Ferienakademie 1999
Kurs 2: Bäume - Algorithmik und Kombinatorik

# LOWEST COMMON ANCESTORS

Pham Minh Bolo

**Zusammenfassung**

Die Arbeit von **Baruch Schieber** und **Uzi Vishkin** zum Thema *ON FINDING LOWEST COMMON ANCESTORS: SIMPLIFICATION AND PARALLELIZATION* behandelt folgendes Problem: Gegeben sei ein Baum $T(V, E)$, mit dem man eine Vorverarbeitung durchführen kann. Bestimme den kleinsten gemeinsamen Vorfahren **(Lowest Common Ancestor)** von zwei Knoten $x, y$ in $T$. Die Autoren stellen einen Algorithmus vor, der nach linearer Vorverarbeitung solche LCA-Anfragen in konstanter Zeit beantwortet. Eine Parallelisierung der Vorverarbeitung unter Verwendung einer optimalen Zahl von Prozessoren auf einer EREW PRAM liefert sogar logarithmische Laufzeit für die Vorarbeit.

## 1 Einführung

Betrachte folgendes Problem: Es sei ein Baum $T(V, E)$ gegeben und wir wollen wissen, welcher Knoten der kleinste gemeinsame Vorfahre (Lowest Common Ancestor) von zwei beliebigen Knoten $x, y$ in $T$ ist. In einem Vorbereitungsschritt sammeln wir Informationen zu den Knoten des Baumes $T$ und verwenden sie anschließend zur Beantwortung solcher LCA-Anfragen. In einer Vorarbeit zeichnen wir alle Knoten $v$ in $T$ mit drei Attributen aus: INLABEL($v$), ASCENDANT($v$) und LEVEL($v$). Wenn wir zu jedem Knoten in $T$ die zugehörige INLABEL Nummer bestimmt haben, können wir feststellen, dass einige Knoten gleiche INLABEL Nummern besitzen, die nicht willkürlich im Baum $T$ verteilt sind, sondern zu einem Pfad zusammengefasst werden können. Es gibt natürlich auch Knoten, die für sich selber einen Pfad bilden, weil sie eine INLABEL Nummer besitzen, die ungleich allen anderen INLABEL Nummern in $T$ ist. Jeder Pfad in $T$ ist somit durch die

INLABEL Nummer der den Pfad bildenden Knoten eindeutig. Wenn wir zu zwei Knoten $x, y$ den kleinsten gemeinsamen Vorfahre $z$ (mit $z = LCA(x, y)$) finden wollen, müssen wir zunächst den Pfad, in dem sich $z$ befindet, bestimmen. Dazu führen wir eine Transformation der Knoten in $T$ in Knoten in einem vollständigen Binärbaum $B$ durch. Die Knoten in $B$ sind durch ihre Inorder-Nummer eindeutig. Alle Knoten in einem Pfad in $T$ werden zu einem Knoten zusammengefasst und nach einer bestimmten Abbildungsvorschrift auf einen Knoten in $B$ abgebildet. Der Knoten $x$ in $T$ wird auf den Knoten INLABEL($x$) abgebildet und der Knoten $z$ in $T$ auf den Knoten INLABEL($z$) in $B$. Die Lage der abgebildeten Knoten richtet sich nach dem Wert der INLABEL Nummer des Urbildes. Das heißt zum Beispiel, dass das Bild der Knoten im Pfad mit der INLABEL Nummer eins in $T$ der Knoten mit der Inorder-Nummer eins in $B$ ist. Wir können den Pfad, in dem sich $z$ befindet, ermitteln, indem wir die Inorder-Nummer des Knotens INLABEL($z$) in $B$ ausgehend von der Inorder-Nummer der gegebenen Knoten INLABEL($x$) und INLABEL($y$) bestimmen.
Im Verlauf des Algorithmus benötigen wir zwei Eigenschaften der INLABEL Nummern. (1) Die Path-Partition Eigenschaft besagt, dass Knoten mit gleicher INLABEL Nummer in $T$ nicht willkürlich verteilt sind, sondern so angeordnet sind,dass sie Pfade bilden. (2) Die Descendance-Preservation Eigenschaft besagt folgendes: Wenn der Knoten $x$ Nachkomme vom Knoten $z$ in $T$ ist, dann ist auch der Knoten INLABEL($x$) Nachkomme vom Knoten INLABEL($z$) in $B$.
Wir können den Vorverarbeitungsschritt in linearer Zeit und mit linearem Platzbedarf implementieren. Anschließend können wir jede LCA-Anfrage in $O(1)$ Zeit beantworten. Die Vorarbeit hat den Vorteil, dass man sie auf einfache Weise parallelisieren kann. Dazu betrachtet man das Modell einer EREW PRAM Maschine (**e**xclusive-**r**ead **e**xclusive-**w**rite **p**arallel **r**andom **a**ccess **m**achine). Eine PRAM verfügt über $p$ synchrone Prozessoren, die alle Zugriff zu einem gemeinsamen Speicher besitzen. Eine EREW PRAM dagegen erlaubt keinen gleichzeitigen Lese- order Schreibzugriff von mehr als einem Prozessor auf die gleiche Speicherzelle. Durch Parallelisierung können wir die Laufzeit der Vorverarbeitung auf $O(\log n)$ Zeit verbessern, indem wir $n/\log n$ Prozessoren auf einer EREM PRAM verwenden. Die Grundidee bei der Parallelisierung liegt darin, den Baum $T$ durch das Einfügen von aufwärts und abwärts gerichteten Kanten zu einem neuen Graphen $H$ umzugestalten, der einen Eulerschen Pfad mit der Wurzel $r$ als Start- und Endknoten enthält. Entlang des Eulerschen Pfades weist man den Kanten entsprechend ihrer Richtung Gewichte zu und berechnet durch einen parallelen, in logarithmischer Zeit laufenden list-ranking Algorithmus den Abstand der Kanten zur Wurzel $r$. Die Attribute INLABEL, ASCENDANT und LEVEL, die man allen Knoten in $T$ in der Vorarbeit zuweist, lassen sich dann durch die vorher berechneten Abstände der Kanten ausdrücken.

