Лекция 15

Приближенные алгоритмы (II). Задача о подстроке

(Конспект: О. Ионова)

15.1 Задача о покрытии множествами

Задача. Дано некоторое множество $U = \{u_1, u_2, u_3, \ldots, u_n\}$ и семейство его подмножеств: $S = \{S_1, S_2, \ldots, S_k\}$, $S_i \subseteq U$, каждому из которых сопоставлена стоимость $p_i \geqslant 0$. В сумме все эти множества $S_1, \ldots, S_k$ покрывают множество U; т.е. U содержится в объединении этих множеств. Задача заключается в том, что нам надо выбрать подмножество множества S, покрывающее U, наименьшей суммарной стоимости. Мы предъявим приближенный алгоритм для этой задачи, а затем используем его для решения другой (более практической) задачи.

Алгоритм. На каждом шаге мы будем пытаться покрыть как можно больше элементов. Т.к. некоторые множества стоят дороже, мы будем минимизировать удельную стоимость элементов.

$I := \emptyset$;
while $\bigcup\limits_{j \in I} S_j \neq U$ do
begin
  $\forall i \notin I$ cost[i]$:= p_i / |S_i \setminus \bigcup\limits_{k \in I} S_k|$;
  (Знаменатель — количество элементов, которые не были покрыты до сих пор, но будут покрыты данным множеством. Т.е., мы вычислили для каждого множества удельную стоимость покрываемых им элементов, теперь выбираем наилучшее по эффективности.*)*
  Найдем такое i_0, что cost[i_0]$= \min\limits_{i \notin C}$(cost[$i$]);
  $I := I \cup \{i_0\}$;
end;

Теперь докажем, что этот алгоритм является H_n-приближенным, где $H_n = 1 + 1/2 + \ldots + 1/n$, а $n = |U|$. Для начала мы пронумеруем все элементы множества u в том порядке, как мы их покрывали. $U = \{u_1, u_2, \ldots, u_n\}$. Каждому u_i из этих элементов мы припишем c_i — ту самую стоимость $\mathrm{cost}[i_0]$, которая была у множества, которым этот элемент впервые покрыли.

Лемма 15.1. $c_i < P^*/(n - i + 1)$*, где* P^* *— стоимость оптимального решения задачи.*

Доказательство. Мы собираемся покрыть i-ый элемент множества U, к этому моменту времени у нас уже что-то покрыто, а что-то нет. Но мы точно можем покрыть оставшееся при помощи множеств общей стоимости $\leqslant P^*$ — значит, мы можем найти конкретное множество, эффективность (cost) которого $\leqslant P^*/|U \setminus \bigcup\limits_{i \in I} S_i|$ (ясно из соображений о среднем). Но знаменатель $\geqslant n - i + 1$, поскольку мы покрываем i-ый элемент, т.е. покрыто пока лишь $i - 1$ элементов. □

Теперь при помощи этой леммы докажем, что алгоритм является H_n-приближенным. Для этого мы должны подсчитать суммарную стоимость нашего решения:

$$\sum_{i \in I} p_i = \sum_{j=1}^{n} c_j \leqslant \frac{P^*}{1 + 1/2 + \ldots + 1/n}$$

(равенство — поскольку все элементы можно разделить между множествами в соответствии с тем, каким множеством элемент был впервые покрыт), что и требовалось доказать.

Применим этот алгоритм к решению одной «биологической» задачи.

15.2 Задача о кратчайшей общей надпоследовательности

Задача: дано множество строк $\{s_1, \ldots, s_k\}$; нас интересует самая короткая строка u, которая содержит в качестве подстроки каждую из s_i.

