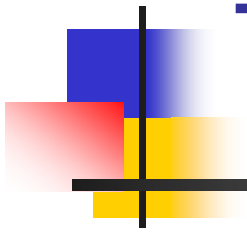
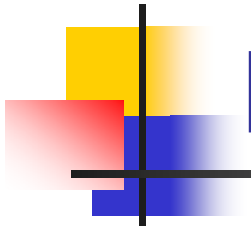


COMP4317: XML & Database Tutorial 3: SAX & DAG

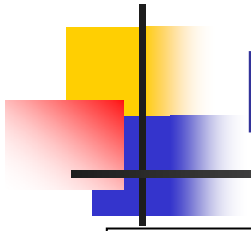
Week 4

Thang Bui @ CSE.UNSW



Hashtable in Java

- A map of key -> value
- Using:
  - import java.util.Hashtable;
  - boolean containsKey(Object key)
  - boolean containsValue(Object value)
  - put(Object key, Object value)
  - Object get(Object key)
  - boolean isEmpty()

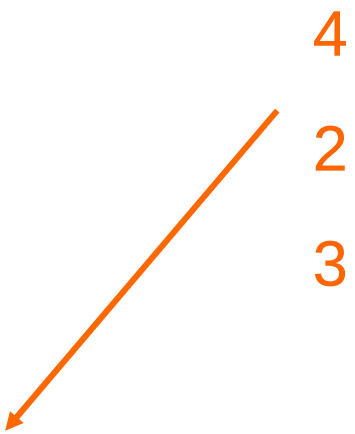


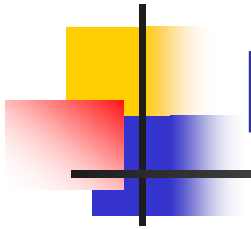
Hashtable in Java

```
import java.util.Hashtable;
class Tester {
 public static void main() {
  Hashtable htbl = new Hashtable();
  htbl.put("First", new Integer(1));
  htbl.put("Second", new Integer(2));
  htbl.put("Third", new Integer(3));
  htbl.put("First", new Integer(4));
  System.out.println("First: " + htbl.get("First"));
  System.out.println("Second: " + htbl.get("Second"));
  System.out.println("Third: " + htbl.get("Third"));
 }
}
```

Hashtable in Java

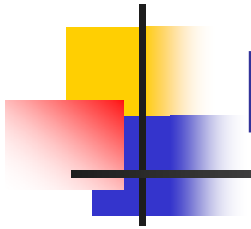
```
import java.util.Hashtable;
class Tester {
 public static void main() {
  Hashtable htbl = new Hashtable();
  htbl.put("First", new Integer(1));
  htbl.put("Second", new Integer(2));
  htbl.put("Third", new Integer(3));
  htbl.put("First", new Integer(4));
  System.out.println("First: " + htbl.get("First"));
  System.out.println("Second: " + htbl.get("Second"));
  System.out.println("Third: " + htbl.get("Third"));
 }
}
```





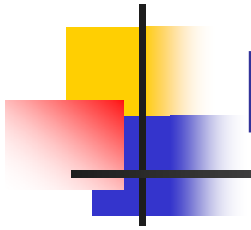
HashMap in Java

- import java.util.HashMap;
- = Hashtable + null key/value



HashMap in Java

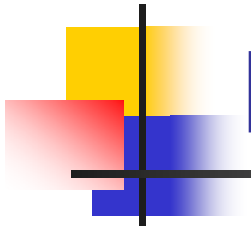
- Example
  - htbl.put(null, “Null key”);
  - htbl.put(“Null value”, null);
  - Error!



HashMap in Java

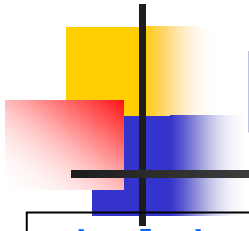
- Example
  - map.put(null, “Null key”);
  - map.put(“Null value”, null);
  - map.put(null, null);

  - Ok!



Map in C++

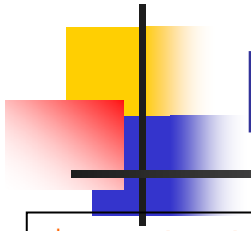
- A map of key -> value
- Using:
  - #include <map>
  - bool empty() const;
  - iterator find(const key_type& key);
  - TYPE& operator[](const key_type& key);



Map in C++

http://www.cppreference.com/cppmap/index.html

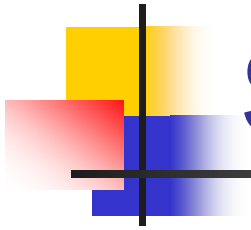
```
#include <map>
#include <string>
using namespace std;
int main() {
  map<string,int> stringCounts;
  string str;

  while( cin >> str ) stringCounts[str]++;

  map<string,int>::iterator iter = stringCounts.find("spoon");
  if( iter != stringCounts.end() ) {
    cout << "You typed '" << iter->first << "' " << iter->second << " time(s)"
<< endl;
  }
  return 0;
}
```



Map in C++

http://www.cppreference.com/cppmap/index.html

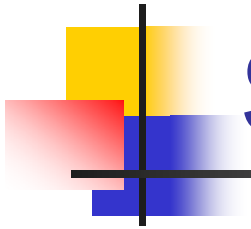
```
$g++ test.cpp
$./a.out
my spoon is too big.
My spoon is TOO big!
My SPOON is TOO big!
^D
You typed 'spoon' 2 time(s)
```



SAX & DAG

- startElement event:
  - know element name
  - know the parent
  - don’t know the children (subtree)

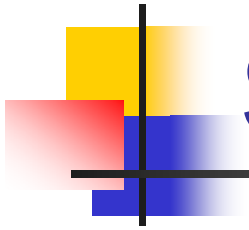
  - => can’t build the DAG from here!



SAX & DAG

- endElement event:
  - know element name
  - know the parent
  - know the children (subtree)

  - => build the DAG from here!



SAX & DAG

- endElement event:
  - know element name
  - know the parent
  - know the children (subtree)

How

  - => build the DAG from here!

Subtree content – on-the-fly

	Events	Subtree content

<book>
  <title>
  TCP/IP Illustrated
  </title>
  <author>
  <first>
  Stevens
  </first>
  <last>
  W.
  </last>
  </author>
</book>
	1. start: “book”	book=“book(”

Subtree content – on-the-fly

	Events	Subtree content

<book>
  <title>
  TCP/IP Illustrated
  </title>
  <author>
  <first>
  Stevens
  </first>
  <last>
  W.
  </last>
  </author>
</book>
	1. start: “book”
	2. start: “title”
		book=“book(”
		title=“title(”

Subtree content – on-the-fly

	Events	Subtree content

<book>
  <title>
  TCP/IP Illustrated
  </title>
  <author>
  <first>
  Stevens
  </first>
  <last>
  W.
  </last>
  </author>
</book>
	1. start: “book”
	2. start: “title”
	
	3. end: “title”
		title=“title()”
		
		⇒ update parent content
		
		book=“book(title()”

Subtree content – on-the-fly

	Events	Subtree content

<book>
  <title>
  TCP/IP Illustrated
  </title>
  <author>
  <first>
  Stevens
  </first>
  <last>
  W.
  </last>
  </author>
</book>
	1. start: “book”
	2. start: “title”
	
	3. end: “title”
	4. start: “author”
		book=“book(title()”
		title=“title()”
		author=“author(”

Subtree content – on-the-fly

	Events	Subtree content

<book>
  <title>
  TCP/IP Illustrated
  </title>
  <author>
  <first>
  Stevens
  </first>
  <last>
  W.
  </last>
  </author>
</book>
	1. start: “book”
	2. start: “title”
	
	3. end: “title”
	4. start: “author
	5. start: “first”
		book=“book(title()”
		title=“title()”
		author=“author(”
		first=“first(”

