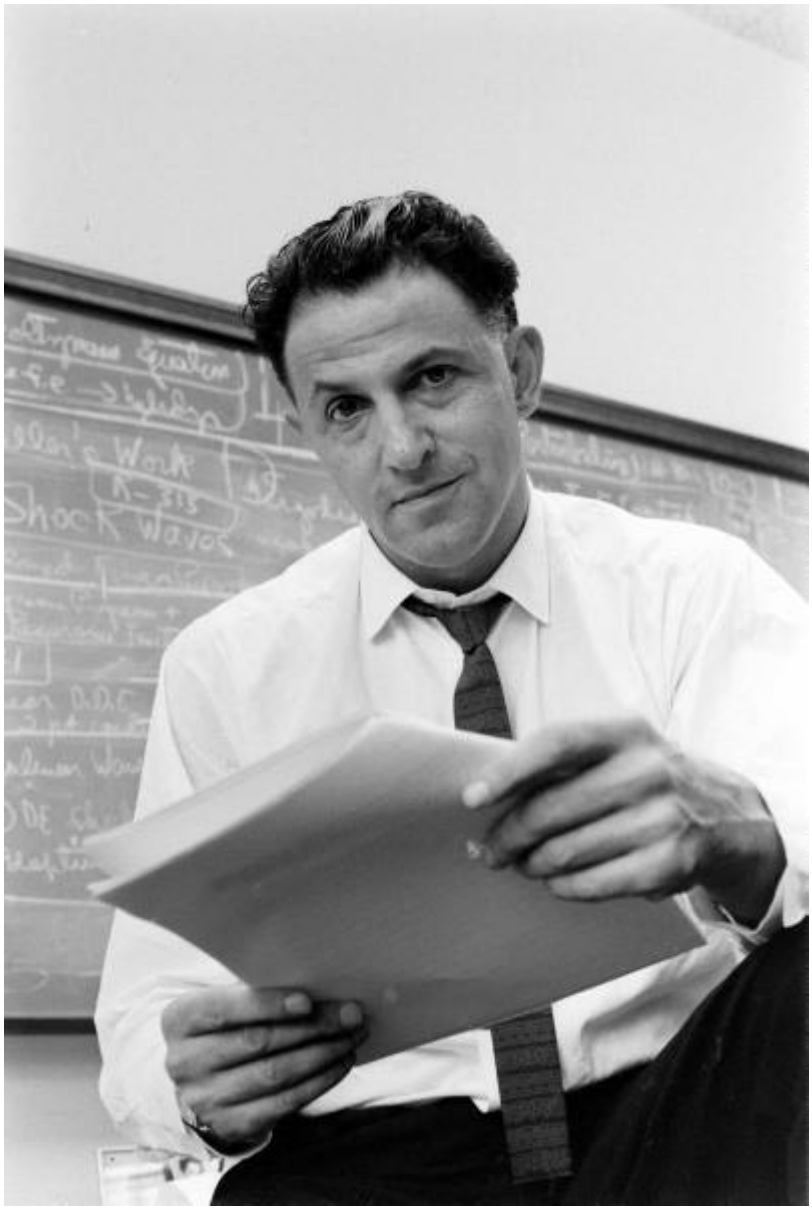
Введение в динамическое программирование (ДП)

Само понятие «динамическое программирование» впервые было использовано в 1940-х годах Ричардом Беллманом для описания процесса нахождения решения задачи, где ответ на одну задачу может быть получен только после решения другой задачи, «предшествующей» ей. Таким образом, американский математик и один из ведущих специалистов в области математики и вычислительной техники — **Ричард Эрнст Беллман** — стал пророделем динамического программирования.



Позднее формулировка понятия была доработана и усовершенствованна до современного вида самим же Беллманом.

Слово «программирование» в контексте «динамическое программирование» на самом деле к классическому пониманию программирования (написанию кода на языке программирования) **практически никакого отношения не имеет**. Слово «Программирование» имеет такой же смысл как в словосочетании «математическое программирование», которое является синонимом слова «оптимизация».

