8. Querying Structured Text

- How to query the structure and contents of structured documents?
  - 8.1 Text-region algebra, and search tool sgrep
    - a simple model and tool for content retrieval of arbitrary structured text files, based on concrete document syntax
  - 8.2 W3C XML query language XQuery
    - a rich query and data manipulation language for all types of XML data sources, based on conceptual content of XML documents ("XML Information Set")

8.1 Region algebra (and sgrep)

- the model: region algebra
  - relatively low-level but appropriate for e.g. XML
  - documents seen as contiguous portions called regions
- an implementation: sgrep
  - "structured grep", command-line based search tool
    - version 0.99 (Unix/Linux): basic features
    - version 1.92a (Unix/Linux/Win32): indexing, SGML/XML/HTML support, other additional features
  - extracts portions of files/input streams

Region algebra: Background

- Pat™ (University of Waterloo; OpenText)
  - efficient full-text index (suffix array)
  - match-point delimited regions
    - result sets of non-overlapping regions only; overlapping results represented as their start points
    - semantic problems
- "generalized concordance lists"
  - Clarke, Cormack and Burkowski 1995
  - overlaps but no nesting in results

Nested region algebra

  - retrieval of document components as dynamically defined regions, based on their mutual ordering and nesting
  - no restrictions on the length, overlapping or nesting of regions
  - no restrictions on document formats
    - regular markup formats (a'la XML) most appropriate

Basics of region algebra data model

- Set of positions \( U = \{0, \ldots, n\} \) comprising a text of length \( n \)
  - characters/bytes (sgrep), or words
- Regions: contiguous (non-empty) sequences of positions (normally fragments of files)
  - typically occurrences of a string, or document elements
  - region \( a = [s, e] \) denoted by \( (s, e) \)
    - \( (a.s, a.e) \), by the start and end position of region \( a \)

Positions in sgrep

- Sample file (under Linux):
  - \% cat abra.txt
  - abracadabra$
- Regions normally occurrences of a string:
  - \% sgrep -o"%(s,\e)" "abra" abra.txt
    - (0,3) (7,10)
  - output format template "%(s,\e)" applied to each result region ("insert start position and end position in parentheses")
Positions in sgrep (2)

- The first position:
  % sgrep 'start' abra.txt
  a
- The last position:
  % sgrep 'end' abra.txt
  5
- The entire document (more on operator '..' later):
  % sgrep 'start .. end' abra.txt
  abracadabra

Relationships of regions a and b

- Language based on relationships between regions:
  1) a precedes b, b follows a:
     - a ends before b starts, \( a.e < b.s \)
  2) a is included in b (b contains a): \( a \subseteq b \) \( (b \supseteq a) \):
     - \( b.s \leq a.s \) and \( a.e \leq b.e \)
     - proper containment: \( a \subset b, \text{ and } b \supset a \)
  3) if neither of the above, a and b overlap

Region algebra

- Region algebra is a set-valued language
  - the value of any expression is a set of regions
  - c.f. relational algebra: set of rows/tuples
- Operations map region sets to region sets
  \( \rightarrow \) a compositional language: operands of expressions can be any other expressions
  \( \rightarrow \) arbitrarily complex queries can be formed

Building new regions

- Nested regions (e.g. document elements)
  - “followed-by” operators
- Non-nested regions
  - “quote” operators
- By removing overlap with other regions
  - “extracting” operator

“Followed-by”

- The most central and characteristic operator in nested region algebra and sgrep
- allows dynamic creation of regions from their bounding regions
- generalises the way how parentheses are paired together, starting from inside and always matching the closest unmatched ones

Followed-by example

- Consider a parenthesised expression:
  % cat expr.txt
  \( (1(2(3)) \{4\}) \)
- Extract parenthesised sub-expressions:
  % sgrep -o"%r//" '"( .. .. *)' expr.txt
  \( (1(2(3)) \{4\}) /// (2(3)) /// \{3\} /// (4) /// \)
- NB: by default, without the \(-o\) output format specification, sgrep outputs each position covered by result regions only once
**Value of $A \ldots B$ formally**

- $A$ and $B$ sets of regions
- $A \ldots B = \{(a.s, b.e) \cup (A\{-a\}) \ldots (B\{-b\})\}$, where $a \in A$, $b \in B$ such that $a.e < b.s$ and the $a.e \ldots b.s$ distance is minimal
  - (and also distance btw $a.s$ and $b.e$ is minimal, if there are otherwise equidistant pairs)
  - empty set, if there are no such $a \in A$ and $b \in B$
  - each $a \in A$ and each $b \in B$ produces at most one result region $(a.s, b.e)$

