INTRODUCTION TO (GENOMIC) ALIGNMENTS

DAVIDE BOLOGNINI

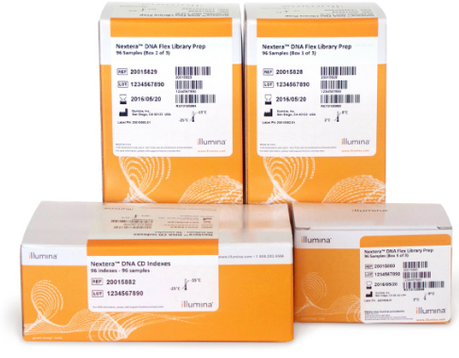
MOLECULAR MEDICINE PHD FELLOW

DEVELOPING @ HTTPS://GITHUB.COM/DAVIDEBOLO1993

DAVIDEBOLOGNINI7@GMAIL.COM

1. BACKGROUND

GENERATING READS



Illumina Nextera DNA Flex Library Prep Kit

Invitrogen QuBit

Illumina HiSeq 4000

SAMPLE PREPARATION

SAMPLE QUANTIFICATION

SAMPLE SEQUENCING

SEQUENCING READS

Oxford Nanopore Technologies DNA sequencing kit

Invitrogen QuBit

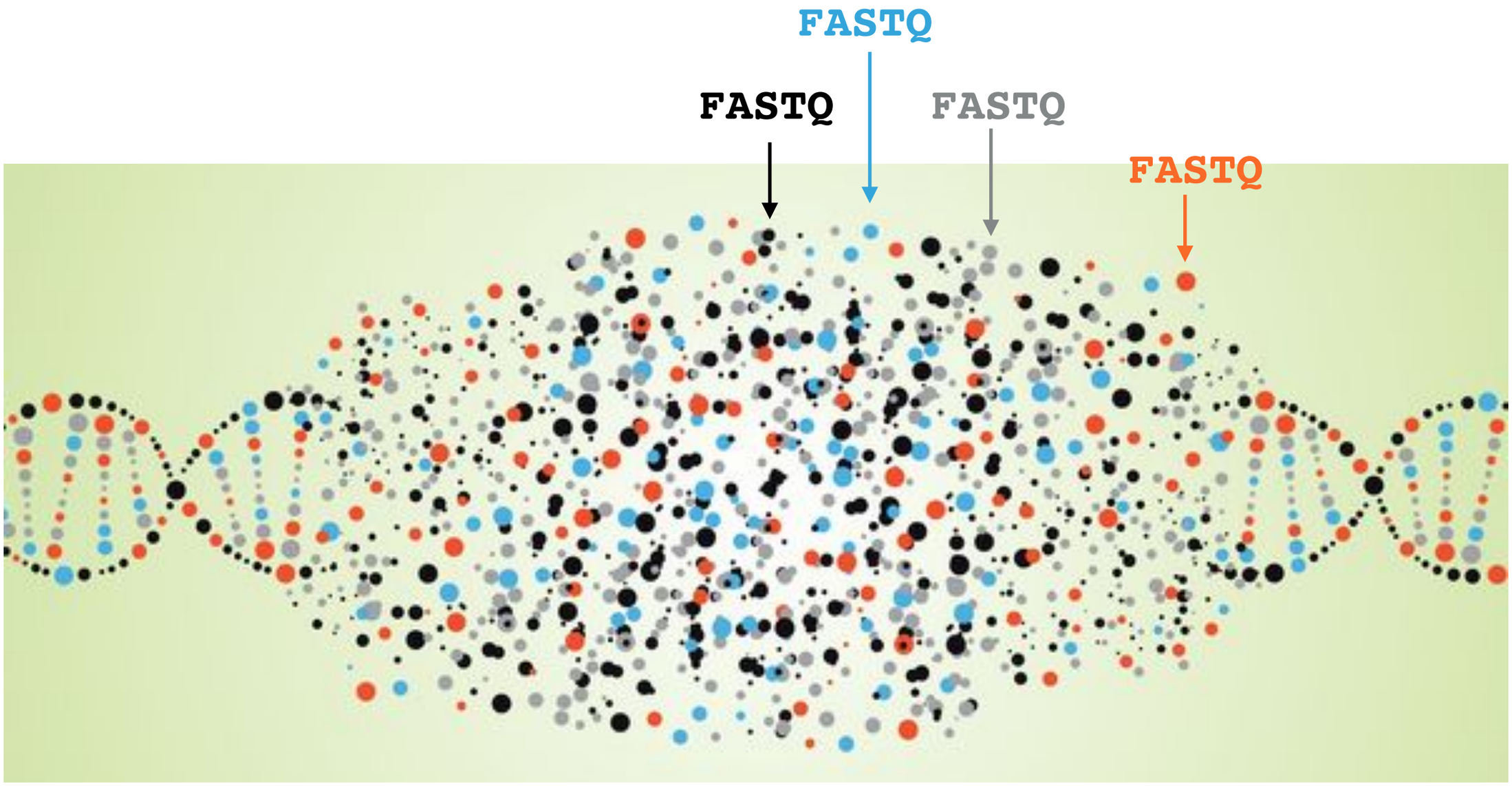
Oxford Nanopore Technologies GridION X5

READ: FASTQ

```
@SEQ_ID @ Sequence ID
TGCCCAGTGGTGGAGGTCTCAGAAATGGTATAAAAGTCAGACAAATCTTGGTTTTGCATT Sequence
+ + (Sequence ID)
!’**((((***+))%%%++)(%%%%).1***-+*’’))**55CCF>>>>>>CCCCCCC65 Sequence quality
```

REFERENCE: FASTA

```
>chr20:17000000-17000539 > Sequence ID
AGAGTGAGACTCTGAAATAAATAAATAAATAAATAAATAAACAATAAACCACGAGCAAGA
GACAAACAGGAAAAAAAAAAAGAATGAACAATTATTTATTTTGGAAGAAACATCTGAGAT
GGTAGGTAGTGTCAAAGATAGCAGTGCCCAGTGGTGGAGGTCTCAGAAATGGTATAAAAG
TCAGACAAATCTTGGTTTTGCATTCTCCAAGTCTGGTTTTTTAGCCTCTGCTGACTGCTG
GAACTACTTGATAATCTTCCAAAAATTACTTCTTGACTAAGGTCACCAAGGCCAGTTTTT  Sequence
GTGGCTTGCACCTGGACTCTTGGCTGGCACTGGGACATGCTGCTTAATGGCGGGAAAACA
ATTAATAATTCCCATTTAAGTCAGGAAAAATACAAGATTGTATATCACCATTACTATTTT
TAGCATTGTCCTGGAAGTAAAATGTAATTATACAAGAAGATTATGTTTGGTTTACATATT
GAAAAGAAGGCAGAAAATTTATCATTATTTGTAGACCATCCTATTATCTACCCAGAAGAT
```



The genome jigsaw

FASTA

Take-home message #1

Genomic alignments map the sequencing reads to the reference genome in order to identify their original genomic *loci*

READ

```
@SEQ_ID
TGCCCAGTGGTGGAGGTCTCAGAAATGGTATAAAAGTCAGACAAATCTTGGTTTTGCATT
+
!'**((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

REFERENCE

