ЛЁГКИЙ МЕТОД РЕШЕНИЯ КУЧ PYTHON

```
def gameResult(x, y):
    if (x, y) in results: return results[(x, y)]  # (2)
    if gameOver(x, y): return 0
    nextCodes = [gameResult(x + KADD, y), gameResult(x * KMUL, y),
                 gameResult(x, y + KADD), gameResult(x, y * KMUL)]
    negative = [c for c in nextCodes if c <= 0]
    if negative:
        res = -max(negative) + 1
    else:
        res = -max(nextCodes)
    results[(x, y)] = res  # (3)
    return res


from math import ceil

ans1 = min(ceil((TARGET - N1) / KMUL / KMUL), ceil(TARGET - N1 * KMUL * KMUL))
ans2 = []
ans3 = []
for S in range(TARGET - N1 - 1, 0, -1):
    r = gameResult(N1, S)
    print("%d: %d" % (S, r))
    if r == 2: ans2.append(S)
    if r == -2: ans3.append(S)
```

Как делать НЕ надо

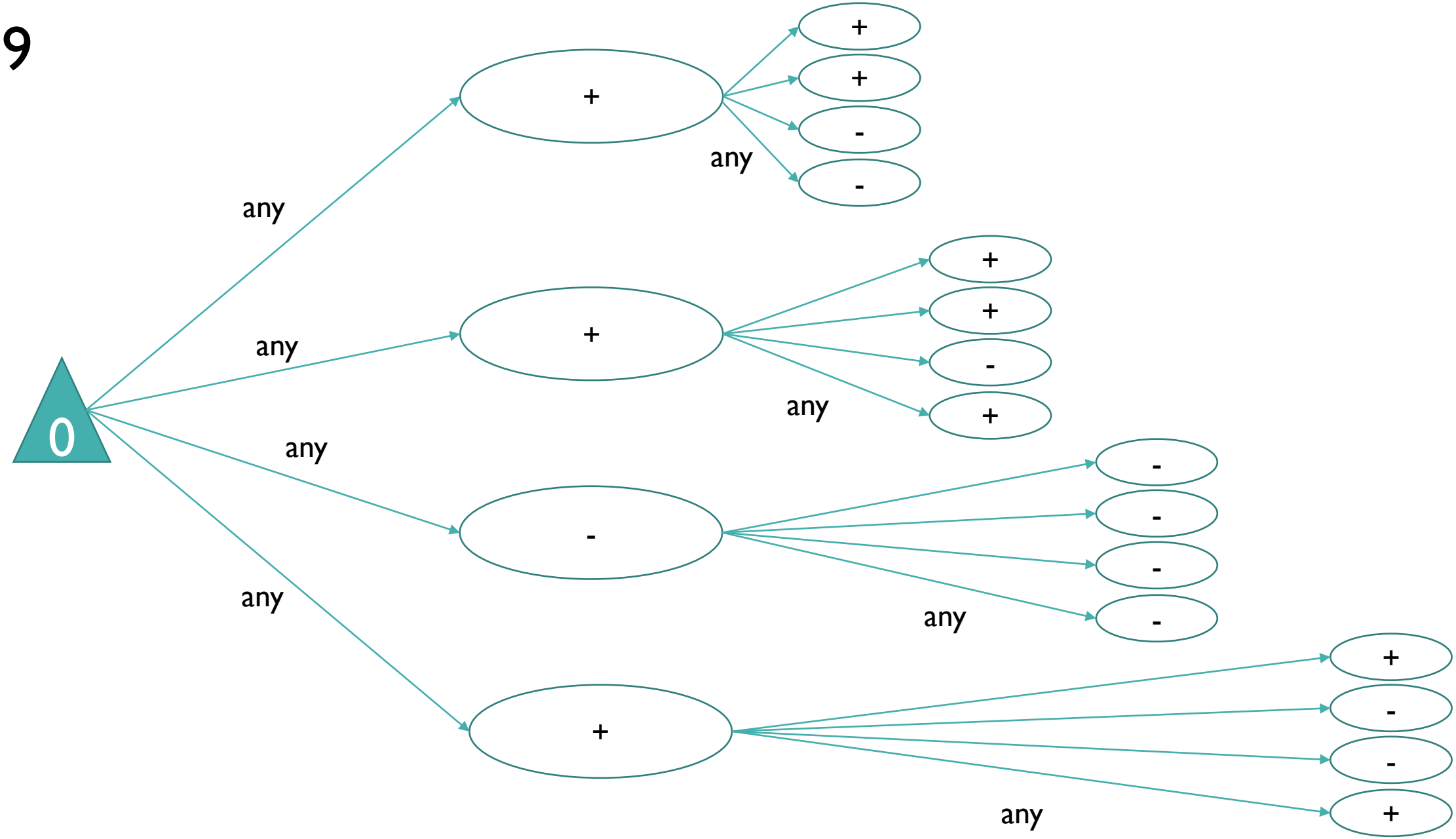
```
def f(a,b,c,m):
    if a==0 or b==0: return c%2==m%2
    if c==m: return 0

    h = []
    if a>=3 and b>=3: h+=[f(a-3,b-3,c+1,m)]
    if a%2==0: h+=[f(a//2,a//2,c+1,m)]
    if b%2==0: h+=[f(b//2,b//2,c+1,m)]
    if len(h)==0: return c%2==m%2

    return any(h) if (c+1)%2==m%2 else all(h)

for k in range(1,60):
    for m in range(1,5):
        if f(20,k,0,m)==1:
            if m==4: print(k,m)
            break
```

