Students: Ussabekov Madiyar, Nurumgaliyev Azatkali

Original solution:   
type writer struct {

    writing\_buf \*[]byte

}

func (w \*writer) write\_to\_temp\_buf(byte byte) {

    \*w.writing\_buf = append(\*w.writing\_buf, byte)

*//fmt.Println("WRITER: Written bytes to writing\_buf", byte)*

}

func (w \*writer) write\_to\_chan(ch chan []byte) {

    ch <- \*w.writing\_buf

*//fmt.Println("WRITER: Send byte slice to chan", writing\_buf)*

    \*w.writing\_buf = nil

}

type reader struct {

    words  \*[]record

    rating \*[]int

}

type record struct {

    word    []byte

    counter int

    checked bool

}

func (r \*reader) contains(element []byte) (bool, int) {

    for index, v := range \*r.words {

        if bytes.Equal(v.word, element) {

            return true, index

        }

        index = index + 1

    }

    return false, 0

}

func (r \*reader) read\_from\_chan(ch chan []byte) {

    for node := range ch {

        state, index := r.contains(node)

        if state {

            (\*r.words)[index].counter++

        } else {

            record := record{node, 1, false}

            \*r.words = append(\*r.words, record)

        }

    }

}

func (r \*reader) get20mostfrequentwords() {

    list := make([]int, 20)

    r.rating = &list

    for index, \_ := range \*r.rating {

        temp := 0

        inss := 0

        for index, v := range \*r.words {

            if (v.checked == false) && (v.counter > temp) {

                temp = v.counter

                inss = index

            }

        }

        (\*r.words)[inss].checked = true

        (\*r.rating)[index] = inss

    }

}

func (r \*reader) print() {

    for \_, v := range \*r.rating {

        fmt.Print((\*r.words)[v].counter, " ", string((\*r.words)[v].word))

        fmt.Println()

    }

}

*//I created two structs with the methods to write and read from the shared channel of []bytes. String is not allowed, so we assume that slice of bytes is a word.*

*//Writer and reader works in different goroutines.*

*//Writer goes through the text and collects bytes to buffer until it reaches the space(which means the end of the word),*

*//or reaches the symbol which is not a letter(in that case we check do we already have a word in our buffer).*

*//After this it sends the content of the buffer to channel, and continues to go through the text*

*//At the same time reader listens to the channel. Whenever it gets the word ([]byte) it checks does it have the same word inside the slice of already written words.*

*//If it does, it increases the counter of this record, if no, it appends the record to it.*

*//After writer and reader both finished working with a channel, reader goes through the slice of words to determine the most 20 frequent words. I did this with an assumption that,*

*//any sorting between approximately 74k elements while reading, or quick sorting 10k elements after reading,*

*//will lead to way more comparisons than going through the slice and just retrieving 20 elements with the largest counter(10k elements in slice, 74k words in text)*

*//P.S Using one goroutine showed better execution time, idk why, but I decided to go with two goroutines version.*

*//I used approx 5 hours to code version with no goroutines, and one extra hour to code version that u see*

func main() {

    start := time.Now()

    file, err := os.Open("mobydick.txt") *//open file*

    if err != nil {

        log.Fatal(err)

    }

    defer func() {

        if err = file.Close(); err != nil {

            log.Fatal(err)

        }

    }()

    readingBuf := make([]byte, 1) *//read file by one letter only*

    words := make([]record, 0)

    reader := reader{words: &words} *//creating reader*

    writingBuf := make([]byte, 0)

    writer := writer{&writingBuf} *//writer*

    ch := make(chan []byte) *//channel that we will use to pass slices of bytes from writer to reader*

*//btw reader listens in range of elements that are passed to channel, it will stop working when there are no elements left, so we don't need any wait groups*

    go func() {

        for {

*//reading file's letters one by one*

            n, err := file.Read(readingBuf)

            if n > 0 {

                byteVal := readingBuf[0]

                if byteVal >= 65 && byteVal <= 90 { *//if symbol is uppercase letter*

                    byteVal = byteVal + 32

                    writer.write\_to\_temp\_buf(byteVal) *//writing to temporary buffer*

                } else if byteVal >= 97 && byteVal <= 122 { *//if symbol is lowercase letter*

                    writer.write\_to\_temp\_buf(byteVal) *//writing to temporary buffer*

                } else if byteVal == 32 && len(writingBuf) != 0 { *//if symbol is [space], and we have letters in our buffer*

                    writer.write\_to\_chan(ch) *//send temporary buffer content to channel, empty the temporary buffer*

