

# Succinct Data Structures

Ian Munro



# 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  $[m] = 0, \dots, m-1$  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 .. **Arborphilic**  
**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

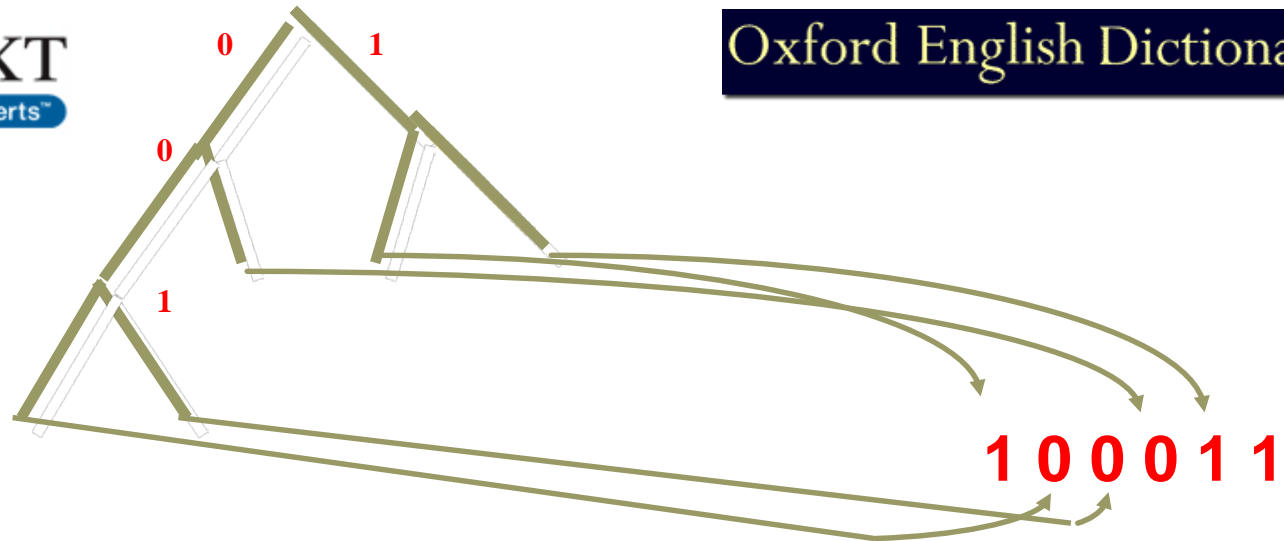
**Search indices for text** (including DNA)

# 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 last 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  $6n \lg n$  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**  $\longrightarrow$

= # 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 ... but operations?

# What you learned about Heaps

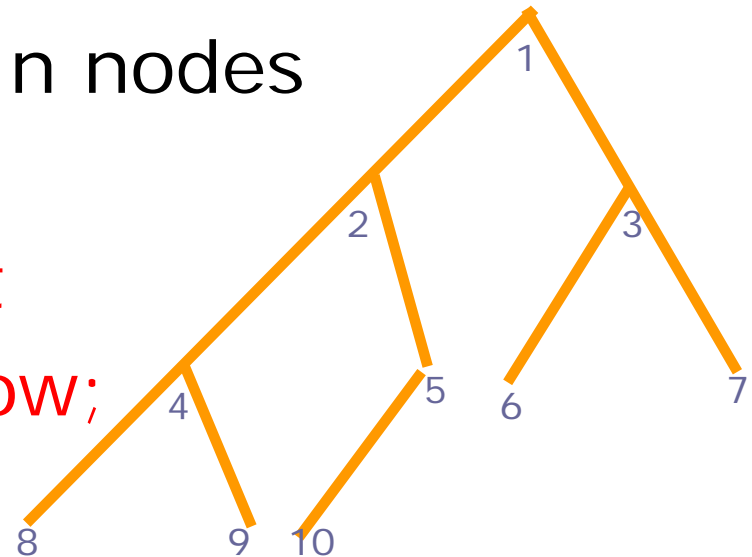
Only 1 heap (shape) on n nodes

Balanced tree,

bottom level pushed left

number nodes row by row;

