МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Сортировки

Студент гр. 9303 _____ Муратов Р.А,

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Познакомиться с алгоритмами сортировки данных, реализовать один из них. Описать реализованный алгоритм, указать его теоретические и практические достоинства и недостатки.

Задание.

Вариант 16.

Сортировка массивов слиянием – естественное слияние.

Описание алгоритма.

Ключевые термины:

- Внешняя сортировка – это сортировка данных, которые расположены на внешних устройствах и не вмещающихся в оперативную память.
- Двухпутевое слияние – это сортировка, в которой данные распределяются на два вспомогательных файла.
- Двухфазная сортировка – это сортировка, в которой отдельно реализуется две фазы: распределение и слияние.
- Длина серии – это количество элементов в серии.
- Естественное слияние – это сортировка, при которой всегда сливаются две самые длинные из возможных серий.
- Несбалансированное слияние – это естественное слияние, у которого после фазы распределения количество серий во вспомогательных файлах отличается друг от друга более чем на единицу.
- Однофазная сортировка – это сортировка, в которой объединены фазы распределения и слияния в одну.
- Простое слияние – это одна из сортировок на основе слияния, в которой длина серий фиксируется на каждом шаге.
- Распределение – это процесс разделения упорядоченных серий на два и несколько вспомогательных файла.

- Сбалансированное слияние – это естественное слияние, у которого после фазы распределения количество серий во вспомогательных файлах отличается друг от друга не более чем на единицу.
- Серия (упорядоченный отрезок) – это последовательность элементов, которая упорядочена по ключу.
- Слияние – это процесс объединения двух (или более) упорядоченных серий в одну упорядоченную последовательность при помощи циклического выбора элементов доступных в данный момент.
- Фаза – это действия по однократной обработке всей последовательности элементов.

В случае простого слияния частичная упорядоченность сортируемых данных не дает никакого преимущества. Это объясняется тем, что на каждом проходе сливаются серии фиксированной длины. При естественном слиянии длина серий не ограничивается, а определяется количеством элементов в уже упорядоченных подпоследовательностях, выделяемых на каждом проходе.

Сортировка, при которой всегда сливаются две самые длинные из возможных последовательностей, является естественным слиянием. В данной сортировке объединяются серии максимальной длины.

Алгоритм сортировки естественным слиянием:

Шаг 1. Исходный файл f разбивается на два вспомогательных файла f1 и f2. Распределение происходит следующим образом: поочередно считываются записи a_i исходной последовательности (неупорядоченной) таким образом, что если значения ключей соседних записей удовлетворяют условию $f(a_i)<=f(a_{i+1})$, то они записываются в первый вспомогательный файл f1. Как только встречаются $f(a_i)>f(a_{i+1})$, то записи a_{i+1} копируются во второй вспомогательный файл f2. Процедура повторяется до тех пор, пока все записи исходной последовательности не будут распределены по файлам.

Шаг 2. Вспомогательные файлы f1 и f2 сливаются в файл f, при этом серии образуют упорядоченные последовательности.

Шаг 3. Полученный файл f вновь обрабатывается, как указано в шагах 1 и 2.

Шаг 4. Повторяя шаги, сливаем упорядоченные серии до тех пор, пока не будет упорядочен целиком весь файл.

Символ "|" обозначает признак конца серии.

Признаками конца сортировки естественным слиянием являются следующие условия:

- количество серий равно 1 (определяется на фазе слияния).
- при однофазной сортировке второй по счету вспомогательный файл после распределения серий остался пустым.

Естественное слияние, у которого после фазы распределения количество серий во вспомогательных файлах отличается друг от друга не более чем на единицу, называется сбалансированным слиянием, в противном случае – несбалансированное слияние.

Число чтений или перезаписей файлов при использовании метода естественного слияния будет не хуже, чем при применении метода простого слияния, а в среднем – даже лучше. Но в этом методе увеличивается число сравнений за счет тех, которые требуются для распознавания концов серий. Помимо этого, максимальный размер вспомогательных файлов может быть близок к размеру исходного файла, так как длина серий может быть произвольной.

