Comp 555 - BioAlgorithms - Spring 2020



- PROBLEM SET #2 IS DUE TONIGHT
- PROBLEM SET #3
 SHOULD BE
 POSTED BY
 TONIGHT
- MIDTERM IS SET FOR MARCH S

Multi-String BWTs

MSBWT



A BWT of a string collection instead of just a single string

- Earliest: Mantaci et al. (2005), used concatenation approach
- Bauer et al. (2011) proposed version we will discuss today

Analogy:

- Instead of searching for a substring within a single book, search every book of a library
 - Each book has it's own text, suffix array, and end-of-text delimiter
 - Searching allows us to find how many times a substring appears and in which texts

Bioinformatics?

- Search all genomes? You could, but that's not the main application.
- Search multiple chromosomes of an organism? You should, but even that is not the killer app

Naive Construction



- Create all rotations for all strings in the collection
- Sort all rotations together (Suffix Array)
- Store the predecessor of each suffix
- Strings are "cyclic"
- The predecessor is always from the same string
- Impossible to "jump" from one string to another
- Strings can have different lengths

| Sorted | MSBWT |
|--------|--|
| \$ACCA | Α |
| \$CAAA | Α |
| A\$ACC | С |
| A\$CAA | Α |
| AA\$CA | Α |
| AAA\$C | С |
| ACCA\$ | \$ |
| CA\$AC | Ċ |
| CAAA\$ | \$ |
| CCA\$A | Á |
| | |
| | |
| | Sorted \$ACCA \$CAAA A\$ACC A\$CAA AA\$CA AAA\$C ACCA\$ CA\$AC CAAA\$ CCA\$A |

MSBWT's FM-index

Identical Definition

- Find k-mer "CA"
- Initialize to full range (") •
- lo, hi = 0, 10•
- Find occurrences of 'A'
 - lo = Offset[A] + FMindex[lo][A] = 2 + 0 = 2Ο
 - hi = Offset[A'] + FMindex[hi][A'] = 2 + 5 = 7Ο
- Find occurrences of "CA" •
 - lo = Offset['C'] + FMindex[lo]['C'] = 7 + 0 = 7Ο
 - hi = Offset['C'] + FMindex[hi]['C'] = 7 + 2 = 90
- Searching and extracting suffixes are identical to a BWT

| | | Г | ш - т | nue | ~ |
|--------|---|---|---|---|---|
| Sorted | MSBW | Т | \$ | А | С |
| \$ACCA | Α | 0: | 0 | 0 | Θ |
| \$CAAA | Α | 1: | 0 | 1 | 0 |
| A\$ACC | С | 2: | 0 | 2 | 0 |
| A\$CAA | Α | 3: | 0 | 2 | 1 |
| AA\$CA | Α | 4: | 0 | 3 | 1 |
| AAA\$C | С | 5: | 0 | 4 | 1 |
| ACCA\$ | \$ | 6: | 0 | 4 | 2 |
| CA\$AC | C | 7: | 1 | 4 | 2 |
| CAAA\$ | \$ | 8: | 1 | 4 | 3 |
| CCA\$A | A | 9: | 2 | 4 | 3 |
| | | 10: | 2 | 5 | 3 |
| | | Offset: | Θ | 2 | 7 |
| | Sorted \$ACCA \$CAAA A\$ACC A\$CAA AA\$CA AAA\$C ACCA\$ CA\$AC CA\$A | SortedMSBW\$ACCAA\$CAAAAA\$ACCCA\$CAAAA\$CAAAAA\$CAAAA\$CAAAA\$CASCA\$ACCCA\$ACCCA\$AS\$CCA\$AA | Sorted MSBWT \$ACCA A 0: \$CAAA A 1: A\$ACC C 2: A\$CAA A 3: AA\$CA A 3: AA\$CA A 4: AAA\$C C 5: ACCA\$ \$ 6: CA\$AC C 7: CAAA\$ \$ 8: CCA\$A A 9: 10: Offset: | Sorted MSBWT \$ \$ACCA A 0:0 0 \$CAAA A 1:0 0 A\$ACC C 2:0 0 A\$CAA A 3:0 0 A\$CAA A 3:0 0 AA\$CA A 4:0 0 AA\$C C 5:0 0 ACCA\$ \$ 6:0 0 CA\$AA\$ \$ 8:1 1 CCA\$A A 9:2 10:2 0ffset:0 0 0 10:2 | Sorted MSBWT \$ A \$ACCA A 0: 0 \$CAAA A 1: 0 1 A\$ACC C 2: 0 2 A\$CAA A 3: 0 2 A\$CAA A 3: 0 2 A\$CA A 3: 0 2 AA\$CA A 4: 0 3 CA\$A\$A\$ \$ 8: 1 4 CCA\$A A 9: 2 4 10: 2 5 |



