Prologo

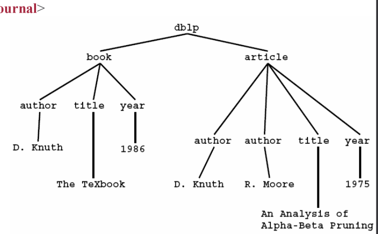
Paolo Ferragina
Dipartimento di Informatica, Università di Pisa

Pre-history of string processing !

- Collection of strings
  - Documents
  - Books
  - Emails
  - Source code
  - DNA sequences
  - …

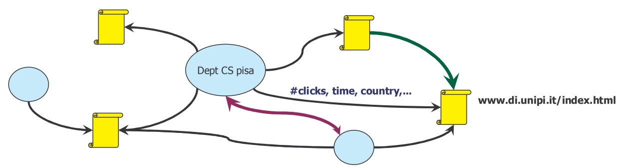
An XML excerpt

```
<dblp>
  <book>
        <author> Donald E. Knuth </author>
        <title> The TeXbook </title>
        <publisher> Addison-Wesley </publisher>
        <year> 1986 </year>
  </book>
  <article>
        <author> Donald E. Knuth </author>
        <author> Ronald W. Moore </author>
        <title> An Analysis of Alpha-Beta Pruning </title>
        <pages> 293-326 </pages>
        <year> 1975 </year>
        <journal> Artificial Intelligence </journal>
  </article>
...
</dblp>
```



size ≈ 100Mb
#leaves ≥ 7Mil for 75Mb
#internal nodes ≥ 4 Mil for 25Mb
depth ≤ 7

The Query-Log graph



- QueryLog (Yahoo! dataset, 2005)
  - #links: ≈70 Mil
  - #nodes: ≈50 Mil
  - Dictionary of URLs: 24 Mil, 56.3 avg/chars, 1.6Gb
  - Dictionary of terms: 44 Mil, 7.3 avg/chars, 307Mb
  - Dictionary of Infos: 2.6Gb

In all cases...

- Some structure: relation among items
  - Trees, (hyper-)graphs, ...
- Some data: (meta-)information about the items
  - Labels on nodes and/or edges

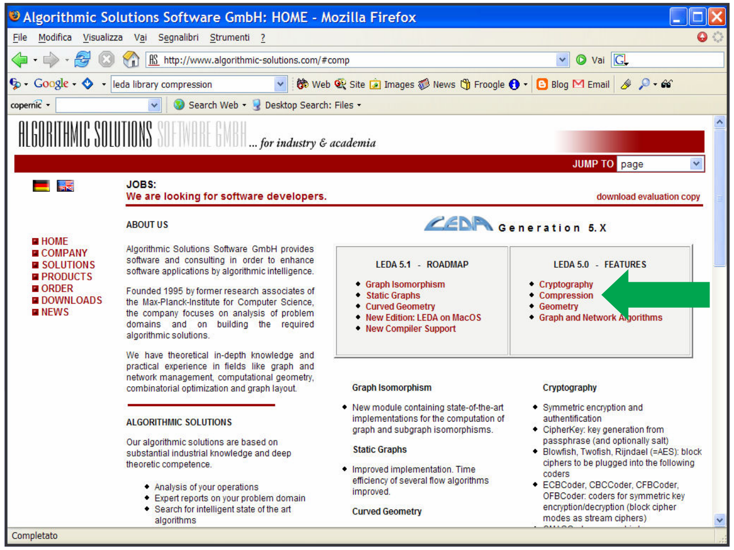
Large space (I/O, cache, compression,...)

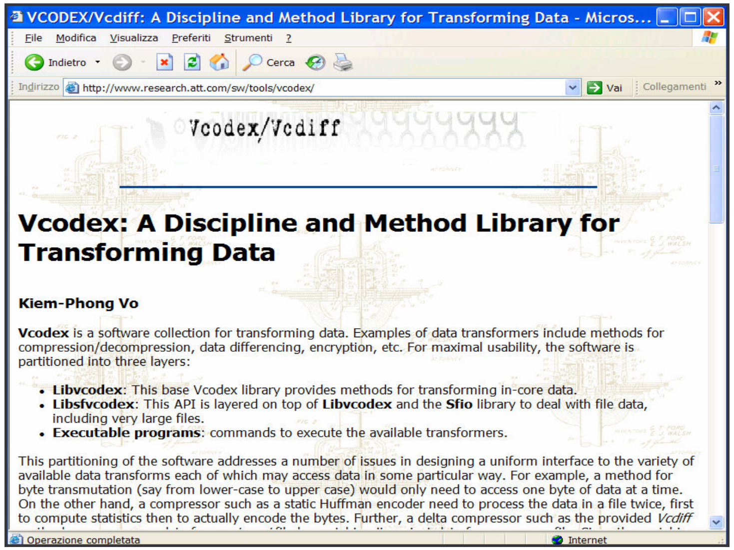
- Various operations to be supported
  - Given node u
    - Retrieve its label, Fw(u), Bw(u),...
  - Given an edge (i,j)
    - Check its existence, Retrieve its label, ...

Id ⇆ String

  - Given a string p:
    - search for all nodes/edges whose label includes p
    - search for adjacent nodes whose label equals p

Index





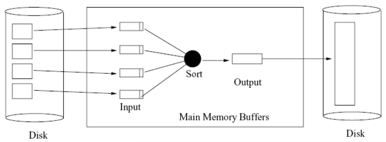
Conclusions

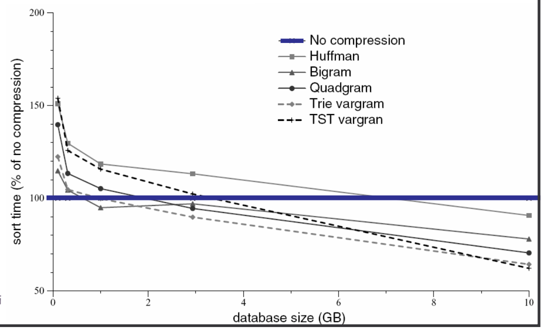
Systems should automatically compress data whenever the benefits of storing or transmitting the compressed data outweigh the costs

- It's time to "teach" systems how to do this

Virtually enlarge M

[Zobel *et al*, '07]

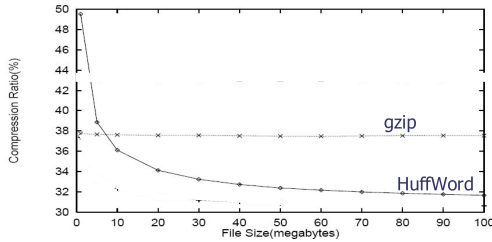




Paolo Ferragina, Università di

Do you use (z)grep?

