МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

# «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. Шухова»

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

# Лабораторная работа № 13

По дисциплине

Основы программирования

По теме: «Множества»

Выполнил: ст. группы КБ – 231

Давыденко Кирилл Иванович



Белгород, 2023 г.

**Цель работы:** закрепление навыков работы со структурами, изучение простых способов представления множеств в памяти ЭВМ.

**Структура отчёта:**

Структура проекта / CMakeList………………………………….……3

Содержание array………………………………………………………4

Содержание bitset………………………………………………………8

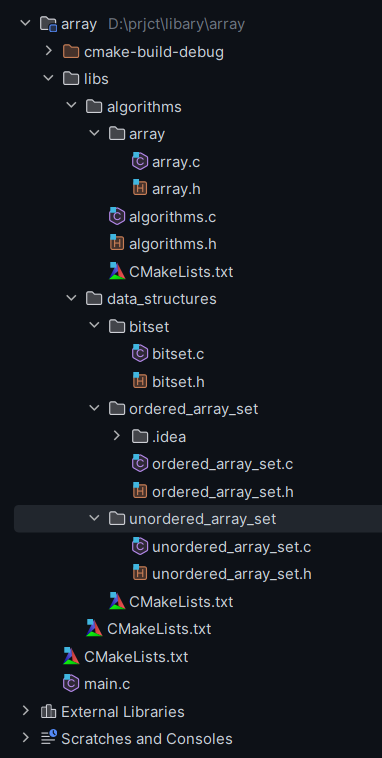
Содержание ordered\_array\_set…………………………………………11

Содержание unordered\_array\_set ……………...………………………17

Тест……………...………………………………………………………21

Задачи с Codeforces.……………………………………………………42

Вывод……………………………………………………………………55

Структура проекта:  


CMakeList проекта:

cmake\_minimum\_required(VERSION 3.26)  
project(array C)  
  
set(CMAKE\_C\_STANDARD 11)  
  
add\_executable(array main.c  
 libs/algorithms/array/array.c  
 libs/algorithms/array/array.h  
 libs/algorithms/algorithms.c  
 libs/algorithms/algorithms.h)  
  
add\_subdirectory(libs)  
target\_link\_libraries(array data\_structures)  
target\_link\_libraries(array algorithms)

CMakeList папки libs:

add\_subdirectory(algorithms)  
add\_subdirectory(data\_structures)

CMakeList папки algorithms:

add\_library(algorithms  
 algorithms.c  
 array/array.c  
)

CMakeList папки data\_structures:

add\_library(data\_structures  
 bitset/bitset.c  
 bitset/bitset.h  
 unordered\_array\_set/unordered\_array\_set.h  
 unordered\_array\_set/unordered\_array\_set.c  
 ordered\_array\_set/ordered\_array\_set.c  
 ordered\_array\_set/ordered\_array\_set.h  
)

Содержание array.h:

#ifndef ARRAY\_ARRAY\_H  
#define ARRAY\_ARRAY\_H  
  
#include <stddef.h>  
  
  
// ввод массива a размера n  
void inputArray\_(int a[], size\_t n);  
  
// вывод массива a размера n  
void outputArray\_(const int a[], size\_t n);  
  
// возвращает индекс первого вхождения элемента x  
// в массиве a размера n при наличии, иначе n  
size\_t linearSearch\_(const int a[], const size\_t n, int x);  
  
// возвращает индекс первого вхождения элемента x  
// в отсортированном массиве а размера n при наличии, иначе n  
size\_t binarySearch\_(const int a[], const size\_t n, int x);  
  
// возвращает индекс первого элемента равного или большего x  
// в отсортированном массиве a размера n  
// при отсутствии возврощает n  
size\_t binarySearchMoreOrEqual\_(const int a[], const size\_t n, int x);  
  
// вставка элемента со значением value  
// в массив а размера n на позицию pos  
void insert\_(int a[], size\_t\* n, size\_t pos, int value);  
  
// вставка элемента со значением value  
// в конец массива a размера n  
void append\_(int a[], size\_t\* n, int value);  
  
// удаление из массива a размера n элемента на позиции pos  
// с сохранением порядка оставшихся элементов  
void deleteByPosSaveOrder\_(int a[], size\_t\* n, size\_t pos);  
  
// удаление из массива a размера n элемента на позиции pos  
// без сохранения порядка оставшихся элементов  
// размер массива a уменьшается на единицу  
void deleteByPosUnsaveOrder\_(int a[], size\_t\* n, size\_t pos);  
  
// возвращает истина, если все элементы массива а размера n  
// удовлетворяют функции-предикату predicate  
// иначе ложь  
int all\_(const int a[], size\_t n, int (\*predicate)(int));  
  
// возвращает истина, если один элемент массива а размера n  
// удовлетворяют функции-предикату predicate  
// иначе ложь  
int any\_(const int a[], size\_t n, int (\*predicate)(int));  
  
// применяет фукнцию-предикат predicate ко всем элементам массива source  
// сохраняет результат в массив dest размера n  
void forEach\_(const int source[], int dest[], size\_t n, int(\*predicate)(int));  
  
// возвращает количество элементов массива a размера n  
// удовлетворяющих функции-предикату predicate  
int countIf\_(const int a[], size\_t n, int(\*predicate)(int));  
  
// удаляет из массива a размера n все элементы, соответсвующие  
// функции-предикату deletePredicate, записывает в n  
// новый размер массива  
void deleteIf\_(int a[], size\_t\* n, int(\*deletePredicate)(int));  
  
#endif //ARRAY\_ARRAY\_H

Содержание array.c:

#include <stdio.h>  
#include <limits.h>  
#include <assert.h>  
#include "array.h"  
  
  
void inputArray\_(int a[], const size\_t n) {  
 for (size\_t i = 0; i < n; i++)  
 scanf("%d", a + i);  
}  
  
  
void outputArray\_(const int a[], const size\_t n) {  
 for (size\_t i = 0; i < n; i++)  
 printf("%d ", \*(a + i));  
 printf("\n");  
}  
  
  
size\_t linearSearch\_(const int a[], const size\_t n, int x) {  
 for (size\_t i = 0; i < n; i++)  
 if (a[i] == x)  
 return i;  
 return n;  
}  
  
  
size\_t binarySearch\_(const int a[], const size\_t n, int x) {  
 if (a[0] > x || n == 0)  
 return n;  
  
 size\_t left = 0;  
 size\_t right = n - 1;  
  
 while (left <= right) {  
 size\_t middle = left + (right - left) / 2;  
  
 if (a[middle] < x)  
 left = middle + 1;  
 else if (a[middle] > x)  
 right = middle - 1;  
 else  
 return middle;  
 }  
  
 return n;  
}  
  
  
size\_t binarySearchMoreOrEqual\_(const int a[], const size\_t n, int x) {  
 if (a[0] >= x)  
 return 0;  
  
 size\_t left = 0;  
 size\_t right = n;  
  
 while (right - left > 1) {  
 size\_t middle = left + (right - left) / 2;  
  
 if (a[middle] < x)  
 left = middle;  
 else  
 right = middle;  
 }  
  
 return right;  
}  
  
  
void insert\_(int a[], size\_t\* n, size\_t pos, int value) {  
 assert(pos < \*n);  
  
 if (\*n != 0) {  
 size\_t low\_bound = (pos == 0) ? SIZE\_MAX : pos;  
 (\*n)++;  
  
 for (size\_t i = \*n; i != low\_bound; i--)  
 a[i] = a[i - 1];  
  
 a[pos] = value;  
 } else {  
 (\*n)++;  
 a[pos] = value;  
 }  
}  
  
  
void append\_(int a[], size\_t\* n, int value) {  
 a[\*n] = value;  
 (\*n)++;  
}  
  
  
void deleteByPosSaveOrder\_(int a[], size\_t\* n, size\_t pos) {  
 for (size\_t i = pos; i < \*n; i++)  
 a[i] = a[i + 1];  
 (\*n)--;  
}  
  
  
void deleteByPosUnsaveOrder\_(int a[], size\_t\* n, size\_t pos) {  
 a[pos] = a[\*n - 1];  
 (\*n)--;  
}  
  
int all\_(const int a[], size\_t n, int (\*predicate)(int)) {  
 for (size\_t i = 0; i < n; i++)  
 if (!predicate(a[i]))  
 return 0;  
 return 1;  
}  
  
int any\_(const int a[], size\_t n, int (\*predicate)(int)) {  
 for (size\_t i = 0; i < n; i++)  
 if (predicate(a[i]))  
 return 1;  
 return 0;  
}  
  
void forEach\_(const int source[], int dest[], size\_t n, int(\*predicate)(int)) {  
 for (size\_t i = 0; i < n; i++)  
 dest[i] = predicate(source[i]);  
}  
  
  
int countIf\_(const int a[], size\_t n, int(\*predicate)(int)) {  
 int res = 0;  
  
 for (size\_t i = 0; i < n; i++)  
 if (predicate(a[i]))  
 res++;  
  
 return res;  
}  
  
void deleteIf\_(int a[], size\_t\* n, int(\*deletePredicate)(int)) {  
 size\_t i\_read = 0;  
 while (i\_read < \*n && !deletePredicate(a[i\_read]))  
 i\_read++;  
  
 size\_t i\_write = i\_read;  
 while (i\_read < \*n) {  
 if (!deletePredicate(a[i\_read])) {  
 a[i\_write] = a[i\_read];  
 i\_write++;  
 }  
  
 i\_read++;  
 }  
  
 (\*n) = i\_write;  
}

Содержание bitset.h:

#ifndef ARRAY\_BITSET\_H  
#define ARRAY\_BITSET\_H  
  
#include <stdbool.h>  
#include <stdint.h>  
  
typedef struct bitset {  
 uint32\_t values;  
 uint32\_t max\_value;  
} bitset;  
  
// возвращает пустое множество с универсумом 0, 1, ..., max\_value  
bitset bitset\_create(unsigned max\_value);  
  
// возвращает множество, состоящее из элементов массива a размера size, и количеством элементов max\_value  
bitset bitset\_create\_from\_array(const unsigned int a[], size\_t size, unsigned max\_value);  
  
// возвращает true, если value является элементов set  
// иначе false  
bool bitset\_in(bitset set, unsigned value);  
  
// возвращает true, если set1 и set2 равны  
// иначе false  
bool bitset\_isEqual(bitset set1, bitset set2);  
  
// возвращает true, если subset является подмножеством set  
// иначе false  
bool bitset\_isSubset(bitset subset, bitset set);  
  
// вставка элемента value в множество set  
void bitset\_insert(bitset\* set, unsigned value);  
  
