2-INF-237 Vybrané partie z dátových štruktúr

2-INF-237 Selected Topics in Data Structures

- Instructor: Broňa Brejová
- E-mail: brejova@fmph.uniba.sk
- Office: M163
- Course webpage: http://compbio.fmph.uniba.sk/vyuka/vpds/

Succinct data structures

We usually count memory in **words** of some size $w \geq \lg n$

each word can hold pointer, index, count, symbol etc.

Now we will count memory in **bits**

Lower bound: to store any $x \in \mathcal{U}$, we need at least $\mathrm{OPT} = \lg |\mathcal{U}|$ bits

Compact data structure uses $\mathrm{O}(\mathrm{OPT})$ bits

Succinct data structure uses $\mathrm{OPT} + \mathrm{o}(\mathrm{OPT})$ bits

Leading constant 1 plus some lower-order terms

Implicit data structure uses OPT bit plus $\mathrm{O}(1)$ words

Uses ordering of elements in an array

Example: binary heap, sorted array

Succinct structure for binary 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 $n + o(n)$ bits of memory

we will concentrate on rank

Succinct structure for rank (Jacobson 1989)

- Divide bit vector to super blocks of size $t_1 = \lg^2 n$
- Divide each super block to blocks of size $t_2 = \frac{1}{2} \lg n$
- 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

Succinct structure for rank (Jacobson 1989)

R1: array of ranks at superblock boundaries

R2: array of ranks at block boundaries within superblocks

R3: precomputed rank for each block type and each position

B: bit array

```
rank(i) {
    superblock = i/t1;   //integer division
    block = i/t2;
    index = block*t2;
    type = B[index..index+t2-1];
    return R1[superblock]+R2[block]+R3[type, i%t2]
}
```

Succinct structure for select

- Let $t_1 = \lg n \lg\lg n$, $t_2 = (\lg\lg n)^2$.
- Store select($t_1 \cdot i$) for $i = 0, \ldots, n/t1$;

  this divides bit vector into super-blocks of unequal size.
- Large super-blocks of size $\geq t_1^2$: store array of indices of 1 bits.
- Small super-block of size $\leq t_1^2$: repeat with t_1:

  store select($t_2 \cdot i$) within super-block for $i = 0, \ldots, n/t2$;

  this divides small super-blocks into blocks of unequal size.
- Large blocks of size $\geq t_2^2$: store relative indices of all 1 bits.
- Small blocks of size $< t_2^2$: store as t_2^2-bit integer,

  plus a lookup table of all answers.

Succinct data structures

- Data structure uses $\mathrm{OPT} + o(\mathrm{OPT})$ bits of memory

  and supports fast operations

- Rank and select on a binary vector of length n in $O(1)$ time

Next:

- Compressed data structures (for rank)

- Wavelet tree for rank over larger alphabet

- Succinct data structure for binary trees

Entropy and compression

Consider alphabet Σ of size σ, probability of $a \in \Sigma$ is p_a

Entropy of this distribution is: $-\sum_{a \in \Sigma} p_a \lg p_a$

Measure of randomness:

Uniform distribution has entropy $\log_2 \sigma$ (max)

If $p_a = 1$ for some $a \in \Sigma$, then entropy 0 (min)

Lossless **compression** of a text consisting of independent identically distributed random symbols with entropy H, needs roughly H bits per symbol

Goal: use $-\log_2 p_a$ bit to encode a

Huffman encoding close to that but needs rounding

Arithmetic coding avoids rounding

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 (class)

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

  $\lceil \lg \binom{t_2}{x} \rceil \le \lg 2^{t_2} = t_2$ bits (overall at most $\frac{n}{t_2} t_2 = n$ bits)
- Rearrange the table with answers for all possible blocks of size t_2

  Add signature boundaries in compressed bit vector $o(n)$

Compressed structure for rank (RRR)

$t_2 = 3$

Cl	Sig	Length	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 (within superblock): 0000|0000|0010|0100|0100

Compressed structure for rank (RRR)

rank(i):

- superblock = i/t1 (integer division)
- block = i/t2
- index = S1[superblock]+S2[block]
- class = C[block]
- length = L[class]
- signature = B[index..index+length-1]
- return R1[superblock]+R2[block]+R3[class, signature, i%t2]