# 2 Vorverarbeitung

Im Vorverarbeitungsabschnitt stellen wir eine look-up Tabelle **HEAD** auf und bestimmen zu jedem Knoten $v$ in $T$ folgende drei Attribute : **INLABEL**($v$)**, ASCENDANT**($v$) **und LEVEL**($v$).

Zu jedem Knoten $v$ in $T$ bestimmen wir **INLABEL**($v$) in zwei Schritten:
*Schritt* 1. Bestimme PREORDER($v$) und SIZE($v$). Die PREORDER-Numerierung numeriert die Knoten in $T$ in der Reihenfolge, in der sie durch die Tiefensuche zum ersten Mal berührt werden. SIZE($v$) gibt die Anzahl der Knoten im Unterbaum mit der Wurzel $v$ an. Die PREORDER Nummern der Knoten im Unterbaum mit Wurzel $v$ liegen dabei im geschlossenen Intervall [PREORDER($v$),PREORDER($v$)+SIZE($v$)−1], dem *Intervall von* $v$.
*Schritt* 2. Wandle alle Dezimalzahlen im Intervall von $v$ in Binärzahlen um und ermittle diejenige Binärzahl, die von rechts gezählt die meisten Bits "0" bis zum ersten Bit "1" aufweist. Diese Zahl ist der Wert von INLABEL($v$).
Abbildung 1 zeigt als Beispiel einen Baum, dessen Knoten mit der PREORDER-Nummer und den anfangs des Abschnitts erwähnten Attributen ausgezeichnet sind.

