Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2021 · tcs.uni-frankfurt.de/algo1

Wochenplan: Disjunkte Mengen, Union-Find *Revision 393abf7 (2021-06-08)*

Vorbereitung

Lies CLRS Kapitel 21 ohne 21.4 (oder Algorithms 4ed. Kapitel 1.5) und schau das Video der Woche.

Dienstag

Aufgabe 1 (Union Find von Hand laufen lassen). Betrachte die folgende Sequenz von Operationen: `Init(7)`, `Union(3,4)`, `Union(5,0)`, `Union(4,5)`, `Union(4,3)`, `Union(0,1)`, `Union(2,6)`, `Union(0,4)` und `Union(6,0)`.

a) (einfach) Führe die Sequenz mittels ***Quick Find*** von Hand durch. Zeige, wie die Inhalte des Feldes `id` nach jedem Schritt aussehen. Die Operation `Union`(i,j) verändert `id` hierbei immer für die Menge, die durch i gegeben ist.

b) (einfach) Führe die Sequenz mittels ***Quick Union*** von Hand durch. Zeige, wie der Baum nach jedem Schritt aussieht. Die Operation `Union`(i,j) setzt hierbei immer die Wurzel von Baum von i als ein Kind der Wurzel des Baumes von j.

c) Führe die Sequenz mittels ***Weighted Quick Union*** von Hand durch. Zeige, wie der Baum nach jedem Schritt aussieht. Die Operation `Union`(i,j) setzt hierbei immer die Wurzel von Baum von i als ein Kind der Wurzel des Baumes von j, wenn die Größe der beiden Bäume gleich ist.

d) Zeige das Resultat der ***Pfadverkürzung*** nach einer Operation `Find`(x) in einem der Bäume aus den Beispielen in a) und b), wobei x einmal ein Blatt sein soll, einmal ein interner Knoten von Tiefe 1, und einmal ein interner Knoten mit Höhe 1.

e) Gib eine Sequenz von Operationen an, die in einem Baum maximaler Tiefe resultieren, wenn die abstrakte Datenstruktur durch ***Quick Union*** implementiert ist.

f) Gib eine Sequenz von Operationen an, die in einem Baum maximaler Tiefe resultieren, wenn die abstrakte Datenstruktur durch ***Weighted Quick Union*** implementiert ist.

g) Schreibe den Pseudo-Code für ***Pfadverkürzung***. *Hinweis: Durchlaufe den Pfad zweimal.*

Aufgabe 2 (Alternative zum *Quick Find* Algorithmus). Eine Kommilitonin stellt die folgende, intuitive Variante von ***Quick Find*** `Union` vor. Funktioniert sie?

```
procedure UNION(i, j)
    if FIND(i) ≠ FIND(j) then
        for all k ∈ {0, ..., n − 1} do
            if id[k] == id[i] then
                id[k] = id[j]
```

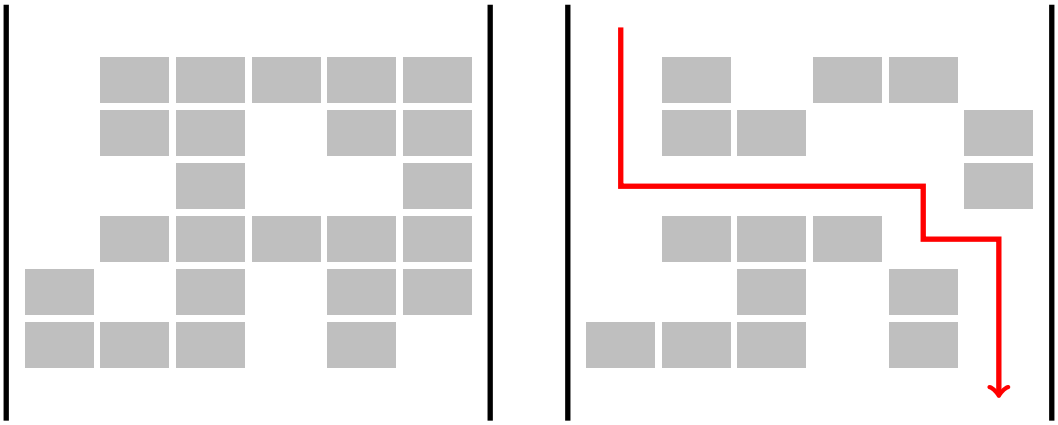
Aufgabe 3 (Dynamische Zusammenhangskomponente und Suche in Graphen). Mit Tiefen- und Breitensuche können wir die Zusammenhangskomponenten eines Graphen finden. Entwirf einen einfache Implementierung der abstrakten Datenstruktur für den dynamischen Zusammenhang (mit den Operationen INIT, CONNECTED, INSERT) mittels Suche in Graphen und vergleiche die Komplexität deiner Lösung mit der auf Union-Find basierenden Lösungen.

