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 №5а

по дисциплине: Основы программирования
тема: «Множества»

Выполнил: ст. группы ПВ-211
Чувилко Илья Романович

Проверили:
Притчин Иван Сергеевич
Черников Сергей Викторович

Белгород 2021 г.

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

Содержание отчета:

- Тема лабораторной работы.
- Цель лабораторной работы.
- Решения задач:
  - Текст задания.
  - Исходный код библиотек (исходный и заголовочный файлы).
  - Исходный код тестов для любого (одного) произвольного типа представления множеств.
  - Решение задач с *codeforces* требуется для получения максимального балла по лабораторной.
  - – Представление множеств на упорядоченных массивах требуется для получения максимального балла по лабораторной.

Задания к лабораторной работе:

Задача №1. Выполнить реализацию множества на типе `uint32_t`. Содержимое файла `bitset.h`:

Код программы:

```c
#include "bitSet.h"

// Выводит значения множества set
void bitset_output(bitset set) {
  printf("[");
  for (size_t i = 0; i <= set.maxValue; i++) {
    if ((set.value >> i) & 1)
      printf("%zd, ", i);
  }
  printf("\b\b]");
}

// возвращает пустое множество с универсумом 0, 1,..., maxValue
bitset bitset_create(unsigned maxValue) {
  return (bitset) {~0, maxValue};
}

// возвращает значение 'истина', если значение value имеется в множестве set
// иначе - 'ложь'
bool bitset_in(bitset set, unsigned int value) {
  assert(set.maxValue >= value);
  return set.value >> value & 1;
}

// возвращает значение 'истина', если множества set1 и set2 равны
// иначе - 'ложь'
bool bitset_isEqual(bitset set1, bitset set2) {
  assert(set1.maxValue == set2.maxValue);
  return (set1.value & set2.value) == set1.value;
}

// возвращает значение 'истина' если множество subset
// является подмножеством множества set, иначе - 'ложь'.
bool bitset_isSubset(bitset subset, bitset set) {
  assert(set.maxValue == subset.maxValue);
  return (set.value & subset.value) != 0;
}

// добавляет элемент value в множество set
void bitset_insert(bitset *set, unsigned int value) {
  assert(set->maxValue >= value);
  set->value |= 1 << value;
}
```

```
// удаляет элемент value из множества set
void bitset_deleteElement(bitset *set, unsigned int value) {
  assert(set->maxValue >= value);
  set->value &= ~(1 << value);
}

// возвращает объединение множеств set1 и set2
bitset bitset_union(bitset set1, bitset set2) {
  assert(set1.maxValue == set2.maxValue);
  bitset set;
  set.value = set1.value | set2.value;
  return set;
}

// возвращает пересечение множеств set1 и set2
bitset bitset_intersection(bitset set1, bitset set2) {
  assert(set1.maxValue == set2.maxValue);
  bitset set;
  set.value = set1.value & set2.value;
  set.maxValue = set1.maxValue;
  return set;
}

// возвращает разность множеств set1 и set2
bitset bitset_difference(bitset set1, bitset set2) {
  assert(set1.maxValue == set2.maxValue);
  return (bitset) {(set1.value & ~set2.value), set1.maxValue};
}

// возвращает симметрическую разность множеств set1 и set2
bitset bitset_symmetricDifference(bitset set1, bitset set2) {
  assert(set1.maxValue == set2.maxValue);
  return (bitset) {(set1.value & ~set2.value) | (~set1.value & set2.value),
                   set1.maxValue};
}

// возвращает дополнение до универсума множества set
bitset bitset_complement(bitset set) {
  bitset u = bitset_create(set.maxValue);
  return (bitset) {u.value ^ set.value, set.maxValue};
}
```

Задача №2. На неупорядоченном массиве:

Код программы:

```c
#include <malloc.h>
#include "unorderedset.h"
#include "array.h"

// возвращает пустое множество для capacity элементов
unordered_array_set unordered_array_set_create(size_t capacity) {
  unordered_array_set s;
  s.data = (int *) calloc(capacity, sizeof(int));
  s.size = 0;
  s.capacity = capacity;
  return s;
}

// возвращает множество, состоящее из элементов массива a размера size
unordered_array_set unordered_array_set_create_from_array(
  const int *a, size_t size) {
  unordered_array_set s = unordered_array_set_create(size);
  for (size_t i = 0; i < size; i++)
    s.data[i] = a[i];
  s.size = size;
  s.capacity = size;
  return s;
}

// 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) {
  for (size_t i = 0; i < set->size; i++)
    if (set->data[i] == value)
      return i;
  return set->size;
}

// возвращает значение 'истина', если элементы множеств set1 и set2 равны
// иначе - 'ложь'
bool unordered_array_set_isEqual(
  unordered_array_set set1, unordered_array_set set2
) {
  return set1.size == set2.size && unordered_array_set_isSubset(set1, set2);
}

// возвращает значение 'истина' если множество subset
// является подмножеством множества set, иначе - 'ложь'.
bool unordered_array_set_isSubset(
  unordered_array_set subset, unordered_array_set set
) {
  for (size_t i = 0; i < subset.size; i++)
    if (unordered_array_set_in(&set, subset.data[i]) == set.size)
      return false;
  return true;
}

// возбуждает исключение, если в множество по адресу set
// нельзя вставить элемент
void unordered_array_set_isAbleAppend(unordered_array_set *set) {
  if (set->size == set->capacity) {
    set->data = realloc(set->data, sizeof(int) * (set->size + 1));
    set->capacity++;
  }
}
```

```c
// добавляет элемент value в множество set
void unordered_array_set_insert(
  unordered_array_set *set, int value
) {
  unordered_array_set_isAbleAppend(set);
  if (isUniqueNumber(set->data, set->size, value))
    set->data[set->size++] = value;
}

// удаляет элемент value из множества set
void unordered_array_set_deleteElement(
  unordered_array_set *set, int value
) {
  assert(set->size > 0);
  size_t index = unordered_array_set_in(set, value);
  if (index != set->size)
    set->data[index] = set->data[--set->size];
}

// возвращает объединение множеств set1 и set2
unordered_array_set unordered_array_set_union(
  unordered_array_set set1, unordered_array_set set2
) {
  unordered_array_set set3 =
    unordered_array_set_create(set1.size + set2.size);
  for (size_t i = 0; i < set1.size; i++)
    unordered_array_set_insert(&set3, set1.data[i]);
  for (size_t i = 0; i < set2.size; i++)
    unordered_array_set_insert(&set3, set2.data[i]);
  return set3;
}

// возвращает пересечение множеств set1 и set2
unordered_array_set unordered_array_set_intersection(
  unordered_array_set set1, unordered_array_set set2
) {
  unordered_array_set set3 = unordered_array_set_create(set1.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(&set3, set1.data[i]);
  return set3;
}

// возвращает разность множеств set1 и set2
unordered_array_set unordered_array_set_difference(
  unordered_array_set set1, unordered_array_set set2
) {
  unordered_array_set set3 = unordered_array_set_create(set1.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(&set3, set1.data[i]);
  }
  return set3;
}

// возвращает симметрическую разность множеств set1 и set2
unordered_array_set unordered_array_set_symmetricDifference(
  unordered_array_set set1, unordered_array_set set2
) {
  unordered_array_set set3 = unordered_array_set_difference(set1, set2);
  for (size_t i = 0; i < set2.size; i++)
    if (unordered_array_set_in(&set1, set2.data[i]) == set1.size)
      unordered_array_set_insert(&set3, set2.data[i]);
  return set3;
}
```

```c
// освобождает память, занимаемую множеством set
void unordered_array_set_delete(unordered_array_set set) {
  free(set.data);
  set.size = 0;
  set.capacity = 0;
}

// вывод множества set
void unordered_array_set_print(unordered_array_set set) {
  printf("[");
  if (set.size == 0)
    printf("]");
  else {
    for (size_t i = 0; i < set.size; i++)
      printf("%d, ", set.data[i]);
    printf("\b\b]\n");
  }
}

// возвращает дополнение до универсума множества set
unordered_array_set unordered_array_set_complement(
  unordered_array_set universumSet, unordered_array_set set
) {
  return (unordered_array_set)
    unordered_array_set_difference(universumSet, set);
}
```

