NOTES

Karatsuba algorithm

Karatsuba's algorithm reduces the multiplication of two n-digit numbers to at most $n^{\log_2 3}$ that is approx $n^{1.585}$. Say we want to multiply x and y then as per the karatsuba. We have to follow following steps

Let x and y be represented as n-digit strings in some base B. For any positive integer m less than n, one can write the two given numbers as

$x = x_1B^m + x_0$
$y = y_1B^m + y_0$

where x_0 and y_0 are less than B^m. The product is then

$xy = (x_1B^m + x_0)(y_1B^m + y_0)$
$xy = x_1y_1B^{2m} + B^m (x_1y_0 + y_1x_0) + x_0y_0$

Say

$z_2 = x_1y_1$
$\mathbf{z_1 = x_1y_0 + x_0y_1}$
$z_0 = x_0y_0$

Accordingly

$xy = z_2B^{2m} + z_1B^m + z_0$

These formulae require four multiplications, and were known to Charles Babbage. Karatsuba observed that xy can be computed in only three multiplications, at the cost of a few extra additions. With z_0 and z_2 as before we can calculate

$$\mathbf{z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0}$$

which holds since

$$z_1 = (x_1 + x_0)(y_1 + y_0) - x_1y_1 - x_0y_0$$

A more efficient implementation of Karatsuba multiplication can be set as

$$xy = (b^2 + b)x_1y_1 - b(x_1 - x_0)(y_1 - y_0) + (b + 1)x_0y_0, \text{ where } b = B^m.$$

An Example

To compute the product of 12345 and 6789, choose B = 10 and m = 3. Then we decompose the input operands using the resulting base (B^m = 1000), as:

12345 = 12 x 1000 + 345
6789 = 6 x 1000 + 789

Only three multiplications, which operate on smaller integers, are used to compute three partial results:

$z_2 = 12 \times 6 = 72$
$z_0 = 345 \times 789 = 272205$
$z_1 = (12 + 345) \times (6 + 789) - z_2 - z_0 = 357 \times 795 - 72 - 272205 = 283815 - 72 - 272205 = 11538$

We get the result by just adding these three partial results, shifted accordingly (and then taking carries into account by decomposing these three inputs in base 1000 like for the input operands):

result = $z_2B^{2m} + z_1B^m + z_0$
result = $72 \cdot 10002 + 11538 \cdot 1000 + 272205 = 83810205$.

Programming implementation

```
def multiplication(X, Y):
    # convert numbers into string
    x = str(X)
    y = str(Y)
    result = 0
    # looping over y
    for i in range(len(y)):
        carry = 0  # Intermediate carry
        inter_res = ""  # Intermediate result
        # looping over x
        for j in range(len(x) - 1, -1, -1):
            # intermediate multiplication of each digit and addition of carry
            num = int(y[i]) * int(x[j]) + carry
            # if intermediate multiplication is of two digits and j>0
            #   then second digit is appended to intermediate result
            #   and first digit is stored as carry
            if num > 9 and j > 0:
                inter_res = str(num % 10) + inter_res
                carry = num // 10
            # else the digit is append to intermediate result
            # And assign carry as zero
            else:
                inter_res = str(num) + inter_res
                carry = 0
        # Adding the intermediate results
        result *= 10
        result += int(inter_res)
    return result

print(multiplication(1234, 8765))
```

Newton's Method to find square root

Let N be any number then the square root of N can be given by the formula:

$$root = 0.5 * (X + (N / X))$$

where X is any guess which can be assumed to be N or 1. In the above formula, X is any assumed square root of N and root is the correct square root of N. Tolerance limit is the maximum difference between X and root allowed.

1. Assign X to the N itself.
2. Now, start a loop and keep calculating the root which will surely move towards the correct square root of N.
3. Check for the difference between the assumed X and calculated root, if not yet inside tolerance then update root and continue.
4. If the calculated root comes inside the tolerance allowed then break out of the loop.
5. Print the root.

```
def squareRoot(n, l) :
    x = n
    count = 0

    while (1) :
        count += 1
        root = 0.5 * (x + (n / x))
        if (abs(root - x) < l) :
```

```
            break
        x = root
    return root

n = 327
l = 0.00001
print(squareRoot(n, l))
```

Deletion in Red-Black Tree

To understand deletion, notion of **double black** is used. When a black node is deleted and replaced by a black child, the child is marked as double black.

