Succinct Data Structures

RAJEEV RAMAN

University of Leicester

Datum Cura

Data entrusted by “user” to “our” care:

a. Represent the data;

b. Retrieve data in response to queries.

Minimise resources: a. SPACE b. TIME

Large Data Sets

- Astronomical catalogues
- Biological sequences
- Terrestrial imaging data
- WWW linkage data
- Telephone company call logs

Large Data Sets

- Astronomical catalogues
- Biological sequences
- Terrestrial imaging data
- WWW linkage data
- Telephone company call logs

▷ Such data is almost invariably stored in compressed format to save space.

- Performance improvement (cf. IBM MXT):
  - Better use of memory levels close to processor;
  - Improved use of disk and memory bandwidth;
  - Reduced (mechanical) seek time.

Operating on Compressed Data

- Large inputs are invariably (?) compressed.
- Decompressing before operating has disadvantages:
  - Expensive (time), especially if only part of input is to be read;
  - May not be feasible (not enough disk space);
  - In-memory representation needed for complex computations?

Can (must?) we operate directly on compressed data?

Compression vs. Retrieval

Fast retrieval requires an *index* or *data structure* that may be large:

- *Suffix tree:* data structure for indexing a text of n bytes.
  - Supports many indexing and search operations.
  - ▷ good implementation: $10n$ bytes of index data [Kurtz, *SPrEx '99*]
- *Range Trees:* data structures for answering 3-D orthogonal range queries.
  - Provably good performance in the worst case.
  - ▷ $\Theta(n \lg n)$ space for storing n points.

Are rapid indexing and compression incompatible?

Compressed Data Structures

Succinct/Compressed Data Structures

Data Structures

- Preprocess input data so as to answer (long) series of *retrieval* or *update* operations.
- Want to minimize:
  1. Query time.
  2. Space usage of data structure.
  3. Time of pre-processing.
  4. Space for pre-processing.

  We do not consider updates, nor (3) or (4).

Succinct/Compressed Data Structures

Data: sets of integers; strings; trees; graphs; DAGs; relations . . .

- can be stored very space-efficiently and
- can support retrieval operations very efficiently.
- lower bounds prove the optimality of some solutions.

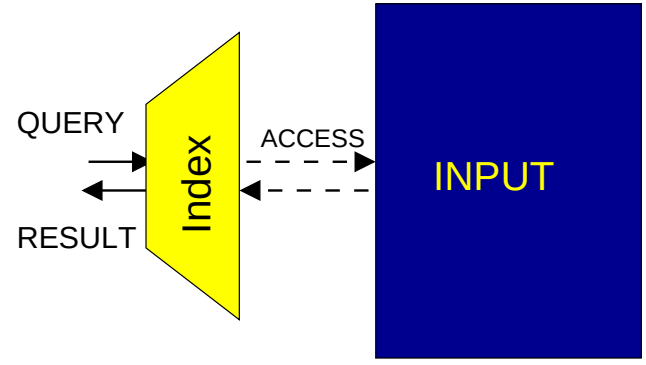
Model Random-Access Machine (RAM), logarithmic word size; uniform cost; space in bits (more later).

Memory Usage Models: General

- Measure total space usage after preprocessing.
- No restriction on access to data structure memory.
- Redundancy = memory usage − “space for data”.
  - “space for data” is a design parameter.
  - Data structure depends *crucially* on operations to be supported, and on the space target.
  - Trade-off: redundancy vs. query time.

ndexing model/Systematic Memory Mode

- Preprocess input to get *index*.
- Queries read index and *access* input (may be expensive).
- Redundancy = index size.



- Index size should be $o(\text{input size})$. Motivations for indexing model in [Barbay et al., *SODA '07*].
- Trade-off: redundancy vs. query time + access time. Easier to prove lower bounds.
- Index depends on operations to be supported and *access* functionality, many indices on same data.

Space Measures

Data Size	Redundancy	
Naive	$O(1)$	(implicit/in-place)
Information-theoretic	$O(\|\text{data}\|)$	(compact)
Information-theoretic	lower-order	(succinct)
Entropy (H_0)	"lower-order"	(density-sensitive)
Entropy (H_k)	"lower-order"	(compressed)
Other measure	"lower-order"	(compressed)

- Implicit DS: e.g. array representation of binary heap.
- Want *constant-time* operations: can be incompatible with redundancy being smaller than data size.
- NB: above discussion not really for succinct index.

Notions of Compressibility

Information-theoretic: Count total number of instances of a given size; take the logarithm base 2 ($\lg$).

Classical (Shannon) Entropy: Postulate (randomised) model of instance generation and calculate information content of instances generated by this model.

Empirical Entropy: Give (simple, uniform, computable) measure of compressibility of an *instance*. [cf. Kolmogorov Complexity.]

- Uniform distribution $\Rightarrow$ (Shannon $\equiv$ Info-Theoretic).
- Uniform distribution $\Rightarrow$ 'Shannon incompressible'
- Succinct data structures are key building blocks.

