УДК 372.862

ПОИСК ЭФФЕКТИВНЫХ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧ ПРИ ОБУЧЕНИИ ПРОГРАММИРОВАНИЮ

Дмитриев В.Л.
Стерлитамакский филиал Башкирского государственного университета
Российская Федерация, г. Стерлитамак

В данной работе рассматривается несколько эффективных алгоритмов решения некоторого класса задач при обучении программированию. Под эффективностью алгоритма здесь понимается выполнение требований по минимальному использованию памяти программой (в рамках условия задачи) и быстроте работы алгоритма (минимальное время выполнения программы).

В большинстве случаев при изучении программирования, как в школе, так и в вузе, редко обращается внимание на поиск наиболее эффективных решений той или иной задачи. По крайней мере, этот вопрос обычно не рассматривается, если предложенный алгоритм решения задачи дает более-менее хорошие результаты в плане скорости вычислений (при этом, зачастую, об экономии памяти речи вообще не идет).

Однако многие задачи, даже являющиеся достаточно простыми, могут быть решены более эффективными алгоритмами. Простота таких задач, очевидно, и является причиной того, что решение, не лежащее на поверхности, а требующее некоторых мыслительных операций, так и остается не найденным.

Ниже мы рассмотрим пару задач, демонстрирующих возможности использования эффективных алгоритмов. Похожие задачи могут быть предложены при обучении программированию как олимпиадные задачи, причем в условиях оценки задач такой олимпиады должно быть обязательно указано, что максимальный балл будет давать именно оригинальный эффективный алгоритм. Так, автором была организована олимпиада по программированию в дистанционной форме среди студентов физико-математического факультета продолжительностью шесть дней. Задания (по одному в день) высылались участникам ежедневно в одно и то же определенное положением о проведении олимпиады временем на адреса электронной почты, и в течение двух часов участник должен был прислать решение. При подведении итогов олимпиады оценивалась эффективность использования памяти, скорость работы алгоритма, его оригинальность, и, конечно же, время решения задачи участником. Результаты олимпиады показали, что только около 20% студентов смогли предложить эффективные алгоритмы решения некоторых задач. Стоит отметить, что при проведении дистанционной олимпиады необходимо придумывать новые задачи, не встречающиеся в сети Интернет, а также активно использовать при проверке решений программы, позволяющие выявить плагиат (например, система eTXT Антиплагиат).

Задачи на поиск эффективных алгоритмов пользуются большим успехом у участников, развивают и стимулируют их мыслительные и конструктивные способности, алгоритмическое мышление, способствуют развитию творческих способностей.

Рассмотрим теперь две задачи, демонстрирующие скорость и уникальность эффективных алгоритмов. Первая задача предложена автором данной статьи, как и ее решение; вторая задача достаточно известна и часто встречается на просторах Интернета. Входные и выходные данные обеих задач размещаются в файлах.

Задача 1. Дан одномерный массив, содержащий n целых положительных элементов, причем значение каждого из элементов этого массива не менее 1 и не более n. Найти элементы массива, которые встречаются в нем ровно по 1 разу.

Входные данные: В первой строке входного файла INPUT.TXT записано число n –

количество элементов в массиве $1 \le n \le 10^6$. В остальных строках файла через пробел идет перечисление элементов.

Выходные данные: В выходной файл OUTPUT.TXT нужно вывести элементы, встречающиеся в массиве только по 1 разу (последовательность вывода элементов не имеет значения).

