Динамическое программирование

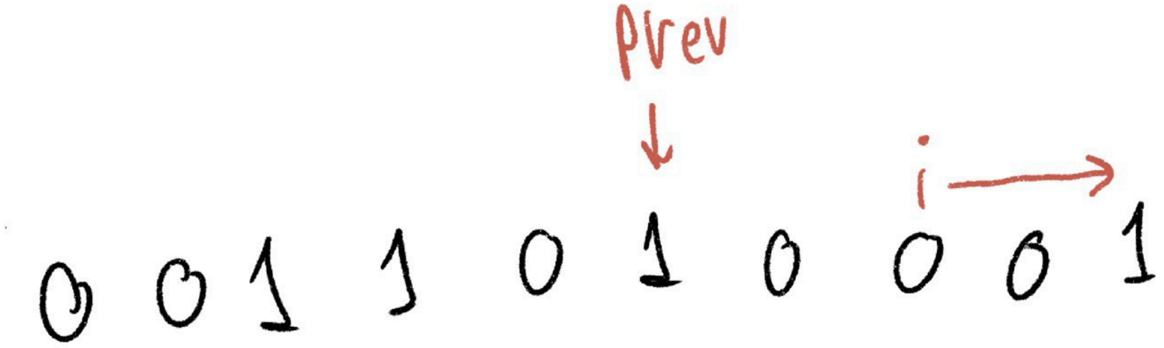
30/03/2024

Задача 0

Дан массив из нулей и единиц. Нужно для каждого нуля вывести ближайшую единицу слева и ближайшую единицу справа.

Вопрос: как такое решать?