Тесты для неупорядоченного массива:

```c
#include "bitSet.h"
#include "unorderedset.h"
#include "orderedarrayset.h"


void test_unordered_array_set_in1() {
  unordered_array_set set =
    unordered_array_set_create_from_array((int[]) {1, 2}, 2);
  size_t resTest = 1;
  assert(unordered_array_set_in(&set, 2) == resTest);

  unordered_array_set_delete(set);
}

void test_unordered_array_set_in2() {
  unordered_array_set set =
    unordered_array_set_create_from_array((int[]) {1, 2}, 2);
  size_t resTest = 2;
  assert(unordered_array_set_in(&set, 3) == resTest);

  unordered_array_set_delete(set);
}

void test_unordered_array_set_in() {
  test_unordered_array_set_in1();
  test_unordered_array_set_in2();
}

void test_unordered_array_set_isSubset1() {
  unordered_array_set set =
    unordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  unordered_array_set subset =
    unordered_array_set_create_from_array((int[]) {2, 3, 4}, 3);
  bool isSubset = true;
  assert(unordered_array_set_isSubset(subset, set) == isSubset);

  unordered_array_set_delete(set);
  unordered_array_set_delete(subset);
}

void test_unordered_array_set_isSubset2() {
  unordered_array_set set =
    unordered_array_set_create_from_array((int[]) {1, 2, 3, 9, 5}, 5);
  unordered_array_set subset =
    unordered_array_set_create_from_array((int[]) {2, 3, 4}, 3);
  bool isSubset = false;
  assert(unordered_array_set_isSubset(subset, set) == isSubset);

  unordered_array_set_delete(set);
  unordered_array_set_delete(subset);
}

void test_unordered_array_set_isSubset() {
  test_unordered_array_set_isSubset1();
  test_unordered_array_set_isSubset2();
}
```

```
void test_unordered_array_set_insert1() {
  unordered_array_set set =
    unordered_array_set_create(1);
  unordered_array_set_insert(&set, 4);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {4}, 1);
  assert(unordered_array_set_isEqual(set, resSet));

  unordered_array_set_delete(set);
  unordered_array_set_delete(resSet);
}

void test_unordered_array_set_insert2() {
  unordered_array_set set =
    unordered_array_set_create(5);
  unordered_array_set_insert(&set, 4);
  unordered_array_set_insert(&set, 56);
  unordered_array_set_insert(&set, 1000);
  unordered_array_set_insert(&set, -42);
  unordered_array_set_insert(&set, 42);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {42, -42, 1000, 56, 4}, 5);
  assert(unordered_array_set_isEqual(set, resSet));

  unordered_array_set_delete(set);
  unordered_array_set_delete(resSet);
}

void test_unordered_array_set_insert() {
  test_unordered_array_set_insert1();
  test_unordered_array_set_insert2();
}

void test_unordered_array_set_deleteElement1() {
  unordered_array_set set =
    unordered_array_set_create_from_array((int[]) {1, 2, 3}, 3);
  unordered_array_set_deleteElement(&set, 2);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {1, 3}, 2);
  assert(unordered_array_set_isEqual(set, resSet));

  unordered_array_set_delete(set);
  unordered_array_set_delete(resSet);
}

void test_unordered_array_set_deleteElement2() {
  unordered_array_set set =
    unordered_array_set_create_from_array((int[]) {1, 2, 3}, 3);
  unordered_array_set_deleteElement(&set, 2);
  unordered_array_set_deleteElement(&set, 1);
  unordered_array_set_deleteElement(&set, 4);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {3}, 1);
  assert(unordered_array_set_isEqual(set, resSet));

  unordered_array_set_delete(set);
  unordered_array_set_delete(resSet);
}

void test_unordered_array_set_deleteElement() {
  test_unordered_array_set_deleteElement1();
  test_unordered_array_set_deleteElement2();
}
```

```
void test_unordered_array_set_union1() {
  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[]) {2, 3}, 2);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {1, 2, 3}, 3);
  unordered_array_set testSet =
    unordered_array_set_union(set1, set2);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(set2);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}

void test_unordered_array_set_union2() {
  unordered_array_set set1 =
    unordered_array_set_create_from_array((int[]) {9, 2}, 2);
  unordered_array_set set2 =
    unordered_array_set_create_from_array((int[]) {7, 3}, 2);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {2, 3, 7, 9}, 4);
  unordered_array_set testSet =
    unordered_array_set_union(set1, set2);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(set2);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}

void test_unordered_array_set_union3() {
  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, 2}, 2);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {1, 2}, 2);
  unordered_array_set testSet =