Example: Sequences

Data: $x \in \Sigma^n$ for some finite set of symbols $0, 1, \ldots, \sigma - 1$

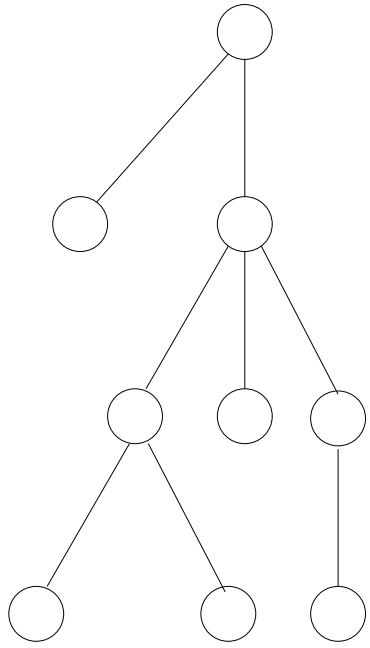
- Naive: $n \lceil \lg \sigma \rceil$ bits.
- Information-Theoretic: $\lg \sigma^n = \lceil n \lg \sigma \rceil$ bits [Dodis, Patrascu, Thorup, *STOC'10*]
- H_0: $nH_0 \sim n_0 \lg(n/n_0) + n_1 \lg(n/n_1)$ (binary).
  - $n_0 = n/2$, $nH_0 \sim n$ bits. $n_0 = 0.2n$, $nH_0 \sim 0.72n$ bits.
  - ▷ Closely related to $\lg \binom{n}{n_1}$.
- H_k: Separate frequencies for all *contexts* of length k:

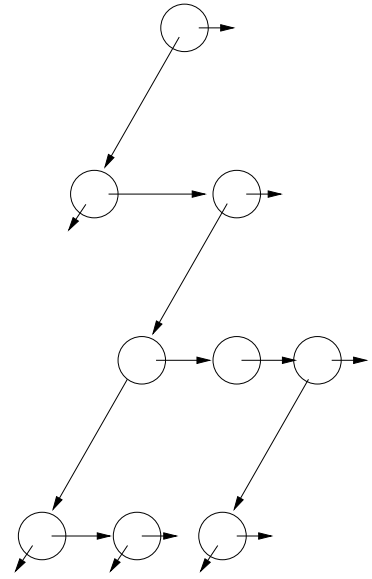
  ...1011...1010...1011...

  - For $0^{n/2}1^{n/2}$, $H_0 \sim n$ bits, $H_1 = O(\lg n)$ bits.
- Others: SLP compression, BDD compression, RLE compression etc.

Example: Ordinal Trees

Data: Given object x is an *ordinal* tree with n nodes.





- Ordinal: rooted tree, arbitrary # children, order matters.
- Naive: $\geq 2n$ pointers; $\Omega(n \lg n)$ bits.
- Information-Theoretic: $\lg\left(\frac{1}{n+1}\binom{2n}{n}\right) = 2n - O(\lg n)$ bits.

Example: Ordinal Trees

- Entropy (H_0): Only "degree sequence" entropy:

$$nH_0 \sim \sum_i n_i \lg(n/n_i)$$

  [Rote, *'97*] [Sadakane/Jansson, *JCSS'12*].

  - n_i = # nodes of degree i, e.g. $\lceil n/2 \rceil = n_0 = n_2 + 1$, $nH_0 \sim n$ bits.

- Entropy (H_k): ???

  ▷ *Labelled* ordinal trees [Luccio et al., *JACM'05*].

- Others: Tree-sharing compression [Bille et al., *SODA'11*].

Bit Vectors

Bit Vectors

Data: Sequence X of n bits, $x_0, \ldots, x_{n-1}$.

- $m = n_1$. Assume $m \leq n/2$.

Operations:

- $\mathit{rank}_1(i)$: number of 1s in $x_0, \ldots, x_i$.
- $\mathit{select}_1(i)$: position of ith 1.

Also rank_0, select_0.

Example: X = 01101001, $\mathit{rank}_1(4) = 3$, $\mathit{select}_0(4) = 6$.

Want *rank* and *select* to take $O(1)$ time.

Bitvectors: Space Usage

Recall space usage may be:

- n bits (succinct).
- $\sim nH_0(X)$ bits = $\lg \binom{n}{m} + o(n) = m \lg(n/m) + O(m)$ bits.
  - $m = n/\lg n \Rightarrow nH_0(X) \sim n \lg\lg n / \lg m = o(n)$ bits.
  - $m \ll n \Rightarrow nH_0(X) \sim m \lg n \ll n$ bits.
  - $o(n)$ term is *not necessarily* a "lower-order" term.
  - Unavoidable if $O(1)$ time *rank* or *select*$_0$ has to be supported [Patrascu/Thorup, *STOC '06*].
- $H_k(X)$, or other measures.
- We will also consider *systematic* or *succinct index* data structures.

