CSCI 5454: Final Project Presentation

Boyer-Moore's algorithm for String Matching

By
Aditya Chandra
Ashutosh Gandhi

Objectives

- Naive Approach for String Matching
- Introduction
- Boyer-Moore algorithm
- Galil Rule Optimization
- Complexity Analysis
- Boyer-Moore-Horspool algorithm
- Real-world applications of Boyer-Moore algorithm
- Comparison with other string matching algorithms

Naive approach to String matching

Text: TCACT, Pattern: ACT

T	C	A	C	T
A	C	T		

T	**C**	A	C	T
	A	C	T	

T	C	**A**	**C**	**T**
		A	**C**	**T**

Time Complexity: O(m*n)
n - length of Text, m - length of pattern

Introduction

- The Boyer-Moore (BM) algorithm is a string matching algorithm that was developed by Robert S. Boyer and J Strother Moore in 1977. It is a powerful and widely used algorithm for finding occurrences of a pattern within a text, and is particularly effective when the pattern is long or has many repeating characters.

- Although the BM algorithm has a worst-case time complexity of O(nm) in the general case, where n is the length of the text and m is the length of the pattern, it can achieve sublinear performance in practice due to its use of heuristics and precomputed tables.

- This makes it an attractive choice for many applications that require efficient string matching. It has been widely used in applications such as text editors, search engines, and bioinformatics, where efficient string matching is crucial.

Boyer-Moore algorithm - Bad Character Heuristic

Case: 1 The mismatch becomes a match

T	T	G	**A**	**C**	T	G	A	C	T	C	A	C	T
P	A	C	**T**	**C**									

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

Boyer-Moore algorithm - Bad Character Heuristic

Case: 2 Pattern moves past the mismatch character

T	T	G	A	C	T	**G**	A	C	T	C	A	C	T
P			A	C	T	**C**							

T	T	G	A	C	T	G	**A**	**C**	**T**	**C**	A	C	T
P							**A**	**C**	**T**	**C**			

Boyer-Moore algorithm - Bad Character Heuristic

Construction of shift table (last occurrence of character in the pattern)

Pattern - ACTC

c	A	C	T	G
delta1(c)	0	3	2	-1

shift = max(1, delta1(Text(c)))

c - mismatch character

Boyer-Moore algorithm - Bad Character Heuristic

Shift table construction code

```
def bad_character_table(self, pattern):
  delta_1 = [-1]*CHAR_SIZE

  for idx,p in enumerate(pattern):
    # delta_1[ord(p)-ord('a')] = idx
    delta_1[ord(p)] = idx

  return delta_1
```

Time complexity: O(n)

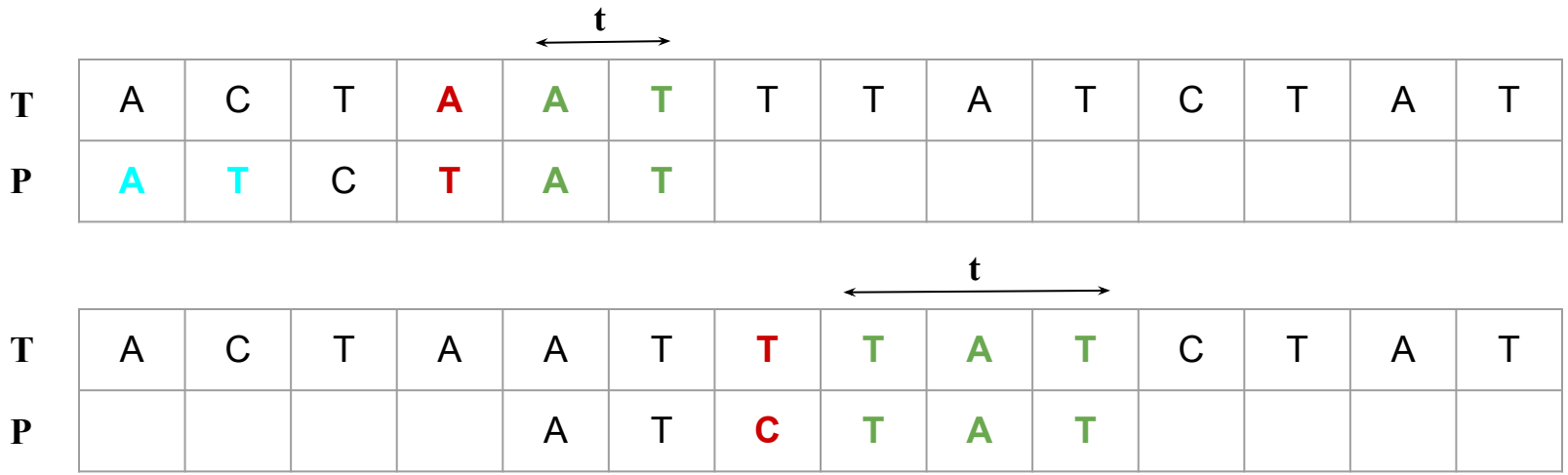
Space complexity: O(|Σ|)

