**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**ALGORITMŲ SUDARYMAS IR ANALIZĖ**

Laboratorinis darbas nr. 1

Atliko:

IFF – 7/2 gr. studentas Giedrius Kristinaitis

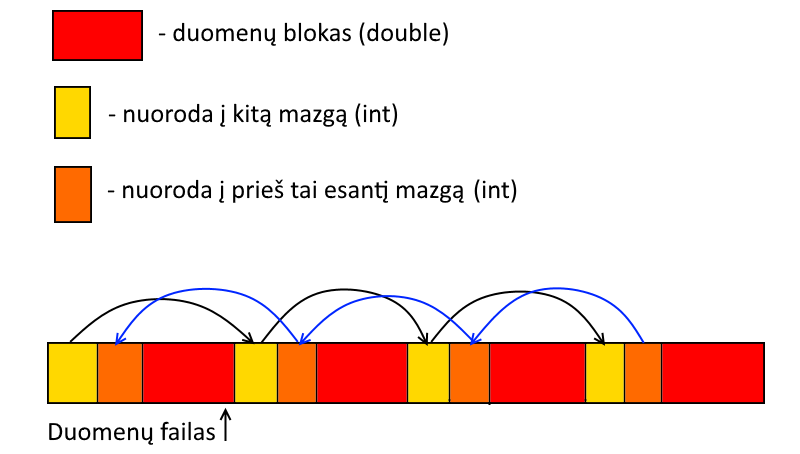
**KAUNAS, 2019**

1. Užduotis

Darbo užduotis – palyginti rikiavimo algritmus (Insertion sort ir Merge sort) masyve ir susietąjame sąraše dviem atvejais: kai duomenų struktūra realizuota operatyvinėje atmintyje ir diskinėje atmintyje. Taip pat reikia atlikti paieškos analizę maišos lentelės su sąrašais atveju. Suskaičiuoti algoritmų sudėtingumą remiantis programos tekstu ir padaryti išvadas.

1. Susietasis sąrašas išorinėje atmintyje

Diskinėje atmintyje realizuotas susietasis sąrašas atrodo taip:



Elemento nuskaitymas realizuotas šiuo kodu:

**public double** next() {  
 **if** (**current** == -1) {  
 **current** = **first**;  
 } **else** {  
 **try** {  
 **data**.seek(**current**);  
  
 **current** = **data**.readInt();  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 **return** get();  
}

**public double** get() {  
 **try** {  
 **data**.seek(**current** + 8);  
  
 **return data**.readDouble();  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 }  
  
 **return** 0;  
}

Čia data – duomenų failas, current – esamas sąrašo mazgas.

1. Insertion sort analizė

Insertion sort – tai rikiavimo algoritmas, kai duomenys surikiuojami iš eilės imant kiekvieną elementą ir jį įdedant į reikiamą vietą.

Teorinis algoritmo sudėtingumas: O(n2)

Suskaičiuojame algoritmo sudėtingumą pagal programos tekstą (array.length() pavadinkime n):

|  |  |  |
| --- | --- | --- |
| **public void** sortArray(DataArray array) {  **for** (**int** i = 1; i < array.length(); i++) {  **int** a = i;  **int** b = i - 1;   **while** (b >= 0 && array.get(b) > array.get(a)) {  array.swap(a, b);  a--;  b--;  }  } } | Kaina  C1  C2  C3  C4  C5  C6  C7 | Kartai  n+1  n  n |

Suskaičiuotas algoritmo sudėtingumas sutampa su teoriniu sudėtingumu.

Masyvo operatyvinėje atmintyje eksperimento rezultatai:

|  |  |
| --- | --- |
| Masyvo dydis | Vykdymo laikas (ms) |
| 1000 | 11 |
| 2000 | 16 |
| 4000 | 23 |
| 8000 | 63 |
| 16000 | 223 |
| 32000 | 808 |
| 64000 | 3219 |

Masyvo diskinėje atmintyje eksperimento rezultatai:

|  |  |
| --- | --- |
| Masyvo dydis | Vykdymo laikas (ms) |
| 10 | 5 |
| 20 | 22 |
| 40 | 48 |
| 80 | 165 |
| 160 | 655 |
| 320 | 2431 |
| 640 | 10659 |

