федеральное государственное автономное образовательное учреждение высшего профессионального образования «Казанский (Приволжский) федеральный университет»

Институт информационных технологий и интеллектуальных систем

Семестровая работа

дисциплина “Алгоритмы и структуры данных”

Фильтр Блума

Выполнила студентка
1 курса группы 11-002
Мартазова О.А.

Руководитель
Зиятдинов М.Т.

Казань, 2021 г.

Фильтр Блума (англ. Bloom filter) — это вероятностная структура данных, придуманная Бёртоном Блумом в 1970 году, позволяющая проверять принадлежность элемента к множеству. Ответ может быть:

- элемент точно не принадлежит множеству,
- элемент возможно принадлежит множеству.

При этом существует возможность получить ложноположительное срабатывание (элемента в множестве нет, но структура данных сообщает, что он есть), но не ложноотрицательное. Фильтр Блума может использовать любой объём памяти, заранее заданный пользователем, причем больше элементов, тем выше вероятность ложного срабатывания.
Поддерживается операция добавления новых элементов в множество, но не удаления существующих.

Основной принцип устройства

Фильтр Блума представляет собой битовый массив из m бит. Изначально, когда структура данных хранит пустое множество, все m бит обнулены.
Также определены k различных хеш-функций h1…hk, каждому элементу хеш-функция сопоставляет число от 1 до m.
Для добавления элемента e необходимо записать единицы на каждую из позиций h1(e)…hk(e) битового массива.

Возможные операции

Добавление элемента в Фильтр Блума:

1. Вычисление значения хеш-функций для добавляемого элемента;
2. Использование этих значений для установления единиц на конкретных битах в Фильтре (значение хеш-функции - индекс битового массива).

Проверка принадлежности элемента *e* к множеству элементов в Фильтре:

1. Проверка состояния k битов h1(e)…hk(e) массива;
2. Если хотя бы один из этих битов равен нулю, элемент точно не принадлежит множеству.
3. Если все биты равны единице, структура данных сообщает, что есть вероятность принадлежности элемента к множеству.

   Есть 2 возможных варианта:

   - элемент действительно принадлежит множеству
   - все биты были установлены при добавлении других элементов. Такая ситуация называется ложноположительное срабатывание (false-positive result).

После добавления элемента в фильтр Блума его нельзя удалить из фильтра, потому что есть вероятность, что другие элементы, соответствующие этим k битам, удаляются.

Оптимальные значения параметров фильтра

Отличительной особенностью фильтра Блума является возможность изменения коэффициента ложноположительного срабатывания фильтра.
Вычисление ложно положительного коэффициента ошибок:
Для любого элемента *x* каждая хеш-функция h_i назначает ему одно из мест в битовом массиве с равной вероятностью

$$\Pr(h_i(x) = p) = \frac{1}{m}, \quad p = 1 \dots m,$$

.
Вероятность того, что в некоторый *p*-й бит не будет записана единица во время операции вставки очередного элемента равна

$$\Pr(h_1(x) \neq p \cap \dots \cap h_k(x) \neq p) = \Pr(h_i(x) \neq p)^k = \left(1 - \frac{1}{m}\right)^k.$$

После вставки n различных элементов $x_1, \dots, x_n$ в изначально пустой фильтр Блума вероятность примерно равна

$$\Pr\left(\forall i \in \{1 \dots k\},\ \forall j \in \{1 \dots n\},\ h_i(x_j) \neq p\right) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$

Ложноположительное срабатывание состоит в том, что для некоторого элемента *q*, не равного ни одному из вставленных, все *k* бит с номерами $h_i(q)$ окажутся ненулевыми, и фильтр Блума ошибочно ответит, что *q* входит в множество вставленных элементов.
Вычисление ложно положительного коэффициента ошибок вычисляется по формуле:

$$p = (1 - e^{-kn/m})^k$$

где n — предполагаемое количество элементов, хранящихся в фильтре, m - размер массива, и k - количество хеш-функций.
Оптимальный размер массива в битах:

