## Kauno Technologijos Universitetas

### Informatikos fakultetas



**Algoritmų sudarymas ir analizė**

**Laboratorinio darbo Nr. 1**

**ataskaita**

Studentas: Giedrius Vičkus, IFE 7

Tikrino: Dalius Makackas

**Kaunas, 2010**

1. **Darbo užduotis ir variantas**

Rikiavimo uždavinys.

Palyginkite tris rūšiavimo algoritmus, kai rūšiavimas atliekamas masyve ir dinaminiame sąraše.

14 variantas.

Rikiavimui naudojami algoritmai:

1. Rikiavimas suliejimu
2. „Piramidė“ (heap sort),
3. Radix sort.
4. **Algoritmoanalizėirteorinisįvertinimas**
   1. **Rūšiavimas suliejimu** - n elementų dalinami į 2 dalis: ir Šios dalys turėtų būti surūšiuojamos ir sujungiamos. Tačiau jos vėl savo ruožtu suskaidomos iki vieno vieno elemento ir atliekamas jų sujungimas. Šios rekurentinės lygties sprendimas yra toks: f(n)=n ⎡log n⎤ - 2 ⎡log n⎤ +1.  
      Ši rekurentinė formulė sutampa su binarinio algoritmo įterpimo blogiausiu atveju.   
      Trūkumas: reikia papildomos atminties masyvui Save.

Algoritmo sudėtingumas *O*(*n* log *n*).

Pseudo kodas:

**function** merge\_sort(m)

**if** length(m) ≤ 1

**return** m

**var***list* left, right, result

**var***integer* middle = length(m) / 2

**for each** x **in** m **up to** middle

add x to left

**for each** x **in** m **after** middle

add x to right

left = merge\_sort(left)

right = merge\_sort(right)

result = merge(left, right)

**return** result

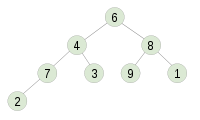
* 1. **Piramidės rikiavimo algoritmas** ([angl.](http://lt.wikipedia.org/wiki/Angl%C5%B3_kalba)*heapsort*) – [rikiavimo algoritmas](http://lt.wikipedia.org/wiki/Rikiavimo_algoritmas), kai rikiuojama duomenis sukeliant į [piramidės](http://lt.wikipedia.org/wiki/Kr%C5%ABva) (krūvos) struktūrą.

**Algoritmo etapai:**

1. Sudaroma [krūvos](http://lt.wikipedia.org/wiki/Kr%C5%ABva)[duomenų struktūra](http://lt.wikipedia.org/wiki/Duomen%C5%B3_strukt%C5%ABra)
2. Iteraciškai iš sudarytos krūvos pašalinamos šaknys. Jas išsaugome mum reikiama tvarka: pvz., realizuojant masyvu, galime ją įrašyti į masyvo gale atsilaisvinusią vietą (pašalinus šaknį ji yra pakeičiama mažiausiu krūvos lapu).

Gauta šaknų seka yra surikiuota.

**Krūvos sudarymas:**Tarkim, turime duomenų masyvą{a1, a2, ..., an}, Pirmiausia sukeiskime jo elementus vietomis, kad gautume{a1', a2', ..., an'), kur ai' >= a2i' ir ai' >= a2i+1'.

Grafiškai tai atrodys kaip dvejetainis medis, kurio kiekvienos viršūnės sūnūs yra ne didesni už tėvus.

Tarkime, turime pradinius duomenis {6, 4, 8, 7, 3, 9, 1, 2}. Masyvo elementų skaičius n=9. Pradėsime nuo pradinio elemento i=n/2=4.

Pav. 1 Sudarytapiramidė (krūva)

Elementų perkėlimas bus vykdomas taip:

i = 4

i = 2

6 4 9 7 3 8 1 2

^ ^ ^

| |

<--->

6 7 9 4 3 8 1 2

i = 1

6 7 9 4 3 8 1 2

^ ^ ^

| |

<--->

9 7 6 4 3 8 1 2

^ ^ ^

| |

<----->

9 7 8 4 3 6 1 2

6 4 8 7 3 9 1 2

^ ^

i = 3

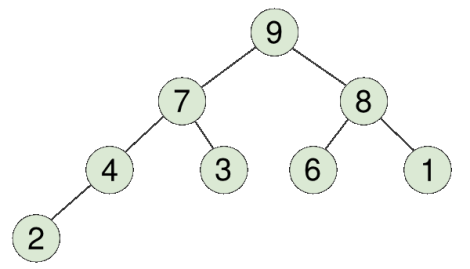
6 4 8 7 3 9 1 2

^ ^ ^

| |

<----->

6 4 9 7 3 8 1 2



Pav. 2 Surikiuotapiramidė (krūva)