Bit-Vectors: Reductions

- $\mathit{rank}_0(i) + \mathit{rank}_1(i) = i + 1$.
- Reduction of to rank_1 to select_0:

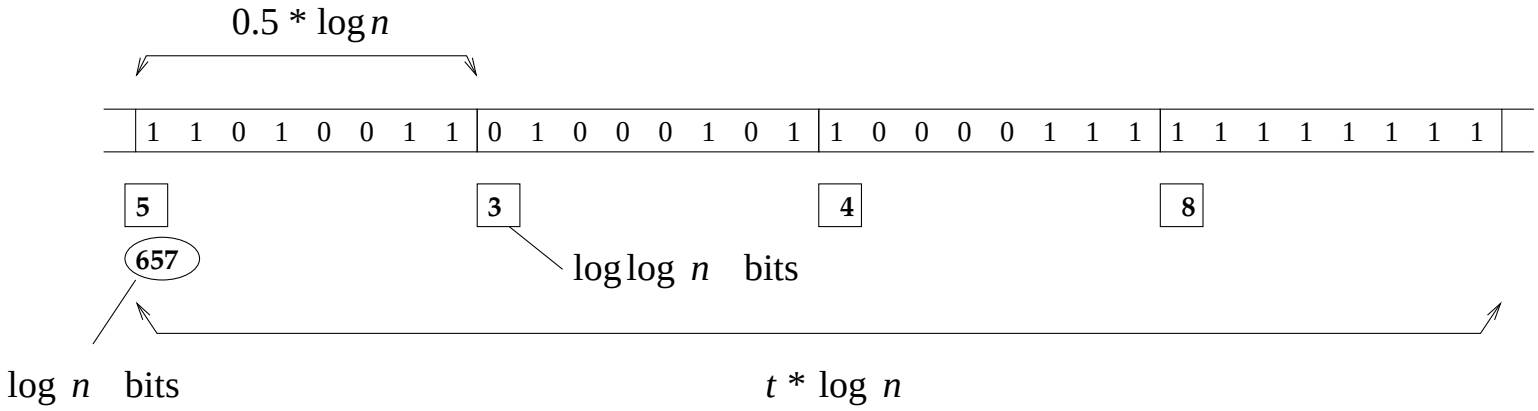
```
X    0 1 1 0 1 0 0 0 1                  (n)
Y    0 1 0 1 0 0 1 0 0 0 0 1 0          (n+m)
```

  $\mathit{select}_0(Y, j) - j + 1 = \mathit{rank}_1(X, j)$ [Elias, *J. ACM '75*]. Implications for compressed BVs.

- Can also reduce select_0 and select_1 to just select_1 + *rank*. [Delpratt et al., *WEA '06*].

Bit-Vector Data Structures

Aim: to support *rank*(i) and *select*(i) using $n + o(n)$ bits.



$$\text{Space} = n + O\left(\frac{n}{(\lg n)^2} \cdot \lg n + \frac{n}{\lg n} \cdot \lg \lg n\right) + O(\sqrt{n} \cdot \lg n)$$
$$= n + o(n) \text{ bits.}$$

- *rank*(x) $O(1)$ time via "table lookup".
- *select*(x) : ?

Compressed Bit Vectors: Lower Bounds

Aim: to support *rank*(i) and *select*(i) in $O(1)$ time using $nH_0(X) + \ldots$ bits. $m = n_1$, assume $m \ll n$.

- $nH_0(X) \sim \lg \binom{n}{m} = m \lg(n/m) + O(m)$ bits.
- Consider $X = x_0, \ldots, x_{n-1}$ as characteristic vector of $S \subseteq \{0, \ldots, n-1\}$. Then:

$$\mathit{rank}_1(i) = |y \in S | y \leq i|$$

  closely related to "predecessor search".
- Lower bounds [Patrascu/Thorup, *STOC 06*] imply: $O(nH_0)$ bits space possible only when $m = n/(\lg n)^{O(1)}$, for $O(1)$-time *rank* (and *select*$_0$).
- Compressed $O(1)$-time BVs exist only for dense sequences.

Compressed Bit Vectors: Results

- Space $nH_0 + O(n \lg\lg n / \lg n)$ bits, supports *rank*, *select*$_0$, *select*$_1$. [Raman et al., *TALG'07*].
  - Space $o(n)$ bits whenever $m = o(n)$, many applications.
  - Uses more than nH_0 bits if m is small.
- $nH_0 + o(m)$ bits supports *rank*, *select*$_0$, *select*$_1$. [Golynski et al., *ESA'07*].
  - $O(1)$ time only for not too sparse BVs $m = n/(\lg n)^{O(1)}$.
- $nH_0 + o(n)$ bits supports only *select*$_1$. [Elias, *J. ACM'75*], [Grossi/Vitter, *SICOMP'06*], [Raman et al., *TALG'07*].
  - Space is always $O(nH_0)$.

Compressed BVs

Aim: Using $nH_0 + o(n)$ bits support *rank*, *select* in $O(1)$ time [Raman et al., *TALG'07*]. (Ideas below are older.)

- Divide input bit-string into $O(n/\lg n)$ blocks of size $b = (\lg n)/2$. Group $\lg n$ blocks into superblock.
- Let number of 1s in i-th block be m_i ($\sum_i m_i = m$).
- For each block store:
  - sum from start of superblock until start of block and m_i using $O(\log \log n)$ bits each.
  - a bit string of $\left\lceil \lg \binom{b}{m_i} \right\rceil$ bits that specifies the block.
    - ▷ may be $\ll b$ bits. *Variable-length* encoding.
- Key point: $\sum_i \lg \binom{b}{m_i} \leq \lg \binom{\sum_i b}{\sum_i m_i} = \lg \binom{n}{m}$.
- "Lower-order" term is $O(n \lg \lg n / \lg n) = o(n)$.