Subtree content – on-the-fly

	Events	Subtree content

<book>
  <title>
  TCP/IP Illustrated
  </title>
  <author>
  <first>
  Stevens
  </first>
  <last>
  W.
  </last>
  </author>
</book>
	1. start: “book”
	2. start: “title”
	
	3. end: “title”
	4. start: “author
	5. start: “first”
	
	6. end: “first”
		book=“book(title()”
		title=“title()”
		first=“first()”
		
		⇒ update parent content
		
		author=“author(first()”

Subtree content – on-the-fly

	Events	Subtree content

<book>
  <title>
  TCP/IP Illustrated
  </title>
  <author>
  <first>
  Stevens
  </first>
  <last>
  W.
  </last>
  </author>
</book>
	1. start: “book”
	2. start: “title”
	
	3. end: “title”
	4. start: “author
	5. start: “first”
	
	6. end: “first”
	7. start: “last”
		book=“book(title()”
		title=“title()”
		author=“author(first()”
		first=“first()”
		last=“last(”

Subtree content – on-the-fly

	Events	Subtree content

<book>
  <title>
  TCP/IP Illustrated
  </title>
  <author>
  <first>
  Stevens
  </first>
  <last>
  W.
  </last>
  </author>
</book>
	1. start: “book”
	2. start: “title”
	
	3. end: “title”
	4. start: “author
	5. start: “first”
	
	6. end: “first”
	7. start: “last”
	
	8. end: “last”
		book=“book(title()”
		title=“title()”
		first=“first()”
		last=“last()”
		
		⇒ update parent content
		
		author=“author(first(),last()”

Subtree content – on-the-fly

	Events	Subtree content

<book>
  <title>
  TCP/IP Illustrated
  </title>
  <author>
  <first>
  Stevens
  </first>
  <last>
  W.
  </last>
  </author>
</book>
	1. start: “book”
	2. start: “title”
	
	3. end: “title”
	4. start: “author
	5. start: “first”
	
	6. end: “first”
	7. start: “last”
	
	8. end: “last”
	9. end: “author”
		title=“title()”
		author=“author(first(),last())”
		first=“first()”
		last=“last()”
		
		⇒ update parent content
		
		book=“book(title(), author(first(),last())”

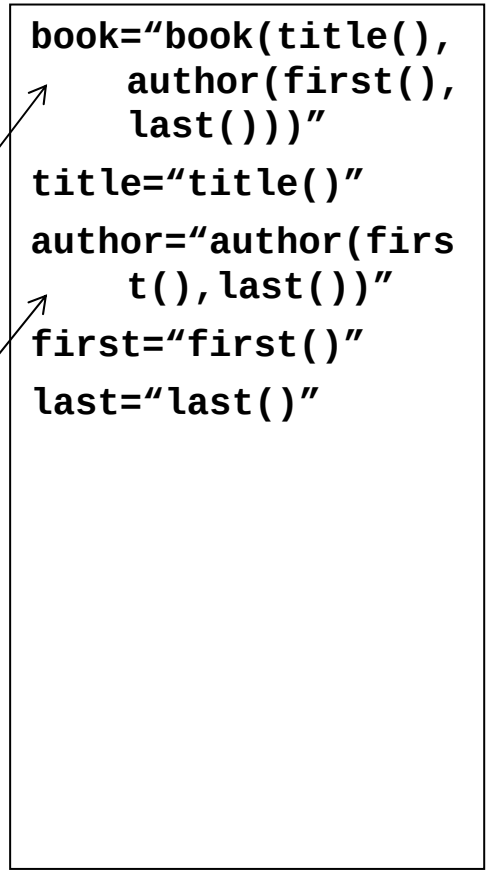
Subtree content – on-the-fly

	Events	Subtree content

<book>
  <title>
  TCP/IP Illustrated
  </title>
  <author>
  <first>
  Stevens
  </first>
  <last>
  W.
  </last>
  </author>
</book>
	1. start: “book”
	2. start: “title”
	3. end: “title”
	4. start: “author
	5. start: “first”
	6. end: “first”
	7. start: “last”
	8. end: “last”
	9. end: “author”
	10. end: “book”
		book=“book(title(),
		author(first(),last()
		))”
		title=“title()”
		author=“author(first(),las
		t())”
		first=“first()”
		last=“last()”

Subtree content – Efficient encoding

- Hastable:
  - “title()” → pointer p2
  - “author(…)” → pointer p3
  - “first()” → pointer p4
  - “last()” → pointer p5
- New encoding:
  - "‘book’(p1,p2)"
  - "‘author’(p3,p4)"



Subtree content – Efficient encoding

- The Hashtable

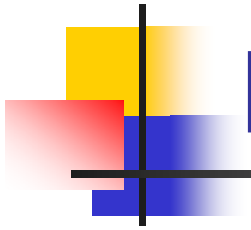
Key (as “label(list of pointers)”)	Value (as pointer)
‘title’()	p2
‘first’()	p4
‘last’()	p5
‘author’(p4,p5)	p3
‘book’(p2,p3)	p1

Subtree content – Efficient encoding

- Search for a key
  - get("'title'()") -> p2
  - get("'author'(p4,p5)") -> p3

Key (as "label(list of pointers)")	Value (as pointer)
'title'()	p2
'first'()	p4
'last'()	p5
'author'(p4,p5)	p3
'book'(p2,p3)	p1

Subtree content – Efficient encoding



- Suppose we have function getContent:
  - getContent(p2) => ‘title’()
  - getContent(p1) => ‘book’(p2,p3)

Key (as “label(list of pointers)”)	Value (as pointer)
‘title’()	p2
‘first’()	p4
‘last’()	p5
‘author’(p4,p5)	p3
‘book’(p2,p3)	p1

Subtree content – Efficient encoding

- New Hashtable

Old Key (as "label(list of pointers)")	Key (as pointer)	Value (as pointer)
'title'()	p2	p2
'first'()	p4	p4
'last'()	p5	p5
'author'(p4,p5)	p3	p3
'book'(p2,p3)	p1	p1

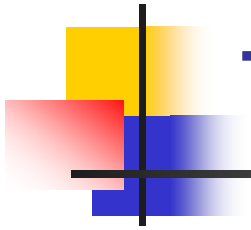
Subtree content – Efficient encoding

- get(pointer x) → return a pointer/NULL

```
x_content = getContent(x)
for each key y (as pointer) in hashtable
   y_content = getContent(y)
   if (y_content == x_content) return y
```

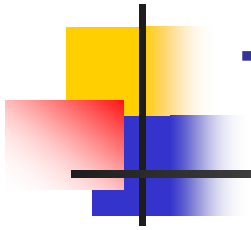
Old Key (as "label(list of pointers)")	Key (as pointer)	Value (as pointer)
'title'()	p2	p2
'first'()	p4	p4
'last'()	p5	p5
'author'(p4,p5)	p3	p3
'book'(p2,p3)	p1	p1

Efficient encoding – The new Hashtable



- Hashtable<pointer, pointer>

Key (as pointer)	Value (as pointer)
p2	p2
p4	p4
p5	p5
p3	p3
p1	p1

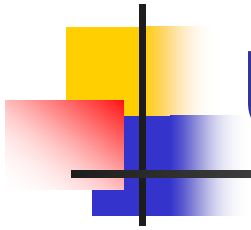


Efficient encoding – The new Hashtable

- More efficient Hashtable search?
  - Provided:
    - public int hashCode()
    - public boolean equals(Object x)
      - Compare **label**
      - Compare the **pointers** of all children
  - Using the get method of Hashtable class
    - That's it !!!