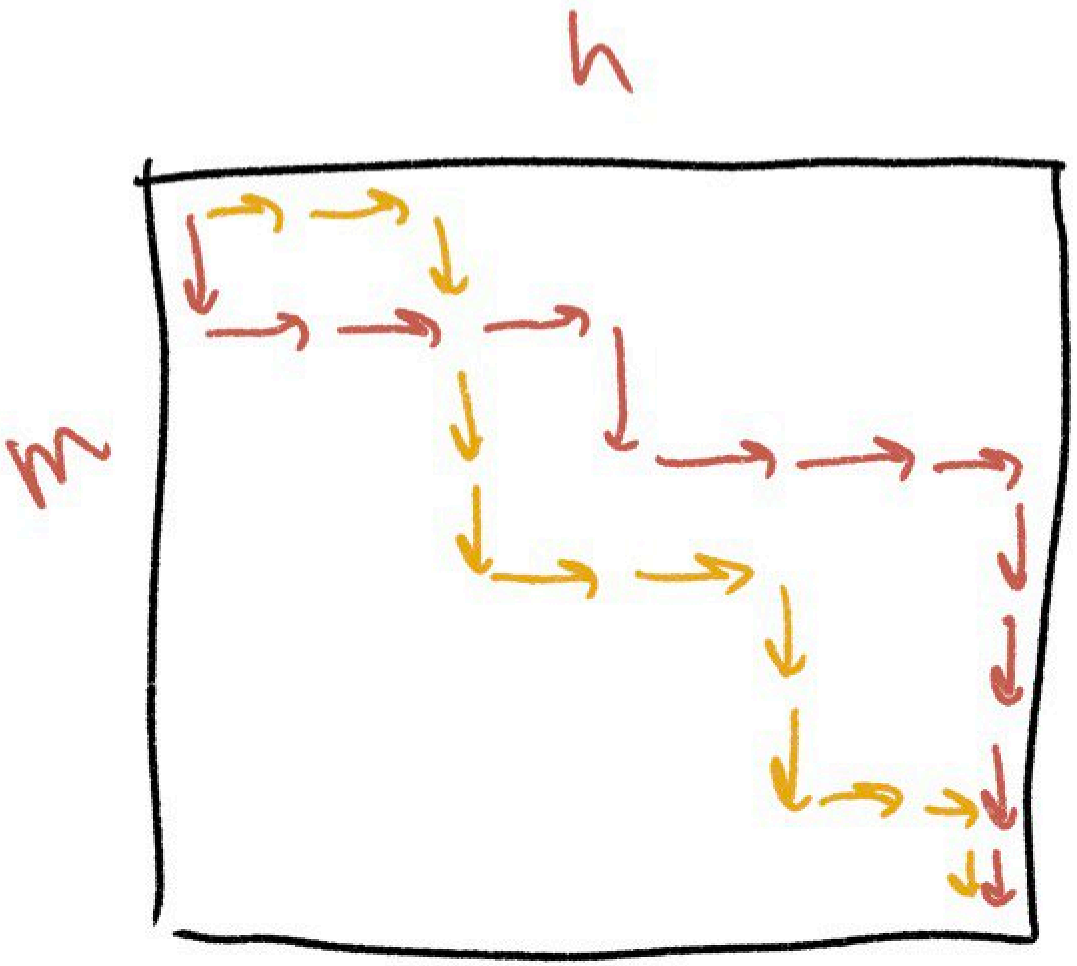
Ближайшая 1

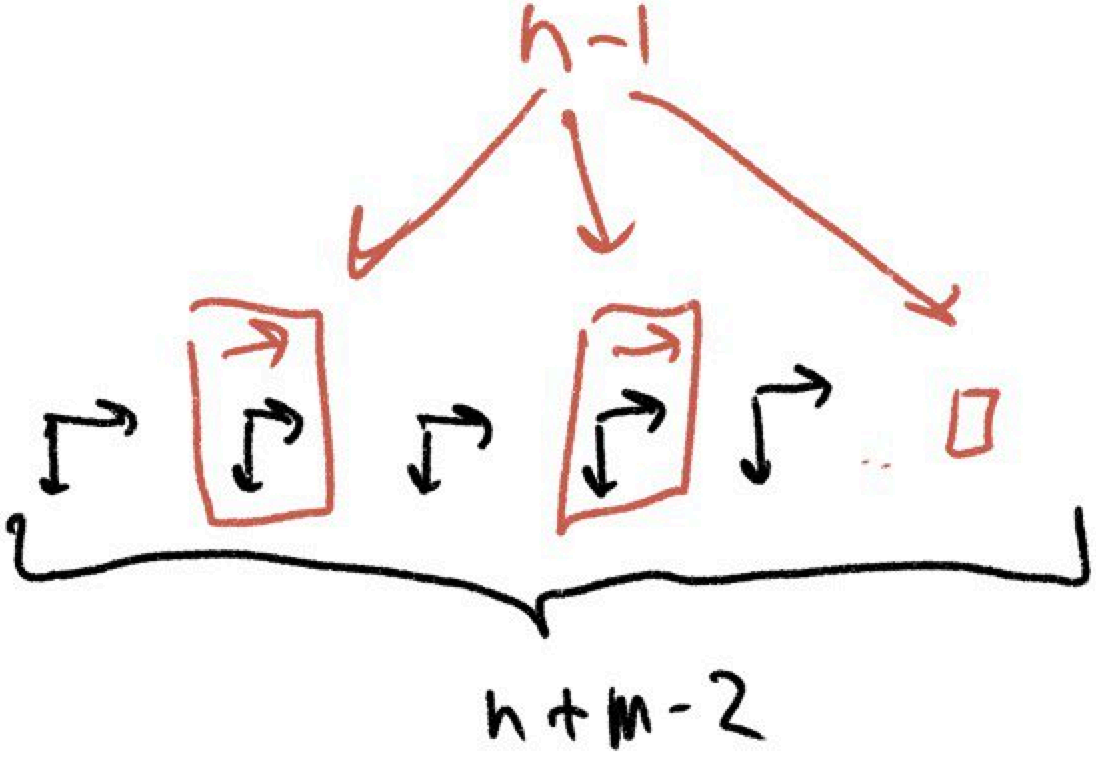


Задача 1

Дана таблица $n \times m$, некоторые клетки заблокированы. Нужно посчитать количество способов дойти от левой верхней клетки до правой нижней.

Вопрос: чему равен ответ, если нет заблокированных клеток?





$$\boxed{C^{n-1}_{n+m-2}}$$

Задача 2

Даны две строки, s и t, нужно понять, сколько минимально надо сделать удалений, добавлений или замен символов, чтобы из s получить t.

$$dist(\text{"kek"}, \text{"break"}) = 3$$

Это называется редакторским расстоянием

Пример неверного решения задачи 2

- Искать первую пару совпадающих символов

Контрпример: abaacd $\to$ bcdcd

"Хотелки" для организации вычислений

- Сохранение промежуточных результатов
- Удовлетворение (обычно) *локальным* ограничениям
- Получение *глобального* результата

