

# Как мы решали задачу 1803 с Тимуса на скорость

## Техники оптимизации

Берсенёв Александр

ENG RUS

# Timus Online Judge



## Online Judge

- [О системе](#)
- [Часто задаваемые вопросы](#)
- [Новости сайта](#)
- [Форум](#)
- [Ссылки](#)

## Задачи

- [Архив задач](#)
- [Отправить на проверку](#)
- [Состояние проверки](#)
- [Руководство](#)

## Авторы

- [Регистрация](#)
- [Исправить данные](#)
- [Рейтинг авторов](#)

## Соревнования

- [Текущее соревнование](#)
- [Расписание](#)
- [Прошедшие соревнования](#)
- [Правила](#)

## Добро пожаловать на Timus Online Judge, архив задач с проверяющей системой

**Timus Online Judge** — это крупнейший в России архив задач по программированию с автоматической проверяющей системой. Основной источник задач для архива — соревнования Уральского федерального университета, Чемпионаты Урала, Уральские четвертьфиналы ICPC, Петрозаводские сборы по программированию. Чтобы начать решать задачи, прочитайте [руководство](#).

**Timus Online Judge** позволяет принять участие в онлайн-версиях большинства соревнований, которые регулярно проходят в Уральском федеральном университете. Перед участием в онлайн-соревнованиях ознакомьтесь с [правилами](#) их проведения.

Сайт создан и поддерживается студентами и выпускниками Уральского федерального университета.

Если вы хотите разместить свои задачи в архиве или провести онлайн-соревнование, напишите по адресу [timus\\_support@acm.timus.ru](mailto:timus_support@acm.timus.ru).

# 1803. Винтовки белочехов

Ограничение времени: 3.0 секунды

Ограничение памяти: 64 МБ

Белочехи решили прекратить ожесточённые бои в Сибири и вернуться домой. Но покинуть Россию было непросто — чтобы добраться на кораблях из Владивостока в Европу, были нужны деньги. Белочехи решили раздобыть необходимую сумму, продав свои винтовки наступающим войскам красных. Всего на продажу они выставили  $n$  винтовок. Первые две винтовки были самыми обычными, и белочехи просили за каждую из них только один рубль. Зато  $i$ -я винтовка ( $i \geq 3$ ) стоила столько же, сколько  $(i-1)$ -я и  $(i-2)$ -я вместе взятые.

В стране имеют хождение только банкноты с номиналами, равными степеням числа  $k$  (банкноты в один рубль,  $k$  рублей,  $k^2$  рублей и т. д.) В распоряжении большевиков уже достаточно типографий, поэтому напечатать нужное количество денег не составило труда. Они заплатили за каждую винтовку минимально возможное количество банкнот, в сумме дающее ровно столько, сколько просили за эту винтовку белочехи.

Когда винтовки попали в руки красных, Чапаев попросил Анку упорядочить их по возрастанию количества банкнот, которое было за них заплачено. Если за две винтовки было заплачено одинаковое количество банкнот, то раньше должна идти винтовка с меньшим номером. Помогите Анке выполнить просьбу Чапаева.

## Исходные данные

В единственной строке через пробел записаны целые числа  $k$  и  $n$  ( $2 \leq k \leq 10$ ;  $3 \leq n \leq 50000$ ).

## Результат

Выведите перестановку чисел от 1 до  $n$  — номера винтовок, упорядоченные по возрастанию количества уплаченных за них банкнот.

## Пример

исходные данные	результат
10 8	1 2 3 4 8 7 5 6

## Замечания

Когда Анка выполнит просьбу Чапаева, стоимости винтовок в рублях будут выглядеть так: {1, 1, 2, 3, 21, 13, 5, 8}.

**Автор задачи:** Виктор Камашев

**Источник задачи:** NEERC 2010, Четвертьфинал Восточного подрегиона

**Метки:** нет ([скрыть метки для нерешенных задач](#))

# 1803. Винтовки белочехов

Ограничение времени: 3.0 секунды

Ограничение памяти: 64 МБ

Белочехи решили прекратить ожесточённые бои в Сибири и вернуться домой. Но покинуть Россию было непросто — чтобы добраться на кораблях из Владивостока в Европу, были нужны деньги. Белочехи решили раздобыть необходимую сумму, продав свои винтовки наступающим войскам красных. Всего на продажу они выставили  $n$  винтовок. Первые две винтовки были самыми обычными, и белочехи просили за каждую из них только один рубль. Зато  $i$ -я винтовка ( $i \geq 3$ ) стоила столько же, сколько  $(i-1)$ -я и  $(i-2)$ -я вместе взятые.

В стране имеют хождение только банкноты с номиналами, равными степеням числа  $k$  (банкноты в один рубль,  $k$  рублей,  $k^2$  рублей и т. д.) В распоряжении большевиков уже достаточно типографий, поэтому напечатать нужное количество денег не составило труда. Они заплатили за каждую винтовку минимально возможное количество банкнот, в сумме дающее ровно столько, сколько просили за эту винтовку белочехи.

Когда винтовки попали в руки красных, Чапаев попросил Анку упорядочить их по возрастанию количества банкнот, которое было за них заплачено. Если за две винтовки было заплачено одинаковое количество банкнот, то раньше должна идти винтовка с меньшим номером. Помогите Анке выполнить просьбу Чапаева.

## Исходные данные

В единственной строке через пробел записаны целые числа  $k$  и  $n$  ( $2 \leq k \leq 10$ ;  $3 \leq n \leq 50000$ ).

## Результат

Выведите перестановку чисел от 1 до  $n$  — номера винтовок, упорядоченные по возрастанию количества уплаченных за них банкнот.

## Пример

исходные данные	результат
10 8	1 2 3 4 8 7 5 6

## Замечания

Когда Анка выполнит просьбу Чапаева, стоимости винтовок в рублях будут выглядеть так: {1, 1, 2, 3, 21, 13, 5, 8}.