Поэтому **программы** будут использоваться в качестве оптимальной последовательности действий для получения решения задачи.

В общем же для начала, **неформальное определение понятия динамического программирования** может звучать так:

Динамическое программирование — это техника или метод, которая позволяет решать некоторые задачи комбинаторики, оптимизации и другие задачи, обладающие определенным свойством (свойством сооптимальности у подзадач).

Задачи оптимизации, как правило, связаны с задачей максимизации или минимизации той или иной целевой функции (например, максимизировать вероятность того, что система не сломается, максимизировать мат. ожидание получения прибыли и т.д.).

Задачи комбинаторики, как правило, отвечают на вопрос, сколько существует объектов, обладающих теми или иными свойствами, или сколько существует комбинаторных объектов, обладающих заданными свойствами.

То есть, ДП решает не все задачи, а лишь некоторые, определенный класс подзадач. Но этот класс подзадачи используется во многих областях знаний: программирование, математика, лингвистика, статистика, теория игр, экономика, в компьютерных науках и т.п.

Задачи, решаемые при помощи динамического программирования, должны обладать **свойством сооптимальности**, о котором будет сказано в дальнейших уроках.

Неформальное объяснение свойства оптимальности у подзадач может быть продемонстрировано с помощью диаграммы:
Есть задача, которую мы хотим решить при помощи ДП, т.е. найти какой-то план ее решения. Допустим эта задача сложна и сразу решить мы ее не можем. Мы берем малую подзадачу и решаем сначала ее (для `x1`). Затем используя это малое решение `x1`, и не меняя структуру этого решения, решаем следующую задачу уже с `x1` и `x2`. И т.д.

Рис. 1.1. Неформальное объяснение свойства оптимальности у подзадач

Более подробно неформальное объяснение рассматривается ниже.

Примеры, решаемых при помощи динамического программирования задач

Сначала рассмотрим задачи оптимизации (задачи 1-5):

1. Маршрут оптимальной длины

Пример: Есть некоторая карта дорог, представленная в виде графа. Наша цель: добраться из пункта **А** в пункт **Б**. Это сделать надо так, чтобы минимизировать расстояние или потраченное топливо.

Целевой функцией здесь является расстояние от **А** до **Б**. Т.е. наша цель — минимизировать расстояние.

А что является **переменной выбора**? Для того, чтобы найти кратчайший путь, надо каждый раз принимать решения. Т.е. в каждой точке или на каждом перекрестке необходимо принимать решения: куда повернуть или ехать прямо.

> **Важно:** Из этой задачи уже можно увидеть общую структуру задач, решаемых при помощи динамического программирования: **в каждой задаче есть целевая функция и переменная выбора**.

2. Замена машины (минимизация расходов)

> **Пример:** Каждый год мы принимаем решение, ездить ли на старой машине еще год и понести при этом издержки на поддержку и обслуживание старой машины или же продать эту машину и купить новую (и понести при этом издержки на покупку).

Целевая функция: минимизация расходов (либо на издержки на поддержку старого автомобиля, либо на покупку нового).

Переменная выбора: каждый год принимать решение продать машину или оставить.

3. Биржевой портфель

> **Пример:** Игра на бирже, приобретение акций каких-либо компаний

Целевая функция: максимизация **средних** доходов, т.к. на бирже доход получается вероятностным путем, т.е. это статистический процесс, вероятностный.

Переменная выбора: то, какой портфель вложений будет: сколько акций и какой фирмы нам необходимо купить.

4. Составление плана оптимального производства (логистика)

> **Пример:** Есть завод, изготавливающий мебель. На заводе работает определенное количество работников, которые могут изготовить соответствующее кол-во определенной мебели (стулья, столы, шкафы и т.п.)

Целевая функция: максимизация прибыли.

Переменная выбора: выбор того, сколько необходимо изготовить стульев или столов, чтобы хватило рабочей силы.

5. Игры (вероятностные или не вероятностные)

> **Пример:** Участие в различных играх

Целевая функция: максимизация вероятности выигрыша или максимизация среднего выигрыша и т.д.

