Kodutöö 3 Otsingualgoritmid

Ülesanne 1: Lineaarotsing

2. Ajakomplekssus on O(n) kuna ta võrdeleb iga elementi massiivis sihiväärtusega, kuni leiab. Mida suurem massiiv seda kauem aega võtab kuna n on massivi pikkus.

Ruumiskomplekssus on O(1) kuna ta ei nõua lisamälu.

3. Reaalmaailma rakendused

Telefoniraamatu otsing - Lineaarset otsingut saab kasutada telefoniraamatust otsimiseks, et leida inimese nimi, võttes arvesse tema telefoninumbrit.

Õigekirjakontroll - algoritm võrdleb iga sõna dokumendis õigesti kirjutatud sõnade sõnastikuga, kuni leitakse vaste.

Minimaalsete ja maksimaalsete väärtuste leidmine - lineaarset otsingut saab kasutada massiivi või loendi minimaalsete ja maksimaalsete väärtuste leidmiseks.

Piirangud

Peamine piirang on selle aja komplekssus, eriti suurte massiivide puhul. Kui element mida otsitakse asub massiivi lõpus või seda pole üldse siis ei ole see kõige efektiivsem lähenemis meetod.

Ülesanne 2: Kahendotsing

2. Lineaarotsingu ajakomplekssus on O(n) kuna võrdleb võrdeleb iga elementi massiivis sihiväärtusega, kuni leiab. Mida suurem massiiv seda kauem aega võtab kuna n on massivi pikkus.

Kahendotsingu ajakomplekssus on tavaliselt O(log n). See jagab sisendmassiivi igal sammul pooleks, vähendades otsinguruumi poole võrra ja seetõttu on sellel logaritmilise järjekorra ajaline keerukus. Parimal juhul on ajakomplekssus O(1), kui otsitav element asub täpselt keskel.

3. Näide

BInary Search on kasulikum näiteks, kui meil on väga suur massiiv. Miljon elementi ja otsitav element asub täisesti massiivi lõppus. Siis lineaarotsing käib ükshaaval kõik elemendid läbi, et leida otsitav või tõestada, et element puudub. Binary Search poolitab massiivi ja võrdleb. Miljoni elemendi puhul oleks tal vaja teha ainult 20 võrdlust, et otsitav leida.

Ülesanne 3: Jump Search

1. Töötab sorteeritud massiividel. Jagab massiivi blokkideks ja hüppab iga bloki lõpus, kontrollides, kas otsitav element asub selles blokis. Kui otsitav element leitakse blokist, kasutatakse lineaarset otsingut ja leitakse elemendi täpne asukoht.

Põhiprintsiibid:

- Algoritm töötab ainult sorteeritud sisendmassiivide puhul.
- Vahelejäetava ploki optimaalne suurus on √n, mille tulemuseks on ajaline keerukus O(√n2).
- Selle algoritmi ajaline keerukus jääb lineaarse otsingu (O(n)) ja binaarse otsingu (O(log n)) vahele

2. Ajaline keerukus jääb lineaarse otsingu ja binaarse otsingu vahele(binaarne olles kõige parem). Lineaarset ületab, sest hüppab sammude kaupa, vähendades lineaarse otsingu vajadust ja muutes selle efektiivsemaks suurte andmekogumite korral.

3. Jump Search on efektiivsem suuremate massiivide korral üle linear searchi, sest teeb hüppeid samal ajal kui lineaar search käib igat elementi läbi.

Jump Search võib olla efektiivsem osaliselt sorteeritud massiivide puhul kui Binary Search, sest Jump Search suudab ületada osalise sorteerituse. Binary Search eeldab täielikku sorteerimist ja võib anda ebatäpseid tulemusi osaliselt sorteeritud massiivide korral.

Kui otsitav element on väike võib olla Jump Search kiirem, kui Binary Search.

Ülesanne 4. Kolmikotsing ja Kahendotsing

1. Idee on jagada massiiv kolmeks võrdseks osaks ja kontrollida, millises osas on otsitav element. See toimib samamoodi nagu binaarne otsing. Ainus erinevus seisneb selles, et see vähendab aja keerukust ja jagab massiivi pigem kolmeks kui kaheks.

Põhiprintsiibid:

- Algoritmi ajaline keerukus on O(2 * log3n) ja see on tõhusam kui lineaarne otsing ja binaarne otsing suuremate massiivide puhul. Väiksemate massiivide puhul võivad lineaarne ja binaarne kiiremad olla.
- Massiiv peab olema sorteeritud.

