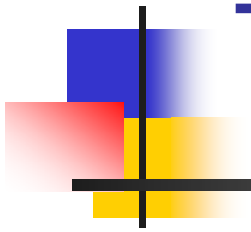
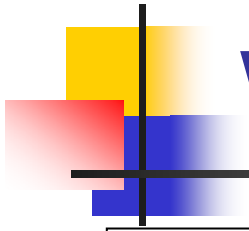


COMP4317: XML & Database
Tutorial 5: Database => XML

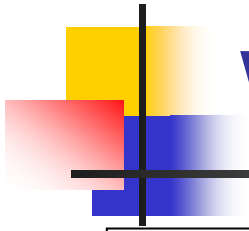
Week 6

Thang Bui @ CSE.UNSW



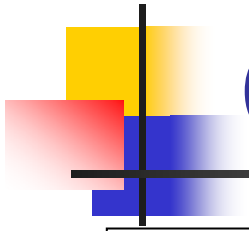
What are your tablenames?

```
mysql> SHOW TABLES;
+--------------------+
| Tables_in_student  |
+--------------------+
| test               |
| z1234567_book_attr |
| z1234567_book_tbl  |
| z3168150_book_attr |
| z3168150_book_tbl  |
+--------------------+
5 rows in set (0.00 sec)

mysql>
```



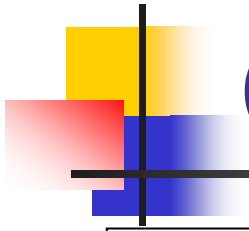
What are your tablenames?

```
mysql> SHOW TABLES LIKE 'z1234567%';
+--------------------------------+
| Tables_in_student (z1234567%) |
+--------------------------------+
| z1234567_book_attr             |
| z1234567_book_tbl              |
+--------------------------------+
2 rows in set (0.00 sec)

mysql>
```



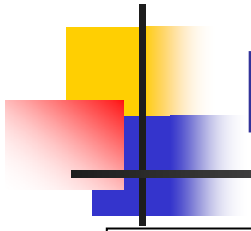
Create your table

```
mysql> CREATE TABLE test(id INTEGER);
ERROR 1050 (42S01): Table 'test' already exists

mysql>
```



Create your table

```
mysql> DROP TABLE IF EXISTS test;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE test(id INTEGER);
Query OK, 0 rows affected (0.08 sec)

mysql>
```

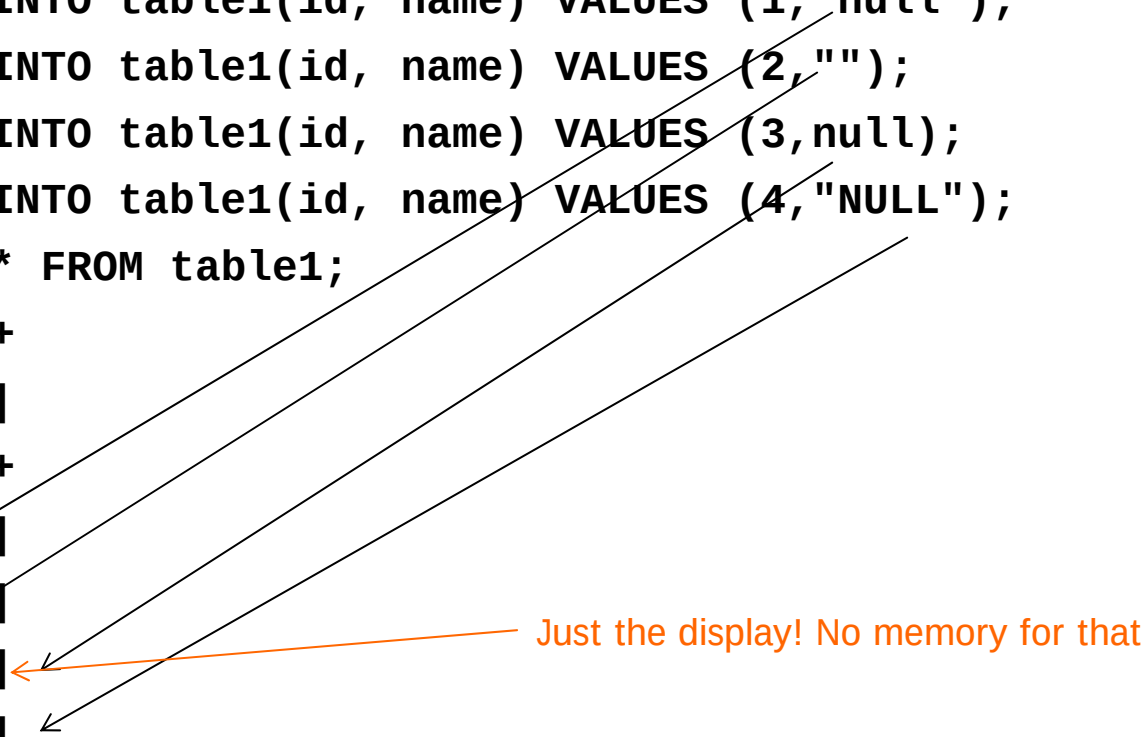


NULL vs Empty in Database

```
mysql> INSERT INTO table1(id, name) VALUES (1,"null");
mysql> INSERT INTO table1(id, name) VALUES (2,"");
mysql> INSERT INTO table1(id, name) VALUES (3,null);
mysql> INSERT INTO table1(id, name) VALUES (4,"NULL");
mysql> SELECT * FROM table1;
+------+------+
| id   | name |
+------+------+
|    1 | null |
|    2 |      |
|    3 | NULL |
|    4 | NULL |
+------+------+
```

