

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**Lygiagreti „Merge sort“ rikiavimo algoritmo implementacija**

Individualus darbas

**Darbą pateikė:**

Nerijus Dulkė IFF-6/11

**KAUNAS, 2018**

# Užduotis

„Merge sort“ rikiavimo algoritmas paremtas „skaldyk ir valdyk“ metodu. Šis algoritmas iš pradžių suskaldo sąraša į dvi lygias dalis ir jas sujungia surikiuota tvarka. Užduoties tikslas palyginti paprastą ir lygiagrečią šio algoritmo realizaciją.

# Užduoties analizė ir sprendimo būdas

Užduočiai realizuoti bus naudojama GO programavimo kalba. Programos testavimui bus naudojami atsitiktinai sugeneruoti sveikūjų skaičių masyvai.

„Merge sort“ algoritmas skaldo masyvą į dvi dalis tol, kol masyve lieka tik vienas elementas, o tik tada pradeda jungti šiuos masyvus ir kartu juos rikiuoja. Dėl šios savybės su dideliais kiekiais duomenų, kiekvieną suskaldytą masyvą rikiuoti atskiroje gijoje būtų neefektyvu, nes daug naujų gijų kurimas užimtų daugiau laiko nei pats rikiavimas. Šiai problemai spresti bus naudojamas parametras *threshold*. Šis parametras pasakys programai kokio mažiausio dydžio turi būti masyvas, kad reikėtų jo rikiavima leisti atskiroje gijoje.

# Programos aprašymas

Programa sudaryta iš 4 pagrindinių funkcijų:

**main**

Ši funkcija yra pagrindinė, joje sugeneruojami duomenys, paleidžiami rikiavimo algoritmai, skaičiuojamas jų veikimo laikas ir atspausdinami rezultatai.

**mergeSort**

Parametrai:

* arr – sveikūjų skaičių masyvas, kurį reikia surikiuoti

Ši funkcija realizuoja paprastą „Merge sort“ algoritmą. Rekursiškai kviečia save, skaldant masyvą per pusę, kol lieka tik vienas elementas, tada kviečiama **merge** funkcija.

**merge**

Parametrai:

* left – pirmasis sveikūjų skaičių masyvas, kurį norima sujungti
* right – antrasis sveikūjų skaičių masyvas, kurį norima sujungti

Ši funkcija sujungia duotus masyvus į viena ir grąžina surikiuotą masyvą.

**parallelMergeSort**

Parametrai:

* arr – sveikūjų skaičių masyvas, kurį reikia surikiuoti
* threshold – sveikasis skaičius nurodantis kokio mažiausio dydžio masyvas turi būti, kad jo rikiavimą vykdyti atskiroje gijoje

Ši funkcija realizuoja „Merge sort“ algoritmą, masyvus rikiuojant lygiagrečiai.

# Programos tekstas

// pagrindinė funkcija

func main() {

    // duomenu skaicius

    count := 1000000

    // nuo kokio masyvo dydzio nustoti skaldyti i atskiras gijas

    // jei -1, skaldyti visada

    threshold := 100000

    var start time.Time

    var elapsed time.Duration

    // uzpildomas masyvas su atsitiktiniais skaiciais

    arr := rand.Perm(count)

    fmt.Println("Rikiuojamo masyvo elementu skaicius:", count)

    fmt.Println("Nuo kokio masyvo dydzio nustoti skaldyti i atskiras gijas:", threshold)

    fmt.Println()

    // matuojamas lygiagretaus merge sort veikimo laikas

    start = time.Now()

    parallelMergeSort(arr, threshold)

    elapsed = time.Since(start)

    fmt.Println("Lygiagretaus merge sort laikas:", elapsed)

    // matuojamas paprasto merge sort veikimo laikas

    start = time.Now()

    mergeSort(arr)

    elapsed = time.Since(start)

    fmt.Println("Paprasto merge sort laikas:", elapsed)

    // fmt.Println("Paprastas", res)

    // fmt.Println("Lygiagretus", res2)

}

// lygiagreti merge sort realizacija

func parallelMergeSort(arr []int, threshold int) []int {

    if len(arr) <= 1 {

        return arr

    }

    // masyvas suskaldomas i dvi dalis

    mid := len(arr) / 2

    left := arr[:mid]

    right := arr[mid:]

    // i atskiras gijas skaidyti tik iki tam tikro dydzio

    if threshold > 0 && len(arr) > threshold {

        var wg sync.WaitGroup

        wg.Add(2)

