SET1 Семестр 2. Задача A3. Взломщик!

Фролов-Буканов Виктор Дмитриевич БПИ-228

13 февраля 2024

1 Условие задачи

Задание A3* (10 баллов) Взломщик!

Для хеширования строковых ключей, которые могут содержать строчные/прописные латинские буквы и цифры, используется следующая **полиномиальная хеш-функция** (ее описание также представлено здесь):

```
const int p = 31;
long long hash = 0, p_pow = 1;
for (size_t i=0; i < s.length(); ++i) {
  hash += (s[i] - 'a' + 1) * p_pow;
  p_pow *= p;
}
```

Вам предлагается предпринять попытку "взлома" этой хеш-функции, то есть подобрать такие строковые ключи, для которых значения этой функции совпадают:

1. Опишите правила, по которым могут быть получены строки, дающие коллизию для этой хеш-функции.
2. Разработайте генератор таких строк и приложите исходный код. В отдельном файле представьте пример работы генератора для $N = 2000$ различных строк.

Ограничений на используемые языки программирования в этом задании нет.

2 Принцип посторения строк

Опишем принцип построения строк с одинаковыми хэшами на примере строки *"acb"*. Согласно правилу взятия хэша букве 'a' будет сопоставлено 1 , 'b' -> 2, 'c' -> 3, то есть можно сказать что строке *"acb"* сопоставляется многочлен $f(x) = 1 + 3x + 2x^2$, а значением хэша этой строки будет $f(31) = 2016$. Наша цель подобрать такие многочлены $f_1(x), f_2(x), \ldots$, что $f_1(31) = f_2(31) = \cdots = f(31)$. При этом коэфициенты многочленов должны быть из того множества, которое нам дали по условию (то есть эти числа должны обратно раскодироваться в строчные/прописные латинские буквы или цифры). Назовем это множество буквой $\mathcal{A} = \{-48, \ldots, -39, -31, \ldots, -6, 1, \ldots, 26\}$ (проверить корректность составления множества можно вызвав функцию *browseAllVariants* из кода ниже)
Мы будем восстанавливать многочлены той же степени. Возьмем произвольный многочлен в точке 31 и приравняем его к значению заданного хэша: $k_1 + 31k_2 + 31^2k_3 = 2016 \iff 31k_2 + 31^2k_3 = 2016 - k_1$. Теперь подберем такое $k_1 \in \mathcal{A}$, что $31|k_1$. Например, $k_1 = -30$. Тогда $31k_2 + 31^2k_3 = 2046 \iff k_2 + 31k_3 = 2046/31 = 66 \iff 31k_3 = 66 - k_2$. Теперь подберем $k_2 \in \mathcal{A}, 31|k_2$. Пусть $k_2 = 4 \Rightarrow 31k_3 = 62 \Rightarrow k_3 = 2$. Таким алгоритмом мы восстановили все k_i. В данном случае мы получили многочлен $f_1(x) = -30 + 4x + 2x^2, f_1(31) = 2016$, который сопоставляется строке "Bdb". Заметим, что на каждом шаге алгоритма для каждого коэффициента мы можем подобрать несколько значений из заданного множества $\mathcal{A}$, что мы и будем делать, а дальше будем рекурсивно вызывать решающую функцию для восстановления следующего коэффициента. В такой реализации главное не забыть в конце,

когда мы восстановили строку, проверить, не восстановили ли мы исходную строку (то есть в примере выше мы должны проверить, не восстановли ли мы в ходе работы алгоритма строку *"acb"*). Ниже представлена реализация полученного алгоритма (метод generate) с применением техники *backtrack*.