[deMoura *et al*, '00]



- ≈1Gb data
- Grep takes 29 secs (scan the uncompressed data)
- Zgrep takes 33 secs (gunzip the data | grep)
- Cgrep takes 8 secs (scan directly the compressed)

Paolo Ferragina, Università di Pisa

IBM Research

Conclusions

Systems should automatically compress data whenever the benefits of storing or transmitting the compressed data outweigh the costs

- It's time to "teach" systems how to do this

In our lectures we are interested **not only** in the storage issue:
- Random Access
- Search

Data Compression + **Data Structures**

15 Toward Ubiquitous Compression © 2004 IBM Corporation

Seven years ago...

[now, *J. ACM* 05]

FOCS 2000

The 41st Annual Symposium on Foundations of Computer Science

Opportunistic Data Structures with Applications

P. Ferragina, G. Manzini

Nowadays several papers: theory & experiments
(see Navarro-Makinen's survey)

Paolo Ferragina, Università di Pisa

Our starting point was...

Ken Church (AT&T, 1995) said "If I compress the Suffix Array with Gzip I do not save anything. But the underlying text is compressible.... What's going on?"

Practitioners use many "squeezing heuristics" that compress data and still support fast access to them

Can we "automate" and "guarantee" the process ?

In these lectures....

Muthu's challenge!!

A path consisting of five steps
1) The problem
2) What practitioners do and why they did not use "theory"
3) What theoreticians then did
4) Experiments
5) The moral ;-)

At the end, hopefully, you'll bring at home:
- ✓ Algorithmic tools to compress & index data
- ✓ Data aware measures to evaluate them
- ✓ Algorithmic reductions: Theorists and practitioners love them!
- ✓ **No ultimate receipts !!**

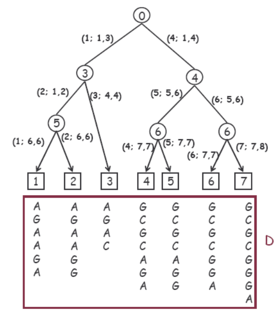
String Storage

Paolo Ferragina
Dipartimento di Informatica, Università di Pisa

A basic problem

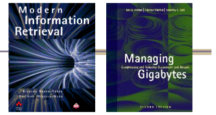
Given a dictionary D of strings, of variable length, compress them in a way that we can efficiently support Id ⇆ string

- Hash Table
  - Need D to avoid *false*-positive and for id→string
- (Minimal) ordered perfect hashing
  - Need D for id→string, or check
- (Compacted) Trie
  - Need D for edge match



Yet the dictionary D needs to be stored
- its space is not negligible
- I/O- or cache-misses in retrieval

Front-coding



Practitioners use the following approach:
- Sort the dictionary strings
- Strip-off the shared prefixes [e.g. host reversal?]
- Introduce some bucketing, to ensure fast random access

uk-2002 crawl ≈250Mb

```
http://checkmate.com/All_Natural/
http://checkmate.com/All_Natural/Applied.html
http://checkmate.com/All_Natural/Aroma.html
http://checkmate.com/All_Natural/Aroma1.html
http://checkmate.com/All_Natural/Aromatic_Art.html
http://checkmate.com/All_Natural/Ayate.html
http://checkmate.com/All_Natural/Ayer_Soap.html
http://checkmate.com/All_Natural/Ayurvedic_Soap.html
http://checkmate.com/All_Natural/Bath_Salt_Bulk.html
http://checkmate.com/All_Natural/Bath_Salts.html
http://checkmate.com/All/Essence_Oils.html
http://checkmate.com/All/Mineral_Bath_Crystals.html
http://checkmate.com/All/Mineral_Bath_Salt.html
http://checkmate.com/All/Mineral_Cream.html

http://checkmate.com/All/Natural/Washcloth.html
...
```

30÷35%

```
0  http://checkmate.com/All_Natural/
33 Applied.html
34 roma.html
38 1.html
38 tic_Art.html
34 yate.html
35
35
33
42
25
25
38
33
0  http://checkmate.com/All/Natural/Washcloth.html...
```

Do we need bucketing ?
→ Experimental tuning ←

gzip ≈ 12%
Be back on this later on!

Locality-preserving FC

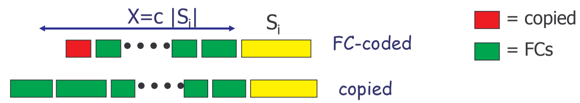
Bender *et al*, 2006

Drop bucketing + optimal string decompression
- Compress D up to $(1+\varepsilon)$ FC(D) bits
- Decompress any string S in $1+|S|/\varepsilon$ time

A simple incremental encoding algorithm [where $\varepsilon = 2/(c-2)$]

I. Assume to have FC($S_1,\ldots,S_{i-1}$)

II. Given S_i, we proceed backward for X=c $|S_i|$ chars in FC
   - Two cases

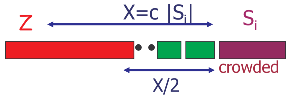


Locality-preserving FC

Bender *et al*, 2006

A simple incremental encoding algorithm [where $\varepsilon = 2/(c-2)$]
- Assume to have FC($S_1,\ldots,S_{i-1}$)
- Given S_i, we proceed backward for X=c $|S_i|$ chars in FC
  - If S_i is decoded, then we add FC(S_i) else we add S_i

➢ Decoding is unaffected!!



---- Space occupancy (sketch)
- **FC-encoded** strings are OK!
- Partition the **copied** strings in (un)crowded
- Let S_i be crowded, and Z its preceding copied string:
  - $|Z| \geq X/2 \geq (c/2)\ |S_i|$
  - Hence, length of crowded strings decreases **geometrically** !!
- Consider **chains** of copied: |uncrowd crowd*| ≤ (c/c-2) |uncrowd|
- Charge chain-cost to X/2 = (c/2) |uncrowd| chars before uncrowd (ie FC-chars)

Random access to LPFC

We call *C* the LPFC-string, n = #strings in *C*, m = total length of *C*

How do we Random Access the compressed *C* ?
- Get(i): return the position of the i-th string in *C* (id→string)
- Previous(j), Next(j): return the position of the string preceding or following character *C*[j]

Classical answers ;-)
- Pointers to positions of copied-strings in *C*
  - Space is *O*(n log m) bits
  - Access time is *O*(1) + *O*(|S|/ε)
- Some form of bucketing... Trade-off
  - Space is *O*((n/b) log m) bits
  - Access time is *O*(b) + *O*(|S|/ε)

No trade-off !

Re-phrasing our problem

C is the LPFC-string, **n** = #strings in *C*, **m** = total length of *C*

Support the following operations on *C*:
- *Get(i)*: return the position of the i-th string in *C*
- *Previous(j), Next(j)*: return the position of the string prec/following *C[j]*

Proper integer encodings

C= 0 http://checkmate.com/All_Natural/ 33 Applied.html 34 roma.html 38 1.html 38 tic_Art.html ....
B= 1 000000000000000000000000000 10 0000000000 10 00000000 10 00000 10 000000000 ....

Rank$_1$(36) = 2 Select$_1$(4) = 51

- Rank$_1$(x) = number of 1 in B[1,x]
- Select$_1$(y) = position of the y-th 1 in B

- *Get(i)* = Select$_1$(i)
- *Previous(j)* = Select$_1$(Rank$_1$(j) -1)
- *Next(j)* = Select$_1$(Rank$_1$(j) +1)

Look at them as pointerless data structures

A basic problem !

