Динамическое программирование - 1

Когда применять

- Когда надо либо подсчитать количество способов сделать что-то и формула не придумывается
- Когда надо получить оптимальный в некотором смысле способ сделать что-то

Задача о кузнечике

- Кузнечик стоит на прямой в точке 0, умеет прыгать только вправо на расстояние 1, 2 или 3
- Сколько способов допрыгать до точки N?
- Обозначим это $f(N)$
- Для первых точек ответы 1, 2, 4, 7
- Заметим, что $f(x) = f(x-1) + f(x-2) + f(x-3)$ для $x \geq 3$, а для $x < 3$ мы знаем ответы: 1,1,2
- Рекурсия будет работать очень долго уже для $N = 50$, т.к. мы посчитали каждый способ в явном виде, а они растут экспоненциально. (Можно доказать, что их примерно 1.83^N, значит мы сделаем $O(1.83^N)$ действия: очень много)

- Идея: можем не пересчитывать заново $f(x)$ если мы его уже считали
- Тогда можем написать (в dp_i будет храниться количество способов прийти в точку i):
```
dp[0]=1 # 1 способ стоять на месте
dp[1]=1 # 1 способ сделать шаг вправо
dp[2]=2 # Можем либо 1+1, либо сразу на 2
for i in range(3, n + 1):
  dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3]
```
- Делаем $O(N)$ действий
- Такой подход называется динамическим программированием

- Что делать с тем, что числа растут очень быстро, и становится *нехорошо* считать, что 1 действие с числом это 1 операция?
- В задачах обычно либо выбирают такое N, что это еще неважно (все влезает в 64-битный тип) или просят посчить по модулю какого-то числа. Это значит, что после каждой математической операции для результата надо взять остаток по данному модулю и числа останутся маленькими.

Задача о черепашке

- Есть поле $N \times M$, в каких-то клетках лежат монетки
- Черепашка стоит в левой верхней клетке и может двигаться вниз или вправо на 1 клетку
- Сколько максимум монеток она сможет собрать?
- Не хотим перебирать все пути
- Пусть, мы стоим в какой-то клетке и хотим найти оптимальный ответ именно для попадания туда
- Мы могли прийти в нее слева или сверху
- Тогда для клетки i, j можем сказать: `dp[i][j]=max(dp[i-1][j], dp[i][j-1]) + [Есть ли монетка в клетке i, j]`
- Ответ на задачу в `dp[N][M]`

Формализуем понятие ДП

- Чтобы решить задачу методом динамического программирования, нам нужно формально выделить следующие части задачи:
  - Состояние динамики (что в ней хранится?)
  - Базу динамики (начальные значения)
  - Переход динамики (формула пересчета)
  - Порядок обхода (в каком порядке вычислять dp?)
  - Где ответ на задачу
- Когда все 5 пунктов выше определены, можно приступать к написанию кода
- Стоит разобраться с ними для 2 предыдущих задач

Восстановление ответа

- Как получить в явном виде оптимальный маршрут черепашки?
- Храним не только `dp[i][j]`, но и `opt[i][j]` - "откуда мы сделали оптимальный переход?"
- Восстанавливаем с конца

Дополнительный параметр динамики

- Задача: посчитать число последовательностей длины N из 0 и 1 без двух 0 подряд.

- Как понять был до этого 0 или 1?
- Внесем это в состояние динамики!
- `dp[i][j], j=0/1` будет означать количество последовательностей длины i и последним символом j

Тогда все становится легче:

- `dp[i][0]=dp[i-1][1]`
- `dp[i][1]=dp[i-1][0]+dp[i-1][1]`
- Ответ в `dp[n][0]+dp[n][1]`

На самом деле для этой задачи от второго параметра можно избавиться, если в `dp[i]` хранить `dp[i][1]` из прошлого подхода, то есть строки длины i которые обязательно кончаются на 1:

- `dp[i] = dp[i-1] + dp[i-2]`: подходят все строки с прошлого шага (продлеваем 1), и с предпредыдущего (продлеваем их 01).
- Ответ хранится в `dp[n]+dp[n-1]`

Общий случай: без K нулей подряд решается аналогично:

- `dp[i] = dp[i-1] + dp[i-2] + ... + dp[i-k]`, здесь асимптотика $O(N \cdot K)$, но это можно улучшить.

Обратите внимание на начальные значения!