Министерство образования и науки Российской Федерации
Новосибирский государственный технический университет
Кафедра прикладной математики

Лабораторная работа №1 – 3
по дисциплине «Сравнительный анализ языков программирования»

Работа с STL C++, C# и Java. Исследование сложности алгоритма

Факультет	ПМИ
Группа	ПМ 33, ПМ 32
Студенты	Жиляков А.
	Антоненко В.
	Губский Е.
Преподаватели	Рояк М. Э.
	Ступаков И. М.
Вариант	5

Новосибирск, 2016

I Указания к выполнению работы

Разработайте программу для решения поставленной задачи на языке **C++**, удовлетворяющую следующим требованиям:

1. программа должна использовать для ввода-вывода потоки **STL**,
2. программа не должна содержать собственных реализаций стандартных алгоритмов и структур данных, а использовать существующие в **STL**,
3. размер каждой подпрограммы не должен превышать 10 строк.

Протестируйте разработанную программу. Исследуйте асимптотические свойства разработанной программы на системе тестов с возрастающей размерностью.

Разработайте программы на языках `C#` и `Java` с использованием аналогичных структур данных и сравните результаты работы.

(`http://dispace.edu.nstu.ru/didesk/course/show/5626/5`)

I.1 Вариант задания

Задан большой текст (книга). Для наиболее часто встречающегося слова найти N наиболее часто встречающихся пар слов.

II Размышления

Для поиска наиболее популярного слова (или пары слов) удобно использовать контейнер std::map — ассоциативный массив с красно-чёрным деревом под капюшоном. Ключ суть слово, значение — частота. (Аналоги в `Java` и `C#` — `TreeMap` и `SortedDictionary`)

Через n обозначим количество слов в нашей входной книжке. Тогда для занесения в контейнер данных потребуется $O(n \log m)$ времени (поиск в дереве осуществляется за логарифм), где $m \leq n$ — количество слов без повторений.

Положим $m = n$ (в такой книжке каждое слово уникально), чтобы проверить на практике логарифмическую сложность — в действительности $m \ll n$ и алгоритм работает практически за линейное время. Будем увеличивать длину книжки в 4 раза при начальном $n = 250000$. Тогда отношение времени работы программы суть

$$\frac{4n \log_2 4n}{n \log_2 n} = 4 \left(1 + \frac{2}{\log_2 n}\right) \searrow 4,\ n \to \infty, \tag{1}$$

то есть предполагается, что программа станет работать медленнее в 4+ раз, при этом при $n \to \infty$ отношение (1) должно становиться всё меньше и меньше.

III Тестирование

III.1 Тестирование на реальной книжке

Для имплементации на всех языках результат один и тот же, во что и хотелось бы верить.

частота	**пара слов**
65	martin eden
44	and martin
42	martin was
40	martin had
27	that martin
25	martin said
20	but martin
19	to martin
18	martin did
18	martin and
17	you martin
13	when martin
13	martin answered
13	as martin
12	martin went

Таблица 1: “Martin Eden,” Jack London. Число слов — 138 754

III.2 Тестирование сложности и проверка «размышлений»

Результаты тестирования при $m = n$.

i	n	t, сек	t_i/t_{i-1}
1	250 000	.742	
2	1 000 000	3.855	5.2
3	4 000 000	19.738	5.12
4	16 000 000	94.566	4.79

Таблица 2: C++

i	n	t, сек	t_i/t_{i-1}
1	250 000	5.01	
2	1 000 000	26.53	5.2
3	4 000 000	101.15	3.8

Таблица 3: C#

i	n	t, сек	t_i/t_{i-1}
1	250 000	5.32	
2	1 000 000	27.01	5.08
3	4 000 000	108.47	4.02

Таблица 4: Java

IV Приложение

Компилируем исходники с помощью MSVC:
`cl -EHsc -Ox testGen.cpp user.cpp`,
генерируем тест из n разных слов (n подаётся на стандартный ввод):
`testGen > dummy.txt`
и наша программа готова к работе:
`user < dummy.txt » out.txt`.