Efficient encoding – The new Hashtable

```
class myTree {
 String label;
 myTree left,right;
 public int hashCode() { return label.hashCode(); }
 public boolean equals(Object x) {
  if (this == x) return true;
  if (x == null) false;
  if (!(x instanceOf myTree)) return false;
  myTree y = (myTree)x;
  return (this.label.equals(y.label))
    && ((this.left == null)?(y.left == null):(this.left == y.left))
    && ((this.right == null)?(y.right == null):(this.right == y.right));
  }
}
Hashtable<myTree, myTree> htbl;
```



Unranked DAG Tree

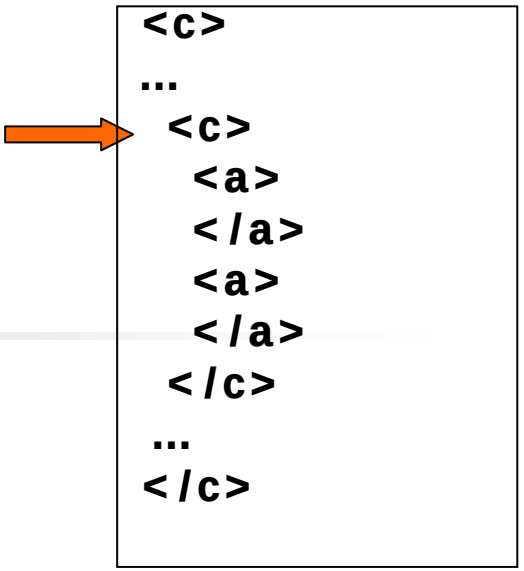
- For each tree node p:
  - Search on the Hashtable -> pointer p1
  - If p1 is already there, use p1
  - Else,
    - put <p,p> to the Hashtable
    - use p

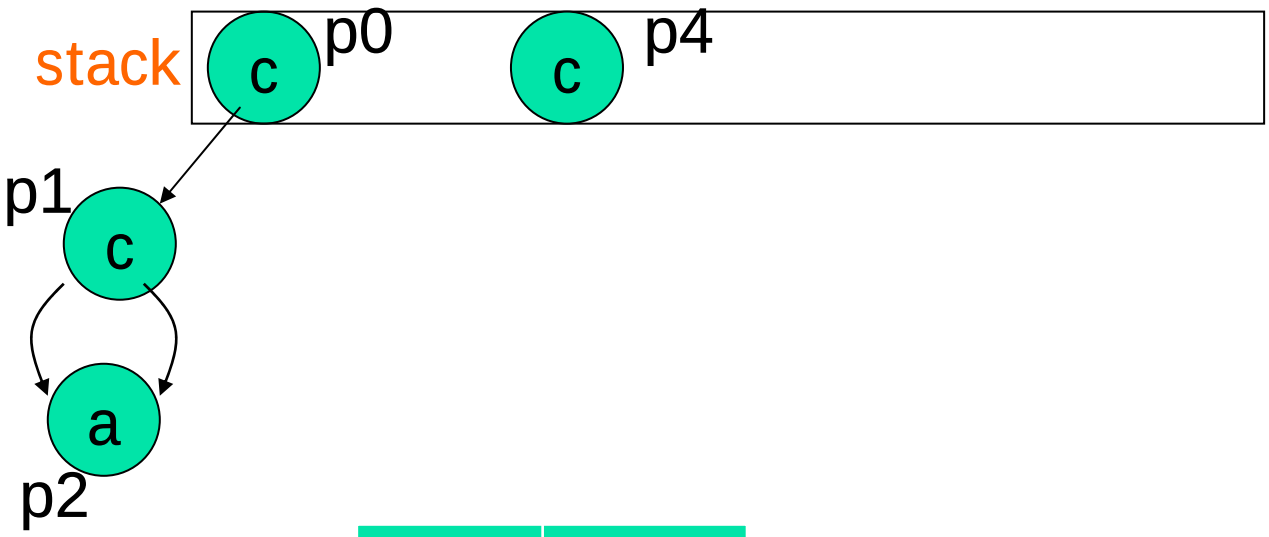
Unranked DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
  <a>
  </a>
 </c>
 ...
</c>
```

Key	Value
p2	p2
p1	p1

Unranked DAG Tree

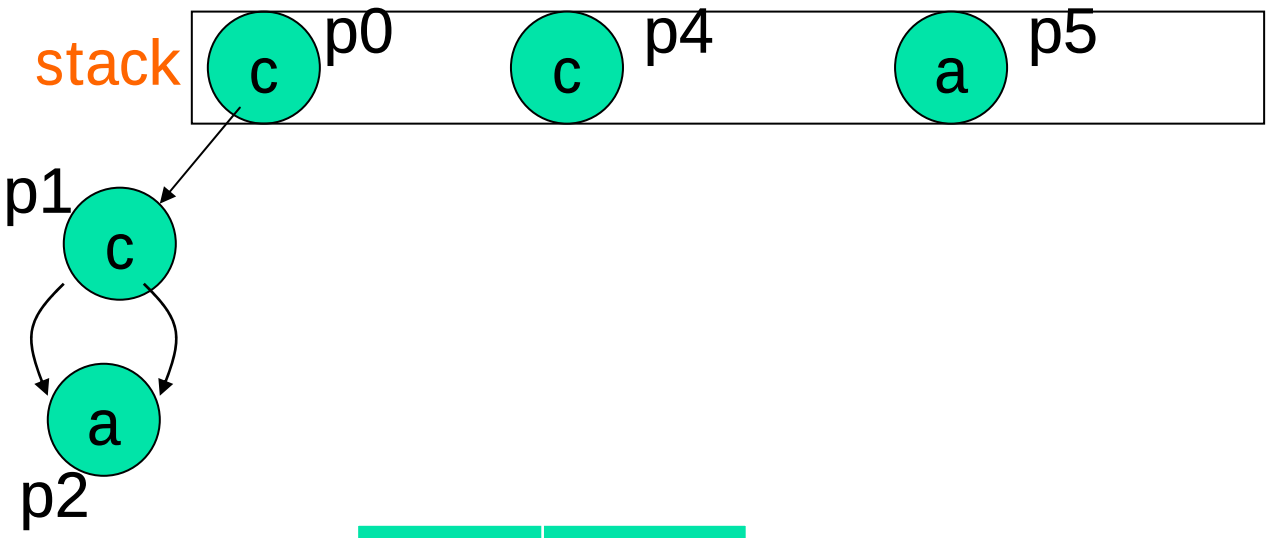




Key	Value
p2	p2
p1	p1

Unranked DAG Tree

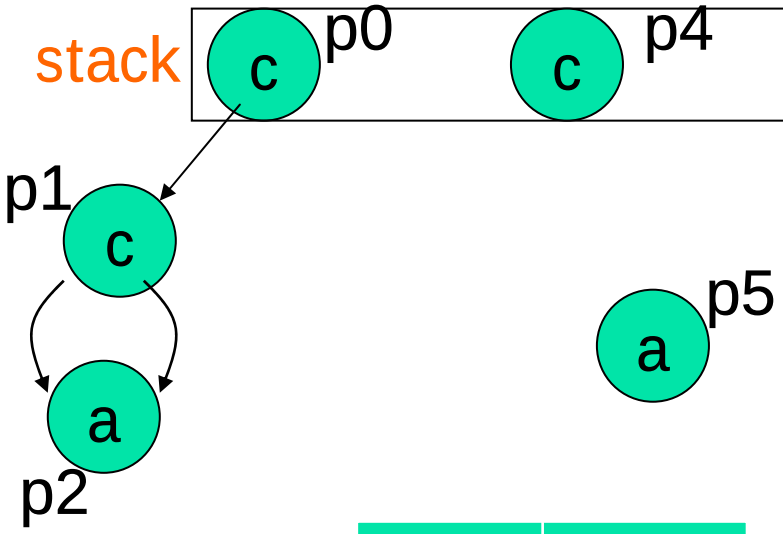
```
<c>
...
 <c>
  <a>
  </a>
  <a>
  </a>
 </c>
 ...
</c>
```



Key	Value
p2	p2
p1	p1

Unranked DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
  <a>
  </a>
 </c>
 ...
</c>
```



Key	Value
p2	p2
p1	p1

htbl.get(p5)

- p1 vs. p5 :

