Kodutöö 4 Paisktabelid, Räsitabelid (Hash map)

Ülesanne 1: Räsimine

1. Kirjuta selgitus räsimise (*hashing*) kontseptsioonist - põhiidee ja eesmärk.

Räsimine on andmete – teksti, numbrite, failide või millegi muu – teisendamine fikseeritud pikkusega tähtede ja numbrite stringiks. Andmed teisendatakse nendeks fikseeritud pikkusega stringideks või räsiväärtusteks, kasutades spetsiaalset algoritmi, mida nimetatakse räsifunktsiooniks.

Näiteks räsifunktsioon, mis loob 32-kohalisi räsiväärtusi, muudab tekstisisestuse alati ainulaadseks 32-kohaliseks koodiks. Olenemata sellest, kas on soov luua räsiväärtus sõnale „Auto" või kogu Shakespeare'i teosele, on räsiväärtus alati 32 tähemärki pikk.

Räsimise peamine eesmärk on andmete kiire otsimine, salvestamine ja leidmine. Samuti kasutatakse räsimist turvalisuses, näiteks paroolide salvestamisel ja andmete terviklikkuse kontrollimisel.

Põhiideed:
Võti-väärtus paar- Andmed salvestatakse võti-väärtus paaridena.
Räsitabel- Andmestruktuur, mida kasutatakse räsimise rakendamiseks.
Indekseerimine- Räsi väärtust kasutatakse andmete salvestamise indeksina.

2. Kirjelda hea räsifunktsiooni omadusi ja selgita, miks need on olulised.

   Räsifunktsioon peab olema kiiresti arvutatav, et tagada andmete tõhus töötlemine.

   Sama sisendi puhul peab räsifunktsioon alati andma sama väljundi.

   Erinevate sisendite puhul peab genereerima erinevaid räsi väärtusi.

   Räsi väärtused peaksid olema jaotatud ühtlaselt kogu räsi ruumis, vältimaks andmete kuhjumist teatud kohtadesse.

Räsi ei tohiks võimaldada algse sisendi kindlakstegemist, eriti krüptograafilistes rakendustes.

3. Selgita kokkupõrgete lahendamise tehnikaid, eriti eraldi aheldamist (separate chaining) ja avatud aadressimist (open addressing).

Eraldi Aheldamine (Separate Chaining):

- See tehnika hõlmab kokkupõrgete lahendamist, lisades samale räsi väärtusele vastavad elemendid ahelasse või loendisse.
- Kui räsifunktsioon annab sama väärtuse mitmele erinevale elemendile, salvestatakse need elemendid samasse asukohta, kuid kujundatakse ahelaks.
- Eeliseks on lihtsus ja paindlikkus, kuid miinuseks võib olla suurem mälu kasutus, kui ahelad on pikad.

Avatud Aadressimine (Open Addressing):

- Selles tehnikas salvestatakse kõik elemendid räsitabeli enda sisse ja kokkupõrgete korral otsitakse tabelist uus vaba asukoht.
- Levinud meetodid avatud aadressimisel on lineaarne uurimine, kvadriline uurimine ja hajutamine kahekordse räsimisega.
- Eeliseks on mälu tõhusam kasutamine võrreldes eraldi aheldamisega, kuid puuduseks võib olla aeglasemad päringud, kui tabel on tugevalt hõivatud.

Ülesanne 2. **Indeksi Kaardistamise (*Index Mapping)* Rakendamine ja Analüüs**

2. Analüüsi oma rakenduse aja- ja ruumikomplekssust.

Ajakomplektsus:

Sisestamisel O(1), kuna see nõuab vaid ühte massiivi indeksile juurdepääsu.

Otsimisel O(1), kuna kontrollitakse otse vastavat indeksit.

Kustutamisel O(1), kuna see nõuab vaid ühte massiivi indeksile juurdepääsu.

Ruumikomplekssus:

O(n)- n on maksimaalne võimalik väärtus. Ehk massiivi suurus on fikseeritud ja sõltub suurimast elemendist, mida soovitakse salvestada.

3.Aruta, kuidas indeksi kaardistamist massiividega saab rakendada reaalses maailmas.

