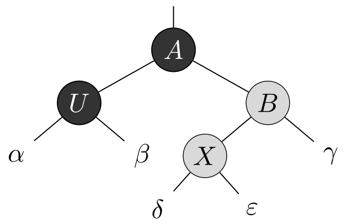
Advanced Algorithms for Data Science
HOMEWORK 2

Krikun Gosha

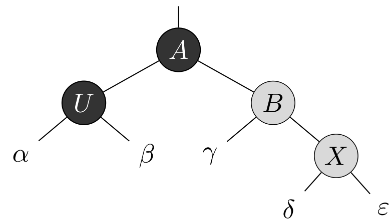
1 Red-Black trees

Question *Describe the two symmetric cases arising when B is the right child of its parent.*

In cases (a) and (b), the color of B's uncle y is black. We distinguish the two cases according to whether X is a left or right child of B.



a) the parent B of X is red, the uncle of X is black, B is the right child of its parent and X is the left child of B



b) the parent B of X is red, the uncle of X is black, B is the right child of its parent and X is the right child of B

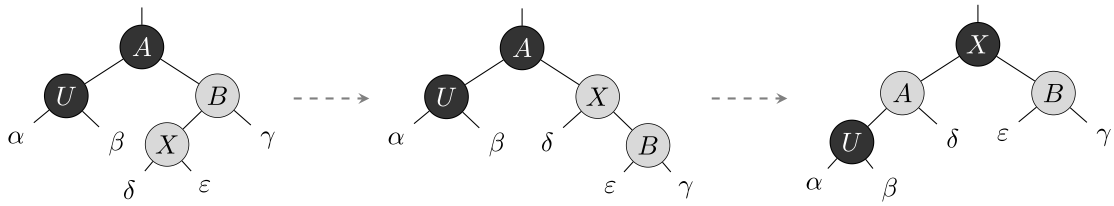
A red-black tree is a binary tree that satisfies the following **red-black properties** [1, 13.1]:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

In this cases property 4 is violated.
For fix-up this cases, we use next algorithm [1, p.320]:

In case (a), node is a left child of its parent. We immediately use a right rotation to transform the situation into case (b), in which node is a right child. Because both X and its parent B are red, the rotation affects neither the black-height of nodes nor property 5.

In case (b), we execute some color changes and a left rotation, which preserve property 5, and then, since we no longer have two red nodes in a row, we are done.



Each of the sub-trees γ, ε, δ, has a black root from property 4 and each has the same black-height the same as uncle sub-tree.

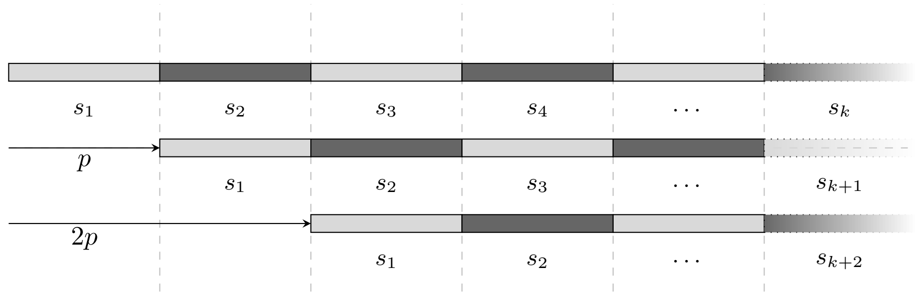
2 Knuth-Morris-Pratt algorithm

Theorem 2.1. *p is a period of T, iff $(T[i-p] = T[i+p])$, where $i \in [1+p \dots n-p]$*

As you can see, I changed little bit statement, to deal with boundaries

Proof. Lets consider graphical representation of period:

This is representation of same string T, but with different shifts.
First one without shift, second with shift p, third with shift $2p$.



Coloring doesn't represent difference, but just for visual separation

String T could be represent as concatenation of their sub-strings $s_1 \cdot s_2 \cdot ... \cdot s_n$

By definition of period p:

$$T[i] = T[i+p], \text{ where } i \in [1 \dots n-p] \tag{1}$$

Thus, $s_2 \cdot s_3 \cdot ... \cdot s_n = s_1 \cdot s_2 \cdot ... \cdot s_{n-1}$, and $s_1 = s_2 = ... = s_n$
This means that if string T, we could represent as $T = s \cdot s \cdot ... \cdot s$, then:

$$T = s \cdot s \cdot ... \cdot s \implies p = k|s|^{*}, \text{ where } k \in N \tag{2}$$

Thus we could generalize equation (1) and (2) as:

$$p_{min} = |s_{min}| \implies T[i] = T[i + kp_{min}] \tag{3}$$

*As soon as multiple shift will give same position in sub-strings and equation (1) is true

where p_{min} is a minimal period, s_{min} - minimal sub-string and $k \in N$.
We could replace the variable $i = j - p_{min}$, then for particular case $k = 2$ equation (3) takes the form:

$$p_{min} = |s_{min}| \implies T[j - p_{min}] = T[j + p_{min}] \tag{4}$$

Lets consider string period definition from other hand:

$$T[i] = T[i + p] \implies p \text{ - period} \tag{5}$$

We could symmetrically apply transformations described above, and get:

$$T[j - p_{min}] = T[j + p_{min}] \implies p_{min} = |s_{min}| \tag{6}$$

Finally from (4) and (6):

$$p \text{ - period} \iff T[j - p_{min}] = T[j + p_{min}]$$

□

To compute the minimal period of the string we could use the Knuth-Morris-Pratt algorithm. More precisely we should use prefix function π described in book[1, p.1003].

That is, $\pi[q]$ is the length of the longest prefix of T that is a proper suffix of T_q. Thus (in case $q = T.length$) we could calculate period as:

$$p_{min} = T.length - \pi[T.length]$$

3 Suffix array

The occurrence of pattern P in string T, correspond to interval $[L_p, R_p]$ in the suffix array for T. As soon as all suffixes of a string sorted lexicographical, thus we can apply binary search to quickly locate occurrence of pattern P in a string, by comparison with each suffix of string.

The starting position L_p of interval is a first suffix that greater than or equals to pattern lexicographical. The ending position R_p is a first suffix, which sub-string of length $|P|$, lexicographical greater than pattern. This definition of right boundary of the interval is fuzzy, but it important to mention that we will compare only prefixes of suffixes to find it.

```
Search-of-R_p(T, P, SA)
l ← 0
r ← SA.Length                                // or T.Length + 1
q ← P.Length
while l < r do
    m ← ⌊(l + r)/2⌋
    if P ≥_lex T[SA[m] ... SA[m] + q] then
        l ← m + 1
    else
        r ← m
return R_p ← m
```

Obviously we could compare $SA[1]$ and $SA[m]$ for handling absence of matching before search (some kind of first looking), but this is premature optimization.

At line 6 we compare prefix of suffix with pattern. Whole while loop ends when l and r positions converge and finally function yields position of first suffix which prefix greater than or equal to pattern.

Then we could analyze compare with L_p. If they are the same - there is no occurrence.

Also we should check boundary case when $R_p = m$, it could be false positive.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 2009.