Rabin-Karp algoritem, Timen Bobnar

OSNUTEK

# Povzetek \*\*\* Končno \*\*\*

Rabin-karp algoritem je algoritem za iskanje pojavitve določenega niza v nekem večjem nizu. Da bomo predelali ta algoritem bomo najprej predstavil kaj zares je problem, ki ga rešujemo s tem algoritmom. Nato si bomo ogledali zgoščevale funkcije. Na temo zgoščevalnih funkcij si bomo ogledali kaj je sprehajajoča se zgoščevalna funkcija in kaj je lažno ujemanje. Na kratko bomo omenil kaj so ASCII tabele. Na koncu bomo predstavili Rabin-karp algoritem in pokazali delovanje našega algoritma na velikem primeru ter analizirali časovno zahtevnost algoritma.

# Problem\*\*\* Končno \*\*\*

Dobimo dva niza, prvi naj bo dolžine n in v njem iščemo pojavitev drugega niza dolžine m. Prvemu nizu recimo besedilo drugemu pa vzorec. Besedilo je po dolžini zmeraj večje oziroma enako dolžini vzorca. Kot rezultat želimo vrniti pozicijo pojavitve vzorca v besedilu.

Če bi se lotili naivno bi primerjali znak po znak, to bi nam podalo časovno zahtevnost O(n\*m), hkrati primerjamo znake med sabo kar je tudi časovno neučinkovito, saj je za računalnik to kar zahteven proces.

# ASCII tabele

Najprej omenimo kaj so ASCII tabele. ASCII tabele so tabele, ki posameznemu znaku priredi številsko vrednost.

Slika, ki vsebuje besede posnetek zaslona, besedilo

Opis je samodejno ustvarjen

Slika 1: Primer ASCII tabele

Kot vidimo na sliki ima vsak znak enolično prirejeno število. Imamo tudi razširjene ASCII tabele, ki vsebujejo šumnike.

# Zgoščevalna funkcija

Preden predstavimo algoritem, povejmo še kaj je zgoščevalna funkcija (Hash funkcija). To je funkcija, ki nizu priredi številsko vrednost. Vse zgoščevalne funkcije morajo imeti lastnost, da istemu nizu priredi isto vrednost. Obstaja veliko različnih zgoščevalnih funkcij, uporabimo lahko katero koli, vendar mi bomo uporabili:

* - ASCII vrednost črke
* b – baza (običajno število znakov ki jih lahko uporabimo)
* p – neko praštevilo

To zgoščevalno funkcijo bomo uporabili, saj jo najlažje uporabimo na primerih, da se vidi kaj se dogaja.

## Primer uporabe:

Naj bo baza 256 in praštevilo 101.

Uporabimo zgoščevalno funkcijo na nizu »abc«

|  |  |
| --- | --- |
|  | ASCII vrednost |
| »a« | 97 |
| »b« | 98 |
| »c« | 99 |

Torej po formuli dobimo da je:

Uporabimo zgoščevalno funkcijo še na primeru »Python«

|  |  |
| --- | --- |
|  | ASCII vrednost |
| »P« | 80 |
| »y« | 121 |
| »t« | 116 |
| »h« | 104 |
| »o« | 111 |
| »n« | 110 |

Torej po formuli dobimo da je:

# Sprehajajoča se zgoščevalna funkcija\*\*\* Končno \*\*\*

Da bi zmanjšali število operacij tako gledanja v ASCII tabela kot vsa množenja bomo sedaj uporabili sprehajajočo se zgoščevalno funkcijo (rolling hash). Tu spet obstaja veliko različnih, vendar mi bomo uporabili sprehajajočo se zgoščevalo funkcijo, ki je izpeljana iz funkcije iz prejšnjega poglavja.

* - ASCII vrednost črke
* b – baza
* p – neko praštevilo

Dokažimo, da je to res isto. Niz ima indekse od 1 do m, novi niz pa od 2 do m+1.

Vstavimo H(niz) iz prejšnjega poglavja.

=

(

## Primer uporabe:

Naj bo baza 256 in praštevilo 101.

Recimo da imamo niz „abcde“ ter naj bo dolžina vzorca 3.

