**Define the problem:**

Many PCR Duplicates are included in the data set received. They are unwanted and can skew results – inaccurate representation of alleles/heterozygosity.

A PCR duplicate, by definition, has the same location, comes from the same chromosome, has the same strandedness and has the same UMI as another fragment. Location/position can be influenced by soft-clipping, which must be accounted for.

In this assignment, we only want to keep the first PCR duplicate that occurs in a set, when reading down the list of the data set.

Left-most location is specified in column 4, and soft-clipping is denoted in the CIGAR string – column 6.

The chromosome is specified in column 3.

The UMI is appended to the end of the column 1 – the QNAME.

The stranded-ness orientation is specified in the bitwise flag, in column 2.

All of these must be taken into account in order to properly identify and remove PCR duplicates.

**INPUT EXAMPLE (2 RECORDS):**

NS500451:154:HWKTMBGXX:1:11101:24260:1121:CTGTTCAC 0 2 76814284 36 71M \* 0 0 TCCACCACAATCTTACCATCCTTCCTCCAGACCACATCGCGTTCTTTGTTCAACTCACAGCTCAAGTACAA 6AEEEEEEAEEAEEEEAAEEEEEEEEEAEEAEEAAEE<EEEEEEEEEAEEEEEEEAAEEAAAEAEEAEAE/ MD:Z:71 NH:i:1 HI:i:1 NM:i:0 SM:i:36 XQ:i:40 X2:i:0 XO:Z:UU

NS500451:154:HWKTMBGXX:1:11101:24260:1121:CTGTTCAC 0 2 76814282 36 2S69M \* 0 0 TCCACCACAATCTTACCATCCTTCCTCCAGACCACATCGCGTTCTTTGTTCAACTCACAGCTCAAGTACAA 6AEEEEEEAEEAEEEEAAEEEEEEEEEAEEAEEAAEE<EEEEEEEEEAEEEEEEEAAEEAAAEAEEAEAE/ MD:Z:71 NH:i:1 HI:i:1 NM:i:0 SM:i:36 XQ:i:40 X2:i:0 XO:Z:UU

**OUTPUT EXAMPLE (1 RECORD – DUPLICATE REMOVED):**

NS500451:154:HWKTMBGXX:1:11101:24260:1121:CTGTTCAC 0 2 76814284 36 71M \* 0 0 TCCACCACAATCTTACCATCCTTCCTCCAGACCACATCGCGTTCTTTGTTCAACTCACAGCTCAAGTACAA 6AEEEEEEAEEAEEEEAAEEEEEEEEEAEEAEEAAEE<EEEEEEEEEAEEEEEEEAAEEAAAEAEEAEAE/ MD:Z:71 NH:i:1 HI:i:1 NM:i:0 SM:i:36 XQ:i:40 X2:i:0 XO:Z:UU

The second record has been removed from the output, because it is a duplicate of the first. The only difference between the two is the CIGAR string and the start position. When taking into account the soft-clipping of the second record, and its influence on the start position, the second record is a PCR duplicate of the first. Only the first record is outputted.

*Modify input file, for organization.  
cat paired\_end.sam | grep "^N" | sort -k 3,3 -k 2,2 | less -S*

*So that it is sorted by chromosomes, and then secondarily by the bit flag*

**Pseudocode**

Set argparse commands:

* -f for input data set of reads file
* -u for input file of list of UMI’s
* -h “Input -f for input read file; input -u for input file of UMI list”

Open file with UMIs as “UMI”, from argparse “-u” argument – read into and store as list – **umi\_list:**

Open 2 output files – as “saved” and “removed”

Open file to process as **data\_set**, from argparse “-f” argument

Read line by line:

Store first line as **record**

Initialize variable **check\_list =** []

Break **record** into list:

Set working\_list = [Current\_line.split()]

Using indexing in working\_list, separate out:

* the QNAME (With UMI)
* Chromosome
* Location
* CIGAR string – for clipping
* Bitwise flag – for strandedness

**qname** = qname index of **working\_list**

**Chromosome** = number from chromosome index of **working\_list**

**Raw\_location** = integer value from location index of **working\_list** (this will still have to be processed further, and is not the final location)

**Cigar =** alphanumeric string from cigar index of **working\_list**

**Bit** = integer value from bitwise flag portion of **working\_list**

**Function – rev\_comp (**astring)**:**

‘’’Given alphabetical string, return the reverse compliment’’’

* Replace method on astring:
  + “A” with ‘t’
  + “T” with ‘a’
  + “G” with ‘c’
  + “C” with ‘g’
* Make all the string uppercase
* return new string **revc**

TEST: rev\_comp(“AGTC”)

Returns “GACT”

**Function – UMI\_processor(astring):**

‘’’ Given a UMI string in the form XXX^XXX, where X is any nucleotide (and can be repeated multiple times) – the UMI processor checks to make sure the left hand and right hand side of the “^” dividing the UMI in the middle are reverse complements of each other.

