# ฟังก์ชันแบบเวียนเกิด

Recursive Functions

# ความสัมพันธ์เวียนเกิด (Recurrences)

- การเขียนความสัมพันธ์ของจำนวนเต็มในลำดับ

0, 1, 2, 3, 4, ... $a_n = a_{n-1} + 1$ เมื่อ $n > 0,\ a_0 = 0$

3, 5, 7, 9, 11, ... $a_n = a_{n-1} + 2$ เมื่อ $n > 0,\ a_0 = 3$

0, 1, 3, 6, 10, 15, ... $a_n = a_{n-1} + n$ เมื่อ $n > 0,\ a_0 = 0$

0, 1, 1, 2, 3, 5, 8,... $f_n = f_{n-1} + f_{n-2}$ เมื่อ $n > 1,\ f_0 = 0,\ f_1 = 1$

# $a_n$ = 0, 1, 3, 6, 10, 15, ...

- รู้ว่า $a_n = (0+1+2+...+n)$

แบบที่ 1

```
def a(n):
  s = 0;
  for i in range(n+1):
    s += i
  return s
```

- รู้ว่า $a_n = a_{n\text{-}1} + n$ เมื่อ $n > 0$, $a_0 = 0$

แบบที่ 2

```
def a(n) {
  if n <= 0:
    return 0
  else:
    return a(n-1) + n;
```

# ฟังก์ชันหาค่า $n!$

แบบที่ 1

```
def fac(n):
   f = 1
   for i in range(1,n+1):
      f = f * i

   return f
```

$$n! = n\times(n-1)\times(n-2)\times \ldots \times 2\times 1$$

แบบที่ 2

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

กรณีเล็กสุด

ตามนิยามของแฟกตอเรียล

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n \geq 1 \end{cases}$$

# ข้อดี-ข้อด้อย

การเขียนแบบ recursive มีทั้งข้อดีและข้อด้อย

ข้อดี

- สั้นกระทัดรัด
- ในบางกรณี มุมมองแบบ recursive จะทำให้เห็นวิธีแก้ปัญหาได้ง่ายขึ้น
- ถ้าจำนวนชั้นของ loop ไม่ "คงที่" การใช้ recursive จะง่ายกว่ามาก
- ได้คำตอบย่อย

ข้อด้อย

- บางครั้งทำงานช้ากว่าแบบ loop
- ใช้หน่วยความจำมากกว่า

# การคำนวณ $a^k$ mod m

- $a^k$ mod m เป็นการคำนวณที่ใช้บ่อยในการเข้ารหัสลับ
- ตัวอย่างที่ 1 : $2^{20}$ % 31 = ?
  - คำนวณ $2^{20}$ ได้ 1048576 จากนั้น % 31 ได้ 1
- ตัวอย่างที่ 2 : $\mathbf{2^{101}}$ % 31 = ?
  - คำนวณ $2^{101}$ ได้ **2535301200456458802993406410752** จากนั้น % 31 ได้ **2**
  - อีกแบบ : $2^{101}$ % 31 = $2(2^{50}$ % $31)^2$ % 31 = $2(1)^2$ % 31 = 2

$$
a^k \% m = \begin{cases} 1 & k = 0 \\ (a^{\lfloor k/2 \rfloor} \% m)^2 \% m & k \text{ is even} \\ a(a^{\lfloor k/2 \rfloor} \% m)^2 \% m & k \text{ is odd} \end{cases}
$$

$$2^{60} \bmod 10 = 4^2 \bmod 10 = 6$$
$$2^{30} \bmod 10 = 8^2 \bmod 10 = 4$$
$$2^{15} \bmod 10 = 2 \times 8^2 \bmod 10 = 8$$
$$2^{7} \bmod 10 = 2 \times 8^2 \bmod 10 = 8$$
$$2^{3} \bmod 10 = 2 \times 2^2 \bmod 10 = 8$$
$$2^{1} \bmod 10 = 2 \times 1^2 \bmod 10 = 2$$
$$2^{0} \bmod 10 = 1$$

# ตัวอย่าง : flatten_list

เขียนฟังก์ชัน **flatten_list** ซึ่งรับ list of lists of lists ...
เช่น [[1,2,3],4,[5,[6,7,[8,9],10],[11,12]]])
เพื่อคืน list ที่มี สมาชิกทุกตัวของ list เดิม เช่น

input: [[1,2,3],4,[5,[6,7,[8,9],10],[11,12]]]

output: [1,2,3,4,5,6,7,8,9,10,11,12]

