

Ausgewählte Kapitel ADS

November 12, 2015

1 Datenstrukturen für Mengen

1.1 Union-Find-Problem

Verwaltung von diskunkten Mengen

Problem

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

- FIND(x): $x \in \{1, \dots, n\}$ 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: $\{\{1\}, \dots, \{n\}\}$ 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 $\text{name}[1..n]$ mit $\text{name}[x] = \text{Name des Blocks der } x$ 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
```

Gesamlaufzeit (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:

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

$\text{FIND}(x)$ bleibt gleich

$\text{UNION}(A,B)$:

```

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: $\text{UNION}(A,B,..)$ hat Laufzeit $\mathcal{O}(\min(|A|, |B|))$

Worst Case eines UNION dieser Folge von UNIONS: $\mathcal{O}\left(\frac{n}{2}\right) = \mathcal{O}(n)$ (kann nur einmal vorkommen)

Wie oft kann sich $\text{name}[x]$ 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)

⇒ Jedes Element $x \in \{1, \dots, n\}$ kann maximal $\log(n)$ mal seinen Namen ändern.

Satz 1: 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.

$\text{UNION}(A,B,E)$: Mache die Wurzel von A zum Kind der Wurzel von B und nenne die Wurzel um in E.

$\text{FIND}(x)$: Starte bei Element (Knoten) x und laufe bis zur Wurzel, dort steht der Name → $\mathcal{O}(\text{Tiefe von } x)$

Realisierung der Datenstruktur durch Felder:

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

$\text{name}[i] = \text{Name des Blocks mit Wurzel } i$ (at nur Bedeutung, falls i Wurzel)

$\text{wurzel}[i] = \text{Wurzel des Blocks mit Namen } i$

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

$\text{size}[i]$ = Anzahl Knoten um Unterbaum mit Wurzel i

Initialisierung:

FIND(x) (wie bei 3):

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

Laufzeit $\mathcal{O}(\log(n))$:

Sei für jeden Knoten x die $\text{höhe}(x)$ die Höhe von x in seinem Baum (maximale Pfad zu Blatt), Blatt=0
 $\text{size}(x)$: Anzahl der Knoten im Unterbaum mit Wurzel x (Gewicht)

Lemma: Bei weighted Union rule gilt stets, dass $\text{size}(x) \geq 2^{\text{höhe}(x)}$ für alle Knoten x.

Beweis: Induktion über $\text{höhe}(x)$:

Voraussetzung:

$$\text{höhe}(x) = 0: x \text{ ist Blatt} \rightarrow \text{size}(x) = 1 = 2^0$$

Anfang:

$$\text{size}(y) \geq 2^{\text{höhe}(y)}$$

Schritt:

Sei $\text{höhe}(x) > 0$

Sei y ein Kind von x mit $\text{höhe}(x)-1$

Betrachte die UNION Operation bei der x zum Vater von y wurde.

Seien $\overline{\text{size}}(x)$ und $\overline{\text{size}}(y)$ die Gewichte vor der UNION Operation, dann gilt:

1) $\text{size}(y) = \overline{\text{size}}(y)$, da sich das Gewicht nur für Wurzeln ändern kann

2) $\overline{\text{size}}(x) \geq \text{size}(y)$ durch weighted union rule

3) Nach der Operation: $\text{size}(x) \geq \overline{\text{size}}(x) + \overline{\text{size}}(y)$

$$\geq 2 * \overline{\text{size}}(y) \text{ wegen 2.}$$

$$\geq 2 * \overline{\text{size}}(y) \text{ wegen 1.}$$

$$\geq 2 * 2^{\text{höhe}(y)} \text{ nach IA}$$

$$= 2^{\text{höhe}(y)+1} = 2^{\text{höhe}(x)}$$

Da Anzahl der Knoten $n \Rightarrow \text{size}(x) \leq n$ gilt:

$$\Rightarrow n \geq \text{size}(x) \geq 2^{\text{höhe}(x)} \text{ für alle } x$$

$$\text{höhe}(x) \leq \log(n)$$

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.

$x = x_0, \dots, x_l = \text{Wurzel}$

Idee: Hänge x_0, \dots, 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]
```

UNION(A,B,C):

```
r1 = wurzel[A]
r2 = wurzel[B]
if size[r1] ≤ size[r2] then
    vater[r1] = r2
    name[r2] = C
    wurzel[C] = r2
    size[r2] += size[r1]
else
    symmetrisch
```

```

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 $\alpha(m, n) = \min\{z \in \mathbb{N} | A(z, \frac{4m}{n}) > \log n\}$

mit A einer Variante der Ackermannfunktion.