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



# What you learned about Heaps

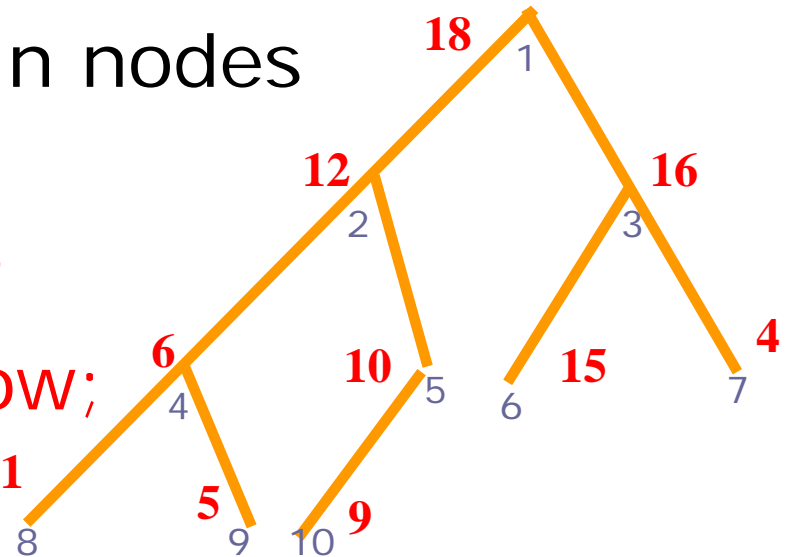
Only 1 heap (shape) on n nodes

Balanced tree,

bottom level pushed left

number nodes row by row;

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



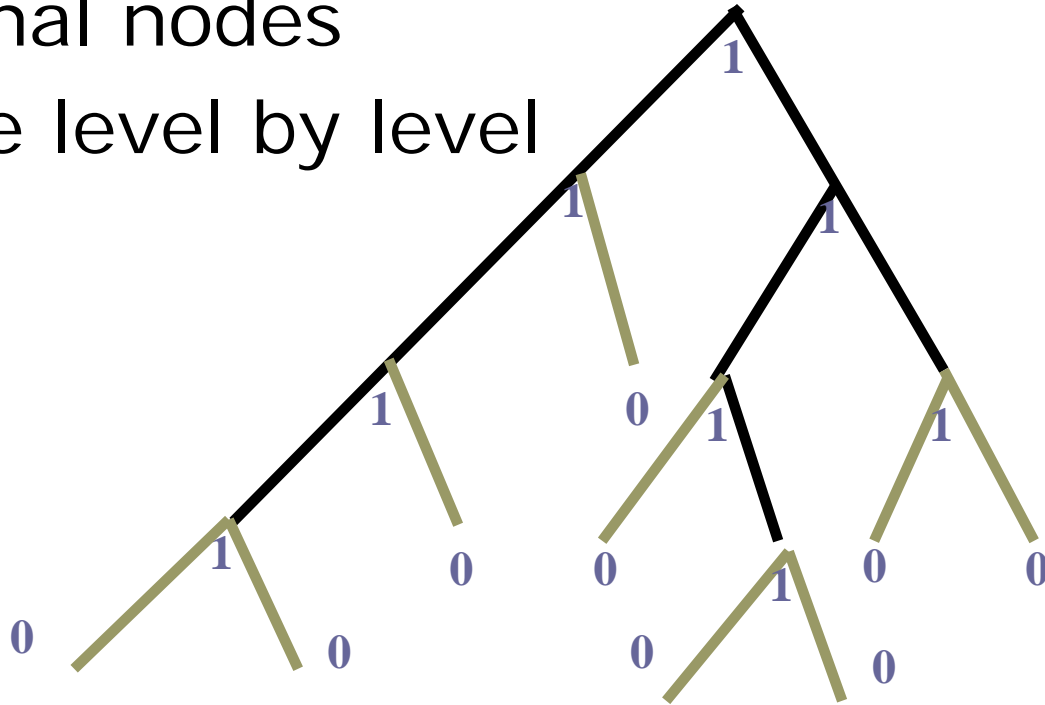
Data: Parent value  $>$  child

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 **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