Die INLABEL Nummern besitzen folgende zwei Eigenschaften, die wir bereits in der Einführung erwähnt haben.
LEMMA 1. *Die* INLABEL *Nummern besitzen die Path-Partition Eigenschaft.*
**Beweis.** Wir stellen fest, dass die Intervalle der Söhne von $v$ paarweise disjunkt sind. Somit liegt INLABEL($v$) in höchstens einem dieser Intervalle. Falls es einen Sohn $u$ gibt, so daß INLABEL($v$) in dessen Intervall liegt, gilt: INLABEL($v$) = INLABEL($u$) (falls $u$ existiert). Für einen anderen Sohn $w$ gilt : INLABEL($w$) $\neq$ INLABEL($v$). Wir haben damit gezeigt, dass höchstens ein Sohn die gleiche INLABEL Nummer wie der Vater besitzt. Dies impliziert die Path-Partition Eigenschaft der INLABEL Nummern. □
LEMMA 2. *Die* INLABEL *Nummern besitzen die Descendance-Preservation Eigenschaft.*
**Beweis.** Sei $d$ ein Nachkomme von $v$ in $T$. Wir zeigen, dass INLABEL($d$) ein Nachkomme von INLABEL($v$) im vollständigen Binärbaum $B$ ist. Betrachte dazu zwei beliebige Knoten $b$ und $c$ in $B$. Zur Darstellung der Inorder-Nummern der Knoten eines vollständigen Binärbaums $B$,der insgesamt $n$ Knoten besitzt, als Binärzahlen benötigen wir $l+1$ Bits, wobei $l = \lfloor \log n \rfloor$. Sei $i$ die Anzahl der rechtesten Bits "0" in $b$. Die Inorder-Nummer von $b$ besteht somit aus den $l-i$ linkesten Bits gefolgt von einer "1" und $i$ Nullen. Wir geben im folgenden zwei Kriterien an, wann ein Knoten ein Nachkomme eines anderen ist.
BEHAUPTUNG. *Ein Knoten $c$ ist Nachkomme eines Knoten $b$ genau dann, wenn (1) die $l-i$ linkesten Bits in $c$ gleich den $l-i$ linkesten Bits in $b$ sind und (2) die Anzahl der Folge von Bits "0" in $c$ (von rechts gezählt) höchstens $i$ ist.*
**Beweis.** $b_L$ und $b_R$ seien linker und rechter Sohn von $b$. Es gilt:
**(i)** $b_L$ ist zusammengesetzt aus den $l-i$ linken Bits von $b$, gefolgt von einer "0", einer "1" und $i-1$ Bits "0".
**(ii)** $b_R$ ist zusammengesetzt aus den $l-i$ linken Bits von $b$, gefolgt von zwei Bits "1" und $i-1$ Bits "0".
Diese Fakten implizieren beide Richtungen der Behauptung. □
Nun können wir den Beweis von Lemma 2 fortsetzen. Sei $i$ die Anzahl der rechtesten Bits "0" in INLABEL($v$). Da INLABEL($d$) im Intervall von $v$ liegt ($d$ ist bekanntlich Nachkomme von $v$) und INLABEL($v$) in diesem Intervall die meisten rechtesten Bits "0" besitzt ($v$ ist ja Wurzel des Unterbaums), folgt, dass die Anzahl der rechtesten Bits "0"