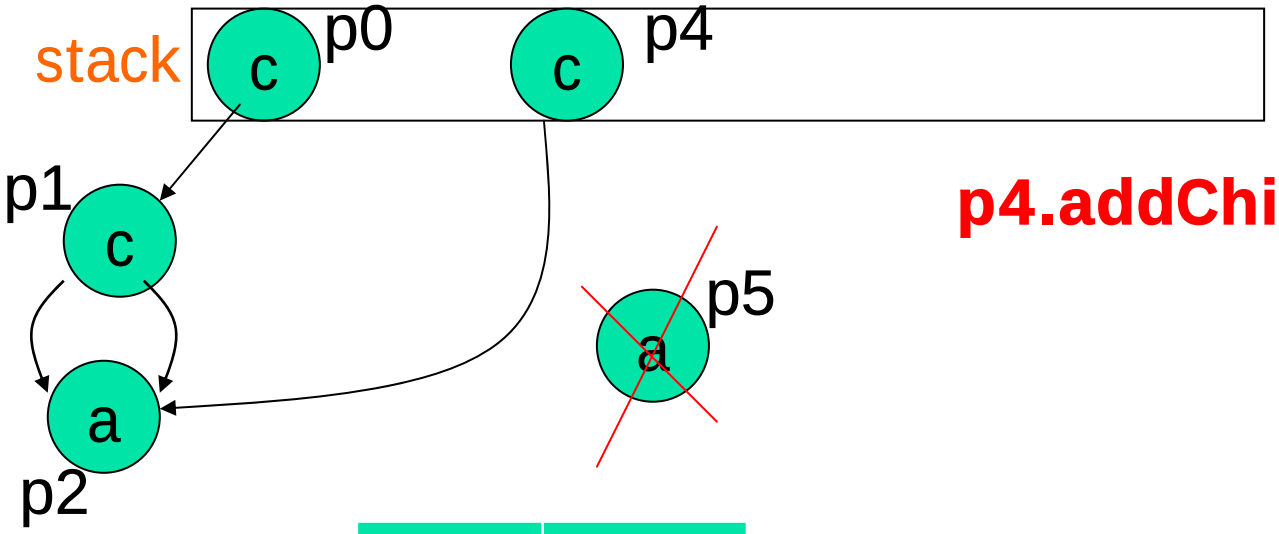
  ‘c’(p2,p2) vs. ‘a’()

- p2 vs. p5 :

  ‘a’() vs. ‘a’() => p2

Unranked DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
  <a>
  </a>
 </c>
 ...
</c>
```

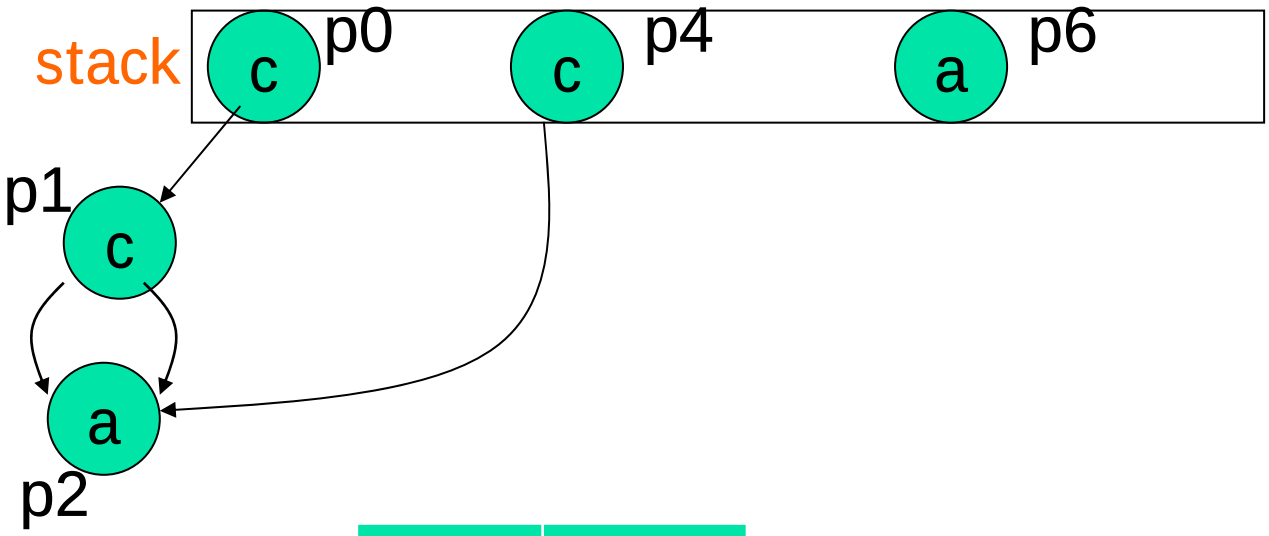


p4.addChild(p2)

Key	Value
p2	p2
p1	p1

Unranked DAG Tree

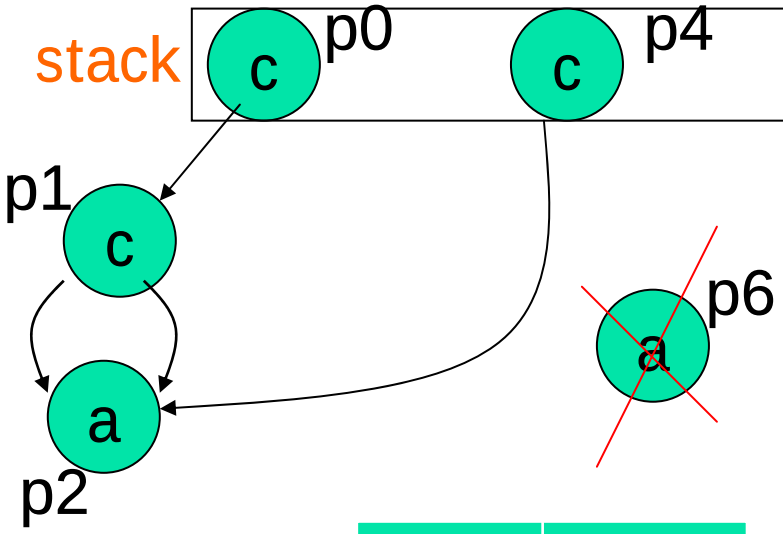
```
<c>
...
 <c>
  <a>
  </a>
  <a>
  </a>
 </c>
 ...
</c>
```



Key	Value
p2	p2
p1	p1

Unranked DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
  <a>
  </a>
 </c>
 ...
</c>
```



Key	Value
p2	p2
p1	p1

htbl.get(p6)

- p1 vs. p6 :

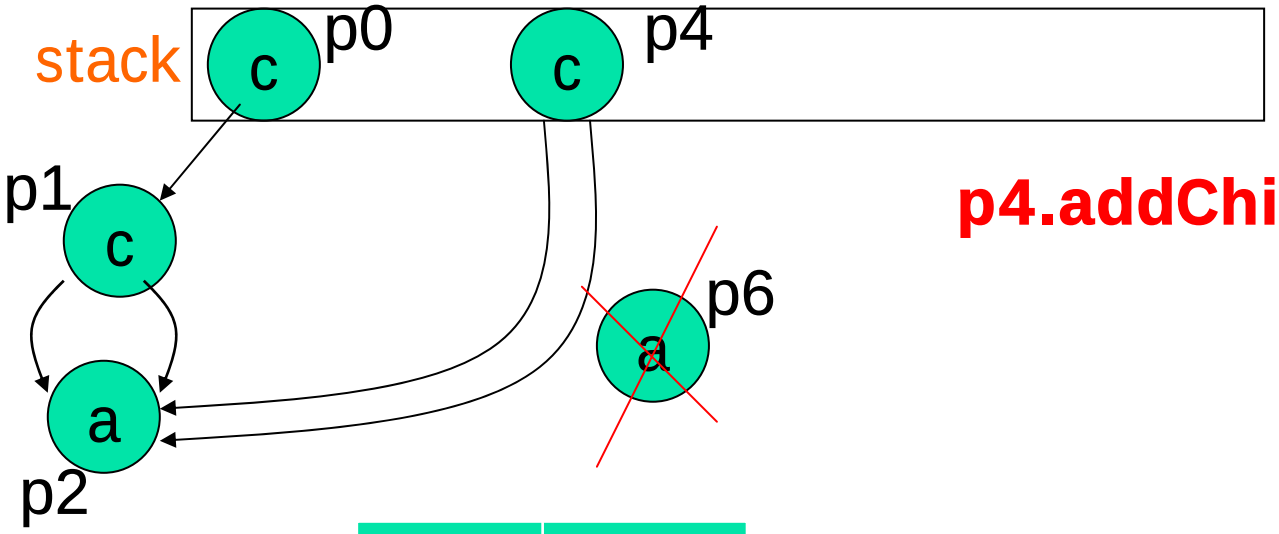
  ‘c’(p2,p2) vs. ‘a’()