Checks to make sure the UMI’s are found in the UMI list

Checks to make sure the UMI’s are of high enough quality to not have any Unknown bases (“N”) in them. ‘’’

* Using Regex, return the portion of **qname** that contains the UMI (ex: GCCTATT^GGCGTAT), as variable **both\_umi**
* If an “N” is present in **both\_umi**:
  + Ex: CCTAG**N**TT^CCTAGCTT
  + break UMI processor function
  + move record to “removed” file
  + break UMI function and move to next record
* Else – split **both\_umi** on “^”
* Initialize variable **Umi\_one =** first half of **both\_umi** (before the “^”)
  + Ex: GCCTATT
* Initialize variable **umi\_two** = second half of **both\_umi** (after the “^”)
  + Ex: GGCGTAT
* Does rev\_comp(**umi\_one == umi\_two**):?
  + Not equivalent:
    - Move record to “removed” file
    - Break UMI function and move to next record
* Are equivalent:
  + Search for **umi\_one** in UMI list
    - **Umi\_one** NOT found in **umi\_list**
      * Move record to “removed” file
      * Break **umi\_processor** and move to next record
    - If **umi\_one** IS found in **umi\_list:**
      * Continue program
      * Return **both\_umi**
      * *NOTE: variable* ***Both\_umi*** *will be used later – although one/two won’t*

TEST: **umi\_processor(GCCGA^TCGGC)**

Return (without breaking) **GCCGA^TCGGC**

TEST: **umi\_processor(GNNGA^TCNNC)**

Returns nothing; breaks and moves to next record

**Function: cigar\_adj(cigar)**:

**‘’’** Takes the string from the CIGAR string portion of the record, and returns a value to adjust the **raw\_location** to the **true\_location** of the read if necessary ‘’’

* Var **cigar\_adjust** = 0
* Using regex, separate the string **cigar** into groups – numeric digits, followed by the first alphabetical character
  + Ex: 17S34M gives 2 groupings – (17S) as first grouping; and (34M) as second
* If the first alphabetical character in the first grouping is “S”:
  + Set **cigar\_adjust** = numeric value previous to the S – only numeric value in that grouping
* If first alphabetical character in the first grouping is not “S”:
  + Leave **cigar\_adjusts** value as 0
* Return **cigar\_adjust**

TEST: **cigar\_adj(17S34M)**

Returns 17

Set variable **true\_location** = **raw\_location** + **cigar\_adjust**

**‘’’ *NOTE:*** *Checked through data set on talapas – count of cigar strings overall is equal to the count of cigar strings that have an “M” in them.*

*When S is present in a cigar string, it is ALWAYS the first alphabetical character in the string – though sometimes also the last alphabetical character (this can be ignored; not doing a paired-end deduper)* ‘’’

**Function: Bit\_flag\_interpreter(an\_int):**

‘’’ Takes an input of the bit flag, and checks to see if the bitwise value for a reverse complement is present in it (16). If it is, it returns the Boolean variable **isrev** as true, and otherwise, returns it as false.”

* Set variable **isrev** = None (Boolean)
* Using bitwise Ampersand, check if 16 is present in variable **bit**
  + If 16 IS present – set **isrev** = True
  + If 16 is NOT present – set **isrev** = False
* Return **isrev**

TEST: bit\_flag\_interpreter(99):

Returns: False

TEST: bit\_flag\_interpreter(147):

Returns: True

Ascertain if current record is a PCR duplicate:

# Holds onto all information necessary for checking to see if a read is or isn’t a duplicate:

Var **add\_list** = [**both\_umi, chromosome, true\_location, isrev]**

if **add\_list** in **check\_list:**

* move record to “removed” file
* move to next record

**if add\_list** not in **check\_list:**

* move record to “saved” file
* append **add\_list** to **check\_list**
* move to next record

close all files