Succinct Data Structures

Ian Munro

University of Waterloo

Joint work with David Benoit, Andrej Brodnik, S. Srinivasa Rao, Rajeev Raman, Venkatesh Raman, Adam Storm et al

How do we encode a large tree or other combinatorial object of specialized information

… even a static one

in a small amount of space

and still perform queries in constant time ???

Example of a Succinct Data Structure: The (Static) Bounded Subset

Given: Universe of n elements $[0,\ldots,n-1]$

and m arbitrary elements from this universe

Create: a static structure to support search in constant time (lg n bit word and usual operations)

Using: Essentially minimum possible # bits ... $\lg\left(\binom{n}{m}\right)$

Operation: Member query in $O(1)$ time

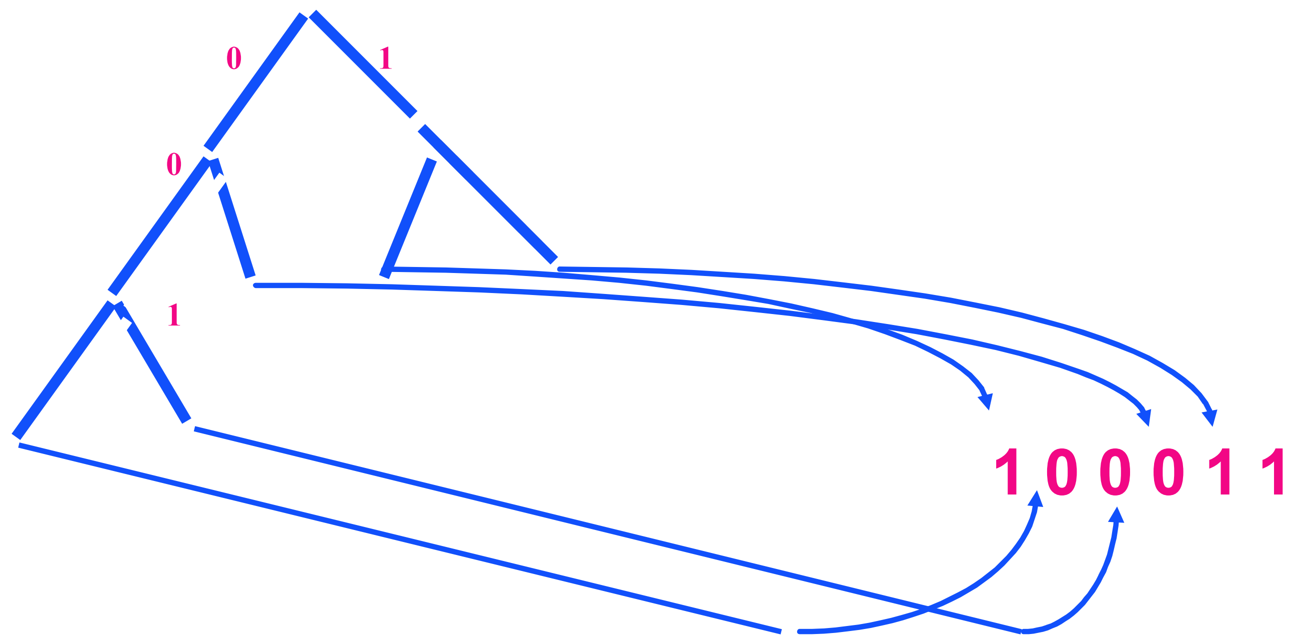
(Brodnik & M.)

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

A Big Patricia Trie / Suffix Trie



- 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 of character boundaries)
- Skip the nodes where there is no branching (so n-1 internal nodes)

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 $6 n \lg n$ 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:

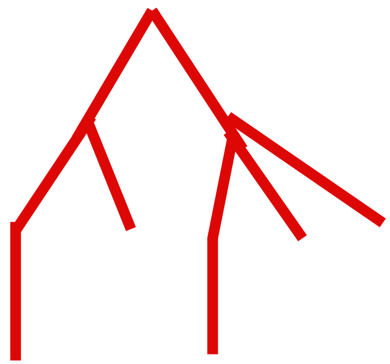
There are about $4^n/(\pi n)^{3/2}$ ordered rooted trees, and same number of binary trees

Lower bound on specifying is about $2n$ bits

What are the natural representations?

