**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н. Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н. Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# Отчёт по лабораторной работе №5
# по дисциплине «Анализ алгоритмов»

**Тема:** Конвейерные вычисления
**Студент:** Княжев А. В.
**Группа:** ИУ7-52Б
**Оценка (баллы):** _____
**Преподаватели:** Волкова Л. Л., Строганов Ю. В.

Москва — 2022 г.

# Оглавление

# Введение

В современном мире часто возникает задача обработки данных в несколько этапов. Так как в этом случаи этапы зависят друг от друга по данным, их нельзя распараллелить привычными способами. В этом случае часто применяют конвейерную обработку данных [1].

## Цель работы

Получение навыков кодирования программного продукта, тестирования и проведения замерного эксперимента работы программы на различных данных. Все это на примере решения задачи конвейерной обработки данных для расчета частоты вхождения термов в набор слов.

## Задачи работы

1) изучение этапов обработки текстов:
   - приведение к начальной форме;
   - дедупликация;
   - расчет частоты вхождения терма в текст;
2) изучение алгоритмов конвейерной обработки данных;
3) разработка алгоритмов параллельной и последовательной конвейерной обработки данных;
4) кодирование данных алгоритмов;
5) проведение замерного эксперимента для данных алгоритмов, с измерением времени работы;
6) проведение сравнительного анализа алгоритмов на основе полученных данных.

# 1. Аналитическая часть

В данном разделе рассмотрены теоретические выкладки по конвейерной обработке данных и этапам обработки текста: приведению к начальной форме, дедупликации и вычисления частоты терма.

## 1.1. Конвейерная обработка данных

Конвейер — способ обработки данных поэтапно: под каждый этап выделен свой обработчик — лента конвейера. Обрабатываемые обработчиком элементы — заявки, помещаются в очередь на обработку. После того, как заявка обрабатывается, она помещается в очередь к следующему обработчику.

Пример схемы конвейера представлен на рис. 1.1

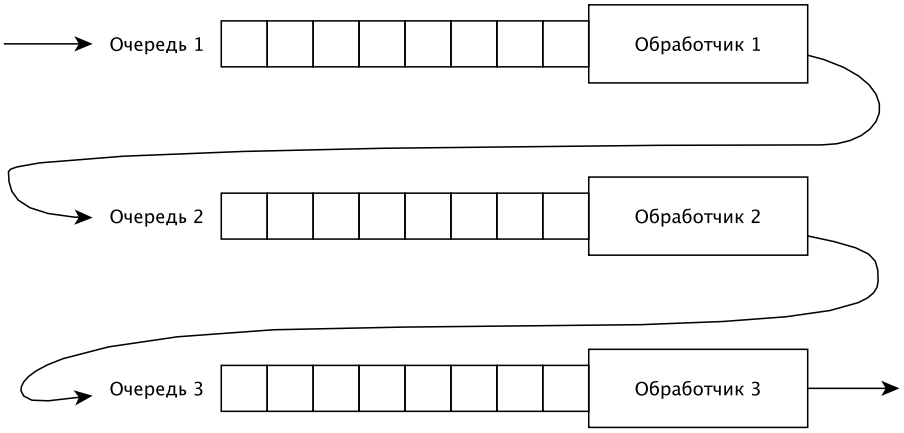


Рисунок 1.1 — Схема конвейера

### 1.1.1. Последовательный конвейер

В последовательном конвейере одновременно может находиться не более одной заявки, то есть пока последний обработчик не вернет заявку, первый не может принять следующую.

### 1.1.2. Параллельный конвейер

В параллельном конвейере обработчики работают параллельно, или квазипараллельно. Такая схема позволяет находиться и обрабатываться в системе более чем одной заявке.

## 1.2. Этапы обработки текста

### 1.2.1. Приведение к начальной форме

Приведение к начальной форме, или нормализация — процесс приведения слова к его начальной форме. Пример такого преобразования: «автомобилем» → «автомобиль». Приведение к начальной форме необходимо для корректного статистического анализа текста, так как нет смысла рассматривать формы одного и того же слова как отдельные слова. Кроме того, нормализация слов сильно упрощает работу с текстом, в том числе поиск [2].