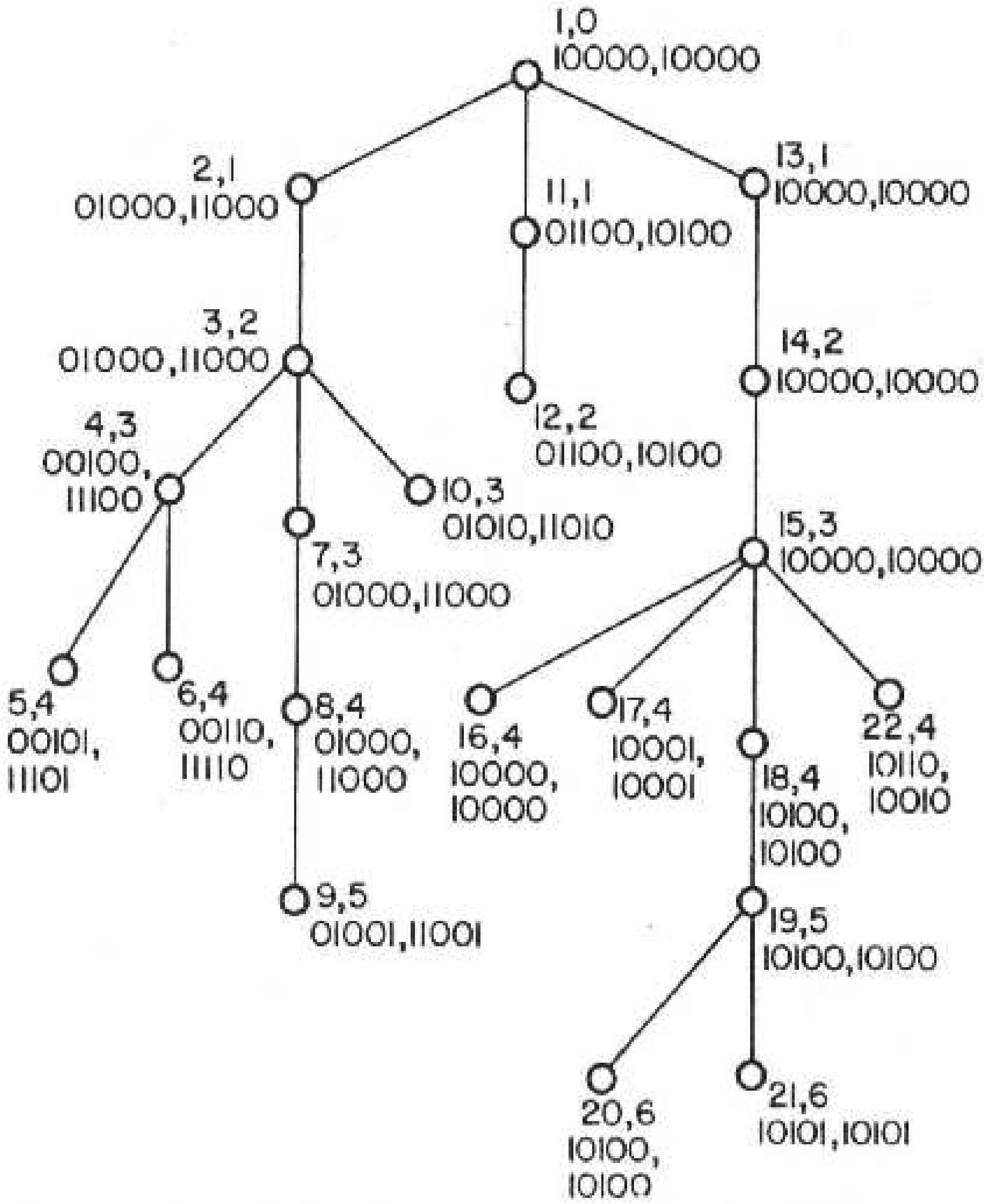


Abbildung 1: Beispiel. Vier Attribute zu jedem Knoten des Baumes: PREORDER,LEVEL,INLABEL und ASCENDANT.

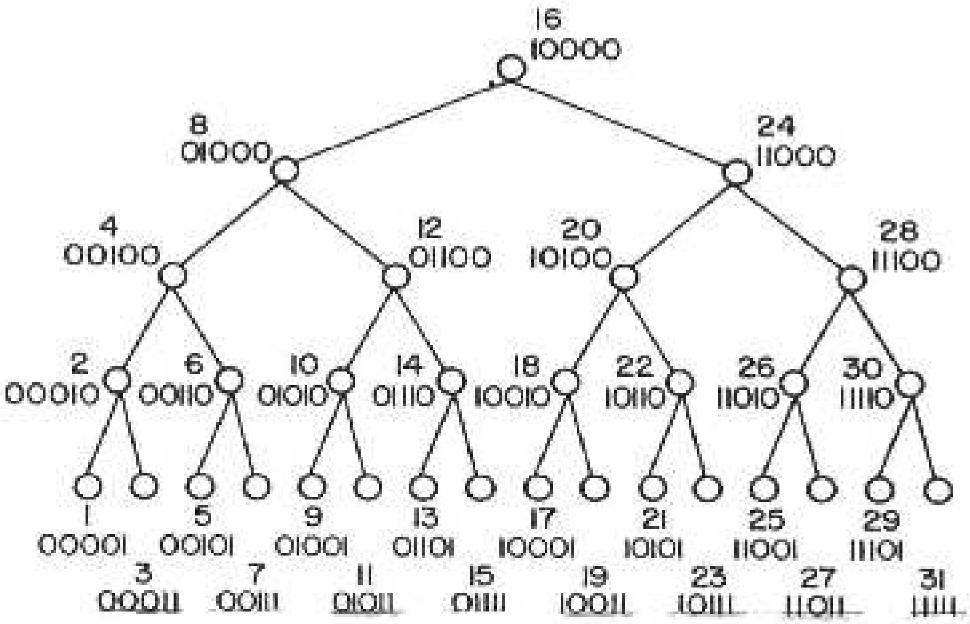


Abbildung 2: Beispiel. Inorder-Numerierung des vollständigen Binärbaums mit 31 Knoten

in INLABEL($d$) höchstens $i$ groß ist. Die $l - i$ linkesten Bits in INLABEL($d$) stimmen mit den $l - i$ linkesten Bits in INLABEL($v$) überein (Eigenschaft der INLABEL Nummern von Knoten im gleichen Unterbaum). Damit erfüllt INLABEL($d$) die obigen Kriterien der Nachkommenschaft und Lemma 2 ist somit bewiesen. □

Der Leser kann anhand von Abbildung 2 die Gültigkeit der Kriterien für die Eigenschaft eines Knotens als Nachkomme eines anderen Knoten verifizieren.

**Implementierung.** Schritt 1. können wir in linearer Zeit und mit linearem Platzbedarf implementieren. Falls zu jedem Knoten $v$ in $T$ PREORDER($v$) und SIZE($v$) gegeben sind, können wir Schritt 2. in konstanter Zeit pro Knoten in zwei Teilschritten durchführen.