Sąrašo operatyvinėje atmintyje eksperimento rezultatai:

|  |  |
| --- | --- |
| Sąrašo dydis | Vykdymo laikas (ms) |
| 1000 | 4 |
| 2000 | 8 |
| 4000 | 27 |
| 8000 | 86 |
| 16000 | 464 |
| 32000 | 3736 |
| 64000 | 12475 |

Sąrašo diskinėje atmintyje eksperimento rezultatai:

|  |  |
| --- | --- |
| Sąrašo dydis | Vykdymo laikas (ms) |
| 10 | 19 |
| 20 | 32 |
| 40 | 86 |
| 80 | 151 |
| 160 | 546 |
| 320 | 2019 |
| 640 | 7125 |

Insertion sort operatyvinėje atmintyje masyvo ir sąrašo palyginimas:

Insertion sort diskinėje atmintyje masyvo ir sąrašo palyginimas:

1. Merge sort analizė

Merge sort – tai skaldyk ir valdyk rikiavimo algoritmas, kai rikiuojami duomenys rekursiškai išskaidomi į mažesnius masyvus ir po to sujungiami reikiama tvarka.

Teorinis algoritmo sudėtingumas: O(n\*logn)

Suskaičiuojame algoritmo sudėtingumą pagal programos tekstą (argumentą right pavadinkime n, nes right yra masyvo ilgis, merge metodą pažymėkime M(n)):

|  |  |  |
| --- | --- | --- |
| **private void** sort(DataArray array, **int** left, **int** right) {  **if** (right > left) {  **int** middle = (left + right) / 2;  sort(array, left, middle); sort(array, middle + 1, right); merge(array, left, right, middle);} } **private void** merge(DataArray array, **int** left, **int** right, **int** middle) {  **int** firstLength = middle - left + 1;  **int** secondLength = right - middle;  **double**[] tempLeft = **new double**[firstLength];  **double**[] tempRight = **new double**[secondLength]; **for** (**int** i = 0; i < firstLength; i++) {  tempLeft[i] = array.get(left + i);  }   **for** (**int** i = 0; i < secondLength; i++) {  tempRight[i] = array.get(middle + 1 + i);  } **int** arrayIndex = left;  **int** i = 0;  **int** j = 0;   **while** (i < firstLength && j < secondLength) {  **if** (tempLeft[i] <= tempRight[j]) {  array.set(arrayIndex, tempLeft[i++]);  } **else** {  array.set(arrayIndex, tempRight[j++]);  }   arrayIndex++;  } **while** (i < firstLength) {  array.set(arrayIndex++, tempLeft[i++]);  }   **while** (j < secondLength) {  array.set(arrayIndex++, tempRight[j++]);  } } | Kaina  C1  C2  T(n/2)  T(n/2)  C3  C4  C5  C6  C7  C8  C9  C10  C11  C12  C13  C14  C15  C16  C17  C18  C19  C20  C21  C22 | Kartai  1  1  1  1  1  1  1  1  firstLength+1  firstLength  secLength+1  secLength  1  1  1  fLen+sLen+1  fLen+sLen  firstLength  secLength  fLen+sLen  2  1  2  1 |

Skaičiuojame merge metodo sudėtingumą (kadangi firstLength + secondLength yra sujungto masyvo ilgis, pavadinkime jį m):

Skaičiuojame T(n):

Suskaičiuotas algoritmo sudėtingumas sutapo su teoriniu sudėtingumu.

Masyvo operatyvinėje atmintyje eksperimento rezultatai:

|  |  |
| --- | --- |
| Masyvo dydis | Vykdymo laikas (ms) |
| 10000 | 8 |
| 30000 | 9 |
| 90000 | 27 |
| 270000 | 74 |
| 810000 | 255 |
| 2430000 | 793 |
| 7290000 | 2536 |

Masyvo diskinėje atmintyje eksperimento rezultatai:

|  |  |
| --- | --- |
| Masyvo dydis | Vykdymo laikas (ms) |
| 100 | 810 |
| 200 | 1742 |
| 400 | 2950 |
| 800 | 5924 |
| 1600 | 12337 |
| 3200 | 24691 |
| 6400 | 52760 |

Sąrašo operatyvinėje atmintyje eksperimento rezultatai:

|  |  |
| --- | --- |
| Sąrašo dydis | Vykdymo laikas (ms) |
| 10000 | 5 |
| 30000 | 7 |
| 90000 | 40 |
| 270000 | 153 |
| 810000 | 601 |
| 2430000 | 3292 |
| 7290000 | 13784 |

