

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Департамент программной инженерии

Микропроект №1 по Архитектуре вычислительных систем.

Пояснительная записка

Исполнено студентом БПИ197

Пике Кирилл

30 октября 2020 года.

Оглавление

1. Задача	3
2. Решение.....	4
2.1 Решето Эратостена^[1].....	5
2.2 Битмапы^[2].	6
2.3 Описание алгоритма решения задачи.	7
2.4 Возможные альтернативы.....	8
3. Проверка работы программы.....	9
4. Ссылки	10
5. Исходный код	11

1. Задача

Разработать программу, вычисляющую число не простых (составных) чисел в диапазоне от 1 до бинарного машинного слова без знака.

2. Решение

Для выполнения задания в первую очередь хотелось применить перебор всех чисел с проверкой на простоту. Однако даже с оптимизациями, такими как игнорирование чётных чисел и проход до квадратного корня числа при проверке числа на простоту, работа программы занимала слишком много времени.

В ходе изучения майнора «Математические структуры» и темы «Битмап» в курсе «Алгоритмы и структуры данных», я узнал об одной из техник для вычисления простых чисел путём применения Решета Эратостена и о структуре данных для применения решета, битмапах. Таким образом я решил выполнять свои вычисления используя данную технику с использованием битмапов.

2.1 Решето Эратостфена^[1].

Решето Эратостфена это алгоритм для нахождения простых чисел, которое отфильтровывает все составные числа. Работает следующим образом: берется первое простое число и вычеркиваются все остальные числа, которые на него делятся. То есть первым число взяв 2, мы сначала вычеркнем все чётные числа. Затем идёт 3, и мы вычёркиваем все числа, которые делятся на 3. Таким образом на каждой итерации, каждое следующее незачеркнутое число является простым.

Дабы уменьшить количество чисел, с которыми нужно работать, я убрал все четные числа из алгоритма.

2.2 Битмапы^[2].

Битмап или как его еще можно назвать битовый массив — это массив из битов, то есть 1 или 0.

Для удобства работы с данной структурой, я разбил числа на куски по 1 байту или по 8 бит. Таким образом один такой кусок мог хранить информацию о простоте 8 чисел.

2.3 Описание алгоритма решения задачи.

Алгоритм разделён на две части. В первой части мы проходим по нечётным числам и фильтруем все числа до 2^{32} . Во второй части мы проходим уже по «отрешетированному» списку чисел и просто отмечаем простые числа. Битмап, отвечающий за изучаемые числа хранит 1 в битах, отвечающих за составные числа и 0 для чисел, которые являются простыми. Сами индексы отвечают не за само число, а за нечетное число с началом в 3. То есть число = индекс * 2 + 1.

Первая часть. Из курса дискретной математики известно, что если у числа и есть простые делители, то хотя бы один из них будет меньше или равен квадратному корню самого числа. Таким образом, чтобы отфильтровать весь список чисел нам достаточно пройтись от 3 до 2^{16} , фильтруя весь список чисел на каждый шаг, когда изучаемое число является простым.

На каждый шаг этой части алгоритма выполняются следующие шаги:

- Проверка значения в битмапе для числа под вопросом.
- Если число простое, то идёт заполнение решета всеми его множителями.
- Если число не является простым, то переходим к следующему элементу.

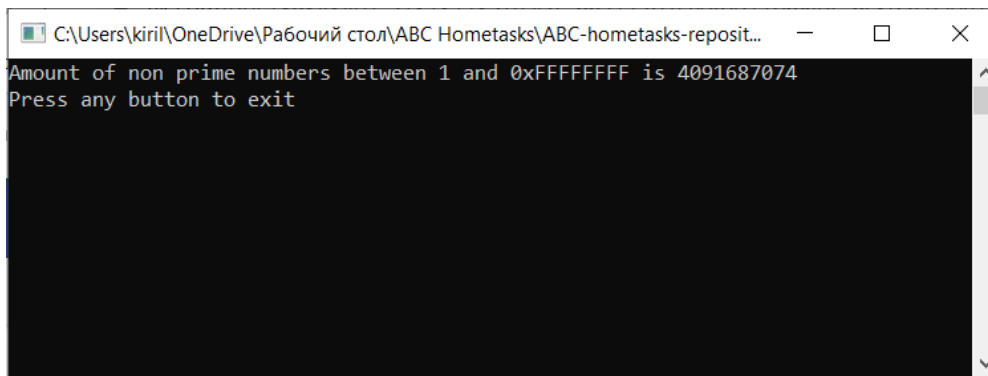
Вторая часть. Мы просто проходим по оставшимся числам, проверяя значение в битмапе для каждого из них и отбираем простые числа и увеличиваем значение счётчика.

Так как задача была посчитать не простые числа, а составные, то в конце мы просто вычитаем из количества изучаемых чисел простые.

2.4 Возможные альтернативы.

В качестве альтернативы, можно было воспользоваться функцией подсчета количества простых чисел разработанной, называемой методом Мейсела, Лемера, Лагариаса, Миллера, Одлизко^[3] или одной из вариаций этого метода, например доработка данного алгоритма Ксавье Гурдоном.

