Lecture 21

Succinct data structures: small space, usually static

Implicit DS: space = information-theoretic OPT + $O(1)$ bits (for rounding)
- typically, data structure is "just the data", permuted in some order

Succinct DS: space = information-theoretic OPT + $o(\text{OPT})$
- lead constant of 1

Compact DS: space = O(information-theoretic OPT)
- still better than "linear-space" data structures on e.g. n-bit strings by factor of $\Theta(w)$

Mini survey:

- *implicit* dynamic search tree: [Franceschini & Grossi — ICALP 2003 & WADS 2003]
  $O(\lg n)$ worst case / insert, delete, search
  (cf. heap & sorted array)
  comparison model on elements
  word RAM for pointers
- *succinct* dictionary: [Brodnik & Munro — SICOMP 1999; Pagh — SICOMP 2001]
  $\lg \binom{u}{n} + O\left(n \frac{(\lg \lg n)^2}{\lg n}\right)$ bits
  $O(1)$ membership query

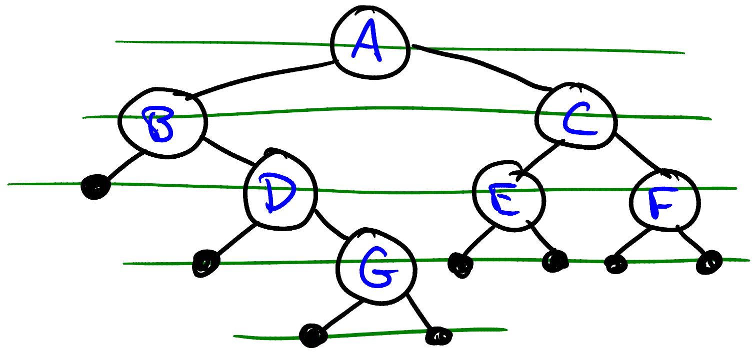
* TODAY *succinct* binary trie: [Munro & Raman — SICOMP 2001]
  $C_n = \binom{2n}{n} / (n+1) \sim 4^n$ such tries
  $\lg C_n + o(\lg C_n) = 2n + o(n)$ bits
  $O(1)$ left child, right child, parent, subtree size
- *compact*/almost succinct k-ary trie: [Benoit, Demaine, Munro, Raman, Raman, Rao — Algorithmica 2005]
  $C_n^k = \binom{kn+1}{n} / (kn+1) \sim 2^{(\lg k + \lg e)n}$ such tries
  $(\lceil \lg k \rceil + \lceil \lg e \rceil) n + o(n)$ bits
  $O(1)$ child with label i, parent, subtree size
- *succinct* rooted ordered tree [Benoit et al. 2005]
  only C_n such trees
  $2n + o(n)$ bits
  $O(1)$ ith child, parent, subtree size

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

→ 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	•	•	•	•	•	•

More useful view:

1. append external node (•) for each missing child
2. visit all nodes in level order
3. write 1 for internal node, 0 for external

→ extra leading 1 (2n+1 bits)

Navigating level-order representation: (with external nodes)

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

Proof:

- suppose ith internal node at position $i+j$
- i.e. j external nodes up to ith external node
- $i-1$ previous internal nodes have $2(i-1)$ children
- $i-1$ already seen as internal nodes (all but root)
- j already seen as external nodes

$\Rightarrow$ left child at position $(i+j)+\underbrace{2(i-1)-(i-1)-j}_{\text{intervening children}}+1 = 2i$. $\square$

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

$$\Rightarrow \begin{aligned} \text{left-child}(i) &= 2\,\text{rank}_1(i) \\ \text{right-child}(i) &= 2\,\text{rank}_1(i)+1 \\ \text{parent}(i) &= \text{select}(\lfloor i/2 \rfloor) \end{aligned}$$

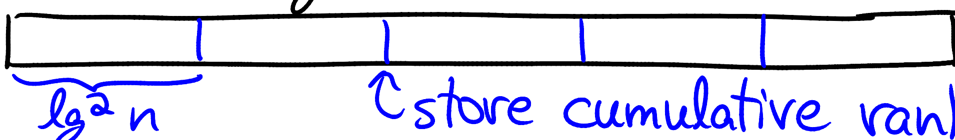
(but no subtree size in level-order rep.)