n - length of the pattern

|Σ| - total number of characters in the text

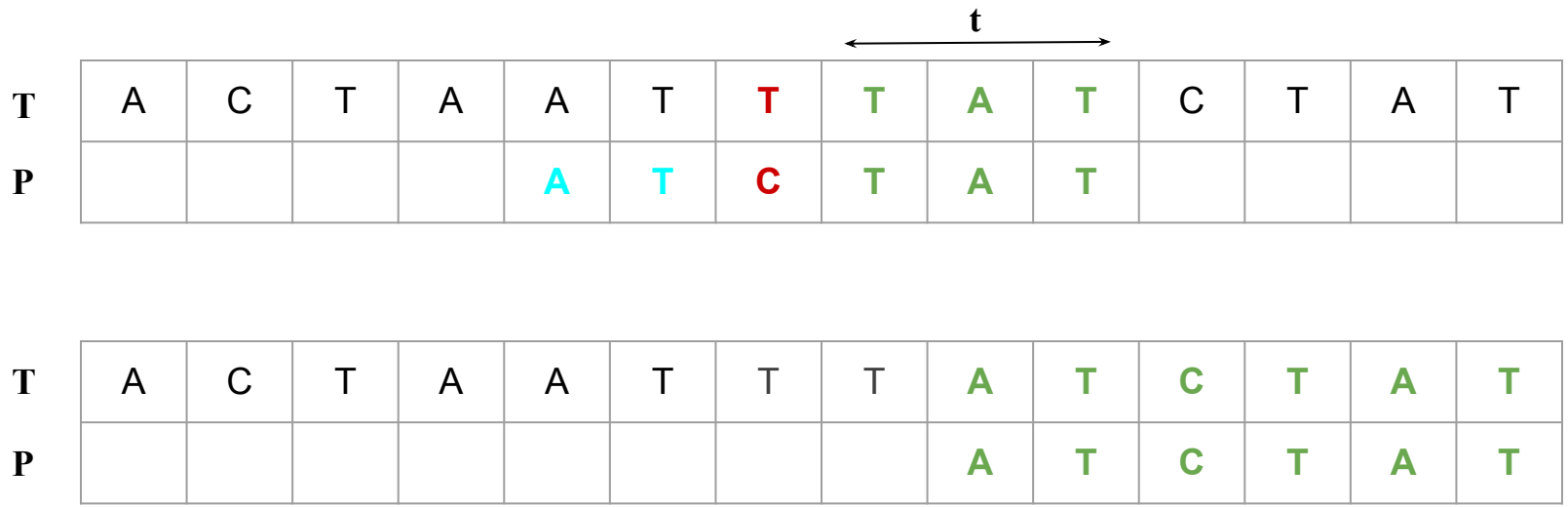
Boyer-Moore algorithm - Good suffix Heuristic

