ДП-2, рекурсия и переборы

18/03/2023

Еще немного про ДП: в прошлый раз

Для некоторых вычислений можно определять состояние, переходы, порядок пересчета, базу, результат.

Так можно делать разные вычисления

Интерпретация результатов ДП: редакторское расстояние

$dp_{n,m}$ - количество действий для того, чтобы перевести s в t.

Но как понять, какая последовательность действий нам нужна?

Задача оптимизации: восстановление ответа

Уже знаем:

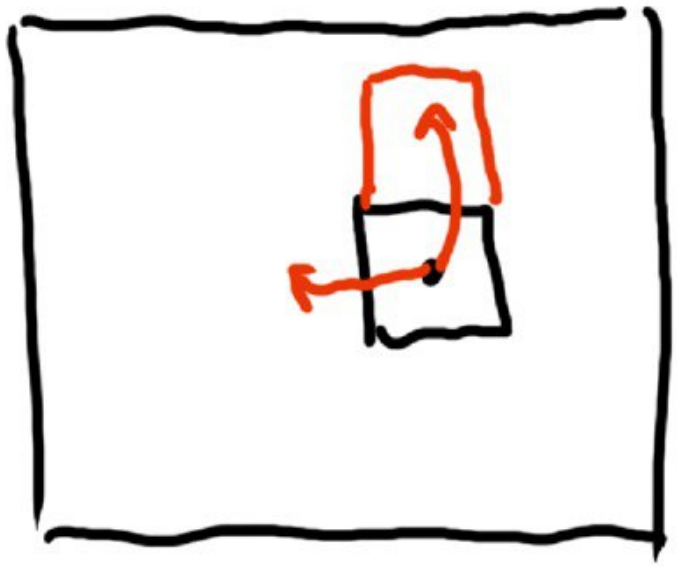
$dp_{i,j}$ - минимальное количество действий, чтобы ...

$$dp_{i,j} = \min(\dots)$$

Хотим:

$$pre_{i,j} = argmin(\dots)$$

Иначе говоря, $pre_{i,j}$ определяет, какое прошлое действие мы сделали, на основе того, какое значение для пересчета dp было минимальным



$dp_{i,j} = \min$ [□] $+1$

$pre_{i,j} =$ opt ↱

Вычисление предков: пример

```
for i in range(n):
    for j in range(m):
        dp[i][j] = min(dp[i-1][j] + 1, dp[i][j-1] + 1, dp[i-1][j-1] + 1)
        if dp[i][j] == dp[i-1][j] + 1:
            pre[i][j] = ERASE_0
        # ...
        if a[i] == b[j]:
            dp[i][j] = min(dp[i][j], dp[i-1][j-1])
            if dp[i][j] == dp[i-1][j-1]:
                pre[i][j] = SIMILAR
```

Восстановление ответа: пример

```
result = (n-1, m-1)
actions = []
while result != (0, 0):
    action = pre[result[0], result[1]]
    actions.append(action)
    if action == ERASE_0:
        result = (result[0] - 1, result[1])
    # ...
    if action == SIMILAR:
        result = (result[0] - 1, result[1] - 1)
actions = actions [::-1]
```

Рекурсия

Функция называется рекурсивной, если ее определение содержит вызов этой же функции.

Минимальный (и бессмысленный) пример

```
def f(args):
    f(args)
```

Пример №1

```
def f(n)
    if n == 0:
        return
    print(n)
    f(n - 1)
```

```
>>> f(3)
3
2
1
```

Пример № 2

```
def f(n)
    if n == 0:
        return
    f(n - 1)
    print(n)
```

```
>>> f(3)
1
2
3
```

Пример № 3 (из ЕГЭ)

```
def f(n)
    if n == 0:
        return
    if n % 2 == 1:
        f(n - 1)
    else:
        f(n // 2)
    print(n)
```

```
>>> f(6)
1
2
3
6
```