```
>chr20:17000000-17000539
AGAGTGAGACTCTGAAATAAATAAATAAATAAATAAATAAACAATAAACCACGAGCAAGA
GACAAACAGGAAAAAAAAAAAGAATGAACAATTATTTATTTTGGAAGAAACATCTGAGAT
GGTAGGTAGTGTCAAAGATAGCAGTGCCCAGTGGTGGAGGTCTCAGAAATGGTATAAAAG
TCAGACAAATCTTGGTTTTGCATTCTCCAAGTCTGGTTTTTTAGCCTCTGCTGACTGCTG
GAACTACTTGATAATCTTCCAAAAATTACTTCTTGACTAAGGTCACCAAGGCCAGTTTTT
GTGGCTTGCACCTGGACTCTTGGCTGGCACTGGGACATGCTGCTTAATGGCGGGAAAACA
ATTAATAATTCCCATTTAAGTCAGGAAAAATACAAGATTGTATATCACCATTACTATTTT
TAGCATTGTCCTGGAAGTAAAATGTAATTATACAAGAAGATTATGTTTGGTTTACATATT
GAAAAGAAGGCAGAAAATTTATCATTATTTGTAGACCATCCTATTATCTACCCAGAAGAT
```

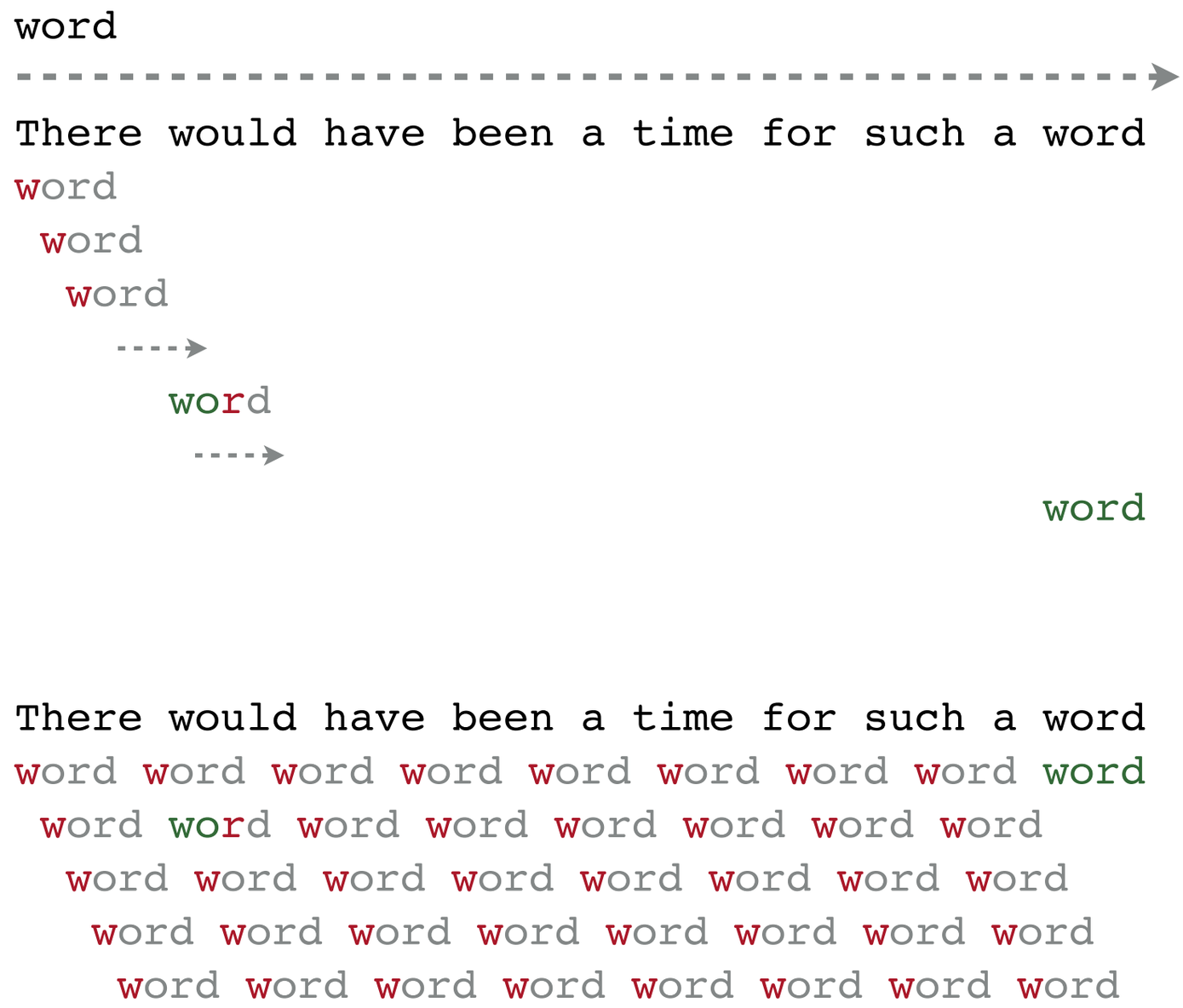
ALIGNMENT

```
                TGCCCAGTGGTGGAGGTCTCAGAAATGGTATAAAAGTCAGACAAATCTTGGTTTTGCATT
TAGCAGTGCCCAGTGGTGGAGGTCTCAGAAATGGTATAAAAGTCAGACAAATCTTGGTTTTGCATTCTCCAA
```

2. EXACT STRING MATCHING ALGORITHMS

P: word

T: There would have been a time for such a word



Q: How many alignments ?

A: |T| - |P| +1

Q: How many character comparison ?

A: Range from |T| - |P| +1 (best case) to |P|*(|T| - |P| +1) (worst case)

Q: How this can be improved ?

A: Learn from character comparisons to skip pointless alignments

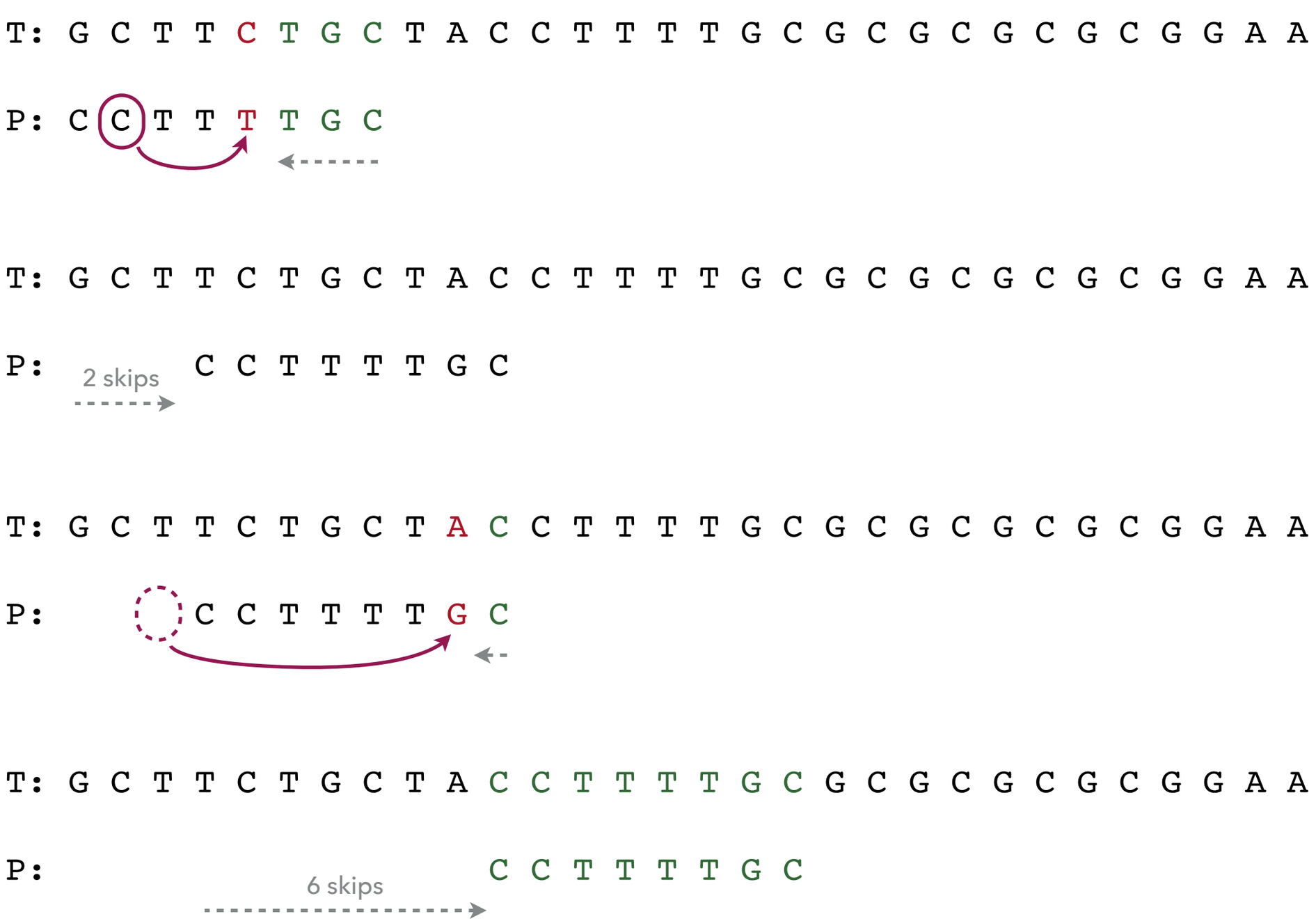
```
word
----------------------------------------->
There would have been a time for such a word
      word
        ----> skip
         ----> skip
          word