3. Проверка работы программы.



```
C:\Users\kiril\OneDrive\Рабочий стол\ABC Hometasks\ABC-hometasks-reposit...
Amount of non prime numbers between 1 and 0xFFFFFFFF is 4091687074
Press any button to exit
```

Если произвести проверочные подсчёты, пользуясь калькуляторами с интернета⁴, то получается, что всего в изучаемом списке чисел от 1 до 4294967295 203280221 простых чисел, что подтверждает результат подсчёта

4. Ссылки

1. Решето Эратостфена. https://ru.wikipedia.org/wiki/Решето_Эратосфена
2. Битмап. https://en.wikipedia.org/wiki/Bit_array
3. COMPUTING $\pi(x)$: THE MEISSEL, LEHMER, LAGARIAS, MILLER, ODLYZKO METHOD. <https://www.ams.org/journals/mcom/1996-65-213/S0025-5718-96-00674-6/S0025-5718-96-00674-6.pdf>
4. Prime Counting Function. <https://www.dcode.fr/prime-number-pi-count>

5. Исходный код

```
format PE console

entry start

include 'win32a.inc'

section '.data' data readable writable

    ;ВАЖНО: так как мы выделяем очень много памяти под решето, в компиляторе нужно
    ;позволить это выделение
    ;для этого заходим в Options -> Compiler Setup и увеличиваем память до максимума
    ;ПРИМЕЧАНИЕ: программа может занимать до 2 МИНУТ для выполнения

    primes dd 1 ;сразу учитываем 2ку, которая не будет входить в алгоритм
    endMessage db 10,'Press any button to exit',0 ;сообщение в конце прграммы
    digit db '%d',0
    answer db ' Amount of non prime numbers between 1 and 0xFFFFFFFF is ',0 ;строка
    ;для вывода числового значения ответа в консоль
    A rt 0xFFFFFFFF ;выделяем необходимую для реализации память
    B rt 0xFFFFFFFF
    C rt 0xFFFFFFFF
    D rd 0xFFFFFFFF

section '.code' code readable writable executable

start:
    ;1) Проходим по числам от 3 до 65536 для нахождения одного из простых множителей
    ;каждого числа из списка
    ;и составляем решето для нечётных чисел.
    call count_primes_h1
    ;2) Проходим по остатку чисел, которые уже отфильтрованы решето. Если число
    ;простое,
    ;то бит индекса равен 0 и мы увеличиваем наш счётчик простых чисел
    call count_primes_h2

    ;3) Получаем количество всех составных чисел от 1 до 0xFFFFFFFF не учитывая 0
    call get_final_result

    ;4)Выводим результат в консоль
    call display_answer

    ;5) Завершаем программу
    call finish_program
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Принимает в регистре eax число для вывода в консоль
;Выводит его по циферке
proc display_answer

    ;выводим сообщение об ответе
    push eax
    push answer
    call [printf]
    add esp, 4
    pop eax

    ;очищаем регистр ecx для счётчика цифр
    xor ecx, ecx

    ;отдаляемся от адреса в стеке, чтобы его не трогать
```

```

sub esp, 4

;делим на 10, остаток пушим в стек, пока число не станет 0
display_push_loop:
xor edx,edx
push ebx
mov ebx,10
div ebx
pop ebx
push edx
inc ecx
cmp eax,0
jne display_push_loop

;выводим по очереди из стека с самого верха цифры в консоль
mov ebx, ecx
display_console:
push digit
call [printf]
add esp, 8
dec ebx
cmp ebx, 0
jne display_console

;возвращаем адрес выхода из процедуры на верх стека
add esp, 4
ret
endp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;работает с ячейкой памяти, в которой хранится количество простых чисел
;возвращает количество составных чисел от 1 до 0xFFFFFFFF
proc get_final_result

    xor eax, eax ;обнуляем регистр
    not eax      ;берём его обратное значение, то есть максимальное значение машинного
слова из 32 бит
    sub eax, [primes] ;вычитаем простые числа
    ret
endp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;:
:::

;принимает на вход регистр ecx в котором хранится индекс числа в битмапе.
;заполняет решето эратосфена до конца изучаемых чисел.
;ничего не возвращает, но меняет значения в битмапе.
proc fill_compound

    ;сохраняем значение регистра ecx
push ecx

    ;разница индексов между числами, которое делит текущее 2*индекс+1, где индекс-
текущий.
mov eax,2
mul ecx
add eax,1

    ;оставляем это значение разницы индексов в стеке, чтобы обращаться к нему по ходу
выполнения программы.
push eax

fill_compound_loop:

```

```

        ;помещаем значение разницы индексов в регистр eax и возвращаем указатель стека
обратно к нему же
        pop eax
        sub esp,4
        ;перемещаемся в индексу следующего числа.
        add ecx, eax

        ;сохраняем его в стеке
        push ecx

        ;берём первые 3 бита из регистра счётчика, так как это числа 0-7
        ;очень удобно для подсчёта битов в байте.
        and cl, 0x7

        ;двигаем 1 на нужную позиция влево, которая соответствует индексу текущего числа
        ;по модулю 8
        mov eax, 1
        shl eax, cl

        ;берём индекс текущего числа и возвращаем указатель на него же.
        pop ecx
        sub esp, 4

        ;выясняем индекс байта в котором хранится бит текущего числа.
        shr ecx, 3

        ;помещаем туда 1 побитовым сложением.
        or byte[A+ecx], al

        ;восстанавливаем индекс текущего числа.
        pop ecx

        ;проверка на окончание цикла
        cmp ecx, 0x7FFFFFFF
        jb fill_compound_loop

        ;конец цикла - возвращаем указатель стека на значение основной программы
        ;чтобы восстановить изначальное значение ecx до входа в процедуру
        end_fill_compound:
        add esp,4
        pop ecx
        ret
        endp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;

;работает с регистрами ecx - хранит индекс нечётных чисел в битмапе
;eax - помогает определить простое число или составное см процедуру check_prime
;[primes] - содержит в себе количество простых чисел от 3 до 0x8000
proc count_primes_h1
    ;инициализируем счётчик индексов
    mov ecx, 1

    ;основной цикл для вычисления простых чисел и заполнения решета эратостена
    count_primes_loop:

    ;проверка текущего числа на простоту ссылаясь на значение его индекса в битмапе
    call check_prime
    cmp eax, 0
    jne count_primes_skip
    jmp count_primes_fill

    ;делаем проверку на окончание цикла
    count_primes_end_check:
    cmp ecx, 0x8000

```

```

        jb count_primes_loop
        jmp end_count_primes

;заполняет решето эратостена числами, кратными текущему простому числу
count_primes_fill:
call fill_compound
;увеличиваем количество простых чисел и индекс числа
inc [primes]
inc ecx
jmp count_primes_end_check

;переходим к следующему числу, если нынешнее не является простым числом
count_primes_skip:
inc ecx
jmp count_primes_end_check

;окончание процедуры. Регистр ecx сохраняет индекс с которого начинается следующий
цикл.
        end_count_primes:
        ret
        endp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;работает с регистрами ecx - хранит индекс нечетных чисел в битмапе
;eax - помогает для определения простого числа или составного см процедуру check_prime
;[primes] ячейка памяти, которая хранит количество простых чисел.
proc count_primes_h2

        ;основной цикл процедуры, где мы проходим по индексам битмапа и проверяем
        ;являются числа простыми и тогда заносим в счетчик или составными.
count_primes_h2_loop:
;проверка на простоту. Если eax = 0, то простое
call check_prime
cmp eax, 0
je inc_prime

;проверка на окончание цикла. Если счётчик в ecx достиг конца, то завершает
исполнение процедуры.
;в ином случае увеличивает значение ecx на 1 продолжает работу.
count_primes_h2_end_check:
inc ecx
cmp ecx, 0x7FFFFFFF
jb count_primes_h2_loop
jmp end_count_primes_h2

;инкрементируем количество простых чисел, встреченных в битмапе
inc_prime:
inc [primes]
jmp count_primes_h2_end_check

;окончание работы процедуры
end_count_primes_h2:
ret
endp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;работает с текущим значением ecx - индекс в битмапе текущих нечётных чисел.
;смотрит значение бита соответствующего числу, если значение бита 1, то число составное,
если 0 то простое.
;устанавливает в индекс eax значение 0 если, число оказалось простым, иное если нет.
proc check_prime

;у нас есть байт, где записана информация о 8 числах. Мы берём 1 и двигаем его на нужное
количество

```

;разрядов влево и перемножаем с известным байтом побитово. Если в искомом бите была 1, то результат умножения
;отличен от 0 и число составное, простое иначе.

```
mov eax, 1
;создаём изначальное значение регистра ecx
push ecx

;отфильтровываем последние три бита ecx, чтобы взять счётчик ecx по модулю 8
;берём именно этот регистр, потому что только он может делать побитовые сдвиги
and cl,0x7

;двигаем единицу влево
shl eax, cl

;восстанавливаем прежнее значение ecx и возвращаем его обратно в стек
pop ecx
push ecx

;вычисляем индекс числа в массиве байтов, содержащих биты
shr ecx, 3

;в младшем байте eax хранится нужное значение для перемножения
;теперь в eax хранится результат побитового произведения.
and al, byte[A+ecx]

;восстанавливаем значение ecx
pop ecx

ret
endp
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
```

;Оканчивает работу программы
proc finish_program

```
push endMessage
call [printf]
call [getch]
push 0
call [ExitProcess]
ret
endp
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
```

section '.idata' import data readable

```
library kernel, 'kernel32.dll',\
    msvcrt, 'msvcrt.dll'

import kernel,\
    ExitProcess, 'ExitProcess'

import msvcrt,\
    printf, 'printf',\
    getch, '_getch'
```