- p2 vs. p6 :

  ‘a’() vs. ‘a’() => p2

Unranked DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
  <a>
  </a>
 </c>
 ...
</c>
```

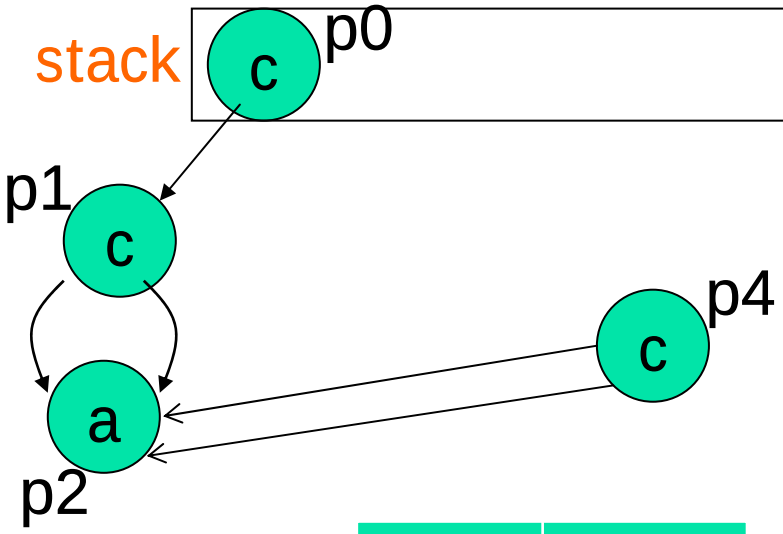


p4.addChild(p2)

Key	Value
p2	p2
p1	p1

Unranked DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
  <a>
  </a>
 </c>
 ...
</c>
```



Key	Value
p2	p2
p1	p1

htbl.get(p4)

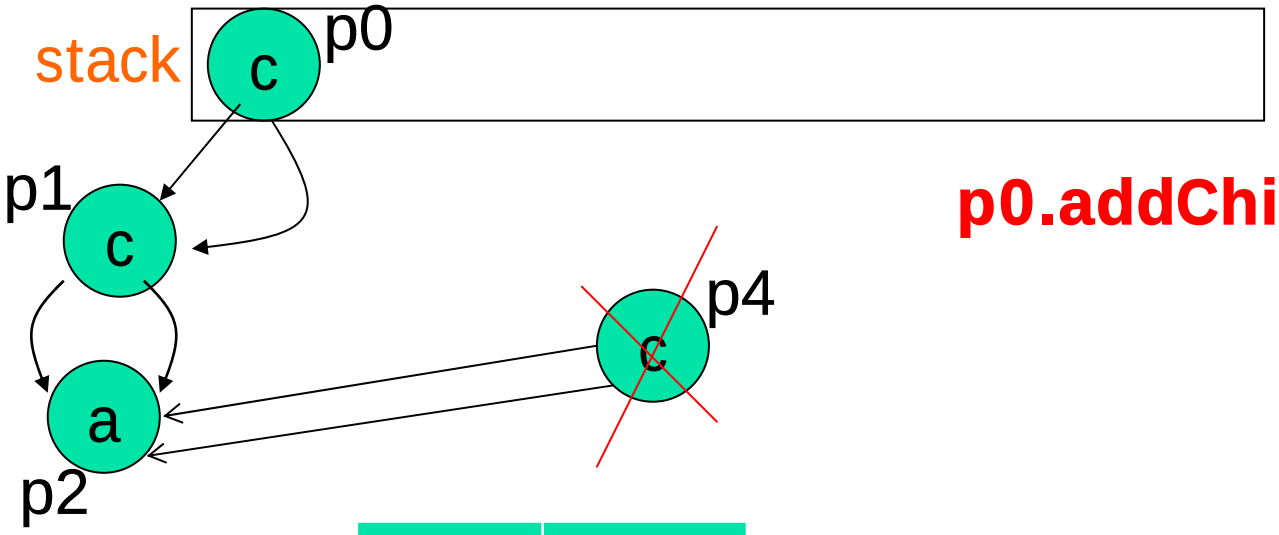
- p1 vs. p4 :

‘c’(p2,p2) vs. ‘c’(p1,p2)