**Автор задачи:** Виктор Камашев

**Источник задачи:** NEERC 2010, Четвертьфинал Восточного подрегиона

**Метки:** нет ([скрыть метки для нерешенных задач](#))

- Найти первые  $n$  чисел Фиббоначи
- Перевести их в  $k$ -ричную систему счисления
- Отсортировать по сумме цифр
- Вывести номера чисел Фиббоначи в отсортированной последовательности

# 1803. Винтовки белочехов

Ограничение времени: 3.0 секунды

Ограничение памяти: 64 МБ

Белочехи решили прекратить ожесточённые бои в Сибири и вернуться домой. Но покинуть Россию было непросто — чтобы добраться на кораблях из Владивостока в Европу, были нужны деньги. Белочехи решили раздобыть необходимую сумму, продав свои винтовки наступающим войскам красных. Всего на продажу они выставили  $n$  винтовок. Первые две винтовки были самыми обычными, и белочехи просили за каждую из них только один рубль. Зато  $i$ -я винтовка ( $i \geq 3$ ) стоила столько же, сколько  $(i-1)$ -я и  $(i-2)$ -я вместе взятые.

В стране имеют хождение только банкноты с номиналами, равными степеням числа  $k$  (банкноты в один рубль,  $k$  рублей,  $k^2$  рублей и т. д.) В распоряжении большевиков уже достаточно типографий, поэтому напечатать нужное количество денег не составило труда. Они заплатили за каждую винтовку минимально возможное количество банкнот, в сумме дающее ровно столько, сколько просили за эту винтовку белочехи.

Когда винтовки попали в руки красных, Чапаев попросил Анку упорядочить их по возрастанию количества банкнот, которое было за них заплачено. Если за две винтовки было заплачено одинаковое количество банкнот, то раньше должна идти винтовка с меньшим номером. Помогите Анке выполнить просьбу Чапаева.

## Исходные данные

В единственной строке через пробел записаны целые числа  $k$  и  $n$  ( $2 \leq k \leq 10$ ;  $3 \leq n \leq 50000$ ).

## Результат

Выведите перестановку чисел от 1 до  $n$  — номера винтовок, упорядоченные по возрастанию количества уплаченных за них банкнот.

## Пример

исходные данные	результат
10 8	1 2 3 4 8 7 5 6

## Замечания

Когда Анка выполнит просьбу Чапаева, стоимости винтовок в рублях будут выглядеть так: {1, 1, 2, 3, 21, 13, 5, 8}.

**Автор задачи:** Виктор Камашев

**Источник задачи:** NEERC 2010, Четвертьфинал Восточного подрегиона

**Метки:** нет ([скрыть метки для нерешенных задач](#))

– Найти первые  $n$  чисел Фиббоначи

– Перевести их в  $k$ -ричную систему счисления

– Отсортировать по сумме цифр

– Вывести номера чисел Фиббоначи в отсортированной последовательности

Пример:  $k=10$ ,  $n=8$

1 1 2 3 5 8 13 21  
1(1) 1(2) 2(3) 3(4) 5(5) 8(6) 13(7) 21(8)  
1(1) 1(2) 2(3) 3(4) 21(8) 13(7) 5(5) 8(6)  
1 2 3 4 8 7 5 6

```

1 k, n = map(int, input().split())
2
3 def fib(a):
4     if a < 3:
5         return 1
6     return fib(a-1)+fib(a-2)
7
8
9 def digits(number, base):
10     ans = []
11     while number:
12         ans.append(number % base)
13         number //= base
14     return ans[::-1]
15
16
17 def count(num):
18     return sum(digits(fib(num), k))
19
20
21 print(" ".join(map(str, sorted(range(1, n+1), key=count)))))

```

- Найти первые n чисел Фибоначчи
- Перевести их в k-ричную систему счисления
- Отсортировать по сумме цифр
- Вывести номера чисел Фибоначчи в отсортированной последовательности

---

Пример: k=10, n=8

```

1 1 2 3 5 8 13 21
1(1) 1(2) 2(3) 3(4) 5(5) 8(6) 13(7) 21(8)
1(1) 1(2) 2(3) 3(4) 21(8) 13(7) 5(5) 8(6)
  1   2   3   4       8   7   5   6

```

```

bay ~/tmp/138_timus $ python3 naive.py
10 8
1 2 3 4 8 7 5 6

```

# Оптимизация

## Алгоритмы и структуры данных

- как хранить числа в памяти?
- как сортировать?

## Особенности CPU

- Кэши
- Хитрые инструкции

## Оптимизация

### Алгоритмы и структуры данных

- как хранить числа в памяти?
- как сортировать?

80%

### Особенности CPU

- Кэши
- Хитрые инструкции

20%



## Как хранить числа в памяти?

Хотим:

- быстро складывать числа
- быстро находить сумму цифр числа в  $k$ -ричной системе

Помним: CPU умеют работать с 64-битными числами

## Как хранить числа в памяти?

### Хотим:

- быстро складывать числа
- быстро находить сумму цифр числа в  $k$ -ричной системе

Помним: CPU умеют работать с 64-битными числами

### Наивный подход:

записать число в виде бит, поделить на группы по 64 бита

складывать "столбиком", т.е. складывать правые 64 бита первого числа с правыми 64 битами второго, если результат не влез в 64 бита, запоминаем это (бит переноса), и добавляем единицу при следующем вычислении

## Как хранить числа в памяти?

The diagram illustrates a naive addition algorithm. It shows the addition of 37 and 28. A red '+1' is written above the tens column. The numbers are written as 37 and 28, with a red '+' sign to the left. A horizontal line is drawn under the 28. Below the line, the result 65 is shown. A green arrow points from the 6 to the expression 3+2+1. A red arrow points from the 5 to the text '7+8=15, "5" пишем в единицах, "1" прибавляем к десяткам.'

$$\begin{array}{r} +1 \\ 37 \\ +28 \\ \hline 65 \end{array}$$

3+2+1

7+8=15, "5" пишем в единицах, "1" прибавляем к десяткам.

