# XML and Databases
# Tutorial session 3: SAX parsing
# DAG Building
# Binary tree encoding

Kim.Nguyen@nicta.com.au

Week 4

# Event-based programming?

Want to execute a sequence of elementary actions, but the order is not known in advance:

# Event-based programming?

Want to execute a sequence of elementary actions, but the order is not known in advance:

- GUI programming

# Event-based programming?

Want to execute a sequence of elementary actions, but the order is not known in advance:

- GUI programming

$\Rightarrow$ user-based event (click on a widget $\Rightarrow$ execute "save file" function, ...)

# Event-based programming?

Want to execute a sequence of elementary actions, but the order is not known in advance:

- GUI programming

$\Rightarrow$ user-based event (click on a widget $\Rightarrow$ execute "save file" function, ...)

- Hardware programming

# Event-based programming?

Want to execute a sequence of elementary actions, but the order is not known in advance:

- GUI programming

$\Rightarrow$ user-based event (click on a widget $\Rightarrow$ execute "save file" function, ...)

- Hardware programming

$\Rightarrow$ Interruption handling, hardware timers,...

# Event-based programming?

Want to execute a sequence of elementary actions, but the order is not known in advance:

- GUI programming

$\Rightarrow$ user-based event (click on a widget $\Rightarrow$ execute "save file" function, ...)

- Hardware programming

$\Rightarrow$ Interruption handling, hardware timers,...

- XML Parsing!

# Event-based programming?

Want to execute a sequence of elementary actions, but the order is not known in advance:

- GUI programming

$\Rightarrow$ user-based event (click on a widget $\Rightarrow$ execute "save file" function, ...)

- Hardware programming

$\Rightarrow$ Interruption handling, hardware timers,...

- XML Parsing!

$\Rightarrow$ a document can arbitrarily mix comments, text, elements, ...

# Event-based programming?

Want to execute a sequence of elementary actions, but the order is not known in advance:

- GUI programming

$\Rightarrow$ user-based event (click on a widget $\Rightarrow$ execute "save file" function, ...)

- Hardware programming

$\Rightarrow$ Interruption handling, hardware timers,...

- XML Parsing!

$\Rightarrow$ a document can arbitrarily mix comments, text, elements, ...

# Event-based programming?

Want to execute a sequence of elementary actions, but the order is not known in advance:

- GUI programming

$\Rightarrow$ user-based event (click on a widget $\Rightarrow$ execute "save file" function, ...)

- Hardware programming

$\Rightarrow$ Interruption handling, hardware timers,...

- XML Parsing!

$\Rightarrow$ a document can arbitrarily mix comments, text, elements, ...

SAX: Simple API for XML

# Event-based programming?

Want to execute a sequence of elementary actions, but the order is not known in advance:

- GUI programming

$\Rightarrow$ user-based event (click on a widget $\Rightarrow$ execute "save file" function, ...)

- Hardware programming

$\Rightarrow$ Interruption handling, hardware timers,...

- XML Parsing!

$\Rightarrow$ a document can arbitrarily mix comments, text, elements, ...

SAX: Simple API for XML

# SAX Parser

Simple loop:

1. read some character from the input

# SAX Parser

Simple loop:

1. read some character from the input
2. try to recognize an XML token

# SAX Parser

Simple loop:

1. read some character from the input
2. try to recognize an XML token
3. call the corresponding *callback* (or *handler*)

# SAX Parser

Simple loop:

1. read some character from the input
2. try to recognize an XML token
3. call the corresponding *callback* (or *handler*)
4. continue until the end of the input

# SAX Parser

Simple loop:

1. read some character from the input
2. try to recognize an XML token
3. call the corresponding *callback* (or *handler*)
4. continue until the end of the input

# SAX Parser

Simple loop:

1. read some character from the input
2. try to recognize an XML token
3. call the corresponding ***callback*** (or *handler*)
4. continue until the end of the input

- Only the callbacks need to be defined by the programmer

# SAX Parser

Simple loop:

1. read some character from the input
2. try to recognize an XML token
3. call the corresponding *callback* (or *handler*)
4. continue until the end of the input

- Only the callbacks need to be defined by the programmer
- The programmer has to handle the storage/buildling

# Sample SAX program in Java

Read an XML Document, count the number of elements.

```
import org.apache.xerces.parsers.SAXParser;
import org.w3c.dom.*;
import java.util.*;
import java.io.*;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;


...
```

# Sample SAX program in Java

```
class SAXExample {

  class MyHandler extends DefaultHandler {
    int count;

    void startElement(String nsuri, String local,
                      String raw, Attributes att)
    {
      count++;
      System.out.println("<"+local+">");
    }
    void endElement(String nsuri, String local,
                    String raw)
    {
      System.out.println("</"+local+">");
    }
```

# Sample SAX program in Java

```
class SAXExample {

  class MyHandler extends DefaultHandler {
    int count;

    void startElement(String nsuri, String local,
                      String raw, Attributes att)
    {
      count++;
      System.out.println("<"+local+">");
    }
    void endElement(String nsuri, String local,
                    String raw)
    {
      System.out.println("</"+local+">");
    }
```

