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»

ОТЧЕТ

по рубежному контролю № 1

по курсу «Анализ алгоритмов»

Студент ИУ7-56Б _____ М. Ю. Вольняга
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата) (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Метод параллельных вычислений на основе нативных потоков — это подход к обработке данных и выполнению программ, при котором задача разбивается на множество подзадач, которые могут быть выполнены одновременно на нескольких процессорах или ядрах. Основным преимуществом нативных потоков является их способность эффективно использовать многопроцессорные и многоядерные системы [1].

Целью данной работы является реализация программного обеспечения, кластеризующего 60 текстов (новостная, художественная и научная тематика) алгоритмом иерархической кластеризации.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- описать алгоритм иерархической кластеризации;
- спроектировать программное обеспечение, реализующее алгоритм и его параллельную версию;
- разработать программное обеспечение, реализующее алгоритм и его параллельную версию.

1 Аналитический раздел

В данном разделе будут рассмотрены многопоточность, алгоритмы классификации полнотекстовых документов, алгоритмы классификации без учителя, иерархические алгоритмы, алгоритм дивизимной иерархической кластеризации, алгоритм дивизимной иерархической кластеризации и алгоритм k-средних.

1.1 Многопоточность

Многопоточность — это возможность центрального процессора одновременно выполнять несколько потоков, используя ресурсы одного процессора. Поток представляет собой набор инструкций, которые могут исполняться параллельно с другими потоками того же процесса [2].

Процесс — это программа в процессе выполнения. Запуск программы или приложения создает процесс, который может включать в себя один или несколько потоков. Поток, будучи частью процесса, выполняет задачи приложения. Процесс завершается, когда все его потоки завершают работу. Каждый поток в операционной системе рассматривается как задача для процессора. Современные процессоры способны выполнять несколько задач на одном ядре, создавая виртуальные ядра, или имеют несколько физических ядер, что характерно для многоядерных процессоров [3].

При разработке многопоточной программы важно учитывать, что простое последовательное выполнение потоков не раскрывает весь потенциал многопоточности. Эффективность достигается за счет параллельного выполнения независимых потоков, что сокращает общее время выполнения процесса.

Основной проблемой многопоточности является совместный доступ к общим ресурсам. Ключевое ограничение заключается в предотвращении одновременной записи в одну и ту же область памяти из нескольких потоков. В этом контексте важную роль играет «мьютекс» (от англ. mutex — mutual exclusion, взаимное исключение), который представляет собой примитив синхронизации. Мьютекс позволяет потокам работать с данными в монопольном режиме, предотвращая одновременный доступ. Если два потока пытаются захватить один и тот же мьютекс, только один из них сможет это сделать, а второй будет ожидать его освобождения [3].

Инструкции, выполняемые между захватом и освобождением мьютекса,

образуют так называемую *критическую секцию*. Поскольку другие потоки не могут выполнить критическую секцию, пока мьютекс захвачен, важно минимизировать объем и продолжительность критической секции во избежание задержек в выполнении [3].

1.2 Алгоритмы классификации полнотекстовых документов

Классификация текстов — ключевая задача в компьютерной лингвистике, охватывающая алгоритмы с учителем и без учителя, и имеющая важное значение для обеспечения информационной безопасности. Алгоритмы с учителем используют предварительно размеченные данные для обучения, в то время как алгоритмы без учителя, такие как кластеризация, организуют данные на основе внутренних закономерностей [4].

В данной лабораторной работе исследуется применение алгоритмов классификации без учителя для определения групп документов с помощью метода иерархической кластеризации с дивизимным подходом. Целью является выявление кластеров документов, таким образом, чтобы документы внутри одного кластера были максимально схожи по смыслу, а документы из различных кластеров — значительно отличались. Особенность данного подхода заключается в отсутствии необходимости в предварительной разметке данных и определении количества кластеров, что открывает широкие возможности для анализа неструктурированных данных [4]. Кластеризация — это разбиение элементов некоторого множества на группы по принципу схожести. Эти группы принято называть кластерами [5].

1.3 Алгоритмы классификации без учителя

Алгоритмы классификации без учителя разбивают набор документов на группы, где одна группа содержит родственные документы, а разные группы содержат разные документы. Без обучающего подмножества и известных категорий, алгоритм кластеризации автоматически определяет количество и состав кластеров, используя расстояния между документами [4].

Кластеризация текстов основана на идее, что похожие документы подходят к одним и тем же запросам, а разные документы подходят к разным запросам [4].