#As ‘u’ not in ‘word’, skip the next 2 alignments
```

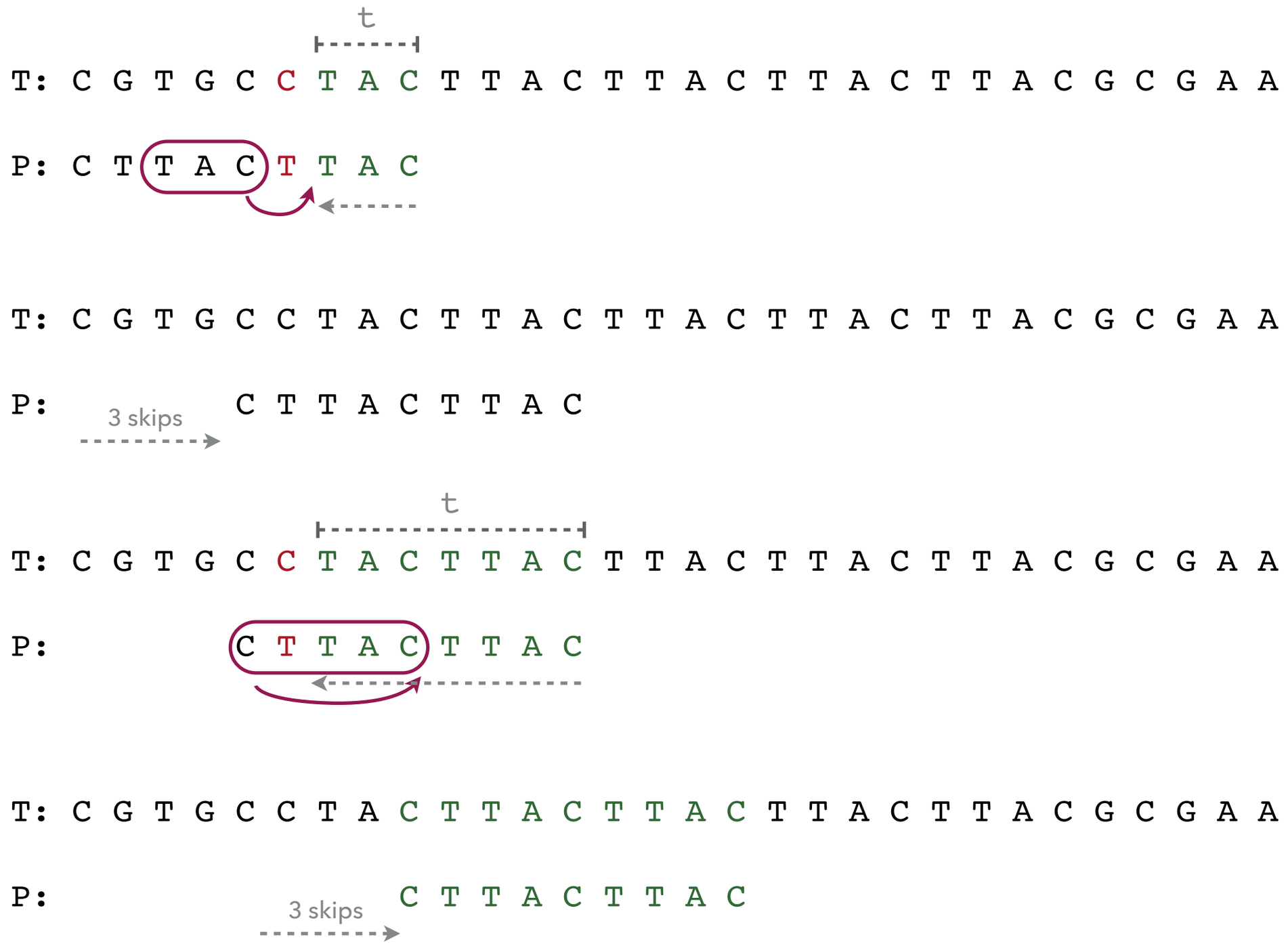
Boyer Moore's rules:

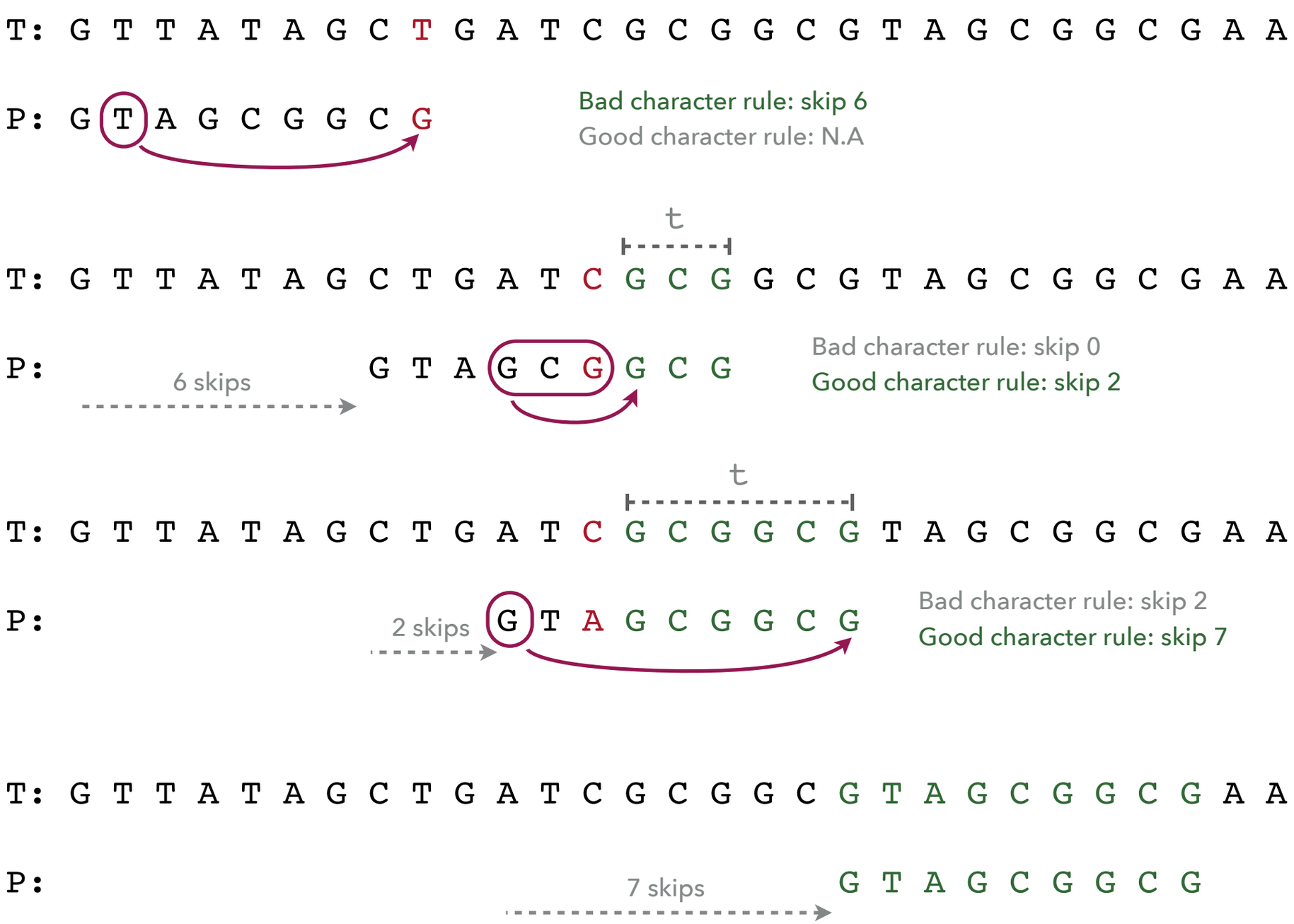
1. Test alignments from left to right, but perform character comparisons in opposite direction

2. Upon hitting a mismatch, skip alignments until mismatch becomes a match or P moves past the mismatch (aka, BAD CHARACTER RULE)

3. Let t be a substring of T matched by a substring of P. Upon hitting a mismatch, skip alignments until there are no mismatches between P and t/a prefix of P matches a suffix of T or P moves past t (aka, GOOD CHARACTER RULE)


T: G C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C
T: G C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: 2 skips C C T T T T G C
T: G C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C
T: G C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: 6 skips C C T T T T G C

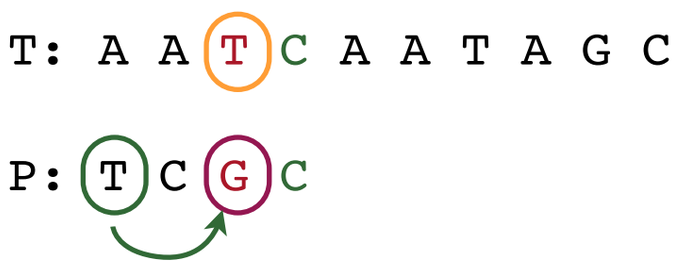

t
T: C G T G C C T A C T T A C T T A C T T A C T T A C G C G A A