Arbitrary Ordered Trees

- Use parenthesis notation
- Represent the tree

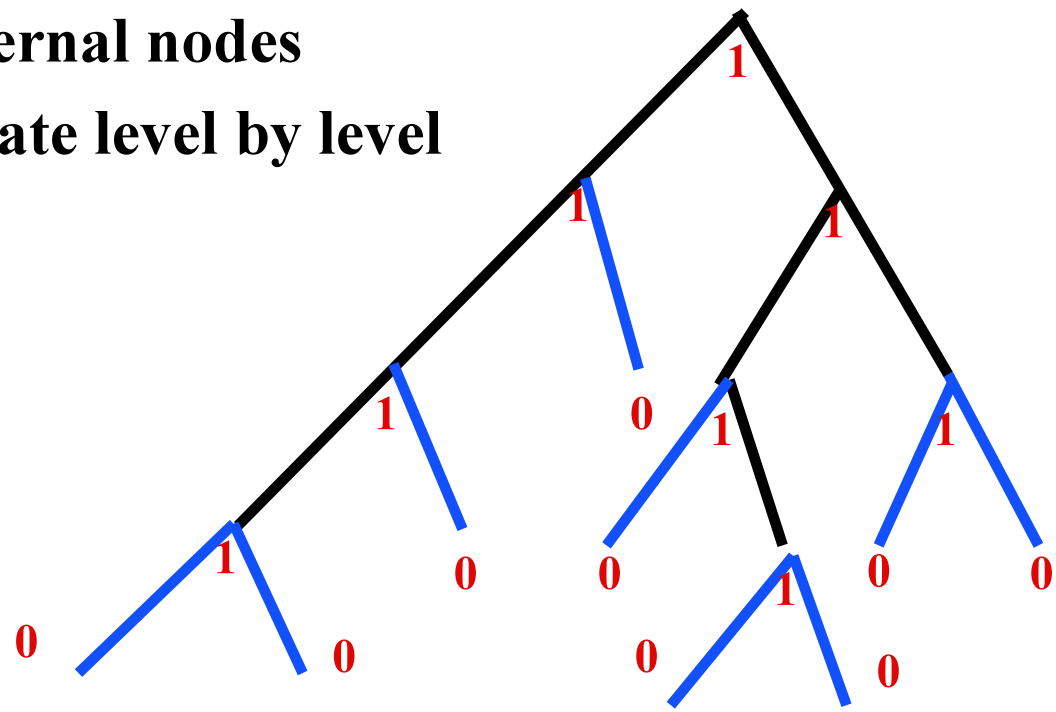


- As the binary string (((())())((())()())): traverse tree as "(" for node, then subtrees, then ")"
- Each node takes 2 bits

Heap-like Notation for a Binary Tree

Add external nodes

Enumerate level by level



Store vector 1111011110010000000 of length 2n+1

(Here 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($\lfloor x/2 \rfloor$)

Rank & Select

Rank - Auxiliary storage ~ 2 n lg lg n / lg n bits

#1's up to each (lg n)^{2} rd bit

#1's within these too each lg nth bit

Table lookup after that

Select - a bit more complicated but similar notions

Key issue: Rank & Select take O(1) time with lg n bit word (M. et al)

Aside: Interesting data type by itself

Other Combinatorial Objects

Planar Graphs (Lu et al)

Permutations [n] $\rightarrow$ [n]

Or more generally

Functions [n] $\rightarrow$ [n]

But what are the operations?

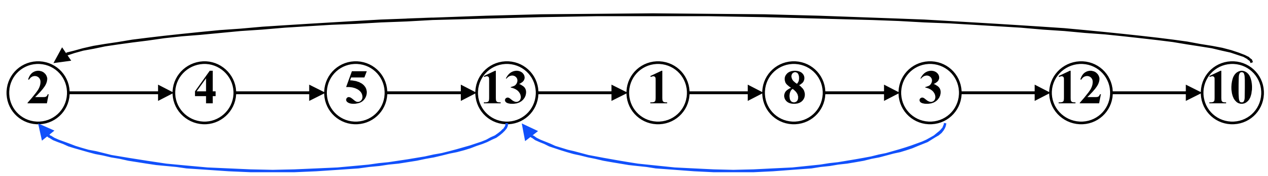
Clearly $\pi(i)$, but also $\pi^{-1}(i)$

And then $\pi^{k}(i)$ and $\pi^{-k}(i)$

Permutations: a Shortcut Notation

Let **P** be a simple array giving π; $P[i] = \pi[i]$

Also have $B[i]$ be a pointer t positions **back** in (the cycle of) the permutation; $B[i] = \pi^{-t}[i]$ .. But only define **B** for every t^{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 indicates no B

