МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Учебно-методическое пособие

ГУАП
Санкт-Петербург
2017

УДК 681.3.01
ББК 32.98
М35

Рецензенты:
кандидат технических наук, доцент *А. А. Ключарев*;
доктор технических наук, профессор *Л. А. Осипов*

Утверждено
редакционно-издательским советом университета
в качестве учебно-методического пособия

Матьяш, В. А.

М35 Структуры и алгоритмы обработки данных: учеб.-метод. пособие / В. А. Матьяш, С. А. Рогачев. – СПб.: ГУАП, 2017. – 66 с.

Рассматриваются методы анализа сложности алгоритмов, хеширование данных, сортировка данных и алгоритмы на деревьях и графах. Изложение материала сопровождается примерами. Методические материалы по выполнению соответствующих лабораторных работ и примеры их выполнения позволяют студенту самостоятельно изучить соответствующий раздел дисциплины.

Предназначены для выполнения лабораторных работ по дисциплинам «Структуры и алгоритмы и обработки данных» и «Структуры и алгоритмы компьютерной обработки данных» студентами различных форм обучения, проходящих подготовку по направлениям «Прикладная математика и информатика» 01.03.02, 02.03.03 «Математическое обеспечение и администрирование информационных систем» и 09.03.04 «Программная инженерия».

УДК 681.3.01
ББК 32.98

ОБЩИЕ УКАЗАНИЯ

Номер варианта задания на лабораторную работу определяется студентом как две последние цифры номера его студенческого билета, взятые по модулю количества вариантов заданий на лабораторную работу, то есть

$$N_{\text{вар}} = (nn \bmod K) + 1,$$

где $N_{\text{вар}}$ – номер варианта; nn – две последние цифры номера студенческого билета; K – количество вариантов заданий на лабораторную работу.

Например, две последние цифры номера студенческого билета – «76», количество вариантов – 25. Тогда номер варианта задания на лабораторную работу, которое необходимо выполнить студенту будет

$$N_{\text{вар}} = (76 \bmod 25) + 1 = 2.$$

Для сдачи лабораторной работы необходим распечатанный отчет, содержание которого определяется соответствующим пунктом, для каждой лабораторной работы, данных методических указаний. Образец титульного листа можно получить на сайте ГУАП в разделе нормативной документации.

Система оценки лабораторных работ доводится до сведения студентов преподавателем на первом занятии учебного семестра.

1. Лабораторная работа «Анализ сложности алгоритмов»

1.1. Цель работы

Целью работы является изучение методов и получение практических навыков анализа сложности алгоритмов.

1.2. Задание на лабораторную работу

Используя память, пропорциональную n, хранить массив целых чисел A, содержащий n элементов.

Элементы массива A могут принимать случайные значения от $-((ndiv2)-1)$ до $(ndiv2)$. То есть, если в массиве хранится *10* элементов, то эти элементы должны быть в диапазоне от -4 до 5.

Разработать алгоритм, который осуществляет заполнение массива A случайными значениями, и по выбору пользователя выполняет одну из двух функций. Состав выполняемых функций и требования к теоретической временной сложности этих функций определяется вариантом задания. Варианты задания приведены в табл. 1.

Таблица 1

№ вар.	Функции	$T(n)$, не более
1	Подсчитать сумму всех элементов, имеющих положительные значения	$O(n)$
	Подсчитать количество элементов с нулевым значением	$O(1)$
2	Подсчитать сумму всех элементов, имеющих положительные значения	$O(n)$
	Подсчитать количество элементов с положительными значениями	$O(1)$
3	Подсчитать сумму всех элементов, имеющих положительные значения	$O(n)$
	Подсчитать количество элементов с отрицательными значениями	$O(1)$
4	Подсчитать сумму всех элементов, имеющих положительные значения	$O(n)$
	Подсчитать количество элементов с четными значениями	$O(1)$

Продолжение табл. 1

№ вар.	Функции	$T(n)$, не более
5	Подсчитать сумму всех элементов, имеющих положительные значения	$O(n)$
	Подсчитать количество элементов с нечетными значениями	$O(1)$
6	Подсчитать сумму всех элементов, имеющих отрицательные значения	$O(n)$
	Подсчитать количество элементов с нулевым значением	$O(1)$
7	Подсчитать сумму всех элементов, имеющих отрицательные значения	$O(n)$
	Подсчитать количество элементов с положительными значениями	$O(1)$
8	Подсчитать сумму всех элементов, имеющих отрицательные значения	$O(n)$
	Подсчитать количество элементов с отрицательными значениями	$O(1)$
9	Подсчитать сумму всех элементов, имеющих отрицательные значения	$O(n)$
	Подсчитать количество элементов с четными значениями	$O(1)$
10	Подсчитать сумму всех элементов, имеющих отрицательные значения	$O(n)$
	Подсчитать количество элементов с нечетными значениями	$O(1)$
11	Заменить все отрицательные значения элементов на модули их значений	$O(n)$
	Подсчитать количество элементов с нулевым значением	$O(1)$
12	Заменить все отрицательные значения элементов на модули их значений	$O(n)$
	Подсчитать количество элементов с положительными значениями	$O(1)$
13	Заменить все отрицательные значения элементов на модули их значений	$O(n)$
	Подсчитать количество элементов с отрицательными значениями	$O(1)$
14	Заменить все отрицательные значения элементов на модули их значений	$O(n)$
	Подсчитать количество элементов с четными значениями	$O(1)$

Окончание табл. 1

№ вар.	Функции	$T(n)$, не более
15	Заменить все отрицательные значения элементов на модули их значений	$O(n)$
	Подсчитать количество элементов с нечетными значениями	$O(1)$
16	Все четные значения элементов уменьшить в два раза	$O(n)$
	Подсчитать количество элементов с нулевым значением	$O(1)$
17	Все четные значения элементов уменьшить в два раза	$O(n)$
	Подсчитать количество элементов с положительными значениями	$O(1)$
18	Все четные значения элементов уменьшить в два раза	$O(n)$
	Подсчитать количество элементов с отрицательными значениями	$O(1)$
19	Все четные значения элементов уменьшить в два раза	$O(n)$
	Подсчитать количество элементов с четными значениями	$O(1)$
20	Все четные значения элементов уменьшить в два раза	$O(n)$
	Подсчитать количество элементов с нечетными значениями	$O(1)$
21	Все значения элементов кратные 10 уменьшить на один порядок	$O(n)$
	Подсчитать количество элементов с нулевым значением	$O(1)$
22	Все значения элементов кратные 10 уменьшить на один порядок	$O(n)$
	Подсчитать количество элементов с положительными значениями	$O(1)$
23	Все значения элементов кратные 10 уменьшить на один порядок	$O(n)$
	Подсчитать количество элементов с отрицательными значениями	$O(1)$
24	Все значения элементов кратные 10 уменьшить на один порядок	$O(n)$
	Подсчитать количество элементов с четными значениями	$O(1)$
25	Все значения элементов кратные 10 уменьшить на один порядок	$O(n)$
	Подсчитать количество элементов с нечетными значениями	$O(1)$

1.3. Порядок выполнения работы

1) выбрать вариант задания из подраздела 1.2 в соответствии с требованиями;
2) изучить теоретический материал, изложенный в учебном пособии;
3) разработать и реализовать на языке программирования высокого уровня алгоритм, выполняющий требования задания;
4) рассчитать теоретические временную и пространственную сложности алгоритма;
5) написать отчет о работе;
6) защитить отчет.

1.4. Содержание отчета

Отчет должен содержать:
1) титульный лист;
2) цель работы;
3) вариант задания;
4) листинг программы, реализующей алгоритм;
5) расчет пространственной и временной сложностей алгоритма;
6) расчет теоретической пространственной и теоретической временной сложностей алгоритма;
7) выводы по работе.

1.5. Пример выполнения работы

Предположим, что необходимо выполнить следующий вариант задания:

№ вар.	Функции	$T(n)$, не более
26	Найти сумму значений всех элементов массива	$O(n)$
	Подсчитать количество элементов со значениями, равными m	$O(1)$

Стоит отметить, что приведенный ниже код программы, нужен только для демонстрации подсчета характеристик сложностей алгоритма, и не реализует всех поставленных задач.

Составляем алгоритм, который выполняет требования задания:

```
#include <stdio.h>

const int NUMBER_TO_COMPARE = 3; // Число для
   сравнений

int calculationAmount(int *mas, int n)
{
 int sum = 0;
 for(int i = 0; i < n; i++)
 {
  sum += mas[i];
 }
 printf("Amount: %i \n", sum);
 return sum;
}
void countingDuplicates(int duplicates)
{
 printf("Count: %i \n",duplicates);
}

int main()
{
 int A[] = {5, 7, 3, 1, 3, 8}; // Исходный массив
 int N = sizeof(A) / sizeof(int); // Количество элементов
   в массиве
 calculationAmount(A, N);
 int count=0;
 for (int i = 0; i < N; i++)
 {
  if (A[i] == NUMBER_TO_COMPARE)
  {
   count++;
  }
 }
 countingDuplicates(count);
};
```

Теперь подсчитываем теоретические сложности алгоритма. Разработанный алгоритм использует следующие данные:

– одну константу;

– один массив размерностью n;

– пять переменных целого типа.

Значит, пространственная сложность алгоритма определяется следующим образом:

$$v = C_{\text{const}} + n * C_{\text{int}} + 5 * C_{\text{int}},$$

где C_{const} – константа, характеризующая объем памяти, отводимый под константу; C_{int} – константа, характеризующая объем памяти, отводимый под переменную целого типа.

Теоретическая пространственная сложность алгоритма составляет:

$$V(n) = O(v) = O(\max(O(C_{\text{const}}),\ \ O(n * C_{\text{int}}), O(5 * C_{\text{int}}))) =$$

$$O(\max(O(1),\ \ O(n),\ \ O(1))) = O(n)$$

Временную сложность алгоритма определяем на основе анализа текста программы, реализующей этот алгоритм. Согласно заданию на лабораторную работу, необходимо реализовать две функции, теоретические временные сложности которых не превышают заданных. Поэтому необходимо рассчитать теоретическую временную сложность функции, реализующих эти алгоритмы:

$$t_{\text{Sum}} = K_7 + n * K_{10} + K_{12} + K_{13},$$

$$t_{\text{Count}} = K_{17},$$

$$t_{\text{Alg}} = K_{22} + K_{23} + K_{25} + n * (K_{28} + \max(K_{30}, 0)) + t_{\text{Sum}} + t_{\text{Count}}$$

где K_i – константа, характеризующая время выполнения операций, помеченных i; t_{Sum}, t_{Count} и t_{Alg} – временные сложности функций calculationAmount, countingDuplicates и всего алгоритма в целом, соответственно.

Теоретическая временная сложность функций составляет:

$$T_{\text{Sum}}(n) = O(t_{\text{Sum}}) = O(\max(O(K_7),\ \ O(n*K_{10}),\ \ O(K_{12}),\ \ O(K_{13}))) =$$

$$O(\max(O(1), O(n), O(1),\ \ O(1))) = O(n)$$

$$T_{\text{Count}}(n) = O(t_{\text{Count}}) = O(K_{18}) = O(1).$$

На основе этих расчетов можно сделать вывод о том, что был разработан алгоритм, характеристики которого соответствуют поставленному заданию.

1.6. Контрольные вопросы

1. Что такое алгоритм и его сложность?
2. Какие существуют способы оценки сложности алгоритма?
3. Что показывает теоретическая временная сложность?
4. Что показывает теоретическая пространственная сложность?
5. Какие факторы оказывают влияние на экспериментальную оценку временной сложности алгоритма?
6. Понятие *O*-символики.
7. Назовите базовые принципы, по которым осуществляется теоретическая оценка временной сложности алгоритмов.
8. Какие плюсы и минусы при использовании *O*-символики?

2. Лабораторная работа «Линейные и циклические списки»

2.1. Цель работы

Целью работы является изучение структур данных «линейный список» и «циклический список», а также получение практических навыков их реализации.