Compressed BVs: Elias-Fano

Aim: Using $nH_0 + o(m)$ bits support *only* *select*$_1$ in $O(1)$ time. [Elias, *J. ACM'75*], [Grossi/Vitter, *SICOMP'06*], [Raman et al., *TALG'07*].

- Space is always $O(nH_0)$.
- Consider bit-vector X as *subset* of $\{0, \ldots, n-1\}$, i.e. as characteristic vector of set $X = \{x_1, \ldots, x_m\}$.
- *select*$_1(i) = x_i$.

MSB Bucketing

Bucket according to most significant b bits: $x \to \lfloor x/2^{\lceil \lg n \rceil - b} \rfloor$.

Example. $b = 3, \lceil \lg n \rceil = 5, m = 7$.

x_1	0	1	0	0	0
x_2	0	1	0	0	1
x_3	0	1	0	1	1
x_4	0	1	1	0	1
x_5	1	0	0	0	0
x_6	1	0	0	1	0
x_7	1	0	1	1	1

Bucket	Keys
000	–
001	–
010	x_1, x_2, x_3
011	x_4
100	x_5, x_6
101	x_7
110	–
111	–

Bucketing saves space

- ▷ Store only low-order bits.
- ▷ (Keep cumulative bucket sizes.)

Example

select(6)

bkt	sz	data
000	0	–
001	0	–
010	0	$\underbrace{00}_{x_1}, \underbrace{01}_{x_2}, \underbrace{11}_{x_3},$
011	3	$\underbrace{01}_{x_4}$
100	4	$\underbrace{00}_{x_5}, \underbrace{10}_{x_6}$
101	6	$\underbrace{11}_{x_7}$

Space Savings

- Choose $b = \lfloor \lg m \rfloor$ bits. In bucket: $\lceil \lg n \rceil - \lfloor \lg m \rfloor$-bit keys.
- $m \lg n - m \lg m + O(m)$ + space for cumulative bucket sizes.

Bucket no:	000	001	010	011	100	101	110	111
Bucket size:	0	0	3	1	2	1	0	0

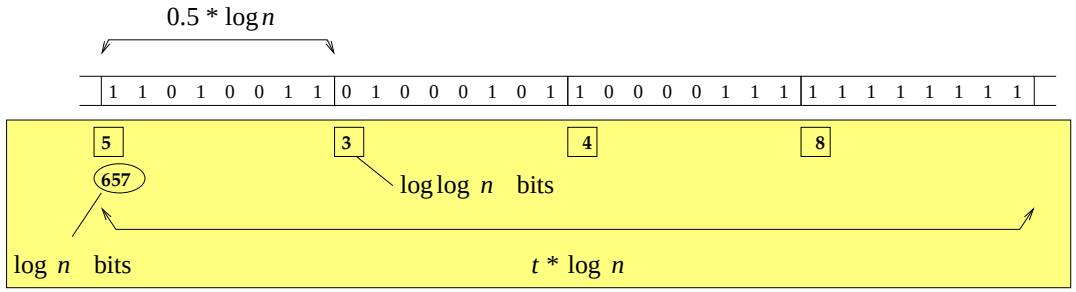
Unary encoding: $0, 0, 3, 1, 2, 1, 0, 0 \rightarrow 110001010010111$.

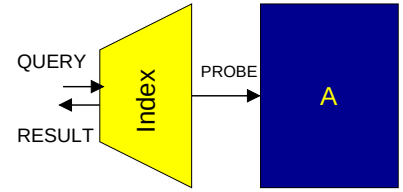
z buckets, total size $m \Rightarrow m + z$ bits ($z = 2^{\lfloor \lg m \rfloor}$).

- Total space = $m \log(n/m) + O(m) = nH_0 + O(m)$ bits.
- In which bucket is the 6th key? ▷"Index of 6th 0" $- 6$.

Higher-Order Compression

The standard approach to bit-vectors is a *succinct index*: the input bit-vector can be in "read-only" memory; works if we can read $(\lg n)/2$ consecutive bits from X.





Theorem: A bit string X can be represented using $nH_k + O(n \lg\lg n / \lg n)$ bits, for any constant $k > 0$, such that $O(\lg n)$ consecutive bits can be decoded in $O(1)$ time. [Ferragina/Venturini, *SODA'07*].

Corollary: A bit string X can be represented using $nH_k + O(n \lg\lg n / \lg n)$ bits, for any constant $k > 0$, such that *rank*/*select* can be aupported in $O(1)$ time.

Redundancy Lower Bounds

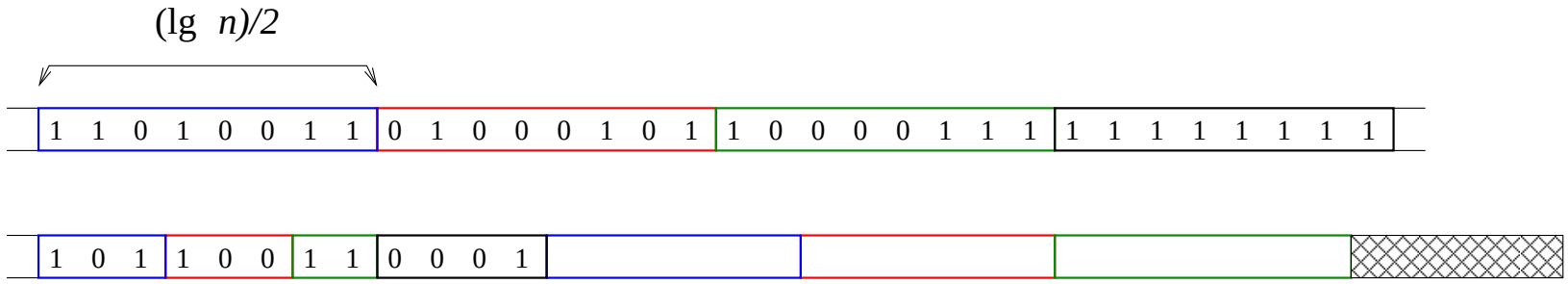
Lower order term for succinct *rank*/*select* is $O(n \lg\lg n / \lg n)$ bits, *optimal* for succinct indices [Golynski, *TCS'07*]. In general model, can do better:

- Block of size $b = (\lg n)/2$ bits represented as pair $\langle m_i, c_i \rangle$; $m_i =$ # 1s in block, c_i is a number from $\binom{b}{m_i}$.
  - Naive representation of $m_i \Rightarrow b + \Theta(\lg\lg n)$ bits.
  - Encode m_i using a Huffman code: $b + O(1)$ bits [Golynski et al., *ESA'07*].
  - Applying again gives $O(n(\lg\lg n)/(\lg n)^2)$ term.
- Approach extended to give $O(n/(\lg n)^c)$ redundancy for any constant c [Patrascu, *FOCS'08*]. Optimal for for *rank*/*select* in $O(1)$ time [Patrascu/Viola, *SODA'10*].

Example of Informative Encoding

Example, $b = 8$, naive encoding of m_i needs 4 bits.

m_i	$\binom{b}{m_i}$	$\lg \binom{b}{m_i}$	Huffmann	Total
0	1	0	0000	4
1	8	3	0010	7
2	28	5	010	8
3	56	6	100	9
4	70	7	11	9
5	56	6	101	9
6	28	5	011	8
7	8	3	0011	7
8	1	0	0001	4



Summary of Results

BVs supporting *rank*, *select* in $O(1)$ time:

- succinct index: redundancy (index size): $\Theta(n \lg \lg n / \lg n)$.
  - succinct index when the bitvector is sparse [Golynski et al., *SWAT'08*].
- general model: $nH_0 + n/(\lg n)^c$ bits (optimal), but compressed BVs exist only for $m = \Omega(n/(\lg n)^c)$ for some constant c.

CBVs supporting *select*$_1$ in $O(1)$ time and using space $nH_0 + o(m)$ exist.

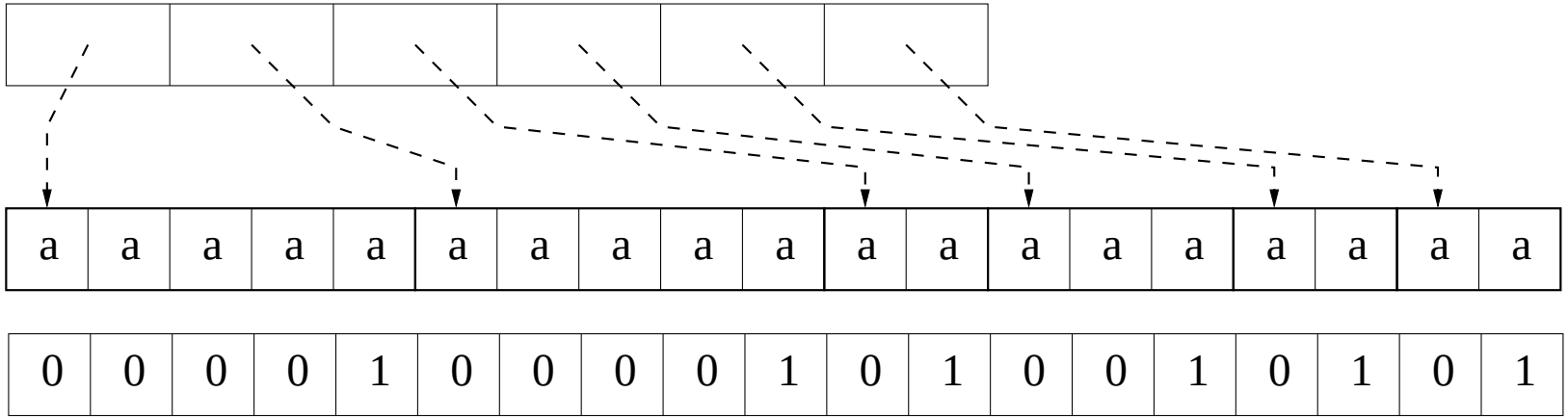
Simple Applications

- Many good implementations of (compressed) BVs: `rsdic`, `Sux4J`.
- Typical performance for *rank* is 1-2 random memory access, select is 2-3 times slower (depends on input).
- Typical tasks:
  - Array of (short) strings.
  - 2D Geometric data.

Array of (short) strings

Data: m variable-length strings, total size n. Access i-th string.

- Naive: $8n$ bits (raw) + $64m$ bits
- Bitvector: $8n$ bits (raw) + n bits.
- Compressed BV: $8n$ bits (raw) + $m \log(n/m) + O(m)$ bits.



▷ XML textual data [Delpratt et al., *EDBT'08*], $n/m \sim 11$.

