# Сравнение коллекций map и unoredred\_map

## Сравнение методов

### Список общих методов

* *begin*
* *cbegin*
* *merge*
* *count*
* *at*
* *emplace*
* *empty*
* *swap*
* *get\_allocator*
* *erase*
* *emplace\_hint*
* *insert*
* *extract*
* *find*
* *size*
* *try\_emplace*
* *insert\_or\_assign*
* *max\_size*
* *contains*
* *end*
* *cend*
* *equal\_range*
* *clear*
* *operator[]*
* *operator=*

### Таблица индивидуальных методов

|  |  |
| --- | --- |
| ***map*** | ***unordered\_map*** |
| *key\_comp*  *rbegin*  *crbegin*  *lower\_bound*  *rend*  *crend*  *upper\_bound*  *value\_comp* | *bucket\_size*  *max\_bucket\_count*  *reserve*  *bucket*  *max\_load\_factor*  *key\_eq*  *bucket\_count*  *load\_factor*  *rehash*  *hash\_function* |

## Функции для измерений

Для автоматизации процесса измерения методов коллекций были написаны вспомогательные функции: *timeit*, *average*, *test*.

### Функция timeit

Листинг 1

***Функция timeit***

int timeit() {

static bool is\_started = false;

static chrono::steady\_clock::time\_point start, end;

if (is\_started) {

end = chrono::steady\_clock::now();

is\_started = false;

return (end - start) / 1ms;

}

start = chrono::steady\_clock::now();

is\_started = true;

return 0;

}

### Функция average

Листинг 2

***Функция average***

int average(int (\*func)(int, int), int num, int seed) {

std::vector<int> vec;

int common\_time = 0;

for (int i = 0; i < 5; i++) {

do {

int tmp = func(num, seed);

common\_time += tmp;

vec.push\_back(tmp);

} while(common\_time < 500);

}

std::sort(vec.begin(), vec.end());

int \_sum = 0;

int count = 0;

int size = vec.size();

int slice = size / 10;

slice = slice == 0 ? 1 : slice;

for (int i = slice; i < size - slice; i++, count++) {

\_sum += vec[i];

}

return \_sum / count;

}

### Функция test

Листинг 3

***Функция test***

void test(

std::fstream& file,

std::string collection,

std::string operation,

int (\*func)(int, int)

) {

int seed = 333;

int elems\_num[4] = { 100'000, 100'000 \* 10, 100'000 \* 30, 100'000 \* 50 };

for (auto num : elems\_num) {

file << "collection " << collection << std::endl;

file << "operation " << operation << std::endl;

file << "elems\_num " << num << std::endl;

file << "time " << average(func, num, seed) << std::endl;

file << std::endl;

seed++;

}

}

## Сравнение показателей

### Метод merge

Листинг 4

***Метод merge***

template<typename T>

int merge(int num, int seed) {

T m1 = create<T>(num, seed);

T m2 = create<T>(num, seed+1);

timeit();

m1.merge(m2);

return timeit();

