Advanced Data Structures

Lecture 12: Dynamic Bit Vectors and Succinct Trees

Florian Kurpicz

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License ©①©: www.creativecommons.org/licenses/by-sa/4.0 | commit c70729e compiled at 2023-07-17-08:26
Recap: Rank Queries on Bit Vectors

\[ \text{rank}_\alpha(i) \] \# of \( \alpha \)s before \( i \)
\[ \text{select}_\alpha(j) \] position of \( j \)-th \( \alpha \)

\[ \text{rank}_0(5) \]
\[ \text{select}_1(5) \]

# of 0s w.r.t. super-block

# of 0s w.r.t. BV
Recap: Succinct Trees

LOUDS

```
ab ch id e j k f g
1011100110011001100000
```

BP

```
ab cd ef g h i j k
((((((((()))))))))((()))
```

DFUDS

```
a bc de fghi jk
((((((()))))))((()))((()))
```
What is a Dynamic Bit Vector?

### Dynamic Bit Vector Operations
- \( \text{insert}(BV, i, b) \) inserts \( b \) between \( BV[i - 1] \) and \( BV[i] \)
- \( \text{delete}(BV, i) \) deletes \( BV[i] \)
- \( \text{bitset}(BV, i) \) sets \( B[i] = 1 \)
- \( \text{bitclear}(BV, i) \) sets \( B[i] = 0 \)
- \( \text{bitset} \) and \( \text{bitclear} \) easy without rank and select
- \( \text{insert} \) and \( \text{delete} \) require more work

- what update time do we want to have?
  - \( O(n) \)
  - \( O(\log n) \)
  - \( O(1) \)
- is doubling the length sufficient? amortized analysis
- why not using a linked list?

Next
- dynamic bit vector including rank and select

10011010001111
01001101001111
for dynamic bit vector of size $n$
- use slowdown factor $O(w)$
- if $n$ is large, $O(w)$ becomes similar to $O(\log n)$

- query time $O(w)$
- $n + O(n/w)$ bits of space
- trade off between query time and space

- use pointer-based balanced search tree
- leaves store pointer to $\Theta(w^2)$ bits
- inner nodes store total number of bits ($num$) and number of ones ($ones$) in left subtree

$BV = 10000010 \ 00000100 \ 10000001 \ 00001010 \ 00001011$
Lemma: Practical Dynamic Bit Vectors 

Space

The dynamic bit vector requires $n + O(n/w)$ bits of space

Proof

- $\Theta(w^2)$ bits per leaf
- $O(n/w^2)$ nodes
- each (inner) node stores 2 pointers (and 2 integers)
- $O(n/w)$ bits of space in addition to $n$ bits
Practical Dynamic Bit Vectors: Access

Access

- follow path based on \( num \)
- requires \( O(\log n) \) time \( \circ \) tree is balanced
- return bit
- example on the board

- can return \( O(w^2) \) bits at the same cost
- unlike std::vector<bool>

\[
BV = 10000010 \ 00000100 \ 10000001 \ 00001010 \ 00001011
\]
Practical Dynamic Bit Vectors: Rank

- keep track of ones to the left
- update based on ones stored in node
- traverse tree accordingly in $O(\log n)$ time
- popcount on the leaf in $O(w)$ time
- example on the board

```
BV = 10000010 00000100 10000001 00001010 00001011
```
Practical Dynamic Bit Vectors: Select

- Select
  - similar to rank
  - keep track of ones
  - or number of bits minus ones for \( \text{select}_0 \)
  - traverse tree accordingly in \( O(\log n) \) time
  - \text{popcount} and scan on the leaf in \( O(w) \) time
  - example on the board

\[
BV = 10000010 \ 00000100 \ 10000001 \ 00001010 \ 00001011
\]

\[
\begin{align*}
\text{num} = 16 & \quad \text{ones} = 3 \\
\text{num} = 8 & \quad \text{ones} = 2 \\
\text{num} = 16 & \quad \text{ones} = 5 \\
10000010 & \quad 00000100 & \quad \text{num} = 8 & \quad \text{ones} = 2 & \quad 10000001 \\
00001010 & \quad 00001011
\end{align*}
\]
Practical Dynamic Bit Vectors: Insert

- inserting bit traverses down to leaf
- update `num` and `ones` on the path
- insert in bit vector at leaf
- allocate additional $w$ bits if necessary
- tracking used space requires $O(n/w)$ bits space

- at most every $w$ inserts a new allocation
- constant time copy of computer word
- are we done?

