Succinct and I/O Efficient Data Structures for Traversal in Trees

Craig Dillabaugh Meng He Anil Maheshwari

School of Computer Science, Carleton University, Ottawa, Ontario, Canada

October 16, 2008

Outline

1. Preliminaries
   - Succinct Data Structures
   - External Memory Model

Outline

1. Preliminaries
   - Succinct Data Structures
   - External Memory Model
2. Our Contributions

Outline

Outline

Outline

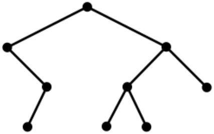
Outline

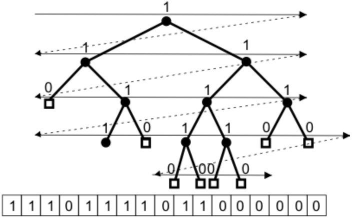
1. Preliminaries
   - Succinct Data Structures
   - External Memory Model
2. Our Contributions
3. Bottom Up Traversal
   - Navigation
   - Analysis
   - Results
4. Top Down Traversal
   - Navigation
   - Results
5. Open Problems
6. References

Succinct Data Structures

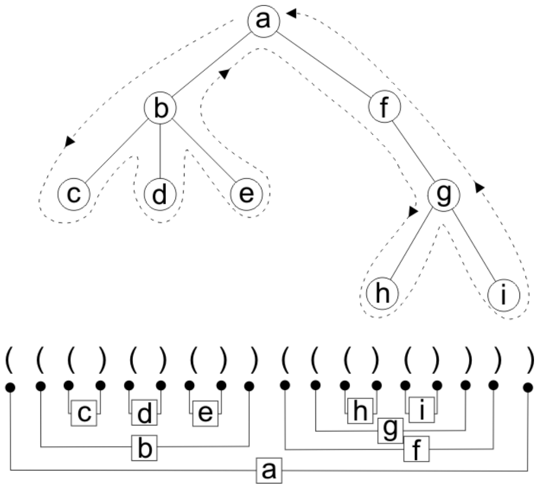
- Succinct data structures seek to encode data structures using space as near information theoretical bounds as possible.
- There are $\binom{2n}{n}/(n+1)$ binary(ordinal) trees on N nodes, approaches have been proposed to represent such trees in $2N + o(N)$ bits.
- Level order binary marked (LOBM) binary trees Jacobson [4].
- Balanced parenthesis sequences Munro and Raman [5].

Level Order Binary Marked





Balanced Parentheses



Binary Rank/Select

Given a bit-vector B we define the following operations:

- $\texttt{rank}_1(B, i)$ and $\texttt{rank}_0(B, i)$ return the number of 1s and 0s in $B[1..i]$, respectively.
- $\texttt{select}_1(B, r)$ and $\texttt{select}_0(B, r)$ return the position of the r^{th} occurrences of 1 and 0, respectively.

Lemma

A bit vector B of length N can be represented using either: (a) $N + o(N)$ bits, or (b) $\lceil \lg \binom{N}{R} \rceil + O(N \lg \lg N / \lg N)$ bits, where R is the number of 1s in B, to support the access to each bit, `rank` and `select` in $O(1)$ time (or $O(1)$ I/Os in external memory).

External Memory Model

- The I/O model of Aggarwal and Vitter [1] splits memory into fast, but finite internal memory, and slow, but infinite external memory (EM).
- Algorithms evaluated in terms of number of I/O operations (block transfers) required to complete a process.
- *Blocking* of data structures refers to partitioning data into blocks that can be transfered in a single I/O operation.

Our Contributions

- Our goal is to develop data structures that are both succinct and efficient in the EM setting.
- We have two main results:
    - A succinct encoding of arbitrary degree trees that permits bottom-up traversal in asymptotically optimal I/Os.
    - A succinct encoding of binary trees that permits top-down traversal in asymptotically optimal I/Os.

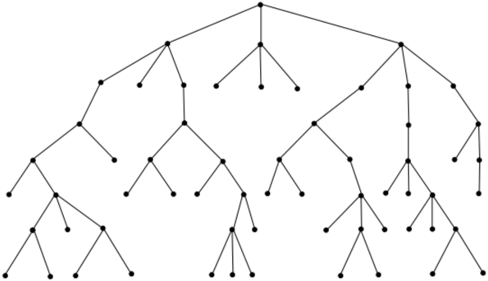
Problem Statement - Bottom Up Traversal

- Given a rooted tree T and a node $v \in T$ report the path from v to the root of T.
- By representing T in a succinct fashion we improve upon the space bound while maintaining the optimal asymptotic bound on I/Os.