IV.1 testGen.cpp

Генератор больших тестов — общий для всех языков.

```cpp
#include <random>
#include <algorithm>
#include <iterator>
#include <iostream>

int main() {
  int n;
  std::cin >> n;
  std::vector<int> v(n);
  for (int i = 0; i < n; ++i)
    v[i] = i;
  std::random_device rd;
  std::mt19937 g(rd());
  std::shuffle(v.begin(), v.end(), g);
  std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " "));
  std::cout << "\n";
  return 0;
}
```

IV.2 C++

user.cpp

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <set>
#include <string>
#include <algorithm>
#include <ctime> //замерить время
#include <cctype> //isalnum

using namespace std; //программа коротенькая, так что можно и расслабиться

//сегмент данных

vector<string> book; //текст книги
string word; //самое популярное слово
set<string> words2ignore; //игнорируемые слова (предлоги, артикли и т.д.)
unsigned N = 15; //количество выводимых сочетаний слов


void getBook() { //считываем текст книжки из стандартного ввода в контейнер
  while (cin >> word) {
    transform(word.begin(), word.end(), word.begin(), ::tolower); //прописные буквы к строчным
    word.erase(remove_if(word.begin(), word.end(), [](char c){return !isalnum(c);}), word.end()); //избавляемся от лишних символов в слове
    if (word.length()) book.push_back(word);
  }
}

void getWord() { //находим самое популярное слово
  map<string, size_t> freq; //ключ - слово, значение - частота появления
  size_t i, maxFreq = 0; //максимальная частота появления слова
  for (i = 0; i < book.size(); ++i)
    if (!words2ignore.count(book[i]) && ++freq[book[i]] > maxFreq) {
      word = book[i];
      maxFreq = freq[word];
    }
}

int main() {
  map<string, size_t> freq; //ключ - сочетание слов, значение - частота появления
  map<string, size_t>::const_iterator iter;
  vector<pair<size_t, string>> res; //отсортированный (см. далее) по частоте контейнер
  size_t i;
  clock_t begTime, endTime;
  ifstream ignoreFile("ignore.txt");
  while (ignoreFile >> word) words2ignore.insert(word);
  getBook();

    begTime = clock();
    getWord();
    for (i = 1; i < book.size() - 1; ++i)
      if (book[i] == word) {
        ++freq[book[i - 1] + " " + word];
        ++freq[word + " " + book[i + 1]];
      }
    for (iter = freq.begin(); iter != freq.end(); ++iter)
      res.push_back(make_pair(iter->second, iter->first));
    sort(res.rbegin(), res.rend());
    endTime = clock();
    N = min(N, res.size());
    for (i = 0; i < N; ++i)
      cout << res[i].first << ' ' << res[i].second << '\n';
    cout << "time diff: " << double(endTime - begTime) / CLOCKS_PER_SEC << '\n'
         << "words count: " << book.size() << "\n\n";

  return 0;
}
```

IV.3 C#

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LAB2
{
  class Program
  {
    static void Main(string[] args)
    {
      Analyzer a = new Analyzer("Martin Eden.txt", "ignore.txt");
      Console.WriteLine("MaxFreeQ={0}", a.MaxFreeq);
      int n = 15;
      for (int i = 0; i < a.Pairs.Count && i < n; ++i)
        Console.WriteLine("{0}\t{1}", i + 1, a.Pairs[i]);
            Console.WriteLine(a.RunTime);
      Console.ReadKey();
    }
  }
}
```