Za prvi niz „abc“ moramo še zmeraj uporabiti navadno zgoščevano funkcijo.

|  |  |
| --- | --- |
|  | ASCII vrednost |
| »a« | 97 |
| »b« | 98 |
| »c« | 99 |

Torej po formuli dobimo da je:

Za naslednji del torej „bcd“ pa bomo uporabili sprehajočo se zgoščevanlo funkciijo.

|  |  |
| --- | --- |
|  | ASCII vrednosti |
| »a« | 97 |
| »d« | 100 |

Torej po formuli dobimo:

Za naslednji del torej „bcd“ pa bomo uporabili sprehajočo se zgoščevanlo funkciijo.

|  |  |
| --- | --- |
|  | ASCII vrednosti |
| »b« | 98 |
| »e« | 101 |

Torej po formuli dobimo:

# Lažno ujemanje

Kot smo že omenili za posamezen niz zgoščevalna funkcija vedno vrne enako vrednost. Vendar občasno lahko naletimo na lažna ujemanja, to je, ko se zgoščeni vrednosti niza ujemata vendar niza nista enaka. Lažna ujemanja lahko nekoliko kontroliramo z izbiro baze in praštevila, vendar se jih nikoli zagotovo ne znebimo.

## Primer lažnega ujemanja:

Naj bo baza 256 in praštevilo 101. Naš niz pa je »abc for elt in range(5)« v njem bomo računali zgoščene vrednosti za nize dolžine 4.

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava, tipografija

Opis je samodejno ustvarjen

Slika 2: izračunane prirejene vrednosti

Opazimo da ima niz »abc« in »elt« enako zgoščeno vrednost vendar nista enaka torej je prišlo do lažnega ujemanja. Podobno situacijo vidimo na nizih »r e« in »lt «.

## Primer vpliva baze in praštevila:

To bomo prikazali s pomočji kode:

Slika, ki vsebuje besede besedilo, posnetek zaslona, programska oprema

Opis je samodejno ustvarjen

Slika 3: koda za prikaz ujemanj pri določeni bazi in praštevilu

Naredili bomo nekaj kombinacij baze in praštevila ter šteli število ujemanj z nizom »abc«. Iz ASCII tabele bomo vzeli vse znake.

|  |  |  |
| --- | --- | --- |
| praštevilo | baza | Število ujemanj |
| 101 | 256 | 27363 |
| 191 | 256 | 14469 |
| 1009 | 256 | 2735 |
| 6073 | 256 | 459 |
| 15485863 | 256 | 1 |
| 101 | 1 | 27441 |
| 101 | 6073 | 6823 |

Opazimo, da če je naše praštevilo veliko, da se izognemo velikemu število lažnih ujemanj, Potrebno je poudariti da smo dobili 1 samo za niz »abc« če bi uporabili kakšen drugačen niz bi prišlo do več ujemanj. Ko izbiramo bazo in praštevilo je torej zaželjeno uporabiti veliko praštevilo, vendar računanje z velikimi praštevili je časovno zalo potrato, tako da je potrebno vzeti neko srednjo pot.

# Rabin-Karp algoritem\*\*\* Končno \*\*\*

Radi bi vrnili indeks pojavitve vzorca v besedilu. Uporabili bomo zgoščevalne funkcije. Torej najprej izračunamo vrednost zgoščevalne funkcije za vzorec. Vzorec je dolžine m. Sedaj se sprehajamo po besedilu z oknom dolžine m (vsakič se premaknemo za eno mesto). Najprej z zgoščevalno funkcijo izračunamo vrednost prvega okna, naslednja okna računamo s sprehajajočo se zgoščevalno funkcijo. Ko pride do ujemanja vrednosti zgoščevalne funkcije za vzorec in okna v besedilu, vemo da ali je prišlo do pravega ujemanja ali pa do lažnega ujemanja. Da prevrimo do kakšnega ujemanja je prišlo, sedaj primerjamo vzorec in okno znak po znak. V primeru, ko ne pride do ujemanja nadaljujemo. V primeru ko pride do ujemanja, si zapomnimo indeks prvega znaka v oknu. Na koncu vrnemo vse indekse, kjer se niza ujemata.

A close-up of a white background

Description automatically generated

Slika 4: Primer

Kot vidimo v primeru je ATC vzorec in ABGJPATCNKAATCH besedilo. Kot rezultat bomo torej vrnili 5 in 12, saj je to indeks kjer se začne ujemanje.