$BV = 10000010 \ 00000100 \ 10000001 \ 00001010 \ 00001011$
Maintaining Leaf Sizes (Insert)

- ensure leaves contain $\Theta(w^2)$ bits
- here $< 2w^2$ bits
- if leaf contains too many bits split leaf
- splitting can require rebalancing of tree
- (left/right) rotation is sufficient
- example on the board

Lemma: Practical Dynamic Bit Vector Insert Time

Inserting a bit in the bit vector requires $O(w + \log n)$ time

Proof

- finding leaf takes $O(w)$ time
- splitting leaf takes $O(w)$ time
- balancing tree takes $O(\log n)$ time
Practical Dynamic Rank Data Structure: Delete

- deleting bit traverses down to leaf
- update \( \text{num} \) and \( \text{ones} \) on the path
- delete in bit vector at leaf
- free \( w \) bits if possible
- tracking used space requires \( O(m/w) \) bits space

- at most every \( w \) deletes a free
- are we done?

**Diagram:**

```
num = 16  ones = 3
  num = 8  ones = 2
    num = 8  ones = 2
      1000
    00000100
    num = 8  ones = 2
    00001010
    00001011
      00001010
      00001011
1000 00000100 10000001 00001010 00001011
```

\( BV = 1000 \ 00000100 \ 10000001 \ 00001010 \ 00001011 \)
Maintaining Leaf Sizes (Delete)

- Ensure leaves contain $\Theta(w^2)$ bits
- Here > $w^2/2$ bits

- If leaf contains not enough bits, steal bits from preceding or following leaf or
- Merge leaves, merging does not result in overflow
- Merging can require rebalancing of tree
- (Left/right) rotation is sufficient
- Example on the board

Lemma: Practical Dynamic Bit Vector Insert Time

Deleting a bit in the bit vector requires $O(w + \log n)$ time

Proof
- Finding leaf takes $O(w)$ time
- Stealing bit requires $O(1)$ time
- Merging leaves takes $O(1)$ time
- Balancing tree takes $O(\log n)$ time
Practical Dynamic Rank Data Structure: Set/Unset

- if bit toggles, traverse and update ones
- toggle bit in leaf
- otherwise (unsure if bit toggles) find bit and
- if necessary backtrack path and update ones
Partial Sums

Definition: Partial Sum
Given an array $A$ containing $n$ non-negative numbers all $\leq \ell$
- $sum(A, i)$ returns $\sum_{j=0}^{i-1} A[j]$ \(\oplus\) $sum(A, 0) = 0$
- $search(A, j)$ returns $\min\{i \geq 0, sum(A, i) \geq j\}$

- what has this to do with rank and select

PINGO

- $sum$ can be answered in $O(1)$ time using $O(wn)$ bits of space
- using $S[i] = sum(A, i)$
- $search$ can be answered in $O(\log n)$ time on $S$

Sampling
- sample every $k$-th sum in $S$ of length $\lceil n/k \rceil$
- $S[i] = sum(A, ik)$
- $sum(A, i) = S[\lfloor i/k \rfloor] + \sum_{j=\lfloor i/k \rfloor k+1}^{i-1} A[j]$

- $sum$ requires $O(k)$ time
- $search$ requires $O(\log n + k)$
- requiring $O(w \lceil n/k \rceil)$ bits of space
Theoretical Dynamic Rank and Select Data Structure

- For $\ell = 1$ partial sums is rank and select on bit vectors
- $O(\log n / \log \log n)$ query time [RRR01]
- $n + o(n)$ bits of space
- Amortized update times

