CS 240 – Data Structures and Data Management

Module 9e: String Matching - Enriched

T. Biedl É. Schost O. Veksler

Based on lecture notes by many previous cs240 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Winter 2021

version 2021−03−30 10:36

Outline

Outline

KMP failure function – fast computation

$F[j]$ is the length of the longest prefix of P that is a suffix of $P[1..j]$.

- How can we compute this faster?
- Recall property of KMP-automaton of P:
  - If we are in state ℓ, then we have just seen $P[0..\ell-1]$

  $\Leftrightarrow$ $P[0..\ell-1]$ is a suffix of what we have just parsed.

  - Also, KMP is always in the rightmost state where this holds.

  $\Leftrightarrow$ $P[0..\ell-1]$ is the *longest* suffix of what we have just parsed.

  $\Leftrightarrow$ ℓ is the length of the longest prefix of P that is a suffix of what we have just parsed.

KMP failure function – fast computation

$F[j]$ is the length of the longest prefix of P that is a suffix of $P[1..j]$.

- How can we compute this faster?
- Recall property of KMP-automaton of P:
  - If we are in state ℓ, then we have just seen $P[0..\ell-1]$

  $\Leftrightarrow$ $P[0..\ell-1]$ is a suffix of what we have just parsed.

  - Also, KMP is always in the rightmost state where this holds.

  $\Leftrightarrow$ $P[0..\ell-1]$ is the *longest* suffix of what we have just parsed.

  $\Leftrightarrow$ ℓ is the length of the longest prefix of P that is a suffix of what we have just parsed.

Combine this with the definition of $F[j]$ to get:

$F[j] = \ell \Leftrightarrow$
we reach state ℓ when parsing $P[1..j]$ on the KMP-automaton for P

KMP failure function – fast computation

$F[j]$ = **the state we reach when parsing** $P[1..j]$

This immediately gives algorithm: For $j = 1, 2, \dots,$

- parse $P[1..j]$ on the KMP-automaton for P
- Set $F[j] = \ell$ if we reach state ℓ

KMP failure function – fast computation

$F[j] =$ **the state we reach when parsing** $P[1..j]$

This immediately gives algorithm: For $j = 1, 2, \ldots,$

- parse $P[1..j]$ on the KMP-automaton for P
- Set $F[j] = \ell$ if we reach state ℓ

Observe: We don't need to re-start the parsing from scratch!

- Assume we have computed $F[j]$ already.
- To compute $F[j+1]$, parse $P[j+1]$ and note reached state.
- So can compute $F[0..m-1]$ with *one* parse of $P[1..m-1]$

KMP failure function – fast computation

$F[j]$ = **the state we reach when parsing** $P[1..j]$

This immediately gives algorithm: For $j = 1, 2, \ldots,$

- parse $P[1..j]$ on the KMP-automaton for P
- Set $F[j] = \ell$ if we reach state ℓ

Observe: We don't need to re-start the parsing from scratch!

- Assume we have computed $F[j]$ already.
- To compute $F[j+1]$, parse $P[j+1]$ and note reached state.
- So can compute $F[0..m-1]$ with *one* parse of $P[1..m-1]$

But isn't this circular?

- We need failure-arcs for parsing, but we compute them only now!
- But: To compute $F[j]$, parse $P[1..j-1]$ first ($j-1$ characters)
  $\Rightarrow$ reach state $\leq j$
  $\Rightarrow$ don't need $F[j]$ (= arc from state $j+1$) to parse $P[j]$