|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«*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 \_\_\_\_\_\_\_\_\_КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по лабораторной работе № 1**

**Дисциплина:** Технология разработки программных систем

**Название лабораторной работы:** Выбор структур и методов обработки данных

Студент гр. ИУ6-45Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_**И.А.Дулина**\_\_\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_**Е.К.Пугачев\_\_\_\_\_\_

(Подпись, дата) (И.О. Фамилия)

Москва, 2024

**Содержание**

[1. Цель лабораторной работы 4](#_Toc162449900)

[2. Описание задания 4](#_Toc162449901)

[2.1. Задания 4](#_Toc162449902)

[2.2. Основные требования 6](#_Toc162449903)

[2.3. Задача 6](#_Toc162449904)

[3. Основной вариант 7](#_Toc162449905)

[3.1. Структура данных 7](#_Toc162449906)

[3.1.1. Реализация структуры на языке C++ 7](#_Toc162449907)

[3.1.2. Расчёт памяти, занимаемой массивом 8](#_Toc162449908)

[3.1.3. Оценка времени доступа к i-ому элементу 8](#_Toc162449909)

[3.1.4. Оценка времени удаления i-ого элемента 9](#_Toc162449910)

[3.2. Метод поиска 9](#_Toc162449911)

[3.2.1. Реализация метода поиска 9](#_Toc162449912)

[3.2.2. Среднее количество сравнений 10](#_Toc162449913)

[3.2.3. Оценка времени поиска 10](#_Toc162449914)

[3.3. Метод упорядочивания 10](#_Toc162449915)

[3.3.1. Реализация метода упорядочивания 10](#_Toc162449916)

[3.3.2. Среднее количество сравнений 11](#_Toc162449917)

[3.3.3. Оценка времени упорядочивания 11](#_Toc162449918)

[3.4. Метод корректировки 11](#_Toc162449919)

[3.4.1. Реализация метода корректировки 11](#_Toc162449920)

[3.4.2. Оценка времени корректировки 12](#_Toc162449921)

[4. Альтернативный вариант 13](#_Toc162449922)

[4.1. Структура данных 13](#_Toc162449923)

[4.1.1. Реализация структуры на языке C++ 13](#_Toc162449924)

[4.1.2. Расчёт памяти, занимаемой массивом 14](#_Toc162449925)

[4.1.3. Оценка времени доступа к i-ому элементу 14](#_Toc162449926)

[4.2. Метод поиска 15](#_Toc162449927)

[4.2.1. Реализация метода поиска 15](#_Toc162449928)

[4.2.2. Среднее количество сравнений 15](#_Toc162449929)

[4.2.3. Оценка времени поиска 15](#_Toc162449930)

[4.3. Метод упорядочивания 16](#_Toc162449931)

[4.3.1. Реализация метода упорядочивания 16](#_Toc162449932)

[4.3.2. Среднее количество сравнений 17](#_Toc162449933)

[4.3.3. Оценка времени упорядочивания 18](#_Toc162449934)

[4.4. Метод корректировки 18](#_Toc162449935)

[4.4.1. Реализация метода корректировки 18](#_Toc162449936)

[4.4.2. Оценка времени корректировки 19](#_Toc162449937)

[5. Таблица результатов и вывод 19](#_Toc162449938)

[5.1. Вывод 20](#_Toc162449939)

[6. Приложения 20](#_Toc162449940)

[6.1. Основной вариант 20](#_Toc162449941)

[6.2. Альтернативный вариант 23](#_Toc162449942)

**Вариант 12**

# **Цель лабораторной работы**

Цель работы: определить основные критерии оценки структуры данных и методов ее обработки применительно к конкретной задаче