**Regions of document elements**

- Regions delimited by start and end tags: % sgrep "<TITLE>" .. "</TITLE>" sgrepman.html
- Delimiting regions can be left-out:
  - using also the built-in markup scanner to recognise tags:
    - % sgrep 'stag("TITLE") . etag("TITLE")' sgrepman.html
    - % sgrep 'stag("TITLE") . etag("TITLE")' sgrepman.html
- (possible empty regions are excluded from $A \ldots B$)

**"Quote" operators**

- Documents typically contain also disjoint regions, often delimited by identical start and end markers
  - e.g. string constants in programming languages
    - % cat hello.txt
      - "Hello," said Bob, "nice day!"
    - % sgrep '""quote""quote""' hello.txt
      - "Hello,""nice day!"
  - starting or ending regions, or both, can be excluded (similarly to ._, __) using _quote, quote_ and _quote_

**Extracting overlapping regions**

- $A$ extracting $B$ = regions that result by removing from regions of $A$ any overlap with regions in $B$
- Contents of sections without subsections:
    - % sgrep "<sect>" .. "</sect>" extracting <"<subsect>" .. "</subsect>" extracted
- Content without markup tags:
    - % sgrep 'start .. end extracting' '([^"\•]*\•)' .html
- Note: No operator precedence; Parenthesised sub-expressions are evaluated first, otherwise left-to-right

**Containment conditions**

- Allow selecting regions based on their context or content
- Operators for selecting regions that
  - appear / do not appear in a given context
  - contain / do not contain a region of another set
- Similar operators in other variations of region algebra

**Containment operators formally**

- $A$ in $B$: \{a$\in A | \exists b \in B: a \subseteq b \}$
- $A$ not in $B$: \{a$\in A | \forall b \in B: a \not\subseteq b \}$
- $A$ containing $B$: \{a$\in A | \exists b \in B: a \supseteq b \}$
- $A$ not containing $B$: \{a$\in A | \forall b \in B: a \not\supseteq b \}$
Set operations

- Union, intersection and difference:
  - A or B, A equal B, A not equal B

- Rather seldom needed (except for 'or'); containment conditions otherwise sufficient for expressing Boolean retrieval (See next)

Expressing Boolean queries

- HTML files (%f) containing “cat” AND “dog”:
  `sgrep -o"%f\n" 'start .. end containing "cat" containing "dog"' *.html`

- HTML files containing “cat” or “dog”:
  `sgrep -o"%f\n" 'start .. end containing ["cat" or "dog"]' *.html`

- HTML files containing “cat” but no “dog”:
  `sgrep -o"%f\n" 'start .. end containing "cat" not containing "dog"' *.html`

Structural retrieval

- Arbitrarily complex queries, e.g., “extract titles of pages which mention cats, but not in the title of the page”:
  `sgrep ""<TITLE>" .. "</TITLE>" in [istart .. end containing ["cat" not in ["<TITLE>" .. "]</TITLE>"]]' *.html`

- But notice: The model supports (and sgrep implements) only extracting the regions that satisfy the query, in order – with restricted modification of each region (using -o)

Additional operations on region sets

- concat(A): minimal set of regions covering exactly the regions of A
  - default result formatting of sgrep

- inner(A) = A not containing A
  - the innermost regions in A

- outer(A) = A not in A
  - the outermost regions in A
  - e.g. to get the document root element:
    `outer(elements)`

Sgrep: some useful options

- -a: Display result regions surrounded by the rest of the file; useful with -o (below)

- -c <style>: Set output format string, possibly containing following place holders:
  - %f: name of the file containing the start of region,
  - %s, %e: start and end position,
  - %r: the content (text) of the region,
  - %n: the ordinal number of the region, (+ few others);
  Output once for each result region, with current values substituted for place holders

Sgrep: more useful options

- -c display only count of matching regions
- -h a short help (list of options)
- -f file read commands from file
- -i ignore case distinctions in phrases
- -S stream mode (regions extend across files)
- -T/-t show statistics about things done/time spent
- For more, see the man page and README
Applications of sgrep

- Simple document assembly
  - Jaakkola and Kipeläinen: Using sgrep for querying structured text files, *SGML Finland’96*
- Analysing element structures
- As a Web site search engine
  - E.g. Chapter 7 in Leventhal, Lewis & Fuchs: Designing XML Internet Applications
  - Using sgrep indexer to speed up querying static files

Analysing element structures

- Structure of element instances in a collection? The DTD does not tell!
- Element statistics:
  - 1lu 1751 (min: 67, avg: 6849.37, max: 115620)
  - 41 -> 98 huom
  - 1751 -> 1751 nu
  - 1748 -> 1748 ot
- 1751 lu elements, with shown minimum, average and maximum lengths; 41 contain huom elements directly, and 98 huom elements are children of lu elements.