$$m = -\frac{n \ln p}{(\ln 2)^2}$$

Вероятность ложноположительного срабатывания уменьшается с ростом *m* и увеличивается с ростом *n*. Для фиксированных *m* и *n* оптимальное число *k* хеш-функций:

$$k = \frac{m}{n} \ln 2$$

Оценка временной сложности

Сложность операции вставки элемента в фильтр Блума зависит от используемых хеш-функций. Во время вставки вычисляется значения k хеш-функций для каждого добавляемого элемента. Временная сложность O(k).
Добавление элемента в фильтр осуществляется за счет изменения k бит в массиве. Доступ происходит по индексу - значению хеш-функции, сложность константная. Проверка принадлежности элемента к множеству также выполняется за время O(k). При фиксированном k временная сложность вставки элемента и проверки принадлежности элемента к множеству с помощью фильтра Блума — O(1).

Фильтр Блума не хранит сами элементы, а только ключи к ним. Фильтр использует битовый массив, который допускает хеш-коллизию. Без коллизий фильтр Блума не был бы эффективным по памяти.

Графики

График зависимости времени выполнения всей структуры (в мкс) от количества входных данных:

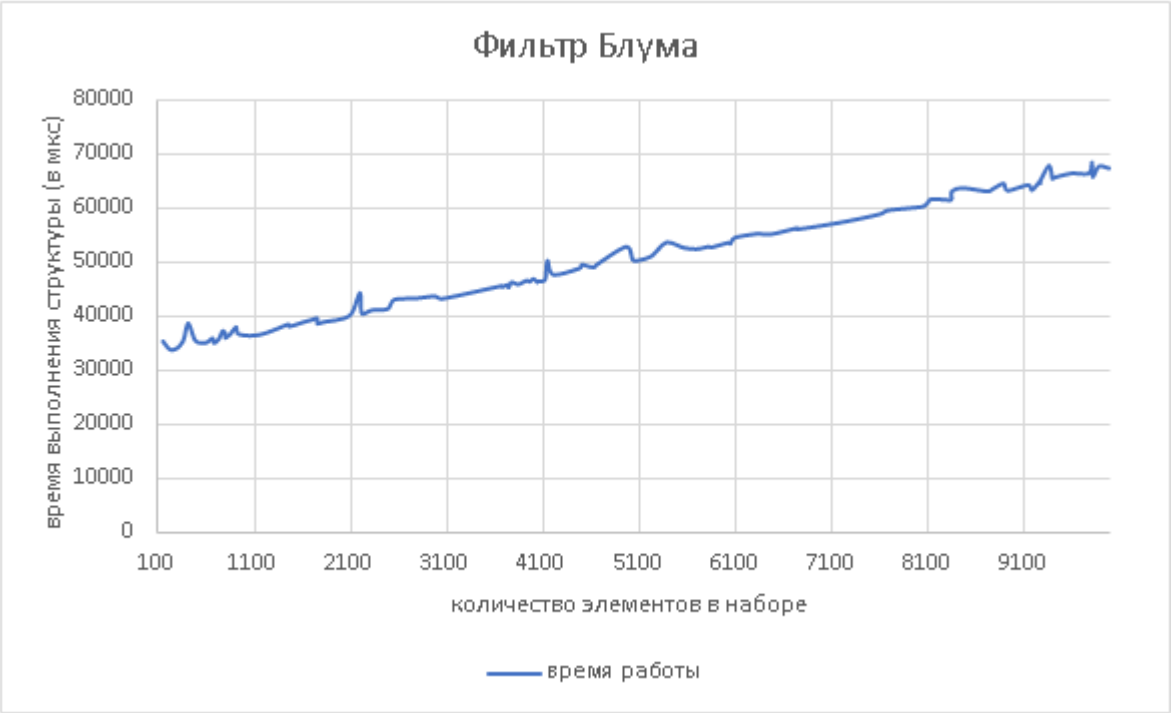
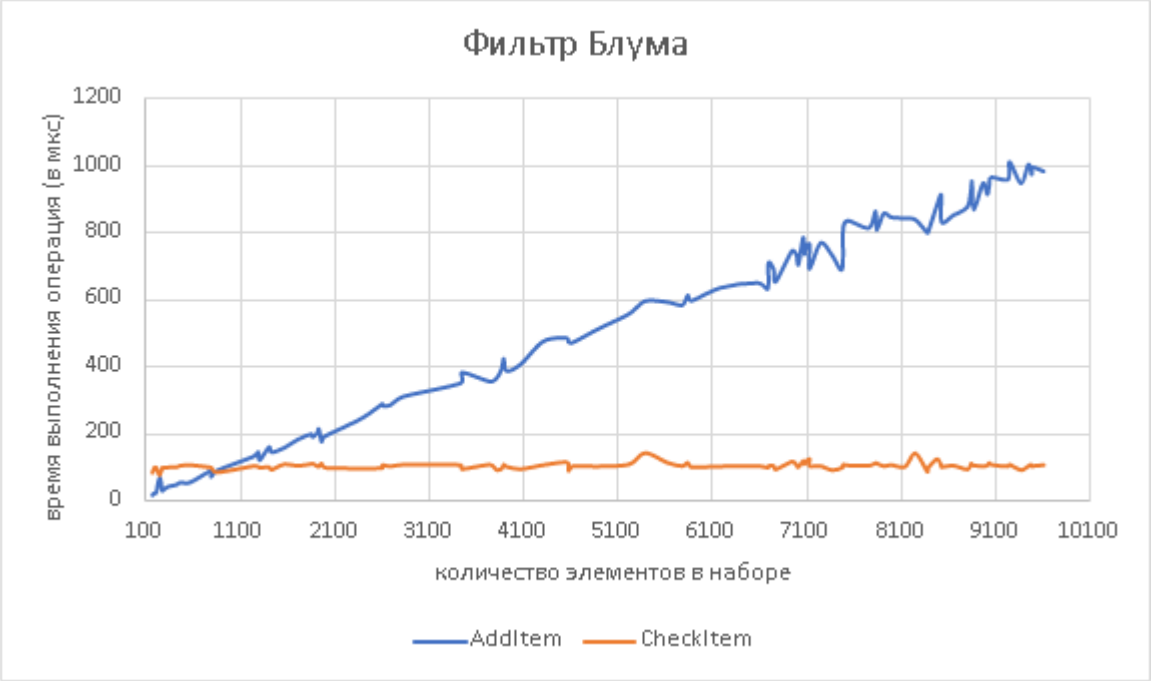


