KMP Applications

Mostafa Saad Ibrahim

Teaching Assistant @ Cairo University

Software Engineer @ NTPSoftware

Palindrome & Failure Function

- Problem: Given a string, What is the longest suffix that is palindrome?
  - aaaa => aaaa
  - abc => c
  - abcdeffe -> effe
- Or, the reverse problem: Given a string, What is the longest prefix that is palindrome?

- We will use the 2 problems to help us solve them.

Palindrome & Failure Function

- abcd**effe** => effe in suffix problem
- **effe**dcba => effe in prefix problem

- Let's assume char @ will not be part of input..let's construct:

- **effe**dcba@ abcd**effe**

- Now problem, find longest prefix that is suffix

- Failure function can answer that easily: F[size-1] is answer

Palindrome & Failure Function

- Problem: Given a string, What is the minimum characters to add to convert string to palindrome?
  - aaaa => aaaa
  - abc => ab**c**ba
  - abcdeffe -> abcd**eff**edcba

- Your turn ☺

Repetition & Failure Function

- Let concatenate(S, N) = SSSS …N times
  - concatenate(abcd, 3) = abcdabcdabcd
- Given String concatenated, find minimum N
- E.g. Given abcdabcdabcd => 3
- E.g. Given **ab**abababab => 2
  - It is also can be abab 2 times, but ab smaller
- E.g Given abc => 1

Repetition & Failure Function

a	**b**	**c**	**d**	a	b	c	d	a	b	c	d

0	0	0	0	1	2	3	4	5	6	7	8

As we match longest proper prefix NOT longest prefix

F[len-1] will fail such that covering all repetition except last one

That is if string is 10 times concatenated, The first 9 blocks = last 9 blocks

=> here abcdabcd = abcdabcd

So we can get the block easily => Len – F[Len-1] = 12 – 8 = 4

Observation: Len % (Len – F[Len-1]) = 0

Repetition & Failure Function

a	**b**	**c**	**d**	a	b	c	d	a	b	c	d

0	0	0	0	1	2	3	4	5	6	7	8

Sol: if Len % (Len – F[Len-1])= 0 then **Len – F[Len-1]** is solution

Why if so, all blocks are equal

Let String S = con(abcd, 5) = abcdabcdabcdabcdabcd
Let's divide string for 5 blocks ABCDE

F[Len-1] = 20, which means first 20 = last 20…
or first 4 blocks = last 4 blocks Then: ABCD = BCDE
Then A = B = C = D = E

Repetition & Failure Function

a	**b**	**c**	**d**	e	g	g	h	a	b	c	d

0	0	0	0	0	0	0	0	1	2	3	4

Len – F[Len-1] = 8 => 12 % 8 != 0

What does this mean? first 4 = last 4, but in between is not like them

Repetition & Failure Function

a	**b**	**c**	**d**	e	g	g	h	a	b	c	d
0	0	0	0	0	0	0	0	1	2	3	4

Len – F[Len-1] = 8 => 12 % 8 != 0

What does this mean? first 4 = last 4, but in between is not like them

Repetition & Failure Function

- Another way, using KMP itself

- Let String input P = abcdabcdabcd
- Let S = p+p = abcdabcdabcdabcdabcdabcd

- Now when we try to find P in S, it should match in 0, k, 2k and so on. So 2nd match is enough for answer

- abcd**abcdabcdabcd**abcdabcd => 2nd match
- abcdabcdabcd

All prefixes & Failure Function

- Problem: Given string P, For each prefix, count its frequency
- E.g aabaaab => 5 3 2 1 1 1 1
  - E.g. a appeared 5 times and aa appeared 3 times
- A basic approach will try every prefix and search whole string in O(n^2)

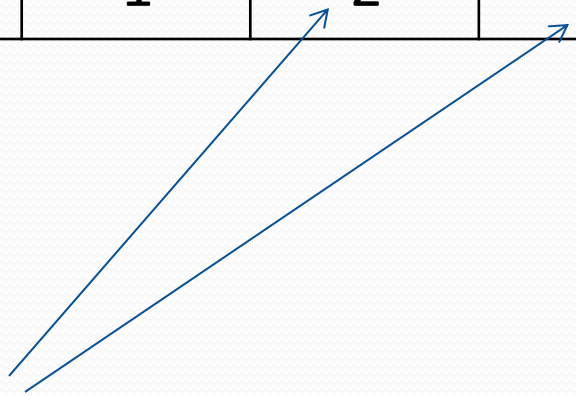
All prefixes & Failure Function

- Remember the prefix function expresses longest proper prefix that is a suffix
- For a prefix to exist in mid of S, it will be suffix for a prefix
- E.g. in aabaaab, prefix **aa** exist as suffix of 3 prefixes: **aa**, aab**aa**, aaba**aa**
- If for each prefix we counted these suffixes, we efficiently listed the prefixes them