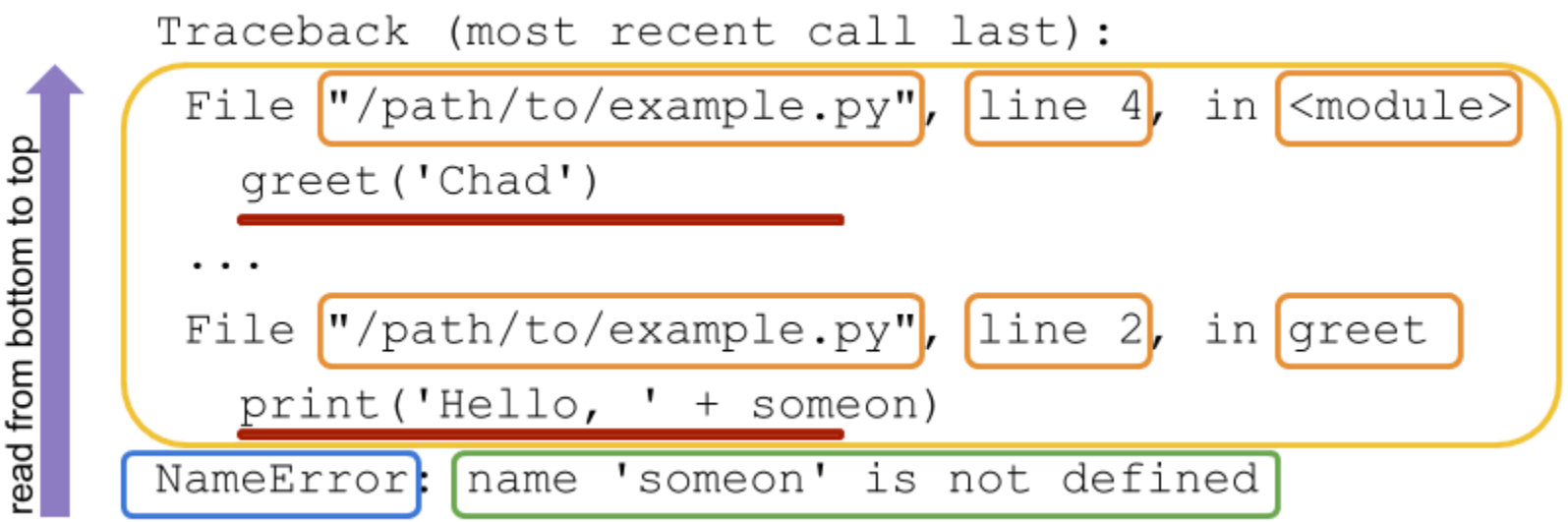
Вызовы

Основная сложность при работе с рекурсией в том, что ваш исполнитель находится в нескольких местах одного и того же кода "одновременно", на разных слоях

Пример №3 (еще раз)

```
def f(n) # 6, 3, 2, 1, 0
    if n == 0:
        return # 0
    if n % 2 == 1:
        f(n - 1) # 3, 1
    else:
        f(n // 2) # 6, 2
    print(n)
```

Стектрейс


Traceback (most recent call last):
File "/path/to/example.py", line 4, in <module>
greet('Chad')
...
File "/path/to/example.py", line 2, in greet
print('Hello, ' + someon)
NameError: name 'someon' is not defined
read from bottom to top

Инварианты

Обычно про рекурсивные алгоритмы стоит думать в формате их определений.

А именно, сначала мы хотим поставить себе задачу, потом сделать какие-то операции и, возможно, свести задачу к задаче поменьше.

Рекурсивный бинпоиск (wat?)

```
def search(a, l, r, x):
    if l + 1 == r:
        return r
    m = (l + r) // 2
    if a[m] < x:
        return search(a, m, r, x)
    else:
        return search(a, l, m, x)
```

Вопрос: почему так не пишут?

Разворот рекурсии

Рекурсия это примерно то же самое, что и цикл + стек.

Плюс:

- Часто понятнее и нагляднее, функция соответствует определению

Минусы:

- Мы зачем-то пользуемся стеком

Переборные алгоритмы

Обычно мы хотим "Перебрать все элементы хитрого множества и":

- ...сделать среди них поиск какого-то особенного
- ...найти элемент, оптимизирующий ответ
- ...собрать с них общую статистику

Хитрое множество

Обычно, хитрое множество - это либо множество, либо мультимножество, либо последовательность чисел, на которое накладываются ограничения

Строки, кстати, это тоже числа - просто "abacaba" = [0, 1, 0, 2, 0, 1, 0]

