Plán semestra

Dnes: *úsporné dátové štruktúry, úvod do editačnej vzdialenosti*

Zajtra: *editačná vzdialenosť, najdlhšia spoločná podpostupnosť*

Do pondelka: výber článku na prezentáciu

Streda 17.4.: *zlepšenia výpočtu editačnej vzdialenosti*

Štvrtok 18.4.: prednáška nebude

Streda 24.4.: *približné výskyty vzorky, lokálne podobnosti, bioinformatika*

Štvrtok 25.4.: *zostavovanie DNA sekvencií, najkratšie spoločné nadslovo*

Streda 1.5.: sviatok

Štvrtok 2.5.: *viacnásobné zarovnanie, opakujúce sa sekvenčné motívy*

Streda 8.5.: sviatok

Štvrtok 9.5.: prezentácie

Streda 15.5.: prezentácie

Štvrtok 16.5.: prezentácie

Succint data structures

- Data structure uses $\mathrm{OPT} + \mathrm{o}(\mathrm{OPT})$ bits of memory
  and supports fast operations
- Rank and select on a binary vector of length n in $\mathrm{O}(1)$ time
- Wavelet tree supports rank over larger alphabet in $\mathrm{O}(\log \sigma)$ time,
  uses binary rank
- FM index counts occurrences of pattern P in T in $\mathrm{O}(\mathrm{m} \log \sigma)$ time,
  uses BWT and wavelet tree

Today:

- Compressed data structures (for rank)
- Succint data structure for binary trees

Succint structure for rank and select

Bit vector A[0..n-1]

rank(i) = number of bits set to 1 in A[0..i]

select(i) = position of the i-th bit set to 1

Example:

i	0	1	2	3	4	5	6	7
$A[i]$	0	1	1	0	1	0	0	1

rank(3) = 2, rank(4) = 3

select(1) = 1, select(3) = 4

Goal:

rank, select in $O(1)$ time

structure needs $n + o(n)$ memory

we have concentrated on rank

Succint structure for rank

- Divide bit vector to superblocks of size $t_1 = \log_2^2 n$
- Divide each super block to blocks of size $t_2 = \frac{1}{2} \log_2 n$
- Keep rank at each super block boundary

  $O(\frac{n}{t_1} \cdot \log n) = O(n/\log n) = o(n)$ bits
- Keep rank within super block at each block boundary

  $O(\frac{n}{t_2} \cdot \log t_1) = O(n \log\log n/\log n) = o(n)$ bits
- Each block stored as a binary number using t_2 bits

  n bits
- For each of 2^{t_2} possible blocks and each query keep the answer

  $O(2^{t_2} \cdot t_2 \cdot \log t_2) = O(\sqrt{n} \log n \log\log n) = o(n)$ bits

Compressed structure for rank (Raman, Raman, Rao 2002)

- Compressed size of bit vector + $o(n)$ bits
- Need to reduce the following part:

  Each block stored as a binary number using t_2 bits
- Blocks with many 0s or many 1s stored using fewer bits
- For each block store the number of 1s

  $O(\frac{n}{t_2} \log t_2) = O(n \log\log n / \log n) = o(n)$ bits
- For a block with x 1s store its index in lexicographic order

  of all binary strings of size t_2 with x 1s

  $\lceil \log_2 \binom{t_2}{x} \rceil \leq \log_2 2^{t_2} = t_2$ bits

  overall at most $\frac{n}{t_2} t_2 = n$ bits
- Also rearrange the table with answers for all possible blocks of size t_2

Compressed structure for rank (RRR)

$t_2 = 3$

1s	Index	Size	Block	Answers
0	0	0	000	0 0 0
1	0	2	001	0 0 1
	1		010	0 1 1
	2		100	1 1 1
2	0	2	011	0 1 2
	1		101	1 1 2
	2		110	1 2 2
3	0	0	111	1 2 3

