Discrete Methods in Mathematical Informatics

Kunihiko Sadakane
The University of Tokyo

http://researchmap.jp/sada/resources/

Problems in Handling Big Data

- Data size (n) is huge
  - human genome: $n = 3G$ (three billion)
  - Web pages: $n > 10G$
- Data do not fit in the main memory of computers
  - parallel/distributed algorithms
  - streaming algorithms
  - data compression

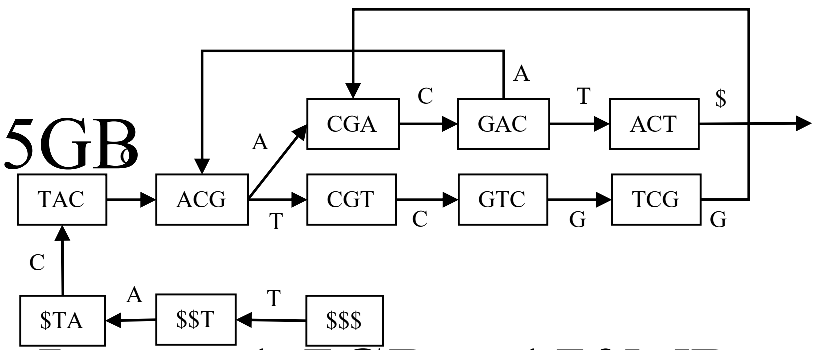
Problems in Data Compression

- Conventional data compression algorithms are not suitable for data processing
  - slow partial decompression
  - not searchable

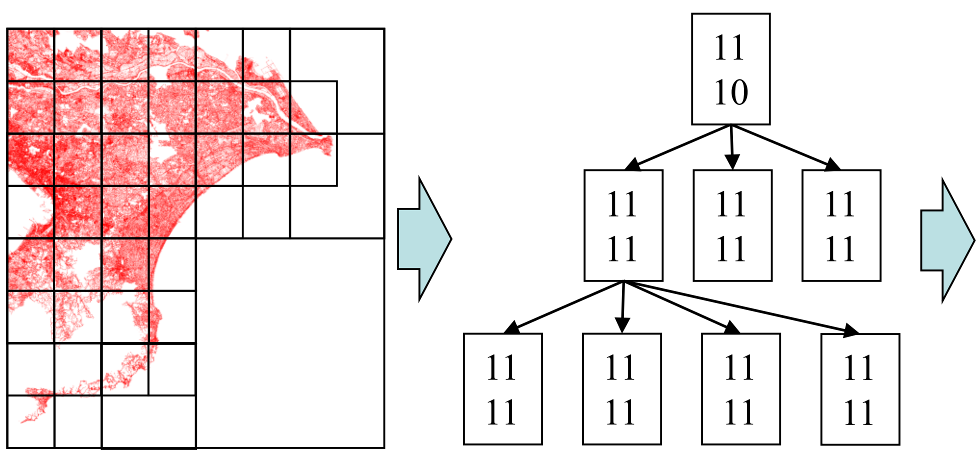
⇒ Succinct Data Structures

Examples of Succinct Data Structures

- String index
  - Suffix array of 100GB text: 680GB→22GB
- Genome assembly
  - Human genome: 300GB→2.5GB