График зависимости времени выполнения операций добавления элемента и проверки на наличие в множестве (в мкс) от количества входных данных:



На графике представлено время выполнения операции вставки для всего набора элементов, а не для одного элемента

```
while (count < n) {
    cin >> x;
    addItem(x, bf);
    count ++;
}
```

Поэтому сложность на графике O(n).

Данные, изображенные на графике, отличаются от теоретических (прослеживаются перепады) из-за сторонних процессов компьютера.

Вывод

Фильтр Блума полезен для предотвращения излишнего выполнения задач, требующих интенсивных вычислений. Алгоритм проверяет, что элемент совершенно точно не входит в множество.

Преимущества фильтра:

- По сравнению с хеш-таблицами, более эффективное использование памяти.
- Фильтр Блума может представлять универсальное множество всех возможных элементов. В этом случае все биты в его битовом массиве равны единице.
- Константная временная сложность операций.

Недостатки фильтра:

- Отсутствует операция удаления элемента из фильтра
- Вероятность ложноположительного срабатывания

Применение фильтра Блума:

- Веб-браузер Google Chrome, используемый для использования фильтра Блума для выявления вредоносных URL-адресов.

- Google BigTable, Apache HBase и Apache Cassandra, и Postgresql используют фильтры Блума, чтобы уменьшить количество запросов на диск для несуществующих строк или столбцов.
- Фильтры Блума широко применялись в контексте распределенных баз данных, сетей, биоинформатики и других областей, где обычные хеш-таблицы занимают слишком много места.