- $nH_0(BV) + o(n)$ bits of space with optimal query [HM14; NS14]
- $H_0$ means 0-th order empirical entropy [KM99]
- More on measurements for compressibility in lecture Text-Indexierung
What is a Dynamic Succinct Tree

<table>
<thead>
<tr>
<th>deletenode((T, v))</th>
<th>insertchild((T, v, i, k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>deletes node (v) such that</td>
<td>insert new (i)-th child of node (v) such that</td>
</tr>
<tr>
<td>(v)'s children are now children of (v)'s parent</td>
<td>the new node becomes parent of</td>
</tr>
<tr>
<td>cannot delete the root</td>
<td>the previously (i)-th to ((i + k - 1))-th child of (v)</td>
</tr>
</tbody>
</table>

- **insertchild(\(T, v, i, 0\))** inserts new leaf
- **insertchild(\(T, v, i, 1\))** inserts new parent of only the previously \(i\)-th child
- **insertchild(\(T, v, 1, \delta(v)\))** inserts new parent of all \(v\)'s children
Example of \textit{insertchild}

which one is the hardest representation to insert and delete

\textit{insertchild}(T, r, 2, 1) \quad \textit{insertchild}(T, r, 3, 0) \quad \textit{insertchild}(T, r, 2, 3)
**Definition: LOUDS**

Starting at the root, all nodes on the same depth are visited from left to right and for node $v$, $\delta(v)$ 1’s followed by a 0 are appended to the bit vector that contains an initial 10.

**deletenode($T, v$)**
- remove 0 representing leaf
- remove 1 representing edge/child
- only works efficiently with leaves

**insertchild($T, v, i, k$)**
- add 1 to node
- add 0 at next level accordingly
- only works efficiently with leaves
**Definition: BP**

Starting at the root, traverse the tree in **depth-first** order and append a
- left parenthesis if a node is visited the first time
- right parenthesis if a node is visited the last time
to the bit vector

**insertchild(T, v, i, k)**
- find parentheses representing subtree under new node
- can be empty if new leaf is inserted
- enclose these parentheses to add new node

**deletenode(T, v)**
- remove both parentheses belonging to node
Definition: DFUDS
Starting at the root, traverse tree in depth-first order and append
- for node $v$, $\delta(v)$ left parentheses and
- a right parenthesis if $v$ is visited the first time

to the bit vector that initially contains a left parenthesis $\inf\cdot$ to make them balanced

insertchild($T, v, i, k$)
- find position where node is inserted
- if $i = \delta(v) + 1$ insert at end of subtree
- insert ($^k$) $O(w)$ time if $k = O(w^2)$
- if $k > 1$ remove $k - 1$ left parentheses from $v$

deletemode($T, v$)
- find node $v$ to delete and remove it from bit vector
- update arity of parent by inserting $(\delta(v) - 1)$ before $v$'s parent
- if $v$ is leaf remove one left parenthesis instead
LOUDS and BP can be updated in time $O(t_{\text{update}})$, where

- $t_{\text{update}}$ is the time to update the bit vector.
- LOUDS can be updated in the same time, if the dynamic bit vector supports updates of blocks of size $\delta(v)$ for any node $v$.

**Dynamic Range Min-Max Tree**

- range min-max trees needed for BP and DFUDS.
- support operations in $O(\log n)$ time.
- now range min-max trees must be dynamic.
- we will see this later when introducing range min-max trees.
Conclusion and Outlook

This Lecture
- dynamic bit vectors with rank and select support
- dynamic succinct trees
- partial sum
- theoretical results for dynamic bit vectors

Next Lecture
- recap
- Q&A
- discussion project

Advanced Data Structures
- String B-tree
- SA & LCP
- PaCHash
- Successor
- CSA
- RMQ
- Kd- & Range Tree
- static BV
- range min-max tree
- static succ. trees
- succ. graphs

Institute of Theoretical Informatics, Algorithm Engineering