- Road networks
  - Positional data of all roads in Japan: 1.7GB→170MB



1110111111111111
1001010110101010

represented by
bitstring

Succinct data structures = Succinct representation of data + Succinct index

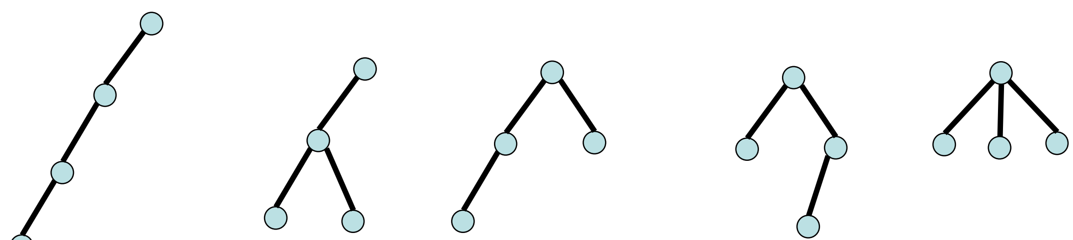
- A representation of data whose size (roughly) matches the information-theoretic lower bound
- If the input is taken from L distinct ones, its information-theoretic lower bound is $\lceil \lg L \rceil$ bits
- Example 1: a lower bound for a set $S \subseteq \{1,2,\ldots,n\}$
  - $\lg 2^n = n$ bits

$$
\begin{array}{llll}
\emptyset & & & \\
\{1\} & \{2\} & \{3\} & n = 3 \\
\{1,2\} & \{1,3\} & \{2,3\} & \\
\{1,2,3\} & & &
\end{array}
$$

$\lg x = \log_2 x$

- Example 2: *n* node ordered tree

$$\lg \frac{1}{2n-1}\binom{2n-1}{n-1} = 2n - \Theta(\lg n) \text{ bits}$$

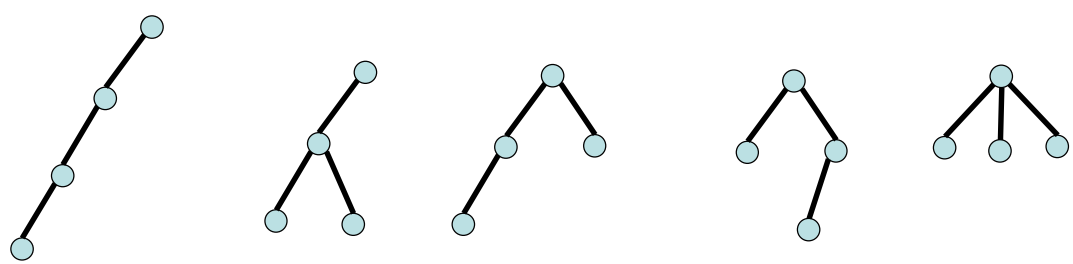


$n = 4$

- Example 3: length-*n* string
  - $n \lg \sigma$ bits (σ: alphabet size)

- For any data, there exists its succinct representation
  - enumerate all in some order, and assign codes
  - can be represented in $\lceil \lg L \rceil$ bits
  - not clear if it supports efficient queries
- Query time depends on how data are represented
  - necessary to use appropriate representations

$$n = 4 \quad \left\lceil \lg \frac{1}{2n-1} \binom{2n-1}{n-1} \right\rceil = \lceil \lg 5 \rceil = 3 \text{ bits}$$



000 001 010 011 100

Succinct Indexes

- Auxiliary data structure to support queries
- Size: o(lg L) bits
- (Almost) the same time complexity as using conventional data structures
- Computation model: word RAM
  - assume word length w = lg lg L
    (same pointer size as conventional data structures)

word RAM

- word RAM with word length w bits supports
  - reading/writing w bits of memory at arbitrary address in constant time
  - arithmetic/logical operations on two w bits numbers are done in constant time
  - arithmetic ops.: +, –, *, /, lg (most significant bit)
  - logical ops.: and, or, not, shift
- These operations can be done in constant time using $O(w^\varepsilon)$ bit tables ($\varepsilon > 0$ is an arbitrary constant)

