

Алгоритмы и структуры данных

Лекция 17

Числа.

Сергей Леонидович Бабичев

Дополнительный код

- Мы будем использовать наборы из фиксированного количества бит — $n = 8, 16, 32, 64$.
- Мы можем трактовать эти наборы как числа без знака. Тогда диапазон представления чисел будет $[0 \dots 2^n)$.
- Мы можем трактовать эти набора как числа со знаком. Тогда диапазон представления будет $[-2^{n-1} \dots 2^{n-1} - 1)$.

Клеточные автоматы

- Существует класс задач, называемый *клеточные автоматы*.
- Имеется *состояние* автомата, которое однозначно определяет следующее состояние.
- Все переходы состояний происходят одновременно.
- Переход между состояниями называется *совершением хода*.

Простой клеточный автомат

- Имеется строка из нулей и единиц, называемых *клетками*.
- Если в клетке находится единица, то она называется *живой*.
- Переход состояния клетки из нуля в единицу называется *рождением*.
- Переход состояния из единицы в ноль называется *смертью*.
- Каждая клетка, кроме самой левой и самой правой, имеет двух *соседей*.
- У самой левой клетки и самой правой клетки имеются фиктивные мёртвые соседи.

Простой клеточный автомат

- Правила перехода такие:
 - 1 Живая клетка, у которой оба соседа живых, умирает от перенаселения.
 - 2 Мёртвая клетка, у которой оба соседа живых воскресает.
 - 3 Живая клетка, у которой оба соседа мёртвых, умирает от одиночества.
 - 4 Мёртвая клетка, у которой оба соседа мёртвых, воскресает.
 - 5 В остальных случаях клетка состояние не изменяет.

Простой клеточный автомат: пример

- Пусть начальное состояние было таким:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|

Добавим фантомные клетки слева и справа:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

Простой клеточный автомат: пример

- Пусть начальное состояние было таким:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|

Добавим фантомные клетки слева и справа:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

Сделаем первый ход.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

Простой клеточный автомат: пример

- Пусть начальное состояние было таким:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|

Добавим фантомные клетки слева и справа:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

Сделаем первый ход.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

После второго хода:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

Где применяются клеточные автоматы

- Моделирования поведения толпы при выходе со стадиона.
- Моделирование решёточных газов.
- Моделирование транспортных потоков (Модель Нагеля-Шрекенберга)
https://keldysh.ru/council/3/D00202403/chechina_aa_diss.pdf
- Моделирование популяций клеток и вирусов.
- ...

Зачем применять уравнения логики в клеточных автоматах?

- Для приемлемой точности нужны большие популяции клеток.
- Нужно много шагов моделирования.
- Последовательное моделирование медленное.
- Моделируют на многих вычислительных потоках и графических картах.
- Требуется создать модель, параллельно вычисляющую все новые состояния.

Как моделировать нашу задачу?

- Составим таблицу нового состояния клетки в зависимости от текущего состояния клетки и её соседей.
- Пусть клетки расположены так: ABC и мы хотим получить новое состояние клетки B' .

| A | B | C | B' |
|-----|-----|-----|------|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Решение задачи

| A | B | C | B' |
|-----|-----|-----|------|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

- Верхняя половина — $A = 0$. Справа — $B \equiv C$ или $\overline{B \oplus C}$.
- Нижняя половина — $A = 1$. Справа — $B \oplus C$.
- Итого: $B' = (\overline{A} \wedge \overline{B \oplus C}) \vee (A \wedge B \oplus C)$
- Упрощая: $B' = (\overline{A} \vee B \oplus C) \vee (A \wedge B \oplus C)$

Битовые операции в дополнительном коде.

Битовые операции в дополнительном коде

- Современная архитектура ЭВМ не любит операций перехода.
- Операции сравнения с присвоением $x = a > 0$; обычно достаточно быстры. Они быстрее операций перехода, но медленнее побитовых операций.
- Некоторые несложные функции можно реализовать на исключительно побитовых операциях.
- Как вычислить абсолютное значение знакового числа x ?

