Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Дискретный анализ»

Студент: А. Ю. Кварацхелия
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-21

Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №1

Задача: Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

Вариант алгоритма: Поиск одного образца при помощи алгоритма Апостолико-Джанкарло.

Вариант алфавита: Числа в диапазоне от 0 до $2^{32} - 1$

1 Описание

Основная суть алгоритма заключается в улучшении оценки времени работы алгоритма Бойера-Мура, в стандартном виде использующем $6m$ сравнений, где m - длина исходного текста. Алгоритм Апостолико-Джанкарло позволяет сократить количесвто сравнений до $2m$.

Важно отметить, что эффективный алгоритм Апостолико-Джанкарло по-прежнему использует «Правило хорошего суффикса» и «Правило плохого символа», описанные в алгоритме Бойера-Мура, нововведение заключается в таблице M, хранящей информацию о количесвте совпавших символов, что позволяет не выполнять каждую фазу лгоритма Бойера-Мура «с нуля».

2 Исходный код

Ниже приведён код с реализованными «Сильным правилом хорошего символа» (класс TGoodSuffix), «Сильным правилом плохого символа» (класс TSymbolTable) и самим алгоритмом с использованием таблицы M, описанными в [1].

```

class TGoodSuffix
{
  public:
  TGoodSuffix(std::vector<int>& lFunctionInput, std::vector<int>& lsFunctionInput) :
  lFunction(lFunctionInput),
  lsFunction(lsFunctionInput),
  size(lFunction.size()) {}

  uint32_t Get(uint32_t idx)
  {
    if (!idx)
    {
      if (size > 2){
        return size - lsFunction[1] - 1;
      }

      return 1;
    }

    if (idx == (uint32_t)this->size - 1){
      return 1;
    }

    if (lFunction[idx + 1]){
      return size - lFunction[idx + 1] - 1;
    }

    return size - lsFunction[idx + 1];
  }

  private:
  std::vector<int> lFunction;
  std::vector<int> lsFunction;
  uint32_t size;
};
```

```

class TSymbolTable
{
  public:
  TSymbolTable(std::vector<uint32_t>& pattern){
    for (unsigned i = 0; i < pattern.size(); ++i){
```

```
      this->table[pattern[i]] = i;
    }
  }

  uint32_t Get(uint32_t symbol, uint32_t idx)
  {
    auto it = table.find(symbol);

    if (it == table.end()){
      return idx + 1;
    }

    return idx - it->second;
  }

  private:
  std::map<uint32_t, uint32_t> table;
};
```

```

  std::vector<int> m(text.size(), 0);

  for (uint32_t j = pattern.size() - 1; j < (uint32_t)text.size();)
  {
    int h = j;
    int i = pattern.size() - 1;

    while (true)
    {
      if (!m[h])
      {
        if (text[h] == pattern[i] && i == 0)
        {
          std::cout << positions[j - pattern.size() + 1] << '\n';

          m[j] = pattern.size();
          j += succesShift.Get();

          break;
        }

        if (text[h] == pattern[i] && i > 0)
        {
          --i;
          --h;

          continue;
        }

        if (text[h] != pattern[i])
```

```
32          {
33            m[j] = j - h;
34
35            j += std::max(goodSuffix.Get(i), symbolTable.Get(text[h], i));
36
37            break;
38          }
39        }
40
41        if (m[h] < nFunction[i])
42        {
43          i -= m[h];
44          h -= m[h];
45
46          continue;
47        }
48
49        if (m[h] >= nFunction[i] && nFunction[i] >= i)
50        {
51          std::cout << positions[j - pattern.size() + 1] << '\n';
52
53          m[j] = j - h;
54          j += succesShift.Get();
55
56          break;
57        }
58
59        if (m[h] > nFunction[i] && nFunction[i] < i)
60        {
61          m[j] = j - h;
62          j += std::max(goodSuffix.Get(i - nFunction[i]), symbolTable.Get(text[h -
                nFunction[i]], i - nFunction[i]));
63
64          break;
65        }
66
67        if (m[h] == nFunction[i] && nFunction[i] > 0 && nFunction[i] < (int)i)
68        {
69          i -= m[h];
70          h -= m[h];
71
72          continue;
73        }
74      }
75    }
```

3 Консоль

```
[axr@pekarnya contest]$ g++ -Wall -o ag ApostolicoGiancarlo.cpp
[axr@pekarnya contest]$ ./ag
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
1,3
1,8
[axr@pekarnya contest]$
```

4 Тест производительности

В качестве тестирующего инструмента были выбраны гугл-тесты, интегрированные в программу при помощи CMake. Входные данные были сгенерированы при помощью вихря Мерсенна. Производительность сравнивалась с методом find() библиотекм STL.

5 Выводы

Выполнив четвёртую лабораторную работу по курсу «Дискретный анализ», я научился оптимизму и жизнерадостности.

Список литературы

[1] Гасфилд Дэн. *Строки, деревья и последовательности в алгоритмах: Ифнорматика и вычислительная биология.* Пер. с англ. И.В. Романовского — СПБ.: Невский Диалект; БХВ-Петербург, 2003. — 654 с.: ил.

[2] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))