6.851	Lecture 17	Apr. 24, 2012

TODAY: Succinct data structures I (of 2)
- survey
- succinct binary tries
  - level-order
  - via balanced parentheses
- succinct rank & select

Goal: small space, often static

Implicit DS: space = $\underbrace{\text{OPT}}_{\text{information theoretic}}$ + $\underbrace{O(1) \text{ bits}}_{\text{for rounding}}$
- typically, DS is "just the data", permuted in some order
- e.g. sorted array, heap

Succinct DS: space = OPT + o(OPT)
- lead constant of 1

Compact DS: space = O(OPT)
- often a factor of w smaller than "linear-space" data structures
  e.g. suffix trees use O(n) words for n-bit string

Minisurvey:

- implicit dynamic search tree:
  [Franceschini & Grossi – ICALP 2003/WADS 2003]
  $O(\lg n)$ worst-case time/insert, delete, predecessor
  also $O(\log_B N)$ cache oblivious
- succinct dictionary: [Brodnik & Munro – SICOMP 1999; Pagh – SICOMP 2001]
  $n \lg \frac{u}{n} = \lg \binom{u}{n} + O\left(n \frac{(\lg \lg n)^2}{\lg n}\right)$ bits
  $O(1)$ membership query (static)

TODAY * succinct binary trie: [Munro & Raman – SICOMP 2001]
  $C_n = \binom{2n}{n} / (n+1) \sim 4^n$ such tries (Catalan)
  $\lg C_n + o(\lg C_n) = 2n + o(n)$ bits
  $O(1)$ left child, right child, parent, subtree size
  – $O(1)$ ins./del. leaf, subdivide edge [Farzan & Munro – TCS 2011]
- succinct k-ary trie: (e.g. suffix tree) [Farzan & Munro – SWAT 2008]
  $C_n^k = \binom{kn+1}{n} / (kn+1)$ tries, $\lg C_n^k + o$ bits
  $O(1)$ child with label i, parent, subtree size, ...
  improving [Benoit, Demaine, Munro, Raman, Raman, Rao – Algorithmica 2005]
- succinct permutations: [Munro, Raman, Raman, Rao – ICALP 2003]
  $\lg n! + o(n)$ bits, $O\left(\frac{\lg n}{\lg \lg n}\right)$ time to compute $\pi^k(x)$ $\forall k$
  OPEN $(1+\varepsilon) n \lg n$ bits, $O(1)$ time π^k (including $k<0$)
  generalizes to functions [Munro & Rao – ICALP 2004]
- compact Abelian groups: [Farzan & Munro – ISSAC 2006]
  $O(\lg n)$ bits for group of order n (!) or elt. in group
  $O(1)$ multiply, inverse, equality testing
- graphs [Farzan & Munro – ESA 2008; Barbay, Aleardi, He, Munro – Alg. 2012]
- implicit n-bit ints: inc./dec. in $O(\lg n)$ bit reads & $O(1)$ bit writes [Rahman & Munro – Alg. 2010]
  OPEN: $O(1)$ word RAM?

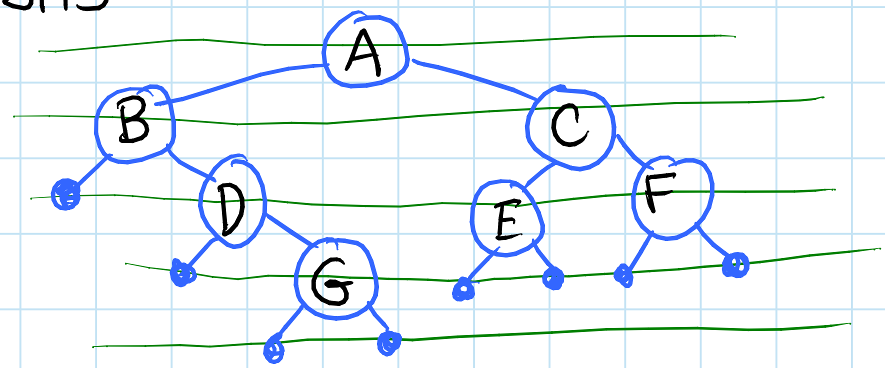
Level-order representation of binary tries: [Munro]

for each node in level order:

- write 0/1 for whether have left child
- write 0/1 for whether have right child

$\Rightarrow$ $2n$ bits

e.g:



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(1)	1	1	0	1	1	1	0	1	0	0	0	0	0	0
A	B	C	•	D	E	F	•	G	•	•	•	•	•	•

Equivalently:

- append external node (•) for each missing child
- for each node in level order:

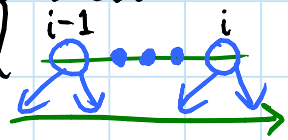
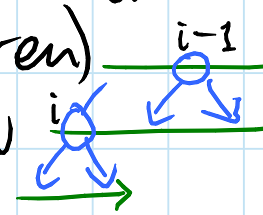
  write 0 if external, 1 if internal

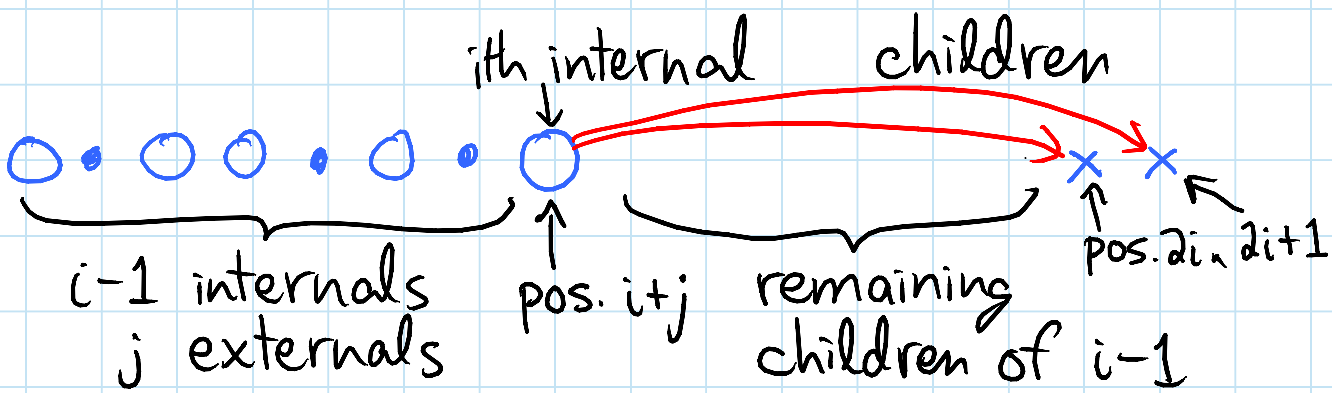
$\Rightarrow$ extra leading 1 ($2n+1$ bits)

Navigation: (in external-node view)

left & right children of ith internal node are at positions $2i$ & $2i+1$

Proof: by induction on i:

- just after $(i-1)$st internal node's children (as external nodes have no children)
- either same level  or new 



$$= 2(i-1) - (i-1) - j$$

all   internal seen   root   external seen

$$= i - j - 1$$

Rank & Select in bit string:

$\text{rank}_1(i)$ = # 1's at or before position i

$\text{select}_1(j)$ = position of jth 1 bit

$\Rightarrow$ $\text{left-child}(i) = 2\,\text{rank}_1(i)$

$\text{right-child}(i) = 2\,\text{rank}_1(i) + 1$

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

(but subtree-size impossible in level-order rep)

Rank: [Jacobsen – FOCS 1989]

① use lookup table for bitstrings of length $\frac{1}{2}\lg n$

$\Rightarrow O(\underbrace{\sqrt{n}}_{\text{bitstring}}\ \underbrace{\lg n}_{\text{query } i}\ \underbrace{\lg\lg n}_{\text{answer}})$ bits of space

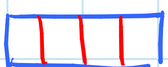
② split into $(\lg^2 n)$-bit chunks:

$\lg^2 n$ ↑ store cumulative rank: $\lg n$ bits

$\Rightarrow O(\frac{n}{\lg^2 n}\lg n) = O(\frac{n}{\lg n})$ bits

(couldn't afford $\lg n$-bit chunks)

③ split each chunk into $(\frac{1}{2}\lg n)$-bit subchunks:



$\frac{1}{2}\lg n$ ↑ store cumulative rank within chunk: $\lg\lg n$ bits

$\Rightarrow O(\frac{n}{\lg n}\lg\lg n) = o(n)$ bits

④ rank = rank of chunk

+ relative rank of subchunk within chunk

+ relative rank of element within subchunk

(via lookup table)

$\Rightarrow O(1)$ time, $O(n\frac{\lg\lg n}{\lg n})$ bits

– $O(n/\lg^k n)$ bits possible for any $k = O(1)$

[Pătraşcu – FOCS 2008]