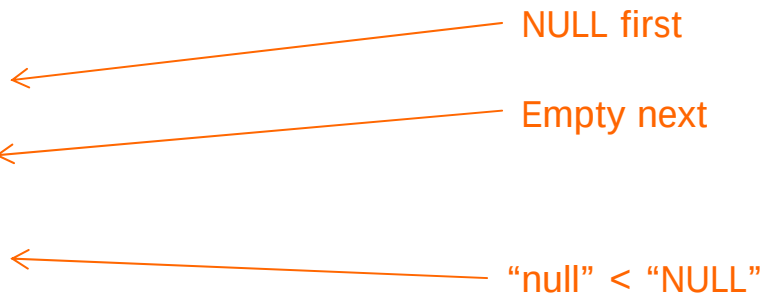
NULL vs Empty in Database

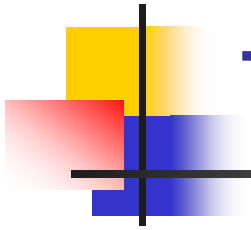
```
mysql> INSERT INTO table1(id, name) VALUES (1,"null");
mysql> INSERT INTO table1(id, name) VALUES (2,"");
mysql> INSERT INTO table1(id, name) VALUES (3,null);
mysql> INSERT INTO table1(id, name) VALUES (4,"NULL");
mysql> SELECT * FROM table1;
+------+------+
| id   | name |
+------+------+
|    1 | null |
|    2 |      |
|    3 | NULL |
|    4 | NULL |
+------+------+
```



NULL vs Empty in Database

```
mysql> SELECT * FROM table1 ORDER BY name;
+------+------+
| id   | name |
+------+------+
|    3 | NULL |
|    2 |      |
|    1 | null |
|    3 | NULL |
+------+------+
```



