The Knuth-Morris-Pratt Algorithm in my own words

Dec 13th, 2009 | Comments

For the past few days, I've been reading various explanations of the Knuth-Morris-Pratt string searching algorithms. For some reason, none of the explanations were doing it for me. I kept banging my head against a brick wall once I started reading "the prefix of the suffix of the prefix of the…".

Finally, after reading the same paragraph of CLRS over and over for about 30 minutes, I decided to sit down, do a bunch of examples, and diagram them out. I now understand the algorithm, and can explain it. For those who think like me, here it is in my own words. As a side note, I'm not going to explain why it's more efficient than na"ive string matching; that's explained perfectly well in a multitude of places. I'm going to explain exactly how it works, as my brain understands it.

The Partial Match Table

The key to KMP, of course, is the partial match table. The main obstacle between me and understanding KMP was the fact that I didn't quite fully grasp what the values in the partial match table really meant. I will now try to explain them in the simplest words possible.

Here's the partial match table for the pattern "abababca":

```
char:  | a | b | a | b | a | b | c | a |
index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
value: | 0 | 0 | 1 | 2 | 3 | 4 | 0 | 1 |
```

If I have an eight-character pattern (let's say "abababca" for the duration of this example), my partial match table will have eight cells. If I'm looking at the eighth and last cell in the table, I'm interested in the entire pattern ("abababca"). If I'm looking at the seventh cell in the table, I'm only interested in the first seven characters in the pattern ("abababc"); the eighth one ("a") is irrelevant, and can go fall off a building or something. If I'm looking at the sixth cell of the in the table… you get the idea. Notice that I haven't talked about what each cell *means* yet, but just what it's referring to.

Now, in order to talk about the meaning, we need to know about **proper prefixes** and **proper suffixes**.

Proper prefix: All the characters in a string, with one or more cut off the end. “S”, “Sn”, “Sna”, and “Snap” are all the proper prefixes of “Snape”.

Proper suffix: All the characters in a string, with one or more cut off the beginning. “agrid”, “grid”, “rid”, “id”, and “d” are all proper suffixes of “Hagrid”.

With this in mind, I can now give the one-sentence meaning of the values in the partial match table:

The length of the longest proper prefix in the (sub)pattern that matches a proper suffix in the same (sub)pattern.

Let’s examine what I mean by that. Say we’re looking in the third cell. As you’ll remember from above, this means we’re only interested in the first three characters (“aba”). In “aba”, there are two proper prefixes (“a” and “ab”) and two proper suffixes (“a” and “ba”). The proper prefix “ab” does not match either of the two proper suffixes. However, the proper prefix “a” matches the proper suffix “a”. Thus, **the length of the longest proper prefix that matches a proper suffix**, in this case, is 1.

Let’s try it for cell four. Here, we’re interested in the first four characters (“abab”). We have three proper prefixes (“a”, “ab”, and “aba”) and three proper suffixes (“b”, “ab”, and “bab”). This time, “ab” is in both, and is two characters long, so cell four gets value 2.

Just because it’s an interesting example, let’s also try it for cell five, which concerns “ababa”. We have four proper prefixes (“a”, “ab”, “aba”, and “abab”) and four proper suffixes (“a”, “ba”, “aba”, and “baba”). Now, we have two matches: “a” and “aba” are both proper prefixes and proper suffixes. Since “aba” is longer than “a”, it wins, and cell five gets value 3.

Let’s skip ahead to cell seven (the second-to-last cell), which is concerned with the pattern “abababc”. Even without enumerating all the proper prefixes and suffixes, it should be obvious that there aren’t going to be any matches; all the suffixes will end with the letter “c”, and none of the prefixes will. Since there are no matches, cell seven gets 0.

Finally, let’s look at cell eight, which is concerned with the entire pattern (“abababca”). Since they both start and end with “a”, we know the value will be at least 1. However, that’s where it ends; at lengths two and up, all the suffixes contain a c, while only the last prefix (“abababc”) does. This seven-character prefix does not match the seven-character suffix (“bababca”), so cell eight gets 1.

How to use the Partial Match Table

We can use the values in the partial match table to skip ahead (rather than redoing unnecessary old comparisons) when we find partial matches. The formula works like this:

*If a partial match of length **partial_match_length** is found and `table[partial_match_length] > 1`, we may skip ahead `partial_match_length - table[partial_match_length - 1]` characters.*

Let’s say we’re matching the pattern “abababca” against the text “bacbababaabcbab”. Here’s our partial match table again for easy reference:

```
char:  | a | b | a | b | a | b | c | a |
index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
value: | 0 | 0 | 1 | 2 | 3 | 4 | 0 | 1 |
```

The first time we get a partial match is here:

```
bacbababaabcbab
 |
 abababca
```

This is a partial_match_length of 1. The value at `table[partial_match_length - 1]` (or `table[0]`) is 0, so we don't get to skip ahead any. The next partial match we get is here:

```
bacbababaabcbab
    |||||
    abababca
```

This is a partial_match_length of 5. The value at `table[partial_match_length - 1]` (or `table[4]`) is 3. That means we get to skip ahead `partial_match_length - table[partial_match_length - 1]` (or `5 - table[4]` or `5 - 3` or 2) characters:

```
// x denotes a skip

bacbababaabcbab
    xx|||
      abababca
```

This is a partial_match_length of 3. The value at `table[partial_match_length - 1]` (or `table[2]`) is 1. That means we get to skip ahead `partial_match_length - table[partial_match_length - 1]` (or `3 - table[2]` or `3 - 1` or 2) characters:

```
// x denotes a skip

bacbababaabcbab
      xx|
        abababca
```

At this point, our pattern is longer than the remaining characters in the text, so we know there's no match.

Conclusion

So there you have it. Like I promised before, it's no exhaustive explanation or formal proof of KMP; it's a walk through my brain, with the parts I found confusing spelled out in extreme detail. If you have any questions or notice something I messed up, please leave a comment; maybe we'll all learn something.

Posted by Jake Boxer Dec 13th, 2009

Comments

177 Comments jBoxer 1 Login

Favorite 84 Tweet Share Sort by Best