Как делать НЕ надо


0
any
any
any
any
+
+
-
-
any
+
+
-
+
any
-
-
-
-
any
+
-
-
+
any

```
class Game:
    def __init__(self, k1, k2):
        self.k1 = k1  // Камней в первой куче
        self.k2 = k2  // Камней во второй куче
        self.state = ('UNKNOWN', 0) // Состояние текущей игры
        self._move_count = 0 // Ходов всего* совершено
```

	Петя →	Ваня →	Петя
move_count: 0	1	2	3

```
def move1(self):
    self.k1 += 1
    return self

def move2(self):
    self.k1 *= 2
    return self
```

Первая куча

```
def move3(self):
    self.k2 += 1
    return self

def move4(self):
    self.k2 *= 2
    return self
```

Вторая куча

Самопроверка:

Nкуч * Nходов = Nфункций

2 кучи * 2 хода (+1, *2) = 4 функции

```
from copy import deepcopy
```

```python
def derive_states(self):
    new_states = []

    s = deepcopy(self).move1().update_state()
    if s.k1 <= 60:
        new_states.append(s)

    s = deepcopy(self).move2().update_state()
    if s.k1 <= 60:
        new_states.append(s)

    s = deepcopy(self).move3().update_state()
    if s.k1 <= 60:
        new_states.append(s)

    return new_states
```

Выдаём варианты, куда можно сходить из позиции

Проверки в более сложных задачах (в данном случае у игроков в сумме 60 камней)

```
def update_state(self):
    self._move_count += 1
    if self.k1 >= 51:
        self.state = ('WIN', self._move_count)
    else:
        self.state = ('UNKNOWN', self._move_count)
    return self
```

2 состояния – WIN или UNKNOWN **(состояния LOSE нет!)**

Ядро программы

```
def move_recursive(curr_game, moves_list, win_condition):
    states = curr_game.derive_states()
    reqs = []
    for st in states:
        if win_condition(st.state):
            reqs.append(True)
        elif len(moves_list) == 1 or st.state[0] == 'WIN':
            reqs.append(False)
        else:
            reqs.append(move_recursive(st, moves_list[1:], win_condition))
    return moves_list[0](reqs)
```

неочевидный момент

тут будет неочевидный момент 2

- Если мы уже в состоянии выигрыша (!!!!! Состояние выигрыша это не только WIN а лямбда-функция вида:
  lambda x: x == ('WIN', 4)
  Почему? Потому что нам важно на каком ходе мы выиграли (от этого зависит ещё кто выиграл). то True
- Если мы в WIN но не на нужном ходу –> (см. неочевидный момент) то это не подходит False
- Если ходы закончились (остался только 1 элемент в массиве) то в рекурсию мы уже не пойдём (другого хода нет уже), текущий нам не подходит. Это состояние когда за данное кол-во ходов никто не выиграл и никто не проиграл, но бессмысленно играть дальше. False
- Если ходы ещё есть идём в рекурсию, передавая следующую функцию all/any из moves_list (проверяем следующие ходы).

```python
for S in range(1, 51):
    if move_recursive(Game(S), [any, any], lambda x: x == ('WIN', 2)):
        print(S)
```

Проверяем все S

any(a,b,c) = a OR b OR c

all(a,b,c) = a AND b AND c

Неочевидный момент 2:

```
>>> any([])
False
>>> all([])
True
```

```python
return moves_list[0](reqs) and len(reqs) > 0
```

\+ фильтруем по длине (если ходить некуда то позиция не выигрывает)