

**Rabin-Karp algoritem**

Timen Bobnar

Ljubljana, 2024

# Povzetek

Rabin-Karp algoritem je algoritem, s katerim ugotavljamo, kje se v nekem večjem nizu pojavi določen niz. V prispevku si najprej točno ogledamo definicijo problema ter zgoščevalne funkcije. Sledita razdelka o sprehajajoči se zgoščevalni funkciji in o lažnem ujemanju. Sledi razdelek kjer predstavimo Rabin-Karp algoritem. Za razdelkom, kjer Rabin-Karp algoritem predstavimo  v celoti , je na vrsti predstavitev njegovega delovanja na velikem primeru.  V zadnjem razdelku pa podamo še analizo časovne zahtevnosti 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 vedno daljše ali vsaj enako dolgo kot vzorec. Velja torej *n >= m*. Kot rezultat želimo vrniti vsa mesta, kjer se naš vzorec pojavi v besedilu. Če se vzorec v besedilu ne pojavi, vrnemo -1 (ali sprožimo izjemo ali na kak drug način javimo - 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 ujemanje.

# Zgoščevalna funkcija

Zgoščevalna funkcija (hash funkcija) je funkcija, ki nizu priredi številsko vrednost.  Praviloma je ta vrednost naravno število z določenega intervala, saj na ta način zgoščeno vrednost lahko uporabimo kot indeks v neki tabeli. Vse zgoščevalne funkcije morajo imeti lastnost, da istemu nizu priredjo isto vrednost. Obstaja veliko različnih zgoščevalnih funkcij. Lahko uporabimo poljubno, a pri tej implementaciji uporabljamo naslednjo:

* - ASCII vrednost črke
* b – baza (običajno število različnih znakov, ki lahko nastopajo v nizu, torej velikost abecede iz katere tvorimo nize)
* p –praštevilo

## Primer uporabe:

Izberimo si bazo 256 in praštevilo 101.

Uporabimo zgoščevalno funkcijo na nizu »abc«

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

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 |

# Sprehajajoča se zgoščevalna funkcija

Če uporabimo standardno obliko te zgoščevalne funkcije, moramo pri nizu dolžine *n* izvesti *n* vpogledov v ASCII tabelo in nato še *n* množenj in seštevanj.

Da zmanjšamo število operacij, uporabimo sprehajajočo se zgoščevalno funkcijo (rolling hash). Ideja te funkcije je, da iz prej naračunane vrednosti izračunamo novo zgoščeno vrednost. Ta postopek sloni na dejstvu, da pri algoritmu primerjamo zgoščene vrednosti nizov, ki so si zelo podobni (nov niz dobimo z dodajanjem znaka na začetek in brisanjem s konca niza).

Obstaja več različnih sprehajajočih se zgoščevalnih funkcij. Uporabili bomo sledečo sprehajajočo se zgoščevalno funkcijo:

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

S to sprehajajočo se zgoščevalno funkcijo zmanjšamo število operacij na dva vpogleda v ASCII tabelo, dve množenji in dve seštevanji.

Dokažimo, da je vrednost sprehajajoča se zgoščevalne funkcije enaka vrednosti navadne zgoščevalne funkcije.

Naj bo 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 tistih z indeksoma 1 in *m+1*.

Vrednost niza izračunamo 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. Ko v sprehajajočo se zgoščevalno funkcijo vstavimo zgoščeno vrednost starega niza, kot rezultat dobimo zgoščeno vrednost novega niza.

## Primer uporabe:

Naj bo baza 256 in praštevilo 101, niz „abcde“ in dolžino vzorca 3.

Za prvi podniz „abc“ moramo uporabiti navadno zgoščevalno funkcijo.

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

Po formuli dobimo, da je:

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

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

Po formuli dobimo:

Naslednji niz je „cde“. Za izračun njegove zgoščene vrednosti spet uporabimo sprehajajočo se zgoščevalno funkcijo.

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

Dobimo:

# Lažno ujemanje

Zgoščevalna funkcija za dani niz vedno vrne enako vrednost. Lahko pa se zgodi, da imata dva različna niza enako zgoščeno vrednost. Temu pravimo lažno ujemanje. Na pogostost lažnih ujemanj lahko nekoliko vplivamo z izbiro baze in praštevila, vendar se jih nikoli ne moremo zagotovo znebiti.

## Primer lažnega ujemanja:

Naj bo baza 256 in praštevilo 101. Vzemimo niz »abc for elt in range(5)«. Izračunajmo zgoščene vrednosti za vse podnize dolžine 3.

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

Opis je samodejno ustvarjen Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava, številka

Opis je samodejno ustvarjen

Slika 2: izračunane zgoščene vrednosti

Opazimo da imata niza »abc« in »elt« enako zgoščeno vrednost. Torej je prišlo do lažnega ujemanja. Podobno se zgodi pri nizih »r e« in »lt «.