Jacobson, '89

Select$_1$(3) = 8

B 00101001010101011111110000011010101010101111000....

Rank$_1$(7) = 4

m = |B|
n = #1s

- Rank$_b$(i) = number of b in B[1,i]
- Select$_b$(i) = position of the i-th b in B

- Considering b=1 is enough: Select$_0$(B,i) = ... Select$_1$ is sim... #1 in B_0 and B_1 ≤ min{m-n,n} $|B_0|+|B_1| = m$
  - Rank$_0$(i)= i – Rank$_1$(i)
  - Any Select → Rank$_1$ and Select$_1$ over two binary arrays:
    - B = 0 1 0 0 0 0 1 1 1 0 0 1 0 0 1 1 1 1 1 1 1 0
    - B_0 = 1 0 0 0 1 0 1 0 1 1, $|B_0|$= m-n
    - B_1 = 1 0 0 1 1 0 0 0 0 0 0 1 , $|B_1|$= n

A basic problem !

Jacobson, '89

Select$_1$(3) = 8

B 00101001010101011111110000011010101010101111000....

Rank$_1$(7) = 4

m = |B|
n = #1s

- Rank$_1$(i) = number of 1s in B[1,i]
- Select$_1$(i) = position of the i-th 1 in B

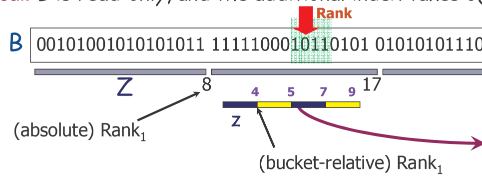
- Given an integer *set*, we set B as its characteristic vector
  - pred(x) = Select$_1$(Rank$_1$(x-1))

LBs can be inherited
[Patrascu-Thorup, '06]

The Bit-Vector Index

m = |B|
n = #1s

Goal. B is *read-only*, and the additional index takes o(m) bits.



block	pos	#1
0000	1	0
....	...	...
1011	2	1
....		

- Setting Z = poly(log m) and z=(1/2) log m:
  - Space is |B| + (m/Z) log m + (m/z) log Z + o(m)
    - m + O(m loglog m / log m) bits
  - Rank time is O(1)
  - The term o(m) is crucial in practice

Ω ??

The Bit-Vector Index

m = |B|
n = #1s

B 00101001010101011111111000001101010101010111000....

size r ⇆ k consecutive 1s

- Sparse case: If $r > k^2$ store explicitly the position of the k 1s
- Dense case: $k \le r \le k^2$, recurse... One level is enough!!

... still need a table of size o(m).

- Setting $k \approx \text{polylog } m$
  - Space is m + o(m), and B is not touched!
  - Select time is O(1)

LPFC + RankSelect
takes [1+o(1)] extra bits per FC-char

There exists a Bit-Vector Index taking |B| + o(|B|) bits and constant time for Rank/Select.
B is read-only!

Compressed String Storage

Paolo Ferragina
Dipartimento di Informatica, Università di Pisa

FC versus Gzip

a	a	c	a	a	c	b	a	b	a	b	c	b	a	c

Dictionary
(all substrings starting here)

<6,3,c>

Two features:

- Repetitiveness is deployed at any position
- Window is used for (practical) computational reasons

On the previous dataset of URLs (ie. uk-2002)

- FC achieves >30%
- Gzip achieves 12%
- PPM achieves 7%

No random access to substrings

May be combine the best of the two worlds?

The emprirical entropy H_0

$$H_0(S) = -\sum_i (m_i/m) \log_2 (m_i/m)$$

Frequency in S of the i-th symbol

- ❖ $m H_0(S)$ is the best you can hope for a memoryless compressor
- ❖ We know that Huffman or Arithmetic come close to this bound

H_0 cannot distinguish between A^xB^y and a random with x A and y B

We get a better compression using a codeword that depends on the k symbols preceding the one to be compressed (**context**)

The empirical entropy H_k

✓ Compress S up to $H_k(S)$

≅ compress all $S[\omega]$ up to their H_0

Use Huffman or Arithmetic

$$H_k(S) = (1/|S|) \sum_{|\omega|=k} |S[\omega]| \; H_0(S[\omega])$$

❖ $S[\omega]$ = string of symbols that follow the substring ω in S

Example: Given S = "mississippi", we have S["is"] = ss

Follow ≈ Precede

How much is "operational" ?

Entropy-bounded string storage [Ferragina-Venturini, '07]

Goal. Given a string S[1,m] drawn from an alphabet Σ of size σ

- encode S within $m H_k(S) + o(m \log \sigma)$ bits, with k ≤ …
- extract any substring of L symbols in optimal $\Theta(L / \log m)$ time

This encoding fully-replaces S in the RAM model !

Two corollaries

- Compressed Rank/Select data structures
  - B was read-only in the simplest R/S scheme
  - We get $|B| H_k(B) + o(|B|)$ bits and R/S in O(1) time
- Compressed Front-Coding + random access
  - Promising: FC+Gzip saves 16% over gzip on uk-2002

The storage scheme

- $b = ½ \log_\sigma m$
- # blocks = $m/b = O(m / \log_\sigma m)$
- #distinct blocks = $O(\sigma^b) = O(m^{½})$

Decoding is easy:

- R/S on B to determine cw position in V
- Retrieve cw from V
- Decoded block is $T[2^{len(cw)} + cw]$

cw	T
ε	α
0	β
1	δ
00	γ
01	…
10	…
11	…
000	…

frequency

S	α	β	α	δ	γ	β	α	
V	--	0	--	1	00	0 …		
B	1	01	1	01	001	01 …		

b

T+V+B take |V| + o(|S| log σ) bits

|B| ≤ |S| log σ, #1 in B = #blocks = o(|S|)

Bounding |V| in terms of $H_k(S)$

- Introduce the statistical encoder $E_k(S)$:
  - Compute F(i)= freq of S[i] within its k-th order context S[i-k,i-1]
  - Encode every block B[1,b] of S as follows
    1) Write B[1,k] explicitly
    2) Encode B[k+1, b] by Arithmetic using the k-th order frequencies

  » Some algebra→ $(m/b) * (k \log \sigma) + m H_k(S) + 2 (m/b)$ bits

  $|S| H_k(S) + o(|S| \log \sigma)$

- $E_k(S)$ is worse than our encoding V
  - E_k assigns unique cw to blocks
  - These cw are a **subset** of $\{0,1\}^*$
  - ➢ Our cw are the **shortest** of $\{0,1\}^*$

Golden rule of data compression

$|V| \le |E_k(S)| \le |S| H_k(S) + o(|S| \log \sigma)$ bits

Part #2: Take-home Msg

- Given a binary string B, we can
  - Store B in $|B|\ H_k(B) + o(|B|)$ bits
  - Support Rank & Select in constant time
  - Access any substring of B in optimal time

Pointer-less data structure

- Given a string S on Σ, we can
  - Store S in $|S|\ H_k(S) + o(|S| \log |\Sigma|)$ bits, where $k \le \alpha \log_{|\Sigma|} |S|$
  - Access any substring of S in optimal time

Always better than S on RAM

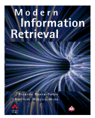
Experimentally
- 10^7 select / sec
- 10^6 rank / sec

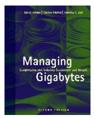
(Compressed) String Indexing

Paolo Ferragina
Dipartimento di Informatica, Università di Pisa

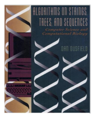
What do we mean by "Indexing" ?

