COMP9319 Web Data Compression and Search

About a2,

Course revision,

Exam

The most important slide

Raymond, please **“RECORD THIS LECTURE!”**

Announcements

- Final exam details and a sample exam released
- No assignment hurdle
- a2 marks to be released end of wk 10.
- Extra consultations (final consultations) next week for a2 marking & exam qns: schedule will be posted on WebCMS

Assessment

```
a1          = mark for assignment 1       (out of 15)
a2          = mark for assignment 2       (out of 35)
asgts       = a1 + a2                     (out of 50)
exam        = mark for final exam         (out of 50)
okEach      = asgts > 20 && exam > 20     (after scaling)
mark        = a1 + a2 + exam
grade       = HD|DN|CR|PS  if mark >= 50 && okEach
            = FL           if mark <  50 && okEach
            = UF           if !okEach
```

One final exam

- One final exam (50 pts)
- If you are ill on the day of the exam, do not attend the exam – c.f. fit-to-site policy. Apply for special consideration asap.
- It's a 3-hr online exam, and you can complete it within 12hrs (9am – 9pm).
- Read the sample exam to get familiar
- Supp exam will be of a similar form, but probably slightly more challenging

About a2 (Testing process)

1. Run the baseline decode & search to benchmark
2. Run your submission & benchmark
3. Check correctness, memory & speed (usr+sys)

Repeat 1-3 above 3 times over 3 days. As far as your solution runs correctly and within the limits in ANY one of the 3 times, it is fine.

About a2 (Test cases)

DECODE (45%)

1MB

5MB

15MB

SEARCH (55%)

500KB	10 search terms
5MB	10 search terms
50MB	15 search terms
100MB	20 search terms

About a2 (Results so far, only ran twice)

- Around 75 submissions
- 6 have makefile or compilation errors (failed to produce binaries by “make”)
- 41 correct and beat my baseline implementation + margin
- 17 beat mine on every test (without the margin)
- 3 produced lossy decoding
- 12 had incorrect search on some tests
- 5 submitted “too” similar solutions
- After discussing with the tutors, it’s already a lenient marking scheme.

About a2 (Baseline optimizations)

- Using blocks (not byte by byte, but a sensible block size) whenever read/write to file
- Read file once to construct C and Occ (with gaps)
- Read file once when one char is decoded
- Read file twice when one char is matched

About a2 (**Further** optimizations)

- Dynamic gap size & no gap for small files
- Alphabet size: 4 not 5
- Buffering for bits instead of bytes
- Reduce the number of counting by halving the gap size for free

Revision

COMP9319 – The foundations of

- how different compression tools work.
- how to manage a large amount of data on small devices.
- how to search gigabytes, terabytes or petabytes of data.
- how to perform full text search efficiently without added indexing.
- how to query distributed data repositories efficiently (optional).

Course Aims

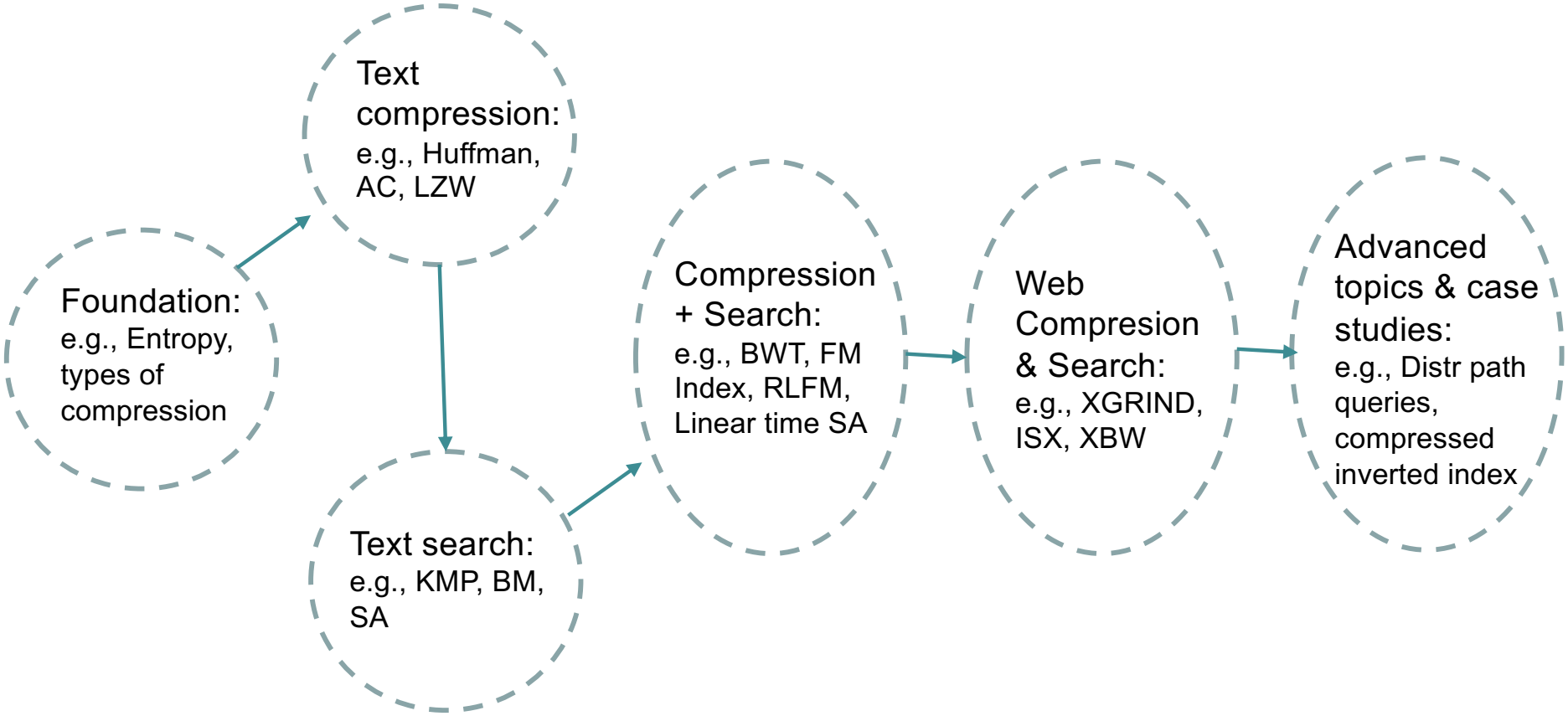
As the amount of Web data increases, it is becoming vital to not only be able to search and retrieve this information quickly, but also to store it in a compact manner. This is especially important for mobile devices which are becoming increasingly popular. Without loss of generality, within this course, we assume Web data (excluding media content) will be in XML and its like (e.g., HTML, JSON).

This course aims to introduce the concepts, theories, and algorithmic issues important to Web data compression and search. The course will also introduce the most recent development in various areas of Web data optimization topics, common practice, and its applications.

Summarised schedule

- 1. Compression
- 2. Search
- 3. Compression + Search
- 4. “Compression + Search” on Web text data
- 5. Selected advanced topics

Topics


Foundation:
e.g., Entropy, types of compression
Text compression:
e.g., Huffman, AC, LZW
Text search:
e.g., KMP, BM, SA
Compression + Search:
e.g., BWT, FM Index, RLFM, Linear time SA
Web Compresion & Search:
e.g., XGRIND, ISX, XBW
Advanced topics & case studies:
e.g., Distr path queries, compressed inverted index

Topic Snapshots

Questions to discuss (www)

- What (is data compression)
- Why (data compression)
- Where

Compression

- Minimize amount of information to be stored / transmitted
- Transform a sequence of characters into a new bit sequence
  - same information content (for lossless)
  - as short as possible

Terminology (Types)

- Block-block
  - source message and codeword: fixed length
  - e.g., ASCII
- Block-variable
  - source message: fixed; codeword: variable
  - e.g., Huffman coding
- Variable-block
  - source message: variable; codeword: fixed
  - e.g., LZW
- Variable-variable
  - source message and codeword: variable
  - e.g., Arithmetic coding

Run-length coding

- Run-length coding (encoding) is a very widely used and simple compression technique
  - does not assume a memoryless source
  - replace runs of symbols (possibly of length one) with pairs of (symbol, *run-length*)

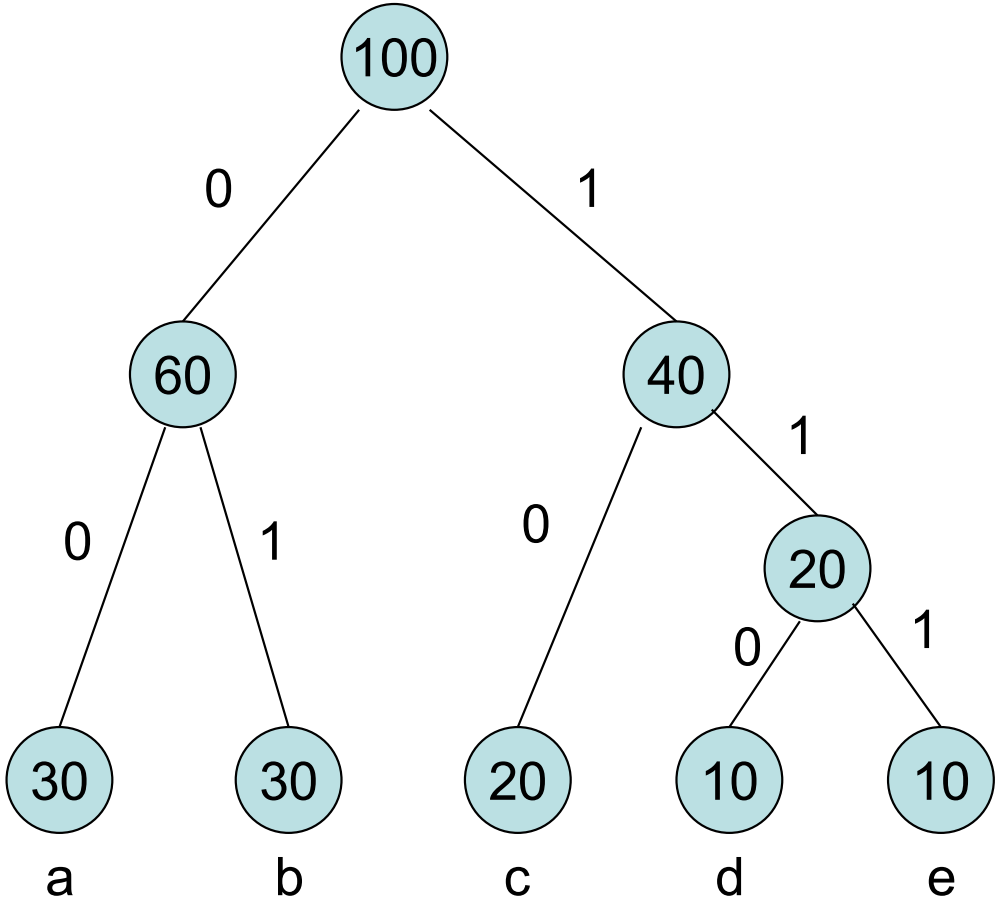
Entropy

- What is the minimum number of bits per symbol?
- Answer: Shannon's result – theoretical minimum average number of bits per code work is known as Entropy (H)

$$\sum_{i=1}^{n} - p(s_i) \log_2 p(s_i)$$

Huffman coding

S	Freq	Huffman
a	30	00
b	30	01
c	20	10
d	10	110
e	10	111



Arithmetic coding (encode)

New Character	Low value	High Value
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
SPACE	0.25720	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

Arithmetic coding (decode)

Encoded Number	Output Symbol	Low	High	Range
0.2572167752	B	0.2	0.3	0.1
0.572167752	I	0.5	0.6	0.1
0.72167752	L	0.6	0.8	0.2
0.6083876	L	0.6	0.8	0.2
0.041938	SPACE	0.0	0.1	0.1
0.41938	G	0.4	0.5	0.1
0.1938	A	0.2	0.3	0.1
0.938	T	0.9	1.0	0.1
0.38	E	0.3	0.4	0.1
0.8	S	0.8	0.9	0.1
0.0				

LZW Compression

```
w = NIL;
  while ( read a character k )
      {
        if wk exists in the dictionary
         w = wk;
        else
          add wk to the dictionary;
          output the code for w;
          w = k;
      }
```

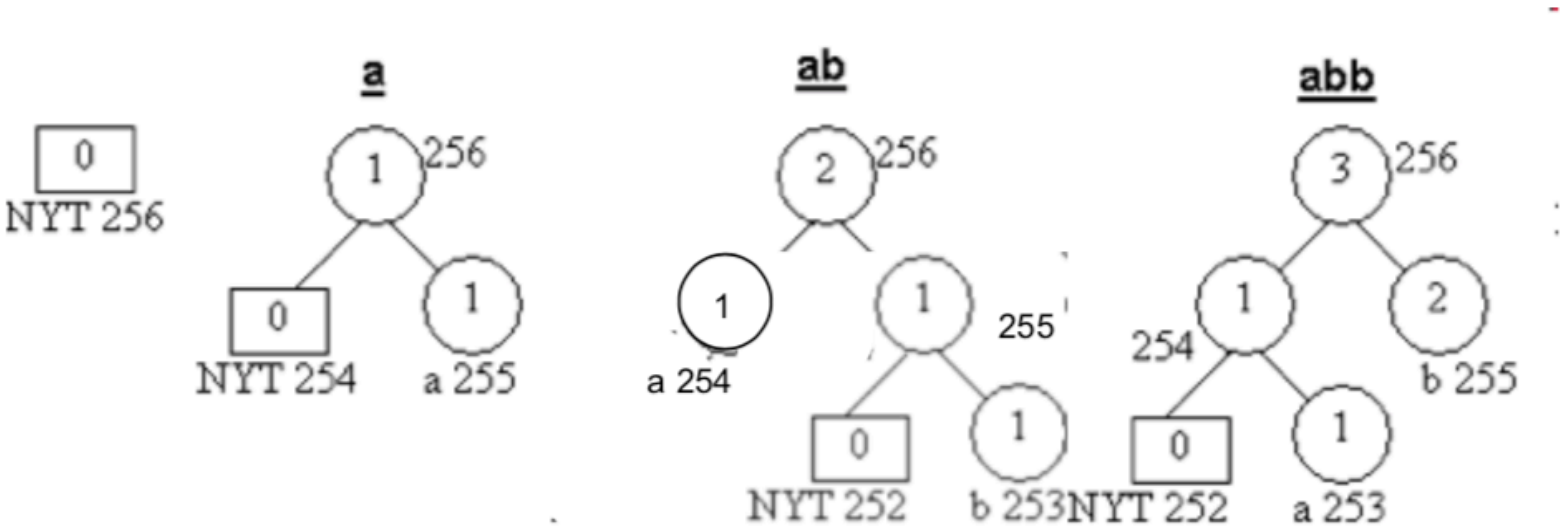
LZW Decompression

```
read a character k;
  output k;
  w = k;
  while ( read a character/code k )
      {
        entry = dictionary entry for k;
        output entry;
        add w + entry[0] to dictionary;
        w = entry;
      }
```

Adaptive Huffman

abbbbba: 01100001011000100110001001100010011000100110001001100001

abbbbba: 011000010011000101111101



a: 01100001

b: 01100010

Modified from Wikipedia

BWT(S)

function BWT (string s)

create a table, rows are all possible rotations of s

sort rows alphabetically

return (last column of the table)

InverseBWT(S)

```
function inverseBWT (string s)
  create empty table

  repeat length(s) times
     insert s as a column of table before first
          column of the table   // first insert creates
          first column
     sort rows of the table alphabetically
  return (row that ends with the 'EOF' character)
```

Other ways to reverse BWT

Consider L=BWT(S) is composed of the symbols $V_0 \ldots V_{N-1}$, the transformed string may be parsed to obtain:

- The number of symbols in the substring $V_0 \ldots V_{i-1}$ that are identical to V_i. (i.e., Occ[ ])
- For each unique symbol, V_i, in L, the number of symbols that are lexicographically less than that symbol. (i.e., C[ ])

Move to Front (MTF)

- Reduce entropy based on local frequency correlation
- Usually used for BWT before an entropy-encoding step
- Author and detail:
  - Original paper at cs9319/Papers
  - http://www.arturocampos.com/ac_mtf.html

BWT compressor vs ZIP

ZIP (i.e., LZW based)

BWT+RLE+MTF+AC

File Name	Raw Size	PKZIP Size	PKZIP Bits/Byte	BWT Size	BWT Bits/Byte
bib	111,261	35,821	2.58	29,567	2.13
book1	768,771	315,999	3.29	275,831	2.87
book2	610,856	209,061	2.74	186,592	2.44
geo	102,400	68,917	5.38	62,120	4.85
news	377,109	146,010	3.10	134,174	2.85
obj1	21,504	10,311	3.84	10,857	4.04
obj2	246,814	81,846	2.65	81,948	2.66

From http://marknelson.us/1996/09/01/bwt/

Pattern Matching

- Brute Force
- Boyer Moore
- KMP

Regular expressions

- L(**001**) = {001}
- L(**0+10***) = { 0, 1, 10, 100, 1000, 10000, … }
- L(**0*10***) = {1, 01, 10, 010, 0010, …} i.e. {w | w has exactly a single 1}
- L(∑∑)* = {w | w is a string of even length}
- L((**0(0+1**))*) = { ε, 00, 01, 0000, 0001, 0100, 0101, …}
- L((**0+ε)(1+ ε**)) = {ε, 0, 1, 01}
- L(1Ø) = Ø ; concatenating the empty set to any set yields the empty set.
- Rε = R
- R+Ø = R

- Note that R+ε may or may not equal R (we are adding ε to the language)
- Note that RØ will only equal R if R itself is the empty set.

Theory of DFAs and REs

- RE. Concise way to describe a set of strings.
- DFA. Machine to recognize whether a given string is in a given set.
- **Duality**: for any DFA, there exists a regular expression to describe the same set of strings; for any regular expression, there exists a DFA that recognizes the same set.

DFA to RE: State Elimination

- Eliminates states of the automaton and replaces the edges with regular expressions that includes the behavior of the eliminated states.
- Eventually we get down to the situation with just a start and final node, and this is easy to express as a RE

Signature files

- Definition
  - Word-oriented index structure based on hashing.
  - Use liner search.
  - Suitable for not very large texts.
- Structure
  - Based on a Hash function that maps words to bit masks.
  - The text is divided in blocks.
    - **Bit mask of block is obtained by bitwise ORing the signatures of all the words in the text block.**
    - **Word not found, if no match between all 1 bits in the query mask and the block mask.**

Suffix tree

Given a string s a suffix tree of s is a compressed trie of all suffixes of s

To make these suffixes prefix-free we add a special character, say $, at the end of s

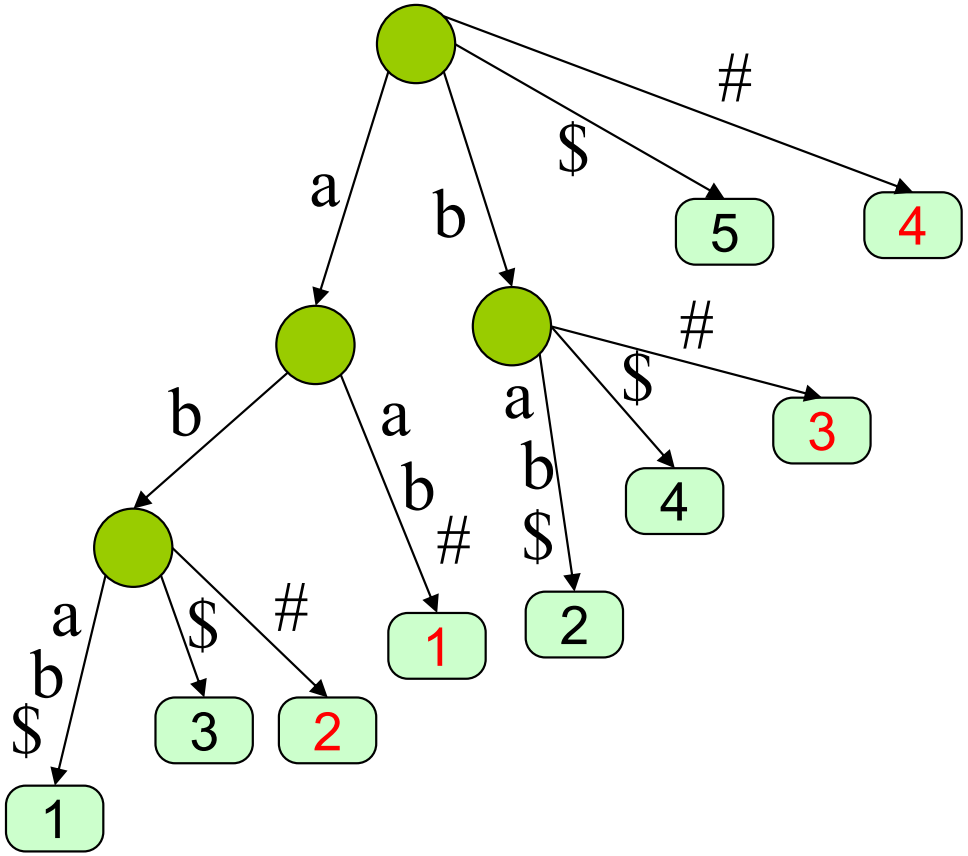
Generalized suffix tree (Example)

Let s_1=abab and s_2=aab here is a generalized suffix tree for s_1 and s_2

{

$	#
b$	b#
ab$	ab#
bab$	aab#
abab$	

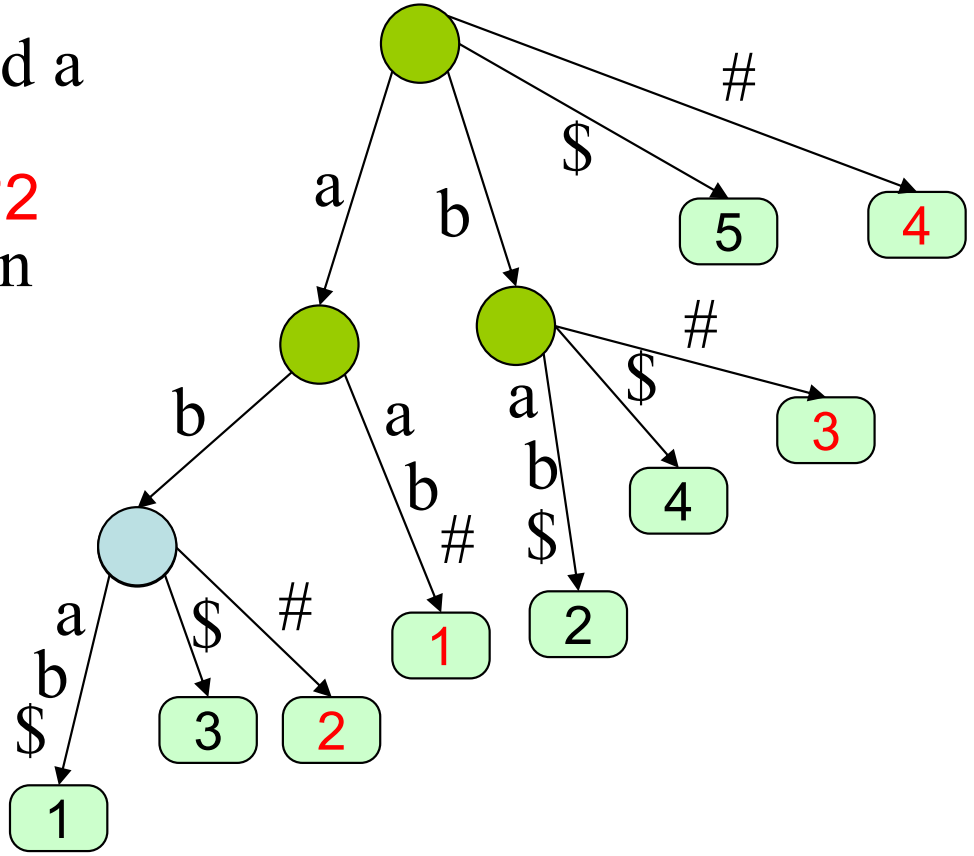
}



Longest common substring (of two strings)

Every node with a leaf descendant from string S_1 and a leaf descendant from string S_2 represents a maximal common substring and vice versa.

Find such node with largest “string depth”



Suffix array

- We loose some of the functionality but we save space.

Let s = abab

Sort the suffixes lexicographically:
ab, abab, b, bab

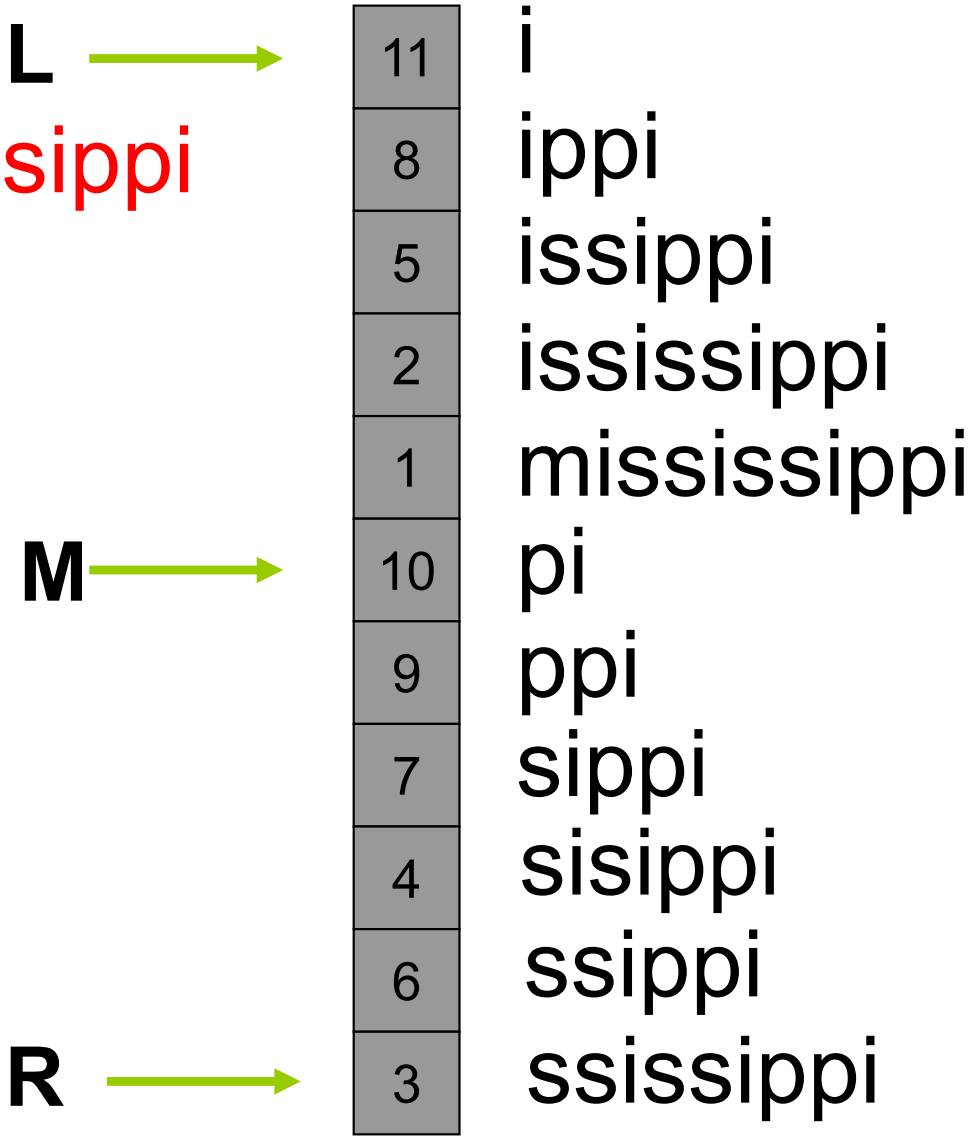
The suffix array gives the indices of the suffixes in sorted order

3	1	4	2

Example

Let S = mississippi

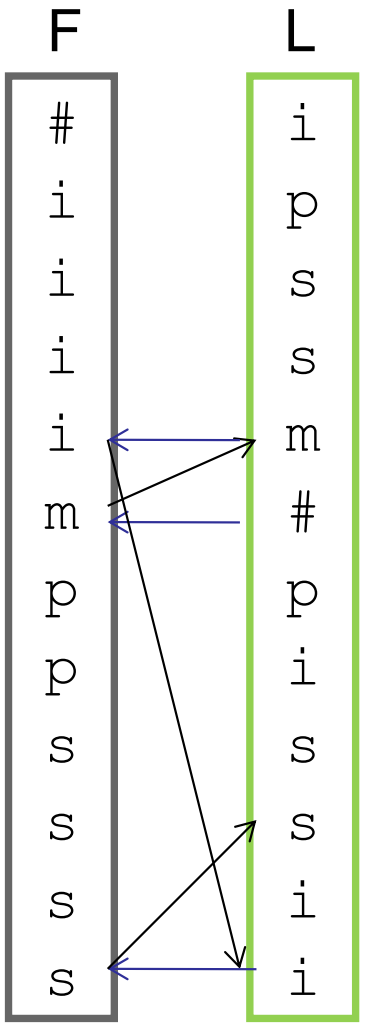
Let P = issa



BURROWS-WHEELER TRANSFORM

Reminder: Recovering T from L

1. Find F by sorting L
2. First char of T? m
3. Find m in L
4. L[i] precedes F[i] in T. Therefore we get
   mi
5. How do we choose the correct i in L?
   - The i's are in the same order in L and F
   - As are the rest of the char's
6. i is followed by s: mis
7. And so on….



BACKWARD-SEARCH EXAMPLE

- **P = pssi**
  - i = 3
  - c = 's'
  - First = C['s'] + Occ('s',1) +1 = 8+0+1 = 9
  - Last = C['s'] + Occ('s',5) = 8+2 = 10
  - (Last – First + 1) = 2

First

Last

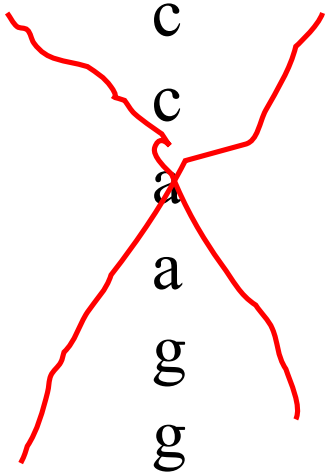
	F	L	
	#	mississipp	i	1
	i	#mississip	p	2
	i	ppi#missis	s	3
	i	ssippi#mis	s	4
	i	ssissippi#	m	5
	m	ississippi	#	6
	p	i#mississi	p	7
	p	pi#mississ	i	8
	s	ippi#missi	s	9
	s	issippi#mi	s	10
	s	sippi#miss	i	11
	s	sissippi#m	i	12

C[ ] =	1	5	6	8
	i	m	p	s

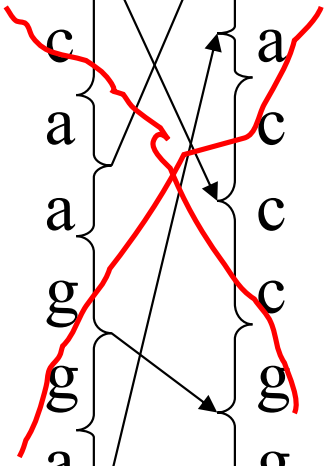
Algorithm backward_search($P[1,p]$)

(1) $i \leftarrow p$, $c \leftarrow P[p]$, First $\leftarrow C[c]+1$, Last $\leftarrow C[c+1]$;
(2) **while** ((First $\leq$ Last) **and** $(i \geq 2)$) **do**
(3) $c \leftarrow P[i-1]$;
(4) First $\leftarrow C[c] + \text{Occ}(c, \text{First}-1) + 1$;
(5) Last $\leftarrow C[c] + \text{Occ}(c, \text{Last})$;
(6) $i \leftarrow i-1$;
(7) **if** (Last $<$ First) **then return** "no rows prefixed by $P[1,p]$" **else return** $\langle$First, Last$\rangle$.

Compressed SA: Run-Length FM-index...



L	B	S
c	1	c
c	0	a
c	0	g
a	1	a
a	0	t
g	1	
g	0	
a	1	
t	1	
t	0	



L →	F	B’
c	a	1
c	a	0
c	a	1
a	c	1
a	c	0
g	c	0
g	g	1
a	g	0
t	t	1
t	t	0

Changes to formulas

- Recall that we need to compute $C_T[c]+rank_c(L,i)$ in the backward search.
- **Theorem:** $C[c]+rank_c(L,i)$ is equivalent to $select_1(B',C_S[c]+1+rank_c(S,rank_1(B,i)))-1$, when $L[i] \neq c$, and otherwise to $select_1(B',C_S[c]+rank_c(S,rank_1(B,i)))+ i\text{-}select_1(B,rank_1(B,i))$.

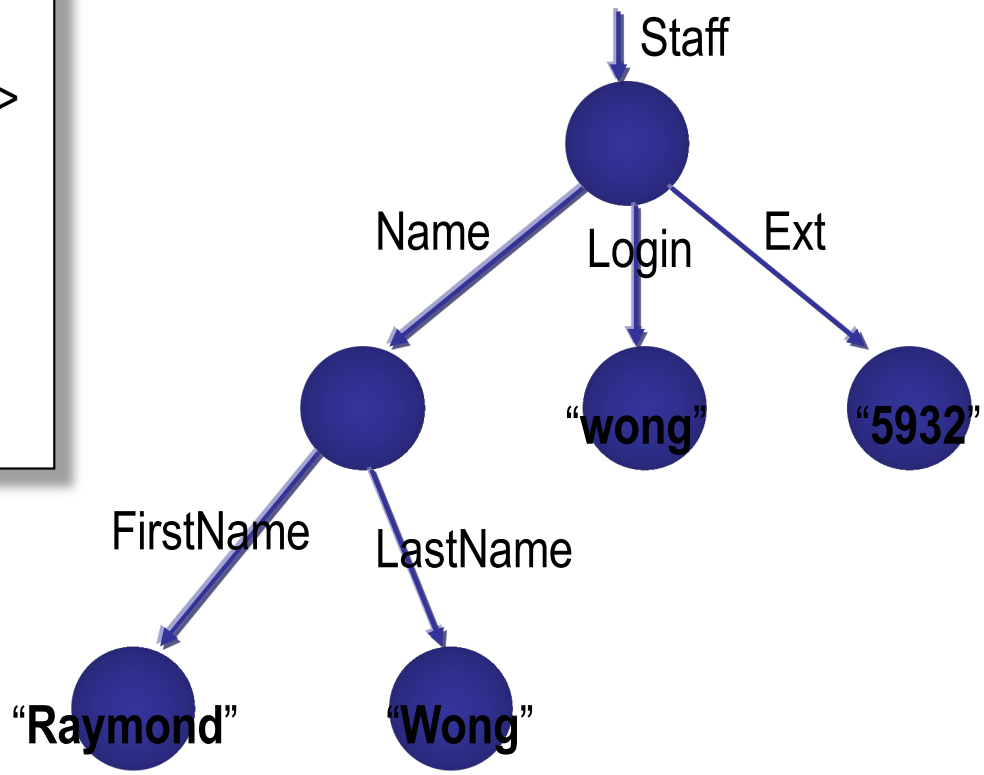
Linear time suffix array construction

- Consider the popular example string S:
- **`bananainpajamas$`**

1. Construct the suffix array of S using the linear time algorithm
2. Then compute the BWT(S)
3. What's the relationship between the suffix array and BWT ? (e.g., SA -> BWT vs BWT -> SA)

Semistructured Data: Tree/HTML/XML/JSON/RDF...

```
<Staff>
   <Name>
      <FirstName> Raymond </FirstName>
      <LastName>  Wong </LastName>
   </Name>
   <Login> wong </Login>
   <Ext> 5932 </Ext>
</Staff>
```



XPath for XML

/bib/book[@price < “60”]

/bib/book[author/@age < “25”]

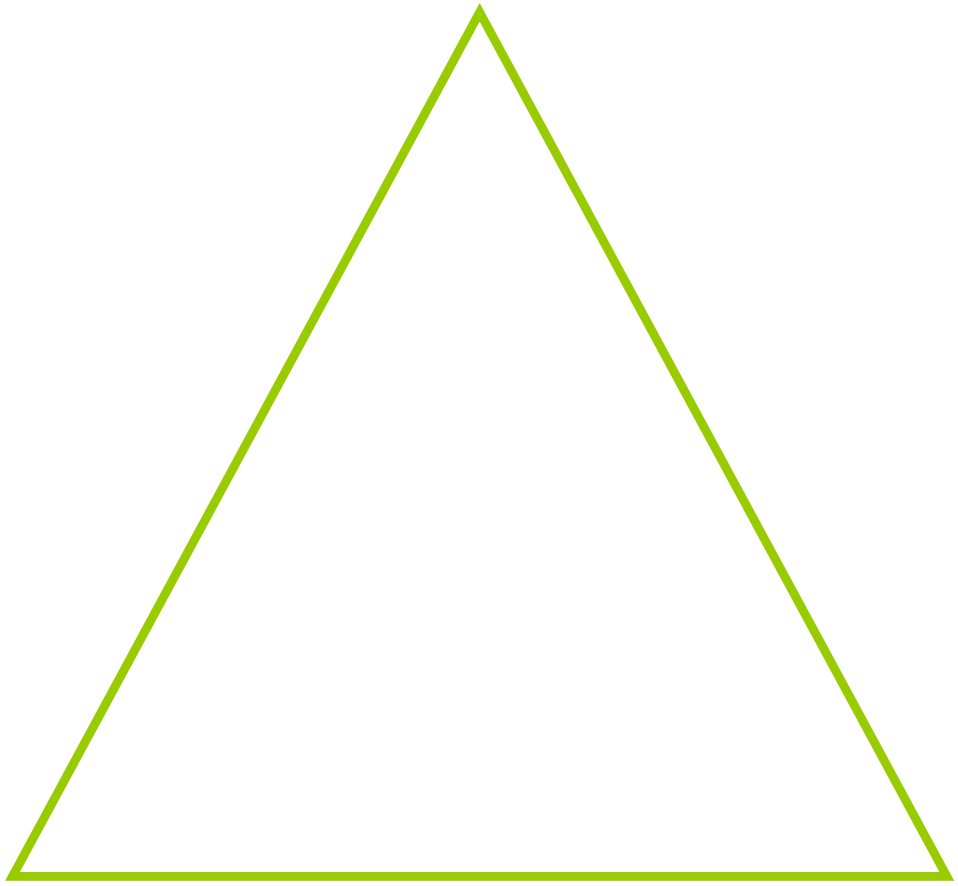
/bib/book[author/text()]

Path query evaluation

Top-down

Bottom-up

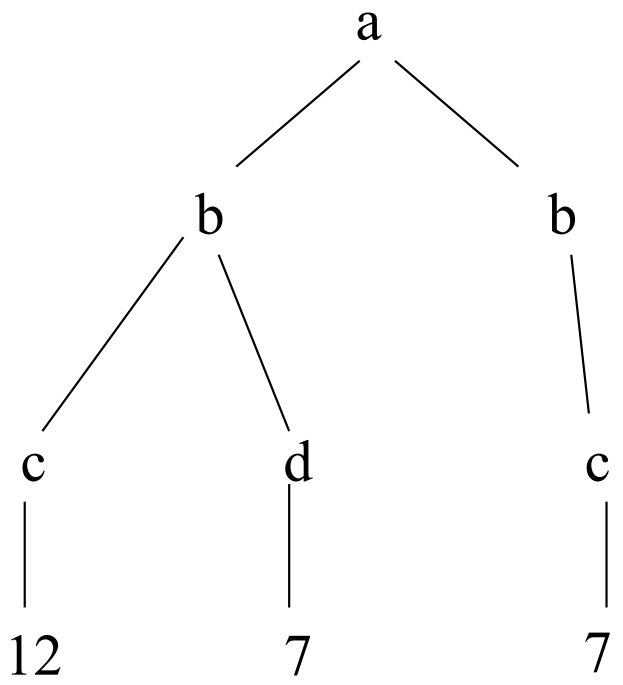
Hybrid



XPath evaluation

<a><c>12</c><d>7</d><c>7</c>

/ a / b [c = “12”]



Path indexing

- Traversing graph/tree almost = query processing for semistructured / XML data
- Normally, it requires to traverse the data from the root and return all nodes X reachable by a path matching the given regular path expression
- Motivation: allows the system to answer regular path expressions without traversing the whole graph/tree

Two techniques

- Based on the idea of language-equivalence
- Data Guide

XMill

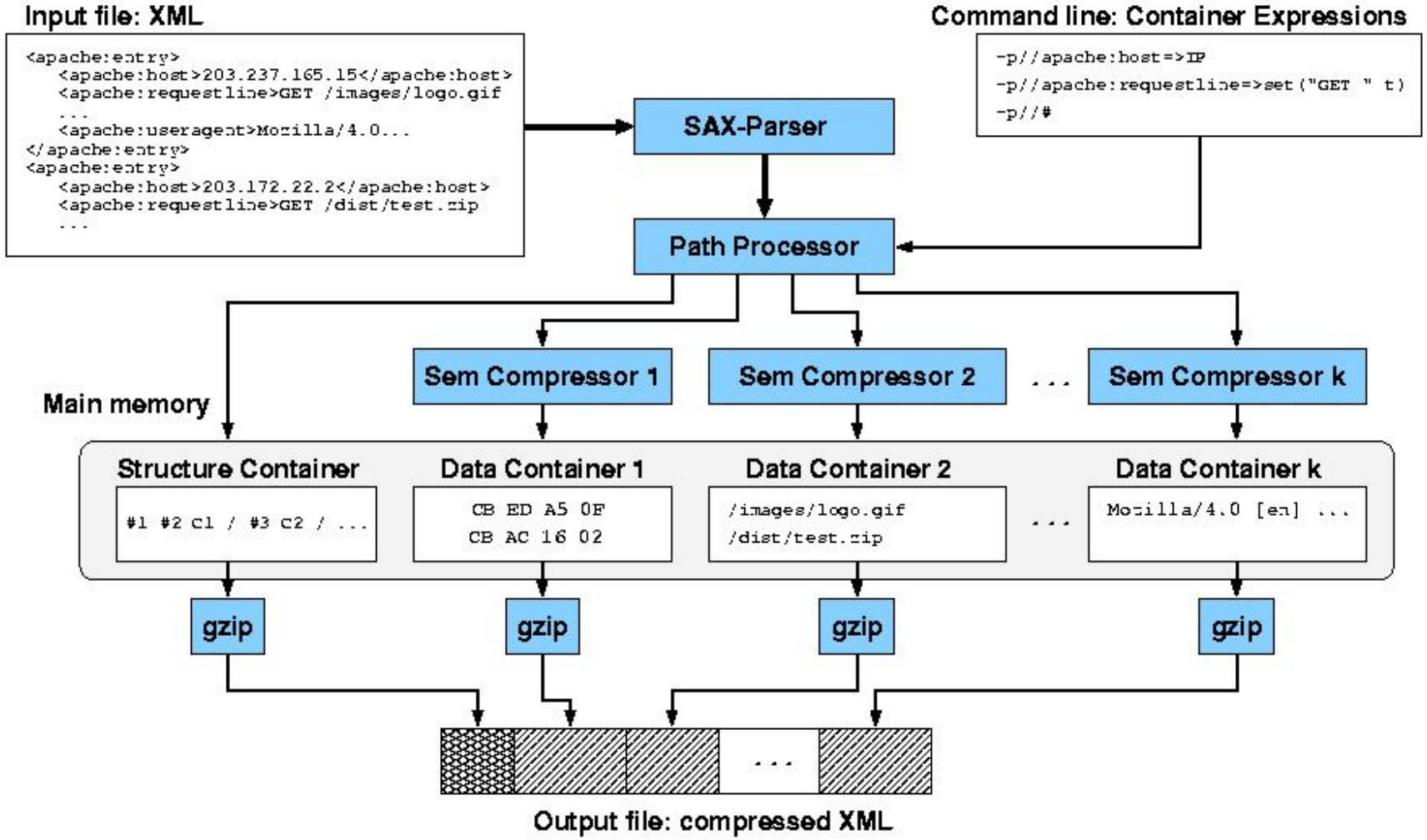


Figure 4: Architecture of the Compressor

XGRIND

Original Fragment:

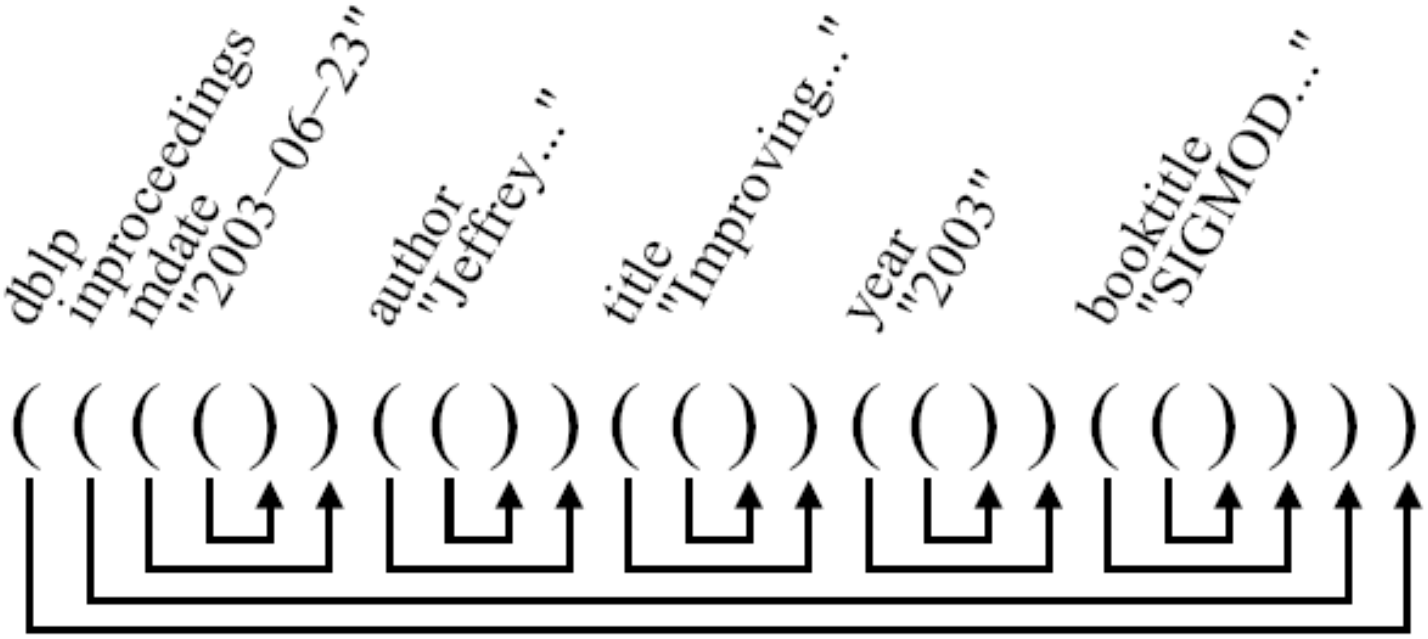
```
<student name=“Alice“>
        <a1>78</a1>
        <a2>86</a2>

    <midterm>91</midterm>
        <project>87</project>
</student>
```

Compressed Fragment:

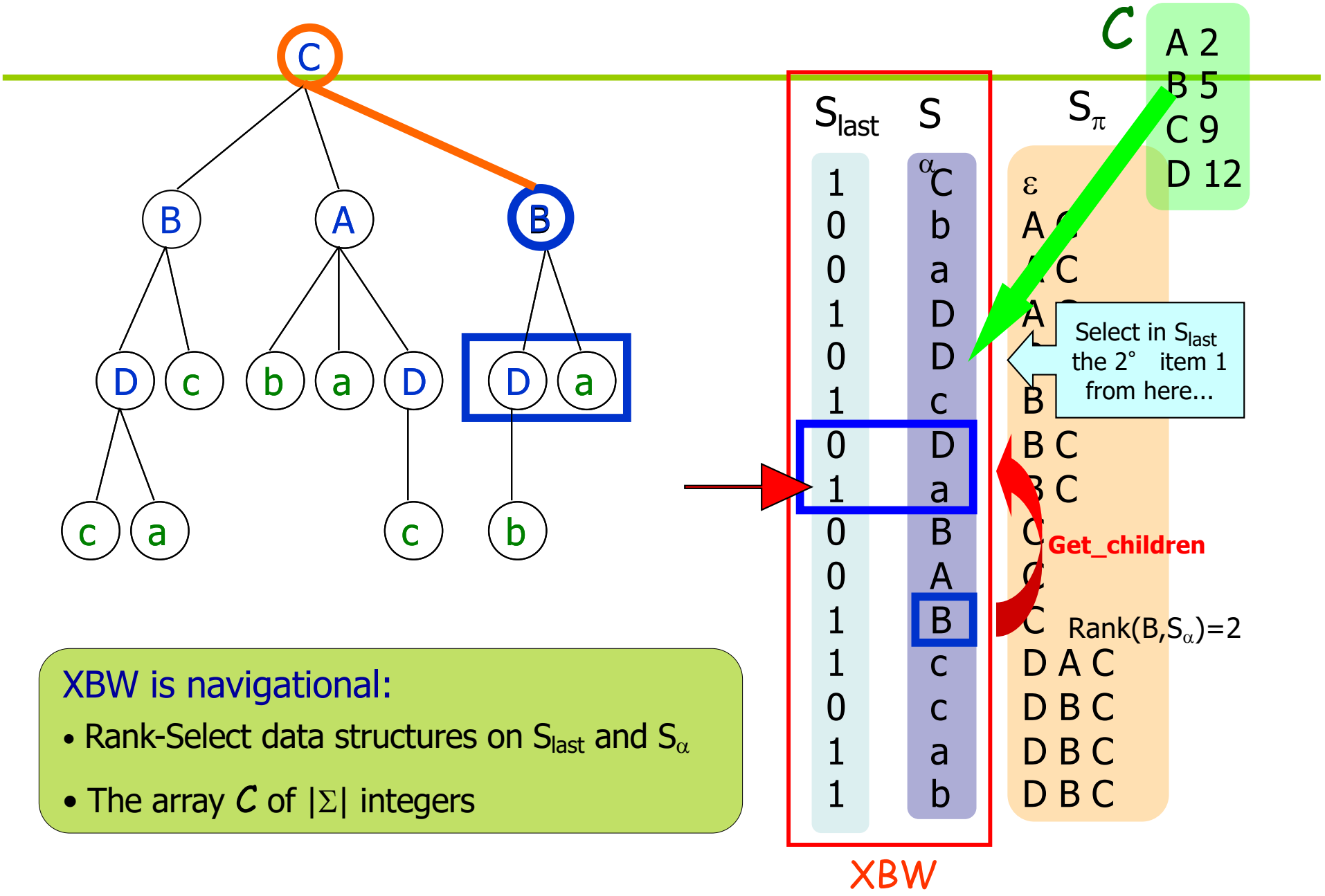
```
T0 A0 nahuff(Alice)
    T1 nahuff(78) /
    T2 nahuff(86) /
    T3 nahuff(91) /
    T4 nahuff(87) /
/
```