На помощь приходит техника **динамического программирования**

Игрушечный пример

Посчитать число Фибоначчи. По определению, n-е число Фибоначчи называется f_n. (Позже мы значение с соответствующим индексом будем называть **состоянием**)

Мы имеем формулу **пересчета**:

$$f_n = f_{n-1} + f_{n-2}$$

Мы знаем какие-то значения заранее, это наша **база**:

$$f_0 = 0$$
$$f_1 = 1$$

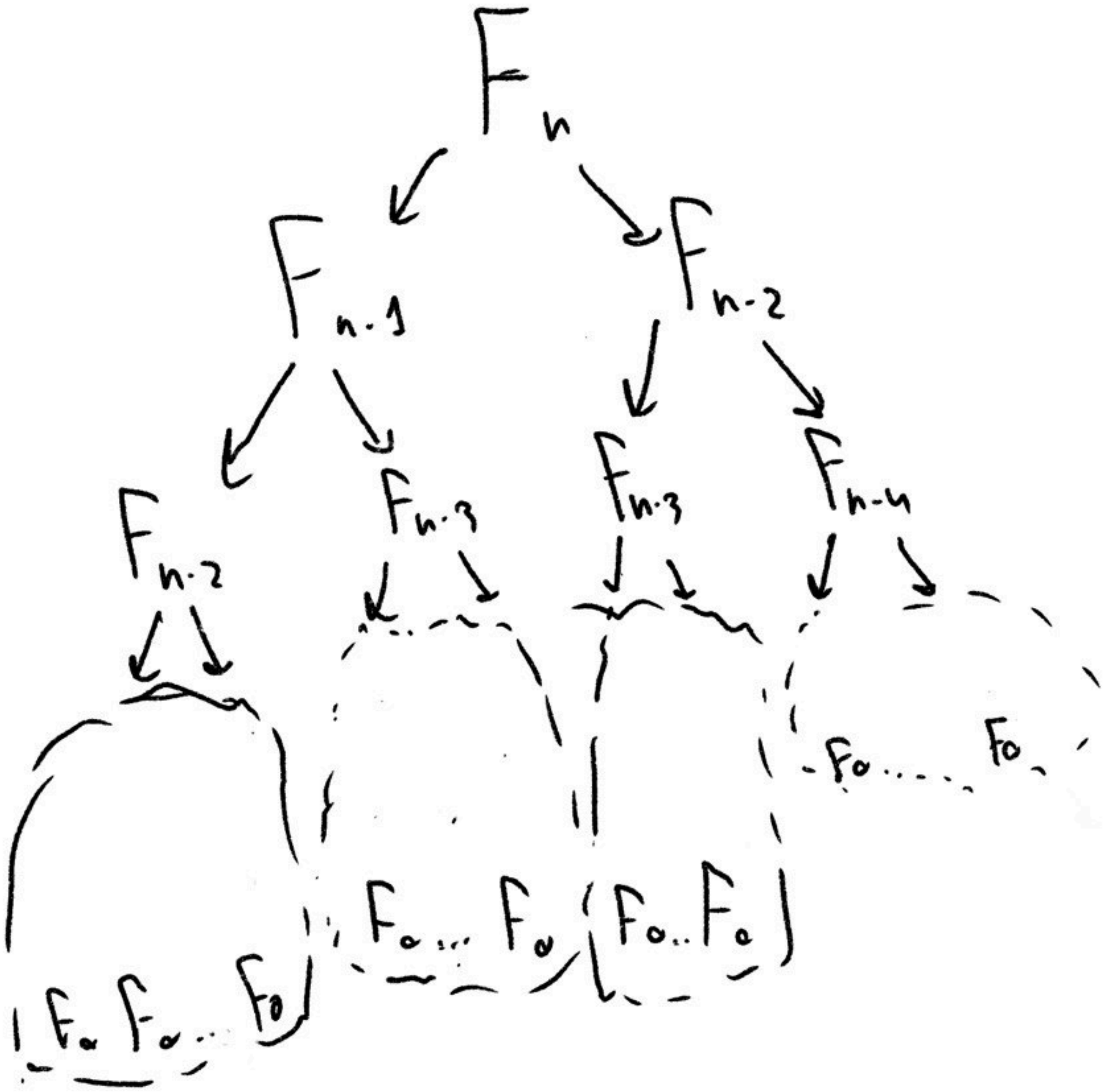
И мы знаем, что желаемое значение (**результат**) равно f_n.