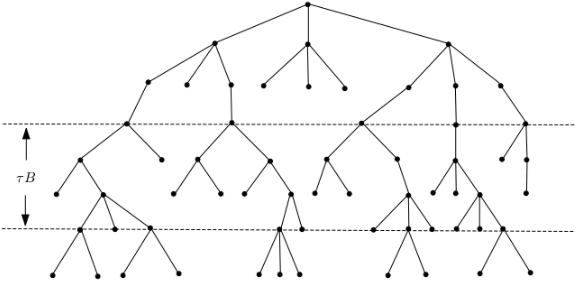
Lemma

A rooted tree T can be stored in $O(N/B)$ blocks on disk such that a bottom-up path of length K in T can be traversed in $K/\tau B$ I/Os, where $0 < \tau < 1$ is a constant. (Hutchinson et al. [3])

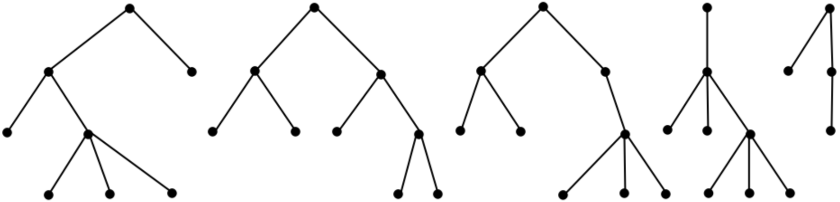
Blocking Strategy



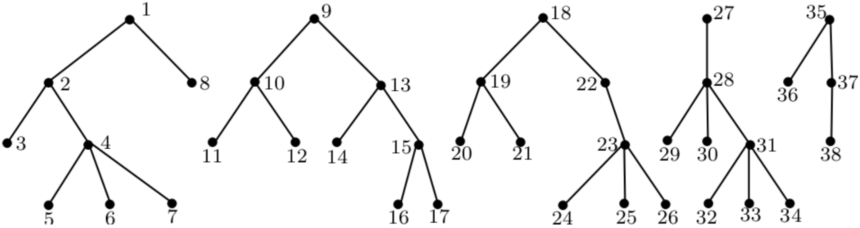
Blocking Strategy



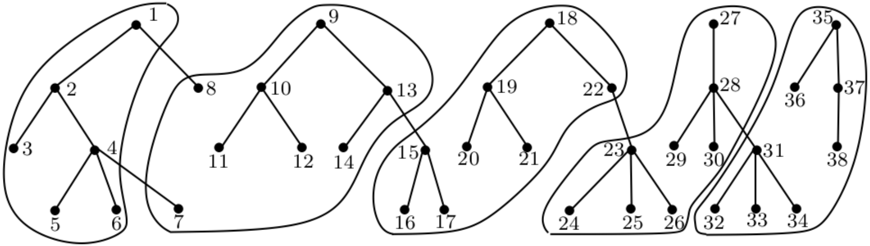
Blocking Strategy



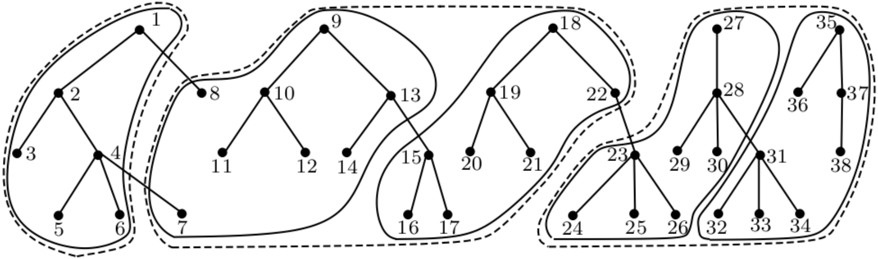
Blocking Strategy



Blocking Strategy



Blocking Strategy

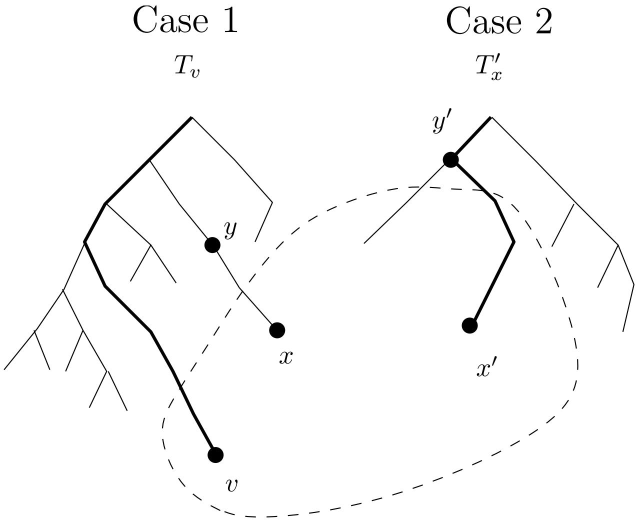


Duplicate Paths

Property 1: Given a block (or superblock) Y, for any node x in Y there exists a path from x to either the top of its layer, or to the *duplicate path* of Y, which consists entirely of nodes in Y.