- ❑ Word-based indexes, here a notion of "word" must be devised !
  - » Inverted files, Signature files, Bitmaps.





- ❑ Full-text indexes, no constraint on text and queries !
  - » Suffix Array, Suffix tree, String B-tree,...



The Problem

Given a text T, we wish to devise a (compressed) representation for T that efficiently supports the following operations:
- ✓ Count(P): How many times string P occurs in T as a substring?
- ✓ Locate(P): List the positions of the occurrences of P in T ?
- ✓ Visualize(i,j): Print T[i,j]

☑ Time-efficient solutions, but not compressed
- ❖ Suffix Arrays, Suffix Trees, ...
- ❖ ...many others...

☑ Space-efficient solutions, but not time efficient
- ❖ ZGrep: uncompress and then grep it
- ❖ CGrep, NGrep: pattern-matching over compressed text

The Suffix Array

Prop 1. All suffixes of T having prefix P are contiguous.
Prop 2. Starting position is the lexicographic one of P.

$\Theta(N^2)$ space

T = mississippi#   (5; suffix pointer)

SA	SUF(T)
12	#
11	i#
8	ippi#
5	issippi#
2	ississippi#
1	mississippi#
10	pi#
9	ppi#
7	sippi#
4	sissippi#
6	ssippi#
3	ssissippi#

P=si

Suffix Array
- SA: $\Theta(N \log_2 N)$ bits
- Text T: N chars

⇨ In practice, a total of 5N bytes

Paolo Ferragina, Università di Pisa

Searching a pattern

Indirected binary search on SA: O(p) time per suffix cmp

T = mississippi#

SA: 12, 11, 8, 5, 2, 1, 10, 9, 7, 4, 6, 3

P is larger

P = si

2 accesses per step

Paolo Ferragina, Università di Pisa

Searching a pattern

Indirected binary search on SA: O(p) time per suffix cmp

T = mississippi#

SA: 12, 11, 8, 5, 2, 1, 10, 9, 7, 4, 6, 3

P is smaller

P = si

Suffix Array search
- $O(\log_2 N)$ binary-search steps
- Each step takes O(p) char cmp

⇨ overall, $O(p \log_2 N)$ time

+ [Manber-Myers, '90]

|Σ| [Cole et al, '06]

Paolo Ferragina, Università di Pisa

Listing of the occurrences

T = mississippi# (4, 7)

SA: 12, 11, 8, 5, 2, 1, 10, 9, 7, 4, 6, 3

Suffix Array search
- listing takes O (occ) time

where # < Σ < $

P# = si#

occ=2 (7: sippi, 4: sissippi)

P$ = si$

Paolo Ferragina, Università di Pisa

Text mining

Lcp[1,N-1] stores the LCP length between suffixes adjacent in SA

T = m i s s i s s i p p i #
1 2 3 4 5 6 7 8 9 10 11 12

Lcp	SA
	12
0	11
0	8
1	5
4	2
0	1
0	10
1	9
0	7
2	4
1	6
3	3

issippi
ississippi

- Does it exist a repeated substring of length ≥ L ?
  - Search for Lcp[i] ≥ L
- Does it exist a substring of length ≥ L occurring ≥ C times ?
  - Search for Lcp[i,i+C-1] whose entries are ≥ L

Paolo Ferragina, Università di Pisa

What about space occupancy?

T = mississippi#

SA: 12, 11, 8, 5, 2, 1, 10, 9, 7, 4, 6, 3

SA + T take $\Theta(N \log_2 N)$ bits

Very far

Do we need such an amount ?

1) # permutations on {1, 2, ..., N} = N!

2) SA cannot be any permutation of {1,...,N}

3) #SA ⇆ # texts = $|\Sigma|^N$

⇨ LB from #texts = $\Omega(N \log |\Sigma|)$ bits

⇨ LB from compression = $\Omega(N H_k(T))$ bits

Paolo Ferragina, Università di Pisa

An elegant mathematical tool

Paolo Ferragina
Dipartimento di Informatica, Università di Pisa

Paolo Ferragina, Università di Pisa

The Burrows-Wheeler Transform (1994)

Take the text T = mississippi#

```
mississippi#
ississippi#m
ssissippi#mi
sissippi#mis
issippi#miss
ssippi#missi
sippi#missis
ippi#mississ
ppi#mississi
pi#mississip
i#mississipp
#mississippi
```

Sort the rows

```
F          L
#mississipp i
i#mississip p
ippi#missis s
issippi#mis s
ississippi# m
mississippi #
pi#mississi p
ppi#mississ i
sippi#missi s
sissippi#mi s
ssippi#miss i
ssissippi#m i
```

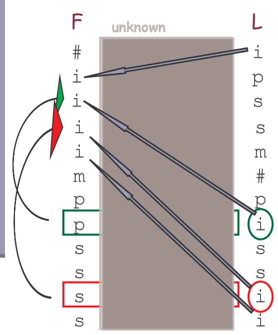
T

Paolo Ferragina, Università di Pisa

A famous example

final char (L)	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set $L[i]$ to be the
i	n turn, set $R[i]$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector $R
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with ch appear in the {\em same order
i	n with ch. In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell~\cite{bell}.

A useful tool: L → F mapping

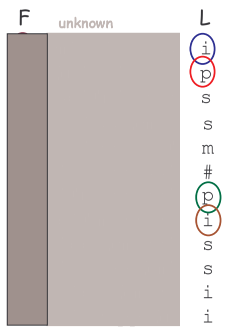


How do we map L's onto F's chars ?

... Need to distinguish equal chars in F...

- Take two equal L's chars
- Rotate rightward their rows
- Same relative order !!

The BWT is invertible



Two key properties:

1. LF-array maps L's to F's chars
2. L[i] precedes F[i] in T

Reconstruct T backward:

T = ... ippi #

```
InvertBWT(L)
Compute LF[0,n-1];
r = 0; i = n;
while (i>0) {
   T[i] = L[r];
   r = LF[r]; i--;
   }
```


How to compute the BWT ?

