Introduction to Wavelet Trees

Giovanni Manzini

rank and select operations

Given a string S[1,n] over and alphabet A we define:

$rank_S(c,i)$ = # occs of symbol c in S[1,i]

$select_S(c,i)$ = position of the i-th c in S[1,n]

Example:

S = abracadabra

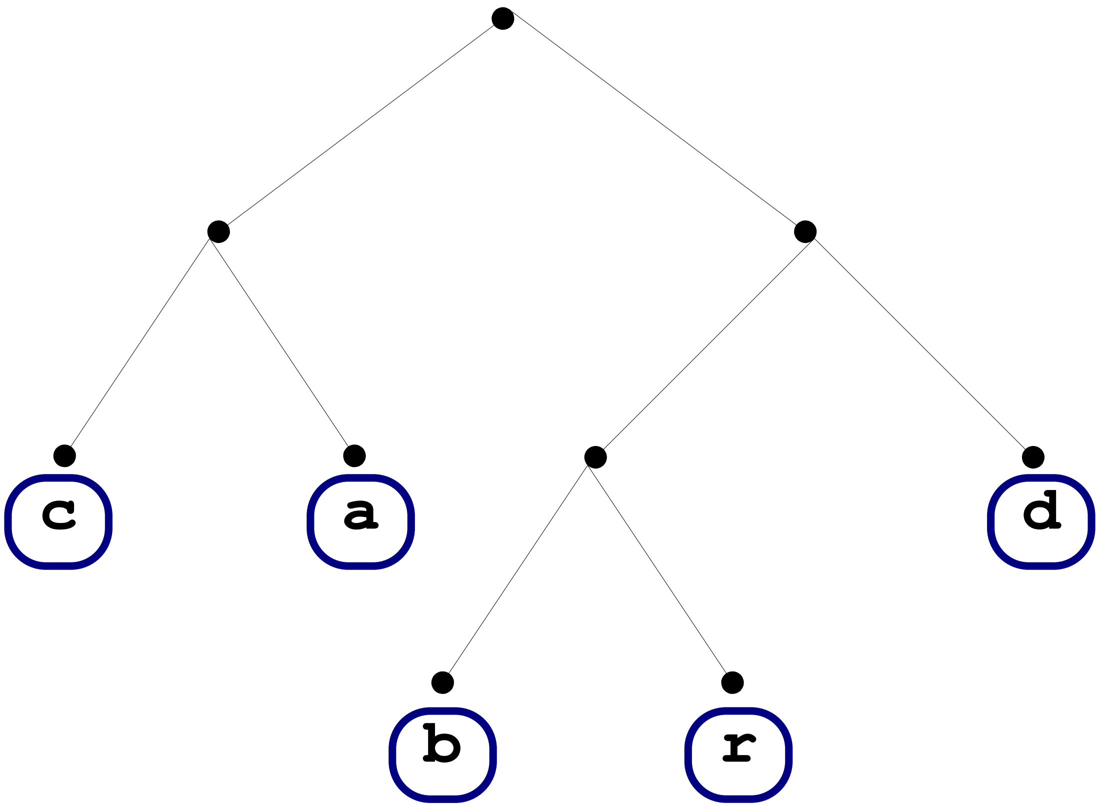
$rank_S(a,3)=1$, $select_S(a,3)=6$

The Wavelet Tree data structure

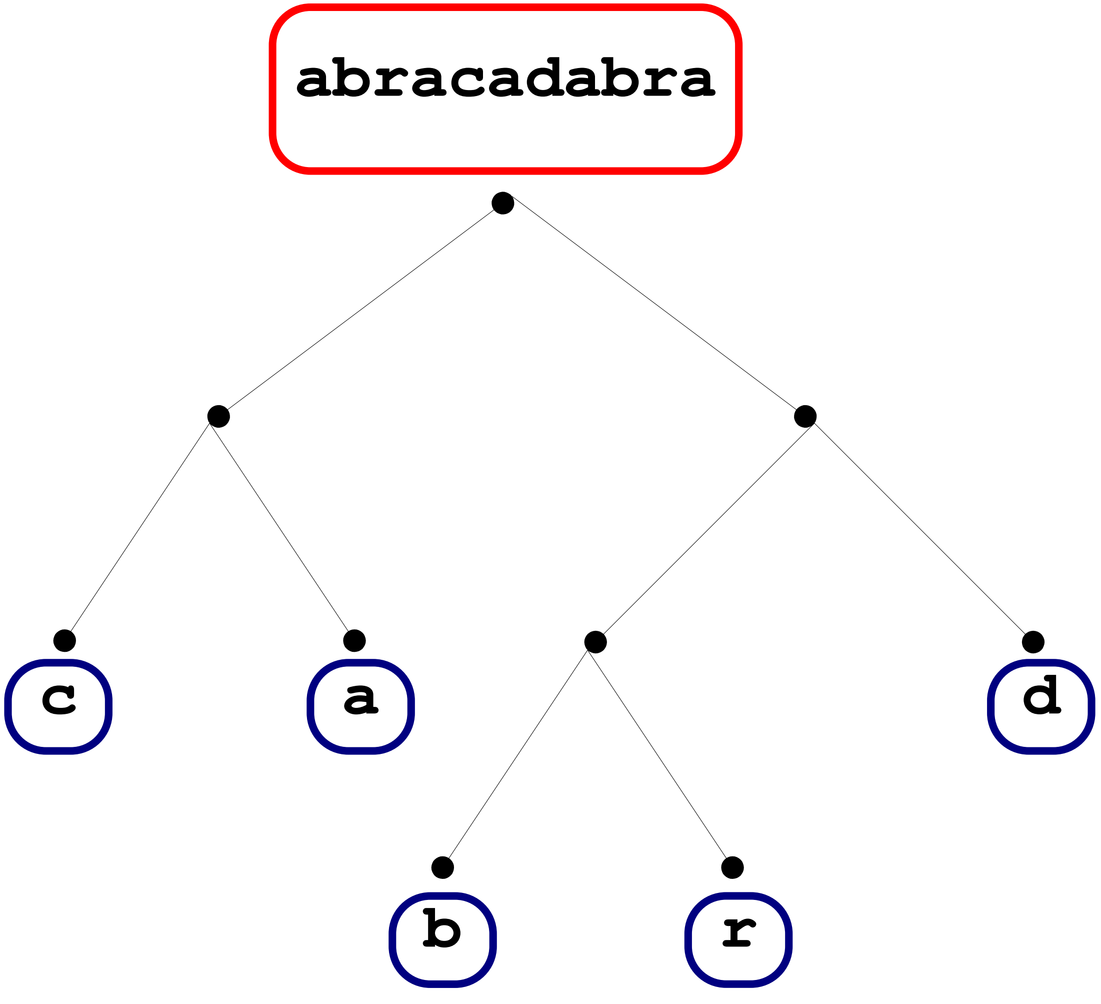
Wavelet Trees have been introduced to represent compactly a string supporting rank/select operations.

They work assuming we can do rank/select on binary strings, To build a Wavelet Tree we start with a complete binary tree with a leaf for each alphabet symbol.

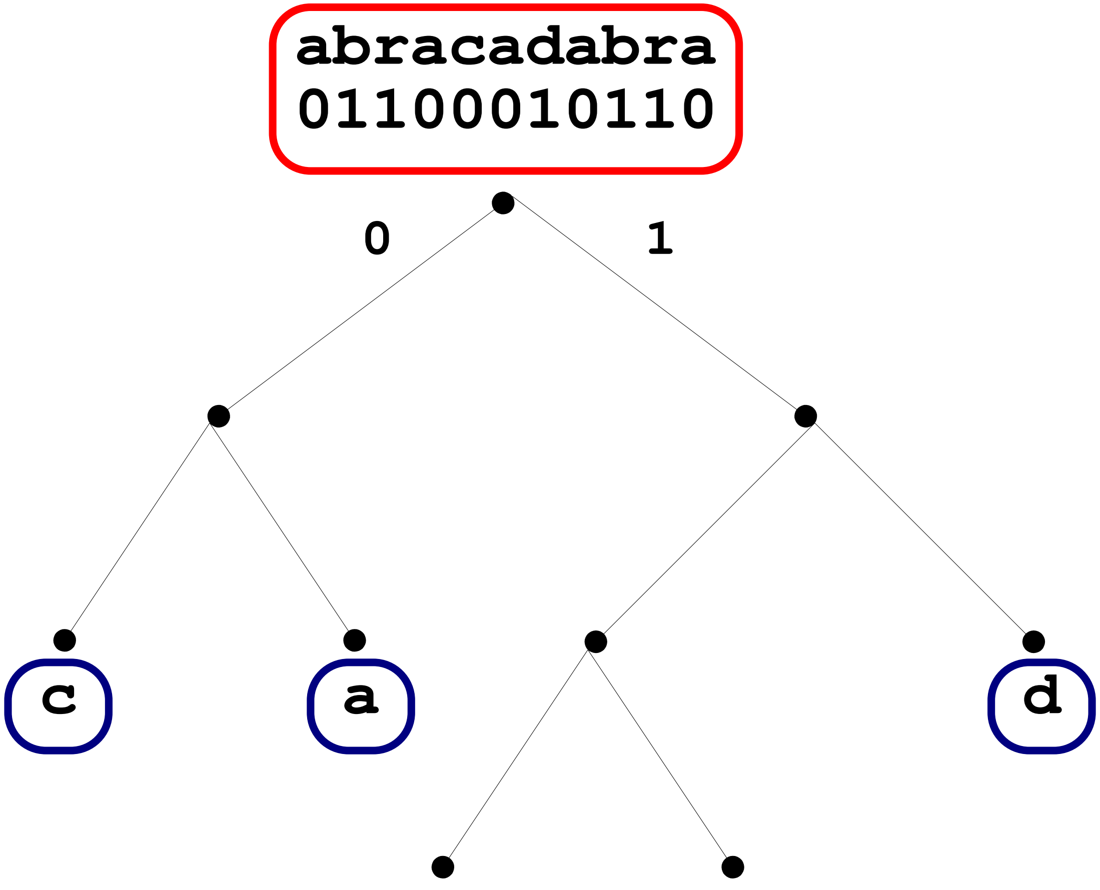
Example. $A = \{a, b, c, d, r\}$



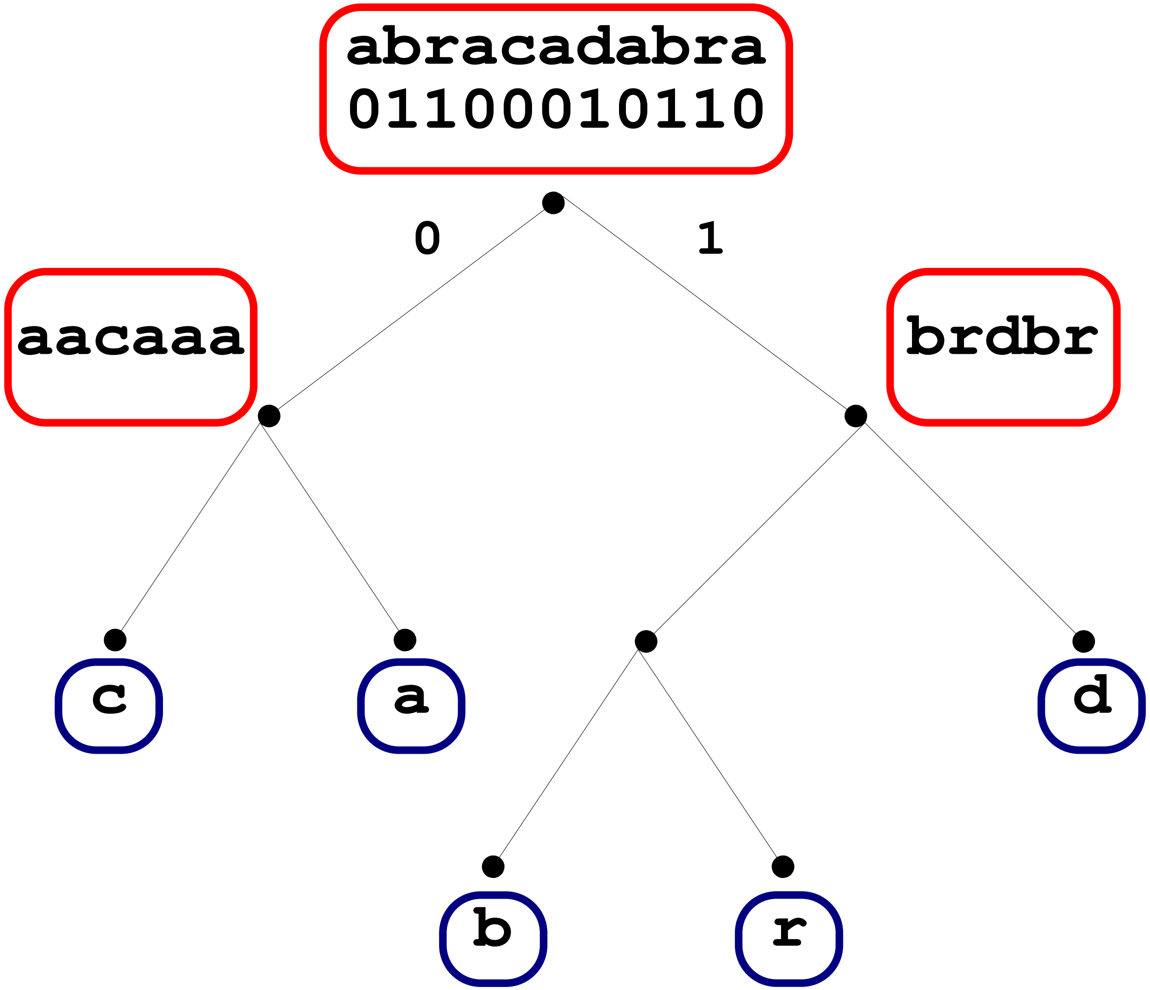
Given the string **abracadabra** we build the corresponding Wavelet Tree as follows:



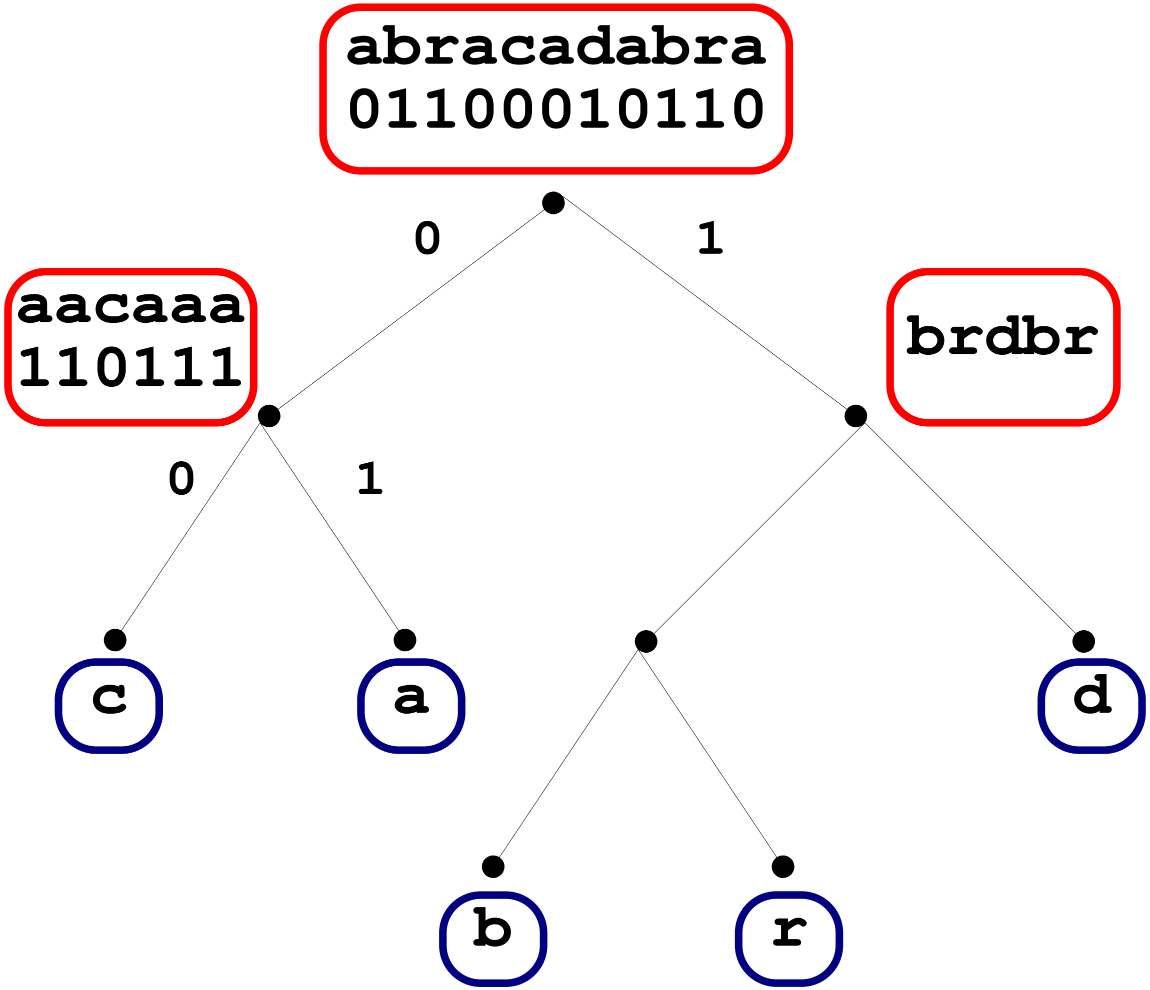
Given the string **abracadabra** we build the corresponding Wavelet Tree as follows:



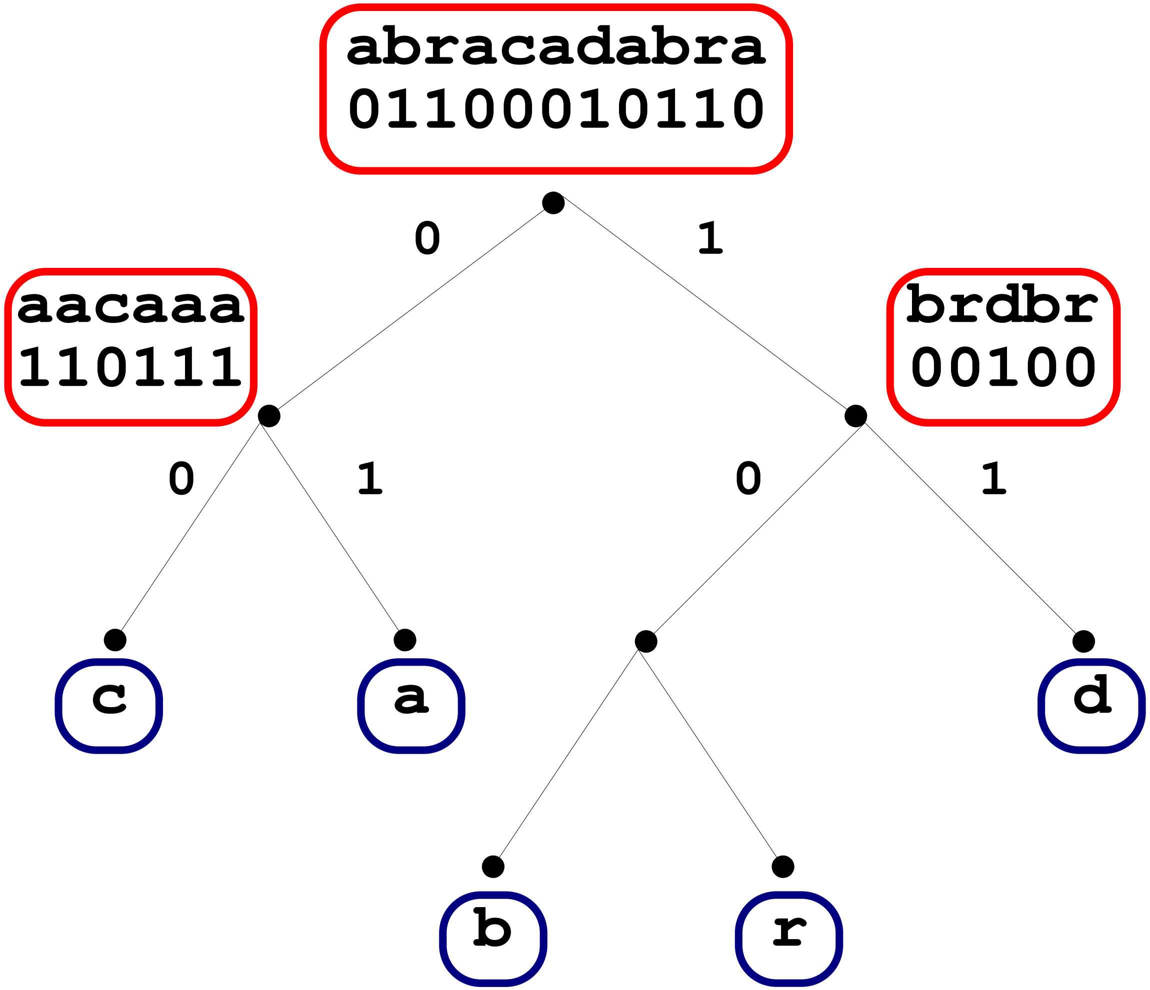
Given the string **abracadabra** we build the corresponding Wavelet Tree as follows:



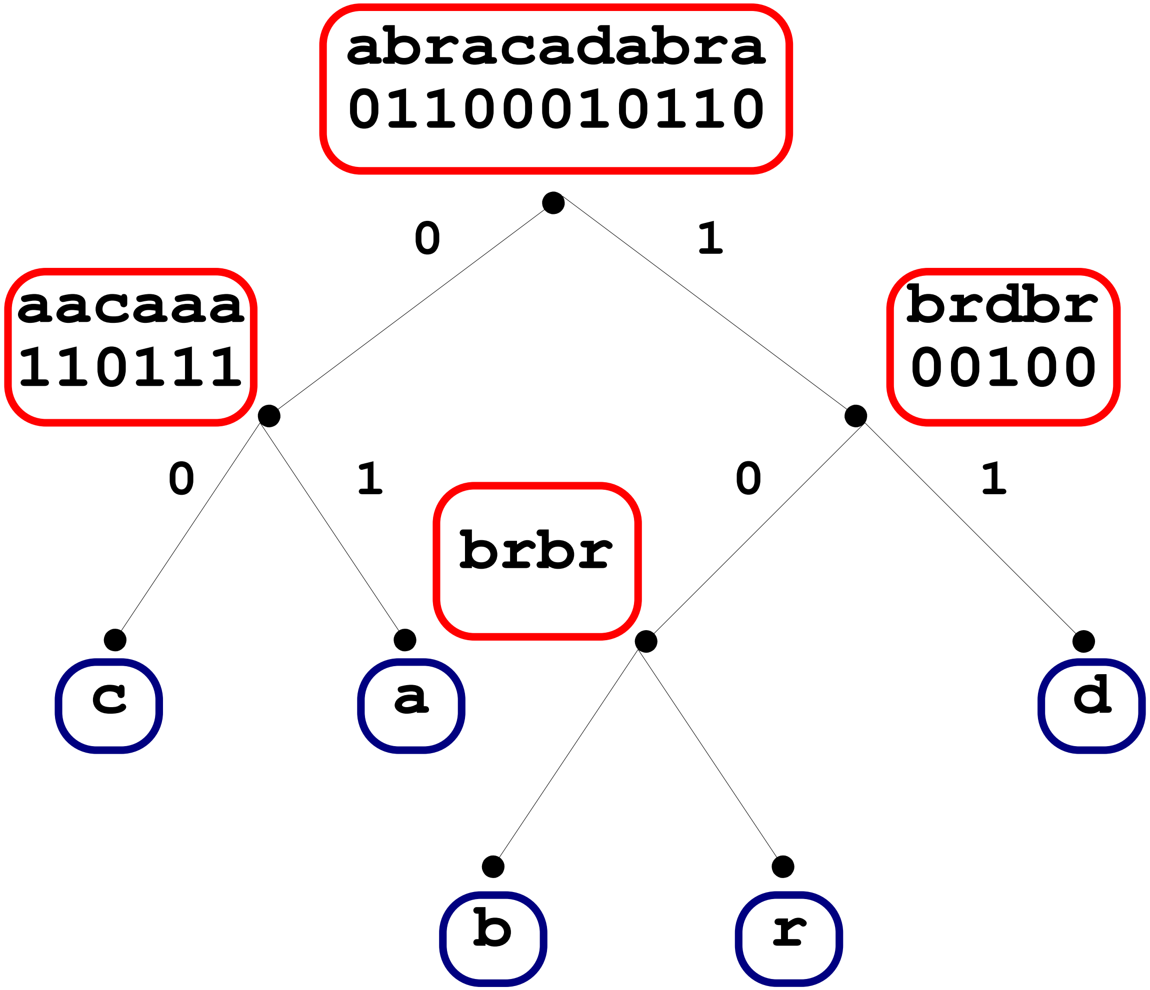
Given the string **abracadabra** we build the corresponding Wavelet Tree as follows:



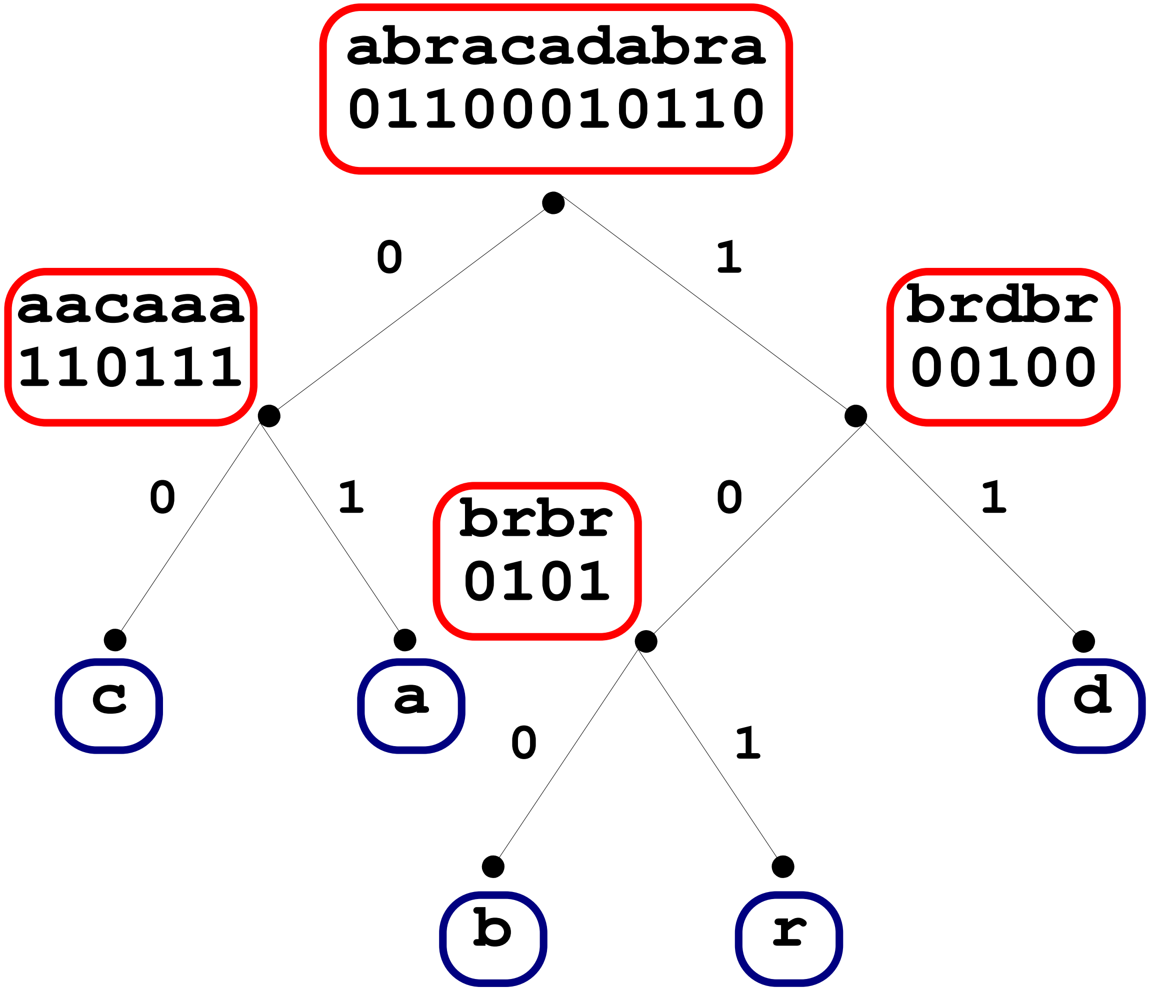
Given the string **abracadabra** we build the corresponding Wavelet Tree as follows:



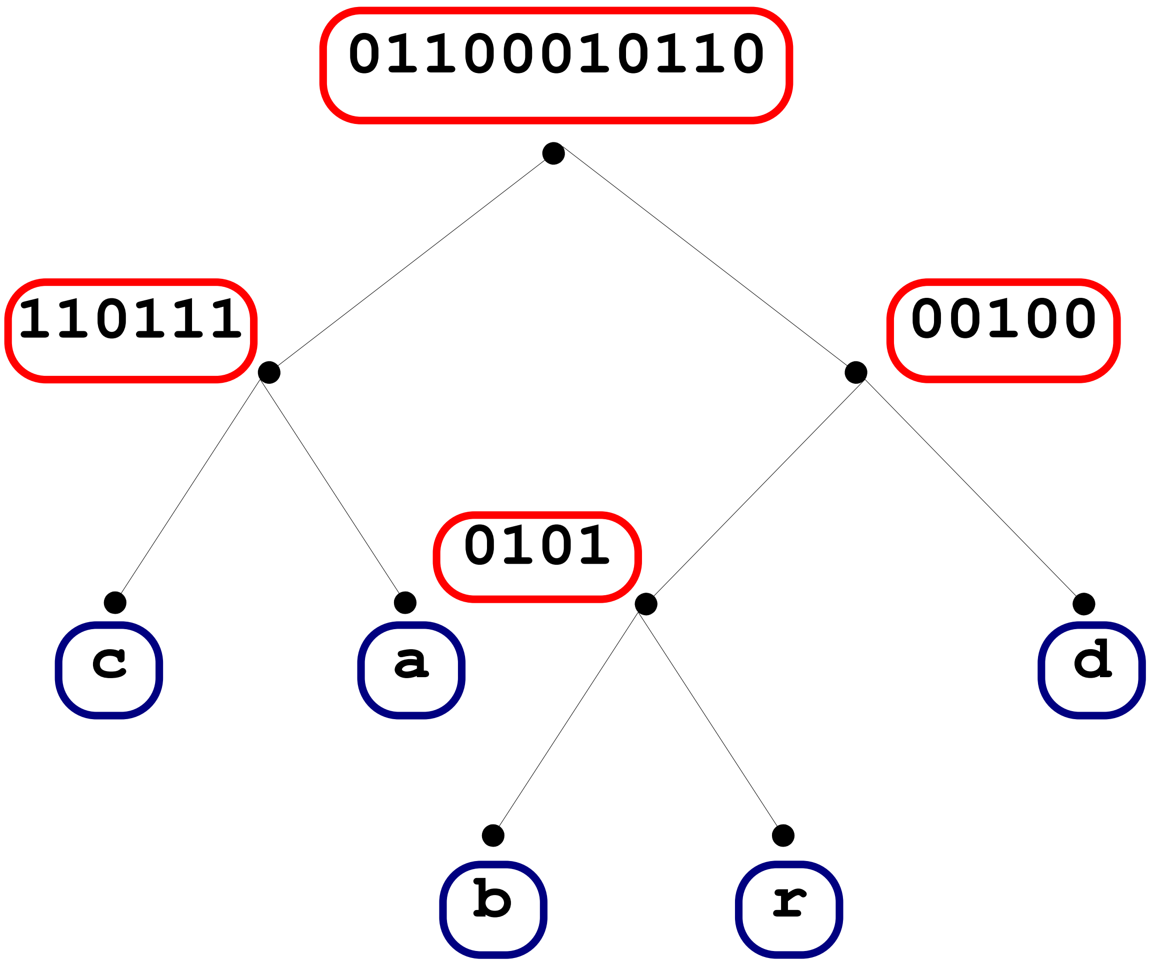
Given the string **abracadabra** we build the corresponding Wavelet Tree as follows:



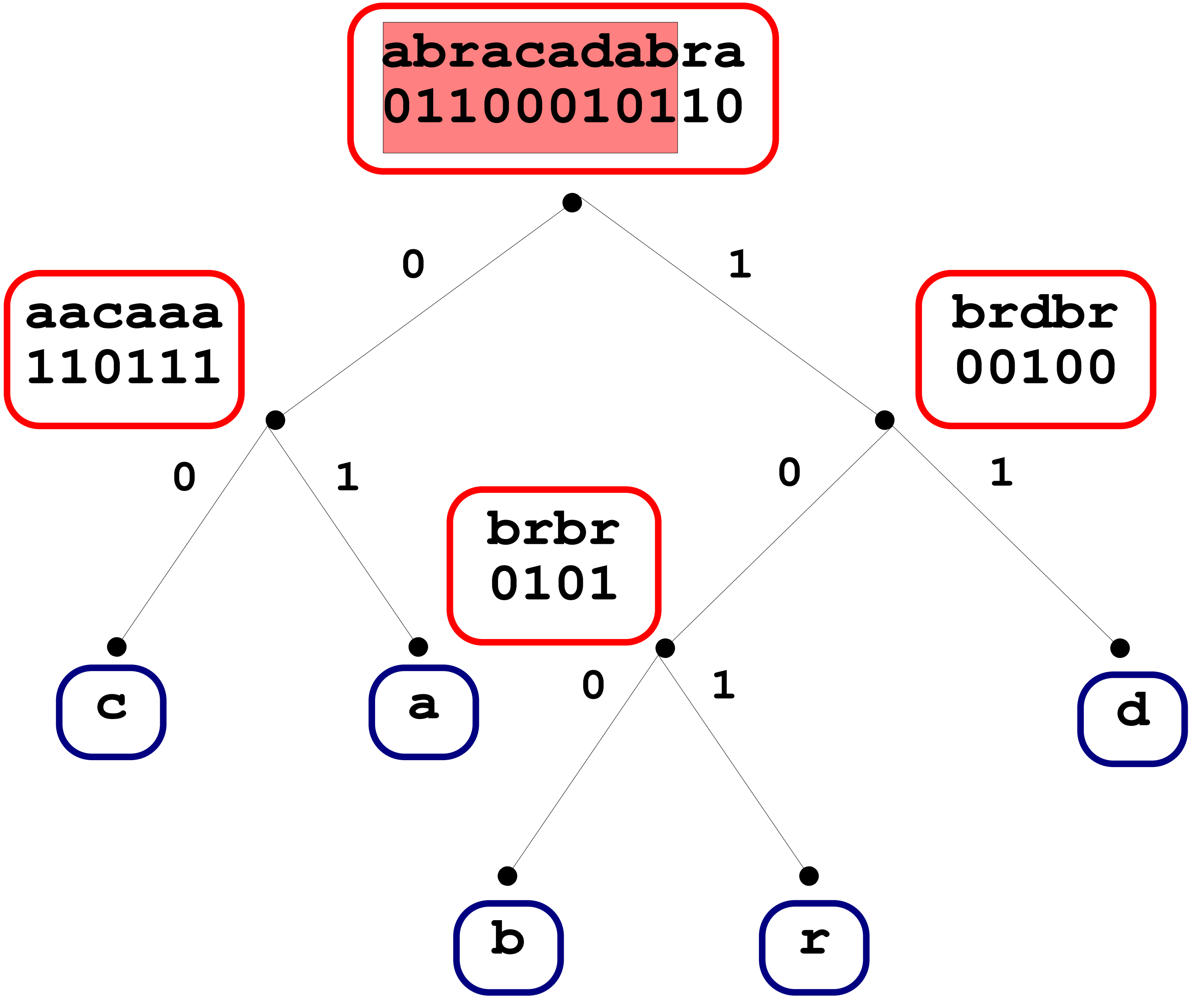
Given the string **abracadabra** we build the corresponding Wavelet Tree as follows:



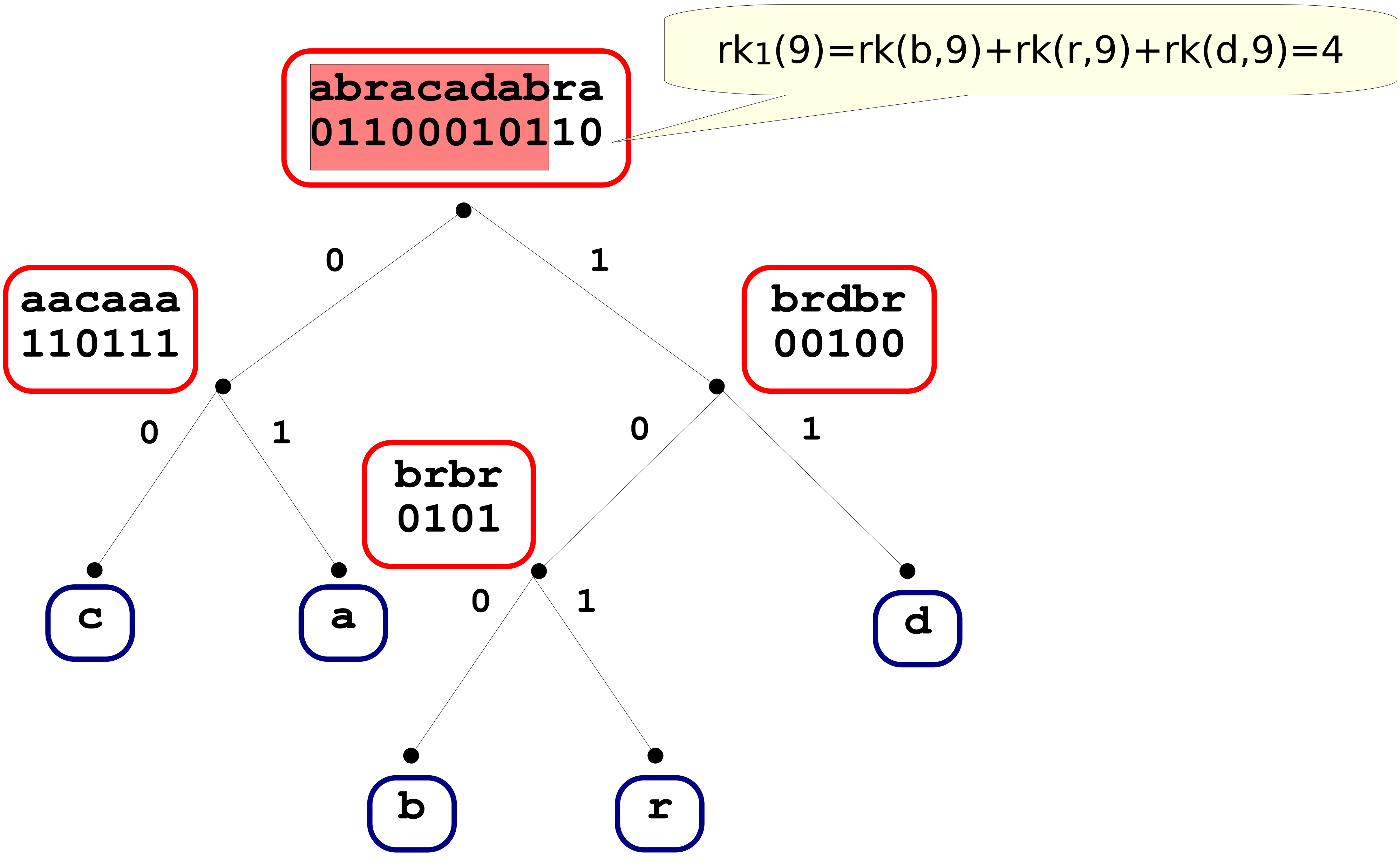
rank/select queries over the original string can be answered via rank/select queries over the binary strings associated to internal nodes.



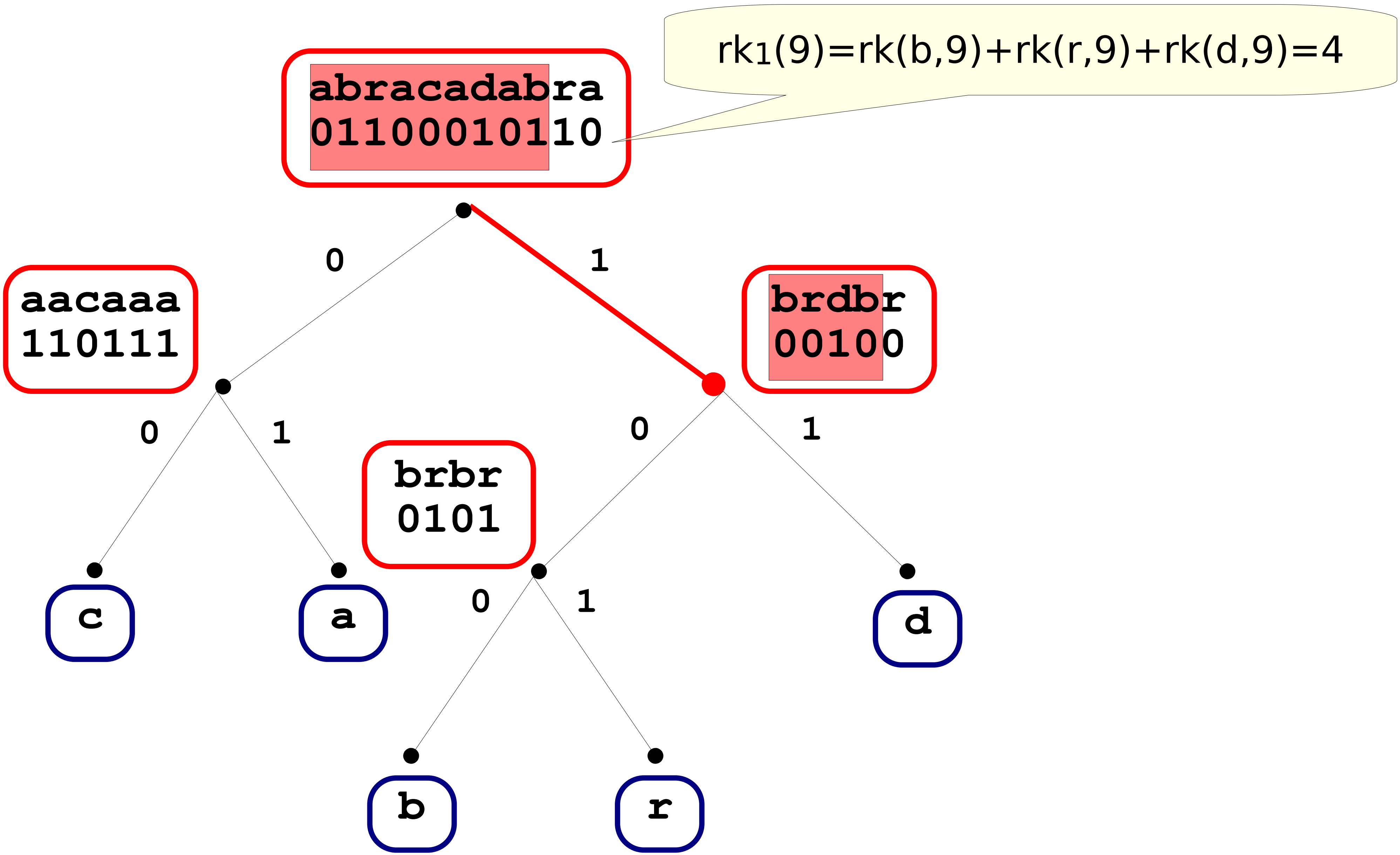
Suppose we want to compute rank(b,9)