P: C T T A C T T A C
T: C G T G C C T A C T T A C T T A C T T A C T T A C G C G A A
P:
3 skips
C T T A C T T A C
t
T: C G T G C C T A C T T A C T T A C T T A C T T A C G C G A A
P: C T T A C T T A C
T: C G T G C C T A C T T A C T T A C T T A C T T A C G C G A A
P:
3 skips
C T T A C T T A C

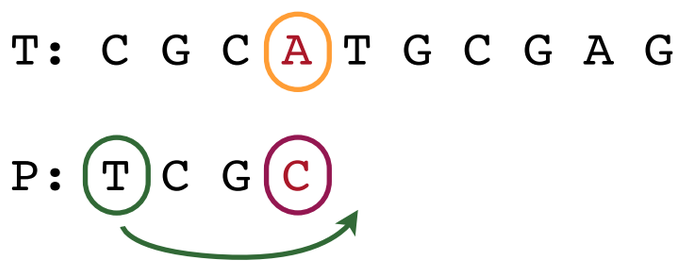

T: G T T A T A G C T G A T C G C G G C G T A G C G G C G A A
P: G T A G C G G C G
Bad character rule: skip 6
Good character rule: N.A
t
T: G T T A T A G C T G A T C G C G G C G T A G C G G C G A A
P:
6 skips
G T A G C G G C G
Bad character rule: skip 0
Good character rule: skip 2
t
T: G T T A T A G C T G A T C G C G G C G T A G C G G C G A A
P:
2 skips
G T A G C G G C G
Bad character rule: skip 2
Good character rule: skip 7
T: G T T A T A G C T G A T C G C G G C G T A G C G G C G A A
P:
7 skips
G T A G C G G C G

P

Σ	T	C	G	C
A	0	1	2	3
C	0	–	0	–
G	0	1	–	0
T	–	0	1	2

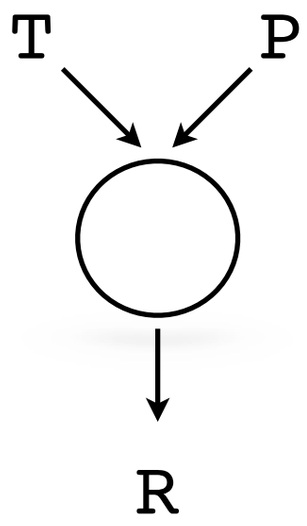
Bad character rule: lookup table for P





