Санкт-Петербургский государственный

электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина) (СПбГЭТУ)

Факультет компьютерных технологий и информатики

Кафедра ВТ

Отчет по теме №3

Комбинированные структуры данных и стандартная библиотека шаблонов

Выполнил: студент гр. 1335 Максимов Ю.Е. \_\_\_\_\_\_\_\_\_\_\_

Проверил: старший преподаватель Манирагена В. \_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2024

**Содержание**

1. Техническое задание 3

2. Формализация задания 3

3. Контрольные тесты 4

4. Временная сложность 5

5. Результаты измерения времени 5

Выводы 6 Список используемой литературы 7

Приложение 1 8

Приложение 2 10

Приложение 3 12

Приложение 4 14

1. **Цель работы**

Получить навыки по работе со структурами данных и стандартными библиотеками шаблонов.

1. **Задание**

Реализовать индивидуальное задание темы «Множества и последовательности» в виде программы, используя свой контейнер для заданной структуры данных (хеш-таблицы или одного из вариантов ДДП), и доработать его для поддержки операций с последовательностями. Для операций с контейнером рекомендуется использовать возможности библиотеки алгоритмов. Программа должна реализовывать цепочку операций над множествами, имеющимися в выражении, взятом по номеру варианта задания из табл. 3.1. с базовым контейнером и операциями с последовательностью из табл. 3.2. Результат каждого шага цепочки операций выводится на экран.

|  |  |  |
| --- | --- | --- |
| 17 | 10 | A \ (B ∩ C) \ D ⊕ E |

Для хранения множеств выбрать контейнер подходящего типа и доработать его для поддержки операций с последовательностями. Для реализации операций с контейнерами использовать возможности библиотеки алгоритмов. Программа должна реализовывать цепочку операций с последовательностями.

|  |  |  |
| --- | --- | --- |
| 17 | Хеш-таб | MERGE, SUBST, MUL |

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 последовательностях в результате не сохраняется.
2. 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.
3. Размножение (*MUL*). Последовательность сцепляется сама с собой заданное количество раз.

Код программы находится в Приложении.

1. **Результат работы программы**

B: 0 1 3 4 6 7 9

< 4 9 4 9 0 1 3 6 4 7 >

C: 0 3 5 7 8

< 5 5 7 5 0 3 8 8 7 7 >

=== B&C === (q\_and = 2)

B: 0 3 7

< 0 3 7 > Press <Enter>

A: 1 2 3 4 5 6 7 8 9

< 3 6 5 9 7 8 8 3 1 6 2 4 >

=== A - (B&C) ===

A: 1 2 4 5 6 8 9

< 1 2 4 5 6 8 9 > Press <Enter>

D: 1 2 3 5 7 8 9

< 7 7 9 5 7 5 8 1 3 2 >

=== A - (B&C) - D ===

A: 4 6

< 4 6 > Press <Enter>

E: 1 2 5 6 7 8

< 2 6 7 7 5 8 5 1 8 2 >

=== A – (B&C) – D xor E

A: 1 2 3 4 5 6 7 8 9

< 1 2 3 4 5 6 7 8 9 > Press <Enter>

R: 0 1 2 3 4 7 8 9

< 0 9 1 2 3 7 0 8 1 2 2 4 >

=== R.Mul(F) ===

R: 0 1 2 3 4 7 8 9

< 0 9 1 2 3 7 0 8 1 2 2 4 >

R: 0 1 2 3 4 7 8 9

< 0 9 1 2 3 7 0 8 1 2 2 4 0 9 1 2 3 7 0 8 1 2 2 4 > Press <Enter>

=== R.Merge(F) ===

R: 0 1 2 3 4 7 8 9

< 0 9 1 2 3 7 0 8 1 2 2 4 0 9 1 2 3 7 0 8 1 2 2 4 >

R: 0 1 2 3 4 5 6 7 8 9

< 0 0 0 0 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 4 4 5 5 6 7 7 7 7 8 8 8 8 9 9 > Press <Enter>

=== R.Subst(F) ===

R: 0 1 2 3 4 5 6 7 8 9

< 0 0 0 0 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 4 4 5 5 6 7 7 7 7 8 8 8 8 9 9 >

R: 0 1 2 3 4 5 6 7 8 9

< 0 0 0 0 1 1 1 1 1 2 6 7 7 5 8 5 1 8 2 2 2 2 2 2 2 2 2 3 3 4 4 5 5 6 7 7 7 7 8 8 8 8 9 9 > Press <Enter>

1. **Описание контейнера**

Контейнер состоит из множества А мощностью power и массива итераторов sA. Множество A представлено в форме контейнера стандартной библиотеки шаблонов set. Данная структура данных хранит множество в виде дерева двоичного поиска с автобалансировкой, причём контейнер set хранит множество ключей.