Taigi gavome {9, 7, 8, 4, 3, 6, 1, 2}. Šiame masyve jau tenkinama mūsų minėta sąlyga.

Tokia duomenų struktūra vadinama [piramide (krūva)](http://lt.wikipedia.org/wiki/Kr%C5%ABva). Pirma algoritmo dalis būtent ir buvo krūvos sudarymas.

Antroji algoritmo dalis yra pats rikiavimas. Eilinėje iteracijoje imama krūvos viršūnė ir sukeičiama vietomis su paskutiniu masyvo elementu. Kadangi krūvos viršūnėje visada bus didžiausias elementas, tai jį sukeitę vietomis su paskutiniu masyvo elementu, žinosime, kad maksimalus masyvo elementas yra jo paskutinėje pozicijoje. Taip sukeitę elementus vietomis, turime atkurti krūvą. Tai padaryti paprasta, nes vietą pakeitęs būna tik vienas elementas (jis atsiduria viršūnėje). Šį elementą, jei jis nėra didžiausias, reikia perstumti keletu lygių žemiau, kol vėl gausime krūvą. Tai atlikę, pradedame naująją iteraciją, tik šį kartą jau nagrinėsime vienetu trumpesnį masyvą, nes didžiausias elementas, kuris anksčiau buvo viršūnėje, jau atsidūrė savo vietoje ir jo nagrinėti nereikia. Tokių iteracijų skaičius yra N-2.

Krūvos aukštis gali būti rastas formule \lceil \log_2 {n +1} \rceilAlgoritmo sudėtingumas: O()

* 1. **Radix sort -** vienasiš rikiavimoalgoritmų, skirtasatvejams, kaiduomenųreikšmėsyraskaitmeninėsirpriklausokokiamnorsskaitiniamintervaluiarišsiskiriapanašiomissavybėmis. Skaitmeniniorikiavimoalgoritmuoseduomenųreikšmėsinterpretuojamoskaipskaičiai M-tainėje (dažniausiai – dvejetainėje) skaičiavimosistemoje.

Algoritmasstabilusirlabaigreitas, sudėtingumas – O(N·k) (*k* – raktoilgis). Pirmasisskaitmeniniorikiavimoalgoritmaskompiuteriuiparašytas 1954 metais.Dažniausiainaudojamosskaitmeniniorikiavimoprocedūros: skaitmeniniokeitimo (*radix exchange sort*) irtiesioginioskaitmeniniorikiavimo (*straight radix sort*). SkaitmeniniokeitimometodasnaudojaapieN·logNbitųlyginimų.Abu skaitmeniniaimetodai, rikiuodami N skaičių, kuriųkiekvienasyra *b* bitųilgio, naudojamažiauneguN·bbitųlyginimų.

1. **Programos išeities tekstai**
   1. **Bubble sort**

publicvoid sortArray()

{

// einam per visą masyvą nuo pradžių

for (int i = 0; i < kiekis; i++)

// einam nuo priešpaskutinio masyvo elemento iki i-tojo

for (int j = kiekis - 1; j > i; j--)

// patikrinam ar reikia elementus sukeisti vietomis

if (mas[j] < mas[j - 1]) //(mas[j].CompareTo(mas[j - 1] < 0), jei nori daugiau, tai ">" o jei lygu, tai "=="

{

int tmp = mas[j];

mas[j] = mas[j - 1];

mas[j - 1] = tmp;

//ops += 3;

}

}

}

}

* 1. **Piramidė (heapsort)**

publicvoid sortArray()

{

int i;

int temp;

for (i = (x / 2) - 1; i >= 0; i--)

{

iApacia(i, x);

}

for (i = x - 1; i >= 1; i--)

{

temp = mas[0];

mas[0] = mas[i];

mas[i] = temp;

iApacia(0, i - 1);

