**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

Algoritmų sudarymas ir analizė (P170B400)

1 laboratorinio darbo ataskaita

(34a variantas)

Atliko:

IF-4/12 gr. studentas

Mangirdas Kazlauskas

2016 m. kovo 21 d.

Priėmė:

Doc. Vytautas Pilkauskas

KAUNAS

2016

1. Laboratorinio darbo užduotis:

Savarankiškai realizuoti bei ištirti sąlygoje nurodytus rikiavimo ir paieškos algoritmus, įvertinti jų efektyvumo priklausomybę nuo duomenų kiekio.

Rikiavimo užduotis: realizuoti rikiavimo algoritmą, kai duomenų faile esančių elementų rikiavimas atliekamas į elementus žiūrint kaip į masyvą bei sąrašą (galimos tik tai duomenų struktūrai būdingos operacijos). Mano užduoties variantas: rikiavimas įterpimo metodu, duomenis sudaro du failai: elementų failas bei rodyklių failas.

Paieškos užduotis: duotas studentų sąrašas. Jame reikia surasti vienodas pavardes. Mano užduoties variantas: paieškai naudoti maišos lentelės su sąrašais duomenų struktūrą.

1. Laboratorinio darbo realizacija:

Laboratoriniam darbui atlikti naudojama programavimo kalba – JAVA. Programavimo aplinka – IntelliJ IDEA 15.0.3.

1. Rikiavimo užduotis:
2. Duomenų generavimas:

Programą Generator.java vykdant konsolėje, yra galimybė nurodyti parametrus: duomenų failo vardas, rodyklių failo vardas, generuojamų duomenų skaičius. Jei parametrai nenurodomi, pagal nutylėjimą duomenų failo vardas yra „Data.txt“, rodyklių failo vardas – „Links.txt“, generuojama 1000 elementų.

Generatoriaus programos kodas:

**import** java.io.FileNotFoundException;  
**import** java.io.IOException;  
**import** java.io.RandomAccessFile;  
**import** java.text.ParseException;  
**import** java.text.SimpleDateFormat;  
**import** java.util.Calendar;  
**import** java.util.Random;  
**import** java.util.StringJoiner;  
  
*// Sugeneruoja sveikuosius skaičius bei rodykles***public class** Generator {  
  
 **private static int** *N* = 1000;  
 **public static** String *dataName* = **"Data.txt"**;  
 **public static** String *linksName* = **"Links.txt"**;  
  
 **public static void** main(String[] args) **throws** ParseException, IOException{  
 **if** (args.**length** == 3){  
 *generateNumbers*(args[0], args[1], Integer.*parseInt*(args[2]));  
 *dataName* = args[0];  
 *linksName* = args[1];  
 *setN*(Integer.*parseInt*(args[2]));  
 }  
 **else if** (args.**length** == 2){  
 *generateNumbers*(args[0], args[1], *getN*());  
 *dataName* = args[0];  
 *linksName* = args[1];  
 }  
 **else if** (args.**length** == 1){  
 *generateNumbers*(args[0], *linksName*, *getN*());  
 *dataName* = args[0];  
 }  
 **else**{  
 *generateNumbers*(*dataName*, *linksName*, *getN*());  
 }  
 }  
  
 *// generuojami sveikieji skaiciai (kiekis - N)* **private static void** generateNumbers(String dataName, String linksName, **int** N) **throws** ParseException, IOException{  
 RandomAccessFile raf = **new** RandomAccessFile(dataName, **"rw"**);  
 raf.setLength(N\*4);  
 Random rand = **new** Random();  
 raf.seek(0);  
 **for** (**int** i=0; i < N; i++) {  
 **int** value = rand.nextInt(1000);  
 *//value = i;* raf.writeInt(value);  
 }  
 System.***out***.print(**"Pirmu 20 sugeneruotu elementu seka:"**);  
 **for** (**int** i = 0; i < 20; i++){  
 raf.seek(i\*4);  
 **int** val = raf.readInt();  
 System.***out***.print(**" "** + val);  
 }  
 System.***out***.println(**"...\nSugeneruotu duomenu kiekis: "**+N);  
 *generateLinks*(linksName, N);  
 raf.close();  
 }  
  
