11. String Matching Sources Cormen 985

Intro String matching bliver brugt inden for mange forskellige felter eksempelvis for at søge efter ord eller for at finde specifikke sequences i DNA. Lad os først notere

Alphabet:

$$\sum$$

Set of all strings finite strings:

$$\sum^*$$

Text:

$$T[0..n]$$

Pattern:

$$P[0..m]$$

Vi ønsker at finde et pattern P i T sådan at vi får returneret et offset s hvor P starter fra. Vi kigger på to parametre, navnligt på preproccesing time og matching time.

Naive string matching Loop over hele teksten T og for hvert index se om om P passer på de næste m karaktere.

Pros: Ingen preprocessing time, den går igang med det samme.

Cons: Har en matching køretid på

$$\mathcal{O}((n-m+1)\cdot m)$$

Den smider vigtig information ud, eksempelvis hvis der er et mismatch, kunne den nogle gange springe

$$i \leq m$$

indices over.

Rabin-Karp Antag at alle bogstaver i

$$\sum$$

kan mappes til en numerisk værdi, lad os antage 0..9. Da ønsker vi så an finde en decimal størrelse for

$$P[0..m]$$

og alle subsets af

$$T[i..j] \subset T[0..n], j-i=m$$

. Vi kan omdanne disse til decimaler eksempelvis med Horners rule. (Vi ganger bare de laveste bit med ti og så for hver higher order bit 10 mere. Nemt.)

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + ... + 10(P[0])))$$
$$t_0 = T[m] + 10(T[m-1] + 10(T[m-2] + ... + 10(T[0])))$$

Dette tager os en preprocessing tid på

$$\theta(m)$$

. Nu skal vi matche, vi itterere over alle alle

$$t_s, s \in \{0...(n-m)\}$$

Og ser om de er lige, for at finde den næste

$$t_{s+1}$$

skal vi lave et bit shift hvilken kan gøres i konstant tid ved at

$$t_{s+1} = 10 \cdot (t_s - T[s] \cdot 10^{m-1}) + T[s+m+1]$$

I første parrantes fjerne vi den highest order bit ved at tage foregående string og trække højeste bit fra. Her efter shifter vi alle en til venstre ved at gange med 10 og ligger din nye lowest order bit til.

Men hvad hvis vi ikke kan lave sammenligninger i konstant tid? Vi kan tage modulo af vores decimaler med et primtal p og sammenligne dem. Hvis vi får et hit, bliver vi dog nødt til at chekke for false positives. Hvis vi antager at vi kun får hits et konstant antal gange har vi altså

Preprocess:

$$\theta(m)$$

Matching:

$$\theta((n-m+1) + c \cdot m)$$

Finite Automaton Build up a finite state machine witch transitions. We can represent transitions in a matrix of size

$$|\sum| \cdot m$$

where we have a start state and an acceptence state. Scanning the text, we will try to move through the transitions untill we get to the acceptence state. Når vi fejler på vejen vil vi gerne gå tilbage til det state der er det længste prefix af P.

Preprocess:

$$\theta(|\sum| \cdot m)$$

Matching:

$$\theta(n)$$