$\text{rank}(x) = \# \text{ 1's up to \& including } x$

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

So in the binary tree

$\text{leftchild}(x) = 2 \text{ rank}(x)$

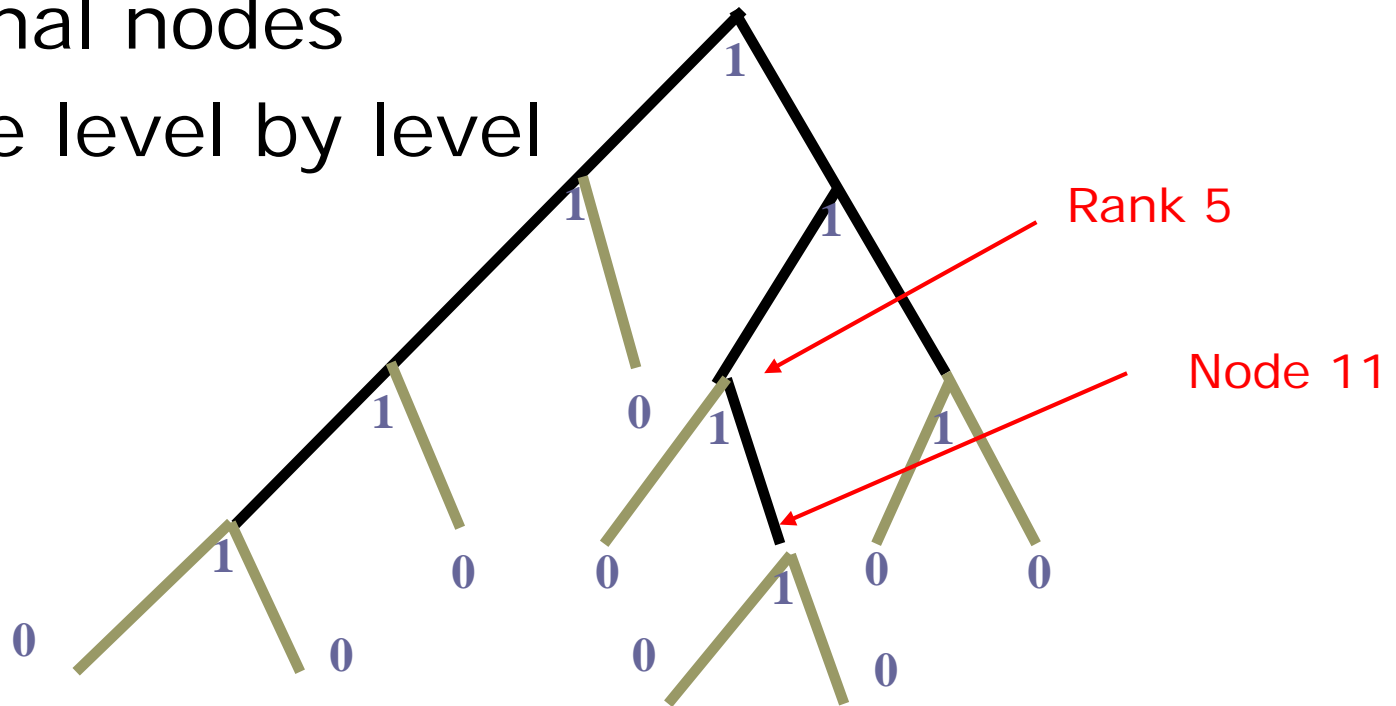
$\text{rightchild}(x) = 2 \text{ rank}(x) + 1$

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



# Heap-like Notation for a Binary Tree

Add external nodes  
Enumerate level by level



Store vector  $11110111001000000$  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  $\sim 2n \lg \lg n / \lg n$  bits

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

#1's within these too each  $\frac{1}{2} \lg n$  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 in  $O(\lg n)$  bit probes, so in  $O(1)$  time.

# 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 ... after all writing the string takes only linear space.

# “Arbitrary” Classes of Trees

---

Consider classes of trees where “all small subtrees” are members of the class.