EM indox



Incremental MSBWT Construction

- A key tool missing from the BWTs toolbox-adding new strings to an existing msBWT
- You could reconstruct the suffix array of the msBWT using suffix(i, fmindex) for all i, and then insert the suffixes of the new string.
- Variant of find(); Find the insertion point of new string's jth suffix, s_i
- Add last character to msBWT
- Update the FMindex



Our original BWT code



```
In [8]: def FMIndex(bwt):
            fm = [{c: 0 for c in bwt}]
            for c in bwt:
                row = {symbol: count + 1 if (symbol == c) else count for symbol, count in fm[-1].items()}
                fm.append(row)
            offset = {}
            N = 0
            for symbol in sorted(row.keys()):
                offset[symbol] = N
                N += row[symbol]
            return fm, offset
        def recoverSuffix(i, BWT, FMIndex, Offset):
            suffix = ''
            c = BWT[i]
            predec = Offset[c] + FMIndex[i][c]
            suffix = c + suffix
            while (predec != i):
                c = BWT[predec]
                predec = Offset[c] + FMIndex[predec][c]
                suffix = c + suffix
            return suffix
        def findBWT(pattern, FMIndex, Offset):
            lo = 0
            hi = len(FMIndex) - 1
            for symbol in reversed(pattern):
                lo = Offset[symbol] + FMIndex[lo][symbol]
                hi = Offset[symbol] + FMIndex[hi][symbol]
            return lo, hi
```

Inserting a new BWT into an existing msBWT



```
0: $ACAT
In [9]:
          1 # Constructing a multistring BWT one suffix at a time
          2 # first let's take a look at the implicit suffix array
                                                                            1: $ATAG
            bwt = "TGAG$TGC$AAA$AA"
                                                                            2: $GAGA
          3
          4 fm, off = FMIndex(bwt)
                                                                            3: A$GAG
          5 for i in range(len(bwt)):
                                                                            4: ACAT$
                print("%2d: %s" % (i, recoverSuffix(i, bwt, fm, off)))
          6
                                                                            5: AG$AT
            print()
                                                                            6: AGA$G
          8
                                                                            7: AT$AC
          9
            # New string to include
                                                                            8: ATAG$
            new = "TATA$"
         10
            inserts = []
                                                                            9: CAT$A
         11
            for i in range(len(new)):
         12
                                                                           10: G$ATA
         13
                rotation = new[i:]+new[:i]
                                                                           11: GA$GA
                1, h = findBWT(rotation, fm, off)
         14
                                                                           12: GAGA$
                inserts.append((h,new[i-1],rotation))
         15
                                                                           13: T$ACA
             print(inserts)
         16
                                                                           14: TAG$A
         17
            # Insert into original BWT in reverse order
            for i, c, rot in sorted(inserts, reverse=True):
         19
                bwt = bwt[:i] + c + bwt[i:]
         20
         21
            # Look at result
            print(bwt)
         23
            print()
         24
         25 fm, off = FMIndex(bwt)
         26 for i in range(len(bwt)):
                print("%2d: %s" % (i, recoverSuffix(i, bwt, fm, off)))
         27
```

