Algoritmid ja andmestruktuurid kodutöö 3

Ülesanne 1

1.1

1.2 Lineaarotsingu nii ajaline kui ruumiline keerukus on O(n).

1.3 Kuna ajaline keerukus sõltub elementide arvust, siis töötab see algoritm hästi just väikeste andmekogude peal, näiteks klassinimekirjast ühe konkreetse õpilase nime leidmiseks või ostunimekirjast mõne toote kontrollimine. Kuna lineaarse otsingu puhul käiakse kõik elemendid järjest läbi kuni otsitava leidmiseni, on see eriti kasulik sorteerimata andmete puhul. Kasutatakse siis, kui vaja on teha ainult ühekordseid või harvu otsinguid või kui eelistatakse lihtsat koodi. Päriselus võidakse kasutada piiratud arvutusvõimsusega süsteemides, näiteks mikrokontrollerites.  
Peamine piirang on suuremate andmemahtude puhul aeglaseks muutumine. Aga samuti ka see, et algoritm ei tööta sorteeritud andmete peal paremini kui sorteerimata andmete peal. Lisaks Ei ole optimaalne mitmekordsete otsingute jaoks: sagedaste otsingute korral võivad teised algoritmid pakkuda paremat jõudlus.

2.1 Kood on eraldi failina Algo\_KT3\_UL2

2.2 Kahendotsingu ajaline keerukus on parem kui lineaar otsingul, sest iga võrdlusega jagatakse massiiv pooleks. O (log n). Mõlema otsinguga on parim tulemus O (1) ejk otsitav võib olla esimene kontrollitav element

2.3 Binary search on kasulikum pikkade sorteeritud loendite puhul. Näiteks siis, kui meil on **miljonite kasutajatega andmebaas**, kus kasutajate nimed on **tähestikulises järjestuses** (ehk **sorteeritud**). Ja me soovime otsida konkreetse kasutaja nime, näiteks "Mark". Sellisel juhul on kahendotsing tõenäoliselt kiirem.

3.1 Hüpikotsingu põhimõte:  
Loendi läbimiseks kasutatakse fikseeritud sammupikkusega hüppeid (tavaliselt √n).  
Algoritm jagab massiivi võrdseteks plokkideks ja hüppab läbi andmete sammude kaupa, et leida plokk, milles otsitav element võiks asuda. Kui plokk, kuhu otsitav väärtus kuulub, on leitud, tehakse selles plokis Linear Search.

Funktsioon JumpSearch(loend, otsitav):

samm = √(n) // Arvuta samm suuruseks ruutjuur massiivi pikkusest

eelmine = 0 // Alusta massiivi algusest

n = loend.pikkus // Massiivi pikkus

// Liigu edasi plokkide kaupa, kuni leiad ploki või jõuad massiivi lõppu

Kuni loend[min(samm, n)-1] < otsitav:

eelmine = samm

samm += √(n)

Kui eelmine >= n:

Tagasta -1 // Elementi pole massiivis

// Tee lineaarotsing plokis

For i = eelmine kuni min(samm, n):

Kui loend[i] == otsitav:

Tagasta i // Tagasta indeksi väärtus

Tagasta -1 // Elementi ei leitud

3.2 Ajalise keerukuse võrdlus Binary kõige kiirem, JumpSearch keskmine ja lineaarotsing kõige aeglasem

|  |  |  |  |
| --- | --- | --- | --- |
|  | Lineaarotsing | Kahendotsing | Hüpikotsing |
| Parim juht | O(1) | O(1) | O(1) |
| Keskmine juhtum | O(n) | O(log n) | O(√n) |
| Halvim juhtum | O(n) | O(log n) | O(√n) |

3.3 Kui andmed on sorteeritud ja mõõduka suurusega, suudab Jump Search vähendada otsinguaega võrreldes lineaarotsinguga.

Näide: Otsing 10000 elemendiga sorteeritud andmestikus. Jump Search nõuab umbes 100 sammu, samas kui Linear Search võib vajada kuni 10000 sammu.

Või siis kui andmete juurdepääs on piiratud või kallis.  
Mõnes süsteemis on juhuslik juurdepääs andmetele kallis ja aeglane (nt mäluseadmetes, nagu magnetlindid või suured hajutatud andmebaasid). Eelis Jump Searchil: See teeb suuri hüppeid esimeses faasis, mis vähendab otsingukordade arvu võrreldes Linear Searchiga. Binary Searchi puhul on aga vaja pidevat keskpunktide arvutamist ja juhuslikku juurdepääsu, mis võib olla ebaefektiivne.

4.1 Kolmikotsing ehk Terinary Search on pisut sarnane kahendotsinguga, kuid jagab otsiva loendi mitte kahkes, vaid kolmeks osaks. Kõigepealt määrab kaks keskkohta ja siis võrdleb keskpunktide väärtusi otsitava väärtusega ja võrdluse põhjal loobub kolmandikust otsinguruumist. Protsessi korratakse kuni leiab otsitava väärtuse või otsinguruum on ammendunud. Kood Lisatud eraldi failina Algo\_KT3\_UL1

4.2 Kahendotsing vs kolmikotsing ajaline keerukus

|  |  |  |
| --- | --- | --- |
|  | Kahendotsing | Kolmikotsing |
| Parim juht | O(1) | O(1) |
| Keskmine juhtum | O(log n) | O(log3 n) |
| Halvim juhtum | O(log n) | O(log3 n) |

4.3 Binaarotsing sobib paremini väiksemate kuni keskmiste andmemassiivide puhul, kolmikotsingu kasu väljendub just suuremate andmekogumite puhul, kuid kolmikotsingu puhul on ka vigade tekkimise tõenäosus suurem. Kolmikotsing teeb igas etapis kaks võrdlust, samas kui kahendotsing teeb ainult ühe võrdluse ja seega pole kolmikotsing alati kiirem kui kahendotsing. Üldiselt kasutatakse pigem kahendotsingut ja kolmikotsing jääb üsna spetsiifiliste ülesannete jaoks.

5.

Sõna otsimine sõnaraamatust. Sobiks hästi kahendotsing. Kuna andmed on sorteeritud ja neid on palju, siis poleks mõteks kasutada lineaarotsingut. Ka hüpikotsik ei pruuki olla nii tõhus,. Kuna sõnaraamatus on väga palju sõnu võiks kahendotsig olla kiirem kui kolmikotsing, sest kolmikotsing vajab rohkem võrdlusi.

LISA.  
Fibonacci Search sarnaneb **Binary Search**-iga, kuid kasutab **Fibonacci arve** otsingupositsioonide määramiseks. Fibonacci arvude abil jaotatakse andmestik osadeks, mis vastavad Fibonacci järjestuse suhetele.

Stsenaarium: Meil ons uur sorteeritud andmestik, mis asub järjestikulises mälus, näiteks: hajutatud andmebaas või väline salvestusseade, kus iga juhuslik mälule ligipääs (nt keskmise indeksi arvutamine Binary Search jaoks) on aeglane ja kallis. Fibonacci Search hoiab võrdlused ja hüpped loogiliselt paigutatuna, mis vähendab aega, kui juhuslik ligipääs on kallis.