Case 1: Another occurrence of matched suffix in the pattern



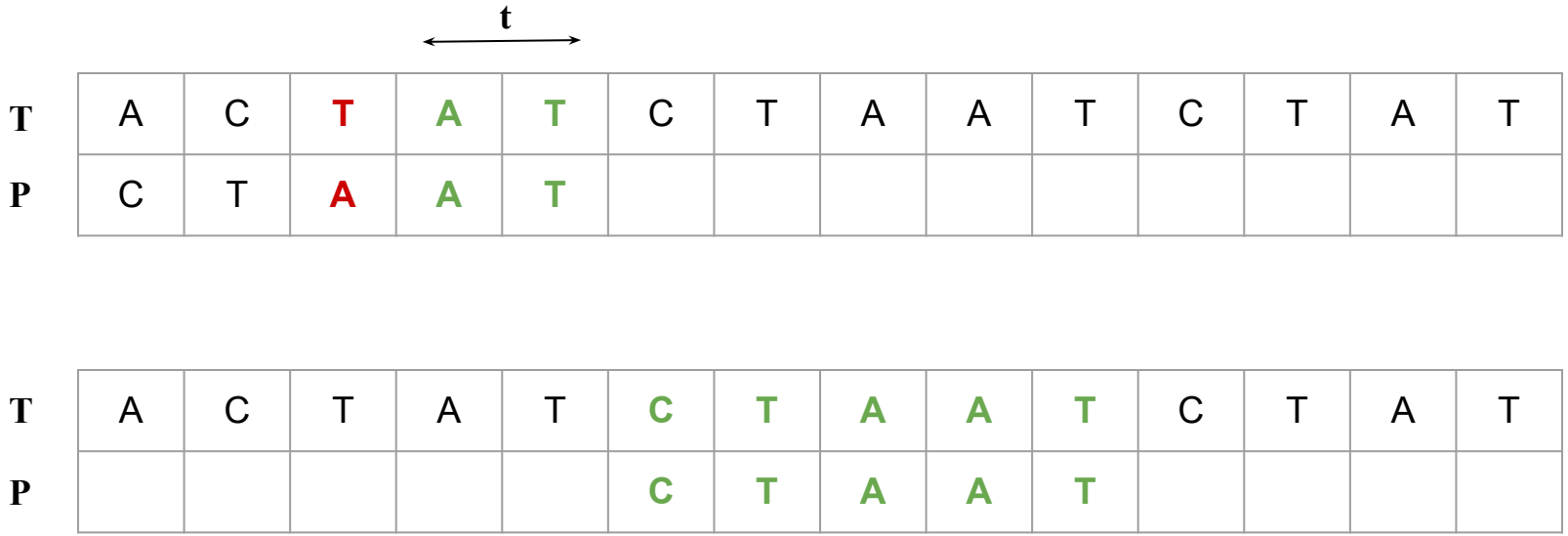
Boyer-Moore algorithm - Good suffix Heuristic

Case 2: A suffix of t in T, matches with a prefix in P



Boyer-Moore algorithm - Good suffix Heuristic

Case 3: P moves past t (case 1 and case 2 don't satisfy)



Galil extension to Boyer-Moore algorithm

Skips comparison of a know failure

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
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	C	G	G	C	G	G	C																					

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
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	C	G	G	C	G	G	C															

Boyer-Moore algorithm - Good suffix Heuristic

Construction of shift table

Pattern - ATCTAT

k	Pattern	delta2(k)
1	ATC$\overline{T}$AT	2
2	$\overline{AT}$CTAT	4
3	$\overline{A}$TCTAT	4
4	$\overline{AT}$CTAT	4
5	$\overline{A}\overline{T}$CTAT	4

shift = delta2(k)

k - len of the pattern suffix that matched

Boyer-Moore algorithm - Good suffix Heuristic

Shift table construction code

Time complexity: O(n)

Space complexity: O(n)

n - length of the pattern

```
def good_suffix_table(self, pattern):
    p = 0
    pat_len = len(pattern)
    i, j = pat_len, pat_len + 1
    delta_2 = [0] * j
    suffix_idx = [0] * j
    suffix_idx[i] = j

    while i > 0:

        while j <= pat_len and pattern[i-1] != pattern[j-1]:
            if delta_2[j] == 0:
                delta_2[j] = j - i

            j = suffix_idx[j]

        i -= 1
        j -= 1
        suffix_idx[i] = j

    offset = suffix_idx[0]

    while p < pat_len + 1:
        if delta_2[p] == 0:
            delta_2[p] = offset

        if p == offset:
            offset = suffix_idx[offset]

        p += 1

    return delta_2
```