Before and After



[(15, '\$', 'TATA\$'), (8, 'T', 'ATA\$T'), (14, 'A', 'TA\$TA'), (4, 'T', 'A\$TAT'), (3, 'A', '\$TATA')] TGAAGT\$TGCT\$AAA\$AAA\$ 0: \$ACAT 0: \$ACAT 1: \$ATAG 1: \$ATAG 2: \$GAGA 2: \$GAGA 3: A\$GAG 3: \$TATA 4: ACAT\$ 4: A\$GAG 5: AG\$AT That was a little tricky... 6: AGA\$G 6: ACAT\$ 7: AT\$AC 7: AG\$AT 8: ATAG\$ 8: AGA\$G 9: CAT\$A 9: AT\$AC 10: G\$ATA 🔶 10: ATA\$T 11: GA\$GA 11: ATAG\$ 12: GAGA\$ 12: CAT\$A 13: T\$ACA 13: G\$ATA 14: GA\$GA 14: TAG\$A 15: GAGA\$ 16: T\$ACA → 17: TA\$TA 18: TAG\$A 19: TATA\$

Merging msBWTs

- BETTER YET! Rather than inserting new strings, build a BWT of the new strings and merge the new and old BWTs
- Suffixes of BTWs are already sorted
- BTWs are interleaved
- In the worse case (ties) the entire suffix must be considered, but general the longest common prefix of suffixes is smaller
- Minimal overhead
- Well suited for divide an conquer approaches (like merge sort)
- Easy to merge multiple data sets!
- Compression improves!



Merging Steps



msBWT merging alternates between sorting and interleaving

- 1. Consider the BWTs as a tuple of (character, BWTid) pairs
- 2. Sort these tuples
- Based on the BWTids after the sort, select a new character For each tuple from the original msBWTs
- 4. Repeat from Step 2 until the sort is stable
- 5. The resulting characters are the merged msBWT
- Number of passes is proportional to largest LCP value.

| | P | Pass #1 | I | Pass #2 | 1 | Pass #3 | Pa | ass #3 | |
|--------------|-----------------|-----------------------|-----------------|-----------------------|-----------------|-----------------------|-----------------|-----------------------|----------------|
| ГТ,0 | \$,0 | Т, 0 | \$,0 | Т, 0 | \$,0 | Т, 0 | \$,0 | Τ, Θ | \$ACAT |
| G, 0 | \$,0 | G, 0 | \$,0 | G, 0 | \$,0 | G, 0 | \$,0 | G, 0 | \$ATA G |
| \$,0 | \$,1 | A,1 | \$,1 | A,1 | \$,1 | A, 1 | \$,1 | A, 1 | \$GAGA |
| Т, 0 | \$,1 | A,1 | \$,1 | A, 1 | \$,1 | A,1 | \$,1 | A, 1 | \$TAT A |
| BWT C,0 | A,0 | \$,0 | A,1 | G, 1 | A, 1 | G,1 | A, 1 | G, 1 | A\$GA G |
| \$,0 | A, 0 | Τ,Θ | A,1 | T,1 | A, 1 | T,1 | A, 1 | T, 1 | A\$TA T |
| A, 0 | A,0 | C,0 | A,0 | \$,0 | A, 0 | \$,0 | A, 0 | \$,0 | ACAT\$ |
| A,0 | A,0 | \$,0 | A,0 | Т,0 | A, 0 | Т, 0 | A,0 | Τ, Θ | AG\$AT |
| A,0 | A,1 | G,1 | A,1 | G, 1 | A, 1 | G,1 | A,1 | G,1 | AGA\$G |
| LA,0 | A, 1 | T,1 | A,0 | C,0 | A, 0 | C,0 | A,0 | C,0 | AT\$AC |
| A,1 | A, 1 | G, 1 | A, 0 | \$,0 | A, 0 | \$,0 | A, 1 | T, 1 | ATA\$ T |
| A,1 | A, 1 | Τ,1 | A, 1 | Τ,1 | A, 1 | Τ,1 | A,0 | \$,0 | ATAG\$ |
| G, 1 | С,0 | Α,Θ | C,0 | Α,Θ | C,0 | Α,Θ | CØ | A, 0 | CAT\$A |
| T,1 | G, 0 | Α,Θ | G, 0 | A,0 | G, 0 | Α,Θ | G, 0 | A, 0 | G\$ATA |
| BWT, G, 1 | G, 1 | A,1 | G,1 | A,1 | G, 1 | A, 1 | G, 1 | A, 1 | GA\$G A |
| T,1 | G, 1 | \$,1 | GAGA\$ |
| A, 1 | Т, 0 | Α,Θ | Т,0 | Α,Θ | Т,0 | Α,Θ | Т,0 | Α,Θ | T\$ACA |
| \$,1 | Т,0 | A,0 | Т,0 | Α,Θ | T, 1 | A, 1 | Т,1 | A, 1 | TAŜTA |
| A, 1 | T, 1 | A,1 | T,1 | A, 1 | Т, 0 | Α,Θ | Т,0 | A, 0 | TAG\$A |
| └\$,1 | Τ,1 | \$,1 | T, 1 | \$,1 | T, 1 | \$,1 | T,1 | \$,1 | TATA\$ |
| | Sort by base | Select from BWT | Done! |

