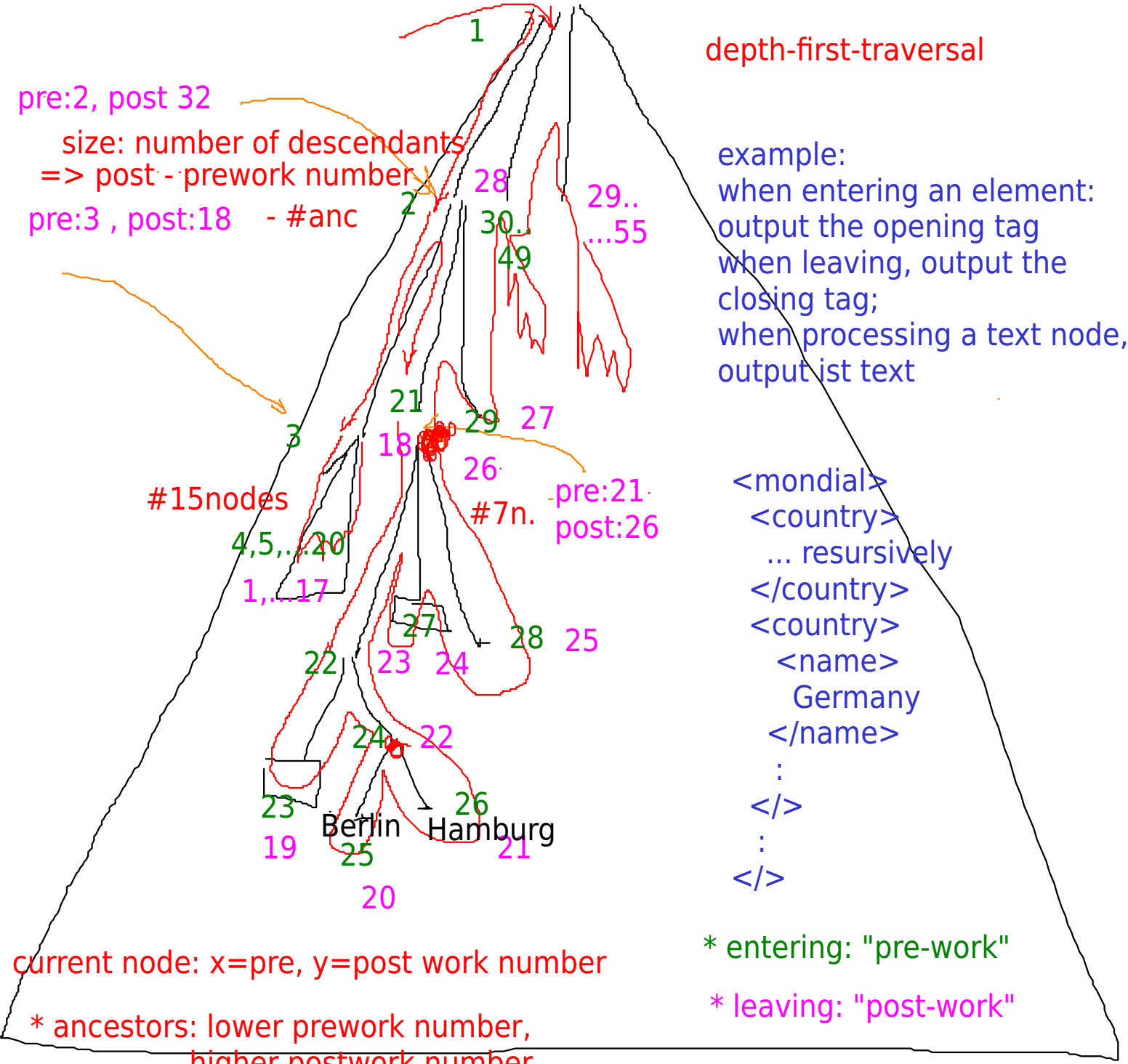
1
depth-first-traversal
pre:2, post 32
size: number of descendants
=> post - prework number
pre:3 , post:18 - #anc
2
28
29..
...55
30..
49
21
3
29
27
18
26
#15nodes
pre:21
post:26
#7n.
4,5,..,20
1,..,17
27
28
25
22
23
24
24
22
23
26
Berlin
Hamburg
19
25
21
20
example:
when entering an element:
output the opening tag
when leaving, output the
closing tag;
when processing a text node,
output ist text
<mondial>
<country>
... resursively
</country>
<country>
<name>
Germany
</name>
:
</>
:
</>
current node: x=pre, y=post work number
* entering: "pre-work"
* leaving: "post-work"
* ancestors: lower prework number,
higher postwork number
* preceding: lower prework number, lower postwork number
* following: higher prework number, higher postwork number
* descendants: higher prework number, lower postwork number

sketch of XML storage (in a relational DB)

nodes

nodeId=pre	post	(or text) elementname	parentid	
21	29	bla	2	
24	22	blubb	22	(first child of 21)
25	20	'Berlin'	24	
26	21	'Hamburg'	24	

consider a query //bla/blubb/text()[contains(.,'burg')]

```
select t.elementname
from nodes e1, nodes e2, nodes t
 where e1.name = 'bla'
   and e2.pre > e1.pre and e2.post<e1.post
   and e2.elementname = 'blubb'
   and t.parent = e2.pre
   and t.elementname like '%burg%'
order by pre
```

=> output in document order

into the storage table

burg

evaluation strategy;
consider there is a fulltext (inverted) index with (frequent) substrings