Original bits: 000|101|001|111|111

Number of 1s in each block: 00|10|01|11|11

Index of block: $\epsilon|01|00|\epsilon|\epsilon$

Where each block starts: 0000|0000|0010|0100|0100

Analysis of RRR structure

- Let S be a string in which $a \in \Sigma$ occurs n_a times
- Its **entropy** is $H(S) = \sum_a \frac{n_a}{n} \log_2 \frac{n}{n_a}$
- $H(S) \leq \log_2 \sigma$ (lower if some characters more frequent than others)
- RRR structure for bit vector B uses $nH(B) + o(n)$ **bits**

Stirling's approximation of $n!$

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + O(1/n))$$

$$\ln(n!) = n \ln(n) - n + O(\ln(n))$$

Uses of RRR structure

- Store binary rank structures in the wavelet tree for text T
  overall $nH(T) + o(nH(T))$ bits
- Instead of wavelet tree, store indicator vector for each $a \in \Sigma$
  overall $nH(T) + \sigma o(n)$ bits
  $O(1)$ per rank query
- FM index needs rank in BWT of T, which might be even better compressible

Succint binary trees

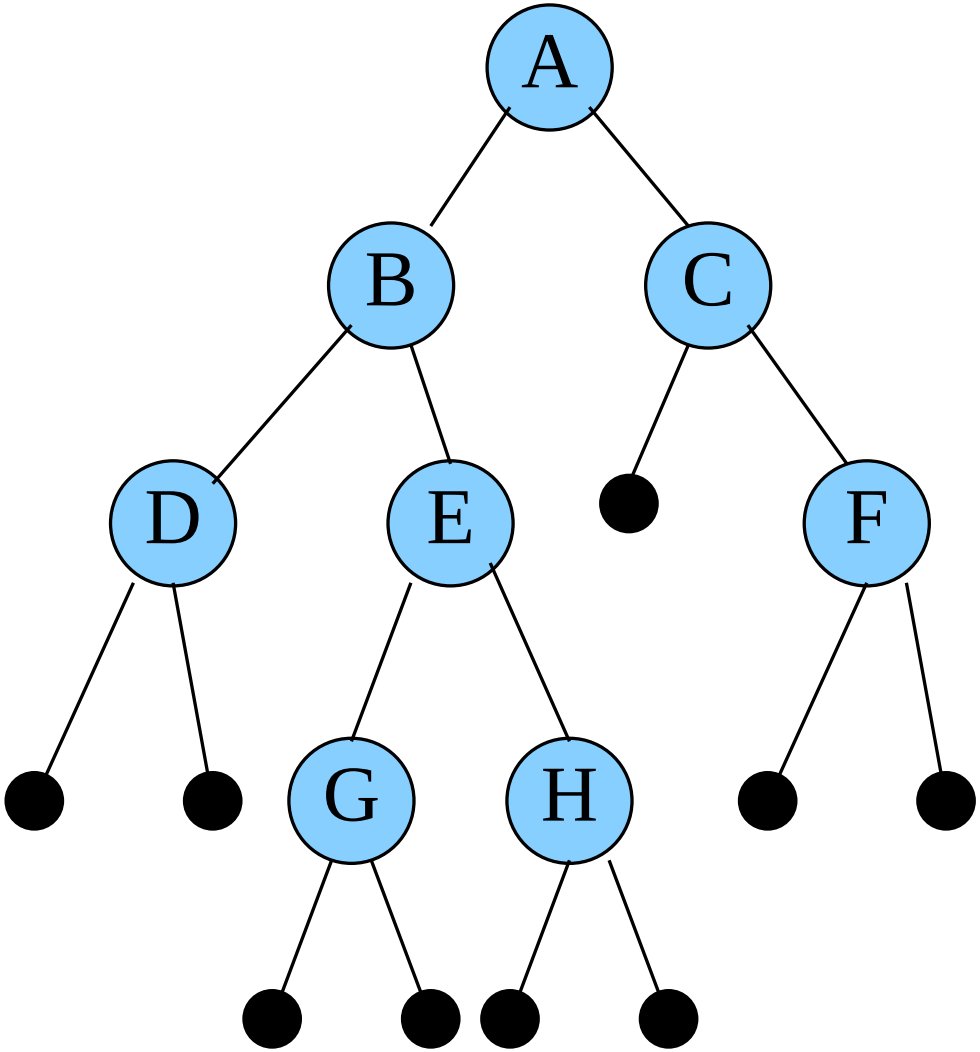
- Consider all binary trees with n nodes
- OPT is cca $2n$ bits (proof later)
- Use $2n + o(n)$ memory
- Support operations left child, right child, parent in $O(1)$
- Nodes are numbers from $\{1, \ldots, 2n + 1\}$

