РЕФЕРАТИВНО-ЭКСПЕРИМЕНТАЛЬНАЯ РАБОТА

ИНФОРМАТИКА

ЭФФЕКТИВНОСТЬ АЛГОРИТМОВ

Выполнил:

Мищенко Алексей Николаевич

учащийся 10 класса

Тверского лицея

Руководитель:

Наумова Алиса Ивановна

учитель информатики высшей категории

Тверского лицея

Оглавление

Введение

В данной работе представлен материал по *оптимизации* составления алгоритмов и программ. Тема достаточно *актуальная* и представляет *повышенный интерес* для учащихся профильных классов.

Хотя в определении алгоритма требуется лишь конечность числа шагов, требуемых для достижения результата, на практике выполнение даже хотя бы нескольких шагов является слишком *медленным*. Также обычно есть и другие ограничения (на размер программы, на допустимые действия). В связи с этим вводят такие понятия, как *сложность алгоритма (временная, по размеру программы, вычислительная и др.)*.

Для каждой задачи может существовать множество алгоритмов, приводящих к цели. *Увеличение эффективности алгоритмов составляет одну из задач современной информатики* [1].

Цель данной работы заключается в том, *чтобы научиться составлять эффективные алгоритмы и программы из различных предметных областей.*

Задача состоит в том, чтобы подобрать *соответствующий материал* с последующей систематизацией, обобщением и иллюстрацией текста.

Работа состоит из двух частей: *теоретической* (дано описание основных параметров оптимизации) и *практической* (приведёны примеры решения задач на компьютере с использованием языка программирования DEV-C++ из курса математики, физики и химии).

Основная часть

Эффективность алгоритмов

§ 1. История разработок эффективных алгоритмов

Ранние электронные компьютеры были очень ограничены как по *скорости*, так и по *памяти*. В некоторых случаях было осознано, что существует *компромисс времени и памяти*, при котором задача должна либо использовать большое количество памяти для достижения высокой скорости, либо использовать более медленный алгоритм, использующий небольшое количество рабочей памяти. В этом случае использовался наиболее *быстрый алгоритм*, для которого было достаточно имеющейся памяти.

Современные компьютеры много *быстрее* тех ранних компьютеров и имеют много *больше* памяти (гигабайты вместо килобайт). Тем не менее, *эффективность* остаётся важным фактором не только с учётом конфигурации компьютера, но и в умении составлять *эффективные* алгоритмы и программы [4].

§ 2. Аппаратные (HandWare) и программные средства (SoftWare)

2.1. Время работы процессора (CPU)

Для анализа работы алгоритма обычно используется анализ *временно́й* сложности алгоритма, чтобы оценить время работы как функцию от *размера входных данных*, особенно в случае обработки большого количества данных.

Многие языки программирования содержат функции, отражающие *время использования процессора (рис.1)*. Для долго работающих алгоритмов затраченное время также может представлять интерес.

Рис.1. Центральный процессор

Немаловажную роль играет также *мультипроцессорная и мультизадачная* среда, т.е. возможность работы других программ в то же самое время.

2.2. Использование основной памяти (RAM)

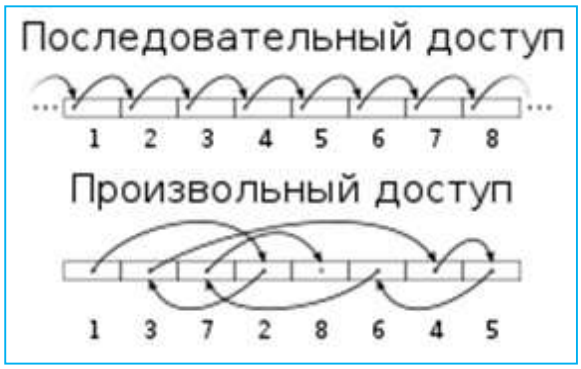


Рис.2. Произвольный доступ

Операти́вная па́мять (англ. *Random Access Memory, RAM*, память с *произвольным* доступом) или операти́вное запомина́ющее устро́йство (ОЗУ) — энергозависимая часть системы компьютерной памяти *(рис.2, рис. 3)*;

во время работы компьютера в ней хранится *выполняемый машинный код* программы, а также *входные, выходные* и *промежуточные данные*, обрабатываемые процессором *(рис.4)* [4].