// удаление элемента value из множества set  
void bitset\_deleteElement(bitset\* set, unsigned value);  
  
// возвращает объединение множества set1 и set2  
bitset bitset\_union(bitset set1, bitset set2);  
  
// возвращает пересечение множеств set1 и set2  
bitset bitset\_intersection(bitset set1, bitset set2);  
  
// возвращает разность множеств set1 и set2  
bitset bitset\_difference(bitset set1, bitset set2);  
  
// возвращает симметричную разность множеств set1 и set2  
bitset bitset\_symmetricDifference(bitset set1, bitset set2);  
  
// возвращает дополнение множества set до универсума  
bitset bitset\_complement(bitset set);  
  
// выводит множество set  
void bitset\_print(bitset set);  
  
#endif //ARRAY\_BITSET\_H

Содержание bitset.c:

#include <stdio.h>  
#include <assert.h>  
#include "bitset.h"  
  
  
bitset bitset\_create(unsigned max\_value) {  
 assert(max\_value < 32);  
 return (bitset) {0, max\_value};  
}  
  
  
bitset bitset\_create\_from\_array(const unsigned int a[], size\_t size, unsigned max\_value) {  
 assert(size < 32);  
  
 bitset set = bitset\_create(max\_value);  
  
 for (size\_t i = 0; i < size; i++)  
 bitset\_insert(&set, \*(a + i));  
  
 return set;  
}  
  
  
bool bitset\_in(bitset set, unsigned value) {  
 return set.values & (1 << value);  
}  
  
bool bitset\_isEqual(bitset set1, bitset set2) {  
 return set1.values == set2.values;  
}  
  
bool bitset\_isSubset(bitset subset, bitset set) {  
 return (set.values & subset.values) == subset.values;  
}  
  
  
void bitset\_insert(bitset\* set, unsigned value) {  
 set -> values = (set -> values) | (1 << value);  
}  
  
  
void bitset\_deleteElement(bitset\* set, unsigned value) {  
 set -> values = (set -> values) & ~(1 << value);  
}  
  
  
bitset bitset\_union(bitset set1, bitset set2) {  
 assert(set1.max\_value == set2.max\_value);  
 return (bitset) {set1.values | set2.values, set1.max\_value};  
}  
  
  
bitset bitset\_intersection(bitset set1, bitset set2) {  
 assert(set1.max\_value == set2.max\_value);  
 return (bitset) {set1.values & set2.values};  
}  
  
  
bitset bitset\_difference(bitset set1, bitset set2) {  
 assert(set1.max\_value == set2.max\_value);  
 return (bitset) {set1.values & ~set2.values};  
}  
  
  
bitset bitset\_symmetricDifference(bitset set1, bitset set2) {  
 assert(set1.max\_value == set2.max\_value);  
 return (bitset) {set1.values ^ set2.values};  
}  
  
  
bitset bitset\_complement(bitset set) {  
 uint32\_t universum = (1 << (set.max\_value + 1)) - 1;  
 return (bitset) {set.values ^ universum, set.max\_value};  
}  
  
  
void bitset\_print(bitset set) {  
 printf ("{") ;  
 int is\_empty = 1;  
  
 for (int i = 0; i <= set.max\_value; i++) {  
 if (bitset\_in(set, i)) {  
 printf("%d, ", i);  
 is\_empty = 0;  
 }  
 }  
  
 if (is\_empty)  
 printf("}\n");  
 else  
 printf("\b\b}\n");  
}

Содержание ordered\_array\_set.h:

#ifndef ARRAY\_ORDERED\_ARRAY\_SET\_H  
#define ARRAY\_ORDERED\_ARRAY\_SET\_H  
  
  
#include <stdint.h>  
#include <assert.h>  
#include <memory.h>  
#include <stdio.h>  
#include <stdbool.h>  
#include "../../algorithms/array/array.h"  
  
  
typedef struct ordered\_array\_set {  
 int\* data;  
 size\_t size;  
 size\_t capacity;  
} ordered\_array\_set;  
  
  
// возвращает пусткое множество для capacity элементов  
ordered\_array\_set ordered\_array\_set\_create(size\_t capacity);  
  
// возвращает множество, состоящее из элементов массива a размера size  
ordered\_array\_set ordered\_array\_set\_create\_from\_array(const int a[], size\_t size);  
  
// возвращает позицию элемента в множестве,  
// если значение value имеется в множестве set, иначе n  
size\_t ordered\_array\_set\_in(ordered\_array\_set\* set, int value);  
  
// возвращает true, если subset является подмножеством set.  
// инчае false  
bool ordered\_array\_set\_isSubset(ordered\_array\_set subset, ordered\_array\_set set);  
  
// возвращает true, если элементы множеств set1 и set2 равны  
// иначе false  
bool ordered\_array\_set\_isEqual(ordered\_array\_set set1, ordered\_array\_set set2);  
  
// возбуждает исплючение, если в множестве по адресу set  
// нельзя вставить элемент  
void ordered\_array\_set\_isAbleAppend(ordered\_array\_set \*set);  
  
// добавляет элемент value в множество set  
void ordered\_array\_set\_insert(ordered\_array\_set\* set, int value);  
  
// удалить элемент value из множества set  
void ordered\_array\_set\_deleteElement(ordered\_array\_set\* set, int value);  
  
// возвращает объединение множеств set1 и set2  
ordered\_array\_set ordered\_array\_set\_union(ordered\_array\_set set1, ordered\_array\_set set2);  
  
// возвращает пересечение множеств set1 и set2  
ordered\_array\_set ordered\_array\_set\_intersection(ordered\_array\_set set1, ordered\_array\_set set2);  
  
// возвращает разность множеств set1 и set2  
ordered\_array\_set ordered\_array\_set\_difference(ordered\_array\_set set1, ordered\_array\_set set2);  
  
// возвращает дополнение множества set до универсума universumSet  
ordered\_array\_set ordered\_array\_set\_complement(ordered\_array\_set set, ordered\_array\_set universumSet);  
  
// возвращает симметричную разность множеств set1 и set2  
ordered\_array\_set ordered\_array\_set\_symmetricDifference(ordered\_array\_set set1, ordered\_array\_set set2);  
  
// вывод множества set  
void ordered\_array\_set\_print(ordered\_array\_set set);  
  
// освобождает память, занимаемую множеством set  
void ordered\_array\_set\_delete(ordered\_array\_set\* set);  
  
  
#endif //ARRAY\_ORDERED\_ARRAY\_SET\_H

Содержание ordered\_array\_set.c:

#include <stdio.h>  
#include <assert.h>  
#include <malloc.h>  
#include <stdlib.h>  
#include <memory.h>  
#include "../../algorithms/array/array.h"  
#include "../../data\_structures/ordered\_array\_set/ordered\_array\_set.h"  
  
  
static int compare\_ints(const void \*a, const void \*b) {  
 return \*(int \*) a - \*(int \*) b;  
}  
  
ordered\_array\_set ordered\_array\_set\_create(size\_t capacity) {  
 return (ordered\_array\_set) {malloc(sizeof(int) \* capacity), 0, capacity};  
}  
  
  
void ordered\_array\_set\_isAbleAppend(ordered\_array\_set \*set) {  
 assert(set->size < set->capacity);  
}  
  
  
size\_t ordered\_array\_set\_in(ordered\_array\_set \*set, int value) {  
 return binarySearch\_(set->data, set->size, value);  
}  
  
  
void ordered\_array\_set\_insert(ordered\_array\_set \*set, int value) {  
 size\_t index = ordered\_array\_set\_in(set, value);  
  
 if (index == set->size) {  
 ordered\_array\_set\_isAbleAppend(set);  
  
 size\_t i;  
 for (i = set->size; (i > 0 && set->data[i - 1] > value); i--)  
 set->data[i] = set->data[i - 1];  
  
 set->data[i] = value;  
 set->size++;  
 }  
}  
  
  
bool ordered\_array\_set\_isEqual(ordered\_array\_set set1, ordered\_array\_set set2) {  
 if (set1.size != set2.size)  
 return 0;  
  
 return memcmp(set1.data, set2.data, sizeof(int) \* set1.size) == 0;  
}  
  
  
void ordered\_array\_set\_shrinkToFit(ordered\_array\_set \*a) {  
 if (a->size != a->capacity) {  
 a->data = (int \*) realloc(a->data, sizeof(int) \* a->size);  
 a->capacity = a->size;  
 }  
}  
  
  
ordered\_array\_set ordered\_array\_set\_create\_from\_array(const int \*a, size\_t size) {  
 ordered\_array\_set set = ordered\_array\_set\_create(size);  
 for (size\_t i = 0; i < size; i++)  
 ordered\_array\_set\_insert(&set, \*(a + i));  
 ordered\_array\_set\_shrinkToFit(&set);  
  
 return set;  
}  
  
  
bool ordered\_array\_set\_isSubset(ordered\_array\_set subset, ordered\_array\_set set) {  
 for (size\_t i = 0; i < subset.size; i++) {  
 bool found = false;  
  
 for (size\_t j = 0; j < set.size; j++)  
 if (subset.data[i] == set.data[j]) {  
 found = true;  
 break;  
 }  
  
 if (!found)  
 return false;  
 }  
  
 return true;  
}  
  
  
void ordered\_array\_set\_deleteElement(ordered\_array\_set \*set, int value) {  
 size\_t index = ordered\_array\_set\_in(set, value);  
  
 if (index != set->size)  
 deleteByPosSaveOrder\_(set->data, &set->size, index);  
}  
  
  
ordered\_array\_set ordered\_array\_set\_union(ordered\_array\_set set1, ordered\_array\_set set2) {  
 size\_t new\_capacity = set1.size + set2.size;  
 ordered\_array\_set set = ordered\_array\_set\_create(new\_capacity);  
  
  
 size\_t i = 0;  
 size\_t j = 0;  
 while (i < set1.size && j < set2.size) {  
 if (j == set2.size || set1.data[i] < set2.data[j]) {  
 set.data[set.size] = set1.data[i];  
 set.size++;  
 i++;  
 } else if (i == set1.size || set1.data[i] > set2.data[j]) {  
 set.data[set.size] = set2.data[j];  
 set.size++;  
 j++;  
 } else {  
 set.data[set.size] = set1.data[i];  
 set.size++;  
 i++;  
 j++;  
 }  
 }  
  
 while (i < set1.size) {  
 set.data[set.size] = set1.data[i];  
 set.size++;  
 i++;  
 }  
  
 while (j < set2.size) {  
 set.data[set.size] = set2.data[j];  
 set.size++;  
 j++;  
 }  
  
 ordered\_array\_set\_shrinkToFit(&set);  
  