(e.g. ordinal trees of degree at most 2)

We can represent such trees in “near optimal space” and navigate in constant time. Even if we don’t know the space lower bound!

(Arash and M)

# Partial Orders

---

**Partial order** ... the transitive closure of a directed graph.

What is the ITLB?

Represent as upper triangular 0-1 matrix.  $n^2/2$

But ~~all~~ **most** of these not “transitive closures”

Right answer  $n^2/4$

Can achieve this bound

**(Nicholson & M)**

# Arbitrary Graphs/Digraphs

---

$n$  vertices and  $m$  edges, support adjacency and degree queries

**Lower bound:** impossible to answer such queries in constant time (per node) ...

In information theory lower bound (unless the graph is very sparse ( $m = o(n^\delta)$  for any constant  $\delta > .0$ ) or (similarly) too dense.

**But** in space  $(1 + \epsilon)ITLB$ , we **can** do it.

(Farzan & M)



# But first ... how about integers

---

Of “arbitrary” size

Clearly  $\lg n$  bits ... if we take  $n$  as an upper bound

But what if we have “no idea”

Elias:  $\lg \lg n$  0's,  $\lg n$  in  $\lg \lg n$  bits,  $n$  in  $\lg n$  bits

Can we do better?

A useful trick in many representations

# Dictionary over $n$ elements [ $m$ ]

---

Brodnik & M

Fredman, Komlós & Szemerédi (FKS)

Hashing gives constant search using  
“keys” plus  $n \lg m + o()$  bits

B&M approach: Information theory lower  
bound is  $\lg \binom{m}{n}$

**Spare** and **dense** cases

*Sparse*: can afford  $n$  bits as initial index  
... several cases for sparse and for dense

# More on Trees

---

“Two” types of trees ... ordinal and cardinal  
i.e. 1<sup>st</sup> 2<sup>nd</sup> 3<sup>rd</sup> versus 1,2,3



**Cardinal trees:** e.g. Binary trees are cardinal trees of degree 2, each location “taken or not”. Number of  $k$ -ary trees

$$C_n^k = \binom{kn + 1}{n} / (kn + 1)$$

So ITLB  $\approx \lg(k - 1) + k \lg(k / (k - 1))n$  bits

# Ordinal Trees

---

Children **ordered**, no bound on number of children,  **$i^{\text{th}}$  cannot exist without  $i-1^{\text{st}}$**

These correspond to balanced parentheses expressions, Catalan number of forests on  $n$  nodes

A variety of representations .....

# But first we need: Indexable Dictionaries

---

Getting that “n” down if there are few 1’s

$S = n$  elements for  $[m]$

$\text{Rank}(i, S)$  gives # elements  $\leq i$

$\text{Select}(i, S)$  gives  $i^{\text{th}}$  smallest

in  $\text{ITLB} = B = \lg \binom{m}{n} \dots$  or so

A problem ... Atai lower bound  $\Omega(\lg \lg n)$

Sidestep by only asking for  $\text{Rank}(i, S)$  if  $i \in S$

Raman, Raman & Rao

# Trees

---

Key rule ... nodes numbered 1 to  $n$ , but data structure gets to choose “names” of nodes

Would like ordinal operations:

parent,  $i^{\text{th}}$  child, degree, subtree size

Plus child  $i$  for cardinal

# Ordinals

---

Many orderings: **L**evel**O**rder**U**nary**D**egree**S**equence

Node:  $d$  1's (child birth announcements)  
then a 0 (death of the node)

Write in level order: root has a "1 in the sky", then birth order = death order

Gives  $O(1)$  time for parent,  $i^{\text{th}}$  child, degree  
Balanced parents gives others, DFUDS ... all



---

Easy approach: Each node gets  $k$  bits,  
saying which children are there  
So  $kn$  bits, say in LOUDS or DFS order

Problem, the space lower bound:

$$\begin{aligned}\lg(C(n, k)) &\approx \lg(k - 1) + k \lg \frac{k}{k - 1} \\ &\approx (\lg k + \lg e) n \text{ bits}\end{aligned}$$

(as  $k$  grows)



