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 **09.03.03 ПРИКЛАДНАЯ ИНФОРМАТИКА**

О Т Ч Е Т

по лабораторной работе № 1
Вариант 7

Название: Исследование структур и методов обработки данных

Дисциплина: Технология разработки программных систем

Студент ИУ6-44Б _____ Я.А. Гришина
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Е.К. Пугачев
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Введение

При разработке алгоритмов программ часто возникает задача выбора структур данных и методов их обработки. Исходными составляющими для решения этой задачи являются описание набора и типов, хранимых данных, а также перечень операций, выполняемых над ними.

Можно выделить следующие основные вопросы, на которые необходимо ответить при решении поставленной задачи:

- Как логически организовать структуру данных? Как ее реализовать?
- Как осуществлять поиск информации? Как упорядочить данные?
- Как выполнить функции корректировки данных?

Цель работы – исследование структур данных, методов их обработки и оценки.

Задание

1. Ознакомиться с теоретическими сведениями по абстрактным структурам данных и методами их обработки.

2. Для указанной задачи и типа данных (см. таблицу 2) предложить способ реализации и определить требуемый объем памяти.

3. Провести анализ заданных методов поиска, упорядочения и корректировки. Оценить время выполнения соответствующих операций.

4. Предложить альтернативный вариант решения задачи, в котором должно быть минимум одно улучшение. Улучшения могут касаться как структуры данных, так и основных операций. Обосновать новые решения, используя количественные и качественные критерии. Количественными критериями являются: объем памяти, среднее количество сравнений и количество тактов. Качественные критерии определяют возможность использования того или иного метода применительно к разработанной структуре. К ним можно отнести: применимость операции только к упорядоченным данным; необходимость знать количество элементов; наличие признака разбивки на гнезда; необходимость в прямом доступе к элементам; знание граничных значений; невозможность создать структуру в соответствии с арифметической прогрессией и др.

Исходные данные:

Задача 2: Дана таблица материальных нормативов, состоящая из K записей фиксированной длины вида: код детали; код материала; единица измерения; номер цеха; норма расхода.

Структура: Таблица;

Поиск: Метод дихотомии (двоичный поиск);

Упорядочение: Вставка;

Корректировка: Удаление сдвигом.

Основная часть

1. Исходные варианты структуры и методов её обработки

Для хранения записи Detail использован статический массив details размера K и индексами от 0 до K-1 (см. рис. 1).

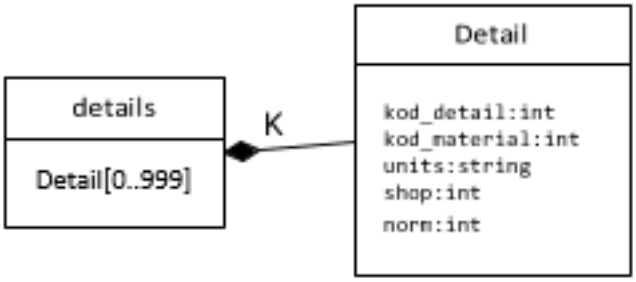

Рисунок 1. Диаграмма статического массива и его элемента

В листинге 1 представлены реализации структуры Details и записи details на языке C++.

Листинг 1

```
struct Detail
{
    int kod_detail;
    int kod_material;
    string units;
    int shop;
    int norm;
};
Detail details[1000];
```

1.1. Определение объёма данных

Объем данных, занимаемой одной записью: 4 * 4 + 28 = 44 Байт.

Объем памяти, занимаемой массивом, содержащим K записей: K * 44 Байт.

Объем памяти, занимаемой массивом, содержащим 1000 записей: 44000 Байт.

1.2. Анализ алгоритма поиска

По заданию необходимо реализовать и оценить способ поиска методом дихотомии. Двоичный (бинарный) поиск (или дихотомия) — классический алгоритм поиска элемента в отсортированном массива, использующий дробление массива на половины.

В листинге 2 представлена реализация алгоритма двоичного поиска в структуре Details на языке C++.

Листинг 2

```
int Search(Detail* arr, int n, int key) {
    int left = 0; // задаем левую и правую границы поиска
    int right = n;
    int search = -1; // найденный индекс элемента равен -1
(элемент не найден)

    while (left <= right) // пока левая граница не "перескочит"
правую
    {
        int mid = (left + right) / 2; // ищем середину отрезка
        if (key == arr[mid].kod_detail) {  // если ключевое поле
равно искомому
            search = mid;     // мы нашли требуемый элемент,
            break;            // выходим из цикла
        }
        if (key < arr[mid].kod_detail)     // если искомое
ключевое поле меньше найденной середины
            right = mid - 1;  // смещаем правую границу,
продолжим поиск в левой части
        else                  // иначе
            left = mid + 1;   // смещаем левую границу,
продолжим поиск в правой части
    }
    return search;

}
```

Оценка времени поиска i-го элемента массива:

$T=t_{\text{установкинач. данных}} + t_{\text{цикла}} = (2*t_{=const} + t_{=}) + m((t_{<=} + (t_{=} + t_{a+b} + t_{/}) + (t_{a[i]} + t_{==} + 1 + t_{.} + 0{,}5(t_{=}) + t_{<} + 1 + t_{a[i]} + t_{.} + 0{,}5(t_{=} + t_{+}) + 0{,}5(t_{=} + t_{+}) = 2 + 2 + m * (2 + 2 + 2+28+2+1+2=2+1+2+1+2+2+3) = 4 + 51m$

1.3. Анализ алгоритма упорядочения

По заданию необходимо реализовать и оценить упорядочение массива методом вставки.