1. Select as the duplicate path the path from the vertex of minimum preorder number.
2. A duplicate path is stored for each block.
3. The duplicate path of the first block in a superblock is the *superblock duplicate path*.

Proof

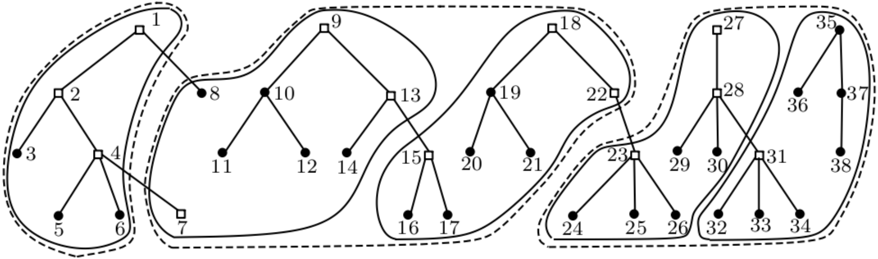


Block Encoding

Each block is encoded by three data structures:

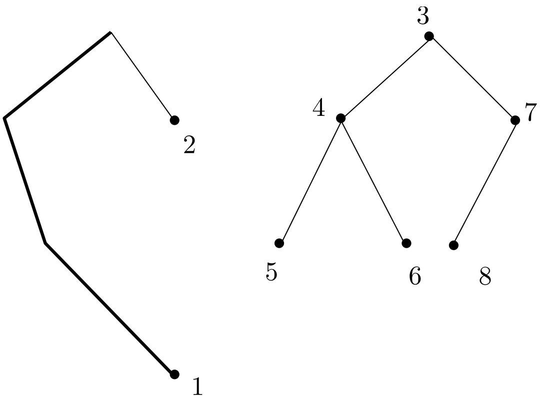
1. Tree structure is encoded using a balanced parentheses sequence.
2. The *duplicate path* is encoded as an array, $D_p[j], 1 < j < \tau B$. Duplicate paths for superblocks store the preorder value within the slice. Duplicate paths for regular blocks store preorder value within the superblock.
3. The *root-to-path array* $R_p[j], 1 < j < \tau B$ encodes the information required to map the roots of subtrees created by the blocking to nodes on the duplicate path or top of the layer.