– $O(\frac{\lg n}{\lg\lg n})$ insert/delete/rank/select

[He & Munro – SPIRE 2010]

Select: [Clark & Munro – Clark's PhD 1996]

① store array of indices of every $(\lg n \lg\lg n)$th 1 bit
$\Rightarrow O\left(\frac{n}{\lg n \lg\lg n} \lg n\right) = O\left(\frac{n}{\lg\lg n}\right)$ bits

② within group of $\lg n \lg\lg n$ 1 bits, say r bits:
if $r \geq (\lg n \lg\lg n)^2$
then store array of indices of 1 bits in group
$$\Rightarrow O\Big(\underbrace{\frac{n}{(\lg n \lg\lg n)^2}}_{\#\text{ such groups}} \underbrace{(\lg n \lg\lg n)}_{\#\text{ 1 bits}} \underbrace{\lg n}_{\text{index}}\Big) = O\left(\frac{n}{\lg\lg n}\right) \text{ bits}$$
else reduced to bitstring of length $r \leq (\lg n \lg\lg n)^2$

③ repeat ①&② on all reduced bitstrings
to reduce to bitstrings of length $(\lg\lg n)^{O(1)}$

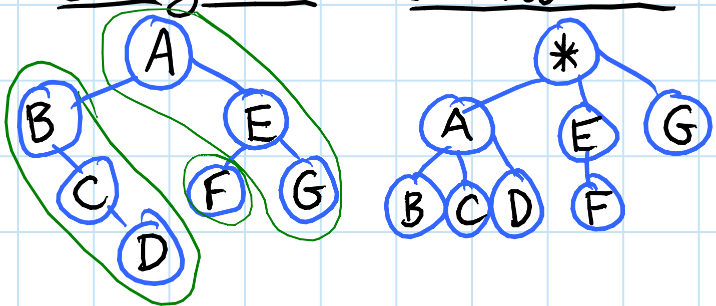
①′ store relative index ($\lg\lg n$ bits) of every
$(\lg\lg n)^2$th 1 bit ($\lg\lg n \lg\lg\lg n$ also OK but bigger)
$\Rightarrow O\left(\frac{n}{(\lg\lg n)^2} \lg\lg n\right) = O\left(\frac{n}{\lg\lg n}\right)$ bits

②′ within group of $(\lg\lg n)^2$ 1 bits, say r bits:
if $r \geq (\lg\lg n)^4$
then store relative indices of 1 bits
$$\Rightarrow O\Big(\underbrace{\frac{n}{(\lg\lg n)^4}}_{\#\text{ such groups}} \underbrace{(\lg\lg n)^2}_{\#\text{ 1 bits}} \underbrace{\lg\lg n}_{\text{rel. index}}\Big) = O\left(\frac{n}{\lg\lg n}\right) \text{ bits}$$
else reduced to bitstring of length $r \leq (\lg\lg n)^4$

④ use lookup table for bitstrings of length $\leq \frac{1}{2}\lg n$
$$\Rightarrow O\Big(\underbrace{\sqrt{n}}_{\#\text{bitstrings}} \underbrace{\lg n}_{\text{query } j} \underbrace{\lg\lg n}_{\text{answer}}\Big)$$

$\Rightarrow O(1)$ query, $O\left(\frac{n}{\lg\lg n}\right)$ bits
– $O(n / \lg^k n)$ bits $\forall k = O(1)$ [Pătraşcu – FOCS 2008]

Binary tries as balanced parentheses: [Munro & Raman – SICOMP 2001]

binary trie	rooted ordered tree	balanced parens (= bitstring)
		((( ) ( ) ( )) (( )) ( )) *ABBCCDDAEFFEGG*
node	node	left paren. [& matching right]
left child	first child	next char. [if (, else none]
right child	next sibling	char. after matching) [if (]
parent	prev. sibling OR parent	prev. char.) ⇒ its matching (prev. char. (⇒ that (
subtree size	size(node) + sizes(right siblings)	½ distance to enclosing)
# leaves in subtree		$\text{rank}_{()}$ of enclosing) − $\text{rank}_{()}$ of here

- similar to (& using) rank & select, can find matching & enclosing parens. in $O(1)$ time, $o(n)$ space

⇒ all operations above in $O(1)$ time

- from subtree size can accumulate index of node for auxiliary data (e.g. pointer to text)