Reaalses maailmas leiab index mapping kasutust näiteks unikaalsete numbrikoodide tuvastamisel. Näiteks kas, mõned ID-d on juba kasutusel või mitte. Veel näiteks ladudes kaupade kontrollimisel, kas teatud toode on veel laos või mitte, kui igal tootel on enda unikaalne kood.

Peamiseseks piiranguks saab mäluhulk, kui elemendid on suured või vahemik väga lai.

Ülesanne 3: *Separate Chaining* Kokkupõrgete Lahendamiseks

2. Võrdle *separate chaningu* efektiivsust *open addressing* meetodiga ajalise ja ruumilise komplekssuse mõttes.

   Ajakomplekssus:

   Separate Chaining: Sisestamine, otsimine ja kustutamine on üldjuhul O(1), aga halvimal juhul O(n), kui kõik võtmed räsivad samasse asukohta.

   Open Addressing: Sisestamine, otsimine ja kustutamine on samuti üldjuhul O(1), kuid võib ulatuda O(n)-ni, eriti kui räsitabel on peaaegu täis.

Ruumikomplekssus:

Separate Chaining: Võib nõuda rohkem ruumi viidete jaoks, iga võti on seotud lingitud nimekirja elemendiga.

Open Addressing: Tavaliselt ruumisäästlikum, sest see salvestab andmed otse piiratud räsitabelisse, kuid vajab laiendamist, kui tabel on peaaegu täis.

3. Aruta *separate chaning* kasutamise plusse ja miinuseid räsitabelites.

   Plussid: Räsitabeli suurus ei ole oluline, kuna nimekirju saab pikendada ilma kogu tabeli ümberkorraldamiseta ja tänu sellele piiramatu elementide arv.

   Sisestamine, otsimine ja kustutamine on efektiivne.

   Suudab käsitleda kõrget koormustegurit ja tagab üsna stabiilse jõudluse ka kõrgema koormusfaktori korral.

   Miinused: Suur mälukasutus(Nõuab lisamälu viidete jaoks LinkedListis või massiivides).

   Kuigi keskmine otsimisaeg on stabiilne, võib see varieeruda suuresti, eriti kui põrgete arv suureneb.

   Toimingute tõhusus sõltub räsifunktsiooni kvaliteedist

   Lingitud nimekirjade haldamine võib osutuda keerukamaks, eriti dünaamilise mälu haldamise kontekstis.

Ülesanne 4: *Open Addressing* Tehnikate Uurimine

1. Kirjuta lühike ülevaade avatud aadressimise meetodist kokkupõrgete lahendamisel räsimises.

Avatud aadressimine on kokkupõrgete lahendamise meetod räsitabelites, kus kõik elemendid salvestatakse otse räsitabelisse. Erinevalt separate chainingust, kus kokkupõrked lahendatakse lingitud nimekirjade kaudu, otsitakse avatud aadressimisel uut vaba asukohta tabelis endas. Kui räsimisfunktsioon annab asukoha, mis on juba hõivatud, kasutatakse teist funktsiooni (või funktsioonide seeriat), et leida alternatiivne asukoht.

2. Võrdle (teooria) kolme tehnikat: lineaarne otsing (linear probing), ruuduline otsing (quadratic probing) ja topelträsimine (*double hashing*).

Lineaarne Otsing: Liigub räsitabelis edasi fikseeritud sammuga (tavaliselt 1) kuni leiab vaba asukoha.

Plussid: Lihtne rakendada, hea jõudlusega väiksema koormusfaktoriga.

Miinused: Kalduvus klastri moodustumisele, mis tähendab, et kokkupõrked kipuvad tekitama "klambreid" täidetud asukohtadest, halvendades jõudlust.

Ruuduline Otsing: Kasutab ruudulist sammufunktsiooni (näiteks 1, 4, 9, 16, ...), et vähendada klastri moodustumist.

Plussid: Vähendab lineaarse otsingu klastri probleemi, tagades parema hajutamise.

Miinused: Võib tekitada sekundaarseid klastreid, mis samuti mõjutavad jõudlust.

Topelträsimine: Kasutab kahte räsimisfunktsiooni – esimene valib algse indeksi ja teine määrab põrke korral edasise indeksi.

Plussid: Pakub parimat hajutamist, vähendades oluliselt klastri moodustumist.