Navigating within a block

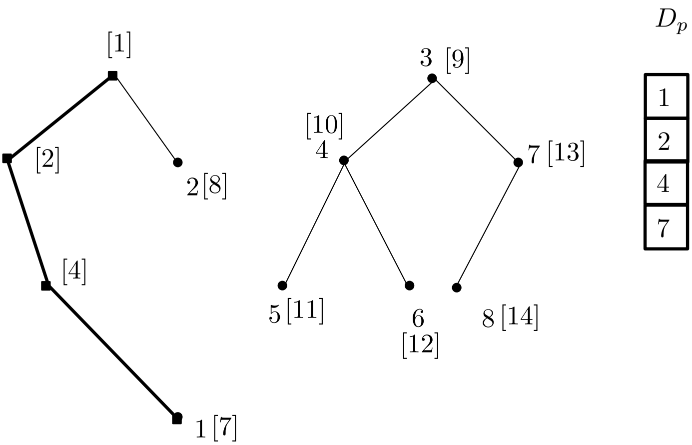


D_p	1	2	4	7
R_p	2	1	1	1

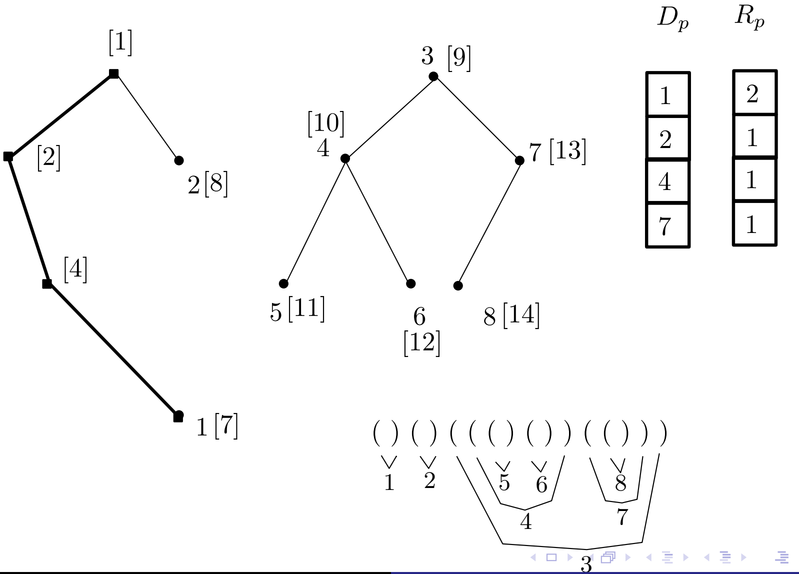
Navigating a within a block



Navigating a within a block



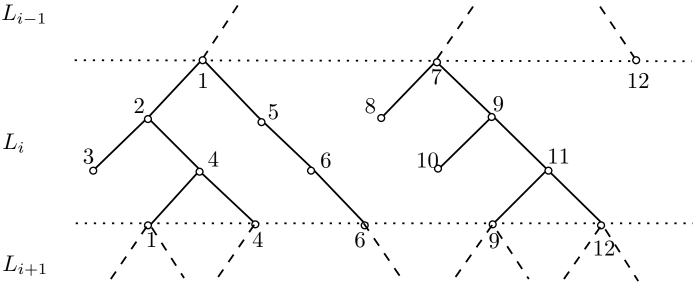
Navigating a within a block



Navigating between blocks - Identifying Nodes

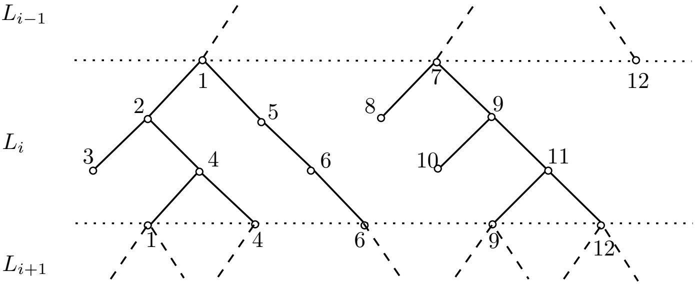
1. For the node $v \in T$ on layer ℓ_v, let p_v be its preorder number within ℓ_v.
2. Each node in T is uniquely represented by the pair (ℓ_v, p_v).
3. Let π define the lexicographic order on these pairs.

Navigating Between Blocks



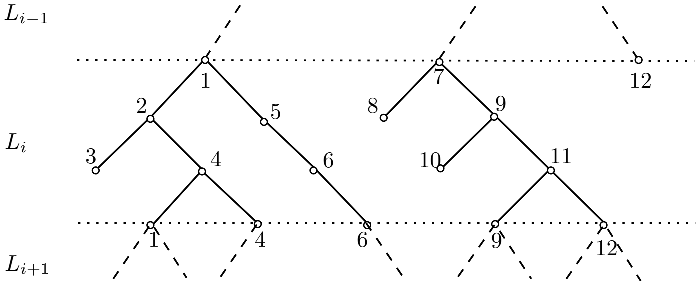
	L_{i-1}	L_i			L_{i+1}			
$\mathcal{V}_{first}$	...	1000	0000	0000	1000	0000	0000	..
$\mathcal{V}_{parent}$	...	0001	0100	0010	0000	0000	0000	..
$\mathcal{V}_{first\_child}$	...	1000	0010	0000	1000	0100	1000	..

Navigating Between Blocks



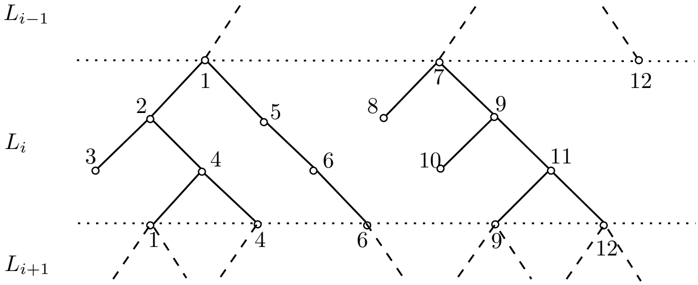
	L_{i-1}	L_i			L_{i+1}			
$\mathcal{V}_{first}$	...	1000	0000	0000	1000	0000	0000	..
$\mathcal{V}_{parent}$	...	0001	0100	0010	0000	0000	0000	..
$\mathcal{V}_{first\_child}$	...	1000	0010	0000	1000	0100	1000	..

Navigating Between Blocks



	L_{i-1}	L_i			L_{i+1}			
$\mathcal{V}_{first}$	. . .	1000	0000	0000	1000	0000	0000	. .
$\mathcal{V}_{parent}$	. . .	0001	0100	0010	0000	0000	0000	. .
$\mathcal{V}_{first\_child}$	. . .	1000	0010	0000	1000	0100	1000	. .

Navigating Between Blocks



	L_{i-1}		L_i			L_{i+1}		
$\mathcal{V}_{first}$	...	1000	0000	0000	1000	0000	0000	..
$\mathcal{V}_{parent}$	...	0001	0100	0010	0000	0000	0000	..
$\mathcal{V}_{first\_child}$	...	1000	0010	0000	1000	0100	1000	..

