Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПЕРМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра “Информационные технологии и автоматизированные системы”

**ОТЧЕТ**

**по лабораторной работе №16**

Дисциплина: “Основы теории алгоритмов и структуры данных”

Тема: Методы внешней сортировки: метод прямого слияния, сбалансированного слияния, метод многофазной сортировки

Вариант №17

Выполнил:

Студент группы ИВТ-20-2б

Сабуров Павел Алексеевич

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

**Пермь, 2021**

**Цель работы**

Научиться применять внешнюю сортировку многофазным слияниями

**Постановка задачи**

Отсортировать вектор целых чисел по неубыванию

**Анализ задачи**

**(1)**

**1.1** Создать вектор целых чисел с помощью функции CreateVector();

**1.2** Вызов функции многофазной сортировки PolyphaseMergeSort(vec);

**1.3** Разделить исходный вектор на равные серии с помощью функции

SplitToEqualSeries(vec);

**1.4** Задать количество серий в первом и втором файлах с помощью чисел Фибоначчи:

|  |
| --- |
| Fibbonachi(series.size(), size1, size2); |

**1.5** Объявить три файла как “вектора векторов”, где внешний вектор - файл, а вложенные в него - серии:

|  |
| --- |
| vector<vector<int>> F1, F2, F3; |

**1.6** Заполнение двух файлов непустыми сериями

**1.7** Сумма size1 и size2 после функции распределения серий по файлам с помощью чисел Фибоначчи может превышать изначальное количество серий исходной последовательности. Если это так, то в один из двух файлов записывается столько пустых серий, сколько разница между (size1 + size2) и series.size()

**1.8** Внутренняя сортировка в каждой серии файлов с помощью сортировки естественным слиянием

**1.9** Слияние двух непустых файлов в пустой третий

**1.10** Удаление пустых серий из отсортированной последовательности

**(2)**

**2.1** int main():

vector<int> vec - вектор с заданной последовательностью для подстановки в метод многофазной сортировки

**2.2** PolyphaseMergeSort():

vector<vector<int>> series - вектор векторов, где внешний вектор - файл, внутренние - серии в этом файле

int size1 - количество серий в первом файле

int size2 - количество серий во втором файле

vector<vector<int>> F1, F2, F3 - файлы, один из которых всегда должен быть пустым, чтобы сливать серии в него из двух других

vector<int> serie - пустая серия, если нужно

int countEmptyFiles - переменная для контроля окончания сортировки (2 пустых файла)

int indOfEmptyFile - переменная для определения пустого файла, куда сливаются два остальных

**(3)**

**3.1** Исходная последовательность для сортировки представлена в виде вектора целых чисел

**3.2** Файлы представлены как “векторы векторов”, где внешний вектор - файл, внутренние векторы - серии в этом файле

**(4)**

**4.1** Ввод последовательности в вектор производится следующим образом:

|  |
| --- |
| vector<int> res; for (int i = 0; i < size; i++) {  res.push\_back(-100 + rand() % 200); } |

**4.2** Вывод файла, то есть “вектора векторов”:

Проход по каждой серии результирующего файла и вывод каждого элемента соответствующей серии

|  |
| --- |
| for (int i = 0; i < series.size(); i++)  {  cout << "(";  for (int j = 0; j < series[i].size(); j++)  {  cout << series[i][j] << " ";  }  cout << ")";  } |

**(5)** Поставленные задачи стоит решать следующим образом:

**5.1** Создать вектор целых чисел с помощью функции CreateVector():

Задается размер последовательности более двух элементов, после чего объявляется вектор, в который размещается последовательность в цикле:

|  |
| --- |
| vector<int> CreateVector(int size = 10) {  if (size < 2) size = 10;  srand(time(0));  vector<int> res;  for (int i = 0; i < size; i++)  {  res.push\_back(-100 + rand() % 200);  }  return res; } |

**5.2** Разделить исходный вектор на равные серии с помощью функции

SplitToEqualSeries(vec):

Сначала ищется минимальное кратное размеру последовательности число, которое будет определять размер одинаковых серий из исходной последовательности:

|  |
| --- |
| int numberOfElements = 2; while (vec.size() % numberOfElements != 0 && numberOfElements < 1000) numberOfElements++; |