# **Описание задания**

## **Задания**

1. На основе теоретических сведений выделить критерии оценки структур данных, принципы работы и критерии оценки операций поиска, сортировки и корректировки.
2. В соответствии с вариантом задания (см. табл. 1.2) предложить конкретную схему структуры данных (в задании указана абстрактная структура данных) и способ ее реализации на выбранном языке программирования.
3. Определить качественные критерии оценки (универсальность, тип доступа и др.) полученной на шаге 2 структуры данных с учетом специфики задачи по выданному варианту.
4. Определить количественные критерии оценки полученной на шаге 2 структуры данных: требуемый объем памяти на единицу информации, на структуру данных в целом и др.
5. Провести сравнительный анализ структуры данных, предложенной на шаге 2, на основе оценок, полученных на шаге 3 и шаге 4, с другими возможными вариантами реализации с целью поиска лучшей структуры данных к заданию по варианту.
6. Если цель шага 5 достигнута, то необходимо выполнить шаг 2, но для новой абстрактной структуры данных с указанием качественных и количественных критериев.
7. Оценить применимость метода поиска, который указан в варианте задания, с учетом структуры данных.
8. Если метод поиска применим, то необходимо сформулировать его достоинства и недостатки, используя качественные и количественные критерии: универсальность, требуемые ресурсы для реализации, среднее количество сравнений, время выполнения (такты) и др.
9. Предложить альтернативный, более эффективный метод поиска (отличный от задания), если такой существует, с учетом специфики задачи по варианту, а также с учетом структур данных, полученных на предыдущих шагах. Для обоснования выбора альтернативного метода поиска использовать качественные и количественные критерии.
10. Оценить применимость метода упорядочивания, который указан в варианте задания, с учетом структуры данных.
11. Если метод упорядочивания применим, то необходимо сформулировать его достоинства и недостатки, используя качественные и количественные критерии: универсальность, требуемые ресурсы для реализации, среднее количество сравнений, время выполнения (такты) и др.
12. Предложить альтернативный метод упорядочивания, более эффективный и отличный от задания, если такой существует. При этом должны учитываться задача по варианту и структура данных. Для обоснования выбора альтернативного метода упорядочивания использовать качественные и количественные критерии.
13. Оценить применимость метода корректировки, который указан в задании, к структуре данных.
14. Если метод корректировки применим, то необходимо сформулировать его достоинства и недостатки, используя качественные и количественные критерии: универсальность, требуемые ресурсы для реализации, время выполнения (такты) и др.
15. Предложить альтернативный способ корректировки, более эффективный и отличный от задания, если такой существует. При этом должны учитываться задача по варианту, структура данных. Для обоснования выбора альтернативного способа корректировки использовать качественные и количественные критерии.
16. Определить влияние метода корректировки на выполнение операций поиска и упорядочивания.
17. Определить основной режим работы программы и с учетом этого сделать выводы, а итоговые полученные результаты внести в табл. 1.3. Из данной таблицы должно следовать, что предложенный альтернативный вариант решения задачи лучше. Как минимум должно быть одно улучшение, но могут быть заменены все методы обработки и сама структура данных.

## **Основные требования**

Таблица 1 – основные требования в соответствии с вариантом

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер варианта | Номер задачи | Структура данных | Метод | | |
| Поиска | Упорядочения | Корректировки |
| 12 | 3 | Таблица | Дихотомический (двоичный) | Вставкой | Удаление маркировкой |

## **Задача**

Даны M записей вида: код группы, ФИО, дата рождения.

# **Основной вариант**

## **Структура данных**

Необходимо выбрать реализацию структуры данных таблица, элементы которой связаны неявно. Её полями будут являться номер группы, ФИО человека и его дата рождения. Тип данных каждого поля записи разный, поэтому целесообразно использовать структуру.

Таблица 2 – определение типа данных

|  |  |  |  |
| --- | --- | --- | --- |
| Столбец | groupCode | name | birthDate |
| Тип данных | unsigned int | string | string |

Чтобы иметь большую вариативность в установке положительного номера группы, так как по определению не может быть отрицательным, воспользуемся типом unsigned int.

Самым тривиальным способом хранения записей будет являться статический массив. На рисунке 1 показана схема структуры данных, где k – фиксированное число.



Рисунок 1 – схема структуры данных

### **Реализация структуры на языке C++**

Листинг 1 – реализация структуры данных основного варианта

|  |
| --- |
| struct Person {  unsigned int groupCode; //код группы  std::string name; //ФИО  std::string birthDate; //дата рождения  bool delete\_flag; //маркер удаления  static int count;  //пустой конструктор, конструктор с параметрами, деструктор  Person() { count++; }  Person(unsigned int groupCode, std::string name, std::string birthDate) : groupCode(groupCode), name(name), birthDate(birthDate), delete\_flag(false) { count++; }  ~Person() { count--; }  bool operator<(const Person& other) const { return name < other.name; }  };  //определим количество  int Person::count = 0;  // Размер массива  const int maxSize = 20;  …  //Ввод таблицы  Person persons[maxSize] = {  {2033, "Арбузов Иван Сергеевич", "15.04.2004"},  {1015, "Дулина Ирина Арбузова", "01.10.1999"},  {7001, "Абрикосов Владимир Владимирович", "31.12.1978"},  {2033, "Иванова Арбуз Васильевна", "20.03.1970"},  {3005, "Яблонева Карина Облепиховна", "31.02.2002"}  }; |

### **Расчёт памяти, занимаемой массивом**

Объем занимаемой памяти массивом 𝑉 = 𝑘\*𝑉э, где 𝑘 — количество элементов, а 𝑉э — размер одного элемента.

Размер элемента является суммой размера полей элемента. Рассчитаем эти размеры, учитывая, что 1 символ занимает 1 байт (на поле ФИО зарезервируем около 80 байт, в среднем дата рождения занимает 15 символов, что равно 15 байтам, также учитываем байт завершения строки):

𝑉э = 𝑙code + 𝑙name + 𝑙date + *lflag*≃ 4 + 80 + 16 +1 = 101

Получаем 𝑉 = 101𝑘 байт.