 return set;  
}  
  
  
ordered\_array\_set ordered\_array\_set\_intersection(ordered\_array\_set set1, ordered\_array\_set set2) {  
 size\_t new\_capacity = set1.size < set2.size ? set1.size : set2.size;  
 ordered\_array\_set set = ordered\_array\_set\_create(new\_capacity);  
  
 size\_t i = 0;  
 size\_t j = 0;  
  
 while (i != set1.size && j != set2.size) {  
 if (set1.data[i] < set2.data[j])  
 i++;  
 else if (set1.data[i] > set2.data[j])  
 j++;  
 else {  
 set.data[set.size] = set1.data[i];  
 set.size++;  
 i++;  
 j++;  
 }  
 }  
  
 ordered\_array\_set\_shrinkToFit(&set);  
  
 return set;  
}  
  
  
ordered\_array\_set ordered\_array\_set\_difference(ordered\_array\_set set1, ordered\_array\_set set2) {  
 size\_t new\_capacity = set1.size;  
 ordered\_array\_set set = ordered\_array\_set\_create(new\_capacity);  
  
 size\_t i = 0;  
 size\_t j = 0;  
  
 while (i < set1.size) {  
 if (j == set2.size || set1.data[i] < set2.data[j]) {  
 set.data[set.size] = set1.data[i];  
 set.size++;  
 i++;  
 } else if (set1.data[i] > set2.data[j])  
 j++;  
 else  
 i++;  
 }  
  
 ordered\_array\_set\_shrinkToFit(&set);  
  
 return set;  
}  
  
  
ordered\_array\_set ordered\_array\_set\_complement(ordered\_array\_set set, ordered\_array\_set universumSet) {  
 size\_t new\_capacity = universumSet.size;  
 ordered\_array\_set new\_set = ordered\_array\_set\_create(new\_capacity);  
  
 size\_t i = 0, j = 0;  
 while (i < universumSet.size) {  
 if (j < set.size && universumSet.data[i] == set.data[j]) {  
 i++;  
 j++;  
 } else {  
 new\_set.data[new\_set.size] = universumSet.data[i];  
 new\_set.size++;  
 i++;  
 }  
 }  
  
 ordered\_array\_set\_shrinkToFit(&new\_set);  
  
 assert(ordered\_array\_set\_isSubset(new\_set, universumSet));  
  
 return new\_set;  
}  
  
  
ordered\_array\_set ordered\_array\_set\_symmetricDifference(ordered\_array\_set set1, ordered\_array\_set set2) {  
 ordered\_array\_set universum = ordered\_array\_set\_union(set1, set2);  
 ordered\_array\_set intersection = ordered\_array\_set\_intersection(set1, set2);  
  
 ordered\_array\_set symmetric = ordered\_array\_set\_complement(intersection, universum);  
  
 ordered\_array\_set\_delete(&intersection);  
 ordered\_array\_set\_delete(&universum);  
  
 return symmetric;  
}  
  
  
void ordered\_array\_set\_print(ordered\_array\_set set) {  
 printf("{");  
 int is\_empty = 1;  
  
 for (size\_t i = 0; i < set.size; i++) {  
 printf("%d, ", \*(set.data + i));  
 is\_empty = 0;  
 }  
 if (is\_empty)  
 printf("}\n");  
 else  
 printf("\b\b}\n");  
}  
  
  
void ordered\_array\_set\_delete(ordered\_array\_set\* set) {  
 free(set -> data);  
 set -> data = NULL;  
  
 set -> size = 0;  
 set -> capacity = 0;  
}

Содержание unordered\_array\_set.h:

#ifndef ARRAY\_UNORDERED\_ARRAY\_SET\_H  
#define ARRAY\_UNORDERED\_ARRAY\_SET\_H  
  
  
#include <stdint.h>  
#include <assert.h>  
#include <memory.h>  
#include <stdio.h>  
#include <stdbool.h>  
#include "../../algorithms/array/array.h"  
  
  
typedef struct unordered\_array\_set {  
 int\* data;  
 size\_t size;  
 size\_t capacity;  
} unordered\_array\_set;  
  
  
// возвращает пусткое множество для capacity элементов  
unordered\_array\_set unordered\_array\_set\_create(size\_t capacity);  
  
// возвращает множество, состоящее из элементов массива a размера size  
unordered\_array\_set unordered\_array\_set\_create\_from\_array(const int\* a, size\_t size);  
  
// возвращает позицию элемента в множестве,  
// если значение value имеется в множестве set, иначе n  
size\_t unordered\_array\_set\_in(unordered\_array\_set\* set, int value);  
  
// возвращает true, если subset является подмножеством set.  
// инчае false  
bool unordered\_array\_set\_isSubset(unordered\_array\_set subset, unordered\_array\_set set);  
  
// возвращает true, если элементы множеств set1 и set2 равны  
// иначе false  
bool unordered\_array\_set\_isEqual(unordered\_array\_set set1, unordered\_array\_set set2);  
  
// возбуждает исплючение, если в множестве по адресу set  
// нельзя вставить элемент  
void unordered\_array\_set\_isAbleAppend(unordered\_array\_set \*set);  
  
// добавляет элемент value в множество set  
void unordered\_array\_set\_insert(unordered\_array\_set\* set, int value);  
  
// удалить элемент value из множества set  
void unordered\_array\_set\_deleteElement(unordered\_array\_set\* set, int value);  
  
// возвращает объединение множеств set1 и set2  
unordered\_array\_set unordered\_array\_set\_union(unordered\_array\_set set1, unordered\_array\_set set2);  
  
// возвращает пересечение множеств set1 и set2  
unordered\_array\_set unordered\_array\_set\_intersection(unordered\_array\_set set1, unordered\_array\_set set2);  
  
// возвращает разность множеств set1 и set2  
unordered\_array\_set unordered\_array\_set\_difference(unordered\_array\_set set1, unordered\_array\_set set2);  
  
// возвращает дополнение множества set до универсума universumSet  
unordered\_array\_set unordered\_array\_set\_complement(unordered\_array\_set set, unordered\_array\_set universumSet);  
  
// возвращает симметричную разность множеств set1 и set2  
unordered\_array\_set unordered\_array\_set\_symmetricDifference(unordered\_array\_set set1, unordered\_array\_set set2);  
  
// вывод множества set  
void unordered\_array\_set\_print(unordered\_array\_set set);  
  
// освобождает память, занимаемую множеством set  
void unordered\_array\_set\_delete(unordered\_array\_set\* set);  
  
  
#endif //ARRAY\_UNORDERED\_ARRAY\_SET\_H

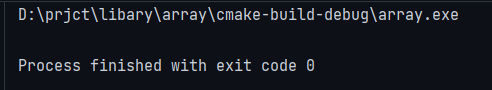
Содержание unordered\_array\_set.c:

#include <stdio.h>  
#include <assert.h>  
#include <malloc.h>  
#include <stdlib.h>  
#include <memory.h>  
#include "../../algorithms/array/array.h"  
#include "../../data\_structures/unordered\_array\_set/unordered\_array\_set.h"  
  
  
static int compare\_ints(const void\* a, const void\* b) {  
 return \*(int\* ) a - \*(int\* ) b;  
}  
  
  
unordered\_array\_set unordered\_array\_set\_create(size\_t capacity) {  
 return (unordered\_array\_set) {malloc(sizeof(int) \* capacity), 0, capacity};  
}  
  
  
void unordered\_array\_set\_isAbleAppend(unordered\_array\_set \*set) {  
 assert(set -> size < set -> capacity);  
}  
  
  
size\_t unordered\_array\_set\_in(unordered\_array\_set\* set, int value) {  
 return linearSearch\_(set -> data, set -> size, value);  
}  
  
  
void unordered\_array\_set\_insert(unordered\_array\_set\* set, int value) {  
 if (unordered\_array\_set\_in(set, value) == set -> size) {  
 unordered\_array\_set\_isAbleAppend(set);  
 append\_(set -> data, &set -> size, value);  
 }  
}  
  
  
bool unordered\_array\_set\_isEqual(unordered\_array\_set set1, unordered\_array\_set set2) {  
 if (set1.size != set2.size)  
 return 0;  
  
 qsort(set1.data, set1.size, sizeof(int), compare\_ints);  
 qsort(set2.data, set2.size, sizeof(int), compare\_ints);  
  
 return memcmp(set1.data, set2.data, sizeof(int) \* set1.size) == 0;  
}  
  
  
static void unordered\_array\_set\_shrinkToFit(unordered\_array\_set\* a) {  
 if (a -> size != a -> capacity) {  
 a -> data = (int\*)realloc(a -> data, sizeof(int) \* a -> size);  
 a -> capacity = a -> size;  
 }  
}  
  
  
unordered\_array\_set unordered\_array\_set\_create\_from\_array(const int\* a, size\_t size) {  
 unordered\_array\_set set = unordered\_array\_set\_create(size);  
 for (size\_t i = 0; i < size; i++)  
 unordered\_array\_set\_insert(&set, a[i]);  
 unordered\_array\_set\_shrinkToFit(&set);  
  
 return set;  
}  
  
  
bool unordered\_array\_set\_isSubset(unordered\_array\_set subset, unordered\_array\_set set) {  
 for (size\_t i = 0; i < subset.size; i++) {  
 bool found = false;  
  
 for (size\_t j = 0; j < set.size; j++)  
 if (subset.data[i] == set.data[j]) {  
 found = true;  
 break;  
 }  
  
 if (!found)  
 return false;  
 }  
  
 return true;  
}  
  
  
void unordered\_array\_set\_deleteElement(unordered\_array\_set\* set, int value) {  
 size\_t index\_value = unordered\_array\_set\_in(set, value);  
  
 if (index\_value < set -> size) {  
 set->data[index\_value] = set->data[set->size - 1];  
 (set->size)--;  
 }  
}  
  
  
unordered\_array\_set unordered\_array\_set\_union(unordered\_array\_set set1, unordered\_array\_set set2) {  
 size\_t new\_capacity = set1.size + set2.size;  
 unordered\_array\_set set = unordered\_array\_set\_create(new\_capacity);  
  
 for (size\_t i = 0; i < set1.size; i++) {  
 set.data[i] = set1.data[i];  
 set.size++;  
 }  
  
 for (size\_t i = 0; i < set2.size; i++)  
 unordered\_array\_set\_insert(&set, set2.data[i]);  
  
 unordered\_array\_set\_shrinkToFit(&set);  
  
