УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите теоретические сведения к лабораторной работе, выполните практическое задание.

Результаты работы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 17.04.2023.

Требования к отчету:

Отчет предоставляется преподавателю в электронном варианте и должен содержать:

- **название работы, постановку цели, вывод;**
- **ответы на контрольные вопросы, указанные преподавателем.**

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лабораторная работа № 14

Тема «Алгоритмы сортировки и поиска целочисленного массива»

Цель: изучить понятие массива, научиться сортировать одномерные массивы.

Теоретические сведения

Сортировка данных может сделать поиск внутри массива более эффективным не только для людей, но и для компьютеров. Например, рассмотрим случай, когда нам нужно узнать, отображается ли определённое имя в списке имён. Чтобы это узнать, нужно проверить каждый элемент массива на соответствие с нашим значением. Поиск в массиве с множеством элементов может оказаться слишком неэффективным (затратным).

Однако, предположим, что наш массив с именами отсортирован в алфавитном порядке. Тогда наш поиск начинается с первой буквы нашего значения и заканчивается буквой, которая идёт следующей по алфавиту. В таком случае, если мы дошли до этой буквы и не нашли имя, то точно знаем,

что оно не находится в остальной части массива, так как в алфавитном порядке нашу букву мы уже прошли!

Не секрет, что есть алгоритмы поиска внутри отсортированных массивов и получше. Используя простой алгоритм, мы можем искать определённый элемент в отсортированном массиве, содержащем 1 000 000 элементов, используя всего лишь 20 сравнений! Недостатком, конечно же, является то, что сортировка массива с таким огромным количеством элементов — дело сравнительно затратное, и оно точно не выполняется ради одного поискового запроса.

В некоторых случаях сортировка массива делает поиск ненужным. Например, мы ищем наилучший результат студента за тест. Если массив не отсортирован, то нам придётся просмотреть каждый элемент массива, чтобы найти наивысшую оценку. Если же массив отсортирован, то наивысшая оценка будет находиться либо на первой позиции, либо на последней (в зависимости от метода сортировки массива: в порядке возрастания или в порядке убывания), поэтому нам не нужно искать вообще!

Сортировка обычно выполняется путём повторного сравнения пар элементов массива и замены значений, если они отвечают определённым критериям. Порядок, в котором эти элементы сравниваются, зависит от того, какой алгоритм сортировки используется. Критерии состоят из того, как будет сортироваться массив (например, в порядке возрастания или в порядке убывания).

Чтобы поменять два элемента местами, мы можем использовать функцию **std::swap()** из стандартной библиотеки C++, которая определена в **заголовочном файле** algorithm. В C++11 std::swap() был перенесён в заголовочный файл utility:

```cpp
#include <iostream>
#include <algorithm> // для std::swap. В C++11 используйте заголовок <utility>

int main()
{
    int a = 3;
    int b = 5;
    std::cout << "Before swap: a = " << a << ", b = " << b << '\n';
    std::swap(a, b); // меняем местами значения переменных a и b
    std::cout << "After swap:  a = " << a << ", b = " << b << '\n';
}
```

Результат выполнения программы выше:

```
Before swap: a = 3, b = 5
After swap: a = 5, b = 3
```

После выполнения операции замены значения переменных `a` и `b` поменялись местами.

Сортировка массивов методом выбора

Существует множество способов сортировки массивов. Сортировка массивов методом выбора, пожалуй, самая простая для понимания, хоть и одна из самых медленных.

Для **сортировки массива методом выбора от наименьшего до наибольшего элемента** выполняются следующие шаги:

Начиная с элемента под индексом 0, ищем в массиве наименьшее значение.

Найденное значение меняем местами с нулевым элементом.

Повторяем шаги №1 и №2 уже для следующего индекса в массиве.

Другими словами, мы ищем наименьший элемент в массиве и перемещаем его на первое место. Затем ищем второй наименьший элемент и перемещаем его уже на второе место после первого наименьшего элемента. Этот процесс продолжается до тех пор, пока в массиве не закончатся неотсортированные элементы.

Вот пример работы этого алгоритма в массиве с 5-ью элементами:

{ 30, 50, 20, 10, 40 }

Сначала ищем наименьший элемент, начиная с индекса 0:

{ 30, 50, 20, **10**, 40 }

Затем меняем местами наименьший элемент с элементом под индексом 0:

{ **10**, 50, 20, **30**, 40 }

Теперь, когда первый элемент массива отсортирован, мы его игнорируем. Ищем следующий наименьший элемент, но уже начиная с индекса 1:

{ *10*, 50, **20**, 30, 40 }

И меняем его местами с элементом под индексом 1:

{ *10*, **20**, **50**, 30, 40 }

Теперь мы можем игнорировать первые два элемента. Ищем следующий наименьший элемент, начиная с индекса 2:

{ *10*, *20*, 50, **30**, 40 }

И меняем его местами с элементом под индексом 2:

{ *10*, *20*, **30**, **50**, 40 }

Ищем следующий наименьший элемент, начиная с индекса 3:

{ *10*, *20*, *30*, 50, **40** }

И меняем его местами с элементом под индексом 3:

{ *10*, *20*, *30*, **40**, **50** }

Ищем следующий наименьший элемент, начиная с индекса 4:

{ *10*, *20*, *30*, *40*, **50** }

И меняем его местами с элементом под индексом 4 (выполняется самозамена, т.е. ничего не делаем):

{ *10*, *20*, *30*, *40* **50** }

Готово!

