NB: Algorithm is valid for only .fastq files with reads not containing empty fields.

Tailcut algorithm is designed to get obtained data with reads from a certain
.fastq file and rewrite them(data) having cut poly(A) suffix or poly(T) prefix.
Python script is run from a command line

Parameters can be variated:
- maximum length of a read (in our example - 101)
- critical percentage of mistakes is allowed to be in a poly(A/T)tail (in our example - 0.10)
- minimum length of a tail - length, beneath which tail cannot be cut (in our example - 6)
- mode of performance – poly(A) tail or poly(T) head we cut (in our example - T)

Here is the common pattern of algorithm's behaviour:

First we calculate CSA (critical S amount) - the maximum amount of S(i.e Strong, non-A/T, W = A/T, Weak) nucleotides can be observed (in our particular case - 101*0.10 = 10.1 -> 11).

NB: always an equal or greater integer is taken for CSA (had we maximum length = 100, CSA would be 100*0.10 = 10 -> 10)

If A mode is run, the read sequence in reversed, so it is prefix that is always analysed.

For current read we get a prefix by scanning every letter. After each letter scan we check whether scanned nucleotide is W or S and enlarge an auxiliary array called Score Array (SA) which is always as long as currently observed prefix. Nucleotide's score which is (amount of W in current prefix/length of current prefix) is stored in SA.

Prefix enlarging is finished when amount of S nucleotides equals CSA.
After that last prefix' letter is checked whether it is S or W (we do not want our algorithm to cut first letters of coding sequence because tail was very pure). If last prefix's letter is S, prefix and SA are shortened one by one nucleotide from the end, until prefix is ended by T.

NB: Algorithm considers tail can end only by W nucleotide.

When prefix is finally ends with W nucleotide, end nucleotide's score is checked in the SA (it should be equal or greater than (1 - critical percentage) (in our particular case - 1 - 0.10 = 0.90)). If score is lower, prefix and SA are shortened again one by one nucleotide from the end, until end nucleotide's score satisfies critical percentage request or prefix is shorter than minimum length desired.

After that the procedure is repeated for next read.
At the end modified (i.e. cut) and not modified reads are written in two separate files.

Following example can display the work of algorithm.

Consider we want to cut a T tail from an array of reads with maximum length 21, and we want tail to be longer than 4 nucleotides and contain not more than 20 % mistakes. Therefore, parameters will be:

maximum length = 21
critical percentage = 0.2
minimum length = 5
mode of performance = "T"
read from file intput.fastq
write cut reads to file output_cut.fastq
write other reads to file output_uncut.fastq

Connand to run the script:
python Tailcut1.1.py 21 5 0.2 T intput.fastq output_cut.fastq output_uncut.fastq

CAX = 21*0.2 = 4.2 -> 5

Consider than we have a read, say,
TTTCTAAGCCCAAAAT
So, here we go:

Step	Prefix	SA
1	T	1.0
2	TT	1.0 1.0
3	TTT	1.0 1.0 1.0
4	TTTC	1.0 1.0 1.0 0.75
5	TTTCT	1.0 1.0 1.0 0.75 0.8
6	TTTCTA	1.0 1.0 1.0 0.75 0.8 0.66
7	TTTCTAA	1.0 1.0 1.0 0.75 0.8 0.66 0.57
8	TTTCTAAG	1.0 1.0 1.0 0.75 0.8 0.66 0.57 0.5
9	TTTCTAAGC	1.0 1.0 1.0 0.75 0.8 0.66 0.57 0.5 0.44

Here prefix enlarging stops, as there are already 5 S nucleotides in prefix and the next one is S as well.

Than we cut prefix' end until it is T:

10	TTTCTAAG	1.0 1.0 1.0 0.75 0.8 0.66 0.57 0.5
11	TTTCTAA	1.0 1.0 1.0 0.75 0.8 0.66 0.57
12	TTTCTA	1.0 1.0 1.0 0.75 0.8 0.66
12	TTTCT	1.0 1.0 1.0 0.75 0.8

Is end nucleotide's score(0.8) satisfactory? It must be >= than 1 - critical percentage = 1 - 0.2 = 0.8.
Length of an obtained tail is 5, which is satisfactory as well.

So we cut off tail TTTCT and live happy with modified read AAGCCCAAAAT.