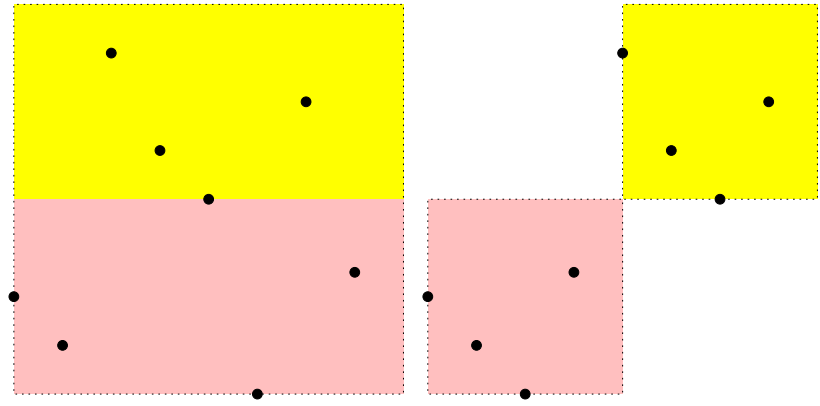
Geometric Data

Data: 2D point set of size $n = 2^k$ in *rank space*.

- x-coordinates $0, \ldots, n-1$.
- y-coordinates $0, \ldots, n-1$, permutation.
- Write first bits of y-coordinates:

  `0 0 1 1 1 0 1 0`

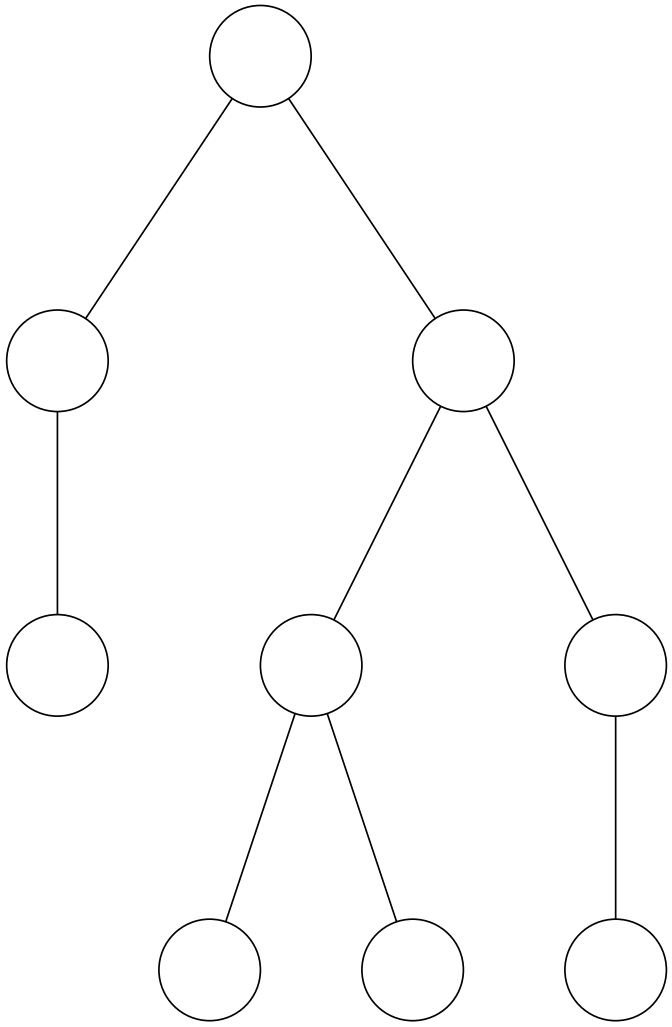
  Store in BV, *rank*/*select* to map.



Basis of “compressed range tree” [Chazelle, *FOCS’83*], precursor to wavelet tree.

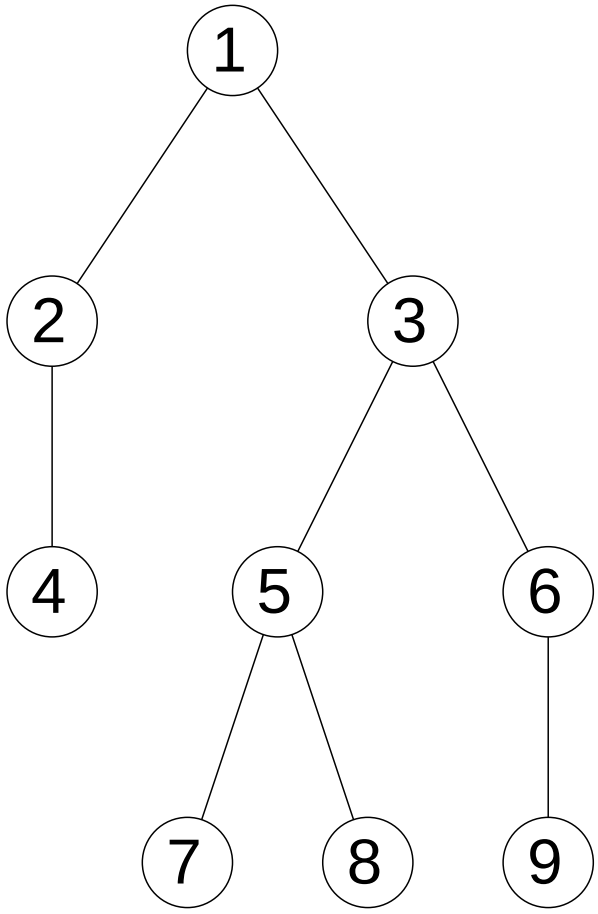
Trees

Ordinal Trees



- Arbitrary rooted (static) tree.
- Ordered children.
- Operations
  - *parent*(x)
  - *child*(x, i)
  - $\ldots$
- Optimal: $2n - O(\lg n)$ bits.

