COMP9319 Web Data Compression and Search

ISX & XBW;
(Web) Graph Compression

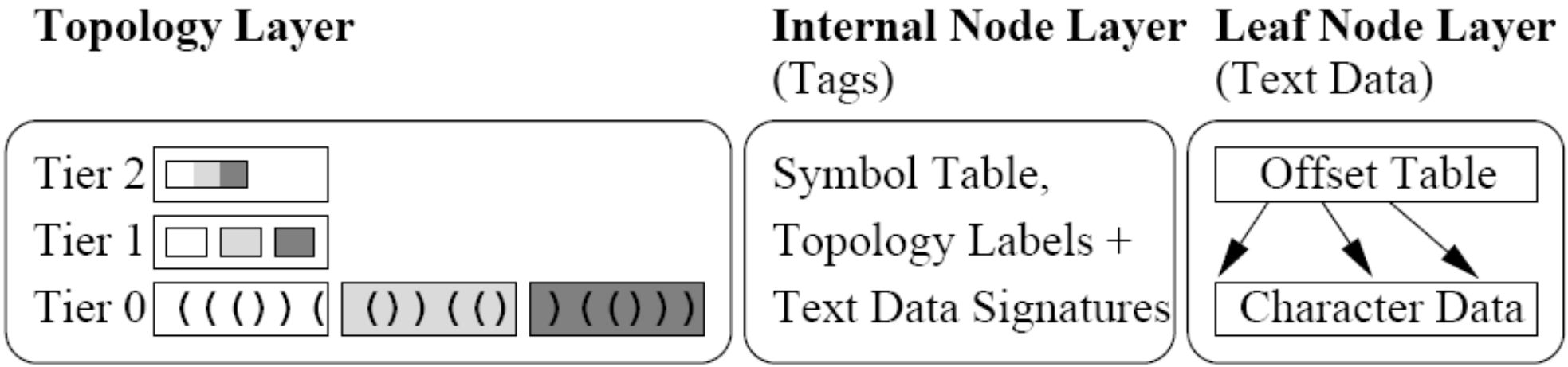
ISX Requirements

1. Space does matter for many applications
2. Generally reducing space improves cache locality
3. Indirection is expensive
4. Support fast navigations
5. Support fast insertion and deletion
6. Support efficient joins
7. Separate topology, text and schema

ISX Goal

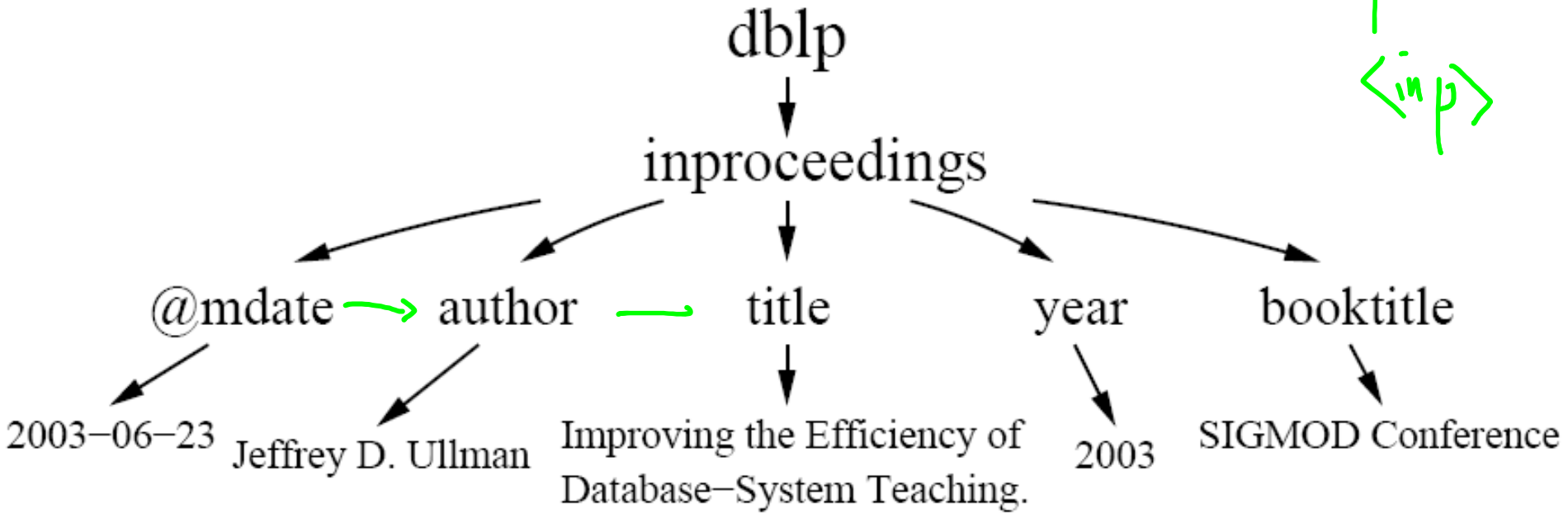
- To find a space-efficient storage scheme for XML data without compromising both query and update performances

Proposed Storage Structure

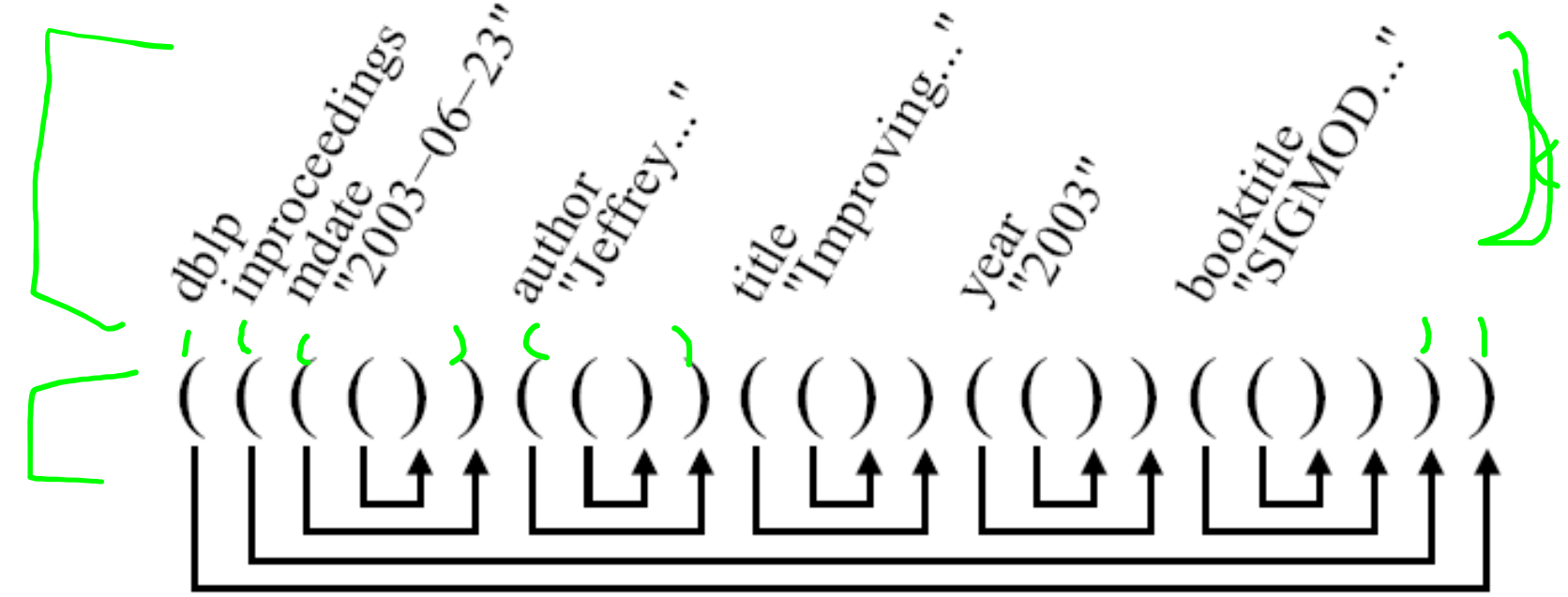


The ISX Structure

Sample DBLP XML Fragment

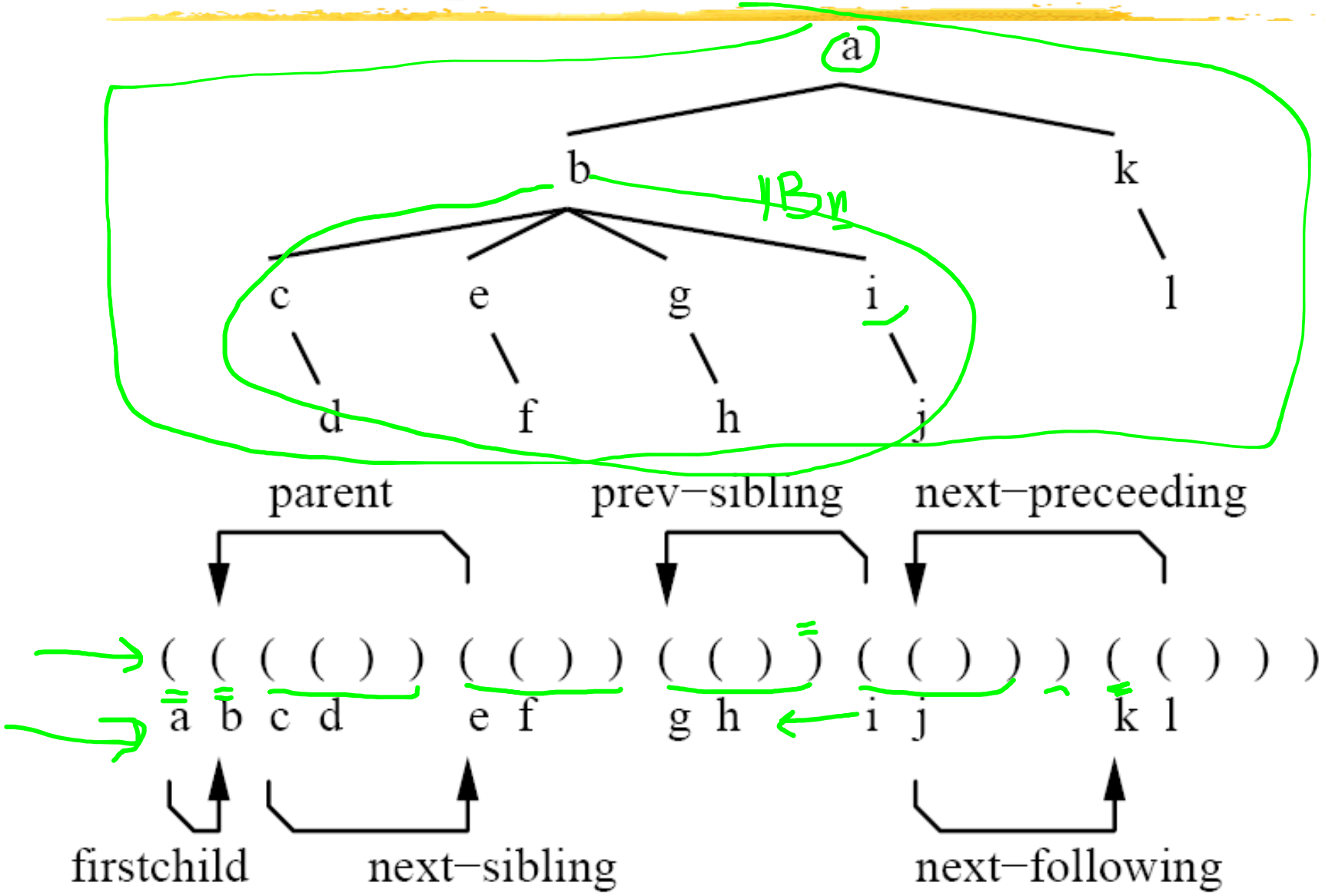


Balanced Parenthesis Encoding



0 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1

Node Navigations



Primitive operators

Algorithm 2 Primitive Operators for Topology Layer Access

```
FORWARDEXCESS(start, end, k)
1   for each current from start to end do
2       if (tier0[current] is an open parenthesis) then
3           k ← k − 1
4       else
5           k ← k + 1
6       if (k = 0) then
7           return current
8   return NOT-FOUND
BACKWARDEXCESS(start, end, k)
1   for each current from start to end step −1 do
2       if (tier0[current] is an open parenthesis) then
3           k ← k − 1
4       else
5           k ← k + 1
6       if (k = 0) then
7           return current
8   return NOT-FOUND
PREV(node)
1   if (node > 0) then return node − 1 else return NOT-FOUND
NEXT(node)
1   if (node < |tier0|) then return node + 1 else return NOT-FOUND
FINDCLOSE(node)
1   return FORWARDEXCESS(node, |tier0|, 0)
FINDOPEN(node)
1   return BACKWARDEXCESS(node, |tier0|, 0)
```

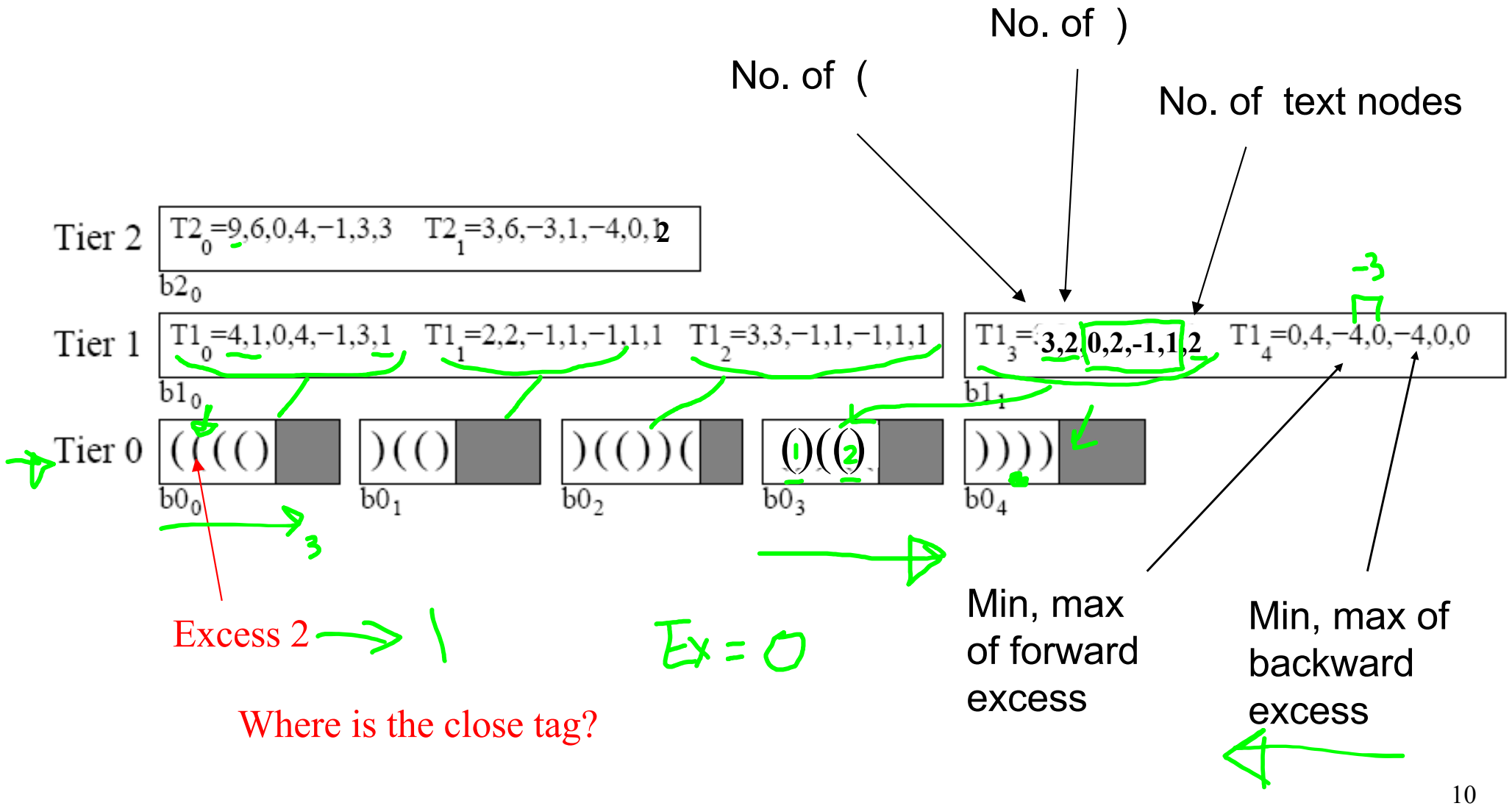
Tier 2 excess

Algorithm 3 Calculate Local Excess in a Tier 2 Block

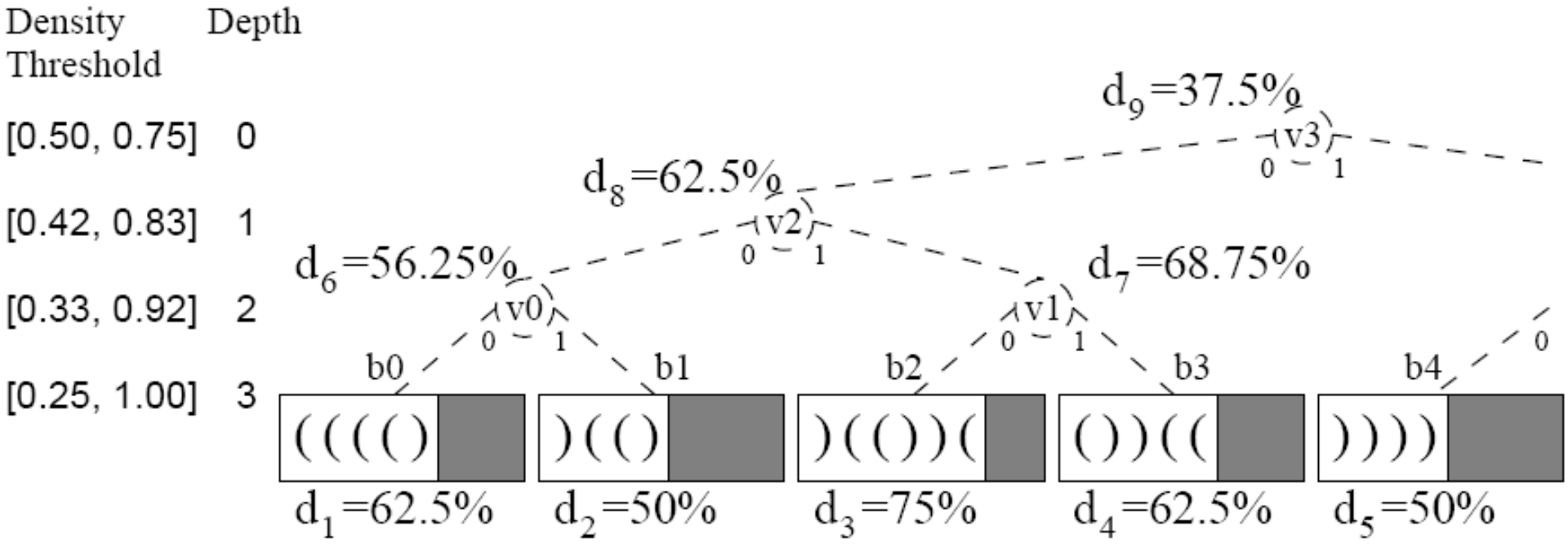
TIER2LOCALEXCESS($t2$)

