer Implementierung Hashing mit Verkettung der Überläufer (*direct chaining hashing*), indem Sie die Einträge mit gleichem Schlüssel in verketteten Listen festhalten. Testen Sie Ihr Programm auch mit der Textdatei `80days`, die Sie auf unserer Website unter der anfangs genannten Adresse finden.
2. Implementieren Sie dieselbe Aufgabe mit Hashing, d.h. im Falle einer Kollision sollen die Datensätze unmittelbar in der Hashtabelle untergebracht werden. Verwenden Sie im Kollisionsfall einmal lineares Sondieren ($h(k), h(k) - 1, h(k) - 2, \dots$) und einmal quadratisches Sondieren ($h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots$).
3. Schreiben Sie ein zugehöriges Makefile, das beim Aufruf von `chainingHashing` ein Programm mit Namen `chainingHashing` erzeugt. Der Aufruf von `make clean` soll alle automatisch erzeugten Dateien löschen. Erweitern Sie das Makefile um die Targets `Hashing1` und `Hashing2`.

4. Erzeugen Sie auf Grundlage Ihrer Hashtabelle eine vollständige Menge aller Wörter des Beispieltextes. Geben Sie zusätzlich die Schlüssel der folgenden Wörter aus: `Bombay`, `caprice`, `watch`.
5. Führen Sie eine Laufzeitmessung Ihrer Programme durch. Starten Sie hierzu Ihre Programme in der `bash` mittels des Kommandos `time <Progname> [Parameter]`.
6. Führen Sie nun ein Profiling durch. Kompilieren Sie hierzu Ihre Programme mit den Flags `-pg`. Starten Sie anschließend Ihr Programm. Es wird zusätzlich eine Datei `gmon` erzeugt. Starten Sie nun das Programm erneut (ohne Argumente) wie folgt: `gprof <Progname>`. Falls Sie einen C++ Compiler verwendet haben, filtern Sie die Ausgabe mittels `c++filt`. Welches ist der geschwindigkeitsbestimmende Schritt in Ihrem Programm? Versuchen Sie, Ihr Programm zu beschleunigen.

Abnahmetermin: Donnerstag, 20. April, ab 14:00 Uhr

Schriftliche Ausarbeitungen werden nicht verlangt. Schicken Sie bitte Ihre kommentierten Quelltexte bereits bis Mittwoch Abend (19. April, 18:00 Uhr) an

bender@informatik.rwth-aachen.de.

Am Donnerstag erläutern Sie uns dann Ihre Lösungen und demonstrieren Ihre Programme im CIP-Pool.