SA	BWT matrix	L
12	#mississipp	i
11	i#mississip	p
8	ippi#missis	s
5	issippi#mis	s
2	ississippi#	m
1	mississippi	#
10	pi#mississi	p
9	ppi#mississ	i
7	sippi#missi	s
4	sissippi#mi	s
6	ssippi#miss	i
3	ssissippi#m	i

Role of #

We said that: L[i] precedes F[i] in T

L[3] = T[7] = T[SA[3] – 1]

Given SA, we have L[i] = T[SA[i]-1]

Elegant but inefficient

Comparison_Based_Construction(char $*T$, int n, char $**SA$)
{ for($i = 0$; $i < n$; i++) $SA[i] = T + i$;
qsort(SA, n, sizeof(char *), Suffix_cmp); }

Suffix_cmp(char **p, char **q){ return strcmp(*p,*q); }

Obvious inefficiencies:
- $O(n^3)$ time in the worst-case
- $O(n^2)$ cache misses or I/O faults

Compressing L seems promising...

final char (L)	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set $L[i]$ to be the
i	n turn, set $R[i]$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector $R
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with ch appear in the {\em same order
i	n with ch. In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell~\cite{bell}.

Key observation:
- L is locally homogeneous

⇨ L is highly compressible

Algorithm Bzip :
1. Move-to-Front coding of L
2. Run-Length coding
3. Statistical coder

☑ Bzip vs. Gzip: 20% vs. 33%, but it is slower in (de)compression !

An encoding example

T = mississippimississippimississippi

L = ipppssssssmmmii#pppiiissssssiiiiii

Mtf = 020030000030030200300300000100000

Mtf = 030040000040040300400400000200000

Bin(6)=110, Wheeler's code

RLE0 = 02131031302131310110

\# at 16

Alphabet |Σ|+1

Arithmetic/Huffman su |Σ|+1 simboli.....

Why it works...

final char (L)	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set $L[i]$ to be the
i	n turn, set $R[i]$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector $R
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with ch appear in the {\em same order
i	n with ch. In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell~\cite{bell}.

Key observation:
- L is locally homogeneous

⇨ L is highly compressible

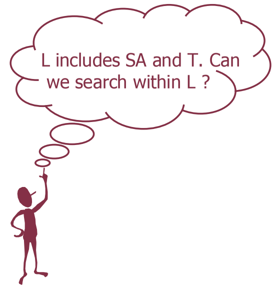
Each piece ⇆ a context

Compress pieces up to their H_0, we achieve $H_k(T)$

MTF + RLE avoids the need to partition BWT

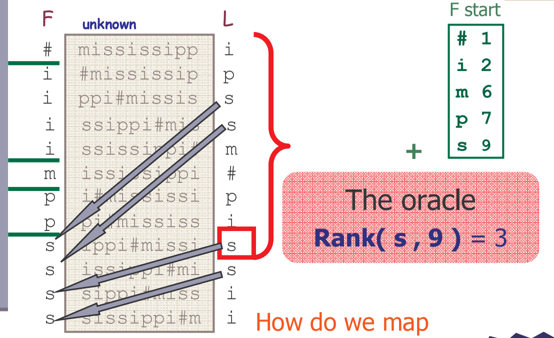
Be back on indexing: BWT ⇆ SA

SA	BWT matrix	L
12	#mississipp	i
11	i#mississip	p
8	ippi#missis	s
5	issippi#mis	s
2	ississippi#	m
1	mississippi	#
10	pi#mississi	p
9	ppi#mississ	i
7	sippi#missi	s
4	sissippi#mi	s
6	ssippi#miss	i
3	ssissippi#m	i



Implement the LF-mapping

[Ferragina-Manzini]



How do we map L[9] → F[11]

We need Generalized R&S

Rank and Select on strings

☑ If Σ is small (i.e. constant)

- ❖ Build binary Rank data structure per symbol of Σ
- ✓ Rank takes O(1) time and entropy-bounded space

☑ If Σ is large (words ?) [Grossi-Gupta-Vitter, '03]

- ❖ Need a smarter solution: Wavelet Tree data structure

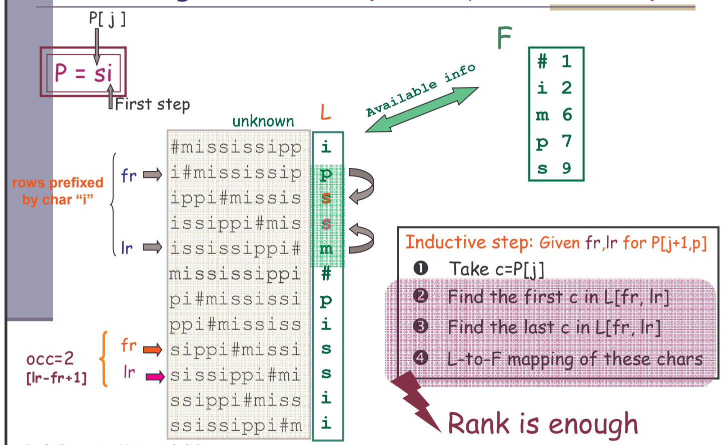
Another step of reduction:

» Reduce Rank&Select over arbitrary strings

... to Rank&Select over binary strings

Binary R/S are key tools
» tons of papers «

Substring search in T (Count the pattern occurrences)



Inductive step: Given fr,lr for P[j+1,p]

1. Take c=P[j]
2. Find the first c in L[fr, lr]
3. Find the last c in L[fr, lr]
4. L-to-F mapping of these chars

Rank is enough

The FM-index

[Ferragina-Manzini, Focs '00]
[Ferragina-Manzini, JACM '05]

The result (on small alphabets):

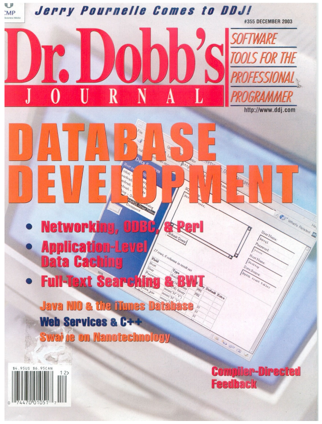
- ✓ Count(P): O(p) time
- ✓ Locate(P): O(occ $\log^{1+\varepsilon} N$) time
- ✓ Visualize(i,i+L): O(L + $\log^{1+\varepsilon} N$) time
- ✓ Space occupancy: O(N $\mathcal{H}_k(T)$) + o(N) bits → o(N) if T compressible

Index does not depend on k
bound holds for all k, simultaneously

New concept: The FM-index is an ***opportunistic*** data structure

Survey of Navarro-Makinen contains many compressed index variants

Is this a technological breakthrough ?