LOUDS



10 110 10 110 0
110 10 0 0 0

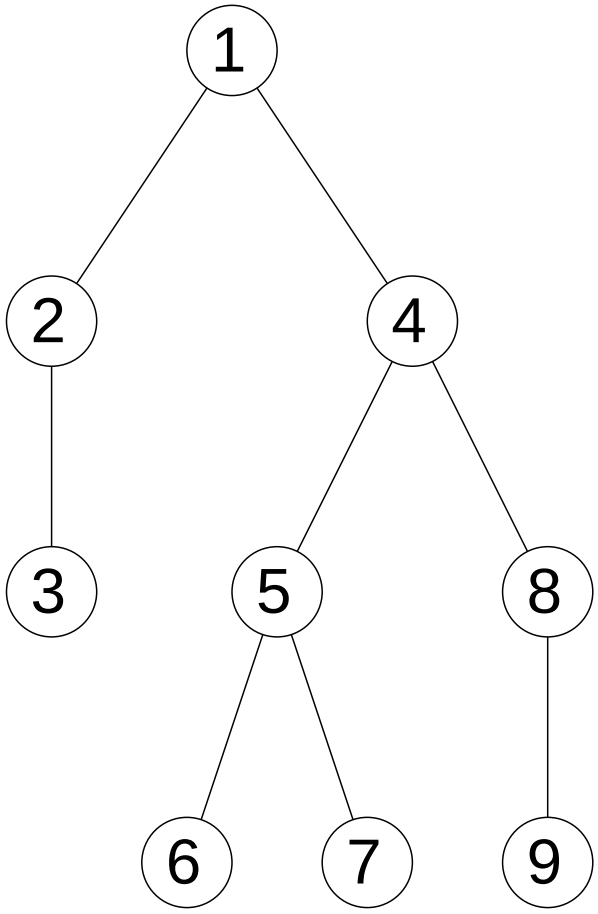
Level-Order Unary Degree Sequence [Jacobson, *FOCS'89*]

- Number nodes in level order (*).
- Write 10, then 1^d0, $2n+1$ bits.
- i-th node "accessed" by position of i-th 1 $1 \to 1$, $5 \to 8$ etc. (*)
- $\mathit{child}(x,i) = \mathit{select}_0(\mathit{rank}_1(x)+i)$
  - Second child of node 3.
  - Node 3 is represented as 4, $\mathit{rank}_1(4)=3$, $\mathit{select}_0(3)=7+2=9$.
  - 9th bit represents node 6 ($\mathit{rank}_1(9)=6$) so 6.
- $\mathit{parent}(x) = \mathit{select}_1(\mathit{rank}_0(x))$
  - Parent of node 8.
  - Node 8 is represented as 14, $\mathit{rank}_0(14)=5$, $\mathit{select}_1(5)=8$.
  - 8th bit represents node 5 ($\mathit{rank}_1(8)=5$) so 5.

Features of Succinct Tree Representations

- For any succinct tree representation, “natural” numbering is fixed by the representation.
- Some representations can support other numberings using indices of size $o(n)$ bits. This is not always easy.
  - ▷ Using LOUDS bitstring, support depth-first numbering is open.
- Representation has “bitstring/physical” numbering, usually from $1..O(n)$, and “natural” numbering from $1..n$.
  - Mapping is “easy” in theory, overhead in practice.
  - Avoid using “double numbering” [Delpratt et al., *WEA'06*].
- Different representations support different operations (?).

Parentheses



((())(()()))(()))))
1232123434323432 10

- $2n$ bits.
- "Natural" numbering: DFS (pre/post).
- Basic navigation by *find-open*, *find-close*, *enclose*operations [Jacobson, *FOCS'89*] [Munro/Raman, *SICOMP'01*].
- More advanced operations by *excess search*. [Sadakane, *SODA'02*], [Munro/Rao, *ICALP'04*],[Sadakane/Navarro, *SODA'10*].
- Excess of a parenthesis string = $\#open - \#close$.
  - ▷ Excess array is implicit in the parenthesis string and is not stored explicitly.
- Lots of operations.