### 1.2.2. Дедупликация

Дедупликация представляет собой удаление дубликатов в наборе. В рамках данной задачи, будет производиться удаление дубликатов в списке терм, то есть слов в начальной форме. Кроме того, будет производится подсчет количества вхождений слова в текст.

### 1.2.3. Вычисление частоты терма

Частота терма $TF$ — величина, являющаяся характеристикой важности терма в тексте [3], которая вычисляется по формуле:

$$TF = \frac{N_w}{N}, \tag{1.1}$$

где

— $N_w$ — количество вхождений данного слова в текст;

— $N$ — количество слов в тексте.

# 2. Конструкторская часть

В данном разделе представлены схемы алгоритмов конвейерной обработки данных и этапов обработки текста.

## 2.1. Разработка алгоритма приведения слова к начальной форме

В рамках данной работы не рассматривается алгоритм приведения слова к начальной форме. Реализован алгоритм будет с использованием внешних библиотек.

## 2.2. Разработка алгоритма дедупликации текста

На рис. 2.1 представлена схема алгоритма дедупликации текста.

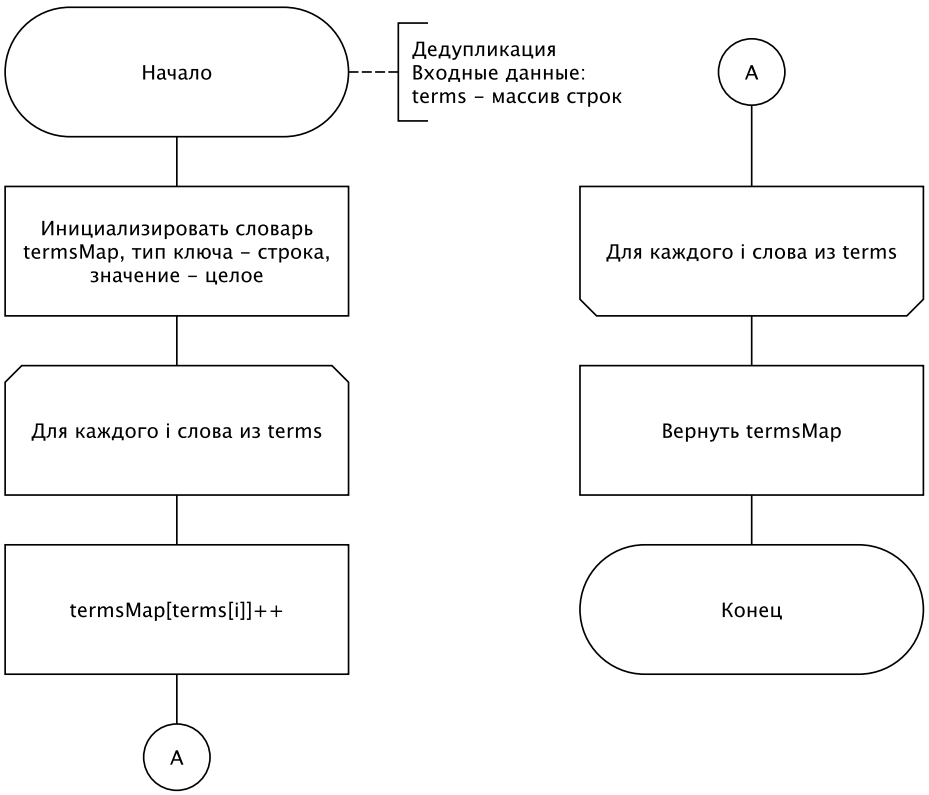


Рисунок 2.1 — Схема алгоритма дедупликации текста

## 2.3. Разработка алгоритма нахождения частоты терма в тексте

На рис. 2.2 представлена схема алгоритма нахождения частоты терма в тексте.