### Наивный подход:

записать число в виде бит, поделить на группы по 64 бита

складывать "столбиком", т.е. складывать правые 64 бита первого числа с правыми 64 битами второго, если результат не влез в 64 бита, запоминаем это (бит переноса), и добавляем единицу при следующем вычислении

Проблема: перевод результата в k-ричную систему – долго

## Как хранить числа в памяти?

### Наивный подход 2:

Хранить цифры числа сразу в  $k$ -ричной системе, например, одна цифра – один байт

Например: число 81 в десятичной системе хранить как два байта: 8 и 1

– быстрое нахождение суммы цифр

## Как хранить числа в памяти?

### Наивный подход 2:

Хранить цифры числа сразу в  $k$ -ричной системе, например, одна цифра – один байт

Например: число 81 в десятичной системе хранить как два байта: 8 и 1

- быстрое нахождение суммы цифр

- медленное сложение т.к. много цифр. Особенно в двоичной системе

Оба наивных подхода не укладываются в лимит времени на задачу в 3 000 мс

## Решение 1

Хранить цифры числа в  $k^8$ -ричной системе, одна цифра – четыре байта

Например, при  $k=10$ , хранить цифры в системе счисления с основанием 100 млн.

– нахождение суммы цифр медленнее т.к. надо каждую цифру делить на 10 восемь раз

– сложение быстрее т.к. цифр в числах стало в 8 раз меньше

2 750 мс

## Решение 2

Хранить цифры числа в  $k^{16}$ -ричной системе если  $k=2$ ,  
в  $k^{10}$ -ричной если  $k=3$ ,  
в остальных случаях в  $k^8$ -ричной

Двоичные и троичные числа имеют меньшую длину, ещё быстрее сложение,  
но медленнее нахождение суммы цифр

2 156 мс

### Решение 3

Ускоряем нахождение суммы цифр.

Делаем таблицу цифра -> сумма цифр k-ричного представления числа

123 -> 6

1234 -> 10

12345 -> 15

Предподсчитываем цифры до 550 000. Выбираем систему счисления, чтобы цифра не превысила это число, т.е. напр. для  $k=2$  берём 524 288, для  $k=10$  берём 100 000

421 мс



## Решение 4

При сложении чисел сначала складываем цифры попарно. В следующем цикле обрабатываем переполнения.

Первый цикл векторизуется компилятором

Функцию сложения чисел делаем inline (т.е. вставляем инструкцию компилятора, вставить тело функции в место её вызова).

```
inline int add(int* a, int* b, const int k, const int pre_k, int &max_len) {  
    for (int i = 0; i < max_len; ++i) {  
        a[i] += b[i];  
    }  
}
```

390 мс

## Решение 5

Предвычисляем суммы цифр до 1 600 000.

375 мс

## Решение 5

Не предвычисляем суммы цифр заранее, а считаем их в момент первого обращения

359 мс

## Решение 6

Снова предвычисляем суммы цифр. Вычисляем сумму цифр по таблице в два этапа (ks – сколько мы предвычислили)

```
int n = a[i];  
sum_digits += sum_tbl[n % ks] ;  
n /= ks;  
sum_digits += sum_tbl[n];
```

Теперь можно взять ещё больше основание системы счисления и считать сумму чисел ещё быстрее (т.к. в них в два раза меньше цифр). Зато сумма цифр – дольше. Например для k=2 можно взять основание  $2^{30}$ .

# 312 мс

## Решение 7

Предвычисляем суммы не всегда для 100 100 значений, а в зависимости от  $k$ .

265 мс

## Решение 8

Вычисляем заранее сумму цифр последних 2000-4000 чисел Фиббоначи для каждого k.

Эти числа записываем в исходный текст в base64.

Экономим время, в худшем случае считаем не 50 000 чисел, а только 48 000.

Проблема: размер кода ограничен 65КБ.

187 мс

## Решение 9

Говорим компилятору сгенерировать код для каждого k с помощью template

```
template<unsigned int KS>
inline int add(unsigned int* f, const unsigned int k, const unsigned int ks, int &max_len, int iter) {
    for (int i = 0; i < max_len; i+=2) {
        f[i] += f[i+1];
        f[i+1] += f[i];
    }
}
```

За один вызов add – два сложения.

Оптимизация расположения цифр в памяти

```
#define genfor(dk, ks) { \
    if (k == dk) { \
        for (int i = 3; i < end; i+=2) { \
            add<ks>(f, kn, ks, max_len, i); \
        } \
    } \
}
```

```
genfor(2, 65536); genfor(3, 59049);
genfor(4, 65536); genfor(5, 78125);
genfor(6, 46656); genfor(7, 16807);
genfor(8, 32768); genfor(9, 59049);
genfor(10, 100000);
```

# 171 мс

## Решение 10

Предвычисляем таблицу сумм на этапе компиляции с помощью constexpr

Предвычисляем суммы цифр последних чисел Фиббоначи, храня их не в виде base64, а в виде байт

```
if (k==3) {  
    end=46200;  
    unsigned char buf[15204]="0êN†N;0 N©0 NvNÉ0>0.0¹0Q0  
010;0*0[NăNÌNó0 NÚOM0]000ÑN 02NñNİN¶NñN<0 0kNμ0.0e0  
\0 Nÿ0 00K0000NÉ040 0 0^0[NÇ0š0M0Õ0ÂNμN00Ž0 Ny0,0C0  
N×0Ė0`Nâ0m0³0,NõPC0pN±0—0N0 0u0\0" "0[0-0J0]N 0r000  
0 0™0CNö0 0é0&0İ0×08090Ÿ0€NÂ0e0@NÝ0[0 NñNý0¾0'0 0¾0
```