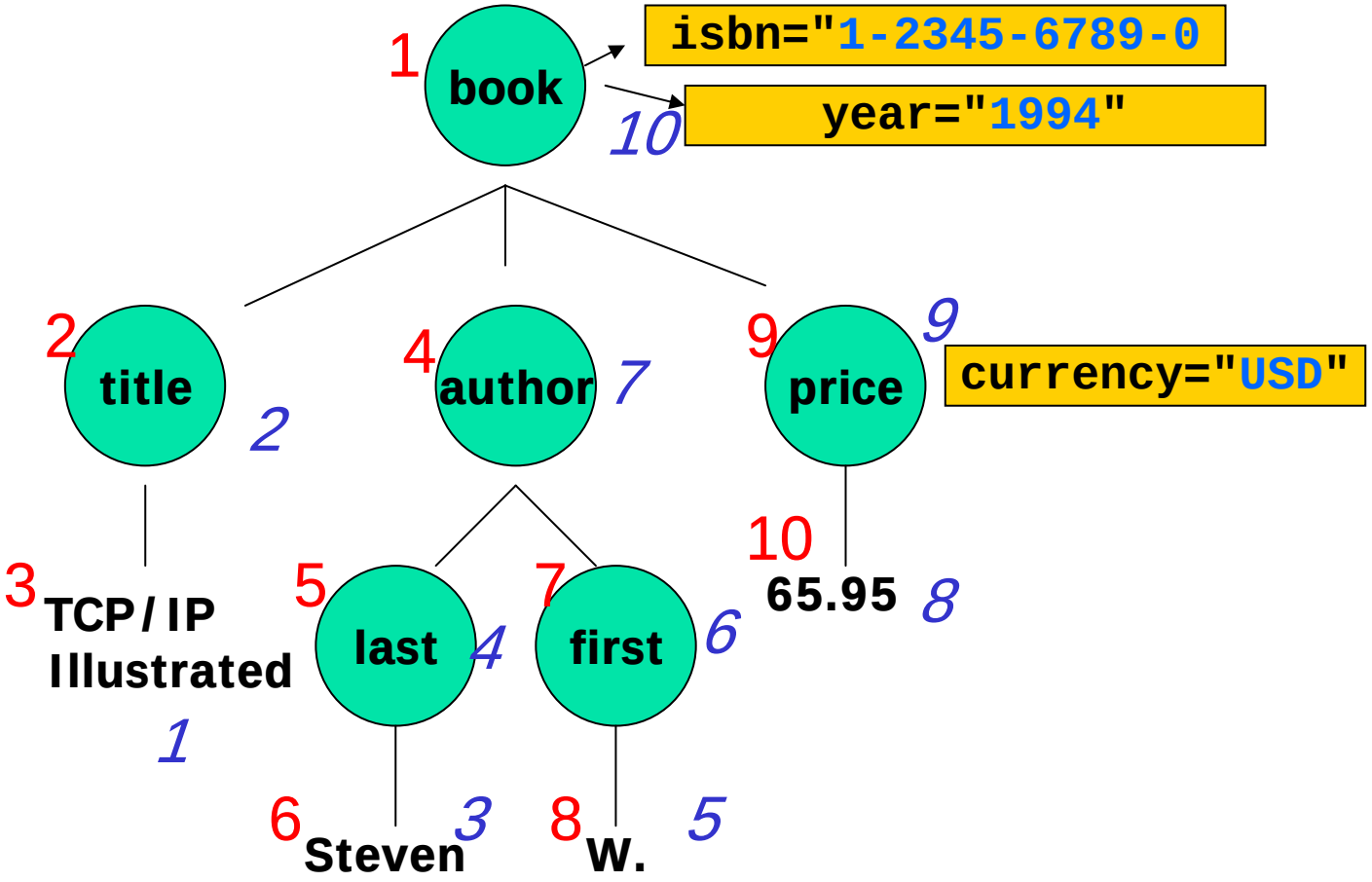


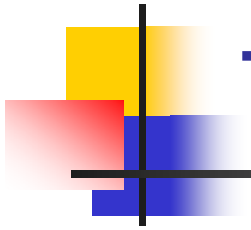
Tables & Tree

pre	post	level	tag	text
1	10	1	book	null
2	2	2	title	null
3	1	3	null	ICP/IP Illustrated
4	7	2	author	null
5	4	3	last	null
6	3	4	null	Steven
7	6	3	first	null
8	5	4	null	W.
9	9	2	price	null
10	8	3	null	65.95

pre	attr_name	attr_value
1	isbn	1-2345-6789-0
1	year	1994
9	currency	USD

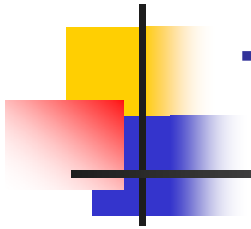
Tables & Tree


1
book
isbn="1-2345-6789-0
10
year="1994"
2
title
2
4
author
7
9
price
9
currency="USD"
3
TCP/IP
Illustrated
1
5
last
4
7
first
6
10
65.95
8
6
Steven
3
8
W.
5



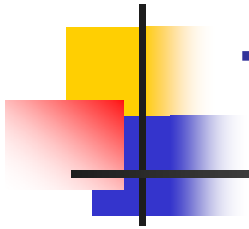
Tables -> Tree

- Features
  - child.pre > parent.pre
  - child.post < parent.post
  - child.level > parent.level



Tables -> Tree

- A simple algorithm
  - Given LP = an order (on pre) list of <pre,post> pairs
  - For every lp =<pre, post> in LP
    - Make a node t (pre, post)
    - Find the “nearest” node p such that p.pre < t.pre and p.post > t.post
    - If such a node, p.addChild(t)



Tables -> Tree

- A simple algorithm
  - Given LP = an order (on pre) list of <pre,post> pairs
  - For every lp =<pre, post> in LP
    - Make a node t (pre, post)
    - Find the “nearest” node p such that p.pre < t.pre and p.post > t.post
    - If such a node, p.addChild(t)

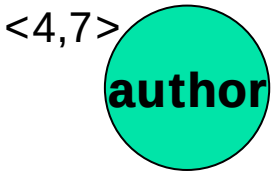
“level” may help?

Tables -> Tree

LP = {<4,7,author>, <5,4,last>, <6,3,Steven>, <7,6,first>, <8,5,W.>}

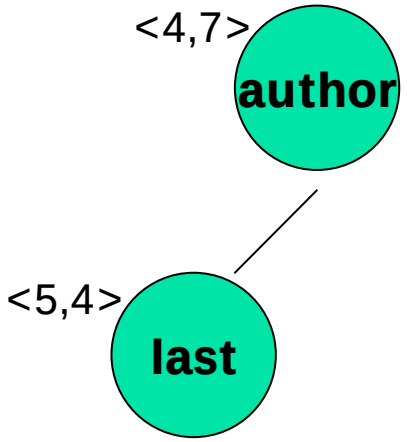
1. lp = <4,7,author>

=> make a first node



Tables -> Tree

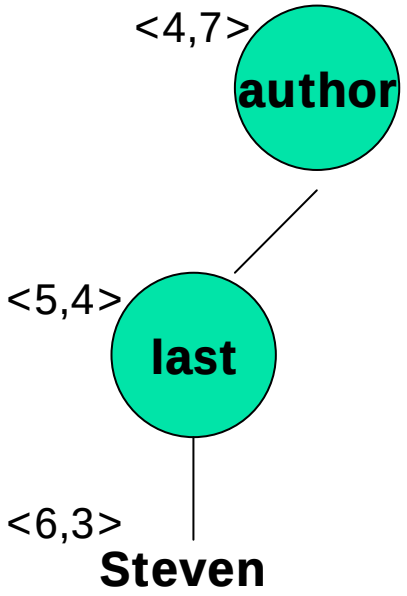
```
LP = {<4,7,author>, <5,4,last>, <6,3,Steven>, <7,6,first>,
<8,5,W.>}

1. lp = <4,7,author>
2. lp = <5,4,last>
        => make a node t
        => find the "nearest" node p:
        p.pre < 5 and p.post > 4
        => p.addChild(t)
```