Исследование проводится над набором документов вида:

$$D = \{d_j \mid j = 1, \ldots, |D|\}, \tag{1.1}$$

содержащей разнообразные тематические классы. Цель алгоритмов классификации без учителя — автоматически классифицировать документы на кластеры вида:

$$C = \{c_j \mid j = 1, \ldots, |C|\}, \tag{1.2}$$

так чтобы каждый кластер представлял собой группу тематически схожих документов. Задача кластеризации сводится к определению оптимального множества кластеров C, удовлетворяющего заданным критериям качества [4].

1.4 Иерархические алгоритмы

Иерархические алгоритмы создают структурированное множество кластеров — иерархию, которое может оказаться весьма информативным для некоторых приложений [4].

В рамках иерархической кластеризации существуют агломеративные (восходящие) и дивизимные (нисходящие) подходы. Агломеративные методы последовательно объединяют мелкие кластеры в более крупные, в то время как дивизимные методы начинают с одного большого кластера и делят его на более мелкие. Дивизимные методы часто требуют дополнительных алгоритмов для определения способа разделения и могут быть более эффективными при ограничении процесса до верхних уровней иерархии без полного разделения на индивидуальные документы. Сложность дивизимного алгоритма зависит от выбранного дополнительного алгоритма плоской кластеризации, если выбран алгоритм k-средних, то $O(|D|)$ [4].

1.5 Представление данных в задаче

В данной лабораторной работе рассматривается использование иерархической кластеризации для анализа текстовых документов. Основа алгоритма — представление документов в виде векторов терминов с весами, вычисленными по методу *TF-IDF*.

Входные данные

Для каждого документа в наборе вида (1.1) формируется вектор:

$$\mathbf{d}_i = (d_{i1}, \ldots, d_{iT}), \tag{1.3}$$

где T — количество уникальных терминов во всех документах, а d_{ij} — вес j-го термина в i-м документе.

Вычисление весов терминов

Веса терминов рассчитываются по формулам:

$$d_{ij} = \frac{w_{ij}}{\|\mathbf{w}_i\|}, \tag{1.4}$$

$$w_{ij} = tf_{ij} \times \log\left(\frac{|D|}{df_j}\right). \tag{1.5}$$

В формулах (1.4) – (1.5) **используются следующие** обозначения: tf_{ij} — частота j-го термина в i-м документе, $|D|$ — общее количество документов в наборе, df_j — документная частота термина j, и $\|\mathbf{w}_i\|$ — евклидова норма вектора весов i-го документа.

Выходные данные

Алгоритм генерирует структуру кластеров, представленную иерархическим деревом, и метки кластеров вида (1.2), характеризующие группы родственных документов.

1.6 Алгоритм дивизимной иерархической кластеризации

1) Вход: множество проиндексированных документов D вида (1.1).
2) Вычислить веса терминов для каждого документа с использованием метода *TF-IDF*, по формуле (1.4).
3) Изначально все элементы принадлежат одному кластеру C_i, $i = 0$ (1.1).
4) Разделить кластер на два подкластера C_{i+1} и C_{i+2}.
5) Применить алгоритм *k-средних* к подкластерам C_{i+1} и C_{i+2}.
6) Увеличить i и повторить шаги 4 и 5 до достижения желаемой структуры кластеризации.

7) Возвратить итоговую структуру кластеров.

1.7 Алгоритм k-средних

При заранее известном числе кластеров k, алгоритм k-средних начинает с некоторого начального разбиения документов и уточняет его, оптимизируя целевую функцию – среднеквадратичную ошибку кластеризации как среднеквадратичное расстояние между документами и центрами их кластеров:

$$e(D, C) = \sum_{j=1}^{k} \sum_{i:d_i \in C_j} \|d_i - \mu_j\|^2, \tag{1.6}$$

где μ_j — центр, или центроид, кластера C_j, $|C| = k$, вычисляющийся по формуле

$$\mu_j = \frac{1}{|C_j|} \sum_{i:d_i \in C_j} d_i, \tag{1.7}$$

где $|C_j|$ — количество документов в C_j. Идеальным кластером алгоритм k - средних считает сферу с центроидом в центре сферы.

Алгоритм k-средних состоит из следующих шагов [4].

1) *Вход:* множество проиндексированных документов D, количество кластеров k.

2) Назначить начальные центры для кластеров $\{\mu_j\},\ j = 1,\ ...,\ k$ случайным образом.

3) Установить каждому кластеру C_j пустой набор, $j = 1,\ ...,\ k$.

4) Для каждого документа $d_i \in D$ выполнить:

   – найти ближайший центр кластера $j^* := \arg\min_j \|\mu_j - d_i\|,\ j = 1,\ ...,\ k$;

   – добавить документ d_i в соответствующий кластер $C_{j^*} := C_{j^*} \cup \{d_i\}$.

5) Для каждого кластера C_j обновить центр как среднее его элементов:

$$\mu_j := \frac{1}{|C_j|} \sum_{i:d_i \in C_j} d_i.$$

6) Если условие остановки не достигнуто, вернуться к шагу 4.