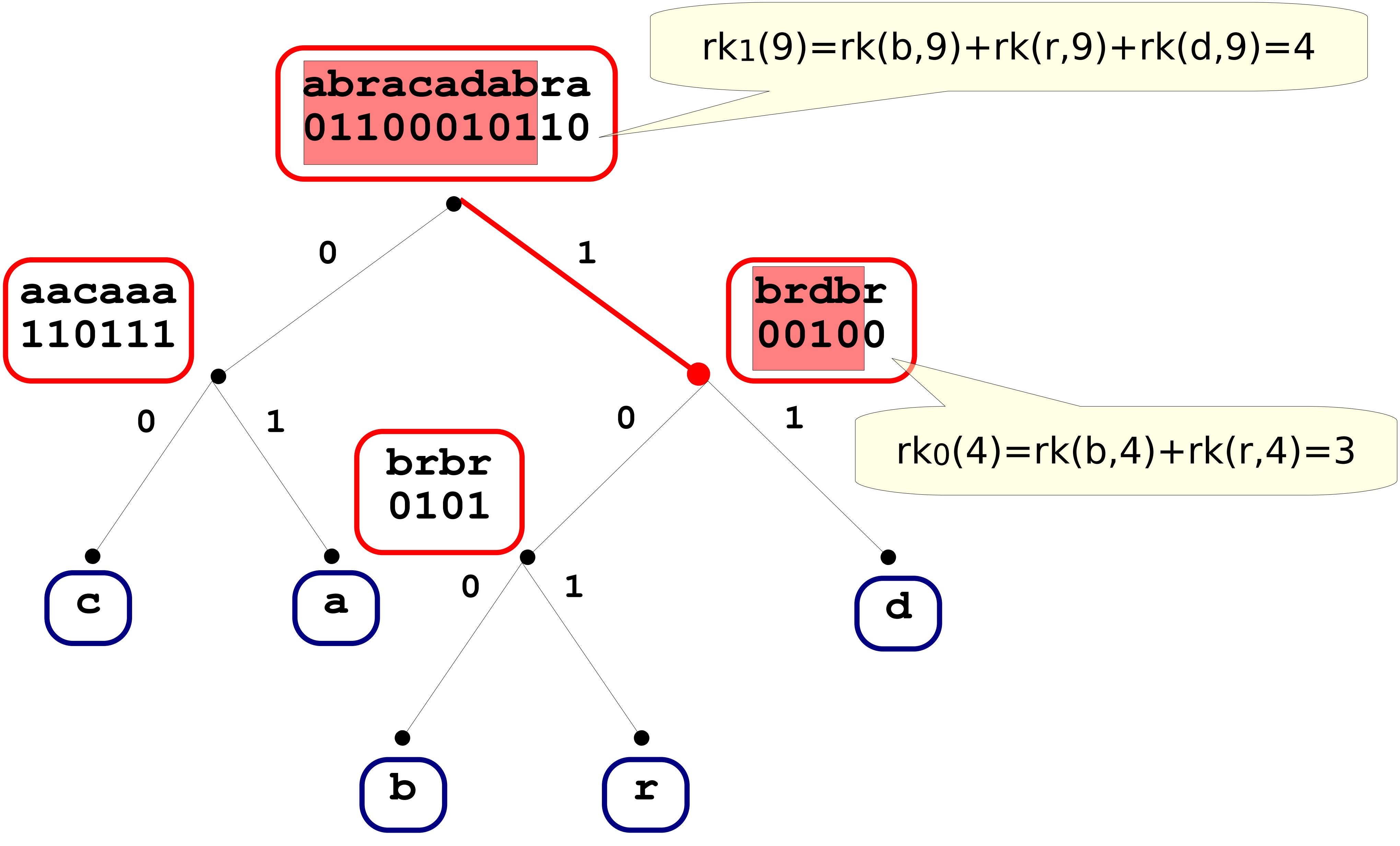
Suppose we want to compute rank(b,9)



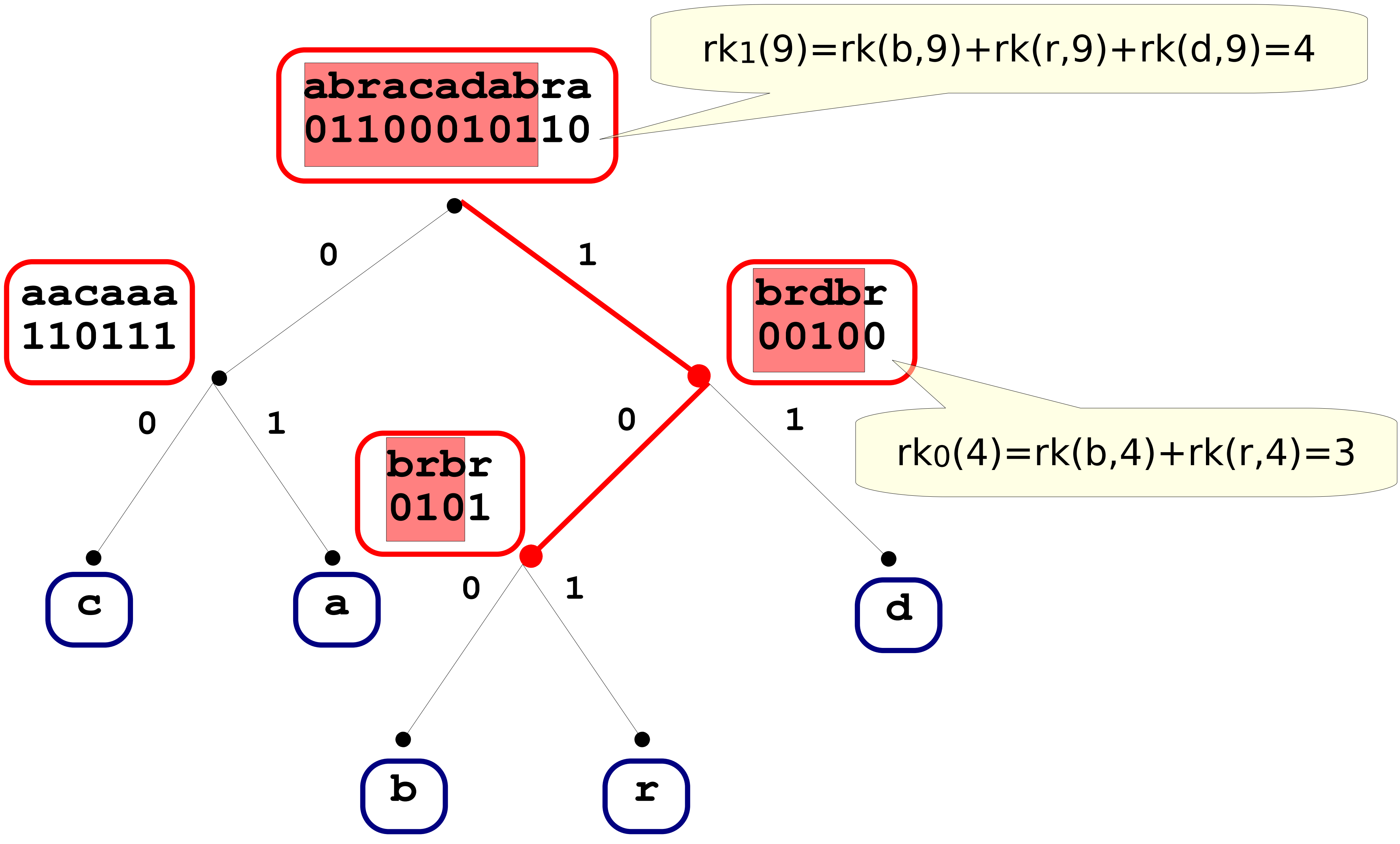
Suppose we want to compute rank(b,9)



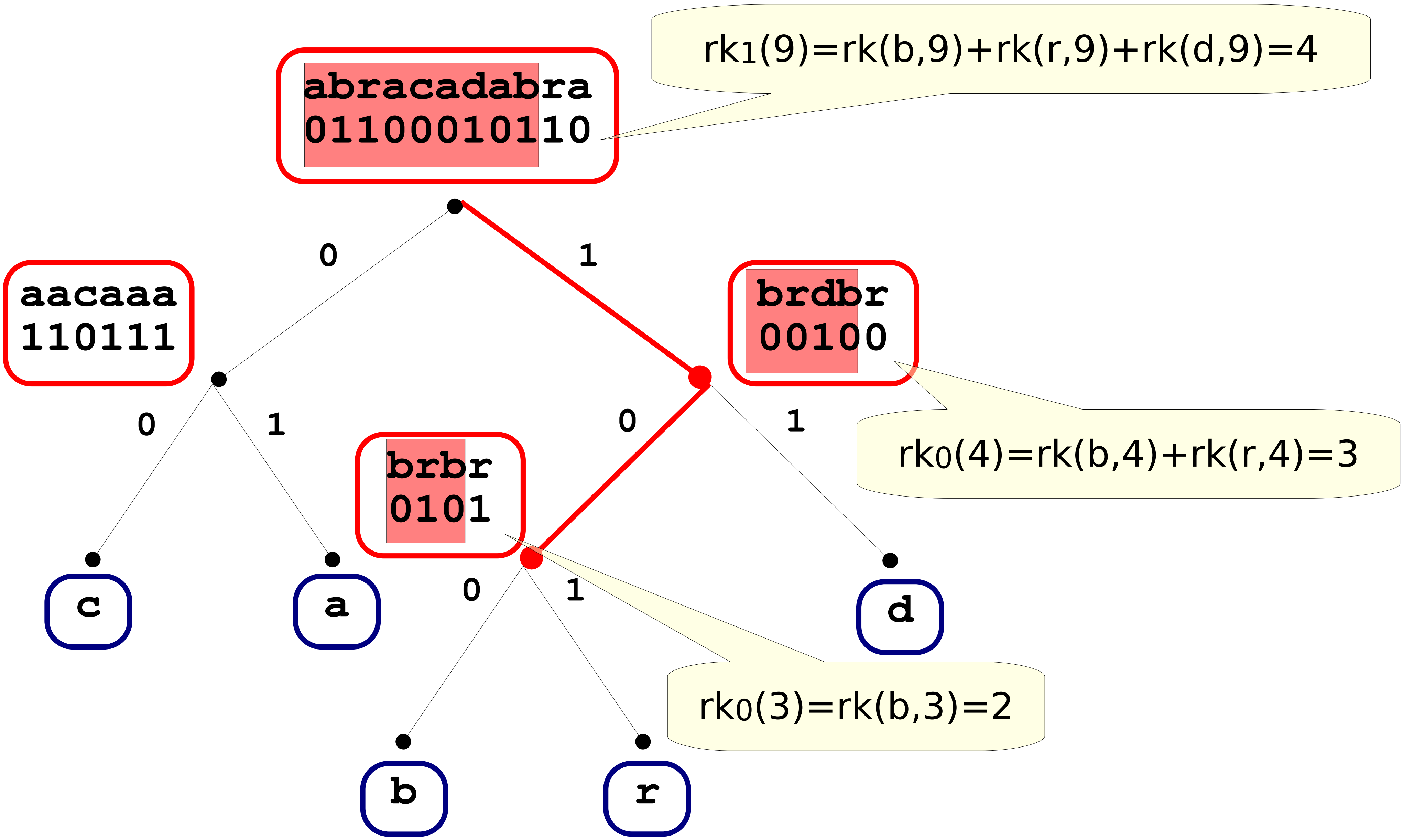
Suppose we want to compute rank(b,9)