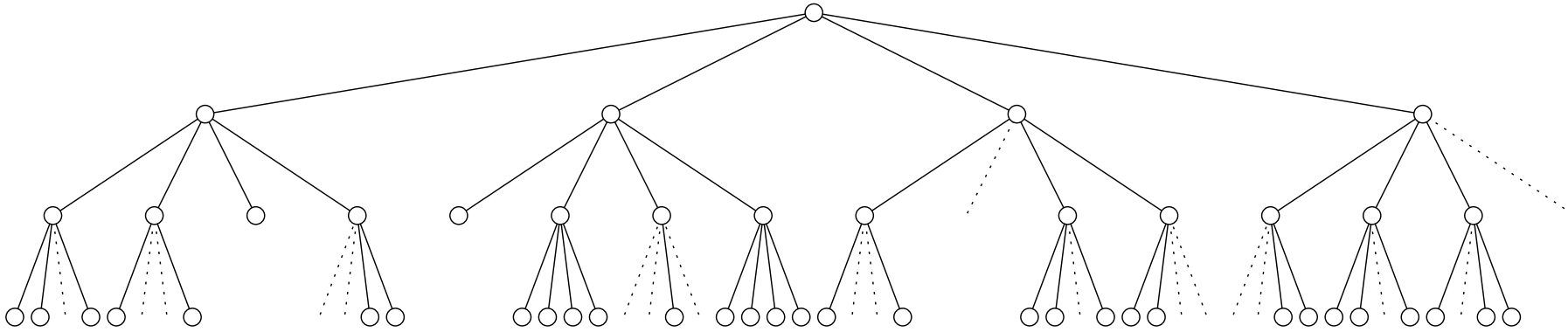
Other Representations

Many other ordinal tree representations:

- DFUDS – depth-first unary degree sequence [Benoit et al., *Algorithmica'05*], [Sadakane/Jansson, *JCSS'12*].
- Partition [Geary et al., *TALG'06*], [He et al., *TALG'11*], [Farzan/Munro, *Algorithmica'12*].
- Parentheses, DFUDS, Partition *seem* to have similar functionality. Very few candidates for separating them, but there seem to be some unknowns.
  - Cannot have "union" of DFUDS and parenthesis since space would become $4n + o(n)$.
- Are they the "same"?
  - Cannot even separate LOUDS from Parenthesis...
- "Universal" representation [Farzan et. al, *ICALP'09*]:
  - Can simulate access to DFUDS, parenthesis and partition bit-string.

Cardinal Trees

n nodes, $n-1$ edges, each edge labelled with $\sigma \in \{0, \ldots, k-1\}$.



$k = 4, n = 50$

- *parent*(x);
- *child*(x, σ).

} in $O(1)$ time.

Succinct space bound: $\left\lceil \lg \left(\frac{1}{kn+1} \binom{kn+1}{n} \right) \right\rceil$

Cardinal trees

Number nodes $0, \ldots, n-1$ in level order

$S = \{\langle i, \sigma \rangle \mid \exists x, i \xrightarrow{\sigma} x\}$

▷ $\{\langle i, \sigma \rangle\} \leftrightarrow \{0, \ldots, kn-1\}$.

▷ $|S| = n - 1$.

Operations

parent(x) =

y : *select*$(x) = \langle y, \sigma \rangle$

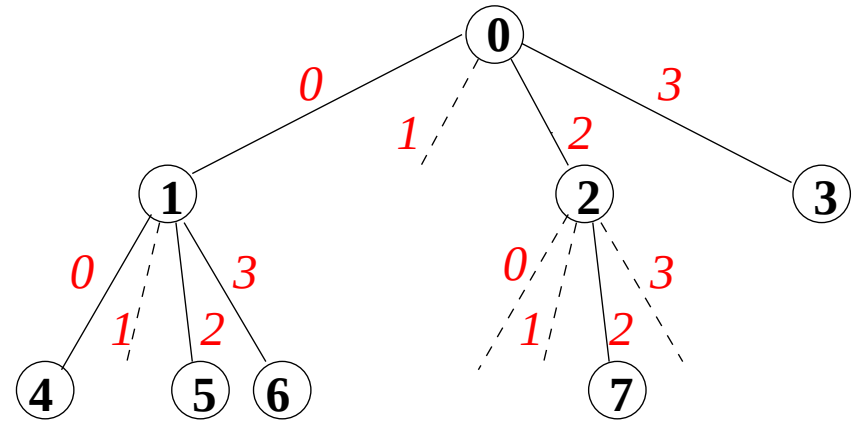
child(x, σ) = *rank*$(\langle x, \sigma \rangle)$ [a]

Space

$\binom{kn}{n-1} = \frac{n}{kn+1}\binom{kn+1}{n}$

[a] Special kind of *rank*.

$k = 4, n = 8$



$\underbrace{\langle 0,0 \rangle}_{1}, \underbrace{\langle 0,2 \rangle}_{2}, \underbrace{\langle 0,3 \rangle}_{3}, \underbrace{\langle 1,0 \rangle}_{4}, \underbrace{\langle 1,2 \rangle}_{5},$

$\underbrace{\langle 1,3 \rangle}_{6}, \underbrace{\langle 2,2 \rangle}_{7}$

[Jacobson, *FOCS'89*], [Raman et al., *TALG'07*].

Summary and Conclusions

Covered some of the basics of succinct and compressed data structures:

- Bitvectors
- Trees

Not covered many “advanced” topics (excluding other invited talks...)

- Dynamization.
- Rank/Select on Large Alphabets.
- Compressed Suffix Arrays/FM-Index.
- Path Searches on Labelled Trees/XBW Transform.
- Representing Permutations and Functions.
- Representing Graph Families.
- Dictionaries and Indexable Dictionaries.
- Grammar-compressed data and “non-linear” data (see ALSIP 2011 talk).
- Implementation issues.
- Lower Bound Techniques.