* [Рекурсия с мемоизацией](https://foxford.ru/lessons/32273/conspects/1)
* [Связь рекурсии с динамическим программированием](https://foxford.ru/lessons/32273/conspects/2)

**Рекурсия с мемоизацией**

При решении задач методом динамического программирования зачастую бывает сложно определить порядок, в котором необходимо заполнять массив значениями. Как правило, это происходит в сложных случаях многомерной динамики, например, в динамическом программировании по подотрезкам.

Мы рассматривали задачу нахождения максимальной подпоследовательности данной строки, являющейся палиндромом, решавшейся при помощи динамического программирования.

В этой задаче была функция f(i,j), значение которой равно длине ответа для подотрезка данной строки начиная от символа s[i] и до символа s[j]. При этом возникает проблема определения порядка обхода двумерного массива.

Вместо того, чтобы определять порядок обхода, попробуем реализовать решение при помощи рекурсии. Для этого запишем функцию вычисления ответа при помощи рекурсии:

**int f(int i, int j)**

**{**

**if (j < i)**

**return 0;**

**if (i == j)**

**return 1;**

**if (s[i] == s[j])**

**return 2 + f(i + 1, j - 1);**

**else**

**return max(f(i + 1, j), f(i, j - 1));**

**}**

Но такое решение будет работать слишком долго, так так функция будет вызываться многократно для повторяющихся наборов параметров. Чтобы избежать повторяющихся вызовов рекурсии, заведем двумерный массив

**ans[n][n]**

, в котором будет храниться ответ, то есть значение f(i,j) будет записываться в элемент массива

**ans[i][j]**

. Если функция будет вызвана для какого-то набора параметров f(i,j), то найдя ответ функция запишет его в соответствующую ячейку массива. При этом в массиве также может быть записано специальное значение, обозначающее, что ответ не был еще посчитан. Будем использовать в качестве такого значения число -1. Таким образом, функция при вызове проверяет, если значение элемента массива не равно -1, то это означает, что функция уже вызывалась для такого набора параметров и ответ был посчитан. В этом случае функция сразу же возвращает вычисленное значение. Иначе функция вызывает себя рекурсивно для нахождения ответа, но после нахождения ответа записывает его в массив, чтобы не вычислять его повторно в следующий раз.

Пример подобной реализации:

**vector <vector <int> > ans(n, vector<int>(n, -1));**

**int f(int i, int j)**

**{**

**if (ans[i][j] != -1)**

**return ans[i][j];**

**if (j < i)**

**ans[i][j] = 0;**

**else if (i == j)**

**ans[i][j] = 1;**

**else if (s[i] == s[j])**

**ans[i][j] = 2 + f(i + 1, j - 1);**

**else**

**ans[i][j] = max(f(i + 1, j), f(i, j - 1));**

**return ans[i][j];**

**}**

Такая технология называется рекурсией с мемоизацией или ленивой динамикой.

#### Связь рекурсии с динамическим программированием

Динамическое программирование часто используется вместо рекурсии, когда необходимо подсчитать количество каких-либо комбинаторных объектов, или количество способов построить какую-либо конструкцию.

Рассмотрим пример - пусть необходимо сгенерировать все разбиения числа n на слагаемые, разбиения, отличающиеся порядком слагаемых, будем считать различными. Поэтому будем записывать все слагаемые в порядке неубывания, а сами разбиения выводить в лексикографическом порядке.

Обычно это делается при помощи рекурсивной функции подобной такой:

**void generate(vector<int> & prefix, int n, int k)**

**{**

**if (n == 0)**

**print(prefix);**

**else**

**for (int i = 1; i <= min(n, k); ++i)**

**{**

**prefix.push\_back(i);**

**generate(prefix, n - i, i);**

**prefix.pop\_back();**

**}**

**}**

Если в этой функции не производить печать найденного разбиения, то можно написать функцию, которая возвращает количество найденных разбиений. Заметим, что теперь нам не нужен параметр prefix, так как он используется только для вывода.

**int count (int n, int k)**

**{**

**if (n == 0)**

**return 1;**

**int ans = 0;**

**for (int i = 1; i <= min(n, k); ++i)**

**ans += count(n - i, i);**

**return ans;**

**}**

Эта функция, вызванная с параметрами (n,k) возвращает количество разбиений числа n на слагаемые, которые не превосходят k.

Однако эта функция будет работать достаточно медленно из-за пересечения подзадач. Заменим рекурсию на динамическое программирование (мемоизацию):

**vector <vector<int> > ans(n + 1, vector<int>(n + 1, -1));**

**int count (int n, int k)**

**{**

**if (ans[n][k] != -1)**

**return ans[n][k];**

**if (n == 0)**

**ans[n][k] = 1;**

**else**

**{**

**ans[n][k] = 0;**

**for (int i = 1; i <= min(n, k); ++i)**

**ans[n][k] += count(n - i, i);**

**return ans[n][k];**

**}**