}

}

publicvoid iApacia(int saknis, int apacia)

{

bool baigta = false;

int max;

int temp;

while ((saknis \* 2 <= apacia) && (!baigta))

{

if (saknis \* 2 == apacia)

max = saknis \* 2;

elseif (mas[saknis \* 2] > mas[saknis \* 2 + 1])

max = saknis \* 2;

else

max = saknis \* 2 + 1;

if (mas[saknis] < mas[max])

{

temp = mas[saknis];

mas[saknis] = mas[max];

mas[max] = temp;

saknis = max;

}

else

baigta = true;

}

}

}

}

* 1. **Bucket sort**

publicvoid skirstyti()

{

for (int i = 0; i < kiekis; i++)

{

kuris = 0;

for (int j = 0; j < kiekis; j++)

if (mas[j] < riba \* (i + 1) + min && mas[j] >= riba \* i + min)

{

dalis[kuris] = mas[j];

kuris++;

}

rikiuoti();

pildyti();

}

}

publicvoid pildyti()

{

for (int i = 0; i < kuris; i++)

{

masRez[kiekisRez] = dalis[i];

kiekisRez++;

}

}

publicvoid rikiuoti()

{

int min, minValue;

for (int i = 0; i < kuris; i++)

{

min = i;

minValue = dalis[i];

for (int j = i + 1; j < kuris; j++)

{

if (dalis[j] < minValue)

{

min = j;

minValue = dalis[j];

}

}

dalis[min] = dalis[i];

dalis[i] = minValue;

}

}

}

}

1. **Algoritmų sudėtingumas, remiantis išeities tekstais**
   1. **Bubble sort**

Išorinis cilas bus vykdomas n – 1 kartų, o vidinis n – i – 1, taigi

* 1. **Piramidė (heapsort)**

Padaroma 3\*n\*log n palyginimų, algoritmo vidutinis veikimo laikas O()

* 1. **Bucket sort**

|  |  |
| --- | --- |
| **Eilutės** | **Iteracijos** |
| 1-3 | O(m) |
| 4-5 | O(n) |
| 6-8 | O(n+m) |
| **Viso:** O(n+m) | |

taigi, algoritmo sudėtingumas yra O(n + m).

1. **Rezultatai**
   1. **C++**

Rikiavimas dinaminiame sąraše

Laikas, ms

Imtis

Pav. 3. Algoritmų rikiavimo laiko priklausomybės nuo imties dydžio

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rikiavimo algoritmas** | **Imties dydis** | | | | | | | |
| **1000** | **5000** | **10000** | **15000** | **20000** | **25000** | **30000** | **40000** |
| Bubble sort | 22 | 338 | 1409 | 3261 | 6717 | 11282 | 16503 | 30510 |
| Piramidė (heap sort) | 17 | 235 | 746 | 1707 | 3344 | 5761 | 9486 | 18421 |
| Bucket sort | 29 | 653 | 2586 | 4680 | 9150 | 12964 | 22279 | 41196 |

Pav. 4. Algoritmų rikiavimo laiko priklausomybės nuo imties dydžio

Imtis

Rikiavimas masyve

Laikas, ms

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rikiavimo algoritmas** | **Imties dydis** | | | | | | | |
| **1000** | **5000** | **10000** | **15000** | **20000** | **25000** | **30000** | **40000** |
| Bubble sort | 24 | 321 | 1235 | 2796 | 4889 | 7780 | 11145 | 19472 |
| Piramidė (heap sort) | 6 | 17 | 32 | 35 | 58 | 66 | 79 | 93 |
| Bucket sort | 33 | 577 | 2396 | 5863 | 9293 | 14177 | 23874 | 38063 |

1. **Išvados**

Iš grafikų matyti, kad greičiausias rikiavimo algoritmas iš trijų yra „Heap sort“. Jis greitas tiek masyve, tiek dinaminiame sąraše, ne taip, kaip „Bubble sort“ ir „Bucket sort“ rikiavimo algoritmai. Peršasi išvada, kad „Piramidės“ (heap sort) algoritmas pritaikytas dirbti su masyvais. Palyginus bucket sort ir heapsort su didelėmis imtimis masyve, matome, kad Heap sort yra žymiai greitesnis.