Succinct Data Structures

Ian Munro

University of Waterloo

General Motivation

- In Many Computations ...
  Storage Costs of Pointers and Other Structures Dominate that of Real Data
- Often this information is not “just random pointers”
- How do we encode a combinatorial object (e.g. a tree) of specialized information … even a static one

  in a small amount of space & still perform queries in constant time ???

Succinct Data Structure

Representation of a combinatorial object:

Space requirement of representation "close to" information theoretic lower bound

and

Time for operations required of the data type comparable to that of representation without such space constraints ($O(1)$)

Example : Static Bounded Subset

Given: Universe [m]= 0,…,m-1 and n arbitrary elements from this universe

Create: Static data structure to support "member?" in constant time in the lg m bit RAM model

Using: Close to information theory lower bound space, i.e. about $lg\binom{m}{n}$ bits

(Brodnik & M)

Careful .. Lower Bounds

Beame-Fich: Find largest less than i is tough in some ranges of m(e.g. $m \approx 2^{\sqrt{\lg n}}$)

But OK if i is present this can be added (Raman, Raman, Rao etc)

Focus on Trees

.. Because Computer Science is .. Arbophilic

Directories (Unix, all the rest)

Search trees (B-trees, binary search trees, digital trees or tries)

Graph structures (we do a tree based search)

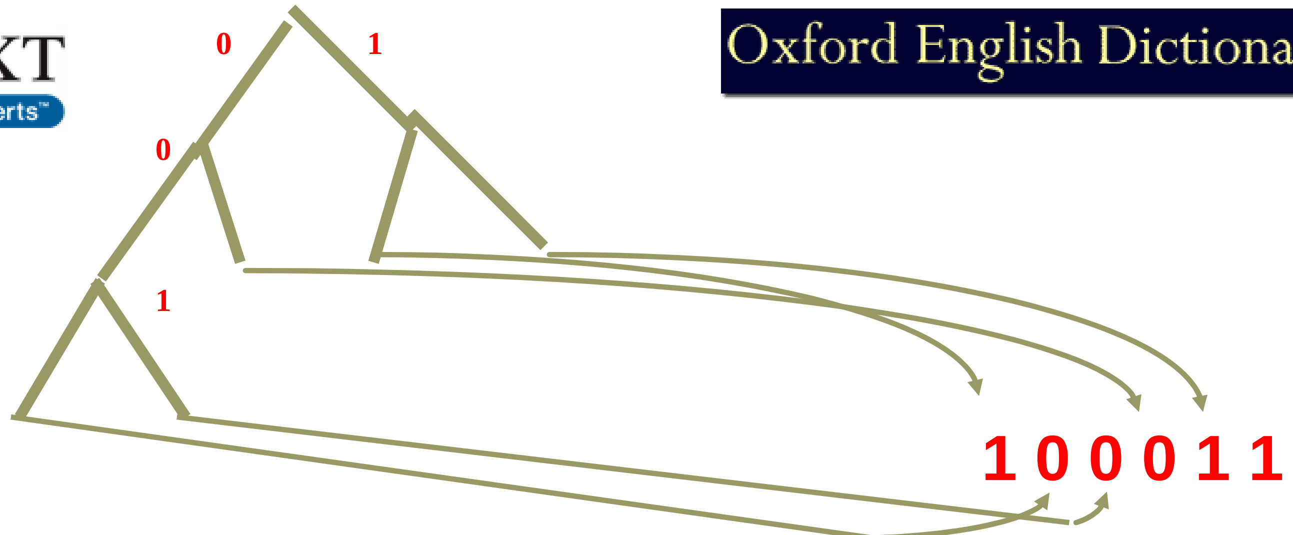
and a key application

Search indices for text (including DNA)

Preprocess Text for Search
A Big Patricia Trie/Suffix Trie

OPEN TEXT
The Content Experts™

Oxford English Dictionary



Given a large text file; treat it as bit vector

Construct a trie with leaves pointing to unique locations in text that "match" path in trie (paths must start at character boundaries)

Skip the nodes where there is no branching (n-1 internal nodes)

So the *basic* story on text search

A suffix tree (*40 years old this year*) permits search for any arbitrary query string in time proportional to the query string. But the *usual* space for the tree can be prohibitive

Most users, especially in Bioinformatics as well as Open Text and Manber & Myers went to suffix arrays instead.

