Algorithmik

Funke/Bahrdt/Krumpe/Mendel/Seybold SS 2015

http://www.fmi.informatik.uni-stuttgart.de/alg

Institut für Formale
Methoden der Informatik
Universität Stuttgart

Übungsblatt 2

Punkte: **50**

Problem 1 - Untere Schranke für vergleichsbasiertes Closest Pair (10 P)
Das Element Distinctness Problem ist das folgende Entscheidungsproblem:

Geg.: Eine Liste von Zahlen: $x_1, \dots x_n \in \mathbb{R}$.

Ges.: Gibt es ein Paar $i \neq j$ mit $x_i = x_j$?

Es ist bekannt, dass jeder vergleichsbasierte Algorithmus für dieses Problem mindestens $\Omega(n \log n)$ Operationen benötigt[1]. Zeigen Sie, dass jeder vergleichsbasierte Algorithmus für das Berechnungsproblem Closest Pair auch mindestens $\Omega(n \log n)$ Operationen benötigt.

Problem 2 - Landau Symbole (10 P)

Es sei folgendes Programmfragment gegeben, das Routinen A_1, A_2, A_3, A_4, A_5 und A_6 mit Laufzeiten $t_1(n)$, $t_2(n)$, $t_3(n)$, $t_4(n)$, $t_5(n)$ und $t_6(n)$ definiert. Geben Sie das asymptotische Wachstum dieser Laufzeiten in Θ-Notation an und begründen Sie jeweils Ihre Antwort. Die Laufzeit sei hierbei durch die Anzahl der ausgeführten arithmetischen Operationen gegeben.

```
function A_1(n):
  x := 0;
  for i = 1 to 5 · n do
    for j = i to 3 · n do
      x := x + 1 od od;
  return x

function A_2(n):
  if n ≤ 2 then
    return 1 fi;
  return A_2(n − 2) + n
```

```
function A_3(n):
  x := A_1(n);
  if n ≤ 1 then
    return n fi;
  return x + A_3(n − 1)

function A_4(n, a):
  x := a;
  if n ≤ 1 then
    return a fi;
  for i = 1 to 7 do
    x := x + A_4(⌊n/5⌋, i) od;
  return x
```

```
function A_5(n, a):
  if n = 0 then
    return a fi;
  return A_5(n − 1, 1) +
         A_5(n − 1, 2) +
         A_5(n − 1, 3)

function A_6(n):
  if n < 6 then
    return 0 fi;
  return 2 + A_6(⌊n/8⌋)
```

Problem 3 - Kargers Algorithmus (10 P)

Beweisen Sie, dass der in der Vorlesung vorgestellte randomisierte MinCut Algorithmus (Kargers Algorithmus) eine Laufzeit von $\mathcal{O}(n^2)$ hat.
Geben Sie dazu im Detail an, wie man einen Kontraktionsschritt in $\mathcal{O}(n)$ ausführen kann:

- Welche Datenstruktur verwenden Sie?
- Wie wählen Sie die Kante aus? Wie finden Sie die zugehörigen Knoten?
- Wie kontrahieren Sie die Knoten? Was müssen Sie in Ihrer Datenstruktur ändern?

Problem 4 - Markov Ungleichung (10 P)

Beweisen Sie die Markov Ungleichung für eine nicht negative Zufallsvariable X und ein $a > 0$:

$$\Pr(X \geq a) \leq \frac{E(X)}{a}.$$

Problem 5 - Closest Pair Implementierung (3 + 7 P)

Sie können alle Implementierungsaufgaben grundsätzlich in der Sprache Ihrer Wahl lösen, allerdings muss Ihr Tutor die Implementierung im Zweifelsfall verstehen. Außerdem muss es einen freien Compiler für Ubuntu geben, der die Sprache compiliert. Völlig problemlos sind also C, C++, Java oder Ada. Das Testsystem läuft mit Ubuntu 12.04 oder neueren.

Erstellen Sie ein Programm, welches für eine Eingabeinstanz aus n Punkten $p_0, p_1, \ldots, p_{n-1}$ im $\mathbb{N}^2$ dasjenige Punktepaar (i, j) zurück gibt, welches den minimalen euklidischen Abstand realisiert. Es soll außerdem den Abstand sowie die Punktekoordinaten dieser Beiden ausgeben.
Die Eingabeinstanz ist eine Textdatei aus $2n$ integer Werten mit einem Wert pro Zeile, wobei die x, y Koordinate von p_k in Zeile $2k$ und $2k + 1$ stehen. Alle Werte sind aus dem Bereich $0 \leq x, y < 2^{22}$.
Eine Instanz aus 3 Punkten würde also wie folgt aussehen:

```
1035638
2411038
1930588
3969569
66805
165280
```

Und die 3 Punkte $p_0 = (1035638, 2411038), p_1 = (1930588, 3969569)$ und $p_2 = (66805, 165280)$ enthalten.
Den Kleinsten Abstand haben p_0 und p_1 mit $\text{dist}(p_0, p_1) = \sqrt{(1035638 - 1930588)^2 + (2411038 - 3969569)^2} \approx 1797207.384$.

Auf der Algorithmik Webseite finden Sie zwei Dateien, von denen Sie **eine** benötigen:

(a) `instances.tgz`

(b) `generator.tgz`

`instances.tgz` ist groß (7MB) und enthält direkt `small.txt` und `large.txt`.
`generator.tgz` ist sehr klein und enthält ein C++ Programm sowie ein Makefile, welches diese beiden Instanzen direkt generiert.

Lösen Sie die beiden Instanzen `small.txt` (1000 Punkte) und `large.txt` (1000000 Punkte) und geben Sie sowohl die **Ausgabe** Ihres Programmes, sowie den **Quellcode** ab.
Zu dem Quellcode gehört ein Shellscript/Makefile, welches Ihr Programm compiliert und auf den beiden Instanzen ausführt. Geben Sie die Instanzen selber **nicht** mit ab. Gehen Sie stattdessen davon aus, dass diese im gleichen Ordner wie Ihre Programm liegen.

Eine Anmerkung:
Im Optimalfall ist Ihre abgegebene Ausgabe korrekt. In diesem Fall werden wir Ihren Quellcode nur grob auf Plausibilität überprüfen. Ihr Programm sollte auf dem Computer des Tutors in akzeptabler Zeit terminieren und keine Libraries verwenden, die schon die geforderte Funktionalität mitbringen. Sollte Ihre Ausgabe nicht korrekt sein, hängt es von der Qualität Ihres Codes ab, inwieweit wir diesen inspizieren werden.

References

[1] Ben-Or, Michael. *Lower Bounds for Algebraic Computation Trees.* Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, p80–86, 1983.