Miinused: Keerulisem rakendada ja võib olla aeglasem, kuna nõuab kahe räsimisfunktsiooni kasutamist

3. Aruta, millistes olukordades iga tehnika oleks kõige efektiivsem.

Lineaarne otsing:

Efektiivne, kui koormusfaktor on madal ja tabeli suurus on suhteliselt suur võrreldes hoiustatavate elementide arvuga. Lineaarne otsing töötab hästi, kui kokkupõrked on harvad.

Ruuduline otsing:

Efektiivne, kui koormusfaktor on keskmine ja on oluline vältida suurte klastrite moodustumist. Ruuduline otsing on kasulik, kui räsitabelis on mõõdukas hulk kokkupõrkeid.

Topelträsimine:

Parim valik kõrgema koormusfaktoriga tabelite jaoks, kus hajutamine on kriitilise tähtsusega. Topelträsimine on efektiivne ka siis, kui on vajalik minimeerida kokkupõrgete mõju jõudlusele, eriti suurtes andmekogumites.

Ülesanne 5: Topelt Räsimise (*Double hashing)* Rakendamine

1. Kuidas Topelt Räsimine Aitab Ületada Ühekordse Räsimise Piiranguid

   Topelt räsimine vähendab oluliselt klastrite moodustumist ja parandab hajutamist, eriti suure koormusfaktoriga räsitabelites. Erinevalt lineaarsest ja ruudulisest otsingust, kus sammude suurus on fikseeritud või sõltub ainult proovide arvust, sõltub topelt räsimises iga samm kahe erineva räsimisfunktsiooni tulemusest, tagades parema hajutamise.

2. Analüüsi oma rakenduse aja- ja ruumikomplekssust.

Ajakomplekssus:

Sisestamine: O(1) keskmiselt, kuid võib ulatuda O(n) halvimal juhul.

Otsimine: O(1) keskmiselt, kuid võib ulatuda O(n) halvimal juhul.

Kustutamine: O(1) keskmiselt ja O(n) halvimal juhul.

Ruumikomplekssus:

O(n) – n tabeli suurus.

3. Paku välja stsenaarium, kus topelt räsimine oleks eriti efektiivne.

   E-kaubanduse platvorm

   E-kaubanduse platvormid haldavad tohutut hulka tooteid ja klientide andmeid. Igal tootel või kliendil on unikaalne identifikaator (ID).

   Platvorm peab kiiresti vastama päringutele, nagu tooteinfo hankimine, kliendi andmete uuendamine või ostude ajalugu.

   Sobib laiendatavatele süsteemidele, kus andmebaasi suurus ja päringute hulk kasvavad pidevalt.

Boonusülesanne: Koormustegur ja *Rehashing (Double hashing vs Rehashing)*

1. Koormustegur(Load Factor)- Räsitabelis salvestatud elementide arvu ja räsitabeli kogumahu suhe.

   Koormustegur =Tabeli mahutavus / Elementide arv

   Näitab sisuliselt, kui koormatud või täidetud räsitabel on. Sellejärgi määrab, millal räsitabel vaja suurendada on.

2. **Rehashing**-u käigus kantakse räsitabeli elemente üle uude, tavaliselt suuremasse tabelisse. See toimub siis, kui koormustegur ületab kindla lävendi. Rehashing aitab säilitada tabeli jõudlust, vähendades kokkupõrgete arvu ja parandades andmete hajutamist.

3. Rehashingu mõju räsitabeli jõudlusele on oluline, eriti kui tegemist on dünaamiliselt muutuva andmehulgaga.

   Positiivne mõju:
   Vähendab Kokkupõrkeid, kuna rehashing hõlmab üleminekut suuremale tabelile, jaotuvad võtmed üle laiema vahemiku, vähendades kokkupõrgete tõenäosust. See parandab otsingu, sisestamise ja kustutamise operatsioonide efektiivsust.

   Parandab Jõudlust, sest suurema mahutavusega tabelis on iga võtme jaoks rohkem potentsiaalseid asukohti, mis aitab vähendada klastrite moodustumist ja parandab üldist hajutamist.

   Negatiivne mõju:
   Ajakulu, sest kogu tabeli elementide ümberpaigutamine uude tabelisse on ajamahukas operatsioon.

   Nõuab palju mälu.

   Võib esineda jõudluse kõikumisi, kuna kõik operatsioonid peatatakse, kuni andmed on üle viidud uude tabelisse.

Matthias Paluteder