Tables -> Tree

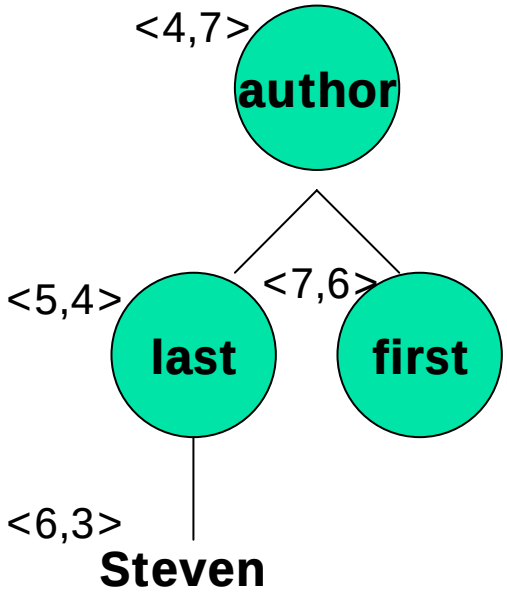
```
LP = {<4,7,author>, <5,4,last>, <6,3,Steven>, <7,6,first>,
<8,5,W.>}
```

1. lp = <4,7,author>
2. lp = <5,4,last>
3. lp = <6,3,Steven>
   => make a node t
   => find the “nearest” node p
   p.pre < 6 and p.post > 3



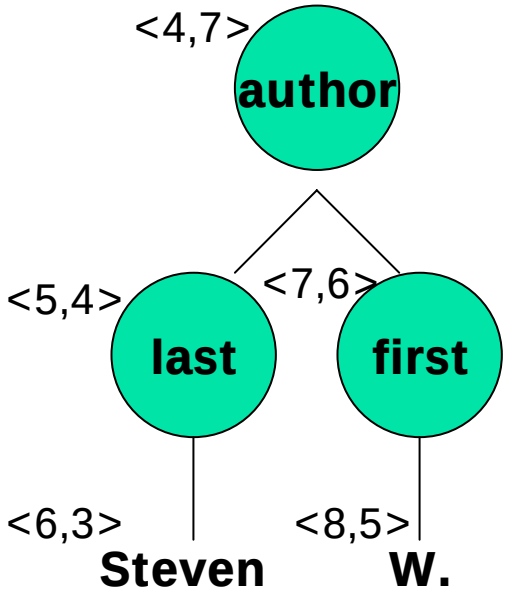
Tables -> Tree

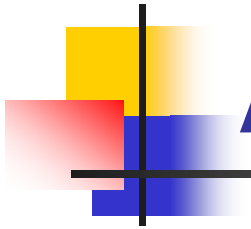
```
LP = {<4,7,author>, <5,4,last>, <6,3,Steven>, <7,6,first>,
<8,5,W.>}

1. lp = <4,7,author>
2. lp = <5,4,last>
3. lp = <6,3,Steven>
4. lp = <7,6,first>
        => make a node t
        => find the "nearest" node p
        p.pre < 7 and p.post > 6
        => p.addChild(t)
```



Tables -> Tree

```
LP = {<4,7,author>, <5,4,last>, <6,3,Steven>, <7,6,first>,
<8,5,W.>}

1. lp = <4,7,author>
2. lp = <5,4,last>
3. lp = <6,3,Steven>
4. lp = <7,6,first>
5. lp = <8,5,W.>
        => make a node t
        => find the "nearest" node p
        p.pre < 8 and p.post > 5
        => p.addChild(t)
```





Another Algorithm

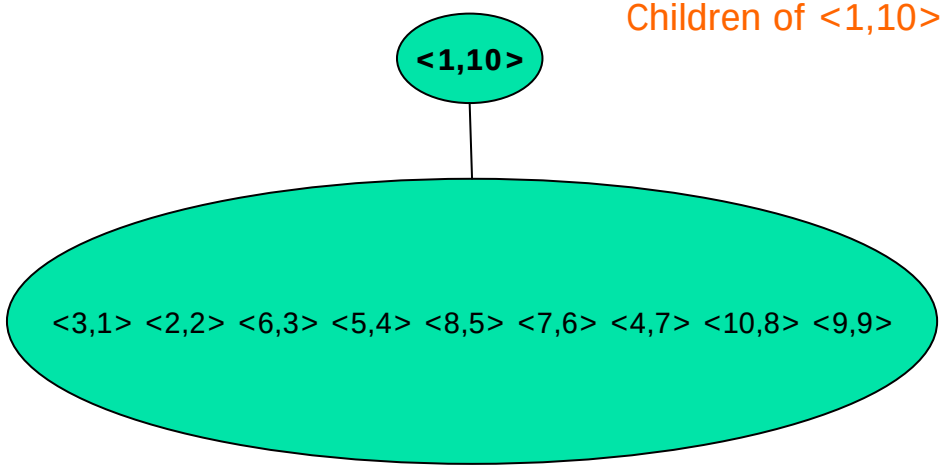
- pre-order: <1,10> <2,2> <3,1> <4,7> <5,4> <6,3> <7,6> <8,5> <9,9> <10,8>