 **private static void** setN(**int** value){  
 *N* = value;  
 }  
  
 **public static int** getN(){  
 **return** *N*;  
 }  
  
 *// generuojamos rodykles* **private static void** generateLinks(String linksName, **int** N) **throws** ParseException, IOException{  
 RandomAccessFile raf = **new** RandomAccessFile(linksName, **"rw"**);  
 raf.writeInt(0);  
 **for** (**int** i = 0; i < N; i++){  
 **if** (i != N - 1) {  
 raf.writeInt(i - 1);  
 raf.writeInt(i + 1);  
 }  
 **else**{  
 raf.writeInt(i-1);  
 raf.writeInt(-1);  
 }  
 }  
 }  
}

1. Rikiavimo realizavimas:

Programą Sorting.java vykdant konsolėje, reikia nurodyti, kokiu būdu duomenys rikiuojami: jei į duomenis žiūrime kaip į masyvą, reikia nurodyti „Array“ bei duomenų failą, kuriame patalpinti duomenys, jei kaip į sąrašą, parametrais reikia nurodyti „List“, tada duomenų failą, kur talpinami duomenys bei rodyklių failą. Atlikus duomenų rikiavimą, programa išveda laiką, kurį užtruko juos rikiuojant.

Rikiavimo funkcija, kai duomenys rikiuojami kaip masyvas:

**private static void** insertionSortArray(String dataName)

**throws** FileNotFoundException, IOException { | laikas| kartai  
 RandomAccessFile raf = **new** RandomAccessFile(dataName, **"rw"**); 1| c1 | 1  
 **int** key, j; 2| c2 | 1  
 **for** (**int** i = 1; i < raf.length() / 4; i++) { 3| c3 | n  
 key = *getNumber*(raf, i); 4| T\*(k) | n-1  
 **for** (j = i - 1; (j >= 0) && (*getNumber*(raf, j) > key); j--) 5| c4 |   
 *setNumber*(raf, j + 1, *getNumber*(raf, j)); 6| T\*(k) |   
 *setNumber*(raf, j + 1, key); 7| T\*(k) | n-1  
 } 8| |  
 raf.close(); 9| c1 | 1  
}

Funkcija T\* pažymėtos rašymo ir skaitymo į failą funkcijos, atrodančios taip:

**public static int** getNumber(RandomAccessFile raf, **int** i) **throws** IOException {  
 raf.seek(i\*4);  
 **return** raf.readInt();  
}

**public static void** setNumber(RandomAccessFile raf, **int** i, **int** value) **throws** IOException {  
 raf.seek(4 \* i);  
 raf.writeInt(value);  
}

Abiejų šių funkcijų sudėtingumas lygus: .

Algoritmo sudėtingumo funkcija:

Kadangi funkcijoje yra sąlygos sakinys, nuo kurio priklauso, kiek kartų bus kartojamas ciklo kodas, todėl apskaičiuosiu geriausią ir blogiausią funkcijos veikimo atvejus. Geriausiu atveju (kai duomenys jau išrikiuoti) tj = 1, tada nebus vykdoma 6 eilutė, o

Funkciją T(n) galima išsireikšti pavidalu *an+b,* todėl, geriausiu atveju, šio algoritmo sudėtingumas – tiesinis.

Blogiausiu atveju (kai duomenys išsidėstę priešinga tvarka nei reikia juos išrikiuoti) tj=j, tada

Funkciją T(n) galima išsireikšti pavidalu *an2+bn+c,* todėl, blogiausiu atveju, šio algoritmo sudėtingumas – kvadratinis.