1 $\{t1_{start}, t1_{end}\} \leftarrow \{\frac{t2*|T^2|}{|T^1|}, \frac{(t2+1)*|T^2|}{|T^1|} - 1\}$
2 $\{tier2[t2].m, tier2[t2].M\} \leftarrow \{tier1[t1_{start}].m, tier1[t1_{start}].M\}$
3 $excess \leftarrow tier1[t1start].L - tier1[t1start].R$
4 **for each** $t1$ **from** $t1_{start} + 1$ **to** $t1_{end}$ **do**
5 **if** $(excess + tier1[t1].m < tier2[t2].M)$ **then**
6 $tier1[t1].m \leftarrow excess + tier1[t1].m$
7 **if** $(excess + tier1[t1].M > tier2[t2].M)$ **then**
8 $tier1[t1].M \leftarrow excess + tier1[t1].M$
9 $excess \leftarrow excess + tier1[t1].L - tier1[t1].R$

Topology Tiers



Efficient Updates



Example

- 100 MB DBLP document
- 5 million XML nodes

- ISX: 1MB topology

Another example

- Core Duo 1.83GHz
- 1GB RAM
- 5400 RPM Harddrive
- MS Vista

5M DBLP	MSXML	ISX
Runtime (loading)	15MB	4MB
Loading time	0.54s	0.035s
Runtime (//www)	21MB	4MB
//www	0.096s	0.004s

100M DBLP	MSXML	ISX
Runtime (loading)	329MB	67MB
Loading time	17.8s	0.67s
Runtime (//www)	333MB	67MB
//www	1.814s	0.143s

ISX Features

Features	XMill	XGrind	NoK	TIMBER	ISX
Compression	√	√			√
Document Traversal	√	√	√	√	√
Node Navigation of All Axes			uncertain	√	√
Update Operation				√	√
Support XPath Query		√	√	√	√

TABLE I

COMPARISON OF SUPPORTED FEATURES

Experiments

Setup

- Fixed at **64MB memory buffer**
- Up to 16 GB XML document
- E.g. 16 GB DBLP contains > 770 million nodes
- **NO** index or query optimization has been employed for ISX (*except for ISX Stream where TurboXPath algorithm has been employed*)

Storage Size (ISX vs NoK)

Document Size	DBLP		PSD		TreeBank	
(MB)	NoK	ISX	NoK	ISX	NoK	ISX
5	18	3.64	17.91	3.36	19	3.21
10	35	7.23	36.12	6.82	38	6.36
50	181	36.1	182.42	34.14	196	31.78
100	367	72.1	377.52	68.74	389	63.43
250	918	180.2	950	171.9	974	159

TABLE II

STORAGE SIZE OF ISX VS. NOK

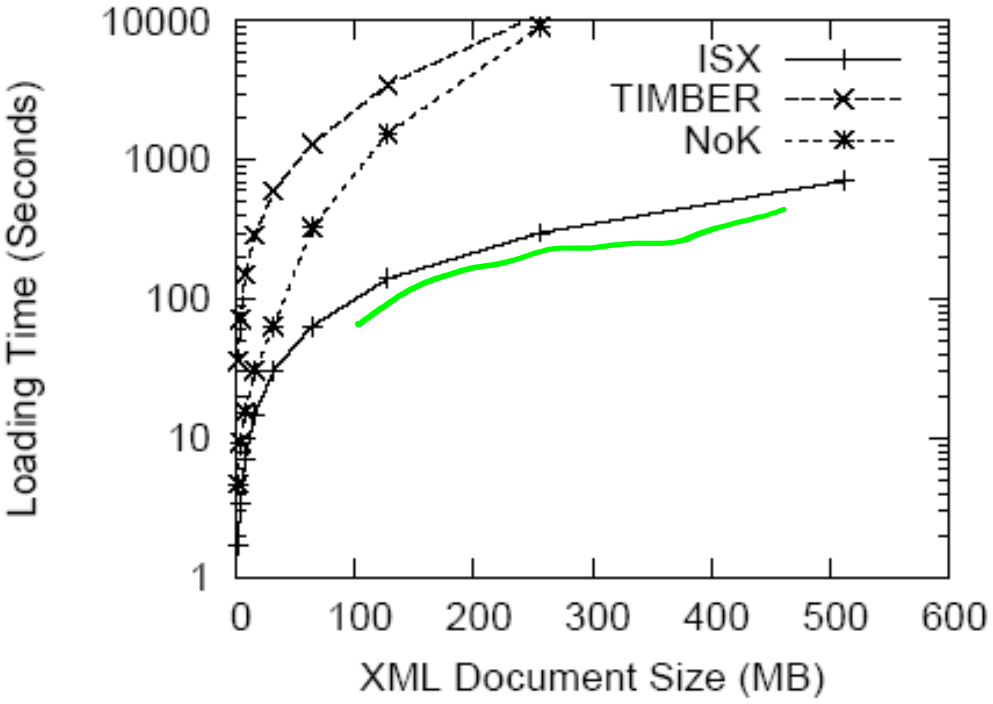
Storage Size (ISX, XMill, XGrind): DBLP

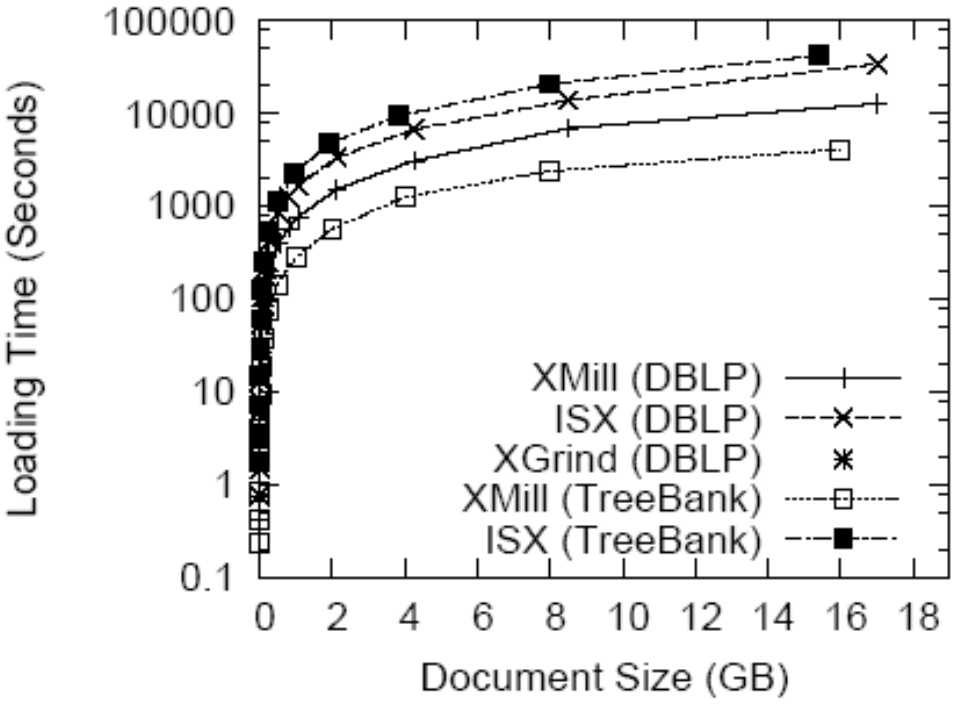
Source Data (MB)	ISX (MB)	ISX Compressed (MB)	XMill (MB)	XGrind (MB)	Source Data (MB)	ISX (MB)	ISX Compressed (MB)	XMill (MB)	XGrind 1 (MB)
1	1	0.4	0.1	0.3	**256**	182	82.7	31.5	75.0
2	1	0.7	0.3	0.6	**500**	363	163.7	62.6	Failed
5	3	1.5	0.5	1.3	**750**	549	249.7	94.0	Failed
8	5	2.5	0.9	2.1	**1000**	726	327.5	125.3	Failed
16	10	5	1.8	4.3	**2000**	1452	654.9	250.5	Failed
32	21	10	3.7	8.6	**4000**	2903	1309.8	501.0	Failed
64	42	20	7.2	17.4	**8000**	5807	2619.6	978.48	Failed
128	87	40.2	14.9	35.8	**16000**	9411	4629.9	1952.81	Failed

TABLE III

STORAGE SIZE OF ISX (WITH AND WITHOUT TEXT COMPRESSION), XMILL AND XGRIND ON DBLP

Bulk Loading Performance



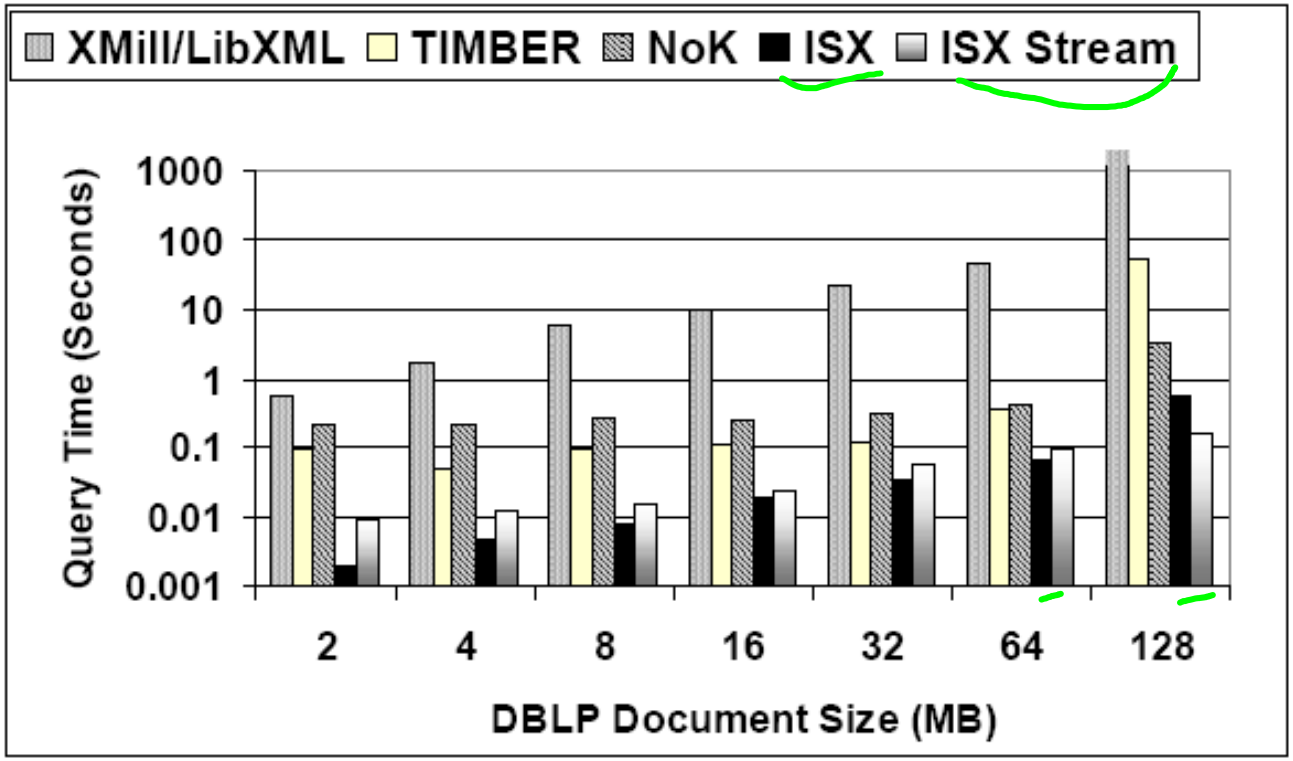


Queries

Query #	XPath Expression	1 GB	2 GB	4 GB	8 GB	16 GB
		$\|Final\|$	$\|Final\|$	$\|Final\|$	$\|Final\|$	$\|Final\|$
Q1	$//inproceedings$	402667	981484	2012761	4160339	8453066
Q2	$//masterthesis$	74	156	315	627	1251
Q3	$/dblp/article$	442184	717449	1379945	2630711	5135130
Q4	$//inproceedings/title$	402667	981484	2012761	4160339	8453066
Q5	$//article[.//month/text() = "July"]//title$	857309	1729184	3454708	6920136	13848372
Q6	$//inproceedings[.//ee]//pages$	796742	1607116	3210628	6430194	12868471

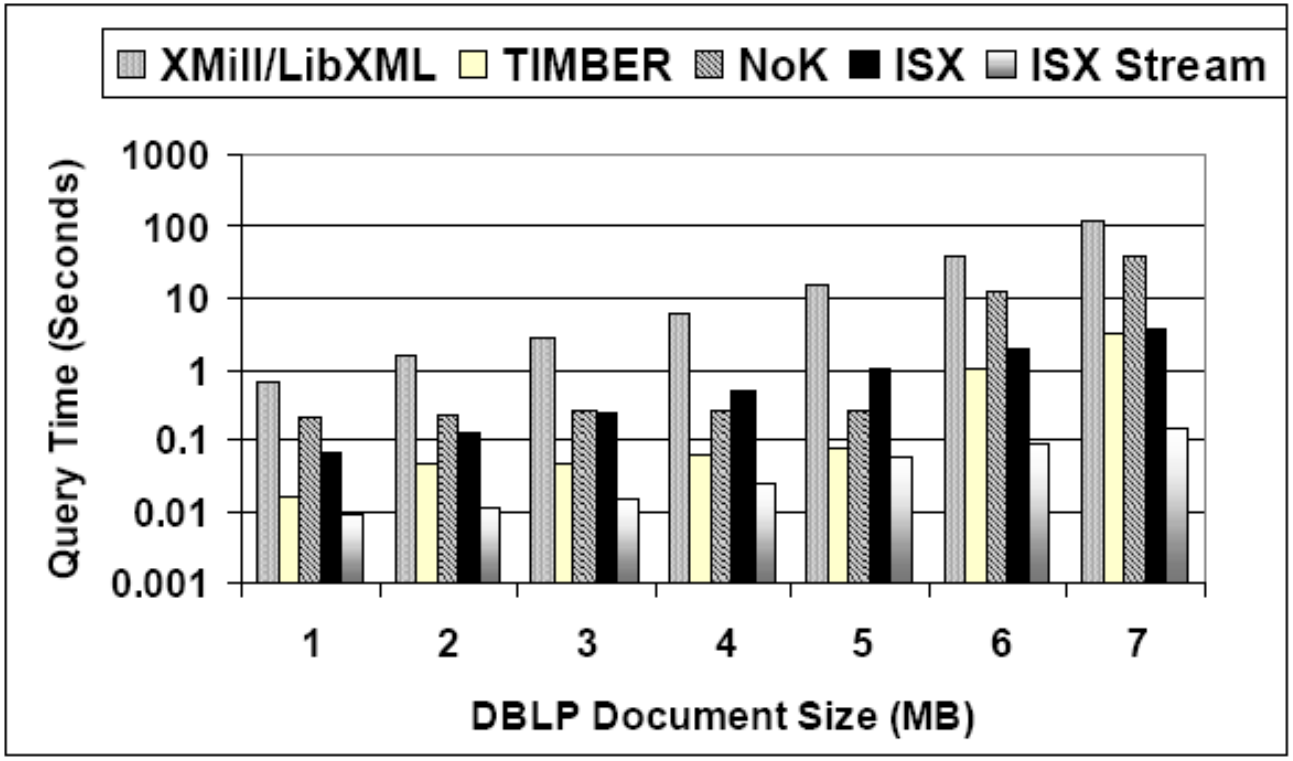
Table 4: Test Queries and Final Result Sizes

