Fusion Tree
Random access machine (RAM)

- Memory divided into cells, each containing $w$ bits.
- $w \geq \log n$.
- Given an index of a cell, the content of the cell can be obtained in constant time.
- Standard operations in constant time: +, -, *, /, $\ll, \gg$, AND, OR, XOR...
A static ordered dictionary stores a set $S$ whose items are from an ordered universe $U$, and supports the following queries:

**Successor**($S, q$) Find the smallest $s \in S$ s.t. $s \geq q$.

**Predecessor**($S, q$) Find the largest $s \in S$ s.t. $s \leq q$.

**Example**

$S = \{1, 4, 8, 10\}$.

Successor($S, 5$) = 8

Predecessor($S, 5$) = 4

A sorted array gives $\Theta(\log n)$ time per operation.
A static ordered dictionary stores a set $S$ whose items are from an ordered universe $U$, and supports the following queries:

**Successor**($S$, $q$) Find the smallest $s \in S$ s.t. $s \geq q$.

**Predecessor**($S$, $q$) Find the largest $s \in S$ s.t. $s \leq q$.

**Example**

$S = \{1, 4, 8, 10\}$.

Successor($S$, $5$) = 8

Predecessor($S$, $5$) = 4

Suppose that $U$ is the set of numbers with $w$ bits. Are there static ordered dictionaries with linear space and $o(\log n)$ time per operation?
A static ordered dictionary stores a set $S$ whose items are from an ordered universe $U$, and supports the following queries:

- **Successor**($S, q$) Find the smallest $s \in S$ s.t. $s \geq q$.
- **Predecessor**($S, q$) Find the largest $s \in S$ s.t. $s \leq q$.

**Example**

$S = \{1, 4, 8, 10\}$.

- $\text{Successor}(S, 5) = 8$
- $\text{Predecessor}(S, 5) = 4$

$y$-fast trie implements a (dynamic) ordered dictionary with $\Theta(\log \log u)$ time per operation. Taking $u = 2^w$ gives $\Theta(\log w)$ time.
An internal node contains $k$ items, and has $k + 1$ children.

The height of a balanced tree is $\Theta(\log_k n)$.

To perform Predecessor($S$, $q$), find the pred. of $q$ in the the items of the current node $v$. If the pred. is the $i$-th item, continue to the $i + 1$-th child of $v$, and if there is no pred., continue the 1-st child.

At the end, return the last predecessor found in a node.

Time complexity (balanced tree):
An internal node contains $k$ items, and has $k + 1$ children.

The height of a balanced tree is $\Theta(\log_k n)$.

To perform Predecessor($S$, $q$), find the pred. of $q$ in the items of the current node $v$. If the pred. is the $i$-th item, continue to the $i + 1$-th child of $v$, and if there is no pred., continue the 1-st child.

At the end, return the last predecessor found in a node.

Time complexity (balanced tree): $\Theta(\log_k n)$.
(k+1)-ary search tree.

An internal node contains $k$ items, and has $k + 1$ children.

The height of a balanced tree is $\Theta(\log_k n)$.

To perform Predecessor($S$, $q$), find the pred. of $q$ in the items of the current node $v$. If the pred. is the $i$-th item, continue to the $i + 1$-th child of $v$, and if there is no pred., continue the 1-st child.

At the end, return the last predecessor found in a node.

Time complexity (balanced tree):
A $(k+1)$-ary search tree.

- An internal node contains $k$ items, and has $k + 1$ children.
- The height of a balanced tree is $\Theta(\log_k n)$.
- To perform $\text{Predecessor}(S, q)$, find the predecessor of $q$ in the items of the current node $v$. If the predecessor is the $i$-th item, continue to the $i + 1$-th child of $v$, and if there is no predecessor, continue the 1-st child.
  At the end, return the last predecessor found in a node.
- Time complexity (balanced tree):

$$\Theta(\log_k n \cdot \log_k n) = \Theta(\log n).$$
An internal node contains \( k \) items, and has \( k + 1 \) children.

The height of a balanced tree is \( \Theta(\log_k n) \).

To perform \( \text{Predecessor}(S, q) \), find the pred. of \( q \) in the the items of the current node \( v \). If the pred. is the \( i \)-th item, continue to the \( i + 1 \)-th child of \( v \), and if there is no pred., continue the 1-st child.

At the end, return the last predecessor found in a node.