Hint: ถ้าอยากทดสอบว่า x เป็น list หรือไม่ ใช้

- `type(x) is list` หรือ
- `isinstance(x,list)`

# flatten_list

```
def flatten_list(d):
    flat = []
    for e in d:
        if type(e) is list:
            flat += flatten_list(e)
        else:
            flat.append(e)
    return flat

x = [1,[[[2,3],4]],[[5,6],7],8]
print(flatten_list(x))
```

# flatten_list (อีกแบบ)

```
def flatten_list(x):
    if len(x) == 0:
        return x
    h = x[0:1]
    if typeof(x[0]) is list:
        h = flatten_list(x[0])
    return h + flatten_list(x[1:])

x = [1,[[[2,3],4]],[[5,6],7],8]
print(flatten_list(x))
```

# Edit distance

# เทียบความยาวของสองสตริงที่เกิดจากการพิมพ์ผิด

| | |
|---|---|
| apple | ความแตกต่าง |
| aple | |
| | |
| a-a ตรง | 0 |
| p-p ตรง | 0 |
| p-l การแทนที่ | 1 |
| l-e การแทนที่ | 1 |
| e-_ การลบ | 1 |
| | = 3 |

| | |
|---|---|
| apple | ความแตกต่าง |
| aple | |
| | |
| a-a ตรง | 0 |
| p-p ตรง | 0 |
| p-_ การลบ | 1 |
| l-l ตรง | 0 |
| e-e ตรง | 0 |
| | = 1 |

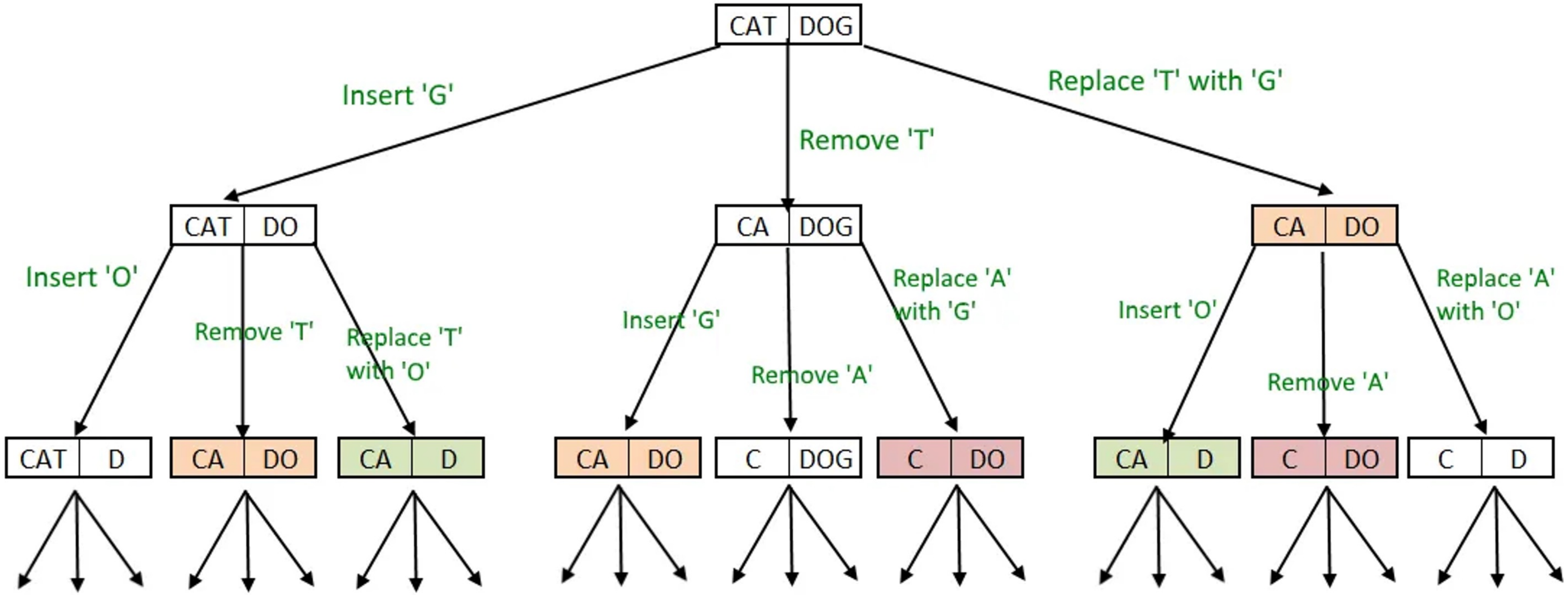
# เทียบความยาวของสองสตริงที่เกิดจากการพิมพ์ผิด

