Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
Y. Maus, O. Saukh

01 Juli, 2016

Algorithmen und Datenstrukturen
Sommersemester 2016
Übungsblatt 10

Abgabe bis 12:00, Freitag, 15 Juli, 2016

Achtung: Bitte fügen Sie Ihrer Lösung eine (kurze) Datei *erfahrungen.txt* mit Ihren Erfahrungen mit dem jeweiligen Aufgabenblatt hinzu. Das Forum können (und sollen) Sie nicht nur für Fragen zur Übung, sondern auch für Fragen zur Vorlesung benutzen.
Achtung: Für dieses Übungsblatt haben Sie zwei Wochen Zeit.

Aufgabe 1: Kürzeste Wege zur Arbeit (über den Berg) (12 Punkte)

Geben Sie jeweils einen Algorithmus an, um die folgenden Probleme zu lösen (dabei dürfen in der Vorlesung behandelte Algorithmen als Blackbox verwendet werden):
Gegeben sei ein Straßennetz als ein ungerichteter, gewichteter Graph $G = (V, E, w)$, $n = |V|$, $m = |E|$.

(a) (2 Punkt) Sie wohnen in Punkt $a \in V$ und arbeiten in Punkt $b \in V$. Wie berechnen Sie den kürzesten Weg zur Arbeit?

Geben Sie außerdem die asymptotische Laufzeit Ihres Algorithmus (mit Begründung) an.

(b) (4 Punkte) Sie möchten umziehen in eine neue Wohnung $a' \in W$, wobei $W \subseteq V$ eine Menge von potentiellen Wohnungen ist. Wie finden Sie a', wenn diese von allen Wohnungen in W den kürzesten Weg zur Arbeit $b \in V$ haben soll?

Geben Sie außerdem die asymptotische Laufzeit Ihres Algorithmus an (mit Begründung).

Hinweis: Ihre Lösung soll besser sein als $|W|$ mal die Lösung aus a zu wiederholen.

(c) (6 Punkte) Sie haben sich wegen der schönen Lage für eine andere Wohnung $c \in V$ mit Fahrradnähe zu $b \in V$ entschieden. Jeder Knoten v im Graphen hat eine Höhe $h_v \in \mathbb{R}$ (der Einfachheit halber seien alle Knotenhöhen voneinander verschieden) und je nachdem, in welcher Richtung man über eine Kante $e = \{u, v\}$ geht, ist diese eine Bergaufkante oder eine Bergabkante. Aus unbekanntem Grund wollen Sie von Ihrer Wohnung c aus zunächst ausschließlich bergauf fahren – bis zu einem Punkt $x \in V$ – und von dort aus ausschließlich bergab bis zum Punkt b.

Beschreiben Sie einen (effizienten) Algorithmus, welcher herausfindet, ob so ein Weg existiert und welcher bei Existenz den kürzesten aller solchen Wege, inklusive des höchsten Punkts x, zurückgibt. Welche Laufzeit hat Ihr Algorithmus (ohne Begr"undung)?

Aufgabe 2: Bellmann-Ford Variation (10 Punkte)

Betrachten Sie den Algorithmus 1, *Fast-Bellman-Ford*, welcher als Eingabe einen *gerichteten*, *gewichteten* und *kreisfreien* Graphen $G = (V, E, w)$ sowie $s \in V$ erhält (w ist hierbei die Gewichtsfunktion $w : E \mapsto \mathbb{R}$, mit erlaubten negativen Gewichten).

Algorithm 1 Fast-Bellman-Ford

```
Sortiere V topologisch und speichere die sortierte Liste im Array B ab.
for all v ∈ V do
    δ(s, v) ← ∞
    parent(v) ← NIL
δ(s, s) = 0; i = 1
while i ≤ n do
    u ← B[i]
    for all out-neighbors v of u do
        if δ(s, u) + w(u, v) < δ(s, v) then
            δ(s, v) = δ(s, u) + w(u, v)
    i ← i + 1
```

Welche Laufzeit hat der Algorithmus? Begründen Sie Ihre Antwort. Argumentieren Sie zudem, dass der Algorithmus tatsächlich alle kürzesten Pfade berechnet.
Hinweis: Überlegen Sie, inwieweit die topologische Sortierung hilft und welche Invariante der Algorithmus dadurch aufrechterhält.