                } else if ((byteVal > 122 || byteVal < 65) || (byteVal > 90 && byteVal < 97)) && len(writingBuf) != 0 { *//if symbol is any other than letter or space, and we have letters in our buffer*

                    writer.write\_to\_chan(ch) *//send temporary buffer content to channel, empty the temporary buffer*

                } else {

                    continue

                }

            }

            if err == io.EOF {

                writer.write\_to\_chan(ch) *//send temporary buffer content to channel, empty the temporary buffer*

                break

            }

        }

        close(ch) *//close channel, so our that our reader will stop working after there are no elements left, in other case reader will cause deadlock*

    }()

    reader.read\_from\_chan(ch) *//reading from channel in range of elements in channel*

    reader.get20mostfrequentwords() *//getting 20 most frequent words, and write it to rating slice*

    reader.print()                  *//print elements from words according to the rating list*

    fmt.Printf("Process took %s\n", time.Since(start))

}

Our solution:

type readerMy struct {

    words \*[]recordMy

}

type recordMy struct {

    word    []byte

    counter int

}

func (r \*readerMy) get20mostfrequentwordsMy() {

    sort.Slice(\*r.words, func(i, j int) bool {

        return (\*r.words)[i].counter > (\*r.words)[j].counter

    })

    for i := 0; i < 20; i++ {

        fmt.Println((\*r.words)[i].counter, " ", string((\*r.words)[i].word))

    }

}

func main() {

file, err := os.Open("mobydick.txt") *//open file*

    if err != nil {

        log.Fatal(err)

    }

    defer func() {

        if err = file.Close(); err != nil {

            log.Fatal(err)

        }

    }()

    readingBuf := make([]byte, 1) *//read file by one letter only*

    words := make([]recordMy, 0)

    reader := readerMy{words: &words} *//creating reader*

    writingBuf := make([]byte, 0)

    ch := make(chan []byte) *//channel that we will use to pass slices of bytes from writer to reader*

*//btw reader listens in range of elements that are passed to channel, it will stop working when there are no elements left, so we don't need any wait groups*

    r := bufio.NewReader(file)

    go func() {

        for {

*//reading file's letters one by one*

            n, err := r.Read(readingBuf)

            if n > 0 {

                byteVal := readingBuf[0]

                if byteVal >= 65 && byteVal <= 90 { *//if symbol is uppercase letter*

                    byteVal = byteVal + 32

                    writingBuf = append(writingBuf, byteVal) *//writing to temporary buffer*

                } else if byteVal >= 97 && byteVal <= 122 { *//if symbol is lowercase letter*

                    writingBuf = append(writingBuf, byteVal) *//writing to temporary buffer*

                } else if byteVal == 32 && len(writingBuf) != 0 { *//if symbol is [space], and we have letters in our buffer*

                    ch <- writingBuf

                    writingBuf = nil

                } else if ((byteVal > 122 || byteVal < 65) || (byteVal > 90 && byteVal < 97)) && len(writingBuf) != 0 { *//if symbol is any other than letter or space, and we have letters in our buffer*

                    ch <- writingBuf

                    writingBuf = nil

                } else {

                    continue

                }

            }

            if err == io.EOF {

                ch <- writingBuf

                writingBuf = nil *//send temporary buffer content to channel, empty the temporary buffer*

                break

            }

        }

        close(ch) *//close channel, so our that our reader will stop working after there are no elements left, in other case reader will cause deadlock*

    }()

    reader.read\_from\_chan(ch) *//reading from channel in range of elements in channel*

    reader.get20mostfrequentwordsMy()

}

Benchmarks:

Original: BenchmarkMain-12 1 1 530 847 600 ns/op 2 690 248 B/op 79 657 allocs/op

Our solution: BenchmarkAzatkali2 3 404 281 300 ns/op 2 444 576 B/op 79 620 allocs/op

Difference: additional 2 more times execution, 1 112 566 300 ns/op, 245 672B/op and 37 allocs/op

The process of optimization:

1. removed

writer := writer{&writingBuf}

and it’s methods calls, instead working with

writingBuf := make([]byte, 0)

directly.

2. instead of reading from file directly, used Reader:

r := bufio.NewReader(file)

3. In get20mostfrequentwords, we used different approach: through the sort.Slice:

func (r \*readerMy) get20mostfrequentwordsMy() {

    sort.Slice(\*r.words, func(i, j int) bool {

        return (\*r.words)[i].counter > (\*r.words)[j].counter

    })

    for i := 0; i < 20; i++ {

        fmt.Println((\*r.words)[i].counter, " ", string((\*r.words)[i].word))

    }

}

And removed from structs unused attributes:

type readerMy struct {

    words \*[]recordMy

}

type recordMy struct {

    word    []byte

    counter int

}