Datenstrukturen und Algorithmen: Blatt 11

Bernhard Dietrich (6256800)
Lars Fernhomberg (6256030)
Sebastian Kniesburges (6257120)
Marcus Köthenbürger (6258550)

Übungsgruppe 11
Mittwoch, 11:00-13:00 Uhr in D1.303
Matthias Ernst

Aufgabe	1	2	3	4	5	Σ	Korrektor
Punkte							
von	6	4	6	4	4	24	

Aufgabe 1

Sei $G = (V, E)$ ein gerichteter Graph. Sei $G^T = (V, E^T)$ definiert durch

$$E^T := \{(v,u) \in V \times V : (u,v) \in E\}.$$

G^T heißt der zu G *transponierte* Graph. Beschreiben Sie sowohl für die Adjazenzlisten-Darstellung als auch für die Adjazenzmatrix-Darstellung Algorithmen, die G^T aus G berechnen. Für die Adjazenzlisten-Darstellung soll Ihr Algorithmus Laufzeit $O(|V| + |E|)$ besitzen. Ihr Algorithmus für die Adjazenzmatrix-Darstellung soll Laufzeit $O(|V|^2)$ besitzen.

Adjazenzlisten-Darstellung

Dien Adjazenzliste besteht aus einem Array mit $|V|$ Elementen, welche die einzelnen Knoten des Graphs repräsentieren. In den einzelnen Elementen des Arrays werden verkettete Listen gespeichert, in denen wiederum gespeichert wird, mit welchen Knoten eine gerichtete Verbindung existiert. Dabei sind die Knoten, die im Array gespeichert werden, die Ausgangsknoten und die Knoten, welche in den verketteten Listen gespeichert werden, die Zielknoten, auf die gezeigt wird.
Bei der Transposition einer Adjazenzliste muss nun diese Zuordnung umgekehrt werden, so dass die Zielknoten zu Startknoten werden und die Startknoten zu Zielknoten.
Erreicht werden kann dies durch eine zweite Adjazenzlistenstruktur (B), welche genau wie die Ursprungsliste (A) aufgebaut ist. Wird nun beim Durchlauf der Ursprungsliste eine Zuordnung (u,v) gefunden, wobei u durch das Array repräsentiert wird und v sich in der verketteten Liste befindet, so wird in der neuen Struktur in der Liste, welches sich an v-ter Stelle im Array befindet, ein Verweis auf u eingefügt.

```
CreateTransListe(A)
1.    for i ← 1 to |V|
2.          do while next[A[i]] ≠ Nil
3.                B[key[i]] ← i
```

In der for-Schleife wird jedes Element des Arrays, welches aus den Knoten des Graphs besteht, einmal durchlaufen, so dass $|V|$ Durchläufe erfordern. In der while-Schleife werden anschließend die Verweise, die von einem Knoten abgehen, durchlaufen, wobei es maximal im gesamten Knoten $|E|$ derartige Verweise gibt, so dass die while-Schleife genau $|E|$ mal durchlaufen wird.

Es ergibt sich somit $O\left(|V|+|E|\right)$.