Далее объявляется “вектор векторов”, внутренние векторы которого заполняются элементами вектора исходной последовательности, а внешний - заполненными внутренними:

|  |
| --- |
| vector<vector<int>> res;  for (int i = 0; i < vec.size() / numberOfElements; i++)  {  vector<int> newSerie;  for (int j = 0; j < numberOfElements; j++)  {  newSerie.push\_back(vec[i \* numberOfElements + j]);  }  res.push\_back(newSerie);  }   return res; |

**5.3** Задать количество серий в первом и втором файлах с помощью чисел Фибоначчи:

|  |
| --- |
| Fibbonachi(series.size(), size1, size2): |

Распределение будет продолжаться, пока size1 + size2 не станет равным или больше series.size(). Сначала задается начальное распределение серий по двум файлам: 0 и 1 соответственно. Далее в цикле количество серий в файлах будет расти в последовательности чисел Фибоначчи: после 0 и 1 станет 1 и 1, потом 1 и 2, после 2 и 3 и так далее:

|  |
| --- |
| size1 = 0;  size2 = 1;  while (size1 + size2 < numberOfSeries)  {  cout << "\n" << size1 << " " << size2 << " " << numberOfSeries << endl;  int t = size2;  size2 = size2 + size1;  size1 = t;  } |

**5.4** Заполнение двух файлов непустыми сериями:

|  |
| --- |
| for (int i = 0; i < size1; i++)  {  F1.push\_back(series[i]);  }  for (int i = size1; i < series.size(); i++)  {  F2.push\_back(series[i]);  } |

**5.5** Сумма size1 и size2 после функции распределения серий по файлам с помощью чисел Фибоначчи может превышать изначальное количество серий исходной последовательности. Если это так, то в один из двух файлов записывается столько пустых серий, сколько разница между (size1 + size2) и series.size():

|  |
| --- |
| for (int i = series.size(); i < size1 + size2; i++)  {  vector<int> serie;  for (int j = 0; j < F2[0].size(); j++)  serie.push\_back(INT32\_MAX);  F2.push\_back(serie);  } |

**5.6** Внутренняя сортировка в каждой серии файлов с помощью сортировки естественным слиянием:

|  |
| --- |
| for (int i = 0; i < F1.size(); i++)  {  F1[i] = NaturalMergeSort(F1[i]);  }  for (int i = 0; i < F2.size(); i++)  {  F2[i] = NaturalMergeSort(F2[i]);  } |

**5.7** Слияние двух непустых файлов в пустой третий:

2 файла будут сливаться в пустой третий, который определяется пустым простой проверкой на пустоту файла:

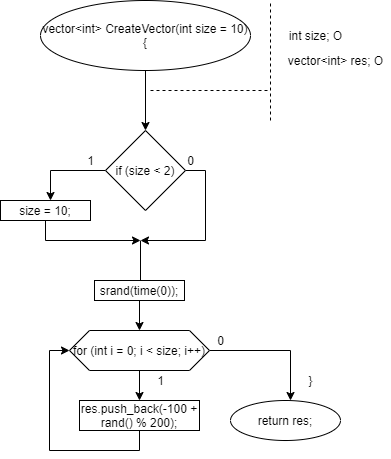
|  |
| --- |
| switch (indOfEmptyFile) {  case 1: F1 = MergeFiles(F2, F3); break;  case 2: F2 = MergeFiles(F1, F3); break;  case 3: F3 = MergeFiles(F1, F2); break; } |

**5.8** Удаление пустых серий из отсортированной последовательности:

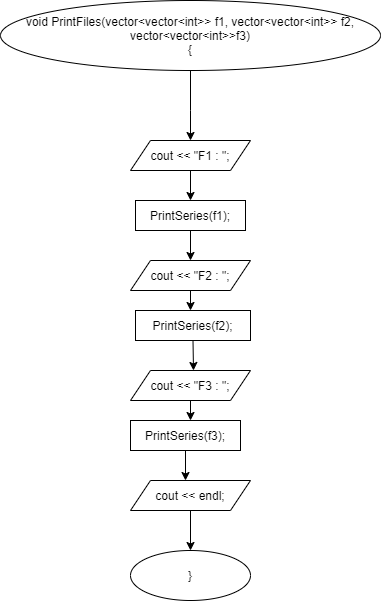
|  |
| --- |
| vector<int> DeleteEmptyElements(vector<int> vec) {  for (int i = vec.size() - 1; i >= 0; i--)  if (vec[i] == INT32\_MAX)  vec.erase(vec.begin() + i);  return vec; } |

