

**Rabin-Karp algoritem**

Timen Bobnar

Ljubljana, 2024

# Povzetek

Rabin-Karp algoritem je algoritem s katerim ugotavljamo kje se določen niz pojavi v nekem večjem nizu. V prispevku si najprej točno ogledamo kaj zares je problem, ki ga rešujemo s tem algoritmom. Nato si bomo ogledali zgoščevalne 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

Dana sta dva niza. Prvi naj bo dolžine *n*. V njem iščemo pojavitev drugega niza dolžine *m*. Prvemu nizu rečemo ***besedilo***, drugemu pa ***vzorec***. Besedilo je po dolžini zmeraj večje oziroma enako dolžini vzorca. Velja torej *n >= m*. Kot rezultat želimo vrniti mesto, kjer se vzorec pojavi v besedilu. Če se vzorec v besedilu ne pojavi kot položaj, vrnemo -1 (to je odvisno od implementacije)

Primer:

|  |  |  |
| --- | --- | --- |
| Vzorec | Besedilo | Rešitev |
| »nov« | »novinbenov« | 0, 7 |
| »Gol« | »novinbenov« | -1 |
| »novica« | »novincanov« | -1 |

Kot vidimo, se mora niz pojaviti v celoti, brez morebitnih vmesnih znakov. Če se niz pojavi večkrat vrnemo vse indekse kjer se začne skladanje.

# 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 in druge znake.

Sedaj se veliko uporablja UTF kodiranje črk oziroma znakov. Pri tej seminarski nalogi bomo uporabljali ASCII tabele, saj je bilo to lažje implementirati. Seveda se sama ideja rešitve ne spremeni.

# 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 priredjo isto vrednost. Obstaja veliko različnih zgoščevalnih funkcij. Lahko uporabimo katerokoli, vendar pri tej implementaciji bomo uporabili naslednjo:

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

To zgoščevalno funkcijo bomo uporabili, saj jo najlažje uporabimo na primerih.

## Primer uporabe:

Izberimo 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

Pri navadni zgoščevalni funkciji, če imamo niz dolžine *n* imamo *n* vpogledov v ASCII tabelo in nato še *n* množenj in seštevanj.

Da bi zmanjšali število operacij, bomo sedaj uporabili sprehajajočo se zgoščevalno funkcijo (rolling hash). S sprehajajočo se zgoščevalno funkcijo sedaj zmanjšamo število operacij na dva vpogleda v ASCII tabelo, dve množenji in dve seštevanji. Obstaja veliko različnih sprehajajočih se zgoščevalnih funkcij. Uporabili bomo sprehajajočo se zgoščevalno funkcijo, ki jo bomo izpeljali iz zgoščevalne funkcije.

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

Dokažimo, da je sprehajajoča se zgoščevalna funkcija enaka navadni zgoščevalni funkciji.

Imejmo besedilo sestavljeno iz *m+1* znakov. Niz je sestavljen iz prvih *m* znakov torej od znaka z indeksom 1 do znaka z indeksom *m*. Za ta niz imamo že izračunano zgoščeno vrednost, ki jo označimo s *H(niz)*. Radi bi izračunali zgoščeno vrednost za niz, ki se začne z indeksom 2 in konča z indeksom *m+1*. Niz in novi niz imata vse znake enake, razen tiste z indeksi 1 in *m+1*.

Niz smo izračunali po formuli:

Vstavimo *H(niz)* sedaj v formulo za sprehajajočo se zgoščevalno funkcijo.

=

(

Kot vidimo se formula preoblikuje v našo osnovno formulo za zgoščevalno funkcijo vendar za niz od indeksa 2 do indeksa m+1. Torej če v sprehajajočo se zgoščevalno funkcijo vstavimo zgoščeno vrednost starega niza bomo dobili kot rezultat zgoščeno vrednost novega niza.

## Primer uporabe:

Naj bo baza 256 in praštevilo 101.

Recimo, da imamo niz „abcde“ in dolžino vzorca 3.

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

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

Torej po formuli dobimo da je:

Za naslednji niz torej „bcd“ pa bomo uporabili sprehajajočo se zgoščevalno funkcijo.

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

Torej po formuli dobimo:

Za naslednji del, torej „cde“, pa bomo uporabili sprehajajočo se zgoščevalno funkcijo.

|  |  |
| --- | --- |
|  | 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 pa lahko občasno pridemo do lažnega ujemanja. Lažno ujemanje se zgodi, 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 moremo znebiti.

## 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 3.

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 na pojavitve lažnega ujemanja:

To bomo prikazali s pomočjo naslednje 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 se velikemu številu lažnih ujemanj izognemo, če je naše praštevilo veliko. Potrebno je poudariti, da smo dobili 1 ujemanje samo za niz »abc«, če bi uporabili kakšen drugačen niz bi lahko prišlo do več ujemanj. Opazimo da z večanjem praštevila se število ujemanj manjša. Vendar je računanje z velikimi praštevili je časovno zalo potrato, tako da je potrebno ubrati neko srednjo pot.

# Rabin-Karp algoritem

Rabin-Karp algoritem je algoritem s katerim ugotavljamo kje se določen niz (***vzorec***) pojavi v nekem daljšem nizu (***besedilo***). Besedilo je po dolžini zmeraj daljše oziroma enako dolžini vzorca. Velja torej *n >= m,* kjer *n* dolžina besedila in *m* dolžina vzorca . Radi bi vrnili *indeks*, kjer se začne pojavitev vzorca v našem besedilu.