Также для реализации функций над последовательностями был использован контейнер вектор указателей sA на значения элементов множества.

Из библиотеки algorithm были использованы методы merge для слияния двух последовательностей.

1. **Временная сложность**

Временная сложность алгоритмов STL представлена в таблице 1.

Временная сложность функций программы представлена в таблице 2.

Таблица 1. Временная сложность алгоритмов STL.

|  |  |
| --- | --- |
| *Функция* | *Временная сложность* |
| vector::push\_back | O(1) |
| set::insert | O(ln n) |
| swap | O(1) |
| size | O(1) |
| merge | O(NlogN) |

Таблица 2. Временная сложность функций программы.

|  |  |
| --- | --- |
| *Функция* | *Средняя* |
| & | O(n) |
| | | O(n) |
| ^ | O(3\*n) |
| Mul | O(n) |
| Merge | O(n) |
| Subst | O(n) |

1. **Сравнение различных структур**

Для замера время выполнения кода используем библиотеку <chrono>.

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

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

auto dt = duration\_cast<duration<double>>(t2 - t1);

|  |  |
| --- | --- |
| Название структуры | Среднее время выполнение вычисления (кол-во воспроизведений 10) |
| MyCont(хеш-таблица) | T=0.0229568 |
| std::set(двоичное дерево поиска) | T=0.0254127 |
| 1-2 дерево | T=0.0342917 |

Пример расчетов среднего времени по MyCont.

(0.022454 + 0.022653 + 0.022126 + 0.022421 + 0.022465+ 0.022396 + 0.022392 + 0.022454 + 0.022467 +0.022459)/10 = 0.022454

1. **Вывод**

Как мы можем видеть, хеш-таблица прекрасно справляется с множеством(учитывая время исполнения кода), так как в самом лучшем случае вставка, извлечение, поиск справляется за O(1), пока не происходит коллизия, тут уже зависит от структуры хеш-таблицы, но чаще всего в бакет засовывают голову листа, и тогда уже вхождение будет за O(n) кол-во узлов списка.

1. **Список используемой литературы**

1. Колинько П. Г. Алгоритмы и структуры данных. Часть 2: Пользовательские контейнеры. Вып. 2020 (для заочников). –– СПб., 2020. — 60 с.

**7. Приложение**

#include **<iostream>**#include **<algorithm>**#include **<set>**#include **<ctime>**#include **<iterator>**#include **<chrono>**#include **<vector>  
  
using namespace** std;  
**using** MySet = std::set<**int**>;  
**using** MyIt = std::set<**int**>::iterator;  
**using** MySeq = std::vector<MyIt>;  
  
**const int** lim = 10; *//ОГРАНИЧИТЕЛЬ для множества ключей***class** MyCont {  
 **int** power;  
 **char** tag;  
 MySet A;  
 MySeq sA;  
  
 MyCont &**operator**=(**const** MyCont &) = **delete**;  
 MyCont &**operator**=(MyCont &&) = **delete**;  
**public**:  
 MyCont(**int**, **char**);  
 MyCont(**const** MyCont &);  
 MyCont(MyCont &&);  
 MyCont &**operator**|=(**const** MyCont &); *// объединение* MyCont **operator**|(**const** MyCont &rgt) **const** {  
 MyCont result(\***this**);  
 **return** (result |= rgt);  
 }  
 MyCont &**operator**&=(**const** MyCont &); *// пересечение* MyCont **operator**&(**const** MyCont &rgt) **const** {  
 MyCont result(\***this**);  
 **return** (result &= rgt);  
 }  
 MyCont &**operator**-=(**const** MyCont &); *// разность* MyCont **operator**-(**const** MyCont &rgt) **const** {  
 MyCont result(\***this**);  
 **return** (result -= rgt);  
 }  
 MyCont& **operator** ^= (**const** MyCont & rgt); *// исключающее или  
  