Freitag

Aufgabe 4 (Implementierung von Union-Find, alleine probieren). Wir wollen Datenstrukturen für Union-Find (mit den Operationen `Init`, `Union` und `Find`) in einer beliebigen Programmiersprache implementieren.

a) Implementiere ***Quick Union***.
b) Erweitere deine Implementierung mit ***Weighted Quick Union***.
c) Erweitere deine Implementierung mit ***Pfadverkürzung***.

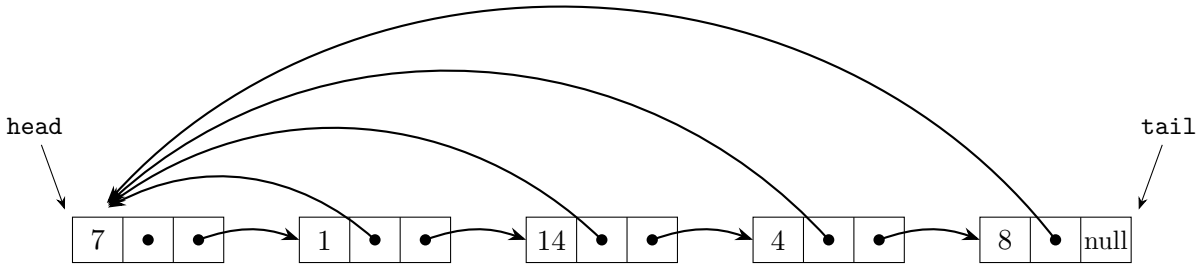
Aufgabe 5 (Zombieinvasion, schwer). Der erkenntnisgierige Prof. Dr. Regloh hat bei unethischen Zombieduellexperimenten versehentlich einige Zombies entkommen lassen und damit die postapokalyptische Zukunft eingeläutet. Du hast dich mit einer kleinen Gruppe an Überlebenden in einem kleinen Gebäude verbarrikadiert. Das Einzige, das zwischen euch und einer brutalen und hungrigen Horde Zombies steht, ist eine starke Befestigung. Die Befestigung besteht aus einem $k \times k$ Gitter von Wänden, hier illustriert durch ein 6×6 Gitter von Wänden (*graue Rechtecke*):



Oberhalb des Gitters warten die Zombies, während deine Gruppe und du unterhalb sind. Unglücklicherweise sind die Wände alt und spröde, und stürzen regelmäßig ein. Sobald ein Pfad von ganz oben nach ganz unten verläuft, kommen die Zombies durch. Um eure Evakuierung vorzubereiten, willst du durchgehend überwachen, ob es derzeit einen Pfad durch die Befestigung gibt. Entwirf eine Datenstruktur, die effizient den Überblick behält, während die Wände (Zellen des Gitters) eine nach der anderen einstürzen.

Aufgabe 6 (Rekursive *Pfadverkürzung*, schwer). Entwirf eine rekursive Variante von ***Pfadverkürzung*** in Pseudo-Code.

Aufgabe 7 (Union-Find mit verketteten Listen und Gewichtungen). Wir wollen eine Variante von ***Quick Find*** mittels verketteten Listen auf die folgende Art und Weise implementieren. Jede Menge wird durch eine einfach verkettete Liste repräsentiert. Der Repräsentant jeder Menge ist das erste Element der jeweiligen Liste und jedes Element der Liste hat einen Zeiger auf den Repräsentanten. Des Weiteren haben wir einen Zeiger `tail` auf das letzte Element der Liste. Zum Beispiel könnte die Datenstruktur für die Menge $\{1, 4, 7, 8, 14\}$ mit Repräsentant 7 wie folgt aussehen:



a) Zeige wie man mit dieser Darstellung der Mengen die Operationen `Init(`n`)` in Zeit $O(n)$, `Find(`i`)` in Zeit $O(1)$ und `Union(`i, j`)` in Zeit $O(|S(i)|)$ implementieren kann, wobei $S(i)$ diejenige Menge ist, die i enthält.

b) Zeige, wie man die Lösung erweitern kann, sodass `Init` und `Find` dieselbe Zeitkomplexität wie zuvor haben, aber `Union(`i, j`)` nun nur Zeit $O(\min(|S(i)|, |S(j)|))$ benötigt. *Hinweis: Speichere ein paar Zusatzinformationen.*

c) (schwer) Zeige, dass für die Lösung aus b) jede Sequenz von p `Find` und m `Union` Operationen auf n Elementen eine Laufzeit von $O(p + m \log n)$ hat. Zu Beginn der Sequenz sind alle Mengen einelementig.