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 (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

ОТЧЕТ

по лабораторной работе № 1

Вариант 11

Название: Методы обработки данных и оценки программ

Дисциплина: Технологии разработки программных продуктов

Студент	ИУ6-43Б		В.К. Залыгин
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

1 Цель лабораторной работы

Целью данной работы является определение основных критериев оценыки структуры данных и методов ее обработки применительно к конкретной задаче.

2 Описание задания

2.1 Задание

Дана таблица материальных нормативов, состоящая из K записей фиксированной длины вида: код детали, код материала, единица измерения, номер цеха, норма расхода.

2.2 Основные требования

Основной вариант задания включает в себя следующие требования:

- структура данных – таблица;
- поиск – вычисление адреса;
- упорядочение – пузырьком;
- корректировка – удаление сдвигом.

3 Описание основного варианта задания

3.1 Структура данных

Структуру данных по заданию (таблицу) реализуем с помощью одномерного статического массива записей. Его устройство изображено на рисунке 1.

1	Запись 1
	. . . . .
i	Запись *i*
	. . . . .
k	Запись *k*

Рисунок 1 – Схема структуры данных

У каждой записи есть следующие поля:

- код детали (целое число от 0 до 65536, unsigned short);
- код материала (строка в 8 символов, char[8]);
- единицы измерения (строка в 8 символов, char[8]);
- номер цеха (целое число, int);
- норма расхода (целое число, int).

Будем исходить из того, что код детали должен быть уникальным.

3.1.1 Реализация структуры на C++

Реализация структуры данных на языке C++ представлена на рисунке 2.

```
#define MAX 65535
#define SIZE 128

typedef struct {
  unsigned short code;     // код детали
  char material_code[8];   // код материала
  char measure[8];         // единицы измерения
  int plant_number;        // номер цеха
  int consumption;         // норма расхода
} detail;

detail table[SIZE];
```

Рисунок 2 – Определение структуры данных

3.1.2 Расчет потребления памяти

Размер памяти, занимаемой массивом, рассчитывается формулой $V = k \cdot V_э$, где k – количество элементов в массиве, $V_э$ – размер памяти, которую занимаем 1 элемент массива. Размер памяти, занимаемой элементов, определяется как сумма размеров полей этого элемента:

$$V_э = l_{\text{код дет.}} + l_{\text{код мат.}} + l_{\text{единицы измерения}} + l_{\text{номер цеха}} + l_{\text{норма расхода}},$$

где l – размер памяти, которое занимает некоторое поле (название поля подписывается). Тогда потребление памяти на 1 элемент: $V_э = 2 + 8 \cdot 1 + 8 \cdot 1 + 4 + 4 = 26$ байт, при условии, что размер символа в строке равен 1 байту. Тогда потребление памяти массивом $V = 26k$ байт.

3.1.3 Оценка времени доступа к i-му элементу

Доступ выполняется по индексу. Следовательно, достаточно потратить 2 такта на получение элемента из массива:

$$T_{\text{д}} = t_i = 2 \text{ такта.}$$

3.2 Анализ применимости метода поиска

По заданию необходимо использовать поиск по вычислению адреса. Данный метод поиска применим к таблице, так как она позволяет осуществлять прямой доступ к хранящимся элементам. Поскольку размер таблицы может быть меньше, чем множество всех возможных значений код детали, будем считать, что одной ячейке в таблице соответствует группа значений кода детали. При этом в ячейке одномоментно может храниться только один элемент из группы – сильное ограничение, связанное с предложенной структурой данных и методов их обработки.

3.2.1 Реализация метода поиска

Реализация, удовлетворяющая заданию и результатами анализа применимости метода поиска, приведена на рисунке 3

```
detail * find(unsigned short code) {
  return table + ((code * SIZE) / MAX);
}
```

Рисунок 3 – Метод поиска элемента через вычисление его адреса

3.2.2 Среднее количество сравнений

Поскольку метод поиска предполагает непосредственное вычисление адреса элемента количество сравнений равно нулю.

3.2.3 Оценка времени поиска

Ниже приведена оценка времени поиска.

$$T_{\text{п}} = T_{*} + T_{/} + T_{+} = 20 + 28 + 2 = 50 \text{ тактов.}$$

3.3 Анализ метода упорядочивания

В задании предлагается реализовать сортировку методом пузырька. С учетом работы метода поиска выгодно брать произвольный элемент массива, вычислять адрес, на котором он должен стоять, а потом менять его с тем элементом, который стоит на данном адресе.