- post-order: <3,1> <2,2> <6,3> <5,4> <8,5> <7,6> <4,7> <10,8> <9,9> <1,10>

Another Algorithm

- pre-order: <1,10> <2,2> <3,1> <4,7> <5,4> <6,3> <7,6> <8,5> <9,9> <10,8>

- post-order: <3,1> <2,2> <6,3> <5,4> <8,5> <7,6> <4,7> <10,8> <9,9> <1,10>



Another Algorithm

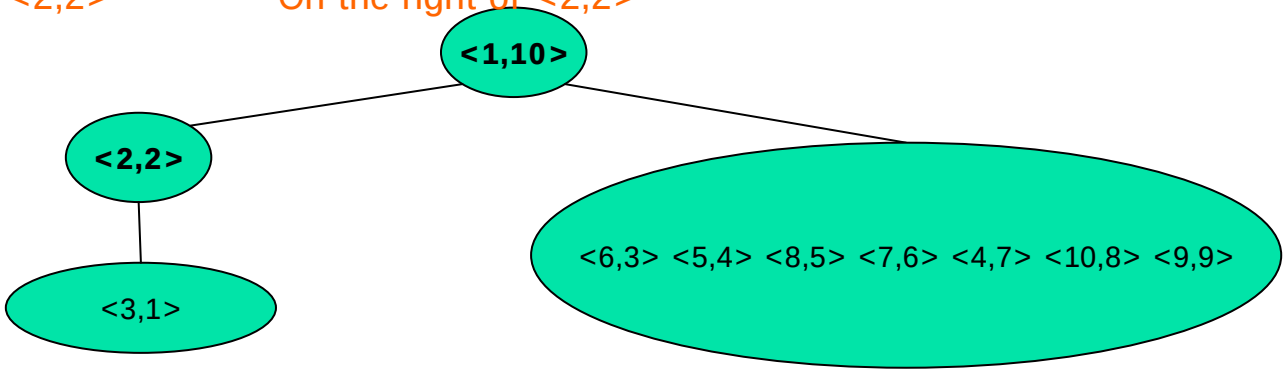
- pre-order: <2,2> <3,1> <4,7> <5,4> <6,3> <7,6> <8,5> <9,9> <10,8>

The subroot

- post-order: <3,1> <2,2> <6,3> <5,4> <8,5> <7,6> <4,7> <10,8> <9,9>

Children of <2,2>

On the right of <2,2>

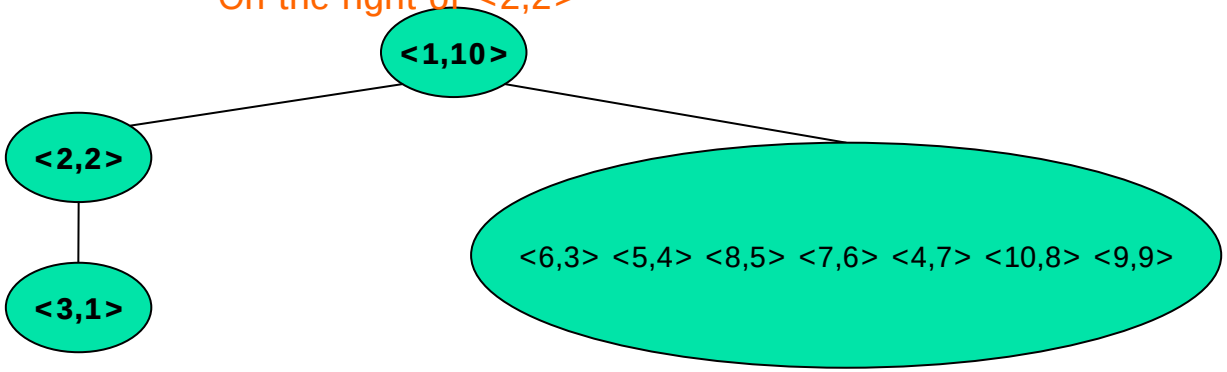


Another Algorithm

- pre-order: <3,1> <4,7> <5,4> <6,3> <7,6> <8,5> <9,9> <10,8>
- post-order: <3,1> <6,3> <5,4> <8,5> <7,6> <4,7> <10,8> <9,9>

The subroot

On the right of <2,2>



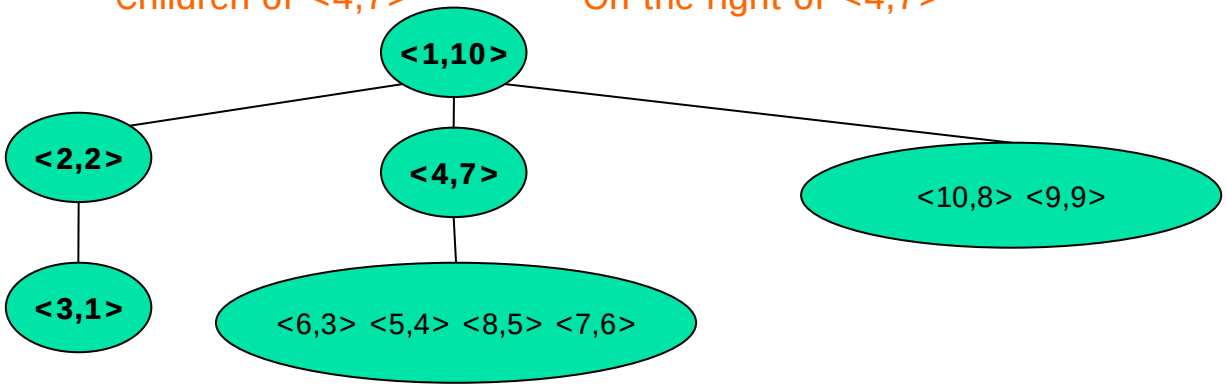
Another Algorithm

The subroot

- pre-order: <4,7> <5,4> <6,3> <7,6> <8,5> <9,9> <10,8>
- post-order: <6,3> <5,4> <8,5> <7,6> <4,7> <10,8> <9,9>