Rank: [Jacobsen – FOCS 1989]

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

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

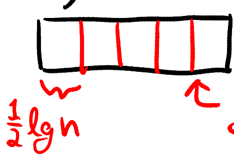
② split into $n/\lg^2 n$ chunks:



store cumulative rank, $\lg n$ bits

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

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



store cumulative rank within chunk

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

④ rank query sums rank of chunk

\+ relative rank of subchunk

\+ relative rank of element within subchunk (lookup table)

$\Rightarrow O(1)$ time, $o(n)$ bits of space

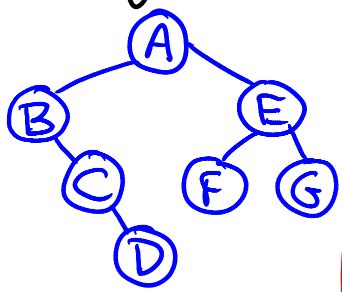
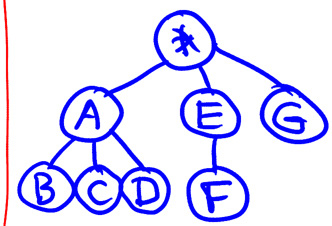
Note: can't afford $\Theta(\frac{n}{\lg n})$ chunks with $(\lg n)$-bit cumulative ranks

Select: [Clark & Munro — Clark's Ph.D. 1996]

1. store index 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
   $\Rightarrow O\left(\underbrace{\frac{n}{(\lg n \lg\lg n)^2}}_{\#\text{such groups}} \underbrace{(\lg n \lg\lg n)}_{\#\text{1 bits}} \underbrace{\lg n}_{\text{index size}}\right) = O\left(\frac{n}{\lg\lg n}\right)$ bits
   #such groups / #1 bits / index size
   else reduced to bitstring of length $r \leq (\lg n \lg\lg n)^2$
3. repeat steps 1 & 2 on all reduced bitstrings to reduce to bitstrings of length $O(\lg\lg n \lg\lg\lg n)$:
   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 array of indices of 1 bits in group
   $\Rightarrow O\left(\underbrace{\frac{n}{(\lg\lg n)^4}}_{\#\text{such groups}} \underbrace{(\lg\lg n)^2}_{\#\text{1 bits}} \underbrace{\lg\lg n}_{\text{index size}}\right) = O\left(\frac{n}{\lg\lg n}\right)$ bits
   #such groups / #1 bits / index size
   else reduced to bitstring length $r \leq (\lg\lg n)^4$
4. use lookup table for bitstrings of length $\leq \frac{1}{2}\lg n$
   $\Rightarrow O(\underbrace{\sqrt{n}}_{\#\text{bitstrings}} \underbrace{\lg n}_{\text{query } j} \underbrace{\lg\lg n}_{\text{answer}})$ bits
   #bitstrings / query j / answer

$\Rightarrow O(1)$ query, $o(n)$ bits of space

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

binary trie	rooted ordered tree	balanced parens (=bitstring)
		((( ) ( ) ( )) (( )) ( )) * A B B C C D D A E F F E G G *
node	node	left paren. ([& matching)]
left child	first child	next char. [if (, else none]
right child	next sibling	char. after matching) [if (]
parent	prev. sibling or parent	prev. char.) ⇒ matching ((⇒ that (
subtree size	size(node) + sizes(right siblings)	$\frac{1}{2}\cdot$distance to enclosing)

- like (& using) rank & select, can find matching/enclosing parens in $O(1)$ time with $o(n)$ space

⇒ all operations above in $O(1)$ time

- from subtree size can accumulate index of node