Datenstrukturen und Algorithmen
Heimübung 10

Eli Kogan-Wang (7251030)
David Noah Stamm (7249709)
Daniel Heins (7213874)
Tim Wolf (7269381)

18. Juni 2022

Aufgabe 1

Für die Adjazenzlistendarstellung:
Sei G_{new} ein neuer leerer Graph in Adjazenzlistendarstellung. Er wird im Laufe des Algorithmus mit den Einträgen des Transponierten Graphen G^T erweitert.
Für jeden Knoten v aus $V(G)$: ($|V|$ mal)
Wir fügen v zu G_{new} hinzu. ($O(1)$)
Für jeden Knoten v aus $V(G)$: ($|V|$ mal)
Wir iterieren über die ursprüngliche Adjazenzliste $A(v)$ mit v_{adj} adjazent zu v: ($|A(v)|$ mal)
Wir fügen zur Adjazenzliste von v_{adj} den Knoten v hinzu. ($O(1)$)
Nun ist G_{new} eine Adjazenzlistendarstellung von G^T.
Bemerkung: $|V| \cdot |A(v)| = O(|E|)$
Das heißt wir üben $O(|V| + |E|)$ Elementare Operationen aus.
Der Algorithmus ist korrekt, weil er alle und nur diese Kanten aus G^T dem Graphen G_{new} hinzufügt.

Für die Adjazenzmatrixdarstellung:
Sei G_{new} ein neuer kantenloser Graph in Adjazenzmatrixdarstellung über dieselben Knoten V aus G.
Nun sei A die Adjazenzmatrix von G und A_{new} die Adjazenzmatrix von G_{new}.
Nun für $1 \leq i \leq |V|$: ($O(|V|)$) Und $1 \leq j \leq |V|$: ($O(|V|)$)
$A_{new}(j, i) = A(i, j)$ ($O(1)$)
Nun ist $A_{new} = A^T$ und G_{new} eine Adjazenzmatrixdarstellung von G^T.
Der Algorithmus übt $O(|V|^2)$ Elementare Operationen aus.
Die Korrektheit des Algorithmus lässt sich aus der Vertauschung der Indizes i und j begründen.

Aufgabe 2

Der Algorithmus Bipartite-Check-Breadth-First-Search nimmt einen Graphen G und einen Startknoten $s \in V$.

Algorithm 1 Bipartite-Check-Breadth-First-Search(G, s)

```
1: for jeden Knoten u in V\{s} do
2:     color[u] ← WHITE
3:     π[u] ← NIL
4: color[s] ← GRAY
5: π[s] ← NIL
6: Q ← {}
7: Enqueue(Q, s)
8: while Q ≠ ∅ do
9:     u ← Dequeue(Q)
10:    if color[u] = GRAY then
11:        otherColor ← BLACK
12:    else
13:        otherColor ← GRAY
14:    for jeden Knoten v in A(u) do          ▷ A(u): Nachbarn von u
15:        if color[v] = WHITE then
16:            color[v] ← otherColor
17:            π[v] ← u
18:            Enqueue(Q, v)
19:        else if color[v] ≠ otherColor then
20:            return FALSE
21: Return TRUE
```

Wir reduzieren das Problem der Bipartitheitsprüfung auf den der 2-Färbbarkeit. Mithilfe der Breitensuche könenn wir die 2-Färbung auf G versuchen und bei einem Konflikt abbrechen.
Ein Graph ist genau dann 2-Färbbar, wenn er bipartit ist.
Unser Algorithmus ist damit korrekt, weil true rückgibt, genau dann wenn der Graph 2-Färbbar ist. Und weil er false rückgibt, wenn der Graph nicht 2-Färbbar ist.
Die Laufzeit einer Breitensuche ist aus der Vorlesung mit $O(|V|+|E|)$ Elementaren Operationen bekannt und wurde hierbei nur durch Elementare Operationen Erweitert, weswegen dieser weiterhin in O(|V|+|E|) liegt.

Aufgabe 3

Der vorgeschlagene Algorithmus besteht aus 3 Phasen:
1. Phase:
Aufteilen der Kanten mit Gewichtung $w \neq 1$ in w-Kanten mit Gewichtung 1.

Man fügt zusätzliche Knoten hinzu und ballert dadurch zusätzliche Kanten rein.
Laufzeit: $O(|E| \cdot k)$ (da $w \in [1, k]$)
2. Phase:
Breitensuche
Laufzeit: $O((|V| + |E|) \cdot k)$ da wir Knoten und Kanten hinzugefügt haben.
3. Phase:
Vergessen der neu hinzugefügten Knoten+Kanten.
Wir vergessen die neu hinzugefügten Knoten und Kanten in den gefundenen kürzesten Pfaden.
Laufzeit: $O(|V| \cdot k)$ (da wir nur Ergebnisse für jeden Knoten speichern und maximal k Knoten+Kanten pro Knoten vergessen)
Die Gesamtlaufzeit ist damit $O(|V| + |E| \cdot k)$