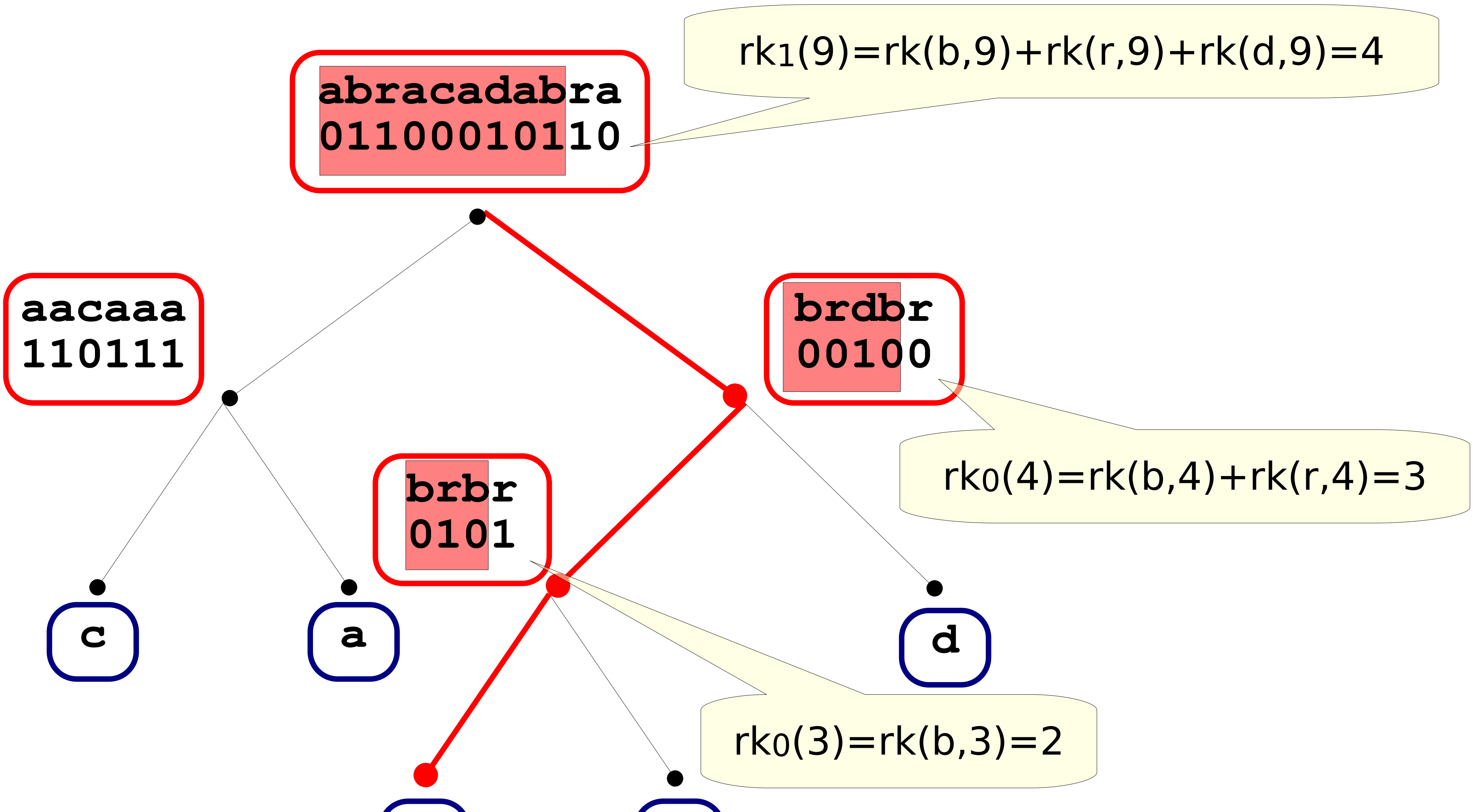
Suppose we want to compute rank(b,9)



Suppose we want to compute rank(b,9)

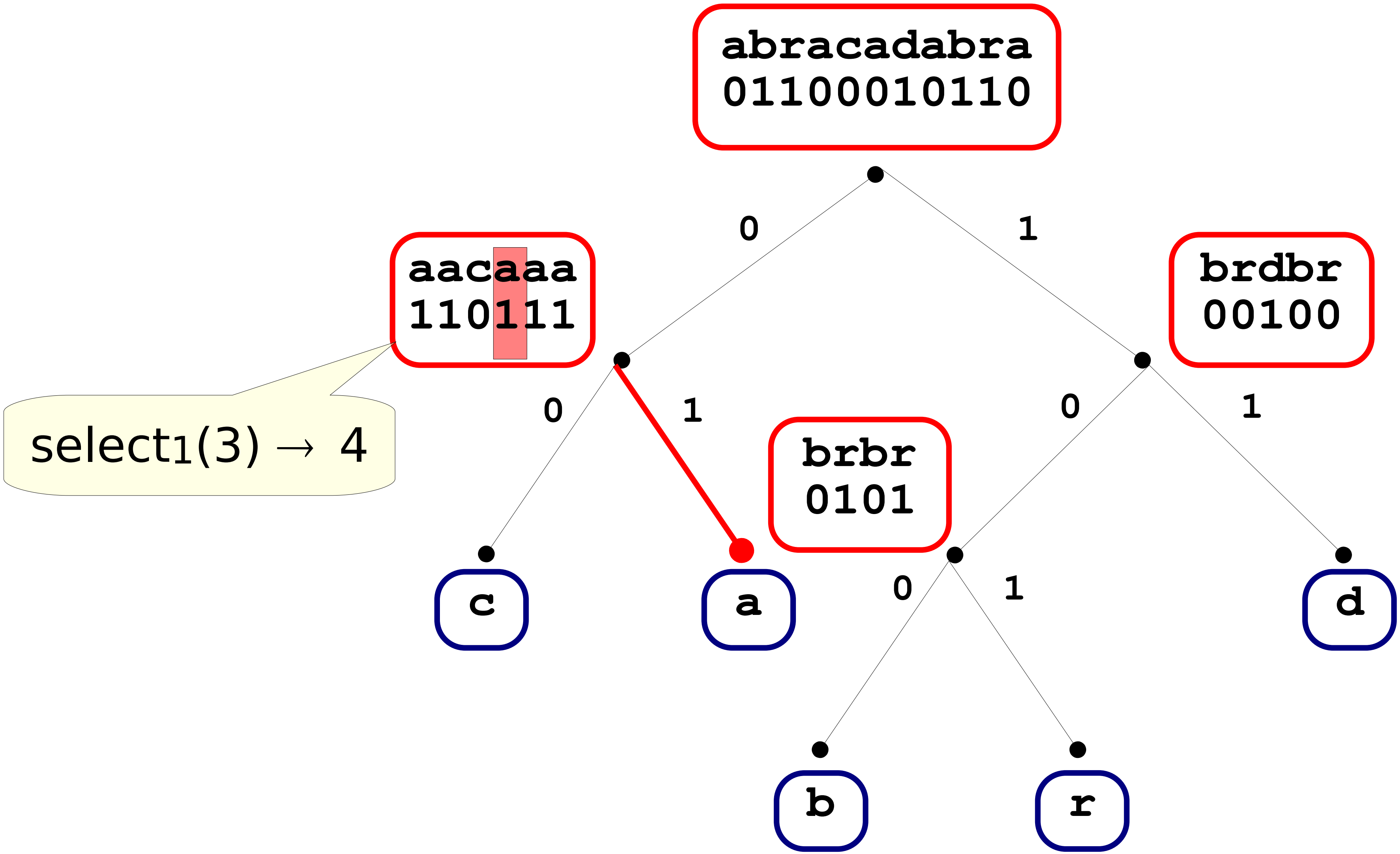


Suppose we want to compute rank(b,9)

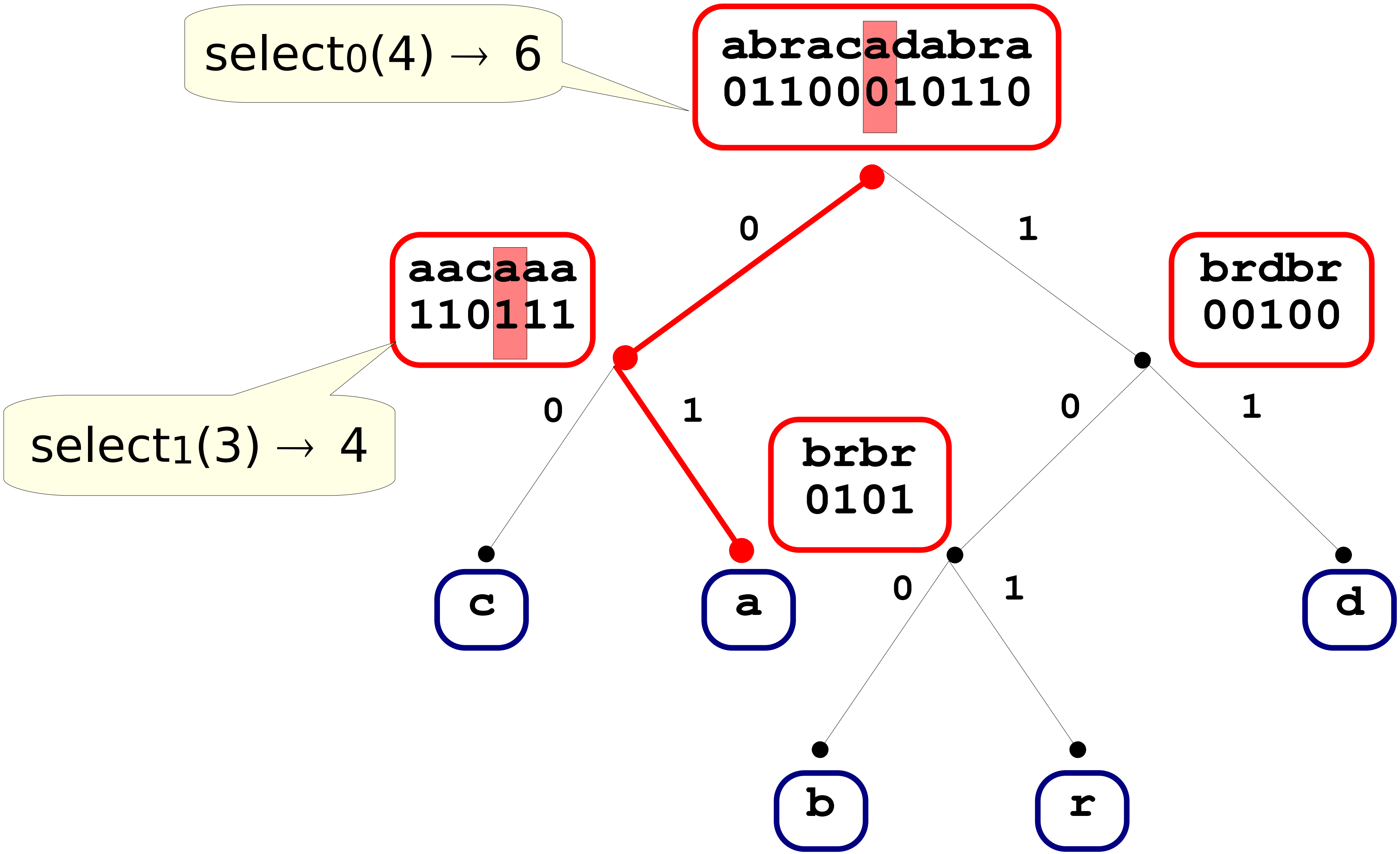


Answer: rank(b,9)=2

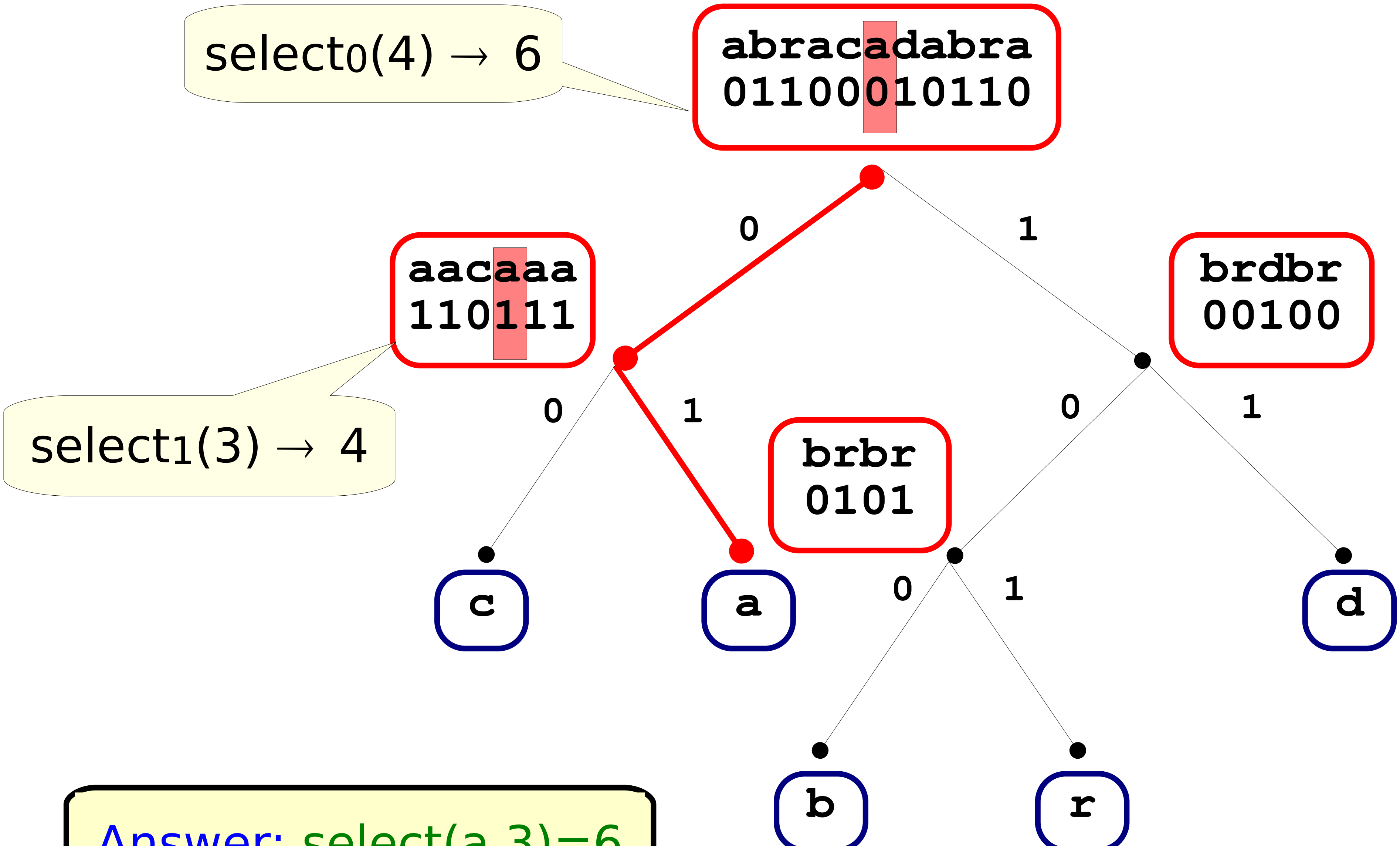
Select queries go upward. To compute select(a,3)



Select queries go upward. To compute select(a,3)