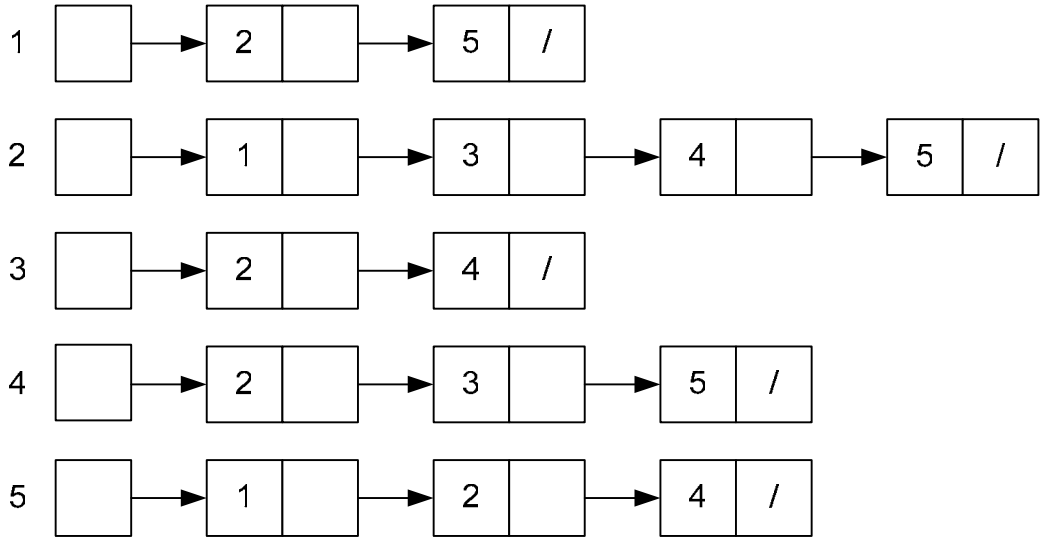
Adjazenzmatrix-Darstellung
Bei einem Graphen, der in Adjazenzmatrix-Darstellung vorliegt, bewirkt die Transposition, dass die Spalten zu Zeilen und die Zeilen zu Spalten werden. Der entsprechende Algorithmus braucht also lediglich die entsprechenden Zahlen zu vertauschen.
Der Einfachheit halber benutzt unser Algorithmus eine zweite Matrix B, in der die transponierte Matrix (A) aus der Eingabematrix erstellt wird, in dem jede Stelle des Graphen (also jede mögliche Kante) einmal abgefragt wird:

```
CreateTransMatrix(A)
1.    for i ← 1 to |V|
2.          do for j ← 1 to |V|
3.                do B(j,i) ← A(i,j)
```

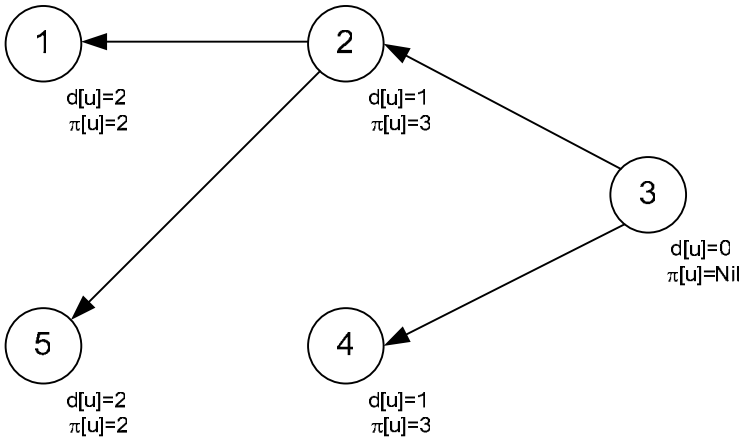
Da eine Adjazenzmatrix die Form $n \times n$ hat, also eine gleiche Anzahl an Zeilen und Spalten besitzt, werden beide Schleifen jeweils n-mal durchlaufen. Es ergibt sich somit $O\left(n \cdot n\right) = O\left(n^2\right)$. Da $n = |V|$, folgt automatisch $O\left(|V|^2\right)$.

Aufgabe 2

Betrachten Sie den Graphen G_2 in Abbildung 1. Der Graph G_1 sein in Adjazenzlisten-Darstellung gegeben. In jeder Adjazenzliste eines Knotens sind die benachbarten Knoten aufsteigend sortiert. Bestimmen Sie den Breitensuch-Baum, wenn Algorithmus BFS mit Graph G_1 und Quelle 3 aufgerufen wird. Bestimmen Sie die Werte $d[u]$ und $\pi[u]$ für jeden Knoten u.
Adjazenzliste für G_1:



Knoten u	$d[u]$	$\pi[u]$
3	0	NIL
2	1	3
4	1	3
1	2	2
5	2	2



Aufgabe 3

Gegeben ist ein zusammenhängender, ungerichteter Graph $G = (V, E)$. Sie möchten die Knoten des Graphen so mit den Farben rot und blau färben, dass Knoten, die durch eine Kante verbunden sind, immer unterschiedliche Farben besitzen. Natürlich ist dieses nicht bei allen Graphen möglich. Wenn es möglich ist, sagen wir, dass der Graph G 2-färbbar ist. Entwerfen Sie einen Algorithmus, der gegeben einen Graphen G entscheidet, ob G 2-färbbar ist. Ihr Algorithmus soll Laufzeit $O(|V| + |E|)$ besitzen.