Analysis of RRR structure

- Let S be a string in which $a \in \Sigma$ occurs n_a times
- Its **entropy** is $H(S) = \sum_a \frac{n_a}{n} \lg \frac{n}{n_a}$
- RRR structure for bit vector B uses $nH(B) + o(n)$ **bits**

Stirling's approximation of $n!$

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

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

Wavelet tree (Grossi, Gupta, Vitter 2003)

$$\Sigma_0 = \{\$, ., a\} \quad \Sigma_1 = \{e, m, u\}$$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S[i]	e	m	a	.	m	a	.	m	a	m	u	.	m	a	m	a	.	m	a	.	e	m	u	$
B[i]	1	1	0	0	1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	0	1	1	1	0

S0 a.a.a.aa.a.$ S1 emmmmummmemu

$\Sigma_{00} = \{\$\}, \Sigma_{01} = \{., a\}, \quad \Sigma_{010} = \{.\}, \Sigma_{011} = \{a\}$
$\Sigma_{10} = \{e\}, \Sigma_{11} = \{m, u\}, \quad \Sigma_{110} = \{m\}, \Sigma_{111} = \{u\}$

i	0	1	2	3	4	5	6	7	8	9	10	11
S0[i]	a	.	a	.	a	.	a	a	.	a	.	$
B0[i]	1	1	1	1	1	1	1	1	1	1	1	0

S00 a.a.a.aa.a. S01 $

Store

B[i] =110010010110101001001110

B0[i] = 111111111110 B1[i] = 011111111011

B01[i] = 10101011010 B11[i] = 000100001

Combination of RRR structure and wavelet trees

- 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 $nH(T) + O(n) + o(\sigma n)$ bits
  $O(1)$ per rank query

Dynamic texts (Navarro, Nekrich 2014)

Access, rank, select

Insert/delete character

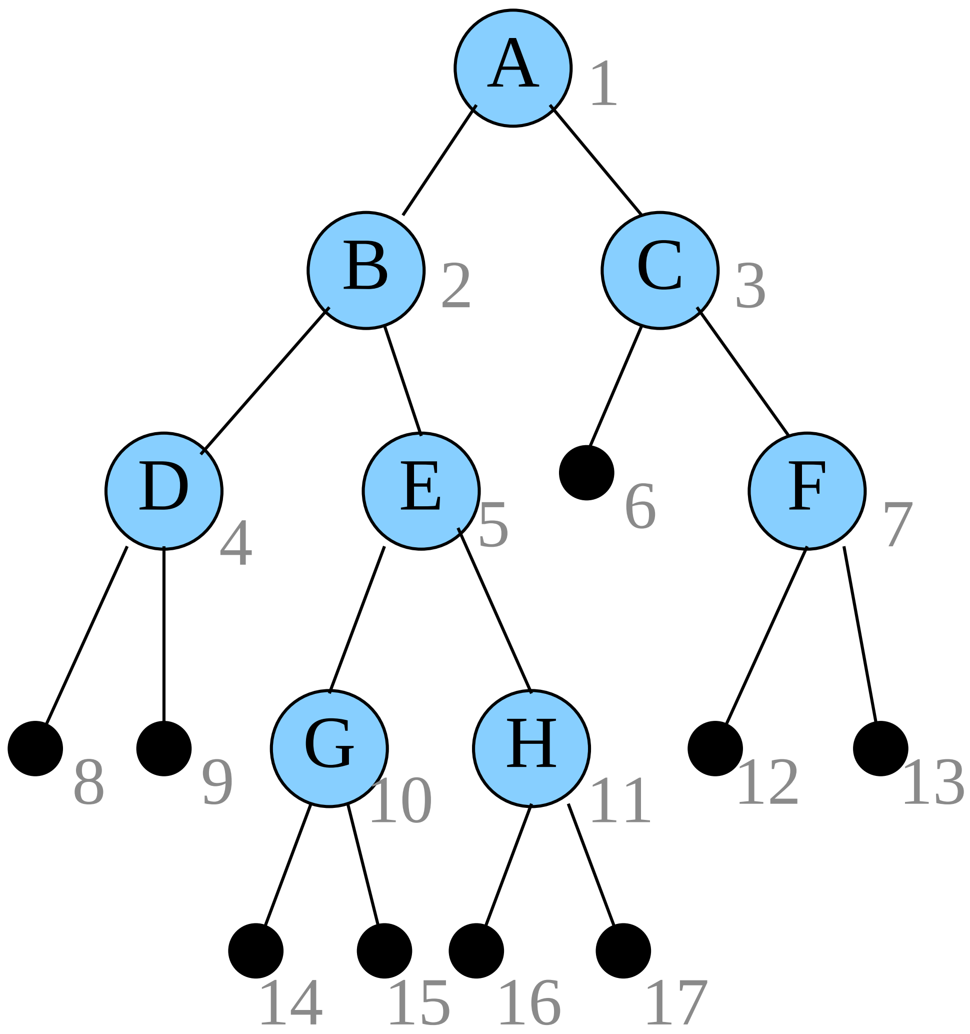
All in $O(\log n / \log \log n)$ amortized