Битовые операции в дополнительном коде

- Пусть имеются операции `bitand`, `bitor`, `bitor`, `bitnot`, `bitshl` и `bitshr`. Тогда в дополнительном коде:
 - $-x = \text{bitnot } x + 1$
 - $-x = \text{bitnot}(x - 1)$
- Можно выделить знак числа знаковым сдвигом вправо $x : s = x \text{ bitshr } 31$. Это будет а) число из N единиц, `bitxor` с которым эквивалентен операции `bitnot` или б) N нулей, операция `bitxor` с которым не изменит число.
- Тогда $\text{abs}(x) = s \text{ bitxor}(x + s)$.

Подсчёт количества бит в числе

Постановка задачи

- Имеется число x , представленное n битами.
- Требуется найти количество единичных битов.
- Имеется набор операций:
 - ▶ поразрядного или \vee , `bitor`;
 - ▶ поразрядного и \wedge , `bitand`;
 - ▶ поразрядного исключающего или \oplus , `bitxor`;
 - ▶ поразрядного отрицания (инверсии) *not*, `bitnot`;
 - ▶ поразрядного сдвига *shift left*, `bitshl` и *shift right*, `bitshr`.
- Результаты этих операций — новый набор из n битов, представляющий новое число.
- Операция истинности расширена на наборы как $b_0 \vee b_1 \vee \dots \vee b_{n-1}$.
- Любое отличное от 0 число трактуется как истина.

Простое решение: просто считаем биты числа.

```
1: function COUNT1(x)
2:   count  $\leftarrow$  0
3:   for all bit  $\in$  [0..n) do
4:     if x bitand(1 bitshl bit) then
5:       count  $\leftarrow$  count + 1
6:     end if
7:   end for
8:   return  $\leftarrow$  count
9: end function
```

Количество операций пропорционально n .

Альтернативный вариант

```
1: function COUNT2( $x$ )
2:    $count \leftarrow 0$ 
3:   for all  $bit \in [0..n)$  do
4:      $count \leftarrow count + ((x \text{ bitshr } bit) \text{ bitand } 1)$ 
5:   end for
6:    $return \leftarrow count$ 
7: end function
```

Количество операций пропорционально n .

Ещё один альтернативный вариант

```
1: function COUNT3( $x$ )
2:    $count \leftarrow 0$ 
3:   for all  $bit \in [0..n)$  do
4:      $count \leftarrow count + (x \text{ bitand } 1)$ 
5:      $x \leftarrow x \text{ bitshr } 1$ 
6:   end for
7:    $return \leftarrow count$ 
8: end function
```

Количество операций пропорционально n .

Улучшения

- Как уменьшить количество операций?
- Использовать только единичные биты.
- Поэкспериментируем с операциями сложения и вычитания на 1 (инкремента и декремента).

Эксперимент с операцией инкремента

| | | | | | | | | |
|---------|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| $x + 1$ | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |

| | | | | | | | | |
|---------|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| $x + 1$ | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

| | | | | | | | | |
|---------|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $x + 1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Эксперимент с операцией инкремента

Попробуем проделать операции `bitor` над парами.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|--------------------------|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| $x + 1$ | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| $x \text{ bitor}(x + 1)$ | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|--------------------------|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| $x + 1$ | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| $x \text{ bitor}(x + 1)$ | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|--------------------------|---|---|---|---|---|---|---|---|
| x | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $x + 1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $x \text{ bitor}(x + 1)$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Эксперимент с операцией инкремента

- Операция `bitor` числа с его инкрементом устанавливает самый правый из нулевых битов в 1.

Эксперимент с операцией декремента

| | | | | | | | | |
|---------|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| $x - 1$ | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |

| | | | | | | | | |
|---------|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| $x - 1$ | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

| | | | | | | | | |
|---------|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $x - 1$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Эксперимент с операцией декремента

Попробуем проделать операции `bitand` над парами.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---------------------------|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| $x - 1$ | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| $x \text{ bitand}(x - 1)$ | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---------------------------|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| $x - 1$ | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| $x \text{ bitand}(x - 1)$ | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---------------------------|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $x - 1$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $x \text{ bitand}(x - 1)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Эксперимент с операцией декремента

- Операция `bitand` числа с его декрементом устанавливает самый правый единичный бит в 0.
- Как нам это поможет ускорить алгоритм в среднем?
- Будем уничтожать по одному биту до тех пор, пока число не обнулится.