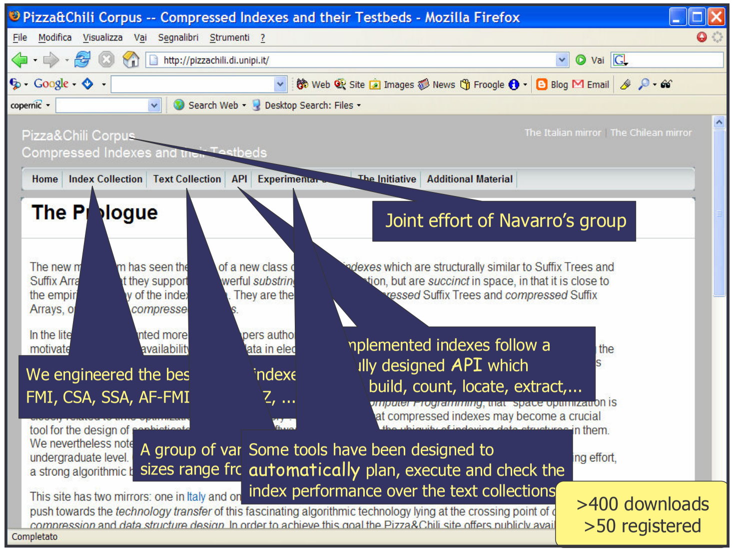


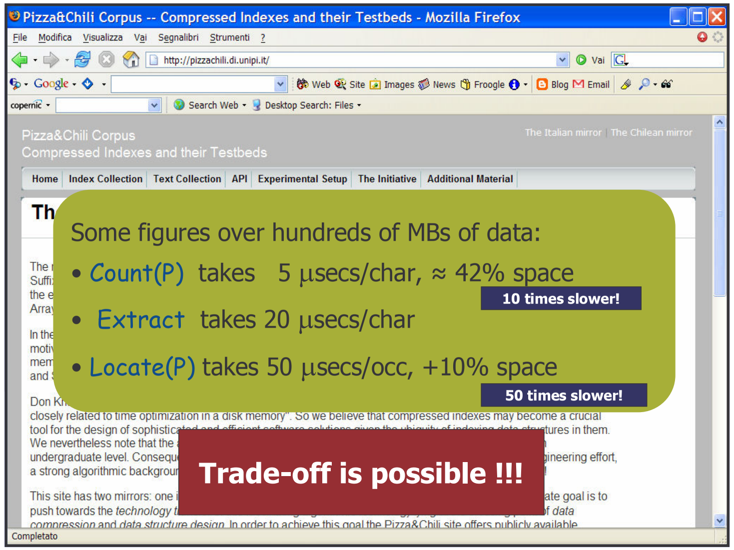
[December 2003] [January 2005]

The question then was...

How to turn these challenging and mature theoretical achievements into a technological breakthrought ?

- ☑ Engineered implementations
- ☑ Flexible API to allow reuse and development
- ☑ Framework for extensive testing





We need your applications...

Part #5: Take-home msg...

Data type: Text

This is a powerful paradigm to design compressed indexes:

1. Transform the input in few arrays
2. Index (+ Compress) the arrays to support rank/select ops

- Compression and I/Os
- Compression and query distribution/flow
- Other data types: Labeled Trees 2D

(Compressed) Tree Indexing

Paolo Ferragina
Dipartimento di Informatica, Università di Pisa

Where we are...

A data structure is "opportunistic" if it indexes a text T within compressed space and supports three kinds of queries:

- ✓ Count(P): Count the occurrences of P occurs in T
- ✓ Locate(P): List the occurrences of P in T
- ✓ Display(i,j): Print T[i,j]

☑ Key tools: Burrows-Wheeler Transform + Suffix Array

☑ Key idea: reduce P's *queries* to few rank/select queries on BWT(T)

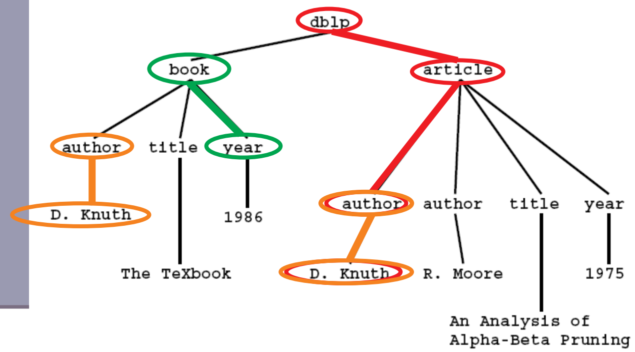
☑ Space complexity: function the *k*-th order empirical entropy of T

Another data format: XML

[W3C '98]

```
<dblp>
    <book>
        <author> Donald E. Knuth </author>
        <title> The TeXbook </title>
        <publisher> Addison-Wesley </publisher>
        <year> 1986 </year>
    </book>
    <article>
        <author> Donald E. Knuth </author>
        <author> Ronald W. Moore </author>
        <title> An Analysis of Alpha-Beta Pruning </title>
        <pages> 293-326 </pages>
        <year> 1975 </year>
        <volume> 6 </volume>
        <journal> Artificial Intelligence </journal>
    </article>
...
</dblp>
```


A tree interpretation...

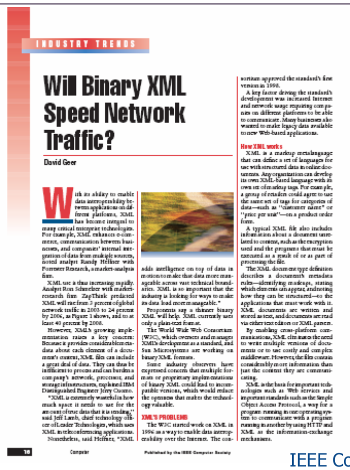


☑ XML document exploration ≡ Tree navigation

☑ XML document search ≡ Labeled subpath searches

Subset of XPath [W3C]

A key concern: Verbosity...


INDUSTRY TRENDS

Will Binary XML Speed Network Traffic?

David Geer

IEEE Computer, April 2005

The problem, in practice...

We wish to devise a (compressed) representation for T that efficiently supports the following operations:

- ✓ Navigational operations: parent(u), child(u, i), child(u, i, c)
- ✓ Subpath searches over a sequence of k labels
- ✓ Content searches: subpath search + substring

☑ XML-aware compressors (like XMill, XmlPpm, ScmPpm,...) need the **whole decompression** for navigation and search

☑ XML-queriable compressors (like XPress, XGrind, XQzip,...) achieve **poor compression** and need the **scan** of the whole (compressed) file

Theory?