Bit Vectors

- B: 0,1 vector of length n $B[0]B[1]\dots B[n-1]$
- lower bound of size = $\lg 2^n = n$ bits
- queries
  - $rank(B, x)$: number of ones in $B[0..x]=B[0]B[1]\dots B[x]$
  - $select(B, i)$: position of i-th 1 from the head ($i \geq 1$)
- basics of all succinct data structures
- naïve data structure
  - store all the answers in arrays
  - $2n$ words ($2\,n \log n$ bits)
  - O(1) time queries

$$\begin{array}{l} \quad\ \ 0 \quad\ \ 3 \ \ 5 \quad\ \ \ 9 \\ B = 1001010001000000 \end{array}$$

$$n = 16$$

Succinct Index for Rank:1

- Divide B into blocks of length $\lg^2 n$
- Store $rank(x)$'s at block boundaries in array R

$$rank(x) = \sum_{i=1}^{x} B[i] = \underbrace{\sum_{i=1}^{\lfloor x/\lg^2 n \rfloor} B[i]}_{R[x/\lg^2 n]} + \sum_{i=\lfloor x/\lg^2 n \rfloor + 1}^{x} B[i]$$

- Size of R

$$\frac{n}{\lg^2 n} \cdot \lg n = \frac{n}{\lg n} \quad \text{bits}$$

- $rank(x)$: $O(\lg^2 n)$ time

Succinct Index for Rank:2

- Divide each block into small blocks of length ½lg n
- Store *rank*(x)'s at small block boundaries in R_2

$$rank(x) = \sum_{i=1}^{x} B[i] = \underbrace{\sum_{i=1}^{\lfloor x/\lg^2 n \rfloor} B[i]}_{R_1[x/\log^2 n]} + \underbrace{\sum_{i=\lfloor x/\lg^2 n \rfloor+1}^{\lfloor x/\frac{1}{2}\lg n \rfloor} B[i]}_{R_2[x/\log n]} + \sum_{i=\lfloor x/\frac{1}{2}\lg n \rfloor+1}^{x} B[i]$$

- Size of R_2

$$\frac{n}{\frac{1}{2}\lg n} \cdot \lg\left(\lg^2 n\right) = \frac{4n \lg\lg n}{\lg n} \quad \text{bits}$$

- *rank*(x): O(lg n) time

Succinct Index for Rank:3

- Compute answers for all possible queries for small blocks in advance, and store them in a table

$$\sum_{i=\lfloor x/\frac{1}{2}\lg n \rfloor+1}^{x} B[i]$$

All patterns of ½ lg n bits

	x = 0	1	2
000	0	0	0
001	0	0	1
010	0	1	1
011	0	1	2
100	1	1	1
101	1	1	2
110	1	2	2
111	1	2	3

- Size of table

$$2^{\frac{1}{2}\lg n} \cdot \tfrac{1}{2}\lg n \cdot \lg\left(\tfrac{1}{2}\lg n\right)$$
$$= O\left(\sqrt{n}\lg n \lg\lg n\right)$$
$$= o\left(\frac{n \lg\lg n}{\lg n}\right) \text{ bits}$$

Theorem: Rank on a bit-vector of length n is computed in constant time on word RAM with word length $\Theta(\lg n)$ bits, using $n + O(n \lg \lg n / \lg n)$ bits.

