In-Place Binary Counters

Amr Elmasry[1,2] **and** **Jyrki Katajainen**[2,3]

[1] Alexandria University
[2] University of Copenhagen
[3] Jyrki Katajainen and Company

These slides are available from my research information system (see http://www.diku.dk/~jyrki/ under Presentations).

Numeral systems

Digit set: the values that the digits can take

Weight set: the weights that the digits represent when determining the decimal value of the underlying integer

Rule set: the rules that the representation of each integer must obey

Operation set: the operations that are to be supported

$$\begin{array}{rcccc} & 2 & 0 & 1 & 3_{10} \\ + & & & & 1_{10} \\ \hline & 2 & 0 & 1 & 4_{10} \end{array}$$

Decimal digits: $\{0, 1, 2, \ldots, 9\}$

Binary weights: $w_i = 2^i$ for $i = 0, 1, \ldots$

Regular system: a redundant binary system where every two 2's have at least one 0 in between

Operations relevant for us: ++, --, and +

The binary system we love

$$
\begin{array}{rccccccccc}
 & 1 & 1 & 1 & 1 & 1 & 1 & \ldots & 1_2 & \ell \text{ bits} \\
+ & & & & & & & & 1_2 & \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0_2 &
\end{array}
$$

Representation: $\ell + O(\lg \ell)$ bits

++/--: $\Theta(\ell/w)$ worst-case time

+: ℓ_1, ℓ_2 bits, $\Theta((\ell_1 + \ell_2)/w)$ worst-case time

w**:** machine word size in bits

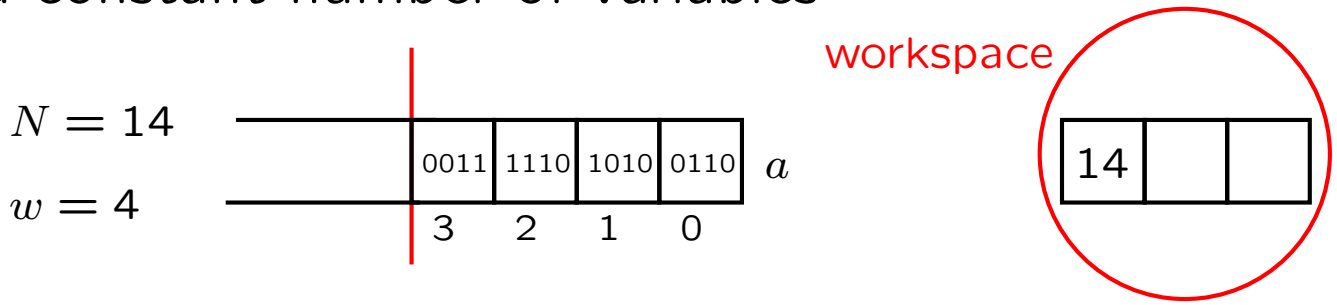
We have developed a numeral system that

- is equally space efficient as the binary system
- supports ++ and -- in $O(1)$ worst-case time
- supports + in $O((\ell_1 + \ell_2)/w)$ worst-case time.

Model of computation

A word RAM with the operations available in the C programming language; **time** means the number of word operations executed:

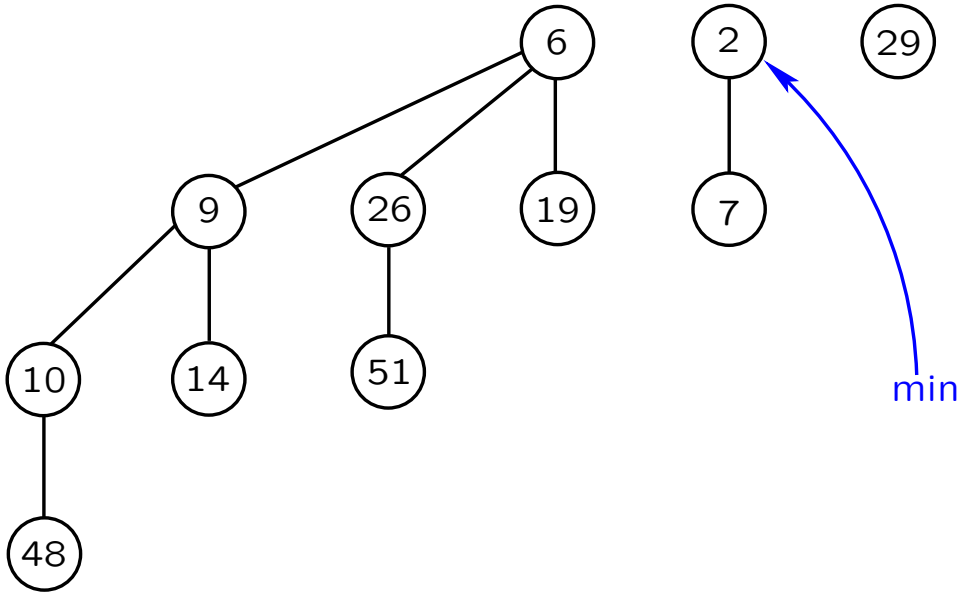
- an **infinite** array a for storing data
- a constant number of variables



- if a counter uses N bits, these must be kept in the first $\lceil N/w \rceil$ locations of a
- the word size $w \geq \lceil \lg(1 + N) \rceil$, where N is the problem size.

