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 3

Punkte: **60**

Problem 1 - Tschebyscheff Ungleichung (5 P.)

Sei X eine Zufallsvariable mit einem Erwartungswert μ und einer endlichen Varianz σ^2. Dann gilt für alle reellen Zahlen $k > 0$:

$$\Pr(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}.$$

Beweisen Sie diese Aussage.

Problem 2 - Las Vegas Algorithmen (10 P)

Sie haben einen Monte Carlo Algorithmus `A(n)`, der in $f(n)$ Zeit mit einer Wahrscheinlichkeit von $p(n)$ das korrekte Ergebnis liefert und möchten daraus einen Las Vegas Algorithmus `B(n)` entwerfen.
Mit einem Verifikator `V(n)` können Sie das Ergebnis von `A(n)` in $g(n)$ Zeit auf Korrektheit überprüfen.
Ihr Algorithmus `B(n)` führt dazu so lange wiederholt `A(n)` und dann `V(n)` auf dessen Ausgabe aus, bis `V(n)` Korrektheit zertifiziert.

Wie hoch ist die Erwartete Laufzeit von `B(n)`?
Beweisen Sie.

Problem 3 - Pi per Monte-Carlo (10 P.)

Folgender Monte-Carlo Algorithmus approximiert $\pi/4$:

```
1: procedure MC-PI(N)
2:     k ← 0
3:     for i ← 1 to N do
4:         pick p u.a.r. from [0,1) × [0,1)
5:         if |p| < 1 then
6:             k ← k + 1
7:         end if
8:     end for
9:     return k/N
10: end procedure
```

Schätzen Sie die Wahrscheinlichkeit $P(|\text{MC-PI}(n) - \pi/4| > \varepsilon)$ sowohl mit der Tschebyscheff- als auch der Chernoff-Ungleichung [1][2] nach oben ab. Modellieren Sie hierzu den Algorithmus als Bernoulli-Prozess. Berechnen Sie auch alle Terme, die für die Ungleichungen benötigt werden (z.B. Varianzen).
Wie schnell nimmt der Fehler mit größerem n ab? Wie groß müssen Sie n wählen um $\pi/4$ mit einer Wahrscheinlichkeit von mindestens 95% auf 0.1% genau zu approximieren?
Welche der beiden Schrank ist schärfer?

Problem 4 - Quicksort (7 P.) Zeigen Sie, dass es für jede deterministische Pivotselektierungstrategie, die das Pivotelement unabhängig von der Eingabe wählt, eine Eingabe gibt, für die dieser Quicksort $O(n^2)$ Vergleiche benötigt. Nehmen Sie an, dass in jedem Partitionierungsschritt die verbleibenden Elemente nur mit dem Pivotelement verglichen werden.

Problem 5 - Algorithmus von Kruskal (5 + 3 + 2 P.)

Gegeben sei folgender Algorithmus (von Kruskal):

```
INPUT: Graph G(V, E), f(e aus E) -> R

mst_kanten = SET(); //Menge von Kanten
mst_wald = SET(); //Menge von Mengen von Knoten
for vertex in V:
    mst_wald.insert( SET(vertex) )
sort(E) //sortiere E nach Kantengewicht
while E.size() > 0:
    e(u,w) = E.first() //Kante kleinsten Gewichts aus E
    E.remove(e) //entferne e aus E
    U = mst_wald.find_set(u) //finde Menge, die u enthaelt
    W = mst_wald.find_set(w) //finde Menge, die w enthaelt
    if U != W:
        mst_kanten.insert(e)
        mst_wald.replace(U, W, union(U, W)) //ersetze U und W durch deren Vereinigung
```

Und der Graph $G(V, E)$ mit Kantenfunktion $f : E \mapsto \mathbb{N}$ mit:
$V = \{a, b, c, d, e, f, g, h, i\}$,
Kanten $e_i, i \in \{1 \dots 14\}$ mit Gewicht: $e_1 = (\{a, b\}, 4)$, $e_2 = (\{a, h\}, 8)$, $e_3 = (\{b, h\}, 11)$, $e_4 = (\{b, c\}, 8)$, $e_5 = (\{h, i\}, 7)$, $e_6 = (\{g, h\}, 1)$, $e_7 = (\{c, i\}, 2)$, $e_8 = (\{g, i\}, 6)$, $e_9 = (\{c, d\}, 7)$, $e_{10} = (\{c, f\}, 4)$, $e_{11} = (\{f, g\}, 2)$, $e_{12} = (\{d, f\}, 14)$, $e_{13} = (\{d, e\}, 9)$ $e_{14} = (\{e, f\}, 10)$

a) Führen Sie den Algorithmus auf dem gegeben Graphen aus und geben Sie den MST und sein Gewicht an.

b) Welche Bedeutung hat die Abfrage U != W? Was bedeutet es, wenn U = W wäre?

c) Welche Laufzeit hat dieser Algorithmus in Abhängigkeit von find_set, insert, union und replace?

Problem 6 - Münzwurf (8 P.) Gegeben sei ein Bernoulli-Experiment (Münzwurf) mit Erfolgswahrscheinlichkeit p. Geben Sie den Erwartungswert für die Anzahl der Würfe bis ein Erfolg eintritt an.

Problem 7 - MinCut 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 kompiliert. Völlig problemlos sind also C, C++, Java oder Ada. Das Testsystem läuft mit Ubuntu 12.04 oder neueren.

Implementieren Sie den randomisierten MinCut Algorithmus (Kargers Algorithmus) von Aufgabenblatt 2, Aufgabe 3, der für einen ungerichteten, ungewichteten, schleifenfreien Multigraphen $G = (V, E)$ mit $|V| = n$ und $|E| = m$ die Größe des gefundenen MinCuts zurück gibt.
Implementieren Sie einen Algorithmus `LoopCut(k, G=(V,E))`, der den MinCut Algorithmus k mal wiederholt und in jeder Runde die Größe des gefundenen MinCuts, sowie das bisherige Optimum zurück gibt.

Die Eingabeinstanz ist eine Textdatei aus $2m$ integer Werten mit einem Wert pro Zeile, wobei source und target von Kante k in Zeile $2k$ und $2k + 1$ stehen. Alle Werte sind aus dem Bereich $0 \leq s, t < n$.
Hinweis: Es können in der Eingabe Kanten der Form $e_k = (x, x)$ vorkommen. Fügen Sie diese ***nicht*** *zum Graphen hinzu.*

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

(a) `instances.tgz`

(b) `generator.tgz`

`instances.tgz` ist groß (2.5MB) 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` ($n = 32$, $m = 100$) und `large.txt` ($n = 256$, $m = 1000000$) und geben Sie sowohl die **Ausgabe** Ihres Programms, sowie den **Quellcode** ab. Lassen Sie den `LoopCut` Algorithmus dazu mit $k = \frac{n}{2}$ laufen.
Zu dem Quellcode gehört ein Shellscript/Makefile, welches Ihr Programm kompiliert 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] http://de.wikipedia.org/wiki/Tschebyscheff-Ungleichung

[2] http://de.wikipedia.org/wiki/Chernoff-Ungleichung