Additional memory $o(n \lg \sigma) + O(\sigma \lg n)$

Succinct binary trees

- Consider all binary trees with n nodes
- Classical trees with pointers use $\Omega(n \log n)$ bits
- OPT is cca $2n$ bits (proof later)
- **Goal:** Use $2n + o(n)$ memory,
  support operations left child, right child, parent in $O(1)$
- Add $n + 1$ auxiliary leaves
- Nodes are numbers $\{1, \ldots, 2n + 1\}$ in level order (BFS)

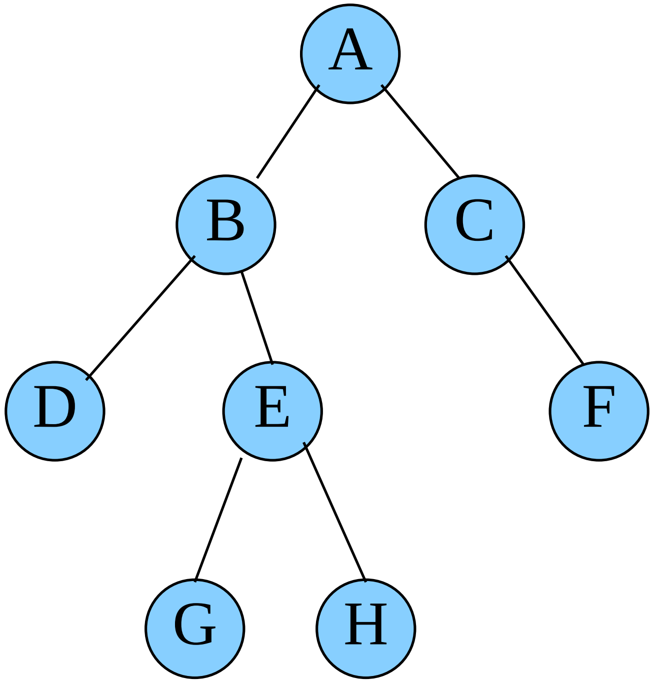
Succinct binary trees: level order representation

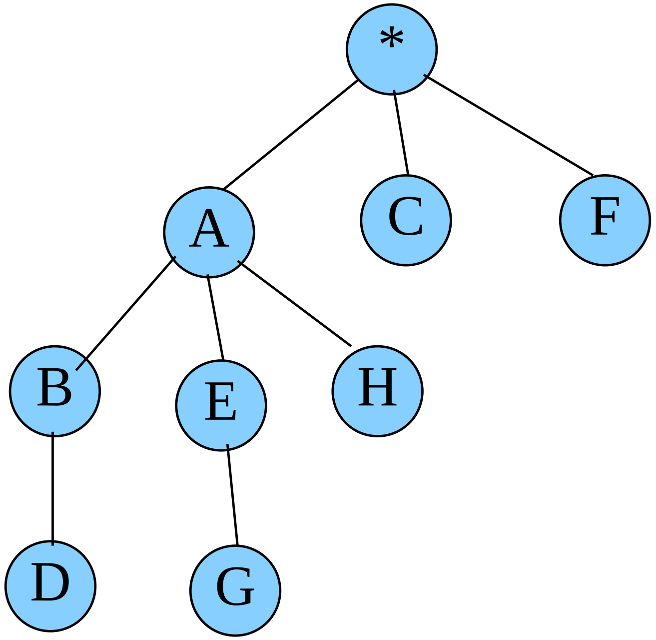


Succinct binary trees

- Consider all binary trees with n nodes
- **Goal:** Use $2n + o(n)$ memory,
  support operations left child, right child, parent in $O(1)$
- Add $n + 1$ auxiliary leaves
- 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
- Static trees only, construction requires more memory