1 indicates a B

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

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

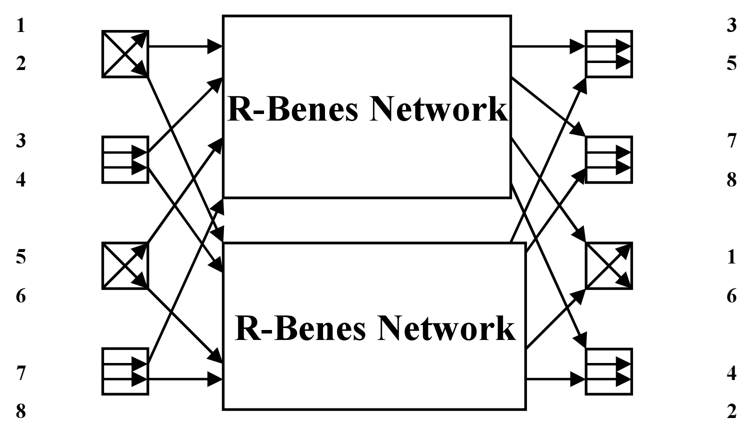
Getting n lg n Bits: an Aside

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

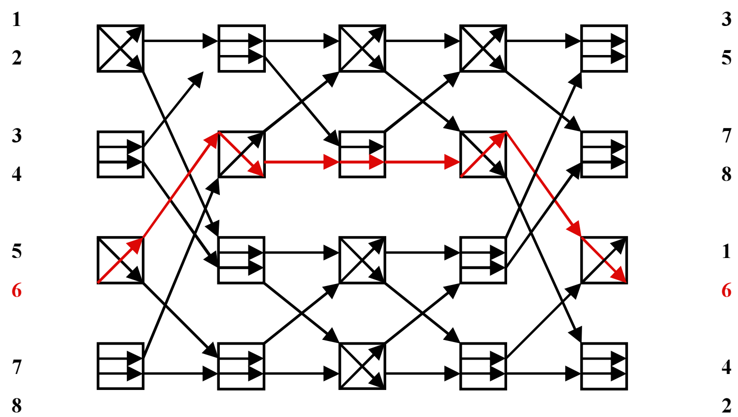
r+1-Benes network … join tops to tops



A Benes Network

Realizing the permutation

(3 5 7 8 1 6 4 2)



What can we do with it?

Divide into blocks of lg lg n gates … and encode their actions in a word .. Taking advantage of the regularity of the address mechanism

and

Also modify the approach to avoid power of 2 issue

So we can trace across a path in time O(lg n/(lg lg n)

This is the best time we can get for π and π^{-1} in minimum space

Back to the main track: Powers of π

Consider the cycles of π

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

Keep a bit vector to indicate the start of each cycle

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

Ignoring parentheses, view as new permutation, ψ.

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

So we have ψ and ψ^{-1} as before

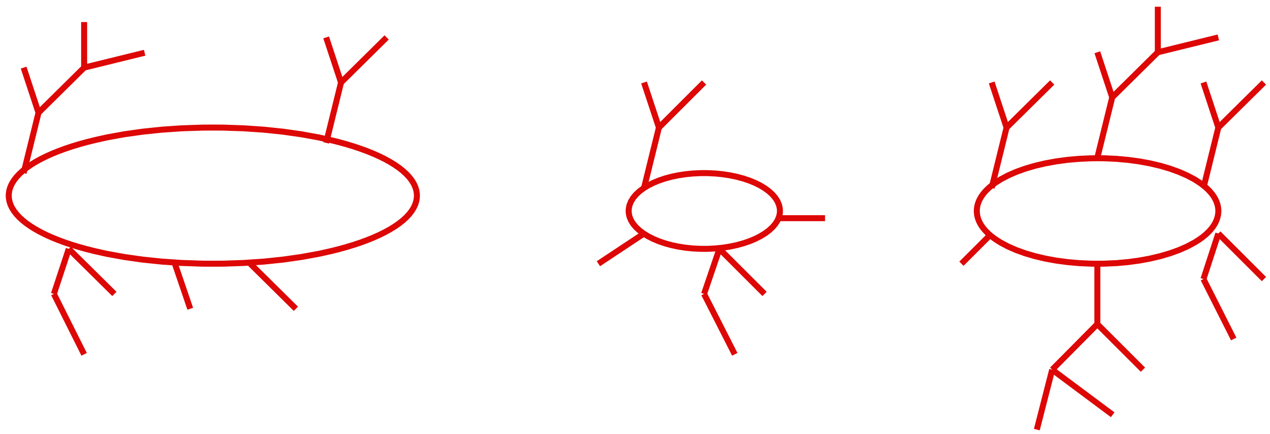
Use $\psi^{-1}(i)$ to find i, then bit vector (rank, select) to find π^{k} or π^{-k}

Functions

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

"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 …

f^{-k} is a set

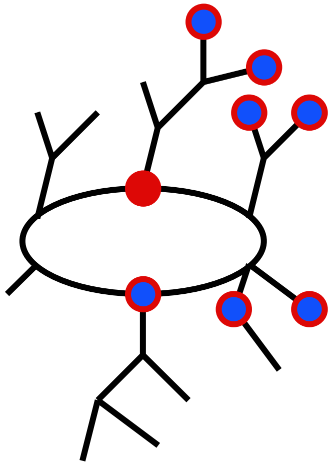
Moving Down the tree requires care

f^{-3}(●) = (●)

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 $O(1 +$ size of answer$)$.

General Conclusion

Interesting, and useful, combinatorial objects can be:

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

So that

Natural queries are performed in O(1) time

This can make the difference between using them and not …