2.2. Задание на лабораторную работу

Реализовать структуры данных «линейный список» и «циклический список» в соответствии с заданным вариантом. Дополнительно программа должна осуществлять следующие операции:

1) добавление/удаление элемента в список (с клавиатуры);

2) вывод исходного и результирующего списков на экран;

3) если списки являются многочленами, в выводе должна быть отражена степень каждого элемента.

Варианты задания приведены в табл. 2. Элементы последовательности, или коэффициенты многочлена (в зависимости от варианта) – числовые значения элемента списка, количество таких элементов списка равно длине последовательности, или количеству коэффициентов многочлена.

Таблица 2

№ вар.	Задача	Вид списка
1	Дана последовательность повторяющихся целых чисел a_1, a_2,..., a_n. Получить последовательность k_1, k_2, ..., k_m, содержащую повторяющиеся в исходной последовательности элементы (c) в порядке убывания частоты их повторения в исходной последовательности (count). Неповторяющиеся элементы не включать. $k_1 = \max(\text{count}(c))$, $k_2 = \min(\max(\text{count}(c), k_1)$....	Линейный односвязный
2	Дана последовательность неповторяющихся чисел a_1, a_2,... a_m. Получить две последовательности. Первая – список целых частей каждого числа в исходной последовательности. Вторая – список дробных частей, если элемент исходной последовательности является целым числом, такой элемент не включается в результирующие последовательности. $B = \text{whole}(a_1), \text{whole}(a_2)...$ $K = \text{fract}(a_1), \text{fract}(a_2)$....	Линейный односвязный

Продолжение табл. 2

№ вар.	Задача	Вид списка
3	Даны натуральное число n и символы $s_1, s_2, \ldots, s_n$. Получить символы, принадлежащие последовательности $s_1, s_2, \ldots, s_n$, которые входят в нее по одному разу	Линейный односвязный
4	Даны 2 многочлена. Каждый многочлен $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ с целыми коэффициентами можно представить в виде списка, причем если $a_i = 0$, то соответствующее звено не включать в список. Определить процедуру, которая строит многочлен p – сумму многочленов q и r	Линейный односвязный
5	Дано натуральное число n и целые числа $a_1, a_2, \ldots, a_n$. Вычислить $\min\limits_{1 \le i \le n} \lvert a_i - \overline{a} \rvert$, где $\overline{a}$ среднее арифметическое чисел $a_1, a_2, \ldots, a_n$	Линейный односвязный
6	Дано натуральное число n и целые числа $a_1, a_2, \ldots, a_n$. Требуется получить последовательность $x_1, x_2, \ldots x_k, y_1, y_2, \ldots, y_k$, где $x_1, \ldots, x_k$ – взятые в порядке следования (слева на право) четные члены последовательности $a_1, \ldots, a_n$, а $y_1, \ldots, y_k$ – нечетные члены	Линейный односвязный
7	Дана последовательность неповторяющихся целых чисел $a_1, a_2, \ldots, a_n$. Получить последовательность, содержащую наименьший элемент исходной последовательности, и все предшествующие ему. $K = \min(a_1, a_2, \ldots, a_n), a_{\min-1}, a_{\min-2}, \ldots, a_1, \ldots$	Линейный двусвязный
8	Дана последовательность неповторяющихся целых $a_1, a_2, \ldots, a_n$, где $n > 4$. Получить последовательность, содержащую элементы исходной последовательности с удаленными двумя наименьшими и двумя наибольшими элементами. $K = A - (\max(a), \min(a), \max(a - \max), \min(a - \min))$	Линейный двусвязный
9	Дана последовательность латинских букв, оканчивающаяся точкой. Среди букв есть специальный символ, появление которого означает отмену предыдущей буквы; n знаков подряд отменяют n предыдущих букв, если такие есть. Учитывая вхождение этого символа преобразовать последовательность	Линейный двусвязный
10	Даны натуральные числа k, m, n и последовательности символов $s_1, s_2, \ldots, s_k, t_1, t_2, \ldots, t_m, u_1, u_2, \ldots, u_n$. Получить по одному разу те символы, которые входят во все три последовательности, но расположить их по возрастанию	Линейный двусвязный

Продолжение табл. 2

№ вар.	Задача	Вид списка
11	Дана последовательность неповторяющихся чисел $a_1, a_2, ..., a_n$ и некое число c, принадлежащее данной последовательности. Составить 2 последовательности. Первая – все числа, находящиеся до указанного числа в обратном порядке. Вторая – все числа после указанного числа в прямом порядке	Линейный двусвязный
12	Дана последовательность неповторяющихся чисел $a_1, a_2, ..., a_n$. Получить последовательность, содержащую все числа в заданном диапазоне $[f...k]$, которые встречаются в исходной последовательности	Линейный двусвязный
13	Дана последовательность повторяющихся целых чисел $a_1, a_2, ..., a_n$. Получить последовательность, содержащую среднее арифметическое n элементов исходной последовательности, затем $n-1$ элементов и т. д. K = average($a_1, ..., a_n$), average($a_1, ..., a_{n-1}$), ... average(a_1, a_2), a_1	Циклический односвязный
14	Дана последовательность неповторяющихся чисел $a_1, a_2, ..., a_n$. Получить 3 последовательности. Первая содержит элементы исходной последовательности, делящиеся без остатка на 3. Вторая – элементы, делящиеся без остатка на 2. Третья последовательность содержит элементы первой и второй полученных последовательностей, за исключением элементов, которые в них дублируются. Пример: $A = 1, 3, 6, 7, 4, 2$; $K = 3, 6$; $B = 6, 4, 2$; Res = 3, 4, 2	Циклический односвязный
15	Дана последовательность неповторяющихся целых чисел $a_1, a_2, ..., a_n$. Получить последовательность, содержащую $n/2$ элементов. Первый элемент такой последовательности – минимальный элемент исходной последовательности b_1, второй – ближайшее число к $b_1 * 2$, третий – ближайшее к $b_2 * 3$ и т. д. b_1 = min(a), min($a_1 - b_1 * 2$, $a_2 - b_1 * 2$, ..., $a_n - b_1 * 2$),...	Циклический односвязный
16	Многочлен $P(x) = a_n x^n + a_n - {}_1 x^{n-1} + ... + a_1 x + a_0$ с целыми коэффициентами можно представить в виде списка, причем если $a_i = 0$, то соответствующее звено не включать в список. Определить процедуру, которая строит многочлен q – производная многочлена p	Циклический односвязный
17	Многочлен $P(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ с целыми коэффициентами можно представить в виде списка, причем если $a_i = 0$, то соответствующее звено не включать в список. Определить логическую функцию *Equal*(p, q), проверяющие на равенство многочлены p и q	Циклический односвязный

Окончание табл. 2

№ вар.	Задача	Вид списка
18	Проверить, удовлетворяют ли элементы списка (базовый тип integer) закону $x = f(x_0, h)$, где x – элемент списка, h – шаг, x_0 – начальный элемент списка. Пример: $x_0 = 5$, $h = 1$. $x_1 = 6$, $x_2 = 7$, $x_3 = 8$... Элементы списка удовлетворяют закону $x = h(5,1)$	Циклический односвязный
19	Дана последовательность неповторяющихся целых чисел $a_1, a_2, ..., a_n$. Получить последовательность, содержащую поочередно каждый четный элемент исходной последовательности слева – направо, и каждый нечетный элемент последовательности справа – налево. Если n – четное, то: $a_1, a_m, a_2, a_{n-1}, ...$	Циклический двусвязный
20	Дана последовательность чисел $a_1, a_2, ..., a_n$ и число K. Необходимо суммировать элементы исходной последовательности, пока сумма не будет больше или равна K, после чего выводятся все просуммированные элементы	Циклический двусвязный
21	Дана последовательность неповторяющихся целых чисел $a_1, a_2, ..., a_n$. Получить последовательность длиной, содержащую разность между первым и последним элементом, вторым и предпоследним и т. д. Если n – нечетное, центральный элемент копируется без изменений	Циклический двусвязный
22	Дана последовательность неповторяющихся чисел $a_1, a_2, ..., a_n$. Получить 2 последовательности, первая из которых является зеркальным отображением исходной. Вторая – поэлементная разница между исходной и второй последовательностью	Циклический двусвязный
23	Дана последовательность неповторяющихся чисел $a_1, a_2, ..., a_n$. Получить 3 последовательности. Первая – копия исходной последовательности с инвертированными знаками. Вторая – копия исходной последовательности со взятыми по модулю значениями. Третья – поэлементная сумма первой и второй последовательностей, с удаленными из неё нулевыми значениями	Циклический двусвязный

2.3. Порядок выполнения работы

1) выбрать вариант задания из подраздела 2.2 в соответствии с требованиями;

2) изучить теоретический материал, изложенный в учебном пособии;

3) разработать на языке программирования высокого уровня программу, выполняющую поставленную задачу с использованием заданной структуры данных;

4) написать отчет о работе;

5) защитить отчет.

К защите отчета по лабораторной работе, включающую демонстрацию работы программы, необходимо сформировать два или более контрольных примера.

2.4. Содержание отчета

Отчет должен содержать:

1) титульный лист;

2) цель работы;

3) вариант задания;

4) листинг программы, реализующей поставленную задачу с использованием заданных структур данных;

5) контрольные примеры;

6) выводы по работе.

2.5. Пример выполнения работы

Предположим, что необходимо выполнить следующий вариант задания:

№ вар.	Задача	Вид списка
24	Дано натуральное число n и целые числа $a_1, a_2, \ldots, a_n$. Требуется получить последовательность $x_1, x_2, \ldots, x_k$, $y_1, y_2, \ldots, y_k$, где $x_1, \ldots, x_k$ – взятые в порядке следования (слева на право) четные члены последовательности $a_1, \ldots, a_n$, а $y_1, \ldots, y_k$ – нечетные члены	Линейный

Для выполнения поставленного задания потребуется организовать два списка:

– линейный список A, состоящий из n элементов и содержащих исходные целые числа;

– линейный список X, состоящий из n элементов и содержащих в первых k элементах числа $x_1, x_2, \ldots, x_k$, а в последующих элементах – числа $y_1, y_2, \ldots, y_k$.

Элементы обоих списков описываются одинаково.

Теперь можно разработать программу, которая должна выполнять следующие шаги:

Содержимое структуры:

```
struct LinearList
{
 int coeff;
 LinearList *next;
};
```

1) создание и заполнение линейного списка *A*:

```
void AddListElem(int NewListElem, LinearList *&First)
{
 LinearList *NewElem = new LinearList;
 NewElem->next = NULL;
 NewElem->coeff = NewListElem;
 if (First == NULL)
  First = NewElem;
 else
 {
  LinearList *tmp = First;
  while (tmp->next != NULL)
  {
   tmp = tmp->next;
  }
  tmp->next = NewElem;
 }
}
```

2) создание и заполнение линейного списка *X*;

Perv – указатель на первый элемент списка.

FinalSequence – указатель на первый элемент результирующего списка

```
void GetSequence(LinearList *Perv, LinearList *&FinalSequence)
{
  LinearList *ResultCarriage = new LinearList;
  LinearList *Carriage = new LinearList;
  Carriage = Perv;
//Участок с обходом четных элементов
  if (Carriage->next != NULL)
{
```

```
   Carriage = Carriage->next;
   FinalSequence = new LinearList;
   FinalSequence->coeff = Carriage->coeff;
   ResultCarriage = FinalSequence;
   while (Carriage->next != NULL)
   {
     if (Carriage->next->next != NULL)
     {
      Carriage = Carriage->next->next;
      ResultCarriage->next = new LinearList;
      ResultCarriage->next->coeff = Carriage->coeff;
      ResultCarriage = ResultCarriage->next;
     }
     else break;
   }
  }
  else
  {
   FinalSequence = new LinearList;
   FinalSequence->coeff = Carriage->coeff;
   return;
  }
//Участок с обходом нечетных элементов
  Carriage = Perv;
  ResultCarriage->next = new LinearList;
  ResultCarriage->next->coeff=Carriage->coeff;
  ResultCarriage = ResultCarriage->next;
  while (Carriage->next != NULL)
  {
   if (Carriage->next->next != NULL)
   {
     Carriage = Carriage->next->next;
     ResultCarriage->next = new LinearList;
     ResultCarriage->next->coeff = Carriage->coeff;
     ResultCarriage = ResultCarriage->next;
   }
   else break;
  }
 ResultCarriage->next = NULL;
}
```

Для выполнения поставленной задачи совершается последовательный обход исходного линейного списка сначала по чётным элементам, затем по нечётным. При этом необходимо проверять наличие элемента в участке памяти, в который происходит обращение программы при очередной итерации. При значении NULL происходит выход из цикла. Такой подход позволит сохранить исходный список, а так же не выполнять лишние итерации при обходе.