Улучшенный вариант алгоритма

```
1: function COUNT4( $x$ )
2:    $count \leftarrow 0$ 
3:   while  $x \neq 0$  do
4:      $count \leftarrow count + 1$ 
5:      $x \leftarrow x \text{ bitand}(x - 1)$ 
6:   end while
7:    $return \leftarrow count$ 
8: end function
```

Количество операций пропорционально числу ненулевых битов в x .

Улучшенный вариант алгоритма

```
1: function COUNT4( $x$ )
2:    $count \leftarrow 0$ 
3:   while  $x \neq 0$  do
4:      $count \leftarrow count + 1$ 
5:      $x \leftarrow x \text{ bitand}(x - 1)$ 
6:   end while
7:    $return \leftarrow count$ 
8: end function
```

Количество операций пропорционально числу ненулевых битов в x .

Победа?

Попытка параллельности

- Мы исходим из предположения, что существуют элементарные операции, обрабатывающие всё множество одновременно.
- Вначале мы работали с единичными битами.
- Затем мы стали работать с группой бит.
- Попробуем работать со всеми битами сразу.
- Можно рассматривать 8 бит как одно дизъюнктивное множество в 8 бит.
- Можно рассматривать 8 бит как два дизъюнктивных множества по 4 бита.
- Можно рассматривать 8 бит как четыре дизъюнктивных множества по 2 бита.

- Как извлечь числовые представления этих множеств?

Извлечение числовых подмножеств: пример

- Рассмотрим число $x = 00101100$
- Чему равно число, представленное множеством бит $[2..4]$?
- Создадим *маску*, содержащую единицы в нужных позициях и нули в остальных.
- Проведём конъюнкцию x с маской.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|--------------------------|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| $mask$ | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| $x \text{ bitand } mask$ | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

- Последний этап — сдвинуть результат вправо на два бита.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|--|---|---|---|---|---|---|---|---|
| $(x \text{ bitand } mask) \text{ bitshr } 2$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Извлечение числовых подмножеств

```
1: function EXTRACT(x, start, end)  
2:   length  $\leftarrow$  end - start + 1  
3:   mask  $\leftarrow$  (1 bitshl length) - 1  
4:   return  $\leftarrow$  (x bitshr start) bitand mask  
5: end function
```

▷ obtain length right ones

Считаем единичные биты: продолжение

- Рассмотрим x как 8 дизъюнктивных множеств (кортежей) длиной 1 бит.
- Задача: просуммировать представления этих множеств.
- Разобьём на чётные (красные) и нечётные (зелёные) элементы.
- Требуется сложить попарно соседние значения из красных и зелёных представлений.
- Результат сложения однозначных чисел не превосходит 10_2 .
- Для хранения результата достаточно двух разрядов.

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |

Считаем единичные биты: продолжение

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |

- Как получить четыре набора из красных значений?
- Наложив маску, удаляющую зелёные значения.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| $mask_u$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| $u \leftarrow x \text{ bitand } mask_u$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

- Мы получили четыре дизъюнктивных множества по два элемента.
- Они представляют четыре двухбитных числа — 0, 0, 1, 0.

Считаем единичные биты: продолжение

- Как теперь получить такие же четыре числа, порождённые зелёными элементами?
- Наложив маску, удаляющую красные значения.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| $mask_v$ | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| $x \text{ bitand } mask_v$ | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| $v \leftarrow (x \text{ bitand } mask_v) \text{ bitshr } 1$ | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

- Осталось сдвинуть результат вправо на 1 бит.
- Мы опять получили четыре дизъюнктивных множества по два элемента.
- Они представляют четыре двухбитных числа — 0, 1, 1, 0.