Aufgabe 3: Pretty Print (18 Punkte)

In der Vorlesung wurde ein Algorithmus vorgestellt, der *optimale*[1] Zeilenumbrüche für die Blocksatzausgabe eines Textes berechnet. Wir wiederholen den Algorithmus kurz: Die Wörter sind in einem Array *text[]* gespeichert, so dass z.B. *text[3]* das dritte (bzw. je nach Implementierung vierte) Wort des Textes als String ist. Wenn eine Zeile die Worte i bis $j-1$ enthält so definieren wir *badness(i, j)* durch

$$badness(i, j) = (W - chars - (words - 1))^3,$$

sofern die Wörter i bis $j-1$ in eine Zeile passen. Falls Sie nicht in eine Zeile passen so gilt $badness(i, j) = \infty$. Hierbei ist W die Zeilenlänge, $chars$ die Gesamtzeichenanzahl der Wörter i bis $j-1$ und $words - 1$ die Anzahl der Abstände zwischen den Wörtern.
Um die *badness* des gesamten Textes zu minimieren muss die Summe der $badness'$ über alle Zeilen im Text minimiert werden. Sei $blocksatz(i)$ die optimale *badness* für einen Blocksatz der Worte ab dem Wort i, falls man mit dem i-ten Wort eine neue Zeile beginnt. Dann gilt die folgende Rekursion:

$$blocksatz(n) = 0 \tag{1}$$
$$blocksatz(i) = \min_{j>i}(badness(i, j) + blocksatz(j)), \tag{2}$$

Der Algorithmus 2 beschreibt in Pseudocode wie die optimalen Zeilenumbruchspositionen dynamisch berechnet werden können.

[1] Die Optimalität bezieht sich immer auf die jeweils genuzte *badness*-Funktion

Algorithm 2 Blocksatz

```
memo = {}; nextline = {}
if i in memo then
    return memo[i]
if i ≥ len(text) then
    cost = 0; return cost;
b = badness(i, i+1)
cost = ∞
j = i+1
min_idx = i+1
while j ≤ n and b < ∞ do
    c = b + blocksatz(j)
    if c < cost then
        cost = c;
        min_index = j;
    j += 1;
    b = badness(i, j);
memo[i] = cost;
nextline[i] = min_idx;
```

In dieser Übung sollen Sie den Blocksatz Algorithmus implementieren und auf den Text in der Datei *input.txt* anwenden. Gehen Sie dabei nach den unten stehenden Teilaufgaben vor (Reihenfolge beliebig).
Eine Zeile soll 81 Zeichen enthalten (80 Buchstaben/Leerzeichen und "$\backslash n$" als Zeilenumbruch am Ende). Der Sonderfall, dass ein Wort mehr Zeichen hat als eine Zeile enthalten darf, muss nicht betrachtet werden.

a) (4 Punkte) Implementieren Sie die Funktion $badness(i, j, l)$, welche, abhängig von der Zeilenlänge l, und dem Wortarray die *badness* einer Zeile zurück gibt, die mit dem i-ten Wort beginnt und mit Wort $j - 1$ endet.

b) (8 Punkte) Implementieren Sie den dynamischen Algorithmus, die Funktion *blocksatz*. Berechnen Sie die optimalen Positionen der Zeilenumbrüche und die Gesamtbadness. Fügen Sie Ihre Gesamtbadness der *erfahrungen.txt* hinzu.

   Hinweis: Sie können gerne mit einer anderen badness-Definition experimentieren und die Unterschiede betrachten.

c) (6 Punkte) Damit der ausgebene Text gut aussieht, dŕfen sich die Abstände zwischen Wörtern in einer Zeile nicht um mehr als 1 unterscheiden, d.h. wenn x und y die Grössen beliebiger Abstände einder Zeile sind, so soll $|x - y| \leq 1$ gelten.

   Implementieren Sie die Funktion $pretty\_print(i, j, l)$, welche die Wörter von i bis $j - 1$ in ein Stringobjekt schreibt und die Abstände der richtigen Grösse zwischen Wörter einfügt. Die Methode muss nicht ordnungsgemäß funktionieren, falls die Wörter inklusive der notwendigen Leerzeichen nicht in eine Zeile passen.

   Lesen Sie die Datei *input.txt*, nutzen Sie Aufgabe *a*) und *b*) und schreiben das Text im optimalen Blocksatz in die Date *output.txt*. Laden Sie die Datei *output.txt* in den SVN.

Hinweis: Im SVN befinden Sich noch weitere Dateien, welche zum Testen benuzt werden können.