Suffix array: reference to each index point in order by what is pointed to

The Issue

Suffix tree/ array methods remain extremely effective, especially for single user, single machine searches.

So, can we represent a tree (e.g. a binary tree) in substantially less space?

Space for Trees

Abstract data type: binary tree

Size: n-1 internal nodes, n leaves

Operations: child, parent, subtree size, leaf data

Motivation: "Obvious" representation of an n node tree takes about 6 n lg n bit words (up, left, right, size, memory manager, leaf reference)

i.e. full suffix tree takes about 5 or 6 times the space of suffix array (i.e. leaf references only)

Succinct Representations of Trees

Start with Jacobson, then others:

Catalan number

= # ordered rooted forests

Or # binary trees

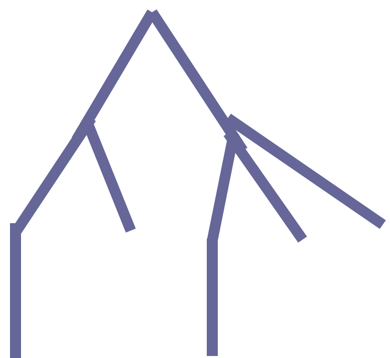
$= \frac{1}{n+1}\binom{2n}{n} \approx 4^n/(\pi n)^{3/2}$

So lower bound on specifying is about $2n$ bits

What are natural representations?

Arbitrary Order Trees

- Use parenthesis notation
- Represent the tree



- As the binary string (((())())((())()())):
  traverse tree as “(“ for node, then subtrees, then “)”
- Each node takes 2 bits

What you learned about Heaps

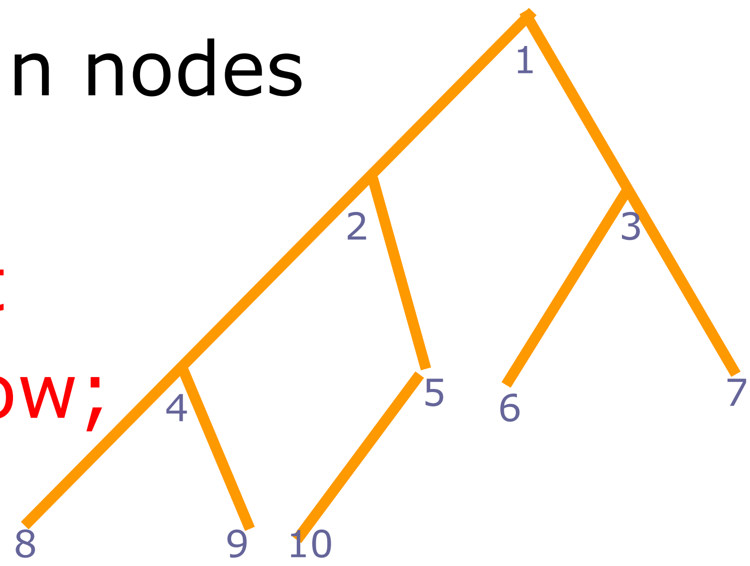
Only 1 heap (shape) on n nodes

Balanced tree,

bottom level pushed left

number nodes row by row;