Analyzer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;

namespace LAB2
{
    /// <summary>
    /// анализирует файл
    /// </summary>
    public class Analyzer
    {
        public long RunTime;
        public class WordFreeq
        {
            public readonly string Word;
            public readonly uint Freeq;

            public WordFreeq(string Word, uint Freeq)
            {
                this.Word = Word;
                this.Freeq = Freeq;
            }
            public override string ToString()
            {
                return "[" + Word + ", " + Freeq + "]";
            }
        }
        List<string> book = new List<string>();                  //строки книги
        public HashSet<string> wordsToIgnore = new HashSet<string>();      //слова, которые не учитываются
        public SortedDictionary<string, uint> freeq = new SortedDictionary<string, uint>(); //набор слов и их встречаемость
        public List<WordFreeq> Pairs = new List<WordFreeq>();                //пары элементов сортированные по встречаемости

        public KeyValuePair<string, uint> MaxFreeq
        {
            get
            {
                KeyValuePair<string, uint> max = new KeyValuePair<string, uint>("", 0);
                foreach (var v in freeq)
                {
                    if (max.Value == 0 || v.Value > max.Value) max = v;
                }
                return max;
            }
        }

        public readonly string FileName;
        public Analyzer(string fileName, string ignoreFile)
        {
            LoadExcludeWords(ignoreFile);
            this.FileName = fileName;
            foreach (var v in wordsToIgnore)
                this.wordsToIgnore.Add(v.ToLower());
            ReadBook(fileName);
        }
        void LoadExcludeWords(string fileName)
        {
            wordsToIgnore.Clear();
            System.IO.StreamReader r = new System.IO.StreamReader(fileName);
            foreach (var word in r.ReadToEnd().Split(' ', ',', '\n', '\t', '\0', '\r'))
                wordsToIgnore.Add(word.ToLower());
            r.Close();
        }

        void ReadBook(string fileName)
        {
            //открываем файловый поток
            System.IO.StreamReader r = new System.IO.StreamReader(fileName);
            //ограничитель
            //читаем файл по словам в нижнем регистре
            book.Clear();
            foreach (var word in r.ReadToEnd().Split(' ', ',', '\n', '\t', '\0', '\r'))
                book.Add(word.ToLower());
```

```
            r.Close();
            //удаляем все ненужное
            book.RemoveAll(x => string.IsNullOrEmpty(x) || wordsToIgnore.Contains(x));
            //определяем сколько раз встретилось каждое слово
            freeq.Clear();
            var watch = System.Diagnostics.Stopwatch.StartNew();
            Console.WriteLine(watch.ElapsedMilliseconds);
            Console.WriteLine(GC.GetTotalMemory(true));

            unchecked
            {
                for (int i = 0; i < book.Count; i++)
                {
                    if (freeq.ContainsKey(book[i])) freeq[book[i]]++;
                    else freeq[book[i]] = 1;
                }
            }
            //берем макс слово
            Console.WriteLine(watch.ElapsedMilliseconds);
            string maxWord = MaxFreeq.Key;
            Console.WriteLine(watch.ElapsedMilliseconds);
            //вытаскиваем все пары
            SortedDictionary<string, uint> pairs = new SortedDictionary<string, uint>();    //пары слов и их количества
            for (int i = 1; i < book.Count - 1; ++i)
            {
                //отсеиваем то, что не содержит макс слова
                if (book[i] != maxWord && book[i + 1] != maxWord) continue;
                //создаем пару
                string p = book[i] + " " + book[i + 1];
                //вставка пары и количества пар
                if (pairs.ContainsKey(p)) pairs[p]++;
                else pairs[p] = 1;

               /* p = book[i-1] + " " + book[i];
                if (pairs.ContainsKey(p)) pairs[p]++;
                else pairs[p] = 1;*/
            }
            //создаем список сортированных пар
            Pairs.Clear();
            foreach (var v in pairs)
                Pairs.Add(new WordFreeq(v.Key, v.Value));
            Console.WriteLine(watch.ElapsedMilliseconds);
            //сортируем по убыванию встречаемости
            Pairs.Sort((a, b) => (int)(b.Freeq - a.Freeq));
            watch.Stop();
            Console.WriteLine(GC.GetTotalMemory(true));
            Console.WriteLine(watch.ElapsedMilliseconds);
            RunTime = watch.ElapsedMilliseconds;
        }
    }
}
```

IV.4 Java

Main.java

```
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.*;

class WordFreeq {
    public String   word;
    public Integer  freeq;

    public WordFreeq( String w, int f ) {
        word    = w;
        freeq   = f;
    }

    public String toString() {
        return "[" + word + ", " + freeq + "]";
    }
}

class KeyValuePair<A, B> {
    public A first;
    public B second;
}

public class Main {
    //Fields
    public static ArrayList<String>     words;

    public static String                uniqueWord;
    public static int                   wordsCount;
    public static int                   uniqueCount;

    public static HashSet<String>       wordsToExclude;
    public static TreeMap<String, Integer> freeq;
    public static ArrayList<WordFreeq>  pairs;


    //Methods
    public static void readExcludeWords( String path ) throws IOException {
        byte[] bytes = Files.readAllBytes( Paths.get( path ) );
        String allWords = new String( bytes );
        wordsToExclude = new HashSet<String>( Arrays.asList( allWords.split( "[\n ]" ) ) );
    }

    public static void readText( String path ) throws IOException {
```

```
46          byte[] bytes = Files.readAllBytes( Paths.get(path) );
47          String fullText = new String( bytes );
48          words = new ArrayList<String>( Arrays.asList( fullText.split( "[\n ]" ) ) );
49      }
50
51      public static void main(String[] args) throws IOException {
52          readExcludeWords( "ignore.txt" );
53          for ( String s : wordsToExclude ) {
54              wordsToExclude.add( s.toLowerCase() );
55          }
56          readText( "Martin Eden.txt" );
57
58          words.removeAll( wordsToExclude );
59
60          long begin = System.currentTimeMillis();
61
62          //Частота встречи слов
63          for ( int i = 0, n = words.size(); i < n; ++i ) {
64              if ( freeq.containsKey( words.get(i) ) ) {
65                  Integer k = freeq.get( words.get(i) );
66                  freeq.put( words.get( i ), k );
67              }
68              else {
69                  freeq.put( words.get( i ), 1 );
70              }
71          }
72
73          String maxWord = "";
74
75          TreeMap<String, Integer> pairsTree = new TreeMap<String, Integer>();
76          for ( int i = 1, n = words.size() - 1; i < n; ++i ) {
77              if ( words.get(i) != maxWord && words.get(i+1) != maxWord )
78                  continue;
79
80              String p = words.get(i) + " " + words.get(i+1);
81              if ( pairsTree.containsKey( p ) ) {
82                  Integer k = pairsTree.get( p );
83                  pairsTree.put( p, k );
84              }
85              else
86                  pairsTree.put( p, 1 );
87          }
88
89          pairs.clear();
90          for ( Map.Entry<String, Integer> entry : pairsTree.entrySet() ) {
91              pairs.add( new WordFreeq( entry.getKey(), entry.getValue() ) );
92          }
93
94          Collections.sort( pairs, new Comparator<WordFreeq>() {
95              public int compare( WordFreeq s1, WordFreeq s2 ) {
96                  return s2.freeq - s1.freeq;
97              }
98          });
99
100         long end = System.currentTimeMillis() - begin;
101
102         //writeAnswer();
103         System.out.println( "Time " + end );
104
105         //writer.close();
106     }
107 }
```

V Делаем выводы

Как видно из таблиц (2 – 4), асимптотика, предсказанная в секции (II), работает всегда — независимо от языка программирования (другого мы и не ожидали).

Так же стоит отметить, что `C++`, будучи языком компилируемым, справляется с задачей быстрее — особенно это заметно на больших тестах (с тестом на 16 000 000 он расправился в 5 раз быстрее, чем `Java` и `C#` справились с тестом в 4 раза меньшего размера). Это можно объяснить тем, что `Java` и `C#` — языки с виртуальной машиной. Интерпретация байткода для машин в исполняемый код занимает своё время. Этим же можно оправдать то, что оба языка показали себя примерно одинаково.

Было замечено, что в случае использования неупорядоченных ассоциативных контейнеров (хэш-таблиц), — `std::unordered_map` и `Dictionary` в `STL C++` и `C#` соответственно, — на небольших тестах `C#` обогнал `C++` приблизительно в 2 раза. Что наводит на мысль, что майкрософтовская реализация хэш-таблиц в `MSVC` для `STL` несколько хуже их же реализации для `C#`.