Sąrašo diskinėje atmintyje eksperimento rezultatai:

|  |  |
| --- | --- |
| Sąrašo dydis | Vykdymo laikas (ms) |
| 100 | 97 |
| 200 | 174 |
| 400 | 445 |
| 800 | 926 |
| 1600 | 2125 |
| 3200 | 4313 |
| 6400 | 9425 |

Merge sort operatyvinėje atmintyje masyvo ir sąrašo palyginimas:

Duomenų kiekis

Merge sort išorinėje atmintyje masyvo ir sąrašo palyginimas:

1. Paieškos analizė

Paieška buvo atlikta maišos lentelėje su sąrašais operetyvinėje ir diskinėje atmintyje.

Buvo ieškomas kiekvienas maišos lentelės elementas.

Paieškos maišos lentelėje vidutinis teorinis sudėtingumas: Θ(1)

Paieškos maišos lentelėje blogiausio atvejo teorinis sudėtingumas: O(n)

Skaičiuojame paieškos sudėtingumą pagal programos kodą (getNodeWithKey metodą pažymėkime G, taip pat kadangi getNodeWithKey gali praeiti tarp 1 ir visų grandinės mazgų, grandinės ilgį pažymėkime n):

|  |  |  |
| --- | --- | --- |
| **public** String get(String key) {  **int** index = hash(key);   **if** (**nodes**[index] != **null**) {  Node existing = getNodeWithKey(**nodes**[index], key);  **if** (existing != **null**) {  **return** existing.**value**;  }  }   **return null**; }  **private** Node getNodeWithKey(Node node, String key) {  **if** (node != **null**) {  **do** {  **if** (node.**key**.equals(key)) {  **return** node;  }   } **while** ((node = node.**next**) != **null**);  }   **return null**; } | Kaina  C1  C2  G(n)  C3  C4  C5  C6  C7  C8  C9  C10 | Kartai  1  1  1  1  1  1  1  n  1  n  1 |

Skaičiuojame G(n) sudėtingumą:

Skaičiuojame T(n):

Kadangi paieškos laikas priklauso nuo grandinės ilgio, kuris yra n, geriausiu atveju paieškos sudėtingumas yra Θ(1), o blogiausiu – O(n).

Paskaičiuotas įvertinimas sutampa su teoriniu įvertinimu.

Paieškos operatyvinėje atmintyje rezultatai:

|  |  |
| --- | --- |
| Duomenų kiekis | Paieškos laikas (ms) |
| 50000 | 17 |
| 100000 | 21 |
| 200000 | 22 |
| 400000 | 48 |
| 800000 | 98 |
| 1600000 | 234 |
| 3200000 | 518 |

Paieškos išorinėje atmintyje rezultatai:

|  |  |
| --- | --- |
| Duomenų kiekis | Paieškos laikas (ms) |
| 300 | 27 |
| 600 | 48 |
| 1200 | 82 |
| 2400 | 135 |
| 4800 | 346 |
| 9600 | 671 |
| 19200 | 1649 |

1. Išvados

Merge sort algoritmo sudėtingumas yra O(n\*logn) ir jis yra daug greitesnis dideliems duomenų kiekiams nei insertion sort, kurio sudėtingumas yra O(n2). Merge sort iki 7 290 000 elementų rikiavimo laikas išaugo tik iki 2,5 sekundės, kai tuo tarpu insertion sort iki 64 000 elementų rikiavimo laikas išaugo iki 3,2 sekundės. Taip yra todėl, nes funkcija n2 auga daug greičiau nei funkcija n\*logn.

Rikiavimo algoritmai buvo greitesni masyvo atveju, nes masyve galima tiesiogiai manipuliuoti reikšmėmis, kai, tuo tarpu, sąraše reikia sukeitinėti mazgų rodykles, kas kainuoja daugiau laiko.

Visais algoritmų atvejais operatyvinė atmintis buvo greitesnė už išorinę atmintį, nes paimti duomenis iš operatyvinės atminties yra daug greičiau nei nuskaityti iš failo.

Paieškos vidutinis sudėtingumas maišos lentelėje yra Θ(1), todėl paieška buvo labai greita ir atlikti paiešką visiems 3 200 000 lentelės elementams tereikėjo tik 0,5 sekundės.