Сириус
IT-Колледж

# Лекция 4. Динамическое программирование

Задача «Калькулятор»: ленивое ДП и ДП назад. Общие принципы динамического программирования. Восстановление ответа. Наибольшая возрастающая подпоследовательность. Дискретная задача о рюкзаке. Редакционное расстояние. Задача коммивояжёра.

Сириус IT-Колледж

# Динамическое программирование

Словосочетание **«динамическое программирование»** впервые было использовано в 1940-х годах Ричардом Беллманом для описания процесса нахождения решения задачи, где ответ на одну задачу может быть получен только после решения задачи, «предшествующей» ей.

Первоначально эта область была основана, как системный анализ и инжиниринг.

Слово «программирование» в словосочетании «динамическое программирование» в действительности к «традиционному» программированию (написанию кода) почти никакого отношения не имеет и имеет смысл как в словосочетании «математическое программирование», которое является синонимом слова «оптимизация». Поэтому слово «программа» в данном контексте скорее означает оптимальную последовательность действий для получения решения задачи.

Сириус IT-Колледж

# Динамическое программирование

**Динамическое программирование** — способ решения сложных задач путём разбиения их на более простые подзадачи. Он применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной. В этом случае время вычислений, по сравнению с «наивными» методами, можно значительно сократить.

Сириус
IT-Колледж

**Динамическое программирование — это когда у нас есть задача, которую непонятно как решать, и мы разбиваем ее на меньшие задачи, которые тоже непонятно как решать.**

**(c) А.Кумок**

Сириус IT-Колледж

# Принципы ДП

Метод динамического программирования сверху (мемоизация, ленивая динамика) — это простое запоминание результатов решения тех подзадач, которые могут повторно встретиться в дальнейшем.

Динамическое программирование снизу (табулация) включает в себя переформулирование сложной задачи в виде рекурсивной последовательности более простых подзадач.

# Динамическое программирование

Сириус IT-Колледж

Основная идея состоит в том, чтобы

1. свести задачу для N к задаче для чисел, меньших, чем N (с помощью формулы)
2. хранить все ответы в массиве
3. заполнить начало массива вручную (для которых формула не работает)
4. обойти массив и заполнить ответы по формуле
5. вывести ответ откуда-то из этого массива

Сириус
IT-Колледж

# Наибольшая общая подпоследовательность (НОП, Longest Common Subsequence, LCS)

Рассмотрим последовательность чисел $a_1, a_2, \ldots, a_n$. Если вычеркнуть из этой последовательности часть чисел, мы получим другую последовательность, которую называют **подпоследовательностью** данной последовательности.

Рассмотрим теперь еще одну последовательность $b_1, b_2, \ldots, b_m$. Требуется найти длину самой длинной подпоследовательности последовательности $\{a_i\}$, которая одновременно является и подпоследовательностью последовательности $\{b_i\}$. Такую последовательность называют **наибольшей общей подследовательностью (НОП)**. Например, для последовательностей *1, 2, 3, 4, 5* и *2, 7, 3, 2, 5* НОП является подпоследовательность *2, 3, 5*, состоящая из трёх членов.

Сириус IT-Колледж

# Наибольшая общая подпоследовательность (НОП, Longest Common Subsequence, LCS)

Опишем подзадачи, на которые мы будем разбивать нашу задачу. Мы напишем функцию LCS(p, q), которая находит длину НОП для двух начальных участков $a_1, \ldots, a_p$ и $b_1, \ldots, b_q$ наших последовательностей. Пусть для всех пар q и p ($p < n, q < m$), мы задачу решать уже научились. Попробуем вычислить LCS(n, m). Рассмотрим два случая:

1. $a_n = b_m$. Тогда LCS(n, m)=LCS(n-1, m-1)+1.
2. $a_n \neq b_m$. Тогда LCS(n, m)=max(LCS(n, m-1), LCS(n-1, m)).

Пользуясь этими формулами, мы можем заполнить таблицу значений LCS(p, q) для всех p и q последовательно: сначала заполняем первую строчку слева направо, затем вторую и т. д. Последнее число в последней строке и будет ответом на поставленную задачу.

Данный алгоритм требует порядка **O(nm)** операций.

Сириус IT-Колледж

# Задача о рюкзаке

**Задача о рюкзаке** (Knapsack problem) — дано $N$ предметов, $n_i$ предмет имеет массу $w_i>0$ и стоимость $p_i>0$. Необходимо выбрать из этих предметов такой набор, чтобы суммарная масса не превосходила заданной величины $W$ (вместимость рюкзака), а суммарная стоимость была максимальна.

Задачу о рюкзаке можно решить несколькими способами:

- Перебирать все подмножества набора из N предметов. Сложность такого решения $O(2^N)$
- Методом Meet-in-the-middle. Сложность решения $O(2^{N/2}N)$
- Метод динамического программирования. Сложность — $O(NW)$

Сириус IT-Колледж

# Задача о рюкзаке

Для решения построим таблицу размерности ***N*** на ***W***, где столбцы соответствуют объему рюкзака, а строки отдельным предметам.

В общем случае формула для стоимости в каждой ячейке выглядит так:

***S[i, j] = max (S[i−1, j], цена i-го предмета + S[i−1, j−вес i-го предмета])***, где ***i*** — номер строки, ***j*** — столбца.

Сириус IT-Колледж

# Редакционное расстояние

**Расстояние Левенштейна** (редакционное расстояние или дистанция редактирования) между двумя строками в теории информации и компьютерной лингвистике — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Нахождения расстояния Левенштейна аналогично нахождению НОП для двух строк, за исключением того, что мы ищем не максимальное, а минимальное количество.

Сириус IT-Колледж

# Задача коммивояжёра

**Задача коммивояжёра** (или TSP, travelling salesman problem) — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город только один раз.