{ 10, 20, 30, 40, 50 }

Обратите внимание, последнее сравнение всегда будет одиночным (т.е. самозамена), что является лишней операцией, поэтому, фактически, мы можем остановить выполнение сортировки перед последним элементом массива.

Сортировка массивов методом выбора в C++

Вот как этот алгоритм реализован в C++:

```cpp
#include <iostream>
#include <algorithm> // для std::swap. в C++11 используйте заголовок <utility>

int main()
{
    const int length = 5;
    int array[length] = { 30, 50, 20, 10, 40 };

    // Перебираем каждый элемент массива
    // (кроме последнего, он уже будет отсортирован к тому времени, когда мы до него доберёмся)
    for (int startIndex = 0; startIndex < length - 1; ++startIndex)
    {
        // В переменной smallestIndex хранится индекс наименьшего значения, которое мы нашли в этой итерации
        // Начинаем с того, что наименьший элемент в этой итерации - это первый элемент (индекс 0)
        int smallestIndex = startIndex;

        // Затем ищем элемент поменьше в остальной части массива
        for (int currentIndex = startIndex + 1; currentIndex < length; ++currentIndex)
        {
            // Если мы нашли элемент, который меньше нашего наименьшего элемента,
            if (array[currentIndex] < array[smallestIndex])
                // то запоминаем его
                smallestIndex = currentIndex;
        }

        // smallestIndex теперь наименьший элемент
                // Меняем местами наше начальное наименьшее число с тем, которое мы обнаружили
        std::swap(array[startIndex], array[smallestIndex]);
    }

    // Теперь, когда весь массив отсортирован - выводим его на экран
    for (int index = 0; index < length; ++index)
        std::cout << array[index] << ' ';

    return 0;
}
```

Наиболее запутанной частью этого алгоритма является **цикл** внутри другого цикла (так называемый вложенный цикл). Внешний цикл (startIndex) перебирает элементы один за другим (поочерёдно). В каждой итерации внешнего цикла внутренний цикл (currentIndex) используется для поиска наименьшего элемента среди элементов, которые остались в массиве (начиная со startIndex + 1). smallestIndex отслеживает индекс наименьшего элемента, найденного внутренним циклом. Затем smallestIndex меняется значением со startIndex. И, наконец, внешний цикл (startIndex) передаёт этот элемент, и процесс повторяется.

Подсказка: Если у вас возникли проблемы с пониманием того, как работает программа выше, то попробуйте записать её выполнение на листке бумаги. Запишите начальные (неотсортированные) элементы массива горизонтально в строке в верхней части листа. Нарисуйте стрелки, указывающие, какие элементы являются startIndex, currentIndex и smallestIndex на данный момент. Прокрутите выполнение программы

вручную и перерисуйте стрелки по мере изменения индексов. После каждой итерации внешнего цикла нарисуйте новую строку, показывающую текущее состояние массива (расположение его элементов).

Сортировка текста выполняется с помощью того же алгоритма. Просто измените тип массива из **int**-а в **std::string** и инициализируйте его с помощью соответствующих значений.

Ход работы.

Разработанный одномерный массив в лабораторных работах №№ 12-13 отсортировать в соответствии с таблицей 1.

Таблица 1

Вариант	Одномерный массив	
	Сортировка	
1	Простой обмен	Отсортировать по возрастанию только четные элементы массива.
2	Простой выбор	Удалить из массива все четные элементы.
3	Простое включение	Найти количество простых чисел в массиве.
4	Простой обмен	Найти количество чисел Фибоначчи в массиве.
5	Простой выбор	Удалить все простые числа из массива.
6	Простое включение	Удалить из массива все числа Фибоначчи.
7	Простой обмен	Отсортировать по возрастанию только положительные элементы массива.
8	Простой выбор	Удалить из массива все элементы с четными номерами.

9	Простое включение	Отсортировать по возрастанию только те элементы массива, которые являются простыми числами.
10	Простой обмен	Удалить из массива все элементы равные min(a[l],a[3],...a[2n- 1]).
11	Простой выбор	Создать новый массив из номеров элементов, значения которых равны 0.
12	Простое включение	Сформировать массив, в котором будут только элементы исходного массива, заканчивающиеся на цифру 4.
13	Простой обмен	Отсортировать по возрастанию только четные элементы массива.
14	Простой выбор	Удалить из массива все четные элементы.
15	Простое включение	Найти количество простых чисел в массиве.
16	Простой обмен	Найти количество чисел Фибоначчи в массиве.
17	Простой выбор	Удалить все простые числа из массива.
18	Простое включение	Удалить из массива все числа Фибоначчи.
19	Простой обмен	Отсортировать по возрастанию только положительные элементы массива.
20	Простой выбор	Удалить из массива все элементы с четными номерами.
21	Простое включение	Отсортировать по возрастанию только те элементы массива, которые являются простыми числами.

22	Простой обмен	Удалить из массива все элементы равные min(a[l],a[3],...a[2n- 1]).
23	Простой выбор	Создать новый массив из номеров элементов, значения которых равны 0.
24	Простое включение	Сформировать массив, в котором будут только элементы исходного массива, заканчивающиеся на цифру 5.
25	Простой обмен	Сформировать два массива. В первый массив включить элементы из исходного массива с четными номерами, а во второй с нечетными.