Kodutöö 4

Taiger Kala

Ülesanne 1: Räsimine

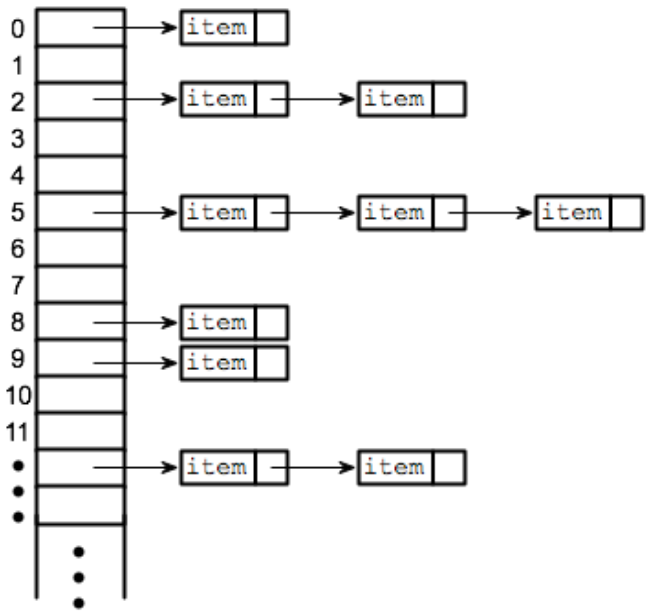
Räsimine tähendab ühesuunalist tehet, kus samast sõnast (paroolist, dokumendist) moodustub alati ühesugune fikseeritud suurusega väärtus, kuid teistpidi tegevus on võimatu – räsist pole võimalik tuletada algset sõnumit. Näiteks, kui parool on "**qwerty**", siis moodustuv räsi näeb välja selline: "**d8578edf8458ce06fbc5bb76a58c5ca4**".

(https://blog.ria.ee/rasist-ja-rasimisest/)

Hea räsifunktsioonil on mitu omadust, et tagada efektiivsus. Võtmed tuleb genereerida nii, et väärtused räsitabelis oleksid jaotatud võrdselt. Samuti peab sama sisend genereerima alati sama räsi. Kõige olulisem on genereerida unikaalseid võtmeid ja need ka võrdselt ära jaotada, et vältida põrkeid ning see tagab räsitabeli efektiivsuse ja kiiruse.

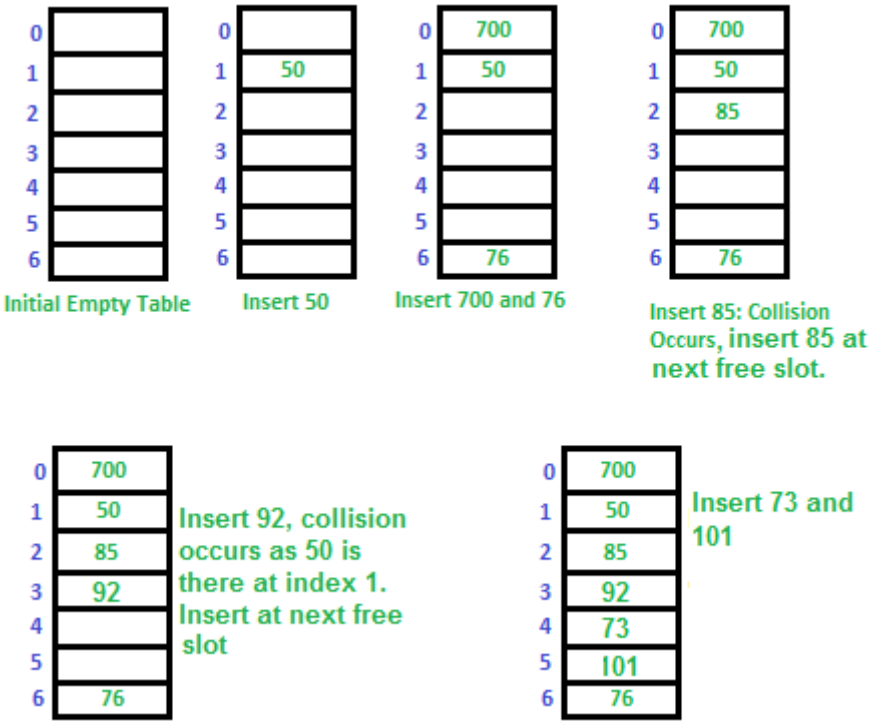
Vahel juhtub räsimise käigus olukord, kus genereeritud võtmetel on sama väärtus, seda nimetatakse põrkeks. Põrke lahendamiseks kasutatakse kas Avatud adresseerimist (open addressing) või eraldi aheldamist (separaate chaining).

Separate Chaining tööpõhimõte on, et räsitabeli iga pilu on linked list, kus andmesõlm koosneb kahest osast. Üks sisaldab andmeid ja teine viitab järgmisele sõlmele.



