ЛАБОРАТОРНАЯ РАБОТА № 4
«ХЕШИРОВАНИЕ ДАННЫХ»

1.1 Цель работы

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

1.2 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 хэш-таблицы, в программе должен быть реализован алгоритм, позволяющий искать элементы, вызвавшие коллизию с удаленным;
- в программе должен быть реализован алгоритм, обрабатывающий ситуации с переполнением хэш-таблицы.

Метод разрешения коллизий выбрать в соответствии с заданным вариантом.

Варианты задания приведены в таблице 4.

Таблица 1

№ 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
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	ББццББ	1500	Открытое хеширование
14	ББццББ	2000	Линейное опробование
15	ББццББ	3000	Квадратичное опробование
16	ББццББ	2500	Двойное хеширование
17	ццББцц	1500	Открытое хеширование
18	ццББцц	2000	Линейное опробование
19	ццББцц	3000	Квадратичное опробование
20	ццББцц	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.

1.3 Порядок выполнения работы

1) выбрать вариант задания в соответствии с требованиями;
2) изучить теоретический материал;
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

высокого 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.

1.4 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.

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

```
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;
  }
}
```

Результатом работы этой программы является массив, который содержит столько элементов, сколько сегментов в хеш-таблице. Каждая строка содержит число, показывающее, сколько раз вычислялся адрес соответствующего сегмента. Данный массив 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.

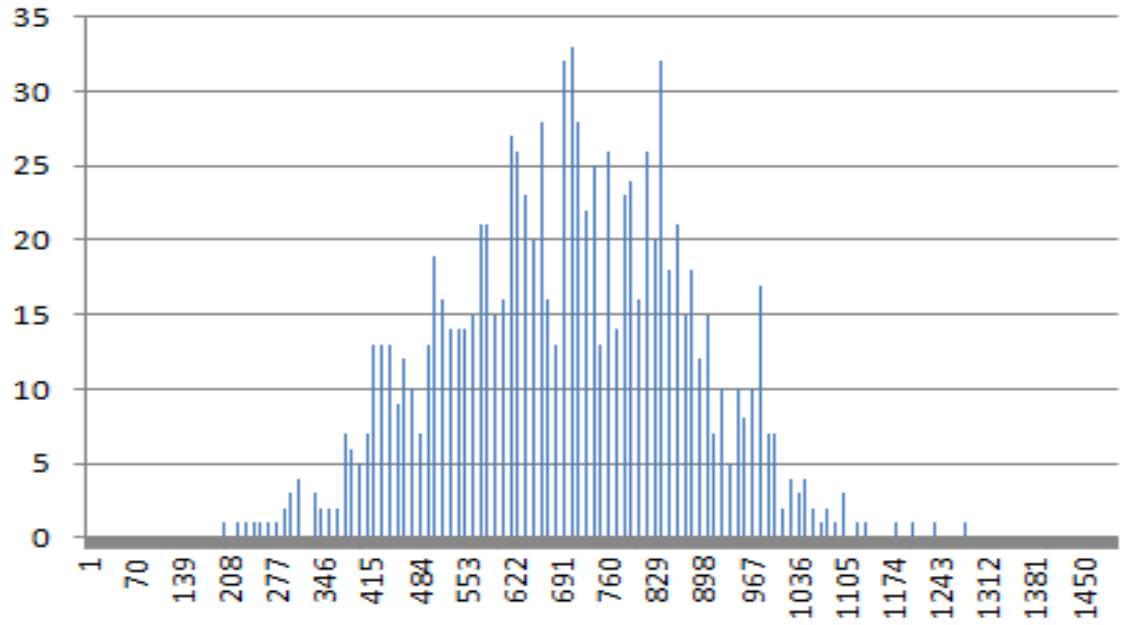

Рисунок 1. Результаты экспериментального анализа хеш-функции. Сгенерировано 1000 ключей на 1500 сегментов.

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 полученное при такой операции число превышает количество сегментов. Итоговую сумму квадратов возьмём по модулю от числа сегментов.

```
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 увеличено до 4000.

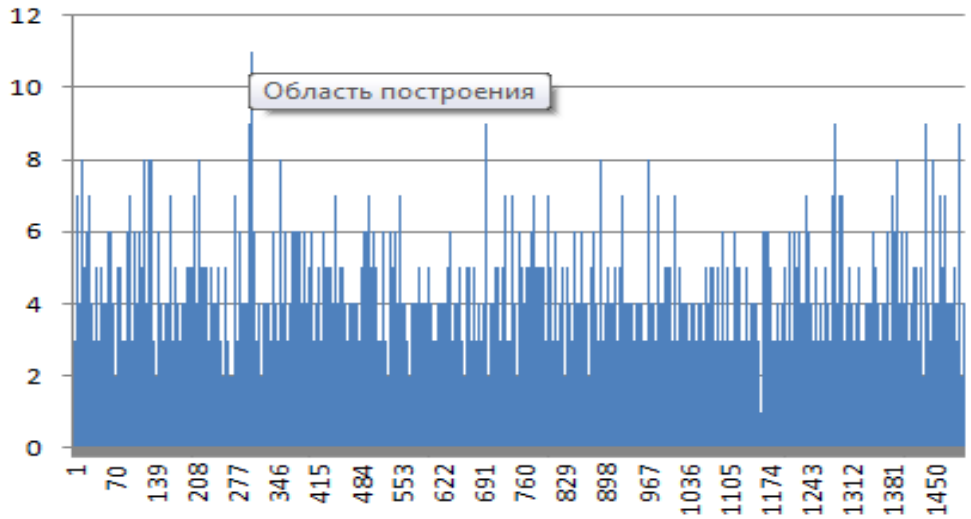

Рисунок 2. Результаты экспериментального анализа доработанной хеш-функции

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).

1.6 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.