Данная сортировка относится к внешним сортировкам, которые применяются к данным, которые хранятся во внешней памяти. Внешние сортировки применяются, если объем сортируемых данных превосходит допустимое место в ОЗУ.

Внешние сортировки, по сравнению с внутренними, характеризуются проигрышем по времени за счет обращения к внешним носителям.

К наиболее известным алгоритмам внешних сортировок относятся: сортировки слиянием (простое слияние и естественное слияние); улучшенные сортировки (многофазная сортировка и каскадная сортировка).

Увеличение количества вспомогательных аппаратурных элементов не позволяет ускорить сортировку. Соответствующий график приведен на рис. 1.

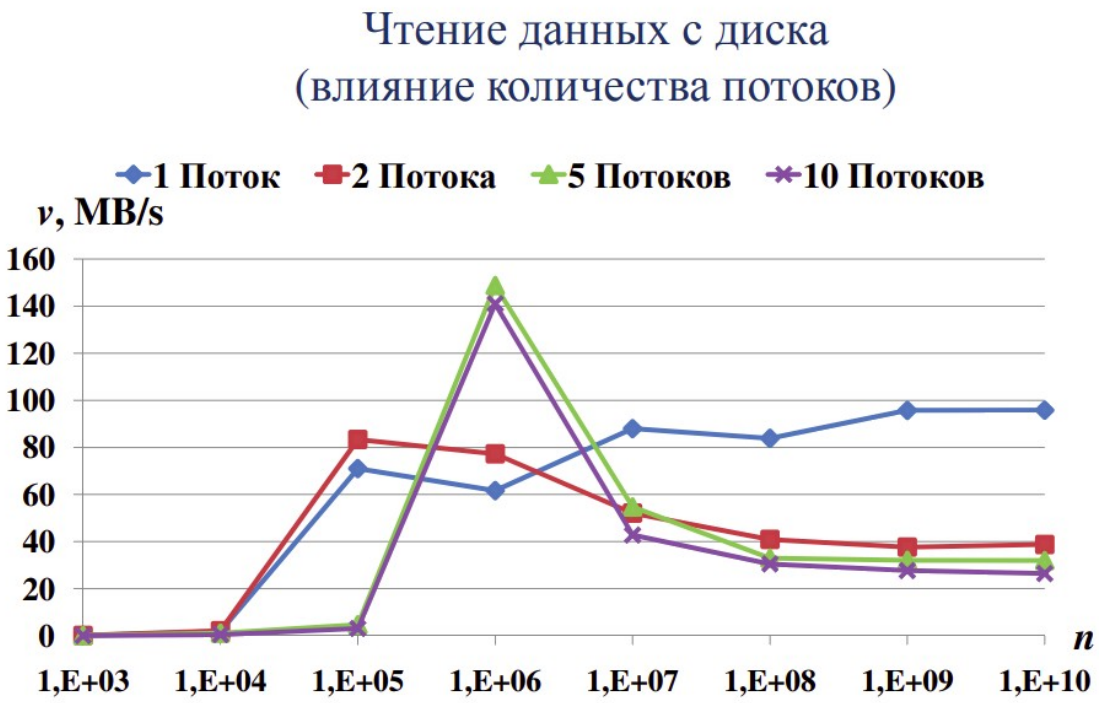

Рисунок 1 – График зависимости скорости чтения от числа элементов

Сложность данного алгоритма равна $O(n^2)$, это говорит о том, что алгоритм достаточно медленный. Также к недостаткам алгоритма можно отнести затраты на дополнительную память.

Выполнение работы.

`bool isSeriesEnd(std::fstream& file)` – данная функция проверяет, достигнут ли конец серии (то есть символ "|") во вспомогательном файле.

`void printFile(const std::string& fileName, std::ostream& os)` – данная функция распечатывает в выходной поток `os` содержимое файла с именем `fileName`.