Children of <4,7>

On the right of <4,7>



Another Algorithm

- pre-order: <5,4> <6,3> <7,6> <8,5> <9,9> <10,8>

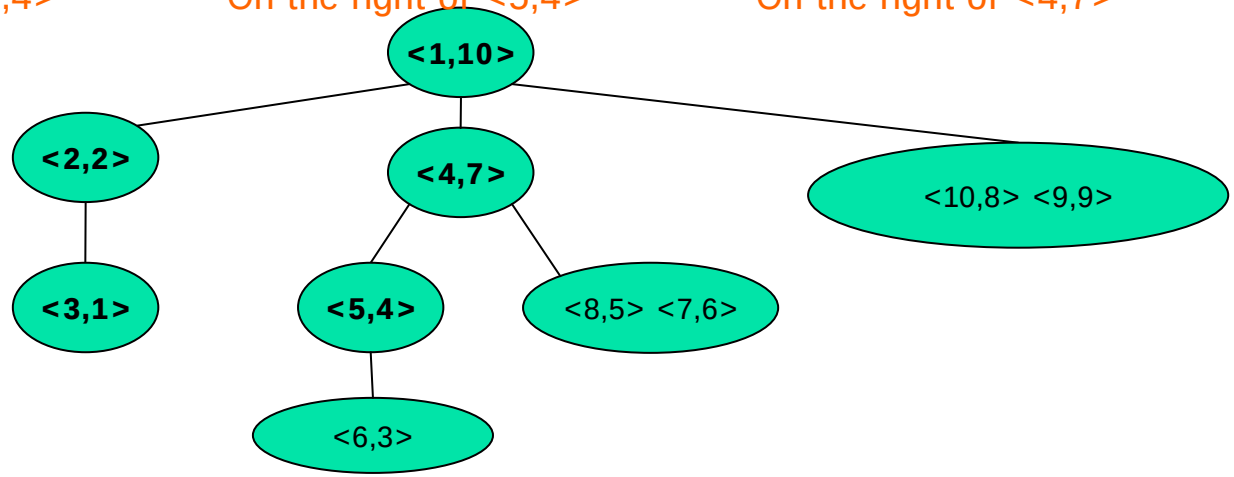
The subroot

- post-order: <6,3> <5,4> <8,5> <7,6> <10,8> <9,9>

Children of <5,4>

On the right of <5,4>

On the right of <4,7>



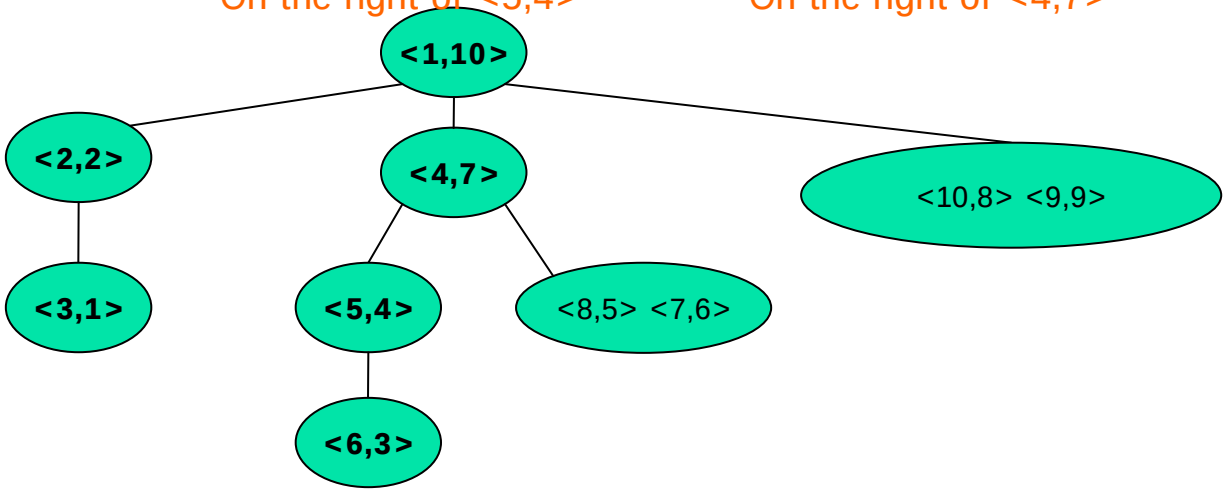
Another Algorithm

- pre-order: <6,3> <7,6> <8,5> <9,9> <10,8>
- post-order: <6,3> <8,5> <7,6> <10,8> <9,9>