3.3.1 Реализация метода

Реализация приведена на рисунке 4.

```
void sort() {
  for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
      detail * a = table + j;
      detail * b = find(a->code, t);
      if (a != b) {
        swap(a, b);
      }
    }
  }
}
```

Рисунок 4 – Реализация метода сортировки

3.3.2 Оценка количества сравнений

Так как сравнения происходят в двойном цикле, то количество сравнений при размере массива N будет: $C = N \cdot N = N^2$

3.4 Метод корректировки

Предлагается использовать удаление сдвигом. То есть все элементы, стоящие за удаляемым сдвигаются назад на один шаг.

3.4.1 Реализация метода

На рисунке 5 приведена реализация метода.

```
void remove(unsigned short idx) {
  for (int i = idx; i < SIZE-1; i++) {
    table[i] = table[i+1];
  }
}
```

Рисунок 5 – Реализация метода удаления элементов

3.4.2 Расчет времени удаления

Если требуется удалить элемент под номером i в массиве, размером n, то имеет место быть следующая формула:

$$DT_i = t_{for} = t_{\text{усл}} + t_{\text{пр}} + 1 + (n - i - 1)(t_{\text{тело}} + t_{\text{пр}} + 2) =$$

$$= 2 + 1 + 2 + 1 + (n - i - 1)(1 + 2 + 2 + 2 + 1 + 2 + 2) =$$

$$= 6 + 12(n - i - 1) \text{ тактов.}$$

4. Альтернативный вариант

4.1 Выбор альтернативной структуры данных

В качестве альтернативной структуры данных предлагается гнездовой способ организации с массивом гнезд и массивом элементов внутри гнезда. Структура данных изображена на рисунке 6.

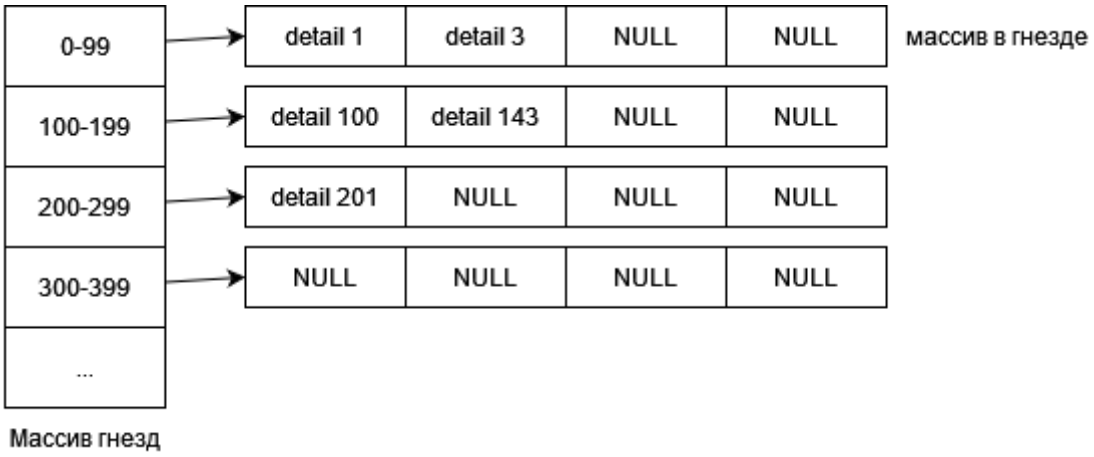

Рисунок 6 – Схема альтернативной структуры данных

Детали разбиваются на гнезда по номеру. Тогда решается главный недостаток предложенного способа – невозможность записи нескольких деталей из одной группы в структуру данных.

Альтернативный вариант организации методом обработки и доступа:

- поиск – вычисление адреса;
- упорядочение – пузырьком;
- корректировка – удаление сдвигом.

У каждой записи есть следующие поля:

- код детали (целое число от 0 до 65536, unsigned short);

- код материала (строка в 8 символов, char[8]);
- единицы измерения (строка в 8 символов, char[8]);
- номер цеха (целое число, int);
- норма расхода (целое число, int).