**Блок-схемы**

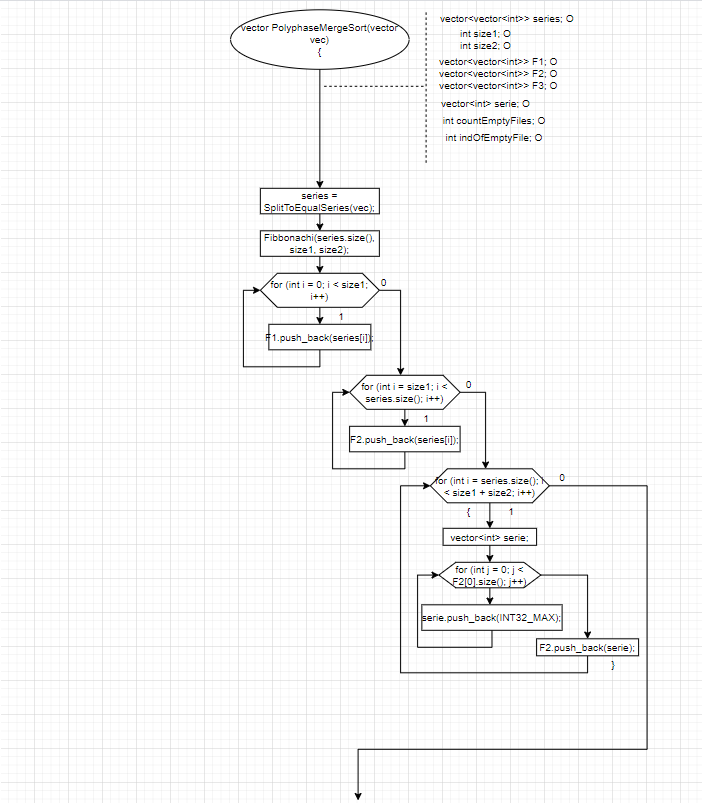
CreateVector():



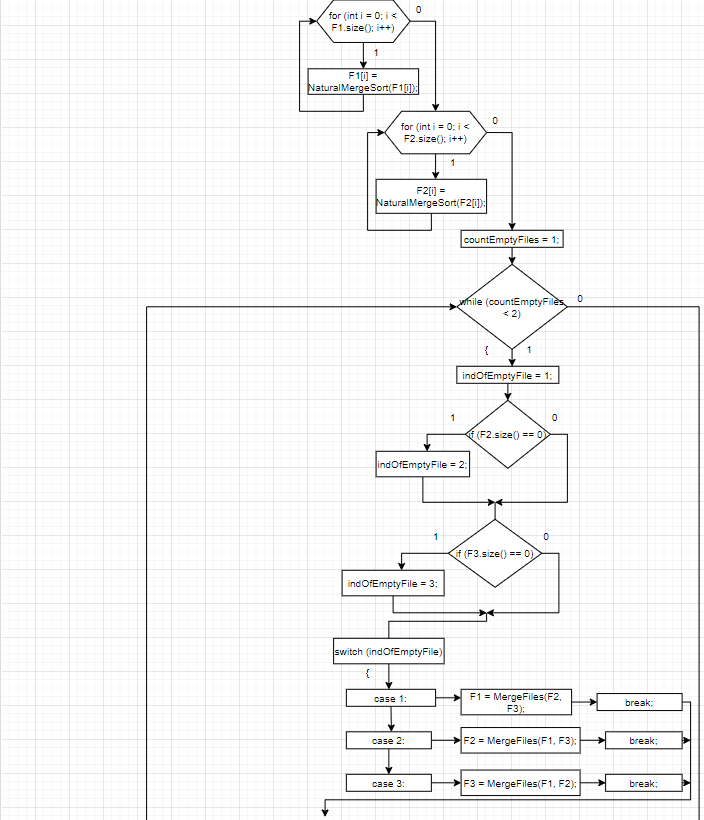
PrintVector():