Der Algorithmus färbt zunächst einen Knoten rot oder blau und fügt diesen Knoten in eine Queue ein. Anschließend wird die Queue abgearbeitet, indem der erste Knoten aus der Queue extrahiert wird, um dessen Nachbarknoten in der jeweils anderen Farbe zu färben und diese dann ebenfalls in die Queue einzufügen. Sollte ein Nachbarknoten bereits gefärbt sein, so wird dieser, wenn er eine andere Farbe als der aktuelle Knoten, haben, nicht mehr in die Queue eingefügt. Wenn der betroffene Nachbarknoten die gleiche Farbe hat, so bedeutet dies, dass der Graph nicht 2-färbbar ist, da es Kreise gibt, die dieses verhindern. Sollte die Queue vor einem Schleifendurchlauf leer sein, so bedeutet dies, dass alle Knoten gefärbt wurden und keine illegale Färbung auftrat, so dass der Graph 2-färbbar ist.

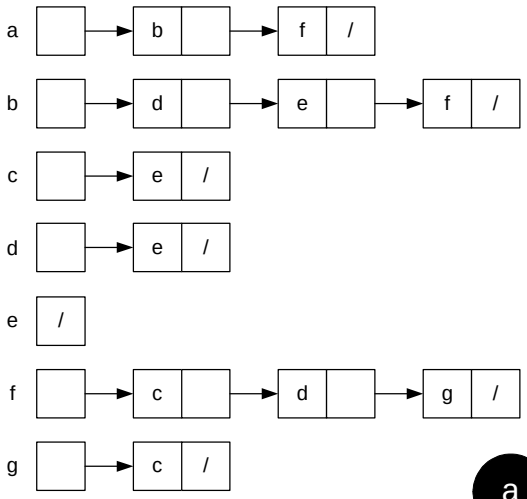
```
ColorGraph(G)
1.    for each Knoten u ∈ V[G]
2.         do   color[u] ← white
3.    u ← zufälliger Knoten aus V[G]
4.    color[u] ← red
5.    Q ← Ø
5.    Enqueue(Q,u)
6.    While Q ≠ Ø
7.         do   u ← Dequeue(Q)
8.              if color[u] = red
9.                   then c ← blue
10.                  else c ← red
11.             for each v ∈ Adj[u]
12.                  if color[v] = white
13.                       then color[v] ← c
14.                            Enqueue[Q,u]
15.                       else if color[v] = c
16.                            then Error „Zwei Knoten mit gleicher
                                          Farbe benachbart!"
```

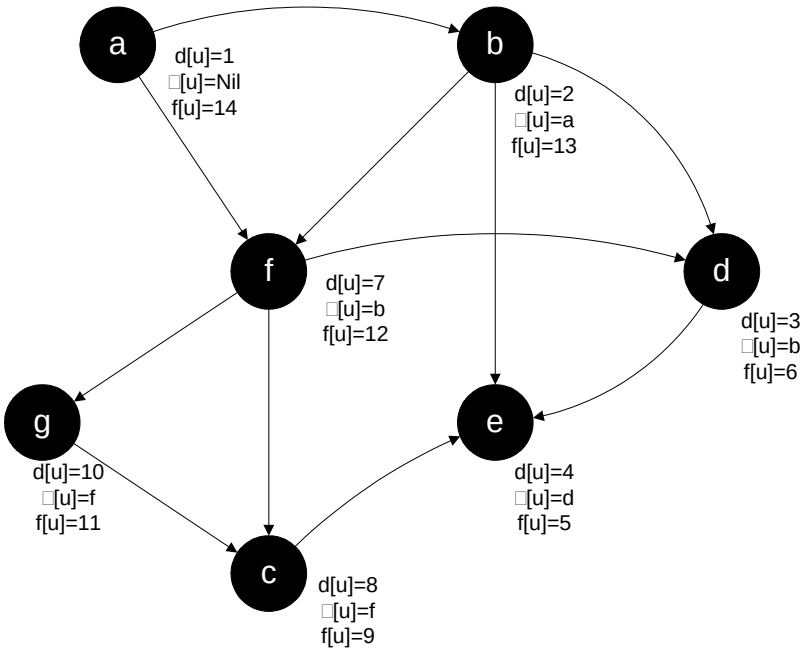
Für die Aufwandsabschätzung sind nur die beiden for-Schleifen und die while-Schleife interessant, da alle anderen Anweisungen eine konstante Laufzeit $O(1)$ haben.