# Uporaba na primeru \*\*\* Končno \*\*\*

Kot osnovne podatke dobimo:

* vzorec: abc
* besedilo: jvnewjoxabcfk ptrphjabcj
* baza: 256
* praštevilo: 101

Najprej poračunamo prirejeno vrednost za vzorec:

|  |  |
| --- | --- |
|  | ASCII vrednost |
| »a« | 97 |
| »b« | 98 |
| »c« | 99 |

Sedaj poračunamo prirejeno vrednost za prvo okno besedila torej za »jvn«

|  |  |
| --- | --- |
| Korak 1 | ASCII vrednost |
| »j« | 106 |
| »v« | 118 |
| »n« | 110 |

54 ni enako 90 torej nadaljujemo na okno »vne«. To okno zdaj že računamo s sprehajajočo se zgoščevalno funkcijo.

|  |  |
| --- | --- |
| Korak 2 | ASCII vrednost |
| »j« | 106 |
| »e« | 101 |

Spet ni prišlo do ujemanja torej nadaljujemo. Tako pridemo do koraka 6.

|  |  |
| --- | --- |
| Korak 6 | ASCII vrednost |
| »w« | 119 |
| »x« | 120 |

Sedaj je prišlo do ujemanja, saj smo dobili 90, torej preverimo nize, ker „abc“ ni enako „jox“ je prišlo do lažnega ujemanja in nadaljujemo.

Tako nadaljujemo dokler ne pridemo do koraka 9.

|  |  |
| --- | --- |
| Korak 9 | ASCII vrednost |
| »x« | 120 |
| »c« | 99 |

Spet je prišlo do ujemanja. Ko preverimo znak po znak vidimo, da pride do pravega ujemanja torej si zapomnemo indeks, tega lahko dobimo iz koraka, ki je trenutno 9-ti. Shranimo indeks 9 oziroma 8 odvisno od tega ali začnemo šteti z 0 ali 1.

Spet nadaljujemo do koraka 21.

|  |  |
| --- | --- |
| Korak 9 | ASCII vrednost |
| »j« | 106 |
| »c« | 99 |

Spet je prišlo do ujemanja. Ko preverimo znak po znak vidimo, da pride do pravega ujemanja torej si zapomnemo indeks tega lahko dobimo iz koraka ki je trenutno 21-ti. Shranimo indeks 21 oziroma 20 odvisno od tega ali začnemo šteti z 0 ali 1.

Naredimo še en korak in vrnemo indeksa 9 in 21 oziroma 8 in 20.

# Časovna zahtevnost Rabin Karp algoritme \*\*\* Končno \*\*\*

Najprej je potrebno analizirati časovne zahtevnosti zgoščevalnih funkcij.

Časovna zahtevnost navadne zgoščevalne funkcije je O(m), saj moramo za vsak element v oknu pogledati v ASCII tabelo. Poleg pogledov v ASCII tabelo je časovna zahtevnost tudi odvisna od praštevila in baze ter od posamezne zgoščevalne funkcije.

Časovna zahtevnost sprehajajoče se zgoščevalne funkcije je O(1). Do take časovne zahtevnosti pride, saj pri sprehajajoči se zgoščevalni funkciji imamo neodvisno od dolžine okna zmeraj 2 vpogleda v ASCII tabelo in nekaj računskih operacij. Seveda tudi tukaj je časovna zahtevnost odvisna od določene sprehajajoče se zgoščevalne funkcije.

Časovna zahtevnost Rabin Karp algoritma. V večini primerov se hash vrednost izračuna v O(1) časa, primerjava dveh hash vrednosti je tudi O(1), če se naš vzorec pojavi malokrat in je naše besedilo dolgo n znakov dobimo časovno zahtevnost O(n) + O(m) za izračun vzorca. V večini primerov je m veliko manjši, kot n torej lahko rečemo da je časovna zahtevnost O(n). Poglejmo še najslabši primer. Najslabši primer se zgodi ko imamo ujemanja na vsakem koraku v tem primeru, dobimo časovno zahtevnost O(n\*m) + O(m).

# Viri (opomba – ta razdelek mora biti obvezno končen!!!)

* slika1: https://simple.m.wikipedia.org/wiki/File:ASCII-Table-wide.svg