

**VILNIAUS KOLEGIJA**

ELEKTRONIKOS IR INFOMATIKOS FAKULTETAS

**Programinės įrangos katedra**

Programu sistemos (vals. kodas 6531BX028)

Gabrielė Tamaševičiūtė

PI23S **grupė**

RIKIAVIMO ALGORITMŲ LYGINAMOJI ANALIZĖ

Kursinis darbas

Darbo vadovas (-ė) Dr. Romanas Tumasonis

Pateikimo data: 2024-03-15

Vilnius

2024

**Įvadas**

Duomenų rikiavimas yra esminis programavimo mokslų elementas, kuris leidžia manipuliuoti didelius duomenų kiekius pagal įvairius parametrus, pavyzdžiui, reikšmių dydį ar abėcėlinę tvarką. Duomenų rikiavimas programavime yra būtinas daugybėje kasdienių užduočių. Duomenų mokslinikai ir analitikai dažniausiai rikiuoja duomenenų rinkinius prieš tolesnę analizę, prekių sąrašai internetinese parduotuvėse rikiuojamos pagal kainą, populiarumą, vartotojo vertinimus arba daugelį kitų kriterijų, socialinių tinklų platformose įrašai, komentarai, pranešimai – ir tie yra rikiuojami pagal tam tikrus kriterijus. Galime suvokti, kad rikiavimo algoritmų tyrimas ir optimizacija yra svarbus uždavinys, nes net ir nedidelis efektyvumo pagerėjimas gali turėti reikšmingą poveikį sistemų veikimo greičiui ir resursų naudojimui. Šio kursinio darbo tema yra "Rikiavimo algoritmų lyginamoji analizė", kurioje dėmesys skiriamas dviem skirtingų tipų algoritmams: paprastajam Burbulo rikiavimo algoritmui ir sparčiajam Suliejimo rikiavimo algoritmui. Ši tema yra itin aktuali, nes rikiavimo algoritmų pasirinkimas ir jų veikimo efektyvumas gali turėti esminę įtaką programų, kurios juos naudoja, veikimo spartai ir efektyvumui.

Kursinio darbo tikslas – išanalizuoti ir palyginti Burbulo ir Suliejimo rikiavimo algoritmų efektyvumą, taikant juos skirtingų dydžių ir išdėstymo duomenų rinkiniams. Šis tikslas yra pasirinktas siekiant suprasti, kaip šie du algoritmai veikia įvairiose situacijose ir kuris iš jų yra efektyvesnis konkrečiomis sąlygomis. Kursinio darbo uždaviniai:

1. Apibrėžti ir aprašyti Burbulo ir Suliejimo rikiavimo algoritmų teorinius principus ir jų veikimo principus, realizuoti algoritmus C++ programavimo kalba.
2. Burbulo ir Suliejimo rikiavimo algoritmų veikimo rezultatų analizė ir palyginimas.
3. Pateikti išsamias išvadas apie kiekvieno algoritmo privalumus ir trūkumus, remiantis atlikta analize.

Šio darbo tyrimo metodai apima teorinę mokslinės literatūros analizę, duomenų rikiavimo algoritmų realizacija C++ programavimo kalba ir eksperimentinį tyrimą, naudojant sukurtas programas.

**Burbulo rikiavimo algoritmo apibrėžimas ir aprašymas**

Burbulo rikiavimo algoritmas yra vienas iš paprasčiausių rikiavimo algoritmų, kurį naudojame, kai rūšiuojame mažą kiekį elementų. Šis algoritmas veikia iteruodamas per masyvą ir keisdamas vietomis gretimus elementus, jei pirmasis elementas yra didesnis už antrąjį, jie keičiami vietomis.Šis procesas kartojamas kiekvienai porai masyvo elementų, pradedant nuo pirmojo iki paskutiniojo ir tesiasi kol nebus išrikiuotas visas masyvas. Pagal Raimondą Čiegį (2007) “kadangi didžiausio elemento judėjimas panašus į vandens burbulo kilimą į paviršių, tai toks rūšiavimo algoritmas vadinamas burbulo metodu” (p.133). Burbulo rūšiavimo didėjančia tvarka algoritmo pavyzdinė programa:

#include <iostream>

using namespace std;

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

int arr[] = { 64, 34, 25, 12 };

int n = sizeof(arr) / sizeof(arr[0]);

cout << "Nesurusiuotas masyvas: \n";

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

cout << arr[i] << " ";

cout << endl;

bubbleSort(arr, n);

cout << "Surusiuotas masyvas: \n";

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

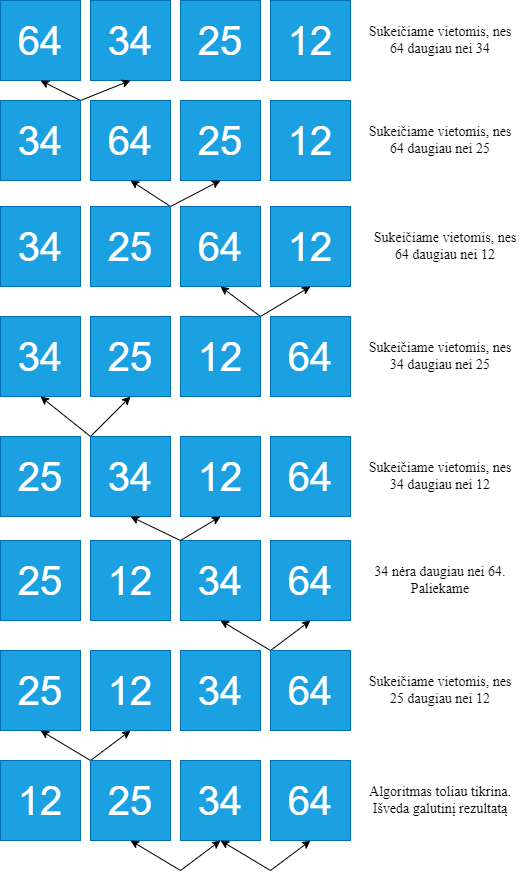
cout << arr[i] << " ";

cout << endl;

return 0;

}

Algoritmo veikimo principas:

1. Deklaruota „bubbleSort“ funkcija priima du argumentus: sveikųjų skaičių masyvą „int arr[]“ (žr 1 Pav.) ir masyvo ilgį „int n“. Funkcijos tikslas - surūšiuoti masyvą didėjimo tvarka naudojant burbulo rūšiavimo metodą.
2. Vykdome dvigubą ciklą:
   1. Išorinis ciklas („for (int i = 0; i < n - 1; i++)“) nustato, kiek kartų reikia peržiūrėti visą masyvą. Kadangi didžiausias elementas kiekvieno peržiūros metu "nuskęsta" į masyvo galą, kiekvieną kartą užtenka peržiūrėti vienu elementu mažesnę masyvo dalį, todėl ir yra „i < n - 1“.
   2. Vidinis ciklas („for (int j = 0; j < n - i - 1; j++)“) eina per masyvo elementus, kuriuos reikia palyginti ir, jei reikia, apkeisti vietomis (žr. 2 Pav.).
3. Vidiniame cikle kiekvienas elementas „arr[j]“ yra palyginamas su kitu šalimai esančiu elementu „arr[j + 1]“. Jei elementas „arr[j]“ yra didesnis už elementą „arr[j + 1]“, elementai apkeičiami vietomis naudojant kintamąjį „temp“. Tai užtirkina, kad po kiekvieno ciklo didžiausias elementas iš nesurūšiuotu nukeliautų į masyvo galą.
4. Prieš iškviečiant „bubbleSort“ funkciją atspausdiname pradinį masyvą „arr[]“, kad lengviau galėtume palyginti surušiuotą ir nesurušiuotą masyvą.
5. Iškviečiame „bubbleSort“ funkciją perduodant „arr[]“ kaip parametrą, kad funkcija surušiuotų masyvą.
6. Po rūšiavimo funkcijos iškvietimo, masyvas „arr[]“ yra vėl atspausdinamas, bet šį kartą kaip „Surūšiuotas masyvas“, demonstruodamas, kaip elementai buvo išrikiuoti didėjimo tvarka.
7. Rezultatas, toks, kad konsolėje gauname išvestus du masyvus, pirminį bei išrikiuotą (žr. 3).