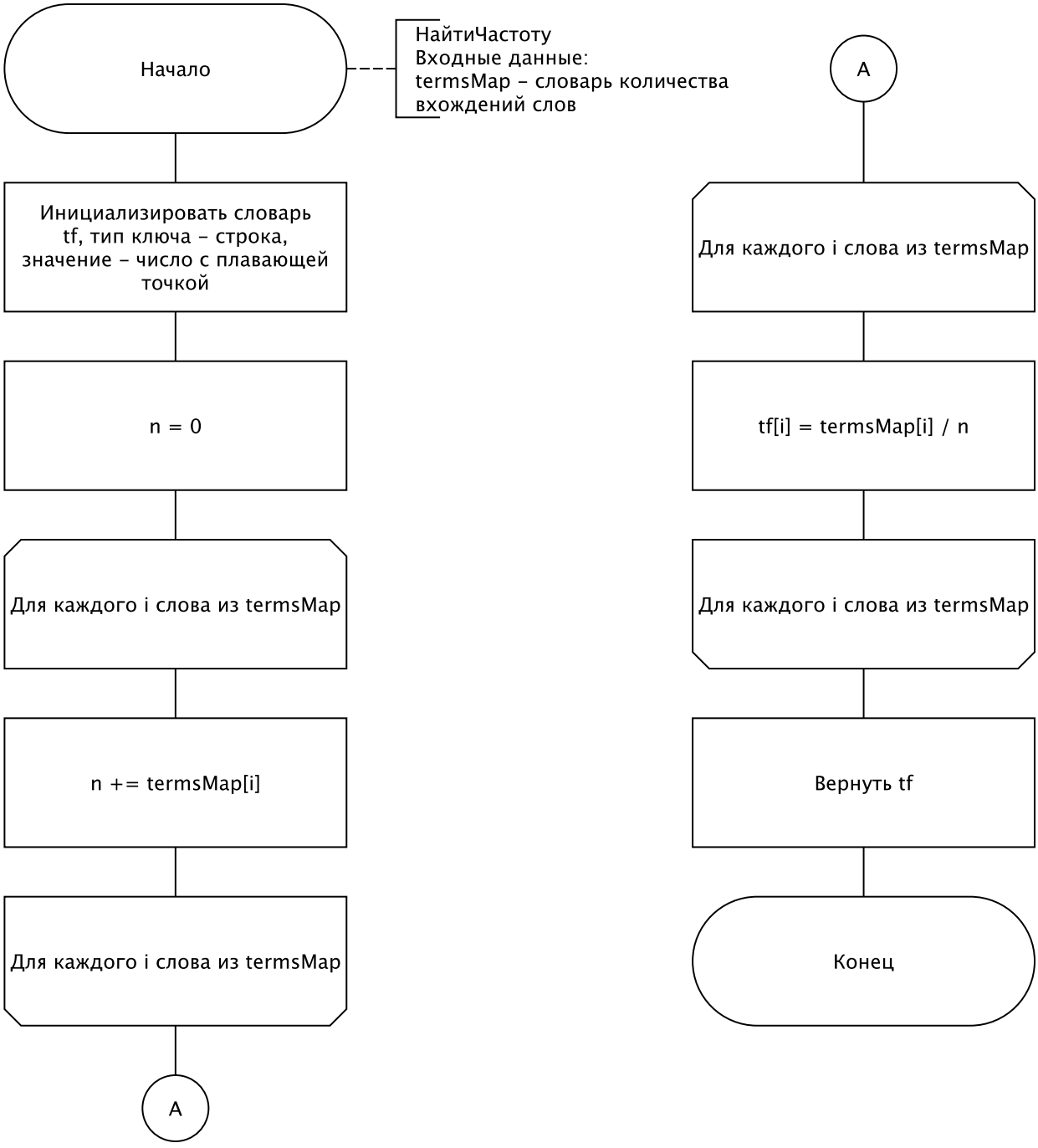


Рисунок 2.2 — Схема алгоритма нахождения частоты терма в тексте

## 2.4. Разработка алгоритма последовательной конвейерной обработки текста

На рис. 2.3 представлена схема алгоритма последовательной конвейерной обработки текста.