}

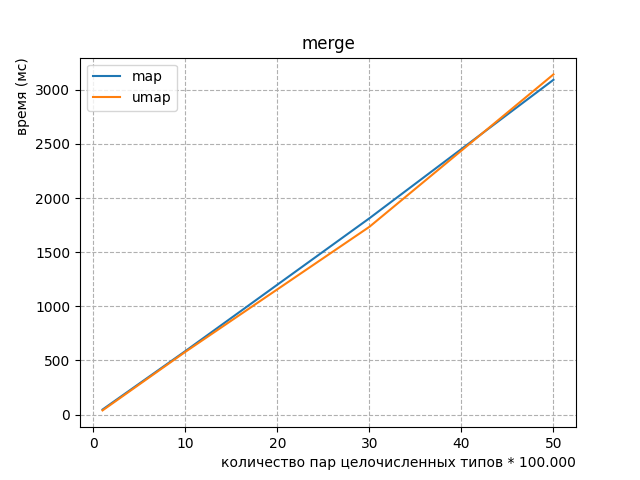
-

Рис. 1. Сравнение показателей метода merge

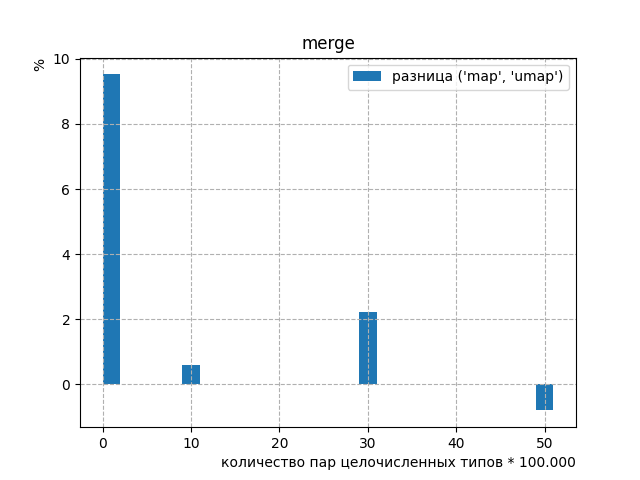


Рис. 2. Относительное сравнение показателей метода merge

### Итерирование

Листинг 5

***Итерирование***

template<typename T>

int forward\_iteration(int num, int seed) {

T m = create<T>(num, seed);

timeit();

for (auto i = m.begin(); i != m.end(); i++) {

int tmp = i->second;

int res\_tmp = tmp \* tmp;

}

return timeit();

}

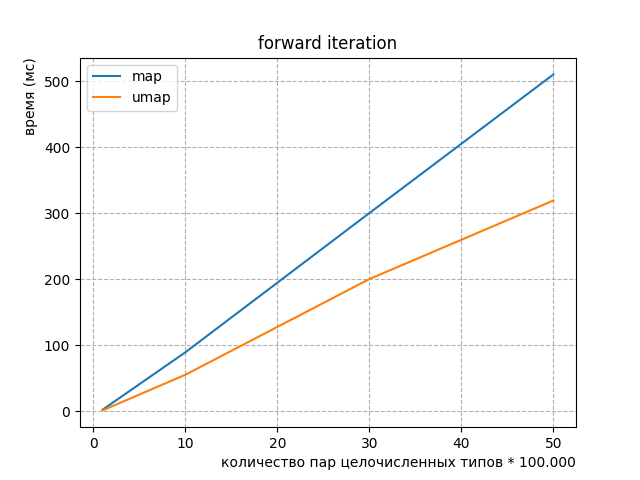


Рис. 3. Сравнение показателей итерирования

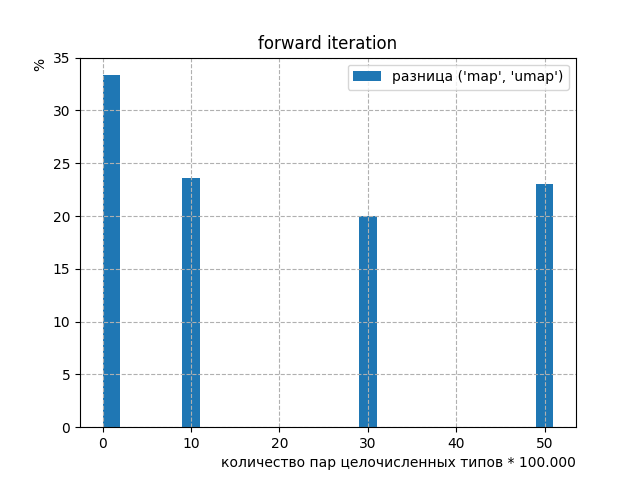


Рис. 4. Относительное сравнение показателей итерирования

### Метод insert

Листинг 6

***insert***

template<typename T>

int insert(int num, int seed) {

T m1 = create<T>(num, seed);

T m2 = create<T>(num, seed);

auto it = m2.begin();

for (int i = 0; i < 50'000; i++, it++);

timeit();

m1.insert(it, m2.end());

return timeit();

}

Листинг 7

***insert***

template<typename T>

int insert\_strings(int num, int seed) {

T m1 = create\_by\_strings<T>(num, seed);