Implementation of sgrep

- Steps in executing an sgrep query?
  1. Preprocessing
     - Macro expansion; external & optional
  2. Query parsing
  3. Query optimisation
  4. String retrieval on the text
  5. Operator evaluation
  6. Data delivery
     - Outputting of result regions

Structure analysis: implementation

- Generated by a Python script
- Each datum (element type name, count, length) computed by generating and executing an sgrep query

1. Preprocessing

- Consider the below query Q:
  - `sgrep -f macros -e 'outer("\<sec\>\<\ sec\>\")' a.xml b.xml`
- With suitable macro definitions Q becomes `outer("\<sec\>\<\ sec\>\") in S[1] a.xml b.xml`

2. Query parsing

- Query into an operator tree:
  - operators as internal nodes, string phrases as leaves

3. Query optimisation

- common subexpression elimination
- operator tree to an operator graph (DAG)
  - e.g., an operator tree of 735 nodes (for BibTeX records) reduces to a DAG of 103 nodes

4. String retrieval (1)

- A deterministic Aho-Corasick automaton $M$ built of string patterns in the query $M$

4. String retrieval (2)

- Input files (or streams) scanned using $M$
  - each pattern simultaneously
  - region list of pattern occurrences attached to the leaves of the operator graph

5. Operator evaluation

- Bottom-up traversal of the operator graph
  - value of node: a region list
    - of (start, end) index pairs
    - internal representation of region sets
    - maintained in increasing (start, end) order
    - each node evaluated once
    - (sub)queries with large results may require lots of main memory!
6. Data delivery

- Finally, the regions in the result region list are output in document order:
  - text of regions retrieved by indexing target files by start and end positions of the result regions
  - result regions possibly modified according to the output style specification (given with option `-o`)

Evaluating region algebra operators

- Synchronised (merge-like, linear) traversal of operand lists, except that ...:
  - $A .. B$ requires sorting of $A$ (by region end positions) if $A$ contains nested regions
  - $A \text{ extracting } B$ may require considering the same regions of $A$ multiple times

Evaluation complexity

- Worst-case time $O(t \cdot n)$, where $n$ maximum size and $t$ maximum "thickness" (number of regions overlapping at any position) of any region set:
- In practise linear time;
- comparable to Unix greps

Extended features in sgrep 2

- (pre-release v. 1.92a)
- documented (only!) in the README file
- XML/HTML/SGML support:
  - scanner to recognise markup tokens
  - SGML/HTML-mode (default)
    - markup names converted to UPPER CASE
  - XML-mode: case-sensitive retrieval of tag names
  - simple-minded parser to recognise elements
  - 16-bit wide characters in XML documents
- NB: currently no expanding of entity references;
  - No validation or well-formedness checking

Extended sgrep features (2)

- Direct containment:
  - $A \text{ childrening } B$, $A \text{ parenting } B$
- Restricting the number of result regions:
  - $\text{first}(n, E)$, $\text{last}(n, E)$
- Restricting the length of result regions:
  - $\text{first_bytes}(n, E)$, $\text{last_bytes}(n, E)$
- Nearness operators:
  - $A \text{ near }(n)$, $A \text{ near}_{\text{before}}(n)$
- Indexing of both structure and content

Restricting result regions

- Display first 30 bytes of first 3 paragraphs:

  ```
  turn> sgrep -o"tr ...\n" -g xml "
  'first_bytes(30, first(3, stag("p") ..
  etag("p")))' REC-xml-19980210.xml
  <p>The Extensible Markup Langua...
  <p>This document has been revie...
  <p>This document specifies a s...
  ```
Restricting results (2)

- Get end tags of the children of the document element:

```
turni> sgrep -o"%r\n" -g xml \
'etag("**") containing last_bytes(1, 
outer[elements in elements])"' \
RCX-xml-19980210.xml
</header>
</body>
</back>
```

Indexing in sgrep 2.0

- Both the structure and content (words) indexed (max file size 2 GB)
- Creates a separate index (postings file)
  - terms with region lists of their occurrences
  - index a compressed binary file, size 30-60% of the original files
- Increases the efficiency of accessing static document collections considerably
  - see the next example

An indexing example

- Index a 64-fold copy of the XML Rec (S64, 10 MB):
  > sgrep -I -g xml -c S64.index S64
  > ls -l S64.index
  ...
  3621954 Feb  8 21:01 S64.index

- How often word "Bray" occurs in file S64?
  > time sgrep -c 'word("Bray")' S64
  320
  3.80user 0.07system 0:03.87elapsed

- The same using the index:
  > time sgrep -c -x S64.index 'word("Bray")'
  320
  0.02user 0.01system 0:00.03elapsed

- A 100-fold speed-up!