lchild(i)=2i; rchild(i)=2i+1



What you learned about Heaps

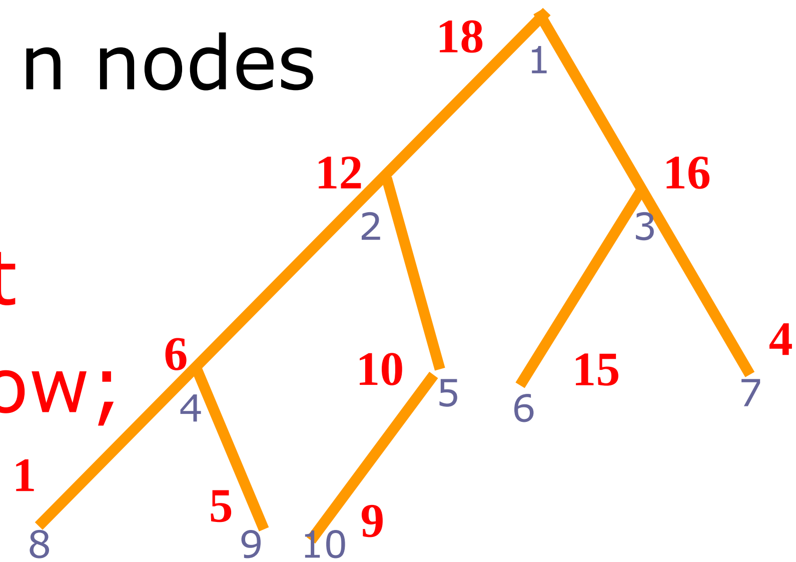
Only 1 heap (shape) on n nodes

Balanced tree,

bottom level pushed left

number nodes row by row;