# Another approach

---

Ordinal for underlying structure (say DFUDS)

Gives parent,  $i^{\text{th}}$  child, degree, subtree size

Now have to deal with child  $j$

Suppose a node has  $d$  of  $k$  children

“just” need  $i = \text{rank}(j)$ , use indexable dict.

i.e.  $d \lg k + o(d) + O(\lg \lg k)$  bits each

Can be made  $n \lg k + o(n) + O(\lg \lg k)$

no  $n$

# More on Trees

---

**Dynamic trees:** Tough going, mainly memory management

M, Storm and Raman and Raman, Raman & Rao

**Other classes:** Decomposition into **big tree** ( $o(n)$  nodes); **minitrees** hanging off (again  $o(n)$  in total); and **microtrees** (most nodes here) microtrees small enough to be coded in table of size  $o(n)$

If microtrees have "**special feature**", encoding can be optimal.. Even if you don't know what that means.

(Farzan & M)

# Permutations and Functions

---

Permutation  $\pi$ , write in natural form:

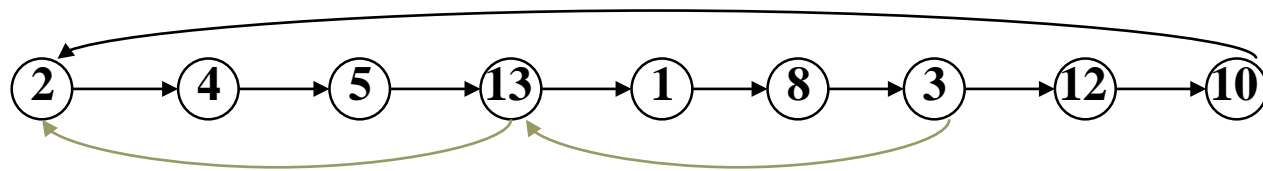
$\pi(i)$   $i = 1, \dots, n$ : space  $n \lg n$  bits, good!

Great for computing  $\pi$ , but how about  $\pi^{-1}$  or  $\pi^k$

Other option: write in cycles, mildly worse for space, much worse for any calculations above

# Permutations: a Shortcut Notation

Let  $P$  be a simple array giving  $\pi$ ;  $P[i] = \pi[i]$   
Also have  $B[i]$  as a pointer  $t$  positions back in (the **cycle** of) the permutation;  
 $B[i] = \pi^{-t}[i]$  .. But only define  $B$  for every  $t^{\text{th}}$  position in cycle. ( $t$  is a constant; ignore cycle length "round-off")



So array representation

$P = [8 \ 4 \ 12 \ 5 \ 13 \ x \ x \ 3 \ x \ 2 \ x \ 10 \ 1]$   
1 2 3 4 5 6 7 8 9 10 11 12 13

# Representing Shortcuts

In a cycle there is a  $B$  every  $t$  positions ...

But these positions can be in "arbitrary" order

Which  $i$ 's have a  $B$ , and how do we store it?

Keep a vector of all positions:  $0 = \text{no } B$   $1 = B$

**Rank** gives the position of  $B["i"]$  in  $B$  array

So:  $\pi(i)$  &  $\pi^{-1}(i)$  in  $O(1)$  time &  $(1+\epsilon)n \lg n$  bits

**Theorem:** Under a **pointer machine model** with space  $(1+\epsilon)n$  references, we need time  $1/\epsilon$  to answer  $\pi$  and  $\pi^{-1}$  queries; i.e. this is as good as it gets ... **in the pointer model.**

# Getting it $n \lg n$ Bits

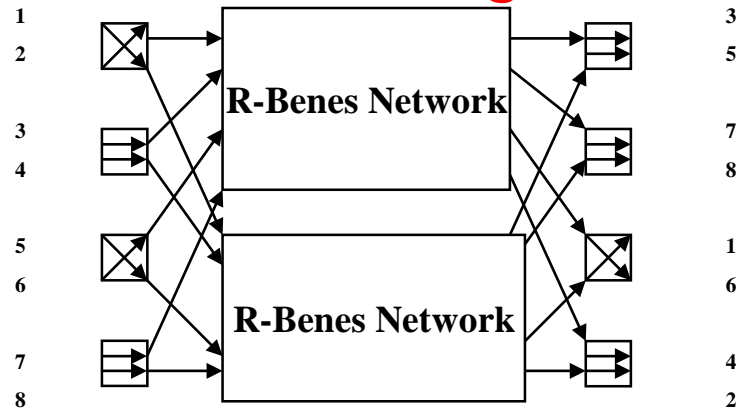
This is the best we can do for  $O(1)$  operations