2.6. Контрольные вопросы

1. Что такое список?
2. Назовите основные характеристики линейного однонаправленного списка.
3. На какой элемент линейного однонаправленного списка необходимо обеспечивать позиционирование какого-либо указателя? Обоснуйте ответ.
4. Назовите основные характеристики линейного двунаправленного списка.
5. Сравните линейный однонаправленный список и линейный двунаправленный список.
6. Назовите основные характеристики циклического однонаправленного списка.
7. Сравните линейный однонаправленный список и циклический однонаправленный список.
8. Назовите основные характеристики циклического двунаправленного списка.
9. Сравните линейный двунаправленный список и циклический двунаправленный список.

3. Лабораторная работа «Стек и очередь»

3.1. Цель работы

Целью работы является изучение структур данных «стек» и «очередь», а также получение практических навыков их реализации.

3.2. Задание на лабораторную работу

Реализовать структуры данных «стек» и «очередь» в соответствии с заданным вариантом. Дополнительно программа должна удовлетворять следующим требованиям:

1) Вывод на экран состояния моделируемой системы на каждой итерации работы (содержимое стека(ов), очереди(ей), процессора(ов));

2) Для каждой задачи из списка входных задач должно быть определено время поступления;

3) Необходимо наличие, как автоматического генератора задач, так и возможность ручного добавления задач, с указанием их параметров (в зависимости от задания);

4) Необходимо обработать ситуации, при которых какая-либо структура данных может быть переполнена.

Варианты задания приведены в табл. 3 (формулировки задач приведены после таблицы).

Таблица 3

№ вар.	Номер задачи	Реализация стека и очереди
1	1	Стек – статический; очередь – статическая
2	1	Стек – статический; очередь – динамическая
3	1	Стек – динамический; очередь – статическая
4	1	Стек – динамический; очередь – динамическая
5	2	Стек – статический; очередь – статическая
6	2	Стек – статический; очередь – динамическая
7	2	Стек – динамический; очередь – статическая
8	2	Стек – динамический; очередь – динамическая

Окончание табл. 3

№ вар.	Номер задачи	Реализация стека и очереди
9	3	Стек – статический; очередь – статическая
10	3	Стек – статический; очередь – динамическая
11	3	Стек – динамический; очередь – статическая
12	3	Стек – динамический; очередь – динамическая
13	4	Стек – статический; очередь – статическая
14	4	Стек – статический; очередь – динамическая
15	4	Стек – динамический; очередь – статическая
16	4	Стек – динамический; очередь – динамическая
17	5	Стек – статический; очередь – статическая
18	5	Стек – статический; очередь – динамическая
19	5	Стек – динамический; очередь – статическая
20	5	Стек – динамический; очередь – динамическая
21	6	Стек – статический; очередь – статическая
22	6	Стек – статический; очередь – динамическая
23	6	Стек – динамический; очередь – статическая
24	6	Стек – динамический; очередь – динамическая
25	7	Стек – статический; очередь – статическая
26	7	Стек – статический; очередь – динамическая
27	7	Стек – динамический; очередь – статическая
28	7	Стек – динамический; очередь – динамическая
29	8	Стек – статический; очередь – статическая
30	8	Стек – статический; очередь – динамическая
31	8	Стек – динамический; очередь – статическая
32	8	Стек – динамический; очередь – динамическая

Задача 1.

Система состоит из процессора P, трёх очередей $F0$, $F1$, $F2$ и стека S. В систему поступают запросы на выполнение задач.

Поступающие запросы ставятся в соответствующие приоритетам очереди. Сначала обрабатываются задачи из очереди $F0$. Если она пуста, можно обрабатывать задачи из очереди $F1$. Если и она пуста, то можно обрабатывать задачи из очереди $F2$. Если все оче-

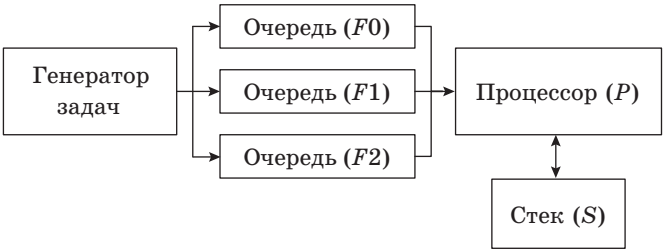

Рис. 3.1

реди пусты, то система находится в ожидании поступающих задач (процессор свободен), либо в режиме обработки предыдущей задачи (процессор занят). Если поступает задача с более высоким приоритетом, чем обрабатываемая в данный момент, то обрабатываемая помещается в стек и может обрабатываться тогда и только тогда, когда все задачи с более высоким приоритетом уже обработаны.

Задача 2.

Система состоит из трех процессоров $P0$, $P1$, $P2$, очереди F, стека S и распределителя задач R. В систему поступают запросы на выполнение задач трёх типов – $T0$, $T1$ и $T2$, каждая для своего процессора.

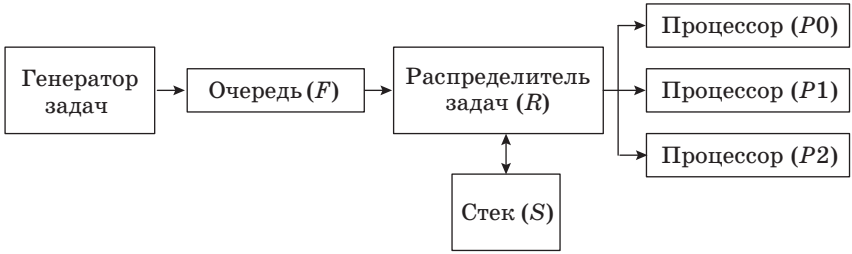

Рис. 3.2

Поступающие запросы ставятся в очередь. Если в начале очереди находится задача T_i и процессор P_i свободен, то распределитель R ставит задачу на выполнение в процессор P_i, а если процессор P_i занят, то распределитель R отправляет задачу в стек и из очереди извлекается следующая задача. Если в вершине стека находится задача, процессор которой в данный момент свободен, то эта задача извлекается и отправляется на выполнение.

Задача 3.

Система состоит из процессора P, трёх очередей $F0$, $F1$, $F2$ и стека S. В систему поступают запросы на выполнение задач.

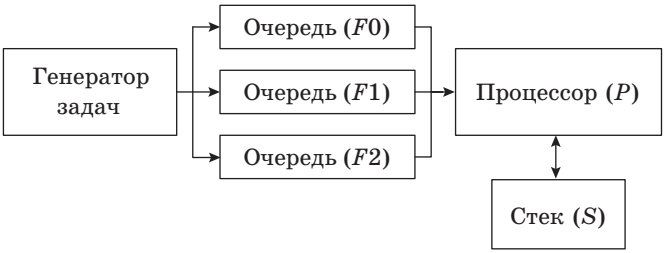

Рис. 3.3

Поступающие запросы ставятся в соответствующие приоритетам очереди. Сначала обрабатываются задачи из очереди $F0$. Если она пуста, можно обрабатывать задачи из очереди $F1$. Если и она пуста, то можно обрабатывать задачи из очереди $F2$. Если все очереди пусты, то система находится в ожидании поступающих задач (процессор свободен), либо в режиме обработки предыдущей задачи (процессор занят). Если поступает задача с более высоким приоритетом, чем обрабатываемая в данный момент, то обрабатываемая помещается в стек и может обрабатываться тогда и только тогда, когда все очереди пусты.

Задача 4.

Система состоит из трех процессоров $P0$, $P1$, $P2$, очереди F, стека S и распределителя задач R. В систему поступают запросы на выполнение задач трёх типов – $T0$, $T1$ и $T2$, каждая для своего процессора.

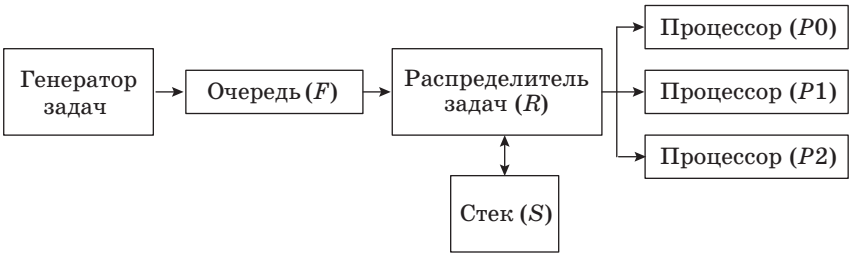

Рис. 3.4

Поступающие запросы ставятся в очередь. Если в начале очереди находится задача T_i и процессор P_i свободен, то распределитель R ставит задачу на выполнение в процессор P_i, а если процессор P_i занят, то распределитель R отправляет задачу в стек и из очереди извлекается следующая задача. Задача из стека поступает в соответствующий ей свободный процессор только тогда, когда очередь пуста.

Задача 5.

Система состоит из двух процессоров $P0$ и $P1$, стека S и очереди F. В систему могут поступать запросы на выполнение задач, причем время выполнения задачи каждым из процессоров, может отличаться. Поступающие запросы ставятся в очередь.

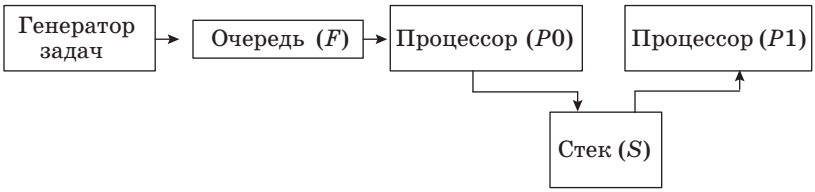

Рис. 3.5

Если процессор $P0$ свободен, то в него поступает на обработку задача из очереди. После обработки задачи процессором $P0$, задача помещается в стек. Если стек не пустой и процессор $P1$ свободен, то задача извлекается из стека, и обрабатывается процессором.

Задача 6.

Система состоит из процессора P, трёх очередей $F0$, $F1$, $F2$ и стека S. В систему поступают запросы на выполнение задач.

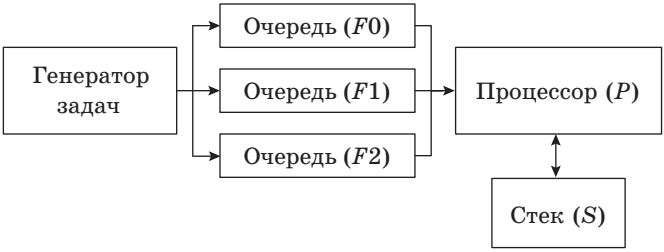

Рис. 3.6

Поступающие запросы ставятся в соответствующие приоритетам очереди. Сначала обрабатываются задачи из очереди $F0$. Если она пуста, можно обрабатывать задачи из очереди $F1$. Если и она пуста, то можно обрабатывать задачи из очереди $F2$. Если все очереди пусты, то система находится в ожидании поступающих задач (процессор свободен), либо в режиме обработки предыдущей задачи (процессор занят). Если поступает задача с более высоким приоритетом, чем обрабатываемая в данный момент, то обрабатываемая помещается в стек, если она выполнена менее чем на половину по времени, и может обрабатываться тогда и только тогда, когда все задачи с более высоким приоритетом уже обработаны.

Задача 7.

Система состоит из двух процессоров $P0$ и $P1$, стека S и очереди F. В систему могут поступать запросы на выполнение задач, причем время выполнения задачи каждым из процессоров, может отличаться. Поступающие запросы попадают в стек.