    unordered_array_set_union(set1, set2);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(set2);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}

void test_unordered_array_set_union() {
  test_unordered_array_set_union1();
  test_unordered_array_set_union2();
  test_unordered_array_set_union3();
}
```

```
void test_unordered_array_set_intersection1() {
  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[]) {2, 3}, 2);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {2}, 1);
  unordered_array_set testSet =
    unordered_array_set_intersection(set1, set2);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(set2);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}

void test_unordered_array_set_intersection2() {
  unordered_array_set set1 =
    unordered_array_set_create_from_array((int[]) {9, 2}, 2);
  unordered_array_set set2 =
    unordered_array_set_create_from_array((int[]) {7, 3}, 2);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {}, 0);
  unordered_array_set testSet =
    unordered_array_set_intersection(set1, set2);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(set2);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}

void test_unordered_array_set_intersection() {
  test_unordered_array_set_intersection1();
  test_unordered_array_set_intersection2();
}

void test_unordered_array_set_difference1() {
  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[]) {2, 3}, 2);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {1}, 1);
  unordered_array_set testSet =
    unordered_array_set_difference(set1, set2);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(set2);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}
```

```c
void test_unordered_array_set_difference2() {
  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[]) {2, 3}, 2);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {1}, 1);
  unordered_array_set testSet =
    unordered_array_set_difference(set1, set2);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(set2);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}

void test_unordered_array_set_difference() {
  test_unordered_array_set_difference1();
  test_unordered_array_set_difference2();
}

void test_unordered_array_set_symmetricDifference1() {
  unordered_array_set set1 =
    unordered_array_set_create_from_array((int[]) {1, 2, 8}, 3);
  unordered_array_set set2 =
    unordered_array_set_create_from_array((int[]) {2, 3, 1}, 3);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {3, 8}, 2);
  unordered_array_set testSet =
    unordered_array_set_symmetricDifference(set1, set2);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(set2);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}

void test_unordered_array_set_symmetricDifference2() {
  unordered_array_set set1 =
    unordered_array_set_create_from_array((int[]) {1, 42}, 2);
  unordered_array_set set2 =
    unordered_array_set_create_from_array((int[]) {5}, 1);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {1, 5, 42}, 3);
  unordered_array_set testSet =
    unordered_array_set_symmetricDifference(set1, set2);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(set2);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}

void test_unordered_array_set_symmetricDifference() {
  test_unordered_array_set_symmetricDifference1();
  test_unordered_array_set_symmetricDifference2();
}
```

```
void test_unordered_array_set_complement1() {
  unordered_array_set set1 =
    unordered_array_set_create_from_array((int[]) {1}, 1);
  unordered_array_set setCompliment =
    unordered_array_set_create_from_array((int[]) {1, 42}, 2);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {42}, 1);
  unordered_array_set testSet =
    unordered_array_set_complement(setCompliment, set1);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(setCompliment);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}

void test_unordered_array_set_complement2() {
  unordered_array_set set1 =
    unordered_array_set_create_from_array((int[]) {1, 53}, 2);
  unordered_array_set setCompliment =
    unordered_array_set_create_from_array((int[]) {1, 42, 53, 12}, 4);
  unordered_array_set resSet =
    unordered_array_set_create_from_array((int[]) {42, 12}, 2);
  unordered_array_set testSet =
    unordered_array_set_complement(setCompliment, set1);
  assert(unordered_array_set_isEqual(resSet, testSet));

  unordered_array_set_delete(set1);
  unordered_array_set_delete(setCompliment);
  unordered_array_set_delete(resSet);
  unordered_array_set_delete(testSet);
}

void test_unordered_array_set_complement() {
  test_unordered_array_set_complement1();
  test_unordered_array_set_complement2();
}
```