lchild(i)=2i; rchild(i)=2i+1



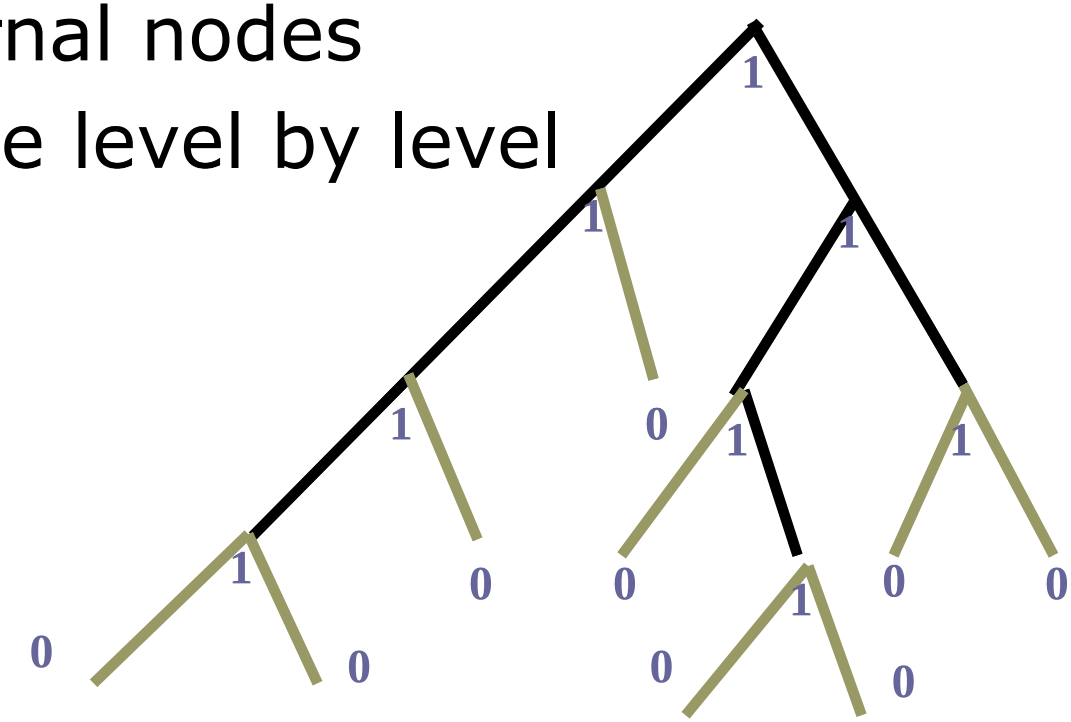
Data: Parent value > child

This gives an implicit data structure for priority queue

Generalizing: Heap-like Notation for any Binary Tree

Add external nodes

Enumerate level by level



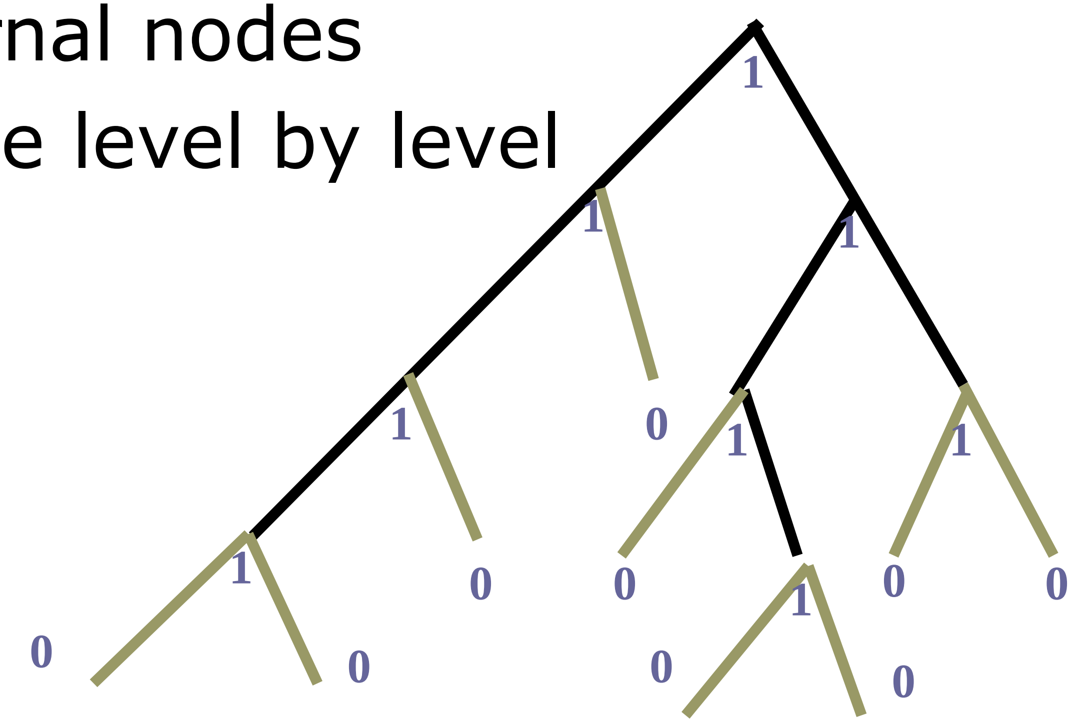
Store vector 111101111001000000 length 2n+1

(Here we don’t know size of subtrees; can be overcome. Could use isomorphism to flip between notations)

What you didn't know about Heaps

Add external nodes

Enumerate level by level



Store vector 11110111001000000 length 2n+1

(Here we don't know size of subtrees; can be overcome. Could use isomorphism to flip between notations)

How do we Navigate?

Jacobson's key suggestion:
Operations on a bit vector

rank(x) = # 1's up to & including x

select(x) = position of x^{th} 1

So in the binary tree

leftchild(x) = 2 rank(x)