Equivalence of binary trees and rooted ordered trees





Rooted ordered tree as a well-parenthesized expression:

`( ( ( ) ) ( ( ) ) ( ) ) ( ) ( )`

`ABDDBEGGEHHACCFF`

Counting well-parenthesized expresions

$X(n, m, k)$: set of all sequences containing n times 1, m times -1 with all prefix sums $\geq k$

Easy: $|X(n, m, -\infty)|$

Want: $|X(n, n, 0)|$

Prove: $|X(n, n, -\infty) \setminus X(n, n, 0)| = |X(n-1, n+1, -\infty)|$

Then: $|X(n, n, 0)| = \binom{2n}{n} - \binom{2n}{n-1} = \frac{(2n)!}{n!n!} - \frac{(2n)!}{(n-1)!(n+1)!}$

$= \frac{(2n)!(n+1-n)}{n!(n+1)!} = \binom{2n}{n}/(n+1)$

Example:

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

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

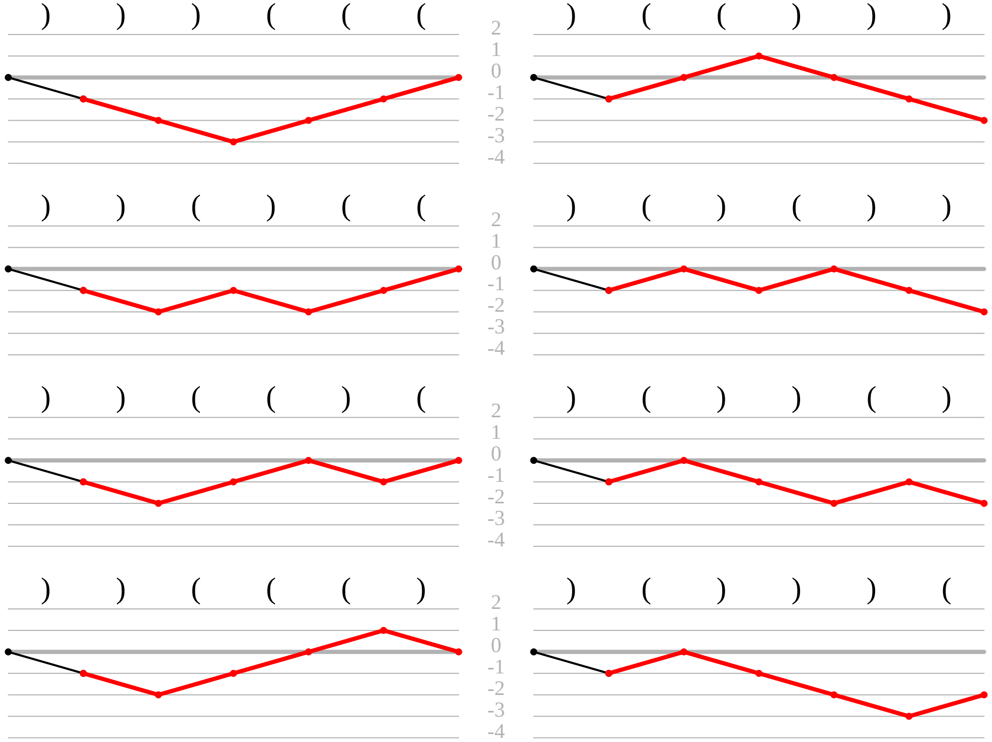
$|X(3, 3, 0)| = 20 - 15 = 5 = C_3$

Counting well-parenthesized expresions

$X(n, m, k)$: set of all sequences containing n times 1, m times -1 with all prefix sums $\geq k$

$$|X(n, n, -\infty) \setminus X(n, n, 0)| = |X(n-1, n+1, -\infty)|$$

Consider $n = 3$, first four examples out of 15:



Back to trees

The number of binary trees with n nodes is $C_n = \binom{2n}{n}/(n+1)$

Recall Stirling's approximation of $n!$:

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

$\ln(C_n) = \ln((2n)!) - 2\ln(n!) + O(\log n)$

$= 2n\ln(2n) - 2n - 2n\ln(n) + 2n + O(\log n)$

$= 2n\ln(2) + O(\log n)$

$\lg(C_n) = \ln(C_n)/\ln(2) = 2n + O(\log n)$

Thus OPT for representing binary trees is 2 bits per node