Московский авиационный институт
(государственный технический университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа №3

по спецкурсу «Криптография»:
Факторизация

Выполнил:	Карпова В.А.
Группа:	08-306
№ по списку:	9
Преподаватель:	Рисенберг Д.В.
Оценка:	
Дата:	

Москва
2012 г.

Задание

Необходимо написать программу на языке C++, C# или Python, реализующую данный алгоритм факторизации в соответствии с вариантом. Должны поддерживаться числа длиннее 64 бит (длинная арифметика). Разрешается использование сторонних реализаций длинной арифметики и генераторов случайных чисел.
Вариант №2: метод р-Полларда.

Метод решения.

Факторизация предполагает полное разложение числа на множители. Однако, в данном задании, мы ограничиваемся разложением на два простых множителя: n=p*q.
Перед тем, как непосредственно использовать р-метод Полларда, необходимо получить дополнительную информацию о числе (чтобы ускорить вычисления и отбросить тривиальные случаи).

- Проверяем, является ли число составным. Если число простое, то в качестве разложения выводим 1 и n (где n -- простое число).
- Проверяем, является ли число четным. Если четное, то сразу возвращаем в качестве разложения 2 и n/2.
- Иначе выполняем метод р-Полларда.

Алгоритм р-метод Полларда.

1. Выбираем отображение $f: Z/nZ \to Z/nZ$

Обычно f -- многочлен степени большей или равной 2. В качестве стандартного примера берется $f(x) = x^2 + 1$. На практике удобнее использовать $f(x) = x^2 + const$, где const -- некоторая константа заданного Z/nZ.

2. Случайно выбираем $x_0 \in Z/nZ$ и вычисляем члены рекурентной последовательности: $x_0, x_1, \ldots$ по правилу:
$$x_i \equiv f(x_{i-1}) mod\ n$$
3. Сформировав последовательность, будем проверять условие:
$$1 < GCD(x_j - x_k, n) < n$$

до тех пор, пока не будет найден делитель числа *n*, или пока не закончится время работы алгоритма.
Один из ключевых моментов -- выбор таких *j* и *k*. Если для каждого *j* перебирать *k<j,* то это займет слишком много времени.
Поэтому рассматривают пары $1 < GCD(x_{2k} - x_k) < n$.

Существует также аналог этого алгоритма, который повторяет описание в точности до метода формирования последовательности.
Благодаря следующему алгоритму последовательность не хранится в памяти в виде массива (списка), а формируется на ходу.

1. Выбирается многочлен *f(x)* с целочисленными коэффициентами, степени не выше 2.
Обычно берется многочлен вида $f(x) = x^2 + c (mod\ n)$
2. Случайно выбирается $x_0 = y_0$ меньше *n*.
3. Вычисляются значения $x_i = f(x_{i-1}) (mod\ n)$, $y_i = f(f(y_{i-1})) mod\ n$
4. Находится $d = GCD(|x_i - y_i|, n)$.
5. Если d = 1, происходит переход на шаг 3, если d = n, происходит остановка - факторизацию провести не удалось. Если 1 < d < n, то найдено разложение числа n.

Оценка сложности.

р-метод Полларда имеет эвристическую оценку сложности O($n^{\frac{1}{4}}$) арифметических операций. Обычно используется для отделения небольших простых делителей факторизуемого числа n.

Для получения оценки сложности воспользуемся следующими теоремами:

1) Пусть S — фиксированное множество из r элементов, f — какое-либо отображение $f\,:\,S \to S$, $x_0 \in S$, последовательность $x_0, x_1, x_2, \ldots$ определяется соотношением $x_j = f\,(x_{j-1})$. Пусть $\lambda > 0, l = 1 + \left[\sqrt{2\lambda r}\right] < r$. Тогда для тех пар (f, x_0)(где f пробегает, все отображения из S в S и x_0 пробегает все множество S), у которых $x_0, x_1, x_2, \ldots x_l$ попарно различны будет

$$r(r-1)\ldots(r-l)\cdot r^{r-l}.$$

Доля таких пар составляет величину

$$t = r^{-r-1}\cdot r^{r-l+1}\prod_{j=1}^{l}(r-j) = \prod_{j=1}^{l}\left(1-\frac{j}{r}\right).$$

Поскольку при $0 < x < 1$ выполнено неравенство $log(1-x) < -x$, то

$$logt = \sum_{j=1}^{l}\log\left(1-\frac{j}{r}\right) < -\sum_{j=1}^{l}\frac{j}{r} = -\frac{l(l+1)}{2r} < -\frac{l^2}{2r} < -\frac{2\lambda r}{2r} = -\lambda$$

