МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САПР

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Измерение времени сложности алгоритм

|  |  |  |
| --- | --- | --- |
| Преподаватель |  | \_\_\_\_\_\_\_\_\_\_Манирагена В. |
|  |  |  |

Студент гр. 1335 \_\_\_\_\_\_\_\_\_\_Максимов Ю.Е.

Санкт-Петербург

2024

Оглавление

[1. Задание 2](#_Toc20146)

[2. Дополнительные сведения 2](#_Toc20147)

[3. Теоретическая оценка временной сложности 2](#_Toc20148)

[4. Результаты эксперимента 4](#_Toc20149)

[5. Выводы 5](#_Toc20150)

[6. Список используемых источников 5](#_Toc20151)

[7. Приложение 5](#_Toc20152)

# Задание

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

Доработать программу таким образом, чтобы она генерировала множества мощностью, меняющейся, например, от 10 до 200, измеряла время выполнения цепочки операций над множествами и последовательностями и выводила результат в текстовой файл. Затем эти данные обрабатываются, и по результатам обработки делается заключение о временной сложности алгоритма.

Для повышения надежности эксперимента предусмотреть в программе перехват исключительных ситуаций таким образом, чтобы сбой сводился просто к пропуску очередного шага эксперимента.

# Дополнительные сведения

Начальный размер входа – 5.

Увеличение размера за шаг алгоритма – 4. Количество шагов – 201.

# Теоретическая оценка временной сложности

Операции над множествами:

* Объединение – O(n) + O(n ln n) + O(n) = O(n ln n);
* Пересечение – O(n) + O(n ln n) + O(n) = O(n ln n);

Операции над последовательностями

(*прим. – n – первая последовательность, s – вторая*):

* Исключение – O(n\*s) + O(s) + O(n) = O(n);
* Включение – O(n+s) = O(n);
* Замена – O(n);

**Цепочка операций:**

Цепочка операций была составлена так, чтобы количество операций с последовательностями и множествами было равное.

Выбор операции и ее элементов был сделан с помощью сайта random.org, который позиционирует себя как «True Random Number Service».

Получившаяся цепочка операций:

uset\_union(H[1], H[4], H[3]); utransform(H[4], H[0], H[2], 3, 5); utransform(H[0], H[2], H[1], 2, 3); uset\_inter(H[1], H[0], H[4]); utransform(H[4], H[1], H[1], 1); utransform(H[2], H[4], H[0], 3, 2); utransform(H[1], H[0], H[0], 2); uset\_union(H[2], H[3], H[4]); uset\_union(H[0], H[0], H[4]); utransform(H[3], H[1], H[2], 1); Где:

uset\_union(const hasht &A, const hasht &B, hasht &C) – объединение; uset\_inter(const hasht &A, const hasht &B, hasht &C) – пересечение;

utransfrom(const hasht &A, const hasht &B, hasht &C, int action, int pos = 0) – операции над последовательностями; action – [1; 3] - соответственно исключение, включение, замена, pos – позиция, с которой начинаются преобразования.

**Временная сложность этой последовательности:**

Объединение + Замена + Включение + Пересечение + Исключение + Замена +

Включение + Объединение + Объединение + Исключение; Из теоретической оценки имеем:

O(n ln n) + O(n) + O(n) + O(n ln n) + O(n) + O(n) + O(n) + O(n ln n) + O(n ln n) + O(n) В итоге: **O(n ln n).**

# Результаты эксперимента

Табл. 1. Вспомогательная таблица.

Определение подходящего уравнения регрессии

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | Отношение дисперсий | | | | |  |  |  |  |  |  |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| **1** | 1 | 0,32 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 |
| **2** | 3,16 | 1,00 | 0,24 | 0,22 | 0,22 | 0,22 | 0,22 | 0,22 | 0,22 | 0,22 | 0,22 | 0,22 | 0,22 | 0,23 | 0,22 | 0,22 |
| **3** | 13,37 | 4,23 | 1,00 | 0,95 | 0,93 | 0,93 | 0,93 | 0,93 | 0,94 | 0,93 | 0,93 | 0,93 | 0,94 | 0,97 | 0,91 | 0,92 |
| **4** | 14,10 | 4,46 | 1,05 | 1,00 | 0,98 | 0,98 | 0,98 | 0,98 | 0,99 | 0,98 | 0,98 | 0,98 | 0,99 | 1,02 | 0,96 | 0,97 |
| **5** | 14,37 | 4,54 | 1,07 | 1,02 | 1,00 | 1,00 | 1,00 | 1,00 | 1,01 | 1,00 | 1,00 | 1,00 | 1,01 | 1,04 | 0,98 | 0,99 |
| **6** | 14,36 | 4,54 | 1,07 | 1,02 | 1,00 | 1,00 | 1,00 | 1,00 | 1,01 | 1,00 | 1,00 | 1,00 | 1,01 | 1,04 | 0,98 | 0,99 |
| **7** | 14,33 | 4,53 | 1,07 | 1,02 | 1,00 | 1,00 | 1,00 | 1,00 | 1,01 | 0,99 | 1,00 | 1,00 | 1,01 | 1,04 | 0,98 | 0,99 |
| **8** | 14,32 | 4,53 | 1,07 | 1,02 | 1,00 | 1,00 | 1,00 | 1,00 | 1,01 | 0,99 | 1,00 | 1,00 | 1,01 | 1,04 | 0,98 | 0,98 |
| **9** | 14,24 | 4,50 | 1,07 | 1,01 | 0,99 | 0,99 | 0,99 | 0,99 | 1,00 | 0,99 | 0,99 | 0,99 | 1,00 | 1,03 | 0,97 | 0,98 |
| **10** | 14,43 | 4,56 | 1,08 | 1,02 | 1,00 | 1,00 | 1,01 | 1,01 | 1,01 | 1,00 | 1,01 | 1,01 | 1,01 | 1,04 | 0,99 | 0,99 |
| **11** | 14,35 | 4,54 | 1,07 | 1,02 | 1,00 | 1,00 | 1,00 | 1,00 | 1,01 | 0,99 | 1,00 | 1,00 | 1,01 | 1,04 | 0,98 | 0,99 |
| **12** | 14,31 | 4,53 | 1,07 | 1,02 | 1,00 | 1,00 | 1,00 | 1,00 | 1,01 | 0,99 | 1,00 | 1,00 | 1,01 | 1,04 | 0,98 | 0,98 |
| **13** | 14,24 | 4,50 | 1,07 | 1,01 | 0,99 | 0,99 | 0,99 | 0,99 | 1,00 | 0,99 | 0,99 | 1,00 | 1,00 | 1,03 | 0,97 | 0,98 |
| **14** | 13,81 | 4,37 | 1,03 | 0,98 | 0,96 | 0,96 | 0,96 | 0,96 | 0,97 | 0,96 | 0,96 | 0,96 | 0,97 | 1,00 | 0,94 | 0,95 |
| **15** | 14,61 | 4,62 | 1,09 | 1,04 | 1,02 | 1,02 | 1,02 | 1,02 | 1,03 | 1,01 | 1,02 | 1,02 | 1,03 | 1,06 | 1,00 | 1,00 |
| **16** | 14,54 | 4,60 | 1,09 | 1,03 | 1,01 | 1,01 | 1,02 | 1,02 | 1,02 | 1,01 | 1,01 | 1,02 | 1,02 | 1,05 | 1,00 | 1,00 |

Начиная с номера 5 в последующих столбцах таблицы на всех строках изменение отношения дисперсий незначительное - +-0.1.

Иногда в 14 столбце можно наблюдать скачок – скорее всего из-за того, что в какой-то момент ХТ автоматически расширяется и перестраивается.

График регрессии представлен ниже.

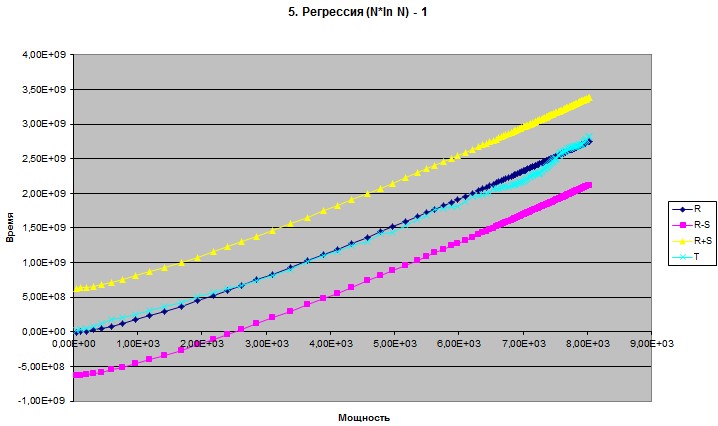


Рис. 1. Регрессия (n ln n) - 1

Все измерения попали в доверительную область.

# Выводы

Экспериментальная оценка временной сложности алгоритма – O(n ln n), а теоретическая – O(n ln n). Т.е. эксперимент подтвердил теорию.

Это объясняется тем, что в цепочку операций включены операции над множествами, временная сложность которых O(n ln n), т.к. они реализованы с помощью стандартных алгоритмов set\_union и set\_intersection, которые требуют, чтобы перед их использованием элементы были упорядочены. Для этого я использую list.sort(), временная сложность которого O(n ln n), а уже из полученного списка формирую множество. Этим и объясняется получившаяся временная сложность.

Данная курсовая повысила мои навыки по оценке временной сложности алгоритмов, закрепила мои навыки по работе со стандартной библиотекой шаблонов, научила анализировать временную сложность алгоритмов экспериментальным путем.

# Список используемых источников

1. Измерение времени. / Лекция по АиСД
2. Генератор случайных чисел. https://www.random.org/
3. Критерий Фишера. https://en.wikipedia.org/wiki/F-test

# Приложение

1. main7.cpp – исходный код курсовой работы.
2. my\_hash.h – определение объекта hasht для работы с множествами и последовательностями.

myhash.h

#include <ctime>

#include <string>

#include <list>

#include <iostream>

#include <unordered\_set>

#include <algorithm> #include <fstream> using namespace std;

int p = 5; class hasht{

unordered\_set <int> table; list <int> al;

public:

void ShowHash(char N) const; int Fill();

friend int uset\_union(const hasht &A, const hasht &B, hasht &C); friend int uset\_inter(const hasht &A, const hasht &B, hasht &C);

friend int utransform(const hasht &A, const hasht &B, hasht &C, int action, int

pos); void FillManualy(); int size() const

{return(table.size() + al.size());}

};

int hasht:: Fill()

{ for(int i = 0; i < p; ++i) al.push\_back(rand()%1000);

al.unique(); list <int> tmp (al); while(!tmp.empty())

{ table.insert(tmp.front()); tmp.pop\_front();

}

return size();

}

/\*void hasht :: FillManualy()

{ int goodstr = 1; char ch[10]; while(goodstr)

{

ShowHash(); cout << "\nHash function will automaticly determinate a position of element.

Just write elements.\nWrite a symbol to stop\n"; cin.get(ch, 10).get();

for(int j = 0; goodstr && ch[j] != '\0'; ++j) goodstr = isdigit(ch[j]);

if(goodstr)

{ table.emplace(stoi(ch)); al.push\_back(stoi(ch));

}

system("cls");

} system("cls"); ShowHash(); cin.get(); cin.get(); system("cls");

}\*/

int uset\_union(const hasht &A, const hasht &B, hasht &C)

{

C.al.clear(); C.table.clear(); list<int> tmp (A.al); list<int> stmp (B.al); tmp.sort(); stmp.sort();

set\_union(tmp.begin(), tmp.end(), stmp.begin(), stmp.end(), back\_inserter(C.al));

tmp = C.al; while(!tmp.empty())

{

C.table.insert(tmp.front()); tmp.pop\_front();

}

return C.size();

}

int uset\_inter(const hasht &A, const hasht &B, hasht &C)

{

C.al.clear(); C.table.clear(); list<int> tmp (A.al); list<int> stmp (B.al); stmp.sort(); tmp.sort();

set\_intersection(tmp.begin(), tmp.end(), stmp.begin(), stmp.end(),

back\_inserter(C.al));

tmp = C.al; while(!tmp.empty())

{

C.table.insert(tmp.front()); tmp.pop\_front();

}

return C.size();

}

int utransform(const hasht &A, const hasht &B, hasht &C, int action, int pos = 0) // 1 - excl (delete second from first if it includes second), 2 - subst (includes from P position, do not replace stuff), 3 - change (includes from p WITH replacing)

{

C.al.clear(); C.table.clear(); switch(action)

{ case 1:

{

//cout << "\nEXCL\n..."; list<int> Btmp (B.al); list<int> tmp (A.al); list <int>:: iterator p;

p = search(tmp.begin(), tmp.end(), Btmp.begin(), Btmp.end()); for(int i = 0; p!=tmp.end() && i < Btmp.size(); ++i) p = tmp.erase(p); C.al = tmp; while(!tmp.empty())

{

C.table.insert(tmp.front()); tmp.pop\_front();

}

//cout << "\nDone!\n";

} break;

case 2:

{

//cout << "\nSUBST...\nEnter the position number to add before "; list <int> tmp(A.al); list <int> stmp(B.al); while(pos && !tmp.empty())

{

C.al.push\_back(tmp.front());

C.table.insert(tmp.front()); tmp.pop\_front();

--pos;

}

while(!stmp.empty())

{

C.al.push\_back(stmp.front());

C.table.insert(stmp.front()); stmp.pop\_front();

}

while(!tmp.empty())

{

C.al.push\_back(tmp.front());

C.table.insert(tmp.front()); tmp.pop\_front();

}

//cout << "\nDone!\n";

} break;

case 3:

{

//cout << "\nCHANGE...\nEnter the position number FROM "; list <int> tmp(A.al); list <int> stmp(B.al); while(pos && !tmp.empty())

{

C.al.push\_back(tmp.front());

C.table.insert(tmp.front()); tmp.pop\_front();

--pos;

}

while(!stmp.empty())

{

C.al.push\_back(stmp.front());

C.table.insert(stmp.front()); stmp.pop\_front(); if(!tmp.empty()) tmp.pop\_front();

}

while(!tmp.empty())

{

C.al.push\_back(tmp.front());

C.table.insert(tmp.front()); tmp.pop\_front();

}

} break;

default:

//cout << "\nWrong index"; break;

}

return C.size();

}

void hasht :: ShowHash(char N) const{ list <int> tmp;

cout << "\n" << N << '(' << p << ") = [ "; for(auto q = table.begin(); q != table.end(); ++q) cout << \*q << ' ';

cout << ']'; cout << "\nSequence: <"; tmp = al; while(!tmp.empty())

{ cout << " " << tmp.front(); tmp.pop\_front();

}

cout << " >\n"; };

main #include "my\_hash.h" const int U = 6;

int main(){ srand(time(0)); ofstream out("in.txt");

if(!out) { cout << "\nError!"; return(1);} unsigned \_\_int64 t1, t2; int q = 300, dp = 2; hasht H[U];

out << static\_cast <int> ((q-p)/dp+1); do

{ try

{ size\_t k = 0, sets = 0; for(int i = 0; i < U; ++i)

{ k += H[i].Fill(); sets++;

}

// for(int i = 0; i < U; ++i)

// H[i].ShowHash(i+65);

t1 = \_\_rdtsc();

// union, inter, excl, sub, change5 k += uset\_union(H[1], H[4], H[3]); k += utransform(H[4], H[0], H[2], 3, 5); k += utransform(H[0], H[2], H[1], 2, 3); k += uset\_inter(H[1], H[0], H[4]); k += utransform(H[4], H[1], H[1], 1); k += utransform(H[2], H[4], H[0], 3, 2); k += utransform(H[1], H[0], H[0], 2); k += uset\_union(H[2], H[3], H[4]); k += uset\_union(H[0], H[0], H[4]); k += utransform(H[3], H[1], H[2], 1);

t2 = \_\_rdtsc(); //H[4].ShowHash('E');

k /= sets;

cout << "\np = " << p << " k = " << k << " Dt = " << t2 - t1;

out << '\n' << k << ' ' << t2-t1;

} catch(...)

{ cout << "\nCycle error";

}

}

while((p += dp) <= q); out.close(); cin.get(); cin.get(); return 0;

}