**Suliejimo Rikiavimo Algoritmo Apibrėžimas ir Aprašymas**

Suliejimo rikiavimo algoritmas yra efektyvus algoritmas, naudingas didelių duomenų rinkinių rikiavimui, kuris naudoja „sakldyk ir valdyk“ principą padalindamas problemą į mažesnes, lengviau sprendžiamas dalis. Pagal Raimondą Čiegį (2007) „nesunku sujungti su surūšiuotus poaibius į vieną surūšiuotą aibę. Todėl metodas ir vadinamas *suliejimo* algoritmu“. Suliejimo rūšiavimo didėjančia tvarka algoritmo pavyzdinė programa:

#include <iostream>

#include <vector>

using namespace std;

void merge(vector<int>& arr, int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

vector<int> L(n1), R(n2);

for (int i = 0; i < n1; i++)

L[i] = arr[l + i];

for (int j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

int i = 0;

int j = 0;

int k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(vector<int>& arr, int l, int r) {

if (l >= r) {

return;

}

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

int main() {

vector<int> arr = { 12, 11, 13, 5 };

int arr\_size = arr.size();

cout << "Nesurusiuotas masyvas: \n";

for (int i = 0; i < arr\_size; i++)

cout << arr[i] << " ";

cout << endl;

mergeSort(arr, 0, arr\_size - 1);

cout << "Surusiuotas masyvas: \n";

for (int i = 0; i < arr\_size; i++)

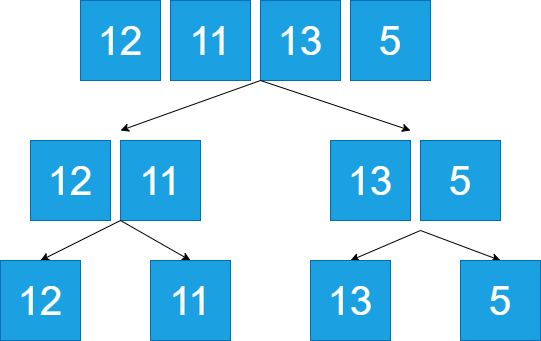
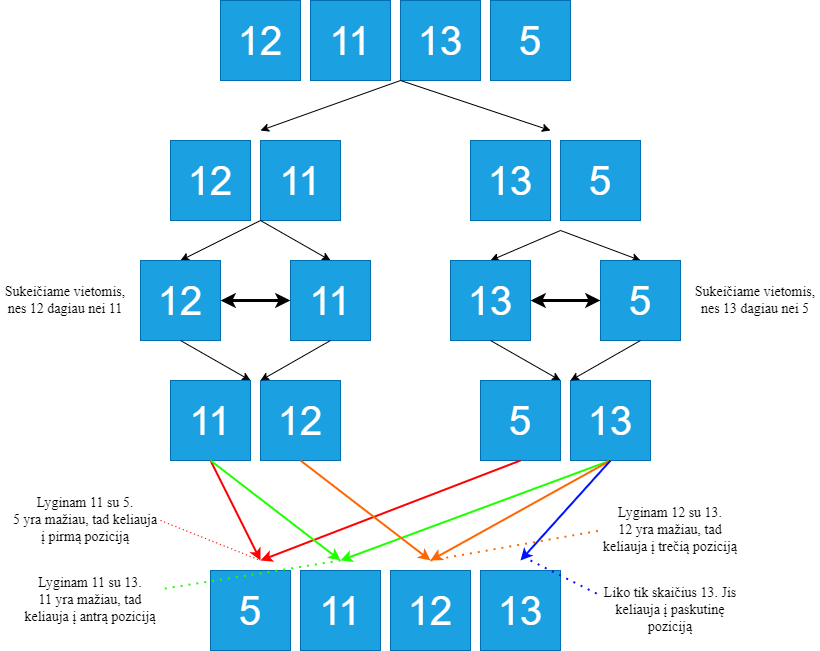
cout << arr[i] << " ";

cout << endl;

return 0;