7) *Выход:* множество центров кластеров $\{\mu_j\}$ и множество самих кластеров C.

Вывод

В данном разделе были рассмотрены многопоточность, алгоритмы классификации полнотекстовых документов, алгоритмы классификации без учителя, иерархические алгоритмы, алгоритм дивизимной иерархической кластеризации, алгоритм дивизимной иерархической кластеризации и алгоритм k-средних.

2 Конструкторский раздел

В данном разделе будут представлены требования к программному обеспечению, описание используемых типов данных и схемы реализуемых алгоритмов.

2.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

– должен присутствовать интерфейс для выбора действий;

– считывание данных должно производиться из файла;

– результат должен записываться в файл.

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры и типы данных:

– массив символов для хранения терма;

– вещественное число для хранения *TF-IDF* терма;

– мьютекс — примитив синхронизации.

2.3 Разработка алгоритмов

На рисунке 2.1 приведена схема дивизимной иерархической кластеризации. На рисунке 2.2 приведена схема алгоритма k-средних. На рисунке 2.3 представлена схема алгоритма создания потоков для алгоритма дивизимной иерархической кластеризации. На рисунке 2.4 представлена схема многопоточного алгоритма кластеризации документов.

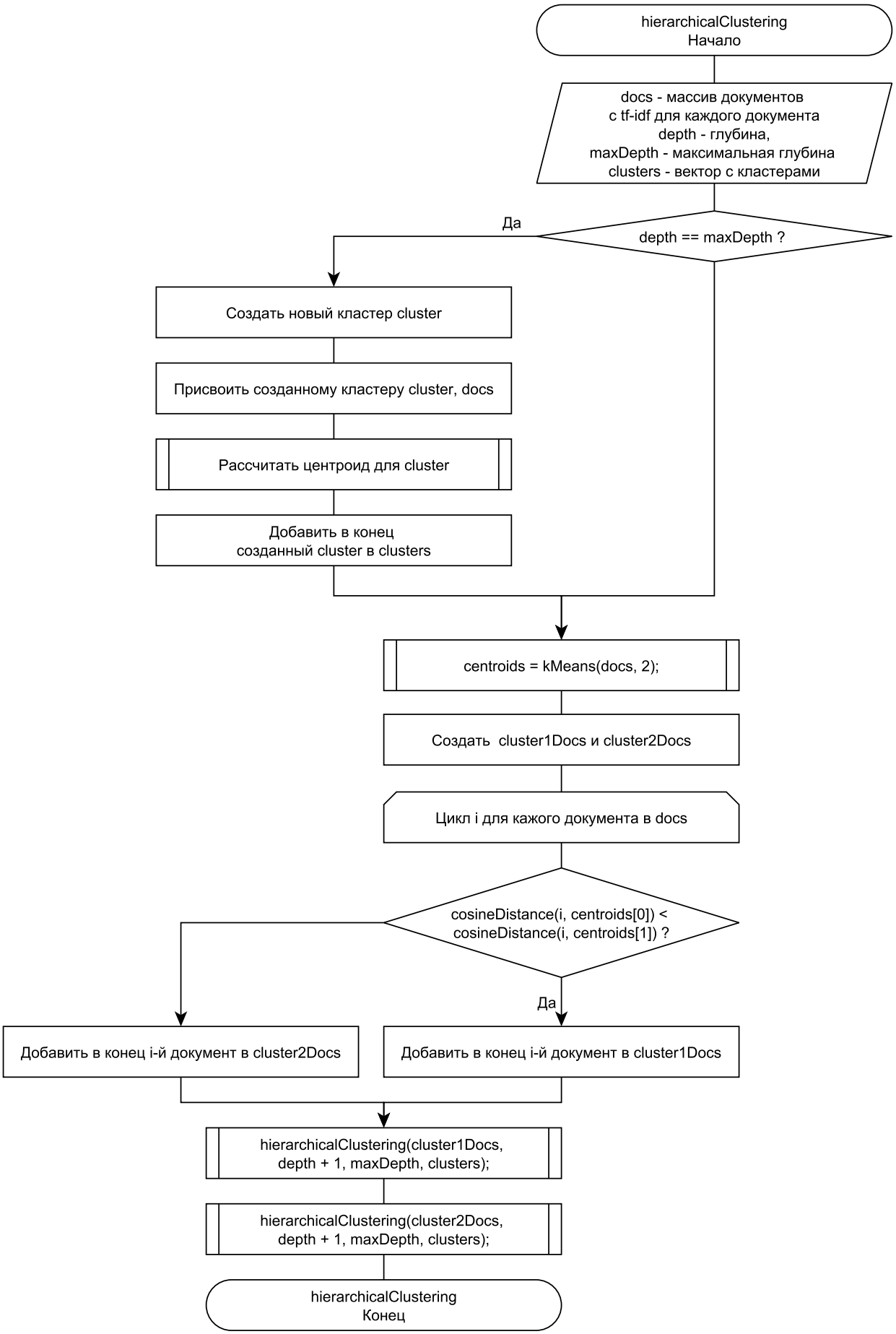

Рисунок 2.1 – Схема дивизимной иерархической кластеризации

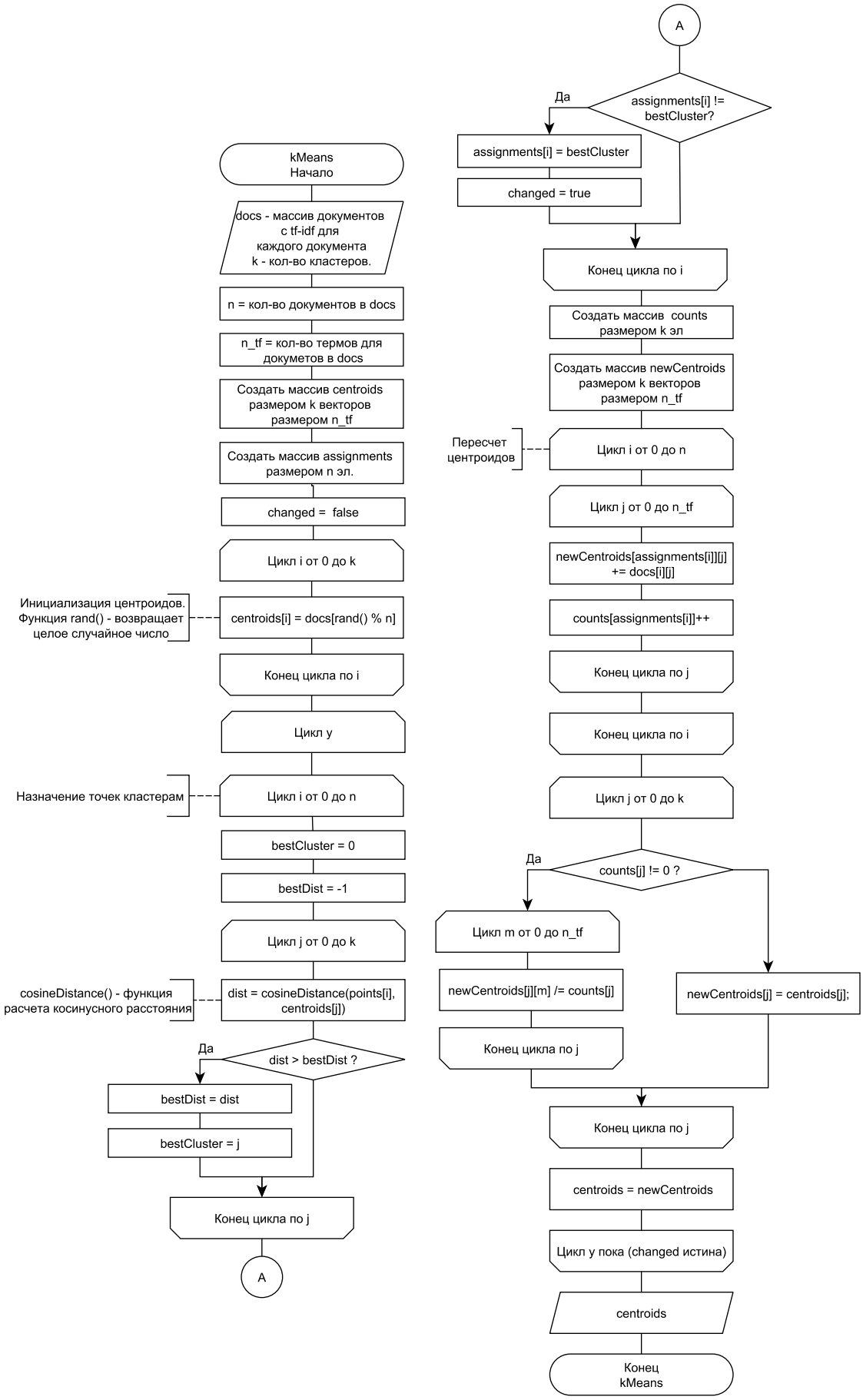

Рисунок 2.2 – Схема алгоритма k-средних

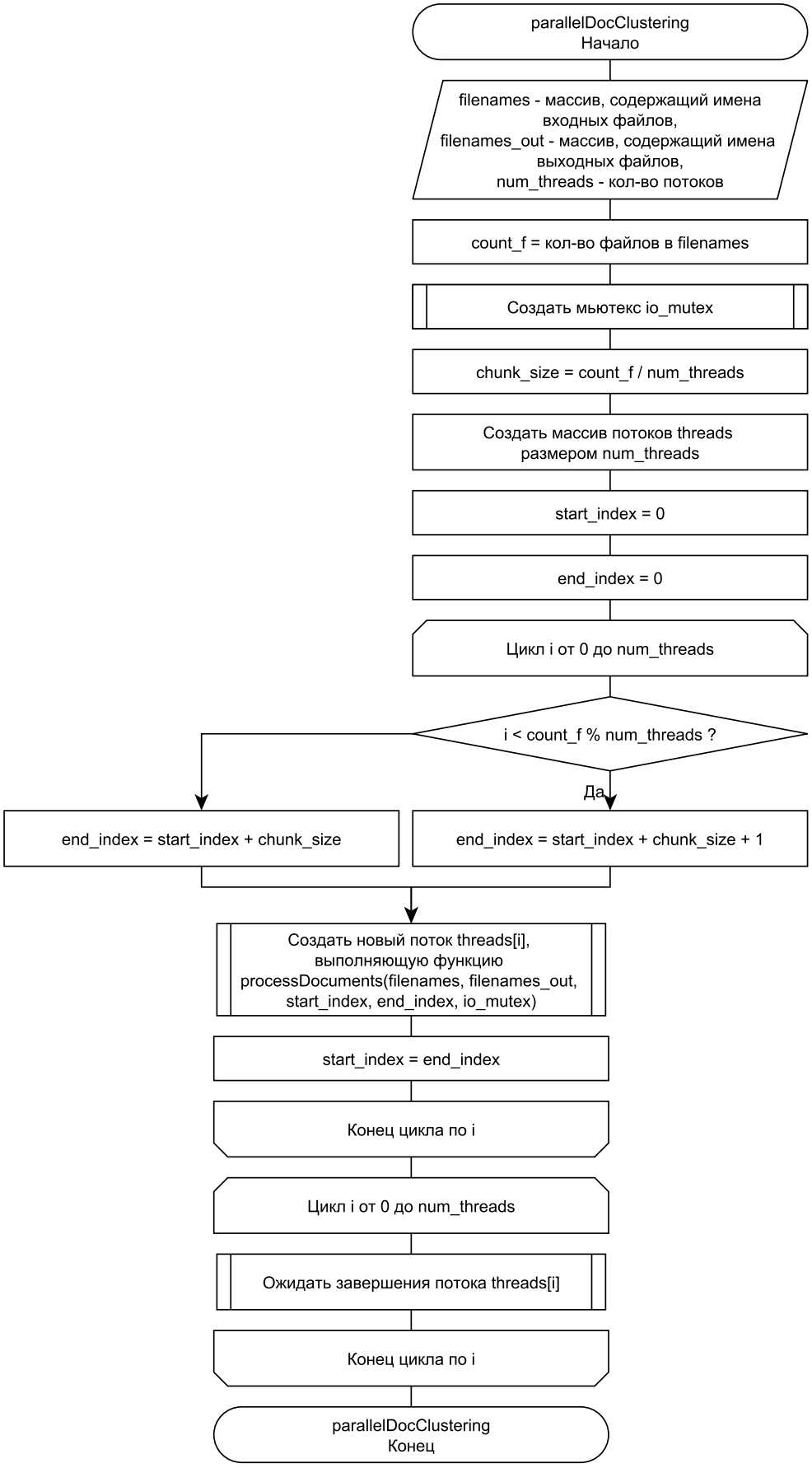

Рисунок 2.3 – Схема алгоритма создания потоков

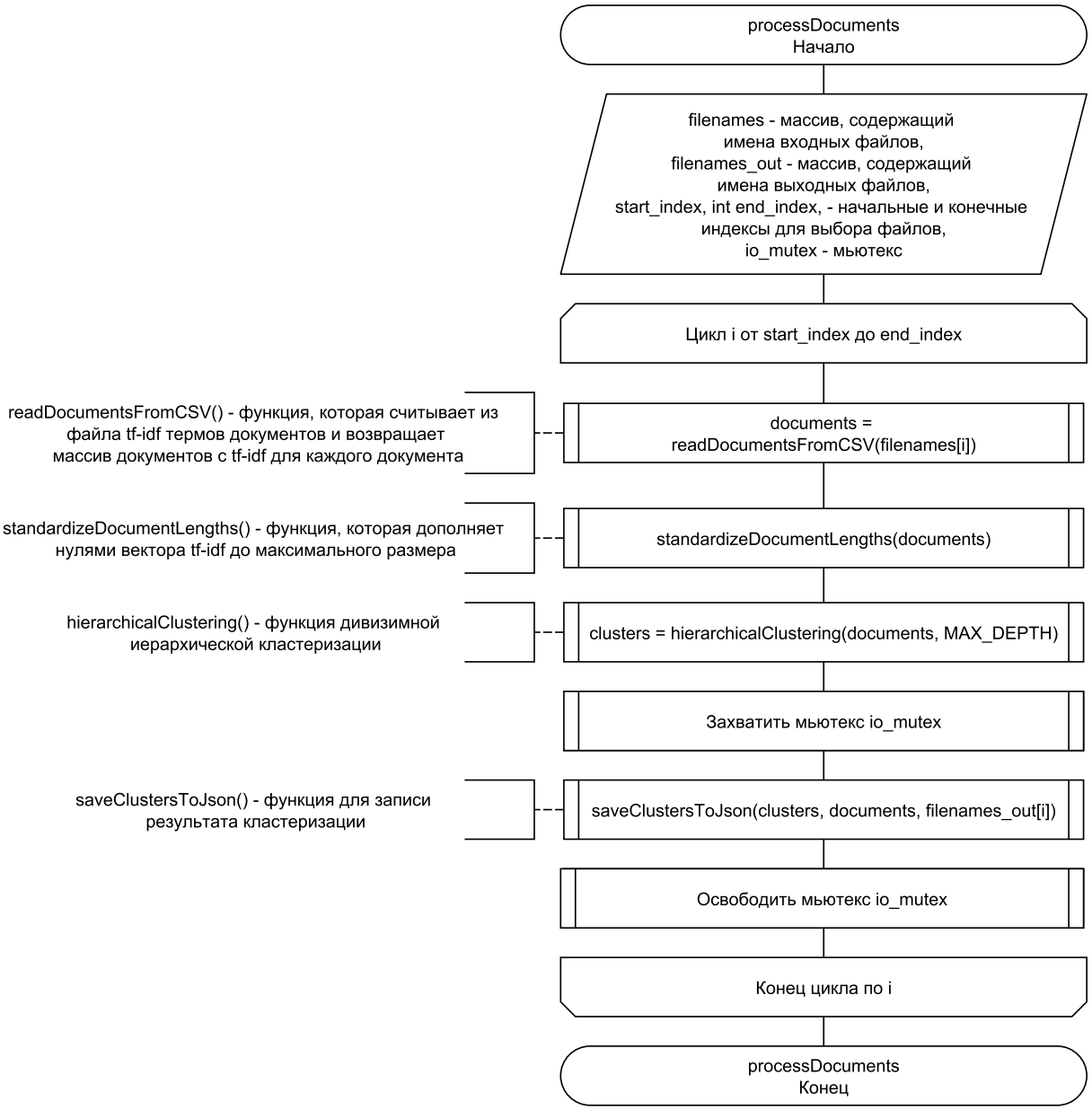

Рисунок 2.4 – Схема многопоточного алгоритма кластеризации документов

Вывод

В данном разделе были представлены требования к программному обеспечению, описание используемых типов данных и схемы реализуемых алгоритмов.

3 Технологический раздел

В данном разделе будут представлены средства реализации, сведения о модулях программы и листинги кода реализации алгоритмов.

3.1 Средства реализации

В качестве языка программирования для разработки программного обеспечения был выбран язык *C++* [6].

Данный выбор обусловлен следующим:

- язык позволяет реализовать все алгоритмы, выбранные в результате проектирования;
- язык поддерживает все типы и структуры данных, которые были выбраны в результате проектирования;
- язык позволяет работать с нативными потоками [7].

Время выполнения реализаций было замерено с помощью функции *clock* [8]. Для хранения термов использовалась структура данных *string* [9], в качестве массивов использовалась структура данных *vector* [10]. В качестве примитива синхронизации использовался *mutex* [11].

Для создания потоков и работы с ними был использован класс *thread* из стандартной библиотеки выбранного языка [7]. В листинге 3.1, приведен пример работы с описанным классом, каждый объект класса представляет собой поток операционной системы, что позволяет нескольким функциям выполняться параллельно [7].

Листинг 3.1 – Пример работы с классом thread

```
#include <iostream>
#include <thread>

void foo(int a){
    std::cout << a << '\n';
}

int main(){
    std::thread thread(foo, 10);
    thread.join();

    return 0;
}
```

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

- *main.cpp* — файл, который содержит точку входа в программу;
- *Cluster.cpp* и *Cluster.h* — модуль, который реализует класс *Cluster*;
- *kMeans.cpp* и *kMeans.h* — модуль, содержащий реализацию функции *kMeans*;
- *Klastering.cpp* и *Klastering.h* — модуль, содержащий реализации функций дивизимной иерархической кластеризации: *hierarchicalClustering* и многопоточная версия кластеризации*parallelDocClustering*;
- *io.cpp* и *io.h* — модуль, содержащий реализации функций для работы входными и выходными файлами;
- *Document.cpp* и *Document.h* — модуль, который реализует класс *Document*;
- *Term.cpp* и *Term.h* — модуль, который реализует класс *Term*;

3.3 Реализация алгоритмов

В листинге 3.2 приведена реализация дивизимной иерархической кластеризации. В листинге А.1 приведена реализация алгоритма k-средних. В

листинге 3.3 приведена реализация алгоритма создания потоков для алгоритма дивизимной иерархической кластеризации. В листинге 3.4 приведена реализация многопоточного алгоритма кластеризации документов.