Bei näherer Betrachtung der Zahlen im Intervall von $v$ können wir INLABEL($v$) finden, wenn wir den Index des rechtesten Bit "1" bestimmen. Rechts von diesem markanten Bit füllen wir die INLABEL Nummer mit Bit "0" auf. Den Index dieses rechtesten Bit "1" finden wir, indem wir die einzelnen Bits von PREORDER($v$) und PREORDER($v$)+SIZE($v$)$-$ 1 , d.h. von der ersten und letzten Zahl im Intervall, jeweils miteinander vergleichen und dabei eine bitweise logische Exklusiv-ODER-Operation durchführen. Diese logische Operation gibt uns Auskunft, an welchen Stellen sich die Bits unterscheiden. Die linkeste Stelle, an der die Bits verschieden sind, ist der gesuchte Index. Wir bestimmen $i$, den gesuchten Index des rechtesten Bits "1", wie folgt:

*Schritt* 2.1. Berechne $i = \lfloor \log[(\mathrm{PREORDER}(v)-1)\ \mathbf{xor}\ (\mathrm{PREORDER}(v)+\mathrm{SIZE}(v)-1)] \rfloor$.

Wir benutzen in der obigen Operation nicht PREORDER($v$) sondern PREORDER($v$) $-1$, damit die Rechung auch für Knoten $v$ mit INLABEL($v$) = PREORDER($v$) funktioniert. Da wir wissen, daß bei allen INLABEL Nummern im Intervall von $v$ die $l-i$ linkesten Bits übereinstimmen, können wir sofort INLABEL($v$) hinschreiben:
*Schritt* 2.2. INLABEL($v$) setzt sich zusammen aus den $l-i+1$ linkesten Bits von PREORDER($v$)+ SIZE($v$) $-1$ (zu den $l-i$ linkesten Bits kommt noch das markante rechteste Bit "1" hinzu) gefolgt von $i$ Bits "0".
Wir kommen nun zur Bestimmung der **ASCENDANT** Nummern. Genau wie die INLABEL Nummern sind die ASCENDANT Nummern Binärzahlen der Länge $l+1$. Die ASCENDANT Nummer eines Knoten $v$ dient dazu, Informationen über die INLABEL Nummern aller Vorfahren von $v$ zu speichern. Bei Betrachtung des Beispielbaumes in Abbildung 1. können wir schnell feststellen, dass die INLABEL Nummer der Vorfahren von $v$ durch den Index deren markanten rechtesten "1" eindeutig sind. Das heißt, wenn in ASCENDANT($v$) an der Bitstelle zwei eine "1" eingetragen ist, können wir sofort sagen, von welcher INLABEL Nummer der Vorfahren von $v$ diese "1" stammt, da unter allen INLABEL Nummern der Vorfahren genau eine existiert, die eine rechteste "1" an der Bitstelle zwei besitzt. Sei $A_l(v), \cdots, A_0(v)$ die zugehörige Binärsequenz von ASCENDANT($v$). Wir setzen $A_i(v) = 1$, falls $i$ der Index des am weitesten rechts stehenden Bits "1" in der INLABEL Nummer eines Vorfahrens von $v$ in $T$ ist. Bei der Durchnumerierung der Knoten in $T$ mit der ASCENDANT Nummer gehen wir wie folgt vor: Sei $r$ die Wurzel von $T$. Setze ASCENDANT($r$) $= 2^l$. Betrachte nun einen inneren Knoten $v$ in $T$. $F(v)$ sei der Vater von $v$ in $T$. Wir setzen ASCENDANT($v$) = ASCENDANT($F(v)$), falls INLABEL($v$) =INLABEL($F(v)$) und ASCENDANT($v$) = ASCENDANT($F(v)$) $+ 2^i$, sonst, wobei $i$ der Index der rechtesten "1" in INLABEL($v$) ist.
Zu jedem Knoten $v$ in $T$ berechnet LEVEL($v$) die Anzahl der Kanten im Pfad von $v$ zur Wurzel $r$.
Die Tabelle **HEAD** bestimmen wir wie folgt: HEAD($k$) beinhaltet denjenigen Knoten im Pfad mit der INLABEL Nummer $k$, der der Wurzel am nächsten ist. Wir gehen alle Knoten durch und setzen HEAD(INLABEL($v$)) auf einen Knoten $v$, falls gilt:
INLABEL($v$) $\neq$ INLABEL($F(v)$). In diesem Fall liegt der Vaterknoten in einem anderen Pfad und somit ist $v$ der höchste Knoten im INLABEL($v$)-Pfad.