 // последовательности* **void** Merge(**const** MyCont &);  
 **void** Concat(**const** MyCont &);  
 **void** Mul(**int**);  
 **void** Erase(size\_t, size\_t);  
 **void** Excl(**const** MyCont &);  
 **void** Subst(**const** MyCont &, size\_t);  
 **void** Change(**const** MyCont &, size\_t);  
 **void** Show() **const**;  
  
 size\_t Power() **const** { **return** sA.size(); }  
  
 **void** PrepareExcl(**const** MyCont &); *// Подготовка excl* **friend void** PrepareAnd(MyCont &, MyCont &, **int**); *// подготовка and и sub*};  
  
MyCont::MyCont(**int** p, **char** t = **'R'**) : power(p), tag(t) {  
 **for** (**int** i = 0; i < power; ++i) {  
 sA.push\_back(A.insert(std::rand() % lim).first);  
 }  
}  
  
MyCont::MyCont(MyCont &&source) *// Копия "с переносом"* : power(source.power), tag(source.tag),  
 A(std::move(source.A)), sA(std::move(source.sA)) {}  
  
MyCont::MyCont(**const** MyCont &source) *// Конструктор копии* : power(source.power), tag(source.tag) {  
 **for** (**auto** x : source.A) sA.push\_back(A.insert(x).first);  
}  
  
**void** MyCont::Show() **const** {  
 **using** std::cout;  
 cout << **"\n"** << tag << **": "**;  
  
 **for** (**auto** x : A) cout << x << **" "**; *// Выдача множества* cout << **"\n < "**;  
 **for** (**auto** x : sA) cout << \*x << **" "**; *// Выдача последовательности* cout << **">"**;  
}  
  
**void** PrepareAnd(MyCont &first, MyCont &second, **const int** quantity) {  
 **for** (**int** i = 0; i < quantity; ++i) { *// Подготовка пересечения:* **int** x = rand() % lim; *// добавление общих эл-тов* first.sA.push\_back(first.A.insert(x).first);  
 second.sA.push\_back(second.A.insert(x).first);  
 }  
}  
  
MyCont &MyCont::**operator**-=(**const** MyCont &rgt) { *// Разность мн-в* MySet temp;  
 MySeq stemp;  
 **for** (**auto** x : A)  
 **if** (rgt.A.find(x) == rgt.A.end())  
 stemp.push\_back(temp.insert(x).first);  
 temp.swap(A);  
 stemp.swap(sA);  
 **return** \***this**;  
}  
  
MyCont &MyCont::**operator**&=(**const** MyCont &rgt) { *// Пересечение* MySet temp;  
 MySeq stemp;  
 **for** (**auto** x : A)  
 **if** (rgt.A.find(x) != rgt.A.end())  
 stemp.push\_back(temp.insert(x).first);  
 temp.swap(A);  
 stemp.swap(sA);  
 **return** \***this**;  
}  
  
MyCont &MyCont::**operator**|=(**const** MyCont &rgt) { *// Объединение* **for** (**auto** x : rgt.A) sA.push\_back(A.insert(x).first);  
 **return** \***this**;  
}  
  
MyCont& MyCont::**operator**^=( **const** MyCont & rgt) { *// Исключающее ИЛИ* MySet temp;  
 MySeq stemp;  
  
 MySet R;  
 MySeq sR;  
 *// пересечь* **for** (**auto** x : A)  
 **if** (rgt.A.find(x) != rgt.A.end())  
 sR.push\_back(R.insert(x).first);  
  
 *// объединить* **for** (**auto** x : rgt.A) sA.push\_back(A.insert(x).first);  
 *// вычесть* **for** (**auto** x : A)  
 **if** (R.find(x) == R.end())  
 stemp.push\_back(temp.insert(x).first);  
  
 temp.swap(A);  
 stemp.swap(sA);  
 **return** \***this**;  
}  
  
**void** MyCont::Erase(size\_t p, size\_t q) { *// Исключение фр-та от p до q* **using** std::min;  
 size\_t r(Power());  
 p = min(p, r);  
 q = min(q + 1, r);  
 **if** (p <= q) {  
 MySet temp;  
 MySeq stemp;  
 **for** (size\_t i = 0; i < p; ++i)  
 stemp.push\_back(temp.insert(\*sA[i]).first);  
 **for** (size\_t i = q; i < r; ++i)  
 stemp.push\_back(temp.insert(\*sA[i]).first);  
 A.swap(temp);  
 sA.swap(stemp);  
 }  
}  
  