3 Исходный код

main.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <unordered_set>
#include <algorithm>
#include <chrono>
#include <random>

std::vector<std::string> ans;
std::unordered_set<std::string> set;
std::ofstream fout;

const int p = 31;

long long hash(const std::string &str) {
  long long hash = 0, pPow = 1;
  for (auto i = 0; i < str.size(); ++i) { // NOLINT
    hash += (str[i] - 'a' + 1) * pPow;
    pPow *= p;
  }
  return hash;
}

void writeAns() {
  for (auto &el : ans) {
    fout << el << '\n';
  }
}

void checkEqualHash() {
  long long initialHash = hash(ans[0]);
  for (auto i = 1; i < ans.size(); ++i) {
    if (hash(ans[i]) != initialHash) {
      fout << "Different hashes!" << '\n';
      return;
    }
  }

  fout << "All hashes are equal!" << '\n';
}

void generate(std::string &str, std::string &curr, long long currHash)
  { // NOLINT
  if (curr.size() == str.size() - 1) {
    curr.push_back(static_cast<char>(currHash - 1 + 'a'));
    if (curr != str) {
      ans.emplace_back(curr);
      set.insert(curr);
    }
    curr.pop_back();
```

```
    return;
  }
  for (auto i = -48; i != -38; ++i) {
    if ((currHash - i) % p == 0) {
      char sym = static_cast<char>(i - 1 + 'a');
      curr.push_back(sym);
      generate(str, curr, (currHash - i) / p);
      curr.pop_back();
    }
  }
  for (auto i = -p; i != -5; ++i) {
    if ((currHash - i) % p == 0) {
      char sym = static_cast<char>(i - 1 + 'a');
      curr.push_back(sym);
      generate(str, curr, (currHash - i) / p);
      curr.pop_back();
    }
  }

  for (auto i = 1; i != 27; ++i) {
    if ((currHash - i) % p == 0) {
      char sym = static_cast<char>(i - 1 + 'a');
      curr.push_back(sym);
      generate(str, curr, (currHash - i) / p);
      curr.pop_back();
    }
  }

}

void browseAllVariants() {
  std::string digits = "0123456789";
  std::string lower = "qwertyuiopasdfghjklzxcvbnm";
  std::string upper = lower;

  for (auto i = 0; i < lower.size(); ++i) {
    upper[i] -= 32;
  }

  std::vector<int> nums;

  for (auto i = 0; i < lower.size(); ++i) {
    nums.emplace_back(lower[i] - 'a' + 1);
    nums.emplace_back(upper[i] - 'a' + 1);
  }

  for (auto i = 0; i < digits.size(); ++i) { // NOLINT
    nums.emplace_back(digits[i] - 'a' + 1);
  }

  std::sort(nums.begin(), nums.end());

  for (auto &el : nums) {
    std::cout << el << '\n';
  }
}

int main() {
  auto startTime = std::chrono::high_resolution_clock::now();
```

```
  fout.open("hacked.txt");
  std::unordered_set<std::string> processedStrings;
  std::string initial = "0123456789"; // length = 12 abcdefghijkl
  int count = 0;
  unsigned int seed = 1;
  while (count != 100) {
    auto rnd = std::default_random_engine{seed};
    auto tmp = initial;
    std::shuffle(tmp.begin(), tmp.end(), rnd);
    if (processedStrings.contains(tmp)) {
      ++seed;
      continue;
    }
    std::string starter;
    generate(tmp, starter, hash(tmp));
    ++count;
    processedStrings.insert(tmp);
    fout << "Source string = " << tmp << '\n';
    writeAns();
    fout << "Total amount of strings = " << ans.size() << '\n';
    if (ans.size() == set.size()) {
      fout << "All strings are different!" << '\n';
    } else {
      fout << "Some strings are equal!" << '\n';
    }
    checkEqualHash();
    fout << "##########################################\n";
    ans.clear();
    set.clear();
    std::cout << count << '\n';
  }
  fout.close();
  auto endTime = std::chrono::high_resolution_clock::now();
  std::cout << "Total time to hack = " << std::chrono::duration_cast<
      std::chrono::seconds>(endTime - startTime).count() << " seconds";

  return 0;
}
```

4 Замечание

По условию надо было взломать хэш-функцию для 2000 различных строк, но в таком случае получается очень большой файл (примерно на 330МБ), что не позволяет мне его загрузить в мой github-репозиторий, так что лишь с этой целью, я число 2000 заменил на 100