`void naturalMergingSort(const std::string fileName, std::ostream& os)` – основная шаблонная функция, которая сортирует с помощью алгоритма естественной сортировки слиянием данные, содержащиеся в файле с именем `fileName`, и выводит лог процесса сортировки в выходной поток `os`.

В переменной `iterCounter` хранится количество итерации алгоритма (это нужно для более содержательного вывода в лог), в переменных `s1` и `s2` хранится текущее количество серий в первом и втором файлах соответственно, переменная `fileSelect` задает номер вспомогательного файла, в который сейчас производится запись, переменные `curElem` и `curElem2` выполняют разные роли в разных фазах: на фазе «распределение» `curElem` и `curElem2` – предыдущий и текущий элементы соответственно, на фазе «слияние» `curElem` и `curElem2` – элементы из первого и второго вспомогательных файлов соответственно. Булевы переменные `seriesEnd1` и `seriesEnd2` показывают, закончилась ли серия во вспомогательном файле или нет.

Далее в цикле сначала обнуляется количество серий в каждом из файлов, для записи (изначально) выбирается первый вспомогательный файл и открываются соответствующие файлы.

После этого последовательно считываются два элемента из основного файла (случай, когда файл пуст, в файле лишь один символ также учитываются, `s1` и `s2` инкрементируются именно для этого). Далее в цикле, если текущий элемент меньше предыдущего, то файл для записи меняется на противоположный, печатается символ конца серии и инкрементируется соответствующая переменная `s1` или `s2`, после этого в соответствующий

файл печатаются элементы из основного файла. После окончания цикла печатается символ конца последовательности в соответствующем файле. Далее используются два вложенных сравнения для корректной обработки случая пустого файла и файла с одним символом. На данный момент завершена фаза «распределение». Файлы закрываются и их одержимое печатается в заданный пользователем выходной поток.

Затем те же файлы открываются, но уже в других режимах (`in` → `out`, `out` → `in`). Считываются элементы из первого и второго вспомогательных файлов. Далее в цикле проходим по «парным» сериям и печатаем упорядоченно их содержимое в основной файл (если в одной из «парных» серии больше элементов, чем в другой, то в другом цикле эти «лишние» элементы печатаются в основной файл). Если в одном из вспомогательном файлов больше серии (сортировка сбалансирована, поэтому разница количества серий может быть равна 0 или 1), то печатаем эту серию в основной файл. Файлы закрываются и завершается фаза «слияние». Печатается содержимое основного файла, то есть итоговый результат, и удаляются вспомогательные файлы, созданные для выполнения сортировки.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Познакомился с алгоритмами сортировки данных, реализовал один из них: естественная сортировка слиянием. Описал реализованный алгоритм, указал его теоретические и практические достоинства и недостатки.

Разработана программа, выполняющая считывание с клавиатуры имя файла, в котором содержаться данные для сортировки, и выполняющая сортировку этих данных.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <cstdio>

void printFile(const std::string& fileName, std::ostream& os) {
  std::fstream file(fileName, std::ios_base::in);
  if (!file.is_open()) {
      os << "Can't open the file with name: " << fileName <<
std::endl;
    return;
  }
  char c;
  c = file.get();
  while (!file.eof()) {
    if (c != '\n')
      os << c;
    c = file.get();
  }
  os << std::endl;
  file.close();
}

bool isSeriesEnd(std::fstream& file) {
  auto pos = file.tellg();
  char x;
  file >> x;
  if (x == '|') {
    file.seekg(1, std::ios_base::cur);
    return true;
  } else {
    file.seekg(pos);
    return false;
  }
}

template<class T>
void     naturalMergingSort(const     std::string&     fileName,
std::ostream& os) {
  size_t iterCounter = 1;
  size_t s1 = 0;
  size_t s2 = 0;

  size_t fileSelect = 1;

