**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе №5,6**

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

Тема: «**ПОДДЕРЖКА ПРОИЗВОЛЬНОЙ ПОСЛЕДОВАТЕЛЬНОСТИ В СТРУКТУРЕ ДАННЫХ ДЛЯ МНОЖЕСТВ»**

**И**

**«ИСПОЛЬЗОВАНИЕ СТАНДАРТНОЙ БИБЛИОТЕКИ ШАБЛОНОВ»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 7307 | Шалугин Е.Д. Васильев А.В. |  |
| Преподаватель | Колинько П.Г. |  |

Санкт-Петербург

2019

Оглавление

[Цель работы 3](#__RefHeading___Toc188_4150899979)

[Задание 3](#__RefHeading___Toc190_4150899979)

[Обоснование по выбору размера хэш-таблицы и коэффициентов хэш-функции 3](#__RefHeading___Toc302_2376616454)

[Оценка временной сложности 3](#__RefHeading___Toc967_306870553)

[Результаты работы программы 3](#__RefHeading___Toc969_306870553)

[Код программы 6](#__RefHeading___Toc306_2376616454)

[Вывод 9](#__RefHeading___Toc308_2376616454)

# **Цель работы**

Научиться выбирать правильные способы доработки Хэш-таблиц, чтобы получать эффективные алгоритмы. Научиться применять стандартные библиотеки шаблонов.

**Задание**

Дополнить программу, составленную по теме «Хеш-таблицы» или «Деревья двоичного поиска», операциями над последовательностями. Использовать шаблоны стандартной библиотеки (unordered\_map)

# **Обоснование способа дополнения базовой СД. Обоснование выбора шаблона для СД.**

# Базовая структура данных была дополнена функциями-членами, позволяющими выполнять операции:

1. Слияние (MERGE). Объединение двух упорядоченных последовательностей в третью с сохранением упорядоченности. От операции объединения множеств отличается только возможностью появления дубликатов ключей. Если исходные последовательности не упорядочены, можно после их слияния просто упорядочить результат. Исходный порядок ключей в множествах в результате не сохраняется.

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

3. Размножение (MUL). Последовательность сцепляется сама с собой заданное количество раз.

4. Исключение (EXCL). Вторая последовательность исключается из первой, если она является её частью.

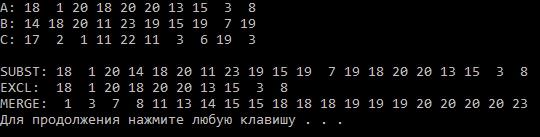
5. Включение (SUBST). Вторая последовательность включается в первую с указанной позиции p. Операция похожа на конкатенацию. Сперва берётся начало первой последовательности до позиции p, затем идёт вторая последовательность, а за ней — остаток первой.

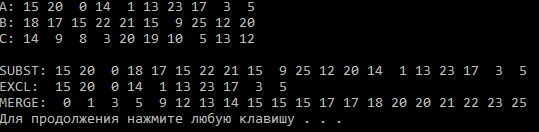
# Так же была произведена перегрузка операторов для более удобной работы со СД.

# unordered\_set не имеет оператора [], поэтому нет возможности осуществлять поэлементный доступ, а так же нету связи ключ-значение, тогда как в unordered\_map есть и то и другое.

# Поскольку существенное отличие unordered\_map от unordered\_multimap заключается в том, что первый шаблон хранит ключ-значение 1 к 1 (т.е. ключи не могут повторяться для разных значений), а второй наоборот может хранить разные значения для одного ключа, а для нас не играет роли это различие, был выбран первый шаблон.

# **Результаты работы программы**





# **Код программы**

1. #include "pch.h"
2. #include <iostream>
3. #include "HashTable.h"
4. #include <ctime>
5. #include <windows.h>
6. using std::cin;
7. using std::cout;
8. using std::endl;
9. void DisplayHashTable(HashTable& table)
10. {
11. for (int i = 0; i < table.size(); ++i)
12. {
13. cout.width(3);
14. cout << table[i];
15. }
16. cout << endl;
17. }
18. int main()
19. {
20. std::srand(unsigned(std::time(0)));
21. HashTable setA(10, 25);
22. cout << "A:";
23. DisplayHashTable(setA);
24. HashTable setB(10, 25);
25. cout << "B:";
26. DisplayHashTable(setB);
27. HashTable setC(10, 25);
28. cout << "C:";
29. DisplayHashTable(setC);
30. HashTable setD(0, 0); //this is for merge
31. cout << endl;
32. cout << "SUBST:";
33. setA.subst(setB,3);
34. DisplayHashTable(setA);
36. cout << "EXCL: ";
37. setA.excl(setB);
38. DisplayHashTable(setA);
39. cout << "MERGE:";
40. setD.merge(setA, setB);
41. DisplayHashTable(setD);
42. //setA.concat(setB);
43. //cout << "1:";
44. //DisplayHashTable(setA);
45. //setA.mul(1);
46. //cout << "2:";
47. //DisplayHashTable(setA);
48. //setA.concat(setC);
49. //cout << "3:";
50. //DisplayHashTable(setA);
51. //setA.excl(setB);
52. //cout << "4:";
53. //DisplayHashTable(setA);
54. system("pause");
55. return 0;
56. }
57. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#pragma once

