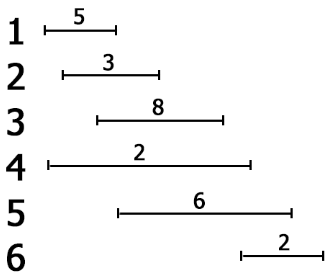
Лекция 9 от 9.02.2016

Продажа земли

Предположим, что у нас есть участок земли у берега и мы хотим его продать. При этом у нас есть несколько покупателей и каждый из них готов отдать некоторую сумму за некоторый фрагмент участка. Как максимизировать выгоду?

Формализуем: у нас есть n предложений и каждое характеризуется тремя числами — началом s, концом f и весом w. Таким образом, вход выглядит так:
$s_1, \dots, s_n$ — начала
$f_1, \dots, f_n$ — концы
$w_1, \dots, w_n$ — веса

Пример:



На выходе мы хотим получить максимальную сумму весов непересекающихся интервалов:

$$\max_{T \subseteq \{1,\dots,n\}} \sum_{i \in T} w_i$$

$$T : \forall i < j \implies f_i \leqslant s_j \vee f_j \leqslant s_i$$

Перебором задача решается за $O(2^n)$
Давайте в качестве первого шага отсортируем по правым концам за $O(n \log n)$.
Введём обозначения:
O — оптимальное решение
$OPT(i)$ — максимальаня стоимость оптимального решения для первых i интервалов
$OPT(n)$ — Максимальная стоимость оптимального решения длв всех интервалов

Пример:

$$\text{OPT}(6) = \max \begin{cases} \text{OPT}(5), 6 \notin O; \\ 2 + \text{OPT}(3), 6 \notin O; \text{(так как все до третьего не пересекаются с шестым)} \end{cases}$$

Пусть $p(j) = \max\{i < j \mid f_i \leqslant s_j\}$ — первый интервал до j-го, совместимый с ним (то есть не пересекающийся).

Эффективное вычисление p остается в качестве упражнения.

Тогда общая формула для OPT такова:

$$\text{OPT}(i) = \max \begin{cases} \text{OPT}(i-1); \\ w_i + \text{OPT}(p(i)); \end{cases}$$

Считая, что данные уже отсортированы, получим что сложность равна $O(n)$, но только если мы сохраняем результаты вычислений. Иначе мы делаем много лишних вычислений, и

Algorithm 1 Подсчёт $OPT(i)$

function ComputeOpt(i)
  if $i = 0$ **then**
    return 0
  return $\max\{\text{ComputeOpt}(i-1), w_i + \text{ComputeOpt}(p(i))\}$

время будет таким: $T(n) = T(n-1) + T(n-2) + c$. Очень похоже на числа Фибоначчи, а они растут экспоненциально; это выражение — тоже.

Значит надо сохранять вычисления в некоторый массив OPT. Инициализируем его так:

$$\text{OPT} = [0, -1, \ldots, -1]$$

Algorithm 2 Модифицированный подсчёт $OPT(i)$

function ComputeOpt(i)
  if $OPT(I) < 0$ **then**
    $OPT[i] = \max\{\text{ComputeOpt}(i-1), w_i + \text{ComputeOpt}(p(i))\}$
  return $OPT[i]$

Вот теперь сложность алгоритма — $O(n)$

Так как мы вычисляем 1 раз каждый элемент OPT, мы можем избавиться от рекурсии:

Algorithm 3 Модифицированный подсчёт $OPT(i)$ без рекурсии

function ComputeOpt(i)
  $OPT = [0, -1, \ldots, -1]$
  for $OPT(I) < 0$ **to** n **do**
    $OPT[i] = \max\{OPT[i-1], w_i + OPT[p(i)]\}$
  return $OPT[n]$

Как теперь определить, какие именно участки нужно продать? Можно хранить в OPT на i-ом месте необходимые участки, но это замедлит нашу программу (не асимптотически), так как придётся тратить на запись не константное время, а некоторое $O(n)$. Однако, можно восстановить номера участков по массиву OPT:

В общем о динамическом программировании

Чем оно отличается от “Разделяй и властвуй”? А тем, что задачи могут пересекаться. Ведь при использовании классического “разделяй и властвуй” мы бы получили экспоненциальное решение.

Для эффективного использования этого принципа необходимы следующие условия:

- Небольшое число задач; например, полиномиальное;
- Возможность их упорядочить и выразить решения следующих через предыдущие.

Задача с прошлой лекции — выравнивание текста

Дано:
$w_1, \ldots, w_n$ — длины слов.

Algorithm 4 Восстановление решения

```
function FindSolution(OPT)
    T = ∅
    i = n
    while i > 0 do
        if OPT[i − 1] > w_i + OPT[p(i)] then
            i = i − 1
        else
            T = T ∪ i
            i = p(i)
    return p(i)
```

$c(i, j)$ — штраф за размещение $w_i, \ldots, w_j$ на одной строке.

Преобразуем наше рекурсивное решение в итеративное.
OPT(i) — оптимальное размещение $w_i, \ldots, w_n$.
$\text{OPT(i)} = \min\limits_{i \leqslant j \leqslant n} \{c(i, j) + \text{OPT}(j + 1))\}$.

Запишем итеративный алгоритм для этой формулы. Так как i-ая задача зависит от задач с большим индексом, будем заполнять массив с конца.

Algorithm 5 Выравнивание текста

```
function ComputeOpt(w_1, ..., w_n)
    best = [0] × n
    OPT = [+∞] × (n + 1)
    OPT[n + 1] = 0
    for i := n downto 1 do
        for j := i to n do
            if c(i, j) + OPT[j + 1] ⩽ OPT[i] then
                OPT[i] = c(i, j) + OPT[j + 1]
                best[i] = j
    return OPT[i]
```

Как видно, в данном алгоритме решение строится по ходу, потому что в данном случае это допустимо.