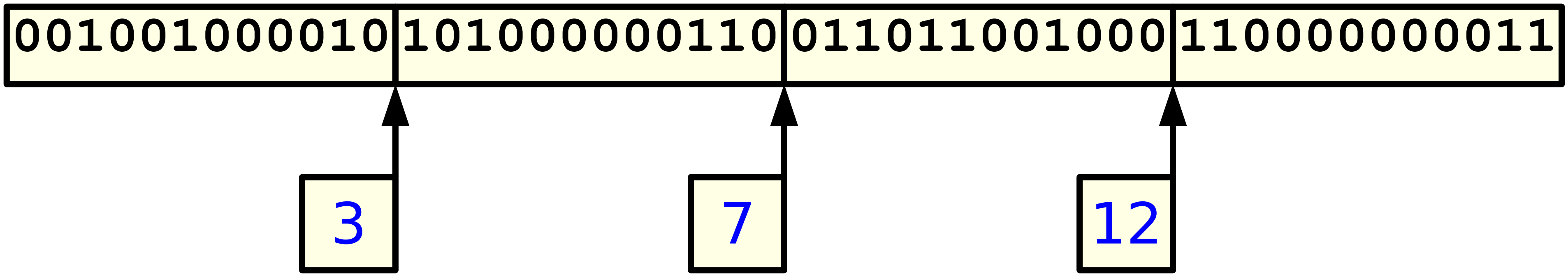
Select queries go upward. To compute select(a,3)



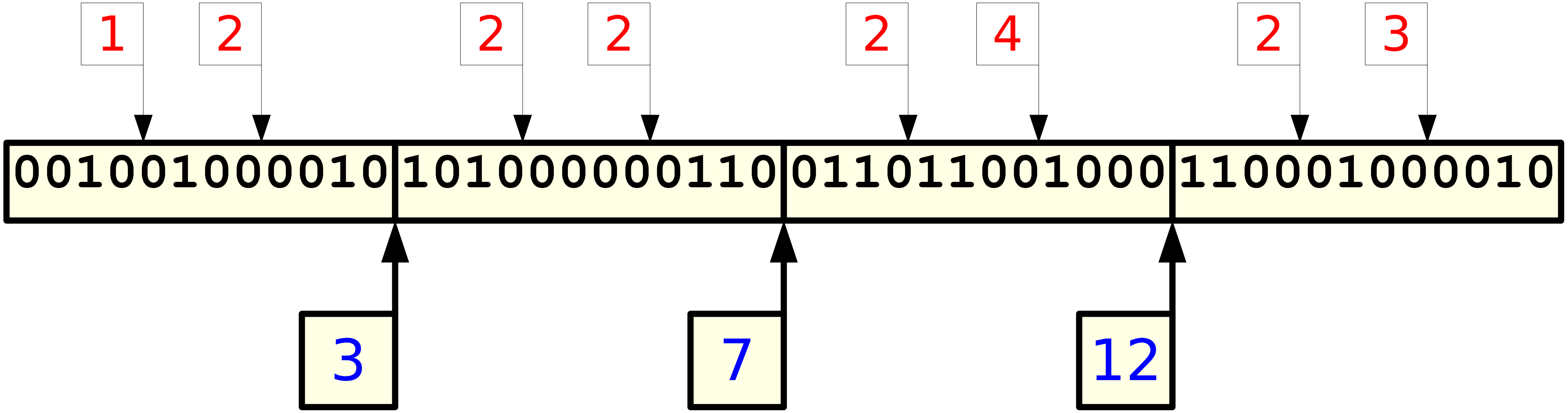
Answer: select(a,3)=6

Rank$_1$ queries on binary strings

Simplest idea: split the binary string into blocks and store a partial sum for each block:

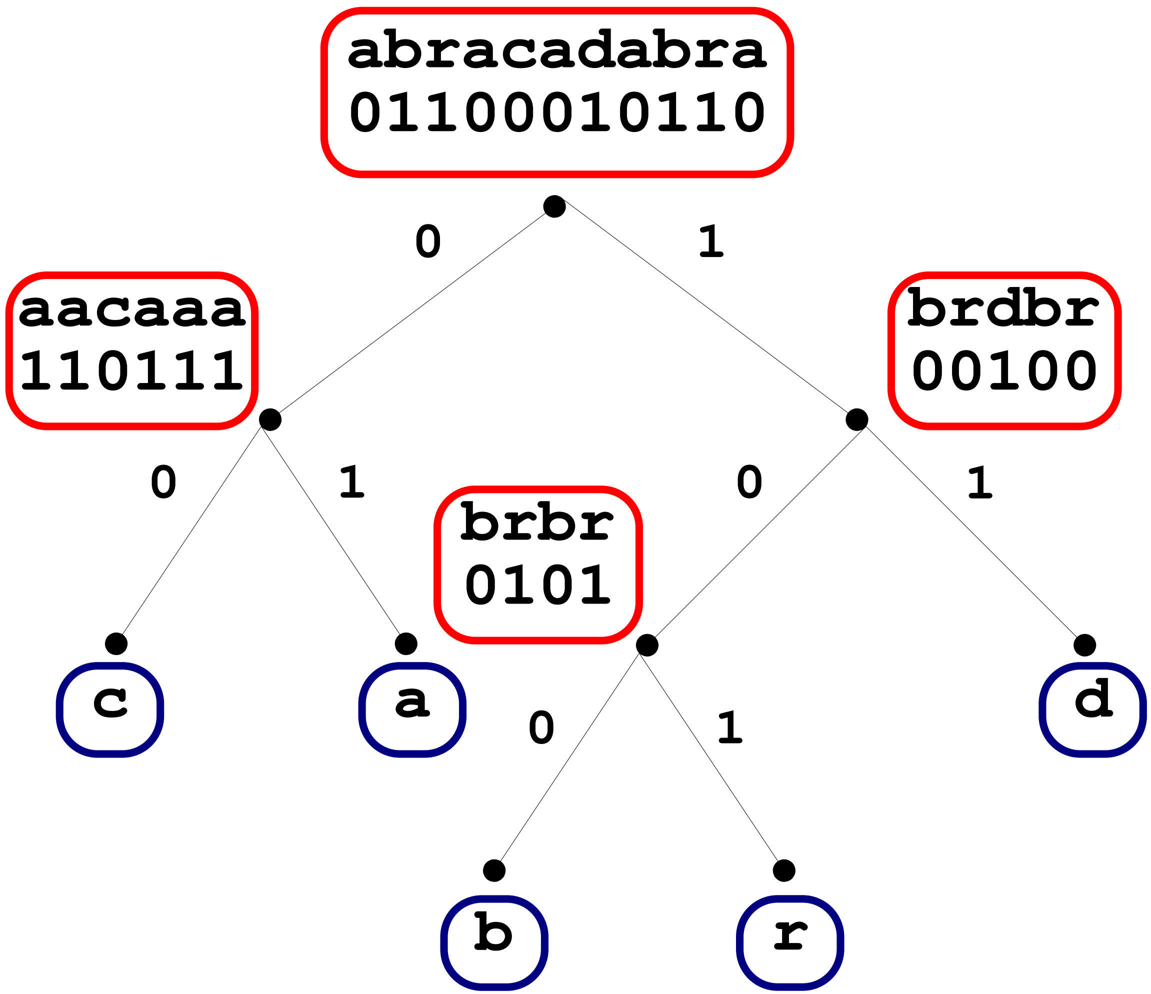


We can refine the idea using blocks and mini-blocks:



Similar techniques for select queries!

Wavelet Trees as scrambled prefix codes



We are implicitly using the encoding:

c*→*00* a→*01* *b*→*100* *r*→*101* *d*→*11

Wavelet Trees as scrambled prefix codes:

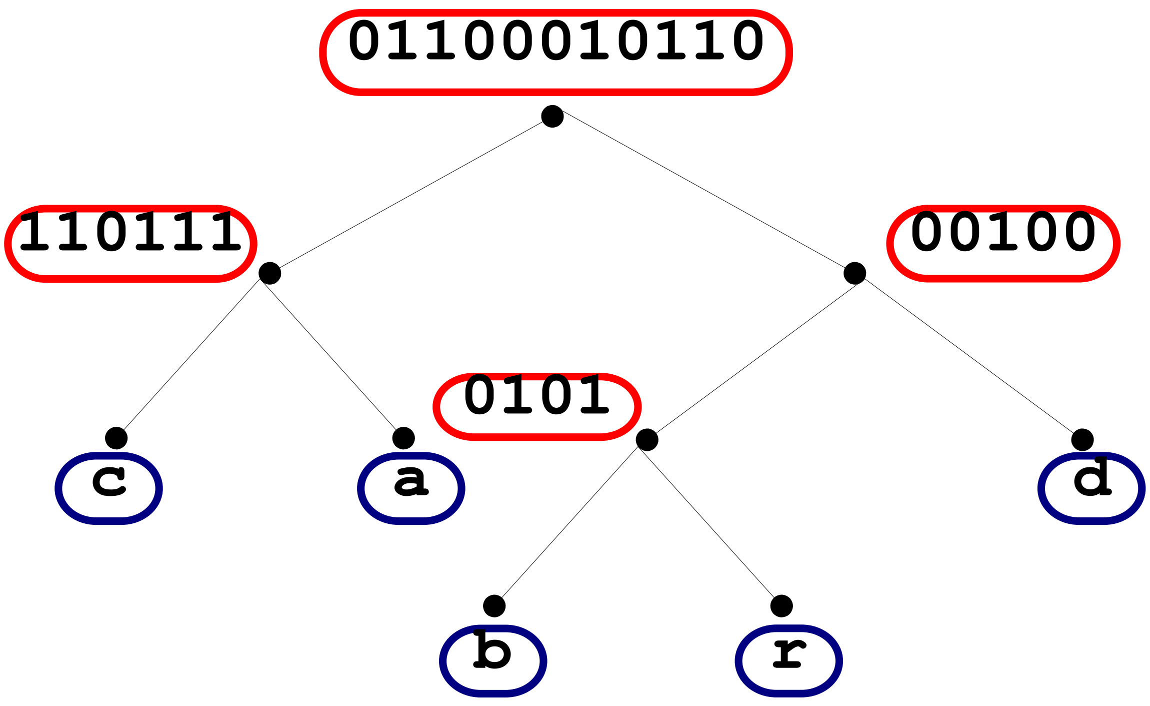
We are implicitly using the encoding:

a→01 c→00 b→100 r→101 d→11

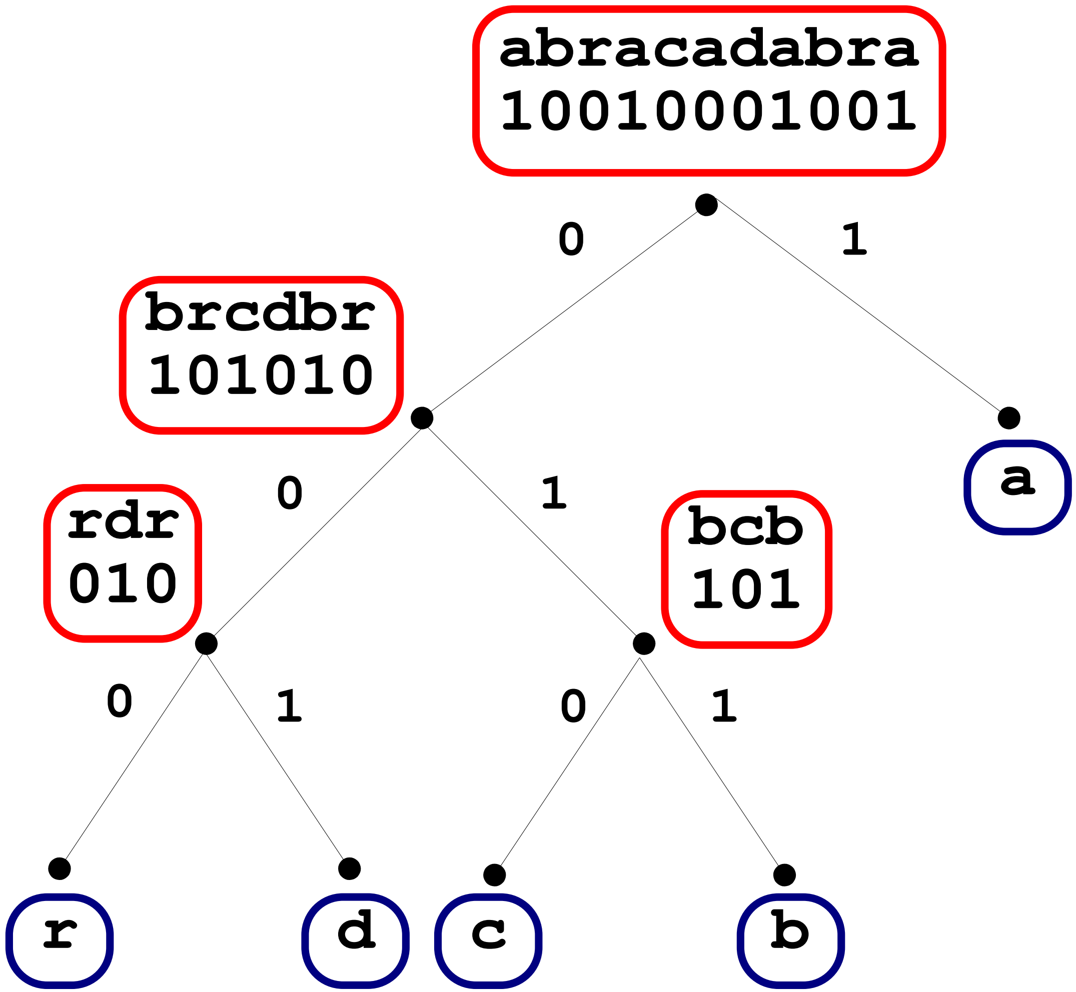
Traditional prefix codes:

abracadabra → 01 100 101 01 00 01 11 . . .

Wavelets:



Changing the shape of the the Wavelet Tree, all properties mentioned above still hold with a different prefix code.



r→*000*
d→*001*
c→*010*
b→*011*
a→*1*

We can choose any prefix-free binary encoding of the characters and build the corresponding Wavelet Tree.

If we choose Huffman codes we have compression for free and direct access to arbitrary positions in the text.