Рис.3. Модули ОЗУ для ПК

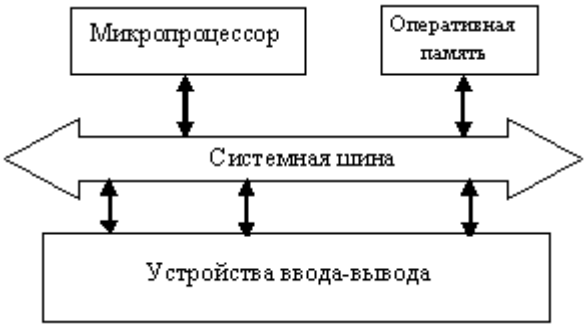


Рис. 4. Схема взаимодействия RAM и CPU

Существует *четыре* аспекта использования основной памяти:

- Количество памяти, необходимой для хранения кода алгоритма.
- Количество памяти, необходимое для входных данных.
- Количество памяти, необходимое для любых выходных данных (некоторые алгоритмы, такие как сортировки, часто переставляют входные данные и не требуют дополнительной памяти для выходных данных).

- Количество памяти, необходимое для вычислительного процесса во время вычислений (сюда входят именованные переменные и любое стековое пространство, необходимое для вызова подпрограмм).

Ранние электронные компьютеры и домашние компьютеры имели относительно *малый объём* рабочей памяти. Так, в 1949 EDSAC имел максимальную рабочую память 1024 17-битных слов, а в 1980 Sinclair ZX80 выпускался с 1024 байтами рабочей памяти *(табл.1)*.

Таблица №1. Первые ЭВМ и ПК

EDSAC	ZX80
EDSAC (англ. ***E**lectronic **D**elay **S**torage **A**utomatic **C**alculator*) — электронная вычислительная машина, созданная в 1949 году в Кембриджском университете (Великобритания) группой разработчиков во главе с Морисом Уилксом.	домашний персональный компьютер, выпущенный на рынок в 1980 году компанией Sinclair Research (Кембридж, Англия).
	

2.3. Использование кэш-памяти (высокая скорость и минимизация памяти)

Алгоритм, необходимая память для которого укладывается в кэш компьютера, работает много быстрее, чем алгоритм, умещающийся в основную память, который, в свою очередь, будет много быстрее алгоритма, который использует виртуальное пространство. Усложняет ситуацию факт, что некоторые системы имеют до трёх уровней кэша. Различные системы имеют

различное количество этих типов памяти, так что эффект памяти для алгоритма может существенно отличаться *при переходе от одной системы к другой*.

В ранние дни электронных вычислений, если алгоритм и его данные не помещались в основную память, он не мог использоваться. В наши дни использование виртуальной памяти обеспечивает огромную память, но за счёт производительности. Если алгоритм и его данные умещаются в кэш, можно получить *очень высокую скорость*, так что минимизация требуемой памяти помогает минимизировать время [4].

2.4. Выбор языка программирования (реализация алгоритма)

Вопросы *реализации* алгоритма могут также повлиять на *фактическую эффективность*. Это касается *выбора языка программирования* и способа, каким алгоритм фактически закодирован, выбора *транслятора* для выбранного языка или используемых *опций компилятора (рис.5)*. Большинство компиляторов переводит программу с некоторого высокоуровневого языка программирования в *машинный код*, который может быть непосредственно выполнен физическим процессором. Как правило, этот код также ориентирован на исполнение в среде *конкретной операционной системы*, поскольку использует предоставляемые ею возможности. Результат компиляции — исполнимый программный модуль — обладает *максимально возможной производительностью*.