### **Оценка времени доступа к i-ому элементу**

В массиве доступ выполняется по индексу. Для удобства мы храним адрес лишь первого элемента массива, поэтому для доступа мы должны прибавить к этому адресу 𝑖, что займет 𝑡++ = 1 такт, доступ к данным по вычисленному адресу совершается за 𝑡→ = 1 такт процессора.

𝑇д = 𝑡++ + 𝑡→ = 1 + 1 = 2 такта.

### **Оценка времени удаления i-ого элемента**

При удалении i-го элемента выполняют сдвиг остальной части массива, т.е. циклическое смещение элементов, после выхода из цикла необходимо также уменьшить значение переменной, хранящей текущее количество элементов.

Листинг 2 – сдвиг массива при удалении элемента

|  |
| --- |
| for (int j = 0; j < n - 1; j++) {  a[j] = a[j + 1];  }  n-=1; |

Количество тактов при удалении элемента массива равно:

tуст: int i=0 – 1 такт

tпров: j<n-1 – 2 такта

tмод: j++ – 1 такт

DTi=tfor+t++=tуст+tпров+2+(k-i)(t=lэл+tмод+tпр+1)+t++=1+2+2+(k-i)(2\*101+1+2+1)+1=5+206(k-i) тактов

## **Метод поиска**

Необходимо реализовать бинарный поиск – классический алгоритм поиска элементов в отсортированном массиве, что и является основным недостатком данного метода. Однако данный метод предоставляет высокую скорость выполнения благодаря уменьшению количества сравнений и итераций, что особенно заметно при работе с большими объемами данных, а также минимизация времени выполнения поиска в сравнении с линейным поиском.

### **Реализация метода поиска**

Листинг 3 – реализация метода бинарного поиска

|  |
| --- |
| int binarySearch(std::string name, Person\* persons) {  int left = 0;  int right = Person::count - 1;  while (left <= right) {  int middle = (left + right) / 2;  if (!persons[middle].delete\_flag && persons[middle].name == name) {  return middle;  }  else if (persons[middle].name < name) {  left = middle + 1;  }  else {  right = middle - 1;  }  }  return -1;  } |

### **Среднее количество сравнений**

Среднее количество сравнений для операции поиска равно:

C=

### **Оценка времени поиска**

Ниже приведена оценка времени поиска.

T = t=+t=+C(t++t=+t++t/+tif) = 2 + 2 + C(2+2+2+28+11) = 4+45\*С

p1t1 : if (persons[middle].name == name) {return middle;}

p2tif: if (persons[middle].name < name) { left = middle + 1;}

tif = tусл + 1 + p1t1 + p2tif = t=+t[] + tp + 1 + p1(1) + p2(tусл+1+p1t1+p2t2) =

= t=+t[] + tp + 1 + p1 + p2(t++t[] + tp +1+p1(t=+t++)+p2(t=+t++)) =

= 2 + 2 + 1+ 1 + 0.5 + 0.5(2+2+1+ 1+0.5(2+1)+0.5(2+1))=11

В итоге: T = 4+45([(k+1)log2(k+1)]/(k-1))

## **Метод упорядочивания**

По зданию необходимо реализовать и оценить упорядочивание массива методом вставки. Среди преимуществ можно выделить простоту в реализации и понимании, а также эффективность в сравнении со сложными алгоритмами при работе с небольшими массивами. Из этого следует, что использование постоянного перемещения элементов будет не таким быстрым для больших данных.

### **Реализация метода упорядочивания**

Листинг 4 – реализация метода сортировки вставкой

|  |
| --- |
| void insertionSort(Person\* arr) {  for (int i = 1; i < Person::count; ++i) {  Person current = arr[i];  int j = i - 1;  // Сдвигаем элементы, пока не найдем место для вставки  while (j >= 0 && !current.delete\_flag && current < arr[j]) {  arr[j + 1] = arr[j];  j--;  }  // Вставляем элемент на его место, если он не помечен для удаления  if (!current.delete\_flag) {  arr[j + 1] = current;  }  }  } |

### **Среднее количество сравнений**

Среднее количество сравнений для операции сортировки вставкой равно:

C = N (N-1)/4

### **Оценка времени упорядочивания**

T = tfor = tуст + tпров + 2 + n(tтела+tмод+tпров+1) = 1+2+2+k(tтела +1+2+1) = 5 + k(304 + 52\*k(k-1)+4) = 52k3 – 52k2 +308k+5

tтела = t=lэл + t= + C (t+ + tp + t+ + t[] + t=lэл + t++) + tif = 2\*101 + 2 + C(2 + 1 + 2 + 2 + 2\*101 + 1) + tif = 204 + C\*210 +103 = 307 + 210\*C = 307 + 210\*k(k-1)/4 = 77 + 53\*k\*(k-1)

tif =tусл + 1 + p1t1 = tp + 1 + 0.5\*t=\*lэл = 1 + 1 + 0.5\*2\*101 = 103

## **Метод корректировки**

Удаление маркировкой заключается в помечании в специальном поле delete\_flag удаляемого элемента. Однако он физически не удаляется, поэтому приходиться отдельно удалять помеченные записи и сжимать массив. При этом нам не нужно каждый раз вызывать цикл для каждого удаляемого элемента, за один вызов удаляются все сразу за счёт маркировки.