Рекурсивные числа фибоначчи

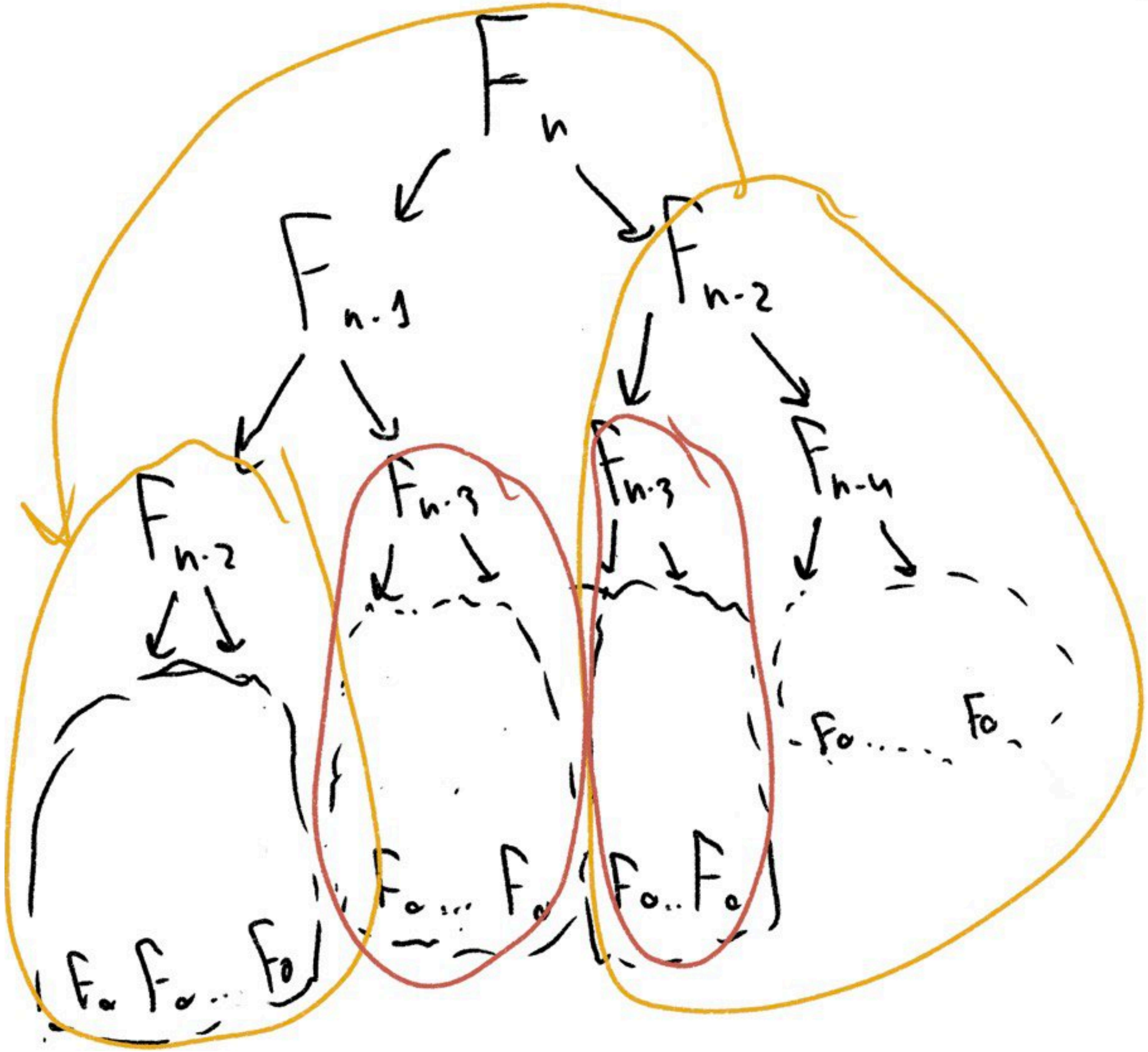
```
def f(n);
    if n <= 1:
        return n
    return f(n - 1) + f(n - 2)

print(f(20))
```