=> p1

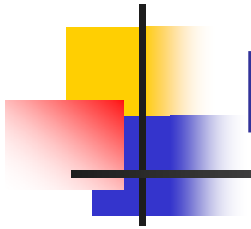
Unranked DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
  <a>
  </a>
 </c>
 ...
</c>
```



p0.addChild(p1)

Key	Value
p2	p2
p1	p1

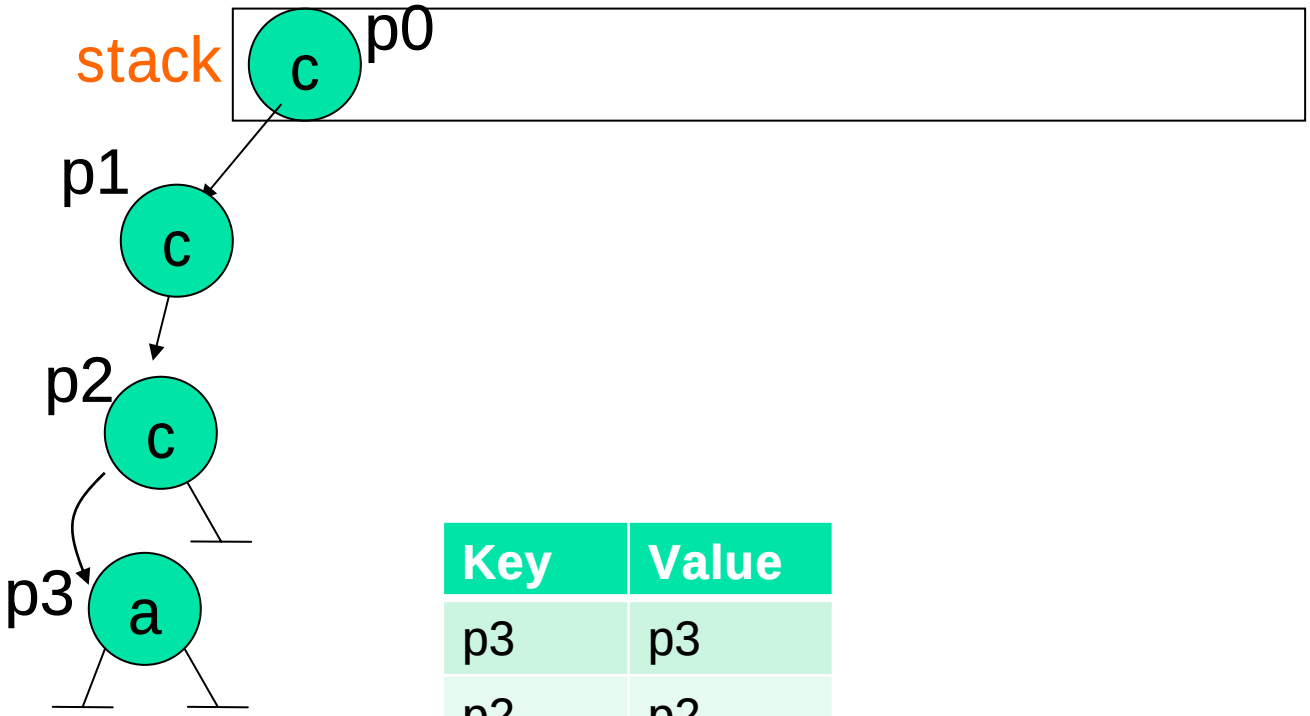


Binary DAG Tree

- Wait until we close the parent:
  - Two role of a child: a child and a previous sibling
  - Collapse the child from the right most child

Binary DAG Tree

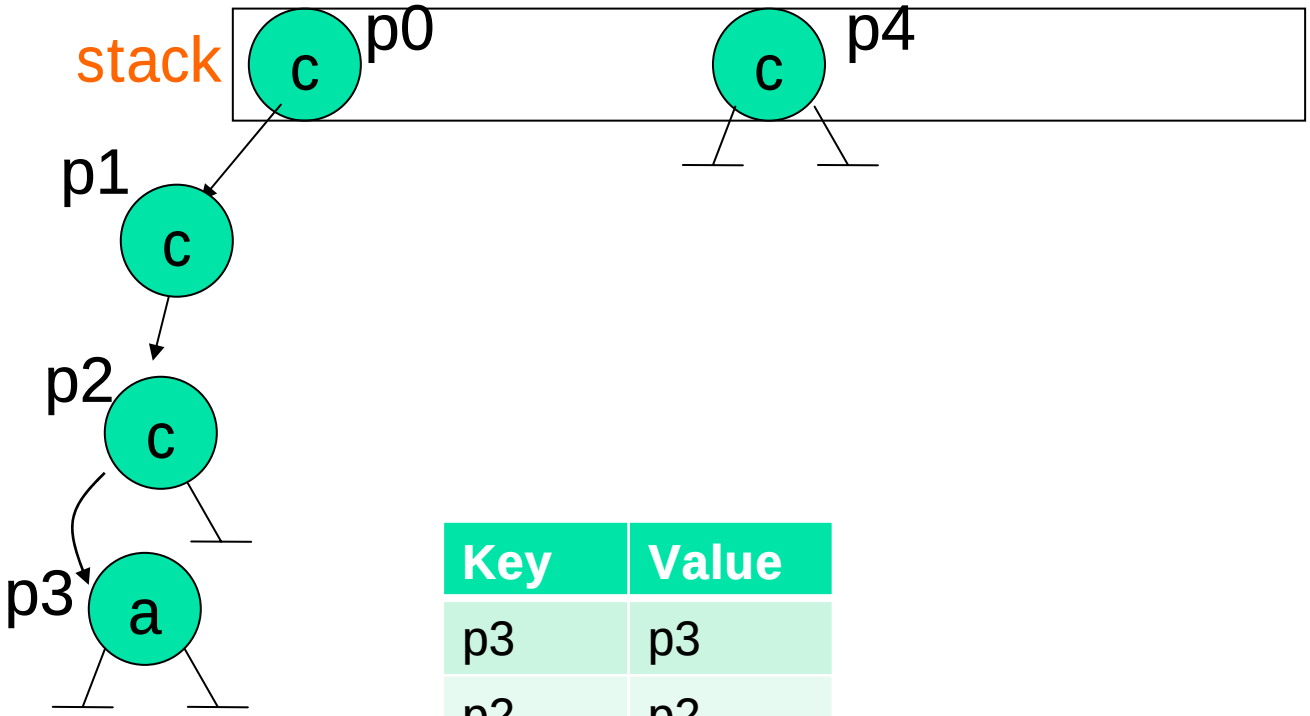
```
<c>
...
 <c>
  <a>
  </a>
 </c>
</c>
```



Key	Value
p3	p3
p2	p2

Binary DAG Tree

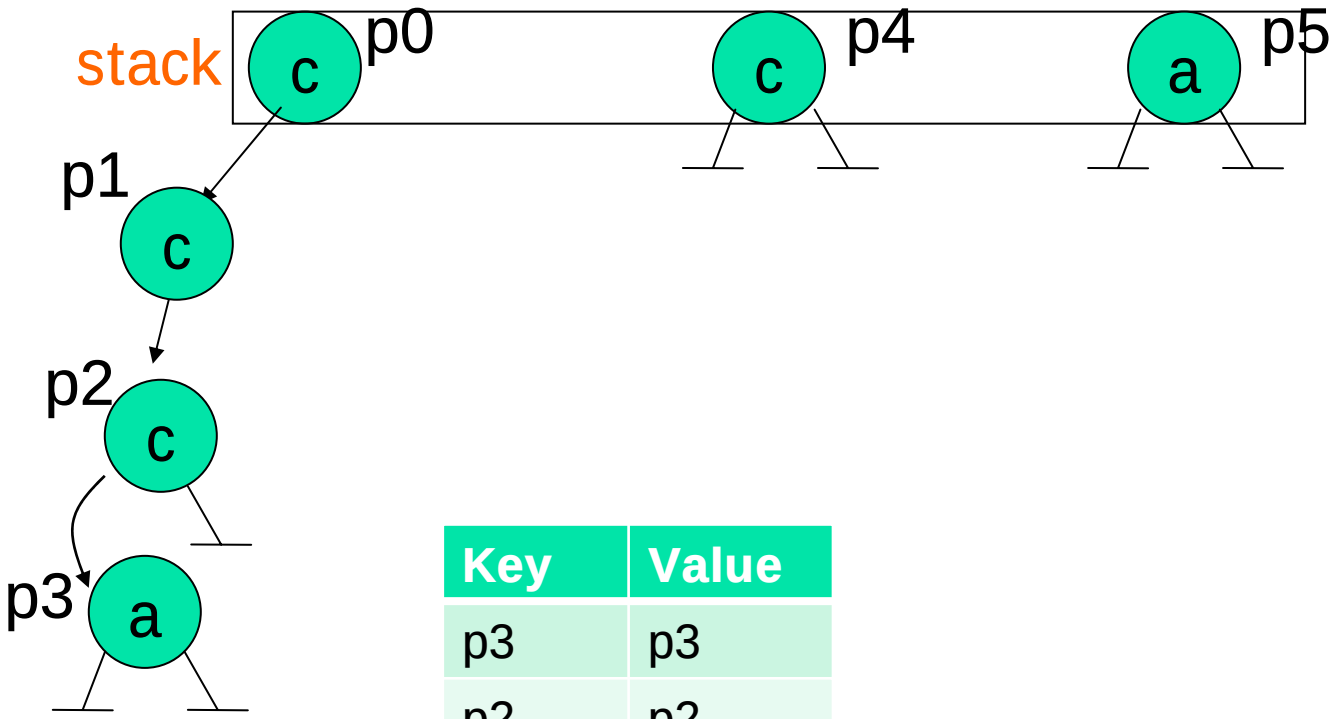
```
<c>
...
 <c>
  <a>
  </a>
 </c>
</c>
```



Key	Value
p3	p3
p2	p2

Binary DAG Tree

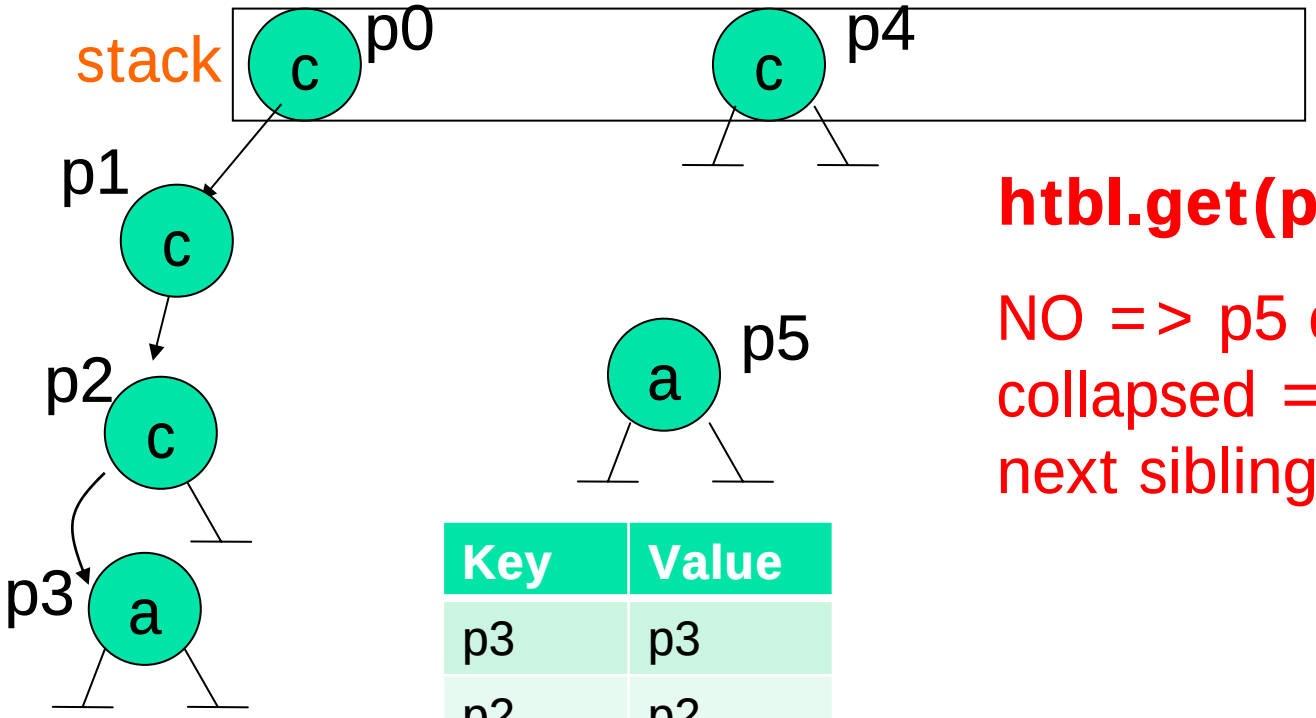
```
<c>
...
  <c>
    <a>
    </a>
  </c>
</c>
```



Key	Value
p3	p3
p2	p2

Binary DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
 </c>
</c>
```