Rikiavimo funkcija, kai duomenys rikiuojami kaip sąrašas:

**private static void** insertionSortList(String dataName, String linksName) **throws** FileNotFoundException, IOException { | laikas| kartai  
 RandomAccessFile rafd = **new** RandomAccessFile(dataName, **"rw"**); 1-2| c1 | 2  
 RandomAccessFile rafl = **new** RandomAccessFile(linksName, **"r"**); | |  
 **int** key, currentId, previousId; 3| c2 | 1  
 **int** head = *getFirstLink*(rafl); 4-5| T\*(k) | 2  
 **int** currentLinkF = *getLinkNext*(rafl, head); | |

**do** { 6| c3 | n  
 currentId = currentLinkF; 7-9| c2 | 3(n-1)  
 previousId = currentLinkF - 1; | |  
 **int** previousLinkP; | |  
 currentLinkF = *getLinkNext*(rafl, currentId); 10-11| T\*(k) | 2(n-1)  
 key = *getNumber*(rafd, currentId); | |  
 **do** { 12| c4 |   
 previousLinkP = *getLinkPrevious*(rafl, previousId); 13| T\*(k) |  
 **if** (*getNumber*(rafd, previousId) > key) { 14-15| T\*(k)|  
 *setNumber*(rafd, previousId + 1, *getNumber*(rafd, previousId)); | |  
 previousId = previousLinkP; 16| c2 |   
 } 17| |   
 } **while** (previousLinkP != -1 && *getNumber*(rafd, previousId) > key); 18| |  
 *setNumber*(rafd, previousId + 1, key); 19| T\*(k) | n-1   
 } **while** (currentLinkF != -1); 20| |  
 rafd.close(); 21-22| c1 | 2  
 rafl.close(); | |  
}

Kaip ir ankstesniame algoritme, funkcija T\* pažymėtos rašymo ir skaitymo į failą funkcijos. Šių funkcijų sudėtingumas lygus: .

Algoritmo sudėtingumo funkcija:

Geriausiu atveju algoritmo sudėtingumas, kaip ir pirmajame algoritme, bus tiesinis, todėl įrodysiu tik blogiausio atvejo sudėtingumą, kad jis taip pat yra kvadratinis.

Blogiausiu atveju (kai duomenys išsidėstę priešinga tvarka nei reikia juos išrikiuoti) tj=j, tada

Funkciją T(n) galima išsireikšti pavidalu *an2+bn+c,* todėl, blogiausiu atveju, šio algoritmo sudėtingumas – kvadratinis.

1. Duomenų (rezultatų) išvedimas:

Norint į konsolės langą išvesti failuose saugomus duomenis ar jau išrikiuotus rezultatus, vykdoma programa Printing.java. Programą vykdant konsolėje, per parametrus reikia nurodyti, kokiu būdu išvesti duomenis: jei norime išvesti kaip masyvą, pirmasis parametras nurodomas „Array“, kitas – duomenų failas, jei kaip sąrašą – nurodoma „List“, toliau nurodomi duomenų bei rodyklių failai. Išvedama pirmųjų 40 duomenų faile esančių reikšmių.

**import** java.io.FileNotFoundException;  
**import** java.io.IOException;  
**import** java.io.RandomAccessFile;  
**import** java.util.Objects;  
**import** java.util.zip.Inflater;  
  
  
**public class** Printing **extends** Sorting{  
  
 **public static** String *dataName* = **"Data.txt"**;  
 **public static** String *linksName* = **"Links.txt"**;  
  
 **public static void** main(String[] args) **throws** IOException {  
 **if** (args.**length** == 2) {  
 **if** (Objects.*equals*(args[0], **"Array"**))  
 *spausdintiMasyva*(args[1]);  
 }  
 **else if** (args.**length** == 3) {  
 **if** (Objects.*equals*(args[0], **"List"**))  
 *spausdintiSarasa*(args[1], args[2]);  
 }  
 }  
 **private static void** spausdintiMasyva(String dataName) **throws** FileNotFoundException, IOException{  
 RandomAccessFile raf = **new** RandomAccessFile(dataName, **"r"**);  
 **for** (**int** i = 0; i < 40; i++){  
 System.***out***.print(i + **"= "** + *getNumber*(raf, i) + **" :"**);  
 }  
 raf.close();  
 }  
 **private static void** spausdintiSarasa(String dataName, String linksName) **throws** FileNotFoundException, IOException{  
 RandomAccessFile rafd = **new** RandomAccessFile(dataName, **"r"**);  
 RandomAccessFile rafl = **new** RandomAccessFile(linksName, **"r"**);  
 System.***out***.print(*getFirstLink*(rafl) + **" -> "**);  
 **for** (**int** i = 0; i < 40; i++){  
 System.***out***.print(**"<- "** + *getLinkPrevious*(rafl, i) + **":"** +  
 *getNumber*(rafd, i) + **":"** + *getLinkNext*(rafl, i) + **" -> "**);  
 }  
 System.***out***.print(**"..."**);  
 rafd.close();  
 rafl.close();  
 }  
}

