Substring Search

Goal: find pattern of length M in text of length N

pattern → needle

text → haystack

* screen scraping → extract relevant data from web page

Brute-Force Substring Search

- check for pattern starting at each position
- worst case $M \times N$
- in many applications we want to avoid backup in text stream

Approaches

1) maintain buffer of last M characters

=> alternate implementation

* (i) points to end of sequence of already matched chars in text

* j stores # of already-matched chars

Knuth-Morris-Pratt

- always avoids backup
- Deterministic finite state automaton (DFA)

=> DFA = abstract string-searching machine
  - finite number of states (including start and halt)
  - exactly one transition for each char in alphabet
  - accept if sequence of transitions leads to halt state

* if in state j reading char c:
  if j is 6 halt and accept

else move to state dfa[c][j]

state = # of characters in pattern that have been matched

* differences between brute-force implementation

- need to precompute dfa[][] from pattern
- text pointer i never decrements
- could use input stream

Constructing the DFA

- include one state for each character in pattern (plus accept state)

* Match transition

if in state j and next char c == pattern.charAt(j) -> go to j+1

	0	1	2	3	4	5
	A	B	A	B	A	C
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6

pattern.charAt(j)

dfa[ ][j] → position in pattern

dfa[ ] → character

* Mismatch Transition : back up if c != pattern.charAt(j)

=> then the last j-1 characters of input are pattern[1...j-1] followed by character c

=> to compute dfa[c][j] => simulate pattern[1...j-1] and take transition c

ex: dfa[A][5] = 1

simulate 'BABA' take transition 'A'

= dfa['A'][3]

dfa['B'][5] = dfa[B][3]

* Running time → seems to require j steps

→ takes only constant time if we maintain state x

Implementation

for each state j

- copy dfa[][x] to dfa[][j]

for mismatch case

- dfa[pat.charAt(j)][j] = j+1 for match

- update x

* Running time : M character accesses
(but space / time proportional to RM)

KMP ⟶ O(M+N) (linear)

Boyer – Moore

- scan characters in pattern from right to left
- can skip as many as M text chars when finding one not in the pattern

How much do we skip?

Case 1) mismatch character not in pattern.

* increment i one character beyond 'T'

before

```
txt   . . . . T L E
pat   N E E D L E
```

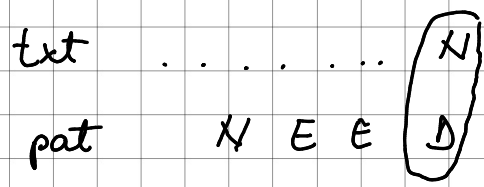
after

```
txt   . . . . . . . . T L E
pat                   N E E D L E
```

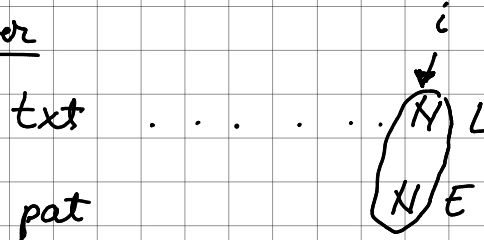
case 2a) Mismatch character in pattern

* align text 'N' with rightmost pattern 'N'

before



after



case 2b) Mismatch character in pattern

=> align with rightmost char might involve backup => we don't want that

=> increment (i) by 1

* you can precompute index of rightmost occurence of character c in pattern (-1 if not in pattern)

* sublinear but in the worst case it ~ N/M

is as bad as the brute force $\sim N M$

Rabin - karp

- basic idea = modular hashing
- compute a hash of pattern characters 0 to $(M-1)$
- if pattern hash = text substring hash => check for a match

pattern

0	1	2	3	4
2	6	5	3	5

$\%$ 997 $= 613$

997 is prime

t_i = text.charAt(i)

$$x_i = t_i \cdot R^{M-1} + t_{i+1} R^{M-2} + .. t_{i+M-1} R^0 \mod Q$$

Horner's Method

Challenge : how to efficiently compute x_{i+1} given we know x_i

$$x_{i+1} = (x_i - t_i R^{M-1}) R + t_{i+M}$$

Monte Carlo version : return if hash match

⇒ low probability (hash collisions)

Las Vegas version : check for substring match if hash match → continue search if false collision

⇒ if Q is a sufficiently large prime (random, about $M \times N^2$) ⇒ the probability of a false collision is about $1/N$