**void** MyCont::Mul(**int** k) { *// Размножение (не более чем в 5 раз)* **auto** p = sA.begin(), q = sA.end();  
 **if** (p != q && (k = k % 5) > 1) { *//П ропуск, если мн-во пусто или k < 2* std::vector<**int**> temp(A.begin(), A.end());  
 MySeq res(sA);  
 **for** (**int** i = 0; i < k - 1; ++i) {  
 std::copy(p, q, back\_inserter(res));  
 A.insert(temp.begin(), temp.end());  
 }  
 sA.swap(res);  
 }  
}  
  
**void** MyCont::Merge(**const** MyCont &rgt) { *// Слияние* **using** std::sort;  
 MySeq temp(rgt.sA), res;  
 **auto** le = [](MyIt a, MyIt b) -> **bool** { **return** \*a < \*b; };*// Критерий* sort(sA.begin(), sA.end(), le);  
 sort(temp.begin(), temp.end(), le);  
 std::merge(sA.begin(), sA.end(), temp.begin(), temp.end(),  
 std::back\_inserter(res), le); *// Слияние для последовательностей...* A.insert(rgt.A.begin(), rgt.A.end()); *// ... и объединение множеств* sA.swap(res);  
}  
  
**void** MyCont::PrepareExcl(**const** MyCont &rgt) {  
 *// Подготовка объекта исключения в пустом контейнере...* **int** a = rand() % rgt.Power(), b = rand() % rgt.Power();  
 *// ... из случайного [a, b] отрезка rgt* **if** (b > a) {  
 **for** (**int** x = a; x <= b; ++x) {  
 **int** y = \*(rgt.sA[x]);  
 sA.push\_back(A.insert(y).first);  
 }  
 }  
}  
  
**void** MyCont::Excl(**const** MyCont &rgt) { *// Исключение подпоследовательности* size\_t n(Power()), m(rgt.Power());  
 **if** (m)  
 **for** (size\_t p = 0; p < n; ++p) { *// Поиск первого элемента* **bool** f(**true**);  
 *// int a(\*sA[p]), b(\*rgt.sA[0]); //ОТЛАДКА* **if** (\*sA[p] == \*rgt.sA[0]) { *// Проверка всей цепочки* size\_t q(p), r(0);  
 **if** (m > 1)  
 **do** {  
 ++q, ++r;  
 size\_t c(\*sA[q]), d(\*rgt.sA[r]);  
 f &= c == d;  
 } **while** ((r < m - 1) && f);  
 **if** (f) { *// Цепочки совпали, удаляем* MySet temp;  
 MySeq stemp;  
 **for** (size\_t i = 0; i < p; ++i)  
 stemp.push\_back(temp.insert(\*sA[i]).first);  
 **for** (size\_t i = p + m; i < Power(); ++i)  
 stemp.push\_back(temp.insert(\*sA[i]).first);  
 A.swap(temp);  
 sA.swap(stemp);  
 **break**;  
 }  
 }  
 }  
}  
  
**void** MyCont::Concat(**const** MyCont &rgt) { *// Сцепление* **for** (**auto** x : rgt.sA) sA.push\_back(A.insert(\*x).first);  
}  
  
**void** MyCont::Subst(**const** MyCont &rgt, size\_t p) { *// Подстановка* **if** (p >= Power()) Concat(rgt);  
 **else** {  
 MySeq stemp(sA.begin(), sA.begin() + p); *// Начало* std::copy(rgt.sA.begin(), rgt.sA.end(), back\_inserter(stemp)); *// Вставка* std::copy(sA.begin() + p, sA.end(), back\_inserter(stemp)); *// Окончание* MySet temp;  
 sA.clear();  
 **for** (**auto** x : stemp) sA.push\_back(temp.insert(\*x).first);  
 A.swap(temp);  
 }  
}  
  