 return set;  
}  
  
  
unordered\_array\_set unordered\_array\_set\_intersection(unordered\_array\_set set1, unordered\_array\_set set2) {  
 size\_t new\_capacity = set1.size < set2.size ? set1.size : set2.size;  
 unordered\_array\_set set = unordered\_array\_set\_create(new\_capacity);  
  
 for (size\_t i = 0; i < set1.size; i++)  
 if (unordered\_array\_set\_in(&set2, set1.data[i]) != set2.size)  
 unordered\_array\_set\_insert(&set, set1.data[i]);  
  
 return set;  
}  
  
  
unordered\_array\_set unordered\_array\_set\_difference(unordered\_array\_set set1, unordered\_array\_set set2) {  
 size\_t new\_capacity = set1.size;  
 unordered\_array\_set set = unordered\_array\_set\_create(new\_capacity);  
  
 for (size\_t i = 0; i < set1.size; i++)  
 if (unordered\_array\_set\_in(&set2, set1.data[i]) == set2.size)  
 unordered\_array\_set\_insert(&set, set1.data[i]);  
  
 return set;  
}  
  
  
unordered\_array\_set unordered\_array\_set\_complement(unordered\_array\_set set, unordered\_array\_set universumSet) {  
 size\_t new\_capacity = universumSet.size;  
 unordered\_array\_set new\_set = unordered\_array\_set\_create(new\_capacity);  
  
 for (size\_t i = 0; i < universumSet.size; i++)  
 if (unordered\_array\_set\_in(&set, universumSet.data[i]) == set.size)  
 unordered\_array\_set\_insert(&new\_set, universumSet.data[i]);  
  
 assert(unordered\_array\_set\_isSubset(new\_set, universumSet));  
  
 return new\_set;  
}  
  
  
unordered\_array\_set unordered\_array\_set\_symmetricDifference(unordered\_array\_set set1, unordered\_array\_set set2) {  
 unordered\_array\_set universum = unordered\_array\_set\_union(set1, set2);  
 unordered\_array\_set intersection = unordered\_array\_set\_intersection(set1, set2);  
  
 unordered\_array\_set symmetric = unordered\_array\_set\_complement(intersection, universum);  
  
 unordered\_array\_set\_delete(&intersection);  
 unordered\_array\_set\_delete(&universum);  
  
 return symmetric;  
}

Тест работоспособности библиотек:

#include <stdio.h>  
#include "libs/algorithms/array/array.h"  
#include "libs/data\_structures/bitset/bitset.h"  
#include "libs/data\_structures/unordered\_array\_set/unordered\_array\_set.h"  
#include "libs/data\_structures/ordered\_array\_set/ordered\_array\_set.h"  
  
  
// тесты с битовыми множествами  
typedef unsigned int uint;  
  
// тест на наличие элемента в множестве  
void test\_bitset\_in\_1() {  
 bitset set = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 uint value = 3;  
  
 bool index = bitset\_in(set, value);  
  
 assert(index == 1);  
}  
  
  
void test\_bitset\_in\_2() {  
 bitset set = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 uint value = 4;  
  
 bool index = bitset\_in(set, value);  
  
 assert(index == 0);  
}  
  
  
void test\_bitset\_in() {  
 test\_bitset\_in\_1();  
 test\_bitset\_in\_2();  
}  
  
  
// тест на является ли множество подмножеством другого множества  
void test\_bitset\_isSubset\_1() {  
 bitset subset = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set = bitset\_create\_from\_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);  
  
 assert(bitset\_isSubset(subset, set));  
}  
  
  
void test\_bitset\_isSubset\_2() {  
 bitset subset = bitset\_create\_from\_array((uint[]){1, 2, 10}, 3, 10);  
 bitset set = bitset\_create\_from\_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);  
  
 assert(!bitset\_isSubset(subset, set));  
}  
  
  
void test\_bitset\_isSubset\_3() {  
 bitset subset = bitset\_create\_from\_array((uint[]){}, 0, 10);  
 bitset set = bitset\_create\_from\_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);  
  
 assert(bitset\_isSubset(subset, set));  
}  
  
  
void test\_bitset\_isSubset\_4() {  
 bitset subset = bitset\_create\_from\_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);  
 bitset set = bitset\_create\_from\_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);  
  
 assert(bitset\_isSubset(subset, set));  
}  
  
  
void test\_bitset\_isSubset() {  
 test\_bitset\_isSubset\_1();  
 test\_bitset\_isSubset\_2();  
 test\_bitset\_isSubset\_3();  
 test\_bitset\_isSubset\_4();  
}  
  
  
// тест на вставку элемента в множество  
void test\_bitset\_insert\_1() {  
 bitset set = bitset\_create\_from\_array((uint[]){7, 8}, 2, 10);  
 uint value = 4;  
  
 bitset\_insert(&set, value);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){4, 7, 8}, 3, 3);  
  
 assert(bitset\_isEqual(set, check\_set));  
}  
  
  
void test\_bitset\_insert\_2() {  
 bitset set = bitset\_create\_from\_array((uint[]){7, 8}, 2, 10);  
 uint value = 7;  
  
 bitset\_insert(&set, value);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){7, 8}, 2, 2);  
  
 assert(bitset\_isEqual(set, check\_set));  
}  
  
  
void test\_bitset\_insert() {  
 test\_bitset\_insert\_1();  
 test\_bitset\_insert\_2();  
}  
  
  
// тест на удаление элемента  
void test\_bitset\_deleteElement\_1() {  
 bitset set = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 uint value = 3;  
  
 bitset\_deleteElement(&set, value);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 2}, 2, 10);  
  
 assert(bitset\_isEqual(set, check\_set));  
}  
  
  
void test\_bitset\_deleteElement\_2() {  
 bitset set = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 uint value = 5;  
  
 bitset\_deleteElement(&set, value);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 2,3}, 3, 2);  
  
 assert(bitset\_isEqual(set, check\_set));  
}  
  
void test\_bitset\_deleteElement() {  
 test\_bitset\_deleteElement\_1();  
 test\_bitset\_deleteElement\_2();  
}  
  
  
// тест на объединение множеств  
void test\_bitset\_union\_1() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){3, 4, 5}, 3, 10);  
  
 bitset res\_set = bitset\_union(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 2, 3, 4, 5}, 5, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_union\_2() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){4, 5, 6}, 3, 10);  
  
 bitset res\_set = bitset\_union(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_union\_3() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
  
 bitset res\_set = bitset\_union(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_union\_4() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){}, 0, 10);  
  
 bitset res\_set = bitset\_union(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_union() {  
 test\_bitset\_union\_1();  
 test\_bitset\_union\_2();  
 test\_bitset\_union\_3();  
 test\_bitset\_union\_4();  
}  
  
  
// тест на пересечение множеств  
void test\_bitset\_intersection\_1() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){2, 3, 4}, 3, 10);  
  
 bitset res\_set = bitset\_intersection(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){2, 3}, 2, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_intersection\_2() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){4, 5, 6}, 3, 10);  
  
 bitset res\_set = bitset\_intersection(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){}, 0, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_intersection\_3() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
  
 bitset res\_set = bitset\_intersection(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_intersection() {  
 test\_bitset\_intersection\_1();  
 test\_bitset\_intersection\_2();  
 test\_bitset\_intersection\_3();  
}  
  
  
// тест на разность двух множеств  
void test\_bitset\_difference\_1() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 3, 7}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){3}, 1, 10);  
  
 bitset res\_set = bitset\_difference(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 7}, 2, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_difference\_2() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 3, 7}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){}, 0, 10);  
  
 bitset res\_set = bitset\_difference(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 3, 7}, 3, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_difference\_3() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 3, 7}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){1, 3, 7}, 3, 10);  
  
 bitset res\_set = bitset\_difference(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){}, 0, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_difference() {  
 test\_bitset\_difference\_1();  
 test\_bitset\_difference\_2();  
 test\_bitset\_difference\_3();  
}  
  
  
// тест на симметричную разность двух множеств  
void test\_symmetricDifference\_1() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){2, 3, 4}, 3, 10);  
  
 bitset res\_set = bitset\_symmetricDifference(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 4}, 2, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_symmetricDifference\_2() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){4, 5, 6}, 3, 10);  
  
 bitset res\_set = bitset\_symmetricDifference(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_symmetricDifference\_3() {  
 bitset set1 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
 bitset set2 = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
  
 bitset res\_set = bitset\_symmetricDifference(set1, set2);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){}, 0, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_symmetricDifference() {  
 test\_symmetricDifference\_1();  
 test\_symmetricDifference\_2();  
 test\_symmetricDifference\_3();  
}  
  
  
// тест на дополнение множества  
void test\_bitset\_complement\_1() {  
 bitset set = bitset\_create\_from\_array((uint[]){1, 2, 3}, 3, 10);  
  
 bitset res\_set = bitset\_complement(set);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){0, 4, 5, 6, 7, 8, 9, 10}, 8, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_complement\_2() {  
 bitset set = bitset\_create\_from\_array((uint[]){}, 0, 10);  
  
 bitset res\_set = bitset\_complement(set);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){0, 1, 2, 3,4, 5, 6, 7, 8, 9, 10}, 11, 12);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
void test\_bitset\_complement\_3() {  
 bitset set = bitset\_create\_from\_array((uint[]){0, 1, 2, 3,4, 5, 6, 7, 8, 9, 10}, 11, 10);  
  
 bitset res\_set = bitset\_complement(set);  
  
 bitset check\_set = bitset\_create\_from\_array((uint[]){}, 0, 10);  
  
 assert(bitset\_isEqual(res\_set, check\_set));  
}  
  
  
  
void test\_bitset\_complement() {  
 test\_bitset\_complement\_1();  
 test\_bitset\_complement\_2();  
 test\_bitset\_complement\_3();  
}  
  
  
void test\_bitset() {  
 test\_bitset\_in();  
 test\_bitset\_isSubset();  
 test\_bitset\_insert();  
 test\_bitset\_deleteElement();  
 test\_bitset\_union();  
 test\_bitset\_intersection();  
 test\_bitset\_difference();  
 test\_symmetricDifference();  
 test\_bitset\_complement();  
}  
  
  
// тест для упорядоченных множеств  
  
  
  
// тест на наличие элемента в множестве  
void test\_ordered\_array\_set\_in\_1() {  
 ordered\_array\_set set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
 int value = 3;  
  
 size\_t index = ordered\_array\_set\_in(&set, value);  
  
 assert(index == 2);  
  
 ordered\_array\_set\_delete(&set);  
}  
  
  
void test\_ordered\_array\_set\_in\_2() {  
 ordered\_array\_set set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
 int value = 5;  
  
 size\_t index = ordered\_array\_set\_in(&set, value);  
  
 assert(index == 3);  
  
 ordered\_array\_set\_delete(&set);  
}  
  
  
void test\_ordered\_array\_set\_in() {  
 test\_ordered\_array\_set\_in\_1();  
 test\_ordered\_array\_set\_in\_2();  
}  
  