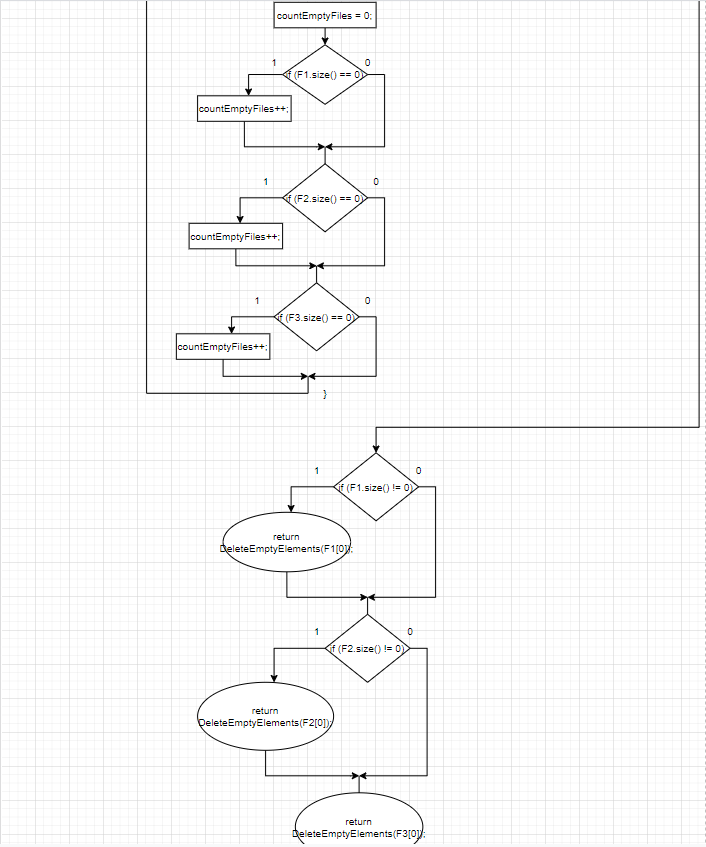
1) PolyphaseMergeSort():



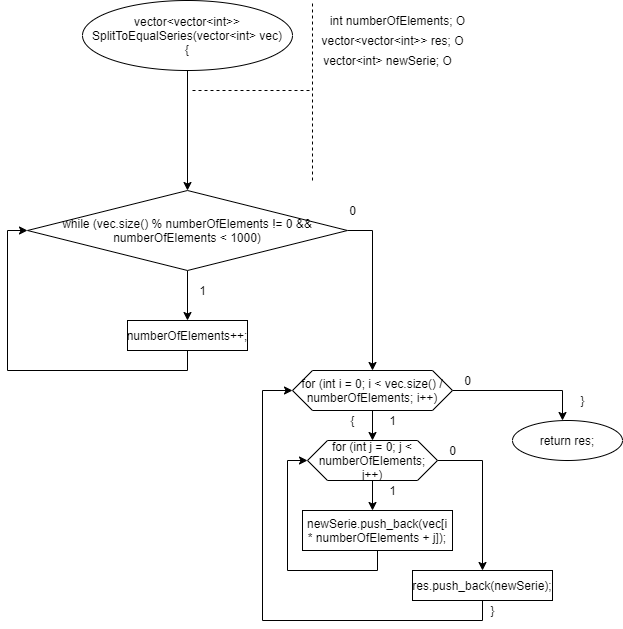
2) PolyphaseMergeSort():



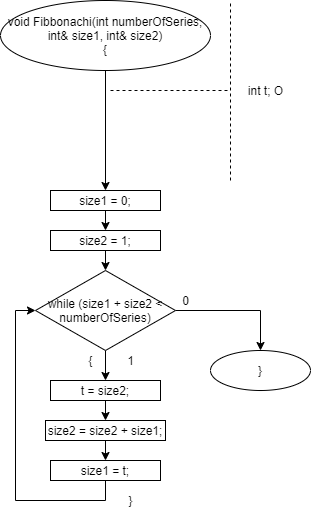
3) PolyphaseMergeSort():