■

Т.е. мы доказали, что если у n есть небольшой простой делитель p, то величина $l(n) = 1 + \left[\sqrt{2\lambda n}\right]$ имеет порядок $n^{\frac{1}{2}}$, в то время как величина $l(p) = 1 + \left[\sqrt{2\lambda p}\right]$ существенно меньше. Доля наборов $\left(f, x_0(mod\ n)\right)$,где $f\!: Z/nZ \to Z/nZ$, у которых элементы длинного набора $x_0(mod\ n), \ldots, x_{l(n)}(mod\ n)$ различны, не превосходят той же величины $e^{-\lambda}$, что и доля наборов $\left(f, x_0(mod\ p)\right)$, где $f\!: Z/pZ \to Z/pZ$, у которых различны элементы короткого набора $x_0(mod\ p), \ldots, x_{l(p)}(mod\ p)$. Следовательно, получаем следующую теорему.

2) Пусть n – нечетное составное число, p – простой делитель n, $p < \sqrt{n}$, $f(x) \in Z[x], x_0 \in Z,$ причем f хорошо редуцируется к модулю n, т.е. f корректно определяет отображение:
$f\!: Z/nZ \to Z/nZ.$
Предположим также, что f(x) хорошо редуцируется к модулю p. Если пара $\left(f, x_0(mod\ p)\right)$ является не слишком редкой по своим статическим свойствам, то p-метод Полларда найдет p за $O(n^{\frac{1}{4}}\log^3 n)$ битовых операций. Т.е., существует константа c, такая, что для любого $\lambda > 0$ вероятность не найти нетривиальный делитель n за $c\cdot\sqrt{\lambda}n^{\frac{1}{4}}\cdot\log^3 n$ битовых операций будем меньше, чем $e^{-\lambda}$.

■

Все методы получения оценки сложности данного алгоритма являются эвристическими и иного метода получения оценки для метода p-Полларда нет.

<u>Реализация</u>.

Факторизация выполняется в следующей функции:

```
def fact(n):
    if isprime(n): return n
    if n%2==0: return 2
    return pollard(n)
```

Для проверки числа на простоту использовался метод Рабина-Миллера, суть которого состоит в следующем:
Алгоритм Миллера — Рабина параметризуется количеством раундов *r*. Рекомендуется брать *r* порядка величины $\log_2(m)$, где *m* — проверяемое число.
Для данного *m* находятся такие целое число *s* и целое нечётное число *t*, что $m-1 = 2^s t$. Выбирается случайное число *a*,$1 < a < m$. Если *a* не является свидетелем простоты числа *m*, то выдается ответ *«m составное»*, и алгоритм завершается. Иначе, выбирается новое случайное число *a* и процедура проверки повторяется. После нахождения *r* свидетелей простоты, выдается ответ *«m, вероятно, простое»*, и алгоритм завершается.

Из теоремы Рабина следует, что если *r* случайно выбранных чисел оказались свидетелями простоты числа m, то вероятность того, что m составное, не превосходит 4^{-r}.

```
def isprime(p): return miller_rabin(p)
def miller_rabin(n, r=100):
      if n==2: return True

      t = n-1
      s = 0
      while t % 2 == 0:
            t //= 2
            s += 1

      for i in range(r):
            a = randint(2,n-1)
            x = pow(a,t,n)
            if x==1 or x==n-1: continue
            for k in range(1,s):
                  x = pow(x,2,n)
                  if x==1: return False
                  if x==n-1: break
            if x==n-1: continue
            return False
      return True
```

Задаем функцию из кольца вычетов в виде $x^2 + const$

```
def f(x,n): return ( x*x+randint(2,n) )%n
```

Находим НОД в соотвествии с алгоритмом Евклида

```
def gcd(a,b): return a if b==0 else gcd(b,a%b)
```

Р-Метод Полларда возвращает единственный делитель числа n. Для получения 2го делителя, выполняем тривиальную операцию n/d.
Для ограничения времени выполнения, в случае невозможности факторизовать число, взяли k == 2**15

```
def pollard(n, k = 2**15):
      """Pollard factorization"""
      x = y = randint(2,n) #not circle
      for i in range(k):
            x = f(x,n) формирование последовательности x
            y = f(y,n)
            y = f(y,n) формирование последовательности y, которая
удовлетволяет условию: y[i] = x[2*i]
            d = gcd(abs(x-y),n)
            if 1 < d < n:
                  return d
            elif d == n:
                  return None
      return None
```

<u>Выводы.</u>

Задача факторизации, сама по себе, является вычислительно сложной. По сложности алгоритмы факторизации разбиваются на две группы: экспоненциальные и субэкспоненциальные. Вопрос о существовании алгоритма факторизации с полиномиальной сложностью до сих пор остается открытым. В то же время факторизация с полиномиальной сложностью возможна на квантовом компьютере с помощью алгоритма Шора.