In Python



```
In [12]: M def mergeBWT(bwt1, bwt2):
                 interleave = [(c, 0) for c in bwt1] + [(c, 1) for c in bwt2]
                 passes = min(len(bwt1), len(bwt2))
                 for p in range(passes):
                     i, j = 0, 0
                     nextInterleave = []
                     for c, k in sorted(interleave, key=lambda x: x[0]):
                         if (k == 0):
                             b = bwt1[i]
                             i += 1
                          else:
                              b = bwt2[j]
                             i += 1
                         nextInterleave.append((b, k))
                     if (nextInterleave == interleave):
                          break
                     interleave = nextInterleave
                 return ''.join([c for c, k in interleave])
             bwt1 = "TG$TC$AAAA"
             bwt2 = "AAGTGTA$A$"
             bwt12 = mergeBWT(bwt1, bwt2)
             print(bwt12)
             FM, Offset = FMIndex(bwt12)
             for i in range(len(bwt12)):
                 j = (i >> 2) + (i \otimes 3) * (len(bwt 12)//4)
                 print("%2d: %s" % (j, recoverSuffix(j,bwt12,FM,Offset)), "\n" if (i % 4 == 3) else "", end='')
```

TGAAGT\$TGCT\$AAA\$AAA\$

0: \$ACAT 5: A\$TAT 10: ATA\$T 15: GAGA\$ 1: \$ATAG 6: ACAT\$ 11: ATAG\$ 16: T\$ACA 2: \$GAGA 7: AG\$AT 12: CAT\$A 17: TA\$TA 3: \$TATA 8: AGA\$G 13: G\$ATA 18: TAG\$A 4: A\$GAG 9: AT\$AC 14: GA\$GA 19: TATA\$

MSBWT Applications



- Instead of building a BWT of a reference genome, build a MSBWT of every sequenced reads
- Arbitrary exact-match k-mer queries
- O(k) time
- Enables fast searches/counting
- Recover an arbitrary read of length L from MSBWT
- O(L) time
- Enables extraction of user-selected reads

Compression of high-throughput sequencing



- Using Run-length encoding again
- Reasons we expect compression:
 - True genomic repeats: gene families, long repeats, etc.
 - Over-sampling: 30x coverage means we expect 30 copies of every k-mer pattern
- Sequencing errors may break up runs
- Technical errors may cause biases for or against a particular pattern
- Real Mouse DNA-seq:
 - 368654191 × 151 × 2 = ~112 Giga-bases
 - Compresses to ~15.3 GB using RLE (1.09 bits/base)
- Real Mouse RNA-seq:
 - ~8.9 Giga-bases
 - ~1.2 GB using RLE (1.05 bits/base)

K-mer Search & Read Extraction



Basic Use:

CC Genome: CC001M4363_UNC_NYGC 799,869,982 strings with 121,580,237,264 bases and index size of 16,944,071,172 bytes (1.11 bits per base) Target: GACAGGAATGTCGAAATTACTGAAAAACGTGAAAAATGAGAAATG