Строить такие множества проще всего поэлементно

Пример: перестановка

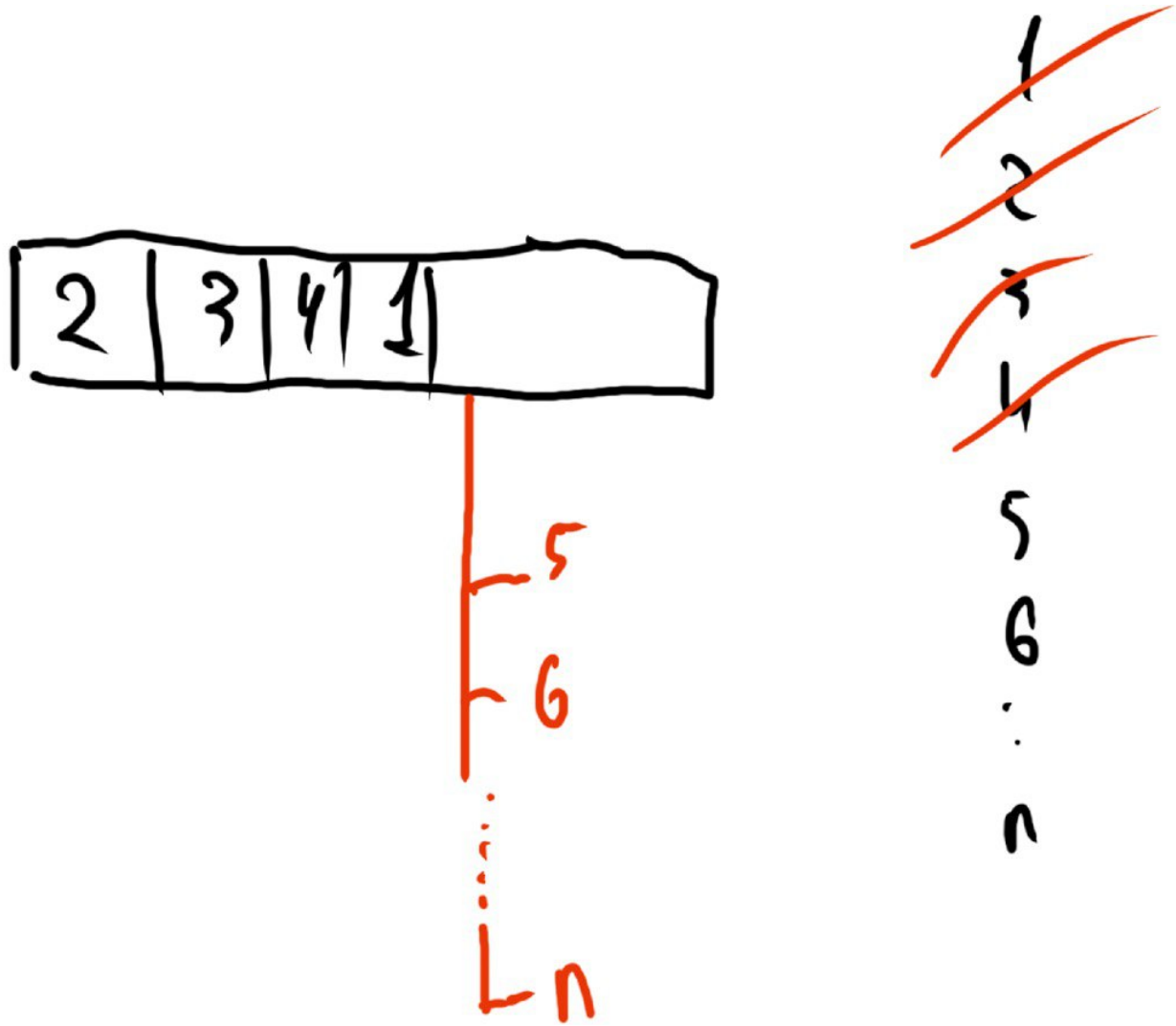
Имея начало перестановки, мы можем добавить в нее элемент, если его не было раньше

```
def gen_perm(n, cur=0, prefix=[]):
    if cur == n:
        return prefix
    next_elem = random.choice(range(n))
    while next_elem in prefix:
        next_elem = random.choice(range(n))
    prefix.append(next_elem)
    return gen_perm(n, cur + 1, prefix)
```

Пример: перебор всех перестановок

```
def gen_perm(callback, n, cur, prefix):
    if cur == n:
        callback(prefix)
        return
    for i in range(n):
        if i not in prefix:
            prefix.append(i)
            gen_perm(callback, n, cur + 1, prefix)
            prefix.pop()
```

```
>>> gen_perm(print, 3, 0, [])
[0, 1, 2]
[0, 2, 1]
[1, 0, 2]
[1, 2, 0]
[2, 0, 1]
[2, 1, 0]
```


2
3
4
1
5
6
Ln
1
2
3
4
5
6
n

Отсечения

Если вы можете заранее понять, что идти перебирать дальше нет смысла - не перебирайте

Вопрос: Есть ли в этом коде бесполезные ветки?

```
def gen_choose_k(callback, prefix, n, k, cur):
    if cur == n:
        if len(prefix) == k:
            callback(prefix)
        return
    gen_choose_k(callback, prefix, n, k, cur + 1)
    gen_choose_k(callback, prefix + [cur], n, k, cur + 1)
```

Отсечения

Если вы можете заранее понять, что идти перебирать дальше нет смысла - не перебирайте

```
def gen_choose_k(callback, prefix, n, k,  cur):
    if len(prefix) > k:
        return
    if k - len(prefix) > n - cur:
        return
    if cur == n:
        if len(prefix) == k:
            callback(prefix)
        return
    gen_choose_k(callback, prefix, n, k, cur + 1)
    gen_choose_k(callback, prefix + [cur], n, k, cur + 1)
```

Такой код можно превратить в решение рюкзака без ограничений на веса.