Задача №3. На упорядоченном массиве:

Код программы:

```c
#include <malloc.h>
#include "orderedarrayset.h"
#include "array.h"

// возвращает пустое множество, в которое можно вставить capacity элементов
ordered_array_set ordered_array_set_create(size_t capacity) {
  ordered_array_set s;
  s.data = (int *) calloc(capacity, sizeof(int));
  s.size = 0;
  s.capacity = capacity;
  return s;
}

// возвращает множество, состоящее из элементов массива a размера size
ordered_array_set ordered_array_set_create_from_array(const int *a,
                                                      size_t size) {
  ordered_array_set s = ordered_array_set_create(size);
  for (size_t i = 0; i < size; i++)
    s.data[i] = a[i];
  s.size = size;
  s.capacity = size;
  return s;
}

// возвращает значение 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) {
  return binarySearch_(set->data, set->size, value);
}

// возвращает значение 'истина', если элементы множеств set1 и set2 равны
// иначе - 'ложь'
bool ordered_array_set_isEqual(ordered_array_set set1,
                               ordered_array_set set2) {
  if (set1.size != set2.size)
    return false;
  for (size_t i = 0; i < set1.size; i++)
    if (set1.data[i] != set2.data[i])
      return false;
  return true;
}

// возвращает значение 'истина', если subset является подмножеством set
// иначе - 'ложь'
bool ordered_array_set_isSubset(ordered_array_set subset,
                                ordered_array_set set) {
  for (size_t i = 0; i < subset.size; i++)
    if (ordered_array_set_in(&set, subset.data[i]) == set.size)
      return false;
  return true;
}

// возбуждает исключение, если в множество по адресу set
// нельзя вставить элемент
void ordered_array_set_isAbleAppend(ordered_array_set *set) {
  if (set->size == set->capacity) {
    set->data = realloc(set->data, sizeof(int) * (set->size + 1));
    set->capacity++;
  }
}
```

```c
// добавляет элемент value в множество set
void ordered_array_set_insert(ordered_array_set *set, int value) {
  if (isUniqueNumber(set->data, set->size, value)) {
    size_t index = binarySearchMoreOrEqual_(set->data, set->size, value);
    ordered_array_set_isAbleAppend(set);
    insert_(set->data, &set->size, index, value);
  }
}

void ordered_array_set_deleteElement(ordered_array_set *set, int value) {
  if (set->size == 0)
    return;
  size_t index = binarySearch_(set->data, set->size, value);
  if (index < set->size)
    deleteByPosSaveOrder_(set->data, &set->size, index - 1);
}

ordered_array_set ordered_array_set_union(ordered_array_set set1,
                                          ordered_array_set set2) {
  ordered_array_set set3 =
    ordered_array_set_create(set1.size + set2.size);
  for (size_t i = 0; i < set1.size; i++)
    set3.data[i] = set1.data[i];
  set3.size = set1.size;
  for (size_t i = 0; i < set2.size; i++)
    ordered_array_set_insert(&set3, set2.data[i]);
  return set3;
}

ordered_array_set ordered_array_set_intersection(
  ordered_array_set set1, ordered_array_set set2
) {
  ordered_array_set set3 =
    ordered_array_set_create(set1.size);
  for (size_t i = 0; i < set1.size; i++)
    if (ordered_array_set_in(&set2, set1.data[i]) != set2.size)
      ordered_array_set_insert(&set3, set1.data[i]);
  return set3;
}

ordered_array_set ordered_array_set_difference(
  ordered_array_set set1, ordered_array_set set2
) {
  ordered_array_set set3 =
    ordered_array_set_create(set1.size);
  for (size_t i = 0; i < set1.size; i++)
    if (ordered_array_set_in(&set2, set1.data[i]) == set2.size)
      ordered_array_set_insert(&set3, set1.data[i]);
  return set3;
}

ordered_array_set ordered_array_set_symmetricDifference(
  ordered_array_set set1, ordered_array_set set2
) {
  ordered_array_set set3 = ordered_array_set_difference(set1, set2);
  for (size_t i = 0; i < set2.size; i++)
    if (ordered_array_set_in(&set1, set2.data[i]) == set1.size)
      ordered_array_set_insert(&set3, set2.data[i]);
  return set3;
}

ordered_array_set ordered_array_set_complement(
  ordered_array_set universumSet, ordered_array_set set
) {
  return (ordered_array_set)
    ordered_array_set_difference(universumSet, set);
}
```

```c
// освобождает память, занимаемую множеством set
void ordered_array_set_delete(ordered_array_set set) {
  free(set.data);
  set.size = 0;
  set.capacity = 0;
}

// вывод множества set
void ordered_array_set_print(ordered_array_set set) {
  printf("[");
  if (set.size == 0) {
    printf("]");
    return;
  }
  for (size_t i = 0; i < set.size; i++)
    printf("%d, ", set.data[i]);
  printf("\b\b]\n");
}
```

