Ausgewählte Kapitel ADS

November 5, 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
→ 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
        $= 2^{\text{höhe}(y)+1} = 2^{\text{höhe}(x)}$

Da Anzahl der Knoten $n \Rightarrow size(x) \leq n$ gilt:
$\Rightarrow n \geq size(x) \geq 2^{hoehe(x)}$ für alle x
$hoehe(x) \leq log(n)$

Satz: Bei UNION-FIND mit weighted UNION ist die Laufzeit einer beliebigen Folge n-1 Unions und m Finds $\mathcal{O}(n + log(n))$
Beweis: 1. UNION $\mathcal{O}(1)$ worst-case, 2. Find $\mathcal{O}(log(n))$ worst case (Lemma)

5. Lösung (Verbesserung von FIND):

Pfad-Komprimierung (path compression)
Ein FIND(x) durchläuft den Pfad von x zur Wurzel.
$x = x_0, ..., x_l = Wurzel$
Idee: Hänge $x_0, ..., x_{l-1}$ direkt an die Wurzel an.
Erhöht die Kosten dieses Finds um einen konstanten Faktor.
Algorithmus:

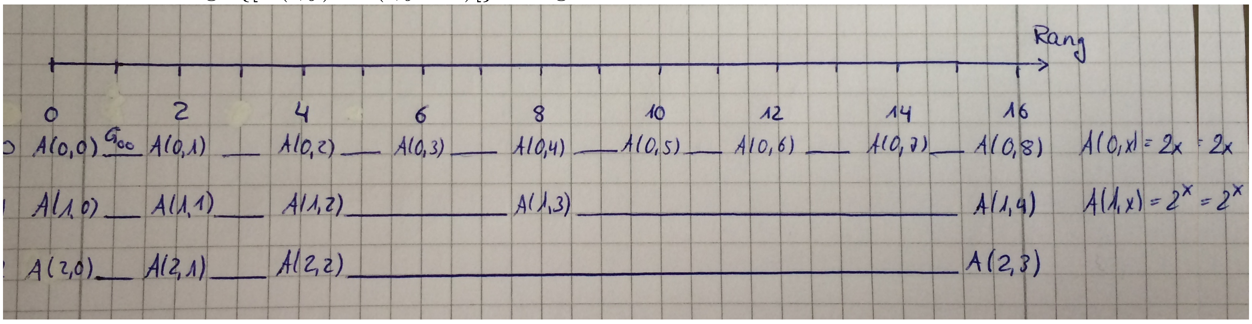
```
FIND(x)
        r <- x;
        while vater[r] != 0 do
                r <- Vater[r]
        od
        while x != r do
                y <- vater[x]
```

```
                vater[x] <- r
                x <= y
        od
```

Ganz klar: $\mathcal{O}(log(n))$ worst case
Satz (Tarjan): Bei UNION-FIND mit weighted UNION und path compression hat eine beliebige Folge von n-1 Unions und m Finds mit $m \geq n$, die Gesamtkosten $\mathcal{O}(m * \alpha(m,n))$,
wobei $\alpha(m,n) = min\{z \in \mathbb{N} | A(z, \frac{4m}{n}) > logn\}$
mit A einer Variante der Ackermannfunktion.
α ist eine Art Inverse der Ackermannfunktion $\Rightarrow$ Ist extrem langsam wachsend.
Definition von A:
$A : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$
$A(i,0) = 0$ für alle $i \in \mathbb{N}_0$
$A(0,x) = 2x$ für alle $x \geq 1$
$A(i,1) = 2$
$A(i,x) = A(i-1, A(i,x-1))$ für $i \geq 1, x \geq 2$

$$\text{A(i,x) als Matrix } i \times x: \begin{array}{cccccc} 0 & 2 & 4 & 6 & 8 & 10 \\ 0 & 2 & 4 & 8 & 16 & 32 \\ 0 & 2 & 4 & 16 & 65536 & 2^{65536} \\ 0 & 2 & 4 & 65536 & 2 \uparrow\uparrow 65536 & . \\ 0 & 2 & . & . & . & . \end{array}$$

