Ausgewählte Kapitel ADS

October 29, 2015

1 Datenstrukturen für Mengen

1.1 Union-Find-Problem

Verwaltung von diskunkten Mengen

Problem

Verwalte eine Partition (Zerlegung in disjunkte Teilmengen) der Menge {1,...,n} unter folgenden Operationen. Jede Teilmenge (Block) besitzt einen eindeutigen Namen aus {1,..,n}.

- FIND(x): $x \in \{1, .., n\}$ Liefert den Namen der Teilmenge, die x enthält
- UNION(A,B,C): Vereinigt die Teilmengen mit Namen A und B zu einer Teilmenge mit dem Namen C.

Initialisierung

Wir starten mit der Partitionierung: $\{\{1\}, .., \{n\}\}$ mit dem Namen i für $\{i\}$,$1 \leq i \leq n$
Analyse: Kosten für 1 Union (worst case)
Amortisiert: Kosten für $n - 1$ mögliche UNIONs
$\rightarrow$ Kosten von $n - 1$ UNIONs und m FINDs

Lösungen

1. Lösung (einfach)
Verwende ein Feld name[1..n] mit name[x] = Name des Blocks der x enthält. $1 \leq x \leq n$

```
for i=1 to n do
        name[i] <- i
od
```

FIND(x): return name[x] : $\mathcal{O}(n)$
UNION(A,B,C): $\mathcal{O}(n)$

```
for i=1 to n do
        if name[i] = A OR name[i] = B
        then name[i] <- C
        fi
od
```

Gesamtlaufzeit (Lemma 1):
$n - 1$ UNIONs und m FINDs kosten $\mathcal{O}(n^2 + m)$
2. Lösung (Verbesserung)
1. Find unverändert
2. Ändere den Namen der kleineren Menge in den Namen der größeren (Relabel the smaller half)
Zusätzliche Felder:

- size[1..n]: size[A] = Anzahl Elemente im Block A, initialisiert mit 1
- L[1..n]: L[A] = Liste aller Elemente in Block A, initialisiert L[i] = {i}

FIND(x) bleibt gleich
UNION(A,B):

```
if size[A] ≤ size[B]
then
          forall i in L[A] do
                    name[i] ← B
          od
          size[B] += size[A]
          L[B] ← L[B] concatenate L[C]
else
          symmetrisch
```

Die Menge heißt jetzt A oder B
Effekt: UNION(A,B,..) hat Laufzeit $\mathcal{O}(min(|A|,|B|))$
Worst Case eines UNION dieser Folge von UNIONs: $\mathcal{O}(\frac{n}{2}) = \mathcal{O}(n)$ (kann nur einmal vorkommen)
Wie oft kann sich name[x] für ein bestimmtes $x : 1 \leq x \leq n$ ändern?
Beobachtung:

- Am Anfang ist jedes Element x in einer ein-elementigen Menge
- Am Ende sind alle Elemente in einer Menge der Größe n
- Immer wenn ein Element x seinen Namen ändert befindet es sich danach in einer doppelt so großen Menge (nach dem UNION)

$\Rightarrow$ Jedes Element $x \in \{1, .., \}$ kann maximal $log(n)$ mal seinen Namen ändern.
<u>Satz 1:</u> Bei UNION-FIND mit "Relabel the smaller half" sind die Gesamtkosten einer beliebigen Folge von n-1 UNIONS und m Finds $\mathcal{O}(m + n * log(n))$
Im Schnitt (amortisiert) kostet ein UNION $log(n)$

3. Lösung

Lösung 1 und 2 haben FIND effizient gelöst, hier UNION
Jeder Block wird als Baum dargestellt. Die Knoten repräsentieren die Elemente des Blocks. In der Wurzel steht der Name des Blocks.
UNION(A,B,E): Mache die Wurzel von A zum Kind der Wurzel von B und nenne die Wurzel um in E.
FIND(x): Starte bei Element (Knoten) x und laufe bis zur Wurzel, dort steht der Name $\rightarrow \mathcal{O}(Tiefe\ von\ x)$

Realisierung der Datenstruktur durch Felder:

$$vater[i] = \begin{cases} Vater\ von\ i\ in\ seinem\ Baum \\ 0,\ falls\ i\ Wurzel \end{cases}$$

$name[i]$ =Name des Blocks mit Wurzel i (at nur Bedeutung, falls i Wurzel)
$wurzel[i]$ = Wurzel des Blocks mit Namen i

Initialisierung:

```
for i=1 to n do
          vater[i] = 0
          name[i] = i
          wurzel[i] = i
od
```

FIND(x):

```
while vater[x] != 0 do
          x = vater[x]
od
return name[x]
```

UNION(A,B,C):

```
r1 = wurzel[A]
r2 = wurzel[B]
vater[r1] = r2
name[r2] = C
wurzel[C] = r2
```

Analyse:

- UNION: $\mathcal{O}(1)$ worst case
- FIND(x): Tiefe von x (max Höhe des entstehenden Baums, n-1 möglich)

4. Lösung (Weighted Union rule):

Vermeide große Tiefen, dafür hänge den kleineren Baum (Anzahl Knoten) an den größeren
Alternativ: Hänge den Baum mit kleinerer höhe an den tieferen.

Realisierung: Zusätzliches Feld
size[i] = Anzahl Knoten um Unterbaum mit Wurzel i

Initialisierung:

```
for i=1 to n do
        vater[i] = 0
        name[i] = i
        wurzel[i] = i
        size[i] = 1
od
```

FIND(x) (wie bei 3):

```
while vater[x] != 0 do
        x = vater[x]
od
return name[x]
```

UNION(A,B,C):

```
r1 = wurzel[A]
r2 = wurzel[B]
if size[r1] |$\leq $| size[r2] then
        vater[r1] = r2
        name[r2] = C
        wurzel[C] = r2
        size[r2] += size[r1]
else
        symmetrisch
```

Zeige Laufzeit $\mathcal{O}(log(n))$:
Sei für jeden Knoten x die höhe(x) die Hohe von x in seinem Baum (maximale Pfad zu Blatt), Blatt=0
size(x): Anzahl der Knoten im Unterbaum mit Wurzel x (Gewicht)
Lemma: Bei weighted Union rule gilt stets, dass $size(x) \geq 2^{\text{höhe}(x)}$ für alle Knoten x.
Beweis: Induktion über höhe(x):

Vorasussetzung:
    höhe(x) = 0: x ist Blatt -> size(x)=1=2^0
Anfang:
    size(y) $\geq 2^{\text{höhe}(y)}$
Schritt:
    Sei höhe(x) > 0
    Sei y ein Kind von x mit höhe(x)−1
    Betrachte die UNION Operation bei der x zum Vater von y wurde.
    Seien $\overline{size}(x)$ und $\overline{size}(y)$ die Gewichte vor der UNION Operaion, dann gilt:
    1) size(y) = $\overline{size}(y)$, da sich das Gewicht nur für Wurzeln ändern kann
    2) $\overline{size}(x) \geq \overline{size}(y)$ durch weighted union rule
    3) Nach der Operation: $size(x) \geq \overline{size}(x) + \overline{size}(y)$
        $\geq 2 * \overline{size}(y)$ wegen 2.
        $\geq 2 * size(y)$ wegen 1.
        $\geq 2 * 2^{\text{höhe}(y)}$ nach IA
        $= 2^{\text{höhe}(y)+1} = 2^{\text{höhe}(x)}$