### **Реализация метода корректировки**

Листинг 5 – реализация метода удаления маркировкой

|  |
| --- |
| // Маркировка нужного элемента и удаление всех с delete\_flaf == true  void defineDelete(int index, Person\* persons) {  if (index != -1) {  persons[index].delete\_flag = true;  }  }  void deletePerson(Person\* persons) {  int write\_index = 0;  for (int read\_index = 0; read\_index < Person::count; read\_index++) {  if (!persons[read\_index].delete\_flag) {  persons[write\_index++] = persons[read\_index];  }  }  Person::count = write\_index;  } |

### **Оценка времени корректировки**

Для физического удаления i-го элемента необходимо сначала его пометить, а затем выполнить сжатие массива. Для сжатия необходимо выполнить поиск этого элемента с начала массива. Итого, получим следующее время физического удаления элемента:

* время для маркировки:

MT = t= +tif = t= + tусл + 1 + p1t1 = t= + t=c + 1 + 0.5(tp + t[] + t= ) = 2 + 1 + 1 + 0.5 (1 + 2 + 2) = 4 + 2.5 = 6.5;

* время удаления промаркированных элементов:

DT = t= + tfor + t= = 2\*t= + tуст + tпров + 2 + n(tтела+tмод+tпров+1) = 2\*t= + tуст+tпров+2+k(tif+tмод+tпров+1)=2\*2+1+2+2+k(105.5+1+2+1)=9+109.5k тактов

tif = tусл + 1+ p1t1 = t[] + tp + 1 + 0.5(t=lэл + t++) = 2 + 1 + 1 + 0.5(2\*101 + 1) = 105.5

Тогда T = MT+DT = 6.5+9+109.5k = 15.5 + 109.5k

# **Альтернативный вариант**

* Структура данных – таблица;
* Метод поиска – последовательный;
* Метод упорядочивания – квадратичной выборки;
* Метод корректировки – удаление маркировкой.

## **Структура данных**

В качестве альтернативного варианта возьмём ту же структуру данных с неявными связями – таблицу, однако реализованную не в виде статического массива записей, а в виде динамического. Данное решение позволит нам сэкономить место памяти, так как под каждый новый элемент она будет выделяться по мере необходимости, что позволит более эффективно работать с данными, объём которых заранее неизвестен, а также практически не ограничивает нас в рамках занимаемой памяти. Однако они обладают более низкой производительностью из-за нужды в постоянной управлении памятью: освобождении и выделении.

На рисунке 2 показана схема структуры данных, где k – изменяемое число.



Рисунок 2 – схема структуры данных

### **Реализация структуры на языке C++**

Листинг 6 – реализация структуры данных альтернативного варианта

|  |
| --- |
| struct Person {  unsigned int groupCode; //код группы  std::string name; //ФИО  std::string birthDate; //дата рождения  bool delete\_flag; //маркер удаления  Person() {}  Person(unsigned int groupCode, std::string name, std::string birthDate) :  groupCode(groupCode), name(name), birthDate(birthDate), delete\_flag(false)  {}  bool operator<(const Person& other) const { return name < other.name; }  };  …  // Динамический массив  std::vector<Person> persons;  // Инициализация  persons.push\_back({ 2033, "Иванов Иван Сергеевич", "15.04.2004" });  persons.push\_back({ 1015, "Дулина Ирина Алексеевна", "01.10.1999" });  persons.push\_back({ 7001, "Абрикосов Владимир Владимирович", "31.12.1978" });  persons.push\_back({ 2033, "Иванова Надежда Васильевна", "20.03.1970" });  persons.push\_back({ 3005, "Яблонева Карина Кирилловна", "31.02.2002" }); persons.push\_back({1015, "Дулина Ирина Алексеевна", "01.10.1999"});  persons.push\_back({7001, "Абрикосов Владимир Владимирович", "31.12.1978"});  persons.push\_back({2033, "Иванова Надежда Васильевна", "20.03.1970"});  persons.push\_back({3005, "Яблонева Карина Кирилловна", "31.02.2002"}); |

В виде изменений также были внесено удаления поля, маркирующее удаляемый элемент, убран счётчик, так как теперь у нас нет нужды считать актуальное количество элементов в массиве за счёт существующей функции size(), вследствие этого изменены параметры конструкторов и удалён деструктор, так как теперь за освобождение памяти отвечает динамический массив.

### **Расчёт памяти, занимаемой массивом**

Объем занимаемой памяти массивом 𝑉 = 𝑘\*𝑉э, где 𝑘 — количество элементов, а 𝑉э — размер одного элемента.