The subroot

On the right of <5,4>

On the right of <4,7>



Another Algorithm

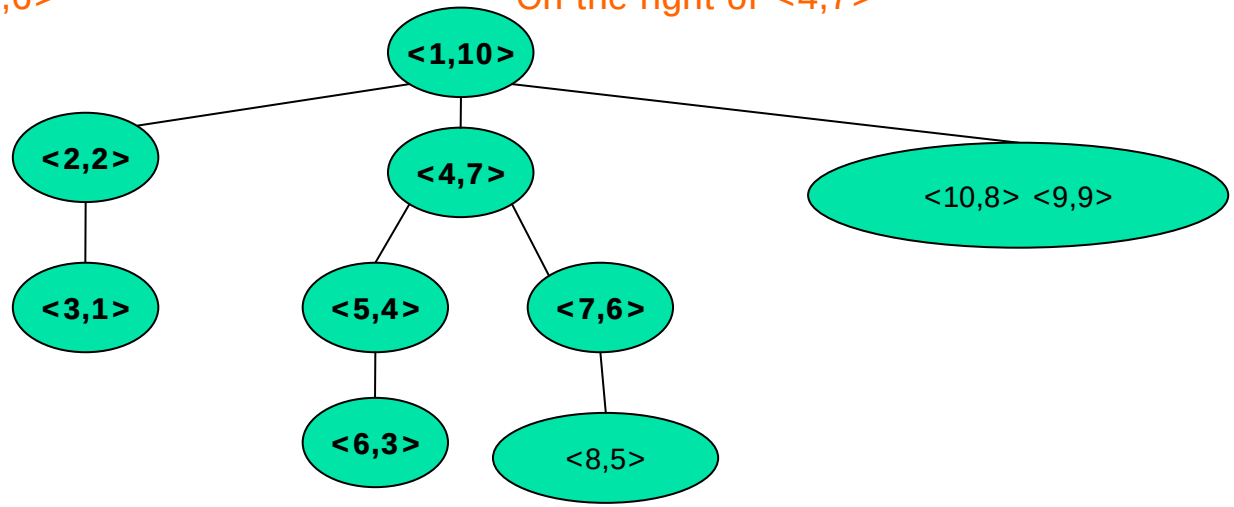
- pre-order: <7,6> <8,5> <9,9> <10,8>

The subroot

- post-order: <8,5> <7,6> <10,8> <9,9>

Children of <7,6>

On the right of <4,7>



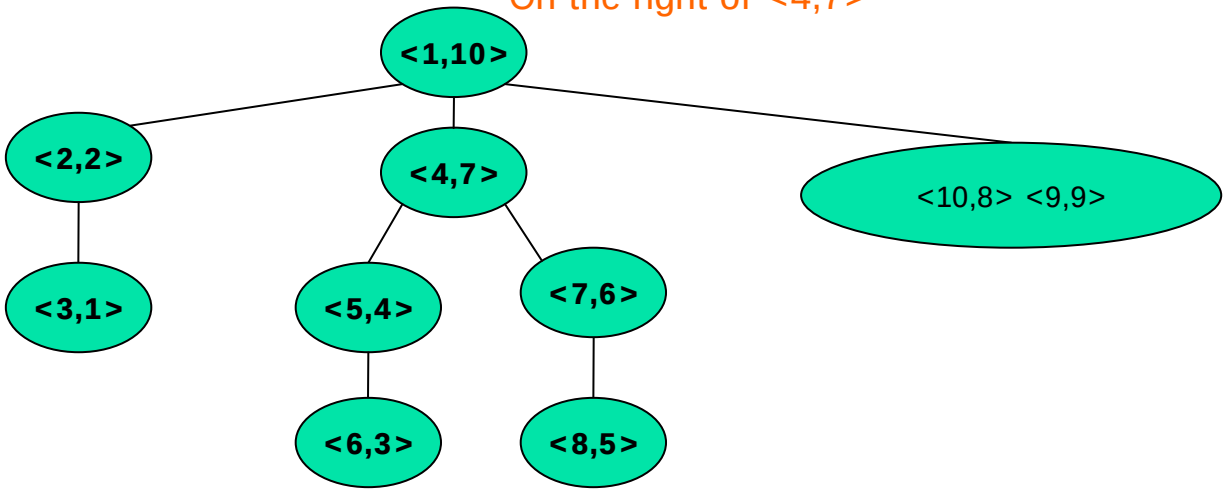
Another Algorithm

The subroot

- pre-order: <8,5> <9,9> <10,8>
- post-order: <8,5>

<10,8> <9,9>

On the right of <4,7>

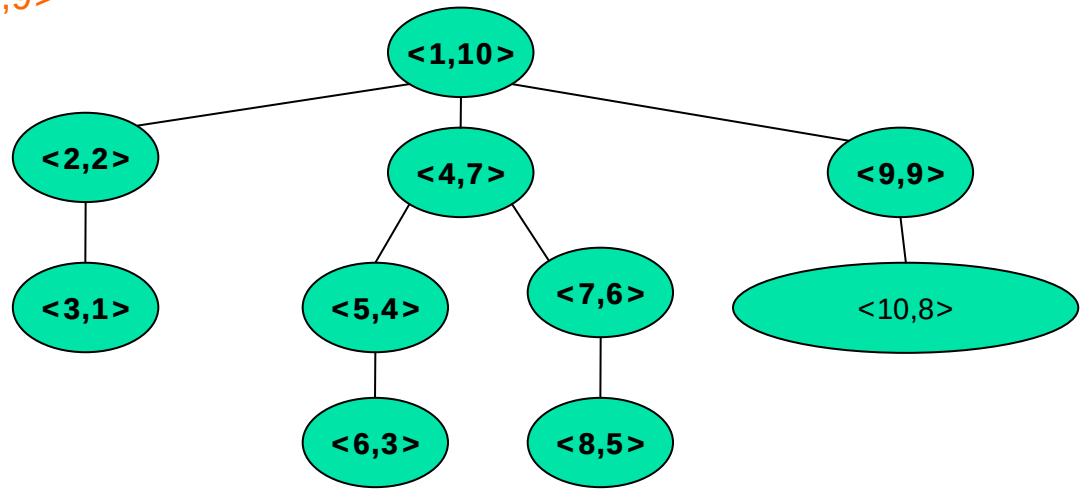


Another Algorithm

The subroot

- pre-order: <9,9> <10,8>
- post-order: <10,8> <9,9>

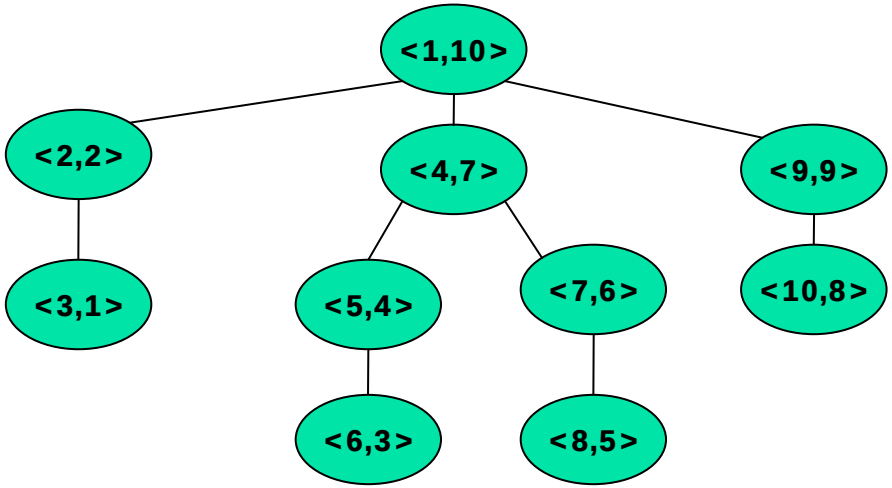
Children of <9,9>



Another Algorithm

The subroot

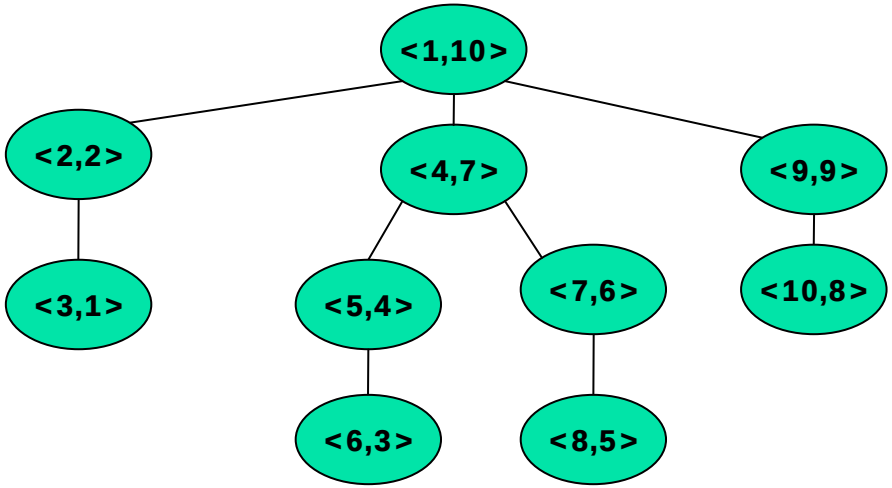
- pre-order: <10,8>
- post-order: <10,8>

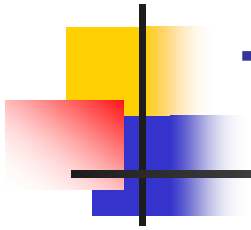


Another Algorithm

- pre-order:

- post-order:



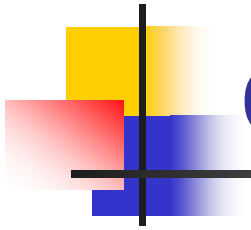


Tree -> XML

- Travel the tree
- Print as –p2 of assignment 1

<author>
<last>
Steven
</last>
<first>
W.
</first>
</author>

Mind your tablenames in –q option



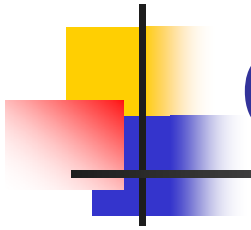
- SELECT pre, post
- FROM book_tbl
- WHERE pre IN
- (SELECT pre FROM book_attr
- WHERE attr = 'isbn' AND value = '1-2345-6789-0')

- SELECT pre, post
- FROM **z1234567_book_tbl**
- WHERE pre IN
- (SELECT pre FROM **z1234567_book_attr**
- WHERE attr = 'isbn' AND value = '1-2345-6789-0')

Filename => Tablename

- XML2Rel –t testcases/book.xml
  - tablename: book_tbl, book_attr

Extract the filename only



Q & A

- DO NOT use any default database, username, password in your program code
  - Receive them from the parameters
- We may create tables for many XML files before deleting them
  - Only retrieve and fix the necessary tablenames.
    - Insert data from XML files a1.xml, a2.xml, a3.xml
    - Query data from table a1_tbl, a1_attr: fix names for a1
    - Query data from table a3_tbl, a3_attr: fix names for a3
    - Delete all data (remove all a*i*_tbl, a*i*_attr tables)