All prefixes & Failure Function

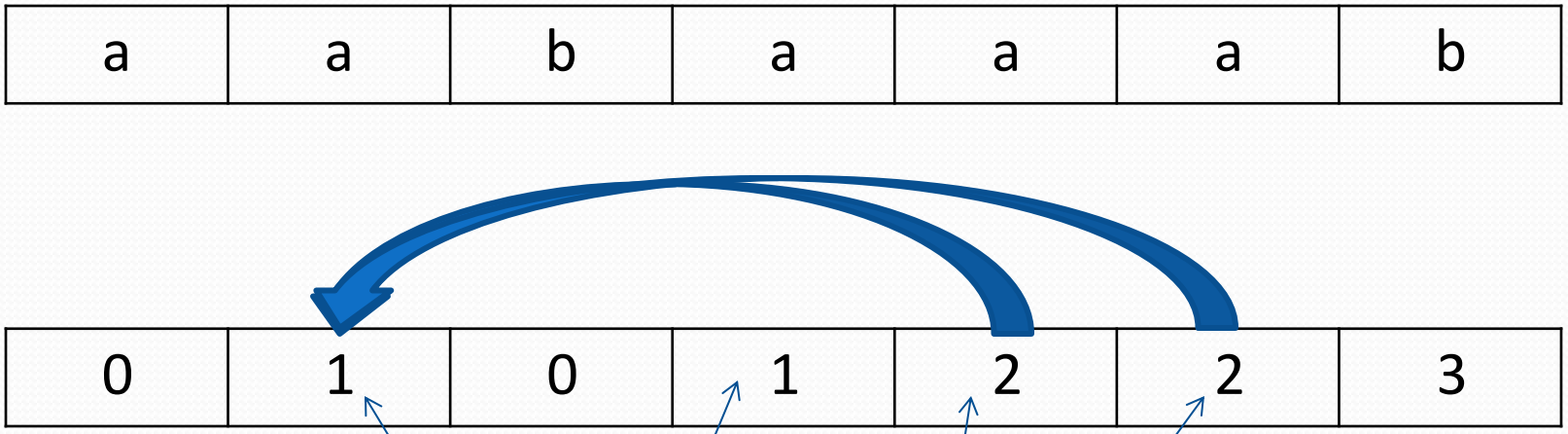
a	a	b	a	a	a	b

0	1	0	1	2	2	3



2 locations fail at 2
Prefix 2 appeared 1 time + 2 from failure
aa = 3

All prefixes & Failure Function



2 locations fail at 1
2 locations fail at 2 which fail at a
2+2 + 1
a = 5

Think A fails over B,
Which fails over C,
Which fails over D
And so on

All prefixes & Failure Function

```
vector<int> pi = computePrefix(pat);

int n = sz(pi);

vector<int> freq(n, 1);

for (int i = 0; i < n; ++i) {
    // For each prefix, find all its sub-prefixes
    int k = pi[i];

    while (k > 0) {
        freq[k - 1]++;
        k = pi[k - 1];
    }
}
```

All prefixes & Failure Function

```
vector<int> pi = computePrefix(pat);

int n = sz(pi);

vector<int> freq(n + 1);

for (int i = 0; i < n; ++i)
    ++freq[pi[i]];

for (int i = n - 1; i > 0; --i)
    freq[pi[i - 1]] += freq[i];

freq.erase(freq.begin());
```

Your Turn ☺
How this O(n) works

All prefixes & Failure Function

- Problem: Given string P, For each prefix, count its frequency in String T

- Hint1, make use of the previous processing
- Hint2: Construct string P@T where @ is character that won't appear in P or T

Count # of unique prefixes

- Given String, count # of suffixes that don't appear again as substring….Or problem reverse
- Given String, count # of unique prefixes

- Remember, When prefix fails on location, it means this location will be prefix later

- So, mark all location that we fail on as repeated prefixes…remaining are unique

Count # of unique prefixes

- For example, count unique prefixes for **abab**
  - a -> appears later -> ignore
  - ab -> appears later -> ignore
  - aba Unique
  - abab Unique

Count # of unique prefixes

```
vector<int> pi = computePrefix(pat);

vector<bool> v(sz(pi));

rep(i, pi)
{
    int k = pi[i];

    if(k)
        v[k-1] = 1; // this location is a repeated prefix
}

cout<<"\n";
rep(i, v) if(!v[i])
    cout<<pat.substr(0, i+1)<<"\n";
```

Count # of distinct substring

- abc => has a, b, c, ab, bc, abc
- aaa => has a, aa, aaa
- aabab => a, b, aa, ab, ba, aab, aba, bab, aaba, abab, aabab
- Think Incrementally: If we know the answer for the first N letters…Could we know for the N+1?
- When we add the N+1 character, we have N+1 suffix. We need only the unique suffixes of them.

- For each prefix P => O(n^2)
  - Count += CountUniquePrefixes(reverse(P))

KMP Cheat Sheet

- Key words: substring, prefix, suffix, string equalities
- Typically processing for the Pattern, similar to Input S
- Try to find solution in: S, SS, rev(S), S@S, and so on