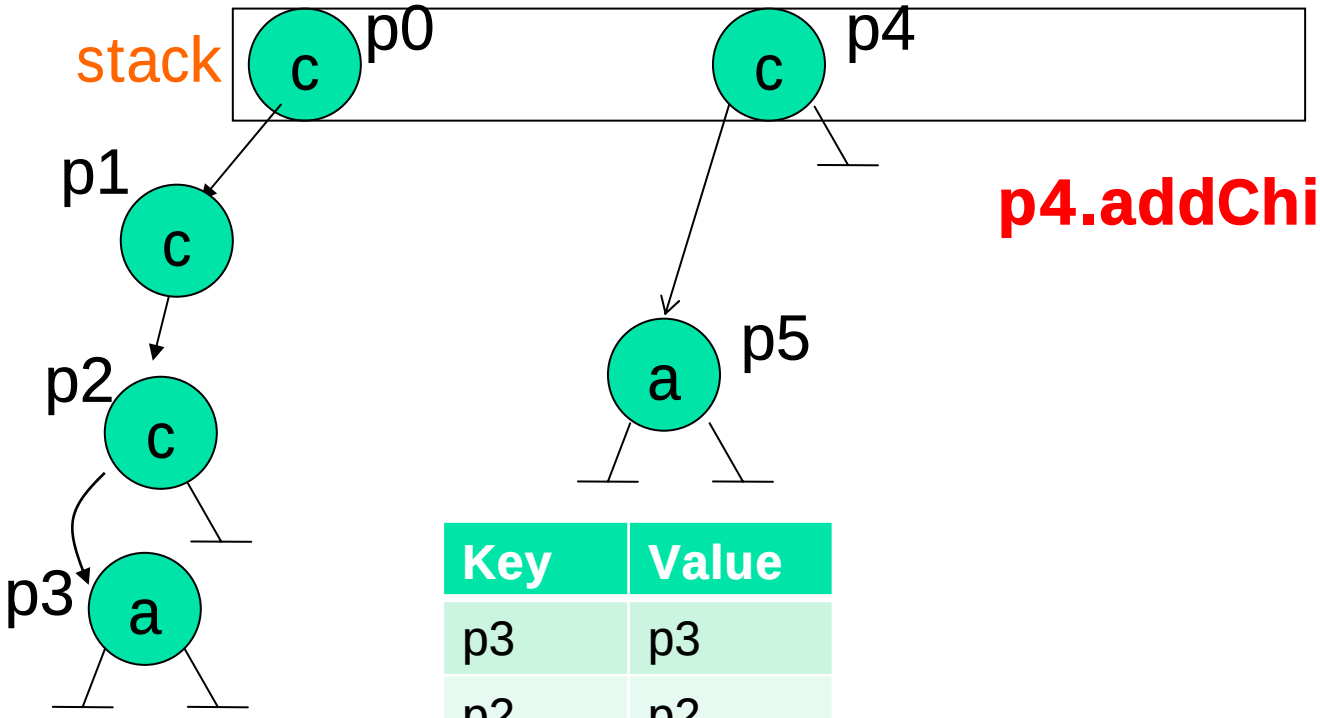
Key	Value
p3	p3
p2	p2

htbl.get(p5) ?

NO => p5 cannot be collapsed => wait for its next sibling

Binary DAG Tree

```
<c>
...
  <c>
    <a>
    </a>
  </c>
</c>
```

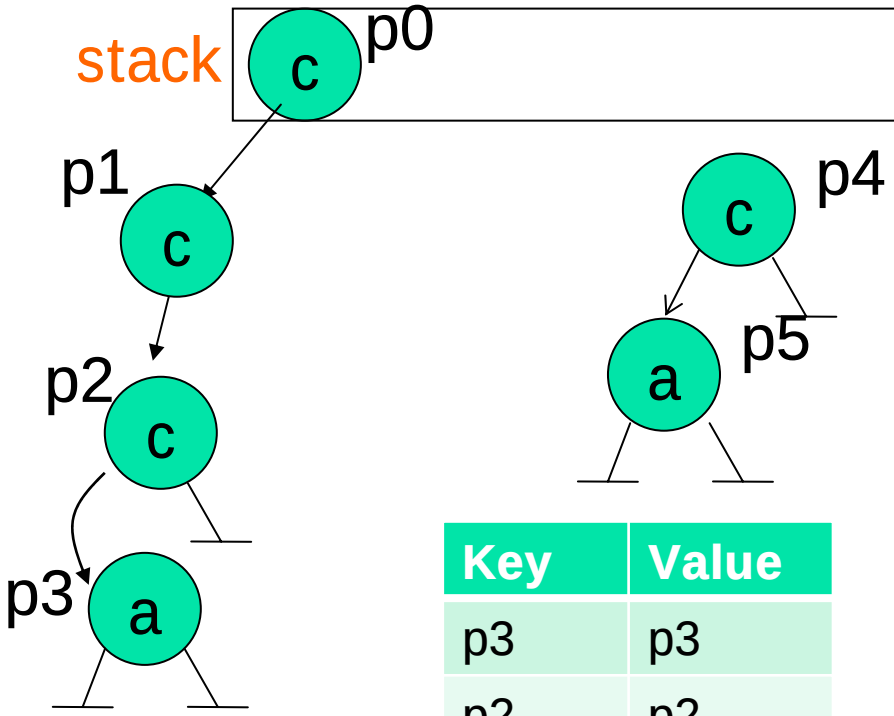


p4.addChild(p5)

