Abstract and Introduction summary :

- The pattern matching is a very practical operation ,its enables users to find the locations of particular DNA subsequences in a database or DNA sequence so , high-speedpattern matching algorithms are needed .
- The problem is, biological data are enormous in size and very complex, we need efficient ways to analyze the data such that Human genome contains approximately 3 billion base pairs (high-computational costs) .
- The experimental results demonstrate the superiority of the presented algorithms over the other simulated algorithms and the development of efficient algorithms is still required .
- Pattern matching algorithms play a vital role in computational biology , Its necessary to study ourselves and learn about variant characteristics .
- The present paper introduces three pattern matching algorithms that are specially formulated to speed up searches on large DNA sequences , focuses on the exact pattern matching problem which finds all the occurrences of a pattern in a text .
- In the pattern matching problem, a text, sequence or database is scanned to detect the locations of a pattern in the text .
- The pattern matching problem arises in the different scopes of computational bioinformatics, which include the basic local alignment search, biomarker discovery, sequence alignment, proteogenomic mapping, and homologous series detection. In these disciplines, there is a need to recognize the locations of multiple patterns, including those of amino acids and nucleotides in databases .
- The comparison of a particular gene with similar genes of the same or different organisms and the prediction of its function .
- In another application, the functionality of a recently discovered DNA sequence can be prespecified by investigating its similarity to known sequences of DNA. This approach has been used in various research studies and medical applications.
- The operation of the proposed algorithms is divided into a preprocessing and a matching phase. In the preprocessing phase, the potential intervals of the text to be matched with the pattern are recognized , These intervals are called windows , during the matching phase, the windows are carefully scanned in order to be matched with the pattern .
- The fewer windows found in the preprocessing phase, the less time taken for verifying the windows in the matching phase , the present work's primary aim is to decrease the number of recognized windows .
- The present study introduces a third algorithm that focuses on the word of the pattern having the fewest repetitions in the text. In other words, the algorithm searches the text for a low-frequency word of the pattern. This technique further advances the algorithm's efficiency by decreasing the number of discovered windows.
- The KMP algorithm which performs the comparison from the left side. In the event of a mismatch, KMP moves the sliding window to the right by holding the longest overlap of a suffix of the matched text and a prefix of the pattern , This algorithm has a linear performance.
- The Boyer-Moore algorithm and its variants search the pattern in the text from right to left. In other words, this algorithm first matches the pattern's last character. At the end of the matching phase, it computes the shift increment.
  To decrease the number of comparisons when a mismatch occurs, two useful rules (bad character and good suffix) are utilized .
- The Divide and Conquer Pattern Matching (DCPM) is a comparison-based algorithm. At the beginning of the DCPM's preprocessing phase, the text is scanned for the rightmost character of the pattern. The index of the findings is stored in the rightmost character table. Then, to detect the leftmost character of the pattern, the text is scanned again.
  In the case of sameness, the indexes are saved in the leftmost character table.

Related Work summary :

- In pattern matching literature, Brute Force (BFpattern matching literature) is a primary method. After either a match or mismatch, the sliding window is shifted one position to the right. The high consumption of time is a significant disadvantage of pattern matching. The use of a dynamic automaton, these types of programs are not often scalable for large sequences.

PROBLEM STATEMENT :

- In recent years, the size of biological data has significantly grown and yet these large volumes of data must be analyzed within a reasonable time. This issue is encountered because sequences of amino acids or nucleotides are often applied to approximate biological sequences. In addition, a pattern matching algorithm is appropriate in fields such as phylogenetic and evolutionary biology. In these applications, specific applications are extracted from the genomic data of specific species

FLPM is an enhancement of DCPM [22] and is composed of preprocessing and matching phases. An example is then presented to complete the complete the end boundary of the search. The algorithm acts based on comparisons, the FLPM preprocessing phase scans text t to distinguish windows of the windows of text t which are later utilized by the matching phase. In the pre processing phase, the first character of p[0]. p is searched in text t; the search is performed during interval t [0].