Листинг 3.2 – Реализация дивизимной иерархической кластеризации

```
void hierarchicalClustering(const
   std::vector<std::vector<double>> &docs,
                    int depth,
                    int maxDepth,
                    std::vector<Cluster> &clusters) {
    if (depth == maxDepth) {
        Cluster cluster;
        cluster.docs = docs;
        cluster.calculateCentroid();
        clusters.push_back(cluster);
        return;
    }

    // Применяем k-средних для разделения на 2 кластера
    std::vector<std::vector<double>> centroids = kMeans(docs, 2);
    std::vector<std::vector<double>> cluster1Docs, cluster2Docs;

    for (const auto &doc: docs) {
        if (cosineDistance(doc, centroids[0]) <
           cosineDistance(doc, centroids[1])) {
            cluster1Docs.push_back(doc);
        }
        else {
            cluster2Docs.push_back(doc);
        }
    }

    // Рекурсивно разделяем каждый подкластер
    hierarchicalClustering(cluster1Docs, depth + 1, maxDepth,
       clusters);
    hierarchicalClustering(cluster2Docs, depth + 1, maxDepth,
       clusters);
}
```

Листинг 3.3 – Реализация алгоритма создания потоков

```
void parallelDocClustering(const std::vector<std::string>
    &filenames,
                           const std::vector<std::string>
                              &filenames_out,
                           int num_threads) {
    int count_f = filenames.size();
    std::mutex io_mutex;

    // Расчет размера подгруппы для каждого потока
    int chunk_size = count_f / num_threads;
    std::thread threads[num_threads]; // Массив потоков

    int start_index = 0;
    for (int i = 0; i < num_threads; ++i) {
        int end_index = start_index + chunk_size + (i < count_f
            % num_threads ? 1 : 0);

        // Создание потока для обработки подгруппы документов
        threads[i] = std::thread(processDocuments,
            std::ref(filenames), std::ref(filenames_out),
                                 start_index, end_index,
                                    std::ref(io_mutex));
        start_index = end_index;
    }

    // Дождаться завершения всех потоков
    for (int i = 0; i < num_threads; ++i) {
        threads[i].join();
    }
}
```

Листинг 3.4 – Реализация многопоточного алгоритма кластеризации документов

```
void processDocuments(const std::vector<std::string> &filenames,
                      const std::vector<std::string>
                         &filenames_out,
                      int start_index, int end_index,
                      std::mutex &io_mutex) {
    for (int i = start_index; i < end_index; ++i) {
        // Загрузка и обработка документа
        std::vector<Document> documents =
           readDocumentsFromCSV(filenames[i]);
        standardizeDocumentLengths(documents);

        // Выполнение иерархической кластеризации
        std::vector<Cluster> clusters =
           hierarchicalClustering(documents, MAX_DEPTH);

        // Синхронизированное сохранение результатов
        std::lock_guard<std::mutex> lock(io_mutex);
        saveClustersToJson(clusters, documents,
           filenames_out[i]);
    }
}
```

Вывод

В данном разделе были представлены средства реализации, сведения о модулях программы и листинги кода реализации алгоритмов.

4 Исследовательская часть

В данном разделе будет приведена демонстрация работы программы.

4.1 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного программного обеспечения. Пример входных файлов представлен на рисунке 4.2. Пример результата кластеризации в виде json файла, представлен на рисунке 4.3.

```
                Menu
1) Run the sequential version of document clustering.
2) Run the parallel version of document clustering.
3) Perform timing measurements of implemented algorithms.
0) Exit.

Select an option (0-3):3

+--------------------+----------------------------------------+----------------------------------------+
| Number of threads  | Sequential version (ms)                | Parallel version (ms)                  |
+--------------------+----------------------------------------+----------------------------------------+
| 1                  | 4771                                   | 4284                                   |
| 2                  | 4446                                   | 2656                                   |
| 3                  | 4399                                   | 2247                                   |
| 4                  | 4475                                   | 2128                                   |
| 5                  | 4340                                   | 2206                                   |
| 6                  | 4476                                   | 2324                                   |
| 7                  | 4450                                   | 2358                                   |
```