В листинге 3 представлена реализация алгоритма упорядочения записей структуры Details на языке C++.

Листинг 3.

```
void Sort(Detail* arr, int n) {
    int counter = 0;
    for (int i = 1; i < n; i++) {
        for (int j = i; j > 0 && arr[j - 1].kod_detail > 
arr[j].kod_detail; j--) {
            counter++;
            Detail tmp = arr[j - 1];
            arr[j - 1] = arr[j];
            arr[j] = tmp;
        }
    }
    cout << counter << endl;
     }
```

Оценка времени упорядочения структуры методом вставки

Размер массива	Количество перестановок
2	1
4	6
8	16
16	69
32	251
64	1107
128	3940
256	16520
512	63856

1.4. Анализ алгоритма удаления

По заданию необходимо реализовать и оценить удаление сдвигом.

В листинге 4 представлена реализация алгоритма удаления записей из структуры Details на языке C++.

Листинг 4

```
void Del(Detail* arr, int n, int index) {
    for (int i = index+1; i < n; i++) arr[i - 1] = arr[i];
     }
```

Оценка времени удаления i-го элемента структуры

$t_{++} + t_{=} + t_{+} + 2 + (K-1)*(t_{++} + 2*t_{[]} + t_{+} + 1) = 7+8(K-1)$

Выводы: использование статического массива подразумевает выделение объема памяти, иногда больше, чем необходимо; сортировка вставками обеспечивает упорядочение структуры за квадратичное время.

2. Альтернативные варианты структуры и методы ее обработки

Для хранения записи Detail использован динамический массив details (см. рис. 2).

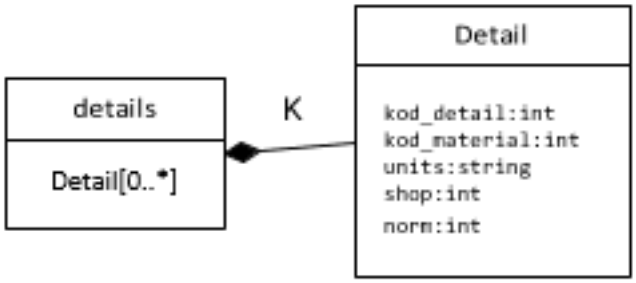

Рисунок 2. Диаграмма динамического массива и его элемента

В листинге 5 представлены реализации структуры Details и записи details на языке C++.

Листинг 5

```
struct Detail
{
    int kod_detail;
    int kod_material;
    string units;
    int shop;
    int norm;
};
Detail* details = new Detail[k];
```

2.1. Определение объёма данных

Объем данных, занимаемой одной записью: 4 * 4 + 28 = 44 Байт.

Объем памяти, занимаемой массивом, содержащим K записей: K * 44 Байт.

Всего система может выделить на программу 2 Гб (2 147 483 647 Байт). Таким образом, максимальное количество элементов в массиве может быть 2 147 483 647 / 44 = 48 806 446.

2.2. Анализ алгоритма упорядочения

Реализуем упорядочение массива методом Шелла.

В листинге 6 представлена реализация алгоритма упорядочения записей структуры Details на языке C++.

Листинг 6.

```
void Sort(Detail* arr, int n) //сортировка Шелла
{
   int d = n / 2;
   while (d > 0)
    {
        for (int i = 0; i < n - d; i++)
        {
            int j = i;
            while (j >= 0 && arr[j].kod_detail >
arr[j+d].kod_detail)
            {

                Detail tmp = arr[j];
                arr[j] = arr[j+d];
                arr[j+d] = tmp;
                j--;
            }
        }
        d = d / 2;
    }
}
```

Оценка времени упорядочения структуры методом Шелла

Размер массива	Количество перестановок
2	1
4	5
8	17
16	49
32	129
64	321
128	769
256	1793
512	4097

Выводы: использование динамических массивов позволяет добиться более эффективного использования памяти, выделяемой под структуру, по сравнению со статическими массивами; сортировка Шелла в худшем случае обеспечивает упорядочение массива за O(n2), но с меньшим коэффициентом амортизации по сравнению с сортировкой вставками.

Заключение

	Структура данных	Алгоритм поиска	Алгоритм упорядочивания	Корректировка
Исходный вариант	Статический массив (статическая таблица)	Дихотомический	Метод вставки	Удаление сдвигом
	44000 Байт	$C_{ср} = (N+1)*\log2(N+1)/N - 1$ $T = 4 + 51m$	$C = N(N-1)/N$	$T = 7+8(K-1)$
Альтернативный вариант	Динамический массив (динамическая таблица)	Дихотомический	Метод Шелла	Удаление сдвигом
	44*N Байт, N – число введенных записей	$C_{ср} = (N+1)*\log2(N+1)/N - 1$ $T = 4 + 51m$	$C = N^{3/2}/2$	$T = 7+8(K-1)$