technische universität dortmund

SoSe 2018

M. Buchin
A. Krivošija, D. Mezlaf
L. Pfahler, A. Rey

DAP2 – Heimübung 11

Ausgabedatum: 15.06.2018 — Abgabedatum: Mo. 25.06.2018 bis 12 Uhr

Abgabe:

Schreiben Sie unbedingt immer Ihren vollständigen Namen, Ihre Matrikelnummer und Ihre Gruppennummer auf Ihre Abgaben! Beweise sind nur dort notwendig, wo explizit danach gefragt wird. Eine Begründung der Antwort wird allerdings *immer* verlangt.

Aufgabe 11.1 (4 Punkte): (Breitensuche: Durchmesser eines Graphen)

Sei $G = (V, E)$ ein ungerichteter Graph. Für $s, t \in V$ bezeichnet $\delta(s, t)$ die Länge des kürzesten Weges in G von s nach t. Der *Durchmesser* d des Graphen G ist nun definiert als das Maximum der Längen kürzester Wege über alle Knotenpaare $s, t \in V$:

$$d := \max\{\delta(s, t) \mid s, t \in V\}$$

Bob erinnert sich aus seiner Zeit als Studierender an die Breitensuche und formuliert folgenden Algorithmus, um den Durchmesser eines zusammenhängenden Graphen G in Laufzeit $\mathcal{O}(|V| + |E|)$ zu ermitteln:

`BerechneDurchmesser`**(Graph** $G = (V, E)$**):**

```
1  s ← beliebiger Knoten aus V
2  "Initialisiere BFS mit Startknoten s"
3  while Q ≠ ∅ do
4      u ← head[Q]
5      foreach v ∈ Adj[u] do
6          if color[v] = weiß then
7              color[v] ← grau
8              d[v] ← d[u] + 1
9              enqueue(Q, v)
10     dequeue(Q)
11     color[u] ← schwarz
12 return max{d[v] | v ∈ V}
```

a) Liefert dieser Algorithmus tatsächlich für jeden Graphen G den korrekten Durchmesser? Falls ja, begründen Sie Ihre Antwort. Falls Nein, konstruieren Sie ein Gegenbeispiel.

b) Wie kann Bobs Algorithmus angepasst werden, um in Laufzeit $\mathcal{O}(|V| \cdot (|V| + |E|))$ den Durchmesser eines Graphen $G = (V, E)$ zu ermitteln?

Aufgabe 11.2 (6 Punkte + 4 Bonuspunkte): (Breitensuche: Mehr Breitensuche)

In dieser Aufgabe sollen Algorithmen für ungerichtete Graphen konstruiert werden, die sich die Vorgehensweise der Breitensuche zunutze machen. Beide Algorithmen sollen beschrieben und in Pseudocode formuliert werden, und beide Algorithmen sollen bei Eingabe eines Graphen $G = (V, E)$ eine worst-case-Laufzeit von $\mathcal{O}(|V| + |E|)$ haben. Um zu begründen, dass diese Laufzeitschranke eingehalten wird, ist keine vollständige Laufzeitanalyse nötig, sondern lediglich eine Betrachtung der jeweiligen Anpassungen der Breitensuche. Sie dürfen für diese Aufgabe annehmen, dass der gegebene Graph G zusammenhängend ist.

a) Geben Sie einen Algorithmus `istBaum(`G`)` an, der genau dann `TRUE` ausgibt, wenn der eingegebene Graph G ein Baum, also kreisfrei ist. Sonst soll der Algorithmus `FALSE` ausgeben.

b) Ein Graph $G = (V, E)$ heißt *bipartit*, wenn die Knotenmenge V in zwei Teilmengen L und R partitioniert[1] werden kann, sodass es keine Kante $(u, v) \in E$ gibt, für die beide Knoten u und v in L oder beide Knoten in R liegen.

   Geben Sie einen Algorithmus `istBipartit(`G`)` an, der genau dann `TRUE` ausgibt, wenn der eingegebene Graph G bipartit ist. Sonst soll der Algorithmus `FALSE` ausgeben.

c) **(Bonus)** Beweisen Sie die Korrektheit der von Ihnen gegebenen Algorithmen.

[1] Die Knotenmenge V in zwei Teilmengen $L \subseteq V$ und $R \subseteq V$ zu partitionieren, bedeutet, dass die Teilmengen L und R die Eigenschaften $L \cap R = \emptyset$ und $L \cup R = V$ erfüllen. Für jeden Knoten $v \in V$ gilt also **genau eine** der Aussagen $v \in L$ oder $v \in R$.