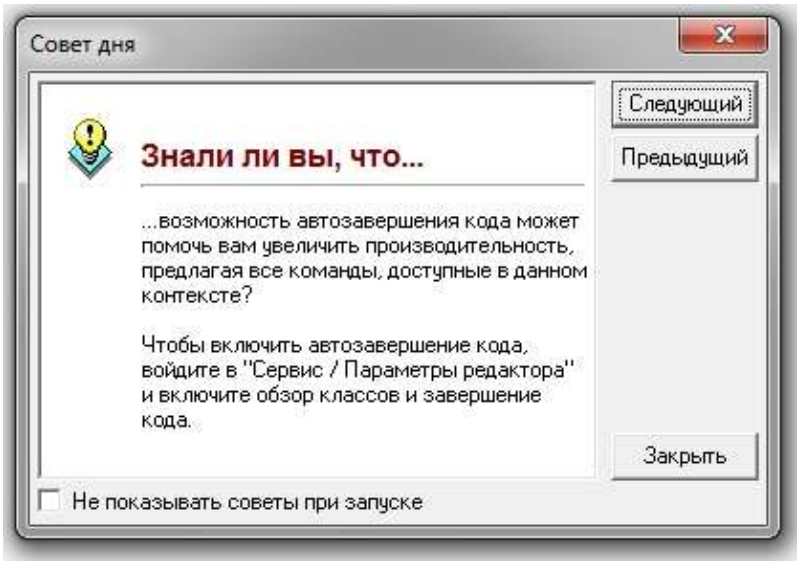


Рис.5. Информация для увеличения производительности в среде DEV-C++

В некоторых случаях язык, реализованный в виде *интерпретатора* (построчный анализ программы), может оказаться *существенно медленнее*, чем язык, реализованный в виде *компилятора* [4].

ПРОЕКТНАЯ ЧАСТЬ

РАЗРАБОТКА ПРАВИЛЬНЫХ И ЭФФЕКТИВНЫХ ПРОГРАММ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ DEV-C++

Подготовка к ЕГЭ. Решение задач в среде программирования Dev-C++ (часть 2, №27 (правильное и эффективное решение задачи))

Учебная литература и интернет-ресурсы:

ЕГЭ. Информатика и ИКТ: Типовые экзаменационные варианты / С.С. Крылов, Т.Е. Чуркина. – М. : Издательство “Национальное образование ”, 2018. – 208 с. – (ЕГЭ. ФИПИ – школе)

Программное обеспечение:

OS Windows 7

Среда программирования Dev-C++

Задание №27 (Б)

§ 3. Решение задачи из курса математики

Имеется набор данных, состоящий из 5 троек положительных чисел. Необходимо выбрать из каждой тройки ровно одно число так, чтобы сумма всех выбранных чисел не делилась на 3 и при этом была максимально возможной.

Задание:

Напишите программу для решения поставленной задачи, которая будет *эффективна*, как по времени, так и по памяти.

Программа считается эффективной по времени, если время работы программы пропорционально количеству троек чисел N, т. е. при увеличении N в k раз время работы программы должно увеличиваться не более чем в k раз.

Программа считается эффективной по памяти, если размер памяти, использованной в программе для хранения данных, не зависит от числа N и не превышает 1 килобайта.

Перед программой кратко опишите использованный алгоритм и укажите версию языка программирования.

3.1. Словесное описание эффективного алгоритма

Чтобы получить максимально возможную сумму, будем брать из каждой тройки наибольшее число. Если полученная при этом сумма будет кратна 3, её придётся уменьшить. Для этого достаточно в одной из троек, где хотя бы два числа имеют разные остатки при делении на 3, заменить ранее выбранное число на число с другим остатком от деления на 3 из той же тройки. При этом модуль разности между старым и новым выбранным из тройки числом должен быть минимально возможным [2].