ISX: 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

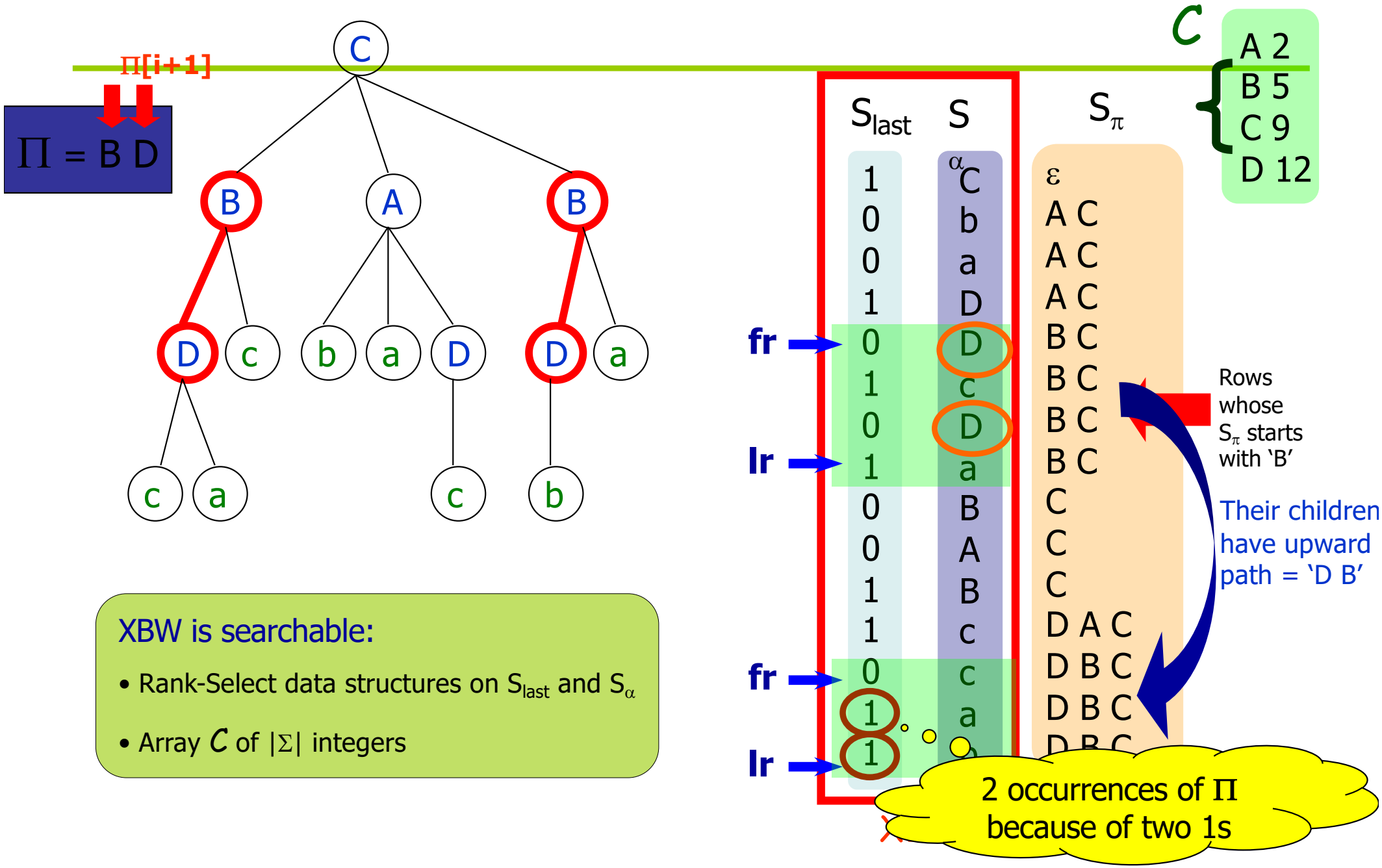
XBW is navigational



XBW is navigational:

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

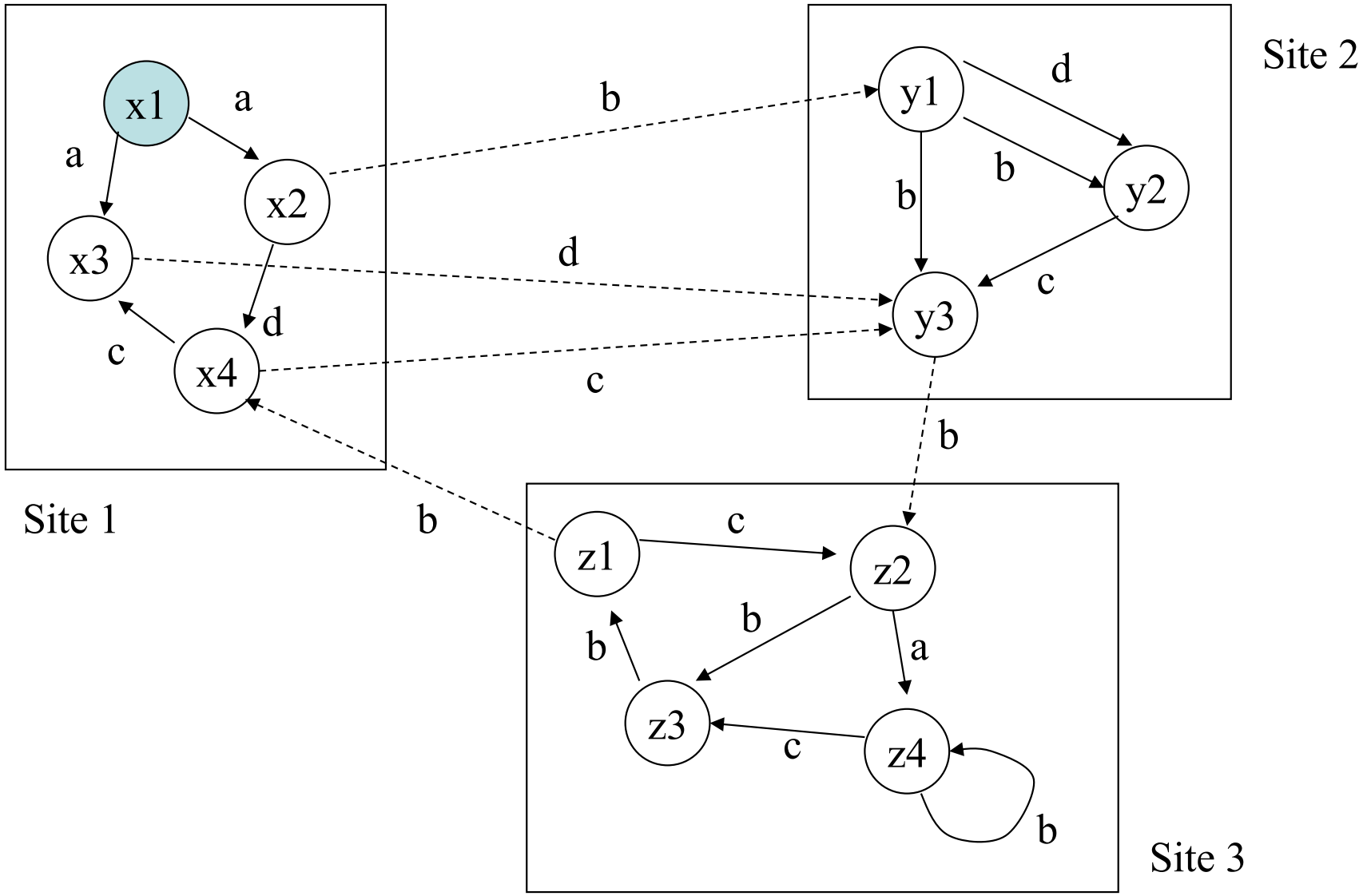
XBW is searchable (count subpaths)



XBW is searchable:

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

Distributed path queries

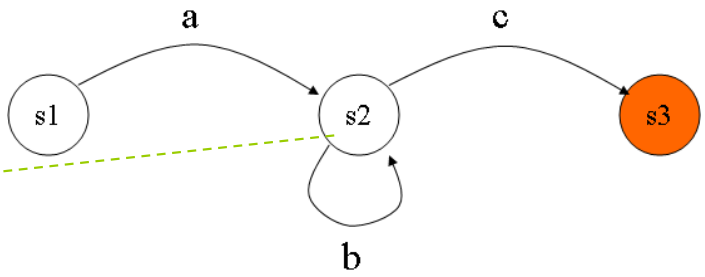


Distributed path query processing

- Given a query, we compute its automaton
- Send it to each site
- Start an identical process at each site
- Compute two sets Stop(n, s) and Result(n, s)
- Transmits the relations to a central location and get their union

Stop and Result sets at site 2

Start	Stop
(y1, s2)	(z2, s2)
(y3, s2)	(z2, s2)



Note: here s2 is State 2, not Site 2

Start	Result
(y1, s2)	y3
(y1, s3)	y1
(y3, s3)	y3

Compression for inverted index

- First, we will consider space for dictionary
  - Main motivation for dictionary compression: make it small enough to keep in main memory
- Then for the postings file
  - Motivation: reduce disk space needed, decrease time needed to read from disk
  - Note: Large search engines keep significant part of postings in memory
- We will devise various compression schemes for dictionary and postings.
- VB code, Gamma code

Web Graph Compression

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.