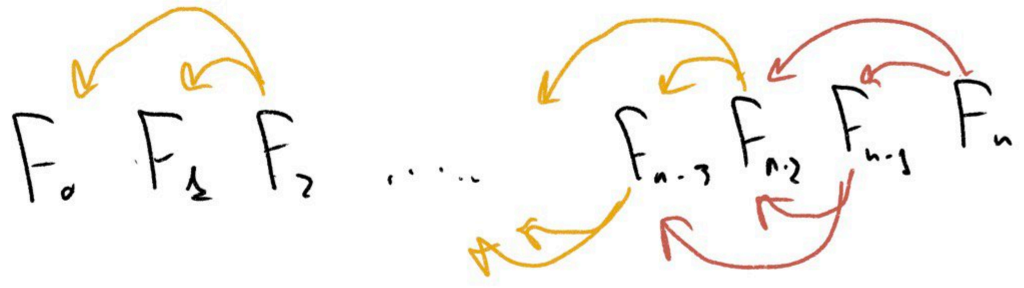
Порядок пересчета



Лишнее



Поменяем порядок пересчета



Апгрейд

```python
f = [i for i in range(21)]
for i in range(2, 21):
    f[i] = f[i - 1] + f[i - 2]
print(f[20])
```

Когда написали, проверяем:

- базу
- значение
- пересчет
- порядок пересчета
- результат

Как решать задачу методом ДП?

- Пытаемся найти *изолированное* состояние системы
- Смотрим, как оно зависит от других состояний
- Мысленно представляем, куда направлены стрелки для пересчета, и на их основе определяем порядок пересчета
- Определяем базу и понимаем, где будет лежать результат
- (*опционально*) Думаем, какие состояния можно забыть и выкинуть, чтобы не тратить память

Блишайшая единица

dp_i - ближайшая единица слева от i

$$dp_i = \begin{cases} dp_{i-1}, & a_i = 0 \\ i, & a_i = 1 \end{cases}$$

$$dp_0 = \begin{cases} \emptyset, & a_0 = 0 \\ 0, & a_0 = 1 \end{cases}$$

Пересчет слева направо (и справа налево для зеркальной задачи)

Число путей в табличке

$dp_{i,j}$ - количество путей до клетки (i, j) от левого верхнего угла

База - $dp_{-1,*} = dp_{*,-1} = 0$, $dp_{0,0} = 1$. Результат в $dp_{n-1,m-1}$.

$$dp_{i,j} = \begin{cases} 0, & a_{i,j} = locked \\ dp_{i,j-1} + dp_{i-1,j}, & else \end{cases}$$

Вопрос: Зачем нам такая большая база?

Число путей в табличке: порядок обхода

Какая разница между этими порядками обхода?