Переменная выбора здесь зависит от конкретной игры.

Задачи 1 — 5 — это примеры задач оптимизации.

Комбинаторика:

6. Графы и деревья

> **Пример:** Задача на решение того, сколько существует деревьев, у которых определенное число листьев; или сколько существует графов для решения такого-то задания и т.п.

7. Задача о размене монет или количество способов вернуть сдачу

> **Пример:** Есть монеты разного достоинства, какими способами можно вернуть сдачу.

Это краткое описание задач для динамического программирования, которые подробно будут рассмотрены позднее.

Понятие динамического программирования

Неформальное объяснение оптимальности подзадач ДП.

Рассмотрим **неформальную** идею ДП.

Итак, возьмем пример с заводом, изготавливающим мебель.

Для **достижения цели максимизации прибыли необходимо решить множество подзадач:**

- сколько стульев произвести — переменная `X1` ,
- сколько столов произвести — переменная `X2` ,
- сколько нанять работников — переменная `X3` ,
- … `Xn` .

При большом количестве подзадач сложно понять, как решать такую задачу. Как правило, **решить одну малую задачу проще, чем решить большую задачу**, состоящую из маленьких.

Поэтому ДП предлагает следующее:

- берем одну подзадачу с переменной `X1` , об остальных подзадачах пока забываем.

  Например, завод производит только стулья. У директора стоит задача получения максимальной прибыли с продажи стульев.

- После того, как найдем оптимальное решение для первой подзадачи, берем подзадачу для двух переменных `X1` и `X2` , и решаем ее **с помощью уже найденного решения для первой подзадачи**.
- Получаем решение уже для большей подзадачи, где фигурируют переменные `X1` и `X2` . Затем, используя полученное решение, берем подзадачи, охватывающие `X1` , `X2` и `X3` .

- И так продолжаем пока не получим решение для всей общей задачи.

> **Важно:** Ключевым моментом здесь является **использование выполненного решения** для малой подзадачи при решении очередного шага — большей подзадачи

ФОРМАЛЬНАЯ ИДЕЯ ДП

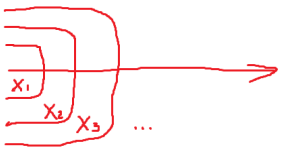
Часто при постановке задачи кажущимся оптимальным решением является **перебор всех возможных вариантов**. Однако, вследствии очень большого количества таких вариантов и, как результат, перегрузки памяти компьютера, такой способ не всегда приемлем.

Кроме того, может возникнуть такой вопрос: для того чтобы найти, например, минимум или максимум, почему бы нам не найти производную? или не использовать множества Ла-Гранжа, или другие методы аппарата математического анализа? Зачем нужно ДП, если есть большой арсенал средств?

Дело в том, что:

> В основе динамического программирования лежит идея решения поставленной задачи путем деления ее на отдельные части (подзадачи, этапы), решение этих подзадач и последующего объединения этих решений в одно общее решение. Часто большинство из подзадач абсолютно одинаковы.

При этом важно, что **при решении более сложной задачи, мы не решаем заново маленькую подзадачу, а используем уже решенный ответ этой подзадачи.**
На графике это может выглядеть так:



> **Важно:** По этой причине разделение задачи на подзадачи и **решение этих подзадач только один раз (!)**, сокращая этим количество общих вычислений — более оптимальный способ, который и заложен в динамическом программировании

Когда мы решаем задачу с производными, множествами Ла-Гранжа и т.п., то мы работаем с непрерывными функциями. При решении же задач ДП мы будем работать в основном с дискретными функциями, поэтому говорить здесь о применении непрерывных функций неуместно.
По этой причине во многих задачах, но не во всех, применение аппарата математического анализа будет неприемлемым.

Простой пример решения задач при помощи ДП

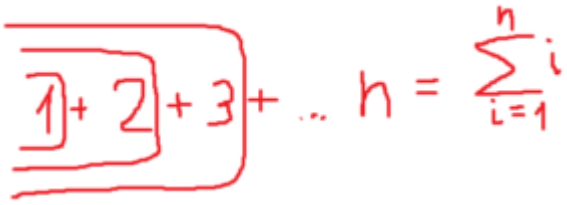
Рассмотрим вариант решения задачи с помощью динамического программирования.