Found 19 times (8 forward, 11 reverse-complemented)

| algacalggggacalggggacalggggacalgagggacalggggacalggggacalggggacalggggacalggggacalggggacalggggacalggggacalggggacalggggacalggggacalgagggacalgggacalgggggacalgggacalgggggacalggggacalggggacalggggacalggggacalggggacalggggacalggggacalggggacalgggggacalgggacalgggacalgggacalgggacalgggacalgggacalggggacalggggacalggggacalggggacalgggacalgggacalgggacalgggacalgggacalgggacalgggagacalgggagacalgggagacalgggacalgggacalgggagacalgggagacalgggagacalgggagacalgggagacalgggagacalgggagacalggagacalggagacalggagacalggagacalggagacalggagacalggagacalggag |
|---|
| |
| .sagaaggaccaaggggattccatgacatgggGACAGGAATGTCGAAAAACGTGAAAAACGAGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAAATGAGAAAATGAGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAATGACGAAAATGACGAAGAACGTGAAAATGACGAATGACGAAGACGACGACGACGGAAATGACGAAGAACGGAAATGACGAAATGACGAACGA |
| .\$ca <mark>a</mark> attcgacactg <mark>g</mark> caagaagggaccaaggggattccatggcGACAGGAATGCCGAAAAACGTGAAAAACG |
| |
| agcggtgactgtcagattcgacatggcaagagggactagggggattccatggcacagggggattccatggacatgggGACAGGAATGTCGAAAAACGTGAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAACGTGAAAACGTGAAAACGTGAAAACGTGAAAACGTGAAAACGTGAAAACGTGAAAAACGTGAAAACGTGAAAACGTGAAAAACGTGAAAACGTGAAAACGTGAAAACGTGAAAACGTGAAAACGTGAAAAACGTGAAAACGTGAAAAACGTGAAAACGTGAAAACGTGAAGAAGTGACGAAGTGACGAAGTGACGAAGTGACGAAGTGACGAAGTGACGAAGTGACGAAGTGACGAAATTACTGAAAAACGTGAAAATTGAGAAATGAGAAATGGCACAGTGAGAAGTGGAAAGTGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAAATGAGAAAATGAGAAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAG |
| |
| agagcggtgactgtcagattcgacactgtcaagaagggaccaaggggattccatggacatgggGACAGGAATGTCGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAGAAATGACGAAGAACTGAGAAATGACGAAGAACGTGAGAAATGACGAGAACTGAGAAATGACGAAAACGTGAGAAATGACGAGAACTGAGAAATGACGAGAACTGAGGAACTGAGGAAATGACGAGAACTGAGAAATGACGAGAACTGAGAAATGACGAGAAATGACGAGAACTGAGAGAACTGAGAGAACTGAGAGAACTGAGAGAACTGAGGAGAACTGAGAGAACTGAGGAGAACTGAGAGAGCGAGAAATGACGAGGAGACTGAGAGAGA |
| |
| |
| |
| |
| SacagcacagggataagaagcaggagcaggtgactgtcaagattcgacatggcactaggggattccatgggGACAGGAAATGCCGAAAAATGCAGAAAATGCacactgcaggacctggaatatggc |
| \$acggcacaggataagaagcagagcggtgactgtcagattcagaattcgacatggggattccatggcacagggggattccatggcaCAGGAATGTCGAAAAACGGGAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAAATGACGAAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAAATGACGAAATGACGAAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAAATGACGAA |
| \$ccttagacggcacagggataagaagcagagcggtgactgtcagattcagaattcgacatggggattccatggcacAggggAttccatgacatgggGACAGGAATGTCGAAAAACGGGAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAATGACGAAAATGACGAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAAATGACGAAAATGACGAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAAATGACGAAATGACGAAATGACGAAAATGACGAAAATGACGAAATGACGAACACTGACGACGACGTGACGACGACGGAACTGACGACGACGACGGAACTGACGACGACGACGACGACGACGACGACGACGACGACGACG |
| |
| StocttagacggcacagggataagaagcaggggtgactgtcagattcgacatgtcaagaaggggctcacagggggattccatgacatgggGACAGGAATGTCGAAAAACGTGAAAACGTGAAAAACGTGAAAACGTGAAAACGTGAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAAACGTGAAAACGTGAAAACGTGAAAAACG |
| |
| tgcgcctgtccttagacggcacaggacaggataagaagcaggggtgactgtcgactgtcaagaagggattccatggcattgccatgggGACAGGAATGTCGAAAAACGTGAAAAAACGTGAAAAAC |
| TEGECCTGTCCTTAGACGGCACAGGGATAAGAAACTGACAGGGGGTGACTGTCAGAATAGGGGACAGGGGATTCGAGAGAATGGGGACAGGGACAGGGACAGGGACAGGGACAGGACATGGGGAAAAATGAGAAAATGAGAATGAGAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAATGAGAATGAGAATGAGAATGAGAATGAGAATGAGAAATGAGAAATGAGAATGAGAGAATGAGAATGAGAAGA |