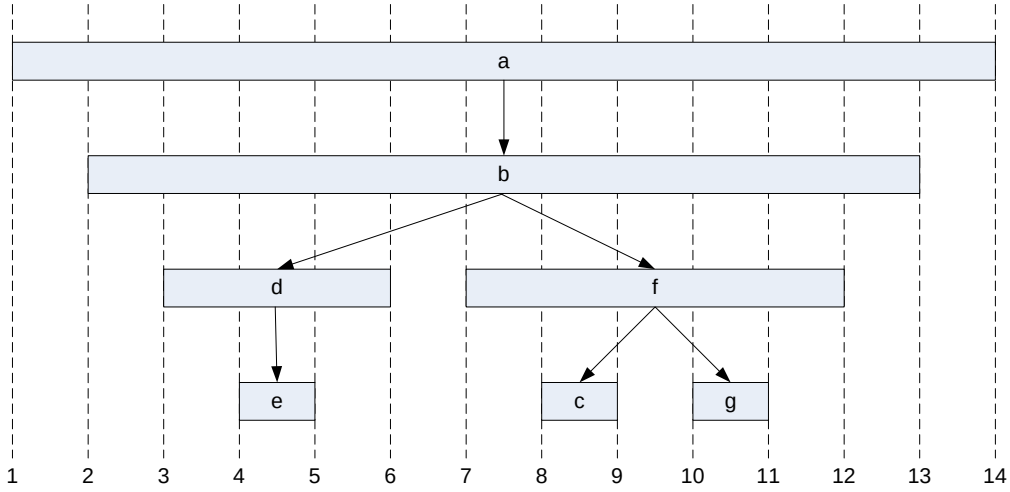
Die erste for-Schleife dient nur zum einfärben aller Knoten in gleicher Farbe und wird $|V|$ mal durchlaufen, sodass $O(|V|)$ gilt.
In der while-Schleife werden sämtliche Knoten einmal durchlaufen, da jeder Knoten einmal eingefärbt werden muss. Somit gilt auch hier $O(|V|)$.
Die innere for-Schleife durchläuft die Adjazenzliste für jeden Knoten. Insgesamt gibt es in allen Adjazenzlisten $|E|$ Einträge, so dass diese Schleife ebenfalls maximal $|E|$-mal durchlaufen wird, so dass sich $O(|E|)$ ergibt. Es ergibt sich $O(|V|)+O(|V|)+O(|E|)=O(2\cdot|V|+|E|)=O(|V|+|E|)$.

Aufgabe 4

Betrachten Sie den gerichteten Graphen G_2 in Abbildung 2. Der Graph G_2 sei in Adjazenzlisten-Darstellung gegeben. Jede Adjazenzliste sei dabei alphabetisch sortiert. Bestimmen Sie den Tiefensuch-Wald, wenn Algorithmus *DFS* mit Graph G_2 aufgerufen wird. In den Zeilen $5-7$ von *DFS* werden dabei die Knoten in alphabetischer Reihenfolge betrachtet.







Aufgabe 5

Der Graph G_2 ist azyklisch. Bestimmen Sie eine topologische Sortierung von G_2.