Time complexity (balanced tree):
\[ \Theta(\log_k n \cdot \log k) = \Theta(\log n). \]
A Fusion tree is a balanced \((k + 1)\)-ary search tree with 
\[ k = \left\lfloor \frac{1}{2} w^{1/5} \right\rfloor. \]
The items in a node are stored in a way that allows finding the
pred. of query \(q\) among these items in \(\Theta(1)\) time.
Let $x_1 < x_2 < \cdots < x_k$ the items of a node $v$.

Let $q$ be a query.

Suppose that $x_1, \ldots, x_k$ and $q$ are $L$-bit integers, where $(L + 1) \cdot k \leq w$.

Pack $x_1, \ldots, x_k$ into $x'$, separated by 1's.

Pack $k$ copies of $q$ into $q'$, separated by 0's.

$x'$ and $q'$ have $\leq w$ bits.

**Example**

$x_1, \ldots, x_4 = 0, 2, 8, 11$, $q = 5$

$x' = 10000100101100011011$

$q' = 00101001010010100101$
Parallel comparison

Example

\[x_1, \ldots, x_4 = 0, 2, 8, 15, \quad q = 5\]

\[x' = 10000100101100011011\]

\[q' = 00101001010010100101\]

Compute \(y = (x' - q') \AND \text{mask}\), where \(\text{mask} = (10^l)^k\).
Parallel comparison

Example

\(x_1, \ldots, x_4 = 0, 2, 8, 15, \ q = 5\)

\[x' = \begin{array}{c} 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \end{array} \]

\[q' = \begin{array}{c} 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \end{array} \]

\[x' - q' = \begin{array}{c} 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \end{array} \]

- Compute \(y = (x' - q') \ \text{AND} \ \text{mask}\), where \(\text{mask} = (10^l)^k\).
Parallel comparison

Example

\[ x_1, \ldots, x_4 = 0, 2, 8, 15, \ q = 5 \]

\[
\begin{align*}
x' &= 10000100101100011011 \\
q' &= 00101001010010100101 \\
x' - q' &= 01011011011001110110 \\
\text{mask} &= 10000100001000010000 \\
y &= (x' - q') \ AND \ mask = 00000000001000010000
\end{align*}
\]

- Compute \( y = (x' - q') \ AND \ mask \), where \( \text{mask} = (10^l)^k \).
### Parallel comparison

**Example**

\[ x_1, \ldots, x_4 = 0, 2, 8, 15, \quad q = 5 \]

\[
\begin{align*}
    x' &= 10000100101100011011 \\
    q' &= 00101001010010100101 \\
    x' - q' &= 01011011011001110110 \\
    \text{mask} &= 10000100001000010000 \\
    y = (x' - q') \text{ AND mask} &= 00000000001000010000
\end{align*}
\]

- Compute \( y = (x' - q') \text{ AND mask} \), where \( \text{mask} = (10^l)^k \).
- The \( i \)th red bit in \( y \) is 1 iff \( q \leq x_i \).
Parallel comparison

Example

\[ x_1, \ldots, x_4 = 0, 2, 8, 15, \; q = 5 \]

\[
\begin{align*}
x' &= 10000100101100011011 \\
q' &= 00101001010010100101 \\
x' - q' &= 01011011011001110110 \\
mask &= 10000100001000010000 \\
y = (x' - q') \text{ AND mask} &= 00000000001000010000
\end{align*}
\]

- Compute \( y = (x' - q') \text{ AND mask} \), where mask = \((10^l)^k\).
- The \( i \)th red bit in \( y \) is 1 iff \( q \leq x_i \).
- Find the leftmost 1 in \((x' - q') \text{ AND mask}\).
  (or count the number of ones).
- How to handle \( w \)-bit integers?

Fusion Tree
Sketches

- Build a trie from $x_1, \ldots, x_k$.

$S = \{0, 2, 56, 59, 63\}$

![Fusion Tree Diagram]
Build a trie from $x_1, \ldots, x_k$.

There are $k - 1$ branching nodes, and $\leq k - 1$ branching levels.

$S = \{0, 2, 56, 59, 63\}$
Sketches

- Build a trie from $x_1, \ldots, x_k$.
- There are $k - 1$ branching nodes, and $\leq k - 1$ branching levels.
- $\text{sketch}_v(x) =$ bits of $x$ corresponding to branching levels.