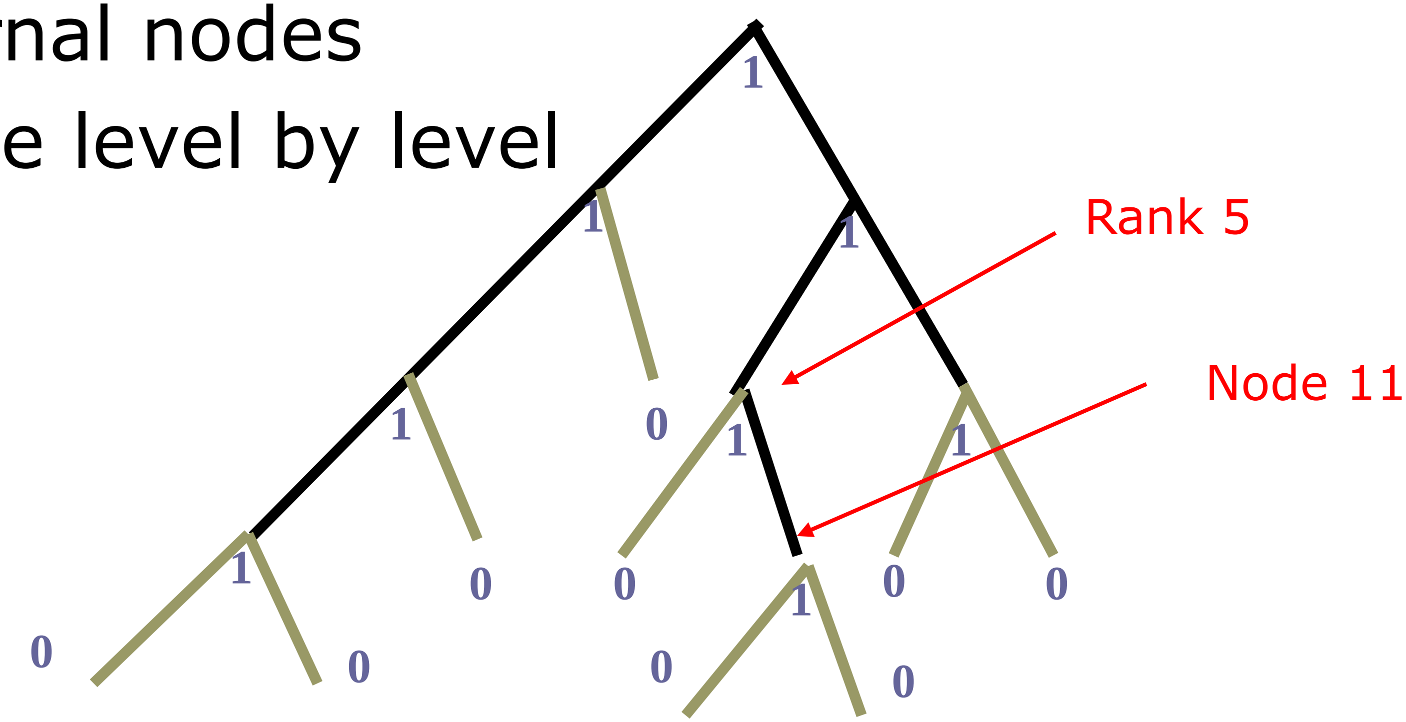
rightchild(x) = 2 rank(x) + 1

parent(x) = select($\lfloor x/2 \rfloor$)

Heap-like Notation for a Binary Tree

Add external nodes

Enumerate level by level



Store vector 1111011110010000000 length 2n+1

(Here don't know size of subtrees; can be overcome. Could use isomorphism to flip between notations)

Rank & Select

Rank: Auxiliary storage ~ 2nlglg n / lg n bits

#1's up to each $(\lg n)^2$ rd bit

#1's within these too each lg n^{th} bit

Table lookup after that

Select: More complicated (especially to get this lower order term) but similar notions

Key issue: Rank & Select take O(1) time with lg n bit word (M. et al)… as detailed on the board

Lower Bound: for Rank & for Select

Theorem (Golynski): Given a bit vector of length n and an "index" (extra data) of size r bits, let t be the number of bits probed to perform rank (or select) then: $r=\Omega(n (\lg t)/t)$.

Proof idea: Argue to reconstructing the entire string with too few rank queries (similarly for select)

Corollary (Golynski): Under the $\lg n$ bit RAM model, an index of size $\Theta(n \lg\lg n/ \lg n)$ is necessary and sufficient to perform the rank and the select operations.

Other Combinatorial Objects

- Planar Graphs (Jacobson; Lu et al; Barbay et al)
- Subset of [n] (Brodnik & M)
- Permutations $[n] \rightarrow [n]$
  - Or more generally
  - Functions $[n] \rightarrow [n]$ But what operations?
  - Clearly $\pi(i)$, but also $\pi^{-1}(i)$
  - And then $\pi^{k}(i)$ and $\pi^{-k}(i)$

More Data Types

Suffix Arrays (special permutations; references to positions in text sorted lexicographically) in linear space

Arbitrary Graphs (Farzan & M)

“Arbitrary” Classes of Trees (Farzan & M)

Partial Orders (M & Nicholson)

Conclusion

Interesting, and useful, combinatorial objects can be:

Stored succinctly … O(lower bound) +o()

So that

Natural queries are performed in O(1) time (or at least very close)

Programs: http://pizzachili.dcc.uchile.cl/index.html

This can make the difference between using the data type and not …