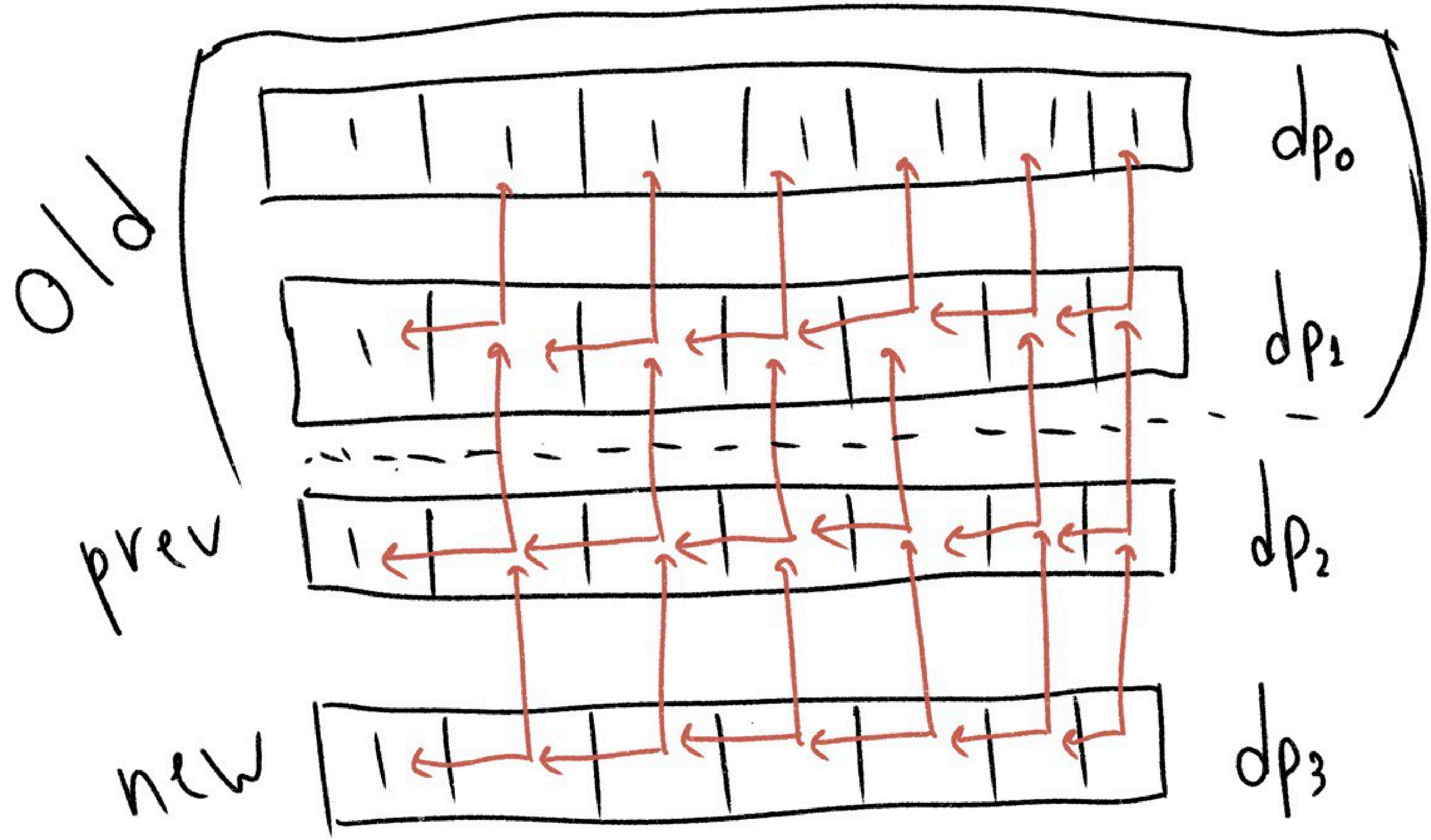
```python
for i in range(1, n):
    for j in range(1, m):
        dp[i][j] = dp[i][j-1] + dp[i-1][j]
        if a[i][j] is None:
            dp[i][j] = 0
```

```python
# swap for's
for j in range(1, m):
    for i in range(1, n):
        dp[i][j] = dp[i][j-1] + dp[i-1][j]
        if a[i][j] is None:
            dp[i][j] = 0
```

Число путей в табличке: жмем данные

На самом деле, мы можем забыть далекие слои

```
for i in range(1, n):
    if a[i][0] is None:
        dp_new[0] = 0
    for j in range(1, m):
        dp_new[j] = dp_new[j-1] + dp[j]
        if a[i][j] is None:
            dp_new[j] = 0
    # hack: no need to clear
    dp_new, dp = dp, dp_new
```


old
prev
new
dp_0
dp_1
dp_2
dp_3

Редакторское расстояние: плохой пример состояния

dp_i - сколько нужно сделать изменений, чтобы первые i символов у обеих строк совпали.

Одна из проблем - в зависимости от количества удалений, ответ может лежать как в dp_n, так и dp_0 (мы знаем про эту проблему даже без формул пересчета).

Редакторское расстояние: решение

$dp_{i,j}$ - сколько нужно сделать изменений, чтобы сделать равными подстроки $a[0:i]$ и $b[0:j]$, не трогая оставшиеся символы

У нас всегда есть три действия и одно бездействие:

1. изменить символ $a_i \to b_j$ (что эквивалентно обратной замене)
2. удалить символ a_i
3. удалить символ b_j
4. если $a_i = b_j$, то мы уже починили эти симвлоы

Редакторское расстояние: формула пересчета

Получаем переходы:

$$dp_{i,j} = \min \begin{pmatrix} dp_{i-1,j-1} + 1 & \\ dp_{i,j-1} + 1 & \\ dp_{i-1,j} + 1 & \\ dp_{i-1,j-1} & if\ a_i = b_j \end{pmatrix}$$

Задача о рюкзаке

Дано n предметов - пар веса и стоимости (w_i, c_i). Вы хотите уложить все в рюкзак с суммарным весом не больше W, при этом максимизировав суммарную стоимость.