Green: query k-mer. Red: forward reads. Blue: reverse-complement reads. Yellow: sequencing errors

- Search for all reads with a given k-mer
- Extract all reads with that k-mer and its reverse-complement
- Build a consensus

Moving over a little bit



CC Genome: CC001M4363_UNC_NYGC 799,869,982 strings with 121,580,237,264 bases and index size of 16,944,071,172 bytes (1.11 bits per base) Target: CGAAATTACTGAAAAACGTGAAAAATGAGAAATGCACACTGCAGGACCTGG Found 46 times (25 forward, 21 reverse-complemented)

| *************************************** |
|--|
| |
| |
| |
| sologa catta aga to t cato a da a a a cato a a a a a a a a a a a a a a a a a a |
| \$aaaatttagaaatgtccactgtagg <mark>aca</mark> tggaatatggcaagaaaactgaaaaactgaaaaatgagaaaaattgagaaaactcacgacatggaaaatgagaaaactgaaaaactg |
| |
| |
| |
| |
| |
| |
| ttgaaaaatgagaaaatgagaaaatgagaaaatgagaaaatgagaaaactgagaaaatgagaa |
| saatcatggaaaatgagaaaatgagaaaatgagaaaatgagaaaatgagaaaatgacGAAATTACTGAAAAACGTGAAAAATGAGAATGGAGAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGGAGAATGA |
| \$aaaatggtaaacatccacttgaagacttgaaaaatgaCGAAATTACTGAAAAACGTGAAAAATGAGAAATGCACACTGCAGGACCTGGaatatggcgagaaaactgaaaatcacggaaaataagggaatacacacatttaggacgtgaaaata |
| |
| |
| |
| |
| |
| tggcaagaaaactgaaaatgagaaaactgaaaatgaggaaaatgagaaaatg |
| ggaaaatttagaaatgtccactgtagg <mark>a</mark> cgtg <mark>a</mark> aatatggcaagaaaatgagaaaatgagaaaatgagaaaatgagaaaatgagaaaatgagaaaatgagaaaatgacaAAAATACGAAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAATG |
| |
| gg <mark>q</mark> aaatttagaaatgtccactgtagq <mark>a</mark> cgtggaatatggcaagaaaatggaaaatgagaaaatgagaaaatgagaaaatgacGAAAATACGGAAAAATGAGAAAATGAGAAAATGCAGGACCTGGAaGA. |
| StctacactenticstacacgadgetetecegatetaaaaataaaaaaataaaaatgacGAAAATAcTGAAAAAACGTGAAAAAACGTGAAAAAATGAGAAATGAGAAAATGAGAAATGAGAAAATGAAAATGAGAAAATGAAAATGAGAAAATGAAAATGAAAATGAAAATGAGAAAATGAAAATGAGAAAATGAAAATGAAAATGAGAAAATGAAAATGAAAAATGAAAAATGAAAAATGAAAAATGAAAAATGAAAAATGAAAAATGAAAATGAAAAATGAAAAATGAAAATGAAAATGAAAATGAAAATGAAAATTAGAAATTAATGAAAATGAAAAATGAAAAATGAAAAATGAAAATGAAAATGAAAAATGAAAATGAAAA |
| |
| att teacht teacht and teacht and teacht and teacht and teacht att att teacht att att att att att att att att att a |
| all trajada gi con cigragga ci ga ad a tig caga ad ci ga ad a cicca ci ga ga ad a cicca ci ga ad a cicca cicca cicca ci ga ad a cicca ci |
| |
| |
| |
| tq <mark>a</mark> caaqaaaactgaaaatcat <mark>t</mark> qaaaatgaqaaaattgaqaaaatgaqaaaattacttaaaatgaCGAAAATTACTGAAAAAACGTGAAAAAATGAGAAAATGCACACTGCAGGACCTGGAaatatgqcgagaaaatgagaaa\$ |