Тесты для неупорядоченного массива:

```c
void test_ordered_array_set_in1() {
  ordered_array_set set =
    ordered_array_set_create_from_array((int[]) {1, 53}, 2);
  assert(ordered_array_set_in(&set, 5) == 2);


  ordered_array_set_delete(set);
}

void test_ordered_array_set_in2() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {1, 53}, 2);
  assert(ordered_array_set_in(&set1, 1) == 0);

  ordered_array_set_delete(set1);
}

void test_ordered_array_set_in() {
  test_ordered_array_set_in1();
  test_ordered_array_set_in2();
}

void test_ordered_array_set_isSubset1() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {1, 53}, 2);
  ordered_array_set set2 =
    ordered_array_set_create_from_array((int[]) {1, 12, 53, 102}, 4);
  assert(ordered_array_set_isSubset(set1, set2));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(set2);
}

void test_ordered_array_set_isSubset2() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {1, 5}, 2);
  ordered_array_set set2 =
    ordered_array_set_create_from_array((int[]) {1, 42, 53, 102}, 4);
  assert(ordered_array_set_isSubset(set1, set2) == 0);

  ordered_array_set_delete(set1);
  ordered_array_set_delete(set2);
}
```

```
void test_ordered_array_set_isSubset() {
  test_ordered_array_set_isSubset1();
  test_ordered_array_set_isSubset2();
}

void test_ordered_array_set_insert1() {
  ordered_array_set set =
    ordered_array_set_create(1);
  ordered_array_set_insert(&set, 4);
  ordered_array_set_insert(&set, 4);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {4}, 1);
  assert(ordered_array_set_isEqual(set, resSet));

  ordered_array_set_delete(set);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_insert2() {
  ordered_array_set set =
    ordered_array_set_create(5);
  ordered_array_set_insert(&set, 4);
  ordered_array_set_insert(&set, 2);
  ordered_array_set_insert(&set, 1);
  ordered_array_set_insert(&set, 5);
  ordered_array_set_insert(&set, 3);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  assert(ordered_array_set_isEqual(set, resSet));

  ordered_array_set_delete(set);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_insert() {
  test_ordered_array_set_insert1();
  test_ordered_array_set_insert2();
}

void test_ordered_array_set_deleteElement1() {
  ordered_array_set set =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set_deleteElement(&set, 4);
  ordered_array_set_deleteElement(&set, 2);
  ordered_array_set_deleteElement(&set, 1);
  ordered_array_set_deleteElement(&set, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {3}, 1);

  assert(ordered_array_set_isEqual(set, resSet));
  ordered_array_set_delete(set);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_deleteElement2() {
  ordered_array_set set =
    ordered_array_set_create_from_array((int[]) {}, 0);
  ordered_array_set_deleteElement(&set, 4);
  ordered_array_set_deleteElement(&set, 2);
  ordered_array_set_deleteElement(&set, 1);
  ordered_array_set_deleteElement(&set, 5);
  ordered_array_set_deleteElement(&set, 3);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {}, 0);
  assert(ordered_array_set_isEqual(set, resSet));
```

```
  ordered_array_set_delete(set);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_deleteElement() {
  test_ordered_array_set_deleteElement1();
  test_ordered_array_set_deleteElement2();
}

void test_ordered_array_set_union1() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {3, 4, 5, 6, 7}, 5);
  ordered_array_set set2 =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5, 6, 7}, 7);
  ordered_array_set testSet =
    ordered_array_set_union(set1, set2);
  assert(ordered_array_set_isEqual(testSet, resSet));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(set2);
  ordered_array_set_delete(testSet);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_union2() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set set2 =
    ordered_array_set_create_from_array((int[]) {6, 7, 8, 9, 10}, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array(
      (int[]) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, 10);
  ordered_array_set testSet =
    ordered_array_set_union(set1, set2);
  assert(ordered_array_set_isEqual(testSet, resSet));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(set2);
  ordered_array_set_delete(resSet);
  ordered_array_set_delete(testSet);
}

void test_ordered_array_set_union() {
  test_ordered_array_set_union1();
  test_ordered_array_set_union2();
}

void test_ordered_array_set_intersection1() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {3, 4, 5, 6, 7}, 5);