156 мс



```

const int DIVS = 1025;
constexpr inline int find_sum_tbl(int num, int k) {
    int ans = 0;
    while(num) {
        ans += num % k;
        num /= k;
    }
    return ans;
}

constexpr std::array<unsigned char, DIVS+1> crtbl(int k, int ks) {
    std::array<unsigned char, DIVS+1> arr = {};
    for (int i = 0; i < ks; ++i) {
        arr[i] = find_sum_tbl(i, k);
    }
    return arr;
}

constexpr std::array<unsigned char, DIVS+1> sums[11] = {
    crtbl(2, DIVS), crtbl(2, DIVS), crtbl(2, DIVS), crtbl(3, DIVS),
    crtbl(4, DIVS), crtbl(5, DIVS), crtbl(6, DIVS), crtbl(7, DIVS),
    crtbl(8, DIVS), crtbl(9, DIVS), crtbl(10, DIVS),
};

```

## Решение 11

Если смотреть на сумму цифр в числах Фиббоначи, можно заметить что они не сильно отличаются. Поэтому можно кодировать не сами числа, а разницы с предыдущим числом.

При этом кодировать таким методом, чтобы чем меньше разница, тем меньше бит занимало закодированное число (типа кода Хаффмана):

от -128 до 128 – 9 бит  
от -256 до 256 – 10 бит  
от -384 до 384 – 11 бит  
иначе 14 бит

49973	46913
49974	46925
49975	46867
49976	47172
49977	47104
49978	46342
49979	47177
49980	46935
49981	46745
49982	47564
49983	46960
49984	47229
49985	47416
49986	46549
49987	47390
49988	47409
49989	46784
49990	47015
49991	46468
49992	47043
49993	47134
49994	47197
49995	47450
49996	46389
49997	47300
49998	47006
49999	47164
50000	46929

# 140 мс

## Решение 12

Сбалансировано предвычисление сумм, чтобы при разных  $k$  время выполнения было бы одинаковым для  $n=50\,000$

125 мс

## Решение 13

Для вычисления суммы цифр в числе использовано три заглядывания в предподсчитанную таблицу, 64-битная арифметика

```
s += sum_tbl[n % KS];  
n /= KS;  
s += sum_tbl[n % KS];  
n /= KS;  
s += sum_tbl[n];
```

109 мс

## Решение 14

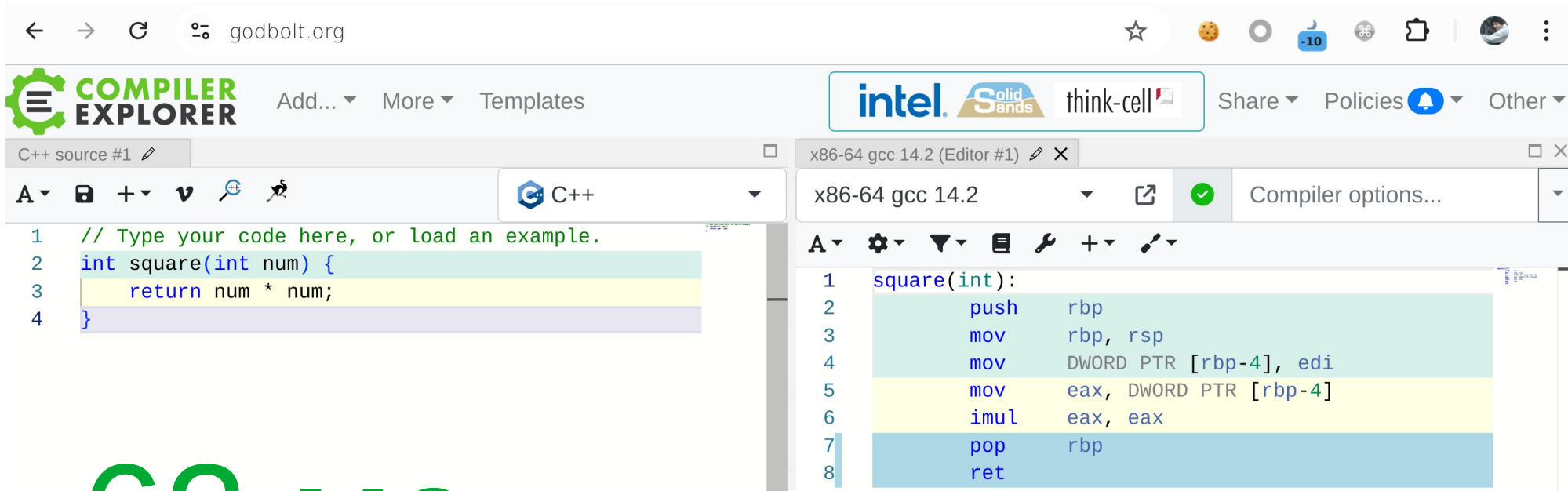
Предвычисляя суммы цифр  $n$ -ного числа Фиббоначи в  $k$ -ричной системе нам не обязательно знать саму сумму. Важен порядок сортировки.

Поэтому можно уменьшать эти суммы так, чтобы порядок сортировки в выводе не менялся. Меньше суммы  $\rightarrow$  меньше бит  $\rightarrow$  больше влезает в 65КБ

93 мс

## Решение 15

Выравнивание структур данных, жонглирование оптимизациями, анализ получившегося ассемблерного кода, случайная перестановка блоков кода. Во все тяжкие. Типа `__attribute__((target("sse4.1")))`. Засылка одного и того же решения несколько раз



The screenshot displays the Godbolt compiler explorer interface. The left pane shows the C++ source code for a function named `square` that takes an integer `num` and returns its square. The right pane shows the generated x86-64 assembly code for the same function, compiled using GCC 14.2. The assembly code includes stack frame setup, pushing the base pointer, moving the stack pointer to `rbp`, storing the argument `edi` at `[rbp-4]`, loading it into `eax`, squaring it with `imul`, popping `rbp`, and returning.

