

LOUDS++

Finn Stutzenstein, Levin Nemesch, Joshua Sangmeister

December 4, 2020

Algorithm Engineering - Übung 2

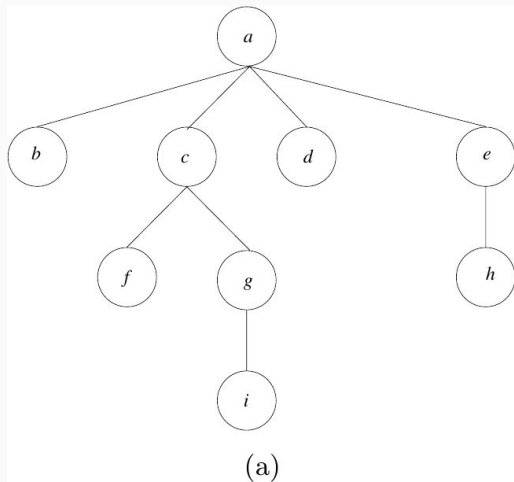
- LOUDS: Level-Order Unary Degree Sequence
- *succinct data structure*: $2n + o(n)$ Bits Speicherbedarf
- Geordneter Baum
- Unterstützt folgende Operationen in $\mathcal{O}(1)$:
 - $\text{parent}(x)$
 - $\text{first-child}(x)/\text{last-child}(x)$
 - $\text{prev-sibling}(x)/\text{next-sibling}(x)$
 - $\text{degree}(x)$: Anzahl der Kinder von x
 - $\text{childrank}(x)$: Rang von x unter seinen Geschwistern
 - $\text{child}(x, i)$: i -tes Kind von x
- Quelle: "Engineering the LOUDS Succinct Tree Representation" von O'Neil Delpratt, Naila Rahman, and Rajeev Raman (<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.4250&rep=rep1&type=pdf>)

- Bitfeld von n Bits
- $\text{rank}_b(x)$ ($b \in \{0, 1\}$): Anzahl der Vorkommnisse von b bis x
- $\text{select}_b(x)$ ($b \in \{0, 1\}$): Position des x -ten Vorkommens von b
- Z.B. von Kim Et Al.: Speicher $n + o(n)$ Bits und Laufzeiten in $\mathcal{O}(1)$

- Bitfeld von n Bits
- $\text{rank}_b(x)$ ($b \in \{0, 1\}$): Anzahl der Vorkommnisse von b bis x
- $\text{select}_b(x)$ ($b \in \{0, 1\}$): Position des x -ten Vorkommens von b
- Z.B. von Kim Et Al.: Speicher $n + o(n)$ Bits und Laufzeiten in $\mathcal{O}(1)$
- Beispiel: 0011011100
 - $\text{rank}_1(7) = 4$
 - $\text{rank}_0(7) = 3$
 - $\text{select}_1(3) = 6$
 - $\text{select}_0(5) = 10$

LOUDS bit string (LBS)

- Speichert unär Baumstruktur
- Jeder Knoten repräsentiert durch 1^d0 Bitstring, d entspricht Anzahl Kinder
- Nummerierung der Knoten ebenenweise (entspricht Breitensuche)
- Beginn markiert durch 10



1 2 | 3 4 5 6 7 | 8 | 9 10 11 | 12 | 13 14 | 15 | 16 17 | 18 | 19 |
1 0 | 1 1 1 1 0 | 0 | 1 1 0 | 0 | 1 0 | 0 | 1 0 | 0 | 0
 (b)

Vertex	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>
BFS	1	2	3	4	5	6	7	8	9
1-based	1	3	4	5	6	9	10	13	16
0-based	7	8	11	12	14	15	17	18	19

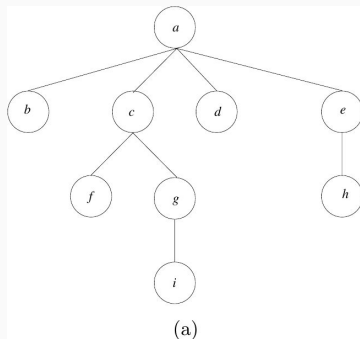
(c)

Vertex	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>
<i>R0</i>	1	0	1	0	1	0	1	0	1
<i>R1</i>	1	0	0	0	1	0	1	1	1

(d)

- Besteht aus $n + 1$ 0-en und n 1-en
- i -ter Knoten:
- *Ones-based numbering*
 - i -te 1 in Kodierung des Elternknotens
 - Knoten i wird Zahl $x = \text{select}_1(i) \in \{1, \dots, 2n + 1\}$ zugewiesen
 - Knoten mit Zahl x : $i = \text{rank}_1(x) \in \{1, \dots, n\}$
- *Zero-based numbering*: $i + 1$ -te 0: Ende der Kodierung des i -ten Knotens
 - $i + 1$ -te 0: Ende der Kodierung des i -ten Knotens
 - Mit select_0 und rank_0 kann analog zum Ones-based numbering eine Zahl zugeordnet werden

Beispiel: Operation $\text{parent}(x)$



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0	0	0

(b)

Vertex	a	b	c	d	e	f	g	h	i
BFS	1	2	3	4	5	6	7	8	9
1-based	1	3	4	5	6	9	10	13	16
0-based	7	8	11	12	14	15	17	18	19

