**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ВТ**

Курсовая работа

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: измерение временной сложности алгоритма**

**Вариант 24**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7308 |  | Замыслов Н.Ю. |
| Студентка гр. 7308 |  | Цебульский С.А. |
| Преподаватель |  | Колинько П.Г. |

Санкт-Петербург

2019

**Цель работы**

Провести эксперимент по определению оценки сложности цепочки операций и проанализировать результат.

**Постановка задачи**

Выполнить статистический эксперимент по измерению временной сложности алгоритма обработки данных, использующего стандартную библиотеку шаблонов.

Доработать программу лабораторной работы 6 с красно-чёрным деревом таким образом, чтобы она генерировала множества мощностью, меняющейся от 10 до 200, измеряла время выполнения цепочки операций над множествами и последовательностями и выводила результат в текстовый файл. Первая строка этого файла должна содержать количество опытов, а последующие — по паре значений «размер входа — время» для каждого опыта. Затем эти данные обрабатываются, и по результатам обработки делается заключение о временной сложности алгоритма.

Для повышения надёжности эксперимента предусмотреть в программе перехват исключительных ситуаций таким образом, чтобы сбой сводился просто к пропуску очередного шага эксперимента. В частности, рекомендуется перехватывать ситуацию *bad\_alloc*, возбуждаемую конструктором при нехватке памяти.

**Описание созданных функций и структур данных**

На основе встроенного контейнера unordered\_set был создан наш собственный контейнер, поддерживающий последовательности и операции над ними. Для поддержки операций над последовательностями используется вектор, хранящий указатели на элементы хеш-таблицы. Поддерживаемые операции:

1) /. Создает новую последовательность, в которой будут находиться элементы, которые содержатся в первом объекте контейнера, но не содержатся во втором.

2) &. Создает последовательность, в которой находятся элементы, которые находятся и в первом объекте контейнера и во втором.

3) ^. Создает последовательность, в которой находятся элементы, которые либо принадлежат и первому и второму контейнерам, либо не принадлежат(xor).

4) Concat. Присоединяет вторую последовательность к первой начиная с конца.

5) Subst. Присоединяет вторую последовательность к первой начинае с указанного места pos.

6) Excl. Вторая последовательность исключается из первой, если она является её частью.

**Временная сложность используемых операций:**

*Таблица 1. Временная сложность операций*

|  |  |
| --- | --- |
| **Название функции** | **Временная сложность** |
| & | O(n) |
| **|** | O(n) |
| **^** | O(n) |
| **/** | O(n) |
| excl | O(n) |
| mul | O(n) |
| **Название функции** | **Временная сложность** |
| erase | O(n) |

**Результат работы программы RG32**

*Таблица 2. Результаты работы программы RG32*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Результаты статистической обработки (из файла out.txt)** | | | | | | |  |  |  |  |  |
|  |  |  |  | (1) | (ln N) | (N) | (N ln N) | (N^2) | (N^3) | (N^4) |  |  |
| **Вариант** | **D** | **S** | **K** | **c0** | **c1** | **c2** | **c3** | **c4** | **c5** | **c6** |  |  |
| **1** | 8,63E+12 | 2,94E+06 | 1 | 6,71E+06 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 189 |
| **2** | 3,06E+12 | 1,75E+06 | 2 | -1,19E+07 | 4,69E+06 | 0 | 0 | 0 | 0 | 0 | 391 | 0 |
| **3** | 2,20E+12 | 1,48E+06 | 2 | -175116 | 0 | 116483 | 0 | 0 | 0 | 0 | 360 | 0 |
| **4** | 2,10E+12 | 1,45E+06 | 3 | 6,81E+06 | -2,63E+06 | 175350 | 0 | 0 | 0 | 0 | 930 | 0 |
| **5** | 2,09E+12 | 1,44E+06 | 3 | 2,99E+06 | 0 | -240371 | 72971,4 | 0 | 0 | 0 | 817 | 0 |
| **6** | 2,10E+12 | 1,45E+06 | 4 | -950399 | 2,54E+06 | -616260 | 138201 | 0 | 0 | 0 | 1122 | 0 |
| **7** | 2,08E+12 | 1,44E+06 | 3 | 1,67E+06 | 0 | 29967,9 | 0 | 824,744 | 0 | 0 | 747 | 0 |
| **8** | 2,10E+12 | 1,45E+06 | 4 | -682083 | 0 | 473865 | -117112 | 2057,61 | 0 | 0 | 1045 | 0 |
| **9** | 2,13E+12 | 1,46E+06 | 5 | -2,63E+06 | -261819 | 911885 | -226264 | 3042,39 | 0 | 0 | 1299 | 0 |
| **10** | 2,10E+12 | 1,45E+06 | 4 | -163310 | 0 | 166695 | 0 | -2041,11 | 17,9821 | 0 | 933 | 0 |
| **11** | 2,10E+12 | 1,45E+06 | 5 | 89057,4 | 49906,4 | 139784 | 0 | -1543,12 | 15,0536 | 0 | 1184 | 0 |
| **12** | 2,10E+12 | 1,45E+06 | 5 | 384861 | 0 | 187901 | -25409,8 | -250,356 | 8,64497 | 0 | 1977 | 0 |
| **13** | 2,11E+12 | 1,45E+06 | 6 | 26524,9 | 222720 | 152392 | -18450,2 | -297,536 | 8,81069 | 0 | 2101 | 0 |
| **14** | 2,10E+12 | 1,45E+06 | 5 | -237099 | 0 | 131748 | 0 | 290,824 | -25,9861 | 0,254123 | 1624 | 0 |
| **15** | 2,12E+12 | 1,46E+06 | 6 | 124295 | 0 | -7885,44 | 51319,1 | -1642,73 | -12,5186 | 0,212551 | 1870 | 0 |
| **16** | 2,13E+12 | 1,46E+06 | 7 | -6018,51 | 46533,2 | 18545,1 | 38790,8 | -990,668 | -17,9179 | 0,231949 | 2140 | 0 |

**Обработка результатов эксперимента**

По значениям отношений дисперсий наиболее подходящим уравнением регрессии выбрано уравнение №3, так как при дальнейшем усложнении уменьшение выборочной дисперсии перестаёт быть значимым. Это видно из следующей таблицы. Это было определено с помощью квантилей распределения Фишера. Количество измерений в работе равняется 190, что приблизительно соответствует значению 1,26 по графику уровней значимости отношения выборочных дисперсий.

*Таблица 3. Отношение дисперсий*

|  |  |  |  |
| --- | --- | --- | --- |
| Отношение дисперсий | | |  |
|  | **1** | **2** | **3** |
| **1** | 1,00 | 0,35 | 0,26 |
| **2** | 2,82 | 1,00 | 0,72 |
| **3** | 3,92 | 1,39 | 1,00 |
| **4** | 4,11 | 1,46 | 1,05 |
| **5** | 4,13 | 1,47 | 1,06 |
| **6** | 4,12 | 1,46 | 1,05 |
| **7** | 4,15 | 1,47 | 1,06 |
| **8** | 4,11 | 1,46 | 1,05 |
| **9** | 4,05 | 1,43 | 1,03 |
| **10** | 4,11 | 1,46 | 1,05 |
| **11** | 4,10 | 1,45 | 1,05 |
| **12** | 4,11 | 1,46 | 1,05 |
| **13** | 4,08 | 1,45 | 1,04 |
| **14** | 4,10 | 1,45 | 1,05 |
| **15** | 4,07 | 1,44 | 1,04 |
| **16** | 4,05 | 1,44 | 1,03 |

Далее результаты эксперимента представлены в виде графика.

Рис. 1. Результаты эксперимента

**Выводы**

Во время выполнения курсовой работы была доработана программа из лабораторной работы №6, был проведён статистический эксперимент по измерению временной сложности алгоритма. Результаты эксперимента показывают, что временная сложность алгоритма O(n). Результаты эксперимента соответствуют теоретической оценке, на основании чего можно считать эксперимент успешным.

**Список использованных источников**

1. Отчет по работе №6.
2. Колинько П. Г. Алгоритмы и структуры данных. Часть 2: Методические указания к практическим занятиям на ПЭВМ и курсовому проектированию. Вып. 1902. — СПб.: СПбГЭТУ «ЛЭТИ», 2019. — 56 с.: ил.

**Листинг**

STL.h

#pragma once

#include <iostream>

#include <algorithm>

#include <unordered\_set>

#include <vector>

#include <iterator>

const int N = 200; // мощность множества

using namespace std;

typedef unordered\_set<int> MySet;

typedef unordered\_set<int>::iterator Mylt;

typedef vector<Mylt> MySeq;

class hash\_table // open hashing

{

MySet data;

MySeq seq;

hash\_table();

bool add(Mylt s);

public:

hash\_table(int p);

hash\_table(const hash\_table &);

~hash\_table() {}

void excl(hash\_table & set);

void subst(hash\_table & set, int pos);

void concat(hash\_table & set);

friend std::ostream & operator<<(std::ostream& os, const hash\_table& set);

hash\_table operator &(const hash\_table&);

hash\_table operator ^(const hash\_table&);

hash\_table operator /(const hash\_table&);

};

STL.cpp

#include "Time.h"

hash\_table::hash\_table()

{

data = MySet();

seq = MySeq();

}

hash\_table::hash\_table(int p)

{

for (int i = 0; i < p; ++i)

seq.push\_back(data.insert(rand() % N).first);

};

hash\_table::hash\_table(const hash\_table & set) // copy constructor

{

for (auto x : set.seq) seq.push\_back(data.insert(\*x).first);

}

int hash\_table::get\_power() {

return data.size();

}

bool hash\_table::add(Mylt s)

{

if (data.find(\*s) != data.end()) {

seq.push\_back(data.find(\*s));

return false;

}

else {

seq.push\_back(data.insert(\*s).first);

return true;

}

}

void hash\_table::excl(hash\_table & set)

{

bool chek = true;

if (set.seq.size() <= seq.size()) {

for (int i = 0; i < set.seq.size() && chek; i++) {

auto k = set.seq[i];

if (data.find(\*k) != data.end())

chek = false;

}

if (chek) {

hash\_table result;

for (int i = 0, j = 0; i < seq.size(); i++) {

if (set.data.find(\*seq[i]) == set.data.end())

result.add(seq[i]);

else

if (j < set.seq.size()) j++;

else result.add(seq[i]);

}

seq.swap(result.seq);

data.swap(result.data);

}

}

}

void hash\_table::subst(hash\_table & set, int pos)

{

hash\_table result;

for (int i = 0; i < pos; i++) {

result.add(seq[i]);

}

for (int j = 0; j < set.seq.size(); j++) {

result.add(set.seq[j]);

}

for (int z = pos; z < seq.size(); z++) {

result.add(seq[z]);

}

seq.swap(result.seq);

data.swap(result.data);

}

void hash\_table::concat(hash\_table & set)

{

hash\_table result;

for (int i = 0; i < seq.size(); i++) {

result.add(seq[i]);

}

for (int j = 0; j < set.seq.size(); j++) {

result.add(set.seq[j]);

}

for (int z = seq.size(); z < seq.size(); z++) {

result.add(seq[z]);

}

seq.swap(result.seq);

data.swap(result.data);

}

hash\_table hash\_table::operator /(const hash\_table& set)

{

hash\_table result;

for (int i = 0; i < data.bucket\_count(); i++)

for (auto it = data.begin(i); it != data.end(i); ++it)

if (set.data.find(\*it) == set.data.end()) {

result.add(it);

}

return result;

}

hash\_table hash\_table::operator &(const hash\_table& set)

{

hash\_table result;

for (int i = 0; i < data.bucket\_count(); i++)

for (auto it = data.begin(i); it != data.end(i); ++it)

if (set.data.find(\*it) != set.data.end()) {

result.add(it);

}

return result;

}

hash\_table hash\_table::operator ^(const hash\_table& set)

{

hash\_table result;

for (int i = 0; i < data.bucket\_count(); i++)

for (auto it = data.begin(i); it != data.end(i); ++it)

{

if (set.data.find(\*it) == set.data.end()) {

result.add(it);

}

}

for (int i = 0; i < set.data.bucket\_count(); i++)

for (auto it = set.data.begin(i); it != set.data.end(i); ++it)

{

if (data.find(\*it) == data.end()) {

result.add(it);

}

}

return result;

}

std::ostream & operator<<(std::ostream & os, const hash\_table & set)

{

std::cout << "HASH:\n";

unsigned n = set.data.bucket\_count();

for (auto i = 0; i < n; ++i) {

cout << "[" << i << "] : ";

for (auto it = set.data.begin(i); it != set.data.end(i); ++it)

cout << " " << \*it;

cout << '\n';

}

std::cout << "SEQUENCE: ";

for (int i = 0; i < set.seq.size(); i++)

{

cout << \*(set.seq[i]) << " ";

}

cout << endl;

return os;

}

Main.cpp

#include "STL.h"

#include <fstream>

#include <chrono>

int main()

{

ofstream out("out.txt");

out << "190" << endl;

for (int i = 10; i < 200; i++)

{

size\_t power = 0;

size\_t middle\_power = 0;

hash\_table a(i), b(i);

auto t1 = std::chrono::high\_resolution\_clock::now();

power = power + a.get\_power() + b.get\_power();

hash\_table res(a / b);

power = power + res.get\_power();

res.excl(a);

power = power + res.get\_power();

res.concat(b);

middle\_power = power / 4;

auto t2 = std::chrono::high\_resolution\_clock::now();

auto t\_diff = t2 - t1;

out << middle\_power << " " << (t\_diff.count()) << endl;

}

out.close();

system("pause");

return 0;

}