Считаем единичные биты: продолжение

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| $u \leftarrow x \text{ bitand } mask_v$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $v \leftarrow (x \text{ bitand } mask_v) \text{ bitshr } 1$ | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

- В четырёх 2-х разрядных кортежах содержатся счётчики количества бит для зелёных и красных половинок отдельно.
- Теперь осталось сложить вектора (четвёрки). Простой операцией +! Переполнения результатов сложения не произойдёт, так как они не превосходят двух и поместятся в двухразрядные числа.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----------------------|---|---|---|---|---|---|---|---|
| $x \leftarrow u + v$ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

Считаем единичные биты: продолжение

Повторим операцию, разбив вектор на чётные и нечётные элементы и сложив уже пары двухразрядных чисел. Опять чётные пары выделим красным цветом, нечётные — зелёным.

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

Снова раскинем их по векторам — на сей раз это будут два 4-битных вектора и сложим:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| x | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| $mask_u$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $u \leftarrow x \text{ bitand } mask_u$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| $mask_v$ | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| $v \leftarrow (x \text{ bitand } mask_v) \text{ bitshr } 2$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $x \leftarrow u + v$ | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

Считаем единичные биты: продолжение

Последний этап: разбиение на два 4-х битных элемента и их сложение.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| $mask_u$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $u \leftarrow x \text{ bitand } mask_u$ | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| $mask_v$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| $v \leftarrow (x \text{ bitand } mask_v) \text{ bitshr } 4$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $x \leftarrow u + v$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Результат: x содержит 3 единичных бита.

Считаем единичные биты: продолжение

- Последний штрих: заметим, что $mask_v$ получается из $mask_u$ сдвигом на то же количество разрядов, что и сдвиг x для получения v .
- Можно считать v как $(x \text{ bitshr } l) \text{ bitand } mask_u$, где l — размер кортежа.

Считаем единичные биты: алгоритм

```
1: function COUNT5( $x$ )
2:    $x \leftarrow (x \text{ bitand } 01010101_2) + ((x \text{ bitshr } 1) \text{ bitand } 01010101_2)$ 
3:    $x \leftarrow (x \text{ bitand } 00110011_2) + ((x \text{ bitshr } 2) \text{ bitand } 00110011_2)$ 
4:    $x \leftarrow (x \text{ bitand } 00001111_2) + ((x \text{ bitshr } 4) \text{ bitand } 00001111_2)$ 
5:   return  $\leftarrow x$ 
6: end function
```


Простые числа и их нахождение.

Решето Эратосфена

Один из древнейших известных алгоритмов.

1. Возьмём битовый массив b размером n , инициализированный нулями, кроме 0 и 1-го элементов.
2. Введём понятие: *текущее простое число*, которое изначально равно 2.
3. Находим с начала массива первый нулевой бит. Его позиция и есть текущее простое число p .
4. Если $p \times p \geq n$ алгоритм закончен.
5. Для всех чисел от n^2 с шагом p устанавливаем в единицу все разряды массива b .
6. Возвращаемся к п. 3.

Сложность алгоритма $O(n \log \log n)$.

Решето Аткинса

Основан на свойствах простых чисел.

- Если $n \equiv 1 \pmod{4}$ и n не кратно квадрату простого число, то n простое тогда и только тогда, когда число корней уравнения $4x^2 + y^2 = n$ нечётно.
- Если $n \equiv 1 \pmod{6}$ и n не кратно квадрату простого число, то n простое тогда и только тогда, когда число корней уравнения $3x^2 + y^2 = n$ нечётно.
- Если $n \equiv 11 \pmod{12}$ и n не кратно квадрату простого число, то n простое тогда и только тогда, когда число корней уравнения $3x^2 - y^2 = n$ нечётно.

Сам алгоритм Аткинса:

1. Возьмём битовый массив b размером n , инициализированный нулями.
2. Для каждой пары (x, y) , $x < \sqrt{n}$, $y < \sqrt{n}$ бит $b[t]$ меняется на противоположный для всех $t = 4x^2 + y^2, t = 3x^2 + y^2, t = 3x^2 - y^2$.
3. Для всех индексов i с ненулевыми $b[i]$ обнуляем $b[i * i]$, если $i * i < n$.
4. Оставшиеся индексы ненулевых элементов есть простые числа.

Сложность алгоритма $O(n)$.