//#include <map>

#include <list>

#include <unordered\_map>

#include <iostream>

#include <algorithm>

//using std::map;

using std::unordered\_map;

using std::list;

using std::find;

using std::advance;

class HashTable

{

private:

unordered\_map<int, int> values;

int hashfunc(int key) const

{

return key % 1000;

}

void clear()

{

values.clear();

}

int index\_operator(int index) const

{

if (values.size() > index)

{

return values.at(index);

}

throw;

}

void delete\_element(int index)

{

for (int i = index; i < size() - 1; ++i)

values[i] = values[i + 1];

values.erase(size() - 1);

}

public:

HashTable() {}

HashTable(int size, int maxValue)

{

maxValue++;

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

add(rand() % maxValue);

}

HashTable(const HashTable& H)

{

values = H.values;

}

HashTable(HashTable&& H)

{

values = H.values;

}

~HashTable()

{

clear();

}

HashTable& operator=(const HashTable& H)

{

if (this != &H)

{

clear();

values = H.values;

}

return \*this;

}

HashTable& operator=(HashTable&& H)

{

if (this != &H)

{

clear();

values = H.values;

}

return \*this;

}

int operator[](int index) const

{

try

{

return index\_operator(index);

}

catch (...)

{

throw;

}

}

void concat(HashTable H)

{

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

add(H[i]);

}

void mul(int n)

{

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

concat(\*this);

}

void subst(HashTable H, int position) {

int start = 0;

int k = 0;

int prev\_size = size();

int val;

//values.concat(H);

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

add(values[i]);

//std::cout << size() << "!!!!!!!!!!!!!!!!!!";

for (int i = position; i < prev\_size + position/\* + 1\*/; ++i) {

//val = values[i];

values[i] = H.values[k];

//values[i] = values[i + prev\_size];

//values[k + prev\_size] = val;

//values.insert(prev\_size + k, val);

//values.insert(i, H.values[k]);

k++;

}

//values[0] = H.values[0];

//int i = position;

//for (int k = 0; k < H.size(); ++k) {

// //add(1);

// values.emplace(i, H.values[k]);

// i++;

//}

}

//void DisplayHashTable(HashTable& table)

//{

// for (int i = 0; i < table.size(); ++i)

// {

// std::cout.width(3);

// std::cout << table[i];

// }

// std::cout << std::endl;

//}

void merge(HashTable A, HashTable B) {

//std::cout << bothsize << "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!";

//int indexA;

//int indexB;

//int minA;// = 999999999;

//int minB;// = 999999999;

int val;

int sizeA = A.size();

int sizeB = B.size();

int bothsize = sizeA + sizeB;

//std::cout << sizeA << sizeB << "!!!!!!!!!!!!!!!!!!!!!";

for (int i = 0; i < sizeA - 1; ++i) {

for (int j = 0; j < sizeA - i - 1; ++j) {

if (A.values[j] > A.values[j + 1]) {

val = A.values[j];

A.values[j] = A.values[j + 1];

A.values[j + 1] = val;

}

}

}

//DisplayHashTable(A);

for (int i = 0; i < sizeB - 1; ++i) {

for (int j = 0; j < sizeB - i - 1; ++j) {

if (B.values[j] > B.values[j + 1]) {

val = B.values[j];

B.values[j] = B.values[j + 1];

B.values[j + 1] = val;

}

}

}

//DisplayHashTable(B);

int j = 0;

int k = 0;

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

if (B.values[k] >= A.values[j] && j != sizeA) {

add(A.values[j]);

j++;

}

else if (k != sizeB) {

add(B.values[k]);

k++;

}

}

}

void excl(HashTable H)

{

int begin = 0;

int start = 0;

int finish = 0;

while (start < size())

{

for (start = begin; start < size(); ++start)

if (values[start] == H[0])

break;

if (start < size())

{

int next = 0;

finish = start;

do {

next++;

finish++;

} while (next < H.size() && finish < size() && H[next] == values[finish]);

begin = finish;

if (next == H.size())

for (int i = start; i < finish; ++i)

delete\_element(start);

}

}

}

void add(int value)

{

values.insert(std::pair<int, int>(int(values.size()), value));

}

int size() const

{

return int(values.size());

}

unordered\_map<int, int>::const\_iterator begin()

{

return values.begin();

}

unordered\_map<int, int>::const\_iterator end()

{

return values.end();

}

};

# **Вывод**

1. В данной работе мы дополнили нашу СД для возможности работы со множествами и применили к ней шаблон из стандартной библиотеки STL .

В процессе разработки были написаны 5 из 7 функций. Дополнительные 2 функции не просто могут выполнять свою задачу, но так же могут использоваться в других более сложных функциях работы с множествами.

Шаблоны позволили не перегружать операторы (присваивания, произвольного доступа…), что упростило разработку программы, а так же уменьшило объем кода.