1 Tiefen- und Breitensuche

Hat man einen Graphen gegeben, dann kommt es vor, dass man ihn irgendwann auch durchlaufen will. Sei es um ihn nach und nach zu färben, oder um einen oder mehrere bestimmte Knoten zu suchen. Hierfür gibt es zwei grundlegende Algorithmen, die man einmal gesehen haben muss, da sie häufig zielführende Ideen liefern.

1.1 Stacks und Queues

Bevor wir aber zu diesen Algorithmen kommen, müssen wir zwei Datenstrukturen wiederholen, die wir gebrauchen können. Im Allgemeinen gilt, je mehr Funktionalitäten eine Datenstruktur zur Verfügung steht, desto mehr Aufwand muss im Hintergrund vom Computer betrieben werden, um diese Funktionalitäten zu gewährleisten. Häufig ist das aber kompletter Overkill, wenn diese Funktionen gar nicht benötigt werden. Stacks und Queues geben uns jeweils nur eine Möglichkeit auf die jeweiligen Daten zuzugreifen und nur eine Möglichkeit Daten hinzuzufügen.

Ein *Stack* (engl. für Stapel) besteht lediglich aus einem obersten Element und einem darunter befindlichen (kleinere) Stapel. Ein Stapel kann dabei auch leer sein. Um ein Element einem Stapel hinzuzufügen, wird das Element auf den Stapel gelegt. Um ein Element von einem Stapel zu entnehmen, wird das oberste Element des Stapels entnommen und vom Stapel entfernt. Das wars. Es ist nicht möglich irgendein anderes Element des Stapels zu nehmen außer dem Obersten. Möchte man das unterste Element nehmen, muss man den gesamten Stapel einmal abräumen. Die Zugriffsbefehle bei einem Stack nennen wir `push(x)` und `pop()`. Der Befehl `push(`x`)` fügt das Element x dem Stapel hinzu, also legt es oben auf ihn drauf. Der Befehl `pop()` entfernt das oberste Element des Stapels und gibt das entsprechende Element zurück. Schreibt man nur `pop()`, dann wird lediglich das Element entfernt, schreibt man y `= pop()`, dann wird das oberste Element entfernt und in der Variable y gespeichert.

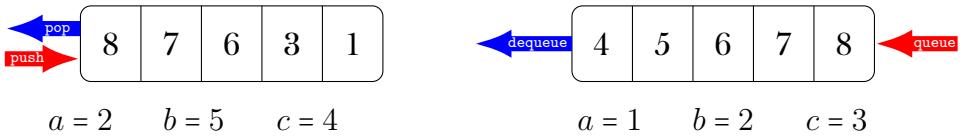
Eine *Queue* (engl. für Warteschlange) besteht aus einem ersten Element und einer dahinterbefindlichen (kleineren) Queue. Der Unterschied zum Stack ist hierbei die Art und Weise, wie Elemente hinzugefügt werden. Der Zugriff erfolgt ähnlich wie beim Stack, das erste Element wird entnommen. Will man ein Element hinzufügen, so wird dieses ans Ende der Queue gepackt (es stellt sich quasi hinten an). Die Zugriffssbefehle nennen wir hier `queue(`x`)` und `dequeue()`. Der Befehl `queue(`x`)` fügt das Element x ans Ende der Queue an. Der Befehl `dequeue()` gibt das erste Element zurück und entfernt es aus der Queue. Auch hier können wir y `= dequeue()` schreiben, um das erste Element in y zu speichern, bevor wir es löschen.

In der Abbildung 1 sieht man beispielhaft die Erzeugung eines Stacks und eines Queues mit den jeweils analogen Befehlen für das Hinzufügen bzw. Entnehmen von Elementen.

Aufgabe 1 Wir schreiben $a_1 \to a_2 \to \ldots \to a_n$ für einen Stack oder einer Queue mit den Elementen a_1 bist a_n. Das vorderste Element ist das oberste in einem Stack bzw. der erste Element einer Queue je nachdem wie wir die Folge interpretieren. Zum Beispiel ergibt sich durch `push(1)`, `push(2)`, `push(3)` der Stack $3 \to 2 \to 1$ und durch `queue(1)`, `queue(2)`,`queue(3)` die Queue $1 \to 2 \to 3$.

```
push(1)
push(2)
a = pop()
push(3)
push(4)
push(5)
b = pop()
c = pop()
push(6)
push(7)
push(8)
```

```
queue(1)
queue(2)
a = dequeue()
queue(3)
queue(4)
queue(5)
b = dequeue()
c = dequeue()
queue(6)
queue(7)
queue(8)
```



$a = 2$ $b = 5$ $c = 4$ $a = 1$ $b = 2$ $c = 3$

Abbildung 1: Stacks und Queues

Transformiere durch möglichst wenig Befehle

(a) den Stack $2 \to 0 \to 2 \to 0$ in den Stack $2 \to 0 \to 2 \to 1$,

(b) die Queue $2 \to 0 \to 2 \to 0$ in die Queue $2 \to 0 \to 2 \to 1$.