Case studies: e.g., Content optimization

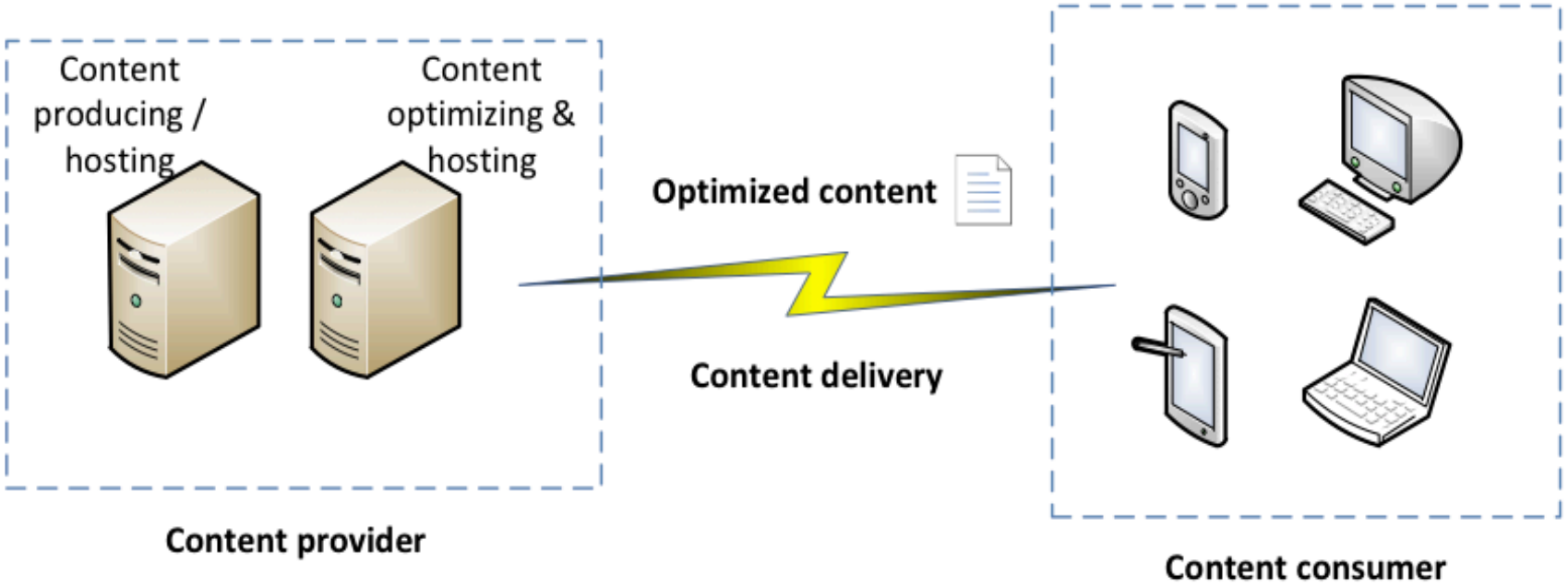


Figure 2. Delivery of content with content optimization

Covered Topics

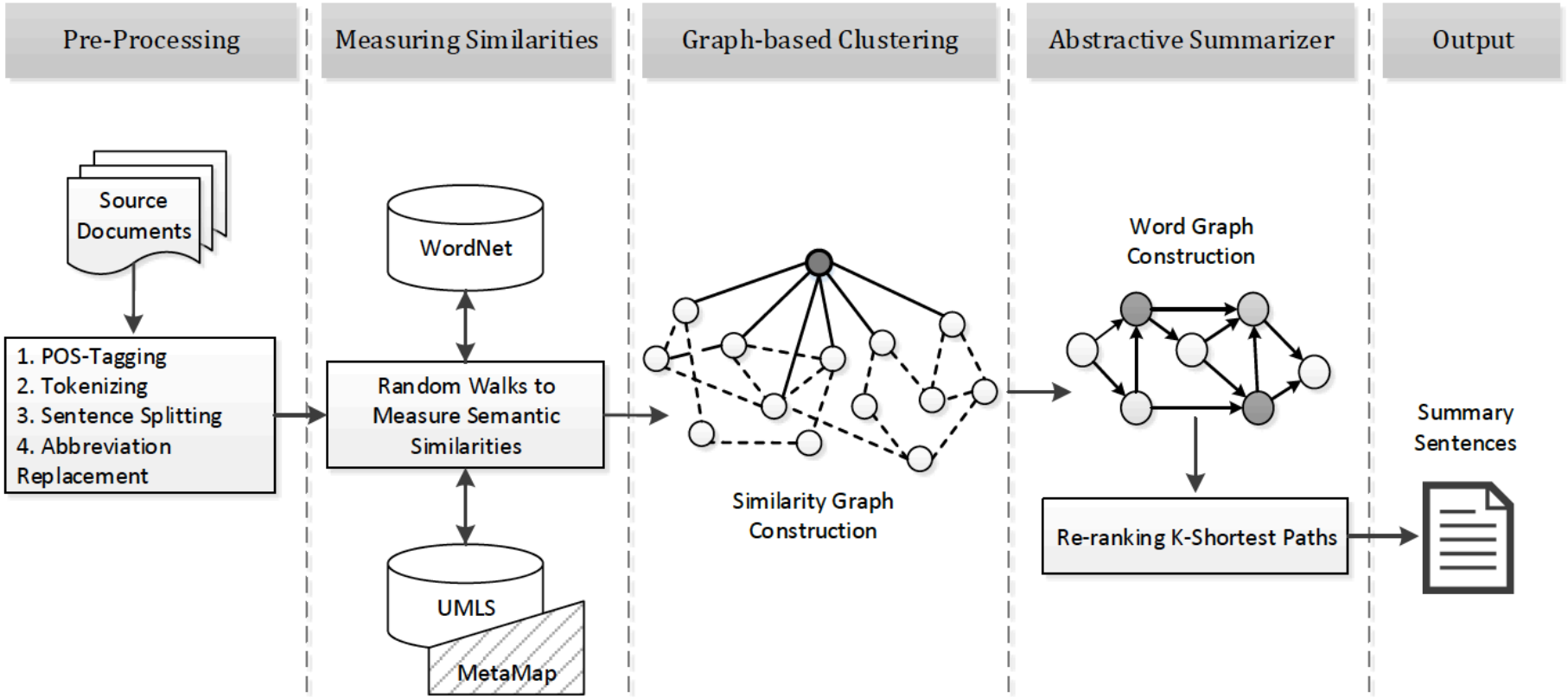
1. Entropy & basic compressions (RLE, Huffman, AC, LZW, Adaptive Huffman)
2. Pattern matching (Brute Force, KMP, BM); Regular Expression & Finite Automata; Inverted Index & Signature Files.
3. Suffix tree, Suffix array, BWT, MTF, FM Index, RLFM, O(n) SA construction.
4. Semistructured Data, XML & XPath; Path indexing; Tree/XML compressions (XMill, XGrind, ISX, XBW).
5. Querying distributed data.
6. Inverted index & its compression; variable length coding; Web graph compression.
7. Case studies: Google Bigtable; Cloud data optimization.

What have not been covered?

- Multimedia data compression (e.g., images, videos)
- Lossy compression

Future Directions

- Lossy (text) compression: summarization, topic modeling, …



Learning outcomes

- have a good understanding of the fundamentals of text compression
- be introduced to advanced data compression techniques such as those based on Burrows Wheeler Transform
- have programming experience in Web data compression and optimization
- have a deep understanding of XML and selected XML processing and optimization techniques
- understand the advantages and disadvantages of data compression for Web search
- have a basic understanding of XML distributed query processing
- appreciate the past, present and future of data compression and Web data optimization

Learning outcomes (a compressed version)

- have a different perception on:
  - ”information” (e.g., entropy)
  - string manipulation (compare, substring, etc)
  - semistructured text data manipulation

- have experience on practical considerations on:
  - efficient algorithms vs efficient implementations
  - computations with limited resources (e.g., when dealing with big text data)

Finally

MyExperience:

Much appreciated if you can provide some constructive feedback !!

The End – *my last lecture exercise for you* ☺

kuo$cdogl