Joonis 1 https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/

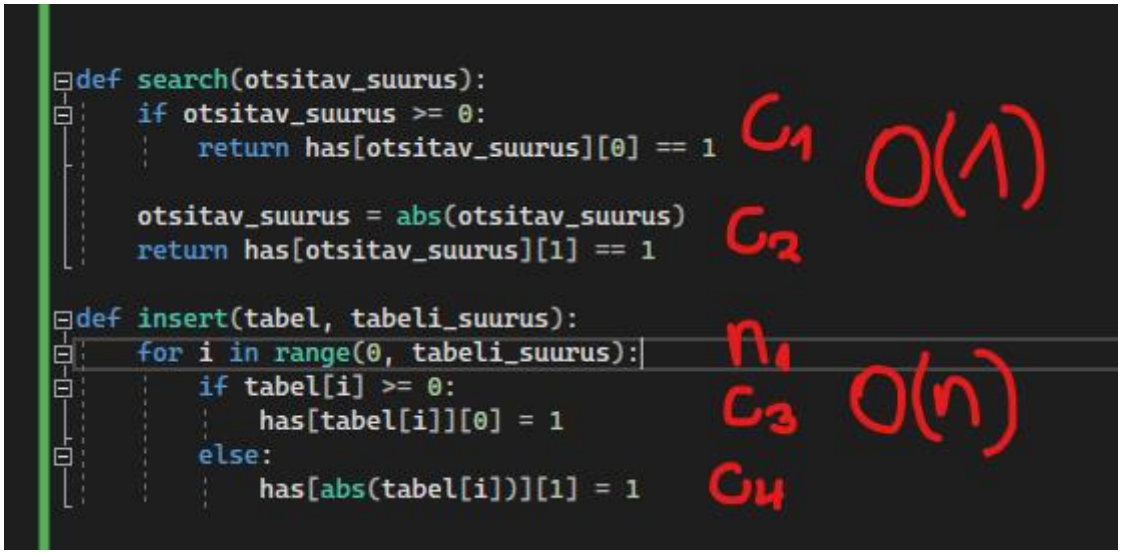
Open Adressingu tööpõhimõte, et kõik elemendid on salvestatud räsitabelis endas.



Joonis 2 https://www.geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/

Ülesanne 2: Indeksi Kaardistamine

Algoritmi ruumikompleksus on O(n), kuna tabeli suurus on seotud ainult massiivi suurusega. Ajakompleksus on O(1), kuna otsime sisendis võtit ja võtme operatsioonis kas võti on olemas tabelis või mitte.



Üks praktiline lahendus indeksi kaardistamisel on ASCII märgid.

Ülesanne 3: Separate Chaining põrke lahendamine

Separate chaining on hea moodus tegeleda kokkupõrgetega, sest saame salvestada mitu väärtust sama võtmega ühte pinusse. Halb külg sellel on kui meil on väga palju ühe võtmeväärtusega räsi, läheb andmete talletamine ebaefektiivseks.

Joonis 3 Halb Separate Chainingu juhtum

https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/

Ülesanne 4: Open Addressing

Open Addressingus salvestame kõik andmed ühte pinusse ning kokkupõrgete lahendamine toimub vaba koha leidmise põhjal.

Lineaarne otsing otsib operatsiooni käigus esimest vaba pinu räsitabelis lineaarselt ehk töötleb tabelit niikaua läbi kuniks leiab endale sobiva koha. Lineaarne otsing saaks kasulik olla raudvara programmeerimisel, kus mälu on piiratud. Lineaarne otsing tagab, et muutujad oleksid salvestatud efektiivselt ilma, et kurnaks liigselt vabamälu.

Ruuduline otsing töötab sarnaselt lineaarsele otsingule kuid erinevalt lineaarse funktsiooni tehtele räsiTabel[x]*i on ruutotsingu tehe räsiTabel[x]*i*i. Ruutotsing võiks olla efektiivne räsitabelites, mille suurus on staatiline, kuna ruutotsing on parem esmase klasterdamise parandamiseks.

Topelträsimise funktsioonil kasutatakse topelt räsimist. Esimene räsimine toimub andmete sisestamisel ning teist räsimist kasutame kui tekib kokkupõrge ning selle lahendamiseks arvutame uue võtme. Topelträsimine on kõige efektiivsem põrgete lahendamisel kuid väga nõudlik ruumi osas. Arvan, et kõige parem on topelträsimist kasutada, kui vabamälu ei ole piiranguks olukorras.

Ülesanne 5: Topelt Räsimine

Topelträsimine aitab kõrvaldada klasterdamist ning minimaliseerib põrkeid kui implementeeritud räsifunktsioonid on efektiivsed.

Topelträsimise ajakompleksus on O(n) ning ruumikompleksus on sõltuv räsitabelist O(n).

Topelträsimist on mõistlik kasutada küberturves, kus on kriitiline, et ühele räsile vastaks mitu sisendit. Põhiliselt siis tuleks kasutada topelräsimist paroolihaldamisel.