But using **Benes networks**:

1-Benes network is a 2 input/2 output switch

$r+1$ -Benes network ... join tops to tops

$$\#bits(n) = 2\#bits(n/2) + n = n \lg n - n + 1 = \min + \Theta(n)$$

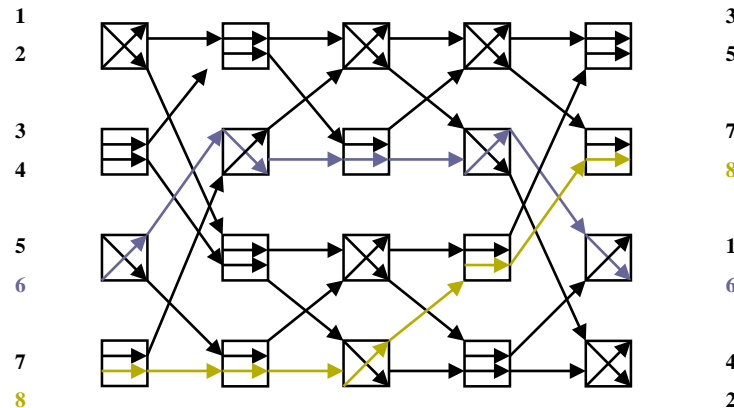


# A Benes Network

Realizing the permutation (std  $\pi(i)$  notation)

$$\pi = (5 \ 8 \ 1 \ 7 \ 2 \ 6 \ 3 \ 4) ; \pi^{-1} = (3 \ 5 \ 7 \ 8 \ 1 \ 6 \ 4 \ 2)$$

Note:  $\Theta(n)$  bits more than "necessary"



# What can we do with it?

Divide into blocks of  $\lg \lg n$  gates ... encode their actions in a word. Taking advantage of regularity of address mechanism

and also

Modify approach to avoid power of 2 issue

Can trace a path in time  $O(\lg n / (\lg \lg n))$

This is the best time we are able get for  $\pi$  and  $\pi^{-1}$  in nearly minimum space.



# Both are Best

Observe: This method “violates” the pointer machine lower bound by using “micropointers”.

But ...

More general Lower Bound (Golynski): Both methods are optimal for their respective extra space constraints

# Back to the main track: Powers of

$\pi$

Consider the cycles of  $\pi$

( 2 6 8)( 3 5 9 10)( 4 1 7)

Bit vector indicates start of each cycle

( 2 6 8 3 5 9 10 4 1 7)

Ignore parens, view as new permutation,  $\psi$ .

Note:  $\psi^{-1}(i)$  is position containing  $i$  ...

So we have  $\psi$  and  $\psi^{-1}$  as before

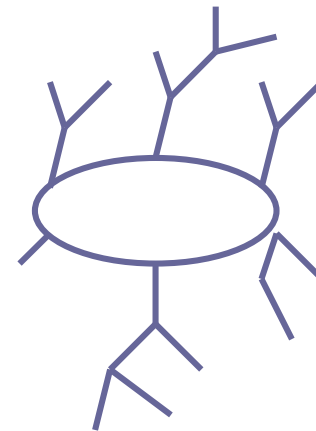
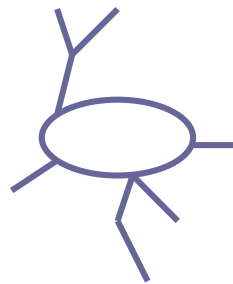
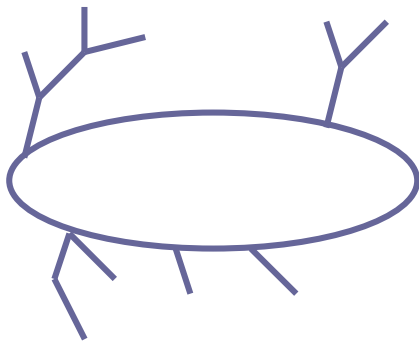
Use  $\psi^{-1}(i)$  to find  $i$ , then  $n$  bit vector (rank, select) to find  $\pi^k$  or  $\pi^{-k}$