XML-native search engines need this tool as a core block for query optimization and (compressed) storage of information

A transform for labeled trees

[Ferragina *et al*, 2005]

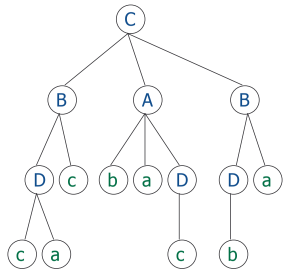
XBW-transform on trees ⇆ BW-transform on strings

The **XBW-transform** linearizes T in **2 arrays** such that:

- ☑ the compression of T *reduces to* the compression of these two arrays (e.g. gzip, bzip2, ppm,...)
- ☑ the indexing of T *reduces to* implement generalized rank/select over these two arrays

Rank&Select are again crucial

The XBW-Transform



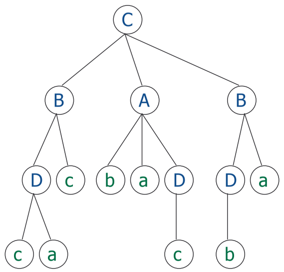
S_α	S_π
C	ε
B	C
D	B C
c	D B C
a	D B C
c	B C
A	C
b	A C
a	A C
D	A C
c	D A C
B	C
D	B C
b	D B C
a	B C

Step 1.
Visit the tree in pre-order.
For each node, write down its label and the labels on its upward path

Permutation of tree nodes

upward labeled paths

The XBW-Transform

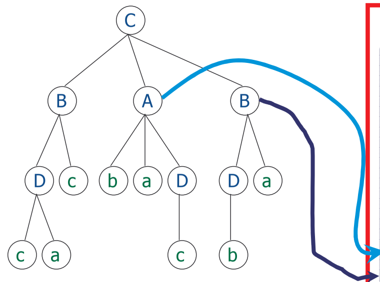


S_α	S_π
C	ε
b	A C
a	A C
D	A C
D	B C
c	B C
D	B C
a	B C
B	C
A	C
B	C
c	D A C
c	D B C
a	D B C
b	D B C

Step 2.
Stably sort according to S_π

upward labeled paths

The XBW-Transform



S_{last}	S_α	S_π
1	C	ε
0	b	A C
0	a	A C
1	D	A C
0	D	B C
1	c	B C
0	D	B C
1	a	B C
0	B	C
0	A	C
1	B	C
1	c	D A C
0	c	D B C
1	a	D B C
1	b	D B C

XBW

XBW can be built and inverted in *optimal* O(t) time

Key fact
Nodes correspond to items in <S_{last},S_α>
rows corresponding to *last* children

XBW takes *optimal* t log |Σ| + t bits

XBW is highly compressible

S_{last} S_α S_π

Donald Knuth /author/article/dblp
Kurt Mehlhorn /author/article/dblp
Kurt Mehlhorn /author/article/dblp
... ...
Kurt Mehlhorn /author/book/dblp
John Kleinberg /author/book/dblp

Theoretically, we could extend the definition of H_k to labeled trees by taking as k-context of a node its leading path of k-length

(related to Markov random fields over trees)

137-157 /pages/article/dblp
... ...
ACM Press /publisher/journal/dblp
ACM Press /publisher/journal/dblp
IEEE Press /publisher/journal/dblp
... ...
1977 /year/book/dblp
2000 /year/journal/dblp
... ...

XBW

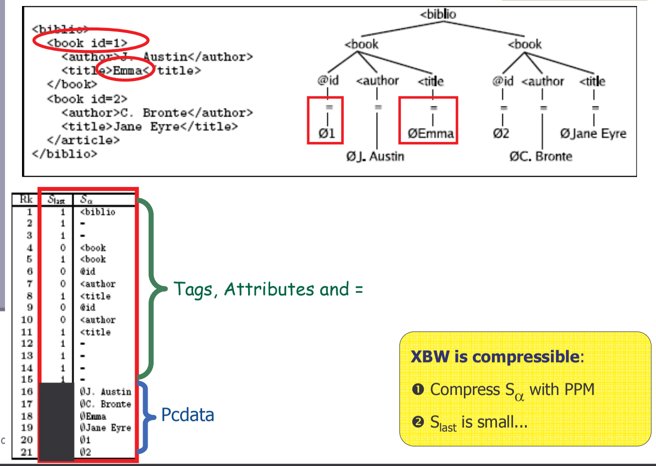
XBW is compressible:

❶ S_α is locally homogeneous

❷ S_{last} has some structure and is small

Paolo Ferragina, Università di Pisa

XBzip – a simple XML compressor



XBW is compressible:

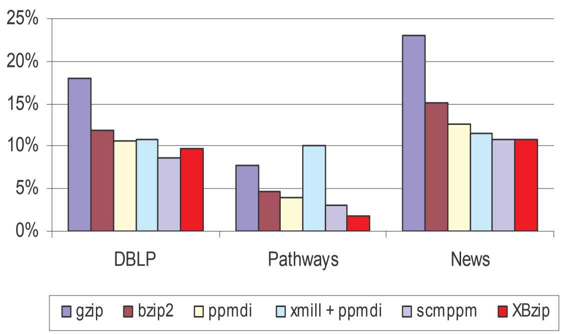
❶ Compress S_α with PPM

❷ S_{last} is small...

Paolo Ferragina, Università di Pisa

XBzip = XBW + PPM

[Ferragina *et al*, 2006]

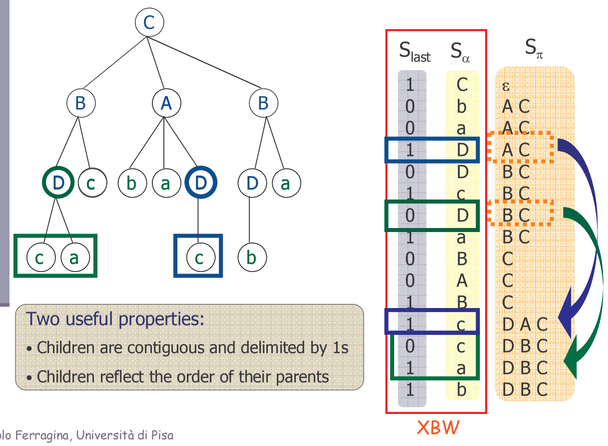


String compressors are not so bad: within 5%

Deploy huge literature on string compression

Paolo Ferragina, Università di Pisa

Some structural properties

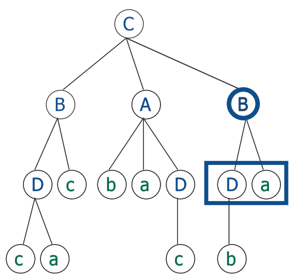


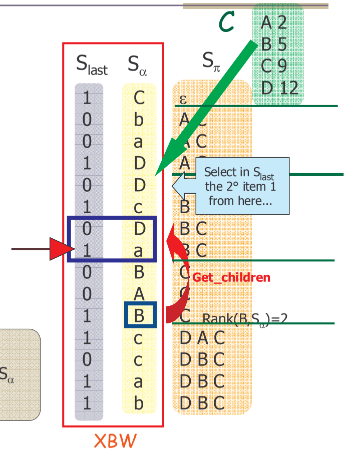
Two useful properties:

- Children are contiguous and delimited by 1s
- Children reflect the order of their parents

Paolo Ferragina, Università di Pisa

XBW is navigational



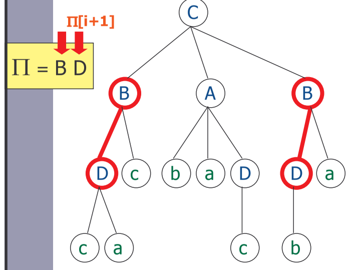


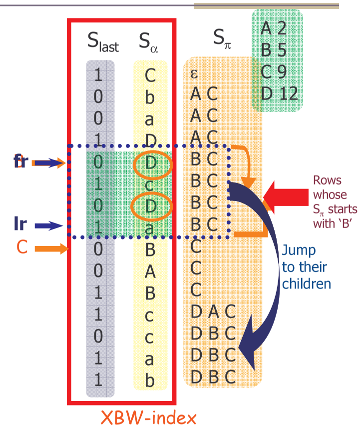
XBW is navigational:

- Rank-Select data structures on S_{last} and S_α
- The array C of $|\Sigma|$ integers

Paolo Ferragina, Università di Pisa

Subpath search in XBW





Inductive step:

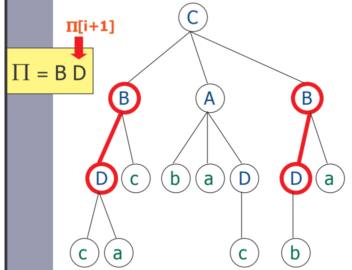
❶ Pick the next char in Π[i+1], i.e. 'D'

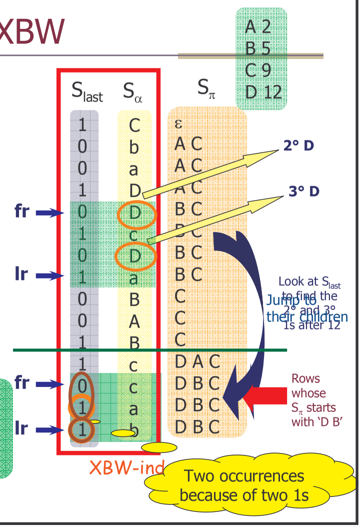
❷ Search for the *first* and *last* 'D' in S_α[fr,lr]

❸ Jump to their *children*

Paolo Ferragina, Università di Pisa

Subpath search in XBW





Inductive step:

XBW indexing [reduction to string indexing]

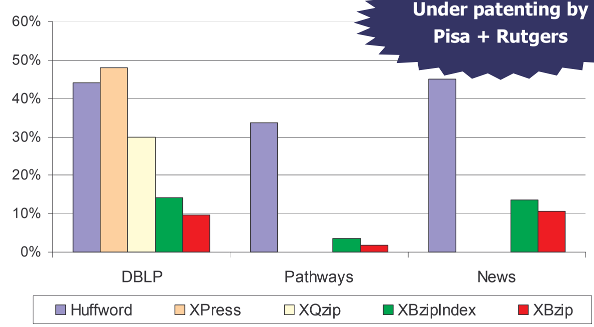
Rank and Select data structures are enough to navigate and search T

Paolo Ferragina, Università di Pisa

XBzipIndex: XBW + FM-index

[Ferragina *et al*, 2006]

Under patenting by Pisa + Rutgers



DBLP: 1.75 bytes/node, Pathways: 0.31 bytes/node, News: 3.91 bytes/node

Upto 36% improvement in compression ratio
Query (counting) time ≅ 8 ms, Navigation time ≅ 3 ms

Paolo Ferragina, Università di Pisa

Part #6: Take-home msg...

Data type: Text

This is a powerful paradigm to design compres

1. Transform the input in few arrays
2. Index (+ Compress) the arrays to support

More ops

More experiments and Applications

Other data types: 2D, Labeled graphs

I/O issues

Paolo Ferragina
Dipartimento di Informatica, Università di Pisa

What about I/O-issues ?

B-tree is ubiquitous in large-scale applications:

- Atomic keys: integers, reals, ...
- Prefix B-tree: bounded length keys (≤ 255 chars)

String B-tree = B-tree + Patricia Trie [Ferragina-Grossi, 95]

- Unbounded length keys
- I/O-optimal prefix searches
- Efficient string updates
- Guaranteed optimal page fill ratio

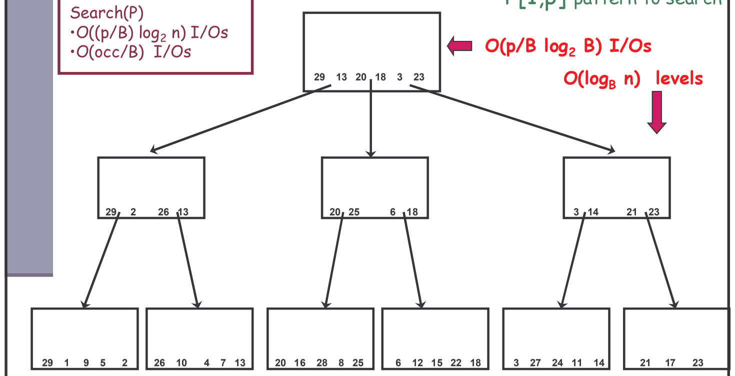
Variants for various models

They are not opportunistic ☹ [Bender et al → FC]

The B-tree

Search(P)
- $O((p/B) \log_2 n)$ I/Os
- $O(occ/B)$ I/Os

P[1,p] pattern to search

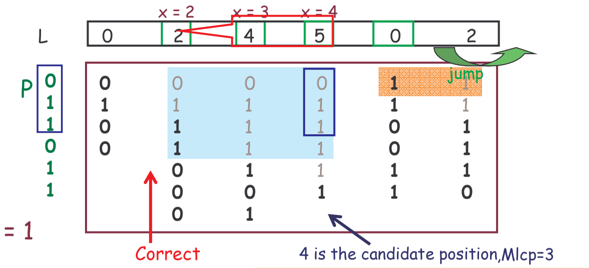


On small sets...

[Ferguson, 92]

Scan FC(D) :

- If P[L[x]]=1, then { x++ } else { *jump*; }
- Compare P and S[x] → Max_lcp
- If P[Max_lcp+1] = 0 go left, else go right, until L[] ≤ Max_lcp



Init x = 1

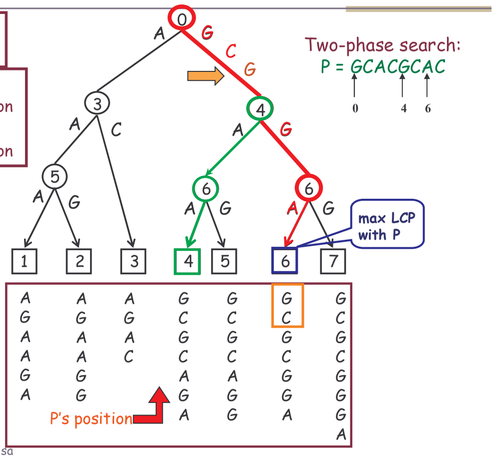
Time is #D + |P| ≤ |FC(D)|
Just S[x] needs to be decoded !!

On larger sets...

Patricia Trie
Space = Θ(#D) words

Search(P):
- Phase 1: tree navigation
- Phase 2: Compute LCP
- Phase 3: tree navigation

1 string checked + Space PT ≈ #D



The String B-tree

Succinct PT → smaller height in practice
...not opportunistic: Ω(#D log |D|) bits

Search(P)
- O((p/B) $\log_B n$) I/Os
- O(occ/B) I/Os

It is dynamic...