Динамическое программирование (примеры)

0.1. Дискретная задача о рюкзаке

Есть рюкзак вместимости W, есть k предметов с некоторым объемом w_i и стоимостью c_i, где i – номер предмета. Нужно найти все предметы такие, чтобы выполнялось условие.

$$max \sum c_i : \sum w_i < W$$

.

Представим набор предметов следующим образом:

Предмет	Стоимость	Объем
p_1	c_1	w_1
p_2	c_2	w_1
p_3	c_3	w_1
$\dots$	$\dots$	$\dots$
p_k	c_k	w_k

Составим таблицу, в столбцах которой будут всевозможные веса, а в строках будут первые i предметов набора. Первую строку и первый столбец, исходя из логики задачи, можно сразу обнулить.

Набор	0	1	2	$\cdots$	W
$\{\varnothing\}$	0	0	0	$\cdots$	0
$\{p_1\}$	0				
$\{p_1, p_2\}$	0				
$\cdots$	$\vdots$				
$\{p_1, \dots p_k\}$	0				

Для заполнения таблицы потребуется рекурентное соотношение. Составим его (C_{ij} – значение в i строке и j столбце таблицы).

$$C_{ij} = \begin{cases} max\{c_i + C_{(i-1)(j-w_i)}; C_{(i-1)j}\}, & i,j > 0 \\ 0, & i \leq 0 \\ 0, & j \leq 0 \end{cases}$$

Прямым или обратным ходом заполняем таблицу. Ответ к исходной задаче будет находится на пересечении k столбца и k строки (В последней клетке таблицы).

Набор	0	1	2	$\cdots$	W
$\{\varnothing\}$	0	0	0	$\cdots$	0
$\{p_1\}$	0	C_{11}	C_{12}	$\cdots$	C_{1W}
$\{p_1, p_2\}$	0	C_{21}	C_{22}	$\cdots$	C_{2W}
$\cdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\{p_1, \ldots p_k\}$	0	C_{k1}	C_{k2}	$\cdots$	C_{kW}

Сложность алгоритма: $O(kW)$. В данной задаче большую роль играют единицы измерения объема. В некоторых случаях динамическое решение может оказаться менее эффективным в сравнении с полным перебором вариантов. Так происходит, если объем не является целым числом. В этом случае нужно брать значение с некоторой погрешностью. И номера столбцов должны быть пропорциональны $gcd(w_1, \ldots, w_k)$.

0.2. Расстояние Левенштейна

Расстояние Левенштейна (редакционное расстояние, дистанция редактирования) – Минимальное количество односимвольных операций (вставки, удаления, замены), необходимых для превращения одной строки в другую. В общем случае, операциям, используемым в этом преобразовании, можно назначить разные цены.

Например, возьмем строки: ABC и BCD. Преобразуем первую из них.

$$ABC \to BC \to BCD$$

Расстояние Левенштейна для этих строк равно 2. Теперь рассмотрим общее динамическое решение задачи на примере. Даны следующие строки.

Строка 1 (S_1) : abcbc
Строка 2 (S_2) : ccba

Будем рассматривать все подстроки этих строк. В столбцах таблицы будут расположены подстроки S_1, а в строках таблицы будут расположены подстроки S_2. Составим таблицу.

	{∅}	a	ab	abc	abcb	abcbc
{∅}						
c						
cc						
ccb						
ccba						

Для заполнения таблицы потребуется рекурентное соотношение. Составим его (D_{ij} – расстояние Левенштейна для подстрок в i строке и j столбце таблицы). Также $S_{1,2}(i)$ это i символ строки.

$$D_{ij} = \begin{cases} min\{D_{i(j-1)} + 1; D_{(i-1)j} + 1; D_{(i-1)(j-1)} + 1 * (S_1(j) \neq S_2(i))\}, & i, j > 0 \\ 0, & i = j = 0 \\ \text{i}, & j = 0 \\ \text{j}, & i = 0 \end{cases}$$

Тепрь заполним таблицу. Ответ на поставленную задачу будет находиться в последней клетке.

	{∅}	a	ab	abc	abcb	abcbc
{∅}	0	1	2	3	4	5
c	1	1	2	2	3	4
cc	2	2	2	2	3	3
ccb	3	3	2	3	2	3
ccba	4	3	3	3	3	3

Получим следующее преобразование строки ccba:

$$\text{ccba} \to \text{accba} \to \text{abcba} \to \text{abcbc}$$

В данной задаче можно также рассматривать операции (удаления, замены, добавления), у которых есть цена, вес. В этом случае алгоритм идентичный, но в таблицу теперь нужно записывать соответствующую стоимость операции.

0.3. Поиск наибольшей общей подпоследовательности

Рассмотрим общее динамическое решение задачи на примере. Даны следующие строки.

Строка 1 (S_1) : abcbc
Строка 2 (S_2) : ccba

Требуется найти их наибольшую общую подпоследовательность. Составим таблицу по тому же принципу, что и в расстоянии Левенштейна. Рассмотрим все возможные подстроки.

	$\{\varnothing\}$	a	ab	abc	abcb	abcbc
$\{\varnothing\}$						
c						
cc						
ccb						
ccba						

Для заполнения таблицы потребуется рекурентное соотношение. Составим его (lcs_{ij} – длина наибольшей общей подпоследовательности подстрок в i строке и j столбце таблицы). Также $S_{1,2}(i)$ это i символ строки.

$$lcs_{ij} = \begin{cases} max\{lcs_{i(j-1)}; lcs_{(i-1)j}; lcs_{(i-1)(j-1)} + 1 * (S_1(j) = S_2(i))\}, & i, j > 0 \\ 0, & i = 0 \text{ or } j = 0 \end{cases}$$

Тепрь заполним таблицу. Ответ на поставленную задачу будет находиться в последней клетке.

	$\{\varnothing\}$	a	ab	abc	abcb	abcbc
$\{\varnothing\}$	0	0	0	0	0	0
c	0	0	0	1	1	1
cc	0	0	0	1	1	2
ccb	0	0	1	1	2	2
ccba	0	1	1	1	2	2

Получили, что cb – наибольшая общая подпоследовательность строк ccba и abcbc.