Лекция 8 от 4.02.2016

Возведение в степень

Пусть x и y содержат по n цифр.

Можно ли за полиномиальное время возвести число x в степень y?

Если мы тривиально перемножим y чисел x, несложно показать, что сложность алгоритма будет $O(2^n)$ (где n — число цифр в числе).

Заметим, что число x^y содержит $n \cdot 10^n$ цифр.

Получается, что один только размер результата экспоненциален, то есть полиномиальной сложности не хватит даже на вывод результата.

А если по модулю?

Вход: x, y, p (по n цифр).

Выход: $x^y \pmod p$.

$x, x^2 \pmod p, x^3 \pmod p \ldots$.

Попробуем *быстрое возведение в степень*.

```
Power(x, y, p)
    if y = 0 then
        return 1
    t:= Power(x, floor(y/2), p)
    if y is even then
        return t^2 mod p
    else
        return x*t^2 mod p
```

Глубина рекурсии — $O(\log y) = O(n)$.

Или вот так:

Пример: $x = 4$, $y = 5$.

x^1, x^2, $x^4 \to x^5 = x^1 \cdot x^4$

Обратная задача

Вход: x, z, p (по n цифр).

Выход: y такой, что $x^y = z \pmod p$.

Такая задача пока не решена за полиномиальное время, но и невозможность этого тоже не доказана. Это всё, вообще говоря, висит на известной проблеме $P \stackrel{?}{=} NP$ 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: $w_1, \dots, w_n; c(i,j)$.

Выход: $j_0, \dots, j_{l+1}$, 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))
```

$$(*)\ OPT(j) = \begin{cases} 0, j > n \\ \min\limits_{i=j\dots n} \{c(j,i) + OPT(i+1)\} \end{cases}$$

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

Будем сохранять результаты:

```
OPT_cache(j):
    if M[j] != NULL then
    else
        M[j] = OPT(j)
    return M[j]
```

Такая методика называется *динамическим программированием.*

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