Knuth Morris Pratt

CS 491 – Competitive Programming

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Spring 2024

Objectives

- Explain the Naïve string matching algorithm and its time complexity.
- Explain the KMP Prefix Array.
- Use the Prefix Array to find a string…
  - … and explain the new runtime.

Code

```
1  void naiveMatching() {
2    for (int i = 0; i < n; i++) {
3      bool found = true;
4      for (int j = 0; j < m && found; j++)
5        if (i + j >= n || P[j] != T[i + j])
6          found = false; // mismatch!
7      if (found) // if P[0..m-1] == T[i..i+m-1]
8        printf("P is found at index %d in T\n", i);
9    } }
```

Example 1 - What happens after check 6?

- Check 1

```
i 01234567890123456789012345678901234567890123456789
T This is a test
P is a
  ^
```

- Check 3

```
i 01234567890123456789012345678901234567890123456789
T This is a test
P   is a
    ^
```

- Check 6

```
i 01234567890123456789012345678901234567890123456789
T This is a test
P   is a
        ^
```

Example 1 ctd

- Check 7

```
i 0123456789012345678901234567890123456789
T This is a test
P    is a
     ^
```

- Check 8

```
i 0123456789012345678901234567890123456789
T This is a test
P     is a
      ^
```

- Check 7 and 8 are useless. We could have told you that nothing will happen.
- Can we skip ahead further?

Example 1b

- Check 6

```
i 01234567890123456789012345678901234567890123456789
T This is a test
P   is a
      ^
```

- Check 7

```
i 01234567890123456789012345678901234567890123456789
T This is a test
P     is a
      ^
```

- We could do this: just start over from the beginning of the pattern where the last match failed.
- What could go wrong?

Example 2 - What happens after check 5?

- Check 1

```
i 01234567890123456789012345678901234567890123456789
T xxaaaabaaayyy
P aaabaa
  ^
```

- Check 2

```
i 01234567890123456789012345678901234567890123456789
T xxaaaabaaayyy
P  aaabaa
   ^
```

- Check 6

```
i 01234567890123456789012345678901234567890123456789
T xxaaaabaaayyy
P    aaabaa
        ^
```

The KMP Method

```
i 01234567890123456789012345678901234567890123456789
T xxaaaabaaayyy
P    aaabaa
       ^
```

- When a match fails, you can advance up to the length of the partial match, minus the lengths of the largest overlapping common prefix and common suffix.
  Proper prefixes of aaa:
  - a and aa

  Proper suffixes of aaa:
  - a and aa

  Match length is 2, so instead of advancing 3 space, we advance 1.

Prefixes

Consider `abababca`, calculate largest overlapping proper prefix / suffix for each prefix.

- a - none, 0
- ab - none, 0
- aba
  - Prefix ab, suffix ba (no overlap)
  - Prefix a, suffix a, so 1
- abab
  - Prefix aba, suffix bab (no overlap)
  - Prefix ab, suffix ab, so 2
- ababa
  - Prefix abab, suffix baba (no overlap)
  - Prefix aba, suffix aba , so 3
- ababab
  - Prefix abab, suffix abab, 4

Prefixes, ctd

Consider `abababca`, calculate largest overlapping proper prefix / suffix for each prefix.

- abababc - none 0
- abababca
  - a, so 1

Code

- Stolen from 'imslavko' on Stack Overflow:
- `https://stackoverflow.com/questions/13792118/kmp-prefix-table`
- 's' is the search string

```
vector<int> prefixFunction(string s) {
    vector<int> p(s.size());
    int j = 0;
    for (int i = 1; i < (int)s.size(); i++) {
        while (j > 0 && s[j] != s[i])
            j = p[j-1];

        if (s[j] == s[i])
            j++;
        p[i] = j;
    }
    return p;
}
```

Using the Prefix Table

- T is the Text
- S is the Search String
- p is the prefix vector

```
1  int i = 0, j = 0;
2  while (i < n) {
3    while (j >= 0 && T[i] != S[j])
4      j = p[j-1]; // different, reset j using b
5    i++; j++;
6    if (j == m) { // a match found when j == m
7      printf("P is found at index %d in T\n", i - j);
8      j = p[j];
9    } } }
```

Example

```
i 01234567890123456789012345678901234567890123456789
T xxaaaabaaayyy
P aaabaa
```

- Prefix array is 0 1 2 0 0 1 2

```
i 01234567890123456789012345678901234567890123456789
T xxaaaabaaayyy
P   aaabaa
      ^
```

- P[j-1] = 2. Size of match is 3, so back up 3-2 = 1 spaces.