Задача о рюкзаке: неверный жадный алгоритм

1. Отсортировать предметы по убыванию весов
2. Брать самый дорогой элемент, который пока что влезает в рюкзак

Задача о рюкзаке: состояние

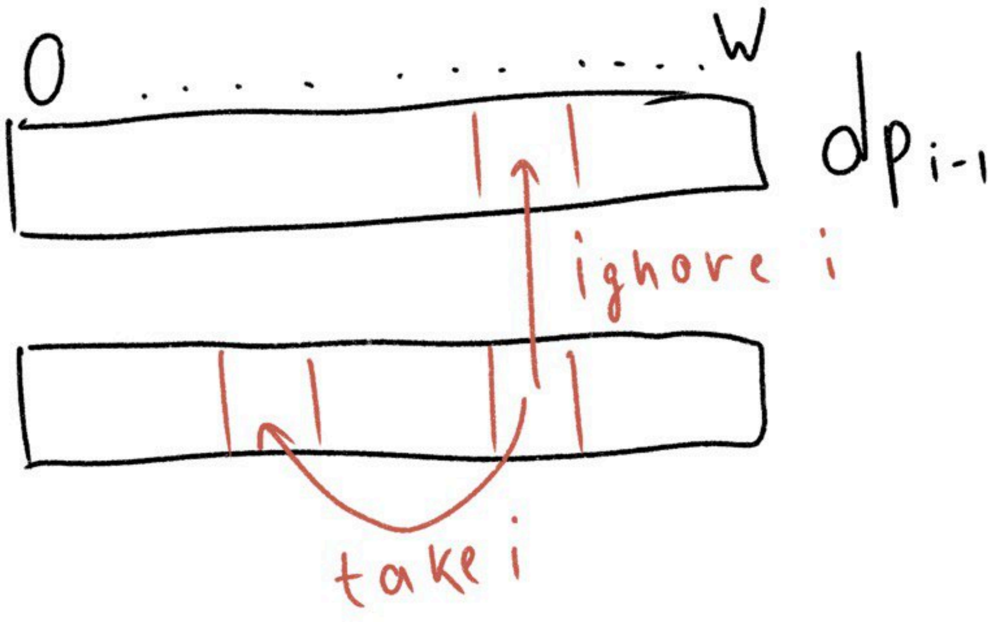
$dp_{i,w}$ - максимальная сумма набора, если мы рассмотрели первые i предметов и пока что набрали множество веса w.

$$dp_{i,w} = \max(dp_{i-1,w}, dp_{i-1,w-w_i} + c_i)$$

Можно хранить только два слоя в виде массивов dp_w, получить $O(W)$ памяти и $O(nW)$ времени.

Рюкзак: повтор предметов

Переходы в задаче о рюкзаке. Сравните, что будет, если перебирать красные клетки справа налево или слева направо.



Задача о рюкзаке: NP

Можно заметить, что если W большое, то данные в память не влезут. Оказывается, если ограничения на W нет, то дальше работают только переборы всех подмножеств за что-то типа $O(2^n n)$ или $O(2^{\frac{n}{2}} n)$ с разными оптимизациями.

Версия рюкзака вида "можно ли найти набор стоимости хотя бы T влезающий в рюкзак" является NP-полной задачей, потому что проверка решения работает за полиномиальное время, и потому что решение пока что существует только на "недетерминированной машине Тьюринга".

Вопрос, можно ли решить рюкзак в общем виде за полином, открыт, решение (или доказательство его отсутствия) стоит $1'000'000'000, а его практическая ценность еще выше ($P = NP$ problem).

Выделение состояния

Обычно сложнее всего выделить независимую подзадачу, которую нужно решать. Хорошо определенное состояние в задаче является независимым:

- Определено на каком-то подмножестве исходных данных
- Имеет иерархическую структуру
- Любые нюансы состояния "спрятаны" в дополнительные измерения
- Любые вспомогательные вычисления "спрятаны" в дополнительные динамики

Подпалиндромы

Задача: Найти количество подстрок-палиндромов.

Подпалиндромы

Можно решать динамикой по подотрезкам.

$dp_{l,r}$ - является ли палиндромом подстрока $s[l:r]$.

Тогда для одного символа мы знаем базу $dp_{i,i} = 1$, $dp_{i,i-1} = 1$, и хотим просуммировать все $dp_{l,r}$.

