Задание 1

Алгоритмы и оценка сложности

Литература:

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.
   Алгоритмы. Построение и анализ.
   2-е изд. М.: Вильямс, 2005. Главы 1-4.

1 Предисловие

Основным учебником по нашему курсу будет книга *CLRS* (Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford). Все ссылки я буду давать на неё. Её стоит взять в библиотеке или скачать с lib.mipt.ru. К каждому семинару я буду говорить какие главы по ней стоит прочесть. Составляя задание я буду считать, что вы приступили к его исполнению, прочитав соответствующие главы. Поэтому, в отличие от прошлого семестра, я практически не буду писать теории, а просто буду акцентировать внимание на некоторых моментах.

2 Анализ алгоритмов

Одна из важных задач нашего курса – оценивать время работы алгоритма. При выполнении этого действия необходимо помнить с какой моделью вычислений мы работаем – например, на двухленточной машине Тьюринга копирование[1] любого слова можно осуществить за $2n$ шагов, что несправедливо для одноленточной МТ.

В ближайшее время (скорее всего до начала изучения классов сложности) мы будем предполагать, что имеем дело с RAM-моделью. Говоря нестрого, можно считать, что мы пишем программы, например, на языке C и примитивные операции, например присвоения, а, в зависимости от контекста, и сложение с умножением стоят константу времени. Например, если нас интересует быстрое умножение чисел, то нельзя за константу принимать сложение и умножение этих чисел. Пока нас будет интересовать только оценка времени работы алгоритма.

[1] На вход подаётся слово x, на выходе должно быть написано слово xx.

Пример 1. На вход подаётся число n, необходимо найти все простые числа до n включительно. Приведён псевдокод решения этой задачи:

```
for i := 2 to n do
|  Prime[i] := True → c1
end
for i := 2 to ⌊√n⌋ do
|  if Prime[i] == True → c2 then
|  |  j := 0 → c3
|  |  while i*i + i*j ⩽ n → c4 do
|  |  |  Prime[i*i+i*j] = False → c5
|  |  |  j = j + 1 → c6
|  |  end
|  end
end
```

После выполнения этого кода Prime[i] будет истинно тогда и только тогда, когда число i простое. Подробно прочитать об этом алгоритме можно, например, в Википедии.

Псевдокод выписан для оценки сложности работы алгоритма. Обозначение $\to c_i$ означает, что данная операция стоит константу c_i. Сейчас мы приведём верхнюю оценку сложности. Для этого суммируем вклад каждой учтённой нами операции в сложность алгоритма:

$$T(n) \leqslant c_1 n + \sum_{i=2}^{i=\lfloor\sqrt{n}\rfloor} (c_2 + c_3 + \lfloor n/i - i \rfloor (c_4 + c_5 + c_6)))$$

Делая эту оценку, мы считали, что кажды раз попадаем внутрь оператора **if**. Как мы видим, часть констант в формуле можно заменить одной, так как сумма констант – константа.

$$T(n) \leqslant c_1 n + (\lfloor\sqrt{n}\rfloor - 1)d_1 + \sum_{i=2}^{i=\lfloor\sqrt{n}\rfloor} \lfloor n/i - i \rfloor d_2$$

Но поскольку при асимптотических оценках, константны нам вообще не особо интересны, то мы можем взять среди всех этих констант максимальную и вынести её за скобки.

$$T(n) \leqslant cn + c(\lfloor\sqrt{n}\rfloor - 1) + c \sum_{i=2}^{i=\lfloor\sqrt{n}\rfloor} \lfloor n/i - i \rfloor$$

$$T(n) \leqslant cn + c(\lfloor\sqrt{n}\rfloor - 1) + c(\lfloor\sqrt{n}\rfloor - 1)\lfloor(\lceil\sqrt{n}\rceil - \lfloor\sqrt{n}\rfloor + n/2 - 2)\rfloor$$

Переходя к асимптотическим оценкам мы можем избавиться от членов cn и $c(\lfloor\sqrt{n}\rfloor - 1)$ и получим, что $T(n) = O(n\lfloor\sqrt{n}\rfloor)$.