Байка: гиперпараметры

```
params = {
    'ORB_FEATURES': [400, 450, 500],
    'FILTER_RELATIVE_THRESHOLD': [0.4, 0.45, 0.5],
    'FILTER_ABSOLUTE_THRESHOLD': [1.1],
    'POINTS_PREDICTIONS': [1],
}

def run(params):
    # ...
    return

def generate_configuration(param_names, i=0, selection={}):
    if i == len(params):
        run(selection)
        return
    for option in params[param_names[i]]:
        selection[param_names[i]] = option
        generate_configuration(param_names, i + 1, selection)
        selection.pop(param_names[i])

generate_configuration(list(params))
```

Нахождение K-й перестановки: опять динамика!

Восстановление ответа для динамики на количество не имеет смысла, зато ей можно находить k-й объект. Для этого нужно отсекать ветки перебора, которые нам не нужны.

```
def get_k(n, k, used=set()):
    if len(used) == n:
        return
    for i in range(n):
        if i+1 in used:
            continue
        if k > fact[n-len(used)]:
            k -= fact[n-len(used)]
        else:
            used.insert(i+1)
            print(i+1)
            get_k(n,k,used)
            return
```

Дп по подмножествам: рюкзак

$dp_{subset,i}$ - максимальная стоимость рюкзака, если мы изучили i предметов и сложили туда подножество $subset$.

Можно заметить две вещи:

- Можно не хранить измерение для i в памяти
- $dp_{subset,i} = \sum_{item \in subset} cost(item) = sum_{subset}$

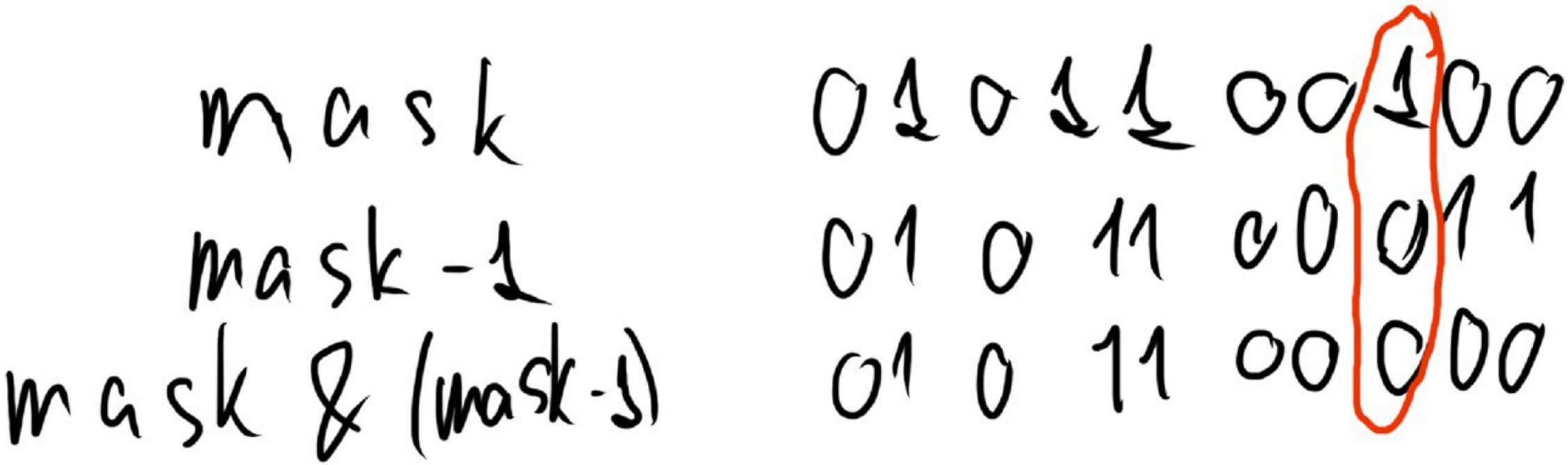
Бинарная магия

`subset` можно хранить как двоичную маску того, есть ли число в множестве.

Например, число $11_{10} = 1011_2$ соответствует тому, что мы взяли предметы 0, 1, 3.