}

Algoritmo veikimo principas:

1. Programa apibrėžia dvi funkcijas: „merge“, kuri atsakinga už sub-masyvų sujungimą ir rūšiavimą, ir „mergeSort“, kuri atsakinga už masyvo skaidymą ir rekursyvų algoritmo kvietimą.
2. Funkcija „mergeSort“ rekursyviai skaido masyvą į mažesnes dalis, naudodama vidurio tašką „m“. Skaidymas tęsiamas, kol sub-masyvuose lieka vienas elementas (žr. 3 Pav.) 
3. Funkcija „merge“ sujungia du surūšiuotus sub-masyvus į vieną surūšiuotą masyvą. Ji naudoja laikinus masyvus „L“ ir „R“, į kuriuos kopijuoja sub-masyvų duomenis. Tada, palyginus elementus, ji juos įterpia atgal į pagrindinį masyvą „arr“ jau išrikiuotai.
4. Funkcijoje „merge“, kai du sub-masyvai sujungiami, kiekvieno sub-masyvo elementai palyginami ir mažesnis iš jų įterpiamas į pagrindinį masyvą (žr. 4 Pav.).
5. Prieš kviečiant „mergeSort“ funkciją, pradinis masyvas „arr“ atspausdinamas konsolėje, kad būtų galima pamatyti pradinę masyvo būklę.
6. Funkcija „mergeSort“ iškviečiama pradiniam masyvui „arr“, pradedant nuo nulinės pozicijos 0 ir baigiant masyvo ilgiu minus vienas.
7. Baigus rūšiavimą, masyvas „arr“ vėl atspausdinamas, bet šį kartą kaip "Surūšiuotas masyvas". Tai parodo, kaip algoritmas išrikiavo elementus didėjimo tvarka.
8. Galiausiai, konsolėje rodomi du masyvai: pradinis nesurūšiuotas ir galutinis surūšiuotas, leidžiantis vizualiai įvertinti algoritmo atliktą rūšiavimą.

**1Rikiavimo algoritmų efektyvumo analizė**

**1.1Tyrimo metodika**

**1.1.1Duomenų generavimas**

Siekiant atlikti rikiavimo algoritmų efektyvumo analizę, buvo parengta programa C++ programavimo kalba, kuri generuoja tris skirtingus duomenų rinkinius: atsitiktinai sugeneruotus, atvirkščiai surūšiuotus ir surūšiuotus duomenis. Duomenys generuojami naudojant atsitiktinių skaičių generatorių, o jų kiekis gali būti 5000, 10000, 50000 arba 100000 elementų. Generuoti duomenys įrašomi į CSV failus, kurių pavadinimai atspindi duomenų rinkinio tipą ir elementų skaičių.

Duomenų generavimo algoritmai aprašyti pagal šiuos principus:

* Nesurikiuoti duomenys: kiekvienam elementui priskiriamas atsitiktinis skaičius nuo 1 iki 100000.
* Atvirkščiai surūšiuoti duomenys: sugeneruojami atsitiktiniai skaičiai, nuo 1 iki 100000, kurie tuomet surikiuojami mažėjimo tvarka.
* Surūšiuoti duomenys: sugeneruojami atsitiktiniai skaičiai, nuo 1 iki 100000, kurie tuomet surikiuojami didėjimo tvarka.