Для получения этой верхней оценки можно было затратить куда меньше сил, а именно оценить последнюю сумму как $n\lfloor\sqrt{n}\rfloor$, поскольку всего в цикле алгоритм делает $\lfloor\sqrt{n}\rfloor$ шагов и внутри второго цикла он делает не более, чем n шагов. Поэтому, любые вложенные циклы можно оценивать сверху как произведение числа их операций, а для более улучшенного результата нужна более тонкая оценка.

Почему при оценки числа шагов некоторые строчки мы не учитывали, например **for** i := 2 **to** n, а некоторые например **while** i*i+i*j $\leqslant$ n учитывали? Потому что изменение аргумента на цикле **for** для нас естественно и незначимо – можно считать, что мы его загнали во все константы внутри **for**, в то же время, проврерка условия i*i+i*j $\leqslant$ n требует нескольких операций, но каждую из них мы считаем константой и поэтому оцениваем проверку выполнения этого условия константой. Легко заметить, что все строчки на одном уровне цикла мы могли оценить в одну константу – нас фактически интересует только уровень вложения операции. Так, не учитывая, например, проверку условия в **while**, мы бы получили тот же результат.

Хороша ли эта оценка? На самом деле нет. Предложение о том, что каждый раз на операторе **if** мы попадаем внутрь цикла сильно завысило нашу оценку. Из теоретико-числовых соображений следует оценка $O(n \log\log n)$ в модели RAM.

3 Алгоритм Карацубы

Этот алгоритм интересен тем, что с его помощью можно перемножить два числа быстрее чем за квадратичное время, которое даёт обычный алгоритм, которому учат в школе. Это простой и изящный алгоритм, который опроверг «гипотезу n^2» о том, что числа не могут быть перемножены быстрее, чем за квадратичное время. Сложность этого алгоритма мы оценим на ближайшем семинаре, а пока опишем сам алгоритм.

Пусть есть два числа x и y. Запись $x = x_1x_0$, $y = y_1y_0$ означает, что $x = x_1 \times 10^m + x_0$, $x_0 \leqslant 10^m$. то есть, число x в десятичной записи разбивается на два числа x_1 и x_0, причём их конкатенация даёт x. Умножение в рамках данной задачи мы бужем обозначать символом $\times$. Заметьте, что

алгоритм не зависит от системы счисления, поэтому вместо основания 10, как например в задании, может быть основание 2.

Алгоритм основан на следующем факте:

$$\begin{aligned} x \times y &= (x_1 \times 10^m + x_0)(y_1 \times 10^m + y_0) = \\ &= x_1 \times y_1 \times 10^{2m} + x_0 \times y_0 + (x_1 \times y_0 + y_1 \times x_0) \times 10^m = \\ &= z_2 \times 10^{2m} + z_0 + z_1 \times 10^m \end{aligned}$$

причём $z_1 = (x_1 \times y_0 + y_1 \times x_0) = (x_1 + x_0) \times (y_1 + y_0) - z_2 - z_0$.

Таким образом, для вычисления произведения $x \times y$ требуется не 4 умножения, а 3: по одному для вычисления z_2 и z_0, а потом ещё одно для вычисления z_1. Алгоритм применяется рекурсивно, то есть вспомогательные произведения z_i тоже вычисляются по данному алгоритму.

4 Домашнее задание

Задачи из канонического задания на первую неделю (**1-5**).

Задача 1. Найдите лучшую оценку на $T(n)$. Обратите внимание, что просто брать $f(n)$ вместо, скажем, $O(f(n))$ и использовав основную теорему о рекурсии писать результат в качестве ответа не корректно!

1. $T(n) = 2T(\frac{n}{3}) + \Theta(n^2)$
2. $T(n) = 4T(\frac{n}{3}) + \Omega(n)$
3. $T(n) = 2T(\frac{n}{3}) + \mathrm{O}(n)$

Задача 2. Приведите алгоритм разложение числа на простые множители и оцените его сложность. Используйте как вспомогательный алгоритм (возможно модифицировнное) решето Эратосфена.

Задача 3*. Докажите оценку $O(n \log\log n)$ в модели RAM для решета Эратосфена. Найти указанную теоретико-числовую оценку можно в Википедии.