# Sample SAX program in Java

```
class SAXExample {

  class MyHandler extends DefaultHandler {
    int count;

    void startElement(String nsuri, String local,
                      String raw, Attributes  att)
    {
      count++;
      System.out.println("<"+local+">");
    }
    void endElement(String nsuri, String local,
                    String raw)
    {
      System.out.println("</"+local+">");
    }
```

# Sample SAX program in Java

```
class SAXExample {

  class MyHandler extends DefaultHandler {
    int count;

    void startElement(String nsuri, String local,
                      String raw, Attributes  att)
    {
      count++;
      System.out.println("<"+local+">");
    }
    void endElement(String nsuri, String local,
                    String raw)
    {
      System.out.println("</"+local+">");
    }
```

# Sample SAX program in Java

```
class SAXExample {

  class MyHandler extends DefaultHandler {
    int count;

    void startElement(String nsuri, String local,
                      String raw, Attributes  att)
    {
      count++;
      System.out.println("<"+local+">");
    }
    void endElement(String nsuri, String local,
                    String raw)
    {
      System.out.println("</"+local+">");
    }
```

# Sample SAX program in Java

```
class SAXExample {

  class MyHandler extends DefaultHandler {
    int count;

    void startElement(String nsuri, String local,
                      String raw, Attributes att)
    {
      count++;
      System.out.println("<"+local+">");
    }
    void endElement(String nsuri, String local,
                    String raw)
    {
      System.out.println("</"+local+">");
    }
```

# Sample SAX program in Java

```
class SAXExample {

  class MyHandler extends DefaultHandler {
    int count;

    void startElement(String nsuri, String local,
                      String raw, Attributes  att)
    {
      count++;
      System.out.println("<"+local+">");
    }
    void endElement(String nsuri, String local,
                    String raw)
    {
      System.out.println("</"+local+">");
    }
```

# Sample SAX program in Java

```
  void characters(char[] buffer, int start,int len)
  {
    System.out.println(new String(buffer, start,len);
  }
  void startDocument()
  {
    count = 0;
  }
  void endDocument()
  {
   System.out.println(count + " elements in the document");
  }
}//MyHandler
```

# Sample SAX program in Java

```
  void characters(char[] buffer, int start,int len)
  {
    System.out.println(new String(buffer, start,len);
  }
  void startDocument()
  {
    count = 0;
  }
  void endDocument()
  {
   System.out.println(count + " elements in the document");
  }
}//MyHandler
```

# Sample SAX program in Java

```
  void characters(char[] buffer, int start,int len)
  {
    System.out.println(new String(buffer, start,len));
  }
  void startDocument()
  {
    count = 0;
  }
  void endDocument()
  {
   System.out.println(count + " elements in the document");
  }
}//MyHandler
```

# Sample SAX program in Java

```
public static void main(char [] args){
  SAXParser parser;
  try {
    parser = new SAXParser();
    MyHandler handle= new MyHandler();
    parser.setContentHandler(handle);
    parser.setErrorHandler(handle);
    parser.parse(args[0]);
  }
  catch (Exception e) {
    System.out.println ("Error during parsing"
                        + e.toString());
  };

} //SAXExample
```

# Sample SAX program in Java

```
public static void main(char [] args){
  SAXParser parser;
  try {
    parser = new SAXParser();
    MyHandler handle= new MyHandler();
    parser.setContentHandler(handle);
    parser.setErrorHandler(handle);
    parser.parse(args[0]);
  }
  catch (Exception e) {
    System.out.println ("Error during parsing"
                        + e.toString());
  };

} //SAXExample
```

# Sample SAX program in Java

```
public static void main(char [] args){
  SAXParser parser;
  try {
    parser = new SAXParser();
    MyHandler handle= new MyHandler();
    parser.setContentHandler(handle);
    parser.setErrorHandler(handle);
    parser.parse(args[0]);
  }
  catch (Exception e) {
    System.out.println ("Error during parsing"
                        + e.toString());
  };

} //SAXExample
```

# Sample SAX program in Java

```
public static void main(char [] args){
  SAXParser parser;
  try {
    parser = new SAXParser();
    MyHandler handle= new MyHandler();
    parser.setContentHandler(handle);
    parser.setErrorHandler(handle);
    parser.parse(args[0]);
  }
  catch (Exception e) {
    System.out.println ("Error during parsing"
                        + e.toString());
  };

} //SAXExample
```

# Sample SAX program in Java

```
public static void main(char [] args){
  SAXParser parser;
  try {
    parser = new SAXParser();
    MyHandler handle= new MyHandler();
    parser.setContentHandler(handle);
    parser.setErrorHandler(handle);
    parser.parse(args[0]);
  }
  catch (Exception e) {
    System.out.println ("Error during parsing"
                        + e.toString());
  };

} //SAXExample
```

# SAX Summary

- extend the class `DefaultHandler`

# SAX Summary

- extend the class `DefaultHandler`
- create a `SAXParser`

# SAX Summary

- extend the class `DefaultHandler`
- create a `SAXParser`
- bind the your handler to the parser (`parser.setContentHandler()`)

