Rabin-Karp algoritam

Pretraživanje teksta

Autor: Martina Alilović
Kolegij: Oblikovanje i analiza algoritama
Mentor: prof. dr. sc. Saša Singer

Općenito o pretraživanju teksta

- Pronalaženje u tekstu t odgovarajući par uzorku p (pattern)
- Oznake: n=|t| m=|p|
- Klasična verzija – sekvencijalno pretraživanje – O(m*(n-m+1))
- Pisati ćemo p[i,…,j] da bi označili podstring stringa p koji počinje na poziciji i, i završava na poziciji j
- U proučavanju Rabin-Karp algoritma, radi jednostavnosti, restringirati ćemo se na binarni alfabet: Σ={0,1}

Uvod u Rabin-Karp algoritam

- U prosjeku: O(n+m)
- prije nego što počnemo tražiti uzorak na nekoj poziciji provjerimo možemo li očekivati uzorak na toj poziciji

Primjer: parnost

- Kažemo da uzorak ima parnost 0 ako sadrži paran broj jedinica, odnosno 1 ako sadrži neparan broj jedinica
- Sve podstringove u teksu duljine m (m=|p|) sad možemo podijeliti po parnosti, pa umjesto da uspoređujemo cijele uzorke, sada uspoređujemo samo male dijelove, njihove otiske prstiju (eng. fingerprints)
- S f[i] označimo parnost od t[i,…,i+m-1]. f poprima vrijednosti 0 i 1

Neka je naš uzorak p:

1	1	0	1

m=4

I neka je dan komad nekog teksta:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	1	1	0	0	1	1	1	0	1	1	0	0

f(1) = t[1]+t[2]+t[3]+t[4] (mod 2) = 1

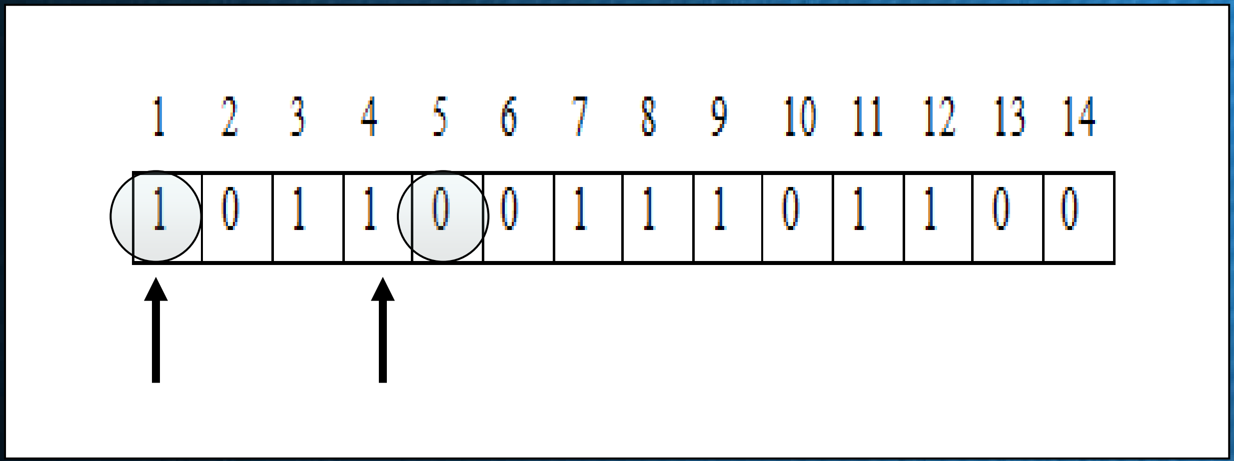
f(p) = 1

i	1	~~2~~	~~3~~	~~4~~	~~5~~	6	7	8	9	~~10~~	~~11~~
f(i)	1	~~0~~	~~0~~	~~0~~	~~0~~	1	1	1	1	~~0~~	~~0~~

Odbacili smo iz razmatranja pozicije u kojima se parnost ne podudara s parnosti našeg uzorka. Sada je dovoljno provjeriti nalazi li se naš uzorak na pozicijama: 1, 6, 7, 8 i 9.