Маска занимает $O(\frac{n}{\omega})$ памяти. С помощью бинарных операций можно, например, удалить из маски последнюю единицу, чтобы сделать пересчет в ДП. Для задач, где $n \leq 30$ маска влезает в одно число, и операции с ней очень быстрые.

$$sum_{mask} = sum_{mask \wedge (mask-1)} + a[\log 2[mask - (mask \ \wedge \ (mask-1))]]$$


mask
mask - 1
mask & (mask - 1)
01011100
01011011
01011000

Метод ветвей и границ

Задача

- Шахматы.

Цель

- Выиграть.

Модель игры

Игроки (A, B) делают ходы по очереди. Каждый из игроков меняет состояние игры S и передает ход другому игроку.

Мы скажем, что $f(S)$ - это "выигрышность" состояния. Чем оптимальность больше, тем лучше. Можно считать, что мы распределяем одну единицу выигрыша между игроками.

$$f(\text{win}) = 1$$
$$f(\text{loose}) = 0$$

Как пересчитать выигрышность состояния?

Мы предполагаем, что оба игрока играют оптимально. Значит, мы считаем, что после нашего хода противник сделает самый плохой ход для нас.

$$f(S) = \max_{white\ move}\left(\min_{black\ move} f(S + w + b)\right)$$

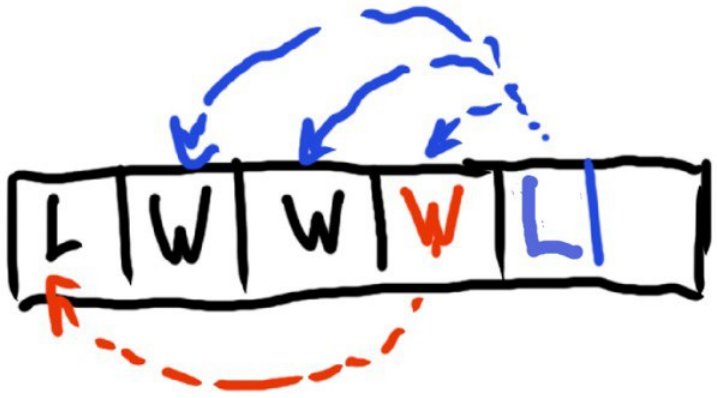
Симметричная игра

Если игра симметрична, то можно переписать еще проще

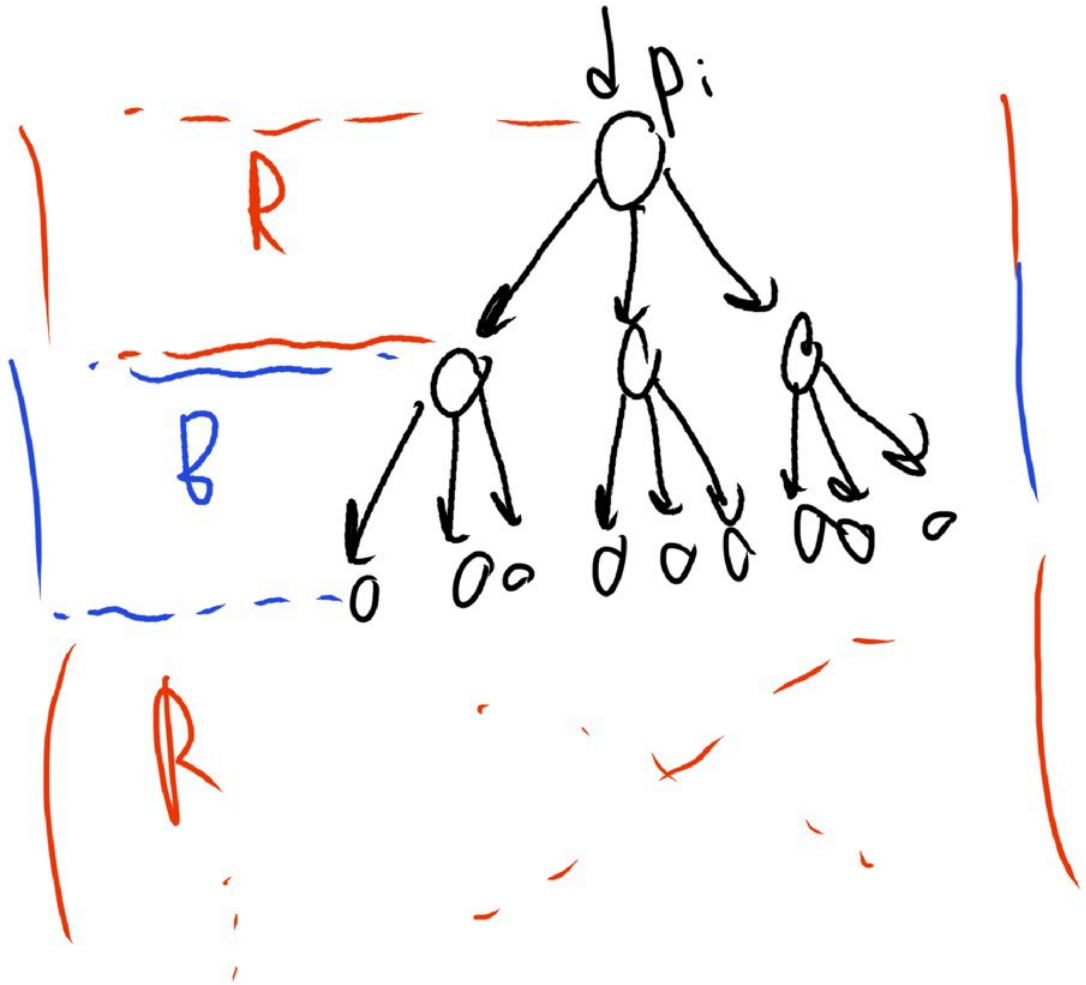
$$f(S) = 1 - \min_{move} f(S + move)$$

$$X \rightarrow \{x-1, x-2, x-3\}$$

Симметричная игра



Игра для двух игроков



Отсечение: идея

Будем "смотреть" вперед на два шага.

Пусть мы уже посчитали, что один из возможных ходов w_0 дает нам в будущем выигрыш f_0.

Попробуем другой ход w_1. Если на него у противника есть хотя бы один ответный ход w_2 такой, что он приведет нас в состояние с выигрышем $f_1 < f_0$, нам ход w_1 точно не интересен.

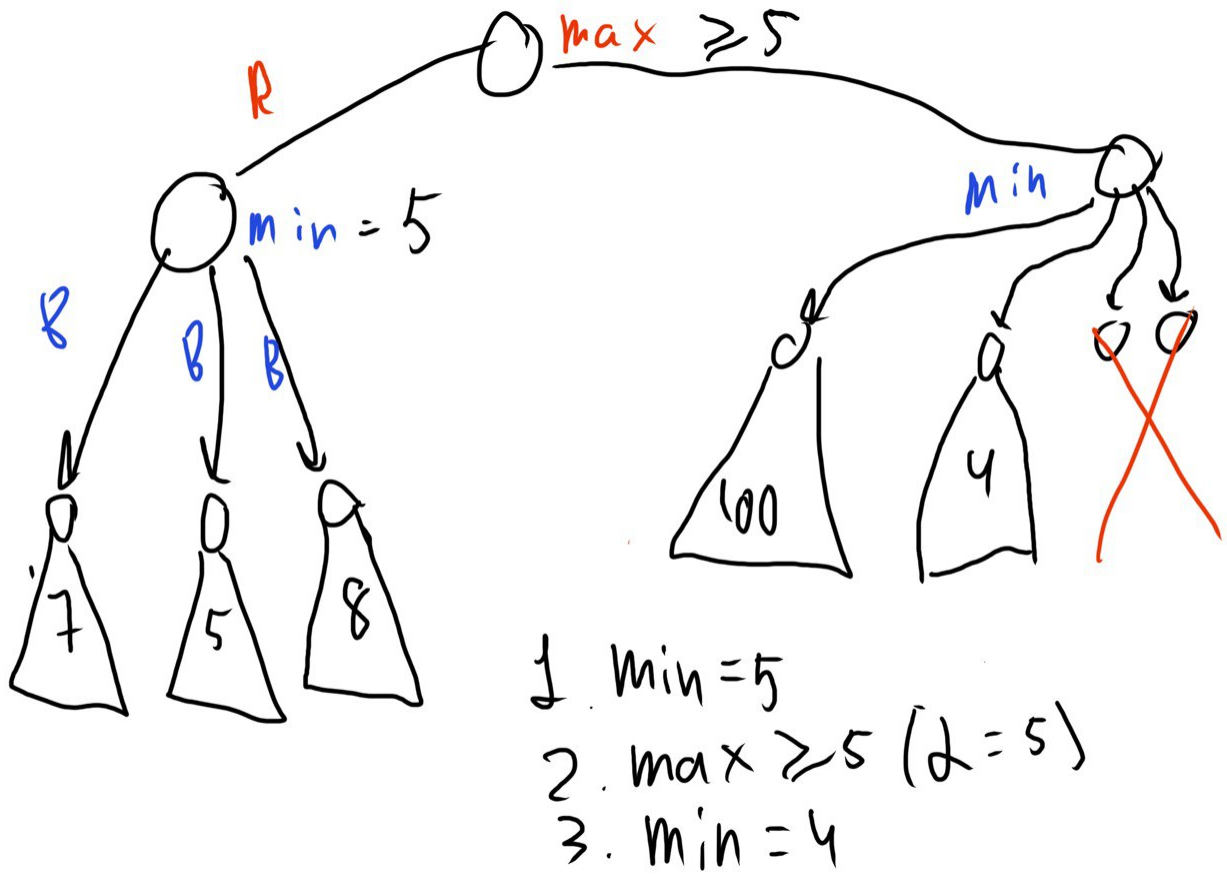
Значит, можно не рассматривать оставшиеся варианты ходов, смежные с w_2

Инвариант

Имеем α - минимальный выигрыш, который мы можем себе гарантировать и β - максимальный выигрыш, который мы можем себе гарантировать.