Решение. Сведем к предыдущей задаче. Мы должны построить универсальное множество U, и множество, которое его покрывает. Не умаляя общности, можно считать, что среди строк нет подстрок друг друга. Построим строки $w_{ijk} = s_i \cdot s_j[k{+}1..|s_j|]$ для тех i, j, k, для которых суффикс s_i длины k совпадает с префиксом s_j длины k. Для любой строки s определим $\mathrm{set}(s) = \{s_i \mid s_i \text{ — подстрока } s\}$. Стоимость множества $\mathrm{set}(s)$ — длина строки s. Вход задачи о покрытии — все множества $\mathrm{set}(s_i)$ и $\mathrm{set}(w_{ijk})$. (Ясно, что $U = \{s_1, \ldots, s_k\}$.) Запускаем алгоритм, он выдает строки, мы их сливаем и получаем строку-ответ. То, что алгоритм является $2H_k$-приближенным, является очевидным следствием доказанного выше про алгоритм для задачи о покрытии множествами и следующей леммы.

Лемма 15.2. *Стоимость оптимального решения этой задачи о покрытии $\leqslant 2\cdot$длина решения исходной задачи.*

Доказательство.

```
--------------------------    s
--------- s1
     -------  s2
       ---------  s3
             ------------  s4
```

Пусть строка s — оптимальное решение исходной задачи. Перенумеруем s_i в порядке их первого вхождения в строку s. Очевидно, каждая следующая строка начинается и заканчивается позже предыдущей.

Разделим наши строки на блоки. Строим 1-ый блок: берем s_1 и все, первые вхождения в строку s которых начинаются до конца первого вхождения s_1. Сделаем из них (точнее, из первой и последней в этом блоке) общую строку (например, $w_{1,3,k}$) — это одно из множеств из условия задачи о покрытии.

Строим следующий блок, начиная его с первой невзятой строки, и т. д. Строки, которые мы сконструировали (по одной для каждого блока) задают некоторое решение задачи о покрытии множествами. Оно может и не быть оптимальным, но оптимальное — разве что еще меньше по стоимости.

Чтобы доказать, что стоимость этого решения — такая, как в формулировке, достаточно показать, что каждый символ строки s входит не более, чем в два блока. Однако, в одной позиции может пересекаться не более двух блоков (по построению). $\square$

15.3 Задача о поиске подстроки (pattern matching)

Задача: даны строки p (образец — pattern) и t (текст — text); $|p| = m$, $|t| = n$. Вопрос: встречается ли подстрока p в строке t?

Алгоритм. Тривиальный алгоритм работает $O(mn)$ шагов. Мы построим алгоритм, которому достаточно $O(m + n)$ шагов.

В алгоритме нам понадобится «таблица откатов» Π,

$$\Pi[q] = \max\{k \mid k < q,\ p[1..k] \text{ — суффикс } p[1..q]\}$$

(если таких k нет, $\Pi[q] = 0$). Как вычислить эту таблицу, мы узнаем чуть позже.

Основной алгоритм. У нас будет два «указателя» q и i; первый указывает на текущий элемент образца; второй — текста.

$q := 1$;
for $i := 1$ to n do
begin
(*) while $q > 1$ and $p[q] \neq t[i]$ do $q := \Pi[q-1]+1$;
if $p[q] = t[i]$ then $q := q+1$;
if $q = m+1$ then «Нашли!»;
end;

Покажем, что этот алгоритм заканчивает свою работу за $O(n)$ шагов. Сомнения может вызывать лишь строка (*), так как в ней имеется вложенный цикл. Однако, в ней уменьшается q. Увеличиться же оно может лишь n раз для каждого i, причем всего на единицу. Значит и тело цикла (*) не может выполниться более n раз за все время работы алгоритма.

Вычисление «таблицы откатов» Π. У нас снова будет два «указателя» k и q; на сей раз оба указывают на текущие элементы образца.

$k := 1$;
$\Pi[1] := 0$;
for $q := 2$ to m do
begin
while $k > 1$ and $p[k] \neq p[q]$ do $k := \Pi[k-1]+1$;
if $p[k] = p[q]$ then $k := k+1$;
$\Pi[q] := k-1$;
end;

То, что таблица будет вычислена за $O(m)$ шагов, показывается аналогично тому, как это было сделано для основного алгоритма (только теперь мы следим за «указателем» k).