# SAX Summary

- extend the class `DefaultHandler`
- create a `SAXParser`
- bind the your handler to the parser (`parser.setContentHandler()`)
- parse the file (`parser.parse()`)

# SAX Summary

- extend the class `DefaultHandler`
- create a `SAXParser`
- bind the your handler to the parser (`parser.setContentHandler()`)
- parse the file (`parser.parse()`)

# SAX Summary

- extend the class `DefaultHandler`
- create a `SAXParser`
- bind the your handler to the parser (`parser.setContentHandler()`)
- parse the file (`parser.parse()`)

Q: How much memory do you need to parse a file?(without validation)

# SAX Summary

- extend the class `DefaultHandler`
- create a `SAXParser`
- bind the your handler to the parser (`parser.setContentHandler()`)
- parse the file (`parser.parse()`)

Q: How much memory do you need to parse a file?(without validation)
Q: How much memory do you need to parse a file?(with validation)

# SAX Summary

- extend the class `DefaultHandler`
- create a `SAXParser`
- bind the your handler to the parser (`parser.setContentHandler()`)
- parse the file (`parser.parse()`)

Q: How much memory do you need to parse a file?(without validation)
Q: How much memory do you need to parse a file?(with validation)

# Building the DAG, Hashtables

Q: What is a DAG ?
A: A list of `DAGNodes`

# Building the DAG, Hashtables

Q: What is a DAG ?
A: A list of `DAGNodes`
Q: How to represent a `DAGNode`

# Building the DAG, Hashtables

Q: What is a DAG ?
A: A list of `DAGNodes`
Q: How to represent a `DAGNode`
A: A unique ID, a tag, and the sequence of unique IDs of it's children.

# Building the DAG, Hashtables

Q: What is a DAG ?
A: A list of `DAGNodes`
Q: How to represent a `DAGNode`
A: A unique ID, a tag, and the sequence of unique IDs of it's children.

Algorithm:
During SAX Parsing, the first time a node is seen, create the corresponding DAGNode, put it in a Hashtable and associate a unique ID.
Then, before inserting an element into a DAG, check that it is not already in the Hashtable

# Building the DAG, Hashtables

Q: How do hashtabl in Java work ?

# Building the DAG, Hashtables

Q: How do hashtabl in Java work ?
A: `.hashCode()` and `.equals()` methods defined for any object.

# Building the DAG, Hashtables

Q: How do hashtabl in Java work ?
A: `.hashCode()` and `.equals()` methods defined for any object.

Any object can be used as a Key for a Hashtable.
When calling `table.put(k,v)` the hashtable computes `k.hashCode()`.
The hashtable checks whether two keys `k1` and `k2` with the same hash are equal using `k1.equals(k2)`.

# Building the DAG, Hashtables

Q: How do hashtabl in Java work ?
A: `.hashCode()` and `.equals()` methods defined for any object.

Any object can be used as a Key for a Hashtable.
When calling `table.put(k,v)` the hashtable computes `k.hashCode()`.
The hashtable checks whether two keys `k1` and `k2` with the same hash are equal using `k1.equals(k2)`.
Let's define our keys:

# Building the DAG, Hashtables

Q: How do hashtabl in Java work ?
A: `.hashCode()` and `.equals()` methods defined for any object.

Any object can be used as a Key for a Hashtable.
When calling `table.put(k,v)` the hashtable computes `k.hashCode()`.
The hashtable checks whether two keys `k1` and `k2` with the same hash are equal using `k1.equals(k2)`.
Let's define our keys:

```
class DAGNode {
  String tag;
  LinkedList<Integer> children
```

```
class DAGNode {
  String tag;
  LinkedList<Integer> children
  DAGNode(String t) {
    tag = t;
    children = new LinkedList<Integer>();
  }
```

```
class DAGNode {
  String tag;
  LinkedList<Integer> children
  DAGNode(String t) {
    tag = t;
    children = new LinkedList<Integer>();
  }
  void appendChild(Integer c){ children.addLast(c); }
  LinkedList<Integer> getChildren() {
    return children;
  }
  String getTag() { return tag; }
}
```

Let's use it!

```
...
//Maps a DAGNode to some ID
HashMap<DAGNode, Integer> table =
        new HashMap<DAGNode, Integer>();
```

Let's use it!

```
...
//Maps a DAGNode to some ID
HashMap<DAGNode,Integer> table =
        new HashMap<DAGNode,Integer>();
// Two "identical" nodes with no children
DAGNode n1 = new DAGNode("c");
DAGNode n2 = new DAGNode("c");
table.put(n1,new Integer(1));
```

Let's use it!

```
...
//Maps a DAGNode to some ID
HashMap<DAGNode,Integer> table =
        new HashMap<DAGNode,Integer>();
// Two "identical" nodes with no children
DAGNode n1 = new DAGNode("c");
DAGNode n2 = new DAGNode("c");
table.put(n1,new Integer(1));
System.out.println(table.get(n1)); //prints 1
```

Let's use it!

```
...
//Maps a DAGNode to some ID
HashMap<DAGNode, Integer> table =
        new HashMap<DAGNode, Integer>();
// Two "identical" nodes with no children
DAGNode n1 = new DAGNode("c");
DAGNode n2 = new DAGNode("c");
table.put(n1,new Integer(1));
System.out.println(table.get(n1)); //prints 1
Integer v = table.get(n2);
if (n == null)
  System.out.println("Object not found");
else
  System.out.println(v);
```