Если мы в ситуации, где $\alpha \geq \beta$, то состояние нет смысла изучать дальше: даже если мы найдем более оптимальный ход, мы все равно не воспользуемся им.

Пример



Реализация

```
def ab_pruning(state, alpha, beta, player, steps):
    if steps == 0:
        return f(state)
    if player == 2:
        for move in state.moves():
            beta = min(beta,
                       ab-pruning(state + move, alpha, beta,
                                  1 - player, steps - 1))
            if beta <= alpha:
                return beta
        return beta
    else:
        # ...
            alpha = max(alpha,
                       ab-pruning(state + move, alpha, beta,
                                  1 - player, steps - 1))
        # ...
```

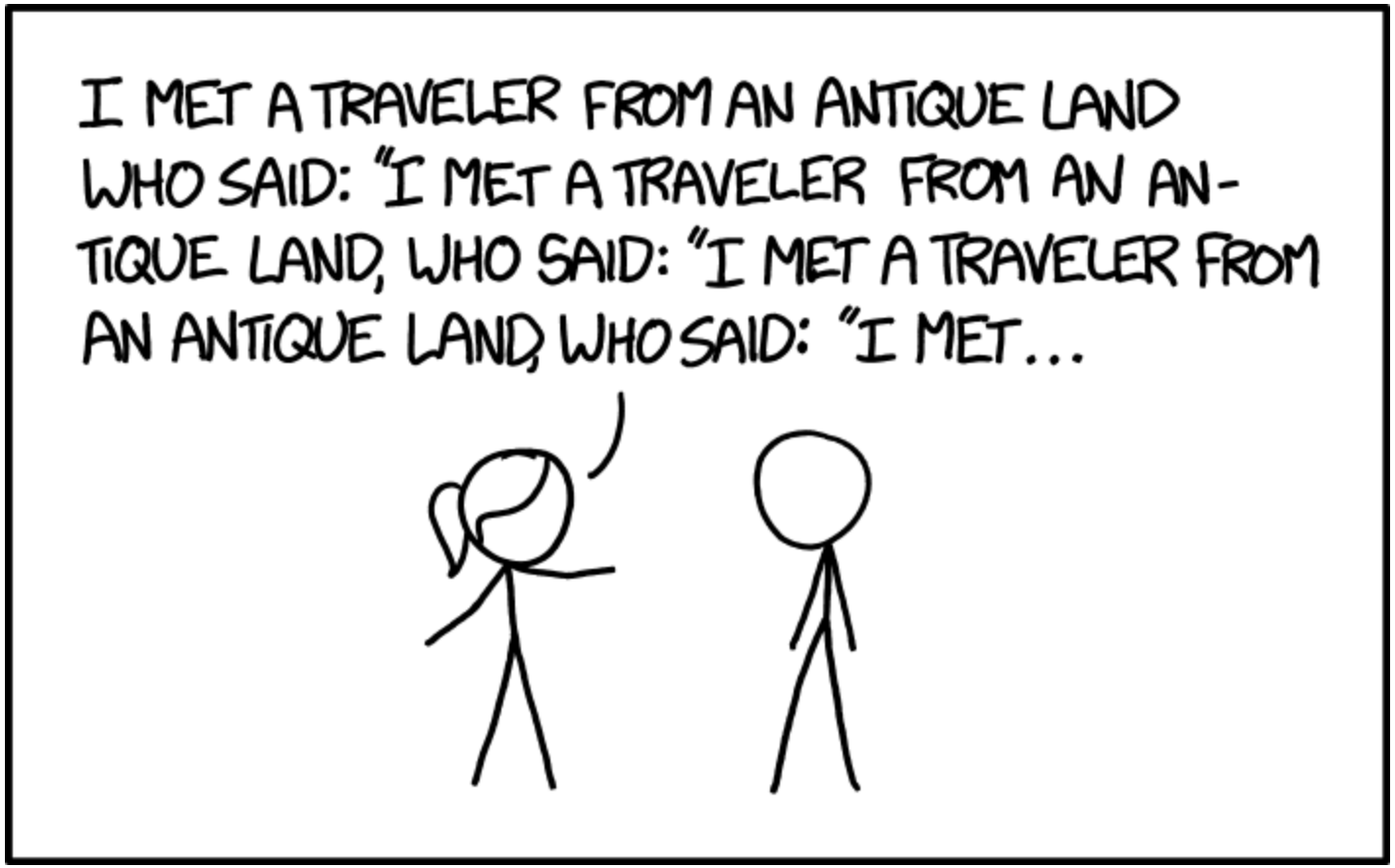