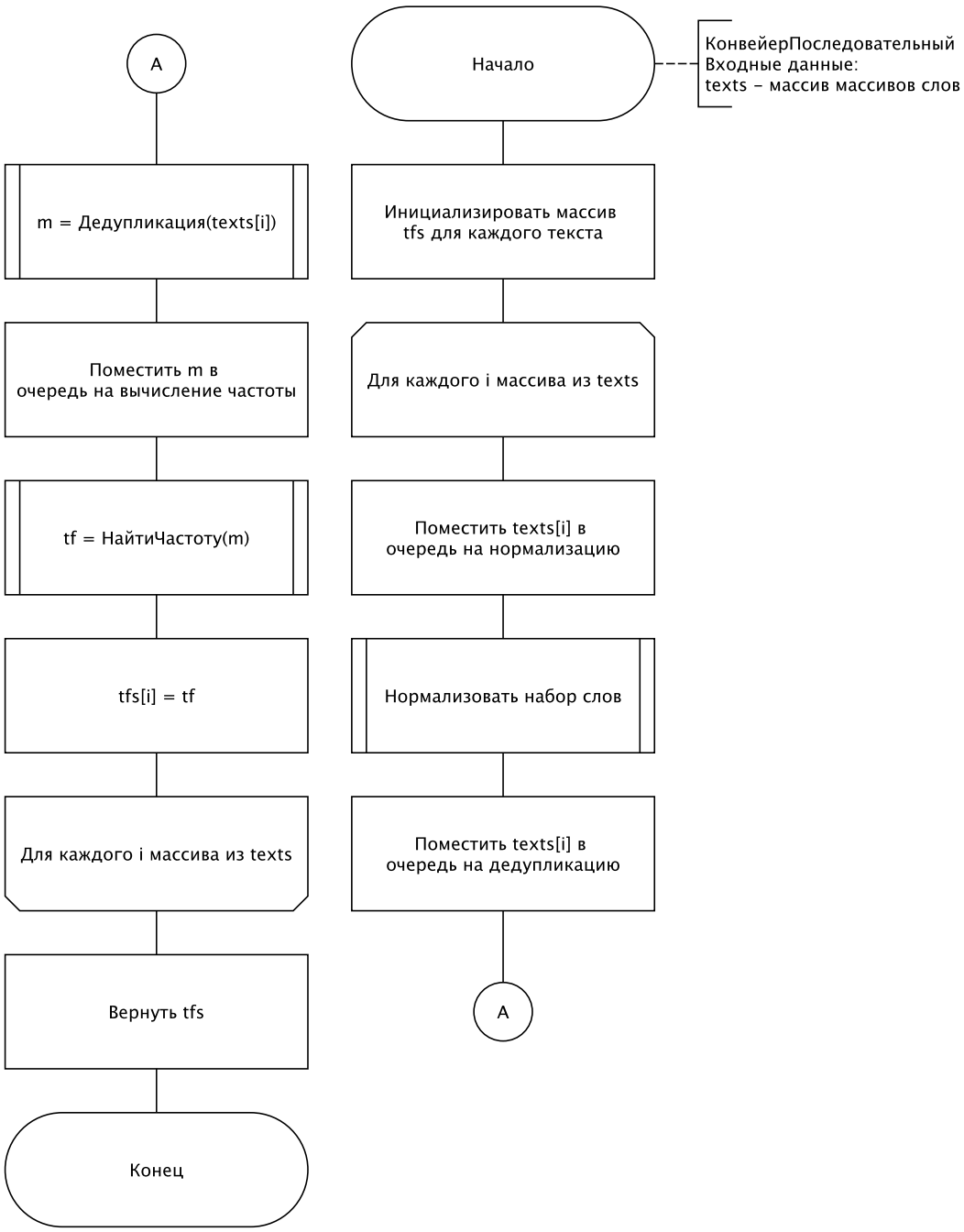


Рисунок 2.3 — Схема алгоритма последовательной конвейерной обработки текста

## 2.5. Разработка алгоритма параллельной конвейерной обработки текста

На рис. 2.4 – 2.5 представлена схема алгоритма параллельной конвейерной обработки текста.