α ist eine Art Inverse der Ackermannfunktion \Rightarrow Ist extrem langsam wachsend.

Definition von A:

$$A : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$$

$$A(i, 0) = 0 \text{ für alle } i \in \mathbb{N}_0$$

$$A(0, x) = 2x \text{ für alle } x \geq 1$$

$$A(i, 1) = 2$$

$$A(i, x) = A(i - 1, A(i, x - 1)) \text{ für } i \geq 1, x \geq 2$$

$$\begin{array}{cccccc} 0 & 2 & 4 & 6 & 8 & 10 \\ 0 & 2 & 4 & 8 & 16 & 32 \end{array}$$

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

$$1. \text{ Zeile: } A(0, x) = 2x; 2. \text{ Zeile: } A(1, x) = 2^x; 3. \text{ Zeile: } A(2, x) = 2^{2^x}$$

Anmerkung: Pfeilschreibweise=(Knuth Up-Arrow)

Beweis des Satzes:

Situation: n Elemente {1,...,n}, beliebige Folge von n-1 Unions und m Finds: $U_1, F_1, F_2, U_2, \dots$

Am Ende: 1 Baum T' (n-1 weighted Unions)

Konzeptuell kann T' anders erhalten werden: Führe zunächst alle Unions aus \rightarrow Baum T. Dann führe m partielle Finds auf T aus (PF_1, \dots, PF_m), die genau den selben Pfad wir F_i durchlaufen, bis zu ihrer 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 durchlaufenden Kanten (mit Mehrfachen)

Zu zeigen: $|F| = \mathcal{O}(m * \alpha(m, n))$

Idee:

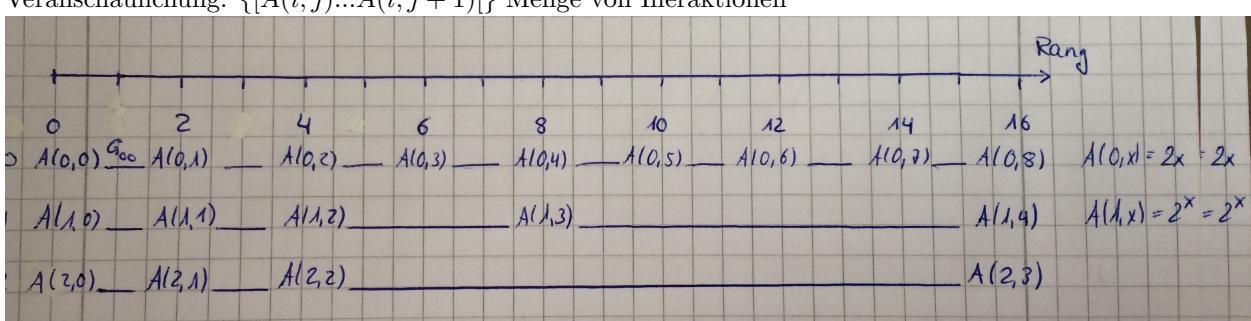
- Teile F in Gruppen nach Rang der Endpunkte der Kanten, wobei $\text{Rang}(x)=\text{Höhe}(x)$ im Baum T' (nicht T)
- Schätzt die Gruppen einzeln ab.

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:

$$G_{i,j} = \{Knoten x | A(i, j) \leq \text{Rang}(x) < A(i, j + 1)\}$$

Veranschaulichung: $\{[A(i, j) \dots A(i, j + 1)]\}$ Menge von Interaktionen



Beispiel: $\text{Rang}(x)=7$, $\text{Rang}(y)=13$

$\Rightarrow x \in G_{0,3}, G_{1,2}, G_{2,2}, \dots$

$\Rightarrow y \in G_{0,6}, G_{1,3}, G_{2,2}, \dots$

Eine Einteilung der Multi-Menge F

Für $0 \leq k \leq z$: $N_k = \{(x, y) \in F | k = \min\{i \geq 0 | \exists j \text{ mit } x, y \in G_{i,j}\}\}$

und $N_{z+1} := F \setminus \bigcup_{0 \leq i \leq z} N_i$

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

- 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 $Rang(u) \geq Rang(t) \geq 2k + 2$

$$\Rightarrow \text{Rangdifferenz zwischen x und dem neuen Vater u} \geq 2$$

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

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

Beobachtungen:

1) $j \geq 2$ weil sonst k=0 die minimale Zeile definiert, sodass (x, y_i) im selben Intervall (die 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) Nach der Pfadkopplermierung ist Vater von $x = y_i + 1$. Außerdem ist y_{i+1} Vorfahr von t_i dabei ist $t_i = y_{i+1}$ möglich.