| SaaatacccaacttattcacgacataaatgaCGAAATTACTGAAAAAACGTGAAAAAACGTGAAAAAATGAGAAAATGCACACTGCAGGAactatggcgagaaatgaggaaggagaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatgaggaaatg |
| |
| |
| |
| |
| |
| a <mark>t gacat ggggacaggaat gt CGAAATTACTGAAAAATGCGAAAATTACTGAAAAATGCAGAAATG</mark> CACACTGCAGGACCTGGGaat at gggcgaaaat ggggaaa ta taggt gaaat at g <mark>a</mark> c gagaaa ta tggagaa ta taggt gaaat at tggagaaa ta tggagaaa ta tggagaaa ta tgacat gacaa taggt gaaat at tgacaa tagacaa tagaca |
| |
| \$aqaaqqaccaaqqqqattccatqqcatqqcatqcaqqqaatqtCGAAAATACGTGAAAAAACGTGAAAAAACGTGAAAAAATGCGGAAAAATGCGGAAAAATGCGAAAAATGCGGAAAAATGCGGAAAATGCGGAAAAATGCGGAAAATGCGGAAAAATGCGGAAAATGCGGAAAATGCGGAAAATGCGGAAAATGCGGGAAAATGCGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGCGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGGAAAATGGGGGAAAATGGGGAGAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAAATGGGGAAGAA |
| |
| St capat t coacact of caagaa do a ccaagaa do a ccaagaa to t cGAAAATACTGGAAAAAACGTGGAAAAAACGTGGAAAAAACGTGGAAGAACATGGACACTGCAGGAACAT CACGACTACCAGGACCTGCAGGAACAT CACGACTACCACACTGCAGGAACAT CACGACAACACGTGGAAAAAACGTGAAAAAACGTGAAAAAACGTGAAAAAACGTGAAAAAAACGTGAAAAAAACGTGGAAAAAACGTGGAAAAAACGTGAAAAAACGTGAAAAAACGTGAAAAAAACGTGAAAAAAACGTGAAAAAAACGTGAAAAAAACGTGAAAAAAAA |
| |
| |
| |
| agaluggina tig caga tuga agga tuga agga tuga agga gu agga agga |
| |
| |
| |
| |
| SacagcacadggataagaagcagaocgqtgactgtcagattcgacatgtcgacatgtcgacatggggattccatgacatgggggatagcaggaatgtCGAAAATGCAAAAACGGAAAAAGGGAAAAAGGGAAAAAGGGAAAAAGGGACCTGGAatatggc |
| |
| GTGGAAAATTTAGAAAATGACAAGAATTTAGAAAATGACAAAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAAATGAGAAAATGAGGAAATGAGAAAATGAGAAATGAGAAATGAGAAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAAATGAGAAAATGAGAAAATGAGAAATGAGAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAAATGAGAAATGAGAAATGAGAAATGAGAAATGAAAATGAGAAATGAGAAATGAAGAA |

Moving over 12 bases shows a different story

Comp 555 - Spring 2020

Reference-based Searches



- Given a reference genome and region of that genome
- Split reference into k-mers
- Count the abundance of each k-mer and plot
- Fast O(k) time per k-mer
- Similar to a post-alignment pileup



Iterative Reference Correction









- Burrows-Wheeler Transform
 - Permutation of characters that represents a suffix array
 - Run-length encoded for compression
- FM-index
 - Derived from BWT
 - Exploits LF-mapping property
 - O(k) search time for arbitrary k-mer, independent of BWT's size
 - Used in many fast aligners
- MSBWT
 - Applies to string collections
 - Enables database-like access to reads via k-mer searches