Динамическое программирование

Динамическое программирование – способ решения сложных задач путём разбиения их на более простые подзадачи.

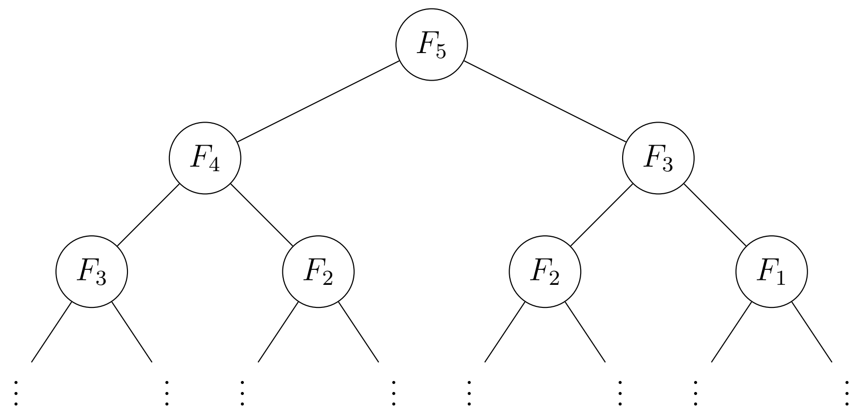
Часто многие из этих подзадач одинаковы. Подход динамического программирования состоит в том, чтобы решить каждую подзадачу только один раз, сократив тем самым количество вычислений.

0.1. Реккурсивное решение

Рассмотрим числа Фибоначчи. Они образуют последовательность $\{F_n\}$, которая задается следующим реккурентным соотношением:

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}, n \geq 2, n \in Z$$

Проанализируем реккурсивный способ получения последовательности. Видно, что данный подход избыточен, так как значения F_i высчитываются по несколько раз.



Приходим к тому, что количество действий в таком алгоритме тоже равно числу Фибоначчи. Воспользуемся формулой Бине, чтобы оценить скорость роста последовательности.

$$F_n = \left(\frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}} \right)$$

Получаем, что скорость роста последовательности чисел фибоначчи: $O(\varphi^n)$, где $\varphi = \frac{1+\sqrt{5}}{2}$. Реккурсия является не самым оптимальным способом решения задач.

0.2. Динамическое решение

Для динамического решения любой задачи нужно:

1) Выразить задачу реккурентно

2) Определить множество требуемых значений

3) Заполнить прямым или обратным ходом

Составим динамическое решение задачи поиска n-ого числа Фибоначчи. Создадим массив $a[n+1] = \{-1, -1, \ldots\}$. Следующая функция обратным ходом заполняет массив, при этом сложность алгоритма $O(n)$.

```
int F(int* array, int n)
{
        if (array[n] == -1)
        {
                array[n] = F(n - 1) + F(n - 2);
        }
        return array[n];
}
```

Теперь рассмотрим прямой ход метода динамического программирования.

```
int F(int* array, int n)
{
        array[0] = 0;
        array[1] = 1;
        for (int i = 2; i <= n; i++)
        {
                array[i] = array[i - 1] + array[i - 2];
        }
        return array[n];
}
```

0.3. Оптимальная подструктура

Задача имеет оптимальную подструктуру, если её оптимальное решение может быть рационально составлено из оптимальных решений её подзадач. Наличие оптимальной подструктуры в задаче используется для определения применимости динамического программирования и жадных алгоритмов для ее решения. Например, задача по нахождению кратчайшего пути между некоторыми вершинами графа содержит в себе оптимальное решение подзадач.

Иногда оптимальная подструктура может отсутствовать в задаче. Рассмотрим задачу, в которой имеется ориентированный граф $G = (V, E)$ и вершины $u, v \in V$, задачу по определению простого пути от вершины u к вершине v, состоящий из максимального количества рёбер.

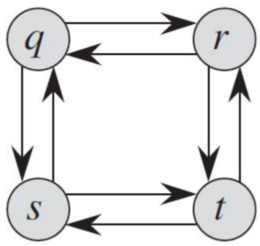


Рис. 1. Задача о самом длинном невзвешенном пути

Рассмотрим путь $q \to r \to t$, который является самым длинным простым путем $q \longrightarrow t$. Является ли путь $q \to r$ самым длинным путем $q \longrightarrow r$? Нет, поскольку простой путь $q \to s \to t \to r$ длиннее. Является ли путь $r \to t$ самым длинным путем $r \longrightarrow t$? Снова нет, поскольку простой путь $r \to q \to s \to t$ длиннее. Таким образом, в задаче о поиске самого длинного невзвешенного пути не возникает никаких оптимальных подструктур.

0.4. Пример

Требуется разложить натуральное число n в сумму натуральных квадратов с наименьшим количеством слагаемых. К данной задаче применимо динамическое решение.

Создадим двумерный массив $a[n+1][sqrt(n)+1]$, номер столбца – это число, а коэффициенты в столбцах – это количество соответствующих квадратов в сумме. Номера строк – это числа, квадраты которых могут присутствовать в разложении. Столбец под номером 0 обнулен. Теперь составим реккурентное соотношение.

$$F(x) = 1 + min(F(x - i^2))$$

Здесь i пробегает все значения от 1 до sqrt(n). Минимальное значение берется по количеству натуральных квадратов в сумме. Массив можно заполнять как прямым, так и обратным ходом.