Let's use it!

```
...
//Maps a DAGNode to some ID
HashMap<DAGNode, Integer> table =
        new HashMap<DAGNode, Integer>();
// Two "identical" nodes with no children
DAGNode n1 = new DAGNode("c");
DAGNode n2 = new DAGNode("c");
table.put(n1,new Integer(1));
System.out.println(table.get(n1)); //prints 1
Integer v = table.get(n2);
if (n == null)
  System.out.println("Object not found");
else
  System.out.println(v);
// Prints "Object not found"
```

It does not seem to work (yet)

```
...
//Let's test
System.out.println(n1.hashCode());
System.out.println(n2.hashCode());
System.out.println(n1.equals(n2));
```

Prints:

It does not seem to work (yet)

```
...
//Let's test
System.out.println(n1.hashCode());
System.out.println(n2.hashCode());
System.out.println(n1.equals(n2));
```

Prints:

```
1569228633
778966024
false
```

http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Object.html#hashCode()

It does not seem to work (yet)

```
...
//Let's test
System.out.println(n1.hashCode());
System.out.println(n2.hashCode());
System.out.println(n1.equals(n2));
```

Prints:

```
1569228633
778966024
false
```

`http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Object.html#hashCode()`

By default, Java uses the *address* of the Object for a hash and == to implement `.equals()`!

```
class DAGNode {
  String tag;
  LinkedList<Integer> children
  int hash;
  DAGNode(String t) {
    tag = t;
    children = new LinkedList<Integer>();
```

```
class DAGNode {
  String tag;
  LinkedList<Integer> children
  int hash;
  DAGNode(String t) {
    tag = t;
    children = new LinkedList<Integer>();
    hash = t.hashCode(); //redefined for the String class
  }
```

```
class DAGNode {
  String tag;
  LinkedList<Integer> children
  int hash;
  DAGNode(String t) {
    tag = t;
    children = new LinkedList<Integer>();
    hash = t.hashCode(); //redefined for the String class
  }
  void appendChild(Integer c){
    children.addLast(c);
    hash = hash *19 + 23 *c;
  }
  LinkedList<Integer> getChildren() {
    return children;
  }
  String getTag() { return tag; }
```

```
class DAGNode {
  String tag;
  LinkedList<Integer> children
  int hash;
  DAGNode(String t) {
    tag = t;
    children = new LinkedList<Integer>();
    hash = t.hashCode(); //redefined for the String class
  }
  void appendChild(Integer c){
    children.addLast(c);
    hash = hash *19 + 23 *c;
  }
  LinkedList<Integer> getChildren() {
    return children;
  }
  String getTag() { return tag; }
  public int hashCode() { return hash; }
  ...
```

```
public boolean equals(Object b){

    if (b instanceof DAGNode){
```

```
public boolean equals(Object b){

    if (b instanceof DAGNode){
      DAGNode n2 = (DAGNode) b;
      if (this == n2) return true;
```

```
public boolean equals(Object b){

    if (b instanceof DAGNode){
      DAGNode n2 = (DAGNode) b;
      if (this == n2) return true;

      if (!tag.equals(n2.tag)) return false;
```

```
public boolean equals(Object b){

    if (b instanceof DAGNode){
      DAGNode n2 = (DAGNode) b;
      if (this == n2) return true;

      if (!tag.equals(n2.tag)) return false;
      Iterator<Integer> i1 = children.iterator();
      Iterator<Integer> i2 = n2.children.iterator();
```

```
public boolean equals(Object b){

   if (b instanceof DAGNode){
     DAGNode n2 = (DAGNode) b;
     if (this == n2) return true;

     if (!tag.equals(n2.tag)) return false;
     Iterator<Integer> i1 = children.iterator();
     Iterator<Integer> i2 = n2.children.iterator();
     while (i1.hasNext()){
       if (!i2.hasNext()) return false;
       if (!i1.next().equals(i2.next())) return false;
     };
     if (i2.hasNext()) return false;
     return true;
   }
```

```
public boolean equals(Object b){

   if (b instanceof DAGNode){
     DAGNode n2 = (DAGNode) b;
     if (this == n2) return true;

     if (!tag.equals(n2.tag)) return false;
     Iterator<Integer> i1 = children.iterator();
     Iterator<Integer> i2 = n2.children.iterator();
     while (i1.hasNext()){
       if (!i2.hasNext()) return false;
       if (!i1.next().equals(i2.next())) return false;
     };
     if (i2.hasNext()) return false;
     return true;
   }
   else return false;
 }
```

General remarks for the assignment:

- Create a `DAGNode` class, that can be a correct `Key`

General remarks for the assignment:

- Create a `DAGNode` class, that can be a correct `Key`
- Initialize a global counter;

General remarks for the assignment:

- Create a `DAGNode` class, that can be a correct `Key`
- Initialize a global counter;
- During parsing, add a `DAGNode` to the hash table with the value of the global counter, if not present and increment the counter;