1. Rezultatai:

Eksperimento metu buvo generuojami bei rikiuojami 100, 200, 400, 800, 1600, 2400, 3200, 4800 bei 6400 duomenų. Duomenys buvo rikiuojami ir kaip masyvas, ir kaip sąrašas. Eksperimento rezultatai pateikti grafikuose (rikiavimo laiko priklausomybė nuo duomenų kiekio):

1. Išvados:

Remiantis svetainių <http://bigocheatsheet.com/> ir <https://en.wikipedia.org/wiki/Insertion_sort> duomenimis, teorinis įterpimo rikiavimo algoritmo efektyvumas geriausiu atveju yra O(n), vidutiniu – O(n2), o blogiausiu – taip pat O(n2). Remiantis teoriniais algoritmų sudėtingumo skaičiavimais bei eksperimento rezultatais, svetainėse pateiktas teorinis algoritmų efektyvumas pasitvirtino - rikiuojant duomenis ir kaip masyvą, ir kaip sąrašą, algoritmo greitis sulėtėja – duomenų skaičiui padidėjus 2 kartus, algoritmo vykdymo sparta sulėtėja 4 kartus – gaunama kvadratinė priklausomybė O(n2).

1. Paieškos užduotis:
2. Duomenų generavimo funkcija (studentų pavardžių generacija):

Generacijos funkcija generuoja studentų pavardes, kurių ilgis - 10 simbolių, pirmasis simbolis – „P“, o likusios pavardės raidės – atsitiktinės. Taip pat generatorius sukurtas taip, kad pirmoji ir paskutinė generuojama pavardė būtų „Paaaaaaaaa“. Taip yra dėl to, kad vykdant paiešką būtų matoma, jog iš tikrųjų surandamos vienodos pavardės, nes jei būtų generuojamos vien tik skirtingos pavardės, būtų neaiškų, ar paieška veikia. Funkcijos kodas:

**private static void** generateStudents(String fileName, **long** n) **throws** FileNotFoundException, IOException{  
 *// Pirmojo ir paskutinio studento pavardes vienodos, kitos - random* RandomAccessFile raf = **new** RandomAccessFile(fileName, **"rw"**);  
 raf.setLength(n\*20);  
 raf.seek(0);  
 raf.writeChar(**'P'**);  
 **for** (**int** i = 1; i < 10; i++) {  
 raf.seek(i \* 2);  
 raf.writeChar(**'a'**);  
 }  
 **for** (**long** i = 1; i < n; i++){  
 **char** []pav = **new char** [10];  
 *// Generacija* pav[0] = **'P'**;  
 **for** (**int** j = 1; j < pav.**length**; j++){  
 Random rand = **new** Random();  
 **int** value = rand.nextInt(122 - 97 + 1) + 97;  
 *// Padaroma taip, kas paskutine pavarde sutaptu su pirmaja* **if** (i != n-1)  
 pav[j] = (**char**)value;  
 **else** pav[j] = **'a'**;  
 *// atkomentuoti, jei norime, kad kazkuri pavarde pasikartotu daugiau nei 2 kartus  
 /\*if (i == n/2 || i == n/3 || i == n/4)  
 pav[j] = 'a';  
 else if (i == n-1)  
 pav[j] = 'a';  
 else  
 pav[j] = (char)value;\*/* }  
 *// Irasymas i faila (viena pavarde faile uzima 20 bitu)* **for** (**int** j = 0; j < pav.**length**; j++){  
 raf.seek(j\*2 + i\*20);  
 raf.writeChar(pav[j]);  
 }  
 }  
 raf.close();  
}