**void** MyCont::Change(**const** MyCont &rgt, size\_t p) { *// Замена* **if** (p >= Power()) Concat(rgt);  
 **else** {  
 MySeq stemp(sA.begin(), sA.begin() + p); *// Начало* std::copy(rgt.sA.begin(), rgt.sA.end(), back\_inserter(stemp));  
 *// Замена* size\_t q = p + rgt.Power();  
 **if** (q < Power())  
 std::copy(sA.begin() + q, sA.end(), back\_inserter(stemp)); *// Окончание* MySet temp;  
 sA.clear();  
 **for** (**auto** x : stemp) sA.push\_back(temp.insert(\*x).first);  
 A.swap(temp);  
 }  
}  
  
**int** main() {  
 **using** std::cout;  
 **using namespace** std::chrono;  
 srand((**unsigned int**) 7); *// Пока здесь константа, данные повторяются  
 //srand((unsigned int)time(nullptr)); //Разблокировать для случайных данных* **bool** debug = **true**; *// false, чтобы запретить отладочный вывод* **auto** MaxMul = 5;  
 **int** middle\_power = 0, set\_count = 0;  
  
 **auto** Used = [&](MyCont &t) {  
 middle\_power += t.Power();  
 ++set\_count;  
 };  
  
 **auto** DebOut = [debug](MyCont &t) {  
 **if** (debug) {  
 t.Show();  
 cout << **" Press <Enter>"** << endl;  
 *// cin.get();* }  
 };  
  
 **auto** rand = [](**int** d) {  
 **return** std::rand() % d;  
 }; *// Лямбда-функция!* **int** p = rand(20) + 1; *// Текущая мощность (место для цикла по p)  
 //=== Данные ===* MyCont A(p, **'A'**);  
 MyCont B(p, **'B'**);  
 MyCont C(p, **'C'**);  
 MyCont D(p, **'D'**);  
 MyCont E(p, **'E'**);  
 MyCont R(p);  
  
 **int** q\_and(rand(MaxMul) + 1);  
 PrepareAnd(A, R, q\_and);  
 **if** (debug) B.Show(); Used(B);  
 **if** (debug) C.Show(); Used(C);  
  
 *// === Цепочка операций ===  
 // (Операция пропускается (skipped!), если аргументы некорректны)  
 // Идёт суммирование мощностей множеств и подсчёт их количества,  
 // измеряется время выполнения цепочки  
 // auto t1 = std::chrono::high\_resolution\_clock::now();  
  
 // 10 A \ (B ∩ C) \ D ⊕ E* **if** (debug) cout << endl << **"=== B&C === (q\_and = "** << q\_and << **") "**;  
 B &= C; DebOut(B); Used(B);  
  
 **if** (debug) A.Show(); Used(A);  
 **if** (debug) cout << endl << **"=== A - (B&C) ==="**;  
 A -= B; DebOut(A); Used(A);  
  
 **if** (debug) D.Show(); Used(D);  
 **if** (debug) cout << endl << **"=== A - (B&C) - D ==="**;  
 A -= D; DebOut(A); Used(D);  
  
 **if** (debug) E.Show(); Used(E);  
 **if** (debug) cout << endl << **"=== A – (B&C) – D xor E"**;  
 A ^= D; DebOut(A); Used(A);  
  
 *//merge, subst, mul* R.Show(); Used(R);  
  
 **if** (debug) cout << endl << **"=== R.Mul(F) ==="**;  
 R.Show(); Used(E);  
 R.Mul(2); DebOut(R); Used(R);  
  
 **if** (debug) cout << endl << **"=== R.Merge(F) ==="**;  
 R.Show(); Used(E);  
 R.Merge(E); DebOut(R); Used(R);  
  
 **int** e = rand(R.Power());  
 **if** (debug) cout << endl << **"=== R.Subst(F) ==="**;  
 R.Show(); Used(E);  
 R.Subst(E, e); DebOut(R); Used(R);  
  
 *// auto t2 = std::chrono::high\_resolution\_clock::now();  
 // auto dt = duration\_cast<duration<double>>(t2 - t1);  
 // middle\_power /= set\_count;  
 // cout << "\n=== Ready === \n(Power = " << p << " Set\_count = " << set\_count << " Middle\_power = " << middle\_power << " DT=" << (dt.count()) << ")\n";  
  
 // cin.get();* **return** 0;  
}