General remarks for the assignment:

- Create a `DAGNode` class, that can be a correct `Key`
- Initialize a global counter;
- During parsing, add a `DAGNode` to the hash table with the value of the global counter, if not present and increment the counter;
- Once parsing is finished, you have a hash table from `DAGNode` to `Integer`;

General remarks for the assignment:

- Create a `DAGNode` class, that can be a correct `Key`
- Initialize a global counter;
- During parsing, add a `DAGNode` to the hash table with the value of the global counter, if not present and increment the counter;
- Once parsing is finished, you have a hash table from `DAGNode` to `Integer`;
- iterate this table and create the reversed one, from `Integer` to `DAGNode` (this is your dag!);

General remarks for the assignment:

- Create a `DAGNode` class, that can be a correct `Key`
- Initialize a global counter;
- During parsing, add a `DAGNode` to the hash table with the value of the global counter, if not present and increment the counter;
- Once parsing is finished, you have a hash table from `DAGNode` to `Integer`;
- iterate this table and create the reversed one, from `Integer` to `DAGNode` (this is your dag!);
- to print the entries of the dag in increasing order of IDs:

  ```
  TreeSet<Integer> keys = new TreeSet<Integer>(dag.getKeys());
  ```

  just iterate this tree set, you visit the IDs in order

# Binary trees: FirstChild/NextSibling encoding

Bijection between an unranked-tree (XML Document) and a binary tree.
Given a node:

- its first child points to its first child in the original document

```
<a>
   <b> <c/> </b>
   <d> <e/> </d>
   <f/>
</a>
```

# Binary trees: FirstChild/NextSibling encoding

Bijection between an unranked-tree (XML Document) and a binary tree.
Given a node:

- its first child points to its first child in the original document
- its second child points to its next sibling in the original document

```
<a>
   <b> <c/> </b>
   <d> <e/> </d>
   <f/>
</a>
```

# Binary trees: FirstChild/NextSibling encoding

Bijection between an unranked-tree (XML Document) and a binary tree.
Given a node:

- its first child points to its first child in the original document
- its second child points to its next sibling in the original document

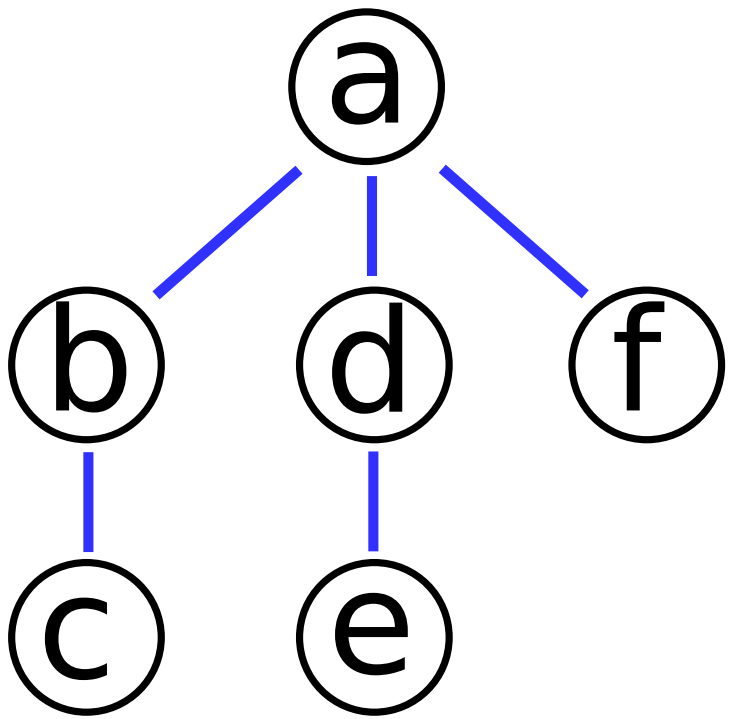
```
<a>
   <b> <c/> </b>
   <d> <e/> </d>
   <f/>
</a>
```

# Binary trees: FirstChild/NextSibling encoding

Bijection between an unranked-tree (XML Document) and a binary tree.
Given a node:

- its first child points to its first child in the original document
- its second child points to its next sibling in the original document

```
<a>
   <b> <c/> </b>
   <d> <e/> </d>
   <f/>
</a>
```

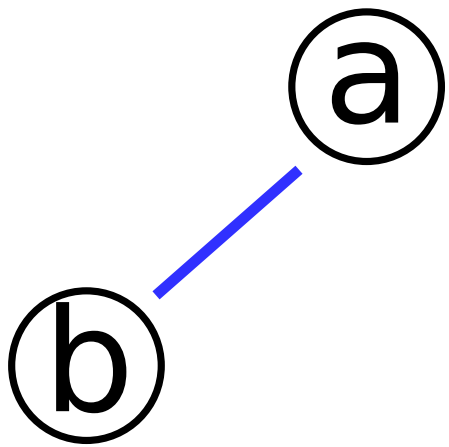


# Binary trees: FirstChild/NextSibling encoding

Bijection between an unranked-tree (XML Document) and a binary tree.
Given a node:

- its first child points to its first child in the original document
- its second child points to its next sibling in the original document

```
<a>
   <b> <c/> </b>
   <d> <e/> </d>
   <f/>
</a>
```

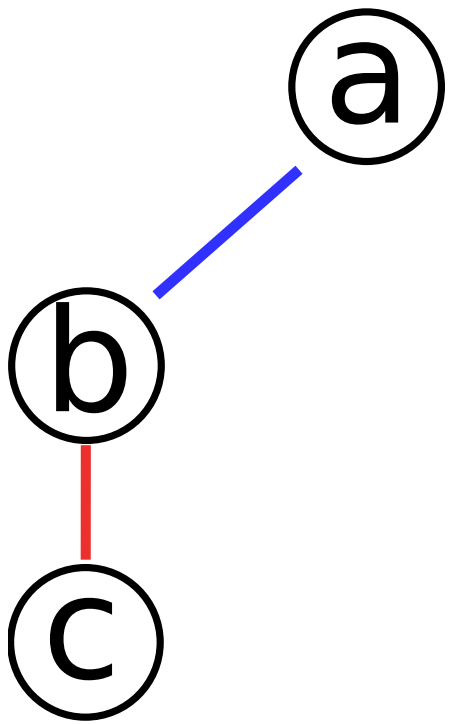


# Binary trees: FirstChild/NextSibling encoding

Bijection between an unranked-tree (XML Document) and a binary tree.
Given a node:

- its first child points to its first child in the original document
- its second child points to its next sibling in the original document

```
<a>
   <b> <c/> </b>
   <d> <e/> </d>
   <f/>
</a>
```

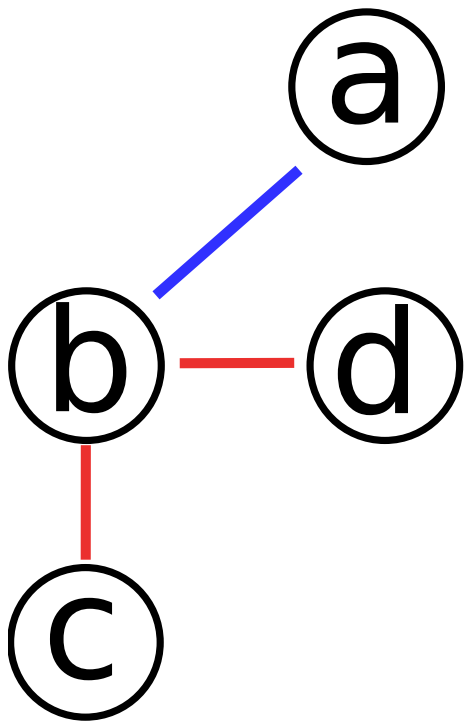


# Binary trees: FirstChild/NextSibling encoding

Bijection between an unranked-tree (XML Document) and a binary tree.
Given a node:

- its first child points to its first child in the original document
- its second child points to its next sibling in the original document

```
<a>
   <b> <c/> </b>
   <d> <e/> </d>
   <f/>
</a>
```

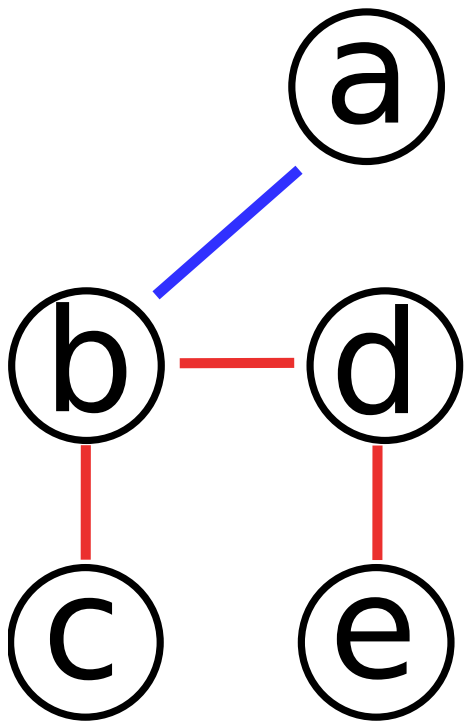


# Binary trees: FirstChild/NextSibling encoding

Bijection between an unranked-tree (XML Document) and a binary tree.
Given a node:

- its first child points to its first child in the original document
- its second child points to its next sibling in the original document

```
<a>
   <b> <c/> </b>
   <d> <e/> </d>
   <f/>
</a>
```