$S = \{0, 2, 56, 59, 63\}$
Finding predecessor using sketches

- $\text{sketch}_v(x_1), \ldots, \text{sketch}_v(x_k)$ can be packed in one word (number of bits is $\leq k^2 = O(w^{2/5})$).

![Diagram showing sketch values and bit comparisons for finding predecessors.](image)
Finding predecessor using sketches

- \( \text{sketch}_v(x_1), \ldots, \text{sketch}_v(x_k) \) can be packed in one word (number of bits is \( \leq k^2 = O(w^{2/5}) \)).
- \( \text{sketch}_v(x_1) < \text{sketch}_v(x_2) < \cdots < \text{sketch}_v(x_k) \).
Finding predecessor using sketches

- \( \text{sketch}_v(x_1), \ldots, \text{sketch}_v(x_k) \) can be packed in one word (number of bits is \( \leq k^2 = O(w^{2/5}) \)).
- \( \text{sketch}_v(x_1) < \text{sketch}_v(x_2) < \cdots < \text{sketch}_v(x_k) \).
- Does the pred. of \( \text{sketch}_v(q) \) in \( \{\text{sketch}_v(x_i)\}_i \) gives the pred. of \( q \) in \( x_1, \ldots, x_k \)?
Finding predecessor using sketches

- $\text{sketch}_v(x_1), \ldots, \text{sketch}_v(x_k)$ can be packed in one word (number of bits is $\leq k^2 = O(w^{2/5})$).
- $\text{sketch}_v(x_1) < \text{sketch}_v(x_2) < \cdots < \text{sketch}_v(x_k)$.
- Does the pred. of $\text{sketch}_v(q)$ in $\{\text{sketch}_v(x_i)\}_i$ gives the pred. of $q$ in $x_1, \ldots, x_k$? **No!**
Finding predecessor using sketches

After the path of $q$ exits the trie, the bits of $\text{sketch}_v(q)$ are “noise”. We have two tasks:

1. Find where $q$ exits the trie.
2. If $q$ exits at node $w$ to the right child of $w$, find the maximum $x_i$ in the subtree of the left child of $w$ (the case when $q$ exits to the left child of $w$ is similar).
Finding predecessor using sketches

- Let pred. of \( \text{sketch}_v(q) \) in \( \{\text{sketch}_v(x_i)\}_i \) be \( \text{sketch}_v(x_j) \).
- \( y \leftarrow \) longest common prefix between \( q \) and \( x_j \) or \( x_{j+1} \) (find leftmost 1 in \( q \) XOR \( x_j \) & \( q \) XOR \( x_{j+1} \)).
- If \( |y| + 1 \)-th bit of \( q \) is 1, let \( q_2 = y_1 \cdots 1 \).
  Find pred. of \( \text{sketch}_v(q_2) \) in \( \text{sketch}_v(x_1), \ldots, \text{sketch}_v(x_k) \).
Finding predecessor using sketches

Let pred. of $\text{sketch}_v(q)$ in $\{\text{sketch}_v(x_i)\}_i$ be $\text{sketch}_v(x_j)$.

$y \leftarrow $ longest common prefix between $q$ and $x_j$ or $x_{j+1}$ (find leftmost 1 in $q \oplus x_j \& q \oplus x_{j+1}$).

If $|y| + 1$-th bit of $q$ is 1, let $q_2 = y1 \cdots 1$.

Find pred. of $\text{sketch}_v(q_2)$ in $\text{sketch}_v(x_1), \ldots, \text{sketch}_v(x_k)$. 
Computing $\text{sketch}_v(q)$

For each visited node $v$ in the search tree, we need to compute $\text{sketch}_v(q)$ in $\Theta(1)$ time!

Example

$q = 1000110$

The search path of $q$ is $v_1, v_2$.

- Keys of $v_1$: 0100100, 1000100, 1010101 $x'_v = 100110111$
- Keys of $v_2$: 1001000, 1001010, 1010010 $x'_v = 100101111$

$\text{sketch}_{v_1}(q) = 10$, $\text{sketch}_{v_2}(q) = 01$
Computing \( \text{sketch}_v(q) \)

For each visited node \( v \) in the search tree, we need to compute \( \text{sketch}_v(q) \) in \( \Theta(1) \) time!

**Example**

\( q = 1000110 \)

The search path of \( q \) is \( v_1, v_2 \).