  T curElem;
  T curElem2;

  std::fstream mergeFile;
  std::fstream supportFile1;
```

```
        std::fstream supportFile2;

        do {
          s1 = s2 = 0;
          fileSelect = 1;

          mergeFile.open(fileName, std::ios_base::in);
          if (!mergeFile.is_open()) {
                std::cerr << "Error: Can't open a merge file!" <<
std::endl;
            return;
          }

           supportFile1.open("/home/mur/Programing/QtProjects/AaDS/Sorti
ng/series_holder_1.txt", std::ios_base::out);
           supportFile2.open("/home/mur/Programing/QtProjects/AaDS/Sorti
ng/series_holder_2.txt", std::ios_base::out);

          mergeFile >> curElem;
          if (!mergeFile.eof()) {
            supportFile1 << curElem << " ";
            ++s1;
          }
          if (!mergeFile.eof()) {
            mergeFile >> curElem2;
            ++s2;
          }

          while (!mergeFile.eof()) {
            if (curElem2 < curElem) {
              switch (fileSelect) {
                case 1:
                  fileSelect = 2;
                  supportFile1 << "| ";
                  ++s1;
                  break;
                case 2:
                  fileSelect = 1;
                  supportFile2 << "| ";
                  ++s2;
                  break;
              }
            }
            if (fileSelect == 1) {
              supportFile1 << curElem2 << " ";
            } else {
              supportFile2 << curElem2 << " ";
            }

            curElem = curElem2;
            mergeFile >> curElem2;
          }

          if (s1 > 0 && fileSelect == 1) {
            supportFile1 << "|";
            ++s1;
```

```
          }
          if (s2 > 0 && fileSelect == 2) {
            supportFile2 << "|";
            ++s2;
          }

          if (s1 != 0) {
            --s1;
            if (s2 != 0)
              --s2;
          }

          mergeFile.close();
          supportFile1.close();
          supportFile2.close();

          os << "Intermediate state at iteration #" << iterCounter++ <<
":\n";
          os << "mergeFile: ";
          printFile(fileName, os);
          os << "supportFile1: ";
           printFile("/home/mur/Programing/QtProjects/AaDS/Sorting/serie
s_holder_1.txt", os);
          os << "supportFile2: ";
           printFile("/home/mur/Programing/QtProjects/AaDS/Sorting/serie
s_holder_2.txt", os);
          os << "s1 = " << s1 << " " << "s2 = " << s2 << std::endl;

          mergeFile.open(fileName, std::ios_base::out);
          if (!mergeFile.is_open()) {
                std::cerr << "Error: Can't open a merge file!" <<
std::endl;
            return;
          }

           supportFile1.open("/home/mur/Programing/QtProjects/AaDS/Sorti
ng/series_holder_1.txt", std::ios_base::in);
          if (!supportFile1.is_open()) {
              std::cerr << "Error: Can't open first support file!" <<
std::endl;
            return;
          }

           supportFile2.open("/home/mur/Programing/QtProjects/AaDS/Sorti
ng/series_holder_2.txt", std::ios_base::in);
          if (!supportFile1.is_open()) {
              std::cerr << "Error: Can't open first support file!" <<
std::endl;
            return;
          }

          supportFile1 >> curElem;
          supportFile2 >> curElem2;

          bool seriesEnd1;
          bool seriesEnd2;
```

```
          while (!supportFile1.eof() && !supportFile2.eof()) {
            seriesEnd1 = seriesEnd2 = false;
            while (!seriesEnd1 && !seriesEnd2) {
              if (curElem <= curElem2) {
                mergeFile << curElem << " ";
                seriesEnd1 = isSeriesEnd(supportFile1);
                supportFile1 >> curElem;
              } else {
                mergeFile << curElem2 << " ";
                seriesEnd2 = isSeriesEnd(supportFile2);
                supportFile2 >> curElem2;
              }
            }
            while (!seriesEnd1) {
              mergeFile << curElem << " ";
              seriesEnd1 = isSeriesEnd(supportFile1);
              supportFile1 >> curElem;
            }
            while (!seriesEnd2) {
              mergeFile << curElem2 << " ";
              seriesEnd2 = isSeriesEnd(supportFile2);
              supportFile2 >> curElem2;
            }
          }

          seriesEnd1 = seriesEnd2 = false;
          while (!supportFile1.eof() && !seriesEnd1) {
            mergeFile << curElem << " ";
            seriesEnd1 = isSeriesEnd(supportFile1);
            supportFile1 >> curElem;
          }
          while (!supportFile2.eof() && !seriesEnd2) {
            mergeFile << curElem << " ";
            seriesEnd2 = isSeriesEnd(supportFile2);
            supportFile2 >> curElem;
          }

          mergeFile.close();
          supportFile1.close();
          supportFile2.close();