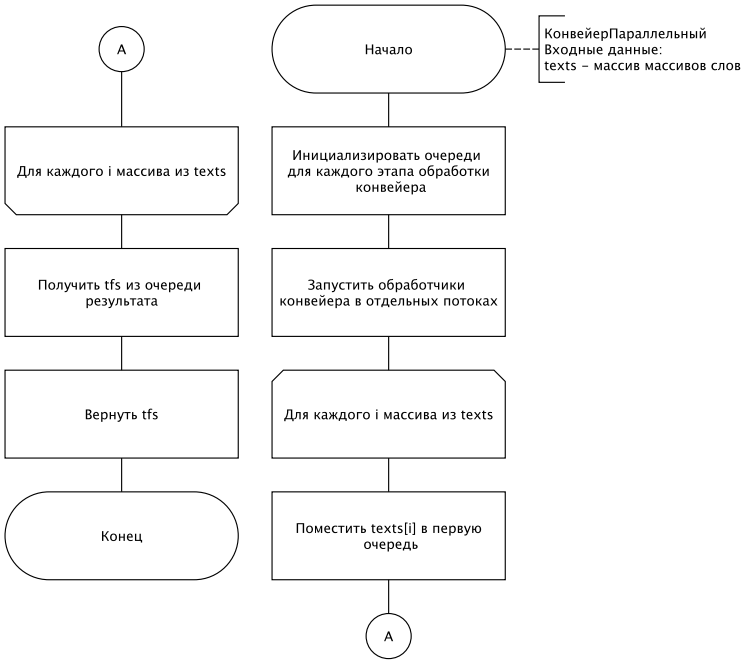


Рисунок 2.4 — Схема алгоритма параллельной конвейерной обработки текста

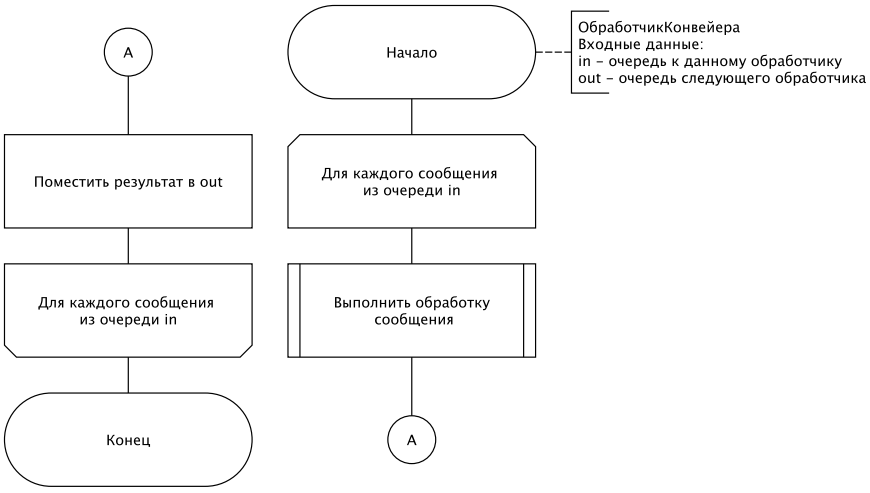


Рисунок 2.5 — Схема алгоритма обработчика конвейера

# 3. Технологическая часть

В данном разделе представлены реализации алгоритмов сортировки. Кроме того, указаны требования к ПО и средства реализации.

## 3.1. Требования к ПО

— программа позволяет вводить имя файла, содержащего информацию о наборах слов, с помощью аргументов командной строки;

— программа аварийно завершается в случае ошибок, выводя сообщение о соответствующей ошибке;

— программа выполняет замеры времени работы реализаций алгоритмов;

— программа строит зависимости времени работы реализаций алгоритмов от размеров входных данных;

— программа принимает на вход файл в следующем формате: наборы слов (тексты) разделены последовательностью символов «***», слова разделены символами переноса строки;

— программа принимает на вход набор массивов слов, нормализует его, удаляет дубликаты и рассчитывает частоту терма для каждого терма в тексте.

## 3.2. Средства реализации

Для реализации данной работы выбран язык программирования Go, так как он содержит необходимые для тестирования библиотеки, а также обладает достаточными инструментами для реализации ПО, удовлетворяющего требованиям данной работы [4].

## 3.3. Реализации алгоритмов

В листингах 3.1 – 3.3 представлены реализации алгоритмов обработки текста.

В листингах 3.4 – 3.6 представлены реализации алгоритмов обработчиков конвейера.

В листингах 3.7 – 3.8 представлены реализации алгоритмов конвейерной обработки данных.

