# **DP1**

Динамическое программирование – это особый подход к решению сложных рекурсивных задач с **сильно перекрывающейся структурой подзадач**. Это означает, что некоторые действия повторяются снова и снова, с одинаковыми входными данными и результатом. DP позволяет избежать повторяющихся вычислений, тем самым уменьшая время выполнения программы.

Рассмотрим работу DP на примере решения задачи про числа Фибоначчи. Требуется найти n-ый элемент последовательности. Каноническая формула для вычисления n-ного элемента ряда выглядит так:

F0=0; F1=1; Fn=Fn-1+Fn-2

Допустим, что нам нужно найти 7 элемент последовательности.

На графе видно, что в процессе поиска возникают одни и те же ветви, т.е. мы будем проделывать повторяющиеся действия. Чтобы не тратить время на одних и тех же действиях, существуют методы оптимизации.

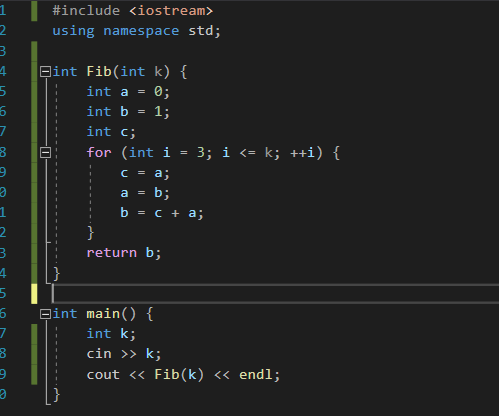
1. Мемоизация. Алгоритм состоит из двух частей.

А. Если значение функции от нужной переменной не находится в памяти, то мы просчитываем и заносим в память.

Б. В противном случае просто воспользуемся уже имеющимся значением от этой переменной.

1. DP снизу вверх. До этого наш алгоритм был построен на противоположном решении. Сейчас же можно идти от первого значения до n-ного, тем самым мы получим результаты от переменных еще до того, как они нам понадобятся.
2. В условиях данной задачи нам не обязательно хранить абсолютно все найденные нами значения по методу снизу-вверх. Достаточно запомнить только последние 2 результата. Таким образом, помимо ускорения по времени, мы получим уменьшение необходимой памяти.

Реализация:



# **M2**

#include<mutex>

using namespace std;

int rc = 0, wc = 0;

mutex m1, m2, w, r;

void Read() {

r.unlock();

m1.unlock();

rc++;

if (rc == 1) {

w.unlock();

}

m1.lock();

r.lock();

//read

m1.unlock();

rc--;

if (rc == 0) {

w.lock();

}

m1.lock();

}

void Write() {

m2.unlock();

wc++;

if (wc == 1) {

r.unlock();

}

m2.lock();

w.unlock();

//write

w.lock();

m2.unlock();

wc--;

if (wc == 0) {

r.lock();

}

m2.lock();

}