  
// тест является ли множество подномножеством другого множества  
void test\_ordered\_array\_set\_isSubset\_1() {  
 ordered\_array\_set subset = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
 ordered\_array\_set set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(ordered\_array\_set\_isSubset(subset, set));  
  
 ordered\_array\_set\_delete(&subset);  
 ordered\_array\_set\_delete(&set);  
}  
  
  
void test\_ordered\_array\_set\_isSubset\_2() {  
 ordered\_array\_set subset = ordered\_array\_set\_create\_from\_array((int[]){4, 5, 3}, 3);  
 ordered\_array\_set set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(ordered\_array\_set\_isSubset(subset, set));  
  
 ordered\_array\_set\_delete(&subset);  
 ordered\_array\_set\_delete(&set);  
}  
  
  
void test\_ordered\_array\_set\_isSubset\_3() {  
 ordered\_array\_set subset = ordered\_array\_set\_create\_from\_array((int[]){5, 3, 2, 4, 1, 6}, 6);  
 ordered\_array\_set set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(ordered\_array\_set\_isSubset(subset, set));  
  
 ordered\_array\_set\_delete(&subset);  
 ordered\_array\_set\_delete(&set);  
}  
  
  
void test\_ordered\_array\_set\_isSubset\_4() {  
 ordered\_array\_set subset = ordered\_array\_set\_create\_from\_array((int[]){10}, 1);  
 ordered\_array\_set set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(!ordered\_array\_set\_isSubset(subset, set));  
  
 ordered\_array\_set\_delete(&subset);  
 ordered\_array\_set\_delete(&set);  
}  
  
  
void test\_ordered\_array\_set\_isSubset() {  
 test\_ordered\_array\_set\_isSubset\_1();  
 test\_ordered\_array\_set\_isSubset\_2();  
 test\_ordered\_array\_set\_isSubset\_3();  
 test\_ordered\_array\_set\_isSubset\_4();  
}  
  
  
// тест на вставку элемента в множество  
void test\_ordered\_array\_set\_insert\_1() {  
 ordered\_array\_set set = ordered\_array\_set\_create(10);  
 int value1 = 2;  
 int value2 = 1;  
 int value3 = 3;  
  
 ordered\_array\_set\_insert(&set, value1);  
 ordered\_array\_set\_insert(&set, value2);  
 ordered\_array\_set\_insert(&set, value3);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
  
 assert(ordered\_array\_set\_isEqual(set, check\_set));  
  
 ordered\_array\_set\_delete(&set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_insert\_2() {  
 ordered\_array\_set set = ordered\_array\_set\_create(10);  
 int value1 = 7;  
 int value2 = 11;  
 int value3 = 2;  
 int value4 = 2;  
  
 ordered\_array\_set\_insert(&set, value1);  
 ordered\_array\_set\_insert(&set, value2);  
 ordered\_array\_set\_insert(&set, value3);  
 ordered\_array\_set\_insert(&set, value4);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){2, 7, 11}, 3);  
  
 assert(ordered\_array\_set\_isEqual(set, check\_set));  
  
 ordered\_array\_set\_delete(&set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_insert() {  
 test\_ordered\_array\_set\_insert\_1();  
 test\_ordered\_array\_set\_insert\_2();  
}  
  
  
// тест на удаление элемента из множества  
void test\_ordered\_array\_set\_deleteElement\_1() {  
 ordered\_array\_set set = ordered\_array\_set\_create\_from\_array((int[]){3, 6, 5, 2}, 4);  
 int value = 2;  
  
 ordered\_array\_set\_deleteElement(&set, value);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){3, 6, 5}, 3);  
  
 assert(ordered\_array\_set\_isEqual(set, check\_set));  
  
 ordered\_array\_set\_delete(&set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_deleteElement\_2() {  
 ordered\_array\_set set = ordered\_array\_set\_create\_from\_array((int[]){3, 6, 5, 2}, 4);  
 int value = 2;  
  
 ordered\_array\_set\_deleteElement(&set, value);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){3, 6, 5}, 3);  
  
 assert(ordered\_array\_set\_isEqual(set, check\_set));  
  
 ordered\_array\_set\_delete(&set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_deleteElement() {  
 test\_ordered\_array\_set\_deleteElement\_1();  
 test\_ordered\_array\_set\_deleteElement\_2();  
}  
  
  
// тест на объединение множеств  
void test\_ordered\_array\_set\_union\_1() {  
 ordered\_array\_set set1 = ordered\_array\_set\_create\_from\_array((int[]){3, 4, 1}, 3);  
 ordered\_array\_set set2 = ordered\_array\_set\_create\_from\_array((int[]){2, 4, 5}, 3);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_union(set1, set2);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5}, 5);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&set1);  
 ordered\_array\_set\_delete(&set2);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_union\_2() {  
 ordered\_array\_set set1 = ordered\_array\_set\_create\_from\_array((int[]){3, 4, 1}, 3);  
 ordered\_array\_set set2 = ordered\_array\_set\_create\_from\_array((int[]){3, 4, 1}, 3);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_union(set1, set2);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){1, 3, 4}, 3);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&set1);  
 ordered\_array\_set\_delete(&set2);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_union\_3() {  
 ordered\_array\_set set1 = ordered\_array\_set\_create\_from\_array((int[]){13, 7, 8}, 3);  
 ordered\_array\_set set2 = ordered\_array\_set\_create\_from\_array((int[]){}, 0);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_union(set1, set2);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){7, 13, 8}, 3);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&set1);  
 ordered\_array\_set\_delete(&set2);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_union() {  
 test\_ordered\_array\_set\_union\_1();  
 test\_ordered\_array\_set\_union\_2();  
 test\_ordered\_array\_set\_union\_3();  
}  
  
  
// тест на пересечение двух множеств  
void test\_ordered\_array\_set\_intersection\_1() {  
 ordered\_array\_set set1 = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
 ordered\_array\_set set2 = ordered\_array\_set\_create\_from\_array((int[]){2, 3, 4}, 3);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_intersection(set1, set2);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){2, 3}, 2);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&set1);  
 ordered\_array\_set\_delete(&set2);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_intersection\_2() {  
 ordered\_array\_set set1 = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
 ordered\_array\_set set2 = ordered\_array\_set\_create\_from\_array((int[]){4, 5, 6}, 3);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_intersection(set1, set2);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){}, 0);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&set1);  
 ordered\_array\_set\_delete(&set2);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_intersection() {  
 test\_ordered\_array\_set\_intersection\_1();  
 test\_ordered\_array\_set\_intersection\_2();  
}  
  
  
// тест на разность множеств  
void test\_ordered\_array\_set\_difference\_1() {  
 ordered\_array\_set set1 = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
 ordered\_array\_set set2 = ordered\_array\_set\_create\_from\_array((int[]){2, 3, 6}, 3);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_difference(set1, set2);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){1, 4, 5}, 3);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&set1);  
 ordered\_array\_set\_delete(&set2);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_difference\_2() {  
 ordered\_array\_set set1 = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
 ordered\_array\_set set2 = ordered\_array\_set\_create\_from\_array((int[]){7, 8, 9}, 3);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_difference(set1, set2);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&set1);  
 ordered\_array\_set\_delete(&set2);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_difference() {  
 test\_ordered\_array\_set\_difference\_1();  
 test\_ordered\_array\_set\_difference\_2();  
}  
  
  
// тест на симметричную разность  
void test\_ordered\_array\_set\_symmetricDifference\_1() {  
 ordered\_array\_set set1 = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
 ordered\_array\_set set2 = ordered\_array\_set\_create\_from\_array((int[]){2, 3, 6, 7, 10, 12}, 6);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_symmetricDifference(set1, set2);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){1, 4, 5,7, 10, 12}, 6);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&set1);  
 ordered\_array\_set\_delete(&set2);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_symmetricDifference\_2() {  
 ordered\_array\_set set1 = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
 ordered\_array\_set set2 = ordered\_array\_set\_create\_from\_array((int[]){4, 5, 6}, 3);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_symmetricDifference(set1, set2);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3,4, 5, 6}, 6);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&set1);  
 ordered\_array\_set\_delete(&set2);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_symmetricDifference() {  
 test\_ordered\_array\_set\_symmetricDifference\_1();  
 test\_ordered\_array\_set\_symmetricDifference\_2();  
}  
  
  
// тест на дополнение множества  
void test\_ordered\_array\_set\_complement\_1() {  
 ordered\_array\_set subset = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 4}, 3);  
 ordered\_array\_set universum = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_complement(subset, universum);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){3, 5, 6}, 3);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&subset);  
 ordered\_array\_set\_delete(&universum);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_complement\_2() {  
 ordered\_array\_set subset = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
 ordered\_array\_set universum = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_complement(subset, universum);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){}, 0);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&subset);  
 ordered\_array\_set\_delete(&universum);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_complement\_3() {  
 ordered\_array\_set subset = ordered\_array\_set\_create\_from\_array((int[]){}, 0);  
 ordered\_array\_set universum = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 ordered\_array\_set res\_set = ordered\_array\_set\_complement(subset, universum);  
  
 ordered\_array\_set check\_set = ordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(ordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 ordered\_array\_set\_delete(&subset);  
 ordered\_array\_set\_delete(&universum);  
 ordered\_array\_set\_delete(&res\_set);  
 ordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_ordered\_array\_set\_complement() {  
 test\_ordered\_array\_set\_complement\_1();  
 test\_ordered\_array\_set\_complement\_2();  
 test\_ordered\_array\_set\_complement\_3();  
}  
  
  
void test\_ordered\_array\_set() {  
 test\_ordered\_array\_set\_in();  
 test\_ordered\_array\_set\_isSubset();  
 test\_ordered\_array\_set\_insert();  
 test\_ordered\_array\_set\_deleteElement();  
 test\_ordered\_array\_set\_union();  
 test\_ordered\_array\_set\_intersection();  
 test\_ordered\_array\_set\_difference();  
 test\_ordered\_array\_set\_symmetricDifference();  
 test\_ordered\_array\_set\_complement();  
}  
  
  
// тесты для неупорядоченного множества  
  
  
  