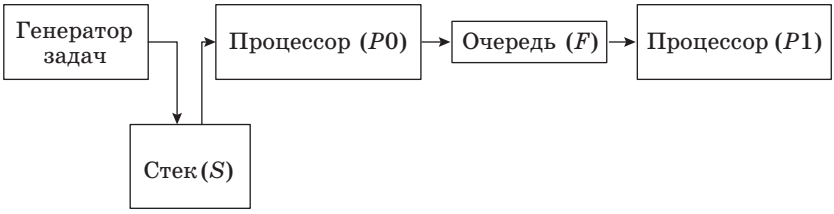

Рис. 3.7

Если процессор *P0* свободен, то в него поступает на обработку задача из стека. После обработки задачи процессором *P0*, задача помещается в очередь. Если очередь не пуста и процессор *P1* свободен, то задача извлекается из очереди, и обрабатывается процессором.

Задача 8.

Система состоит из процессора P, двух стеков *S0* и *S1*, очереди F и распределителя задач R. Поступающие в систему запросы, попадают в очередь.

Распределитель задач (R), получает задачу из очереди и помещает ее либо в стек *S0*, либо в стек *S1* (зависимости от приоритета задачи). Процессор P, обрабатывает задачи из стеков в порядке

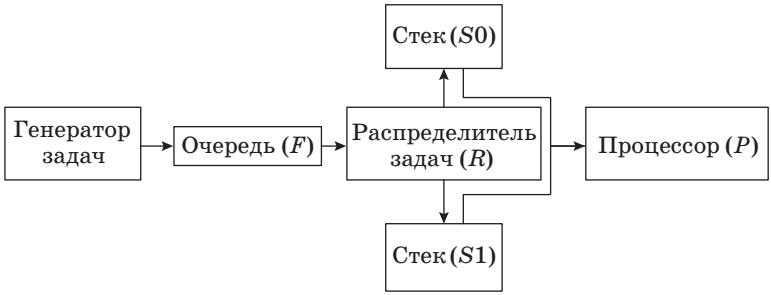

Рис. 3.8

приоритета. Таким образом, если стек *S0* пуст и процессор *P* свободен, то могут быть обработаны задачи из стека *S1*.

3.3. Порядок выполнения работы

1) выбрать вариант задания в соответствии с требованиями раздела;

2) изучить теоретический материал, изложенный в учебном пособии;

3) разработать на языке программирования высокого уровня программу, выполняющую поставленную задачу с использованием заданных структур данных;

4) написать отчет о работе;

5) защитить отчет.

К защите отчета по лабораторной работе, включающую демонстрацию работы программы, необходимо сформировать два или более контрольных примера.

3.4. Содержание отчета

Отчет должен содержать:

1) титульный лист;

2) цель работы;

3) вариант задания;

4) листинг программы, реализующей поставленную задачу с использованием заданных структур данных;

5) контрольные примеры, в которых задействованы все структуры описанные в задании;

6) выводы по работе.

3.5. Пример выполнения работы

Предположим, что необходимо выполнить следующий вариант задания:

№ вар.	Номер задачи	Реализация стека и очереди
33	9	Стек – динамический; очередь – динамическая

Задача 9.

Система состоит из процессора P, очереди F и стека S. В систему поступают запросы на выполнение задач трех приоритетов.

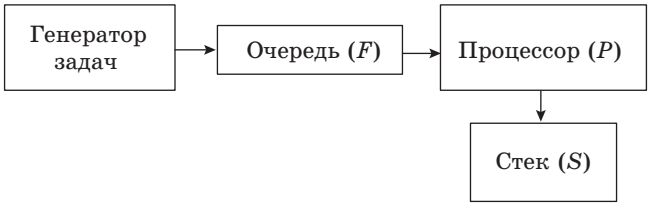

Рис. 3.9

Поступающие запросы ставятся в очередь. Если очередь пуста, то система находится в ожидании поступающих задач (процессор свободен), либо в режиме обработки предыдущей задачи (процессор занят). Если поступает задача с более высоким приоритетом, чем обрабатываемая в данный момент, то обрабатываемая помещается в стек и может обрабатываться тогда и только тогда, когда все задачи с более высоким приоритетом уже обработаны

Генератор задач должен формировать запросы на выполнение задач. В общем случае эти запросы на выполнение задач со случайным приоритетом и случайной длительности должны поступать в случайные моменты времени. Однако, для демонстрации всех свойств системы, необходимо определенная комбинация задач с заданными характеристиками. Поэтому при сдаче лабораторной студенту будет необходимо продемонстрировать работу системы на заранее запланированных примерах.

В приведенном примере, для удобства программирования, будут использованы 2 структуры, Структура, описывающая одну задачу, и структура, позволяющая хранить набор задач. В данном примере у задач нет названия, однако при выполнении лабораторной работы, каждая задача должна иметь своё название.

```
struct Task
{
	uint16_t priority;
	uint16_t taskTime;
	uint16_t durationTime;
};

struct TaskList
{
	Task *taskValues;
	TaskList *next;
};
```

Работа системы моделируется в виде последовательности тактов работы. На каждом такте выполняются следующие действия:

1) проверяется поступление задач на текущий такт, и, если задача поступила, она ставится в очередь;

2) проверяется приоритет ближайшей задачи из очереди, и задачи, обрабатываемой процессором. Далее, в соответствии со схемой работы системы, происходит поступление задач в стек, пока в процессоре не окажется задача с приоритетом более высоким, чем у ближайшей задачи в очереди;

3) увеличивается таймер системы, и уменьшается время выполнения задачи, находящейся в процессоре. Если задача завершена, процессор освобождается.

Текст функции на языке C++, выполняющей данную задачу, приведен ниже:

```
void processorLoop(TaskList *&IncomingTask)
{
 TaskList *Stack = NULL;
 TaskList *Queue = NULL;
 Task *OurProcessor = new Task;
 OurProcessor->priority = 0;
 OurProcessor->durationTime = 0;
 bool emptyQueue = true; //Проверка пустоты Очереди
 bool emptyStack = true; //Проверка пустоты Стека
 bool processorIsFree = true; //Проверка занятости процессора
 bool allTasksGone = false;
 int timer = 1;
 while(true)
```

```
 {
  if (!allTasksGone)
  {
   if (IncomingTask->taskValues->taskTime == timer)
   {
    pushToQueue(IncomingTask, Queue, emptyQueue, allTasksGone);
   }
  }
  if (!emptyQueue)
  {
   if (Queue->taskValues->priority > OurProcessor->priority
|| processorIsFree)
   {
    if (OurProcessor->durationTime > 0)
     pushToStack(Stack, OurProcessor, emptyStack, processor-
IsFree);
    getFromQueue(Queue, OurProcessor, emptyQueue, processor-
IsFree);
    while (true)
    {
     if (!emptyQueue)
     {
      if (Queue->taskValues->priority > OurProcessor->priority)
      {
       pushToStack(Stack, OurProcessor, emptyStack, proces-
sorIsFree);
       getFromQueue(Queue, OurProcessor, emptyQueue, proces-
sorIsFree);
      }
      else
       break;
     }
     else
      break;
    }
   }
  }
  else if (!emptyStack)
  {
   if (processorIsFree)
    getFromStack(Stack, OurProcessor, emptyStack, processor-
IsFree);
```

```
  }
  cout << endl << "Идет " << timer << " такт" << endl;
  if (!allTasksGone)
  {
   cout << "Входные задания" << endl;
   showStruct(IncomingTask);
  }
  if (!emptyStack)
  {
   cout << "Содержимое стэка" << endl;
   showStruct(Stack);
  }
  if (!emptyQueue)
  {
   cout << "Содержимое очереди" << endl;
   showStruct(Queue);
  }
  if (!processorIsFree)
  {
   cout << "Содержимое процессора" << endl;
   showStructElem(OurProcessor);
  }
  else
   cout << «Процессор свободен» << endl;
  if (!processorIsFree)
  {
   if (OurProcessor->durationTime)
    OurProcessor->durationTime--;
   if(OurProcessor->durationTime<=0)
   {
    OurProcessor->durationTime = 0;
    OurProcessor->priority = 0;
    processorIsFree = true;
   }
  }
  timer++;
  if (emptyStack && emptyQueue && processorIsFree && allTasksGone)
   break;
 }
 IncomingTask=NULL;
}
```

Теперь рассмотрим пример работы. Составим, в качестве контрольного примера, следующее расписание поступление запросов на выполнение задач:

Момент поступления	Длительность выполнения	Приоритет
1	2	3
2	1	3
3	2	1
4	3	2
5	3	3

Результат работы программы по заданному расписанию будет следующим:

```
Идет 1 такт
Входные задания
Время поступления задачи 2 Приоритет задачи 3 Такты задачи 1
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 2
Время поступления задачи 4 Приоритет задачи 2 Такты задачи 3
Время поступления задачи 5 Приоритет задачи 3 Такты задачи 3
Содержимое процессора
Время поступления задачи 1 Приоритет задачи 3 Такты задачи 2
Идет 2 такт
Входные задания
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 2
Время поступления задачи 4 Приоритет задачи 2 Такты задачи 3
Время поступления задачи 5 Приоритет задачи 3 Такты задачи 3
Содержимое очереди
Время поступления задачи 2 Приоритет задачи 3 Такты задачи 1
Содержимое процессора
Время поступления задачи 1 Приоритет задачи 3 Такты задачи 1
Идет 3 такт
Входные задания
Время поступления задачи 4 Приоритет задачи 2 Такты задачи 3
Время поступления задачи 5 Приоритет задачи 3 Такты задачи 3
Содержимое очереди
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 2
Содержимое процессора
Время поступления задачи 2 Приоритет задачи 3 Такты задачи 1
```

```
Идет 4 такт
Входные задания
Время поступления задачи 5 Приоритет задачи 3 Такты задачи 3
Содержимое стэка
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 2
Содержимое процессора
Время поступления задачи 4 Приоритет задачи 2 Такты задачи 3
Идет 5 такт
Содержимое стэка
Время поступления задачи 4 Приоритет задачи 2 Такты задачи 2
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 2
Содержимое процессора
Время поступления задачи 5 Приоритет задачи 3 Такты задачи 3
Идет 6 такт
Содержимое стэка
Время поступления задачи 4 Приоритет задачи 2 Такты задачи 2
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 2
Содержимое процессора
Время поступления задачи 5 Приоритет задачи 3 Такты задачи 2
Идет 7 такт
Содержимое стэка
Время поступления задачи 4 Приоритет задачи 2 Такты задачи 2
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 2
Содержимое процессора
Время поступления задачи 5 Приоритет задачи 3 Такты задачи 1
Идет 8 такт
Содержимое стэка
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 2
Содержимое процессора
Время поступления задачи 4 Приоритет задачи 2 Такты задачи 2
Идет 9 такт
Содержимое стэка
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 2
Содержимое процессора
Время поступления задачи 4 Приоритет задачи 2 Такты задачи 1
Идет 10 такт
Содержимое процессора
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 2
Идет 11 такт
Содержимое процессора
Время поступления задачи 3 Приоритет задачи 1 Такты задачи 1
```

Также следует отметить, что результаты показаны по состоянию на конец такта. Таким образом, если задача ставится в очередь и сразу попадает на выполнение, то факт ее пребывания в очереди не отображается. Не отображается и восстановление прерванной задачи из стека, и новое ее прерывание на этом же такте при наличии более приоритетных задач в очереди.

3.6. Контрольные вопросы

1. Что такое стек?
2. Назовите основные способы реализации стека. Сравните их.
3. Что такое очередь?
4. Назовите основные способы реализации очереди. Сравните их.
5. Что такое дек?
6. В каких случаях целесообразно применять стек, а каких очередь?
7. Какие достоинства и недостатки использования статических и динамических структур данных?

4. Лабораторная работа «Хеширование данных»

4.1. Цель работы

Целью работы является изучение методов хеширования данных и получение практических навыков реализации хеш-таблиц.