No, za računanje parnosti stringa duljine m treba nam vrijeme O(m). U tom slučaju, nema poboljšanja u odnosu na obično sekvencijalno pretraživanje.

Poboljšanje ipak možemo postići “pametnim” računanjem vrijednosti funkcije parnosti.



Nakon što smo izračunali vrijednost funkcije parnosti za prvi podstring duljine m, za računanje sljedeće nije nam potrebno svih m provjera, dovoljno je pogledati prvi simbol prethodnog podstringa i zadnji simbol novog podstringa. Ukoliko se oni razlikuju mijenja se parnost, u protivnom, vrijednost funkcije parnosti je jednaka kao i u prethodnom stringu.

- Računanje vrijednosti funkcije parnosti: O(1)
- U prosjeku, provjera parnosti eliminira pola podstringova, pa bi petlji trebalo oko m(n-m+1)/2 prolaza.
- Ubrzanje nije dovoljno jer previše vrijednosti dijeli isti otisak.

Hash funkcija

Ako želimo ubrzanje za faktor q, treba nam funkcija takva da:

1) m-bitne stringove povezuje s q različitih vrijednosti (otisaka)

2) uniformno raspoređuje stringove po q različitih vrijednost

3) jednostavno se sekvencijalno računa, odnosno u O(1) vremenu možemo izračunati novu vrijednost funkcije (nakon pomaka na slijedeći podstring)

Ta svojstva nam omogućuju da u prosjeku odbacimo

(q-1)/q vrijednosti. Svojstvo 3) nam garantira da možemo saznati treba li podstring biti eliminiran u konstantnom vremenu.

Dobra funkcija

- m-bitni binarni string pretvorimo u broj u bazi 10 i računamo ostatak modulo q

$$\sum_{j=0}^{m-1} s_j 2^{m-1-j} \bmod q.$$

- Funkcija očito zadovoljava svojstvo 1) jer su vrijednosti iz skupa {0,1,…,q-1}
- Svojstvo 2) ovisi o odabiru q, pokazuje se da odabir prostog broja q>m radi vrlo dobro i zadovoljava 2)

•Svojstvo 3) provjerimo jednostavnom manipulacijom suma

$$
\begin{aligned}
\sum_{j=0}^{m-1} s_{j+1} 2^{m-1-j} \bmod q &= s_m + \sum_{j=0}^{m-2} s_{j+1} 2^{m-1-j} \bmod q \\
&= s_m + 2 \sum_{j=0}^{m-2} s_{j+1} 2^{m-1-(j+1)} \bmod q \\
&= s_m + 2 \sum_{j=1}^{m-1} s_j 2^{m-1-j} \bmod q \\
&= s_m + 2(-2^{m-1} s_0 + \sum_{j=0}^{m-1} s_j 2^{m-1-j}) \bmod q \\
&= s_m + 2(a - 2^{m-1} s_0) \bmod q.
\end{aligned}
$$

Algoritam

- Algoritam traži uzorak p u tekstu t. Vraća najmanji indeks i takav da t[i,…,i+m-1]=p ili -1 ako takav indeks ne postoji.

Input Parameters: p, t
Output Parameters: None

```
rabin_karp_search(p, t) {
  m = p.length
  n = t.length
  q = prime number larger than m
  r = 2^(m-1) mod q
  // computation of initial remainders
  f[0] = 0
  pfinger = 0
  for j = 0 to m - 1 {
    f[0] = 2 * f[0] + t[j] mod q
    pfinger = 2 * pfinger + p[j] mod q
  }
  i = 0
  while (i + m ≤ n) {
    if (f[i] == pfinger)
      if (t[i..i + m - 1] == p) // this comparison takes time O(m)
        return i
    f[i + 1] = 2 * (f[i] - r * t[i]) + t[i + m] mod q
    i = i + 1
  }
  return -1
}
```

$O(m)$

$O(m*(n-m+1))$

U prosjeku:
$O(1/q*m*(n-m+1)$

Složenost algoritma

Najgori slučaj: Sve vrijednosti dijele isti otisak prsta
Algoritam je tada jednak algoritmu sekvencijalnog pretraživanja, ali uz to još smo vremena utrošili na računanje početne vrijednosti f[0] i vrijednosti pfinger.
Ukupna složenost je tada:
O(m(n-m+1)) + O(m), a to je opet **O(m(n-m+1))**