3.2. Создание эффективной по памяти и времени программы на Dev-C++ 4.9.9.2

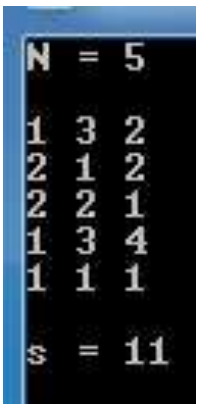
```
#include<iostream>              //для ввода/вывода данных
#include<conio.h>              //необходимо для задержки экрана
using namespace std;
int main()                     //начало исполнения программы (функция main)
{
   const int aMax=10000;       //максимально возможное исходное число
   int N;                      //количество троек
   int a,b,c,tmp;              //тройка чисел
   int s;                      //сумма выбранных чисел
   int Dmin;                   //минимальная разница в тройке, не кратная 3
   s=0;                        //начальное значение переменной s
   Dmin=aMax+1;                //начальное значение переменной Dmin
   cout << "N=";
   cin >> N;cout << endl;       //ввести количество троек
      for (int i = 0; i < N; i++)
      {
         cin >> a >> b >> c;    //ввести три числа через пробел
```

```
        if (a < b) {tmp = a; a = b; b = tmp;}
        if (b < c) {tmp = b; b = c; c = tmp;}
        s = s + a;                  //найти сумму максимальных чисел
        if (((a-b) % 3 != 0) && ((a-b) < Dmin)) Dmin = a-b;
        if (((a-c) % 3 != 0) && ((a-c) < Dmin)) Dmin = a-c;
    }
        if ((s % 3 != 0) && (Dmin > aMax)) s=0;
  cout << endl;
  cout << "s = " << s;            //вывод суммы s, которая не делится на 3
  getch();                        //задержка экрана
  return 0;                       //показывает успешное завершение программы
}                                 //конец функции main
```

3.3. Подготовка тестовых примеров и выполнение

На вход программе в первой строке подаётся количество троек чисел N (1 ≤ N ≤ 100 000). Каждая из следующих строк содержит три натуральных числа (введите через пробел), не превышающих 10 000 (*табл. 2*) [3].

Таблица №2. Тестирование программного кода

Тестовый пример		*Выполнение*
Входные данные	*Выходные данные*	
5 1 3 2 2 1 2 2 2 1 1 3 4 1 1 1	11	

§ 4. Решение задачи из курса физики

Датчик передаёт каждую секунду по каналу связи неотрицательное целое число, не превосходящее 1000, - текущий результат измерений. Временем, в течение которого происходит передача, можно пренебречь.

Необходимо найти в заданной серии показаний датчика *минимальное чётное произведение двух показаний*, между моментами передачи которых прошло не менее 15 секунд.

Если получить такое произведение не удаётся, ответ считается равным -1. Общее количество показаний датчика в серии не превышает 10000.

Задание:

Напишите программу для решения поставленной задачи, которая будет *эффективна*, как по времени, так и по памяти (или хотя бы по одной из этих характеристик).

Программа считается эффективной по времени, если время работы программы пропорционально количеству полученных показаний прибора N, т. е. при увеличении N в k раз время работы программы должно увеличиваться не более чем в k раз.

Программа считается эффективной по памяти, если размер памяти, использованной в программе для хранения данных, не зависит от числа N и не превышает 1 килобайта.

Перед программой кратко опишите использованный алгоритм и укажите версию языка программирования.

4.1. Словесное описание эффективного алгоритма

Чтобы произведение было чётным, хотя бы один сомножитель должен быть чётным, поэтому при поиске подходящих произведений чётные показания прибора можно рассматривать в паре с любыми другими, а нечётные – только с чётными.

Для каждого показания с номером k, начиная с k = 16, рассмотрим все допустимые по условиям задачи пары, в которых данное показание получено вторым. Минимальное произведение из всех этих пар будет получено, если

первым в паре будет взято минимальное подходящее показание среди всех, полученных от начала приёма и до показания с номером k - 15. Если очередное показание чётное, минимальное среди предыдущих может быть любым, если нечётное – только чётным.

Для получения эффективного по времени решения нужно по мере ввода данных помнить абсолютное минимальное и минимальное чётное показание на каждый момент времени, каждое вновь полученное показание умножить на соответствующий ему минимум, имевшийся на 15 элементов ранее, и выбрать минимальное из всех таких произведений [2].