Q1: //inproceedings



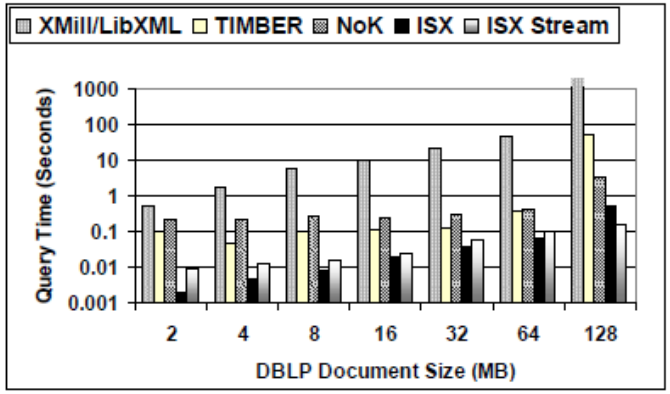
(a) DBLP Q1

Q5: //article[.//month/text() = "July"]//title

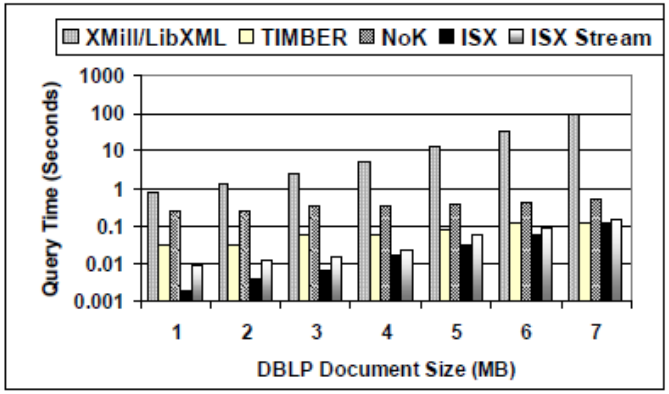


(e) DBLP Q5

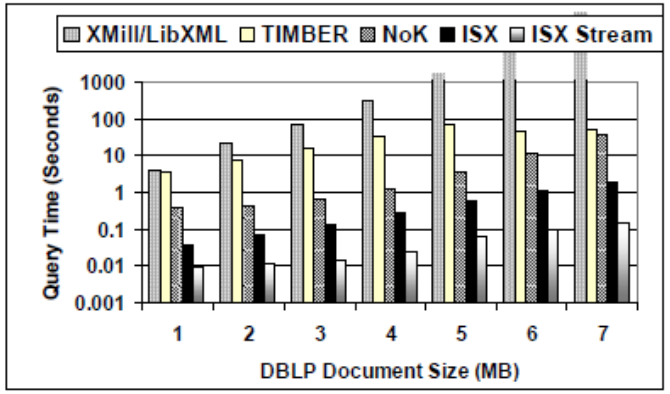
Other queries



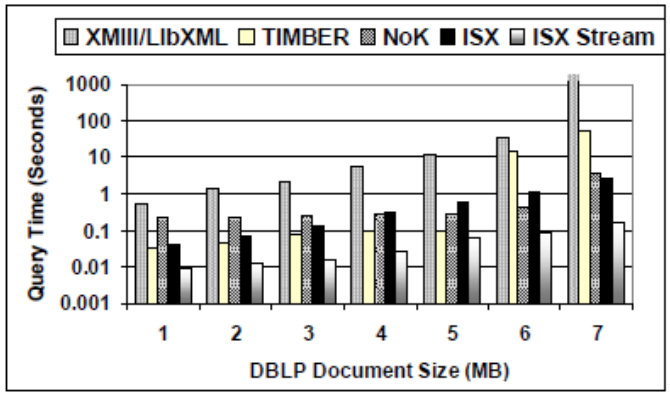
9(a) DBLP Q1



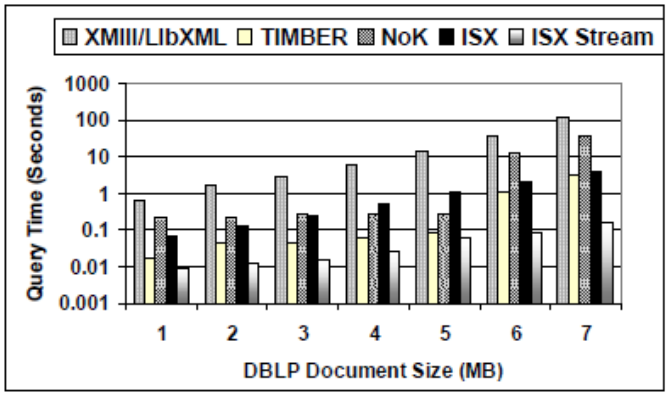
9(b) DBLP Q2



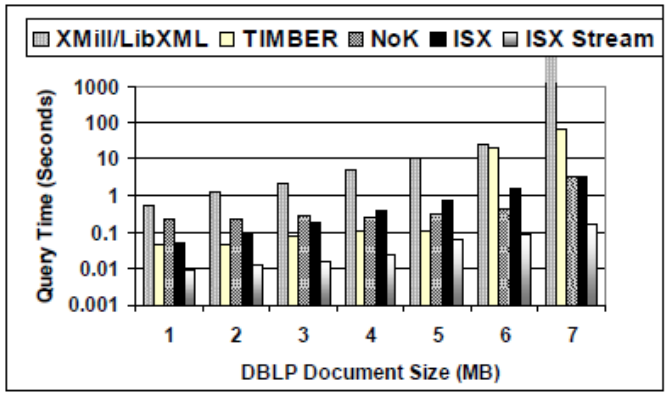
9(c) DBLP Q3



9(d) DBLP Q4



9(e) DBLP Q5

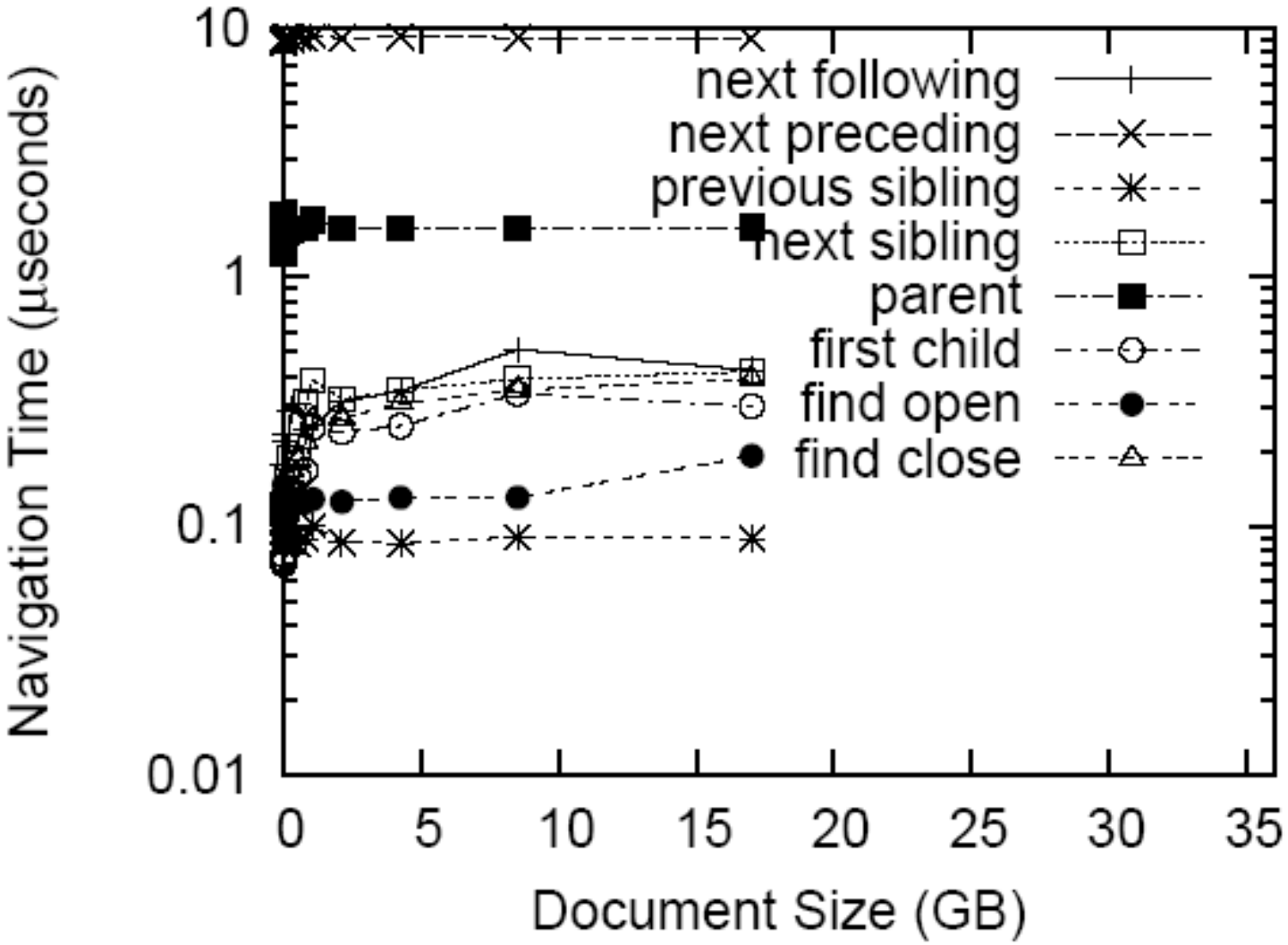


9(f) DBLP Q6

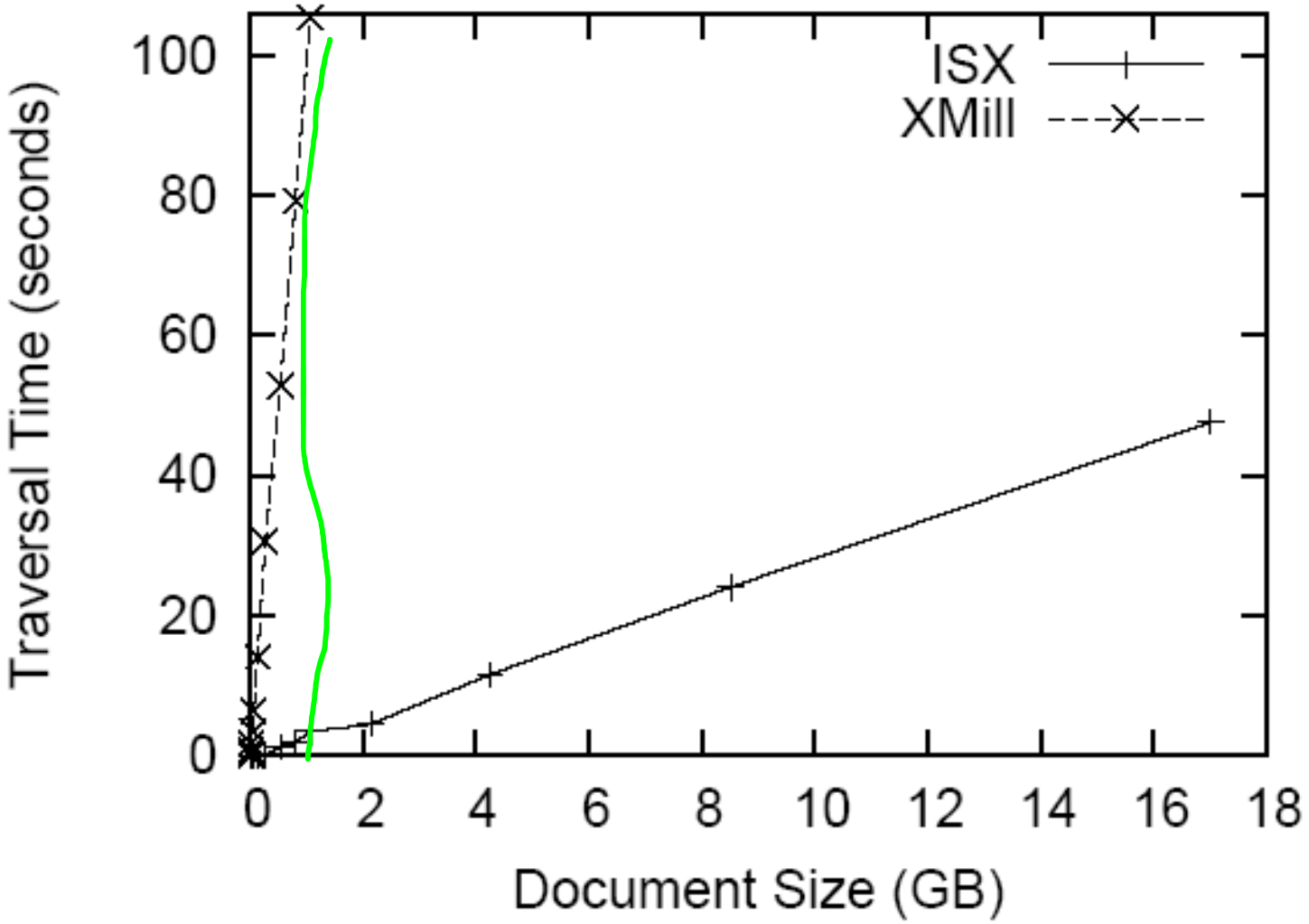
XPath 13 axes

- We can navigate along 13 axes:
  - ancestor
  - ancestor-or-self
  - attribute
  - child
  - descendant
  - descendant-or-self
  - following
  - following-sibling
  - namespace
  - parent
  - preceding
  - preceding-sibling
  - self

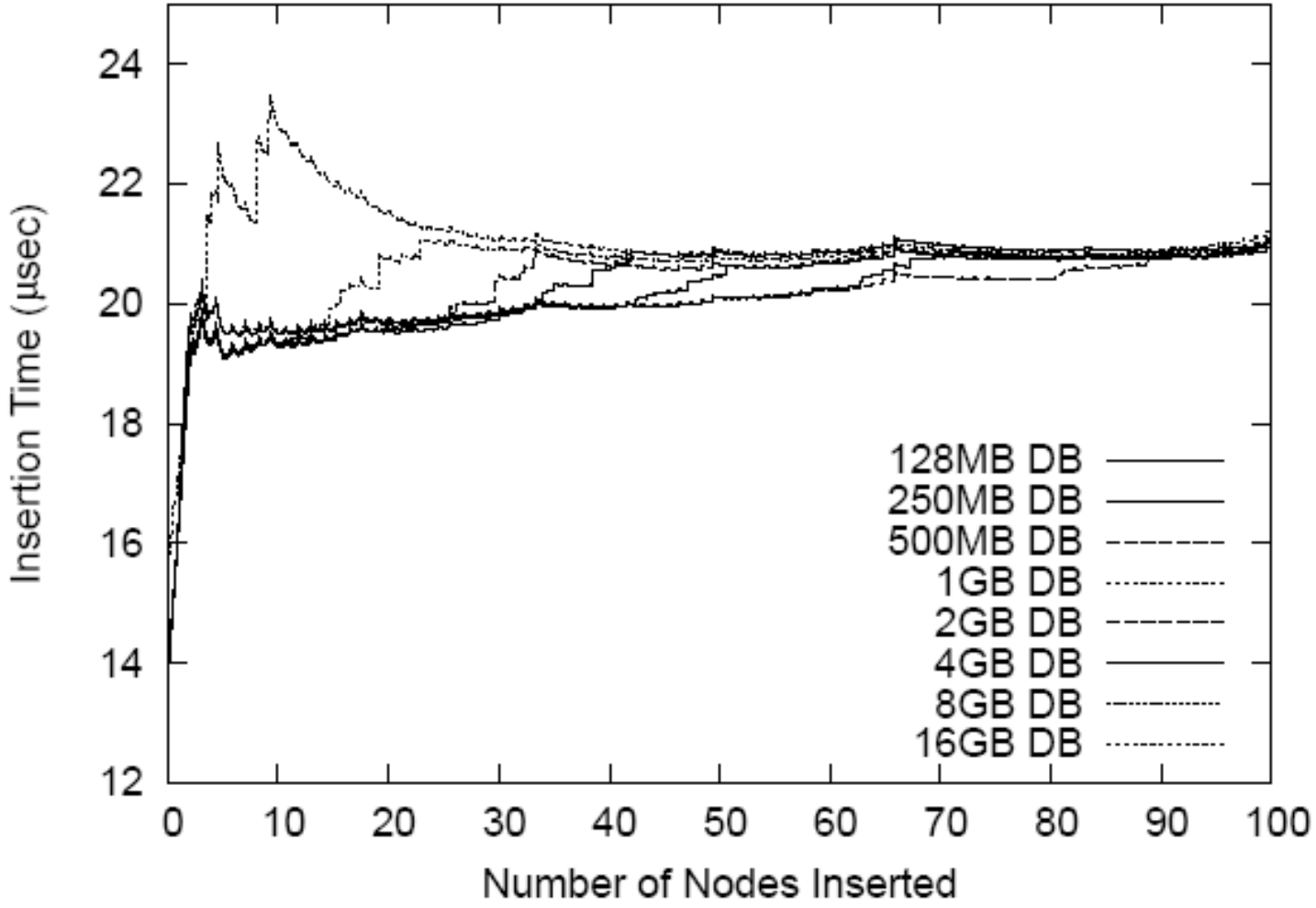
Node Navigation



Full document traversal



Update (Insertion) Performance



ISX Summary

- Small storage footprint
- Small runtime footprint
- Fast and consistent performance on navigational access
- Superior query performance (further indexing / query optimization can be added)
- Superior update performance

Compressing and Searching XML Data Via Two Zips

Paolo Ferragina et al.