**1.1.2Rikiavimo algoritmų analizės įgyvendinimas**

Analizei atlikti sukurtas dar vienas programinis sprendimas C++ programavimo kalba, kuris nuskaito anksčiau sugeneruotus duomenų failus ir taiko du skirtingus rikiavimo algoritmus – burbulo ir suliejimo. Kiekvieno algoritmo rikiavimo procesas kartojamas dešimt kartų, bei apskaičiuojamas vidutinis surūšiavimo laikas, siekiant užtikrinti matavimo tikslumą. Laiko matavimui tyrimo procese taikoma <chrono> C++ standartinės bibliotekos klasė high\_resolution\_clock, kuri leidžia fiksuoti programos vykdymo laiką su tikslumu iki nanosekundžių. Tai užtikrina laiko matavimo tikslumą ir leidžia detaliai įvertinti rikiavimo algoritmų veikimo greitį.

**1.2** **Burbulo ir suliejimo algoritmų efektyvumo lyginimas**

Analizė buvo atlikta su aukščiau minėtais sugeneruotais skirtingais duomenų rinkiniais: atsitiktinai sugeneruoti, atvirkščiai surūšiuoti ir surūšiuoti duomenis. Kiekvienas duomenų rinkinys buvo testuotas su abiejų rikiavimo algoritmų pagalba, pradedant nuo mažiausio (5000 elementų) iki didžiausio (100000 elementų) kiekio. Siekiant aiškumo ir suprantamumo, rezultatai pateikti ir aptariami sekundžių tikslumu.

|  |  |  |
| --- | --- | --- |
| Duomenys | Burbulo algoritmo vidutinis greitis (sekundes) | Suliejimo algoritmo vidutinis greitis (sekundes) |
| 100000 Atvirkščiai srūšiuoti | 377.745 | 0.3165 |
| 100000 Nesurūšiuoti | 278.191 | 0.7354 |
| 100000 Surūšiuoti | 183.1147 | 0.2541 |
| 50000 Atvirkščiai srūšiuoti | 73.5791 | 0.262 |
| 50000 Nesurūšiuoti | 72.8452 | 0.3003 |
| 50000 Surūšiuoti | 27.2987 | 0.1721 |
| 10000 Atvirkščiai srūšiuoti | 5.5438 | 0.0676 |
| 10000 Nesurūšiuoti | 2.4374 | 0.0476 |
| 10000 Surūšiuoti | 0.8385 | 0.0317 |
| 5000 Atvirkščiai srūšiuoti | 1.1632 | 0.0327 |
| 5000 Nesurūšiuoti | 0.5179 | 0.0208 |
| 5000 Surūšiuoti | 0.2137 | 0.0149 |

Lentele 1

Rezultatai rodo, kad burbulo algoritmo veikimo laikas yra gerokai ilgesnis nei suliejimo algoritmo visiems tyrimo atvejams (žr. lentel 1). Suliejimo algoritmas pasirodė ypač efektyvus, nepriklausomai nuo duomenų rinkinio dydžio ir duomenų išdėstymo.

Grafikas nr1 vaizduoja bendrą skirtumą tarp burbulo ir suliejimo algoritmų rikiavimo veikimo laiko. Tai leidžia vizualiai įvertinti ir palyginti abiejų algoritmų našumą pagal skirtingus duomenų rinkinius.

* 5000 elementų duomenys: suliejimo algoritmas surikiavo atvirkščiai surūšiuotus, nesurūšiuotus bei jau surūšiuotus duomenis greičiau nei burbulo algoritmas, rodydamas atitinkamai 1.13, 0.5 ir 0.2 sekundžių pranašumą.
* 10000 elementų duomenys: skirtumas tarp algoritmų dar padidėjo, suliejimo algoritmo pranašumas siekė 5.48, 2.39 ir 0.81 sekundes.
* 50000 elementų duomenys: suliejimo algoritmas parodė dar didesnį efektyvumo skirtumą - 73.32, 72.54 ir 27.13 sekundes greitesnis nei burbulo algoritmas.
* 100000 elementų duomenys: suliejimo algoritmas rikiavo atvirkščiai surūšiuotus, nesurūšiuotus bei jau surūšiuotus duomenis net 377.43, 277.46 ir 182.86 sekundes greičiau nei burbulo algoritmas.

