Pozdravljeni jaz sem Timen Bobnar in ….. . Danes vam bova prestavila RABIN-KARP algoritem.

# Problem

S tem algoritmom iščemo ali nek niz znakov vsebuje nek drug manjši niz znakov. Torej kot vhodne podatke dobimo 2 niza večji je dolžine n krajši je dolžine m. Kot izhodni podatek pa želimo ideks oziroma pozicijo kje se je vzorec dolžine m pojavil v vzorcu dolžine n(v textu).

Oglejmo si en primer:

Kot vhodne podatke dobimo tekst: ABGJPATCNKAATCH in vzorec ATC. Kot izhodne podatke želimo dobiti : 5 in 12.

Omenimo še kaj se zgodi če vzorca ne najdemo v našem textu. V tem primeru samo na nek način povemo da vzorca ni textu naprimer return not found

# Motivacija

# Algoritem

Pa poglejmo kaj se dogaja z našim algoritmom. Osnovna ideja je zelo preprosta in sicer. Sprehodi se čez text in na vsakem koraku primerjaj z vzorcem in mi bomo tudi to naredili v tem algoritmu vendar na malo boljši način.

Vemo da v računalniku veliko lažje primerjamo 2 števili kot dva niza saj primerjava dveh števil ima konstantno časovno zahtevnost dveh niov pa cca O(mn)+O(m). To znanje bomo sedaj uporabili.

Imamo naš niz dolžine m in mu priredimo Hash vrednost. In si ji zapomnemo.

Sedaj se sprehodimo čez text in na vsakem koraku izračunamo hash vrednost oziroma »nizu znakov priredimo haš vrednost« . Tukaj je potrebno omeniti, da so te prirejene hash vrednosti za iste nize enake ter da lahko dobimo enakost pri popolnoma različnih nizih.

Ko se hash vrednosti ujamejo imamo torej možnost da se je zares zgodila ponovitev torej v tem primeru zares preverimo »črko po črko« ali sta niza enaka.

Naš alogitem še optimiziramo z Rolling Hash.

Oglejmo si kratek primer:

Brez rolling hash:

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

Opis je samodejno ustvarjen

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

Opis je samodejno ustvarjen

Z uporabo rooling hush

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

Opis je samodejno ustvarjen

# Hash vrednosti(zgoščene vrednosti)

# Rolling hash

Kot smo že omenili bomo naš algoritme optimizirami z rolling hash.

Zakaj??

Opazimo da se večji del niza ko računamo vrednost ponovi le en znak odstranimo in en znak vstvaimo. Zato bomo uporabili eno izmed Rolling hash funkcij saj je veliko različih.

## Polynomial rolling hash:



Tukaj je a neka konstanta ter c znak ter vse delamo v mofulu n.

V tem primeru se lahko prirejamo nove hash vrednosti samo z množenjem/delejenjem celotnega h z a.

Poznamo tudi Rabin fingerprint ki delujo podobno kot ta ki ga bomo uporabili

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

Opis je samodejno ustvarjen

Večji p bolše je. V tej kodi je H(c) zgoščene vrednost od niza c malo še pvem o kodi.

Oglejmo si to računanje na nekaj primerih.

Primer ko dobimo isto haš vrednost za različnega in en malo daljši.

# Lažno ujemanje

# Časovna zahtevnost

Časovna zahtevnost Rabin-Karp algoritma je O(n+m).

V najslabšem slučaju je O(n\*m), ki bi ga dobili takrat ko bi se vse hash vrednosti ujemale. Takrat bi mogli preverjat črko po črko za vsak podniz.

Kako pridemo do te časovne analize:

Primerjava hash vrednosti kot že omenjeno je O(1).

Časovna zahtevnost izračuna rolling hasha po formuli \(H(c') = ((H(c) \cdot b - c\_1 \cdot b^{(m-1)}) + c\_{m+1}) \mod p\) je odvisna od več dejavnikov, vključno z:

1. \*\*Izračun hasha za prvotno okno \(H(c)\):\*\*

- Časovna zahtevnost za izračun hasha za prvotno okno je odvisna od uporabljene hashe funkcije. Če je h(d) časovna zahtevnost hasiranja za določen del besedila, potem je ta del časovne zahtevnosti odvisen od h(d).

2. \*\*Izračun hasha za novo okno \(H(c')\):\*\*

- Izračun hasha za novo okno vključuje množenje s faktorjem \(b\), odvzem \(c\_1 \cdot b^{(m-1)}\), in dodatek \(c\_{m+1}\). Ta operacija je konstantna časovna operacija in ni odvisna od velikosti besedila. Torej, časovna zahtevnost te operacije je \(O(1)\).

3. \*\*Uporaba modulo operacije:\*\*

- Modulo operacija (\(\mod p\)) se uporablja za omejitev vrednosti hasha na določeno število (v tem primeru \(p\)). Časovna zahtevnost modulo operacije je ponavadi konstantna in ne odvisna od velikosti števil.

Skupna časovna zahtevnost za izračun rolling hasha je tako pogosto zapisana kot \(O(1)\), kar pomeni, da je časovna zahtevnost konstantna in ne odvisna od velikosti besedila. To velja, če se uporablja učinkovita implementacija množenja, odvzema in dodajanja, ter če modulo operacija deluje v konstantnem času.

Torej izračun in primerjava je oboje O(1). Ko to vemo opazimo da ima teks dolžine n, n izračunanih Hash vrednosti in m primerjav torej sledi O(n+m) kar je O(n) .

V najslabšem slučaju je O(n\*m), ki bi ga dobili takrat ko bi se vse hash vrednosti ujemale. Takrat bi mogli preverjat črko po črko za vsak podniz.

# Primer

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

Opis je samodejno ustvarjen

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

Opis je samodejno ustvarjen

In the orchard, the apple trees stood tall. Apples, with their crisp texture and sweet flavor,

were a delight for everyone. As the wind blew, apples fell to the ground, creating a carpet of

red and green. The aroma of ripe apples filled the air. It was a picturesque scene, with the

apple orchard extending as far as the eye could see.

Apple

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

Opis je samodejno ustvarjen

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

Opis je samodejno ustvarjen