  ordered_array_set set2 =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {3, 4, 5}, 3);
  ordered_array_set testSet =
    ordered_array_set_intersection(set1, set2);
  assert(ordered_array_set_isEqual(testSet, resSet));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(set2);
  ordered_array_set_delete(testSet);
  ordered_array_set_delete(resSet);
}
```

```
void test_ordered_array_set_intersection2() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {6, 7, 8, 9, 10}, 5);
  ordered_array_set set2 =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {}, 0);
  ordered_array_set testSet =
    ordered_array_set_intersection(set1, set2);
  assert(ordered_array_set_isEqual(testSet, resSet));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(set2);
  ordered_array_set_delete(testSet);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_intersection() {
  test_ordered_array_set_intersection1();
  test_ordered_array_set_intersection2();
}

void test_ordered_array_set_difference1() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {3, 4, 5, 6, 7}, 5);
  ordered_array_set set2 =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {6, 7}, 2);
  ordered_array_set testSet =
    ordered_array_set_difference(set1, set2);
  assert(ordered_array_set_isEqual(testSet, resSet));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(set2);
  ordered_array_set_delete(testSet);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_difference2() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {3, 4, 5, 6, 7}, 5);
  ordered_array_set set2 =
    ordered_array_set_create_from_array((int[]) {1, 2, 11, 12, 13}, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {3, 4, 5, 6, 7}, 5);
  ordered_array_set testSet =
    ordered_array_set_difference(set1, set2);
  assert(ordered_array_set_isEqual(testSet, resSet));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(set2);
  ordered_array_set_delete(testSet);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_difference() {
  test_ordered_array_set_difference1();
  test_ordered_array_set_difference2();
}
```

```c
void test_ordered_array_set_symmetricDifference1() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {3, 4, 5, 6, 7}, 5);
  ordered_array_set set2 =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {1, 2, 6, 7}, 4);
  ordered_array_set testSet =
    ordered_array_set_symmetricDifference(set1, set2);
  assert(ordered_array_set_isEqual(testSet, resSet));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(set2);
  ordered_array_set_delete(testSet);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_symmetricDifference2() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {6, 7, 8, 9, 10}, 5);
  ordered_array_set set2 =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array(
      (int[]) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, 10
    );
  ordered_array_set testSet =
    ordered_array_set_symmetricDifference(set1, set2);
  assert(ordered_array_set_isEqual(testSet, resSet));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(set2);
  ordered_array_set_delete(testSet);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_symmetricDifference() {
  test_ordered_array_set_symmetricDifference1();
  test_ordered_array_set_symmetricDifference2();
}

void test_ordered_array_set_complement1() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {3, 4, 5, 6, 7}, 5);
  ordered_array_set setComplement =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {1, 2}, 2);
  ordered_array_set testSet =
    ordered_array_set_complement(setComplement, set1);
  assert(ordered_array_set_isEqual(testSet, resSet));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(setComplement);
  ordered_array_set_delete(testSet);
  ordered_array_set_delete(resSet);
}
```

```c
void test_ordered_array_set_complement2() {
  ordered_array_set set1 =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set setComplement =
    ordered_array_set_create_from_array((int[]) {1, 2, 3, 4, 5}, 5);
  ordered_array_set resSet =
    ordered_array_set_create_from_array((int[]) {}, 0);
  ordered_array_set testSet =
    ordered_array_set_complement(setComplement, set1);
  assert(ordered_array_set_isEqual(testSet, resSet));

  ordered_array_set_delete(set1);
  ordered_array_set_delete(setComplement);
  ordered_array_set_delete(testSet);
  ordered_array_set_delete(resSet);
}

void test_ordered_array_set_complement() {
  test_ordered_array_set_complement1();
  test_ordered_array_set_complement2();
}

void test() {
  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();
}

int main() {
  test();

  return 0;
}
```