Space Requirements

The total space required by the data structure is:

1. Space to store the tree succinctly: $2N$ bits.
2. Space to store the bitvectors for navigating between layer: $o(N)$
3. Space to store the duplicate paths: $\frac{12\tau N}{\log_B N}$

Space Requirements for Duplicate Paths

- Duplicate paths store arrays with entries of size $\lceil \lg N \rceil$ (superblocks) or $\lceil \lg B \rceil$ (blocks).
- Our analysis is based on assumption of fixed size full blocks/superblocks.
- We cannot guarantee this so we use non-full *leading* blocks/superblocks.
- We use another set of bit vectors ($o(N)$) to enable packing/lookup of non-full leading blocks.

Results

Space Requirements:

- $2N + \frac{\epsilon N}{\log_B N} + o(N)$ when $0 < \epsilon < 1$.

I/O Efficiency:

- Given a node-to-root path of length K the path can be reported in $O(K/B)$ I/Os

Corollary

A tree T on N nodes with q-bit keys can be represented in $(2+q)N + q \cdot \left[\frac{6\tau N}{\lceil \log_B N \rceil} + \frac{2\tau q N}{\lceil \lg N \rceil} + o(N)\right]$ bits such that given a node-to-root path of length K, that path can be reported in $O(\tau K/B)$ I/Os, when $0 < \tau < 1$.

Top Down Traversal: Problem Statement

- Given a binary tree T in which every node is associated with a q-bit key, we wish to traverse a top-down path of lenght K starting at the root of T and terminating at some node $v \in T$

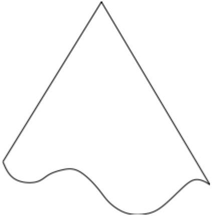
Lemma

For a binary tree T, a traversal from the root to a node of depth K requires the following number of I/Os:

1. $\Theta(K/\lg(1+B))$, *when* $K = O(\lg N)$,
2. $\Theta(\lg N/(\lg(1+B\lg N/K)))$, *when* $K = \Omega(\lg N)$ *and* $K = O(B\lg N)$, *and*
3. $\Theta(K/B)$, *when* $K = \Omega(B\lg N)$.

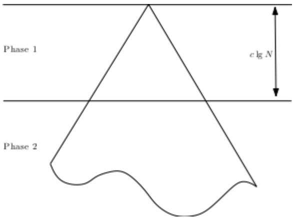
Due to Demaine et al [2]

Blocking Strategy



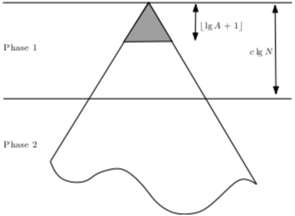
- Blocking in two phases.
- Phase 1 blocks the topmost $c \lg N$, $0 < c < 1$ layers.
- Block the first $\lfloor \lg(A+1) \rfloor$ levels of T.
- Remove blocked nodes and repeat until $c \lg N$ levels are blocked.

Blocking Strategy



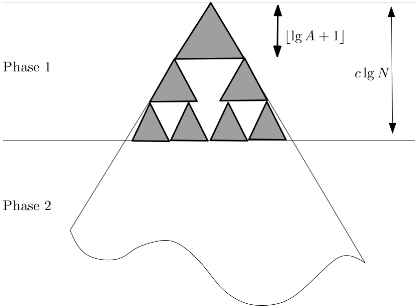
- Blocking in two phases.
- Phase 1 blocks the topmost $c \lg N$, $0 < c < 1$ layers.
- Block the first $\lfloor \lg (A+1) \rfloor$ levels of T.
- Remove blocked nodes and repeat until $c \lg N$ levels are blocked.

Blocking Strategy



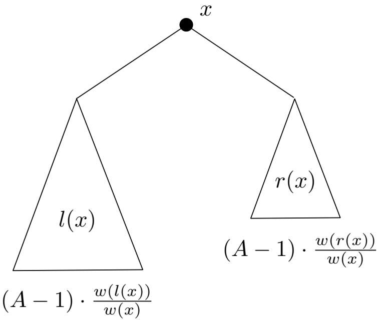
- Blocking in two phases.
- Phase 1 blocks the topmost $c \lg N$, $0 < c < 1$ layers.
- Block the first $\lfloor \lg (A + 1) \rfloor$ levels of T.
- Remove blocked nodes and repeat until $c \lg N$ levels are blocked.