1. Zeile: $A(0,x) = 2x$; 2. Zeile: $A(1,x) = 2^x$; 3. Zeile: $A(2,x) = 2^{2^{\cdot^{\cdot^{2}}}}$
Anmerkung: Pfeilschreibweise=(Knuth Up-Arrow)
Beweis des Satzes:
Situation: n Elemente {1,..,n}, beliebige Folge von n-1 Unions und m Finds: $U_1, F_1, F_2, U_2, ...$
Am Ende: 1 Baum T' (n-1 weighted Unions)
Konzeptuell kann T' anders erhalten werden: Führe zunächst alle Unions aus $\to$ Baum T. Dann führe m partielle Finds auf T aus $(PF_1, ..., PF_m)$, die genau den selben Pfad wir F_i durchlaufen, bis zu ihrere ursprünglichen Wurzel vor den Unions.
Wir schätzen nun die Gesamtkosten dieser Folge (insbesondere der m PF's) ab.
Frage: Wieviele Vater-Verweise (Kanten) werden insgesamt durchlaufen?
Sei F=Multi-Menge aller durch die PF's durchlaufenen Kanten (mit Mehrfachen)
Zu zeigen: $|F| = \mathcal{O}(m * \alpha(m,n))$
Idee:
a) Teile F in Gruppen nach Rang der Endpunkte der Kanten, wobei Rang(x)=Höhe(x) im Baum T (nicht T')
b) Schätze die Gruppen einzeln ab.
Zunächst einteilung der Knoten in die Gruppen nach Rang (nicht disjunkt).
Sei $z \in \mathbb{N}_0$, Für $0 \leq i \leq z, j \geq 0$ sei:
$G_{ij} = \{Knoten\ x | A(i,j) \leq Rang(x) < A(i,j+1)\}$
Veranschaulichung: $\{[A(i,j)...A(i,j+1)[\}$ Menge von Ineraktionen



Beispiel: Rang(x)=7, Rang(y)=13
$\Rightarrow x \in G_{0,3}, G_{1,2}, G_{2,2}, ...$
$\Rightarrow y \in G_{0,6}, G_{1,3}, G_{2,2}, ...$
Eine Einteilung der Multi-Menge F
Für $0 \leq k \leq z : N_k = \{(x,y) \in F | k = min\{i \geq 0 | \exists j\ mit\ x, y \in G_{i,j}\}\}$
und $N_{z+1} := F \setminus \cup_{0 \leq i \leq z} N_i$

Schließlich definieren wir für $0 \leq k \leq z+1 : L_k = \{(x,y) \in N_k | (x,y)$ist letzte (oberste) Kante auf PF-Pfad

<u>Lemma:</u>

a) $|L_k| \leq m$ für $0 \leq k \leq z+1$

b) $|N_0 \setminus L_0| \leq n$

c) $|N_k \setminus L_k| \leq \frac{5}{8}n$ für $1 \leq k \leq z$

d) $|N_{z+1} \setminus L_{z+1}| \leq n * a(z,n)$ mit $a(z,n) = min\{i \geq 0 | A(z,i) > log(n)\}$

Beweis:

a) Für jedes PF gibt es höchstens 1 Kante in L_k. Die Behauptung folgt daraus, dass es insgesamt nur m PFs gibt.

b) Sei $(x,y) \in N_0 \setminus L_0$, dann gilt $\exists j \geq 0$ mit $x, y \in G_{0,j}$, das heißt $A(0,j) \leq Rang(x) < Rang(y) < A(0,j+1)$

$\Rightarrow Rang(x) = 2j, Rang(y) = 2j+1$

$(x,y) \notin L_0 \Rightarrow$ nicht die letzte Kante in diesem PF: Betrachte PF von (x,y), dann existiert eine Kante $(s,t) \in L_0$ auf diesem PF-Pfad

Situation:

$Rang(x) = 2j, Rang(y) = 2j+1$

$Rang(s) \geq Rang(y)$ da letzter Pfad vor dem nicht letzten sein muss

$Rang(t) > Rang(s)$ da Pfad von s nach t

$\Rightarrow Rang(t) \geq 2j+2$

Nach dem PF: x hat neuen Vater (möglicherweise t) u mit $Rang(u) \geq Rang(t) \geq 2k+2$

$\Rightarrow$ Rangdifferenz zwischen x und dem neuen Vater u ≥ 2

$\Rightarrow$ Spätere PFs können keine Kante (x,..) mehr zu N_0 hinzufügen

$\Rightarrow$ Für jeden Knoten wird maximal eine ausgehende Kante (Vaterverweis) in $N_0 \setminus L_0$ gezählt werden.

$\Rightarrow |N_0 \setminus L_0| \leq n$

Beweis c), d):

Idee: Schätze Beitrag eines Knotens $x \in G_{k,j}$ zu $N_k \setminus L_k$ d.h. alle Kanten, die von x ausgehen und in $N_k \setminus L_k$ gezählt werden.

Sei $k \geq 1$ und $x \in G_{k,j}$ beliebig, d.h. $\exists j$ mit $A(k,j) \leq Rang(x) < A(k,j+1)$

und $y_1, ..., y_q$ alle Endknoten mit $(x, y_i) \in N_k \setminus L_k$. Ziel: q nach oben abschätzen.

$\Rightarrow Rang(y_1) \leq ... \leq Rang(y_q) < A(k,j+1)$

Beobachtungen:

1) $j \geq 2$ weil sonst k=0 die minimale Zeile definiert, sodass (x, y_i) im selben Intervall (de ersten 3 Spalten sind immer gleich gefüllt mit 0,2,4). Hier: $k \geq 1$

2) $(x, y_i) \notin L_k$ für $1 \leq i \leq q \Rightarrow \exists (s_i, t_i) \in N_k$ auf PF-Pfad von (x, y_i) oberhalb von (x, y_i)