Пример: Необходимо вычислить сумму `n` чисел: `1 + 2 + 3 + ... + n`

В чем состоит якобы «сложность» данной задачи: в том, что необходимо сразу взять большое количество чисел и получить ответ.

Попробуем применить к задаче идеи ДП и решить ее, разбивая на простые подзадачи.
(В ДП всегда необходимо сначала определить начальные условия или условие)

- Начнем с суммы одного первого элемента, т.е. просто берем первый элемент:
  `F(1) = 1`
- теперь с помощью решения для первого элемента, решим
  `F(2) = F(1) + 2 = 1 + 2 = 3`, т.е. надо взять сумму первого элемента и добавить к нему второй элемент
- `F(3) = F(2) + 3 = 6`
- по аналогии продолжаем и получаем целевую функцию:
  `F(n) = F(n-1) + An`

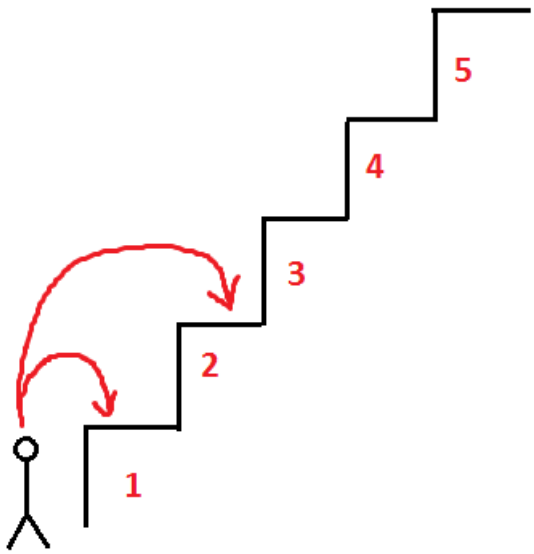


Итак, что мы сделали: определили порядок и вычленили подзадачи, затем решили каждую из них, опираясь на решение предыдущей.

Простой пример, где пока неоправданно используется ДП (искусственно), демонстрирует принцип идей ДП.

Рассмотрим еще один пример.

Пример: имеется лесенка из `n` ступенек, перед которой находится человек, который за **1** шаг умеет подниматься либо на следующую ступеньку, либо перепрыгивает через одну ступеньку. Вопрос: сколькими способами он может попасть на последнюю ступеньку?



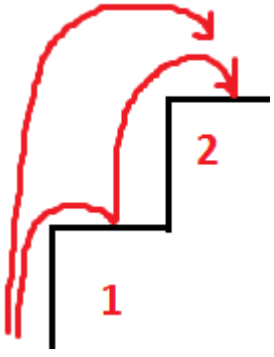
Решение:

Рассмотрим самые простые варианты (подзадачи):

1. Если лесница состоит из **1 ступеньки**:

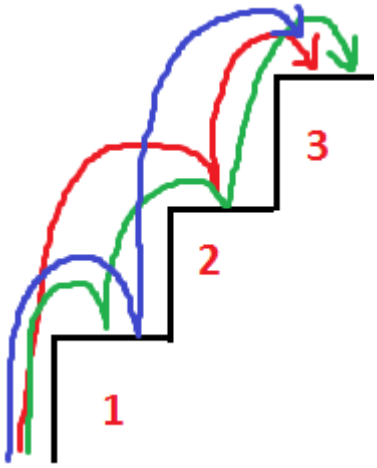
`f(a1) = 1`

2. из **2 ступенек**:



`f(a2) = 2`

3. из **3 ступенек**:



`f(a3) = 3` (1 — перешагнуть первую ступеньку, 2 — шагнуть на первую и перешагнуть вторую, 3 — прошагать все 3 ступеньки).