# Functions

Now consider arbitrary functions  $[n] \rightarrow [n]$

“A function is just a hairy permutation”

All tree edges lead to a cycle



# Challenges here

Essentially write down the components in a convenient order and use the  $n \lg n$  bits to describe the mapping (as per permutations)

To get  $f^k(i)$ :

Find the level ancestor ( $k$  levels up) in a tree

Or

Go up to root and apply  $f$  the remaining number of steps around a cycle

# Level Ancestors

---

There are several level ancestor techniques using

$O(1)$  time and  $O(n)$  WORDS.

Adapt Bender & Farach-Colton to work in  $O(n)$  bits

But going the other way ...

# Level Ancestors

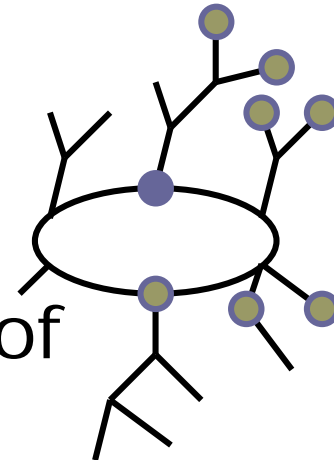
Moving **Down** the tree (toward leaves) requires care

$$f^{-3}(\bullet) = (\bullet)$$

The trick:

Report all nodes on a given level of a tree in time proportional to the number of nodes, and

Don't waste time on trees with no answers



# Final Function Result

Given an arbitrary function  $f: [n] \rightarrow [n]$

With an  $n \lg n + O(n)$  bit representation we can compute  $f^k(i)$  in  $O(1)$  time and  $f^{-k}(i)$  in time  $O(1 + \text{size of answer})$ .

$f$  &  $f^{-1}$  are very useful in several applications

... then on to binary relations (HTML markup)

# Succinct Data Structures for Representing Equivalence Classes

*Moshe Lewenstein*  
*Ian Munro*  
*Venkatesh Raman*  
*Hichen ElZein*





# Succinct Data Structures ...

## The Game

---

Represent a combinatorial object, size  $n$ ,  
using  $\lg(\# \text{ objects})$  “or so” bits

Perform the necessary queries “quickly”

“Naming” can be an issue. Primary interest  
in names  $1\dots n$ .

Not an issue for “dense graphs”, but is for  
trees, planar graphs, and definitely for  
equivalence classes

# Label Space: Direct Equivalence Queries

---

But first ... give elements distinct “names” over “smallest space” so that elements in same class can be recognized, with **no extra space**

Katz, Katz, Korman & ~~Keleg~~ Peleg studied k-connectivity:  $\Omega(k \lg n)$  bit lower bound

For our problem,  $k=1$  so  $i^{\text{th}}$  largest bucket has at most  $n/i$  elements)

Thm: Label space  $\sum_{i=1}^n \lfloor n/i \rfloor$  necessary and sufficient.

# With Care

---

Can support **constant time** equivalence class queries using labels of

$\lg n + \lg \lg n + 2$  bits.