**C++ source #1**

```
1 // Type your code here, or load an example.
2 int square(int num) {
3     return num * num;
4 }
```

**x86-64 gcc 14.2 (Editor #1)**

```
1 square(int):
2     push    rbp
3     mov     rbp, rsp
4     mov     DWORD PTR [rbp-4], edi
5     mov     eax, DWORD PTR [rbp-4]
6     imul    eax, eax
7     pop     rbp
8     ret
```

62 мс



```
using namespace std;int s[50001]={0,1,1};unsigned long long f[1250]={1,1};unsigned char sum_tbl[120001]={0}
inline void add(unsigned long long* f,const unsigned long long k,const unsigned long long ks,int &ml,int i
ml += 4;unsigned long long n;int s1=0,s2=0;for(int i=0;i<ml;i+=2){if(f[i] >= k){f[i] -= k;f[i+2] += 1;if(
+= sum_tbl[n % KS];n/=KS;s1 += sum_tbl[n % KS];n/=KS;s1 += sum_tbl[n];if(f[i+1] >= k){f[i+1] -= k;f[i+1+2
n=f[i+1];s2 += sum_tbl[n % KS];n/=KS;s2 += sum_tbl[n % KS];n/=KS;s2 += sum_tbl[n % KS];n/=KS;s2 += sum_tbl
s[iter]=s1;s[iter+1]=s2;if(f[ml-1]==0 && f[ml-2]==0){ml -= 2;if(f[ml-1]==0 && f[ml-2]==0){ml -= 2;}}}
inline int get_bits(const unsigned char* buf,int &off,int cnt)
{ unsigned int ans=0;for(int i=0;i<cnt;i++){int a=off+i;int c=a/8;int b=7 - (a % 8);unsigned char bit=(buf
inline int get_diff(const unsigned char* buf,int &o){int d=0;if(get_bits(buf,o,1)==0){d=get_bits(buf,o,8)-
=128;else d+=128;}else{if(get_bits(buf,o,1)==0){d=get_bits(buf,o,8)-128;if(d<0)d-=256;else d+=256;}else{d=
int main(){unsigned int k,n;cin>>k>n;const unsigned char *a=0;unsigned long long kn=0,ks=0;if(k==2){kn=0x
else if(k==4){kn=0x4000000000000000;ks=65536;} else if(k==5){kn=0x14adf4b7320334b9;ks=78125;} else if(k==6
{kn=0x3642798750226111;ks=117649;} else if(k==8){kn=0x10000000000000000;ks=32768;} else if(k==9){kn=0x12bf3
unsigned char sum_pretbl[D];for(int i=0;i<D;i+=1){int ans=0;int num=i;while(num){ans += num % k;num /= k;
int div=k;while (div*k<D && div*div<ks)div *= k;for(unsigned int i=0;i<ks;i+=1){int num=i;sum_tbl[i] += s
int ml=2,end=50000;if (k==2) {
end=46400;
unsigned char buf[8358]="4[]00Q4c 00C4]ZQ,00+Fl00H000i0[]%10;40[]0000[]*:[]0E[]>00[]\"400000A0-s0EK000Cw0[
[]001nCZ0H[]Z5V+00[]T0D00G00d000Mu00&e,p0E0[]\0\"\"000-0J0ij[]0'0q00 60i0@B07F[]x0RT[]Ge0.+72i0a[]0N0|,[]b0C[]L0
00Lh0500.c00> []{E}[]M0FN+v0)-+00j0Z0&E0000u0[]0u[]0tNS'0p0%L00000[]kI00U[] []*i0[]e000[]\0\"\"M000J00w0dm000K
0m[]\"0E0[]0y0d00[]Z0j0[]\"0J'r0q00b> []09003V0m(C'0VJD0Ww[]r)0-0.v[]ede,00[]!0:Q0m0hp0000[]00Q+#[J000Gn000RK*03k
[]na[]0f[].[]0Z0)00_D00L0'00[]0'0000[]000[]0[]0w^%PRg20[]0\4[]04C\"\"FI*[]J[]0j0B0[]R0a[]\0\"\"%0oLb000(0000[]01Y0[]0'
[]yKB02,0000[]00F0([]wD0x00LJ0000[]!>005[];!0M0xE0ffq00[]h[]00>L000[]00*+0hI000[]R04Q!083T000Q0/00[]0{F[]Q[]0[]000l
A0[]0nL0T[]00V00000[]00F00B00[]00d[]tD00[]0K[]zh00m0[]00b[]S0[]\0\"\"00[]000[]0a[]00[0d{H0[]0Bc[][[]0(-00Y00B000|A[]00c'
[]?[]0[]V0008[]\nS09000/00^[]0A[]00>00[]0i0 U'?n00rI1000R00w[]00[]00L-0[]y0[]\0$Z00T0B00X0!00[]000000!e>[]00c[]\0\"\"0
00[]\"[]0[]R0000m0000N0+00[]000c []0a0X00[]0o90p0b00T0XJ0]0[]0a0[]0000000[]00kY0[]00r0l<v[]0[]E0[] [] [] []t6J0[]0P00y\
00[]Q0000[]t[]yR00H0EjL0mQT00[]\00R[]T)0A[]00[]00Eh[] []0[]$0)p000i!9[]r0{[]000[]0xG[]00[]:0+,Ii0B'd0[]0+0![]0R5(0[]+000[]\0
0=0000BH0n[]000[]0QN00n0[]00R0000R0%[]0)0E003 []0.0090(6001-0[]00[]0GVPF\"00,0000][]00P0*[]000[]|0a0[]00h0|0Z0
100U0P000[]0[]00g[]\0\"\"00W[]0000*M0[]0VB4[]0[]0#0[]r0|0084[]r[]0t0'.0Cv000Y[]^0u2A[]\0\"\"0l05'0[]0kx0=p000e0h02_00[]te`
0_00005A[]U00[][(0y0A00[]=0b0[]r000)0Xa00&P'K00q0/[]\J00[]0i0;[]0 00ERWq00z$YH0(h0Xf[]0000\n0000'000[]0l0-0[]0
u00[]00G0]BEi0J4u0U0ii[]R0h8[]0[]0dw00Rso0Q0r[]a0XC0cG0P003K0)X0v> []\0Hu[]\?[]n00t0[]\"J00[]u[]\"h[]0000[ ,r0000
00[]900s00[]Q00000km+00vr0000ia[]\Gw000c0[] []00[]e[]0000[]\<W3[]a*%PR0b0[]00[]0)00J0$[]0y[][[]001jv<0\nT000Z000[]0
i[]t00[]00[]0<Zub00[]j0ueC0[]000[]00[]!0IE0#0[]\[]0A0[]h[]000G00G&0j0[]0[]00[]t05q0Xg~8100[]PR[]D0L[]Mf[0g0Ls0s[]00n,\"
[]wA0J006{0 [] ,0x0)iq[]00E0Ly0&0D-A00M2U00*[]0[]?0[]000> H[]0[]000[]0JG0(M0U000[]00NM0f00L0e[]\"00000l0[]`0[]> 30
000[]0:Z0rMz[]\00n[]0@V72L00$ U&00$00G0)r0R0YbU0L000000[]0q[]]|H:N[]r09l00[00,l0G$[]0^qR0200[] 00007Ans00IV[]l
```