Summing up

Binary strings operations + Wavelet Trees +

= Compression and efficient rank support +

BWT & backward search = Compressed full text index

Other Wavelet Trees virtues

Simple solutions for range queries on an integer sequence S[1,n]

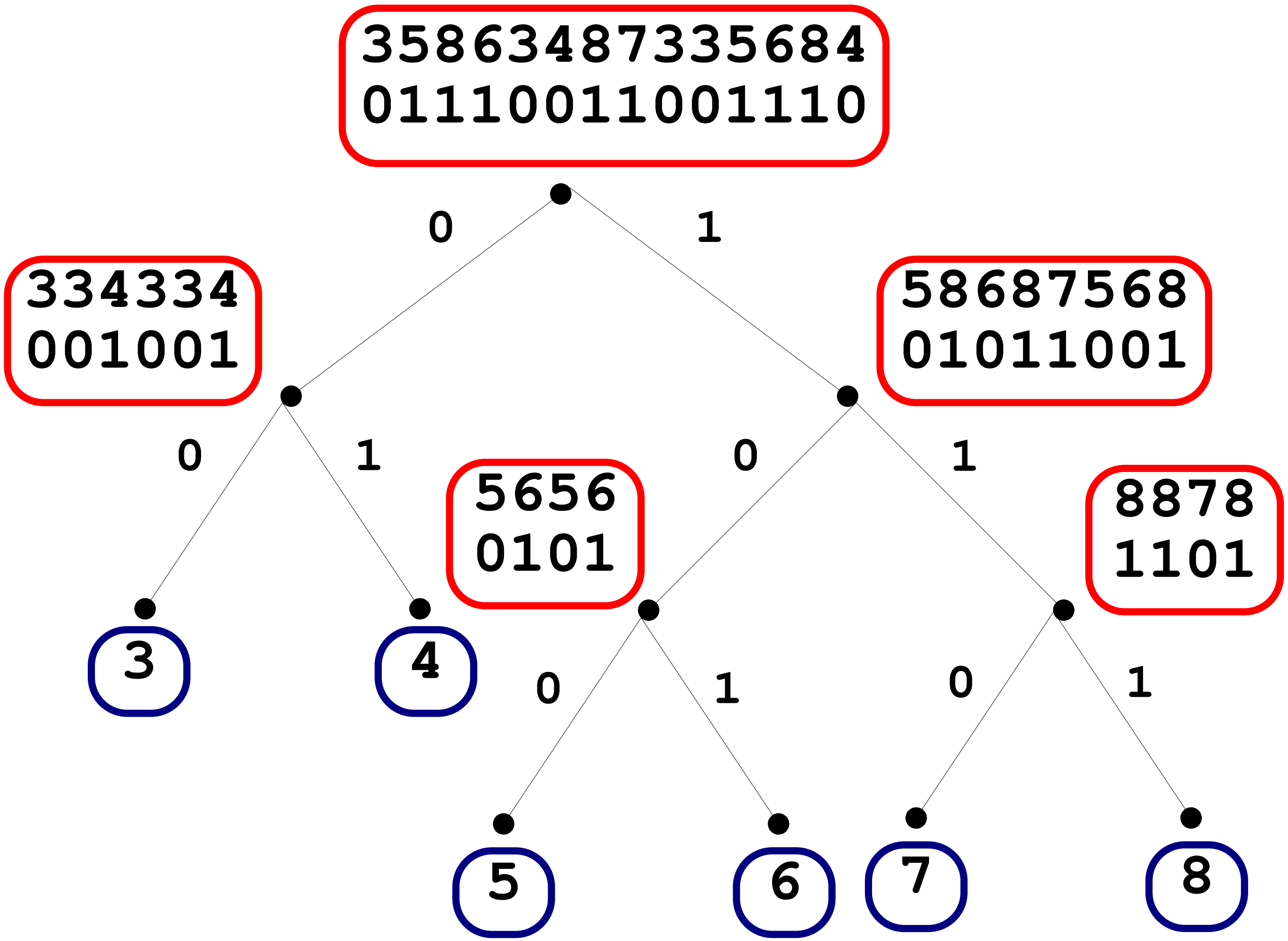
range_quantile(S,i,j,k): return the k-th smallest value in S[i···j]

range_next(S,i,j,x): return the smallest value greater than x in S[i···j]

range_count(S,i,j): return # of distinct values in S[i···j]

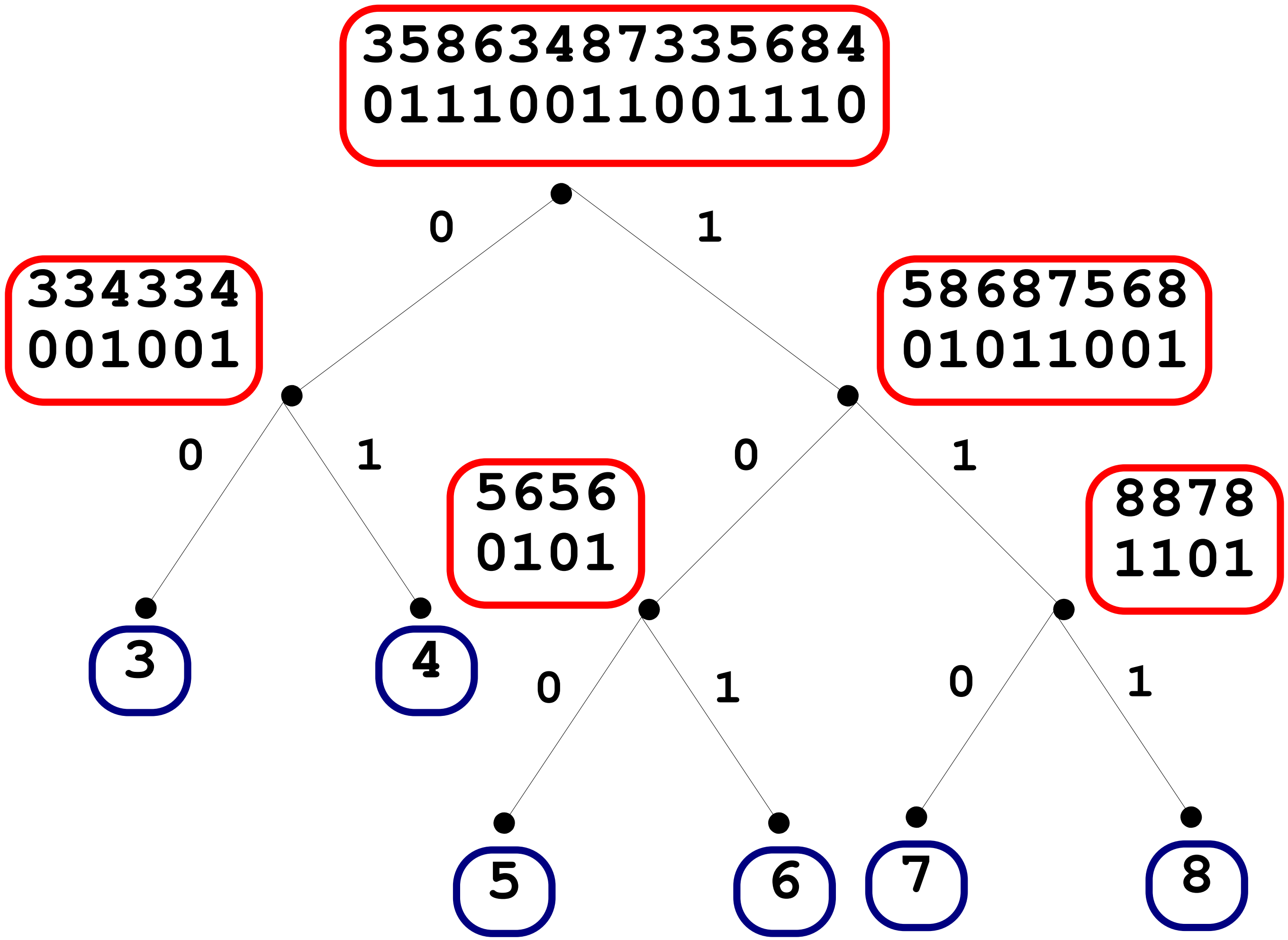
Example.

S = **[3,5,8,6,3,4,8,7,3,3,5,6,8,4]**



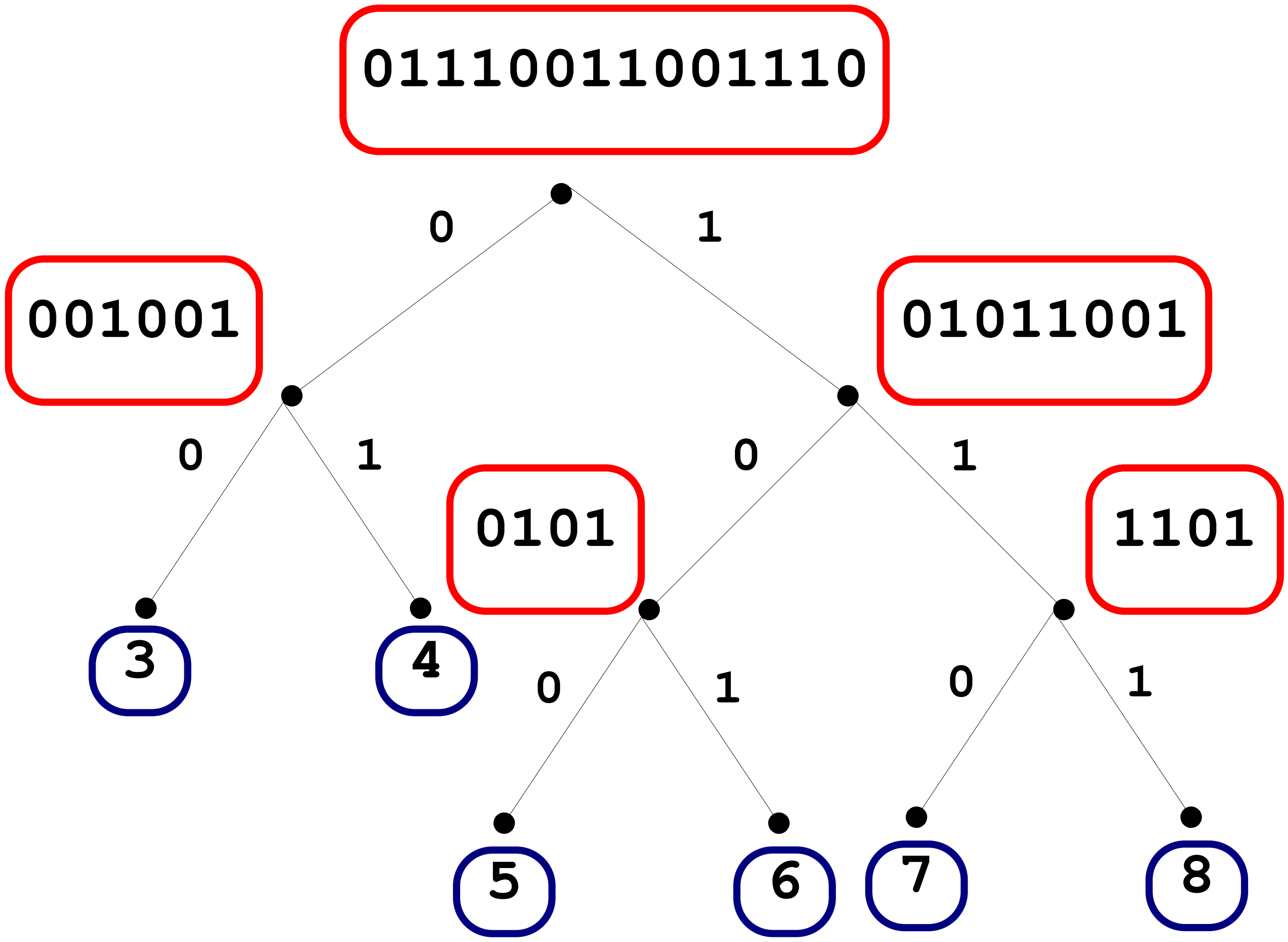
range_quantile(S,2,7,3): 3-rd smallest value in S[2⋯7]

S = [3,5,8,6,3,4,8,7,3,3,5,6,8,4]



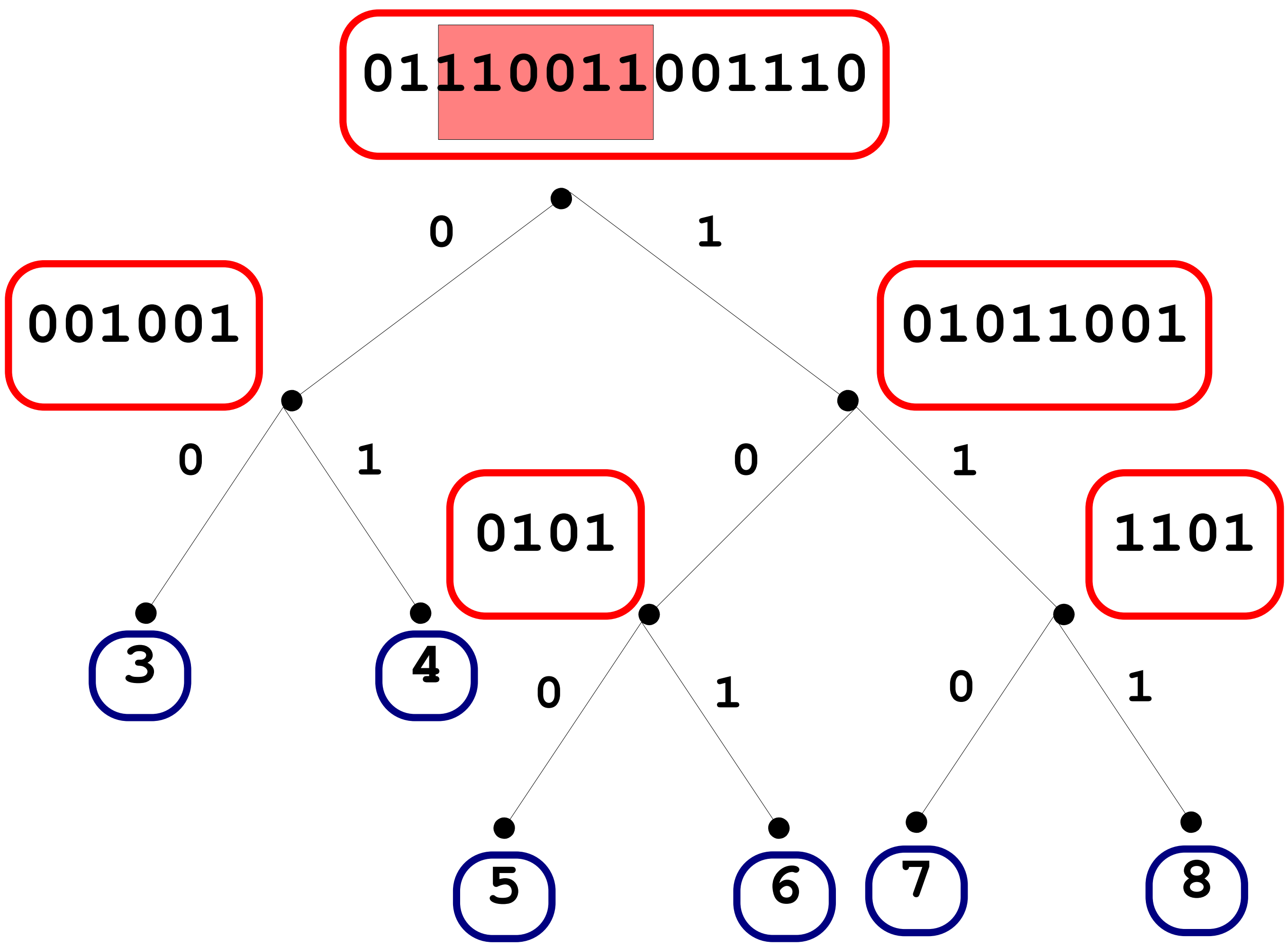
range_quantile(S,2,7,3): 3-rd smallest value in S[2···7]