// тест на наличие элемента в множестве  
void test\_unordered\_array\_set\_in\_1() {  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
 int value = 2;  
  
 size\_t index = unordered\_array\_set\_in(&set, value);  
  
 assert(index == 1);  
  
 unordered\_array\_set\_delete(&set);  
}  
  
  
void test\_unordered\_array\_set\_in\_2() {  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){10, 1, 4}, 3);  
 int value = 5;  
  
 size\_t index = unordered\_array\_set\_in(&set, value);  
  
 assert(index == 3);  
  
 unordered\_array\_set\_delete(&set);  
}  
  
  
void test\_unordered\_array\_set\_in() {  
 test\_unordered\_array\_set\_in\_1();  
 test\_unordered\_array\_set\_in\_2();  
}  
  
  
// тест на является ли одно множество подмножеством другого  
void test\_unordered\_array\_set\_isSubset\_1() {  
 unordered\_array\_set subset = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(unordered\_array\_set\_isSubset(subset, set));  
  
 unordered\_array\_set\_delete(&subset);  
 unordered\_array\_set\_delete(&set);  
}  
  
  
void test\_unordered\_array\_set\_isSubset\_2() {  
 unordered\_array\_set subset = unordered\_array\_set\_create\_from\_array((int[]){4, 5, 3}, 3);  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(unordered\_array\_set\_isSubset(subset, set));  
  
 unordered\_array\_set\_delete(&subset);  
 unordered\_array\_set\_delete(&set);  
}  
  
  
void test\_unordered\_array\_set\_isSubset\_3() {  
 unordered\_array\_set subset = unordered\_array\_set\_create\_from\_array((int[]){5, 3, 2, 4, 1, 6}, 6);  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(unordered\_array\_set\_isSubset(subset, set));  
  
 unordered\_array\_set\_delete(&subset);  
 unordered\_array\_set\_delete(&set);  
}  
  
  
void test\_unordered\_array\_set\_isSubset\_4() {  
 unordered\_array\_set subset = unordered\_array\_set\_create\_from\_array((int[]){10}, 1);  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(!unordered\_array\_set\_isSubset(subset, set));  
  
 unordered\_array\_set\_delete(&subset);  
 unordered\_array\_set\_delete(&set);  
}  
  
  
void test\_unordered\_array\_set\_isSubset() {  
 test\_unordered\_array\_set\_isSubset\_1();  
 test\_unordered\_array\_set\_isSubset\_2();  
 test\_unordered\_array\_set\_isSubset\_3();  
 test\_unordered\_array\_set\_isSubset\_4();  
}  
  
  
// тест на вставку элемента  
void test\_unordered\_array\_set\_insert\_1() {  
 unordered\_array\_set set = unordered\_array\_set\_create(10);  
 int value1 = 1;  
 int value2 = 7;  
 int value3 = 5;  
  
 unordered\_array\_set\_insert(&set, value1);  
 unordered\_array\_set\_insert(&set, value2);  
 unordered\_array\_set\_insert(&set, value3);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){1, 7, 5}, 3);  
  
 assert(unordered\_array\_set\_isEqual(set, check\_set));  
  
 unordered\_array\_set\_delete(&set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_insert\_2() {  
 unordered\_array\_set set = unordered\_array\_set\_create(10);  
 int value1 = 3;  
 int value2 = 7;  
 int value3 = 7;  
 int value4 = 8;  
  
 unordered\_array\_set\_insert(&set, value1);  
 unordered\_array\_set\_insert(&set, value2);  
 unordered\_array\_set\_insert(&set, value3);  
 unordered\_array\_set\_insert(&set, value4);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){3, 7, 8}, 3);  
  
 assert(unordered\_array\_set\_isEqual(set, check\_set));  
  
 unordered\_array\_set\_delete(&set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_insert() {  
 test\_unordered\_array\_set\_insert\_1();  
 test\_unordered\_array\_set\_insert\_2();  
}  
  
  
// тест на удаление элемента из множества  
void test\_unordered\_array\_set\_deleteElement\_1() {  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
 int delete\_value = 2;  
  
 unordered\_array\_set\_deleteElement(&set, delete\_value);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){1, 3}, 2);  
  
 assert(unordered\_array\_set\_isEqual(set, check\_set));  
  
 unordered\_array\_set\_delete(&set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_deleteElement\_2() {  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){4, 12, 3}, 3);  
 int delete\_value1 = 4;  
 int delete\_value2 = 12;  
 int delete\_value3 = 3;  
  
 unordered\_array\_set\_deleteElement(&set, delete\_value1);  
 unordered\_array\_set\_deleteElement(&set, delete\_value2);  
 unordered\_array\_set\_deleteElement(&set, delete\_value3);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){}, 0);  
  
 assert(unordered\_array\_set\_isEqual(set, check\_set));  
  
 unordered\_array\_set\_delete(&set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_deleteElement() {  
 test\_unordered\_array\_set\_deleteElement\_1();  
 test\_unordered\_array\_set\_deleteElement\_2();  
}  
  
  
// тест на объединение неупорядоченных множеств  
void test\_unordered\_array\_set\_union\_1() {  
 unordered\_array\_set set1 = unordered\_array\_set\_create\_from\_array((int[]){1, 2}, 2);  
 unordered\_array\_set set2 = unordered\_array\_set\_create\_from\_array((int[]){1, 3}, 2);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_union(set1, set2);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
  
 assert(unordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 unordered\_array\_set\_delete(&set1);  
 unordered\_array\_set\_delete(&set2);  
 unordered\_array\_set\_delete(&res\_set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_union\_2() {  
 unordered\_array\_set set1 = unordered\_array\_set\_create\_from\_array((int[]){5, 7, 8}, 3);  
 unordered\_array\_set set2 = unordered\_array\_set\_create\_from\_array((int[]){}, 0);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_union(set1, set2);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){5, 7, 8}, 3);  
  
 assert(unordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 unordered\_array\_set\_delete(&set1);  
 unordered\_array\_set\_delete(&set2);  
 unordered\_array\_set\_delete(&res\_set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_union() {  
 test\_unordered\_array\_set\_union\_1();  
 test\_unordered\_array\_set\_union\_2();  
}  
  
  
// тест на пересечение множеств  
void test\_unordered\_array\_set\_intersection\_1() {  
 unordered\_array\_set set1 = unordered\_array\_set\_create\_from\_array((int[]){1 , 3, 4}, 3);  
 unordered\_array\_set set2 = unordered\_array\_set\_create\_from\_array((int[]){ 3, 4, 5}, 3);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_intersection(set1, set2);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){3, 4}, 2);  
  
 assert(unordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 unordered\_array\_set\_delete(&set1);  
 unordered\_array\_set\_delete(&set2);  
 unordered\_array\_set\_delete(&res\_set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_intersection\_2() {  
 unordered\_array\_set set1 = unordered\_array\_set\_create\_from\_array((int[]){1 , 2, 3}, 3);  
 unordered\_array\_set set2 = unordered\_array\_set\_create\_from\_array((int[]){ 4, 5, 6}, 3);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_intersection(set1, set2);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){}, 0);  
  
 assert(unordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 unordered\_array\_set\_delete(&set1);  
 unordered\_array\_set\_delete(&set2);  
 unordered\_array\_set\_delete(&res\_set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_intersection() {  
 test\_unordered\_array\_set\_intersection\_1();  
 test\_unordered\_array\_set\_intersection\_2();  
}  
  
  
// тест на вычитание одного множества из другого  
void test\_unordered\_array\_set\_difference\_1() {  
 unordered\_array\_set set1 = unordered\_array\_set\_create\_from\_array((int[]){1 , 2, 3, 4, 5, 6}, 6);  
 unordered\_array\_set set2 = unordered\_array\_set\_create\_from\_array((int[]){ 4, 5, 6}, 3);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_difference(set1, set2);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3}, 3);  
  
 assert(unordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 unordered\_array\_set\_delete(&set1);  
 unordered\_array\_set\_delete(&set2);  
 unordered\_array\_set\_delete(&res\_set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_difference\_2() {  
 unordered\_array\_set set1 = unordered\_array\_set\_create\_from\_array((int[]){1 , 2, 3, 4, 5, 6}, 6);  
 unordered\_array\_set set2 = unordered\_array\_set\_create\_from\_array((int[]){ 7}, 1);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_difference(set1, set2);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 assert(unordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 unordered\_array\_set\_delete(&set1);  
 unordered\_array\_set\_delete(&set2);  
 unordered\_array\_set\_delete(&res\_set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_difference\_3() {  
 unordered\_array\_set set1 = unordered\_array\_set\_create\_from\_array((int[]){1 , 2, 3}, 3);  
 unordered\_array\_set set2 = unordered\_array\_set\_create\_from\_array((int[]){ 3, 2, 1}, 3);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_difference(set1, set2);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){}, 0);  
  
 assert(unordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 unordered\_array\_set\_delete(&set1);  
 unordered\_array\_set\_delete(&set2);  
 unordered\_array\_set\_delete(&res\_set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_difference() {  
 test\_unordered\_array\_set\_difference\_1();  
 test\_unordered\_array\_set\_difference\_2();  
 test\_unordered\_array\_set\_difference\_3();  
}  
  
  
// тест на симметричную разность двух множеств  
void test\_unordered\_array\_set\_symmetricDifference\_1() {  
 unordered\_array\_set set1 = unordered\_array\_set\_create\_from\_array((int[]){1 , 3, 4}, 3);  
 unordered\_array\_set set2 = unordered\_array\_set\_create\_from\_array((int[]){ 3, 4, 5}, 3);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_symmetricDifference(set1, set2);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){1, 5}, 2);  
  
 assert(unordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 unordered\_array\_set\_delete(&set1);  
 unordered\_array\_set\_delete(&set2);  
 unordered\_array\_set\_delete(&res\_set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_symmetricDifference\_2() {  
 unordered\_array\_set set1 = unordered\_array\_set\_create\_from\_array((int[]){1 , 2, 3}, 3);  
 unordered\_array\_set set2 = unordered\_array\_set\_create\_from\_array((int[]){ 4, 5, 6}, 3);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_symmetricDifference(set1, set2);  
  
 unordered\_array\_set check\_set = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 6, 5}, 6);  
  
 assert(unordered\_array\_set\_isEqual(res\_set, check\_set));  
  
 unordered\_array\_set\_delete(&set1);  
 unordered\_array\_set\_delete(&set2);  
 unordered\_array\_set\_delete(&res\_set);  
 unordered\_array\_set\_delete(&check\_set);  
}  
  
  
void test\_unordered\_array\_set\_symmetricDifference() {  
 test\_unordered\_array\_set\_symmetricDifference\_1();  
 test\_unordered\_array\_set\_symmetricDifference\_2();  
}  
  