Es gilt stets $Rang(x) < Rang(y_i) \leq Rang(s_i) < Rang(t_i) \leq Rang(y_{i+1})$

Definition von N_k k minimal $\Rightarrow (x, y_i), (s_i, t_i) \notin N_{k-1}$

$$\Rightarrow \exists j \text{ mit } Rang(s_i) < A(k-1, j) \leq Rang(t_i)$$

Daher gilt $Rang(y_i) < A(k-1, j) \leq Rang(y_{i+1})$ für $1 \leq i \leq q-1$

Anwendung auf die gesamte Folge y_1, \dots, y_q (d.h. q-1 mal):

$$Rang(y_1) < A(k-1, j_1) \leq Rang(y_2) < A(k-1, j_2) \leq Rang(y_3) < \dots \leq Rang(y_{q-1}) < A(k-1, j_{q-1}) < Rang(y_q)$$

Beobachtung: $j_{i+1} \geq j_i \Rightarrow \exists j_1 \text{ mit } Rang(y_1) < A(k-1, j_1) \leq A(k-1, j_1 + q-1) \leq Rang(y_q)$

I. $\exists j \geq 2 : Rang(y_q) \geq A(k-1, j + q - 1)$

Beweis Teil c):

$k \geq 1, x \in G_{k,j}, (x, y_i) \in N_k \Rightarrow y_1, \dots, y_q \in G_{k,j}, j \geq 2$

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

II. $Rang(y_q) < A(k, j+1)$

Nach I und II:

$$\exists j : A(k-1, j + q - 1) \leq A(k, j+1) = A(k-1, A(k, j))$$

\Rightarrow (Monotonie von A in Zeilen) $j + q - 1 < A(k, j)$

$$\Rightarrow (j \geq 2)q < A(k, j)$$

Wir haben gezeigt: Für jeden Knoten $x \in G_{k,j}, k \geq 1, j \geq 2$ gibt es höchstens A(k,j) Kanten $(x, y) \in N_k \setminus L_k$

$$\Rightarrow |N_k \setminus L_k| \leq \sum_{j \geq 2} |G_{k,j}| * A(k, j) \text{ mit } 1 \leq k \leq z$$

Behauptung: $|G_{k,j}| \leq \frac{2n}{2^{A(k,j)}}$ extrem fallend, n Knoten im Baum.

Daraus folgt: $|N_k \setminus L_k| \leq \sum_{j \geq 2} \frac{2n * A(k, j)}{2^{A(k, j)}}$ wobei $A(k, j) \leq 2^j$, da $k \geq 1$ zweite Zeile.

$\leq 2n \sum_{j \geq 2} \frac{2^j}{2^{2^j}} = 2n \sum_{j \geq 2} \frac{1}{2^{2^j-j}} = 2n(\frac{1}{4} + \frac{1}{32} + \frac{1}{2^{12}} + \dots)$ wobei $(\frac{1}{32} + \dots)$ mit $\frac{1}{16}$ abgeschätzt wird.
 $\Rightarrow = \frac{5}{8}n$

Beweis der Behauptung $|G_{k,j}| \leq \frac{2n}{2^{A(k,j)}}$

Sei l beliebig mit $A(k,j) \leq l < A(k,j+1)$

Zähle zuerst alle Knoten mit $Rang(x) = l$. Dafür sei $G_{k,j,l} = \{x \in G_{k,j} \mid Rang(x) = l\}$

Es gilt:

1. Jeder Knoten x mit $Rang(x) = l$ hat mindestens 2^l Nachkommen (alle Knoten im Unterbaum mit Wurzel x), nach Lemma weighted Union

2. Für $x \neq y$ mit $Rang(x) = Rang(y)$ sind die Nachkommensmengen disjunkt. 1+2: $|G_{k,j,l}| \leq \frac{n}{2^l}$ mit n Gesamtanzahl der Knoten

$$\begin{aligned} \Rightarrow |G_{k,j}| &= \sum_{l=A(k,j)}^{A(k,j+1)-1} |G_{k,j,l}| \\ &\leq \sum_{l=A(k,j)}^{\inf} \frac{n}{2^l} = n \sum_{l=A(k,j)}^{\inf} \frac{1}{2^l} \\ &= n \left(\frac{1}{2^{A(k,j)}} + \frac{1}{2} \frac{1}{A(k,j)} + \frac{1}{4} \frac{1}{A(k,j)} + \dots \right) \\ &= n * \frac{2}{2^{A(k,j)}} \end{aligned}$$

Beweis Teil d:

$k = z + 1$, weighted Union $\Rightarrow Rang(y_q) \leq \log(n)$

Nach I. für $k = z + 1$:

$A(z, j + q - 1) \leq Rang(y_q) \leq \log(n)$

$\Rightarrow j + q - 1 < \alpha(z, n)$, da $\alpha(z, n)$ minimal mit $A(z, \alpha(z, n)) > \log(n)$

$\Rightarrow q < \alpha(z, n)$ mit $j \leq 2$

Also gibt es für jeden Knoten x höchstens $\alpha(z, n)$ Kanten $(x, \dots) \in N_{z+1} \setminus L_{z+1}$ mit $n = \text{Anzahl aller Knoten}$

$\Rightarrow |N_{z+1} \setminus L_{z+1}| \leq n * \alpha(z, n)$

Beweis des Satzes (Tarjan):

Jede beliebige Folge von $n-1$ Unions und $m \geq n$ Finds hat die Gesamtaufzeit von $\mathcal{O}(m * \alpha(m, n))$

Lemma (Kosten aller Finds): Für jedes $z \geq 0$ gilt:

$$\begin{aligned} |F| &= \sum_{k=0}^{z+1} |L_k| + \sum_{k=1}^{z+1} |N_k \setminus L_k| = \sum_{k=0}^{z+1} |L_k| + |N_0 \setminus L_0| + \sum_{k=1}^z |N_k \setminus L_k| + |N_{z+1} \setminus L_{z+1}| \\ &\leq (z+2) * m + n + \frac{5}{8}n * z * + n * \alpha(z, n) \end{aligned}$$

Betrachte $z = \alpha(m, n)$, $n \leq m$ (jedes z liefert eine obere Schranke)

Dann gilt:

$$|F| \leq \mathcal{O}(m * \alpha(m, n)) + n + \mathcal{O}(n * \alpha(m, n)) + n * \alpha(z, n)$$

$$|F| \leq \mathcal{O}(m * \alpha(m, n) + n * \alpha(z, n))$$

$$\Rightarrow \alpha(\alpha(m, n), n) \leq 4 \frac{m}{n}$$

$$\Rightarrow \leq n * \frac{4m}{n} = \mathcal{O}(m)$$

Insgesamt: Kosten aller Find-Operationen sind $|F| = \mathcal{O}(m\alpha(m, n) + m) = \mathcal{O}(m\alpha(m, n))$

Kosten aller Unions: $\mathcal{O}(n) = \mathcal{O}(m)$

Bemerkung:

1) In der Praxis sehr gute Laufzeiten (sehr einfache Algorithmen und Datenstrukturen, $\alpha(m, n) < 4$ für alle in der Praxis vorkommenden Werte von m, n).

2) Die Schranke $m\alpha(m, n)$ ist scharf, d.h. das Union-Find-Problem hat tatsächlich diese Komplexität. Es gibt Beweise für untere Schranke $\Omega(m * \alpha(m, n))$

3) Optimal...yeay.

4) Varianten: Split-Find (für Intervalle),

Union-Split-Find: Split(x) markiere x , Union(x) entfernt Markierung, Find(x) nächste Markierung rechts von x

1.2 Wörterbücher

Wir kennen balancierte Suchbäume.

Problem: Teilmenge $S \subseteq U$ mit U Universum, eventuell linear geordnet.

Speichere S (Schlüssel) in einer Datenstruktur D mit:

D.insert(x) $x \in U$, $S \leftarrow S \cup \{x\}$

D.delete(x) $x \in U$, $S \leftarrow S \setminus \{x\}$

D.lookup(x) $x \in U$, testet ob $x \in S$

Es soll zusätzlich zum Schlüssel Zusatzinformation gespeichert werden.

Wir behandeln 2 Datenstrukturen: Randomisierte Suchbäume, Perfektes Hashing

1.2.1 Randomisierter Suchbaum

Entwickelt von Seidel/Aragon

Idee: Verwende einen Zufallsprozess zur Balancierung von binären Suchbäumen.

Vorteile: Sehr einfache Implementierung, geringer Aufwand zur Verwaltung der Balance, effizient

Definition RST(Randomized Search Tree):

Sei $S = \{x_1, \dots, x_n\}$ eine Menge von n Schlüsseln aus einem linear geordneten Universum U .

Jedem $x_i \in S$ wird zusätzlich eine Zufallszahl $prio(x_i)$, seine Priorität zugeordnet. Diese Zufallszahlen sind gleichverteilte reelle Zahlen aus $[0,1]$ oder praktisch ganze Zahlen aus $[0, \dots, 2^{32} - 1]$.

Ein RST für S ist:

- 1) Ein Knotenorientierter binärer Suchbaum für die Paare $(x_i, prio(x_i))$
- 2) Gleichzeitig ein Maximum-Heap bezüglich der Prioritäten $prio(x_i)$