T m2 = create\_by\_strings<T>(num, seed);

auto it = m2.begin();

for (int i = 0; i < 5'000; i++, it++);

timeit();

m1.insert(it, m2.end());

return timeit();

}

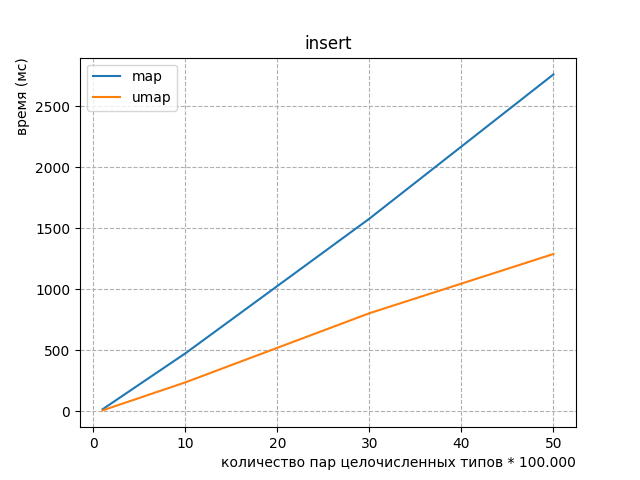


Рис. 5. Сравнение показателей метода insert

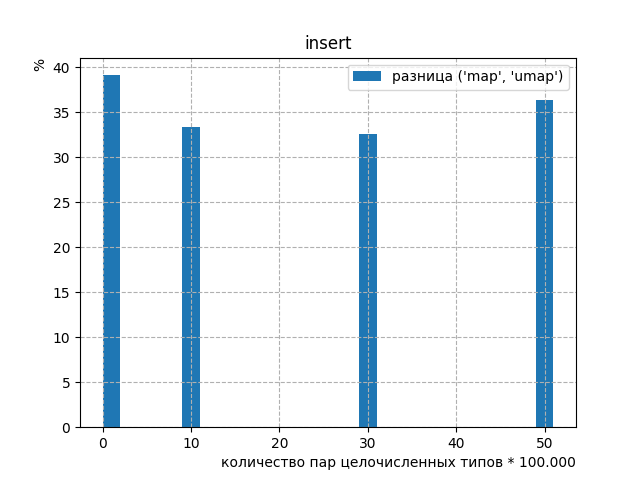


Рис. 6. Относительное сравнение показателей метода insert

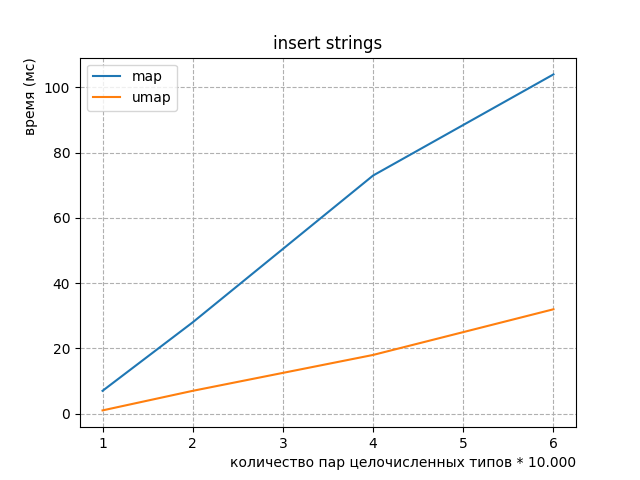


Рис. 7. Cравнение показателей метода insert для строк

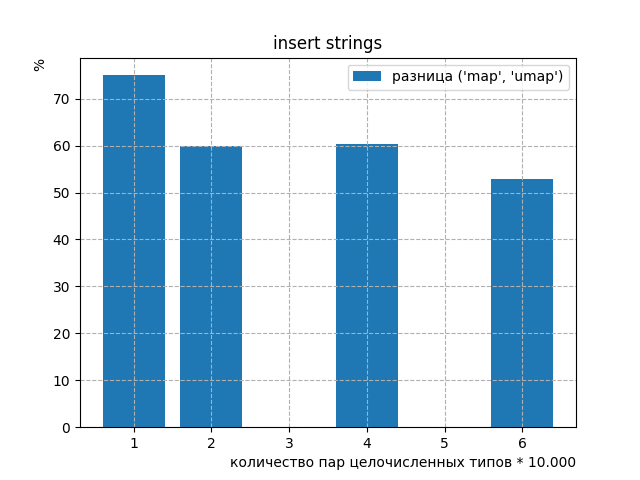


Рис. 8. Относительное сравнение показателей метода insert для строк

### Метод find

Листинг 8

***find***

template<typename T>

int find(int num, int seed) {

T m = create<T>(num, seed);