(c)

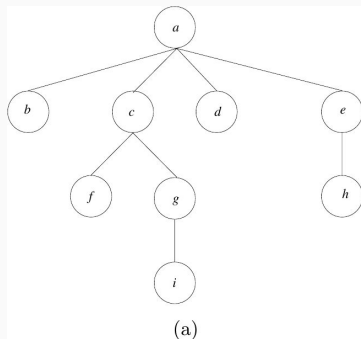
Vertex	a	b	c	d	e	f	g	h	i
R0	1	0	1	0	1	0	1	0	1
R1	1	0	0	0	1	0	1	1	1

(d)

- *Ones-based numbering:*
 $x = \text{select}_1(\text{rank}_0(x))$
- *Zero-based numbering:*
 $\text{select}_0(\text{rank}_0(\text{select}_1(\text{rank}_0(x) - 1) + 1))$

Beispiel $\text{parent}(g)$ (Ones-based):
 Knoten g (BFS 7) $\rightarrow \text{select}_1(7) = 10$
 $\text{rank}_0(10) = 3$
 $\text{select}_1(3) = 4$
 $\text{rank}_1(4) = 3 \rightarrow \text{Knoten c (BFS 3)}$

Beispiel: Operation $\text{parent}(x)$



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0	0	0

(b)

Vertex	a	b	c	d	e	f	g	h	i
BFS	1	2	3	4	5	6	7	8	9
1-based	1	3	4	5	6	9	10	13	16
0-based	7	8	11	12	14	15	17	18	19

(c)

Vertex	a	b	c	d	e	f	g	h	i
R0	1	0	1	0	1	0	1	0	1
R1	1	0	0	0	1	0	1	1	1

(d)

- *Ones-based numbering:*
 $x = \text{select}_1(\text{rank}_0(x))$
- *Zero-based numbering:*
 $\text{select}_0(\text{rank}_0(\text{select}_1(\text{rank}_0(x) - 1) + 1))$

Beispiel $\text{parent}(g)$ (Zero-based):

$$\text{rank}_0(17) - 1 = 8 - 1 = 7$$

$$\text{select}_1(7) + 1 = 10 + 1 = 11$$

$$\text{rank}_0(11) = 4$$

$$\text{select}_0(4) = 11$$

Double-Numbering

- $y = \text{select}_0(x) \implies \text{rank}_0(y) = x \wedge \text{rank}_1(y) = y - x$
- analog für select_1

Double-Numbering

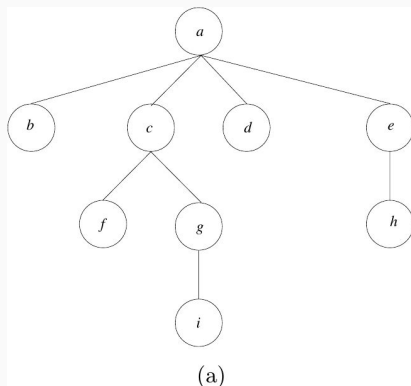
- $y = \text{select}_0(x) \implies \text{rank}_0(y) = x \wedge \text{rank}_1(y) = y - x$
- analog für select_1
- Beispiel: 0011011100
 - $\text{select}_0(4) = 9$
 - $\rightarrow \text{rank}_0(9) = 4$
 - $\rightarrow \text{rank}_1(9) = 9 - 4 = 5$

Double-Numbering

- $y = \text{select}_0(x) \implies \text{rank}_0(y) = x \wedge \text{rank}_1(y) = y - x$
- analog für select_1
- Beispiel: 0011011100
 - $\text{select}_0(4) = 9$
 - $\rightarrow \text{rank}_0(9) = 4$
 - $\rightarrow \text{rank}_1(9) = 9 - 4 = 5$
- Punkte können somit als Paar $\{x, y\}$ gespeichert werden
- Aufrufe der Form $\text{rank}(\text{select}(\dots))$ können durch einfache selects ersetzt werden
- Beispiel für parent:

```
parent({x, y}):  
    rzerox = y - x  
    newy = select1(rzerox)  
    newx = newy - rzerox  
    return {newx, newy}
```

Partitioned Representation



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0	0	0

(b)

Vertex	a	b	c	d	e	f	g	h	i
BFS	1	2	3	4	5	6	7	8	9
1-based	1	3	4	5	6	9	10	13	16
0-based	7	8	11	12	14	15	17	18	19

(c)

Vertex	a	b	c	d	e	f	g	h	i
R_0	1	0	1	0	1	0	1	0	1
R_1	1	0	0	0	1	0	1	1	1

(d)

- R_0 : Nullfolgen (mit Längen $\ell_1, \ell_2 \dots \ell_z$) kombinieren zu $R_0 = 0^{\ell_1-1}10^{\ell_2-1}1 \dots 0^{\ell_z-1}1$
- R_1 : Einsfolgen (mit Längen $\ell_1, \ell_2 \dots \ell_z$) kombinieren zu $R_1 = 0^{\ell_1-1}10^{\ell_2-1}1 \dots 0^{\ell_z-1}1$
- Alle select Operationen auf LBS durchführbar durch select_1 und rank_1 auf R_0 oder R_1
- LOUDS++ nutzt R_0 und R_1

Ergebnisse