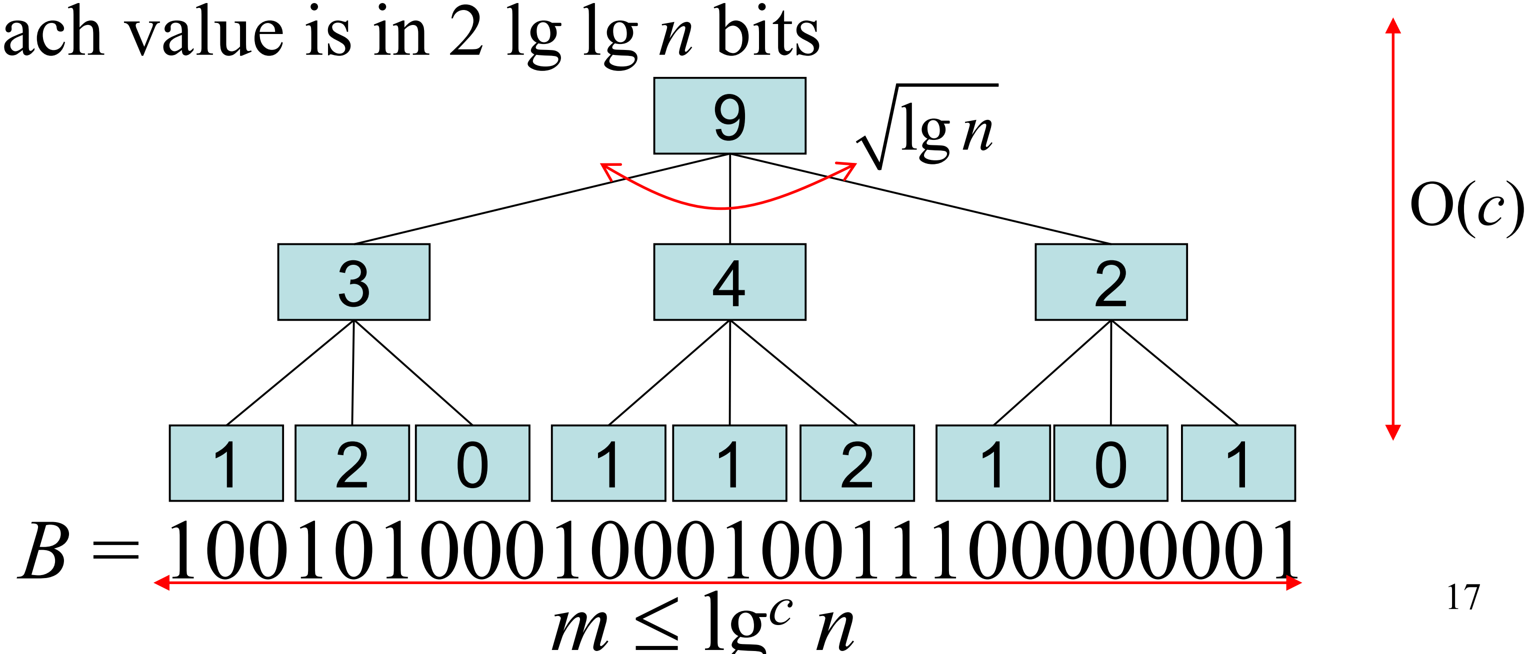
Note: The size of table for computing rank inside a small block is $O\left(\sqrt{n} \lg n \lg \lg n\right)$ bits. This can be ignored theoretically (asymptotically), but not in practice.

Succinct Index for Select

- More difficult than *rank*
  - Let $i = q\,(\lg^2 n) + r\,(\frac{1}{2}\lg n) + s$
  - if i is multiple of $\lg^2 n$, store $select(i)$ in S_1
  - if i is multiple of $\frac{1}{2}\lg n$, store $select(i)$ in S_2
  - elements of S_2 may become $\Theta(n) \rightarrow \Theta(\lg n)$ bits are necessary to store them $\rightarrow \Theta(n)$ bits for the entire S_2
  - *select* can be computed by binary search using the index for *rank*, but it takes $O(\lg n)$ time
- Divide B into large blocks, each of which contains $\lg^2 n$ ones.
- Use one of two types of data structures for each large block

- If the length of a large block exceeds $\lg^c n$
  - store positions of $\lg^2 n$ ones
  - $\lg^2 n \cdot \lg n = \lg^3 n$ bits for a large block
  - the number of such large blocks is at most $\dfrac{n}{\lg^c n}$
  - In total $\dfrac{n}{\lg^c n} \cdot \lg^3 n$ bits
  - By letting $c = 4$, index size is $\dfrac{n}{\lg n}$ bits

- If the length m of a large block is at most $\lg^c n$
  - devide it into small blocks of length ½ lg n
  - construct a complete $\sqrt{\lg n}$ -ary tree storing small blocks in their leaves
  - each node of the tree stores the number of ones in the bit vector stored in descendants
  - number of ones in a large block is $\lg^2 n$
    $\rightarrow$ each value is in 2 lg lg n bits