Key	Value
p3	p3
p2	p2

Binary DAG Tree

```
<c>
...
  <c>
    <a>
    </a>
  </c>
</c>
```



Key	Value
p3	p3
p2	p2

Collapse children of p4

- htbl.get(p5)

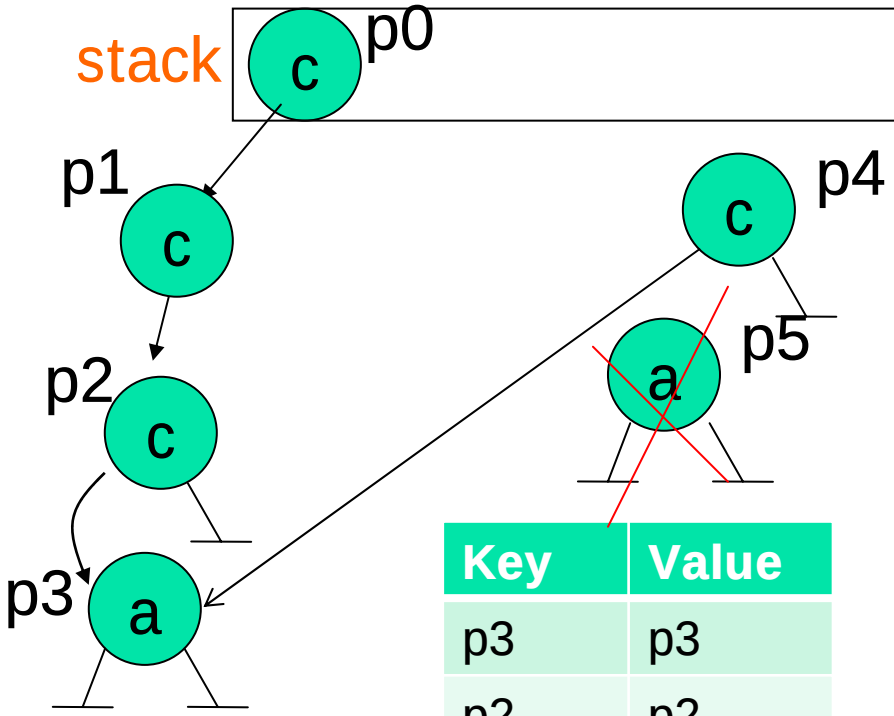
+ p3 vs. p5 :

‘a’(_,_) vs. ‘a’(_,_)

=> p3

Binary DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
 </c>
</c>
```



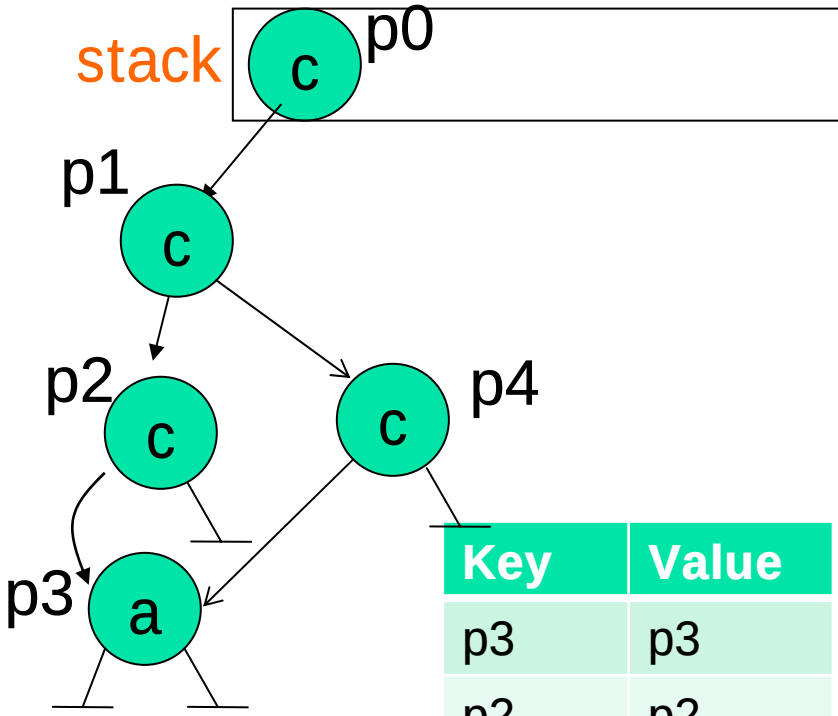
Key	Value
p3	p3
p2	p2

Collapse children of p4

- replace p4.left by p3

Binary DAG Tree

```
<c>
...
  <c>
    <a>
    </a>
  </c>
</c>
```



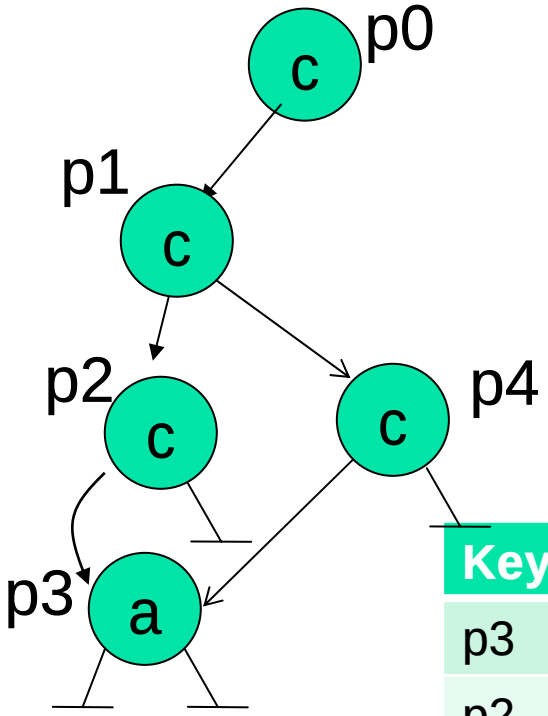
p0.addChild(p4)

Key	Value
p3	p3
p2	p2

Binary DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
 </c>
</c>
```

stack



Key	Value
p3	p3
p2	p2

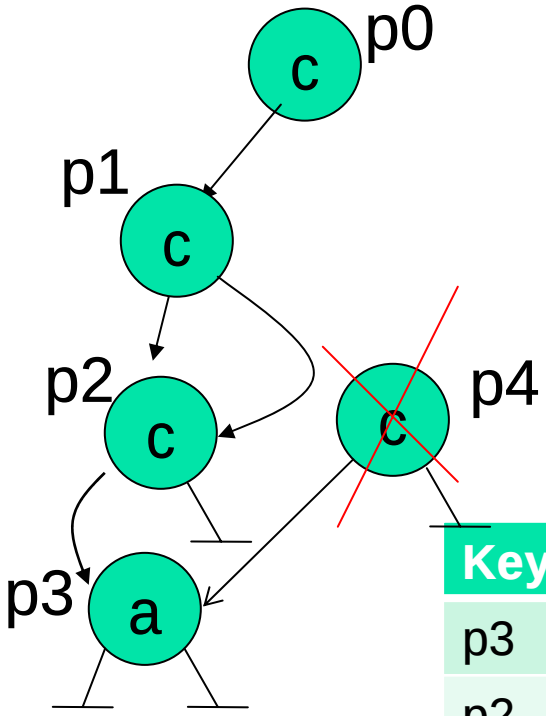
Collapse children of p0

- The right most child: p4
- htbl.get(p4)
  + p2 vs. p4 :
    'c'(p3,_) vs. 'c'(p3,_)
    => p2

Binary DAG Tree

```
<c>
...
 <c>
  <a>
  </a>
 </c>
</c>
```

stack



Key	Value
p3	p3
p2	p2

Collapse children of p0

- Replace p1.right by p2