Didžiausias laiko skirtumas užfiksuotas su 100000 elementų atvirkščiai surūšiuotų duomenų rinkiniu, kur burbulo algoritmas rikiavimui reikalavo daugiau nei šešis šimtus kartų daugiau laiko nei suliejimo algoritmas. Tai parodo, kad burbulo algoritmas yra itin neefektyvus tvarkant didelius atvirkščiai surūšiuotus duomenų rinkinius, lyginant su suliejimo algoritmu.

Mažiausias laiko skirtumas stebėtas su 5000 elementų surūšiuotų duomenų rinkiniu, kur burbulo algoritmas buvo tik šiek tiek lėtesnis už suliejimo algoritmą. Tai rodo, kad burbulo algoritmo veikimo laiko padidėjimas nėra toks reikšmingas su mažesniais ir iš anksto surūšiuotais duomenų rinkiniais.

Vertinant grafiko duomenis, akivaizdu, kad burbulo algoritmo našumas smarkiai krenta su duomenų rinkinio dydžio augimu. Suliejimo algoritmas demonstruoja žymiai greitesnį rūšiavimą palyginti su visais duomenų rinkiniais.

Analizuojant algoritmų gebėjimą nustatyti, kad duomenys jau yra surūšiuoti, suliejimo algoritmas parodė žymiai geresnius rezultatus. Suliejimo algoritmas, palyginti su burbulo algoritmu, greičiau nustatė, kad duomenys jau yra išrikiuoti ir nedidino veikimo laiko, kas yra būdinga burbulo algoritmui, kuriam reikia daugiau laiko patikrinti visą duomenų seką.

Kalbant apie elementų sukeitimą vietomis, burbulo algoritmas, veikiantis pagal pradinį principą, kad kiekviename žingsnyje sukeičiami gretimi elementai, jei jie nėra tinkamoje sekoje, atliko daugiau tokių operacijų nei suliejimo algoritmas. Pastarasis yra efektyvesnis, nes sujungia jau dalinai surūšiuotus duomenų blokus, taip mažindamas sukeitimo operacijų skaičių.

Apibendrinant, galima teigti, kad suliejimo algoritmo vidutinis greitis buvo daugmaž nuoseklus nepriklausomai nuo duomenų rūšies, o burbulo algoritmo greitis smarkiai skyrėsi priklausomai nuo duomenų išsidėstymo. Visais atvejais suliejimo algoritmas parodė esąs greitesnis už burbulo algoritmą.

**Išvada**

Išanalizavus ir palyginus abiejų rikiavimo algoritmų veikimo laikus su skirtingais duomenų rinkiniais, pastebėta, kad suliejimo algoritmo efektyvumas nekinta ženkliai priklausomai nuo duomenų išdėstymo ar jų kiekio, išlaikydamas santykinai pastovų greitį. Tuo tarpu burbulo algoritmo veikimo laikas smarkiai padidėjo su duomenų kiekio didėjimu, ypač su atvirkščiai surūšiuotais duomenimis, kur jis rodė labai ilgus veikimo laikus. Šis stebėjimas parodo, kad burbulo algoritmas yra neefektyvus didelėms duomenų aibėms, o suliejimo algoritmas yra daug stabilesnis ir tinka didesnėms duomenų aibėms rikiuoti.

Analizė taip pat atskleidė, kad suliejimo algoritmas yra žymiai efektyvesnis nei burbulo algoritmas ne tik tvarkant nesurūšiuotus duomenis, bet ir atpažįstant jau išrikiuotus duomenų rinkinius. Tai yra svarbu, nes realiose situacijose dažnai pasitaiko atvejų, kai reikia tvarkyti jau iš dalies surūšiuotus duomenis.