- The height of the tree is $O(c)$
- Space for storing values in the nodes for a large block is $O\left(\frac{cm \lg \lg n}{\lg n}\right)$ bits
- Space for the whole vector is $O\left(\frac{cn \lg \lg n}{\lg n}\right)$ bits
- To compute *select*(i), search the tree from the root
  - the information about child nodes is represented in $\sqrt{\lg n} \cdot 2 \lg \lg n = O(\lg n)$ bits → the child to visit is determined by a table lookup in constant time
- Search time is $O(c)$, i.e., constant

Theorem (Data structure BV): *rank* and *select* on a bit-vector of length n is computed in constant time on word RAM with word length $\Theta(\lg n)$ bits, using $n+O(n \lg \lg n / \lg n)$ bits.

Note: *rank* inside a large block is computed using the index for *select* by traversing the tree from a leaf to the root and summing *rank* values.

Extension of Rank/Select (1)

- Queries on 0 bits
  - $rank_c(B, x)$: number of c in $B[0..x] = B[0]B[1]\ldots B[x]$
  - $select_c(B, i)$: position of i-th c in B ($i \geq 1$)
- from $rank_0(B, x) = (x+1) - rank_1(B, x)$, $rank_0$ is done in O(1) using no additional index
- $select_0$ is not computed by the index for $select_1$ add a similar index to that for $select_1$

Extension of Rank/Select (2)

- Compression of sparse vectors
  - A 0,1 vector of length n, having m ones
  - lower bound of size $\left\lceil \lg \binom{n}{m} \right\rceil \approx m \lg \frac{n}{m} + (n-m) \lg \frac{n}{n-m}$
  - if $m << n$

$$\left\lceil \lg \binom{n}{m} \right\rceil \approx m \lg \frac{n}{m} + 1.44m < n$$

- Operations to be supported
  - $rank_c(B, x)$: number of c in $B[0..x] = B[0]B[1]\ldots B[x]$
  - $select_c(B, i)$: position of i-th c in B ($i \geq 1$)

Entropy of String

Definition: order-0 entropy H_0 of string S

$$H_0(S) = \sum_{c \in A} p_c \lg \frac{1}{p_c}$$ (p_c: probability of appearance of letter c)

Definition: order-k entropy

– assumption: $\Pr[S[i] = c]$ is determined from $S[i-k..i-1]$ (context)

$$nH_k(S) = \sum_{s \in A^k} n_s \sum_{c \in A} p_{s,c} \lg \frac{1}{p_{s,c}}$$

$\underbrace{\text{abcabab}}_{\text{context}}\boxed{\text{c}}$

– n_s: the number of letters whose context is s

– $p_{s,c}$: probability of appearing c in context s

- The information-theoretic lower bound for a sparse vector asymptotically matches the order-0 entropy of the vector

$$
\begin{aligned}
\left\lceil \lg \binom{n}{m} \right\rceil &\approx m \lg \frac{n}{m} + (n-m) \lg \frac{n}{n-m} \\
&= n \sum_i p_i \lg \frac{n}{p_i} \\
&= nH_0(B)
\end{aligned}
$$

Compressing Vectors (1)

- Divide vector into small blocks of length $l = ½ \lg n$
- Let m_i denote number of 1's in i-th small block B_i
  - a small block is represented in $\underbrace{\lg l}_{\text{\#ones}} + \underbrace{\left\lceil \lg \binom{l}{m_i} \right\rceil}_{\text{\#possible patterns with speficied \#ones}}$ bits

#ones

#possible patterns with speficied #ones

- Total space for all blocks is

$$\sum_{i=0}^{n/l} \left(\lg l + \left\lceil \lg \binom{l}{m_i} \right\rceil \right) \le \sum_{i=0}^{n/l} \left(\lg l + \lg \binom{l}{m_i} + 1 \right)$$

$$\le \lg \prod_{i=0}^{n/l} \binom{l}{m_i} + \frac{n}{l}(1 + \lg l)$$

$$\le \lg \binom{n}{m} + O\left(\frac{n \lg \lg n}{\lg n} \right)$$

- indexes for *rank*, *select* are the same as those for uncompressed vectors: $O(n \log \log n / \log n)$ bits
- numbers of 1's in small blocks are already stored in the index for *select*
- It is necessary to store pointers to small blocks. We use the following Theorem.