1. Vienodų studentų pavardžių paieška maišos lentelėje su sąrašais:

Vienodų pavardžių paieška maišos lentelėje veikia tokiu principu: sukuriama nauja tuščia lentelė, į kurią po vieną talpinamos studentų pavardės. Naudojant hash funkciją, kiekvienai pavardei surandama konkreti vieta lentelėje, todėl tampa aišku, kad jei pavardės vienodos, jiems bus paskirta ta pati vieta, todėl programoje patikrinama, ar įterpiama jau lentelėje esanti pavardė ir jei pavardė kartojasi, prie pavardės lentelėje esantis indeksas padidinamas vienetu ir taip žinoma, kad buvo bandoma įterpti identišką pavardę. Paieškos funkcija ir jos teorinis algoritmo sudėtingumo skaičiavimas:

**public static void** studentuPaieska(String dataName, String hashName)

**throws** IOException { | laikas | kartai   
 RandomAccessFile rafData = **new** RandomAccessFile(dataName, **"rw"**); 1-2| c1 | 2   
 RandomAccessFile raf = **new** RandomAccessFile(hashName, **"rwd"**); | |  
 *tableSize* = 0; 3| c2 | 1  
 *createEmpty*(raf,0.75f,20); 4| Tc(20) | 1   
 **long** n = rafData.length()/20; 5| c2 | 1  
 **for** (**long** i = 0; i < n; i++) { 6| c3 | n+1  
 **char** []p = **new char**[10]; 7| c2 | n  
 **for** (**int** j = 0; j < 10; j++) { 8| c4 | 11n  
 rafData.seek(j\*2+i\*20); 9-10| c5 | 20n  
 p[j] = rafData.readChar(); | |  
 } 11| |  
 *put*(raf,p); 12| Tp() | n  
 } 13| |  
 rafData.close(); 14-15| c1 | 2  
 raf.close(); | |  
}

Algoritmo sudėtingumo funkcija:

Tuščios lentelės (talpinančios 20 sąrašų) sukūrimo funkcija Tc(20):

**public static void** createEmpty(RandomAccessFile raf, **float** loadFactor,

**long** size) **throws** FileNotFoundException, IOException{ | laikas | kartai  
 raf.setLength(20 + 8\*size); 1-5| c1 | 5  
 raf.seek(0); | |  
 raf.writeFloat(loadFactor); *// lenteles uzpildymas* | |raf.writeLong(size); *// lenteles dydis* | |raf.writeLong(0); *// nariu skaicius* | |**for** (**int** i = 0; i < size; i++) 8| c2 | 21

raf.writeLong(-1); *// tuscia vieta* 7| c1 | 20}

Funkcijos sudėtingumas: .

Pavardės įtraukimo į lentelę funkcija Tp(j):

**public static void** put(RandomAccessFile raf, **char** []masyvas) **throws** IOException {  
 */\* Lenteles failo sudetis:  
 \* pirmi 4 bitai - load Factor  
 \* tolesni 8 bitai - lenteles dydis  
 \* tolesni 8 bitai - nariu skaicius lenteleje  
 \* Viso: 20 pirmu failo bitu uzimta  
 \* Kiekviena grandinele turi savo indeksa, todel viso tokiu indeksu - 8 bitai\*lenteles dydis  
 \* Kiekviena grandinėle viduje talpina pavardes (10 \* 2 bitai),  
 \* tokiu paciu pavardziu kieki - 8 bitai, adresas i kita - 8 bitai  
 \* Viso: 20 + 8 + 8 = 36 bitai vienam nariui, is viso bitu nariams - 36 bitai \* nariu skaicius \*/* | laikas | kartai