# 3 LCA-Anfragen

Wir zeigen nun, wie man LCA-Anfragen unter Verwendung der Ergebnisse des Vorverarbeitungsabschnittes beantwortet.
Betrachte zu zwei beliebigen Knoten $x, y$ in $T$ eine Anfrage LCA($x, y$). Wir unterscheiden zwei Fälle:
*Fall* A. INLABEL($x$) = INLABEL($y$). In diesem Fall liegen $x$ und $y$ im gleichen Pfad und LCA($x, y$) ist somit $x$, falls LEVEL($x$) $\leq$ LEVEL($y$), und $y$ sonst.
*Fall* B. INLABEL($x$) $\neq$ INLABLE($y$). Sei $z$ LCA($x, y$). Wir finden $z$ in vier Schritten:

*Schritt* 1. Finde $b$, LCA von INLABEL$(x)$ und INLABEL$(y)$, im vollständigen Binärbaum $B$. Sei $i$ der Index der rechtesten "1" in $b$. Da $b$ ein gemeinsamer Vorfahre von INLABEL$(x)$ und INLABEL$(y)$ in $B$ ist, muss $i$ daher folgende zwei Bedingungen erfüllen. (1) Die $l-i$ linkesten Bits in INLABEL$(x)$ und in INLABEL$(y)$ stimmen mit den entsprechenden Bits in $b$ überein. (2) Der Index der rechtesten "1" in INLABEL$(x)$ und in INLABEL$(y)$ ist höchstens $i$. Für $i$ müssen wir den kleinstmöglichen Index wählen, der beide Bedingungen erfüllt. Wir gehen dabei wie folgt vor, um $i$ zu bestimmen.

Bestimme den Index der rechtesten "1" in INLABEL$(x)$, den Index der rechtesten "1" in INLABEL$(y)$ und den Index der linkesten Bitstelle, an der INLABEL$(x)$ und INLABEL$(y)$ sich unterscheiden. Für $i$ wählen wir von den drei Indizes den größten davon. Die gewählte Bitstelle, die von allen drei Indizes die linkeste Stelle ist, gibt uns die Garantie, daß $i$ beide Bedingungen erfüllt.

In Schritt 2 finden wir INLABEL$(z)$ ($z$ ist LCA$(x,y)$). Die Descendance-Preservation Eigenschaft besagt, dass INLABEL$(z)$ ein gemeinsamer Vorfahre von INLABEL$(x)$ und INLABEL$(y)$ ist. INLABEL$(z)$ ist nicht notwendigerweise der kleinste gemeinsame Vorfahre von INLABEL$(x)$ und INLABEL$(y)$ in $B$, aber man kann sich leicht überlegen, dass INLABEL$(z)$ von allen Vorfahren von $b$ in $B$, auf die Vorfahren von $x$ und $y$ in $T$ abgebildet werden, der kleinste ist.

*Schritt* 2. Finde INLABEL$(z)$. Sei $j$ der Index der rechtesten "1" in INLABEL$(z)$. Da $z$ ein gemeinsamer Vorfahre von $x$ und $y$ in $T$ ist, gilt: $A_j(x) = 1$ ($A_j(x)$ ist der $j$-te Eintrag in ASCENDANT$(x)$, das die Indizes der rechtesten "1" in den INLABEL Nummern der Vorfahren von $x$ speichert.) und $A_j(y) = 1$. Da INLABEL$(z)$ von allen Vorfahren von $b$, auf die Vorfahren von $x$ und $y$ in $T$ abgebildet werden, der kleinste ist, und $b$ ein gemeinsamer Vorfahre von $x$ und $y$ ist, muss der Index $j$ gleich dem Index der rechtesten "1" in $A_l(x), \cdots, A_i(x)$ und in $A_l(y), \cdots, A_i(y)$ sein. INLABEL$(z)$ besteht somit aus den $l-j$ linkesten Bits von INLABEL$(x)$ (oder INLABEL$(y)$), gefolgt von einer "1" und $j$ Bits "0". Anschließend finden wir $z$. Dazu bestimmen wir $\hat{x}$, den kleinsten Vorfahren von $x$ im durch INLABEL$(z)$ definierten Pfad und $\hat{y}$, den kleinsten Vorfahren von $y$ im selben Pfad. $z$ ist dann von beiden Knoten derjenige, der der Wurzel näher ist.