4.2. Задание на лабораторную работу

Составить хеш-функцию в соответствии с заданным вариантом и проанализировать ее. При необходимости доработать хеш-функцию. Используя полученную хеш-функцию разработать на языке программирования высокого уровня программу, которая должна выполнять следующие функции:

– создавать хеш-таблицу;

– добавлять элементы в хеш-таблицу;

– просматривать хеш-таблицу;

– искать элементы в хеш-таблице по номеру сегмента/по ключу;

– выгружать содержимое хеш-таблицы в файл для построения гистограммы в MS Excel, или в аналогичном подходящем ПО;

– удалять элементы из хеш-таблицы;

– в программе должна быть реализована проверка формата вводимого ключа;

– при удалении элементов из хеш-таблицы, в программе должен быть реализован алгоритм, позволяющий искать элементы, вызвавшие коллизию с удаленным;

– в программе должен быть реализован алгоритм, обрабатывающий ситуации с переполнением хеш-таблицы.

Метод разрешения коллизий выбрать в соответствии с заданным вариантом.

Варианты задания приведены в табл. 4.

Таблица 4

№ вар.	Формат ключа	Количество сегментов	Метод хеширования (разрешения коллизий)
1	цццБцц	1500	Открытое хеширование
2	цццБцц	2000	Линейное опробование
3	цццБцц	3000	Квадратичное опробование
4	цццБцц	2500	Двойное хеширование

Окончание табл. 4

№ вар.	Формат ключа	Количество сегментов	Метод хеширования (разрешения коллизий)
5	БцццББ	1500	Открытое хеширование
6	БцццББ	2000	Линейное опробование
7	БцццББ	3000	Квадратичное опробование
8	БцццББ	2500	Двойное хеширование
9	БццццБ	1500	Открытое хеширование
10	БццццБ	2000	Линейное опробование
11	БццццБ	3000	Квадратичное опробование
12	БццццБ	2500	Двойное хеширование
13	ББцццББ	1500	Открытое хеширование
14	ББцццББ	2000	Линейное опробование
15	ББцццББ	3000	Квадратичное опробование
16	ББцццББ	2500	Двойное хеширование
17	ццББццц	1500	Открытое хеширование
18	ццББццц	2000	Линейное опробование
19	ццББццц	3000	Квадратичное опробование
20	ццББццц	2500	Двойное хеширование
21	цБББББц	1500	Открытое хеширование
22	цБББББц	2000	Линейное опробование
23	цБББББц	3000	Квадратичное опробование
24	цБББББц	2500	Двойное хеширование
25	цББББББ	1500	Открытое хеширование

Где «ц» – это цифра 0...9; «Б» – это большая буква латиницы A...Z.

4.3. Порядок выполнения работы

1) выбрать вариант задания в соответствии с требованиями;

2) изучить теоретический материал, изложенный в учебном пособии;

3) составить хеш-функцию, используя заданный формат ключа и количество сегментов;

4) проанализировать полученную хеш-функцию, определив распределение коллизий по сегментам таблицы. При концентрации коллизий на определенных сегментах таблицы изменить хеш-функцию с целью устранения этой концентрации;

5) используя полученную хеш-функцию и заданный метод разрешения коллизий разработать на языке программирования вы-

сокого уровня программу, реализующую хеш-таблицу и заданный перечень функции;

6) написать отчет о работе;

7) защитить отчет.

К защите отчета по лабораторной работе, включающую демонстрацию работы программы, необходимо сформировать два или более различных ключа заданного формата, по которым разработанная хеш-функция вычисляет одинаковые адреса (происходит коллизия), и быть готовым продемонстрировать результат разрешения этой коллизии. Также целесообразно сформировать файл с частично заполненной хеш-таблицей. В этом случае программа должна осуществлять загрузку хеш-таблицы из файла.

4.4. Содержание отчета

Отчет должен содержать:

1) титульный лист;

2) цель работы;

3) вариант задания;

4) описание хеш-функции;

5) результаты анализа хеш-функции;

6) листинг программы, реализующей хеш-таблицу и заданный перечень функции;

7) выводы по работе.

4.5. Пример выполнения работы

Предположим, что необходимо выполнить следующий вариант задания:

№ вар.	Формат ключа	Количество сегментов	Метод хеширования (разрешения коллизий)
26	цБББББ	1500	Линейное опробование

Используя заданный формат ключа и количество сегментов в таблице, составляем хеш-функцию. Ключ представляет собой строку, содержащую буквы и цифры в определенных позициях. Попробуем подход, основанный на сложении кодов символов, составляющих строку-ключ.

```
h(key) = (int)key[1] + (int)key[2] + (int)key[3] +
         (int)key[4] +(int)key[5] + (int)key[6]
```

Оценим область определения значений полученной хеш-функции. Эта область должна совпадать с количеством сегментов хеш-таблицы.

Минимальное значение хеш-функция принимает при минимальном значении слагаемых, то есть если строка-ключ содержит символы с минимальным кодом. Это будет символ «0» с кодом 48 и символ «A» с кодом 65. Тогда минимальное значение полученной хеш-функции будет 373, и начальные сегменты хеш-таблицы никогда не будут использованы. Следовательно, необходимо привести хеш-функцию к виду, когда ее минимальное значение давало бы минимальный номер сегмента таблицы, равный 0, то есть вычесть из нее число 373.

Теперь проверим максимальное значение хеш-функции, которое она принимает при максимальном значении слагаемых, то есть если строка-ключ содержит символы с максимальным кодом. Это будет символ «9» с кодом 57 и символ «Z» с кодом 90. Тогда максимальное значение полученной хеш-функции (с учетом вычитания 373) будет 134, и конечные сегменты хеш-таблицы никогда не будут использованы. Следовательно, необходимо привести хеш-функцию к виду, когда ее максимальное значение давало бы максимальный номер сегмента таблицы, равный 1500, то есть умножить ее на число 11. Теперь хеш-функция принимает значения в диапазоне от 0 до 1474, что практически соответствует допустимым номерам сегментов хеш-таблицы.

Теперь проанализируем качество полученной хеш-функции. Для этого осуществим экспериментальное исследование количества коллизий, приходящихся на сегменты хеш-таблицы.

Экспериментальное исследование проводится следующим образом:

1) формируются случайным образом ключи заданного формата в количестве, превышающем количество сегментов хеш-таблицы минимум в 2...3 раза;

2) для каждого сформированного ключа вычисляется хеш-функция, и подсчитывается, сколько раз вычислялся адрес того или иного сегмента хеш-таблицы.

После получения экспериментальных данных осуществляем их анализ и делаем экспертные оценки равномерности распределения коллизий по сегментам хеш-таблицы. При равномерном распреде-

лении коллизий считаем, что сформированная хеш-функция является удачной и можно программно реализовывать хеш-таблицу на ее основе. В противном случае хеш-функция считается неудачной, необходима ее доработка и повторное проведение экспериментального исследования. Доработка и повторное экспериментальное исследование повторяются до тех пор, пока не будет получена удачная хеш-функция.

Приведенный порядок экспериментальных исследований является достаточно трудоемким. Его можно автоматизировать и сделать более эффективным, если разработать специальную программу, позволяющую получать экспериментальные данные в виде текстового файла. Для упрощения оценки равномерности распределения коллизий по сегментам хеш-таблицы целесообразно визуализировать полученные данные с помощью диаграммы. Это можно сделать с использованием MS Excel, если в него импортировать полученный текстовый файл и построить диаграмму.

Текст программы на языке C++ для формирования экспериментальных данных приведен ниже.

```
void HashFunctionSt(Keys *&GeneratedKeys, Keys
*MassiveOfSegments,Keys *&BadKeys)
{
 int hashNum=0;
 Keys *Tmp = GeneratedKeys;
 Keys *TmpBadKeys = BadKeys;
 uint16_t minKeysCode = 0;
 bool flag = false;
 for (int i = 0; i < 6; i++)
 {
  if (TypeOfKey[i] == 'ц')
   minKeysCode = minKeysCode + 48;
  else minKeysCode = minKeysCode + 65;
 }
 uint16_t maxKeysCode = 0;
 for (int i = 0; i < 6; i++)
 {
  if (TypeOfKey[i] == 'ц')
   maxKeysCode = maxKeysCode + 57;
  else maxKeysCode = maxKeysCode + 90;
 }
 uint16_t coeffOfExpansion = NumOfSegments / (maxKeysCode-
minKeysCode);
```

```
for (int i = 0; i < NumOfKeys; i++)
{
 for (int j = 0; j < 6; j++)
  hashNum = hashNum+Tmp->keys[j];
 hashNum = (hashNum - minKeysCode) * coeffOfExpansion;
 MassiveOfSegments[hashNum].numOfH++;
 for (int j = 0; j <= MaxNumOfCollisions; j++)
 {
  if (MassiveOfSegments[hashNum].keys[0] == NULL)
  {
   for (int y = 0; y < 6; y++)
    MassiveOfSegments[hashNum].keys[y] = Tmp->keys[y];
   break;
  }
  else if (j == MaxNumOfCollisions - 1)
  {
   if (BadKeys == NULL)
   {
    TmpBadKeys = new Keys;
    for (int y = 0; y < 6; y++)
     TmpBadKeys->keys[y] = Tmp->keys[y];
    TmpBadKeys->next = NULL;
    BadKeys = TmpBadKeys;
    break;
   }
   else
   {
    TmpBadKeys->next = new Keys;
    TmpBadKeys = TmpBadKeys -> next;
    TmpBadKeys->next = NULL;
    for (int y = 0; y < 6; y++)
     TmpBadKeys->keys[y] = Tmp->keys[y];
    break;
   }
  }
  else
  {
   hashNum += StepOfLinearSampling;
   if (hashNum > NumOfSegments)
    hashNum -= NumOfSegments;
  }
```

```
  }
  Tmp = Tmp->next;
  hashNum = 0;
 }
}
```

Поскольку в примере используется метод закрытого хеширования с методом разрешения коллизий «линейным опробованием», по истечению максимального количества коллизий `MaxNumOfCollisions`, ключ может так и не попасть в сегментную сетку. В таком случае он заноситься в список «плохих» ключей `BadKeys`.

Результатом работы этой программы является массив, который содержит столько элементов, сколько сегментов в хеш-таблице. Каждая элемент содержит число `numOfH`, показывающее, сколько раз вычислялся адрес соответствующего сегмента, а также первый записанный ключ `keys`. Данный массив, содержащий количество попаданий ключей в сегмент, следует выгрузить в текстовый файл.

Полученный текстовый файл импортируем в MS Excel с помощью команды «Файл/Открыть». При этом задействуется мастер импорта текстов. Затем строим диаграмму с помощью команды «Вставка/Диаграмма». При этом задействуется мастер диаграмм. Результат приведен ниже.

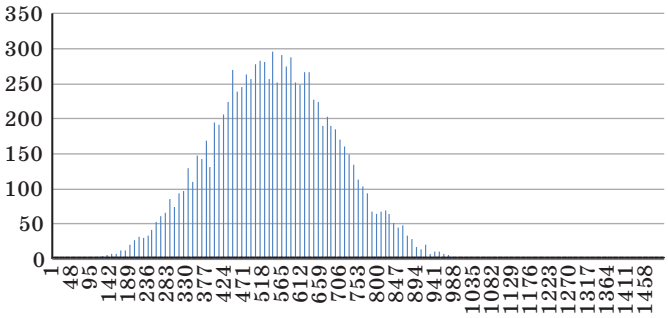

Рис. 4.1. Результаты экспериментального анализа хеш-функции. Сгенерировано 10 000 ключей на 1500 сегментов

Анализ этой диаграммы показывает, что коллизии распределены неравномерно (наблюдается концентрация коллизий в середине хеш-таблицы) и, следовательно, созданная хеш-функция является неудачной.