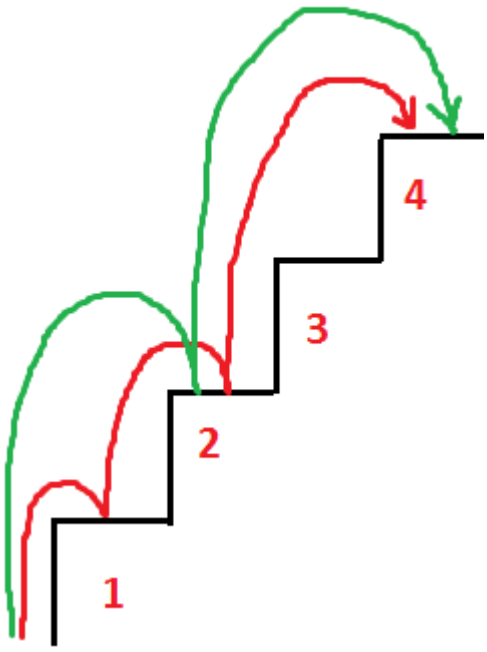
Рассмотрим пример из `i` ступенек

Как мы можем попасть на `i` ступеньку:

1. с `i-1` ступеньки, а на `i-1` ступеньку мы могли попасть a_{i-1} способами
2. с `i-2` ступеньки, а на `i-2` ступеньку мы могли попасть a_{i-2} способами

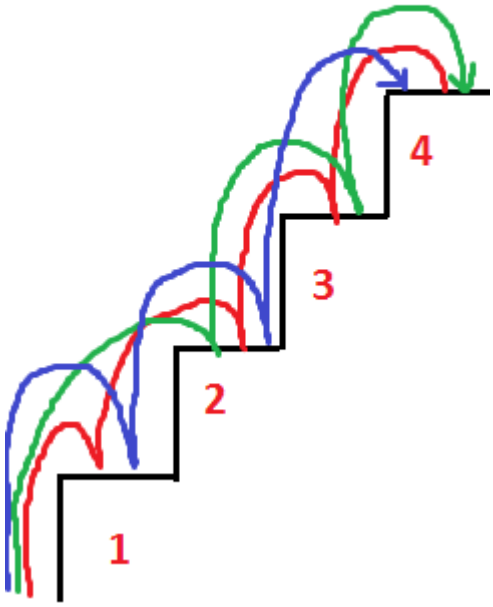
Например, как попасть на **4-ю ступеньку**:

1. У нас есть пример шагов **до второй ступеньки**, к каждому способу которого прибавляем полный шаг через две ступеньки.



т.е. $f(a_i) = f(a_i)-2$ …

2. У нас есть 3 способа попасть **на третью ступеньку**, если мы к каждому из них добавим маленький шажок на одну ступеньку выше, то получим 3 способа попасть на 4 ступеньку.



т.е. $f(a_i) = f(a_i)-1$

Т.о., **общее количество способов** попасть на `i` ступеньку:
$f(a_i) = f(a_{i-1}) + f(a_{i-2})$

Определим начальные значения, с которых следует начинать решать задачу.
Если начинать с 1, то формула соответствует нахождению последовательности чисел Фибоначчи.

$f(a1) = 1$

А можем посчитать и `a0` , начав с 0 , т.е. с нулевой ступеньки попасть на нулевую — такой способ 1, просто стоять на месте.

$a0 = 1$

Мы видим, что задача по сути комбинаторная (т.е. количество способов сделать что-либо) свелась к вычислению некоторой рекуррентной последовательности.

Задание 1: реализовать пример для первых десяти ступенек (по сути, первые 10 чисел ряда Фибоначчи), используя рекурсию.

Дополните код:

```
var c:integer;
  procedure getKolSposob(i,n: integer);
  begin
       writeln (i+n,' ');
       inc(c);
       if  ... then
           getKolSposob(...,...)
  end;

begin
  c:=1;
  getKolSposob(0,1);
end.
```

Задание 2:
Решение 15-го типа заданий ЕГЭ (Графы. Поиск количества путей).