  Using rank can be mapped to $\{1, \ldots, n\}$

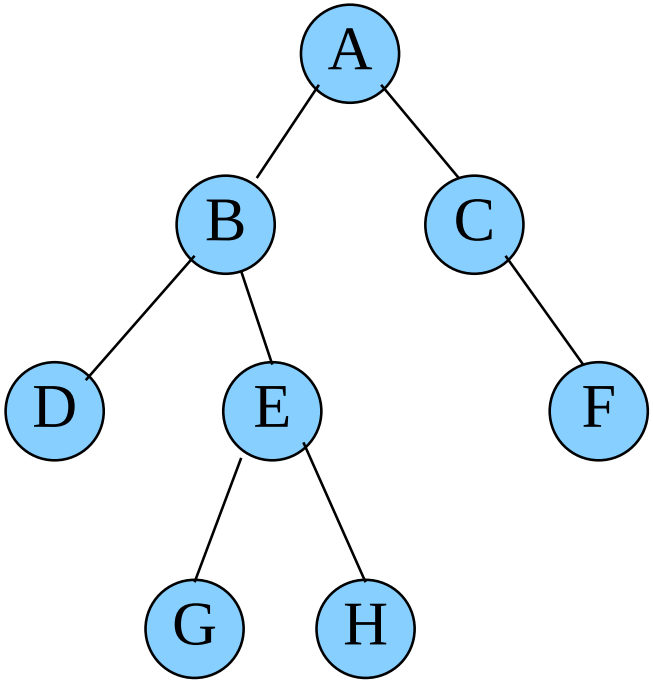
  These can be then used as indices to arrays with additional data
- Can be part of binary tries, binary suffix trees

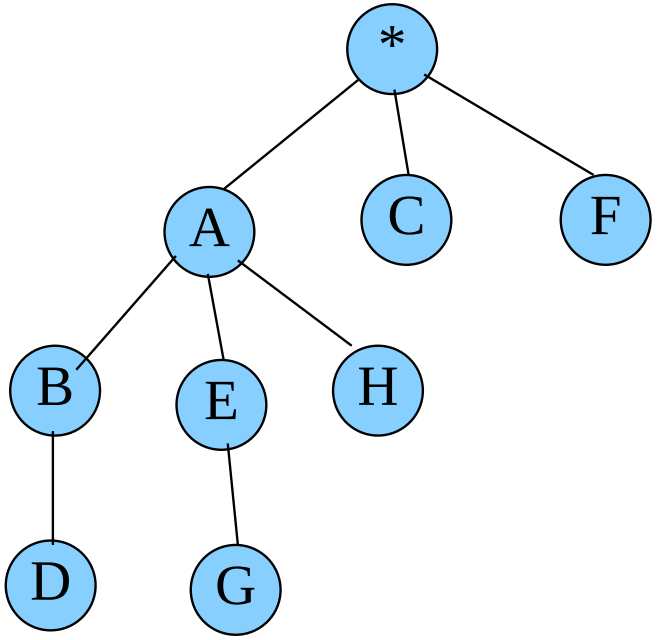
  Extensions to larger alphabets exist
- Static trees only, construction requires more memory
- Classical trees with pointers use $\Omega(n \log n)$ bits

Succint binary trees: level order representation



Equivalence of binary trees and rooted ordered trees





Rooted ordered tree as a well-parenthesized expression:

(((())(()))(()))(()())

ABDDBEGGEHHACCFF

Counting well-parenthesized expresions

Example: $|X(3,3,-\infty) \setminus X(3,3,0)| = |X(2,4,-\infty)| = 15$

$|X(3,3,-\infty)| = 20$

$|X(3,3,0)| = 20 - 15 = 5 = C_3$

First four examples out of 15: