**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

Курсовая работа

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

**Тема: Измерение временной сложности алгоритма**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7307 | Торопов В.А. |  |
| Преподаватель | Колинько П.Г. |  |

**Оглавление**

[**Цель работы** 3](#_Toc10639954)

[**Задание** 3](#_Toc10639955)

[**Описание созданных структур данных и функций, а также особенности проведения эксперимента** 4](#_Toc10639956)

[**Результат работы программы RG32** 5](#_Toc10639957)

[**Обработка результатов эксперимента** 5](#_Toc10639958)

[**Выводы** 7](#_Toc10639959)

[**Приложение:** 8](#_Toc10639960)

[**Список литературы** 11](#_Toc10639961)

# **Цель работы**

Измерить временную сложность цепочки операций.

# **Задание**

Доработать программу из работы №6, выполнить статистический эксперимент по измерению временной сложности алгоритма обработки данных, использующего стандартную библиотеку шаблонов.

# **Описание созданных структур данных и функций, а также особенности проведения эксперимента**

В 6-ой лабораторной работе, была разработана структура данных, совмещающая в себе последовательность и множество, и поддерживающая операции над ними. Структура данных основывается на контейнерах unordered\_set. Помимо стандартных операций над множествами(AND(&),OR(|),XOR(+),DIFFERENCE(/)) были реализованы операции над последовательностями:

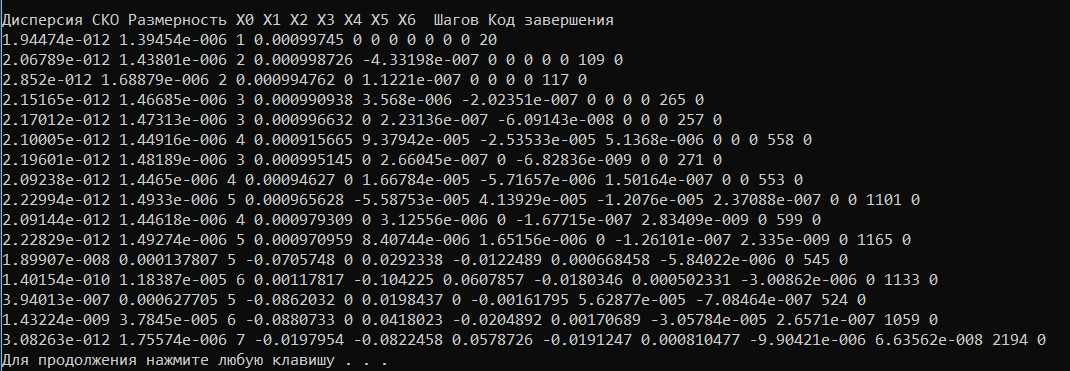
1. Слияние (*MERGE*). Объединение двух упорядоченных последовательностей в третью с сохранением упорядоченности.

2. . Сцепление (*CONCAT*). Вторая последовательность подсоединяется к концу первой, образуя её продолжение.

3. Включение (*SUBST*). Вторая последовательность включается в первую с указанной позиции *p*.

Для проведения эксперимента генерировалось 20 множеств с мощностью от 10 до 133 с шагом 1. Дальше над множествами проводились все вышепредставленные операции (В программе это множество R). После выполнения всех операций в файл записывалась мощность всех использованных множеств, а также время. Если после выполнения операций полученное время оказывалось 0, то опыт проводился заново.

# **Результат работы программы RG32**



# **Обработка результатов эксперимента**

По значениям отношений дисперсий наиболее подходящим уравнением регрессии выбрано уравнение №7, 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 123, что приблизительно соответствует значению 1,26 по графику уровней значимости отношения выборочных дисперсий.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Отношение дисперсий | |  |  |  |  |  |  |  |  |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **1** | 1,00 | 0,37 | 0,06 | 0,02 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| **2** | 2,70 | 1,00 | 0,17 | 0,05 | 0,01 | 0,01 | 0,00 | 0,00 | 0,00 |
| **3** | 15,95 | 5,90 | 1,00 | 0,29 | 0,07 | 0,03 | 0,01 | 0,01 | 0,01 |
| **4** | 55,30 | 20,48 | 3,47 | 1,00 | 0,24 | 0,11 | 0,05 | 0,05 | 0,05 |
| **5** | 234,65 | 86,88 | 14,71 | 4,24 | 1,00 | 0,48 | 0,21 | 0,22 | 0,20 |
| **6** | 489,03 | 181,07 | 30,67 | 8,84 | 2,08 | 1,00 | 0,44 | 0,45 | 0,41 |
| **7** | 1111,42 | 411,52 | 69,70 | 20,10 | 4,74 | 2,27 | 1,00 | 1,03 | 0,93 |
| **8** | 1082,16 | 400,68 | 67,86 | 19,57 | 4,61 | 2,21 | 0,97 | 1,00 | 0,91 |
| **9** | 1190,21 | 440,69 | 74,64 | 21,52 | 5,07 | 2,43 | 1,07 | 1,10 | 1,00 |
| **10** | 970,29 | 359,26 | 60,85 | 17,55 | 4,14 | 1,98 | 0,87 | 0,90 | 0,82 |
| **11** | 1246,86 | 461,66 | 78,19 | 22,55 | 5,31 | 2,55 | 1,12 | 1,15 | 1,05 |
| **12** | 964,01 | 356,93 | 60,45 | 17,43 | 4,11 | 1,97 | 0,87 | 0,89 | 0,81 |
| **13** | 1010,17 | 374,03 | 63,35 | 18,27 | 4,31 | 2,07 | 0,91 | 0,93 | 0,85 |
| **14** | 939,32 | 347,79 | 58,90 | 16,99 | 4,00 | 1,92 | 0,85 | 0,87 | 0,79 |
| **15** | 1034,27 | 382,95 | 64,86 | 18,70 | 4,41 | 2,11 | 0,93 | 0,96 | 0,87 |
| **16** | 1090,08 | 403,61 | 68,36 | 19,71 | 4,65 | 2,23 | 0,98 | 1,01 | 0,92 |

График:

# **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^2). Это соответствует теоретическому значению, определенному в 6 лабораторной работе. Таким образом, эксперимент можно считать успешным.

# **Приложение:**

* 1. Код программы:
  2. main.cpp

1. #include <iostream>  
   #include "Hash\_Table.h"  
     
   **int** main(){  
    srand(time(0));  
    Hash A, B, C, D, E, R, B\_and\_C;  
    ofstream fout;  
    fout.open("tests.txt");  
    fout << 20 << endl;  
    **for**(**int** i = 10; i < 50; i = i + 2){  
    A.create('A',i);  
    B.create('B',i);  
    C.create('C',i);  
    D.create('D',i);  
    E.create('E',i);  
    std::chrono::high\_resolution\_clock::time\_point t1 = std::chrono::high\_resolution\_clock::now();  
    R = B & C;  
    R = R | D;  
    R = R + E;  
    R = A / R;  
    R = A.MERGE(B);  
    R = A.CONCAT(B);  
    R = A.SUBST(B, 3);  
    std::chrono::high\_resolution\_clock::time\_point t2 = std::chrono::high\_resolution\_clock::now();  
    **auto** time = std::chrono::duration\_cast<std::chrono::duration<**double**>>(t2 - t1);  
    **if**(time.count() == 0){  
    i = i - 2;  
    } **else** fout << i << ' ' << time.count() << endl;  
    A.del();  
    B.del();  
    C.del();  
    D.del();  
    E.del();  
    R.del();  
    }  
    fout.close();  
    **return** 0;  
   }
   1. Hash\_Table.h
2. #ifndef LAB\_3\_HASH\_TABLE\_H  
   #define LAB\_3\_HASH\_TABLE\_H  
   #endif //LAB\_3\_HASH\_TABLE\_H  
   #include <set>  
   #include <random>  
   #include <string.h>  
   #include <vector>  
   #include <list>  
   #include <unordered\_set>  
   #include <algorithm>  
   #include <ctime>  
   #include <fstream>  
   #include <chrono>  
   **using** std::sort;  
     
   **enum**{*max\_num* = 100};  
   **using namespace** std;  
     
   **class** Hash{  
    **char** Name;  
    unordered\_set<**int**> table;  
    vector<**int**> keys;  
    **int** N = 0, n\_table = 0;  
   **public**:  
    Hash& **operator** = (**const** Hash & B){  
    (\***this**).del();  
    (\***this**).N = B.N;  
    **for**(**auto** i: B.table) table.insert(i);  
    **for**(**auto** i: B.keys) keys.push\_back(i);  
    **return** (\***this**);  
    };  
    **void** create(**char** Name\_set, **int** \_N);  
    **void** Out();  
    **void** Out(string \_name);  
    **void** Out\_id(string \_name);  
    Hash(){};  
    Hash (**const** Hash &);  
    Hash(Hash &&);  
    Hash MERGE(Hash );  
    Hash CONCAT(Hash );  
    Hash SUBST(Hash , **int** k);  
    Hash **operator**& (**const** Hash &B) **const**{  
    Hash R(\***this**);  
    **return** R &= B;  
    };  
    Hash& **operator** &= (**const** Hash &);  
    Hash **operator** | (**const** Hash &B) **const**{  
    Hash R(\***this**);  
    **return** R |= B;  
    };  
    Hash& **operator** |= (**const** Hash &);  
    Hash **operator** + (**const** Hash &B) **const**{  
    Hash R = (\***this**) & B;  
    Hash A = (\***this**) | B;  
    **return** R / B;  
    };  
    Hash **operator** / (**const** Hash &B) **const**{  
    Hash R(\***this**);  
    **return** R /= B;  
    };  
    Hash &**operator** /= (**const** Hash &);  
    **void** del();  
   };  
     
   Hash::Hash(Hash && B):N(B.N), table(std::move(B.table)), keys(std::move(B.keys)) {};  
   Hash::Hash(**const** Hash &B): N(B.N){  
    **for**(**auto** i: B.table) table.insert(i);  
    **for**(**auto** i: B.keys) keys.push\_back(i);  
   }  
     
   **void** Hash::create(**char** Name\_set, **int** \_N){  
    (\***this**).N = \_N;  
    (\***this**).n\_table = 3\*N;  
    (\***this**).Name = Name\_set;  
    **for** (**int** i = 0; i < N; i++){  
    **int** a = rand() % *max\_num*;  
    **while**(table.find(a) != table.**end**()){  
    a = rand() % *max\_num*;  
    }  
    table.insert(a);  
    }  
    **for**(**auto** i: table){  
    keys.push\_back(i);  
    }  
   };  
     
   **void** Hash::del() {  
    (\***this**).table.clear();  
    (\***this**).keys.clear();  
   }  
     
   Hash& Hash::**operator**&=(**const** Hash & B) {  
    **for** (**auto** i: (\***this**).table) {  
    **if**(B.table.find(i) == B.table.**end**()){  
    (\***this**).table.erase(i);  
    }  
    }  
    **return** (\***this**);  
   }  
     
     
   Hash& Hash::**operator**|=(**const** Hash & B) {  
    **for**(**auto** i: B.table){  
    **if**((\***this**).table.find(i) == (\***this**).table.**end**()){  
    (\***this**).table.insert(i);  
    }  
    }  
    **return** (\***this**);  
   }  
     
     
   Hash& Hash::**operator**/=(**const** Hash & B) {  
    **for** (**auto** i: (\***this**).table) {  
    **if**(B.table.find(i) != B.table.**end**()){  
    (\***this**).table.erase(i);  
    }  
    }  
    **return** (\***this**);  
   }  
     
   Hash Hash::MERGE(Hash B) {  
    Hash R(\***this**);  
    R.table.insert(B.table.begin(), B.table.**end**());  
    R.keys.insert(R.keys.**end**(), B.keys.begin(), B.keys.**end**());  
    sort(R.keys.begin(), R.keys.**end**());  
    **return** R;  
   }  
     
   Hash Hash::CONCAT(Hash B) {  
    Hash R(\***this**);  
    R.table.insert(B.table.begin(), B.table.**end**());  
    R.keys.insert(R.keys.**end**(), B.keys.begin(), B.keys.**end**());  
    **return** R;  
   }  
     
   Hash Hash::SUBST(Hash B, **int** k) {  
    Hash R(\***this**);  
    R.table.insert(B.table.begin(), B.table.**end**());  
    R.keys.insert(R.keys.begin() + k, B.keys.begin(), B.keys.**end**());  
    **return** R;  
   }

# **Список литературы**

1)Лабораторная работа №6, *Торопов В.А*, *г. Санкт-Петербург,* 2019г.

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.