Blocking Strategy



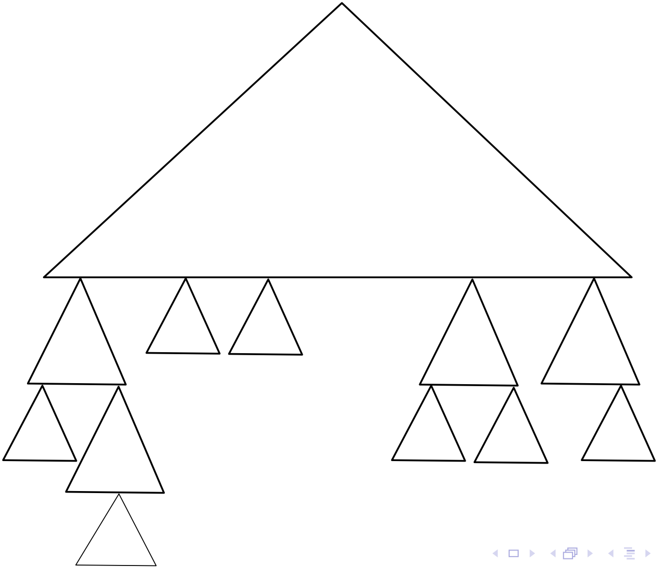
- Blocking in two phases.
- Phase 1 blocks the topmost $c \lg N$, $0 < c < 1$ layers.
- Block the first $\lfloor \lg (A + 1) \rfloor$ levels of T.
- Remove blocked nodes and repeat until $c \lg N$ levels are blocked.

Blocking Strategy: Phase 2

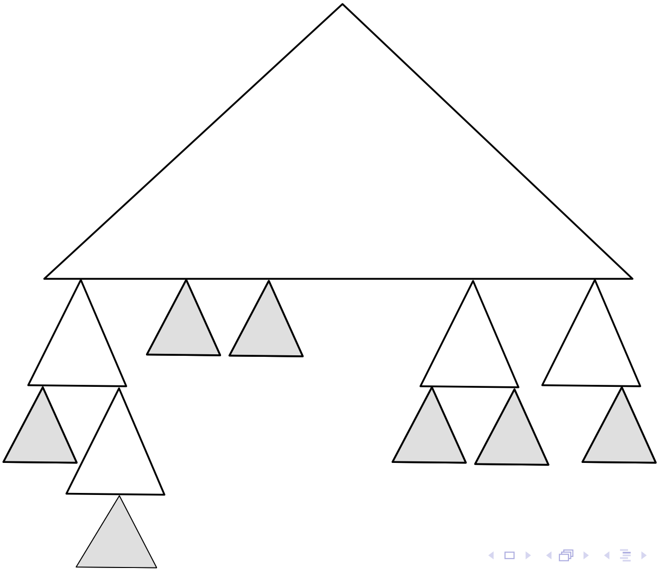


- Remaining nodes are blocked recursively.
- At node x let $w(x)$ be the size of the subtree rooted at x.
- For a block with remaining capacity A, add x and subdivide remaining capacity among x's subtrees proportional to their weights.

