Research Topics in Bioinformatics: Problems related to Bioinformatics

Roger Luis Uy
College of Computer Studies
De La Salle University
Manila, Philippines

Problems related to Bioinformatics

- Challenge: write a program in {C | C++ | CUDA C |assembly language} either in {Windows | Linux} environment
- Challenge:
  - Correctness
  - Fastest execution time

Problem #1: Counting DNA Nucleotides

- Given: A DNA string *s* of length stored in *Fasta* format. Assume there is only one DNA string in the Fasta file (file: ndna_255.fa).
- Output:
  - Frequency of A
  - Frequency of C
  - Frequency of G
  - Frequency of T
  - Frequency of non-ACGT (usually “N” to denote Not sequence)
  - Total length of the DNA string
- Research Question: What is a Fasta file format?
  - In this format, the string is introduced by a line that begins with '>', followed by some labeling information. Subsequent lines contain the string itself.

Problem #2: Computing GC content

- Given: A DNA string *s* of length stored in *Fasta* format. Assume there is only one DNA string in the Fasta file (file: ndna_255.fa).
- Output:
  - GC-content of the DNA string
- Research Question: What is GC content?
  - The GC-content of a DNA string is given by the percentage of symbols in the string that are 'C' or 'G'. For example, the GC-content of "AGCTATAG" is 37.5%.

Problem #3: Transcribing DNA to RNA

- Given: A DNA string *s* of length stored in *Fasta* format. Assume there is only one DNA string in the Fasta file (file: ndna_255.fa).
- Output:
  - The transcribed RNA string of the DNA string
- Research Question: What is RNA?
  - An RNA string is a string formed from the alphabet containing 'A', 'C', 'G', and 'U'
  - Given a DNA string, its transcribed RNA string is formed by replacing all occurrences of "T" with "U"

Problem #4: Complementing a DNA string

- Given: A DNA string *s* of length stored in *Fasta* format. Assume there is only one DNA string in the Fasta file (file: ndna_255.fa).
- Output:
    - The reverse complement of the DNA string
- Research Question: What is reverse complement?
    - In DNA strings, symbols 'A' and 'T' are complements of each other, as are 'C' and 'G'.
    - The reverse complement of a DNA string is the string formed by reversing the symbols, then taking the complement of each symbol
    - Example: The reverse complement of "GTCA" is "TGAC"

Problem #5: Counting Point Mutations (hamming distance)

- Given: Given two DNA strings *s* and *t* of equal length. The two strings are stored in text format (file: Rosalind_hamm.txt)
- Output:
  - The hamming distance of the two strings
- Research Question: What is hamming distance?
  - Hamming distance – the number of differences at each position of the string. Can be viewed as the number of substitutes to transform from one string to another.
  - Example: The hamming distance of (“GTAGCGGCG”,”GTAACGGCG”) is 1

Problem #6: Finding a Motif in DNA

- Given: Given two DNA strings *s* and *t*, *t* is the substring of *s* if *t* is contained as a contiguous collection of DNA symbols in s (obvious, *t* must be no longer than *s*)The two strings are stored in text format (file: Rosalind_subs.txt)
- Output:
  - All locations of *t* as a substring of *s*
- Research Question: Finding exact match in a string
  - Example:

    s = “GATATATGCATATACTT”

    t = “ATAT”

    Location: 2 4 10 (assume 1-indexing)

Problem #7: Enumerating k-mers Lexicographically

- Given: An integer *k*
- Output:
  - Enumerate all *k*-mers of the DNA alphabet {A,C,G,T}
  - Example: 2

    Output: AA, AC, AG, AT, CA,CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, TT
- Research Question: What is *k*-mer?
  - K-mer is basically a substring of a string. Given a string s= “GATATATGCATATACTT”, we can say that “GATA” is a 4-mer of the string *s;* “CTT” is a 3-mer of the string s, etc.

Problem #8: Enumerating overlapping k-mer

- Given: An integer *k* and a string *s*
- Output:
  - Enumerate all overlapping *k*-mers of the string *s*
  - Example: s= "GATAT", *k*=3

    Output: GAT, ATA, TAT
- Research Question: What is *k*-mer?
  - K-mer is basically a substring of a string. Given a string s= "GATATATGCATATACTT", we can say that "GATA" is a 4-mer of the string *s;* "CTT" is a 3-mer of the string *s*, etc.