- keys of \( v_1 \): 0100100, 1000100, 1010101 \( x'_{v_1} = 100110111 \)
- keys of \( v_2 \): 1001000, 1001010, 1010010 \( x'_{v_2} = 100101111 \)

\( \text{sketch}_{v_1}(q) = 10, \text{sketch}_{v_2}(q) = 01 \)
Computing $\text{sketch}_v(q)$

For each visited node $v$ in the search tree, we need to compute $\text{sketch}_v(q)$ in $\Theta(1)$ time!

**Example**

$q = 1000110$

The search path of $q$ is $v_1$, $v_2$.

keys of $v_1$: $0100100$, $1000100$, $1010101$  
$x'_{v_1} = 100110111$

keys of $v_2$: $1001000$, $1001010$, $1010010$  
$x'_{v_2} = 100101111$

$\text{sketch}_{v_1}(q) = 10$, $\text{sketch}_{v_2}(q) = 01$

We can use a modified sketch that contain “few” zeros between the branching bits.

**Example**

$\text{sketch}'_{v_1}(q) = 001$  
$x'_{v_1} = 100011001101$

$\text{sketch}_{v_2}(q) = 0001$  
$x'_{v_2} = 100001000111001$
Computing modified sketch

1. Mask non-branching bits.

   \[ 011010011010 \]
   \[ \text{AND} \quad 010001000010 \]
   \[ = 010000000010 \]
Computing modified sketch

1. Mask non-branching bits.
   
   \[
   \begin{align*}
   011010011010 & \quad \text{AND} \quad 010001000010 \\
   &= 010000000010
   \end{align*}
   \]

2. Use multiplication to shift bits.

   \[
   \begin{align*}
   010000000010 & \quad \times \quad 1001001 \\
   &= 010000000010 \\
   &+ 010000000010 \\
   &+ 010000000010 \\
   &= 010010010010010010
   \end{align*}
   \]

If every column contains at most one red bit, there will be no carry in the addition.
Computing modified sketch

1. Mask irrelevant bits.
   
   \[
   \begin{align*}
   011010011010 \\
   \text{AND} & \quad 010001000010 \\
   & = 010000000010
   \end{align*}
   \]

2. Use multiplication to shift bits.
   
   \[
   \begin{align*}
   010000000010 \\
   \times & \quad 1001001 \\
   & = 010010010010010010
   \end{align*}
   \]

3. Mask irrelevant bits.
   
   \[
   \begin{align*}
   010010010010010010 \\
   \text{AND} & \quad 000000011010000000 \\
   & = 0000000010010000000
   \end{align*}
   \]

Fusion Tree
Computing modified sketch

1. Mask irrelevant bits.
   
   \[
   \begin{align*}
   011010011010 \\
   \text{AND} \quad 010001000010 \\
   = \quad 010000000010
   \end{align*}
   \]

2. Use multiplication to shift bits.
   
   \[
   \begin{align*}
   010000000010 \\
   \times \quad 1001001 \\
   = \quad 010010010010010010
   \end{align*}
   \]

3. Mask irrelevant bits.
   
   \[
   \begin{align*}
   010010010010010010 \\
   \text{AND} \quad 000000011010000000 \\
   = \quad 000000010010000000
   \end{align*}
   \]

4. Truncate zeros.
   
   \[
   \begin{align*}
   000000001001000000000 \gg 7 \quad = \quad 00000001001
   \end{align*}
   \]
Computing modified sketch

- Let $b_1 < b_2 < \cdots < b_r$ be the branching bits of a node $v$ (counting from right, and starting from 0).
- Let $M$ be a number with ones in the indices $m_1 > m_2 > \cdots > m_r$.
- The relevant bits in the product are $b_i + m_i$.

**Example**

$b_1 = 1, b_2 = 6, b_3 = 10, m_1 = 6, m_2 = 3, m_3 = 0$

$$
\begin{align*}
&010000000010 \\
&\times 1001001 \\
=& 010000000010 \quad (\text{shift by } m_3) \\
+& 010000000010 \quad (\text{shift by } m_2) \\
+& 010000000010 \quad (\text{shift by } m_1) \\
=& 010010010010010010 \\
\end{align*}
$$