Подпалиндромы: переходы

$$dp_{l,r} = \begin{cases} dp_{l+1,r-1}, & s_l = s_r \\ 0, else \end{cases}$$

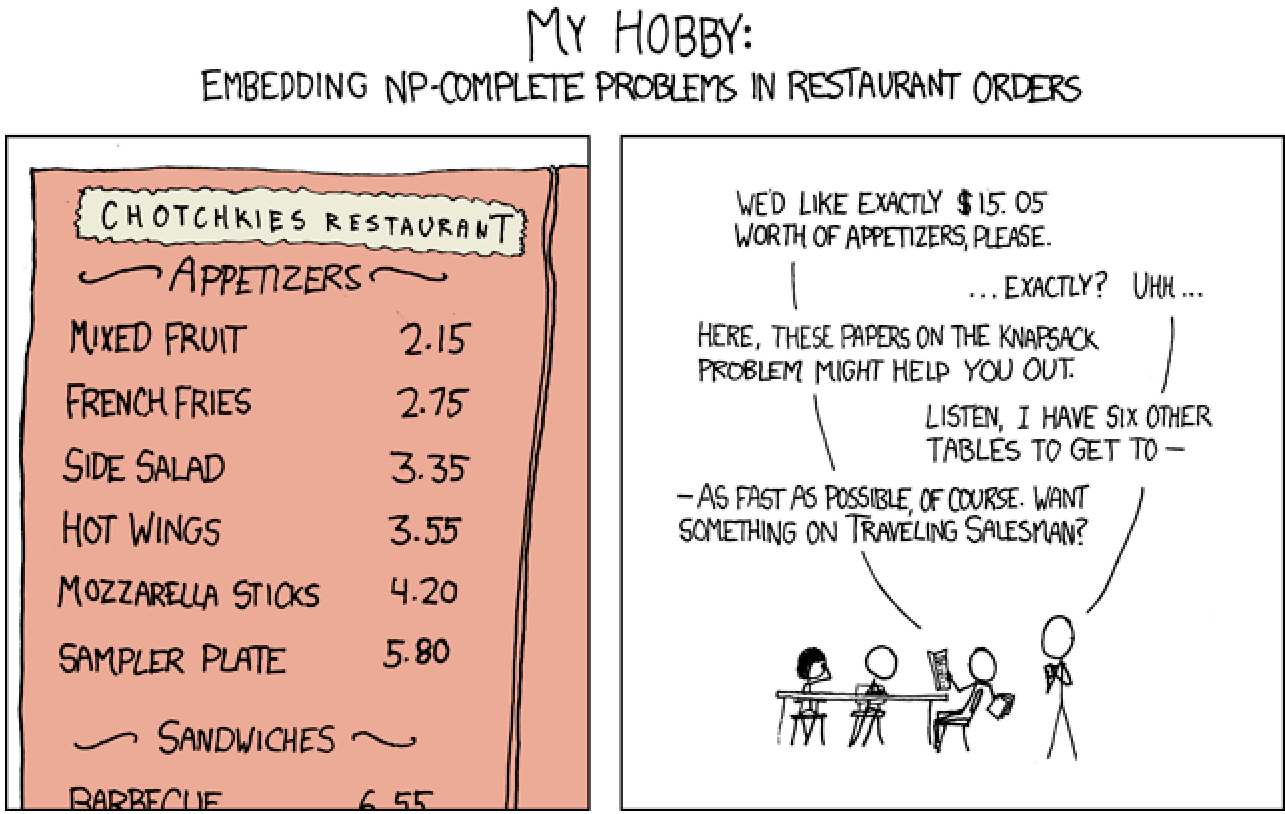
```
for r in range(n):
    for l in range(r, -1, -1):
        # ...
```

Почему такой порядок пересчета?

Другие (абстрактные) примеры:

- `dp[v]` -- поддеревья, пересчет через прямых потомков.
- `dp[l][r]` -- подстроки, пересчет через склеивание двух подстрок меньшего размера
- `dp[mask]` -- подмножества, пересчет через под-подмножества.
- `dp[i][min/max]` -- две динамики, которые зависят друг от друга (например, если можно домножать предыдущее состояние на -1).
- `dp[i][flag]` -- флаг доопределяет состояние (например, показывает состояние кузнечика, если он сначала прыгает только на A, а потом -- только на B)
- `dp[l][r][min/max][flag]` -- продвинутые задачи

Мем


MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS
CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO —
— AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

Для дальнейшего изучения

- Дп по подмаскам
- Дп по поддеревьям
- Дп на ациклических графах
- NP-полные задачи, TSP, SAT, итд
- И решать задачи!!!

QR для фидбека