Таблица 1. Примеры содержимого входного и выходного файлов к задаче 1

№	INPUT.TXT	OUTPUT.TXT
1	10 1 3 7 8 9 3 2 1 7 3	2 8 9
2	31 1 27 30 23 16 3 4 5 7 9 19 30 13 13 12 28 28 17 22 11 2 2 3 4 4 5 16 17 20 29 18	1 7 9 11 12 18 19 20 22 23 27 29
3	100 11 23 45 67 87 65 45 45 45 32 32 34 45 6 7 1 2 3 4 5 6 7 8 9 2 3 4 9 76 65 23 45 6 7 8 9 8 7 6 5 4 3 2 1 1 2 3 4 5 6 7 8 9 90 87 67 56 45 34 56 67 67 67 67 78 78 78 87 87 8 7 3 4 5 1 2 3 4 5 6 7 8 9 34 34 32 32 32 32 56 78 89 99 87 87 12 11 13 14 15	12 13 14 15 76 89 90 99

Как может показаться, для решения задачи достаточно описать массив необходимого размера, заполнить его нулями и затем при чтении данных из файла сразу увеличивать элемент массива с индексом, соответствующим значению считываемого элемента, на единицу. После окончания чтения данных из файла остается просмотреть весь массив и вывести в качестве результата те номера индексов массива, которые будут содержать в качестве значения единицы – это и будут элементы, встречающиеся по одному разу. Однако для данной постановки задачи это – неверное решение, так как в условии сказано, что дан одномерный массив, содержащий элементы. Поэтому прежде, чем решать задачу, мы должны обязательно поместить данные из файла в массив (файл здесь выступает лишь для удобства ввода данных в программу).

Допустим, мы учли отмеченный момент, и теперь для решения задачи нам достаточно использовать еще один массив такой же размерности, и уже в него записывать количество раз встречаемости соответствующих элементов исходного массива. Но в этом случае мы будем нерационально использовать память. Значит, надо попытаться отыскать другой способ решения.

Такой способ существует, и заключается в следующем. Мы знаем, что значение элементов массива не превышает n. Поэтому задача может быть решена в два прохода по массиву. Во время первого прохода к элементу, номер которого соответствует значению элемента первоначального массива, добавляем значение, равное $n+1$. Во время второго прохода остается проверить полученный массив и выбрать номера элементов, значения которых изменились только на $n+1$. Такие номера и будут значениями элементов первоначального массива, которые встречались только по одному разу.

Для того, чтобы во время первого прохода знать первоначальное значение исходного элемента массива (в результате операций суммирования на предыдущих шагах любой элемент исходного массива мог измениться), необходимо рассматривать не сам этот элемент, а остаток от деления этого элемента на значение, равное $n+1$.