vector<int> finds;

for (auto it = m.begin(); finds.size() < 50'000;) {

finds.push\_back(it->first);

int tmp = rand() % 3 + 1;

for (int j = 0; j < tmp; j++) {

it++;

}

}

for (auto k : finds) {

int v = m.find(k)->second;

int tmp = v \* v;

}

return timeit();

}

Листинг 9

***find\_strings***

template<typename T>

int find\_strings(int num, int seed) {

T m = create\_by\_strings<T>(num, seed);

vector<string> finds;

for (auto it = m.begin(); finds.size() < 1'000;) {

finds.push\_back(it->first);

int tmp = rand() % 10 + 1;

for (int j = 0; j < tmp; j++) {

it++;

}

}

timeit();

for (int i = 0; i < 10; i++) {

for (auto k : finds) {

string v = m.find(k)->first;

v = v + " like something useful for compiler";

}

}

return timeit();

}

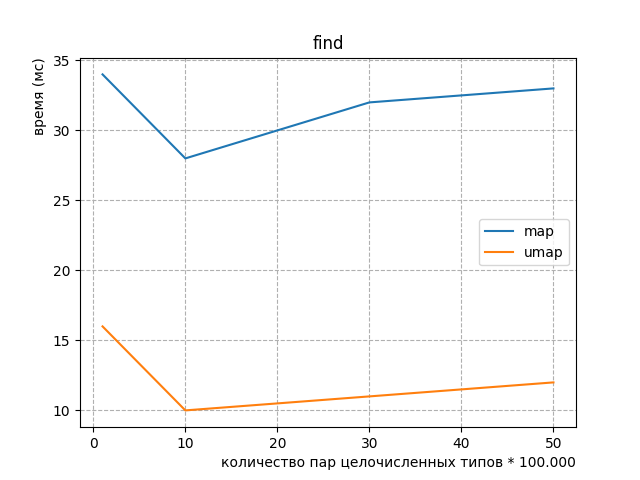


Рис. 9. Сравнение показателей методов find

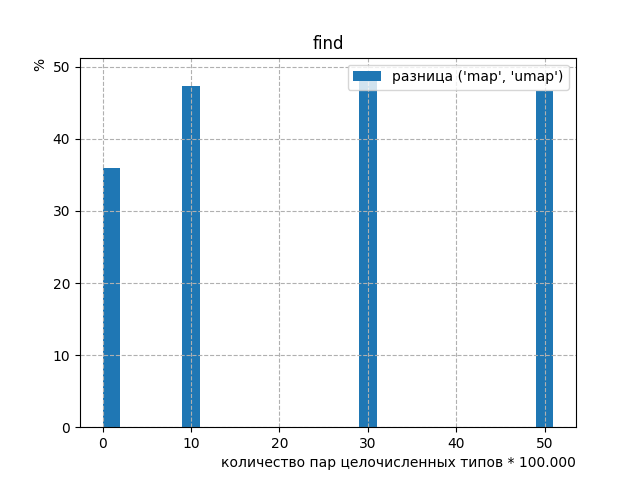