Slides modified from P. Ferragina's

A transform for "labeled trees"

[Ferragina *et al*, *IEEE* Focs '05]

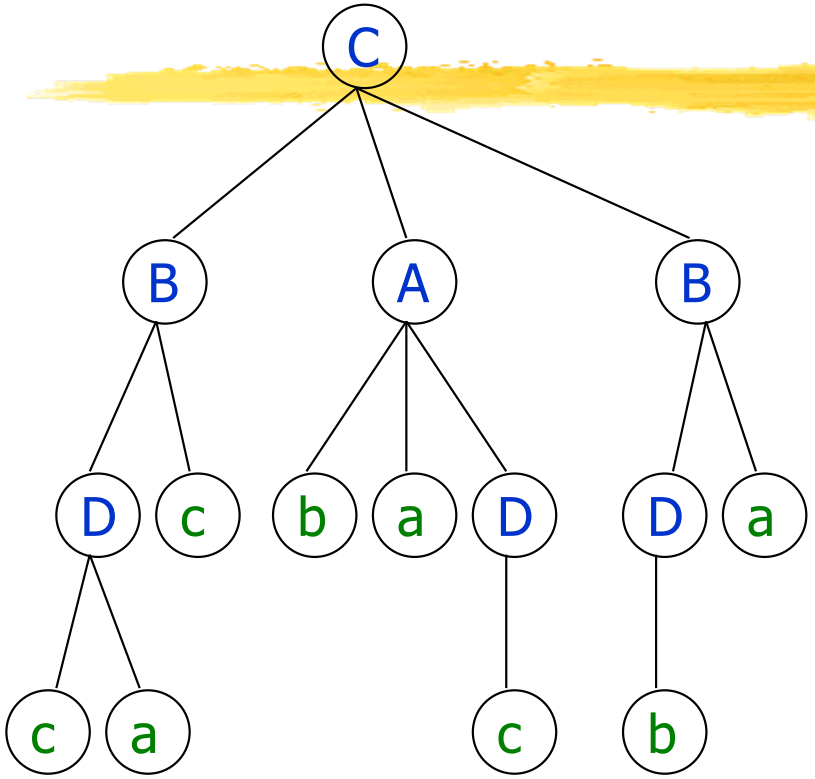
- ☑ We proposed the XBW-transform that mimics on trees the nice structural properties of the Burrows-and-Wheeler Trasform on strings

- ☑ The XBW **linearizes** the tree T in **2 arrays** s.t.:

  - ☑ the compression of T *reduces to* use any compressor *(gzip, bzip,...)* over these two arrays

  - ☑ the indexing of T *reduces to* implement simple rank/select query operations over these two arrays

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

Permutation of tree nodes

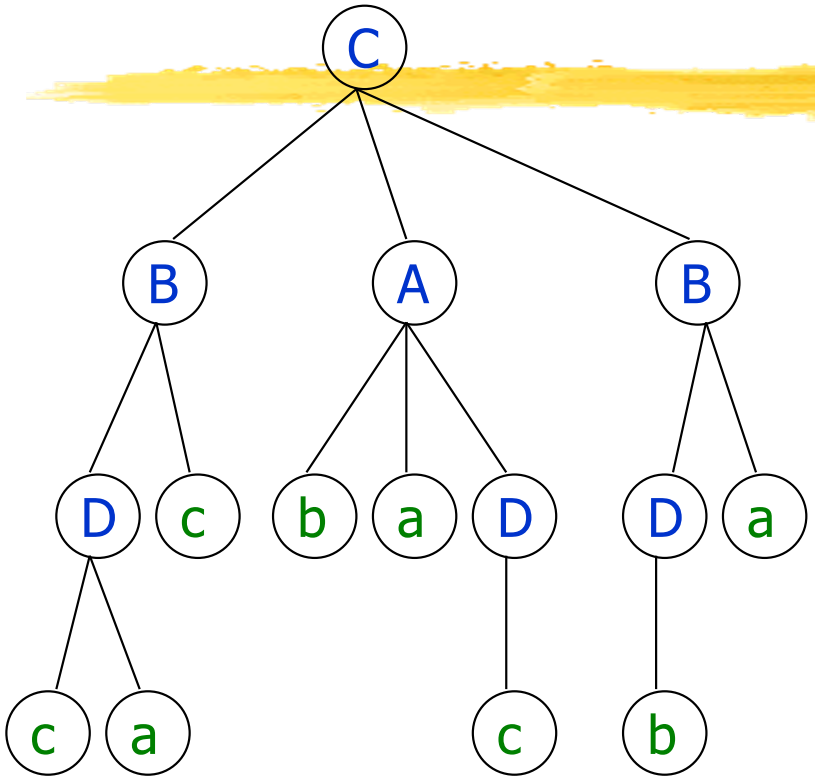
upward labeled paths

Step 1.

Visit the tree in pre-order.

For each node, write down its label and the labels on its upward path

The XBW-Transform

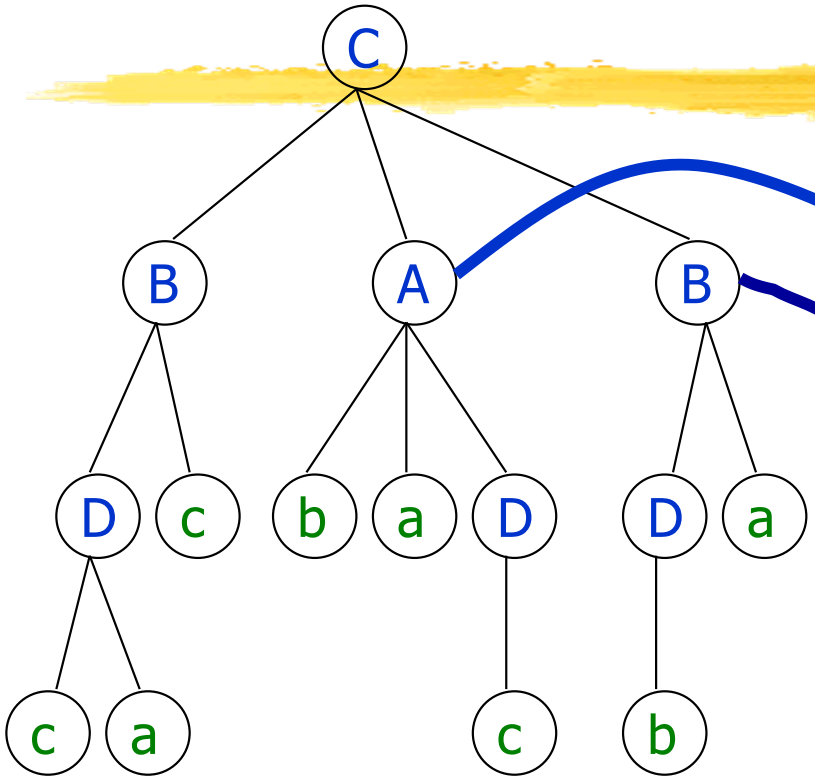


S_α	S_π
C	ε
b	A C
a	A C
D	A C
D	B C
c	B C
D	B C
a	B C
B	C
A	C
B	C
c	D A C
c	D B C
a	D B C
b	D B C

upward labeled paths

Step 2.
Stably sort according to S_π

The XBW-Transform



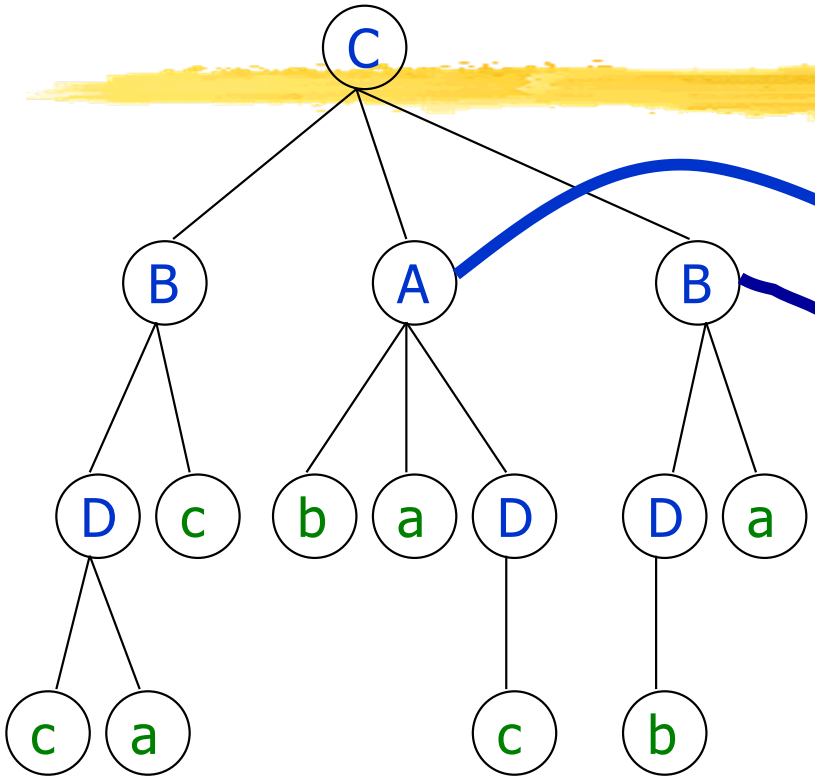
S_{last}	S	S_π
1	C	ε
0	b	A C
0	a	A C
1	D	A C
0	D	B C
1	c	B C
0	D	B C
1	a	B C
0	B	C
0	A	C
1	B	C
1	c	D A C
0	c	D B C
1	a	D B C
1	b	D B C

XBW

Step 3.

Add a binary array S_{last} marking the rows corresponding to *last* children

The XBW-Transform



S_{last}	S	S_π
1	α 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

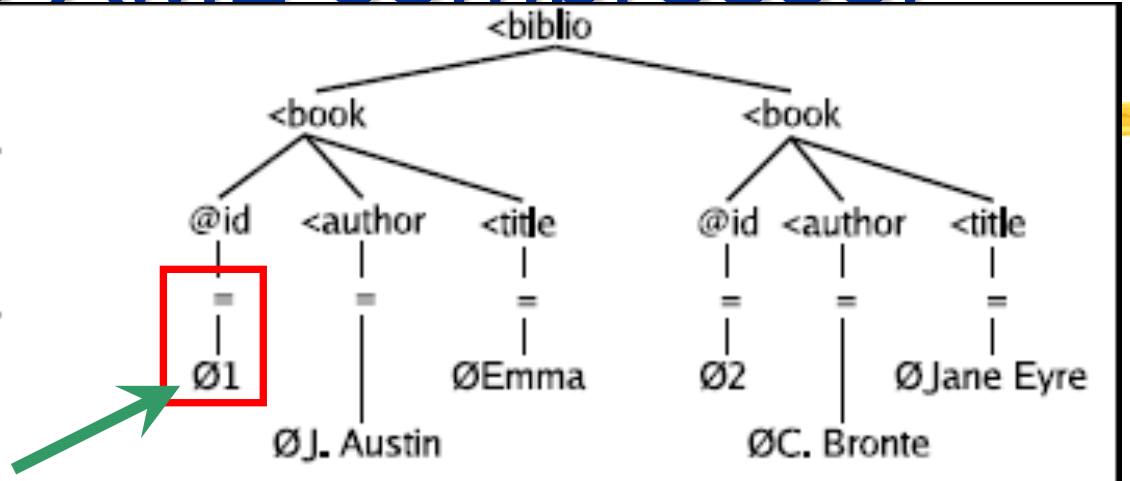
XBW

Key fact

Nodes correspond to items in $<S_{last}, S_\alpha>$

XBzip – a simple XML compressor

```
<biblio>
  <book id=1>
    <author>J. Austin</author>
    <title>Emma</title>
  </book>
  <book id=2>
    <author>C. Bronte</author>
    <title>Jane Eyre</title>
  </article>
</biblio>
```



Rk	S_{last}	S_α	S_π
1	1	<biblio	*empty string*
2	1	=	<author<book<biblio
3	1	=	<author<book<biblio
4	0	<book	<biblio
5	1	<book	<biblio
6	0	@id	<book<biblio
7	0	<author	<book<biblio
8	1	<title	<book<biblio
9	0	@id	<book<biblio
10	0	<author	<book<biblio
11	1	<title	<book<biblio
12	1	=	<title<book<biblio
13	1	=	<title<book<biblio
14	1	=	@id<book<biblio
15	1	=	@id<book<biblio
16		ØJ. Austin	=<author<book<biblio
17		ØC. Bronte	=<author<book<biblio
18		ØEmma	=<title<book<biblio
19		ØJane Eyre	=<title<book<biblio
20		Ø1	=@id<book<biblio
21		Ø2	=@id<book<biblio

Tags, Attributes and symbol =

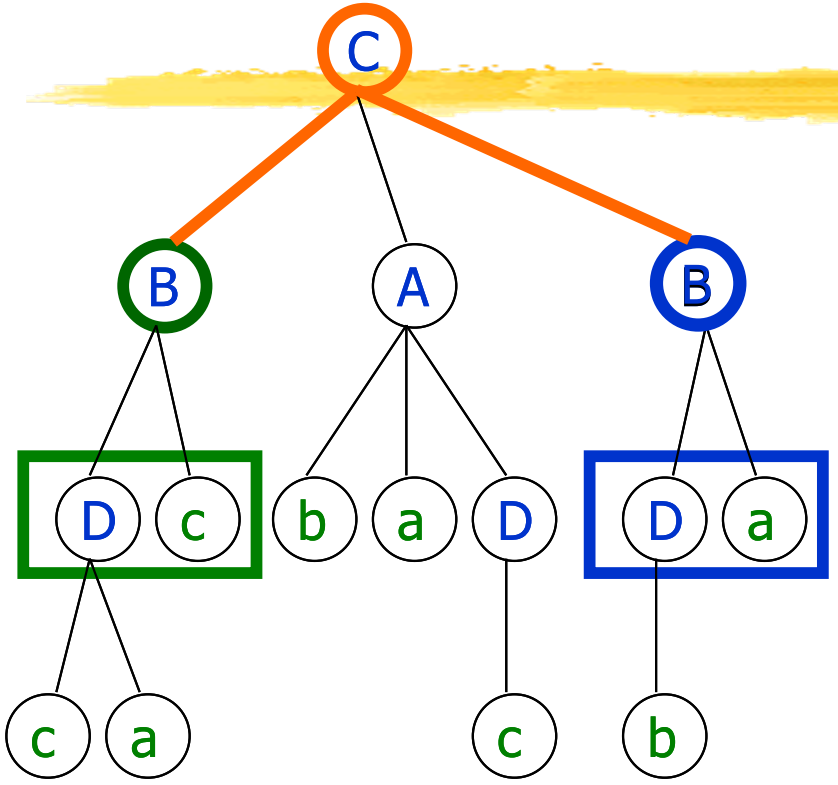
Pcdata

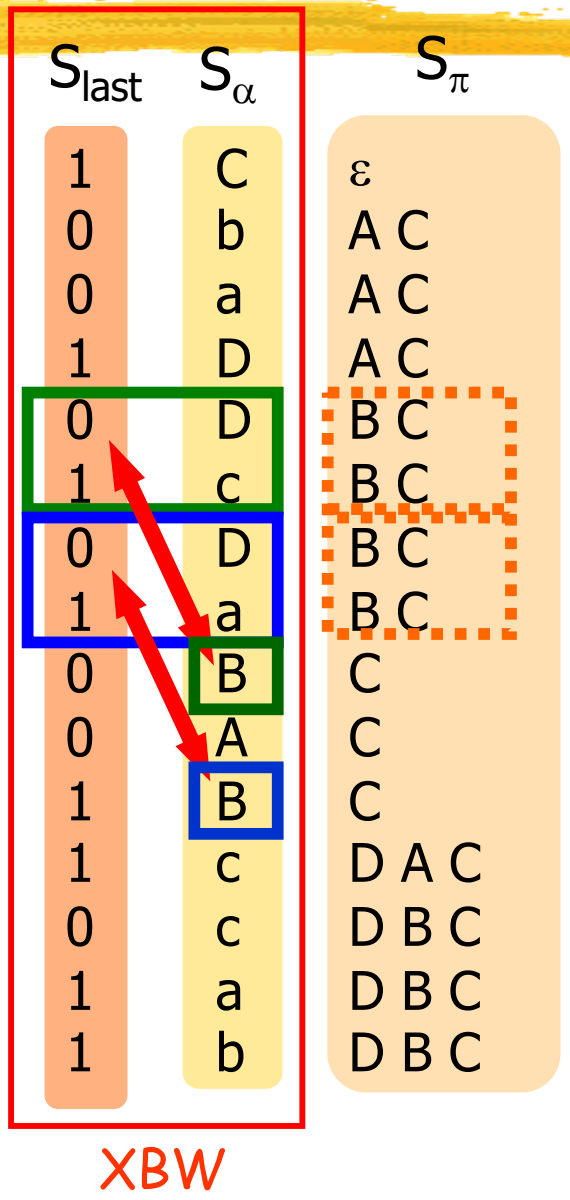
XBW is compressible:

❶ S_α and S_{pcdata} are locally homogeneous

❷ S_{last} has some structure

Some structural properties

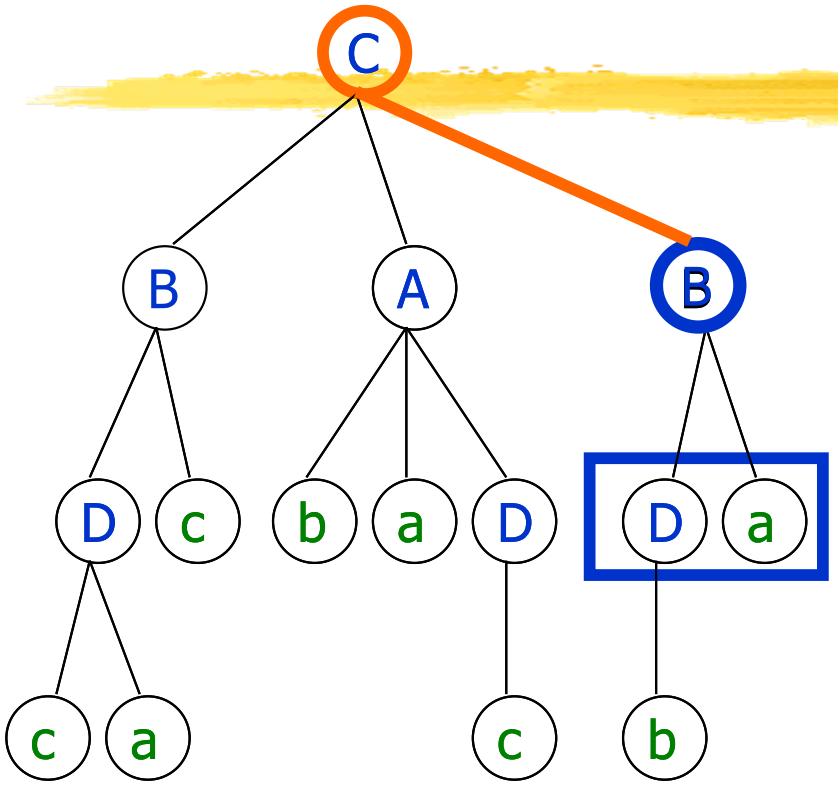


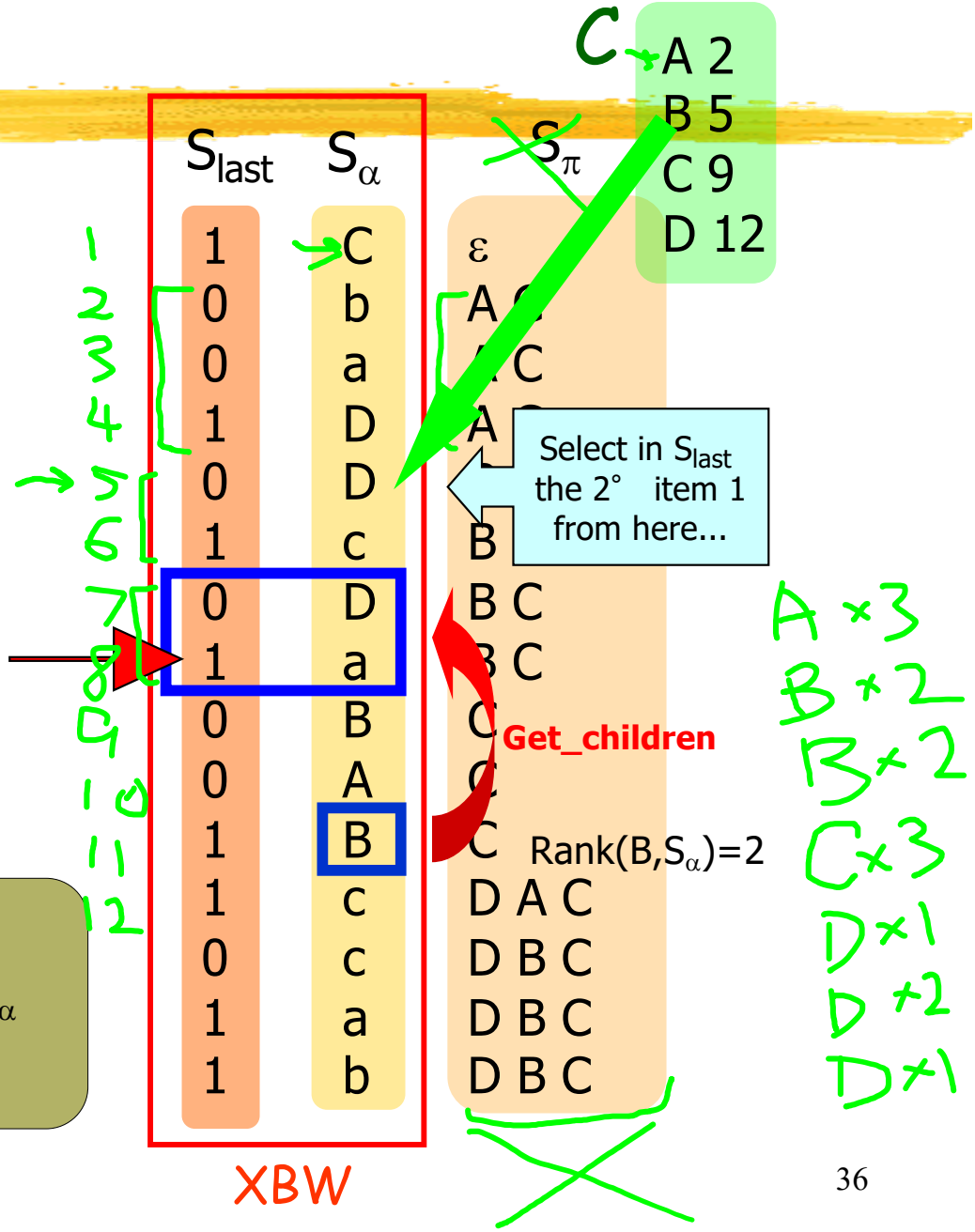


Two useful properties:

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

XBW is navigational

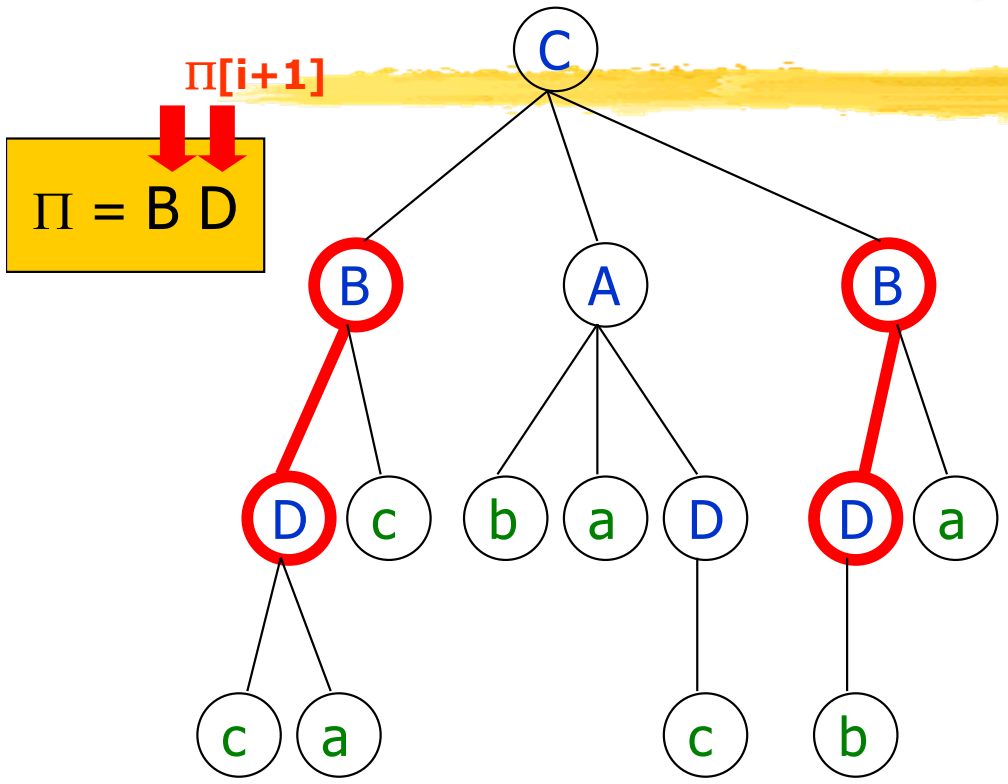




XBW is navigational:

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

XBW is searchable (count subpaths)

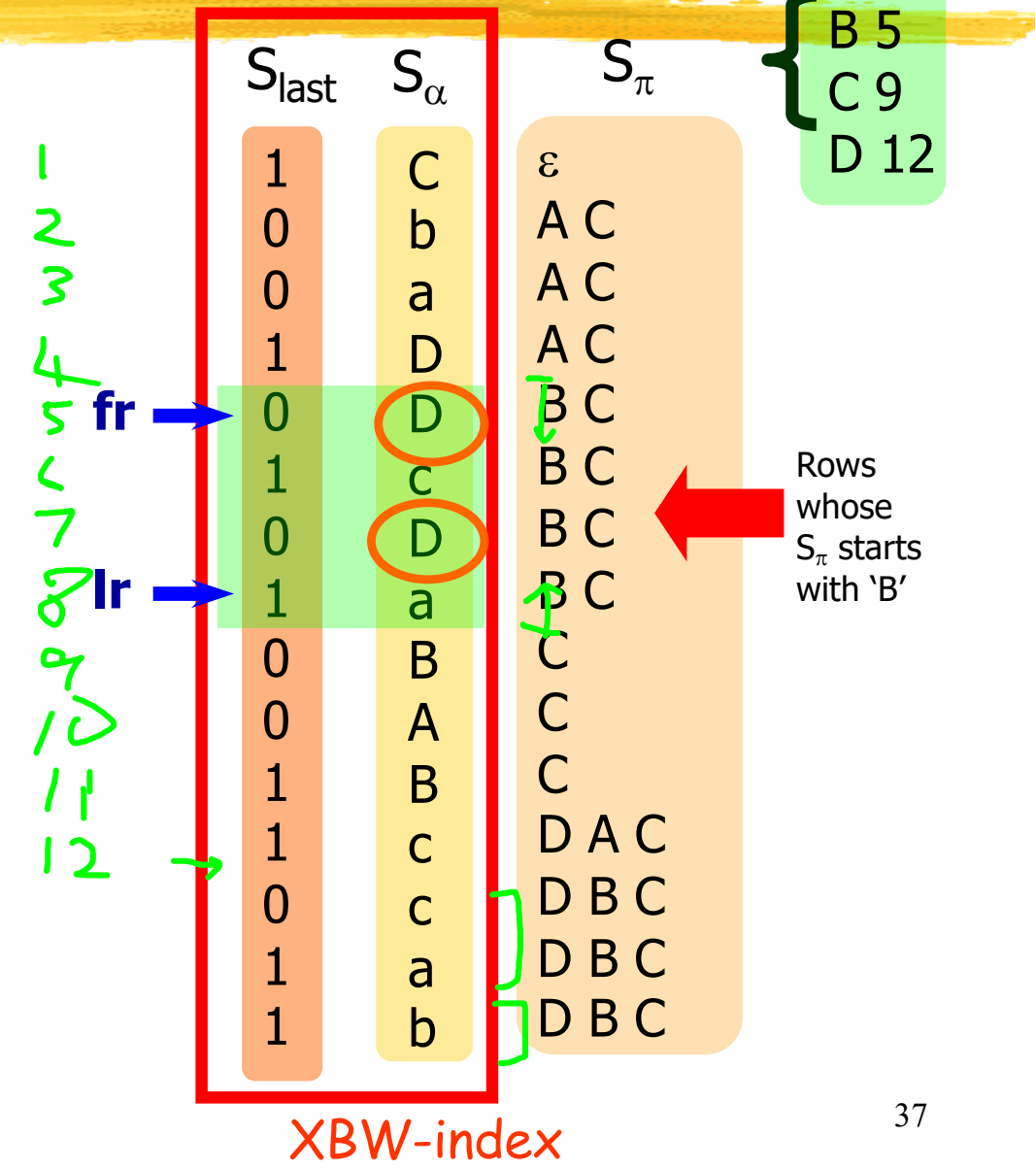


Inductive step:

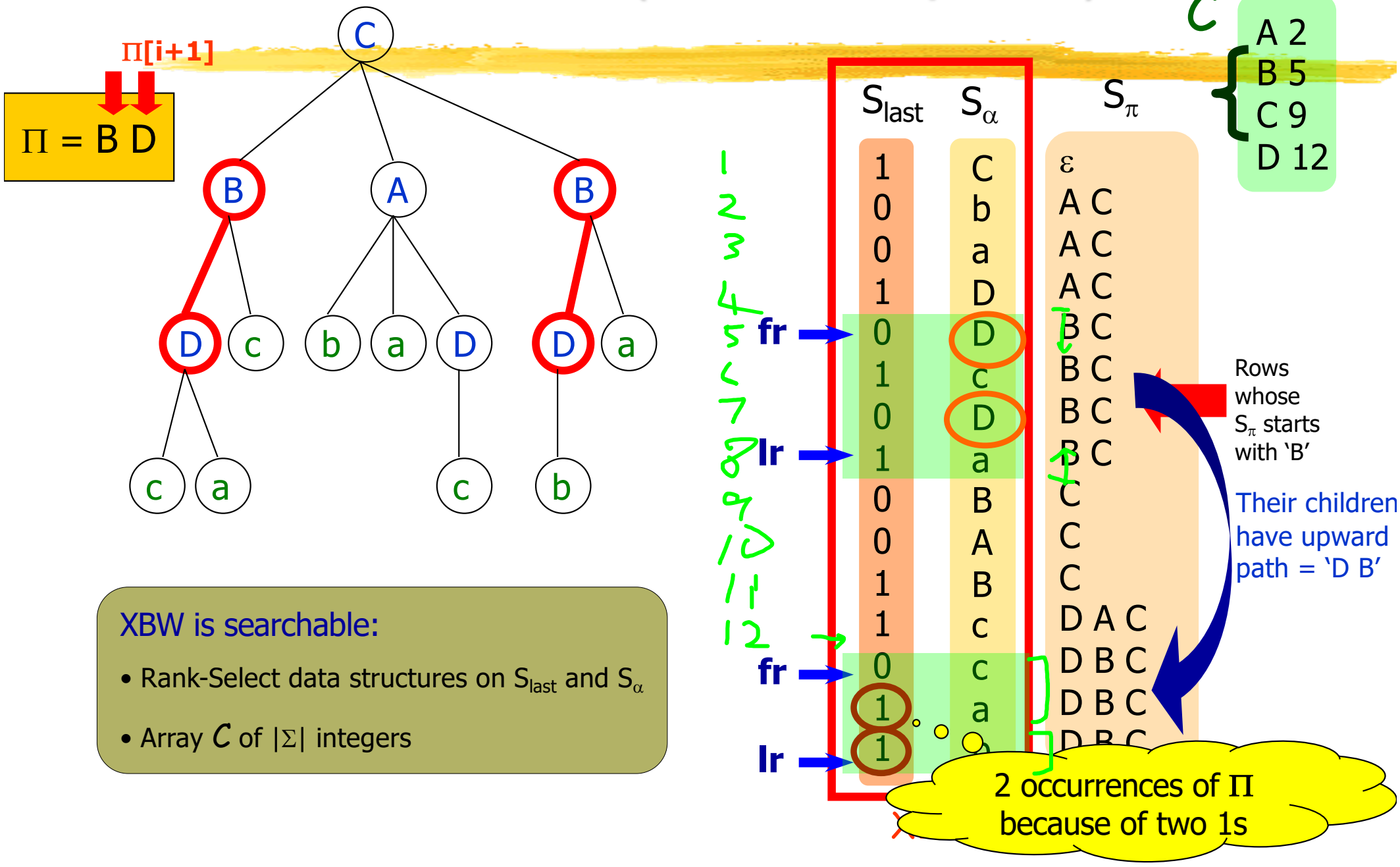
❶ Pick the next char in Π[i+1], i.e. 'D'

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

❸ Jump to their *children*



XBW is searchable (count subpaths)



XBW is searchable:

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

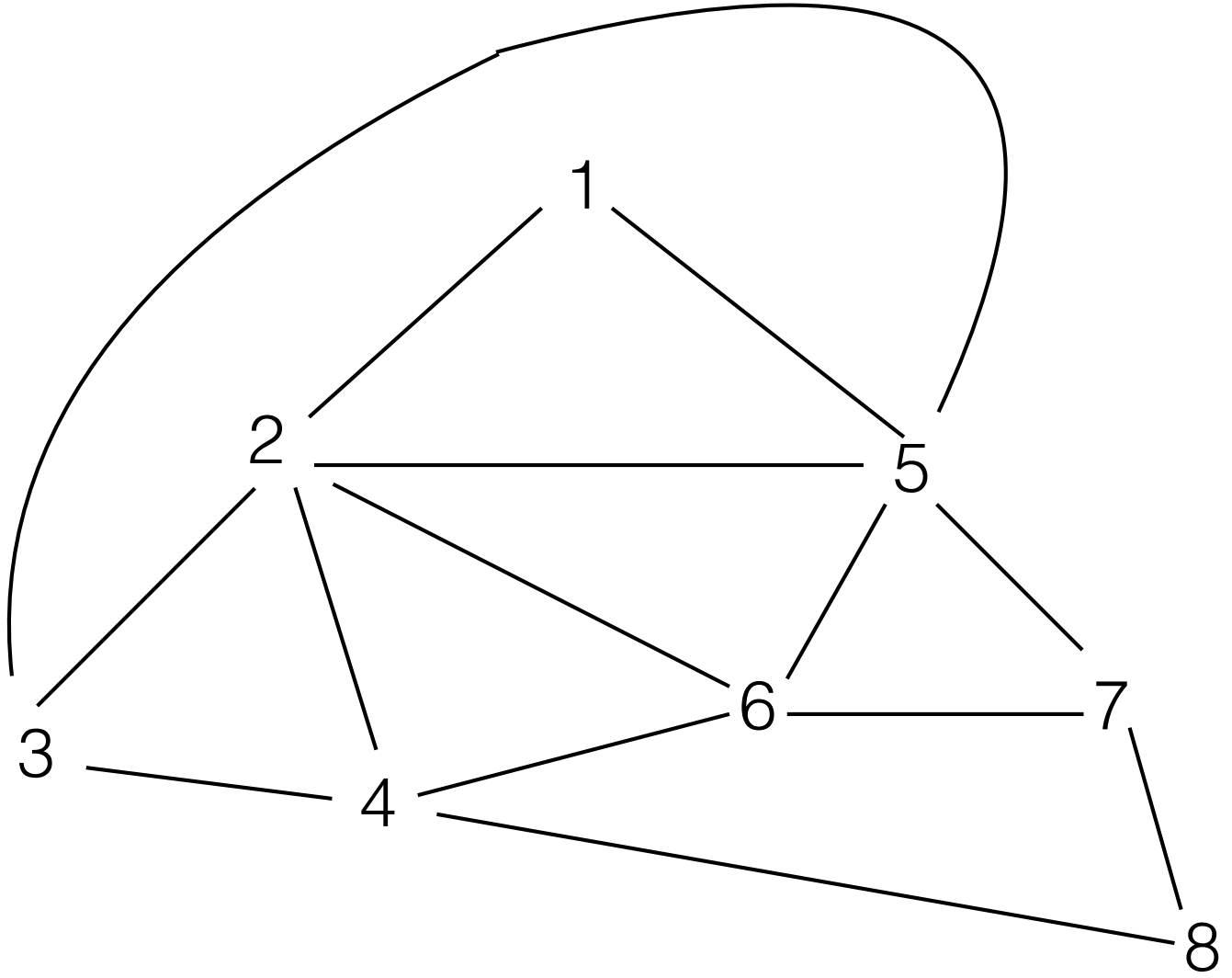
Graph compression

- Useful for many Internet based applications such as:
  - Web graph
  - Social network analysis
  - Chemical & biological applications
  - Graph visualisation and analysis

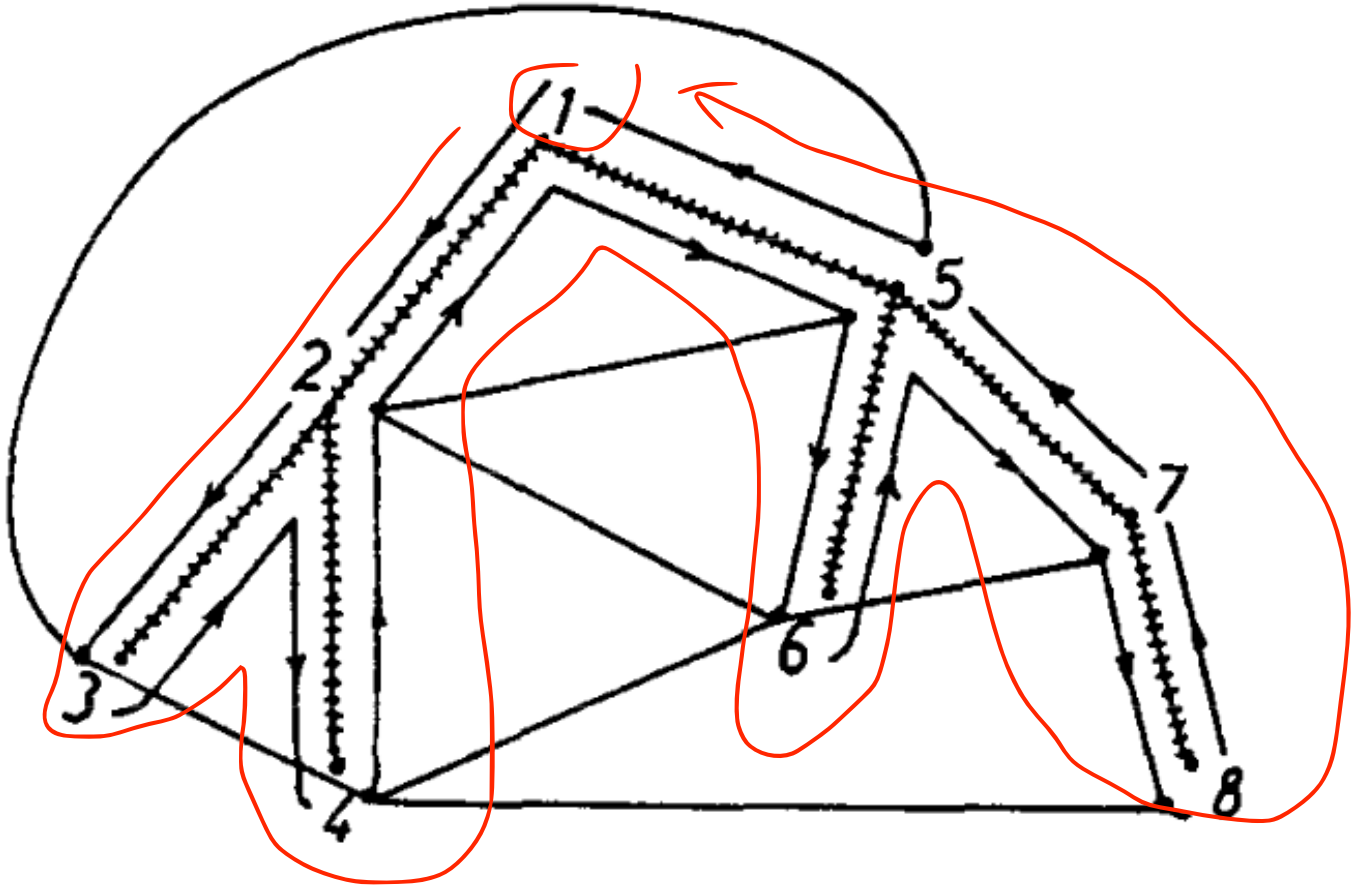
Graph compression

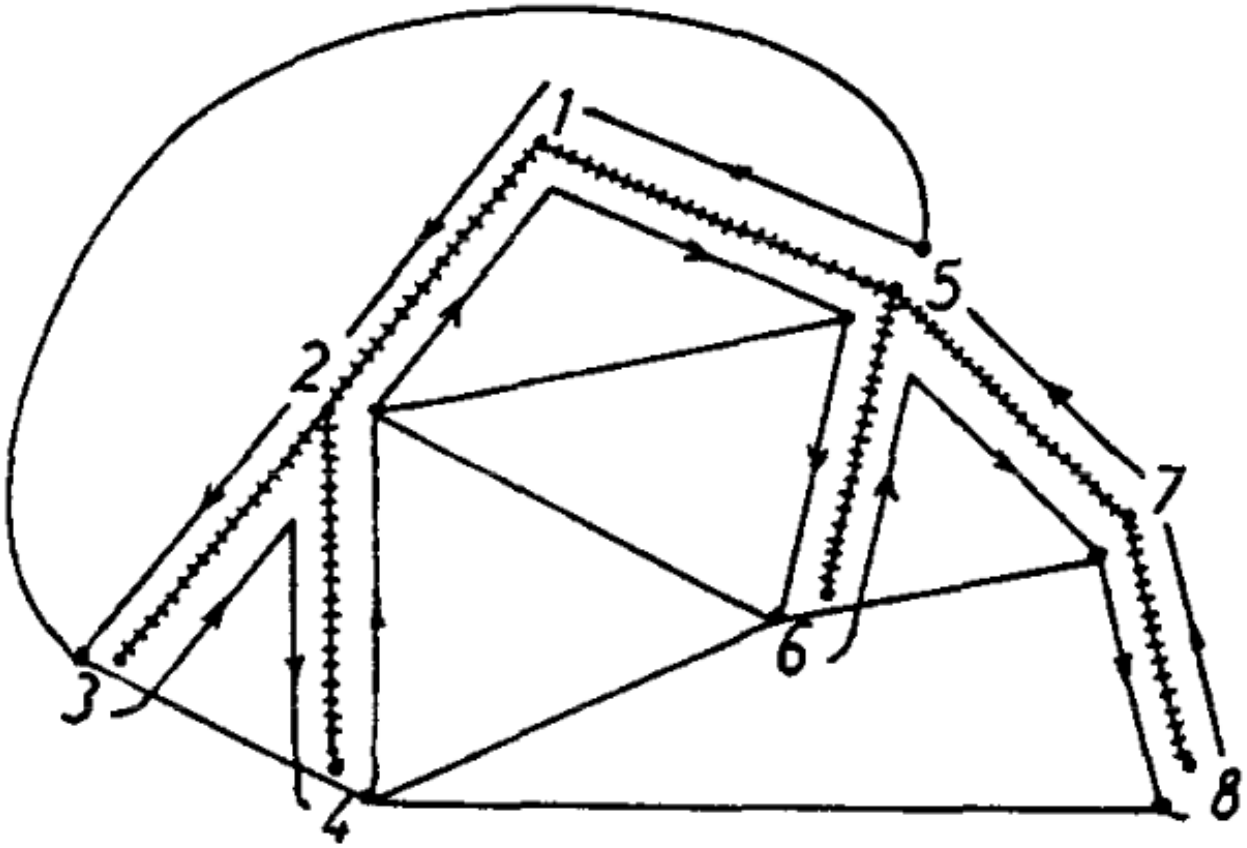
- Many techniques, e.g.,
  - Succinct graph representation
  - Adjacency matrix
  - Adjacency list

A planar graph G

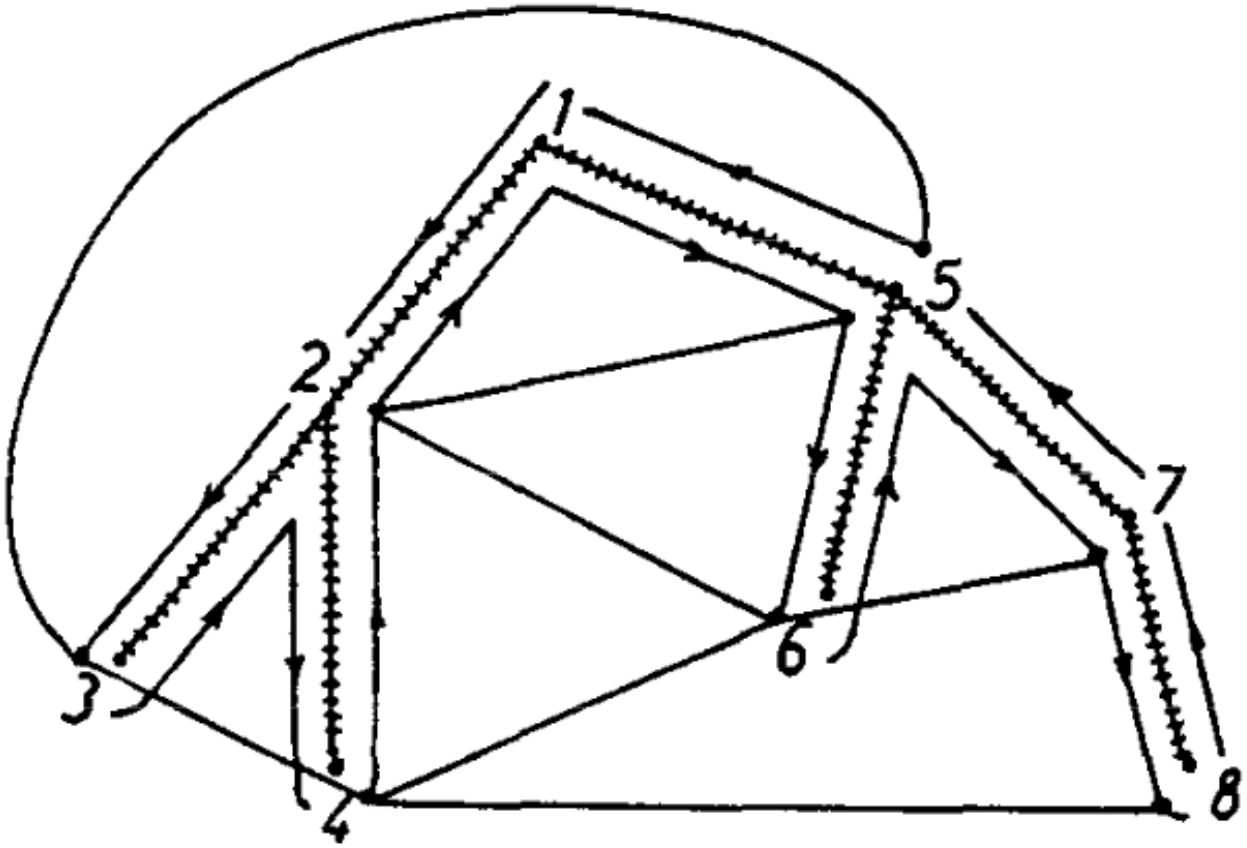


A spanning tree of G

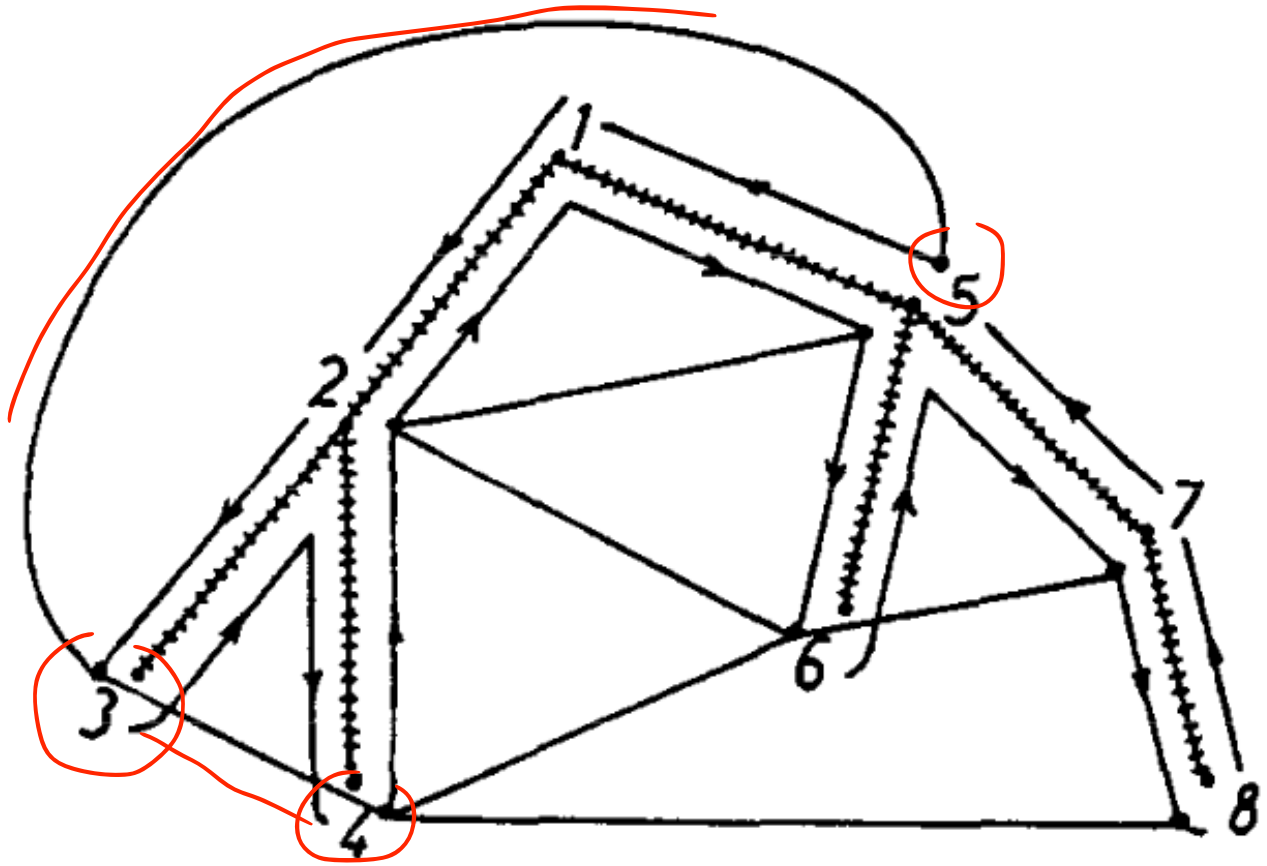




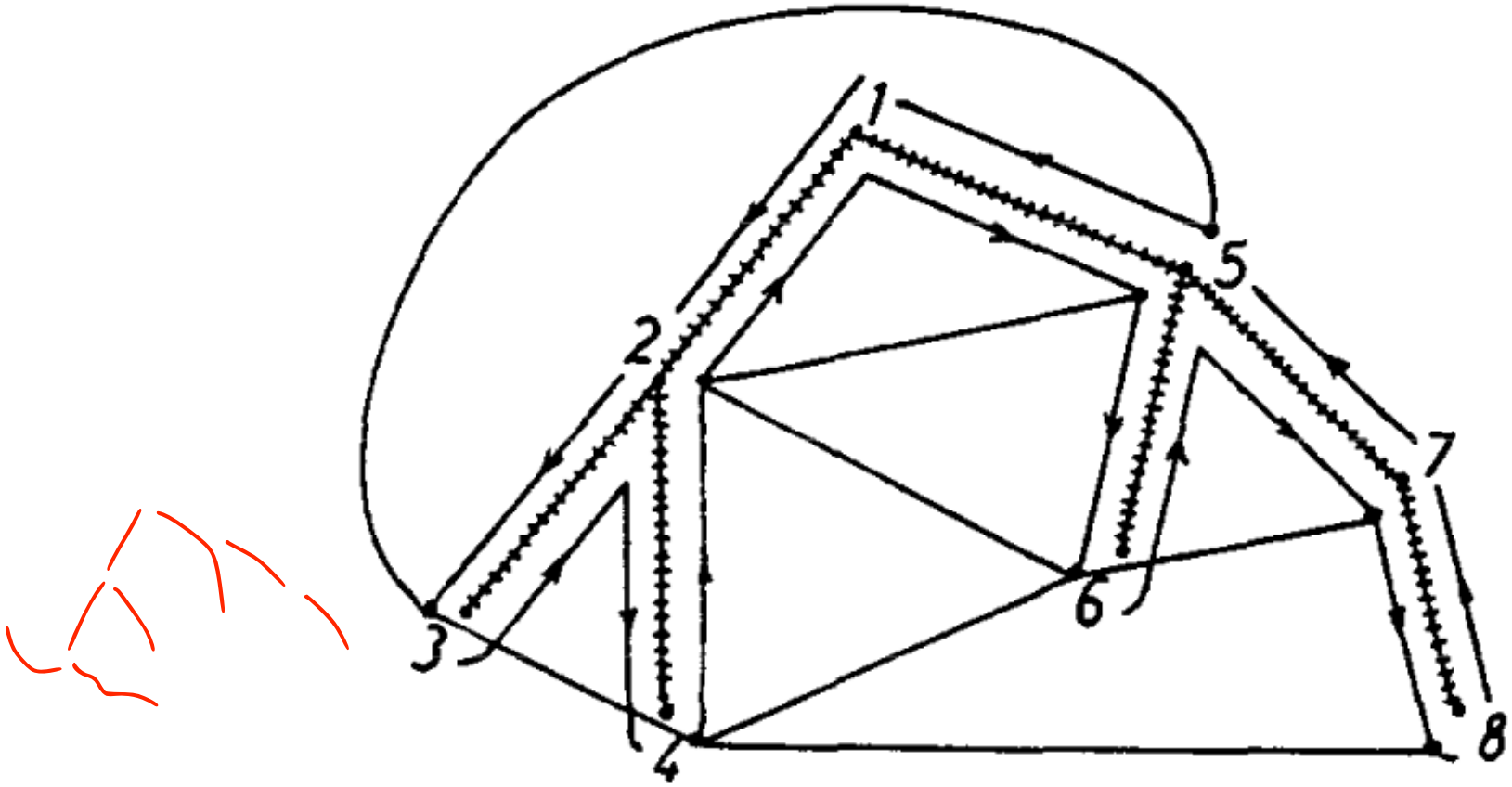
1 2 3 2 4 2 1 5 6 5 7 8 7 5 1



1	2	3	2	4	2	1	5	6	5	7	8	7	5	1
-	-	+	-	+	+	-	-	+	-	-	+	+	+	



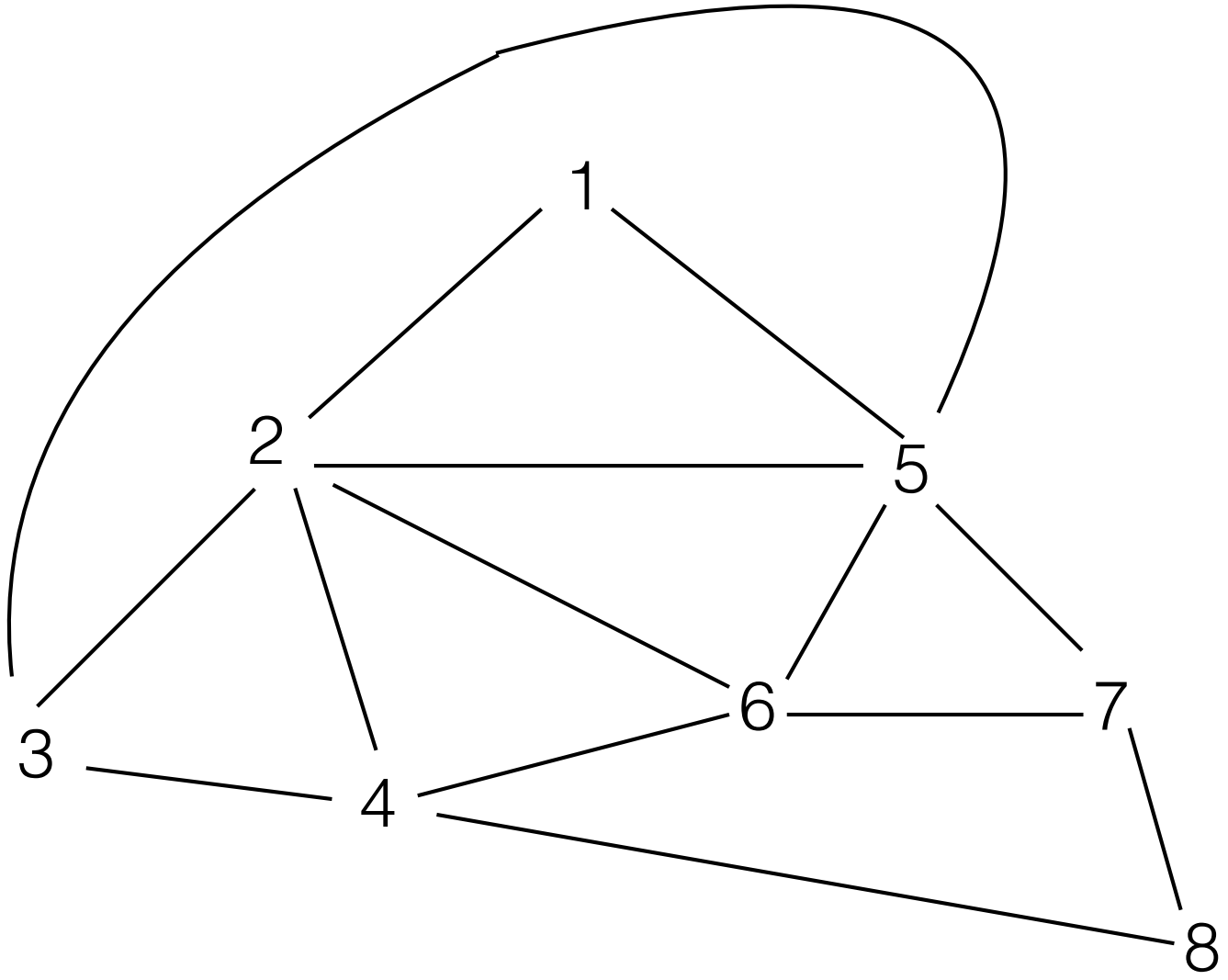
1	2	3	2	4	2	1	5	6	5	7	8	7	5	1
-	-	+	-	+	+	-	-	+	-	-	+	+	+	
		((		)((	((		)	))(		)	)		)	


1
2
3
4
5
6
7
8

- - + - + + - - + - - + + +

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

- - ((+ -)((+ ((+ -) -))(+ -) -) + +) +

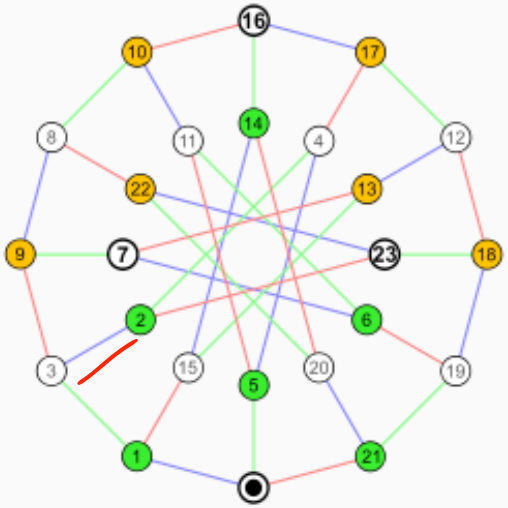


- - ((+ -)((+ ((+ -) -))(+ -) -) + +) +

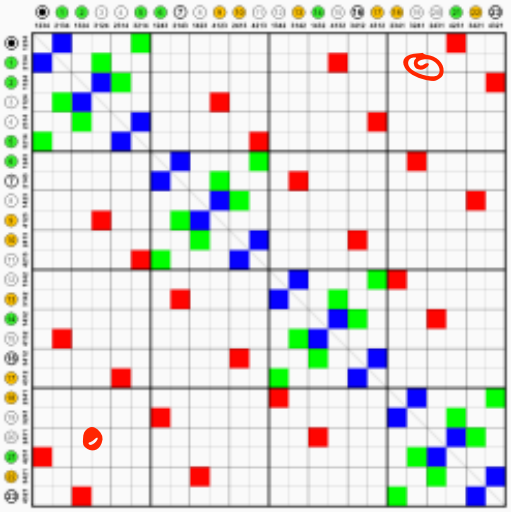
00: - 01: + 10: (11:)

Succinct representation of G

- - ((+ -)((+ ((+ -) -))(+ -) -) + +) +

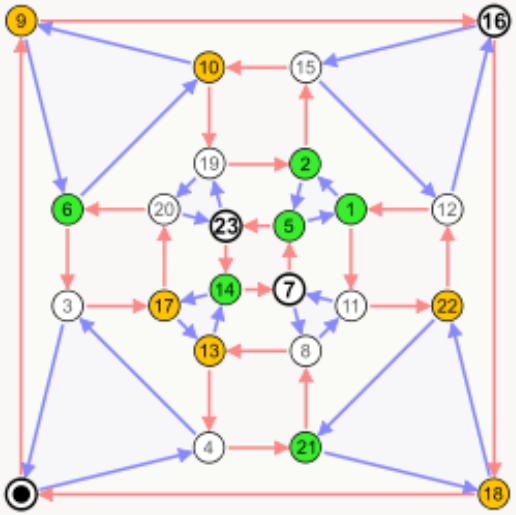


Nauru graph

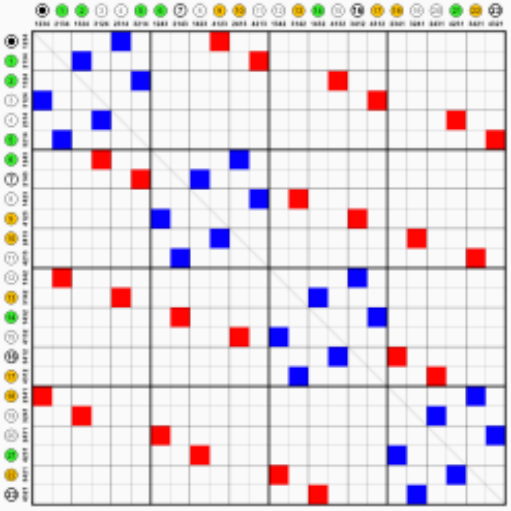


Coordinates are 0–23.
White fields are zeros, colored fields are ones.

Adjacency matrix



Directed Cayley graph of S_4



Coordinates are 0–23.
As the graph is directed, the matrix is not symmetric.

Adjacency list

- Each vertex associated with an (sorted / unsorted) array of adjacent vertices
- More space efficient for sparse graph

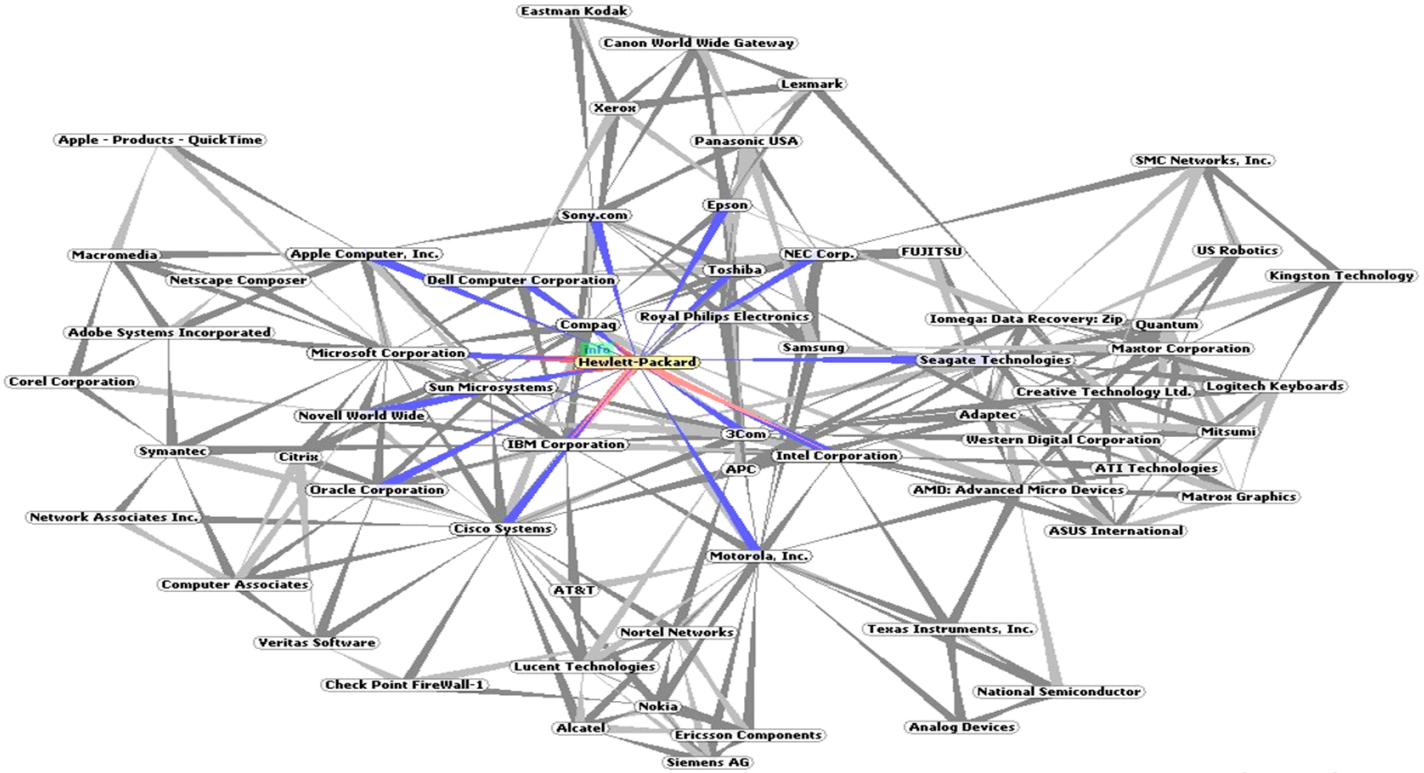
Web Graph representation and compression

Slides modified from L.S. Buriol and D. Donato's

Internet/Web as Graphs

- Graph of the physical layer with routers, computers etc as nodes and physical connections as edges
  - It is limited
  - Does not capture the graphical connections associated with the information on the Internet
- Web Graph where nodes represent web pages and edges are associated with hyperlinks

Web Graph



http://www.touchgraph.com/TGGoogleBrowser.html

Web Graph Considerations

- Graph is highly dynamic
  - Nodes and edges are added/deleted often
  - Content of existing nodes is also subject to change
  - Pages and hyperlinks created on the fly
- Apart from primary connected component there are also smaller disconnected components

Why the Web Graph?

- Example of a large,dynamic and distributed graph
- Possibly similar to other complex graphs in social, biological and other systems
- Reflects how humans organize information (relevance, ranking) and their societies
- Efficient navigation algorithms
- Study behavior of users as they traverse the web graph (e-commerce)

Statistics of Interest

- Size and connectivity of the graph
- Number of connected components
- Distribution of pages per site
- Distribution of incoming and outgoing connections per site
- Average and maximal length of the shortest path between any two vertices (diameter)

Web Graph

A web graph relative to a set of URLs is a directed graph having those URLs as the set of nodes. An arc $u \rightarrow v$ is identified for each hyperlink from a URL *u* towards a URL *v*.

URLs that do not appear either as sources or in more than *T* (4) pages are ignored;

The URLs are normalized by converting hostnames to lower case, cannonicalizes port number, re-introducing them where they need, and adding a trailing slash to all URLs that do not have it.

Main features of Web Graphs

Locality: usually most of the hyperlinks are local, i.e, they point to other URLs on the same host. The literature reports that on average 80% of the hyperlinks are local.

Consecutivity: links within same page are likely to be consecutive respecting to the lexicographic order.

Main features of WebGraphs

Similarity: Pages on the same host tend to have many hyperlinks pointing to the same pages.

Literature

Connectivity Server (1998) – *Digital Systems Reseach Center and Stanford University* – K. Bharat, A. Broder, M. Henzinger, P. Kumar, S. Venkatasubramanian;

Link Database (2001) - *Compaq Systems Research Center* – K. Randall, R. Stata, R. Wickremesinghe, J. Wiener;

WebGraph Framework (2002) – *Universita degli Studi di Milano* – P. Boldi, S. Vigna.

Connectivity Server

- Tool for web graphs visualisation, analysis (connectivity, ranking pages) and URLs compression.
- Used by Alta Vista;
- Links represented by an outgoing and an incoming adjacency lists;

- Composed of:

URL Database: URL, fingerprint, URL-id;

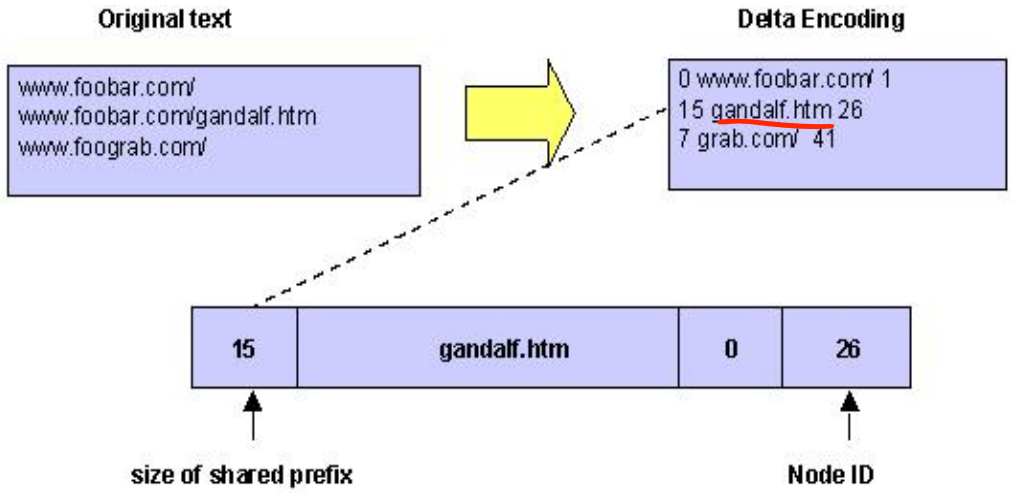
Host Database: group of URLs based on the hostname portion;

Link Database: URL, outlinks, inlinks.

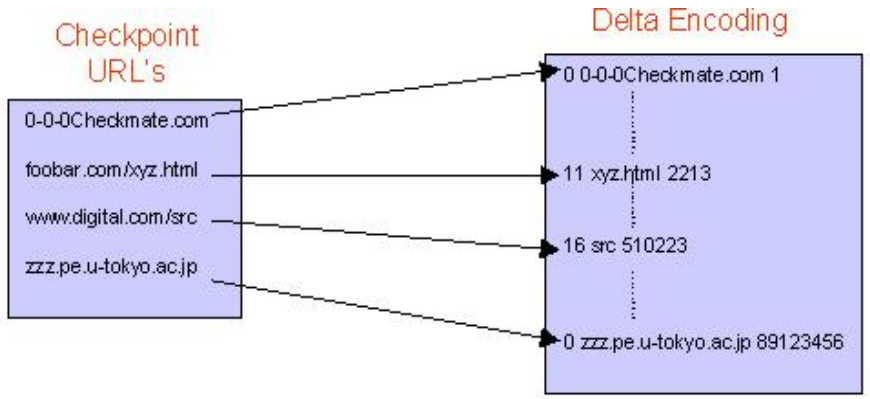
Connectivity Server: URL compression

URLs are sorted lexicographically and stored as a delta encoded entry (70% reduction).

URLs delta encoding



Indexing the delta enconding



Link1: first version of Link Database

No compression: simple representation of outgoing and incoming adjacency lists of links.

Avg. inlink size: 34 bits

Avg. outlink size: 24 bits

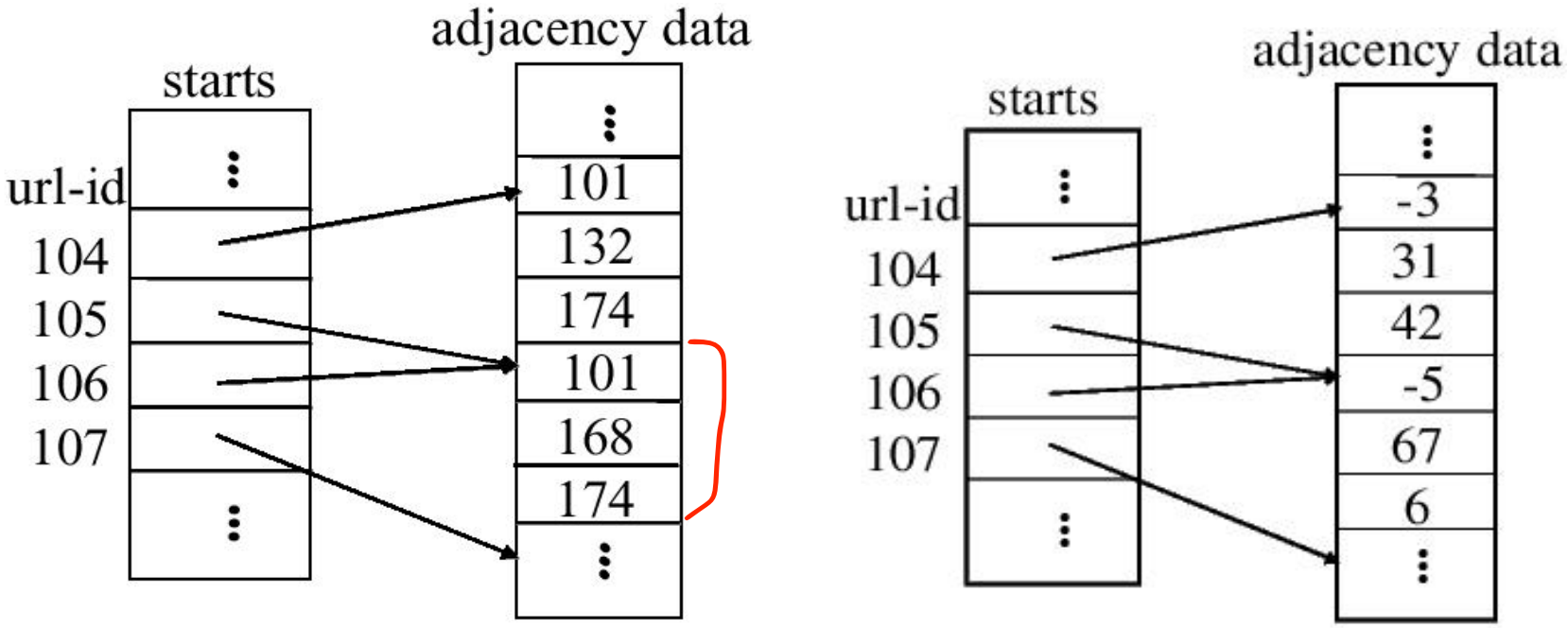
Link2: second version of Link Database

Single list compression and starts compression

Avg. inlink size: 8.9 bits

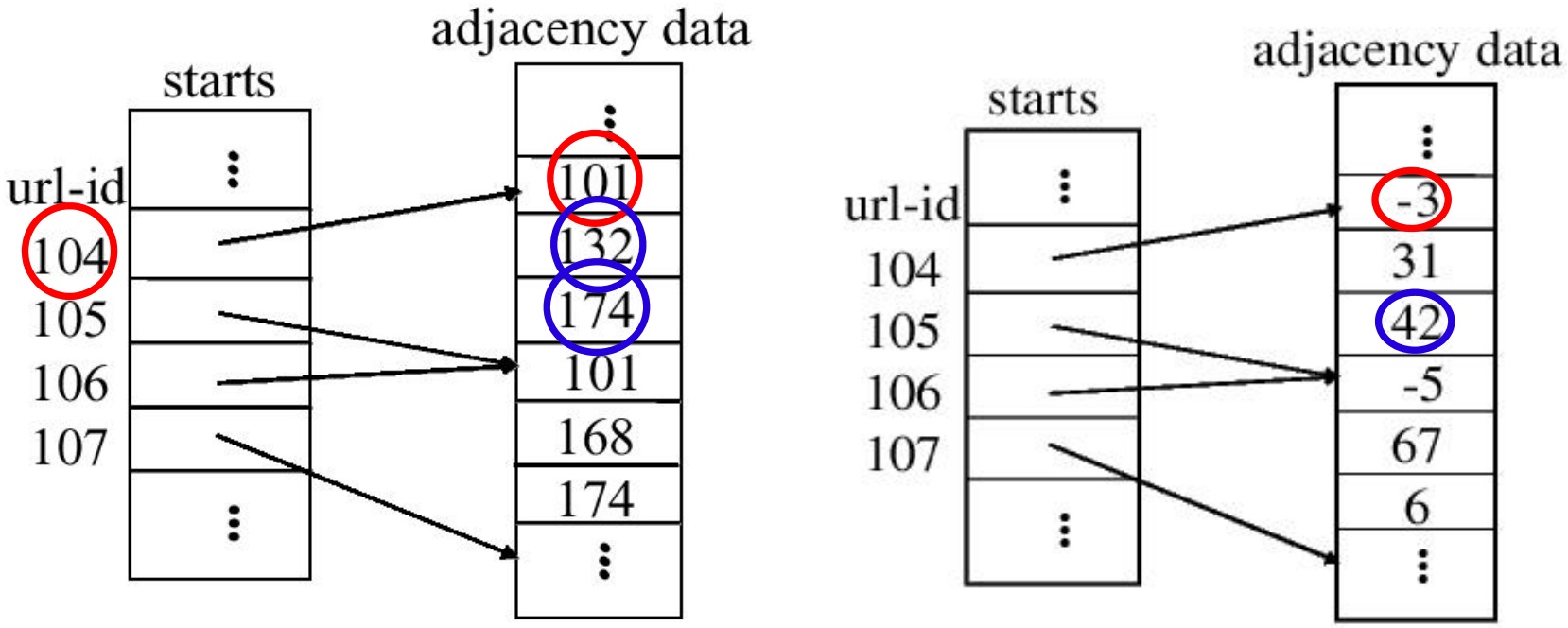
Avg. outlink size: 11.03 bits

Delta Encoding of the Adjacency Lists



Each array element is 32 bits long.

Delta Encoding of the Adjacency Lists



-3 = 101-104 (first item)

42 = 174-132 (other items)

Starts array compression

- The URLs are divided into three partitions based on their degree;

- The literature reports that 74% of the entries are in the low-degree partition.

Link3: third version of Link Database

Interlist compression with representative list

Avg. inlink size: 5.66 bits

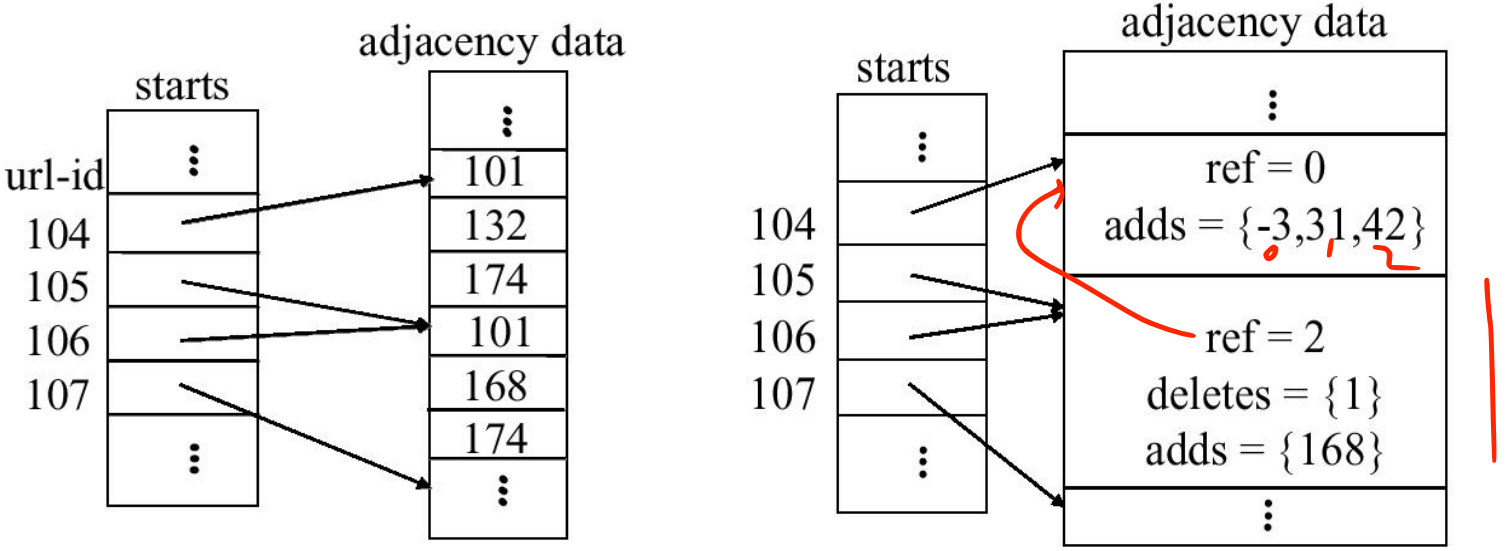
Avg. outlink size: 5.61 bits

Interlist Compression

ref : relative index of the representative adjacency list;

deletes: set of URL-ids to delete from the representative list;

adds: set of URL-ids to add to the representative list.



LimitSelect-K-L: chooses the best representative adjacency list from among the previus *K* (8) URL-ids' adjacency lists and only allows chains of fewer than *L* (4) hops.

ζ -codes (WebGraph Framework)

Interlist compression with representative list

Avg. inlink size: 3.08 bits

Avg. outlink size: 2.89 bits

Using copy lists

Uncompressed adjacency list

Node	Outdegree	Successors
...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...	...	...

Adjacency list with copy lists.

Node	Outd.	Ref.	Copy list	Extra nodes
...	...	...	...	...
15	11	0		13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	1	01110011010	22, 316, 317, 3041
17	0			
18	5	3	11110000000	50
...	...	...	...	...

Each bit on the copy list informs whether the corresponding successor of y is also a successor of x;

The reference list index *ref.* is chosen as the value between 0 and W (window size) that gives the best compression.

Using copy blocks

Adjacency list with copy lists.

Node	Outd.	Ref.	Copy list	Extra nodes
...	...	...	...	...
15	11	0		13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	1	01110011010	22, 316, 317, 3041
17	0			
18	5	3	11110000000	50
...	...	...	...	...

Adjacency list with copy blocks.

Node	Outd.	Ref.	# blocks	Copy blocks	Extra nodes
...	...	...	...	...	...
15	11	0			13, 15, 16, 17, 18, 19, 23, 24 203, 315, 1034
16	10	1	7	0, 0, 2, 1, 1, 0, 0	22, 316, 317, 3041
17	0				
18	5	3	1	4	50
...	...	...	...	...	...

The last block is omitted;

The first copy block is 0 if the copy list starts with 0;

The length is decremented by one for all blocks except the first one.

Conclusions

The compression techniques are specialized for Web Graphs.

The average link size decreases with the increase of the graph.

The average link access time increases with the increase of the graph.

The ζ-codes seems to have the best trade-off between avg. bit size and access time.