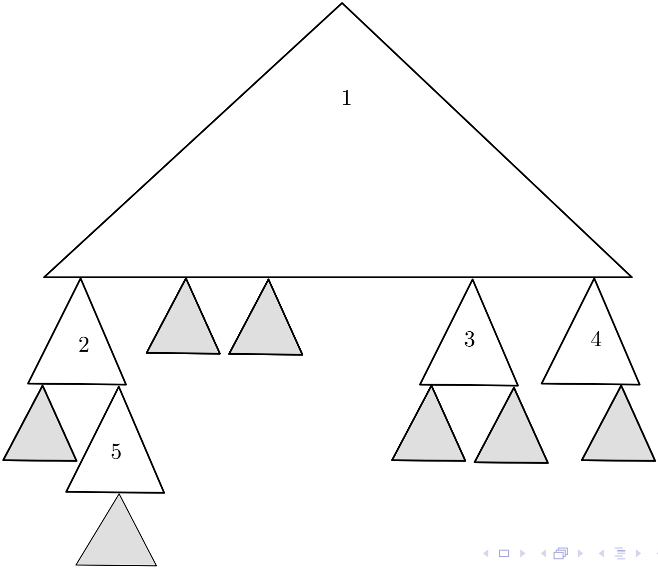
Top Down Blocking



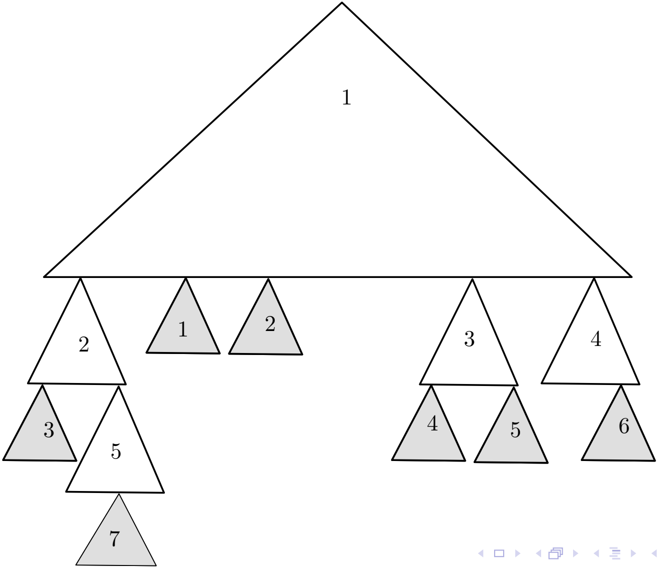
Top Down Blocking



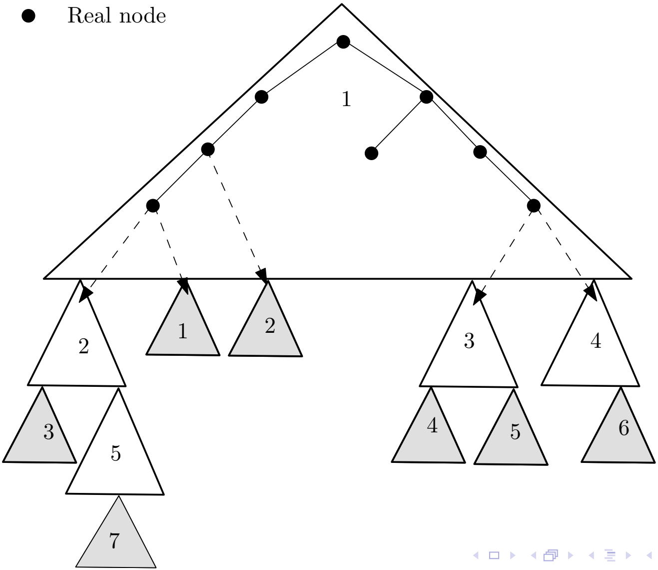
Top Down Blocking



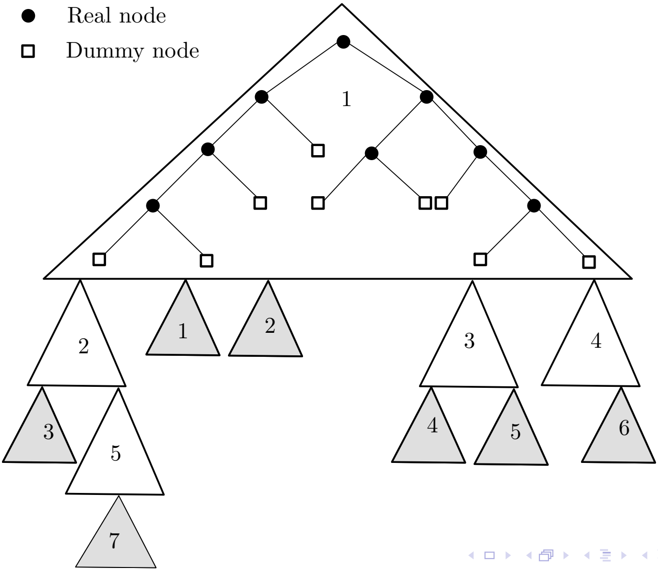
Top Down Blocking



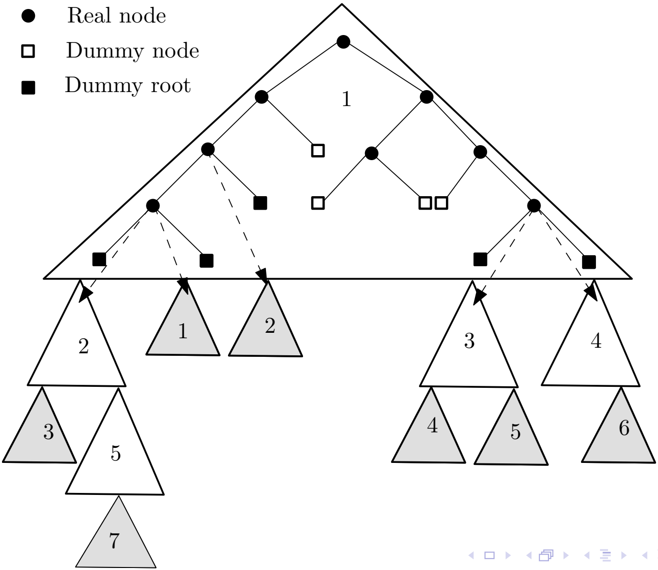
Top Down Blocking



Top Down Blocking



Top Down Blocking



Block Representation

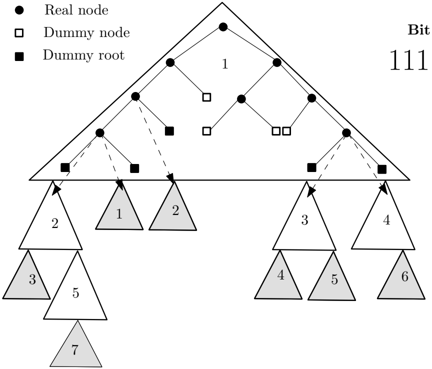
Each internal tree block stores:

- The set of keys for this block (array).
- The tree structure, using LOBM representation.
- The *dummy offset*

Dummy node ordering and dummy offset

- Let Γ be a total order over the set of all dummy nodes in internal blocks. In Γ the order of dummy node d is determined first by its block number, and second by its position within the succinct representation for its block.
- The dummy offset records the position in Γ of the first dummy node in a block.