Поскольку требуется алгоритм, распределяющий ключи по сегментам равномерно. В качестве примера, будем вычислять адрес сегмента путем сложения возведенных в квадрат кодов символов ключа. Поскольку полученный адрес будет превышать размер хеш-таблицы, итоговое число необходимо нормировать относительно количества сегментов в таблице.

```
void HashFunction(Keys *&GeneratedKeys, Keys *MassiveOfSegments,
Keys *BadKeys)
{
 int hashNum = 0;
 Keys *Tmp = GeneratedKeys;
 Keys *TmpBadKeys = BadKeys;
 for (int i = 0; i < NumOfKeys; i++)
 {
  for (int j = 0; j < 6; j++)
  {
   if (j)
    hashNum = hashNum + (j * 3) * Tmp->keys[j];
   else hashNum = Tmp -> keys[j];
  }
  hashNum = hashNum % NumOfSegments;
  MassiveOfSegments[hashNum].numOfH++;
  for (int j = 0; j < MaxNumOfCollisions; j++)
  {
   if (MassiveOfSegments[hashNum].keys[0] == NULL)
   {
    for (int y = 0; y < 6; y++)
     MassiveOfSegments[hashNum].keys[y] = Tmp->keys[y];
    break;
   }
   else if (j == MaxNumOfCollisions - 1)
   {
    if (BadKeys == NULL)
    {
     TmpBadKeys = new Keys;
     for (int y = 0; y < 6; y++)
      TmpBadKeys->keys[y] = Tmp->keys[y];
     TmpBadKeys->next = NULL;
     BadKeys = TmpBadKeys;
     break;
```

```
      }
      else
      {
       TmpBadKeys->next = new Keys;
       TmpBadKeys = TmpBadKeys->next;
       TmpBadKeys->next = NULL;
       for (int y = 0; y < 6; y++)
        TmpBadKeys->keys[y] = Tmp->keys[y];
       break;
      }
     }
     else
     {
      hashNum += StepOfLinearSampling;
      if (hashNum > NumOfSegments)
       hashNum -= NumOfSegments;
     }
    }
    Tmp = Tmp->next;
    hashNum = 0;
  }
}
```

Результаты эксперимента с новой хеш-функции представлены ниже. Для наглядности количество сгенерированных ключей было равно 10 000.

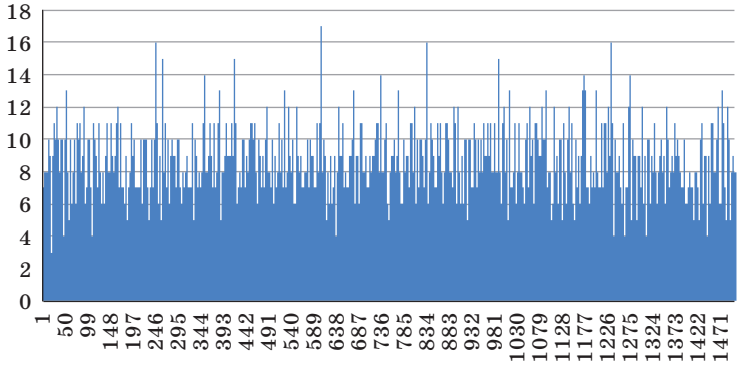

Рис. 4.2. Результаты экспериментального анализа доработанной хеш-функции

Анализ этой диаграммы показывает, что коллизии распределены более равномерно и, следовательно, новую хеш-функцию можно считать приемлемой.

Теперь можно приступать к реализации хеш-таблицы и программы, которая выполняет требуемые действия с этой хеш-таблицей. На этом этапе уже необходимо учитывать заданный метод хеширования (разрешения коллизий).

4.6. Контрольные вопросы

1. Что такое хеш-функция?
2. Каким свойством обладает идеальная хеш-функция?
3. Какие требования предъявляются к хеш-функциям?
4. Что такое открытое хеширование?
5. Что такое закрытое хеширование?
6. Каким образом осуществляется удаление данных при закрытом хешировании и почему?
7. Какие методы повторного хеширования существуют? Сравните их.
8. Каким образом определяется адрес при линейном опробовании?
9. Каким образом определяется адрес при квадратичном опробовании?
10. Каким образом определяется адрес при двойном хешировании?
11. Для чего используется алгоритм хеширования? Его плюсы и минусы.

5. Лабораторная работа «Алгоритмы сортировки»

5.1. Цель работы

Целью работы является изучение алгоритмов внутренней сортировки и получение практических навыков их использования, и анализа их сложности.

5.2. Задание на лабораторную работу

Использовать неупорядоченный массив A, содержащий n целочисленных элементов. Величина n определяется по согласованию с преподавателем. Дополнительно в программе должны быть реализованы следующие функции:

1) поиск элемента либо по его порядковой позиции, либо по его содержимому;

2) добавление/удаление элемента с последующей пересортировкой последовательности;

3) в программе должен быть реализован подсчет количества сравнений и перестановок, при осуществлении сортировки.

Варианты задания приведены в табл. 5.

Таблица 5

№ вар.	Задание	Алгоритм сортировки
1	Найти количество различных чисел среди элементов массива	Подсчетом
2	–″–	Простым включением
3	–″–	Шелла
4	–″–	Простым извлечением
5	–″–	Шейкерная
6	–″–	Чётно-нечётная
7	–″–	Расческой
8	–″–	Быстрая (Хоара)
9	–″–	Слиянием
10	–″–	Распределением
11	Найти количество повторяющихся чисел среди элементов массива	Подсчетом

Окончание табл. 5

№ вар.	Задание	Алгоритм сортировки
12	–"–	Простым включением
13	–"–	Шелла
14	–"–	Простым извлечением
15	–"–	Шейкерная
16	–"–	Чётно-нечётная
17	–"–	Быстрая (Хоара)
18	–"–	Слиянием
19	–"–	Распределением
20	–"–	Расческой
21	Найти k-е по порядку число среди элементов массива	Подсчетом
22	–"–	Простым включением
23	–"–	Шелла
24	–"–	Простым извлечением
25	–"–	Шейкерная
26	–"–	Чётно-нечётная
27	–"–	Расческой
28	–"–	Быстрая (Хоара)

5.3. Порядок выполнения работы

1) выбрать вариант задания из подраздела в соответствии с требованиями;

2) изучить теоретический материал, изложенный в учебном пособии;

3) разработать и реализовать на языке программирования высокого уровня алгоритм, выполняющий требования задания;

4) подсчитать количество выполненных операций сравнения и перестановок элементов массива;

5) подсчитать сложность алгоритма

6) написать отчет о работе;

7) защитить отчет.

5.4. Содержание отчета

Отчет должен содержать:

1) титульный лист;
2) цель работы;
3) вариант задания;
4) листинг программы, реализующей алгоритм;
5) контрольный пример;
6) количество выполненных операций сравнения и перестановок элементов массива;
7) временную и пространственную сложность алгоритма;
8) выводы по работе.

5.5. Пример выполнения работы

Предположим, что необходимо выполнить следующий вариант задания:

№ вар.	Задание	Алгоритм сортировки
29	Найти k-е по порядку число среди элементов массива	Пузырьком

Для нахождения k-го по порядку числа среди элементов массива A необходимо упорядочить, то есть отсортировать, массив, а затем найти число, располагающееся в элементе массива с индексом k. Алгоритмов сортировки выбираем в соответствии с вариантом задания.

Однако программу, реализующую классический алгоритм сортировки, необходимо модифицировать для того, чтобы осуществлять подсчет выполняемых операций сравнения и перестановок элементов массива. Модификация осуществляется путем ввода в текст программы специальных переменных-счетчиков, которые увеличиваются при выполнении соответствующих операций. В примере ниже счетчиком операций сравнений является `numOfCompare`, а счетчиком операций перестановки – `numOfReshuffle`.

```
uint16_t mas[]={ 5, 7, 9, 1, 3, 8 };
uint16_t tmp = 0;
uint16_t numOfReshuffle = 0;
```

```
uint16_t numOfCompare = 0;
bool needtosort = false;
for (uint16_t i = 0; i < sizeof(mas) / sizeof(uint16_t);
i++)
{
 for(uint16_t j=0; j < sizeof(mas) / sizeof(uint16_t) - i;
j++)
  if (mas[j] > mas[j + 1])
  {
   numOfCompare++;
   tmp = mas[j+1];
   mas[j + 1] = mas[j];
   mas[j] = tmp;
   numOfReshuffle++;
   needtosort = true;
  }
  else
  {
   numOfCompare++;
}
 if (!needtosort)
  break;
 needtosort = false;
}
delete[] mas;
```

Теперь подсчитываем теоретические сложности алгоритма.

Разработанный алгоритм использует следующие данные:

– один массив размерностью n = `sizeof(mas)/sizeof(uint16 _ t);`

– две переменные целого типа и одну логическую переменную.

Значит, пространственная сложность алгоритма определяется следующим образом:

$$v = + n * C_{\text{uint16\_}t} + 2 * C_{\text{uint16\_}t} + C_{\text{bool}}$$

где $C_{\text{uint16\_}t}$ – константа, характеризующая объем памяти, отводимый под переменную беззнакового 16-тиразрядного целого типа.

Теоретическая пространственная сложность алгоритма составляет:

$$V(n) = O(v) = O(\max(O(n * C_{\text{uint16\_}t}),$$
$$O(2 * C_{\text{uint16\_}t}),\ O(C_{\text{bool}}))) =$$
$$O(\max(O(n),\ O(1),\ O(1))) = O(n).$$

Теоретическую временную сложность алгоритма определяем на основе анализа текста программы, реализующей данный алгоритм. Для начала рассчитаем теоретическую временную алгоритма сортировки:

$$T_{\text{Sort}} = O(\max(O(K_1),\ O(n^2 * K_2))) = O(\max(O(1),$$

$$O(n^2))) = O(n^2),$$

где K_1 – операции сравнения, присваивания, используемые в алгоритме и имеющие временную сложность «1», K_2 – цикличные операции, занимающие в наихудшем случае n^2 шагов.

5.6. Контрольные вопросы

1. Что такое сортировка?
2. Какие существуют виды сортировки, определяемые местоположением сортируемых данных? Какие особенности каждого из этих видов?
3. Какова временная и пространственная сложность алгоритмов внутренней сортировки?
4. Почему теоретическая оценка быстродействия алгоритма быстрой сортировки носит вероятностный характер?
5. В чем отличительная особенность алгоритма сортировки распределением и в чем основной недостаток вырожденной сортировки распределением?
6. Что такое временная и пространственная сложность алгоритма?
7. Что такое теоретическая временная и теоретическая пространственная сложность алгоритма?

6. Лабораторная работа «АВЛ-деревья поиска»

6.1. Цель работы

Целью работы является изучение деревьев поиска и получение практических навыков их использования.

6.2. Задание на лабораторную работу

Разработать на языке программирования высокого уровня программу, которая должна выполнять следующие функции:

– добавлять элементы в сбалансированное дерево поиска;

– удалять элементы из сбалансированного дерева поиска;

– искать элементы в дереве поиска с выводом количества шагов, за которое осуществляется поиск;

– выводить дерево на экран (любым способом доступным для восприятия);

– выводить список, соответствующий обходу вершин, в соответствии с вариантом задания;

– осуществлять операцию, заданную в таблице 6.

Количество элементов и порядок их ввода при создании сбалансированного дерева поиска определяется по согласованию с преподавателем.

