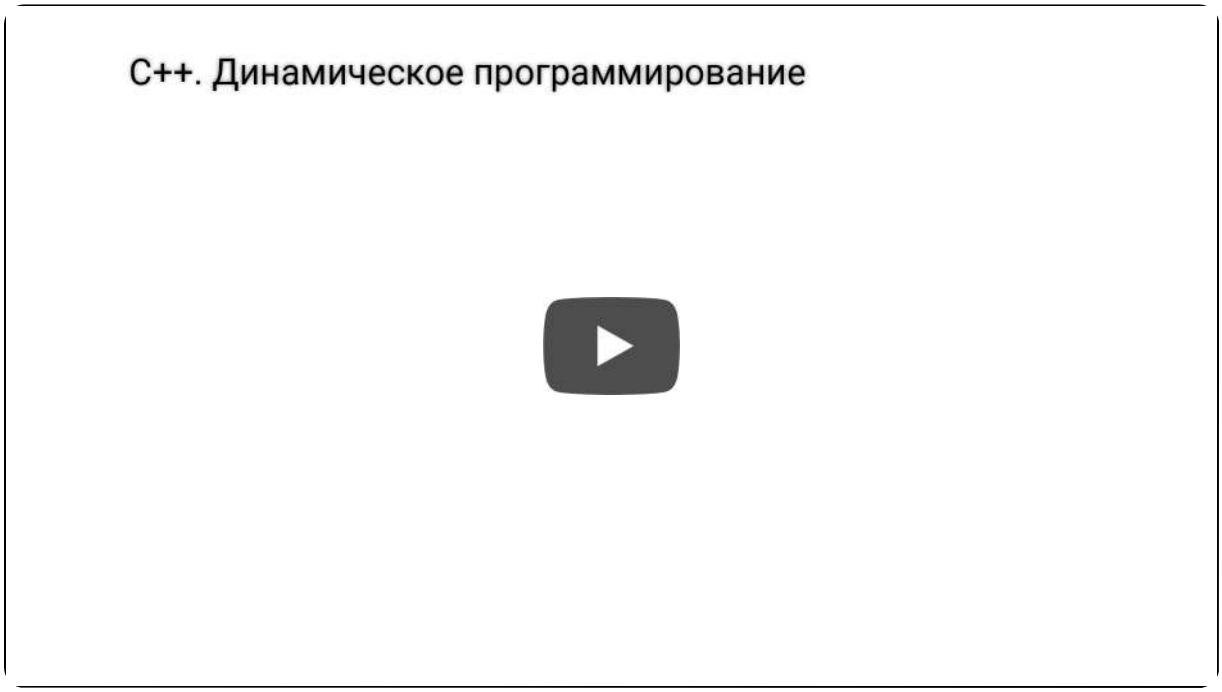
C++. Динамическое программирование.



Динамическое программирование — метод решения задачи путём разбиения её на меньшие подзадачи, которые имеют такую же структуру. Для того чтобы задача решалась методом динамического программирования, она должна обладать следующими свойствами:

- оптимальная подструктура (оптимальное решение задачи может быть составлено из оптимальных решений её подзадач)
- пересекающиеся подзадачи (одна и та же подзадача нужна для решения большого количества других подзадач)
- возможность мемоизации (ограничения на память позволяют запоминать ответы на все решённые подзадачи)

Первое свойство необходимо как для задач на динамическое программирование, так и для задач на рекурсию. Второе и третье свойства отличают задачи на динамическое программирование от задач на рекурсию и, как правило, позволяют сократить время работы с экспоненциального до полиномиального.

Рекурсивные методы решения задач, как правило, производят полный перебор

всех вариантов. Динамическое программирование также производит полный перебор всех вариантов, но делает это оптимизированно за счёт мемоизации (каждая подзадача решается только один раз и её решение сохраняется).

Задача. Нахождение N-го числа Фибоначчи. Числа Фибоначчи — это ряд чисел, который определён первыми двумя членами $F_0 = 0, F_1 = 1$ и рекуррентным соотношением $F_N = F_{N-1} + F_{N-2}, N \geq 2$. Можно выписать несколько первых членов этого ряда:

$F_0 = 0$

$F_1 = 1$

$F_2 = F_1 + F_0 = 1 + 0 = 1$

$F_3 = F_2 + F_1 = 1 + 1 = 2$

$F_4 = F_3 + F_2 = 2 + 1 = 3$

$F_5 = F_4 + F_3 = 3 + 2 = 5$

$F_6 = F_5 + F_4 = 5 + 3 = 8$

$F_7 = F_6 + F_5 = 8 + 5 = 13$

$F_8 = F_7 + F_6 = 13 + 8 = 21$

Рекурсивный алгоритм нахождения N-го числа Фибоначчи выглядит следующим образом:

```
int F(int n) {
    if (n < 2) return n;
    return F(n - 1) + F(n - 2);
}
```

Этот рекурсивный алгоритм работает за экспоненциальное время. Можно показать, что время работы этого алгоритма есть $O(a^N)$, где $a \approx 1.6$.

Можно немного модифицировать этот алгоритм так, чтобы он работал по принципу динамического программирования за время $O(N)$. Для этого необходимо добавить запоминание тех подзадач, которые уже были решены. Необходимо завести массив, назовём его dp, который изначально заполнен

значениями -1. Если $dp[i]$ равно -1, то значит задача для значения i ещё не была решена, и мы будем решать её как в рекурсивном случае, в противном случае сразу вернём значение $dp[i]$, которое уже было найдено ранее и сохранено в нашем массиве. Код имеет следующий вид:

```
int F(int n) {
    if (dp[n] != -1) return dp[n];
    if (n < 2) return n;
    return dp[n] = F(n - 1) + F(n - 2);
}
```

Такой подход к реализации алгоритма принято называть нисходящим, потому что решая задачу, мы переходим к её подзадачам. Есть и восходящий подход, который состоит в решении задач подряд от меньших к большим с запоминанием результатов. Реализация такого алгоритма для $n > 0$ имеет следующий вид:

```
cin >> n;
dp.resize(n + 1);
dp[0] = 0;
dp[1] = 1;
for (int i = 2; i < n + 1; ++i)
    dp[n] = dp[n - 1] + dp[n - 2];
```

Для данного примера может показаться, что восходящий способ проще, чем нисходящий. Но если задача зависит не от одного параметра, а от двух или трёх, то нисходящий способ, как правило, оказывается проще.