# Миниконкурс

← → ↺ 🏠 alexbers.com/minikonkurs.txt



## Миниконкурс

Предыстория: перед Новым годом решал задачу 1803 (<https://acm.timus.ru/problem.aspx?space=1&num=1803>) на Тимусе. Просто решить её мне показалось мало и я старался решить её быстрее всех. Постепенно, оптимизируя код, решение стало проходить за 62 мс, а решение на втором месте проходит за 234 мс.

"Но можно ли лучше?" - подумал я и решил предложить задачу вам.

Чтобы было интереснее придумал такие призы:

- первому кто уделает моё решение по скорости в рейтинге (<https://acm.timus.ru/rating.aspx?space=1&num=1803>) и покажет решение переведу 10 тыс руб на СБП
- при наступлении 14 января в полночь открою таблицу рейтинга и авторы 3 самых быстрых решений (я не в счёт), присланных в 2025 году получат 7.5 тыс, 5 тыс и 2.5 тыс руб.

Что можно: любой язык поддерживаемый платформой, любые предварительные вычисления, адаптация к тестам (тесты там такие - два числа на вход, первое число 2-10, второе можно считать что всегда максимальное т.к. с ним программа работает дольше всего), адаптация к конкретной модели процессора

Что нельзя: пользоваться административным доступом к Тимусу, если он есть (например, менять времена в базе, тесты к задаче, максимальный размер принимаемого файла с решением и т.д), мешать другим (например, создавать избыточную нагрузку на сайт), использовать уязвимости в изоляции системы тестирования (например, читать данные оставшиеся от прошлых попыток сдачи), создавать себе несколько аккаунтов



## Рейтинг решений задачи Винтовки белочехов

[Все языки](#) (491) | [C/C++](#) (441) | [Pascal](#) (6) | [Java](#) (29) | [C#](#) (14) | [Go](#) (2) | [Python](#) (4) | [Ruby](#) (0) | [Haskell](#) (0) | [Scala](#) (0) | [Kotlin](#) (0) | [Rust](#) (7)

Место	ID	Дата	Автор	Язык	Время работы	Выделено памяти
1	10863213	10:41:11 6 янв 2025	<a href="#">Endagorion</a>	Clang++ 17 x64	0.046	1 168 КБ
2	10858858	05:01:19 30 дек 2024	<a href="#">Bersenev Alexander</a>	G++ 13.2 x64	0.062	820 КБ
3	10861619	16:35:13 3 янв 2025	<a href="#">sw1ft</a>	Rust 1.75 x64	0.093	1 932 КБ
4	10861068	02:43:56 3 янв 2025	<a href="#">Kirill`~</a>	G++ 13.2 x64	0.109	1 116 КБ
5	10861878	18:15:42 3 янв 2025	<a href="#">ArtemOrnysh</a>	G++ 13.2 x64	0.109	1 524 КБ



Легендарный гроссмейстер

**Endagorion**

Михаил Тихомиров, [Нидерланды](#), [Амстердам](#)  
Из организации [pinely](#)



Рейтинг: **3003** (макс. **легендарный гроссмейстер**, 3313)



Вклад: **+36**



В друзьях: у 4078 пользователей

Последнее посещение: 41 час назад

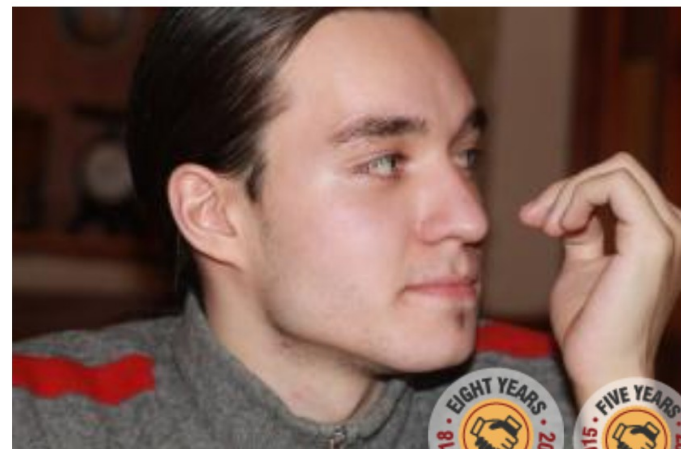
Зарегистрирован: 15 лет назад



[Записи в блоге \(46\)](#), [комментарии](#)



[Переписка](#) | [Послать сообщение](#)



## Решение Endagorion

Следующее число Фиббоначи = сумма двух предыдущих

Сумма цифр  $X+Y$  = Сумма цифр  $X$  + Сумма цифр  $Y$  –  $(k-1)$ \*количество переполнений