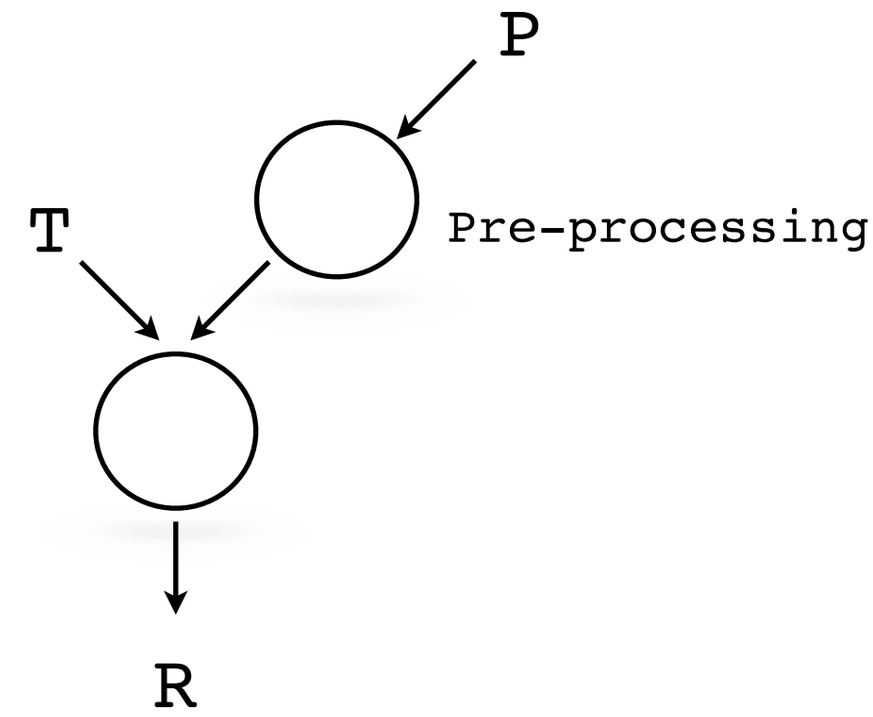
Q: Any benefits?

A: Overall work reduced via reuse

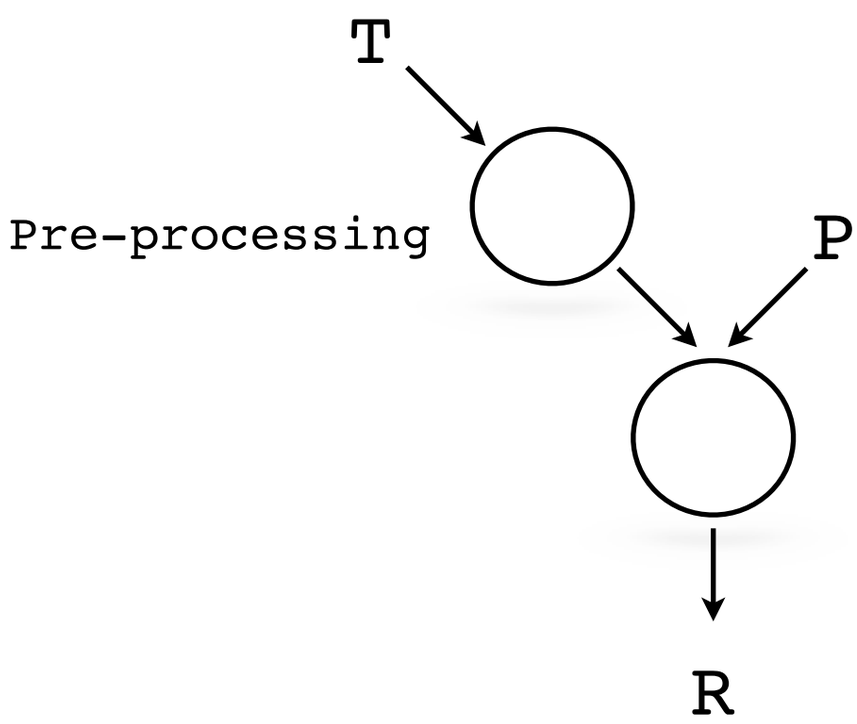


ONLINE

Brute Force String Matching

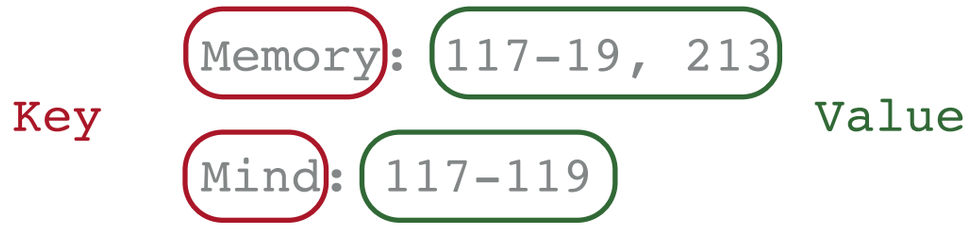


Boyer-Moore String Matching

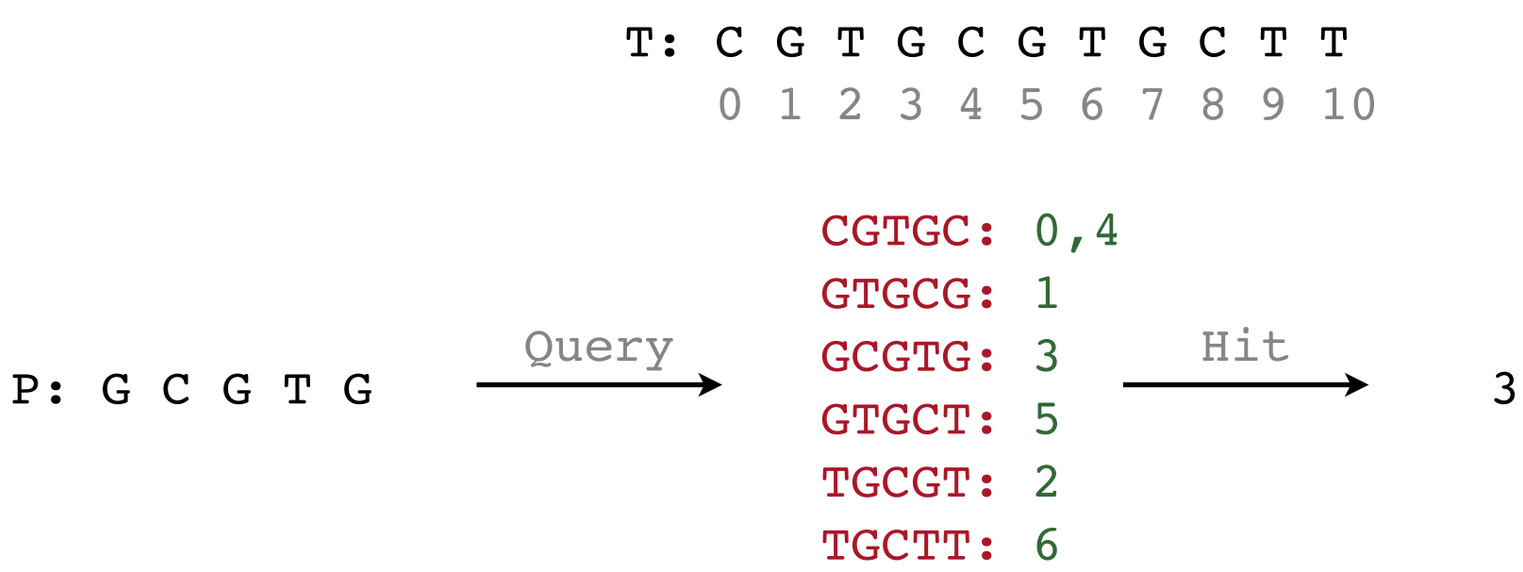


OFFLINE

Index of a book for terms ‘memory’ and ‘mind’



Index of DNA string T for 5-mers substrings



T: G T G C G T G T G G G G

CGT	3
GCG	2
GGG	8,9,10
GTG	0,4,6
TGC	1
TGG	7
TGT	5

TGG > GTG

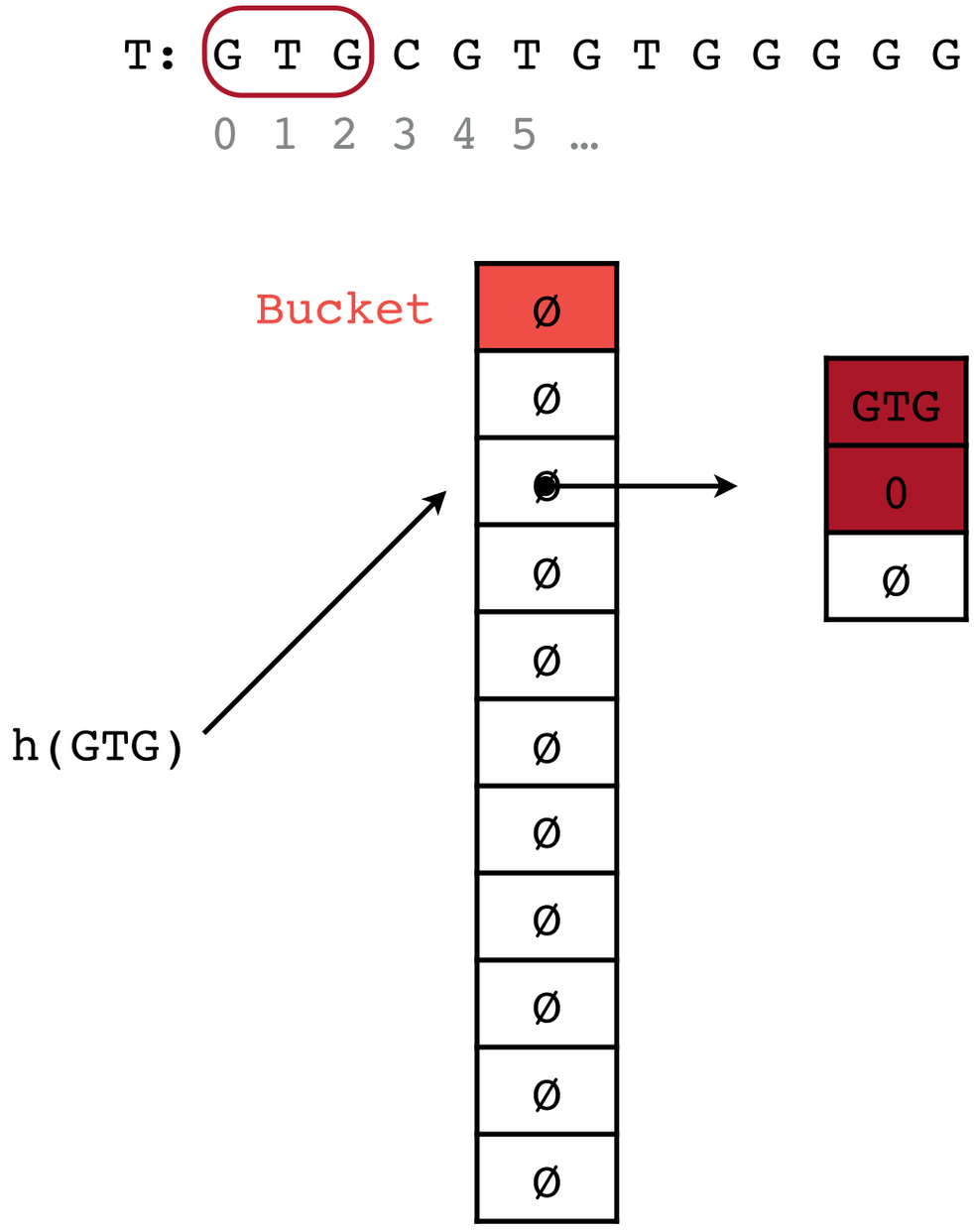
query

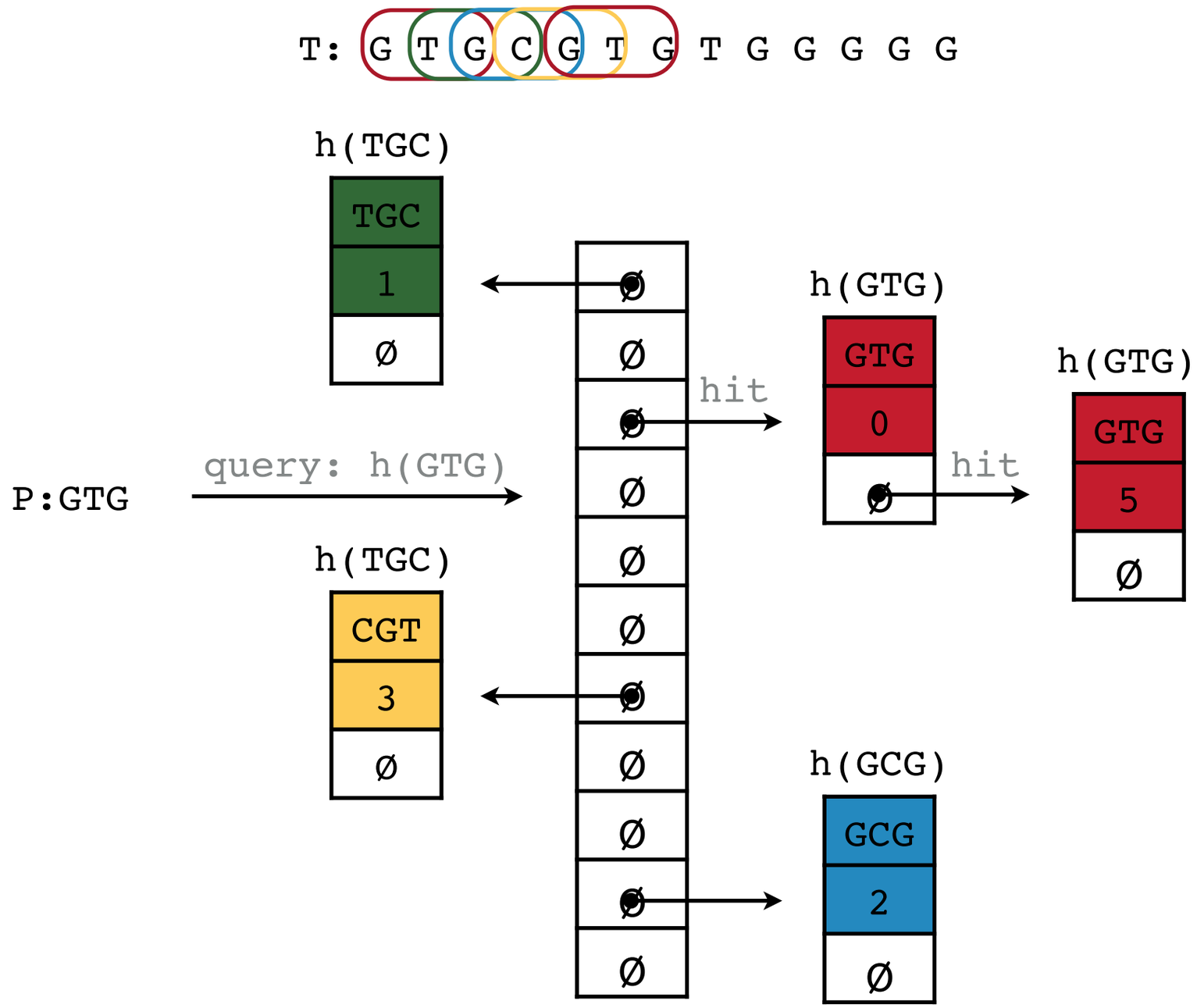
CGT	3
GCG	2
GGG	8,9,10
GTG	0,4,6
TGC	1
TGG	7
TGT	5

TGG = TGG

hit

7


T: G T G C G T G T G G G G G
0 1 2 3 4 5 …
Bucket
Ø
Ø
Ø
Ø
Ø
Ø
Ø
Ø
Ø
Ø
GTG
0
Ø
h(GTG)


T: G T G C G T G T G G G G G
h(TGC)
TGC
1
Ø
P:GTG
query: h(GTG)
h(TGC)
CGT
3
Ø
Ø
Ø
Ø
Ø
Ø
Ø
Ø
Ø
Ø
hit
h(GTG)
GTG
0
Ø
hit
h(GTG)
GTG
5
Ø
h(GCG)
GCG
2
Ø

3. A BIOINFORMATIC EXAMPLE

In [1]:
```
!wget --no-check https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/phix.fa
```