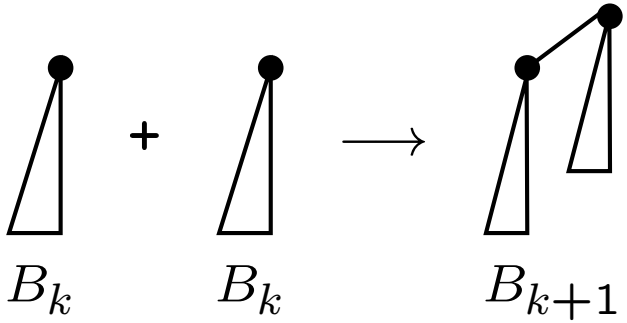
Application: Binomial queues

$N = 11_{10} = 1011_2$

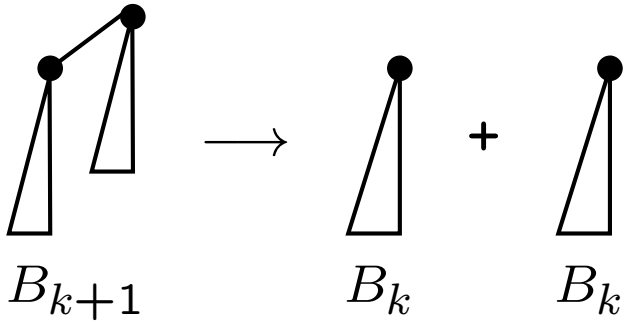


Primitives

join:



split:



Worst-case bounds

minimum: $O(1)$

insert / borrow: $\Theta(\lg N) \rightarrow O(1)$

union: two queues of size M,N,
$O(\lg M + \lg N)$

extract-min, *delete*: $\sim$ *union*

Current knowledge

Binary system

Representation: $\ell + O(\lg \ell)$ bits

++/--: $\Theta(\ell/w)$ worst-case time

+: ℓ_1, ℓ_2 bits, $\Theta((\ell_1 + \ell_2)/w)$ worst-case time

Extended regular system

Representation: $O(\ell)$ words

++d_i/--d_i: $O(1)$ digit changes in the worst case

+: ℓ_1, ℓ_2 bits, $O(\min\{\ell_1, \ell_2\})$ digit changes in the worst case (open how to use word-level parallelism)

Our system [this paper]

Representation: $\ell + O(\lg \ell)$ bits

++/--: $O(1)$ worst-case time

+: ℓ_1, ℓ_2 bits, $O((\ell_1 + \ell_2)/w)$ worst-case time

Known space-economic solutions

- analysed in the bit-probe model
- rely on Gray codes $\rightarrow$ cannot be used in data-structural applications

The regular system

Digits: $\{0, 1, 2\}$

Weights: $w_i = 2^i$ for $i = 0, 1, \ldots$

Rule: the string of digits respects the regular expression $((1|21^*0)0^*)^*$ (seen from right to left); this is not exactly the same rule as that given before!

Operations: ++, --, and +

$$2\ 0\ 5_{10} =$$

128	64	32	16	8	4	2	1	
1	0	1	2	1	1	0	1	regular

block

++ in the regular system

Algorithm

1. $d_0 \leftarrow d_0 + 1$
2. fix the first 2

Fix: $x2 \to (x+1)0$, i.e. move a carry one position forward

Data structure: a stack of the positions of all 2's

$$\begin{array}{rcccccc} & 1 & 2 & 1 & 1 & 0 & 1 \text{ regular} \\ + & & & & & & 1 \text{ regular} \\ \hline & 1 & 2 & 1 & 1 & 1 & 0 \text{ regular} \end{array}$$

Proof of correctness

1. $d_0 = 0$
- **1.1.** part of a block
  - **1.1.1.** after the block comes a singleton 0
  - **1.1.2.** after the block comes a singleton 1
  - **1.1.3.** after the block comes another block
- **1.2.** singleton 0

2. $d_0 = 1$
- **2.1.** followed by a singleton 0
- **2.2.** followed by a singleton 1
- **2.3.** followed by a block

Motivating questions

1. How would you improve the space efficiency of the regular system?
2. How would you extend the regular system to support -- in $O(1)$ worst-case time?

$$\begin{array}{rl} & 1\ 0\ 0\ 0\ 0\ 0\ \ldots\ 0 \text{ regular} \\ - & \qquad\qquad\qquad\quad\ 1 \text{ regular} \\ \hline \end{array}$$

Our in-place system

Let $\boldsymbol{d} = \langle d_{\ell-1}, \ldots, d_1, d_0 \rangle$ and $K = \sum_{i=0}^{\ell-1} d_i w_i$.