V algoritmu si pomagamo z računanjem zgoščenih vrednosti. Najprej vzorcu priredimo zgoščeno vrednost, recimo, da je ta vrednost *Ѳ*. Sprehajali se bomo po našem besedilu z oknom dolžine *m*. Vsakič, ko se premaknemo, spremenljivki *indeks* prištejemo 1. Torej, v prvem koraku imamo okno, ki zajema prvih *m* znakov (od 0 do *m*-1) ter *indeks* 0. V drugem koraku zajamemo znake od 1 do *m* in *indeks* je 1. Tako se sprehajamo dokler je lahko naš vzorec še v celoti v besedilu.

Ob prvem koraku naše zanke izračunamo prirejeno vrednost našemu vzorcu z zgoščevalno funkcijo. Pri vseh ostalih korakih prirejeno vrednost izračunamo s pomočjo sprehajajoče se zgoščevalne funkcije.

Vsakič, ko izračunamo prirejeno vrednost, jo primerjamo z vrednostjo Ѳ. V primeru, ko se vrednosti ne ujemata nadaljujemo, sicer pa vzamemo naš vzorec. Iz našega besedila vzamemo *m* znakov in sicer od vrednosti, ki je shranjena v spremenljivki *indeksa* naprej. Znakom iz besedila ohranimo vrstni red. Temu delu, ki smo ga vzeli iz besedila rečemo okno. Nato primerjamo vzorec in okno. V primeru, ko so znaki v vzorec in v okno različni, nadaljujemo po zanki, saj je prišlo do lažnega ujemanje. V primeru, ko pa pride do ujemanja med vzorcem in oknom, pa si zapomnimo vrednost, ki jo hranimo v spremenljivki *indeks*.

Če na koncu zanke ni prišlo do nobenega ujemanja, samo vrnemo vrednost -1. V nasprotnem primeru vrnemo shranjene indekse.

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

Pri prikazu primera bomo uporabili:

* baza: 256
* praštevilo: 101

Kot osnovne podatke dobimo:

* vzorec: abc
* besedilo: jvnewjoxabcfk ptrphjabcj

Najprej izračunamo zgoščeno vrednost za vzorec:

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

Sedaj izračunamo zgoščeno vrednost za prvo okno besedila torej za »jvn«

|  |  |
| --- | --- |
| Korak 1 / indeks 0 | ASCII vrednost |
| »j« | 106 |
| »v« | 118 |
| »n« | 110 |
| Ujemanja | [-1] |

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

|  |  |
| --- | --- |
| Korak 2 / indeks 1 | ASCII vrednost |
| »j« | 106 |
| »e« | 101 |
| Ujemanja | [-1] |

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

|  |  |
| --- | --- |
| Korak 6 / indeks 5 | ASCII vrednost |
| »w« | 119 |
| »x« | 120 |
| Ujemanja | [-1] |

Sedaj je prišlo do ujemanja, saj smo dobili 90. Torej primerjamo niz „abc“ in okno „jox“ znak po znak. Ker nista enaka, je prišlo do lažnega ujemanja in nadaljujemo.

Tako nadaljujemo dokler ne pridemo do koraka 9.

|  |  |
| --- | --- |
| Korak 9 / indeks 8 | ASCII vrednost |
| »x« | 120 |
| »c« | 99 |
| Ujemanja | [-1] |

Spet je prišlo do ujemanja. Ko preverimo znak po znak, vidimo, da pride do pravega ujemanja. Zapomnimo si torej *indeks*, ki je trenutno 8.

Spet nadaljujemo do koraka 21.

|  |  |
| --- | --- |
| Korak 21 / indeks 20 | ASCII vrednost |
| »j« | 106 |
| »c« | 99 |
| Ujemanja | [8] |

Spet je prišlo do ujemanja. Ko preverimo znak po znak, vidimo, da pride do pravega ujemanja. Zapomnimo si torej *indeks*, ki je trenutno 20.

|  |  |
| --- | --- |
| Ujemanja | [8, 20] |

Naredimo še en korak in končamo, saj bi naslednje okno vsebovalo samo 2 znaka.

Na koncu izpišemo indekse 8 in 20.

# Časovna zahtevnost Rabin-Karp algoritme

## Časovna analiza zgoščevalnih funkcij:

Naj bo *m* dolžina okna. Osnovna operacija pa vpogledi v ASCII tabelo.

Časovna zahtevnost navadne zgoščevalne funkcije je *O(m)*, saj moramo za vsak element v oknu pogledati dolžine *m* v ASCII tabelo. Torej imamo *m* vpogledov.

Časovna zahtevnost sprehajajoče se zgoščevalne funkcije je *O(1)*, saj imamo za vsako novo okno 2 vpogleda v ASCII tabelo. En vpogled je za element, ki zapusti okno in en vpogled za element, ki vstopi v okno.

## Časovna analiza za Rabin-Karp algoritme:

Naj bo *m* dolžina okna in *n* dolžina niza. Osnovna operacija naj bo primerjava tako zgoščenih vrednosti kot posameznih črk, ko pride do ujemanja.

Najboljši primer je, ko nikoli ne pride do ujemanja tako lažnega kot pravega. V tem primeru bomo imeli za primerjati *n-m*, torej *n-m* primerjav oziroma operacij. Časovna zahtevnost je torej *O(n-m)*.

Najslabši primer pa je, ko na vsakem koraku pride do ujemanja. V tem primeru bomo imeli primerjanja zgoščenih vrednosti in primerjanja okna in vzorca. Primerjanje zgoščenih vrednosti nam poda *n-m* operacij. Primerjanje vzorca in okna nam poda *m* primerjav. Torej, če to združimo, dobimo časovno zahtevnost enako *O((n-m)\*m)*.

# Viri

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