Algorithmen und Datenstrukturen 1

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

⭐-Aufgabe: Union Find Move

Revision 364b09b (2021-06-02)

Entwickle eine **möglichst effiziente** Implementierung der folgenden abstrakten Datenstruktur: Verwalte eine Familie von disjunkten Mengen, anfangs die einelementigen Mengen

$$\{0\}, \{1\}, \ldots, \{n-1\},$$

unter den folgenden Operationen:

0 („query“) Die *query* Operation nimmt zwei Argumente s und t, und ermittelt, ob s und t zur selben Menge gehören. Das heißt, wenn $s \in S$ and $t \in T$, dann gibt die Operation `1` aus falls $S = T$ und `0` wenn $S \neq T$.

1 („union“) Die *union* Operation nimmt zwei Argumente s und t, und erzeugt die Vereinigung der beiden Mengen, die s und t enthalten. Das heißt, wenn $s \in S$ und $t \in T$ mit $S \neq T$, dann sollen S und T aus der Familie entfernt und durch die Menge $S \cup T$ ersetzt werden. (Wenn $S = T$, dann passiert nichts.)

2 („move“) Die *move* Operation nimmt zwei Argumente s und t, und verschiebt das Element s in die Menge, die t enthält. Das heißt, wenn $s \in S$ und $t \in T$ mit $S \neq T$, dann sollen S und T aus der Familie entfernt werden und stattdessen die Mengen $S - \{s\}$ (falls diese nichtleer ist) und $T \cup \{s\}$ zur Familie hinzugefügt werden. (Wenn $S = T$, dann passiert nichts.)

Diese Operationen erhalten die Invariante, dass die Mengen in der Familie disjunkt sind.

a) Beschreibe deine Datenstruktur und analysiere sie
b) Implementiere und teste deine Datenstruktur gemäß der Anforderungen in Abschnitt A.

Hinweise zur Abgabe. Mindestens a) muss bearbeitet werden. Die Union-Find Datenstrukturen aus der Vorlesung reichen hier nicht aus, sondern müssen modifiziert werden, daher ist eine klare und anschauliche Beschreibung der Idee besonders wichtig. Kleine Bildchen wären nützlich, um den Zustand deiner Datenstruktur zu verstehen. Erwartet werden ansonsten wie üblich die grobe Idee, Pseudocode oder echter Code, Korrektheitsbeweis, Laufzeitanalyse.

A. Implementierung

Für die Implementierung soll folgendes Ein- und Ausgabeverhalten unterstützt werden.

Eingabe

Die Eingabe startet mit zwei natürlichen Zahlen n und m in der ersten Zeile. Die Zahl n ist die Anzahl an einelementigen Mengen zu Beginn. Dann folgen m Zeilen der Form „`0` s t“ (für *query*) oder „`1` s t“ (für *union*) oder „`2` s t“ (für *move*). Du kannst $0 \leq s < n$ und $0 \leq t < n$ annehmen.

Ausgabe

Für jede *query* Operation wird wie oben beschrieben eine Zeile mit `1` oder `0` ausgegeben.

Beispiel

`example.in`	Zustand der Datenstruktur nach der Operation
`4 9`	$\{0\},\{1\},\{2\},\{3\}$
`0 0 1`	
`1 0 1`	$\{0,1\},\{2\},\{3\}$
`0 0 1`	
`1 1 2`	$\{0,1,2\},\{3\}$
`0 1 2`	
`0 0 3`	
`2 2 3`	$\{0,1\},\{2,3\}$
`0 0 1`	
`0 1 2`	

`example.ans`

```
0
1
1
0
1
0
```

Mehr Tests

Hier gibt es größere Tests:
https://tcs.uni-frankfurt.de/teaching/summer21/algo1/unionfindmove-tests-v2.zip

Die darin enthaltene Datei `032-huge.in` sollte gelöst werden. Ein effizienter Algorithmus braucht auf einem handelsüblichen Laptop nicht lange:

```
$ time python oursolution.py < 032-huge.in > /dev/null

------------------------------------------------------
Executed in    8.85 secs    fish           external
   usr time    8.79 secs  822.00 micros    8.79 secs
   sys time    0.06 secs  309.00 micros    0.06 secs
```

Liefern Sie die SHA-1 Summe von `032-huge.ans`.