Исходный код программы на C++, выполняющей необходимые расчеты, приведен ниже. Для более подробного ознакомления с особенностями языка программирования C++ можно использовать работы [1, 4] – в них достаточно подробно рассматриваются основные приемы программирования на C++, а также приведено большое количество практических примеров.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{ ifstream F;
  long i=0, n;
  long *a= new long [n];
  F.open("INPUT.txt",ios::in);
  if (F) { F>>n;
          while (!F.eof()) F>>a[i++]; } else cout<<"Файл не найден!"<<endl;
  F.close();
  for(i=0; i<n; i++) a[a[i]%(n+1)-1]+=(n+1);  // собственно, само решение
  ofstream f;
  f.open("OUTPUT.txt");
  for(i=0; i<n; i++)  if ((int)a[i]/(n+1) == 1) f<<(i+1)<<" ";
  f.close();
  delete[]a; return 0; }
```

Как видим, приведенный способ основывается на использовании некоторых дополнительных (и, что важно, обратимых) преобразований над элементами исходного массива, что позволяет экономить память.

Задача 2. В сказочной стране по поводу празднования совершеннолетия принцессы решено было устроить грандиозный бал, на который пригласили всех жителей страны. Тем не менее, желающие попасть на бал должны были соблюдать одно простое условие: на бал можно приходить лишь парами. Так как жителей в стране очень много, то на входе пары решено было нумеровать целыми числами, в том числе и отрицательными. Происходило это следующим образом. Пара совместным решением выбирала любое понравившееся ей целое число (по модулю не превышающее 10^9), и, если оно оказывалось не занятым другой парой, то каждый член этой пары получал выбранное число и наклеивал его на свою одежду так, чтобы его было видно всем.

Уже после начала бала служба безопасности короля выяснила, что на бал попал один житель без пары. Так как было неизвестно, какой номер он мог получить, то решено было проверить наличие всех пар. Ситуация осложняется тем, что на балу пары уже разбились, потому жители, относящиеся к одной паре, могли находиться в разных местах огромного зала. Не прерывая торжество, служба безопасности решила вписывать номера встреченных на балу жителей в специально отведенный для этого журнал, а уже потом сравнить, какие из записанных чисел имеют пары, и, таким образом, выявить номер нарушителя. Чтобы не посчитать несколько раз одного и того же жителя, каждому уже учтенному жителю дополнительно наклеивалась на одежду красная звездочка.

Помогите службе безопасности выяснить, кто из жителей проник на бал без пары.

Входные данные: Строки входного файла INPUT.TXT представляют собой журнал службы безопасности, причем последним числом в нем записано n – общее количество жителей, пришедших на бал (в это число входит и нарушитель). Известно, что число n не превышает 10^6+1.

Выходные данные: В выходной файл OUTPUT.TXT нужно вывести единственное число – номер нарушителя, нелегально проникшего на бал.

Таблица 2. Примеры содержимого входного и выходного файлов к задаче 2

№	INPUT.TXT	OUTPUT.TXT
1	1 3 7 8 8 1 5 7 25 25 3 11	5

2	1 3 -7 8 8 1 135 -7 25 25 3 -2 4 17 81 6 -2 17 4 81 9 6 9 23	135
3	-117 67 45 34 34 -117 67 45 1000 234567890 18 18 234567890 1000000000 343 677 87 -19 -25 1000000000 -19 677 343 -25 87 1000000001 23 1000000001 23 455 455 1000 1000000000 999999999 999999999 6789 1000000000 99 6789 39	99

В условии данной задачи, напротив, про массив ничего не сказано. Более того, указано, что строки входного файла и представляют собой журнал службы безопасности. Поэтому задачу можно решать и без записи данных из файла в массив. Вообще, оптимальное решение этой задачи не предусматривает использования какого-либо массива, и основано на применении инструкции XOR (исключающее ИЛИ) к элементам, хранящимся в файле. При решении задачи таким способом (да и другими тоже) необходимо исключить из рассмотрения последний элемент в файле, так как он хранит количество элементов, и при решении не требуется. Исходный код программы на языке C++ представлен ниже.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
 { long a, b=0;
   ifstream F;
   F.open("INPUT.txt",ios::in);
   if (F) {
     while (!F.eof())
     { F>>a;
       if (!F.eof()) b=b^a; } } else cout<<"Файл не найден!"<<endl;
   F.close();
   ofstream f;
   f.open("OUTPUT.txt");
   f<<b;
   f.close();
   return 0; }
```

Рассмотренные примеры показывают, насколько красивым, оригинальным, коротким и эффективным может быть решение той или иной задачи: остается научить учащихся заниматься поиском таких решений. Поэтому одной из главных целей при изучении программирования должно стать приобретение учащимися навыков конструктивного мышления, то есть умения видеть различные варианты реализации поставленной задачи [2, 3].

Список литературы

1. Дмитриев В.Л. Теория и практика программирования на C++. – Стерлитамак: РИО СФ БашГУ, 2013. – 308 с.
2. Дмитриев В.Л., Ахмадеева Р.З. Развитие конструктивного мышления при изучении программирования // Информатика и образование. № 2, 2009. – С. 69-73.
3. Окулов С.М. Программирование в алгоритмах. – М.: БИНОМ. Лаборатория знаний, 2002. – 341 с.
4. Stroustrup Bjarne. The C++ programming language / Bjarne Stroustrup. – Fourth edition. – Boston: Addison-Wesley, 2013. – 1368 p.