raf.seek(4); 1-4| c1 | 4   
 *tableSize* = raf.readLong(); | |  
 raf.seek(12); | |  
 **long** nariuSk = raf.readLong(); | |  
 **long** pabPos = 20+8\**tableSize*+36\*nariuSk; 5| c2 | 1  
 **long** hash = *hash*(masyvas); 6| c3 | 1  
 raf.seek(hash\*8+20); 7-8| c1 | 2  
 **long** kitasNarys = raf.readLong(); | |  
 **if** (kitasNarys==-1) { 9| c4 | 1  
 raf.seek(hash\*8+20); 10-11| c1 | 2  
 raf.writeLong(pabPos); | |  
 **for** (**int** i = 0; i < 10; i++) { 12| c5 | 11  
 raf.seek(i\*2+pabPos); 13-14| c1 | 20  
 raf.writeChar(masyvas[i]); | |  
 } 15| |  
 raf.seek(pabPos+20); 16-19| c1 | 4  
 raf.writeLong(1); | |  
 raf.seek(pabPos+20+8); | |  
 raf.writeLong(-1); | |  
 nariuSk++; 20| c2 | 1  
 raf.seek(12); 21-22| c1 | 2  
 raf.writeLong(nariuSk); | |  
 } | |  
 **else** { | |  
 **boolean** radau = **false**; 25| c2 | 1  
 **while** (!radau) { 26| c6 | j  
 **char** []t = **new char**[10]; 27| c2 | j-1  
 **for** (**int** i = 0; i < 10; i++) { 28| c5 | 11(j-1)  
 raf.seek(i\*2+kitasNarys); 29-30| c1 | 10(j-1)  
 t[i] = raf.readChar(); | |  
 } | |  
 **if** (Arrays.*equals*(t, masyvas)) { 32| c4 | (j-1)  
 raf.seek(20+kitasNarys); 33-36| c1 | 4(j-1)  
 **long** kiekis = raf.readLong()+1; | |  
 raf.seek(20+kitasNarys); | |  
 raf.writeLong(kiekis); | |  
 radau = **true**; 37| c2 | (j-1)  
 } | |  
 **else** { | |  
 raf.seek(kitasNarys+28); 40-41| c1 | (j-1)  
 **long** kitasT = raf.readLong(); | |  
 **if** (kitasT==-1) { 42| c4 | (j-1)   
 raf.seek(kitasNarys+28); 43-44| c1 | (j-1)   
 raf.writeLong(pabPos); | |  
 **for** (**int** i = 0; i < 10; i++) { 45| c5 | 11(j-1)   
 raf.seek(i\*2+pabPos); 46-47| c1 | 10(j-1)   
 raf.writeChar(masyvas[i]); | |  
 } | |  
 raf.seek(pabPos+20); 49-52| c1 | 4(j-1)   
 raf.writeLong(1); | |  
 raf.seek(pabPos+20+8); | |  
 raf.writeLong(-1); | |  
 radau = **true**; 53-54| c2 | 2(j-1)   
 nariuSk++; | |  
 raf.seek(12); 55-56| c1 | 2(j-1)   
 raf.writeLong(nariuSk); | |  
 } | |  
 **else** | |kitasNarys = kitasT; 59| c2 | j-1  
 }  
 }  
 }  
}

Šios funkcijos sudėtingumą paskaičiuoti yra sudėtinga, nes labai sunku nustatyti, kiek kartų suksis while ciklas, kurie sąlygos sakiniai bus vykdomi, tačiau aišku tai, kad bet kokiu atveju funkcijos sudėtingumas galiausiai bus tiesinis, t.y., kuo daugiau elementų bus lentelėje, tuo ilgiau bus ieškoma tokio paties elemento arba vietos, kur reikia įterpti kitą elementą:

Bendras paieškos funkcijos sudėtingumas:

Kadangi funkcija *Op(j)* yra tiesinė, tai sandauga *Op(j)\*n* virs kvadratine funkcija, todėl:

;

Funkciją *T(n)* galime išreikšti pavidalu *an2+bn+c*, todėl šio paieškos algoritmo sudėtingumas – kvadratinis.

1. Rezultatai:

Eksperimento metu buvo generuojamos 100, 200, 400, 800, 1600, 2400, 3200, 4800 bei 6400 studentų pavardžių. Vėliau buvo ieškoma, kiek iš visų sugeneruotų pavardžių yra identiškų kitoms sugeneruotosioms bei jau įterptoms į lentelę pavardėms. Eksperimento rezultatai pateikti grafikuose (paieškos laiko priklausomybė nuo duomenų kiekio):

1. Išvados:

Iš grafiko matome, kad teorinis algoritmo sudėtingumo įvertinimas pasitvirtino – priklausomybė iš tiesų yra kvadratinė O(n2).