S = [3,5,8,6,3,4,8,7,3,3,5,6,8,4]



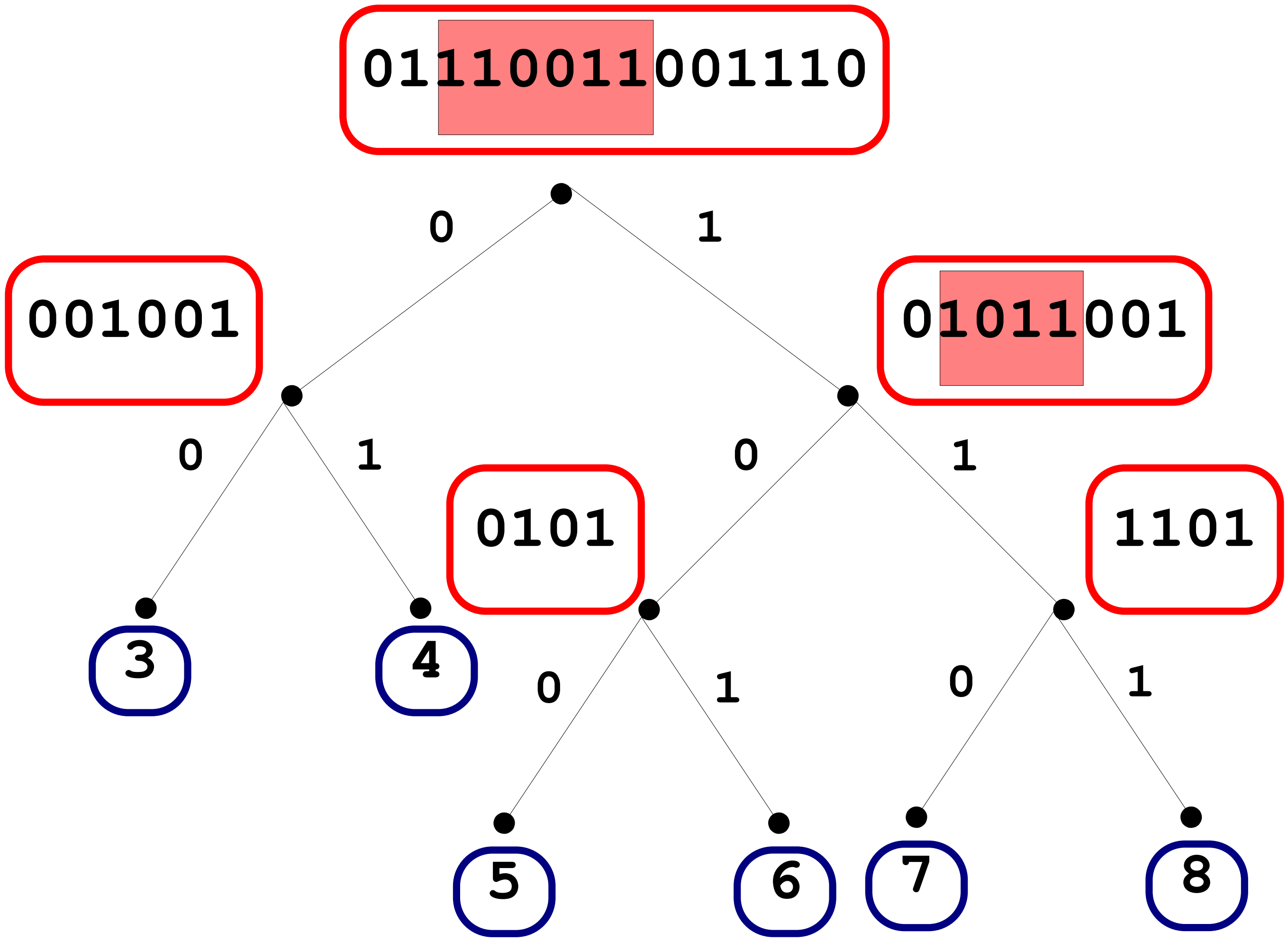
range_quantile(S,2,7,3): 3-rd smallest value in $S[2 \cdots 7]$

S = **[3,5,8,6,3,4,8,7,3,3,5,6,8,4]**



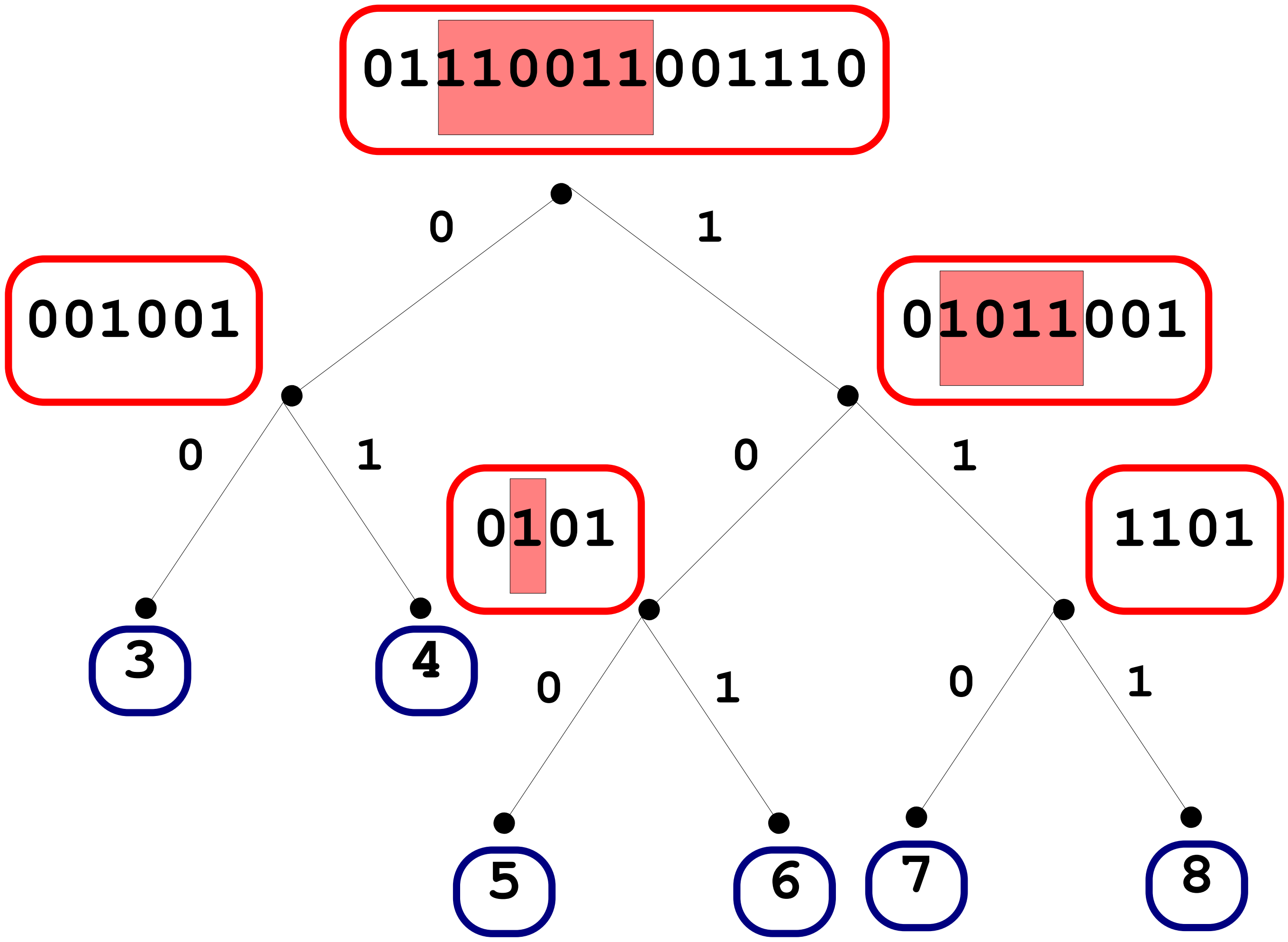
range_quantile(S,2,7,3): 3-rd smallest value in S[2⋯7]

S = [3,5,8,6,3,4,8,7,3,3,5,6,8,4]



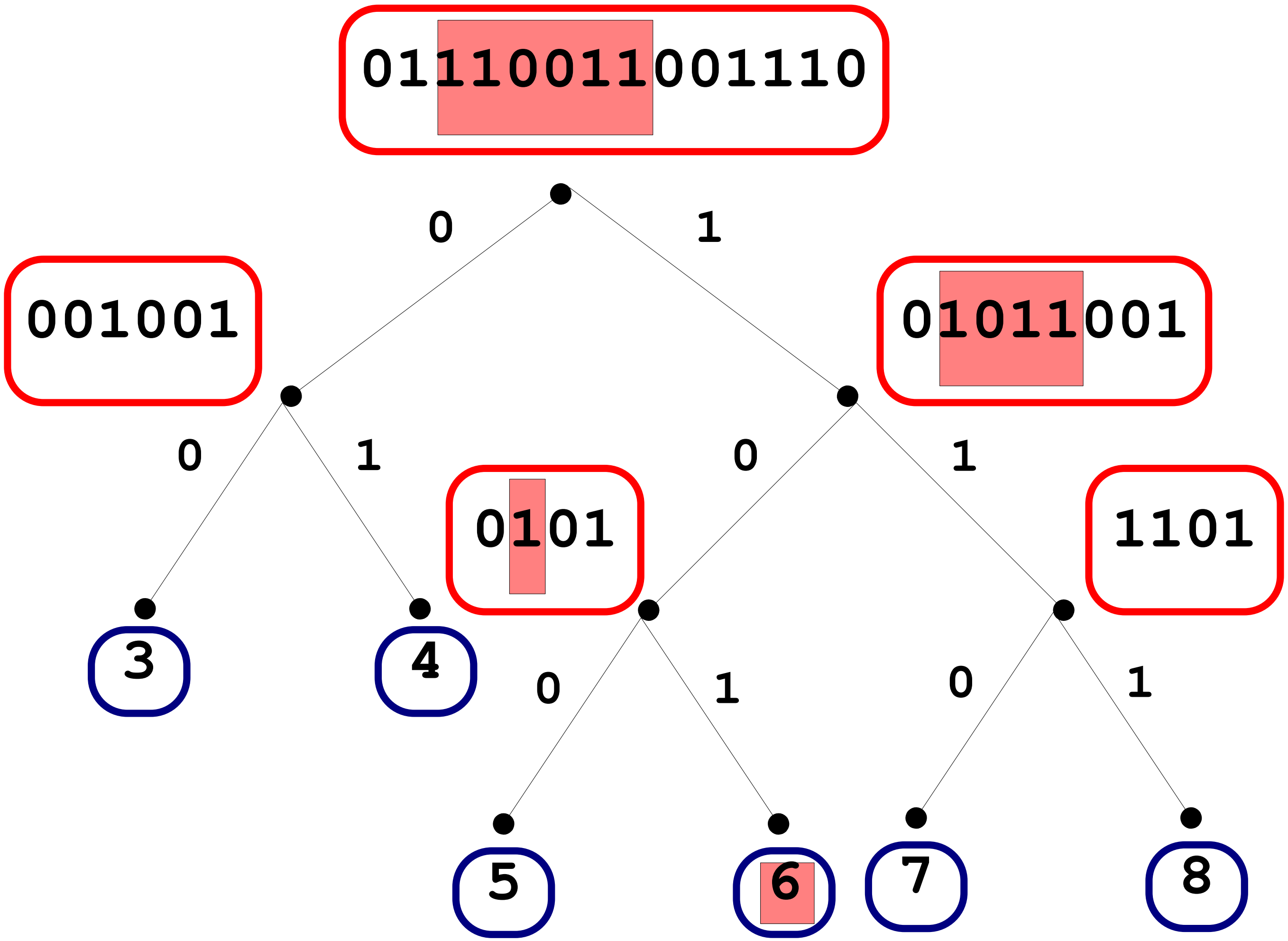
range_quantile(S,2,7,3): 3-rd smallest value in S[2···7]

S = [3,5,8,6,3,4,8,7,3,3,5,6,8,4]



range_quantile(S,2,7,3): 3-rd smallest value in S[2···7]

S = [3,5,8,6,3,4,8,7,3,3,5,6,8,4]



Other applications

- Inverted indices (representation of posting lists)
- Computations Geometry (bidimensional range queries)
- Representation of Graphs, Permutations, ...
- Bioinformatics (maximal repeats)

Implementation

- Conceptually simple, but tricky in a few details
- Several implementations cited in the literature, some available on the web
- Ready to use, open source, modular library:

  https://github.com/simongog/sdsl-lite

References

- R. Grossi, A. Gupta, J. Vitter High-order entropy-compressed text indexes. Proc. SODA 2003 (paper introducing WTs: somewhat hard to read)
- G. Navarro, V. Mäkinen. Compressed Full-text indexes. ACM Comp. Surv. 2007 (main reference for WT and compressed indices)
- G. Navarro. Wavelet Trees for All, Journal Discrete Algorithms, 2014 (recent review paper on WTs)

Thank you!