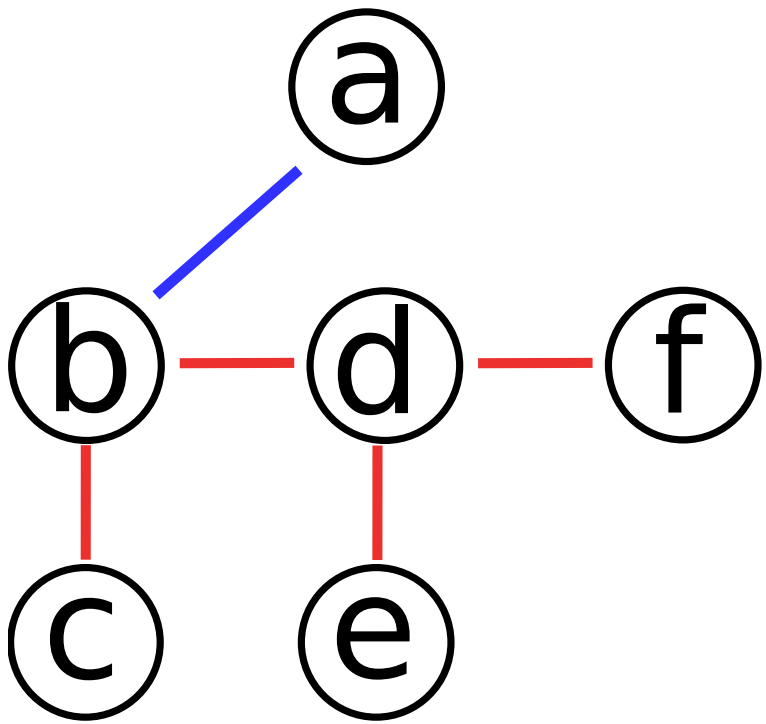


# Binary trees: FirstChild/NextSibling encoding

Bijection between an unranked-tree (XML Document) and a binary tree.
Given a node:

- its first child points to its first child in the original document
- its second child points to its next sibling in the original document

```
<a>
   <b> <c/> </b>
   <d> <e/> </d>
   <f/>
</a>
```

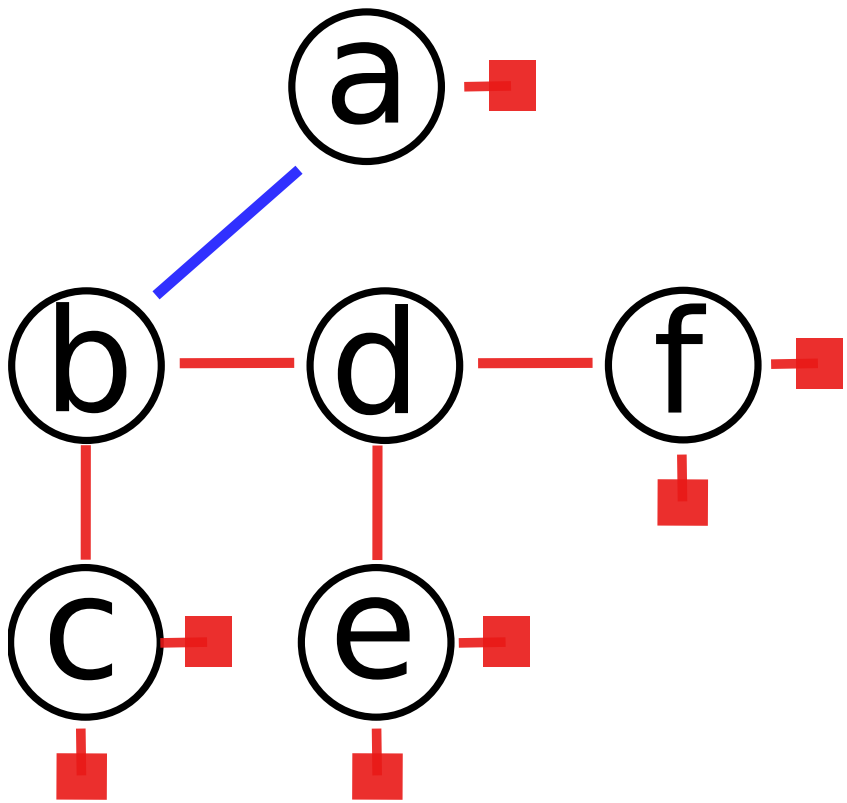


# Binary trees: FirstChild/NextSibling encoding

Bijection between an unranked-tree (XML Document) and a binary tree.
Given a node:

- its first child points to its first child in the original document
- its second child points to its next sibling in the original document

```
<a>
   <b> <c/> </b>
   <d> <e/> </d>
   <f/>
</a>
```



# Binary trees: FirstChild/NextSibling encoding

- Preserves the document order

```
<a>
  <b> <c/> </b>
  <d> <e/> </d>
  <f/>
</a>
```

# Binary trees: FirstChild/NextSibling encoding

- Preserves the document order
- Everything else may be different (height, width, . . . )

```
<a>
  <b> <c/> </b>
  <d> <e/> </d>
  <f/>
</a>
```

# Binary trees: FirstChild/NextSibling encoding

- Preserves the document order
- Everything else may be different (height, width, . . . )

```
<a>
  <b> <c/> </b>
  <d> <e/> </d>
  <f/>
</a>
```

# Binary trees: FirstChild/NextSibling encoding

- Preserves the document order
- Everything else may be different (height, width, . . . )

```
<a>
  <b> <c/> </b>
  <d> <e/> </d>
  <f/>
</a>
```

Q: What is the sequence of SAX events for this document?

# Binary trees: FirstChild/NextSibling encoding

- Preserves the document order
- Everything else may be different (height, width, . . . )

```
<a>
  <b> <c/> </b>
  <d> <e/> </d>
  <f/>
</a>
```

Q: What is the sequence of SAX events for this document?
Q: Write an XML document which represents the binary tree (using <_/>) for the empty tree?