Algorithm 1 Preprocessing Phase of FLPM Algorithm
Input: Text *t* and pattern *p* stored in the arrays of *t*[0*::n*-1] and *p*[0*::m* - 1], respectively, over finite alphabet P.
Output: The number of windows identified in this phase and their start indexes.
1. *count* 0
2. *num_window* 0
3. **WHILE** *count* ≤ *n* - *m* **DO**
4. **IF** *t* [*count*] D *p*[0] **THEN**
5. **IF** *t* [*count* C *m* - 1] D *p*[*m* - 1] **THEN**
6. *window_index*[*num_window*] *count*
7. *num_window num_window* C 1
8. **END-IF**
9. **END-IF**
10. *count count* C 1
11. **END-WHILE**

Phrase matches the matching phase to find all occurrences of the pattern in the text. The number of windows and their start indexes are considered as the inputs of Algorithm 2. The algorithm's outer while loop repeats the instructions for all instructions for each window. If alignment succeeds, then this window is an answer. Otherwise, there is a mismatch or a match or mismatch.

Algorithm 2 Matching Phase of FLPM Algorithm
Input: The number of windows identified in the preprocessing phase and their start indexes.
Output: The start index for all occurrences of pattern *p* in text *t*.
1. *count* 0
2. *num_match*
3. **WHILE** *count < num_window* **DO**
4. *s window_index*[*count*]
5. *c* 1
6. **WHILE** *c* ≤ *m* - 2 **DO**
7. **IF** *p* [*c*] 6D *t*[*s* C *c*] **THEN**
8. BREAK /∗Exit the current loop∗/
9. **END-IF**
10. *c c* C 1
11. **END-WHILE**
12. **IF** *c* D *m* - 1 **THEN**
13. *match_index*[*num_match*] *s*
14. *num_match num_match* C 1
15. **END-IF**
16. *count count* C 1
17. **END-WHILE**

- PAPM algorithm is different from the FLPM algorithm in the way pattern p characters and text t southwesterncharacters are compared. PAPM performs comparisons based on a word comprised of several characters while FLPM uses several words.
- In PAPM, the first word of the pattern is searched in the text. The processor can compare a word (including word_lencharacters) with another by using its registers. The start index for the word comparison is saved in the window_index array. The higher number of characters (a word) searched in a textields a lower number of windows, which may decrease the time necessary for the next phase of processing.

- Least Frequency Pattern Matching (LFPM) algorithm is an enhancement of PAPM that is specialized for DNAapplications. LFPM focuses on a word that is expected to appear less than other words of the pattern.
- In the first step before the preprocessing phase, LFPMcomputes the frequency of all possible words for the related alphabet 6. The frequency of each word is shown in the word_len column of that word. For example, in a 32-bit computer, 256 different words having. the length of four characters from therelated alphabet 6 can be generated. The amount of memory required to maintain the memory is reasonable, so this table can be placed in the main memory.

Algorithm 5 LFPM Algorithm
Input: The filled *freq_table* on reference data. Text *t* and pattern *p* stored in the arrays of *t*[0::*n*-1] and *p*[0::*m*-1], respectively, over finite alphabet P D f*A; C; G; T* }.
Output: The first indexes for all occurrences of pattern *p* in text *t*.
/∗**PREPROCESSING PHASE**∗/
/∗**Step 1: finding the least frequent word in the pattern**∗/
1. *count* 0
2. *min_value* 1
3. *min_index* -1
4. **WHILE** *count* ≤ *m* - *word_len* **DO**
5. *row_count* 0, *col_count* 0
6. *w* [0::*word_len* - 1]
p [*count*::*count* C *word_len* - 1]
7. **WHILE** *col_count* < *word_len* **DO**
8. **SWITCH** *w* [*col_count*]
9. **CASE**‘A’: nothing
10. **CASE**‘C’: Increase *row_count* by 1 × 4*word_len-col_count*-1
11. **CASE**‘G’: Increase *row_count* by 2 × 4*word_len-col_count*-1
12. **CASE**‘T’: Increase *row_count* by 3 × 4*word_len-col_co*

RESULTS

this section compares the performance of the presented algorithms (FLPM, PAPM, and LFPM) with that of other algorithms. The simulation was performed with the C programing language. The amount of overhead was ignored in the calculations because the table of frequency is created only once for any text or database worked on by the algorithm. The specifications of the different simulatedalgorithms were as follows.

CONCLUSION

The current paper introduces three new algorithms: FLPM, PAPM, and LFPM. The proposed algorithms surpass the other simulated pattern matching algorithms in terms of time cost. The presentation of a parallel version of the presented algorithms is left for future work.