If we delete a node, what was the color of the node removed?

if Red then

1. We won't have changed any black height nor
2. will we have created 2 red nodes in a row
3. also, it could not have been the root!

if Black then

1. Could violate any of root rule, red rule, or black-height rule!

Observations

• If we delete a red node, tree is still a red-black tree. A red node is either a leaf node or must have two children then rules are-

1. If node to be deleted is a red leaf, remove leaf, done
2. If it is a single-child parent, it must be black, replace with its child (must be red) and recolor child black
3. If it has two internal node children, swap node to be deleted with its in-order successor
   - if in-order successor is red remove leaf, done
   - if in-order successor is a single child parent, apply second rule

4. if in-order successor is a black leaf, or if the node to be deleted is itself a black leaf, things get complicated. Let us understand the cases-

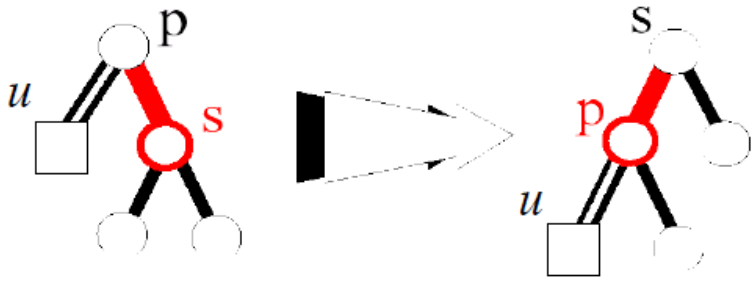
Case A: Black-Leaf Removal

We want to remove v, which is a black leaf, Replace v with external node u, color u double black. To eliminate double black edges-

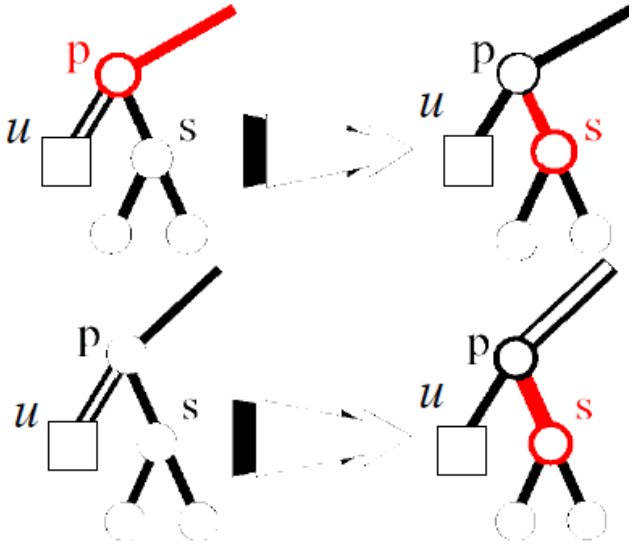
1. Find a red edge nearby, and change the pair (red, double black) into (black, black)
2. As with insertion, we recolor and/or rotate
3. Rotation resolves the problem locally, whereas recoloring may propagate it two levels up!

Case B: Red Sibling

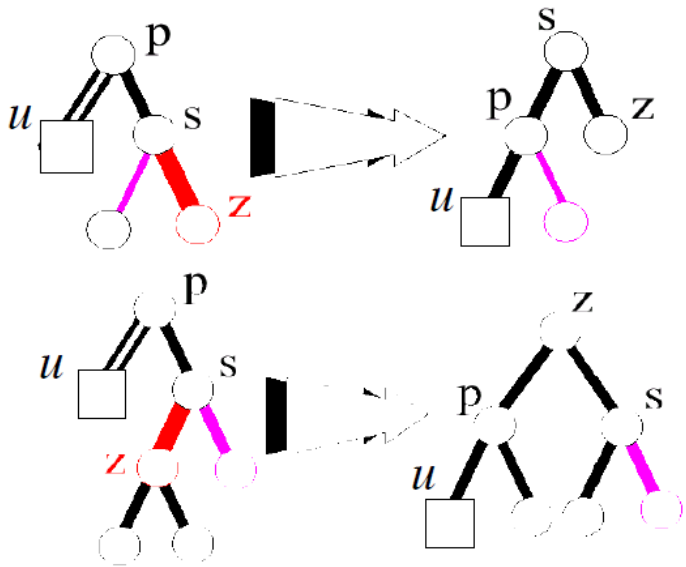
If sibling is red, rotate such that a black node becomes the new sibling then treat it as a black-sibling case (Discussed later)



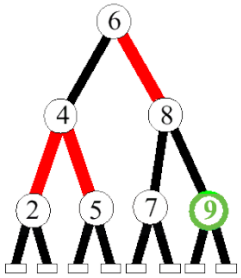
Case C: Black Sibling and Nephew/Niece
If sibling and its children are black, recolor sibling and parent. If parent becomes double black, percolate up!



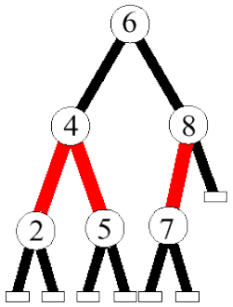
Case D: Black Sibling but Red Nephew
If sibling is black and one of its children is red, rotate and recolor red nephew involved in rotation!



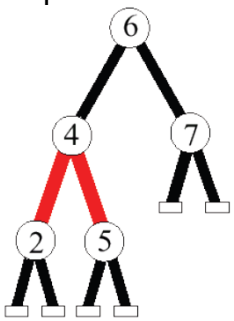
An Example
Consider the following Tree-



Remove 9: For 9, sibling and its children are black so this is case C, recolor sibling and parent. It will look like.



Remove 8: Its inorder successor is red so replace 8 with children and then recolor it.



Remove 7: Sibling is black with red nephew so rotate and recolor red nephew involved in rotation, it will be