U prosjeku, očekujemo da test (f[i] == pfinger) uspije u prosjeku 1/q puta.
Dakle, imamo O((1/q)*m(n-m+1)) + O(m)
Budući da je q>m, očekivana složenost je **O(m+n).**

Input Parameters: p, t
Output Parameters: None

```
rabin_karp_search(p, t) {
  m = p.length
  n = t.length
  q = prime number larger than m
  r = 2^(m-1) mod q
  // computation of initial remainders
  f[0] = 0
  pfinger = 0
  for j = 0 to m - 1 {
    f[0] = 2 * f[0] + t[j] mod q
    pfinger = 2 * pfinger + p[j] mod q
  }
  i = 0
  while (i + m ≤ n) {
    if (f[i] == pfinger)
      if (t[i..i + m - 1] == p) // this comparison takes time O(m)
        return i
    f[i + 1] = 2 * (f[i] - r * t[i]) + t[i + m] mod q
    i = i + 1
  }
  return -1
}
```

- Za sad smo ignorirali problem traženje prostog broja većeg od m.
- Po Bertrandovom postulatu, za svaki broj m, postoji prost broj između m i 2m, dakle, možemo pronaći prost broj veći od m u vremenu $O(m^{3/2})$.
- U praksi, puno je brže koristiti tehnike za nasumičan odabir velikog prostog broja

Testiranje

N=10

n	Rabin- Karp		Obično pretraživanje	
	m=100	m=500	m=100	m=500
128	0.0004 s		0.0009 s	
256	0.0009 s		0.0019 s	
512	0.0008 s	0.0004 s	0.0037 s	0.0015 s
1024	0.0008 s	0.0005 s	0.01 s	0.0146 s
2048	0.001 s	0.0008 s	0.0121 s	0.0477 s
4096	0.0015 s	0.0011 s	0.0232 s	0.1067 s
8192	0.0015 s	0.0013 s	0.0504 s	0.2253 s
16384	0.0025 s	0.0022 s	0.1025 s	0.4031 s
32768	0.0042 s	0.0045 s	0.1468 s	0.7376 s
65536	0.0079 s	0.0089 s	0.2994 s	1.2472 s
131072	0.0159 s	0.0142 s		
262144	0.0365 s	0.0326 s		
524288	0.0622 s	0.0694 s		

Testiranje

N=10

n	Rabin- Karp			
	m=100	**t/(m+n)**	**m=500**	**t/(m+n)**
128	0,00040	0,00000175		
256	0,00090	0,00000253		
512	0,00080	0,00000131	0,0004	0,00000040
1024	0,00080	0,00000071	0,0005	0,00000033
2048	0,00100	0,00000047	0,0008	0,00000031
4096	0,00150	0,00000036	0,0011	0,00000024
8192	0,00150	0,00000018	0,0013	0,00000015
16384	0,00250	0,00000015	0,0022	0,00000013
32768	0,00420	0,00000013	0,0045	0,00000014
65536	0,00790	0,00000012	0,0089	0,00000013
131072	0,01590	0,00000012	0,0142	0,00000011
262144	0,03650	0,00000014	0,0326	0,00000012
524288	0,06220	0,00000012	0,0694	0,00000013

Monte Carlo Rabin Karp Search

Spomenimo još da se u praksi ponekad koristi i Monte-Carlo verzija Rabin-Karp algoritma.

Takav algoritam oslanja se samo na otiske prstiju da odluči da li se odgovarajući par nalazi na poziciji i.
To jako ubrzava algoritam, sada mu je složenost O(n).

Uočavamo da dva stringa mogu biti različita iako imaju isti otisak, ali pokazuje se da je vjerojatnost da će se to Dogoditi, ako odaberemo slučajan prost broj manji od mn^2 za q, manja od 2.53/n.

Literatura:

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest: *Introduction to Algorithms*
- M.H. Alsuwaiyel: *Algorithms: Design Techniques and Analysis*