Использование фильтра Блума - это быстрый и экономичный способ отслеживания объектов, когда нет необходимости хранить эти объекты. Используя достаточно большой битовый массив и правильное количество хеш-функций, можно снизить количество ложных срабатываний.

Список используемой литературы

Викиконспекты
Implementing a simple, high-performance Bloom filter in C++
Bloom Filter implementation in C++
Bloom Filters – Introduction and Implementation
All about Bloom Filters
Что такое Фильтр Блума

Приложение

Код программы

Задача: проверить числа на принадлежность множеству чисел. Возможные результаты: “элемент точно не принадлежит множеству”, “элемент возможно принадлежит множеству”
На вход подается:

- число n - количество элементов в множестве
- n чисел множества (натуральные числа и ноль)
- числа для проверки на принадлежность множеству (натуральные числа и ноль)
- -1 — признак конца ввода

```
#include <iostream>
#include <list>
#include <cmath>
using namespace std;

const int N = 100000;
struct BloomFilter {
  int numberOfCells;
  list<int(*)(int)> hashFunctions;
```

```
  bool* cell;
};

void addItem(int const& x, BloomFilter bf) {
  for (auto it = bf.hashFunctions.begin(); it != bf.hashFunctions.end();
++it) {
     bf.cell[(*it)(x) % bf.numberOfCells] = true;
  }
}

bool checkItem(int const& x, BloomFilter bf) {
  bool intInSet = true;
  for (auto it = bf.hashFunctions.begin(); it != bf.hashFunctions.end();
++it) {
     if (!bf.cell[(*it)(x) % bf.numberOfCells]) {
        intInSet = false;
        break;
     }
  }
  return intInSet;
}

int hash1(int x) {
  return x;
}

int hash2(int x) {
  return (3*x + 5);
}

int hash3(int x) {
  return (5*x + 7);
}

int hash4(int x) {
  return (7*x + 11);
}

int hash5(int x) {
  return (13*x + 17);
}

int main() {
  list<int(*)(int)> hashFunctions;
  hashFunctions.push_back(hash1);
  hashFunctions.push_back(hash2);
  hashFunctions.push_back(hash3);
  hashFunctions.push_back(hash4);
  hashFunctions.push_back(hash5);

  BloomFilter bf;
  bf.hashFunctions = hashFunctions;
  // enter the number of items in the filter
  int n;
  cin >> n;
```

```
    bf.numberOfCells = round (5 * n / log(2));
    bool cells[N];
    for (int i = 0; i < bf.numberOfCells; i ++) {
        cells[i] = false;
    }
    bf.cell = cells;
    int count = 0;
    // enter numbers into the filter
    // possible input: positive integer and 0
    int x;
    while (count < n) {
       cin >> x;
       addItem(x, bf);
       count ++;
    }
    cout << "Bloom filter has filled" << endl;
    cout << "Check if an item was present in set" << endl;

    // testing numbers against the bloom filter
    // possible input: positive integer and 0
    // -1 is the end of input
    int y;
    cin >> y;
    while (y != -1) {
       if (!checkItem(y, bf))
          cout << y << " surely is not present" << endl;
       else cout << y << " was probably present" << endl;
       cin >> y;
    }
    return 0;
}
```

Ссылка на репозиторий [github]

Инструкция по запуску тестов

1. Генератор ста наборов данных

Запуск программы Data.java в папке data_set

Во время выполнения программы создается файл input.txt с данными для 100 наборов

2. Запуск программы time.cpp в папке Semestrovka1

time.cpp для каждого набора создает файл output.txt с одним набором данных. Этот набор подается в качестве входных данных в code_.exe

3. Выполнение code_.cpp

code_.cpp это измененный код структуры code.cpp, засекает время выполнения операций структуры.

Для операции addItem создается файл add.txt.

Для операции checkItem создается файл check.txt.

В файлы записывается размер входных данных и среднее время выполнения операции.

4. Для повторного запуска файла time.cpp

необходимо стереть содержимое файлов add.txt и check.txt. Иначе данные в файлах запишутся ниже.

Таблицы полученных значений времени работы в зависимости от размера данных
table.xlsx
Все входные данные (100 наборов)
input.txt