Таблица 6

№ вар.	Задание	Порядок обхода
1	Вывести глубину самого верхнего листа дерева (maxh) и самого нижнего листа (ов) дерева (minh), а так же их значения. Удалить элементы и сбалансировать дерево. Процедуру повторять до тех пор, пока не выполнится условие maxh = minh	Прямой
2	Вывести глубину самого верхнего листа дерева (maxh) и самого нижнего листа (ов) дерева (minh), а так же их значения. Удалить элементы и сбалансировать дерево. Процедуру повторять до тех пор, пока не выполнится условие maxh = minh	Обратный
3	Вывести глубину самого верхнего листа дерева (maxh) и самого нижнего листа (ов) дерева (minh), а так же их значения. Удалить элементы и перебалансировать дерево. Процедуру повторять до тех пор, пока не выполнится условие maxh = minh	Симметричный

Продолжение табл. 6

№ вар.	Задание	Порядок обхода
4	Вывести глубину самого верхнего листа дерева (maxh) и самого нижнего листа (ов) дерева (minh), а так же их значения. Удалить элементы и перебалансировать дерево. Процедуру повторять до тех пор, пока не выполнится условие maxh = minh	В ширину
5	Подсчитать среднее арифметическое всех листов дерева. Затем вычесть из каждого листа данное значение, затем обойти дерево еще раз и удалить все элементы, делящиеся без остатка на 3 и перестроить дерево с учетом изменений	Прямой
6	Подсчитать среднее арифметическое всех листов дерева. Затем вычесть из каждого листа данное значение, затем обойти дерево еще раз и удалить все элементы, делящиеся без остатка на 3 и перестроить дерево с учетом изменений	Обратный
7	Подсчитать среднее арифметическое всех листов дерева. Затем вычесть из каждого листа данное значение, затем обойти дерево еще раз и удалить все элементы, делящиеся без остатка на 3 и перестроить дерево с учетом изменений	Симметричный
8	Подсчитать среднее арифметическое всех листов дерева. Затем вычесть из каждого листа данное значение, затем обойти дерево еще раз и удалить все элементы, делящиеся без остатка на 3 и перестроить дерево с учетом изменений	В ширину
9	Заменить в дереве все отрицательные элементы на их абсолютные величины, после чего уменьшить в 2 раза все элементы, делящиеся без остатка на 4. Затем обойти дерево повторно в соответствии с вариантом, у каждой обойденной вершины изменить значение $k_i = \|k_i - k_{i-1}\|$, где i – индекс текущего элемента в порядке обхода	Прямой
10	Заменить в дереве все отрицательные элементы на их абсолютные величины, после чего уменьшить в 2 раза все элементы, делящиеся без остатка на 4. Затем обойти дерево повторно в соответствии с вариантом, у каждой обойденной вершины изменить значение $k_i = \|k_i - k_{i-1}\|$, где i – индекс текущего элемента в порядке обхода	Обратный
11	Заменить в дереве все отрицательные элементы на их абсолютные величины, после чего уменьшить в 2 раза все элементы, делящиеся без остатка на 4. Затем обойти дерево повторно в соответствии с вариантом, у каждой обойденной вершины изменить значение $k_i = \|k_i - k_{i-1}\|$, где i – индекс текущего элемента в порядке обхода	Симметричный

Продолжение табл. 6

№ вар.	Задание	Порядок обхода
12	Заменить в дереве все отрицательные элементы на их абсолютные величины, после чего уменьшить в 2 раза все элементы, делящиеся без остатка на 4. Затем обойти дерево повторно в соответствии с вариантом, у каждой обойденной вершины изменить значение $k_i = \|k_i - k_{i-1}\|$, где i – индекс текущего элемента в порядке обхода	В ширину
13	Обойти дерево в соответствии с вариантом и найти максимальную глубину листа. Добавлять элементы в дерево, пока высота всех листов дерева не будет одинакова, но не превысит изначально найденного максимального значения	Прямой
14	Обойти дерево в соответствии с вариантом и найти максимальную глубину листа. Циклично добавлять элементы в дерево, пока высота всех листов дерева не будет одинакова, но не превысит изначально найденного максимального значения	Обратный
15	Обойти дерево в соответствии с вариантом и найти максимальную глубину листа. Циклично добавлять элементы в дерево, пока высота всех листов дерева не будет одинакова, но не превысит изначально найденного максимального значения	Симметричный
16	Обойти дерево в соответствии с вариантом и найти максимальную глубину листа. Циклично добавлять элементы в дерево, пока высота всех листов дерева не будет одинакова, но не превысит изначально найденного максимального значения	В ширину
17	Циклично удалять каждый нечётный узел дерева (Корень – 1, нечетный) и перестраивать дерево, беря элементы по порядку обхода. Данную процедуру выполнять, пока не останется последний элемент. На каждой итерации выводить дерево	Прямой
18	Циклично удалять каждый нечётный узел дерева (Корень – 1, нечетный) и перестраивать дерево, беря элементы по порядку обхода. Данную процедуру выполнять, пока не останется последний элемент. На каждой итерации выводить дерево	Обратный
19	Циклично удалять каждый нечётный узел дерева (Корень – 1, нечетный) и перестраивать дерево, беря элементы по порядку обхода. Данную процедуру выполнять, пока не останется последний элемент. На каждой итерации выводить дерево	Симметричный

Окончание табл. 6

№ вар.	Задание	Порядок обхода
20	Циклично удалять каждый нечётный узел дерева (Корень – 1, нечетный) и перестраивать дерево, беря элементы по порядку обхода. Данную процедуру выполнять, пока не останется последний элемент. На каждой итерации выводить дерево	В ширину
21	Для каждого узла дерева *A*, вывести список потомков в порядке обхода.	Прямой
22	Для каждого узла дерева *A*, вывести список потомков в порядке обхода.	Обратный
23	Для каждого узла дерева *A*, вывести список потомков в порядке обхода.	Симметричный
24	Для каждого узла дерева *A*, вывести список потомков в порядке обхода.	В ширину

6.3. Порядок выполнения работы

1) выбрать вариант задания в соответствии с требованиями;
2) изучить теоретический материал, изложенный в учебном пособии;
3) разработать и реализовать на языке программирования высокого уровня алгоритм, выполняющий требования задания;
4) написать отчет о работе;
5) защитить отчет.

6.4. Содержание отчета

Отчет должен содержать:
1) титульный лист;
2) цель работы;
3) вариант задания;
4) листинг программы, реализующей алгоритм;
5) изображения с демонстрацией работы программы;
6) выводы по работе.

6.5. Пример выполнения работы

Алгоритм добавления нового элемента в сбалансированное дерево будет состоять из следующих трех основных шагов:

1) поиск по дереву.

2) вставка элемента в место, где закончился поиск, если элемент отсутствует.

3) восстановление сбалансированности.

Первый шаг необходим для того, чтобы убедиться в отсутствии элемента в дереве, а также найти такое место вставки, чтобы после вставки дерево осталось упорядоченным.

Третий шаг представляет собой обратный проход по пути поиска: от места добавления к корню дерева. По мере продвижения по этому пути корректируются показатели сбалансированности проходимых вершин, и производится балансировка там, где это необходимо. Добавление элемента в дерево никогда не требует более одного поворота.

Алгоритм удаления элемента из сбалансированного дерева будет выглядеть так:

1) поиск по дереву.

2) удаление элемента из дерева.

3) восстановление сбалансированности дерева (обратный проход).

Первый шаг необходим, чтобы найти в дереве вершину, которая должна быть удалена.

Третий шаг представляет собой обратный проход от места, из которого взят элемент для замены удаляемого, или от места, из которого удален элемент, если в замене не было необходимости.

Операция удаления может потребовать перебалансировки всех вершин вдоль обратного пути к корню дерева, то есть порядка $\log n$ вершин.

Восстановление сбалансированности требует движения от листьев к корню, поэтому будем хранить пути от листьев к корню дерева.

```
struct Tree
{
 int elem; //содержимое узла
 int height; //высота узла
 Tree *Prev; //указатель на предка
 Tree *Left; //указатель на меньшего потомка
 Tree *Right; //указатель на большего потомка
};
```

Примеры функций добавления элемента и балансировки:

```
void BalanceTree(Tree *&Root, Tree *&NewElem)
{
 Tree *MainTree = NewElem;
 while (true)
 {
  MainTree->height = max(GetHeight(MainTree->Left),
GetHeight(MainTree->Right)) + 1;
  if (GetHeight(MainTree->Right) - GetHeight(MainTree->Left) == 2)
  {
    if (GetHeight(MainTree->Right->Right) - GetHeight(MainTree-
>Right->Left) == 0)
    LeftRotate(MainTree);
  else if (GetHeight(MainTree->Right->Right) - GetHeight(MainTree-
>Right->Left) == 1)
     LeftRotate(MainTree);
   else
    BigLeftRotate(MainTree);
  }
  else if (GetHeight(MainTree->Right) - GetHeight(MainTree->Left)
== -2)
  {
     if (GetHeight(MainTree->Left->Right) - GetHeight(MainTree-
>Left->Left) == 0)
    RightRotate(MainTree);
   else if (GetHeight(MainTree->Left->Right) - GetHeight(MainTree-
>Left->Left) == -1)
    RightRotate(MainTree);
   else
    BigRightRotate(MainTree);
  }
  if (MainTree->Prev)
   MainTree = MainTree->Prev;
  else break;
 }
 Root = MainTree;
}

void AddTreeElem(Tree *&Root, Tree *NewElem)
{
 Tree *MainTree = Root;
 if (!Root)
```

```
 {
  Root = new Tree;
  Root->elem = NewElem->elem;
  Root->height = 1;
  Root->Prev = NULL;
  Root->Left = NULL;
  Root->Right = NULL;
  MainTree = Root;
 }
 else
 {
  while (true)
  {
   if (NewElem->elem < MainTree->elem)
   {
    if (MainTree->Left)
     MainTree = MainTree->Left;
    else
    {
     MainTree->Left = NewElem;
     NewElem->Prev = MainTree;
     break;
    }
   }
   else
   {
    if (MainTree->Right)
     MainTree = MainTree->Right;
    else
    {
     MainTree->Right = NewElem;
     NewElem->Prev = MainTree;
     break;
    }
   }
  }
  BalanceTree(Root, MainTree);
 }
}
```

Функция удаления элемента из дерева:

```
void DeleteTreeElem(Tree *&Root, Tree *&DeleteElem)
{
 Tree *MainTree = Root;
 Tree *TmpElem = NULL;
 Tree *OneMoreTmp = NULL;
 int counter = 0;
 while (true)
 {
  if (MainTree)
  {
   if (DeleteElem->elem == MainTree->elem)
   {
    if (MainTree->Prev || GetHeight(MainTree) > 1)
    {
     OneMoreTmp = MainTree;
     if (MainTree->Left || MainTree->Right)
     {
      if (MainTree->Right)
       MainTree = MainTree->Right;
      TmpElem = OneMoreTmp->Prev;
      while (MainTree->Left)
       MainTree = MainTree->Left;
      if (MainTree->Prev)
      {
       if (MainTree->Prev->Right == MainTree)
        MainTree->Prev->Right = NULL;
       else
        MainTree->Prev->Left = NULL;
      }
      if (TmpElem)
      {
       if (TmpElem->Right)
       {
        if (TmpElem->Right == OneMoreTmp)
         TmpElem->Right = MainTree;
        else
         TmpElem->Left = MainTree;
```

```
    }
    else
     TmpElem->Left = MainTree;
   }
   if (MainTree->Prev != OneMoreTmp)
    TmpElem = MainTree->Prev;
   else TmpElem = MainTree;
   MainTree->Prev = OneMoreTmp->Prev;
   if (OneMoreTmp->Right != MainTree)
   {
    MainTree->Right = OneMoreTmp->Right;
    if (MainTree->Right)
     MainTree->Right->Prev = MainTree;
   }
   else
    MainTree->Right = NULL;
   if (OneMoreTmp->Left != MainTree)
   {
    MainTree->Left = OneMoreTmp->Left;
    if (MainTree->Left)
     MainTree->Left->Prev = MainTree;
   }
   else
    MainTree->Left = NULL;
   BalanceTree(Root, TmpElem);
  }
  else
  {
   TmpElem = MainTree->Prev;
   if (TmpElem)
   {
    if (TmpElem->Right)
    {
     if (TmpElem->Right == MainTree)
      TmpElem->Right = NULL;
     else
      TmpElem->Left = NULL;
    }
    else
```

```
        TmpElem->Left = NULL;
       }
       BalanceTree(Root, TmpElem);
      }
      cout << "elem was deleted" << endl;
      ShowTree(counter, Root, 0);
      cout << "Num of elements = " << counter<<endl;
      _getch();
      return;
     }
     else
     {
      Root = NULL;
      cout << "Tree is Empty Now" << endl;
      _getch();
      return;
     }
    }
    else if (DeleteElem->elem > MainTree->elem)
     MainTree = MainTree->Right;
    else
     MainTree = MainTree->Left;
   }
   else
   {
    cout << "No such elem in tree!" << endl;
    _getch();
    return;
   }
  }
}
```

Таким образом, реализуется два первых пункта задания на лабораторную работу. Осталось реализовать функцию поиска в дереве и функцию, заданную вариантом задания.

Поиск в дереве осуществляется классическим способом. Единственным отличием здесь будет добавлением в функцию поиска операции подсчета количества вершин дерева, по которым проходим в процессе поиска, и вывода этого количества по окончанию поиска.

Задание по варианту реализуется с помощью указанного метода обхода. Обход можно реализовывать рекурсивным или нерекурсивным способом по выбору.