Digits: **(a)** d_0 is a basket of 1's, call their number x

(b) there may exist at most one α such that $d_\alpha = 2$

(c) $d_i \in \{0, 1\}$ for all $i \neq 0$ and $i \neq \alpha$

Weights: $w_i = 2^i$ for $i = 0, 1, \ldots$

Rules: **(a)** $\boldsymbol{d}$ is of the form $((1(0|1)^*)^* x \,|\, ((1(0|1)^*)^* 20^* x$

(b) $\sum_{i=0}^{\ell-1} d_i \leq \lceil \lg(1 + K) \rceil$

Operations: ++, --, and +

Our system is as the regular system, but we only allow at most one 2, except that x can become as high as $\lceil \lg(1 + K) \rceil$.

Actual representation

50_{10} when only ++ is supported

1	0	1	1	0	–
b_5	b_4	b_3	b_2	b_1	b_0

$$
\begin{aligned}
\ell &= 6 \text{ (length of the representation)} \\
x &= 2 \text{ (size of the basket of 1's)} \\
carry &= 1 \text{ (indicates that there is a 2)} \\
\alpha &= 2 \text{ (position of the 2, if any)} \\
\zeta &= 2 \text{ (number of zeros)} \\
\beta &= \gamma \text{ (normal mode)} \\
\gamma &= \beta \text{ (normal mode)}
\end{aligned}
$$

Key ideas

Basket $\equiv x$

- not too big: $\sum_{i=0}^{\ell-1} d_i \approx \ell$
- not too small: $x > 0$

First non-zero digit

- position may be lost

$$
\begin{array}{cccccccc}
 & & & & & \downarrow & & \\
 & 1 & 0 & 0 & 0 & 1 & x\,\text{our} \\
- & & & & & & 1\,\text{our} \\
\hline
 & 1 & 0 & 0 & 0 & 0 & (x+1)\,\text{our}
\end{array}
$$

Modes

normal mode: $\gamma = \beta$

search mode: $d_\gamma = 0,\ \gamma \leftarrow \gamma + 2$

borrow mode: $d_\gamma \neq 0,\ \gamma \leftarrow \gamma - 2$

$$
\begin{array}{cccccccc}
 & & & \gamma & & \beta & & \\
1 & 0 & 0 & 0 & 0 & 0 & x\,\text{our}
\end{array}
$$

γ: position up to which the search has reached when searching for the first non-zero digit

β: position up to which the postponed borrows should be executed

++ in our system

Algorithm

1. **if** $\gamma = \beta$ (normal mode)
2. **if** there is a 2 (ignore x)
   fix it
3. **else if** no 2 **and** $x \geq 2$
   $x \leftarrow x - 2$
   ++d_1
4. **else** (other modes)
5. $\beta \leftarrow \beta + 2$
6. $x \leftarrow x + 1$

			α		γ / β	
	1	0	2	0	0	3 our
+						1 our
	1	1	0	0	0	4 our
+						1 our
	1	1	0	0	1	3 our

			γ		β	
	1	1	0	0	0	3 our
+						1 our
	1	1	0	0	0	4 our

The idea for --

Implement -- as the reverse of ++. However, this approach fails when there is a long borrow sequence.

Solution:

- Keep a basket of 1's (x)
- Search for the first d_i, $d_i \neq 0$ and $i \geq 1$, in double speed
- Fix the borrows in double speed (until γ meets β)
- Take 1 from the basket in connection with every --.

Key: $x \geq$ #0's in the front of the first non-zero digit

Note: We should be able to support ++'s when we are not in the normal mode. Whenever $\gamma > \beta$, it means that there have been more --'s than ++'s since we switched to the search mode.

Proof of correctness

- Show that after each operation the form is right:

$$((1(0|1)^*)^* x \mid ((1(0|1)^*)^* 20^* x$$

- Show that a stronger invariant holds:

$$\sum_{i=0}^{\ell-1} d_i = \lceil \lg(1+K) \rceil - (\gamma - \beta)/2$$

$K = \sum_{i=0}^{\ell-1} d_i w_i$

β: position up to which the postponed borrows should be executed

γ: position up to which the search has reached when searching for the first non-zero digit

For details, read the proceedings!

+ in our system

- Add the bit representations, the carries, and the values of the least significant digits using binary addition
- Convert the resulting binary number back to the required form (in particular, x must be larger than ζ)

  ℓ: length of the representation

  x: size of the basket of 1's

  $carry$: 1 if there is a 2

  α: position of the 2, if any

  ζ: number of zeros

  β: position up to which the postponed borrows should be executed

  γ: position up to which the search has reached when searching for the first non-zero digit

Claim: The worst-case running time is $O((\ell_1 + \ell_2)/w)$

Question: Which primitives should the machine support?

Perspective

- Implementation: No random access needed
- Digit changes per ++/--: $O(1)$ if the basket is in unary form
- Other useful operations: == 0: $O(1)$ worst-case time; -, ==, <: $O((\ell_1 + \ell_2)/w)$ worst-case time
- Primary schools in 2063: Our system in use
- Future computer hardware: Our system in use

I am done! Thank you!