# Binary trees: FirstChild/NextSibling encoding

- Preserves the document order
- Everything else may be different (height, width, . . . )

```
<a>
  <b> <c/> </b>
  <d> <e/> </d>
  <f/>
</a>
```

Q: What is the sequence of SAX events for this document?
Q: Write an XML document which represents the binary tree (using <_/>) for the empty tree?
Q: What is the sequence of SAX events for the binary tree?

# Binary trees: FirstChild/NextSibling encoding

- Preserves the document order
- Everything else may be different (height, width, . . . )

```
<a>
  <b> <c/> </b>
  <d> <e/> </d>
  <f/>
</a>
```

Q: What is the sequence of SAX events for this document?
Q: Write an XML document which represents the binary tree (using <_/>) for the empty tree?
Q: What is the sequence of SAX events for the binary tree?
Q: Find an algorithm to convert a document into a binary one during SAX parsing.

# Binary trees: FirstChild/NextSibling encoding

- Preserves the document order
- Everything else may be different (height, width, . . . )

```
<a>
  <b> <c/> </b>
  <d> <e/> </d>
  <f/>
</a>
```

Q: What is the sequence of SAX events for this document?
Q: Write an XML document which represents the binary tree (using <_/>) for the empty tree?
Q: What is the sequence of SAX events for the binary tree?
Q: Find an algorithm to convert a document into a binary one during SAX parsing.

# Binary Tree Wrapper around an Unranked Event Handler

We want to use the previouly defined `MyHandler` to print the binary tree

```
class MyBinaryHandler extends Default Handler{
  MyHandler handler;
```

# Binary Tree Wrapper around an Unranked Event Handler

We want to use the previouly defined `MyHandler` to print the binary tree

```
class MyBinaryHandler extends Default Handler{
  MyHandler handler;
  Stack<Pair<String,Integer>> stack;
```

# Binary Tree Wrapper around an Unranked Event Handler

We want to use the previouly defined `MyHandler` to print the binary tree

```
class MyBinaryHandler extends Default Handler{
  MyHandler handler;
  Stack<Pair<String,Integer>> stack;
  Integer LEFT = new Integer(0);
  Integer RIGHT = new Integer(1);
```

# Binary Tree Wrapper around an Unranked Event Handler

We want to use the previouly defined `MyHandler` to print the binary tree

```
class MyBinaryHandler extends Default Handler{
  MyHandler handler;
  Stack<Pair<String,Integer>> stack;
  Integer LEFT = new Integer(0);
  Integer RIGHT = new Integer(1);

  void startDocument(){
    handler = new MyHandler();
    stack = new Stack<Pair<String,Integer>>();
    stack.push(new Pair<String,Integer>("",LEFT));
  }
```

# Binary Tree Wrapper around an Unranked Event Handler

```
  void startElement(String nsuri, String label,
                    String raw, Attributes atts)
{
   stack.push(new Pair<String,Integer>(label,LEFT));
   handler.startElement(nsuri,label,raw,atts);
  }

  void endElement(String nsuri, String label,
                  String raw)
{
   Pair<String,Integer> top = stack.peek();
   if (top.getSecond().equals(LEFT)){
     top.setSecond(RIGHT);
     handler.startElement(null,"_","_",null);
     handler.endElement(null,"_","_",null);
}
```

# Binary Tree Wrapper around an Unranked Event Handler

```
else { // Direction is RIGHT
      handler.startElement(null,"_","_",null);
      handler.endElement(null,"_","_",null);
```

# Binary Tree Wrapper around an Unranked Event Handler

```
else { // Direction is RIGHT
      handler.startElement(null,"_","_",null);
      handler.endElement(null,"_","_",null);

      while(top.getSecond().equals(RIGHT)){
        handler.endElement(null,top.getFirst(),top.getFi
        stack.pop();
        top = stack.peek();
      };
```

# Binary Tree Wrapper around an Unranked Event Handler

```
else { // Direction is RIGHT
      handler.startElement(null,"_","_",null);
      handler.endElement(null,"_","_",null);

      while(top.getSecond().equals(RIGHT)){
        handler.endElement(null,top.getFirst(),top.getFi
        stack.pop();
        top = stack.peek();
      };

      top.setSecond(RIGHT);
```

# Binary Tree Wrapper around an Unranked Event Handler

```
else { // Direction is RIGHT
      handler.startElement(null,"_","_",null);
      handler.endElement(null,"_","_",null);

      while(top.getSecond().equals(RIGHT)){
        handler.endElement(null,top.getFirst(),top.getFi
        stack.pop();
        top = stack.peek();
      };

      top.setSecond(RIGHT);
    }//else
 }// endElement()
```

# Binary Tree Wrapper around an Unranked Event Handler

```
void endDocument(){
    Pair<String,Integer> top = stack.peek();
    if (top.getSecond().equals(RIGHT)){
      handler.startElement(null,"_","_",null);
      handler.endElement(null,"_","_",null);
      handler.endElement(null,top.getFirst(),top.getFirs
    };
 }

} //end class MyBinHandler
```