Листинг 3.1 — Реализация алгоритма нахождения начальных форм слов

```
func Normalize(words []string) []string {
        terms := make([]string, 0)

        for _, v := range words {
                _, normals, _ := morph.Parse(strings.ToLower(v))
                if len(normals) == 0 {
                        panic("no normals")
                }

                terms = append(terms, normals[0])
        }

        return terms
}
```

Листинг 3.2 — Реализация алгоритма дедупликации

```
func Deduplicate(terms []string) map[string]int {
        termsMap := make(map[string]int)

        for _, v := range terms {
                termsMap[v] = termsMap[v] + 1
        }

        return termsMap
}
```

Листинг 3.3 — Реализация алгоритма подсчета частоты терма

```
func CountTF(termsMap map[string]int) map[string]float64 {
        n := 0

        for _, v := range termsMap {
                n += v
        }

        tfs := make(map[string]float64)

        for k, v := range termsMap {
                tfs[k] = float64(v) / float64(n)
        }

        return tfs
}
```

Листинг 3.4 — Обработчик первой ленты конвейера — нормализация текста

```
func Normalize(in <-chan task.Task, out chan<- task.Task) {
        for t := range in {
                t.StartTime1 = time.Now()
                t.Terms = text.Normalize(t.Text)
                t.EndTime1 = time.Now()

                out <- t
        }

        close(out)
}
```

Листинг 3.5 — Обработчик второй ленты конвейера — дедупликация текста

```
func Deduplicate(in <-chan task.Task, out chan<- task.Task) {
        for t := range in {
                t.StartTime2 = time.Now()
                t.Stat = text.Deduplicate(t.Terms)
                t.EndTime2 = time.Now()

                out <- t
        }

        close(out)
}
```

Листинг 3.6 — Обработчик третьей ленты конвейера — поиск частоты слова

```
func CountTF(in <-chan task.Task, out chan<- task.Task) {
        for t := range in {
                t.StartTime3 = time.Now()
                t.TFs = text.CountTF(t.Stat)
                t.EndTime3 = time.Now()

                out <- t
        }

        close(out)
}
```

Листинг 3.7 — Реализация последовательного конвейера

```
func Pipeline(texts [][]string, log bool) []map[string]float64 {
        tfs := make([]map[string]float64, len(texts))
        resTasks := make([]task.Task, 0, len(texts))
        now := time.Now()

        for _, v := range texts {
                chanNormalize := make(chan task.Task, 1)
                chanDeduplicate := make(chan task.Task, 1)
                chanCountTF := make(chan task.Task, 1)
                chanResult := make(chan task.Task, 1)

                t := task.Task{
                        Text: v,
                        TimeStat: task.TimeStat{
                                StartTime0: now,
                        },
                }

                chanNormalize <- t
                close(chanNormalize)

                workers.Normalize(chanNormalize, chanDeduplicate)
                workers.Deduplicate(chanDeduplicate, chanCountTF)
                workers.CountTF(chanCountTF, chanResult)
                resTasks = append(resTasks, <-chanResult)
        }

        for i, v := range resTasks {
                tfs[i] = v.TFs
        }

        return tfs
}
```

Листинг 3.8 — Реализация параллельного конвейера

```
func Pipeline(texts [][]string, log bool) []map[string]float64 {
        tfs := make([]map[string]float64, len(texts))
        resTasks := make([]task.Task, 0, len(texts))
        now := time.Now()
        chanNormalize := make(chan task.Task, len(texts))
        chanDeduplicate := make(chan task.Task, len(texts))
        chanCountTF := make(chan task.Task, len(texts))
        chanResult := make(chan task.Task, len(texts))
        go workers.Normalize(chanNormalize, chanDeduplicate)
        go workers.Deduplicate(chanDeduplicate, chanCountTF)
        go workers.CountTF(chanCountTF, chanResult)

        for _, v := range texts {
                t := task.Task{
                        Text: v,
                        TimeStat: task.TimeStat{
                                StartTime0: now,
                        },
                }

                chanNormalize <- t
        }
        close(chanNormalize)
        for t := range chanResult {
                resTasks = append(resTasks, t)
        }
        for i, v := range resTasks {
                tfs[i] = v.TFs
        }

        return tfs
}
```

## 3.4. Тестирование

Тестирование проводилось по методологии чёрного ящика. **Тесты пройдены успешно.**

В таблице 3.1 представлены тестовые данные для реализаций алгоритмов конвейерной обработки данных.