например:

$$\begin{array}{ll} 10 + 20 = 30 & 12 + 34 = 46 \\ 1+2 = 3 & 3 + 7 = 10 \end{array}$$

$$\begin{array}{ll} 19 + 10 = 29 & 19 + 11 = 30 \\ 10 + 1 = 11 & 10 + 2 - 1*(10-1) = 3 \end{array}$$

$$\begin{array}{l} 999 + 1 = 1000 \\ 27 + 1 - 3*(10-1) = 1 \end{array}$$

## Решение Endagorion

Представление чисел

Каждая цифра кодируется каким-то количеством бит:

для  $k=2$ : 1 бит

для  $k=3,4$ : 2 бита

для  $k=5,6,7,8$ : 3 бита

для  $k=9,10$ : 4 бита

Каждые 64 бита могут кодировать 15-63 цифр.

При сложении чисел считаем количество переполнений.

## Решение Endagorion

### Представление чисел

Каждая цифра кодируется каким-то количеством бит:

для k=2: 1 бит

для k=3,4: 2 бита

для k=5,6,7,8: 3 бита

для k=9,10: 4 бита

Каждые 64 бита могут кодировать 15-63 цифр.

При сложении чисел считаем количество переполнений.

Пример для k=2. Складываем 8 и 13:

01000

01101

-----

10101

$$1 + 3 - 1 \cdot (2 - 1) = 3$$

## Решение Endagorion

Как считать количество переполнений:

Пример для  $k=2$ . Складываем 8 и 13:


```
01000
01101
-----
10101
```

$X \wedge Y = 00101$

$X + Y = 10101$

$(X \wedge Y) \wedge (X + Y) = 10000$

считаем единичные биты  
инструкция popcount



## Решение Endagorion

Как считать количество переполнений:

Пример для k=4. Складываем 8 и 13:

```
01000
01101
-----
10101
```

$( (X \wedge Y) \wedge (X + Y) ) \& 10101$

## Решение Endagorion

Как считать количество переполнений:

Пример для  $k=8$ . Складываем 8 и 13:

01000	10
01101	15
-----	
10101	25

$( (X \wedge Y) \wedge (X + Y) ) \& 01001$



## Решение Endagorion

Как считать количество переполнений:

Пример для  $k=3$ . Складываем 8 и 13

$8 = 3 \cdot 2 + 2$ , т.е. 22 в троичной системе

$13 = 9 \cdot 1 + 3 \cdot 1 + 1$ , т.е. 111 в троичной системе

$21 = 9 \cdot 2 + 3 \cdot 1$ , т.е. 210 в троичной системе

001010	22
010101	111
-----	
011111 (???)	210

## Решение Endagorion

Как считать количество переполнений:

Пример для  $k=3$ . Складываем 8 и 13

$8 = 3 \cdot 2 + 2$ , т.е. 22 в троичной системе

$13 = 9 \cdot 1 + 3 \cdot 1 + 1$ , т.е. 111 в троичной системе

$21 = 9 \cdot 2 + 3 \cdot 1$ , т.е. 210 в троичной системе

001010	22
010101	111
-----	
011111 (???)	210

Надо добавить к X (или к Y) такое число 010101:  $001010 + 010101 = 011111$

011111	22 (+111)
010101	111
-----	
110100	210

Если цифра переполнилась – результат правильный, если не переполнилась – надо вычесть единицу обратно.

# Фишки решения Endagorion

## Сортировка

```
vector<int> idx;  
idx.resize(n);  
  
for (unsigned int i = 0; i < n; ++i) {  
    idx[i] = i + 1;  
}  
  
sort(idx.begin(), idx.end(), [&](int a, int b) {  
    if (s[a] != s[b])  
        return s[a] < s[b];  
    return a < b;  
});
```

```
u64 M = 49000, off[50000], ans[N];  
for(int i=0, i<n; i++) ++off[ds[i]];  
for(int i=0, i<M; i++) off[i + 1] += off[i];  
for(int i=0, i<n; i++) ans[off[ds[i] - 1]++] = i + 1;
```

## Фишки решения Endagorion

Вывод на экран

```
for (unsigned int i = 0; i < n; ++i) {  
    if (i) cout << ' ';  
    cout << idx[i];  
}  
cout << endl;
```

```
void print_int(int val) {  
    char chars[6];  
    int digits = 0;  
    while (val) {  
        auto v = val / 10;  
        chars[digits++] = (val - v * 10) + '0';  
        val = v;  
    }  
    while (digits) _putchar_nolock(chars[--digits]);  
}  
  
void print_ans(int n) {  
    for(int i=0; i<n; i++) {  
        print_int(ans[i]);  
        _putchar_nolock(" \n"[i + 1 == n]);  
    }  
    _fflush_nolock(stdout);  
}
```

## Фишки решения Endagorion

Формула для длины числа Фиббоначи

```
1 log2 = 0.693147180559
2 log5 = 0.8047189562170;
3 logphi = 0.481211825059603;
4
5
6 def l(n):
7     return int(((n * logphi - log5)) / log2) + 1
```

Косячит не больше чем на 1 бит, для первого 1 млн чисел (я проверил)

46 мс

## Добавляем предпосчёт сумм цифр

Пользуясь формулой длины находим количество цифр в N-ом числе Фиббоначи.

Ожидаем что при сложении предыдущих двух чисел переполнение происходит ровно в половине случаев.

Кодируем разницу фактического числа переполнений и ожидаемого числа

Новый алгоритм кодирования:

числа от -32 до 32 — кодируются 7 битами

от -64 до 64 — 8 битами

итд, до 15 бит

Сумму цифр одного числа Фиббоначи удалось закодировать **1 байтом**.

31 мс

## Проект 15

Идея: вычислить на этапе компиляции суммы цифр чисел Фиббоначи

Проблема: вычисления на этапе компиляции **тормозные**.