4.2. Создание эффективной по памяти и времени программы на Dev-C++ 4.9.9.2

```
#include<iostream>        //для ввода/вывода данных
#include<conio.h>         //необходимо для задержки экрана
using namespace std;
int main()                //начало исполнения программы (функция main)
{
   const int s = 15;      //требуемое расстояние между показаниями
   int amax = 1001;       //больше максимально возможного показания
   int N;
   int a[s];              //хранение s показаний датчика
   int a_;                //ввод очередного показания
   int ma;                //минимальное число без s последних
   int me;                //минимальное чётное число без s последних
   int mp;                //минимальное значение произведения
   int p;
      cout << "N=";
      cin >> N; cout << endl;     //ввести количество показаний
      for (int i = 0; i < s; i++)
      {
         cin >> a[i];             //ввод первых s чисел (значения датчика)
```

```
    }
    //ввод остальных значений, поиск минимального произведения
    ma = amax; me = amax;
    mp = amax * amax;
       cout << endl;
       for (int i = s; i < N; i++)
       {
       cin >> a_;
       if (a[0] < ma)
       ma = a[0];
       if ((a[0] % 2 == 0) && (a[0] < me))   //минимальное чётное произведение
       me = a[0];
       if (a_ % 2 == 0)
       p = a_ * me;
       else
       if (me < amax)
       p = a_ * me;
       else
       p = amax * amax;
       if (p < mp)
       mp = p;
       else
       mp = -1;                                  //заданное произведение не получено
       // сдвигаем элементы вспомогательного массива влево
       for (int j = 0; j < s; j++)
       a[j] = a[j+1];
       a[s] = a_;
       }
//вывод результата (минимальное чётное произведение двух показаний)
cout << endl;
```

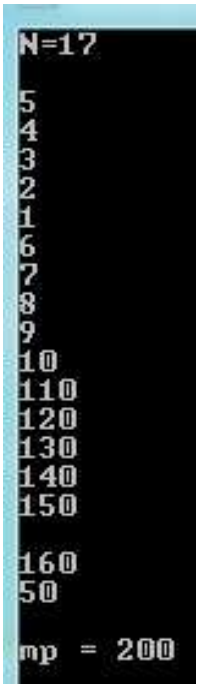
```
    cout << "mp = " << mp;
    getch();                    //задержка экрана
    return 0;                   //показывает успешное завершение программы
}                               //конец функции main
```

4.3. Подготовка тестовых примеров и выполнение

В первой строке задаётся число N – общее количество показаний датчика. Гарантируется, что N > 15. В каждой из следующих N строк задаётся одно *неотрицательное целое* число – очередное показание прибора.

Программа должна вывести одно число – *описанное в условии произведение*, либо -1, если получить такое произведение не удаётся (*табл. 3*) [3].

Таблица №3. Тестирование программного кода

Тестовый пример			*Выполнение*
Входные данные		*Выходные данные*	
17 5 4 3 2 1 6 7 8	9 10 110 120 130 140 150 160 50	200	

§ 5. Решение задачи из курса химии

В химической лаборатории для большого количества органических молекул измеряется количество входящих в молекулу атомов углерода – целое

неотрицательное число, которое будем называть С-индексом молекулы. Исследуемых молекул может быть очень много, но не может быть меньше трёх, С-индексы во всех молекулах различны.

При обработке результатов отбирается так называемое основное множество С-индексов. Это непустое подмножество всевозможных С-индексов (в него могут войти как С-индекс одной молекулы, так и С-индексы всех исследуемых молекул) такое, что сумма значений С-индексов у него чётна и максимальна среди всех возможных непустых подмножеств с чётной суммой. Если таких подмножеств несколько, то из них выбирается то подмножество, которое содержит наименьшее количество элементов.

Задание:

Напишите программу для решения поставленной задачи, которая будет *эффективна*, как по времени, так и по памяти.

Программа считается эффективной по времени, если время работы программы пропорционально количеству входных данных N, т. е. при увеличении N в k раз время работы программы должно увеличиваться не более чем в k раз.

Программа считается эффективной по памяти, если размер памяти, использованной в программе для хранения данных, не зависит от числа N и не превышает 1 килобайта.

Перед программой кратко опишите использованный алгоритм и укажите версию языка программирования.

5.1. Словесное описание эффективного алгоритма

Основное множество состоит из всех значений С-индексов, кроме 0, если он встречается, и кроме минимального нечётного значения, если таких значений нечётное число.