        } while (s1 > 1 && s2 > 0);

        os << "Final result: ";
        printFile(fileName, os);

         std::remove("/home/mur/Programing/QtProjects/AaDS/Sorting/series_holder_1.txt");
         std::remove("/home/mur/Programing/QtProjects/AaDS/Sorting/series_holder_2.txt");
      }

      int main() {
        std::ofstream resFile("./result.txt");
```

```
        std::string fileName;
          std::cout << "Please, enter the input file name: " <<
std::endl;
        std::cin >> fileName;

        naturalMergingSort<int>(fileName, resFile);

        std::cout << "If you entered a valid file's name, result saves
in file result.txt" << std::endl;
        return 0;
      }
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	10 9 8 7 6 5 4 3 2 1	Intermediate state at iteration #1: mergeFile: 10 9 8 7 6 5 4 3 2 1 supportFile1: 10 | 8 | 6 | 4 | 2 | supportFile2: 9 | 7 | 5 | 3 | 1 | s1 = 5 s2 = 5 Intermediate state at iteration #2: mergeFile: 9 10 7 8 5 6 3 4 1 2 supportFile1: 9 10 | 5 6 | 1 2 | supportFile2: 7 8 | 3 4 | s1 = 3 s2 = 2 Intermediate state at iteration #3: mergeFile: 7 8 9 10 3 4 5 6 1 2 supportFile1: 7 8 9 10 | 1 2 | supportFile2: 3 4 5 6 | s1 = 2 s2 = 1 Intermediate state at iteration #4: mergeFile: 3 4 5 6 7 8 9 10 1 2 supportFile1: 3 4 5 6 7 8 9 10 | supportFile2: 1 2 | s1 = 1 s2 = 1 Final result: 1 2 3 4 5 6 7 8 9 10	Изначальная последовательность неотсортирована.
2.	10	Intermediate state at iteration #1: mergeFile: 10 supportFile1: 10 | supportFile2: s1 = 1 s2 = 0 Final result: 10	Один элемент.
3.	1 2 3 4 5 6 7 8 9 10	Intermediate state at iteration #1:	Исходная

		mergeFile: 1 2 3 4 5 6 7 8 9 10 supportFile1: 1 2 3 4 5 6 7 8 9 10 | supportFile2: s1 = 1 s2 = 0 Final result: 1 2 3 4 5 6 7 8 9 10	последовательность отсортирована.
4.		Intermediate state at iteration #1: mergeFile: supportFile1: supportFile2: s1 = 0 s2 = 0 Final result:	Пустой файл.
5.	7 -10 32 4 0 123 -321 32 14 23	Intermediate state at iteration #1: mergeFile: 7 -10 32 4 0 123 -321 32 14 23 supportFile1: 7 | 4 | -321 32 | supportFile2: -10 32 | 0 123 | 14 23 | s1 = 3 s2 = 3 Intermediate state at iteration #2: mergeFile: -10 7 32 0 4 123 -321 14 23 32 supportFile1: -10 7 32 | -321 14 23 32 | supportFile2: 0 4 123 | s1 = 2 s2 = 1 Intermediate state at iteration #3: mergeFile: -10 0 4 7 32 123 -321 14 23 32 supportFile1: -10 0 4 7 32 123 | supportFile2: -321 14 23 32 | s1 = 1 s2 = 1	Произвольная последовательность.

		Final result: -321 -10 0 4 7 14 23 32 32 123	