```
--2019-10-16 16:17:36--  https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/phix.fa
Risoluzione di d28rh4a8wq0iu5.cloudfront.net (d28rh4a8wq0iu5.cloudfront.net)... 143.204.10.105, 143.204.10.216, 143.2
04.10.21, ...
Connessione a d28rh4a8wq0iu5.cloudfront.net (d28rh4a8wq0iu5.cloudfront.net)|143.204.10.105|:443... connesso.
Richiesta HTTP inviata, in attesa di risposta... 200 OK
Lunghezza: 5528 (5,4K) [application/octet-stream]
Salvataggio in: "phix.fa"

phix.fa             100%[===================>]   5,40K  --.-KB/s    in 0s

2019-10-16 16:17:37 (195 MB/s) - "phix.fa" salvato [5528/5528]
```

In [2]:
```
def ReadGenome(filename):

    '''
    Open and read a FASTA, concatenating different sequences into a single one
    '''

    genome=''
    with open(filename, 'r') as f:
        for line in f:
            if line[0] != '>':
                genome+=line.rstrip()
    return genome
```

In [3]:
```
genome=ReadGenome('phix.fa')
```

In [4]:
```
len(genome)
```

Out[4]: 5386

In [1]:
```
!wget --no-check https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/ERR266411_1.first1000.fastq
```

```
--2019-10-16 16:31:07--  https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/ERR266411_1.first1000.fastq
Risoluzione di d28rh4a8wq0iu5.cloudfront.net (d28rh4a8wq0iu5.cloudfront.net)... 143.204.10.21, 143.204.10.216, 143.20
4.10.34, ...
Connessione a d28rh4a8wq0iu5.cloudfront.net (d28rh4a8wq0iu5.cloudfront.net)|143.204.10.21|:443... connesso.
Richiesta HTTP inviata, in attesa di risposta... 200 OK
Lunghezza: 254384 (248K) [audio/mpeg]
Salvataggio in: "ERR266411_1.first1000.fastq"

ERR266411_1.first10 100%[===================>] 248,42K   777KB/s    in 0,3s

2019-10-16 16:31:08 (777 KB/s) - "ERR266411_1.first1000.fastq" salvato [254384/254384]
```

In [2]:
```
def ReadFastq(filename):

        '''
        Open and read a 4-line FASTQ. Discard first/third/fourth line for each sequence
        '''

        sequences=[]
        with open(filename, 'r') as f:
                while True:
                        f.readline()
                        seq=f.readline().rstrip()
                        if seq == '':
                                break
                        sequences.append(seq)
                        f.readline()
                        f.readline()
        return sequences
```

In [3]:
```
reads=ReadFastq('ERR266411_1.first1000.fastq')
```

In [4]:
```
len(reads)
```

Out[4]: 1000

In [6]:
```
len(reads[0])
```

Out[6]: 100

CHECK MATCHES (I)

In [10]:
```
def BruteForce(P, T):

    '''
    Simple implementation of Brute Force String Matching
    '''

    assert len(P) <= len(T)
    occurrences = []
    for i in range(len(T)-len(P)+1):
        match = True
        for j in range(len(P)):
            if T[i+j] != P[j]:
                match = False
                break
        if match:
            occurrences.append(i)
    return occurrences
```

In [11]:
```
total=0
matched=0

for read in reads:
    matches=BruteForce(read,genome)
    if len(matches) > 0:
        matched += 1
    total+=1

print('%d / %d reads matched the genome !' %(matched,total))
```

```
7 / 1000 reads matched the genome !
```

Q: Why this number is so small?

A1: Differences between individuals and sequenced reads

A2: Reads orientation

CHECK MATCHES (II)

In [13]:
```
total=0
matched=0
trans = str.maketrans('AaTtGgCcN', 'TtAaCcGgN')

for read in reads:
    matches=BruteForce(read,genome)
    if len(matches) > 0:
        matched += 1
    else:
        revMatches=BruteForce(read[::-1].translate(trans), genome)
        if len(revMatches) > 0:
            matched +=1
    total+=1


print('%d / %d reads matched the genome !' %(matched,total))
```

```
449 / 1000 reads matched the genome !
```

Q: Why this number is (still) so small?

A3: Sequencing errors

CHECK MATCHES (III)

In [14]:
```
total=0
matched=0
trans = str.maketrans('AaTtGgCcN', 'TtAaCcGgN')

for read in reads:
    read=read[:30]
    matches=BruteForce(read,genome)
    if len(matches) > 0:
        matched += 1
    else:
        revMatches=BruteForce(read[::-1].translate(trans), genome)
        if len(revMatches) > 0:
            matched +=1
    total+=1


print('%d / %d reads matched the genome !' %(matched,total))
```

```
932 / 1000 reads matched the genome !
```

Take-home message #2

Sequencing errors occur (% is ~1% in short reads and ~10% in long reads) and exact string matching algorithms are not the method of choice for mapping

4. APPROXIMATE STRING MATCHING ALGORITHMS

Mismatch

```
T: G G A A A A A G A G G T A G C G G C G T T T A A C A G T A G

P:                     G T A A C G G C G
```

Insertion

```
T: G G A A A A A G A G G T A G C - G C G T T T A A C A G T A G

P:                     G T A G C G G C G
```

Deletion

```
T: G G A A A A A G A G G T A G C G G C G T T T A A C A G T A G

P:                     G T - G C G G C G
```

Given T and P, with |T| = |P|, the Hamming Distance is the minimum #substitutions required to turn one into the other

T: G G A A A A A G A G G T A G C G G C G T T T A A C A G T A G

P: G G T T A A A G A G G G A G C G G C G T T T T A C A G T A G

Hamming distance = 4

Given T and P, the Edit Distance (aka, Levenshtein Distance) is the minimum #edits (substitutions, insertions or deletions) required to turn one into the other

T: G G A A A A A G A G G T A G C G G C G T T T A A C A G T A G

P: G G T A A A A G A G G T A G C G G C G T T T A C A G T A G

Edit distance = 2

Q: What is the relationship between Hamming and Edit Distance if |T| = |P| ?

T: G C G T A T G C G G C T A A C G C

P: G C T A T G C G G C T A T A C G C

T: G C G T A T G C G G C T A A C G C

P: G C T A T G C G G C T A T A C G C

Hamming distance = 10

T: G C G T A T G C G G C T A - A C G C

P: G C - T A T G C G G C T A T A C G C

Edit distance = 2

A: Edit Distance (P,T) <= Hamming Distance (P,T)

Q: What is the minimum Edit Distance if |P| ≠ |T| ?

A: Edit Distance (P,T) >= ||T| - |P||

```
T: G C T A T A C
   0 1 2 3 4 5 6
P: G C G T A T G C
   0 1 2 3 4 5 6 7
```

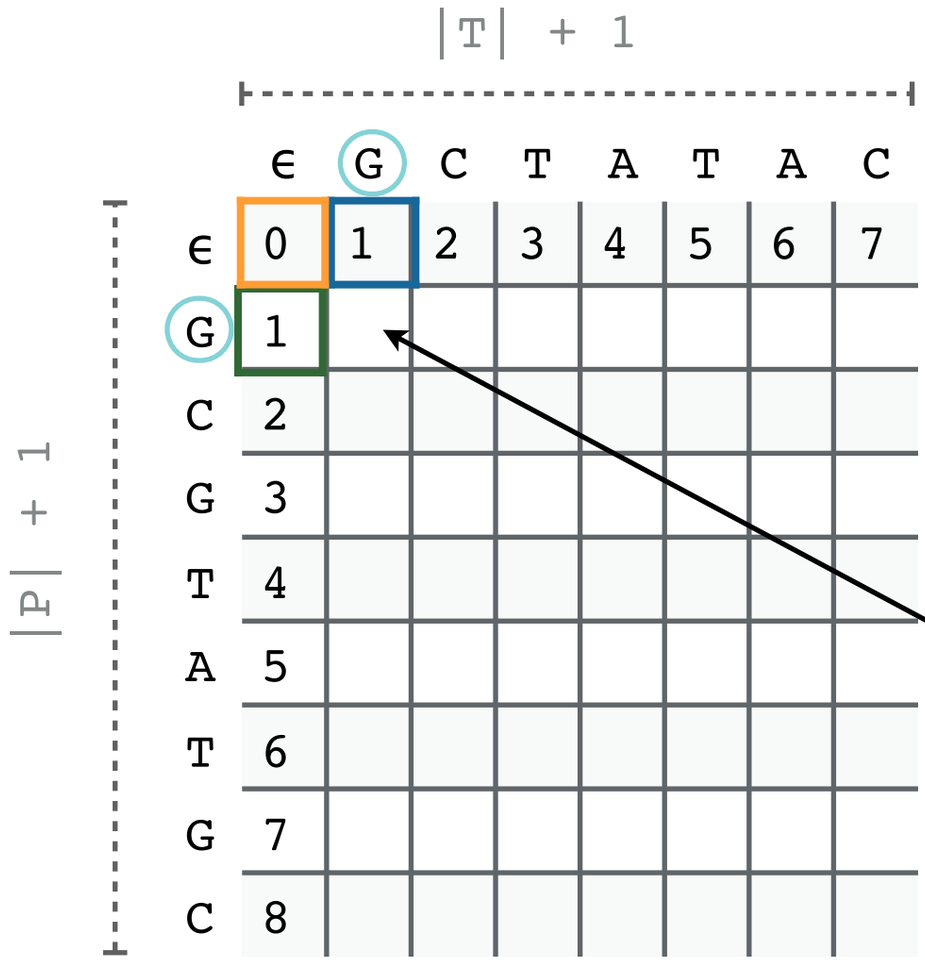


Table D

```
D[i,j] = min(
    D[i-1,j]+1,
    D[i,j-1]+1,
    D[i-1,j-1]+∂,
    )