Рисунок 4.1 – Демонстрация работы программы

```
Document,Term,TF-IDF
news0,мобилизация,0.323565390388597
news0,нелогичный,0.0916643406262363
news0,заявить,0.054831234176486984
news0,депутат,0.06731898150728603
news0,верховный,0.07324778740136165
news0,рада,0.08089134759714936
news0,евгений,0.07324778740136165
news0,шевченко,0.09166434062623631
news0,интервью,0.06731898150728603
news0,украинский,0.1167581850158465
news0,политолог,0.09166434062623631
news0,вадим,0.09166434062623631
news0,закон,0.05837909250792325
news0,загнать,0.09166434062623631
news0,наш,0.027953534093788215
news0,этот,0.025484005145127456
news0,собака,0.09166434062623631
news0,стать,0.027953534093788215
news0,мочь,0.017998127726737652
```

Рисунок 4.2 – Пример входного файла

Рисунок 4.3 – Пример результата кластеризации

ЗАКЛЮЧЕНИЕ

Цель лабораторной работы достигнута, реализовано программного обеспечение, кластеризующее 60 текстов, алгоритмом иерархической кластеризации.

Для достижения поставленной цели были выполнены следующие задачи:

- описан алгоритм иерархической кластеризации;
- спроектировано программное обеспечение, реализующее алгоритм и его параллельную версию;
- разработано программное обеспечение, реализующее алгоритм и его параллельную версию.

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация алгоритма k-средних

```
std::vector<std::vector<double>> kMeans(const
   std::vector<std::vector<double>> &docs, int k) {
    int n = docs.size();
    std::vector<std::vector<double>> centroids(k,
       std::vector<double>(docs[0].size()));
    std::vector<int> assignments(n, 0);
    // Инициализация центроидов
    for (int i = 0; i < k; ++i) {
        centroids[i] = docs[rand() % n];
    }
    bool changed;
    do {
        changed = false;
        // Назначение точек кластерам
        for (int i = 0; i < n; ++i) {
            double bestDist = -1.0;
            int bestCluster = 0;
            for (int j = 0; j < k; ++j) {
                double dist = cosineDistance(docs[i],
                   centroids[j]);
                if (dist > bestDist) {
                    bestDist = dist;
                    bestCluster = j;
                }
            }
            if (assignments[i] != bestCluster) {
                assignments[i] = bestCluster;
                changed = true;
            }
        }
        // Обновление центроидов
        std::vector<int> counts(k, 0);
        std::vector<std::vector<double>> newCentroids(k,
           std::vector<double>(docs[0].size(), 0.0));
        for (int i = 0; i < n; ++i) {
            for (size_t j = 0; j < docs[i].size(); ++j) {
                newCentroids[assignments[i]][j] += docs[i][j];
```

```
34              }
35              counts[assignments[i]]++;
36          }
37          for (int j = 0; j < k; ++j) {
38              if (counts[j] != 0) {
39                  for (size_t m = 0; m < newCentroids[j].size();
                        ++m) {
40                      newCentroids[j][m] /= counts[j];
41                  }
42              }
43              else {
44                  newCentroids[j] = centroids[j];
45              }
46          }
47          centroids = newCentroids;
48      } while (changed);
49
50      return centroids;
51  }
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Grama A., Gupta A., Karypis G., Kumar V.* Introduction to Parallel Computing. — Addison-Wesley, 2003. — С. 1—5.
2. *Stoltzfus J.* Multithreading [Электронный ресурс]. — 2022. — Режим доступа: https://www.techopedia.com/definition/24297/multithreading-computer-architecture (дата обращения: 09.12.2023).
3. *Ричард Стивенс У., Стивен А. Раго.* UNIX. Профессиональное программирование. 3-е издание. — Питер, 2018. — С. 994.
4. *Большакова Е. И., Клышински Э. С., Ландэ Д. В., Носков А. А., Пескова О. В., Ягунова Е. В.* Автоматическая обработка текстов на естественном языке и компьютерная лингвистика: учеб. пособие. — МИЭМ, 2011. — С. 1—272.
5. *Котелина Н. О., Матвийчук Б. Р.* Кластеризация изображения методом K-средних // Вестник Сыктывкарского университета. — 2019. — Т. Выпуск 3 (32). — С. 102—106.
6. C++ reference [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/` (дата обращения: 20.12.2023).
7. Concurrency support library [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/thread` (дата обращения: 20.12.2023).
8. std::clock [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/chrono/c/clock` (дата обращения: 20.12.2023).
9. Strings library [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/string` (дата обращения: 20.12.2023).
10. std::vector [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/string` (дата обращения: 20.12.2023).
11. std::mutex [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/thread/mutex` (дата обращения: 20.12.2023).