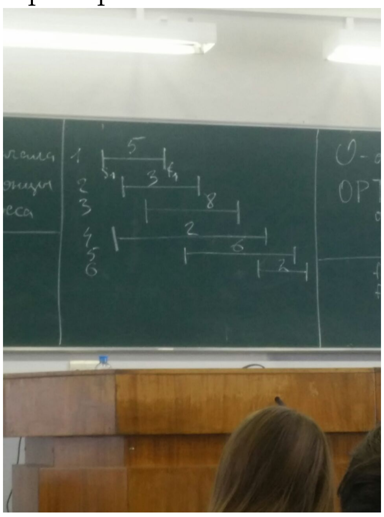
Лекция 9 от 9.02.2016

Продажа земли

Предположим, что у вас есть участок земли у берега и вы хотите его продать; при жтом у вас 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 характеерихуется тремя числами:
$s_1, \dots, s_n$ — начала
$f_1, \dots, f_n$ — концы
$w_1, \dots, w_n$ — веса
Пример:



На выходе мы хотим получить максимальную сумму весов непересекающихся интервалов: $\max\limits_{T \subseteq \{1,\dots,n\}; \forall i<j \implies f_i \leqslant s_j \vee f_j \leqslant s_i} \sum\limits_{i \in T} w_i$

Давайте в качестве первого шага отсортируем по правым концам за $O(n \log n)$. Позже будет ясно, зачем.

Пусть O — оптимальное решение, а $\mathrm{OPT}(i)$ — стоимость оптимального решения для первых i интервалов. А мы хотим вычислить $\mathrm{OPT}(n)$.

Например:

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

Пусть $p(j) = \max\{i < j \mid f_i \leqslant s_j\}$

Эффективное вычисление p остается в качестве упражнения.

Тогда общая формула для OPT такова:

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

```
ComputeOpt(i):
    if i = 0 then return 0
    return max{ComputeOpt(i-1), w_i+ComputeOpt(p(i))}
```

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

```
ComputeOpt(i):
    if OPT[i] < 0 then
        OPT[i] := max{ComputeOpt(i-1), w_i+ComputeOpt(p(i))}
    return OPT[i]
```

Другой вариант просто заполнит массив без рекурсии:

```
ComputeOpt(...)
    OPT:= [0, -1, ..., -1]
    for i:= 1 to n do
        OPT[i]:= max{ComputeOpt(i-1), w_i+ComputeOpt(p(i))}
    return OPT[n]
```

А теперь попробуем восстановить решение по массиву OPT. Можно его сохранять на каждом шаге, конечно, но это замедлит алгоритм.

```
Find_Solution(OPT):
    T:= {}
    i:= n
    while i > 0
        if OPT[i-1] > w_i + OPT[p(i)] then
            i:=i-1
        else
            T:= T \cup {i}
            i:= p(i)
    return T
```

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 нужно вот что:

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

Задача с прошлой лекции — выравнивание текста

Даны длины слов $w_1, \ldots, w_n$ и функция штрафа $c(i, j)$. Мы построили рекурсивное решение с мемоизацией; теперь построим итеративное решение.

$\mathrm{OPT}(i)$ — оптимальное размещение $w_i, \ldots, w_n$.

$\mathrm{OPT}(i) = \min\limits_{i \leqslant j \leqslant n} \{c(i, j) + \mathrm{OPT}(j + 1)\}$.

```
ComputeOpt(w_1, ..., w_n)
    best = [0]*n
    OPT:= [+\infty]*(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
i:= 1
while i < n do
    print best[i]
    i:= best[i]+1
```