Таблица 3.1 — Тестовые данные для алгоритмов конвейерной обработки данных

| № | Набор слов | Результат |
|---|---|---|
| 1 | «Автомобили»<br>«ехали»<br>«и»<br>«едут»<br>«и»<br>«будут»<br>«ехать» | «и»: 0.285714<br>«быть»: 0.142857<br>«автомобиль»: 0.142857<br>«ехать»: 0.428571 |
| 2 | «тридцать»<br>«девять»<br>«около»<br>«девяти» | «тридцать»: 0.250000<br>«девять»: 0.500000<br>«около»: 0.250000 |
| 3 | «летел»<br>«лебедь»<br>«по»<br>«синему»<br>«морю» | «лететь»: 0.200000<br>«лебедь»: 0.200000<br>«по»: 0.200000<br>«синий»: 0.200000<br>«море»: 0.200000 |

# 4. Экспериментальная часть

В данном разделе описаны замерные эксперименты и представлены результаты исследования.

## 4.1. Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование [6]:

— 8 ГБ оперативной памяти;

— процессор Apple M2 (тактовая частота — до 3.5ГГц);

— операционная система macOS Ventura 13.0.

## 4.2. Измерение реального времени выполнения реализаций алгоритмов

Для измерения реального времени выполнения реализаций алгоритмов была использована стандартная библиотека языка программирования Go, позволяющая замерять время работы реализаций алгоритмов в микросекундах [5].

### 4.2.1. Время работы реализаций алгоритмов

В таблице 4.1 представлены результаты измерений реального времени выполнения в зависимости от количества заявок для алгоритмов конвейеризации. На рисунке 4.1 представлена зависимость времени выполнения от количества заявок.

Таблица 4.1 — Результаты замеров реального времени (в мкс)

| Кол-во заявок | Последовательный | Параллельный |
|---|---|---|
| 1 | 416 | 442 |
| 2 | 828 | 838 |
| 3 | 1244 | 1236 |
| 4 | 1649 | 1636 |
| 5 | 2075 | 2030 |
| 6 | 2475 | 2426 |
| 7 | 2888 | 2820 |
| 8 | 3345 | 3217 |
| 9 | 3747 | 3619 |
| 10 | 4126 | 4017 |
| 12 | 4953 | 4807 |
| 14 | 5780 | 5613 |
| 16 | 6671 | 6393 |
| 18 | 7489 | 7189 |
| 20 | 8259 | 7987 |

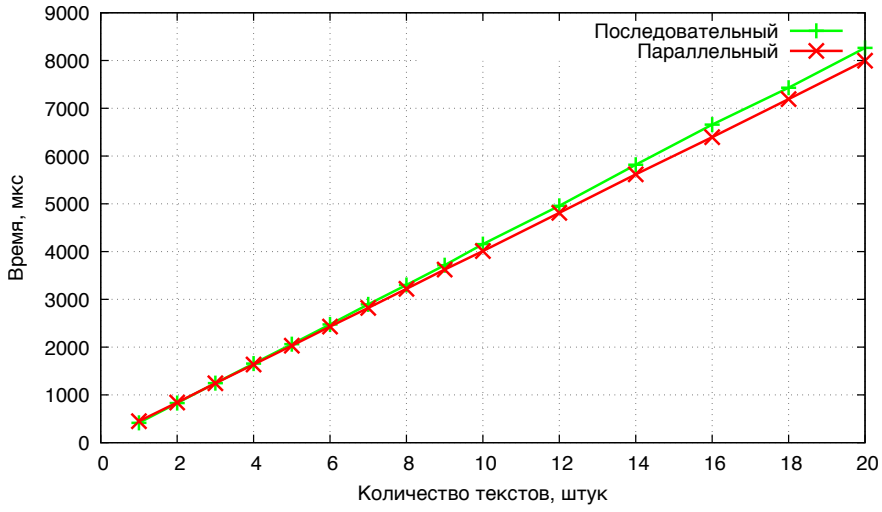


Рисунок 4.1 — Результаты замеров времени

### 4.2.2. Журналирование

В таблицах 4.2 и 4.3 представлен пример журналирования времени обработки заявок каждым обработчиком последовательного и параллельного конвейера для количества заявок, равной трем.