Navigate between internal blocks



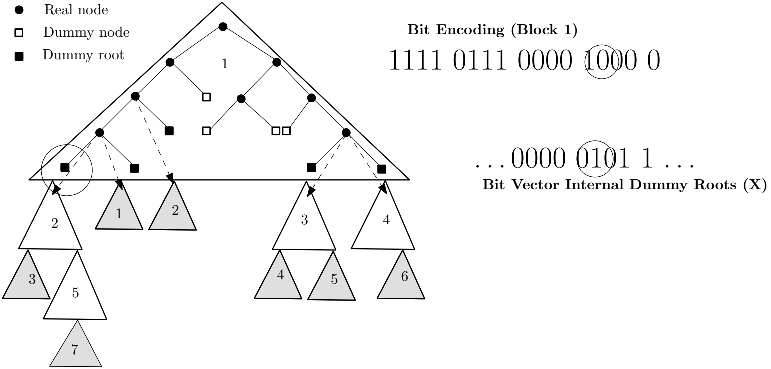
Bit Encoding (Block 1)

1111 0111 0000 1000 0

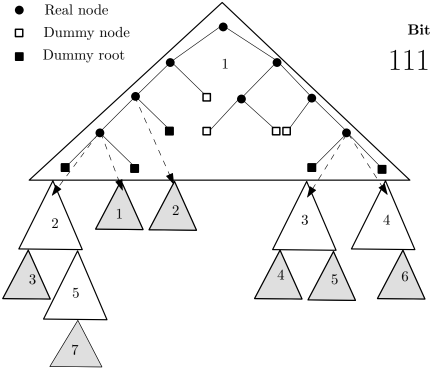
. . . 0000 0101 1 . . .

Bit Vector Internal Dummy Roots (X)

Navigate between internal blocks



Navigate between internal blocks



Bit Encoding (Block 1)

1111 0111 0000 1000 0

. . . 0000 0101 1 . . .

Bit Vector Internal Dummy Roots (X)

Navigate between internal-terminal blocks

- Similar to navigation between internal blocks.
- Use a separate bitvector S to identify roots of terminal blocks.
- Blocks may be non-full so they are packed together on disk, requiring an additional $o(N)$ bit array to identify block locations.

Results

Space requirements:

- For a rooted binary tree of size N with keys of size $q = O(\lg N)$ bits we store T in $(3+q)N + o(N)$ bits.

I/O Efficiency:

- A root to node path of length K can be reported with:
  1. $O\left(\frac{K}{\lg(1+(B \lg N)/q)}\right)$ I/Os, when $K = O(\lg N)$
  2. $O\left(\frac{\lg N}{\lg(1+\frac{B \lg^2 N}{qK})}\right)$ I/Os, when $K = \Omega(\lg N)$ and $K = O\left(\frac{B \lg^2 N}{q}\right)$, and
  3. $O\left(\frac{qK}{B \lg N}\right)$ I/Os, when $K = \Omega\left(\frac{B \lg^2 N}{q}\right)$.

Corollary

Given a rooted binary tree, T, of size N, with keys of size $q = O(1)$ bits, T can be stored using $3N + o(n)$ bits in such a manner that a root to node path of length K can be reported with:

1. $O\left(\frac{K}{\lg(1+(B \lg N))}\right)$ *I/Os when* $K = O(\lg N)$
2. $O\left(\frac{\lg N}{\lg(1+\frac{B \lg^2 N}{K})}\right)$ *I/Os when* $K = \Omega(\lg N)$ *and* $K = O\left(B \lg^2 N\right)$, *and*
3. $O\left(\frac{K}{B \lg N}\right)$ *I/Os when* $K = \Omega(B \lg^2 N)$.

Open Problems

- Top-down traversal in trees of higher bounded degree.
- Improve the I/O bound for bottom-up from $O(K/B)$ to $O(K/A)$ I/Os where A is the number of nodes that can be represented succinctly in a block.
- Improving constants in asymptotic terms for traversal.

References

Alok Aggarwal and S. Vitter Jeffrey.
The input/output complexity of sorting and related problems.
Commun. ACM, 31(9):1116–1127, 1988.

Erik D. Demaine, John Iacono, and Stefan Langerman.
Worst-case optimal tree layout in a memory hierarchy.
arXiv:cs/0410048v1 [cs:DS], 2004.

David A Hutchinson, Anil Maheshwari, and Norbert Zeh.
An external memory data structure for shortest path queries.
Discrete Applied Mathematics, 126:55–82, 2003.

Guy Jacobson.
Space-efficient static trees and graphs.
FOCS, 42:549–554, 1989.

J. Ian Munro and Venkatesh Raman.
Succinct representation of balanced parentheses and static