Пример оценочной функции

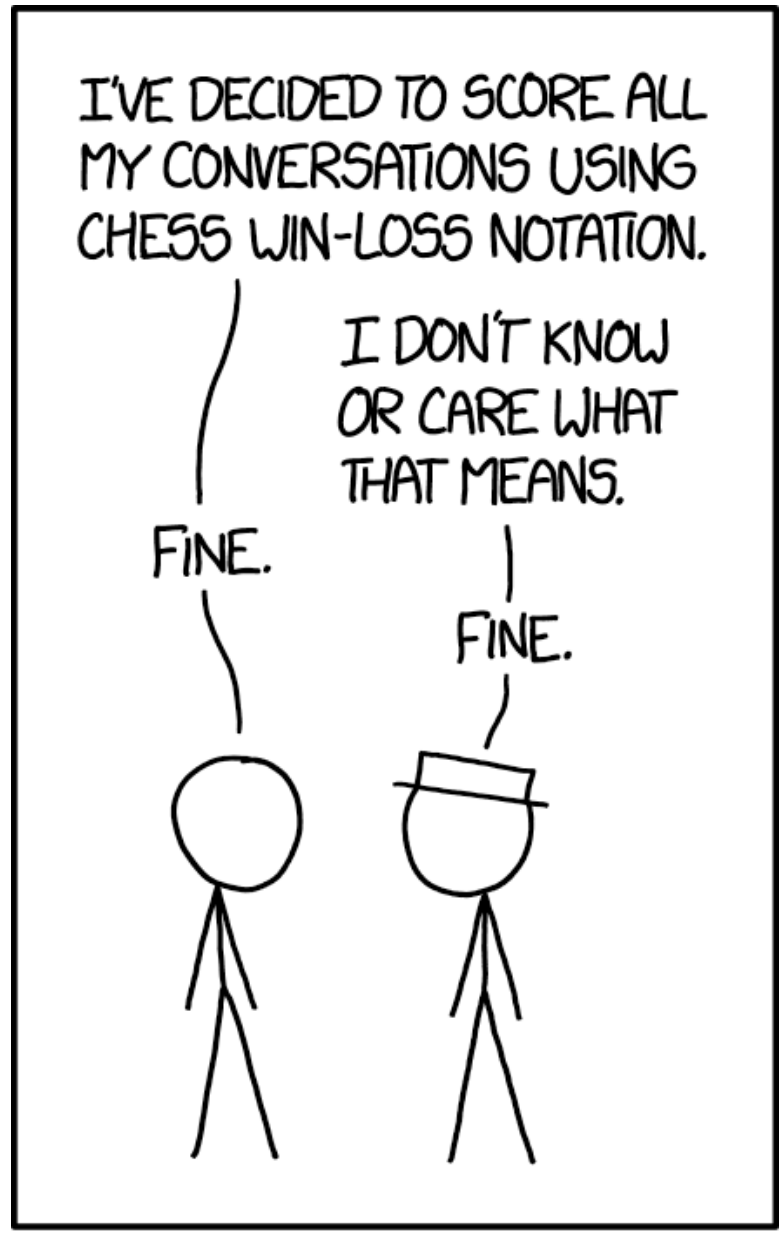
```
def f(state):
    if state.checkmate_white():
        return 1
    if state.checkmate_black():
        return 0
    cost_white = 0
    cost_black = 0
    for piece in state.pieces():
        if piece.white:
            cost_white += piece.cost
        else:
            cost_black += piece.cost
    return cost_white / (cost_white + cost_black)
```

Пет-проджет

Шахматы :)

Мемы!


I MET A TRAVELER FROM AN ANTIQUE LAND
WHO SAID: "I MET A TRAVELER FROM AN AN-
TIQUE LAND, WHO SAID: "I MET A TRAVELER FROM
AN ANTIQUE LAND, WHO SAID: "I MET...



$\frac{1}{2} - \frac{1}{2}$

Для дальнейшего изучения

- Коммивояжер, рюкзак, итд
- https://habr.com/ru/company/timeweb/blog/533642/
- https://habr.com/ru/post/560468/
- Фракталы