Relevant bits are $b_1 + m_1, b_2 + m_2, b_3 + m_3 = 7, 9, 10$
Choosing $m_i$s

### Example

$b_1 = 1, \ b_2 = 6, \ b_3 = 10, \ m_1 = 6, \ m_2 = 3, \ m_3 = 0$

$$
\begin{array}{c}
01000000010 \\
\times \ \\ 1001001 \\
\end{array}
= \begin{array}{c}
01000000010 \quad \text{(shift by } m_3) \\
+ \begin{array}{c}
01000000010 \quad \text{(shift by } m_2) \\
+ \begin{array}{c}
01000000010 \quad \text{(shift by } m_1) \\
= 010010010010010010
\end{array}
\end{array}
\end{array}
$$

Relevant bits are $b_1 + m_1, \ b_2 + m_2, \ b_3 + m_3 = 7, 9, 10$

We need to show that there are $m_i$ that satisfy

1. $b_1 + m_1 < b_2 + m_2 < \cdots < b_r + m_r$.
2. All $b_i + m_j$ values are distinct.
3. $b_r + m_r - (b_1 + m_1)$ is small.
Choosing $m_i$s

Lemma

For every $b_1, \ldots, b_r$ we can choose $m_1, \ldots, m_r$ such that

1. $b_1 + m_1 < b_2 + m_2 < \cdots < b_r + m_r$.
2. All $b_i + m_j$ values are distinct modulo $r^3$.
3. $b_r + m_r - (b_1 + m_1) \leq r^4 - 1$. 
Choosing \( m_i \)s

**Lemma**

For every \( b_1, \ldots, b_r \) we can choose \( m_1, \ldots, m_r \) such that

1. \( b_1 + m_1 < b_2 + m_2 < \cdots < b_r + m_r \).
2. All \( b_i + m_j \) values are distinct modulo \( r^3 \).
3. \( b_r + m_r - (b_1 + m_1) \leq r^4 - 1 \).

To prove the lemma, we first first show the existence of \( 0 \leq m'_1, \ldots, m'_r < r^3 \) that satisfy the 2nd property. Build the \( m'_i \)-s iteratively. The value of \( m'_t \) is chosen from

\[ \{0, \ldots, r^3 - 1\} \setminus \{m'_i + b_j - b_l \mod r^3 : i < t, j < r, l < r\} \]

This set has size \( \geq r^3 - tr^2 > 0 \).
Choosing $m_i$s

Example

$b_1 = 1, b_2 = 6, b_3 = 10$
**Choosing $m_i$s**

**Example**

\[ b_1 = 1, \ b_2 = 6, \ b_3 = 10 \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>6</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \(x+1 \neq 1\)
- \(x+1 \neq 6\)
- \(x+1 \neq 10\)
- \(x+6 \neq 1\)
- \(x+6 \neq 6\)
- \(x+6 \neq 10\)
- \(x+10 \neq 1\)
- \(x+10 \neq 6\)
- \(x+10 \neq 10\)
Choosing $m_i$s

Example

$b_1 = 1, b_2 = 6, b_3 = 10$

<table>
<thead>
<tr>
<th>$m'_i$</th>
<th>0</th>
<th>1</th>
<th>6</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_i$</td>
<td>1</td>
<td>6</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x+1 \neq 1$</th>
<th>$x \neq 1-1 = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x+1 \neq 6$</td>
<td>$x \neq 6-1 = 5$</td>
</tr>
<tr>
<td>$x+1 \neq 10$</td>
<td>$x \neq 10-1 = 9$</td>
</tr>
<tr>
<td>$x+6 \neq 1$</td>
<td>$x \neq 1-6 = 22$</td>
</tr>
<tr>
<td>$x+6 \neq 6$</td>
<td>$x \neq 6-6 = 0$</td>
</tr>
<tr>
<td>$x+6 \neq 10$</td>
<td>$x \neq 10-6 = 4$</td>
</tr>
<tr>
<td>$x+10 \neq 1$</td>
<td>$x \neq 1-10 = 18$</td>
</tr>
<tr>
<td>$x+10 \neq 6$</td>
<td>$x \neq 6-10 = 23$</td>
</tr>
<tr>
<td>$x+10 \neq 10$</td>
<td>$x \neq 10-10 = 0$</td>
</tr>
</tbody>
</table>
### Example

\[ b_1 = 1, b_2 = 6, b_3 = 10 \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>6</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fusion Tree
Choosing $m_i$s