Таблица 4.2 — Пример журналирования на последовательном конвейере

| № заявки | Этап | Начало | Конец |
|---|---|---|---|
| 1 | 1 | 1 | 648 |
| 1 | 2 | 648 | 681 |
| 1 | 3 | 681 | 704 |
| 2 | 1 | 705 | 1098 |
| 2 | 2 | 1099 | 1124 |
| 2 | 3 | 1124 | 1154 |
| 3 | 1 | 1155 | 1549 |
| 3 | 2 | 1549 | 1576 |
| 3 | 3 | 1576 | 1600 |

Таблица 4.3 — Пример журналирования на параллельном конвейере

| № заявки | Этап | Начало | Конец |
|---|---|---|---|
| 1 | 1 | 10 | 695 |
| 1 | 2 | 712 | 752 |
| 1 | 3 | 754 | 783 |
| 2 | 1 | 697 | 1099 |
| 2 | 2 | 1108 | 1138 |
| 2 | 3 | 1139 | 1163 |
| 3 | 1 | 1101 | 1493 |
| 3 | 2 | 1496 | 1525 |
| 3 | 3 | 1526 | 1552 |

### 4.2.3. Анализ характеристик работы реализаций алгоритмов

В таблице 4.4 представлен анализ характеристик работы реализаций алгоритмов конвейерной обработки. Анализ характеристик производится для количества заявок, равному двадцати.

Таблица 4.4 — Анализ характеристик работы реализаций алгоритмов конвейерной обработки

| Характеристика | | Последовательно, мкс | | | Параллельно, мкс | | |
|---|---|---|---|---|---|---|---|
| Линия | | 1 | 2 | 3 | 1 | 2 | 3 |
| Простой очереди | sum | 83232 | 0 | 0 | 80654 | 206 | 20 |
| | min | 1 | 0 | 0 | 12 | 2 | 0 |
| | max | 8065 | 0 | 0 | 7826 | 17 | 2 |
| | avg | 4161 | 0 | 0 | 4032 | 10 | 1 |
| Время заявки в системе | min | 683 | | | 788 | | |
| | max | 8469 | | | 8289 | | |
| | avg | 4584 | | | 4510 | | |

# Заключение

Параллельная конвейерная обработка данных дает преимущество в плане производительности при работе с наборами данных, требующих последовательной обработки. При количестве заявок, равном двадцати, параллельный конвейер работает в 1.03 раз быстрее последовательного.

Среднее время простоя в первой очереди при параллельном конвейере в 1.03 раза меньше, чем при последовательном. При этом времена простоя во второй и третьей очереди равны нулю. Такое поведения связано с тем, что первый обработчик конвейера работает медленнее, чем второй и третий. Поэтому заявки накапливаются перед первым обработчиком, а после выдачи из первого обработчика сразу принимаются вторым, так как второй обработчик данных простаивает в данном случае.

Цель работы была достигнута: были изучены алгоритмы конвейерной обработки данных в контексте решения задачи расчета частоты вхождения термов в набор слов. Были выполнены все задачи:

— изучены этапы обработки текстов;

— изучены алгоритмы конвейерной обработки данных;

— разработаны алгоритмы параллельной и последовательной конвейерной обработки данных;

— кодированы данные алгоритмы;

— проведен замерный эксперимент для данных алгоритмов, с измерением времени работы;

— проведен сравнительный анализа алгоритмов на основе полученных данных.

# Список использованных источников


1. Allan V. H. et al. Software pipelining //ACM Computing Surveys (CSUR). – 1995. – Т. 27. – No. 3. – С. 367-432.

2. Федотов А. М. и др. Модель определения нормальной формы слова для казахского языка //Вестник Новосибирского государственного университета. Серия: Информационные технологии. – 2015. – Т. 13. – №. 1. – С. 107-116.

3. Петровский В. И., Бондарев В. Н. Модуль анализа естественно-языкового текста //Интеллектуальные системы, управление и мехатроника-2016. – 2016. – С. 368-372.

4. Документация по языку программирования *Go* [Электронный ресурс]. Режим доступа: *https://go.dev/doc* (дата обращения: 07.10.2022).

5. Документация по пакетам языка программирования *Go* [Электронный ресурс]. Режим доступа: *https://pkg.go.dev* (дата обращения: 07.10.2022).

6. Техническая спецификация ноутбука *MacBookAir* [Электронный ресурс]. Режим доступа: *https://support.apple.com/kb/SP869* (дата обращения: 08.10.2022).