Размер элемента является суммой размера полей элемента. Рассчитаем эти размеры, учитывая, что 1 символ занимает 1 байт (на поле ФИО зарезервируем около 80 байт, в среднем дата рождения занимает 15 символов, что равно 15 байтам, также учитываем байт завершения строки):

𝑉э = 𝑙code + 𝑙name + 𝑙date + *lflag*≃ 4 + 80 + 16 +1 = 101

Получаем 𝑉 = 101𝑘 байт.

### **Оценка времени доступа к i-ому элементу**

В массиве доступ выполняется по индексу. Для удобства мы храним адрес лишь первого элемента массива, поэтому для доступа мы должны прибавить к этому адресу 𝑖, что займет 𝑡++ = 1 такт, доступ к данным по вычисленному адресу совершается за 𝑡→ = 1 такт процессора.

𝑇д = 𝑡++ + 𝑡→ = 1 + 1 = 2 такта

## **Метод поиска**

В качестве альтернативного варианта поиска был выбран последовательный поиск, так как для небольших неотсортированных и часто изменяющихся данных, на которых мы испытываем наши функции и процедуры, он будет работать эффективнее, чем бинарный поиск.

### **Реализация метода поиска**

Листинг 7 – реализация последовательного поиска

|  |
| --- |
| // последовательный поиск  int poslSearch(std::string name, const std::vector<Person>& persons) {  for (int i = 0; i < persons.size(); ++i) {  const Person& person = persons[i];  if (!person.delete\_flag && person.name == name) {  return i;  }  }  return -1;  } |

### **Среднее количество сравнений**

C =

### **Оценка времени поиска**

T = tfor + t = tуст + tпров + 2 + k(tтела + tмод + tпров +1) + t =

= t=c + t+ + 2 + k(t= + t[] + tif + t++ + t+ + 1) + t =

= t=c + t+ + 2 + k(t= + t[] +tусл + 1 + p1t1 + t++ + t+ + 1) + t =

= t=c + t+ + 2 + k(t= + t[] +tp + t=\*lэл+ 1 + p1t + t++ + t+ + 1) + t =

= 1 + 2 + 2 + k(2 + 2 + 1 + 2\*101 + 1 + 0.5\*1 + 1 + 2 + 1) + 1 =

= 6 + 212.5k

## **Метод упорядочивания**

Упорядочиваемый массив, состоящий из N элементов, делится на групп по элементов в каждой. Для каждой группы необходимо выделить поле памяти, достаточное для размещения значения ключевого признака. Совокупность таких полей образует зону линейного накопления SN. В результате просмотра элементов каждой группы определяют наименьший элемент, который заносят в соответствующее поле зоны линейного накопления SN. Затем просматривают элементы в SN и наименьший из элементов заносят в зону формирования SR. Далее осуществляют просмотр в группе, элемент которой записан в зону SR. Наименьший элемент из оставшихся элементов в группе заносят в зону накопления SN на место элемента, перенесенного в зону формирования SR. После этого снова просматривают элементы в SN и наименьший из элементов заносят в зону формирования SR. Процесс повторяют, пока в зоне SN не останется элементов.

Квадратичная выборка по сравнению с сортировкой выбором имеет меньшее число сравнений элементов, однако требует дополнительного объема памяти.

### **Реализация метода упорядочивания**

Листинг 8 – реализация квадратичной выборки

|  |
| --- |
| // квадратичная выборка  // поиск минимального индекса в 1 группе  int min\_index(const std::vector<Person>& persons, int start, int end) {  int ind\_min = -1;  std::string min\_str(255, 'я');  for (int i = start; i < end; i++) {  if (i < persons.size()) {  if (persons[i].name < min\_str){// && persons[i].name != min\_str) {  min\_str = persons[i].name;  ind\_min = i;  }  }  }  return ind\_min;  }  // вычисляем минимальный индекса в массиве Sn  int min\_index\_sn(const std::vector<Person>& persons, const std::vector<int>& arr, int length) {  int ind\_min = -1;  std::string min\_str(255, 'я');  for (int i = 0; i < length; i++) {  if (arr[i] != -1) {  if (persons[arr[i]].name < min\_str) {  min\_str = persons[arr[i]].name;  ind\_min = i;  }  }  }  return ind\_min;  }  // Упорядочивание квадратичной выборкой  void quadrSort(std::vector<Person>& persons) {  // Количество групп  const int num\_groups = ceil(sqrt(persons.size()));  // Массив для хранения индексов начала каждой группы  std::vector<int> group\_starts(num\_groups);  // Разделение на группы SN и ищем мин индекс в каждой из них, лишние индексы заполняются -1  for (int i = 0; i < num\_groups; i++) {  group\_starts[i] = min\_index(persons, i \* num\_groups, (i + 1) \* num\_groups);  }  // ищем минимальное в готовом массиве Sn  int index\_groups = min\_index\_sn(persons, group\_starts, num\_groups);  std::vector<Person> resultArray(persons.size());  int i = 0;  //теперь в цикле движемся по каждой группе заново  while (index\_groups != -1) {  resultArray[i] = persons[group\_starts[index\_groups]];  persons[group\_starts[index\_groups]].name = std::string(255, 'я');  group\_starts[index\_groups] = min\_index(persons, index\_groups \* num\_groups, (index\_groups + 1) \* num\_groups);  index\_groups = min\_index\_sn(persons, group\_starts, num\_groups);  i++;  }  std::swap(persons, resultArray);  } |

### **Среднее количество сравнений**

C =

### **Оценка времени упорядочивания**

Так как сама функция достаточно сложна подсчитаем только такты, времени выполнения функций *min\_index* и *min\_index\_sn*.