(Put elements in appropriate  $n/i$  size buckets; need care with “breakpoints” between buckets)

Under this model  $\lg n + \lg \lg n - \Theta(1)$  are necessary (from previous theorem)

# Our Real Problem

## Labels 1 ... n (but we pick)

---

This means we have to store a data structure that distinguishes between partitions:

Hardy-Ramanujan formula  $\approx \frac{1}{4n\sqrt{3}} e^{(\pi\sqrt{2n/3})}$

Taking  $\lg$

Lower Bound:  $((\pi\sqrt{\frac{2}{3}}\lg e)\sqrt{n} - \lg n + O(1))$  bits

# $O(n^{1/2})$ bit Structure

## $O(\lg n)$ Time

---

Elements of same class numbered consec.  
Classes of same size ordered consecutively

- $k$  distinct classes sizes:  $s_i$
- $n_i$  classes of size  $s_i$
- $\gamma_i = s_i n_i$  elements in a class of size  $s_i$
- Order by  $\gamma_i$  (non-decreasing)
- Note # class sizes  $k \leq (2n)^{1/2}$

# The Structure

---

We store 2 sequences ( $i=1, k$ )

**S:**  $\delta_i = \gamma_i - \gamma_{i-1} = s_i n_i - s_{i-1} n_{i-1} \quad (s_0 n_0 = 0)$

**M:**  $n_i$

And **Shadows** – bit vector 1 indicates start of new term (play a rank/select game on these)

Also store  $\sum_{j=1}^i s_j n_j$  “occasionally”

# Finding a Class

---

Store  $\sum_{j=1}^i s_j n_j$  for every  $\lg n^{\text{th}}$   $i$  value

So, given  $x$ , we find its class

- Binary search to find the right  **$\lg n$ -block**
- Sequential search in  $\lg n$ -block to find right class size
- Compute class in size group

# Speeding Up Search

---

For  $\Theta(n^{1/2} \lg n / \lg \lg n)$  bits

And  $\Theta(\lg \lg n)$  time

- Store sequences  $\lg \lg n$  apart
- Use y-fast trees



# Getting to Constant Time

---

But  $O(n^{1/2} \lg n)$  bits

More work and more details.

Key point is computing (integer) square roots in constant time, by table lookup and  $O(1)$  extra work

(Curiously avoiding the  $\lg \lg n$  iterations of Newton...also used by Rajeev Raman)

# Updates?

---

Can support unions .. OK, simplicity

Space:  $O(n^{1/2} \lg n)$  bits

Worst case merge:  $O(\lg n / \lg \lg n)$  time

Amortized query time:  $O(\alpha(n))$

How?

Multiple copies, old, updating, new

“Time sharing” etc

# Space-Name Space Tradeoff

---

- For “ideal name space” we have  $\Theta(n^{1/2})$  bits ... and that’s as good as it gets
- For knowing only  $n$ , we need  $n \ln n$  name space
- Are there tradeoffs?

# Space-Name Space Tradeoff

---

- For “ideal name space” we have  $\Theta(n^{1/2})$  bits ... and that’s as good as it gets
- For knowing only  $n$ , we need  $n \ln n$  name space
- Suppose we round each equivalence class size up to **power of 2** in “ideal case”
  - Name space  $2n$
  - Keep track of number of each size
  - Leads to constant time and  $\Theta(\lg^2 n)$  space

# Space-Name Space Tradeoff

---

- For “ideal name space” we have  $\Theta(n^{1/2})$  bits ... and that’s as good as it gets
- For knowing only  $n$ , we need  $n \ln n$  name space
- Name space  $2n$ , Suppose we raise each equivalence class size up to **power of 2** in “ideal case”,  $\Theta(\lg^2 n)$  space
- Raise # classes of “same size” to **power of 2** ...  $\Theta(\lg n \lg \lg n)$  space

# Do We Have to Know $n$ ?

---

- “No memory” method assumes we know  $n$
- But that is  $\lg n$  bits (or so)
- A few more improvements; e.g. label space  $(1 + \epsilon)n$  and no extra space.
- Suppose we raise  $n$  to next power of 2, need only  $\lg \lg n$  bits (or so); just doubles name space, still  $\Theta(n \lg n)$

# Overall Conclusion

---

Interesting & useful, combinatorial objects can be:  
Stored succinctly ...  $O(\text{lower bound}) + o()$  so  
Natural queries are performed in  $O(1)$  time (or at  
least very close)

**Packages:** <http://pizzachili.dcc.uchile.cl/index.html>

Ferragina & Navarro;

(also package (RoSA) of Simon Gog)

This can make the difference between  
using the **data type** and not ...



# Thanks