Theorem (Data structure TY): Let $z_1, z_2, \ldots, z_k$ be an integer sequence s.t. $|z_i| = n^{O(1)}$ and $\min\{|z_i|, |z_i - z_{i-1}|\} = \mathrm{polylog}(n)$. Then this sequence can be represented in $O(k \lg \lg n)$ bits and each z_i is obtained in constant time on $\Omega(\lg n)$-bit word-RAM.

- let p_i denote the pointer to B_i
  - $0 = p_0 < p_1 < \ldots < p_{n/l} < n$ (less than n because compressed)
  - $p_i - p_{i-1} \leq ½ \log n$
- We store p_i's using the theorem. The space is $O(n/l \cdot \lg\lg n) = O(n \lg\lg n / \lg n)$ bits

Theorem (Data structure FID): A bit-vector of length n and m ones is stored in $\lg\binom{n}{m} + \Theta\left(\frac{n \lg\lg n}{\lg n}\right)$ bits, and $rank_0$, $rank_1$, $select_0$, $select_1$ are computed in O(1) time on word RAM with word length $\Theta(\lg n)$ bits.
The data structure is constructed in $O(n)$ time.

This data structure is called FID (fully indexable dictionary) (Raman, Raman, Rao 2005).

Note: if $m << n$, it happens $\lg\binom{n}{m} < \Theta\left(\frac{n \lg\lg n}{\lg n}\right)$, and the size of index for *rank*/*select* cannot be ignored.

Sparse Vectors

- Let B be a bit-vector of length n with m ones. The information-theoretic lower bound of the size of B is $\left\lceil \lg \binom{n}{m} \right\rceil \approx m \lg \frac{n}{m} + (n-m) \lg \frac{n}{n-m}$

$$< m \lg \frac{n}{m} + m \lg e$$

- On the other hand, the size of FID is

$$\lg \binom{n}{m} + \Theta\left(\frac{n \lg \lg n}{\lg n} \right)$$

- The second term is much larger than the lower bound if $m = \mathrm{o}(n / \lg n)$.

Theorem (Data structure GV): For a bit-vector B of length $u = 2^w$ with n ones, $select_1(B, i)$ is computed in constant time using a data structure of $n(2 + w - \lfloor \lg n \rfloor) + O(\frac{n \lg \lg n}{\lg n})$ bits on $\Omega(\lg u)$-bit word-RAM.

Proof: Let $0 \le p_1 < p_2 < \ldots < p_n < u$ be the positions of ones in B. Each p_i is a w-bit integer. Let q_i be its most significant (leftmost) $z = \lfloor \lg n \rfloor$ bits and r_i be the rest.

$w = 4$	p_i	$2 = 0010$	$3 = 0011$	$8 = 1000$	$14 = 1110$
$n = 4$	q_i	00	00	10	11
$z = 2$	r_i	10	11	00	10

The lower part (r_i) is stored in an integer array L.

$n(w - z) = n(w - \lfloor \lg n \rfloor)$ bits.

The upper part (q_i) is stored using a bit-vector H.

We encode $q_1, q_2 - q_1, q_3 - q_2, \ldots, q_n - q_{n-1}$ by unary codes.

$k \geq 0$ is represented by 0^k1.

q_i	0	0	2	3
$q_i - q_{i-1}$	0	0	2	1
H	1	1	001	01

It holds $q_i = select_1(H, i) - i$. By storing H using BV, q_i is computed in O(1) time. H has n many ones and at most $q_i \leq n$ many zeros. Thus the size is at most $2n + O(n \lg \lg n / \lg n)$ bits.