– корень из k раз вызовем тот же цикл for (т.к. в функцию передается группа из элементов.

– аналогично, как в tmin\_index

*while* – отработает за *k* тактов

Tвн = t=\*lэл + t[] + tp + t=c + t[] + tmin\_index + t= + tmin\_index\_sn + t++ = 2\*101 + 2 + 1 + 1 + 2 + + 2 + + 1 = 211+2

T = t=c + tfor + t= + + t=c + k\*Tвн =

= t=c + tуст + tпров + 2 + k(tтела + tмод + tпров +1) + t= + + t=c + k\*Tвн =

= t=c + t=c + t+ + 2 + k(t[] + t= + + t++ + t+ +1) + t= + + t=c + k\*Tвн =

= 1 + 1 + 2 + 2 + k(2 + 2 + + 1 + 2 +1) + 2 + + 1 + k(211+2) =

= 3k + 219k + + 9

## **Метод корректировки**

Удаление маркировкой заключается в помечании в специальном поле delete\_flag удаляемого элемента. Однако он физически не удаляется, поэтому приходиться отдельно удалять помеченные записи и сжимать массив.

### **Реализация метода корректировки**

Листинг 9 – реализация метода удаления маркировкой

|  |
| --- |
| // Маркировка нужного элемента и удаление всех с delete\_flaf == true  void defineDelete(int index, std::vector<Person>& persons) {  if (index != -1) {  persons[index].delete\_flag = true;  }  }  void deletePerson(std::vector<Person>& persons) {  int write\_index = 0;  for (int read\_index = 0; read\_index < persons.size(); read\_index++) {  if (!persons[read\_index].delete\_flag) {  persons[write\_index++] = persons[read\_index];  }  }  persons.resize(write\_index);  } |

### **Оценка времени корректировки**

Для физического удаления i-го элемента необходимо сначала его пометить, а затем выполнить сжатие массива. Для сжатия необходимо выполнить поиск этого элемента с начала массива. Итого, получим следующее время физического удаления элемента:

* время для маркировки:

MT = t= +tif = t= + tусл + 1 + p1t1 = t= + t=c + 1 + 0.5(tp + t[] + t= ) = 2 + 1 + 1 + 0.5 (1 + 2 + 2) = 4 + 2.5 = 6.5;

* время удаления промаркированных элементов:

DT = t= + tfor + t= = 2\*t= + tуст + tпров + 2 + n(tтела+tмод+tпров+1) = 2\*t= + tуст+tпров+2+k(tif+tмод+tпров+1)=2\*2+1+2+2+k(105.5+1+2+1)=9+109.5k тактов

tif = tусл + 1+ p1t1 = t[] + tp + 1 + 0.5(t=lэл + t++) = 2 + 1 + 1 + 0.5(2\*101 + 1) = 105.5

Тогда T = MT+DT = 6.5+9+109.5k = 15.5 + 109.5k

# **Таблица результатов и вывод**

Таблица 3 – таблица результатов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вариант** | **Структура данных** | **Метод поиска** | **Метод упорядоче-ния** | **Метод корректиро-вки** |
| CCОсновной | Статичес-кий массив | Бинарный | Сортировка вставкой | Удаление маркировкой |
| V = 101k | C = | C = | MT = 6.5 |
| T = 4+45 | T = 52k3 – 52k2 +308k+5 | DT = 15.5 + 109.5k |
| Альтерна-тивный | Динамичес-кий массив | Последовательный | Квадратичная выборка | Удаление маркировкой |
| V = 101k | C = | C = | MT = 6.5 |
| T = 6 + 212.5k | T = 3k + 219k + + 9 | DT = 15.5 + 109.5k |

Несмотря на то, что альтернативный вариант выигрывает по времени в методах поиска и упорядочения, мы вынуждены занимать больше места в оперативной памяти для хранения ссылок на предыдущий и следующий элемент динамического массива.

## **Вывод**

В результате выполнения лабораторной работы были изучены различные виды структур данных, методы поиска, упорядочения и корректировки, были сделаны количественные и качественные оценки структур и методов их обработки.