2. Kolmikotsingus jagame massiivi kolmeks osaks (võtame kaks keskelt) ja jätame iga iteratsiooni korral kõrvale kaks kolmandikku otsinguruumist. Esmapilgul tundub, et kolmendotsing võib olla kiirem kui kahendotsing, kuna selle ajaline keerukus sisendil, mis sisaldab n elementi, peaks olema O (log3n), mis on väiksem kui kahendotsingu ajaline keerukus O (log2n).
Praktikas on Kahendotsing väiksemate massiivide puhul kiirem.

3.Üldiselt võib öelda, et Binary Search on praktilistes olukordades sageli efektiivsem kui Ternary Search. Kuna ta teeb võrdlusi kahe osa vahel mitte kolme osa vahel. Tänu sellele vajab see vähem ressurssi ja on efektiivsem.

Mõlemal on logaritmiline keerukus, on kolmendotsing kiirem kui binaarne otsing piisavalt suure puu jaoks. Kuigi kahendotsingul on igas sõlmes 1 võrdlus vähem, on see sügavam kui kolmendotsingu puu.

5. Sõna otsimine sõnaraamatust.

Kõige tõhusam sellise probleemi puhul oleks Binary search.
Kuna tegemist on väga suure massiiviga jääks lineaar search lihtsalt liiga aeglaseks.
Näiteks otsime sõna „Karu“ siis lineaarse otsingu puhul alustaksime A tähest ja käiksime kõik sõnad läbi enne, kui jõuaksime K täheni.

Binaarse otsingu puhul võtame keskpunkti ja vaatame kas sõna „Karu“ on suurem või väiksem keskpunktist. Tänu sellele on meil massiiv juba 2 korda väiksem. Kordame seda kuni leiame sõna karu.

Binaarne otsing on efektiivne suurte andmebaaside korral, sest see vähendab otsinguruumi kiiresti poole võrra iga sammu järel.
Binaarne osting eeldab ka sorteeritud massiive ja sõnaraamat on täpselt selline.

Modifikatsioonid:

Indekseerimine:
Luua indeks või räsitabel, mis vastendab iga sõna algustähe vastavale positsioonile sõnastikus.
Nii saab sõna esimese tähe järgi kiiresti määrata sõnaraamatu vahemiku, kust otsida.

Vahemällu salvestamine:
Rakendada vahemällu salvestamise mehhanism. Salvestada hiljuti otsitud sõnad koos nende positsioonidega sõnastikus. Kui sõna otsitakse sageli, võib see otsinguaega oluliselt lühendada.

BOONUS

Fibonacci Search:

Fibonacci otsingualgoritm algab väikseima Fibonacci arvu genereerimisega, mis on suurem või võrdne otsitava massiivi suurusega. See Fibonacci arv F saab otsingu lähtepunktiks.

Seejärel võrdleb algoritm otsitavat elementi massiivi elemendiga Fibonacci arvu (F-2) asukohas. Kui otsitav element on suurem, välistab algoritm esimesed F-2 elemendid ja kordab protsessi ülejäänud massiiviga. Kui otsitav element on väiksem, välistab see viimased F-1 elemendid ja kordab protsessi ülejäänud massiivi peal.

See protsess kordub, kusjuures algoritm jätkab massiivi osade "eemaldamist", kuni otsitav element leitakse või kuni massiiv on täielikult uuritud.

Ajaline keerukus on O(log n)
Ruumiline keerukus on O(1)

Massiivil kus teatud väärtused esinevad sagedamini kui teised võib fibonacci olla tõhusam. Binaarne otsing jagab massiivi alati pooleks, mis ei pruugi olla optimaalne. Fibonacci otsing on seevastu jaotuspunktide valimisel paindlikum, mis võib ebaühtlaste jaotuste korral otsinguruumi vähendada.

Fibonacci võib veel olla etem kui meil on stsenaarium, kus meil on suuremahuline sorteeritud massiiv ja otsime konkreetset elementi. Massiivis on palju korduvaid elemente, kuid need on järjestatud klastrid.

Näiteks:
[1,2,2,2,3,3,4,5,5,6,7,7,8,8,8,9,10,11,11,11,12,13,13,13,13,14,15,15,15,16]

Sellises stsenaariumis võib Fibonacci Search olla efektiivsem kui binaarne otsing. Kui otsitav element on klastri sees, võib binaarne otsing leida suvalise elemendi klastri algusest, mis ei pruugi olla otsitav element. Fibonacci Search, aga kasutades Fibonacci jada elemente, võib olla parem klastri sees liikumisel ja jõuda kiiremini otsitava elemendini.