Код реализации структуры на языке С представлен на рисунке 7.

```c
#define MAX 65535

typedef struct {
  unsigned short detail_code; // код детали
  char material_code[8];      // код материала
  char measure[8];            // единицы измерения
  int plant_number;           // номер цеха
  int consumption;            // норма расхода
} element;

typedef struct {
  element * elements;
  int size;
} socket;

typedef struct {
  socket * sockets;
  int size;
} table;
```

Рисунок 7 – Код альтернативной структуры данных

4.1.2 Расчет занимаемой памяти

При учете, что размер указателя – 4 байта. Размер гнезда размером k:

$V_{\text{гнездо}} = V_{.} + kV_{\text{деталь}} + V_{int} = 26k + 8$ байт.

Размер структуры данных, если гнезд g штук:

$V = (V_{\text{гнездо}} + V_{.})g + V_{int} = (26k + 4)g + 4$ байт.

Наиболее оптимальным для данных, равномерно распределенных по диапазону ключей, считается k = g, k * g = N, где N – количество деталей.

4.1.3 Оценка времени доступа

$$T_{\text{д}} = T_{i} + T_{i} = 2 + 2 = 4 \text{ такта.}$$

4.2 Метод поиска

Для реализации поиска предлагается использовать вычисление вычисления адреса для нахождения гнезда, а затем метод последовательного поиска детали внутри гнезда.

4.2.1 Реализация метода

Реализация метода поиска показана на рисунке 7.

```
element * find(unsigned short detail_code, table t) {
  socket s = t.sockets[(code * SIZE) / MAX];
  for (int i = 0; i < s.size; i++) {
    if (s.elements[i].detail_code == detail_code) {
      return s.elements[i];
    }
  }
}
```

Рисунок 7 – Метод поиска

4.2.2 Среднее количество сравнений при поиске

Сравнения производятся только при поиске внутри гнезда. Всего элементов в гнезде $\sqrt{N}$ – оптимальный размер гнезда при общем количестве элементов N.

Тогда количество сравнений в среднем определяется формулой:

$$C = \frac{\sqrt{N}}{2}$$

4.3 Метод упорядочивания

Предлагается поочередно сортировать значения в каждом гнезде. Внутри гнезда будет применять сортировку методом Шелла.

4.3.1 Оценка количества сравнений

Известно, что при сортировке Шелла количество сравнений определяется формулой $C = \frac{1}{2}K^{\frac{3}{2}}$, где K – количество элементов в гнезде. В одном гнезде $K = \sqrt{N}$ элементов, где N – количество элементов в целом. Всего гнезд $\sqrt{N}$. Тогда получим итоговую формулу:

$$C = \frac{1}{2}N^{\frac{3}{4}}N^{\frac{1}{2}} = \frac{1}{2}N^{\frac{3}{8}}$$

4.4 Метод корректировки

В альтернативном случае используется метод корректировки заменой. В рамках одного гнезда удаляемый элемент заменяется элементом из конца массива.

4.4.1 Реализация метода

Алгоритм метода удаления представлен на рисунке 9.

```
void remove(unsigned short e_idx, unsigned short s_idx, table t) {
  socket s = t.socket[s_idx]
  s.elements[e_idx] = s.elements[s.size-1];
}
```

Рисунок 9 – Метод удаления элемента.

4.4.1 Оценка времени работы метода корректировки

$$DT = t_{=} \cdot 2 + t_{i} \cdot 3 = 10 \text{ тактов.}$$

Заключение

В результате выполнения лабораторной работы были проведены качественные и количественные оценки структур данных и методов их обработки в соответствии с вариантом задания. В альтернативном варианте предложены решения, которые обеспечат более эффективные сортировку и удаление данных с меньшими ограничениями к данным.

Таблица 1 – Таблица результатов

Варианты	Структура данных	Метод поиска	Метод сортировки	Метод корректировки
Основной	Статический массив. $V = 26k$ байт.	Вычисление адреса. $T_{\text{п}} = 50$ тактов.	Пузырьком. $C = N^2$	Удаление сдвигом. $DT = 6 + 12(n - i - 1)$ тактов.

Альтернати-вный	Гнездовая. $V = (26k + 4)g + 4$ байт.	Вычисление адреса + последовательный. $C = \frac{\sqrt{N}}{2}$	Метод Шелла. $\mathrm{C} = \frac{1}{2}N^{\frac{3}{8}}$	Удаление заменой. $DT = 10$ тактов.

Из недостатков можно назвать больший объем требуемой памяти и более медленный поиск, который все же обладает приемлемой асимптотикой.