# **Приложения**

## Основной вариант

Листинг 10 – реализация основного варианта

|  |
| --- |
| #include <iostream>  #include <locale.h>  #include <string>  #include <cstdlib>  struct Person {  unsigned int groupCode; //код группы  std::string name; //ФИО  std::string birthDate; //дата рождения  bool delete\_flag; //маркер удаления  static int count;  //пустой конструктор, конструктор с параметрами, деструктор  Person() { count++; }  Person(unsigned int groupCode, std::string name, std::string birthDate) : groupCode(groupCode), name(name), birthDate(birthDate), delete\_flag(false) { count++; }  ~Person() { count--; }  bool operator<(const Person& other) const { return name < other.name; }  };  //определим количество  int Person::count = 0;  // Размер массива  const int maxSize = 20;  void printTable(Person\* persons) {  std::cout << "---------------------------------------------------" << std::endl;  std::cout << "| Код группы | ФИО | Дата рождения |" << std::endl;  std::cout << "---------------------------------------------------" << std::endl;  for (int i = 0; i < Person::count; ++i) {  if (!persons[i].delete\_flag) {  std::cout << "| " << persons[i].groupCode << " | " << persons[i].name << " | " << persons[i].birthDate << " |" << std::endl;  }  }  std::cout << "---------------------------------------------------" << std::endl;  }  // Функция сортировки вставкой  void insertionSort(Person\* arr) {  for (int i = 1; i < Person::count; ++i) {  Person current = arr[i];  int j = i - 1;  // Сдвигаем элементы, пока не найдем место для вставки  while (j >= 0 && !current.delete\_flag && current < arr[j]) {  arr[j + 1] = arr[j];  j--;  }  // Вставляем элемент на его место, если он не помечен для удаления  if (!current.delete\_flag) {  arr[j + 1] = current;  }  }  }  // Бинарный поиск  int binarySearch(std::string name, Person\* persons) {  int left = 0;  int right = Person::count - 1;  while (left <= right) {  int middle = (left + right) / 2;  if (!persons[middle].delete\_flag && persons[middle].name == name) {  return middle;  }  else if (persons[middle].name < name) {  left = middle + 1;  }  else {  right = middle - 1;  }  }  return -1;  }  // Маркировка нужного элемента и удаление всех с delete\_flaf == true  void defineDelete(int index, Person\* persons) {  if (index != -1) {  persons[index].delete\_flag = true;  }  }  void deletePerson(Person\* persons) {  int write\_index = 0;  for (int read\_index = 0; read\_index < Person::count; read\_index++) {  if (!persons[read\_index].delete\_flag) {  persons[write\_index++] = persons[read\_index];  }  }  Person::count = write\_index;  }  // Добавление элемента в массив  void addPerson(Person\* persons, unsigned int groupCode, const std::string& name, const std::string& birthDate) {  if (Person::count >= maxSize) {  std::cerr << "Ошибка: Массив переполнен!" << std::endl;  return;  }  persons[Person::count++] = Person(groupCode, name, birthDate);  }  int main()  {  setlocale(LC\_ALL, "Russian");  system("chcp 1251"); // настраиваем кодировку консоли  //Ввод таблицы  Person persons[maxSize] = {  {2033, "Арбузов Иван Сергеевич", "15.04.2004"},  {1015, "Дулина Ирина Арбузова", "01.10.1999"},  {7001, "Абрикосов Владимир Владимирович", "31.12.1978"},  {2033, "Иванова Арбуз Васильевна", "20.03.1970"},  {3005, "Яблонева Карина Облепиховна", "31.02.2002"}  };  //Сортировка таблицы  std::cout << "Несортированная таблица" << std::endl;  printTable(persons);  insertionSort(persons);  std::cout << "Сортированная таблица" << std::endl;  printTable(persons);  //Поиск по имени  std::cout << "Введите ФИО" << std::endl;  std::string name;  std::getline(std::cin, name);  int a = binarySearch(name, persons);  if (a != -1) {  std::cout << "Найденный человек:" << std::endl;  std::cout << persons[a].groupCode << " " << persons[a].name << " " << persons[a].birthDate << std::endl;  }  else std::cout << "Такого человека в таблице нет" << std::endl;  //Удаление маркировкой  defineDelete(a, persons);  a = binarySearch("Иванова Надежда Васильевна", persons);  defineDelete(a, persons);  deletePerson(persons);  std::cout << "Таблица с удалённым значением" << std::endl;  printTable(persons);  addPerson(persons, 1101, "Захаров Арбузер Владимирович", "12.12.1912");  printTable(persons);  return 0;  } |