  
// тест на дополнение множества до универсума  
void test\_unordered\_array\_set\_complement\_1() {  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){1 ,2, 3}, 3);  
 unordered\_array\_set universum = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_complement(set, universum);  
  
 assert(unordered\_array\_set\_isSubset(res\_set, universum));  
  
 unordered\_array\_set\_delete(&set);  
 unordered\_array\_set\_delete(&universum);  
 unordered\_array\_set\_delete(&res\_set);  
}  
  
  
void test\_unordered\_array\_set\_complement\_2() {  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){1 ,2, 3, 4, 5, 6}, 6);  
 unordered\_array\_set universum = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_complement(set, universum);  
  
 assert(unordered\_array\_set\_isSubset(res\_set, universum));  
  
 unordered\_array\_set\_delete(&set);  
 unordered\_array\_set\_delete(&universum);  
 unordered\_array\_set\_delete(&res\_set);  
}  
  
  
void test\_unordered\_array\_set\_complement\_3() {  
 unordered\_array\_set set = unordered\_array\_set\_create\_from\_array((int[]){}, 0);  
 unordered\_array\_set universum = unordered\_array\_set\_create\_from\_array((int[]){1, 2, 3, 4, 5, 6}, 6);  
  
 unordered\_array\_set res\_set = unordered\_array\_set\_complement(set, universum);  
  
 assert(unordered\_array\_set\_isSubset(res\_set, universum));  
  
 unordered\_array\_set\_delete(&set);  
 unordered\_array\_set\_delete(&universum);  
 unordered\_array\_set\_delete(&res\_set);  
}  
  
  
void test\_unordered\_array\_set\_complement() {  
 test\_unordered\_array\_set\_complement\_1();  
 test\_unordered\_array\_set\_complement\_2();  
 test\_unordered\_array\_set\_complement\_3();  
}  
  
  
void test\_unordered\_array\_set() {  
 test\_unordered\_array\_set\_in();  
 test\_unordered\_array\_set\_isSubset();  
 test\_unordered\_array\_set\_deleteElement();  
 test\_unordered\_array\_set\_union();  
 test\_unordered\_array\_set\_intersection();  
 test\_unordered\_array\_set\_difference();  
 test\_unordered\_array\_set\_symmetricDifference();  
 test\_unordered\_array\_set\_complement();  
}  
  
  
  
  
void test() {  
 test\_bitset();  
 test\_ordered\_array\_set();  
 test\_unordered\_array\_set();  
}  
  
  
int main() {  
 test();  
  
 return 0;  
}

Тестовая система не вызвала ошибок, следовательно, функции работают верно:



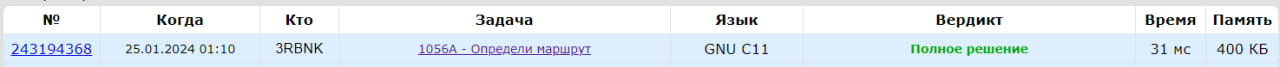
Задания с Codeforces:

Задание 1. Определи маршрут (1056A):

Код программы:

int main() {  
 int n;  
 scanf("%d", &n);  
  
 // считаем первое множество. Будем пересекать остальные множества с этим  
 int r;  
 scanf("%d", &r);  
  
 unordered\_array\_set set = unordered\_array\_set\_create(r);  
 for (int i = 0 ; i < r; i++) {  
 int x;  
 scanf("%d", &x);  
  
 unordered\_array\_set\_insert(&set, x);  
 }  
  
 for (int i = 0; i < n - 1; i++) {  
 scanf("%d", &r);  
  
 unordered\_array\_set subset = unordered\_array\_set\_create(r);  
 for (int j = 0 ; j < r; j++) {  
 int x;  
 scanf("%d", &x);  
  
 unordered\_array\_set\_insert(&subset, x);  
 }  
  
 set = unordered\_array\_set\_intersection(set, subset);  
 }  
  
 unordered\_array\_set\_print(set);  
}

Вердикт тестовой системы:

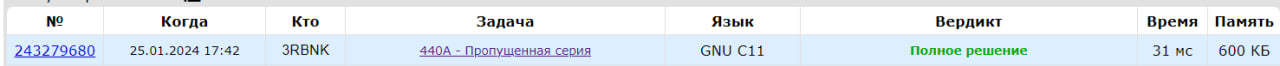


Задание 2. Пропущенная серия (440A):

Код программы:

int main() {  
 int n;  
 scanf("%d", &n);  
  
 // создаём множество и заполняем его элементами от 1 до n  
 ordered\_array\_set set = ordered\_array\_set\_create(n);  
 for (int i = 1; i < n + 1; i++) {  
 ordered\_array\_set\_insert(&set, i);  
 }  
  
 // считываем номера серий, которые просмотренны и удаляем их из множ  
 for (int i = 0; i < n - 1; i++) {  
 int x;  
 scanf("%d", &x);  
  
 ordered\_array\_set\_deleteElement(&set, x);  
 }  
   
 // выводим оставшийся элемент  
 ordered\_array\_set\_print(set);  
  
 return 0;  
}

Вердикт тестовой системы:

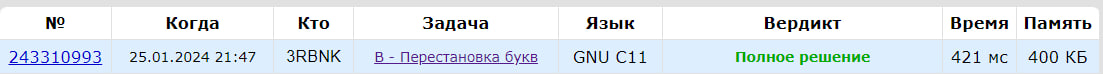


Задание 3. Перестановка букв (1093):

Код программы:

int main() {  
 int t;  
 scanf("%d", &t);  
  
 for (int t\_sets = 0; t\_sets < t; t\_sets++) {  
 // заполняем множество нулями  
 unordered\_array\_set amount = unordered\_array\_set\_create(26);  
 for (int i = 0; i < 26; i++) {  
 amount.data[i] = 0;  
 amount.size++;  
 }  
  
 // считываем строку. Размер ограничен 1000 по условию  
 char str[1000];  
 scanf("%s", str);  
  
 // считываем каждый символ и увеличиваем его счётчик  
 size\_t size\_str = strlen(str);  
 for (size\_t i = 0; i < size\_str; i++) {  
 size\_t char\_index = str[i] - 97;  
 amount.data[char\_index]++;  
 }  
  
 // определяем, возможен ли палиндром из данных букв  
 bool is\_palindrome = true;  
 for (size\_t i = 0; i < 26; i++) {  
 if (amount.data[i] != 0 && amount.data[i] != size\_str)  
 is\_palindrome = false;  
 }  
  
 // если строка - палиндром, вывести -1 иначе, строку  
 if (is\_palindrome)  
 printf("-1\n");  
 else {  
 for (int i = 0; i < 26; i++)  
 for (int j = 0; j < amount.data[i]; j++)  
 printf("%c", 'a' + i);  
 printf("\n");  
 }  
 }  
  
 return 0;  
}

Вердикт тестовой системы:



Задание 4. Тихий класс (1166A):

Код программы:

// возвращает количество пар для числа n  
int counterCouple(int n) {  
 return (n \* (n - 1)) / 2;  
}  
  
  
int main() {  
 // заполняем множество нулями  
 unordered\_array\_set amount = unordered\_array\_set\_create(26);  
 for (size\_t i = 0; i < 26; i++) {  
 amount.data[i] = 0;  
 amount.size++;  
 }  
  
 int n;  
 scanf("%d", &n);  
  
 for (int n\_sets = 0; n\_sets < n; n\_sets++) {  
 // считываем строку. Размер 20 обусловлен ограничениями задачи  
 char s[20];  
 scanf("%s", s);  
   
 // увеличиваем колиество людей по индексу первой буквы имени  
 size\_t char\_index = s[0] - 97;  
 amount.data[char\_index]++;  
 }  
   
 // подсчитываем количество людей в первом/втором классе  
 int result = 0;  
 for (size\_t i = 0; i < 26; i++) {  
 int amount\_first\_class = amount.data[i] / 2;  
 int amount\_second\_class = amount.data[i] % 2 == 0 ? amount.data[i] / 2 : amount.data[i] / 2 + 1;  
  
 result += counterCouple(amount\_first\_class) + counterCouple(amount\_second\_class);  
 }  
   
 printf("%d\n", result);  
  
 return 0;  
}

Вердикт тестовой системы:



Задание 5. Щедрый Кефа (841A):

Код программы:

// возвращает максимальное значение массива a размера n  
int getMax(const int a[], const size\_t n) {  
 int max = a[0];  
  
 for (size\_t i = 1; i < n; i++)  
 if (a[i] > max)  
 max = a[i];  
  
 return max;  
}  
  
  
int main() {  
 int n, k;  
 scanf("%d %d", &n, &k);  
  
 char str[n];  
 scanf("%s", str);  
   
 // создаём множество и заполянем его нулями  
 unordered\_array\_set amount\_balls = unordered\_array\_set\_create(26);  
 for (size\_t i = 0; i < 26; i++) {  
 amount\_balls.data[i] = 0;  
 amount\_balls.size++;  
 }  
  
   
 // заполняем множество количеством шаров определённого цвета  
 for (size\_t i = 0; i < n; i++) {  
 int char\_index = str[i] - 97;  
  
 amount\_balls.data[char\_index]++;  
 }  
  
   
 // опрделяем максимальное количество шаров  
 int max\_amount = getMax(amount\_balls.data, amount\_balls.size);  
  
 if (max\_amount <= k)  
 printf("YES");  
 else  
 printf("NO");  
  
 return 0;  
}

Вердикт тестовой системы:

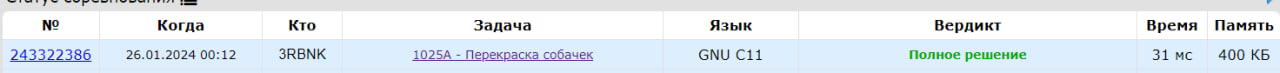


Задание 6. Перекраска собачек (1025A):

Код программы:

// возвращет true, если в массиве a размера n имеется элемент больше двух  
// иначе false  
bool more2(const int a[], const size\_t n) {  
 for (size\_t i = 0; i < n; i++)  
 if (a[i] >= 2)  
 return true;  
 return false;  
}  
  
  
int main() {  
 int n;  
 scanf("%d", &n);  
  
 char str[n];  
 scanf("%s", str);  
  
 // создаём множество и заполянем его нулями  
 unordered\_array\_set amount\_color = unordered\_array\_set\_create(26);  
 for (size\_t i = 0; i < 26; i++) {  
 amount\_color.data[i] = 0;  
 amount\_color.size++;  
 }  
  
  
 // заполняем множество количеством собак определённого цвета  
 for (size\_t i = 0; i < n; i++) {  
 int char\_index = str[i] - 97;  
  
 amount\_color.data[char\_index]++;  
 }  
  
 // определяем, есть ли элемент больше двух  
 bool more\_two = more2(amount\_color.data, amount\_color.size);  
  
 if (n == 1 || more\_two)  
 printf("YES");  
 else  
 printf("NO");  
  
 return 0;  
}

Вердикт тестовой системы:



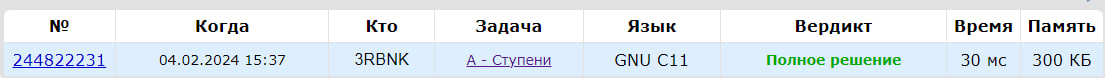
Задание 7. Ступени (1011A):

Код программы:

// возвращет вес ракеты если можно составить ракету размера k из имеющихся ступеней массива a начиная со ступени по индексу pos  
// иначе -1  
int getWeightRocket(const int a[], int pos, int k) {  
 int result = pos + 1;  
 int amount\_stages = 1;  
 int amount\_letter = 26;  
 int i = pos + 2;  
 int last = pos;  
  
 while (amount\_stages < k && i < amount\_letter) {  
 if (a[i] != 0 && i - last > 1) {  
 result += i + 1;  
 amount\_stages++;  
 last = i;  
 }  
  
 i++;  
 }  
 if (amount\_stages != k)  
 return -1;  
 else  
 return result;  
}  
  
int main() {  
 // создаём множество и заполянем его нулями  
 unordered\_array\_set amount = unordered\_array\_set\_create(26);  
 for (size\_t i = 0; i < 26; i++) {  
 amount.data[i] = 0;  
 amount.size++;  
 }  
 int n, k;  
 scanf("%d %d", &n, &k);  
  
 char str[n];  
 scanf("%s", str);  
 // заполняем множество количеством ступеней определённого имени  
 for (size\_t i = 0; i < n; i++) {  
 size\_t char\_index = str[i] - 97;  
  
 amount.data[char\_index]++;  
 }

// определяем индекс первой ступени  
 int index\_first\_element = 0;  
 for (int i = 0; i < 26; i++) {  
 if (amount.data[i] != 0) {  
 index\_first\_element = i;  
 break;  
 }  
 }

int result = getWeightRocket(amount.data, index\_first\_element, k);  
  
 printf("%d", result);  
 return 0;  
}

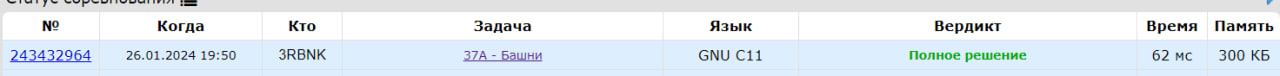
Вердикт тестовой системы:  


Задание 8. Башни (37A):

Код программы:

// возвращает максимальное значение массива a размера n  
int getMax(const int a[], const size\_t n) {  
 int max = a[0];  
  
 for (size\_t i = 0; i < n; i++)  
 if (a[i] > max)  
 max = a[i];  
  
 return max;  
}  
  
  
int main() {  
 // создаём множество и заполянем его нулями  
 unordered\_array\_set amount = unordered\_array\_set\_create(1001);  
 for (size\_t i = 0; i < 1001; i++) {  
 amount.data[i] = 0;  
 amount.size++;  
 }  
  
 int n;  
 scanf("%d", &n);  
  
 // заполняем множество количеством башен определённого размера  
 for (size\_t i = 0; i < n; i++) {  
 int l;  
 scanf("%d", &l);  
  
 amount.data[l]++;  
 }  
  
 // определяем максимальное количество, то есть максимальная высота  
 int max\_amount = getMax(amount.data, amount.size);  
  
 // определяем количество ненулевых башен  
 int amount\_element = 0;  
 for (size\_t i = 0; i < 1001; i++)  
 if (amount.data[i] != 0)  
 amount\_element++;  
  
 printf("%d %d\n", max\_amount, amount\_element);  
  
 return 0;  
}

Вердикт тестовой системы:

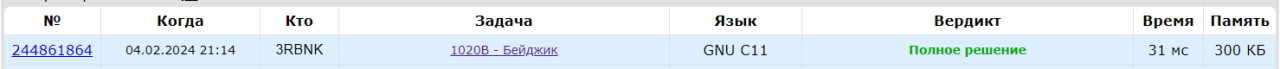


Задание 9. Бейджик (1020B):

Код программы:

// возвращает номер ученика на которого сошлётся ученик из массива a размера n по индексу pos  
int getReference(const int a[], const int n, int pos) {  
 // создаём множество и заполняем его нулями  
 unordered\_array\_set include = unordered\_array\_set\_create(n);  
 for (size\_t i = 0; i < n; i++) {  
 include.data[i] = 0;  
 include.size++;  
 }  
  
 int i = pos;  
  
 include.data[i] = 1;  
  
 // цикл работает до тех пор, пока мы не наткнёмся на ученика, к которому уже обращились  
 while (include.data[a[i]] == 0) {  
 include.data[a[i]] = 1;  
 i = a[i];  
 }  
  
 // очищаем память от множества  
 unordered\_array\_set\_delete(&include);  
  
 return a[i] + 1;  
}  
  
  
int main() {  
 int n;  
 scanf("%d", &n);  
   
 // создаём массив, где i элементы указывает на кого ссылается ученик  
 int pupils[n];  
 for (size\_t i = 0; i < n; i++) {  
 int x;  
 scanf("%d", &x);  
  
 pupils[i] = x - 1;  
 }  
   
 for (int pos = 0; pos < n; pos++) {  
 printf("%d ", getReference(pupils, n, pos));;  
 }  
  
 return 0;  
}

Вердикт тестовой системы:



Задание 10. Разнообразие — это хорошо (672B):

Код программы:

int main() {  
 // создаём множество и заполянем его нулями  
 unordered\_array\_set amount = unordered\_array\_set\_create(26);  
 for (size\_t i = 0; i < 26; i++) {  
 amount.data[i] = 0;  
 amount.size++;  
 }  
  
 int n;  
 scanf("%d", &n);  
  
 char str[n];  
 scanf("%s", str);  
  
 // заполняем множество количеством букв  
 for (size\_t i = 0; i < n; i++) {  
 size\_t char\_index = str[i] - 97;  
  
 amount.data[char\_index]++;  
 }  
   
 // опрделяем количество изменений   
 int res = 0;  
 for (size\_t i = 0; i < 26; i++) {  
 if (amount.data[i] > 1)  
 res += amount.data[i] - 1;  
 }  
   
 // если строка длинне 26, то нельзя будет сделать её только из уникальных элементов  
 if (n > 26)  
 printf("-1");  
 else  
 printf("%d", res);  
  
 return 0;  
}

Вердикт тестовой системы:



Задание 11. Игра: Банковские карты (777B):

Код программы:

unordered\_array\_set create\_set(char\* str, int n) {  
 unordered\_array\_set set;  
 set.size = n;  
 set.capacity = n;  
 set.data = (int\*) malloc(n \* sizeof(int));  
  
 for (int i = 0; i < n; i++) {  
 set.data[i] = str[i] - '0';  
 }  
  
 qsort(set.data, set.size, sizeof(int), compare);  
   
 return set;  
}  
  
int main() {  
 int n;  
 scanf("%d", &n);  
 char sherlock[n+1], moriarty[n+1];  
 scanf("%s", sherlock);  
 scanf("%s", moriarty);  
  
 unordered\_array\_set sherlock\_set = create\_set(sherlock, n);  
 unordered\_array\_set moriarty\_set = create\_set(moriarty, n);  
  
 int min\_slaps = 0, max\_slaps = 0;  
 int j = 0;  
 for (int i = 0; i < n; i++) {  
 while (j < n && moriarty\_set.data[j] < sherlock\_set.data[i]) {  
 j++;  
 }  
 if (j < n) {  
 j++;  
 } else {  
 min\_slaps++;  
 }  
 }  
 j = 0;  
 for (int i = 0; i < n; i++) {  
 while (j < n && moriarty\_set.data[j] <= sherlock\_set.data[i]) {  
 j++;  
 }  
 if (j < n) {  
 max\_slaps++;  
 j++;  
 }  
 }  
  
 printf("%d\n", min\_slaps);  
 printf("%d\n", max\_slaps);  
  
 free(sherlock\_set.data);  
 free(moriarty\_set.data);  
  
 return 0;  
}

Вердикт тестовой системы:

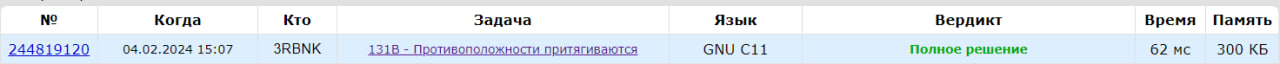


Задание 12. Противоположности притягиваются (131B):

Код программы:

int main() {  
 // создаём множество и заполянем его нулями, для элементов от -10 до 10   
 unordered\_array\_set amount = unordered\_array\_set\_create(21);  
 for (size\_t i = 0; i < 21; i++) {  
 amount.data[i] = 0;  
 amount.size++;  
 }  
  
 int n;  
 scanf("%d", &n);  
   
 // считываем число и увеличиваем количество таких чисел. используем сдвиг +10, чтобы элемент со значением -1 оказался на 0 индексе  
 for (int n\_sets = 0; n\_sets < n; n\_sets++) {  
 long long int x;  
 scanf("%lld", &x);  
  
 amount.data[x + 10]++;  
 }  
   
 // определяем количество пар  
 long long int result = 0;  
 for (size\_t i = 0; i < 10; i++) {  
 result += amount.data[i] \* amount.data[20 - i];  
 }  
   
 // отдельно рассматриваем 0, т.к. он составляем пару с самим собой  
 result += (amount.data[10] \* (amount.data[10] - 1)) / 2;  
  
 printf("%lld", result);  
  
 return 0;  
}

Вердикт тестовой системы:



Вывод: в ходе выполнения лабораторной работы мы успешно закрепили навыки работы с структурами данных и изучили простые способы представления множеств в памяти ЭВМ. Работа с структурами позволяет нам организовывать данные таким образом, чтобы обеспечить удобство и эффективность их обработки. Изучение способов представления множеств в памяти ЭВМ дало нам понимание того, как компьютер хранит и обрабатывает информацию о множествах, что является важным аспектом в разработке программного обеспечения. Эта лабораторная работа помогла нам не только углубить знания о структурах данных, но и понять, как они используются на практике для решения конкретных задач. Мы убедились в важности эффективного управления данными и их представлении для создания эффективных и мощных программ. Благодаря этой работе мы теперь осознаем, что понимание работы со структурами данных и способами представления множеств помогает нам стать более компетентными программистами и решать задачи более эффективно.