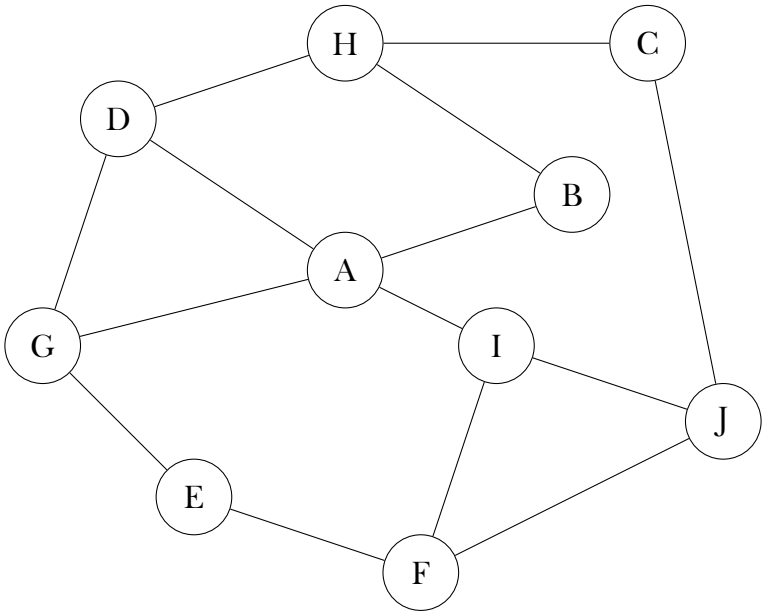
1.2 Breitensuche

Bei der Breitensuche beginnen wir in einem Startknoten als Pivot und färben ihn grau. Danach gehen wir alle Nachbarn des Knotens durch, die noch keine Farbe besitzen und färben sie blau, während wir sie simultan in eine Queue stecken. Jetzt nehmen wir uns den ersten Knoten aus unserer Queue und behandeln ihn als den neuen Pivot. Der Algorithmus endet, wenn alle Element grau gefärbt sind, oder das gesuchte Element gefunden wurde, je nachdem was man erreichen will. Es geht bei der Breitensuche weniger um das Ziel, als um die Art und Weise alle Knoten eines Graphen zu besuchen.

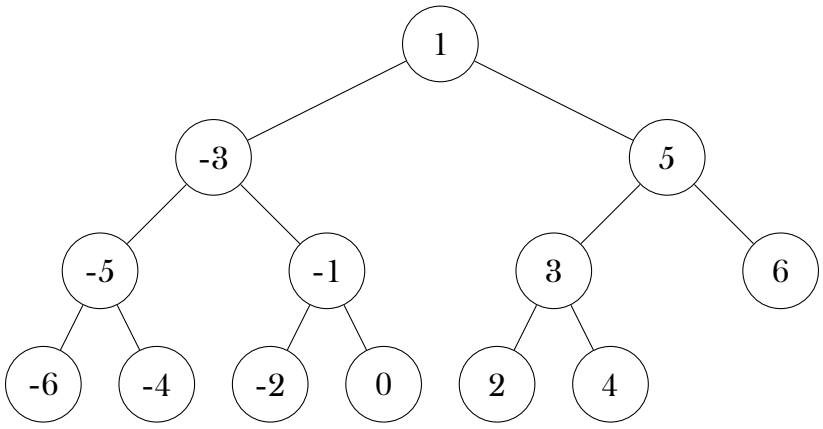
Gegeben : Graph (V, E) und ein Startknoten $v \in V$
Erzeuge eine Queue Q und `queue(v)`
while Q *ist nichtleer* **do**
  Setze p = `dequeue()` und färbe p grau
  foreach w *mit* $\{p, w\} \in E$ **do**
    if w *ist farblos* **then**
      Färbe w blau und `queue(`w`)`
    end
  end
end

Algorithmus 1 : Zentraler Loop der Breitensuche

Aufgabe 2 Vollführe für den folgenden Graphen eine Breitensuche mit Startknoten A. Schreibe als Ergebnis eine Liste auf, wo du angibst in welcher Reihenfolge die Knoten grau gefärbt wurden und zu jedem Knoten angibst, wieviele Knoten in dem Moment grau waren, als der jeweilige Knoten blau gefärbt wurde.



Aufgabe 3 Vollführe für den folgenden Baum eine Breitensuche von der Wurzel ausgehend und suche das Element 2. Als Ergebnis gebe eine Liste mit allen grauen Elementen bis zum Fund der Zahl an. Die Elemente werden von links nach rechts der Queue hinzugefügt.



Aufgabe 4 Die *Tiefensuche* ist vom Algorithmus her beinahe identisch mit der Breitensuche. Der einzige Unterschied ist der, dass anstelle einer Queue ein Stack verwendet wird (und die Befehle `queue` und `dequeue` werden entsprechend durch `push` und `pop` ersetzt).

Erledige nun Aufgabe 2 und 3 erneut, allerdings mit einer Tiefensuche statt einer Breitensuche. Warum heißt wohl Breitensuche Breitensuche und Tiefensuche Tiefensuche?

Zusatzaufgabe Welcher der beiden Algorithmen könnte uns helfen den kürzesten Weg zwischen zwei gegebenen Knoten zu finden?