## Альтернативный вариант

Листинг 11 – реализация альтернативного варианта

|  |
| --- |
| #include <iostream>  #include <vector>  #include <algorithm>  #include <string>  #include <locale.h>  #include <cstdlib>  struct Person {  unsigned int groupCode; //код группы  std::string name; //ФИО  std::string birthDate; //дата рождения  bool delete\_flag; //маркер удаления  Person() {}  Person(unsigned int groupCode, std::string name, std::string birthDate) :  groupCode(groupCode), name(name), birthDate(birthDate), delete\_flag(false)  {}  bool operator<(const Person& other) const { return name < other.name; }  };  //Вывод  void printTable(const std::vector<Person>& persons) {  std::cout << "---------------------------------------------------" << std::endl;  std::cout << "| Код группы | ФИО | Дата рождения |" << std::endl;  std::cout << "---------------------------------------------------" << std::endl;  for (const Person& person : persons) {  std::cout << "| " << person.groupCode << " | " << person.name << " | " << person.birthDate << " |" << std::endl;  }  std::cout << "---------------------------------------------------" << std::endl;  }  // последовательный поиск  int poslSearch(std::string name, const std::vector<Person>& persons) {  for (int i = 0; i < persons.size(); ++i) {  const Person& person = persons[i];  if (!person.delete\_flag && person.name == name) {  return i;  }  }  return -1;  }  // квадратичная выборка  // поиск минимального индекса в 1 группе  int min\_index(const std::vector<Person>& persons, int start, int end) {  int ind\_min = -1;  std::string min\_str(255, 'я');  for (int i = start; i < end; i++) {  if (i < persons.size()) {  if (persons[i].name < min\_str){// && persons[i].name != min\_str) {  min\_str = persons[i].name;  ind\_min = i;  }  }  }  return ind\_min;  }  // вычисляем минимальный индекса в массиве Sn  int min\_index\_sn(const std::vector<Person>& persons, const std::vector<int>& arr, int length) {  int ind\_min = -1;  std::string min\_str(255, 'я');  for (int i = 0; i < length; i++) {  if (arr[i] != -1) {  if (persons[arr[i]].name < min\_str) {  min\_str = persons[arr[i]].name;  ind\_min = i;  }  }  }  return ind\_min;  }  // Упорядочивание квадратичной выборкой  void quadrSort(std::vector<Person>& persons) {  // Количество групп  const int num\_groups = ceil(sqrt(persons.size()));  // Массив для хранения индексов начала каждой группы  std::vector<int> group\_starts(num\_groups);  // Разделение на группы SN и ищем мин индекс в каждой из них, лишние индексы заполняются -1  for (int i = 0; i < num\_groups; i++) {  group\_starts[i] = min\_index(persons, i \* num\_groups, (i + 1) \* num\_groups);  }  // ищем минимальное в готовом массиве Sn  int index\_groups = min\_index\_sn(persons, group\_starts, num\_groups);  std::vector<Person> resultArray(persons.size());  int i = 0;  //теперь в цикле движемся по каждой группе заново  while (index\_groups != -1) {  resultArray[i] = persons[group\_starts[index\_groups]];  persons[group\_starts[index\_groups]].name = std::string(255, 'я');  group\_starts[index\_groups] = min\_index(persons, index\_groups \* num\_groups, (index\_groups + 1) \* num\_groups);  index\_groups = min\_index\_sn(persons, group\_starts, num\_groups);  i++;  }  std::swap(persons, resultArray);  }  // Маркировка нужного элемента и удаление всех с delete\_flaf == true  void defineDelete(int index, std::vector<Person>& persons) {  if (index != -1) {  persons[index].delete\_flag = true;  }  }  void deletePerson(std::vector<Person>& persons) {  int write\_index = 0;  for (int read\_index = 0; read\_index < persons.size(); read\_index++) {  if (!persons[read\_index].delete\_flag) {  persons[write\_index++] = persons[read\_index];  }  }  persons.resize(write\_index);  }  int main() {  setlocale(LC\_ALL, "Russian");  system("chcp 1251"); // настраиваем кодировку консоли  // Динамический массив  std::vector<Person> persons;  // Инициализация  persons.push\_back({ 2033, "Иванов Иван Сергеевич", "15.04.2004" });  persons.push\_back({ 1015, "Дулина Ирина Алексеевна", "01.10.1999" });  persons.push\_back({ 7001, "Абрикосов Владимир Владимирович", "31.12.1978" });  persons.push\_back({ 2033, "Иванова Надежда Васильевна", "20.03.1970" });  persons.push\_back({ 3005, "Яблонева Карина Кирилловна", "31.02.2002" });  std::cout << "Несортированная таблица" << std::endl;  printTable(persons);  std::cout << persons.size() << std::endl;  //Поиск по имени  std::cout << "Введите ФИО" << std::endl;  std::string name;  std::getline(std::cin, name);  int a = poslSearch(name, persons);  if (a != -1) {  std::cout << "Найденный человек:" << std::endl;  std::cout << persons[a].groupCode << " " << persons[a].name << " " << persons[a].birthDate << std::endl;  }  else std::cout << "Такого человека в таблице нет" << std::endl;  //Сортировка таблицы=  quadrSort(persons);  std::cout << "Сортированная таблица" << std::endl;  printTable(persons);    //Удаление маркировкой  defineDelete(a, persons);  a = poslSearch("Иванов Иван Сергеевич", persons);  defineDelete(a, persons);  deletePerson(persons);  std::cout << "Таблица с удалённым значением" << std::endl;  printTable(persons);  persons.push\_back({ 1101, "Захаров Борис Владимирович", "12.12.1912" });  printTable(persons);  return 0;  } |