Succinctness

We describe a implementation of the DOM that does not require the use of node-pointers, and is based upon succinct data structures.

Succinct data structures use the information-theoretically minimum number of bits to encode a object. For example, an ordinal tree on 1 node is a rooted tree, where the children of a node are ordered from left-to-right (XML documents are essentially ordinal trees). The lower bound for representing an ordinal tree on 1 node is 2 - O \log n bits [1,9].

This is much more than the pointer-based representations (encoded address), which would use asymptotically 4-6 log n bits.

Motivations

A major disadvantage of most implementations of the DOM is a high memory footprint, referred to as “XML bloat”. The XML DOM representation of an XML document can be many times larger than the XML itself itself, for example figure 1. This means that even moderately large XML documents cannot be processed within the main memory of a reasonably high-end machine.

Preliminary tree representation [2] of XML doc (figure 2)

SDOM Components

We have implemented Succinct DOM (SDOM) in C++. The process of building the SDOM data structure from an XML document is via a SAX parser. Figure 5 shows the architecture of SDOM. We have already discussed the tree structure (STree), we now give an overview of the other components.

Text DS:

Memory used for representing XML documents (~30-60% of tree).

Start DS:

XML document root.

Name Code DS:

Node identification.

Name Pool:

Node name storage.

Attributes DS:

Node attributes.

Tree DS:

Node tree structure.

SAX Parser

Root node.

SDOM

STree

DecTree

Name Code DS

Name Pool

Attributes DS

XQuery and XSLT processing

XML Document

Figure 3. Overview of parsing XML documents to SDOM components.

Figure 7. Test XML files and their sizes. SDOM space usage with no text compression applied. SDOM memory usage compared to Xerces-C.

UCSD Publications

O’Neil Delpratt

(Joint work with Rajeev Raman, Naila Rahman)

Department of Computer Science, University of Leicester

Introduction

XML is a standard format for data exchange and storage. XML documents are processed by a number of applications in the following manner: the XML document is parsed, and a tree representation is built in main memory. The DOM representation is then accessed through the standard DOM interface.

The DOM interface is very flexible, and is commonly used for XML processing. Our focus is on static XML documents --- while DOM does have functionality that allows (fairly arbitrary) changes to the XML document, this functionality is not frequently used. Indeed, there are few DOM implementations for static documents.

We discuss the advantages and disadvantages of existing implementations of the DOM and describe a new approach in our DOM implementation that is based upon succinct data structures.

Acknowledgments

We thank 4DM and UCSD for supporting our work.

Conclusions

Motivated by succinct representations we have discovered new uses for the application of XML document content and its structure. By the engineering of SDOM: SDOM provides flexibility in XML processing by letting tuning parameters in the SDOM components. This greatly reduces space usage where memory is limited or increases space usage where performance is critical.

We have shown that Succinct trees improve the space complexity without compromising too much on query time. Also, to represent XML trees close to optimal space, while supporting, a wide range of operations efficiently.

XQuery and XSLT processing

BAX-Parser

STree

DecTree

Name Code DS

Name Pool

Attributes DS

XQuery and XSLT processing

XML Document

Figure 3. Overview of parsing XML documents to SDOM components.

Figure 7. Test XML files and their sizes. SDOM space usage with no text compression applied. SDOM memory usage compared to Xerces-C.

UCSD Publications

O’Neil Delpratt

(Joint work with Rajeev Raman, Naila Rahman)

Department of Computer Science, University of Leicester

Introduction

XML is a standard format for data exchange and storage. XML documents are processed by a number of applications in the following manner: the XML document is parsed, and a tree representation is built in main memory. The DOM representation is then accessed through the standard DOM interface.

The DOM interface is very flexible, and is commonly used for XML processing. Our focus is on static XML documents --- while DOM does have functionality that allows (fairly arbitrary) changes to the XML document, this functionality is not frequently used. Indeed, there are few DOM implementations for static documents.

We discuss the advantages and disadvantages of existing implementations of the DOM and describe a new approach in our DOM implementation that is based upon succinct data structures.

Motivations

A major disadvantage of most implementations of the DOM is a high memory footprint, referred to as “XML bloat”. The XML DOM representation of an XML document can be many times larger than the XML itself itself, for example figure 1. This means that even moderately large XML documents cannot be processed within the main memory of a reasonably high-end machine.

Preliminary tree representation [2] of XML doc (figure 2)

SDOM Components

We have implemented Succinct DOM (SDOM) in C++. The process of building the SDOM data structure from an XML document is via a SAX parser. Figure 5 shows the architecture of SDOM. We have already discussed the tree structure (STree), we now give an overview of the other components.

Text DS:

Memory used for representing XML documents (~30-60% of tree).

Start DS:

XML document root.

Name Code DS:

Node identification.

Name Pool:

Node name storage.

Attributes DS:

Node attributes.

Tree DS:

Node tree structure.

SAX Parser

Root node.

SDOM

STree

DecTree

Name Code DS

Name Pool

Attributes DS

XQuery and XSLT processing

XML Document

Figure 3. Overview of parsing XML documents to SDOM components.

Figure 7. Test XML files and their sizes. SDOM space usage with no text compression applied. SDOM memory usage compared to Xerces-C.

UCSD Publications

O’Neil Delpratt

(Joint work with Rajeev Raman, Naila Rahman)

Department of Computer Science, University of Leicester

Introduction

XML is a standard format for data exchange and storage. XML documents are processed by a number of applications in the following manner: the XML document is parsed, and a tree representation is built in main memory. The DOM representation is then accessed through the standard DOM interface.

The DOM interface is very flexible, and is commonly used for XML processing. Our focus is on static XML documents --- while DOM does have functionality that allows (fairly arbitrary) changes to the XML document, this functionality is not frequently used. Indeed, there are few DOM implementations for static documents.

We discuss the advantages and disadvantages of existing implementations of the DOM and describe a new approach in our DOM implementation that is based upon succinct data structures.

Motivations

A major disadvantage of most implementations of the DOM is a high memory footprint, referred to as “XML bloat”. The XML DOM representation of an XML document can be many times larger than the XML itself itself, for example figure 1. This means that even moderately large XML documents cannot be processed within the main memory of a reasonably high-end machine.

Preliminary tree representation [2] of XML doc (figure 2)

SDOM Components

We have implemented Succinct DOM (SDOM) in C++. The process of building the SDOM data structure from an XML document is via a SAX parser. Figure 5 shows the architecture of SDOM. We have already discussed the tree structure (STree), we now give an overview of the other components.

Text DS:

Memory used for representing XML documents (~30-60% of tree).

Start DS:

XML document root.

Name Code DS:

Node identification.

Name Pool:

Node name storage.

Attributes DS:

Node attributes.

Tree DS:

Node tree structure.

SAX Parser

Root node.

SDOM

STree

DecTree

Name Code DS

Name Pool

Attributes DS

XQuery and XSLT processing

XML Document

Figure 3. Overview of parsing XML documents to SDOM components.

Figure 7. Test XML files and their sizes. SDOM space usage with no text compression applied. SDOM memory usage compared to Xerces-C.