∂=1 if T[i-1] ≠ P[j-1] else 0

N.B: i are indexes for T
     j are indexes for P


D[1,1] = min(
    1+1,
    1+1,
    0+0,
    ) = 0
```

T: G C T A T A C

P: G C G T A T G C

	ϵ	G	C	T	A	T	A	C
ϵ	0	1	2	3	4	5	6	7
G	1	0	1	2	3	4	5	6
C	2	1	0	1	2	3	4	5
G	3	2	1	1	2	3	4	5
T	4	3	2	1	2	2	3	4
A	5	4	3	2	1	2	2	3
T	6	5	4	3	2	1	2	3
G	7	6	5	4	3	2	2	3
C	8	7	6	5	4	3	3	2

Edit Distance between T and P

T: T A T T G G C T A T A C G G T T

P: G C G T A T G C

T: T A T T G G C - T A T A C G G T T

P: G C G T A T G C

Initialize first row with 0s

Apply the Edit Distance formula

		ϵ	T	A	T	T	G	G	C	T	A	T	A	C	G	G	T	T
	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	G	1	1	1	1	1	0	0	1	1	1	1	1	1	0	0	1	1
M	C	2	2	2	2	2	1	1	0	1	2	2	2	1	1	1	1	2
I	G	3	3	3	3	3	2	1	1	1	2	3	3	2	1	1	2	2
M	T	4	3	4	3	3	3	2	2	1	2	2	3	3	2	2	1	2
M	A	5	4	3	4	4	4	3	3	2	1	2	2	3	3	3	2	2
M	T	6	5	4	3	4	5	4	4	3	2	1	2	3	4	4	3	2
S	G	7	6	5	4	4	4	5	5	4	3	2	2	3	3	4	4	3
M	C	8	7	6	5	5	5	5	5	5	4	3	3	2	3	4	5	4

Trace back the matrix using the minimum parental value

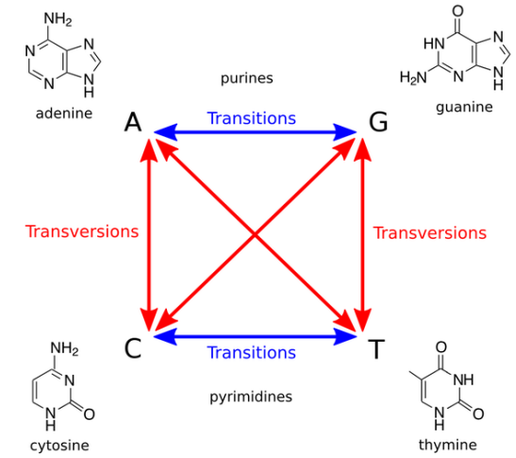
Find the minimum value in the last row

Edit Distance-based approximate string matching penalize different kind of edits we may find with the same amount: that is, there is no difference between a substitution and a deletion or different kind of substitutions

Q: Is this correct?

	AFR		AMR		EAS		EUR		SAS	
Samples	**661**		**347**		**504**		**503**		**489**	
Mean coverage	**8.2**		**7.6**		**7.7**		**7.4**		**8.0**	
	Var. sites	**Singletons**	**Var. sites**	**Singletons**	**Var. sites**	**Singletons**	**Var. sites**	**Singletons**	**Var. sites**	**Singletons**
SNPs	4.31M	14.5k	3.64M	12.0k	3.55M	14.8k	3.53M	11.4k	3.60M	14.4k
Indels	625k	-	557k	-	546k	-	546k	-	556k	-
Large deletions	1.1k	5	949	5	940	7	939	5	947	5
CNVs	170	1	153	1	158	1	157	1	165	1
MEI (Alu)	1.03k	0	845	0	899	1	919	0	889	0
MEI (L1)	138	0	118	0	130	0	123	0	123	0
MEI (SVA)	52	0	44	0	56	0	53	0	44	0
MEI (MT)	5	0	5	0	4	0	4	0	4	0
Inversions	12	0	9	0	10	0	9	0	11	0

A global reference for human genetic variation



Transition/Transversion ~2.1

A: No. We need something similar to the Edit Distance-based approximate string matching but with personalized penalties (that is, give an higher penalty to deletions rather than to substitutions or an higher penalty to transversions rather than to transitions). Global Alignment (aka, Needleman-Wunsch algorithm) and Local Alignment (aka, Smith-Waterman algorithm) serve the purpose.

	A	C	G	T	–
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
–	8	8	8	8	

Penalties:

0 for matches

8 for gaps

2 for transitions

4 for transversions

```
T: T A T G T C A T G C
   0 1 2 3 4 5 6 7 8 9
P: T A C G T C A G G
   0 1 2 3 4 5 6 7 8
```

	A	C	G	T	–
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
–	8	8	8	8	

Initialize first row and first column with gap penalties

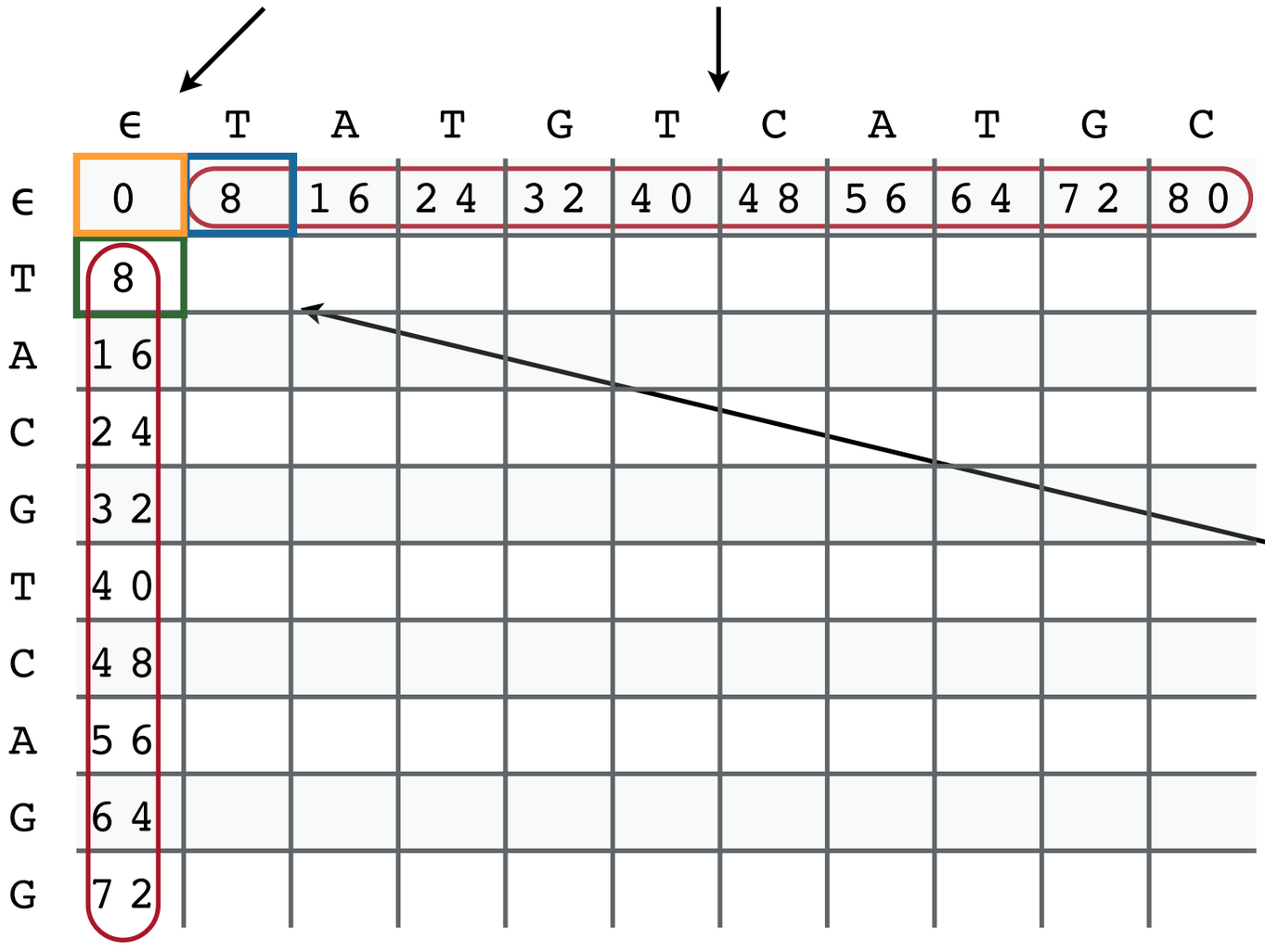


Table D

Apply the formula:

```
D[i,j] = min(
     D[i-1,j]+p(T[i-1],'-'),
     D[i,j-1]+p('-',P[j-1]),
     D[i-1,j-1]+p(T[i-1],P[j-1),
     )
```

N.B i are indexes of T
   j are indexes of P

