Лекция 8 от 4.02.2016

Быстрое возведение в степень

Пусть $x = \overline{x_1 \dots x_n}$, $y = \overline{y_1 \dots y_n}$. Можно ли за полиномиальное время возвести число x в степень y?

Тупо умножать x на себя y совершенно неоптимально — несложно показать, что сложность алгоритма будет $O(2^n)$ (где n — число цифр в числе). При этом само число x^y содержит $10^n n$ цифр. Получается, что один только размер результата экспоненциален, то есть полиномиальной сложности не хватит даже на вывод результата.

А если по модулю (т.е. результатом будет $x^y \pmod p$ для некоторого указанного p)? Прямое умножение всё равно достаточно медленно. Можно ли быстрее? Оказывается, что да.

Algorithm 1 Быстрое возведение в степень

```
function POWER(x, y, p)                    ▷ алгоритм считает x^y (mod p)
    if y = 0 then
        return 1
    t := POWER(x, ⌊y/2⌋, p)
    if y ≡ 0 (mod 2) then
        return t^2
    else
        return xt^2
```

Легко понять, что глубина рекурсии для данного алгоритма равна $O(\log y) = O(n)$.

Покажем, как он работает на примере $x = 4, y = 33$:

$$x^{33} = x(x^{16})^2 = x((x^8)^2)^2 = x(((x^4)^2)^2)^2 = x((((x^2)^2)^2)^2)^2$$

. Как видно, для возведения числа в 33-ю степень достаточно 7 умножений.

Обратная задача

Пусть нам известны числа x, z, p, каждое по n цифр. Можно ли за полиномиальное время найти число y такое, что $x^y = z \pmod p$.

Сказать сложно — с одной стороны, такой алгоритм ещё не смогли придумать. С другой стороны, не могут доказать того, что его нет. Это всё, вообще говоря, висит на известной проблеме $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ и подробнее мы об этом поговорим ближе к концу курса.

Обработка текста

Предположим, у нас есть n слов, и эти слова мы хотим разместить на странице (порядок, разумеется, не меняя — это же, в конце концов, текст). При этом, шрифт моноширинный, а ширина строки ограничена. Что мы хотим — разместить текст так, чтобы он был выровнен по обоим краям. При этом хотелось бы, чтобы пробелы были примерно одинаковы по ширине.

Введём такую ??? (меру? хз): $\varepsilon(i,j) = L - \sum\limits_{t=i}^{j} |w_t| - (j-i)$ — число дополнительных пробелов в строке с i-го по j-ое слово.

Также введём $c(i,j)$ — стоимость размещения.

$$c(i,j) = \begin{cases} +\infty, \varepsilon(i,j) < 0 \\ \left(\frac{\varepsilon(i,j)}{j-i}\right)^3, \varepsilon(i,j) \geqslant 0 \end{cases}$$

И как это решать? Можно попробовать жадным алгоритмом — просто "впихивать" слова в строку, пока влезают. Он тут не работает, так как он вообще не учитывает стоимость.

Попробуем наш извечный "разделяй и властвуй". Базовый случай — слова помещаются в одну строку, а если не помещаются — переносим и повторяем. Но тут тоже не учитывается стоимость, так что вряд ли будет сильно лучше.

Вход: $w_1, \ldots, w_n; c(i, j)$.

Выход: $j_0, \ldots, j_{l+1}$, такие что $j_0 = 1$, $j_{l+1} = n$, $\sum c(j_i, j_{i+1})$ минимальна.

Сколько всего таких наборов? Мест, где в принципе может оказаться разрып строки — $n-1$, в каждом можно поставить или не поставить — итого 2^{n-1} разбиений.

Пусть $OPT(j)$ — стоимость оптимального размещения слов с j-ого по n-ное. Наша задача — вычислить $OPT(1)$. А как?

$$OPT(1) = \min_{i \leqslant n} \{c(1, i) + OPT(i+1)\}$$

```
OPT(j):
    if j = n+1 then return 0
    f:= +inf
    for i:= j to n do
     f:= min(f, c(i, j)+OPT(i+1))
```

$$(*)\ \mathrm{OPT}(3) = \begin{cases} 0, & j > n \\ \min\limits_{i=j\ldots n} \{c(j,i) + \mathrm{OPT}(3)(i+1)\}, & \text{иначе} \end{cases}$$

А сложность? Построив дерево, заметим, что OPT(3) вычисляется два раза; OPT(4) – три раза и так далее.

Будем сохранять результаты:

function OPT_CACHE(t)
  if $M[j] \neq$ NULL **then**
  else
    $M[j] := \mathrm{OPT}(j)$
  return $M[j]$

Такая методика называется *динамическим программированием.*

Основная идея — каждая задача зависит от полиномиального числа других задач.