Example

$b_1 = 1, b_2 = 6, b_3 = 10$

<table>
<thead>
<tr>
<th>$m'_i$</th>
<th>1</th>
<th>6</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$y + 1 \neq 1 \quad y + 1 \neq 3$
$y + 1 \neq 6 \quad y + 1 \neq 8$
$y + 1 \neq 10 \quad y + 1 \neq 12$
$y + 6 \neq 1 \quad y + 6 \neq 3$
$y + 6 \neq 6 \quad y + 6 \neq 8$
$y + 6 \neq 10 \quad y + 6 \neq 12$
$y + 10 \neq 1 \quad y + 10 \neq 3$
$y + 10 \neq 6 \quad y + 10 \neq 8$
$y + 10 \neq 10 \quad y + 10 \neq 12$
Choosing \( m_i \)s

**Lemma**

For every \( b_1, \ldots, b_r \) we can choose \( m_1, \ldots, m_r \) such that

1. \( b_1 + m_1 < b_2 + m_2 < \cdots < b_r + m_r \).
2. All \( b_i + m_j \) values are distinct modulo \( r^3 \).
3. \( b_r + m_r - (b_1 + m_1) \leq r^4 - 1 \).

- The \( m_i' \)s satisfy property 2, but not 1 and 3.
- Divide the integers into bins of size \( r^3 \).
- For \( i = 2, \ldots, r \), set \( m_i = m_i' + \delta_i r^3 \), where \( \delta_i \) is chosen so that \( b_i + m_i \) is in bin \( i \).
Summary

- Build a \((k + 1)\)-ary search tree, with \(k = \lfloor \frac{1}{2} w^{1/5} \rfloor \).
- For a node containing keys \(x_1, \ldots, x_k\), let \(b_1, \ldots, b_r\) be the branching bits, with \(r \leq k - 1\).
- Define a sketch function for \(v\).
- The length of the sketch of one integer is \(\leq r^4\).
- Pack the sketches of \(x_1, \ldots, x_k\) into one word.
- During a query \(q\), traverse the search tree. At each node, compute the sketch of \(q\) (and \(q_2\)), and use it to find the rank of \(q\) among \(x_1, \ldots, x_k\) in \(\Theta(1)\) time.
Build a \((k + 1)\)-ary search tree, with \(k = \lceil \frac{1}{2} w^{1/5} \rceil\).

For a node containing keys \(x_1, \ldots, x_k\), let \(b_1, \ldots, b_r\) be the branching bits, with \(r \leq k - 1\).

Define a sketch function for \(v\).

The length of the sketch of one integer is \(\leq r^4\).

Pack the sketches of \(x_1, \ldots, x_k\) into one word.

During a query \(q\), traverse the search tree. At each node, compute the sketch of \(q\) (and \(q_2\)), and use it to find the rank of \(q\) among \(x_1, \ldots, x_k\) in \(\Theta(1)\) time.

Analysis:

- A query takes \(\Theta(\log_k n) = \Theta(\log n / \log w)\) time.
Build a \((k + 1)\)-ary search tree, with \(k = \left\lfloor \frac{1}{2} w^{1/5} \right\rfloor\).

For a node containing keys \(x_1, \ldots, x_k\), let \(b_1, \ldots, b_r\) be the branching bits, with \(r \leq k - 1\).

Define a sketch function for \(v\).

The length of the sketch of one integer is \(\leq r^4\).

Pack the sketches of \(x_1, \ldots, x_k\) into one word.

During a query \(q\), traverse the search tree. At each node, compute the sketch of \(q\) (and \(q_2\)), and use it to find the rank of \(q\) among \(x_1, \ldots, x_k\) in \(\Theta(1)\) time.

**Analysis:**

- A query takes \(\Theta(\log_k n) = \Theta(\log n / \log w)\) time.
- Combining with \(\Theta(\log w)\) structure gives

\[
\Theta \left( \min \left( \frac{\log n}{\log w}, \log w \right) \right)
\]

time per query.
Summary

- \( \log w \) is monotone increasing as a function of \( w \), and \( \frac{\log n}{\log w} \) is monotone decreasing.

- The maximum of the expression \( \min \left( \frac{\log n}{\log w}, \log w \right) \) is

  \[ \text{when } \frac{\log n}{\log w} = \log w, \text{ and then } \min \left( \frac{\log n}{\log w}, \log w \right) = \sqrt{\log n}. \]

Theorem

There is a static ordered dictionary that answers queries in \( \Theta(\sqrt{\log n}) \) time.