Рис. 10. Относительное сравнение показателей метода find

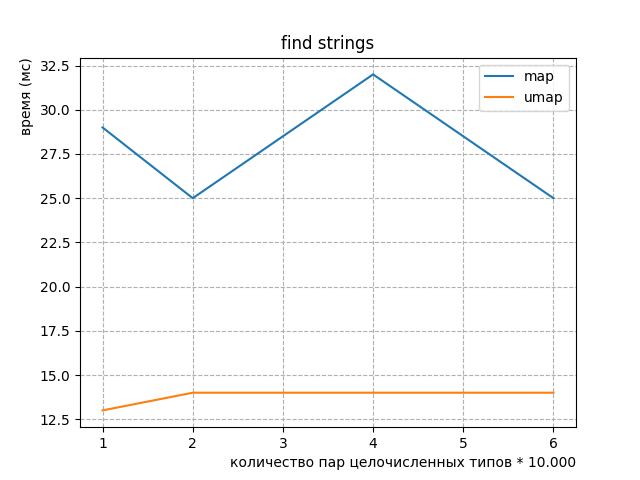


Рис. 11. Сравнение показателей метода find для строк

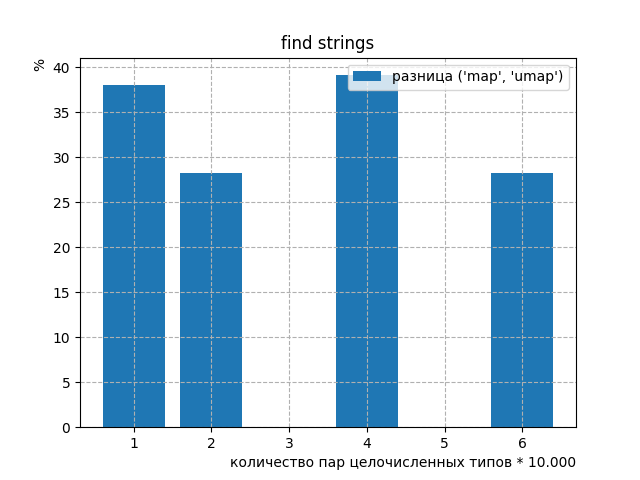


Рис. 12. Относительное сравнение показателей метода find для строк

### Метод erase

Листинг 10

***erase***

template<typename T>

int erase(int num, int seed) {

T m = create<T>(num, seed);

auto it = m.begin();

for (int i = 0; i < 50'000; i++, it++);

timeit();

m.erase(it, m.end());

return timeit();

}

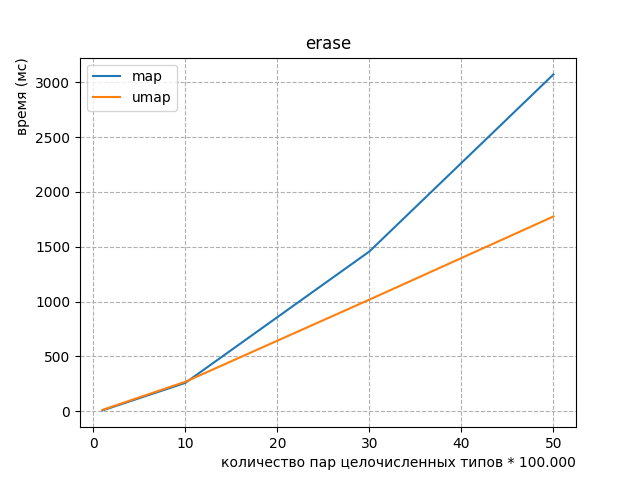


Рис. 13. Сравнение показателей методов erase

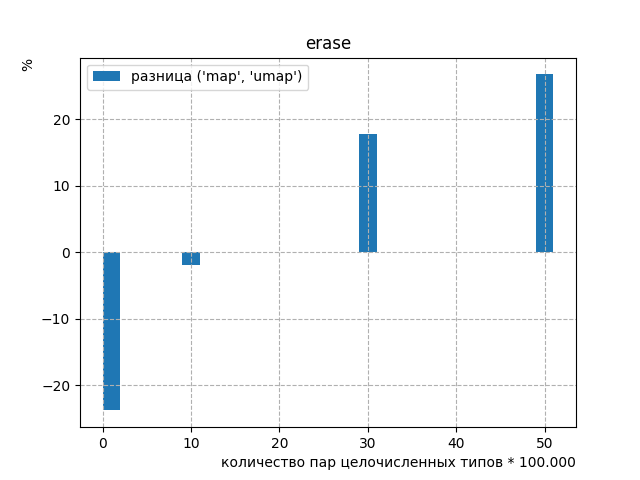


Рис. 14. Относительное сравнение показателей метода erase

### Метод clear

Листинг 11

***clear***

template<typename T>

int clear(int num, int seed) {

T m = create<T>(num, seed);

timeit();

m.clear();

return timeit();