6.6. Контрольные вопросы

1. Что такое дерево? Что такое высота и степень дерева?
2. Назовите способы реализации бинарного дерева. Сравните их.
3. Какие способы обхода бинарного дерева существуют?
4. Что такое дерево поиска?
5. Какова временная сложность алгоритма поиска в дереве поиска?
6. Что такое идеально сбалансированное дерево?
7. Что такое дерево, сбалансированное по АВЛ?
8. Какие операции применяют для восстановления сбалансированности дерева?

7. Лабораторная работа «Алгоритмы на графах»

7.1. Цель работы

Целью работы является изучение графов и получение практических навыков их использования.

7.2. Задание на лабораторную работу

Разработать на языке программирования высокого уровня программу, которая должна выполнять функцию, в соответствии с вариантом задания. Варианты задания приведены в таблице 7 (формулировки задач приведены после таблицы).

Количественные параметры варианта задания определяются по согласованию с преподавателем.

Таблица 7

№ вар.	Задача	Представление графа
1	1	Матрица смежности
2	1	Матрица инцидентности
3	1	Список ребер
4	2	Матрица смежности
5	2	Матрица инцидентности
6	2	Список ребер
7	3	Матрица смежности
8	3	Матрица инцидентности
9	3	Список ребер
10	4	Матрица смежности
11	4	Матрица инцидентности
12	4	Список ребер
13	5	Матрица смежности
14	5	Матрица инцидентности
15	5	Список ребер
16	6	Матрица смежности
17	6	Матрица инцидентности

Окончание табл. 7

№ вар.	Задача	Представление графа
18	6	Список ребер
19	7	Матрица смежности
20	7	Матрица инцидентности
21	7	Список ребер
22	8	Матрица смежности
23	8	Матрица инцидентности
24	8	Список ребер

Задача 1.

Прямоугольное изображение задано графом, так что каждая вершина является пикселем изображения, ребра соединяют соседние пиксели по горизонтали и вертикали. Требуется:

1) проверить соответствует ли описанный граф изображению (нет пропущенных пикселей);

2) Найти и «закрасить» замкнутые контура на изображении.

Задача 2.

На случай экстренной ситуации на предприятии имеется список сотрудников и список оповещения вида: (фамилия оповещающего, фамилия оповещаемого). Один оповещающий может оповестить за день 3 человек. Определить:

1) полон ли список оповещения, то есть будут ли оповещены все сотрудники;

2) сколько людей будет оповещено за K дней.

Задача 3.

Задача Эйлеровых циклов. Дан граф, требуется определить, возможно ли пройти из указанной вершины по всем ребрам графа, и, если возможно, указать данный путь.

Задача 4.

Найти все возможные пути между двумя вершинами в графе, не пересекающиеся по:

1) ребрам;

2) вершинам.

Задача 5.

Решить 2 задачи:

1) задача о минимальном вершинном покрытии;

2) задача о минимальном рёберном покрытии.

Задача 6.

Составить программу для нахождения произвольного разбиения N студентов на M команд, численность которых отличается не более чем в 2 раза, если известно, что в любой команде должны быть студенты, обязательно не знакомые друг с другом. Круг знакомств задается графом, где вершина – это студент, а ребро отображает его знакомство с другим студентом. Решить задачи:

1) определить наименьшее количество команд, на которое можно разбить множество студентов;

2) проверить возможность разбиения множества студентов на заданное количество команд.

Задача 7.

В помещении присутствует N человек. Некоторые из них знакомы между собой. Проверить, можно ли перезнакомить людей между собой? (Незнакомые люди могут познакомиться только через общего знакомого).

Задача 8.

Дано N колец сцепленных между собой. Удалить минимальное количество колец так, чтобы получилась цепочка.

7.3. Порядок выполнения работы

1) выбрать вариант задания в соответствии с требованиями;

2) изучить теоретический материал, изложенный в учебном пособии;

3) разработать и реализовать на языке программирования высокого уровня алгоритм, выполняющий требования задания;

4) написать отчет о работе;

5) защитить отчет.

7.4. Содержание отчета

Отчет должен содержать:

1) титульный лист;

2) цель работы;

3) вариант задания;

4) листинг программы, реализующей алгоритм;

5) контрольный пример с изображением графа;

6) выводы по работе.

7.5. Пример выполнения работы

Предположим, что необходимо выполнить следующий вариант задания:

№ вар.	Задача	Представление графа
25	9	Матрица смежности

Задача 9.

Дана схема алгоритма. Определить по схеме достижимость конечной вершины из начальной.

Дополнительные указания: схема алгоритма – ориентированный граф, возможно содержащий циклы. Воспользоваться волновым алгоритмом.

Для решения поставленной задачи необходимо построить граф, используя заданное представление, а затем применить рекомендуемый алгоритм.

Пусть задана схема алгоритма (рис. 7.1). На основе нее строим матрицу смежности (рис.7.2). Теперь можно воспользоваться волновым алгоритмом, взяв за исходную вершину – вершину «Начало». По окончании работы алгоритма остается проверить, была ли помечена как пройденная вершина «Конец». Если вершина «Конец» была помечена, то можно сделать вывод, что конечная вершина достижима, в противном случае – конечная вершина не достижима.

Следует отметить, что граф может иметь циклы, однако для волнового алгоритма их наличие не является опасным.

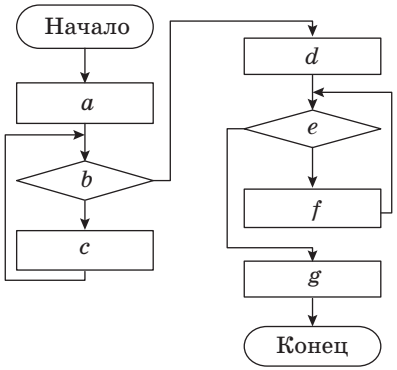

Рис. 7.1

	Начало	*a*	*b*	*c*	*d*	*e*	*f*	*g*	Конец
Начало	0	1	0	0	0	0	0	0	0
a	0	0	1	0	0	0	0	0	0
b	0	0	0	1	1	0	0	0	0
c	0	0	1	0	0	0	0	0	0
d	0	0	0	0	0	1	0	0	0
e	0	0	0	0	0	0	1	1	0
f	0	0	0	0	0	1	0	0	0
g	0	0	0	0	0	0	0	0	1
Конец	0	0	0	0	0	0	0	0	0

Рис. 7.2

7.6. Контрольные вопросы

1. Что такое граф?
2. Назовите основные способы реализации графа. Сравните их.
3. Какие алгоритмы обхода графа существуют? Какова их временная сложность?
4. Какие алгоритмы нахождения кратчайшего пути в графе существуют? Какова их временная сложность?
5. Какие переборные алгоритмы нахождения кратчайшего пути в графе существуют? Какова их временная сложность?
6. Какие алгоритмы нахождения минимального остовного дерева графа существуют? Какова их временная сложность?

Литература

1. *Ключарев А. А., Матьяш В. А., Щекин С. В.* Структуры и алгоритмы обработки данных. – СПб.: ГУАП, 2003*.

2. *Ахо А.* Построение и анализ вычислительных алгоритмов / А. Ахо, Дж. Хопкрофт, Дж. Ульман; пер. А. О. Слисенко; ред. Ю. В. Матиясевич. – М.: Мир, 1979. – 536 с*.

3. *Ахо А.* Структуры данных и алгоритмы / А. Ахо, Д. Э. Хопкрофт, Д. Ульман; пер. с англ. и ред. А. А. Минько. – М.: Вильямс, 2016. – 620 с.

4. *Вирт Н.* Алгоритмы и структуры данных. Новая версия для Оберона + CD / Н. Вирт; пер. Д. Б. Подшивалов. – 2-е изд., испр. – М.: ДМК Пресс, 2012. – 272 с*.

5. *Грин Д. Х.* Математические методы анализа алгоритмов / Д. Х. Грин, Д. Кнут. – М.: Мир, 1987. – 120 с*.

6. *Гудман С.* Введение в разработку и анализ алгоритмов / С. Гудман, С. Хидетниеми; пер.: Ю. Б. Котов, Л. В. Сухарева, Л. В. Ухов; ред. В. В. Мартынюк. – М.: Мир, 1981. – 366 с*.

7. *Демидович Е. М.* Основы алгоритмизации и программирования. Язык СИ / Е. М. Демидович. – 2-е изд., испр. и доп. – СПб.: БХВ-Петербург, 2008. – 440 с*.

8. *Кнут Д.* Искусство программирования: в 3 т. Т. 1. Основные алгоритмы / Д. Кнут; ред. Ю. В. Козаченко. – 3-е изд. – М.: Вильямс, 2014. – 720 с*.

9. *Колдаев В. Д.* Основы алгоритмизации и программирования / В. Д. Колдаев. – М.: ФОРУМ-ИНФРА-М, 2006. – 413 с*.

10. *Кормен, Т.* Алгоритмы: построение и анализ / Т. Кормен и др.; пер.: И. В. Красиков, Н. А. Орехова, В. Н. Романов. – 2-е изд. – М.: Вильямс, 2012. – 1290 с*.

11. *Лафорте Р.* Структуры данных и алгоритмы в JAVA / Р. Лафорте; пер. Е. Матвеев. – 2-е изд. – СПб.: Питер, 2017. – 704 с.

12. *Макконнелл Дж.* Анализ алгоритмов. Активный обучающий подход / Джефри Макконнелл; пер.: С. Кулешов, С.Ландо. – 2-е доп. изд. – М.: Техносфера, 2009. – 416 с.

13. *Матьяш В. А., Путилов В. А., Фильчаков В. В., Щекин С. В.* Структуры и алгоритмы обработки данных. – Апатиты: КФ ПетрГУ, 2000. – 80 с*.

14. *Оре О.* Теория графов / пер. с англ. И. Н. Врублевская; ред. Н. Н. Воробьев. – 2-е изд., стер. – М.: Наука, 1980. – 336 с*.

15. *Рейнгольд Э., Нивергельт Ю., Део Н.* Комбинаторные алгоритмы. Теория и практика. – М.: Мир, 1980*.

16. *Седжвик Р.* Алгоритмы на C++: анализ структуры данных, сортировка, поиск, алгоритмы на графах / Р. Седжвик; конс. К. Ван Вик. – М.: Вильямс, 2014. – 1056 с*.

17. *Сибуя М.* Алгоритмы обработки данных / ред.: В. В. Панферов; пер.: Э. К. Николаева. – М.: Мир, 1986. – 218 с*.

18. *Скиена С.* Алгоритмы. Руководство по разработке. – 2-е изд.: пер. с англ. – СПб.: БХВ-Петербург, 2017. – 720 с.

19. *Успенский В. А., Семенов А. Л.* Теория алгоритмов: основные открытия и приложения. – М.: Наука, 1987*.

20. *Харари Ф.* Теория графов / Ф. Харари; ред.: Г. П. Гаврилов; пер.: В. П. Козырев. – М.: Мир, 1973. – 300 с*.

* – Имеются в наличии в библиотеке ГУАП.

СОДЕРЖАНИЕ

Общие указания 3
1. *Лабораторная работа* «Анализ сложности алгоритмов» 4
2. *Лабораторная работа* «Линейные и циклические списки» 11
3. *Лабораторная работа* «Стек и очередь» 19
4. *Лабораторная работа* «Хеширование данных» 33
5. *Лабораторная работа* «Алгоритмы сортировки» 43
6. *Лабораторная работа* «АВЛ-деревья поиска» 48
7. *Лабораторная работа* «Алгоритмы на графах» 59
Литература 64

Учебное издание

Матьяш Валерий Анатольевич,
Рогачев Сергей Александрович

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Учебно-методическое пособие

Публикуется в авторской редакции.
Компьютерная верстка *И. Н. Мороз*

Сдано в набор 06.12.17. Подписано к печати 20.12.17.
Формат $60\times84^{1}/_{16}$. Усл. печ. л. 3,83.
Уч.-изд. л. 4,12. Тираж 50 экз. Заказ № 531.

Редакционно-издательский центр ГУАП
190000, Санкт-Петербург, Б. Морская ул., 67

ДЛЯ ЗАМЕТОК