Problem #9: Edit distance

- Given: Given two DNA strings *s* and *t* of equal length. The two strings are stored in text format. Assume that all edits {insert | delete | mismatch) has a cost of 1. Hint: use dynamic programming. Ask for user input for *s* and *t*
- Output:
  - The edit distance of two strings
- Research Question: What is edit distance?
  - Please refer to succeeding slides

Levenshtein distance or edit distance

Levenshtein distance or edit distance – the minimal number of insertions, deletions and substitutions to make two strings equal. Also known as “string matching with k differences”

```
T: GGAAGTAGC-GCGCGTT
       ||||| |||
P:     GTAGCGGCG
```

(Insertion)

```
T: GGAAGTAGCGGCGTT
       || ||||||
P:     GT-GCGGCG
```

(deletion)

```
T: GGAAGTAGCGGCGTT
       ||| |||||
P:     GTAACGGCG
```

(substitution/mismatch)

Vladimir I. Levenshtein, "Binary codes capable of connecting deletions, insertions, and reversals," Soviet Physics Doklady, vol. 10, no. 8, pp. 707-710, February 1966.

Levenshtein distance or edit distance

- Levenshtein/edit distance can be implemented using:
  - Dynamic programming (DP) → *O*(mn) time complexity
  - Bit vector algorithm → $O(\left\lceil \frac{m}{w} \right\rceil n)$ time complexity

Levenshtein distance or edit distance

- Using dynamic programming, two matrices can be used:
    - Matrix *H* with size of (*m*+1,*n*+1) is the processing matrix where the distance is computed
    - (optional) Backtrack matrix *B* with the size of (*m*+1,*n*+1) is to keep track the changes of the score to determine the sequence alignment

Levenshtein distance or edit distance

- The DP matrix is calculated as follows:
  1. The first row and column of the DP matrix *H* is initialized as follows:

$$H[0, j] = H[0, j-1] + \sigma$$
$$H[i, 0] = H[i-1, 0] + \sigma$$

Where σ is the gap cost (usually +1 in simple edit distance)

Levenshtein distance or edit distance

- The DP matrix is calculated as follows:
   2. The recursion scheme:

$$H[i,j] = min\begin{cases} H[i-1,j-1] + \delta_i \\ H[i-1,j] + 1 \\ H[i,j-1] + 1 \end{cases}$$

$$\delta_i = \begin{cases} 0, if\ pattern[j] = text[i] \\ 1, if\ pattern[j] \neq text[i] \end{cases}$$

   3. The edit distance cost is located in the lower right cell (i.e., $H_{m,n}$)

Levenshtein distance or edit distance

		A	T	G	C	A	T	G	C	C	G	G
	0	1	2	3	4	5	6	7	8	9	10	11
T	1	1										
C	2											
C	3											
G	4											
A	5											
A	6											
A	7											
C	8											

min[0+1, 1+(+1), 1+(+1)]

Levenshtein distance or edit distance

		A	T	G	C	A	T	G	C	C	G	G
	0	1	2	3	4	5	6	7	8	9	10	11
T	1	1	1	2	3	4	5	6	7	8	9	10
C	2	2	2	2	2	3	4	5	6	7	8	9
C	3	3	3	3	2	3	4	5	5	6	7	8
G	4	4	4	3	3	3	4	4	5	6	6	7
A	5	4	5	4	4	3	4	5	5	6	7	7
A	6	5	5	5	5	4	4	5	6	6	7	8
A	7	6	6	6	6	5	5	5	6	7	7	8
C	8	7	7	7	6	6	6	6	5	6	7	8

Edit distance [$H(m,n)$] = 8

Levenshtein distance or edit distance

		A	T	G	C	A	T	G	C	C	G	G
	0	←0	0	0	0	0	0	0	0	0	0	0
T	0	3	↖3	2	2	2	2	2	2	2	2	2
C	0	1	1	↖3	3	2	2	2	2	2	2	2
C	0	1	1	1	↖3	2	2	2	3	2	2	2
G	0	1	1	3	1 ↑	3	2	3	2	2	3	2
A	0	3	1	1	1	↖3	2	1	3	2	1	3
A	0	1	3	1	1	1	↖3	2	1	3	2	1
A	0	1	1	1	1	1	1	↖3	2	1	3	2
C	0	1	1	1	3	1	1	1	↖3	←2	←2	←2

Backtrack array: 1-means up 2-means left; 3-means diagonal

```
-TCCGAAAC---
ATGC-ATGCCGG
```

Edit distance = 8