}

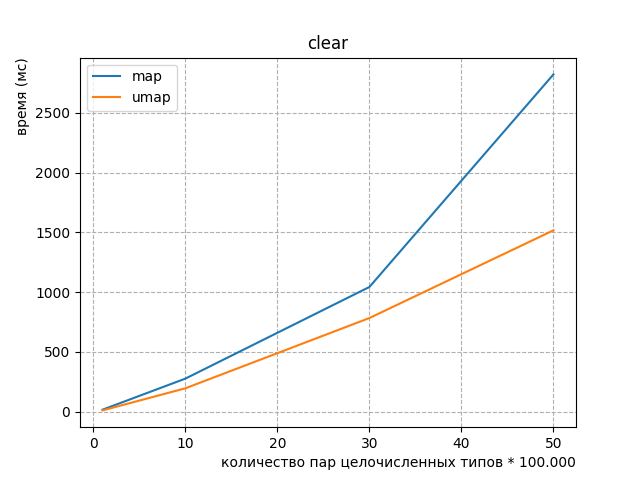


Рис. 15. Сравнение показателей методов clear

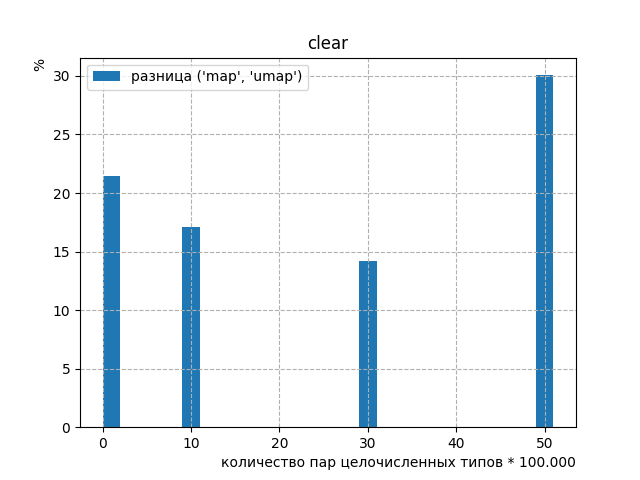


Рис. 16. Относительное сравнение показателей метода clear

### Метод copy

Листинг 12

***copy***

template<typename T>

int copy(int num, int seed) {

T m1 = create<T>(num, seed);

timeit();

T m2 = m1;

return timeit();

}

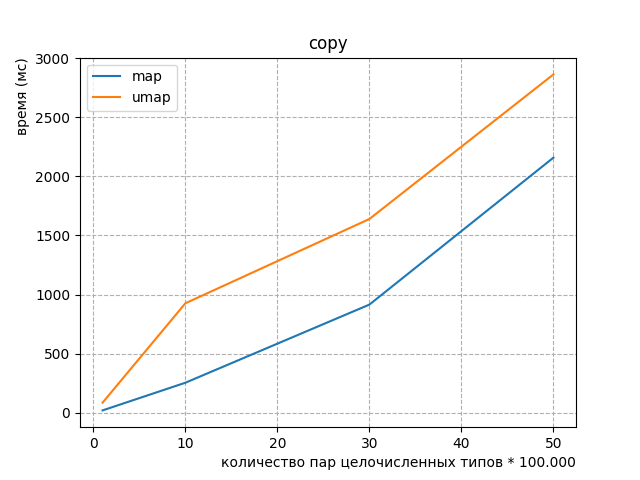


Рис. 17. Сравнение показателей методов copy

## Вывод

Коллекция *unordered\_map* в основном превосходит *map*. Только операция копирования является исключением в рассматриваемых тестах. Однако стоит иметь ввиду, что при больших объёмах данных у коллекции *unorederd\_map*, использующую хеширование, может снизиться эффективность. С коллекцией *map* такого не случится ввиду использования бинарного дерева.