Программа читает все входные данные один раз, не занимая все входные данные в массиве, размер которого равен N. Во время чтения данных запоминается номер 0, если он встретился (по условию все значения различны, поэтому 0 встречается не больше одного раза), подсчитывается количество

нечётных значений и ищется минимальное нечётное значение. После окончания ввода распечатываются все номера, кроме номера 0 и номера минимального нечётного значения, но только в случае, если их количество нечётно [2].

5.2. Создание эффективной по памяти и времени программы на Dev-C++ 4.9.9.2

```
#include<iostream>                //для ввода/вывода данных
#include<conio.h>                 //необходимо для задержки экрана
using namespace std;
int main()                        //начало исполнения программы (функция main)
{
   int N, i, j, k, cnt, min, a;   //объявление переменных
   cout << "N= ";
   cin >> N; cout << endl;        //ввести количество молекул
   min = 1000000001;              //начальное значение переменной min
   //начальные значения переменных k, j, cnt
   k = 0;
   j = 0;
   cnt = 0;
      for (int i = 0; i < N; i++)
      {
        cout << "a= "; cin >> a;          //ввод целого неотрицательного числа
        cout << endl;
        if (a == 0)   //условие проверки введённого числа на нулевое значение
        j = i;
        if ( a % 2 != 0)               //проверка остатка от деления введённого числа
        {
           cnt++;
           if (a < min)                //проверка введённого числа на минимум
           { min = a; k = i;
```

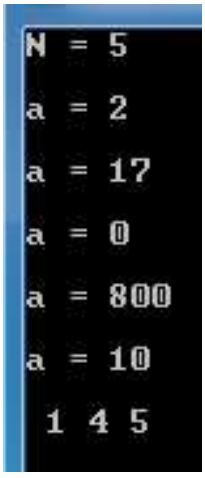
```
            }
          }
        }
            for (int i = 0; i < N; i++)
           //условие отбора основного множества С-индексов
            if ((i != j) && (( cnt % 2 == 0) || (i != k)))
   cout << " " << i + 1;          //вывод номеров молекул по возрастанию
   getch();                       //задержка экрана
   return 0;                      //показывает успешное завершение программы
}                                 //конец функции main
```

5.3. Подготовка тестовых примеров и выполнение

На вход программе в первой строке подаётся количество молекул N. В каждой из последующих строк N записано одно целое неотрицательное число, не превышающее 10^9. Все N чисел различны.

Программа должна вывести в порядке возрастания номера молекул, С-индексы которых принадлежат основному множеству данной серии. Нумерация молекул ведётся с единицы (*табл. 4*) [3].

Таблица №4. Тестирование программного кода

Тестовый пример		*Выполнение*
Входные данные	*Выходные данные*	
5 2 17 0 800 10	1 4 5	

Заключение

Эффективность алгоритма — это свойство алгоритма, которое связано с *вычислительными ресурсами*, используемыми алгоритмом. Алгоритм должен быть *проанализирован* с целью определения *необходимых* алгоритму ресурсов[4].

В данной работе рассмотрены как *аппаратные*, так и *программные* ресурсы.

В практической части даны примеры составления и выполнения *эффективных программ* по времени и памяти на языке программирования DEV-C++ с достаточно *высокой производительностью*.

Список использованных источников и литературы

1. Алгоритм (Эффективность алгоритма) [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Алгоритм
2. ЕГЭ. Информатика и ИКТ: Типовые экзаменационные варианты / С.С. Крылов, Т.Е. Чуркина. – М. : Издательство “Национальное образование ”, 2018. – 208 с. – (ЕГЭ. ФИПИ – школе)
3. И.А. Шаповалова, Е.В. Тишина. Программирование в школьном курсе информатики: учеб. Пособие.-Тверь: Твер. Гос. ун-т, 2013.
4. Эффективность алгоритма [Электронный ресурс]. – Режим доступа: https://ru/wikipedia/org/wiki/Эффективность_алгоритма

Программное обеспечение

1. OS Windows 7
2. Среда программирования Dev-C++