```
D[1,1] = min(
     8+8,
     8+8,
     0+0,
     ) = 0
```

T: T A T G T C A T G C

P: T A C G T C A G G

T: T A T G T C A T G C

P: T A C G T C A - G G

	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	

		ϵ	T	A	T	G	T	C	A	T	G	C
	ϵ	0	8	16	24	32	40	48	56	64	72	80
M	T	8	0	8	16	24	32	40	48	56	64	72
M	A	16	8	0	8	16	24	32	40	48	56	64
S	C	24	16	8	2	10	18	24	32	40	48	56
M	G	32	24	16	10	2	10	18	26	34	40	48
M	T	40	32	24	16	10	2	10	18	26	34	42
M	C	48	40	32	24	18	10	2	10	18	26	34
M	A	56	48	40	32	26	18	10	2	10	18	26
DM	G	64	56	48	40	32	26	18	10	6	10	18
M	G	72	64	56	48	40	34	26	18	12	10	10

Start to trace back from bottom rightmost cell

Given 2 strings (T and P), Local Aligment does not try to find the alignment of P to T but try to identify the substring of T and the substring of P which are the most similar to each other.

T: he will after his sour fashion tell you

P: struts and frets his hour upon the stage

T: he will after **his sour** fashion tell you

P: struts and frets **his hour** upon the stage

	A	C	G	T	-
A	2	-4	-4	-4	-6
C	-4	2	-4	-4	-6
G	-4	-4	2	-4	-6
T	-4	-4	-4	2	-6
-	-6	-6	-6	-6	

Rewards:

2 for matches

-6 for gaps

-4 for substitutions

```
T: T A T A T G C G G C G T T T
   0 1 2 3 4 5 6 7 8 9 …
P: G G T A T G C T G G C G C T A
   0 1 2 3 4 5 6 7 8 9 …
```

	A	C	G	T	-
A	2	-4	-4	-4	-6
C	-4	2	-4	-4	-6
G	-4	-4	2	-4	-6
T	-4	-4	-4	2	-6
-	-6	-6	-6	-6	

Initialize first row and first column with 0s

	ϵ	T	A	T	A	T	G	C	G	G	C	G	T	T	T
ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0														
G	0														
T	0														
A	0														
T	0														
G	0														
C	0														
T	0														
G	0														
G	0														
C	0														
G	0														
C	0														
T	0														
A	0														

Table D

Apply the formula:

```
D[i,j] = max(
    D[i-1,j]+r(T[i-1],'-'),
    D[i,j-1]+r('-',P[j-1]),
    D[i-1,j-1]+r(T[i-1],P[j-1),
    0
    )
```

N.B i are indexes of T
   j are indexes of P

```
D[1,1] = min(
    0+(-6),
    0+(-6),
    0+(-4),
    0
    ) = 0
```

T: T A T A T G C G G C G T T T T: T A T A T G C - G G C G T T T

P: G G T A T G C T G G C G C T A P: G G T A T G C T G G C G C T A

	A	C	G	T	-
A	2	-4	-4	-4	-6
C	-4	2	-4	-4	-6
G	-4	-4	2	-4	-6
T	-4	-4	-4	2	-6
-	-6	-6	-6	-6	

Stop to trace back when reach 0

		ϵ	T	A	T	A	T	G	C	G	G	C	G	T	T	T
	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G	0	0	0	0	0	0	2	0	2	2	0	2	0	0	0
	G	0	0	0	0	0	0	2	0	2	4	0	2	0	0	0
M	T	0	2	0	2	0	2	0	0	0	0	0	0	4	2	2
M	A	0	0	4	0	4	0	0	0	0	0	0	0	0	0	0
M	T	0	2	0	6	0	6	0	0	0	0	0	0	2	2	2
M	G	0	0	0	0	2	0	8	2	2	2	0	2	0	0	0
M	C	0	0	0	0	0	0	2	10	4	0	4	0	0	0	0
I	T	0	2	0	2	0	2	0	4	6	0	0	0	2	2	2
M	G	0	0	0	0	0	0	4	0	6	8	2	2	0	0	0
M	G	0	0	0	0	0	0	2	0	2	8	4	4	0	0	0
M	C	0	0	0	0	0	0	0	4	0	2	10	4	0	0	0
M	G	0	0	0	0	0	0	2	0	6	2	4	12	6	0	0
	C	0	0	0	0	0	0	0	4	0	2	4	6	8	2	0
	T	0	2	0	2	0	2	0	0	0	0	0	0	8	10	4
	A	0	0	4	0	4	0	0	0	0	0	0	0	2	4	6

Start to trace back from cell with the highest value

Q: Is there a unique way of performing Global and Local alignments ?

A: No. It depends on the values we use in penalty and score matrixes. Moreover, usually there is a different score for gap-opening and gap-extending (the cost for gap-extending is usually lower)

Q: What is the time complexity of dynamic programming ?

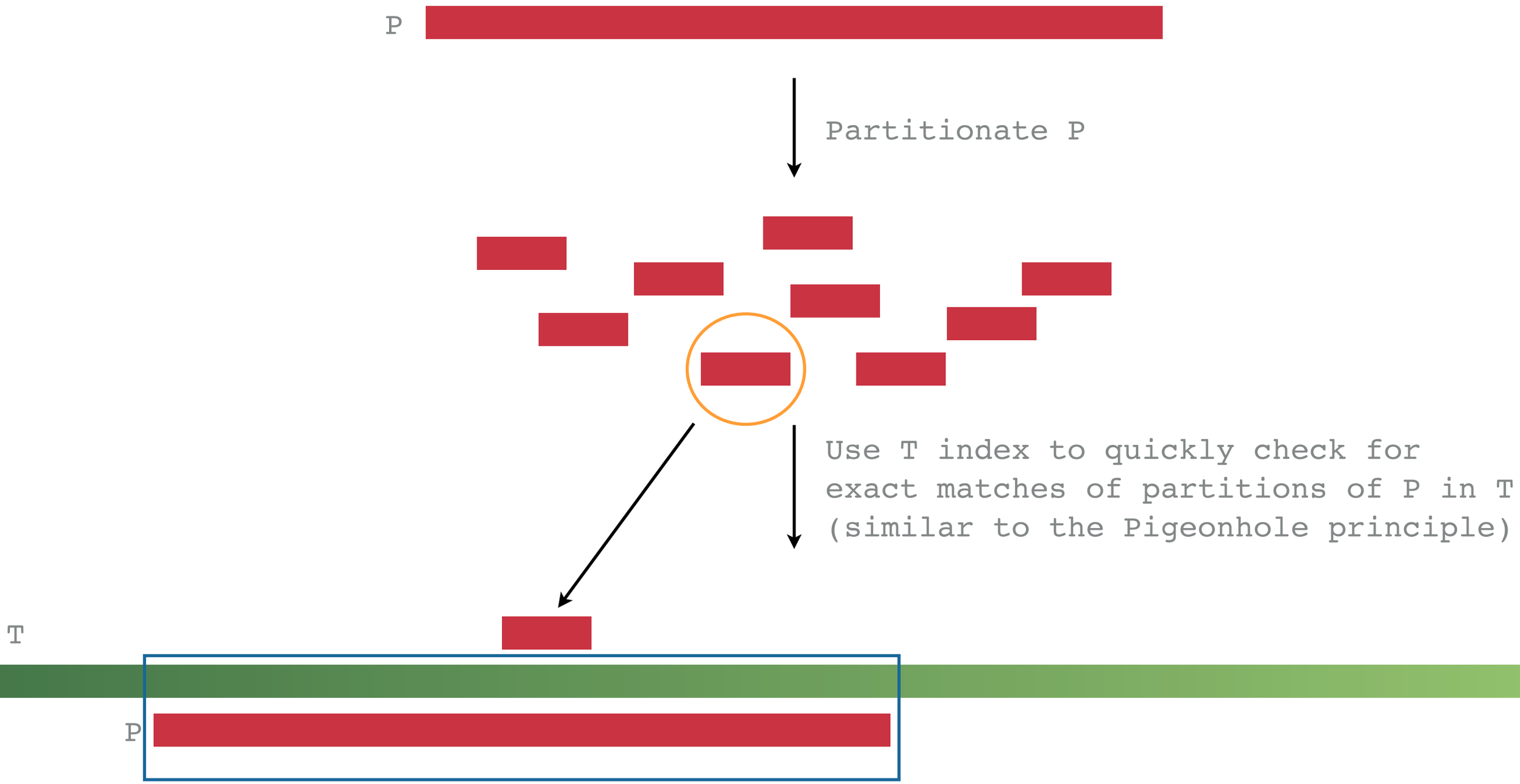
A: O(mn)

Q: What is the time complexity of dynamic programming ?

A: O(mn)

Q: Can Global and Local alignments algorithms be applied as described here ?

A: No. The human reference genome ≈ $3*10^9$ nucleotides (m). 1-week long run of an Illumina HiSeq 4000 produce ≈ $6*10^9$ reads (d), 100 nucleotides each (n). Time (and space, even if we can reduce this somehow) are then O(mnd). With 1000 processors, 3 GHz each, this will require ≈ 2 years.


P
Partitionate P
Use T index to quickly check for
exact matches of partitions of P in T
(similar to the Pigeonhole principle)
T
P
Perform dynamic programming using P
and T in the vicinity of the match

Take-home message #3

Index-assisted global and local alignment algorithms combine dynamic programming's efficacy (that is, they produce the results we want) and indexes' efficiency (that is, they do not waste too much time doing what they do) and are the method of choice for mapping the reads to the genome

5. TEACHING MATERIAL

1. Email me @:
   davidebolognini7@gmail.com

2. Get slides (.key + .pdf) from GitHub:
   https://github.com/davidebolo1993/Classes

THAT'ALL, FOLKS !