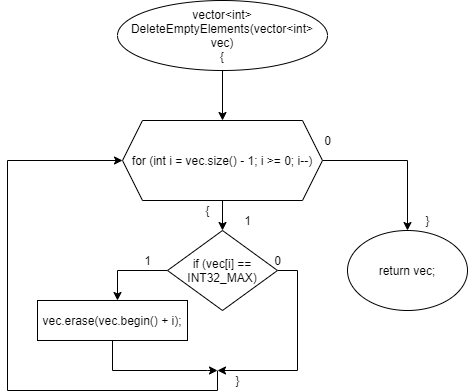
SplitToEqualSeries():



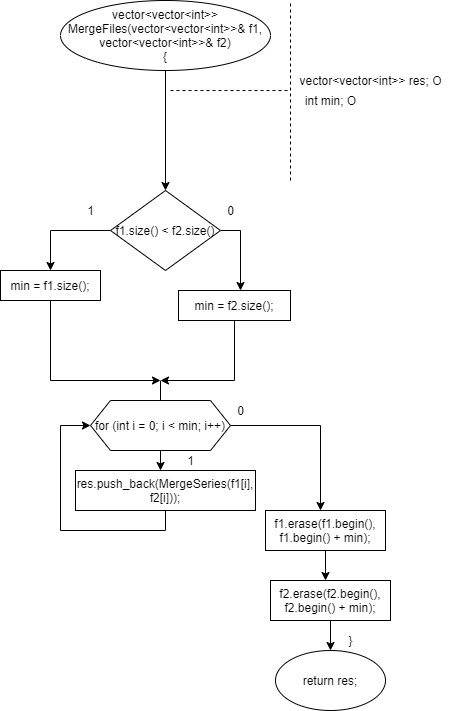
Fibbonachi():



DeleteEmptyElements():



MergeFiles():



**Код на C++**

|  |
| --- |
| vector<int> PolyphaseMergeSort(vector<int> vec) {  vector<vector<int>> series = SplitToEqualSeries(vec);  int size1, size2;  Fibbonachi(series.size(), size1, size2);  vector<vector<int>> F1, F2, F3;   cout << "Разделение на равные серии" << endl;  PrintSeries(series);   for (int i = 0; i < size1; i++)  {  F1.push\_back(series[i]);  }  for (int i = size1; i < series.size(); i++)  {  F2.push\_back(series[i]);  }  for (int i = series.size(); i < size1 + size2; i++)  {  vector<int> serie;  for (int j = 0; j < F2[0].size(); j++)  serie.push\_back(INT32\_MAX);  F2.push\_back(serie);  }   cout << "Разделение серий на файлы F1 и F2 (" << size1 << " " << size2 << ")" << endl;  PrintFiles(F1, F2, F3);   cout << "Сортировка элементов внутри файлов" << endl;  for (int i = 0; i < F1.size(); i++)  {  F1[i] = NaturalMergeSort(F1[i]);  }  for (int i = 0; i < F2.size(); i++)  {  F2[i] = NaturalMergeSort(F2[i]);  }  PrintFiles(F1, F2, F3);   cout << "Слияние файлов" << endl;  int countEmptyFiles = 1;   while (countEmptyFiles < 2)  {  int indOfEmptyFile = 1;  if (F2.size() == 0) indOfEmptyFile = 2;  if (F3.size() == 0) indOfEmptyFile = 3;   switch (indOfEmptyFile)  {  case 1: F1 = MergeFiles(F2, F3); break;  case 2: F2 = MergeFiles(F1, F3); break;  case 3: F3 = MergeFiles(F1, F2); break;  }   countEmptyFiles = 0;  if (F1.size() == 0) countEmptyFiles++;  if (F2.size() == 0) countEmptyFiles++;  if (F3.size() == 0) countEmptyFiles++;   PrintFiles(F1, F2, F3);  }   if (F1.size() != 0) return DeleteEmptyElements(F1[0]);  if (F2.size() != 0) return DeleteEmptyElements(F2[0]);  return DeleteEmptyElements(F3[0]); } |

**Результаты выполнения работы**