        // suskaldyi masyvai rekursiskai perduodami toliau skaldyti

        go func() {

            left = parallelMergeSort(left, threshold)

            wg.Done()

        }()

        go func() {

            right = parallelMergeSort(right, threshold)

            wg.Done()

        }()

        // laukiama kol gijos baigs darba

        wg.Wait()

    } else {

        // suskaldyi masyvai rekursiskai perduodami toliau skaldyti

        left = parallelMergeSort(left, threshold)

        right = parallelMergeSort(right, threshold)

    }

    // masyvai sujungiami ir surikiuojami

    return merge(left, right)

}

// paprasta merge sort realizacija

func mergeSort(arr []int) []int {

    if len(arr) <= 1 {

        return arr

    }

    // masyvas suskaldomas i dvi dalis

    mid := len(arr) / 2

    left := arr[:mid]

    right := arr[mid:]

    // suskaldyi masyvai rekursiskai perduodami toliau skaldyti

    left = mergeSort(left)

    right = mergeSort(right)

    // masyvai sujungiami ir surikiuojami

    return merge(left, right)

}

func merge(left []int, right []int) []int {

    lsize := len(left) - 1

    rsize := len(right) - 1

    size := lsize + rsize + 2

    i, j := 0, 0

    arr := make([]int, size)

    // rikiavimas ir jungimas

    for k := 0; k < size; k++ {

        if i > lsize && j <= rsize {

            arr[k] = right[j]

            j++

        } else if (j > rsize && i <= lsize) || (left[i] < right[j]) {

            arr[k] = left[i]

            i++

        } else {

            arr[k] = right[j]

            j++

        }

    }

    return arr

}

# Testavimo instrukcija

Programos testavimui užtenka keisti du parametrus *main()* funkcijoje:

* count – nurodo kokio dydžio masyvas bus rikiuojamas
* threshold - nurodo kokio mažiausio dydžio masyvas turi būti, kad jo rikiavimą vykdyti atskiroje gijoje

Programa paleidžiama per komandinę eilutę naudojanat komandą:

*go run IFF.6.11.Dulke.Nerijus.IP.go*

# Vykdymo laiko kitimo tyrimas

Tyrimui naudojamas kompiuteris su šiom specifikacijomis:

* Intel I7-8550 Procesorius
* 4 fiziniai branduoliai su „Intel Hyperthreading“ technologija (8 loginiai branduoliai)
* 8GB RAM

|  |  |  |  |
| --- | --- | --- | --- |
| Duomenų masyvo dydis | Threshold parametras | Paprasto „Merge sort“ vykdymo laikas | Lygiagretaus „Merge sort“ vykdymo laikas |
| 1 000 000 | -1 (visi masyvai atskirose gijose) | 147.2533ms | 228.9461ms |
| 1 000 000 | 500 000 | 147.2533ms | 133.3437ms |
| 1 000 000 | 100 000 | 147.2533ms | 77.7489ms |
| 1 000 000 | 10 000 | 147.2533ms | 86.9549ms |
| 1 000 000 | 1 000 | 147.2533ms | 80.8635ms |
| 1 000 000 | 100 | 147.2533ms | 224.9983ms |
| 1 000 000 | 10 | 147.2533ms | 245.9941ms |
| 10 000 000 | -1 (visi masyvai atskirose gijose) | 2.0375171s | 3.0970654s |
| 10 000 000 | 5 000 000 | 2.0375171s | 1.3422544s |
| 10 000 000 | 1 000 000 | 2.0375171s | 758.5562ms |
| 10 000 000 | 10 000 | 2.0375171s | 726.1902ms |
| 10 000 000 | 100 | 2.0375171s | 1.1340168s |
| 10 000 000 | 10 | 2.0375171s | 4.5288578s |

# Išvados

Individualaus darbo užduotis buvo atlikta naudojant GO programavimo kalbą ir *goroutine* priemones. Iš rezultataų galima teigti, kad užduotis įgyvendinta ir priemonės yra tinkamos šios užduoties realizavimui. Iš rezultatų taip pat galime teigti, kad lygiagreti „Merge sort“ realizacija gali veikti greičiau nei paprasta realizacija. Tačiau tai priklauso nuo *threshold* paramtero. Jei egzistuoja labai didelis duomenų kiekis, efektyviau yra jį rikiuoti iki tam tikro gijų skaičiaus.

# Literatūra

* + - <https://golang.org/>
    - <https://gobyexample.com/>