Boyer-Moore algorithm - example

Text - GTTATAGCTGATCGCGGCGTAGCGGCGAA

Pattern - GTAGCGGCG

Bad character shift table

c	A	T	C	G
delta1(c)	2	1	7	8

shift =

max(max(1,j-delta1(Text(shift+j))), delta2(k))

Good suffix shift table

k	**pattern**	**delta2(k)**
1	GTAGCGG̅C̅G	2
2	GTAGC̅G̅GCG	3
3	GTAG̅C̅G̅GCG	3
4	G̅TAGCGGCG	8
5	G̅TAGCGGCG	8
6	G̅TAGCGGCG	8
7	G̅TAGCGGCG	8
8	G̅TAGCGGCG	8

Boyer-Moore algorithm - example

Step 1:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
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																				

delta1(T) = 1, delta2(0) = 0; shift = max(8-delta1(T), 0) = max(8-1, 0) = 7

Step 2:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
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													

delta1(Text(7+5)) = delta1(C)=7, delta2(3) = 3;

shift = max(max(1,5-delta1(C)), 3) = max(max(1, -2), 3) = max(1,3) = 3

Boyer-Moore algorithm - example

Step 3:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
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										

delta1(Text(10+2)) = delta1(C) = 7, delta2(6) = 8;

shift = max(max(1,2-7), 8) = max(1,8) = 8

Step 4:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
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		

Boyer-Moore algorithm

String matching code

Time complexity: O(m+n)

Space complexity: O(n + |Σ|)

n - length of the pattern,

|Σ| - total number of alphabets in the text

```
def search(self, text):
  text = text.lower()
  shift = 0
  pat_len = len(self.pattern)
  txt_len = len(text)
  match_found = list()

  while shift <= txt_len - pat_len :
    j = pat_len - 1

    while j >= 0 and self.pattern[j] == text[shift + j] :
      j -= 1

    if j < 0 :
      # print("Pattern found at index : ", shift)
      match_found.append(shift)
      match_end = shift + pat_len
      shift_1 = (pat_len - self.get_delta_1(text, match_end) if match_end < txt_len else 1)
      shift_2 = self.delta_2[0]

    else :
      shift_1 = max(1, j - self.get_delta_1(text, shift + j))
      shift_2 = self.delta_2[j + 1]

    # print("Delta-1 shift : ", shift_1)
    # print("Delta-2 shift : ", shift_2)

    shift += max(shift_1, shift_2)

  return match_found
```

Boyer-Moore Horspool algorithm

Published in 1980 as an extension to Boyer-Moore algorithm.

- Only applies the bad character heuristic when matching the string
- Makes 2 changes to the bad character heuristic
  - Omits the last character when building the shift table
  - When a mismatch is detected, always use the character in the Text string that is aligned with the last character in the pattern to determine the amount of shift
- It is more efficient than Boyer-Moore for small alphabets sizes.
- It is more easier to implement.

Applications of Boyer-Moore Algorithm

- Used in GNU's grep
- Used in the Golang strings library
- Also part of the standard C++ library since C++17
- Search in text editors and word processors
- Code editors and IDEs
- File and data compression

Comparative Analysis

demo.ipynb

References

- A Fast String Searching algorithm https://dl.acm.org/doi/pdf/10.1145/359842.359859
- On improving the worst case running time of the Boyer–Moore string matching algorithm https://doi.org/10.1145%2F359146.359148
- https://dearxxj.github.io/post/4/
- Ch10 Information Retrieval: Data Structures & Algorithms http://orion.lcg.ufrj.br/Dr.Dobbs/books/book5/chap10.htm
- Good suffix rule in Boyer Moore algorithm explained simply
- Practical fast searching in strings https://onlinelibrary.wiley.com/doi/10.1002/spe.4380100608

THANK YOU!