Время компиляции ограничено – 10 секунд

## Проект 15

Идея: вычислить на этапе компиляции суммы цифр чисел Фиббоначи

Проблема: вычисления на этапе компиляции **тормозные**.

Время компиляции ограничено – 10 секунд

Решение: использование типов `unsigned _BitInt(N)`, которые недавно завезли в `clang`.

Например: `unsigned _BitInt(10000)` – число из 10 000 бит

Таким образом можно написать `a + b` и два больших числа сложатся на этапе компиляции, причём сложение написано **быстро**, на Си



## Проект 15

Идея: вычислить на этапе компиляции суммы цифр чисел Фиббоначи

Проблема: вычисления на этапе компиляции **тормозные**.

Время компиляции ограничено – 10 секунд

Решение: использование типов `unsigned _BitInt(N)`, которые недавно завезли в clang.

Например: `unsigned _BitInt(10000)` – число из 10 000 бит

Таким образом можно написать `a + b` и два больших числа сложатся на этапе компиляции, причём сложение написано **быстро**, на Си

Проблемы подхода: нет инструментария для профилирования. Например, оказалось что умножение такого числа на что-то **супер-медленное**.

## Проект 15

Ещё проблемы подхода: если сразу использовать `unsigned _BitInt(50000)`, операции будут медленными. Надо постепенно увеличивать ширину. С каким шагом?

```
const int NN = 2048;

unsigned _BitInt(NN) X0 = 1, Y0 = 1;
int i = 2;

#define D(N) unsigned _BitInt(N) X##N, Y##N;
#define L(S, E) loop(len_inverse<k>((S-1)/w), X##S, Y##S, S, X##E, Y##E)

D(4096);
D(8192);
D(12288);
D(16384);
D(20480);
D(24576);
D(28672);
D(32768);
D(36864);

loop(len_inverse<k>(2047/w), X0, Y0, 2048, X4096, Y4096);
L(4096, 8192);
L(8192, 12288);
L(12288, 16384);
L(16384, 20480);
L(20480, 24576);
L(24576, 28672);
L(28672, 32768);
loop(pre, X32768, Y32768, 32768, X32768, Y32768);
```

## Проект 15

В Clang 17, который есть на Тимусе, нет функции подсчёта числа единиц в `_BitInt`. Её завезли в Clang 19. Приходится писать примерно такую магию (не вчитывайтесь):

```
c12_s = c12_s - ((c12_s >> 1) & m1)
c12_s = (c12_s & m2) + ((c12_s >> 2) & m2)
c12_s = (c12_s + (c12_s >> 4)) & m4;
c12_s = (c12_s + (c12_s >> 8)) & m8;
c12_s = (c12_s + (c12_s >> 16)) & m16;
cc = c12_s | (c12_s >> par16)
cc += cc >> 32;
cc += cc >> 64;
cc += cc >> 128;
cc += cc >> 256;
if constexpr (par > 512) { cc += cc >> 512; }
if constexpr (par > 1024) { cc += cc >> 1024; }
if constexpr (par > 2048) { cc += cc >> 2048; }
if constexpr (par > 4096) { cc += cc >> 4096; }
if constexpr (par > 8192) { cc += cc >> 8192; }
if constexpr (par > 16384) { cc += cc >> 16384; }
```

## Проект 15

Мы можем предпосчитывать 10 секунд. Мы можем засунуть в исходник значения суммы для 50 000 чисел.

Проблема: что лучше предпосчитать на этапе компиляции, а что засунуть в виде готовых сумм. Задача оптимизации.

Условия: для всех k должно считаться одинаково по времени и как можно быстрее

Усложнение: т.к. вычисление числа единичных бит — дорогая операция, быстрее пропустить первые N итераций, т.е. просто посчитать сами числа, но не считать сумму цифр в них (если они маленькие). Считать её снова на этапе выполнения

Решение: многомерный поиск, на каждом шаге можно засечь время компиляции и время выполнения

```
constexpr int SKIPS[] = {0, 0, 16000, 18000, 14750, 18000, 12250, 10000, 13000, 10500, 7500};  
constexpr int PRES[] = {0, 0, 26000, 32700, 29900, 30750, 33750, 34500, 29900, 23000, 22550};
```

## Рейтинг решений задачи Винтовки белочехов

[Все языки](#) (494) | [C/C++](#) (444) | [Pascal](#) (6) | [Java](#) (29) | [C#](#) (14) | [Go](#) (2) | [Python](#) (4) | [Ruby](#) (0) | [Haskell](#) (0) | [Scala](#) (0) | [Kotlin](#) (0) | [Rust](#) (8)

Место	ID	Дата	Автор	Язык	Время работы	Выделено памяти
1	10874695	20:56:46 26 янв 2025	<a href="#">Bersenev Alexander</a>	Clang++ 17 x64	0.015	1 744 КБ
2	10863213	10:41:11 6 янв 2025	<a href="#">Endagorion</a>	Clang++ 17 x64	0.046	1 168 КБ
3	10861619	16:35:13 3 янв 2025	<a href="#">sw1ft</a>	Rust 1.75 x64	0.093	1 932 КБ
4	10861068	02:43:56 3 янв 2025	<a href="#">Kirill`~</a>	G++ 13.2 x64	0.109	1 116 КБ

15 мс

## Можно ли меньше?

В Clang 19 завезли `__builtin_popcountg`

```
bay ~/tmp/124_tmp $ time clang++-19 -static -std=c++23 -O3 full_compiletime.cpp -o full_compiletime
clang++-19 -static -std=c++23 -O3 full_compiletime.cpp -o full_compiletime  4.05s user 0.05s system 99% cpu 4.098 total
bay ~/tmp/124_tmp $ echo '5 50000' | time ./full_compiletime > /dev/null
./full_compiletime > /dev/null  0.00s user 0.00s system 90% cpu 0.002 total
bay ~/tmp/124_tmp $ █
```

Спасибо