*Schritt* 3. Finde $\hat{x}$ und $\hat{y}$. Wir zeigen im folgenden, wie man $\hat{x}$ ermittelt. Analog dazu findet man $\hat{y}$. Falls INLABEL$(x)$ $=$ INLABEL $(z)$ gilt, liegen $x$ und $z$ im gleichen Pfad und somit ist $x = \hat{x}$. Betrachte den interessanteren Fall INLABEL$(x)$ $\neq$ INLABEL$(z)$. Zur Bestimmung von $\hat{x}$ müssen wir zunächst seinen Sohn $w$ finden, der ebenfalls ein Vorfahre von $x$ ist. Sei $k$ der Index der rechtesten "1" in INLABEL$(w)$. Man kann sich überlegen, dass $k$ der Index der linkesten "1" in $A_{j-1}(x), \cdots, A_0(x)$ ist. INLABEL$(w)$ setzt sich somit zusammen aus den $l-k$ linkesten Bits in INLABEL$(x)$, gefolgt von einer "1" und $k$ Bits "0". Da $w$ in einem anderen Pfad als dessen Vater $\hat{x}$ liegt, folgt, dass $w$ HEAD(INLABEL$(w)$) ist. $w$ lesen wir somit aus der Tabelle HEAD ab. Damit finden wir $\hat{x}$, den Vater von $w$.

*Schritt* 4. LCA$(x,y)$ ist $\hat{x}$, falls LEVEL$(\hat{x})$ $\leq$ LEVEL$(\hat{x})$, und $\hat{y}$ sonst.

Im folgenden geben wir noch Informationen zur Implementierung der obigen Schritte der LCA-Anfragen an:

*Schritt* 1. Bestimme $i$, den Index der rechtesten "1" in $b$ wie folgt:

*Schritt* 1.1. Bestimme $i_1$, den Index der rechtesten "1" in INLABEL$(x)$, und $i_2$, den Index

der rechtesten "1" in INLABEL($y$). $i_1$ gewinnt man wie folgt:
$i_1 :=$ log (INLABEL($x$) – [INLABEL($x$) **and** (INLABEL($x$) – 1)]). $i_2$ erhält man analog.
*Schritt* 1.2. Finde $i_3$, den Index des linkesten Bit, an dem sich INLABEL($x$) und INLABEL($y$) unterscheiden, wie folgt:
$i_3 :=$ $\lfloor$log[ INLABEL($x$) **xor** INLABEL($y$)]$\rfloor$. Diese Operation entspricht dem Schritt 2.1 bei der Bestimmung der INLABEL Nummern im vorherigen Abschnitt.
$i$ ist das Maximum von $i_1, i_2, i_3$. Anhand der Kriterien der Descendance-Preservation Eigenschaft können wir nun die Bitsequenz von $b$ problemlos hinschreiben.
*Schritt* 2. Den Index $j$ gewinnen wir wie folgt:
*Schritt* 2.1. COMMON sei eine $l+1$ lange Binärzahl, die alle Bits "1", die sowohl in ASCENDANT($x$) als auch in ASCENDANT($y$) auftreten, enthält. Wir führen somit folgende bitweise logische Operation durch:
COMMON := ASCENDANT($x$) **and** ASCENDANT($y$).
*Schritt* 2.2. Sei COMMON$_i$ := $2^i\lfloor$COMMON$/2^i\rfloor$. COMMON$_i$ beinhaltet alle Bits "1" sowohl in $A_l(x), \cdots, A_i(x)$ als auch in $A_l(y), \cdots, A_i(y)$. Die Bitstellen rechts von $i$ sind alle mit Bits "0" aufgefüllt. $j$ ist der Index der rechtesten "1" in COMMON$_i$. $j$ bestimmen wir wie folgt:
$j :=$ log (COMMON$_i$ -[ COMMON$_i$ **and** (COMMON$_i$ -1)]).
Die Implementierung von Schritt 3. verwendet die gleichen Techniken.

# 4 Parallelisierung der Vorverarbeitung

Wir werden in diesem Abschnitt zeigen, dass man durch Parallelisierung die Vorverarbeitung in $O(\log n)$ Zeit unter Verwendung von $n/\log n$ Prozessoren durchführen kann.
Wir machen die folgende Annahme, dass alle Kanten in $T$ aufwärts gerichtet sind. Zur Bestimmung der Attribute INLABEL($v$), ASCENDANT($v$) und LEVEL($v$) gehen wir wie folgt vor:
*Schritt* 1. Zu der Kante $(v \to u)$ in $T$ fügen wir eine Kante in Gegenrichtung $(u \to v)$ hinzu. $H$ sei der neue Graph.
Jeder Knoten in $H$ hat gleichen IN- und OUT-Grad und $H$ hat einen Eulerschen Pfad mit Start- und Endpunkt $r$. Anmerkung: Ein Pfad (Kreis) in Graphen (Digraphen) heißt *Eulerscher Pfad*, wenn er jede Kante genau einmal enthält.
*Schritt* 2. Gehe zu jedem Knoten wie folgt vor: $(v \to u_0), \cdots, (v \to u_{d-1})$ seien die ausgehenden Kanten von $v$. Setze $D(u_i \to v) := (v \to u_{(i+1) mod\ d})$, für $i = 0, ...d-1$. Um den damit entstandenen Eulerkreis in $D$ aufzubrechen, führen wir eine Korrektur ein, die einen Eulerpfad mit Start- und Endpunkt bei $r$ liefert: $D(u_i \to r) :=$*end-of-list*.
Wir zeigen im folgenden, wie man den Eulerpfad zur Bestimmung von PREORDER($v$), PREORDER($v$)+SIZE($v$) $-$ 1 und LEVEL($v$) zu dem Knoten $v$ in $T$ verwenden kann.
*Schritt* 3. Wir weisen jeder Kante $e$ im Eulerschen Pfad zwei Gewichte $W_1(e)$ und $W_2(e)$ zu. (1) $W_1(e) = 1$, falls $e$ abwärts zeigt und $W_1(e) = 0$ sonst. (2) $W_2(e) = 1$, falls $e$ abwärts zeigt und $W_2(e) = -1$ sonst.