## Primer vpliva baze in praštevila na pojavitve lažnega ujemanja

Pokažimo, da izbira baze in praštevila vpliva na to, koliko lažnih ujemanj najdemo. V ta namen si oglejmo naslednjo kodo

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

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, ter naredili vse možne kombinacije dolžine 3.

|  |  |  |
| --- | --- | --- |
| 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 uporabimo veliko praštevilo. Na prvi pogled je smiselno, da bi kot praštevilo uporabili 15485863, saj bi potem prišlo največ do enega ujemanja. A potrebno je poudariti, da ima samo eno ujemanje le niz »abc«. Če bi opazovali lažno ujemanje s kakšnim drugačnim nizom, bi lahko pri izbiri tega praštevila lahko prišlo do več ujemanj.

A tudi če eksperimentiramo z drugimi nizi, opazimo, da se z večanjem praštevila število ujemanj manjša. Vendar je računanje z velikimi praštevili časovno zelo potratno. Ko izbiramo bazo, se je torej potrebno odločiti, ali hočemo imeti več ujemanj in je zato računanje hitrejše ali pa manj ujeman in je zato računanje počasnejše.

# Rabin-Karp algoritem

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

V algoritmu si pomagamo z računanjem zgoščenih vrednosti. Najprej vzorcu priredimo zgoščeno vrednost. Označimo jo z *Ѳ*. Po besedilu se torej sprehajamo z oknom dolžine *m*. Vsakič, ko se premaknemo, spremenljivki *indeks* prištejemo 1. 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 okno še v celoti v besedilu.

Ob prvem koraku z zgoščevalno funkcijo izračunamo vrednost niza v oknu našemu vzorcu. Pri vseh ostalih korakih vrednost okna izračunamo s pomočjo sprehajajoče se zgoščevalne funkcije.

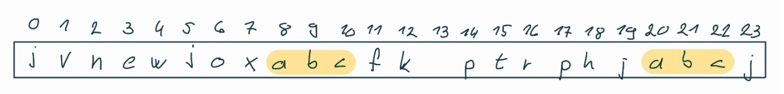
Vsakič, ko izračunamo prirejeno vrednost, jo primerjamo z vrednostjo Ѳ. V primeru, ko se vrednosti ne ujemata, nadaljujemo. Če se zgoščeni vrednosti ne ujemata, namreč ni mogoče, da bi bila pripadajoča niza enaka.

Poglejmo, kaj se zgodi, če se zgoščeni vrednosti ujemata. Načeloma naj bi se potem ujemala tudi niza v oknu in vzorec. A zaradi lažnega ujemanja moramo vseeno izvesti primerjanje znak po znak. Če se niz v oknu in vzorec ujemata tudi pri primerjanju znak po znak, si zapomnimo vrednost, ki jo hranimo v spremenljivki *indeks*.

.

# Uporaba na primeru

V primeru na Sliki 4 je abc vzorec in jvnewjoxabcfk ptrphjabcj besedilo. Rezultat je 8 in 20, saj sta to indeksa, kjer se začne ujemanje vzorca s podnizom v besedilu.



Slika 4: Primer

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 |

Iščemo torej položaj okna, katerega zgoščena vrednost pripadajočega niza je tudi 90.

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 znak po znak primerjamo niza „abc“ in „jox“. Ker nista enaka, je prišlo do lažnega ujemanja. Zato nadaljujemo.

Pri naslednjih korakih ujemanja ni. Zato 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 se tudi tokrat niza zares ujemata. Zapomnimo si torej *indeks*, ki je trenutno 20.

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

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

Algoritem je torej našel indeksa 8 in 20.

# Časovna zahtevnost Rabin-Karp algoritma

## Časovna analiza zgoščevalnih funkcij

Naj bo *m* dolžina okna. Kot osnovno operacijo štejmo vpoglede v ASCII tabelo.

Časovna zahtevnost izračuna vrednosti navadne zgoščevalne funkcije je *O(m)*, saj moramo za vseh *m* znakov pogledati v ASCII tabelo.

Časovna zahtevnost izračuna 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 algoritem

Časovno analizo bomo opravili le za najboljši in najslabši primer. Najboljši primer je, ko nikoli ne pride do ujemanja zgoščenih vrednosti, tako lažnega kot pravega. V tem primeru bomo imeli m operacij za izračun zgoščene vrednosti vzorca. Število operacij za izračun zgoščene vrednosti besedila izračunamo tako, imamo m operacij za izračun prvega okna. Za vsako naslednje okno potrebujemo O(1) operacij, teh oken pa je n-m. Ker nikoli ne pride do ujemanja zgoščenih vrednosti je torej naša časovna zahtevnost

Najslabši primer pa je, ko na vsakem koraku pride do ujemanja. V tem primeru bomo imeli m operacij za izračun zgoščene vrednosti vzorca. Število operacij za izračun zgoščene vrednosti besedila izračunamo tako, imamo m operacij za izračun prvega okna. Za vsako naslednje okno potrebujemo O(1) operacij, teh oken pa je n-m. Ker na vsakem koraku pride do ujemanja imamo na vsakem koraku tudi m primerjav. Torej je naša časovna zahtevnost enaka .

# Viri

*Geeks for geeks.* 16. februar 2024. https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/.

*Programiz.* 16. februar 2024. https://www.programiz.com/dsa/rabin-karp-algorithm.

*Wikipedia.* 16. februar 2024. https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp\_algorithm.