**Тема**: Динамическое программирование

**Источники**:

<https://habr.com/ru/post/191498/>

<https://ru.wikipedia.org/wiki/Динамическое_программирование>

<https://www.youtube.com/watch?v=iKj-xI4enLw> (YouTube нарушает законодательство РФ)

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.

Ключевая идея в динамическом программировании достаточно проста. Как правило, чтобы решить поставленную задачу, требуется решить отдельные части задачи (подзадачи), после чего объединить решения подзадач в одно общее решение. Часто многие из этих подзадач одинаковы. Подход динамического программирования состоит в том, чтобы решить каждую подзадачу только один раз, сократив тем самым количество вычислений. Это особенно полезно в случаях, когда число повторяющихся подзадач экспоненциально велико.

*Оптимальная подструктура* в динамическом программировании означает, что оптимальное решение подзадач меньшего размера может быть использовано для решения исходной задачи. К примеру, кратчайший путь в графе из одной вершины (обозначим s) в другую (обозначим t) может быть найден так: сначала считаем кратчайший путь из всех вершин, смежных с s, до t, а затем, учитывая веса рёбер, которыми s соединена со смежными вершинами, выбираем лучший путь до t (через какую вершину лучше всего пойти). В общем случае мы можем решить задачу, в которой присутствует оптимальная подструктура, проделывая следующие три шага:

1. Разбиение задачи на подзадачи меньшего размера.
2. Нахождение оптимального решения подзадач рекурсивно, проделывая такой же трехшаговый [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC).
3. Использование полученного решения подзадач для конструирования решения исходной задачи.

Подзадачи решаются делением их на подзадачи ещё меньшего размера и т. д., пока не приходят к тривиальному случаю задачи, решаемой за константное время (ответ можно сказать сразу). К примеру, если нам нужно найти n!, то тривиальной задачей будет 1! = 1 (или 0! = 1).

*Перекрывающиеся подзадачи* в динамическом программировании означают подзадачи, которые используются для решения некоторого количества задач (не одной) большего размера (то есть мы несколько раз проделываем одно и то же). Ярким примером является вычисление [последовательности Фибоначчи](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B0_%D0%A4%D0%B8%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D0%B8), {\displaystyle F\_{3}=F\_{2}+F\_{1}} и {\displaystyle F\_{4}=F\_{3}+F\_{2}} — даже в таком тривиальном случае вычисления всего двух чисел Фибоначчи мы уже посчитали {\displaystyle F\_{2}} дважды. Если продолжать дальше и посчитать {\displaystyle F\_{5}}, то {\displaystyle F\_{2}} посчитается ещё два раза, так как для вычисления {\displaystyle F\_{5}} будут нужны опять {\displaystyle F\_{3}} и {\displaystyle F\_{4}}. Получается следующее: простой рекурсивный подход будет расходовать время на вычисление решения для задач, которые он уже решал.

Чтобы избежать такого хода событий, необходимо сохранять решения подзадач, которые мы уже решали, и когда нам снова потребуется решение подзадачи, мы вместо того, чтобы вычислять его заново, просто достанем его из памяти.

Подводя итоги вышесказанного, можно сказать, что динамическое программирование пользуется следующими свойствами задачи:

* перекрывающиеся подзадачи;
* оптимальная подструктура;
* возможность запоминания решения часто встречающихся подзадач.

Формирование решения задачи с помощью динамического программирования

Для того, чтобы сформировать решение задачи с помощью динамического программирования необходимы 3 шага:

1. Определить решаемые значения;
2. Определить рекуррентное соотношение/формулу;
3. Определить начальные значения;
4. Определить порядок вычисления значений
5. Определение/Анализ местонахождения результата

В качестве примера будут использоваться известные многим числа Фибоначчи – F(n) = F(n-1) + F(n-2), при этом F(0) = 0; F(1) = 1.

Для того, чтобы запоминать решённые или первоначальные значения, будет использоваться массив.

Первым шагом определяем решаемые значения чисел Фибоначчи: F(n) – n-ый элемент последовательности чисел Фибоначчи.

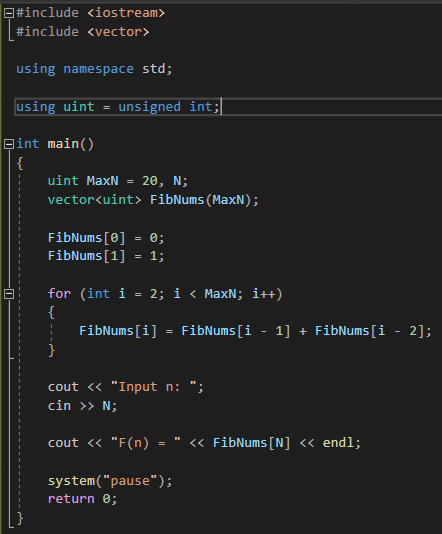
Вторым шагом определяем рекуррентную формулу: F(n) = F(n – 1) + F(n – 2).

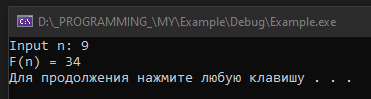
Третьим шагом определяем начальные значения: F(0) = 0 и F(1) = 1.

Четвёртый шаг: определяем линейный порядок вычисления с начала (от 1-ого элемента к последнему).

Пятый шаг: анализируем местонахождение F(n). Оно будет находиться в массиве чисел Фибоначчи под индексом n (Другими словами: n-ым элементом вычисленной нами последовательности чисел Фибоначчи).

Для того, чтобы реализовать данный способ нахождения последовательности чисел Фибоначчи на определённом языке программировании, необходимо задать максимальное n, до которого и будет вычисляться данная последовательность. Иначе она будет вычисляться «бесконечно»…





С помощью применения динамического программирования удалось сократить вычисление последовательности чисел Фибоначчи с ~O(2^n) до O(n).