*Schritt* 4. Wir wenden zweimal einen optimalen parallelen list-ranking Algorithmus mit logarithmischer Laufzeit an, um zu jeder Kante $e$ in $H$ ihren gewichteten Abstand vom Startpunkt des Eulerpfades zu ermitteln. Die Ergebnisse werden in $\text{DISTANCE}_1(e)$ (erste Anwendung) und $\text{DISTANCE}_2(e)$ (zweite Anwendung) gespeichert. Betrachte einen Knoten $v \neq r$ und sei $u$ dessen Vater in $T$. PREORDER($v$) ist dann $\text{DISTANCE}_1(u \rightarrow v) + 1$, PREORDER($v$)+SIZE($v$) $- 1$ ist $\text{DISTANCE}_1(v \rightarrow u) + 1$ und LEVEL($v$) ist $\text{DISTANCE}_2(u \rightarrow v)$. (Der Leser möge diese Behauptungen eigenständig nachprüfen)
*Schritt* 5. Falls zu jedem Knoten $v$ in $T$ PREORDER($v$) und PREORDER($v$)+SIZE($v$)$-1$ gegeben sind, bestimmen wir INLABEL mittels des sequentiellen Algorithmus in konstanter Zeit unter Gebrauch von $n$ Prozessoren.
Im folgenden zeigen wir, wie man ausgehend vom Eulerschen Pfad zu jedem Knoten $v$ in $T$ ASCENDANT($v$) findet.
*Schritt* 6. Wir weisen jeder Kante $e$ im Eulerschen Pfad ein neues Gewicht $W(e)$ zu. Gehe zu jedem Knoten $v \neq r$ wie folgt vor: Sei $u$ der Vater von $v$ in $T$ und sei $i$ der Index der rechtesten "1" in INLABEL($v$). Falls INLABEL($v$) $\neq$ INLABEL($u$), setzen wir $W(u \rightarrow v) = 2^i$ und $W(v \rightarrow u) = -2^i$. Das Gewicht der anderen Kanten setzen wir auf Null.
*Schritt* 7.Wir wenden wiederum einen parallelen list-ranking Algorithmus zur Bestimmung des gewichteten Abstands jeder einzelnen Kante $e$ in $H$ vom Startpunkt des Eulerschen Pfades an. Betrachte hierzu einen Knoten $v \neq r$ und sei $u$ dessen Vater in $T$. ASCENDANT($v$) ist der Abstand der Kante $(u \rightarrow v)$ plus $2^l$. Für die Wurzel $r$ gilt: ASCENDANT($r$) $= 2^l$.
Wenn wir die Attribute zu den Knoten $v$ in $T$ berechnet haben, können wir die Tabelle HEAD in konstanter Zeit mittels $n$ Prozessoren aufstellen.
*Komplexität.* Die Schritte 4 und 7 benötigen jeweils $n/\log n$ Prozessoren und $O(\log n)$ Laufzeit. Die Schritte 1,2,3,5,6 und das Aufstellen der Tabelle HEAD benötigen jeweils $n$ Prozessoren und $O(1)$ Zeit. Diese Schritte können aber auch von $n/\log n$ Prozessoren in $O(\log n)$ Zeit bearbeitet werden. Das heißt, mittels Parallelisierung können wir die Vorverabeitung in $O(\log n)$ Zeit mit $n/\log n$ Prozessoren durchführen.

## LITERATUR

B. SCHIEBER und U. VISHKIN, *On Finding Lowest Common Ancestors: Simplification and Parallelization*, SIAM J. Comput., Vol. 17, No. 6, December 1988.