| apple | ความแตกต่าง | apple | ความแตกต่าง |
|---|---|---|---|
| aple | | aple | |
| | | | |
| a-a ตรง | 0 | a-a ตรง | 0 |
| _-p การเติม | 0 | p-p ตรง | 0 |
| p-l การแทนที่ | 1 | p-l การแทนที่ | 1 |
| p-e การ | 1 | l-e การแทนที่ | 1 |
| l-_ การลบ | 1 | e-_ การลบ | 1 |
| e-_ การลบ | 1 | | = 3 |
| | =4 | | |

# คำนวณได้หลายวิธีเอาอันที่น้อยที่สุด

ใช้ recursion

ช่วย



# คำนวณได้หลายวิธีเอาอันที่น้อยที่สุด

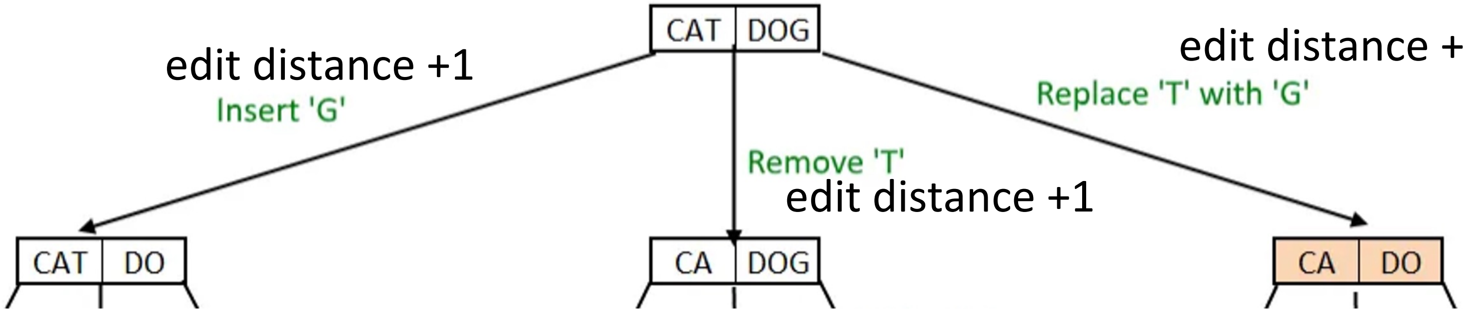
ใช้ recursion

ช่วย

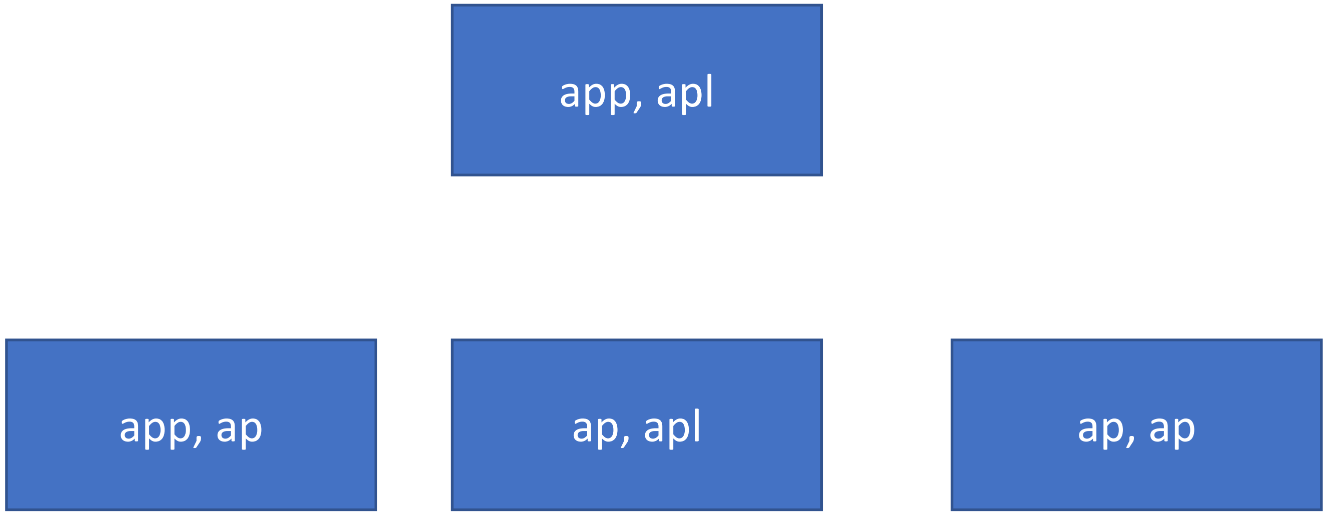
edit distance +1

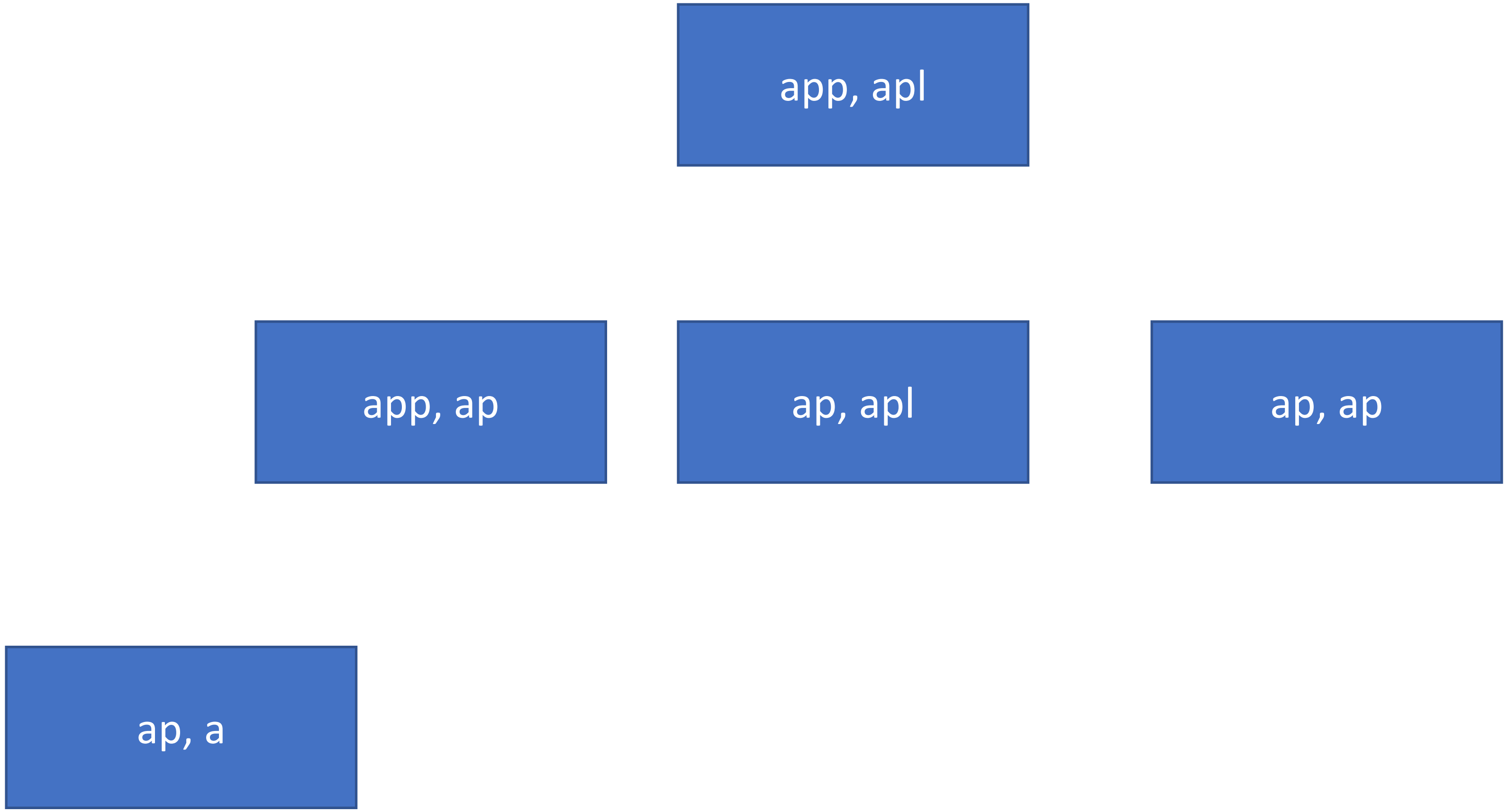
edit distance +1

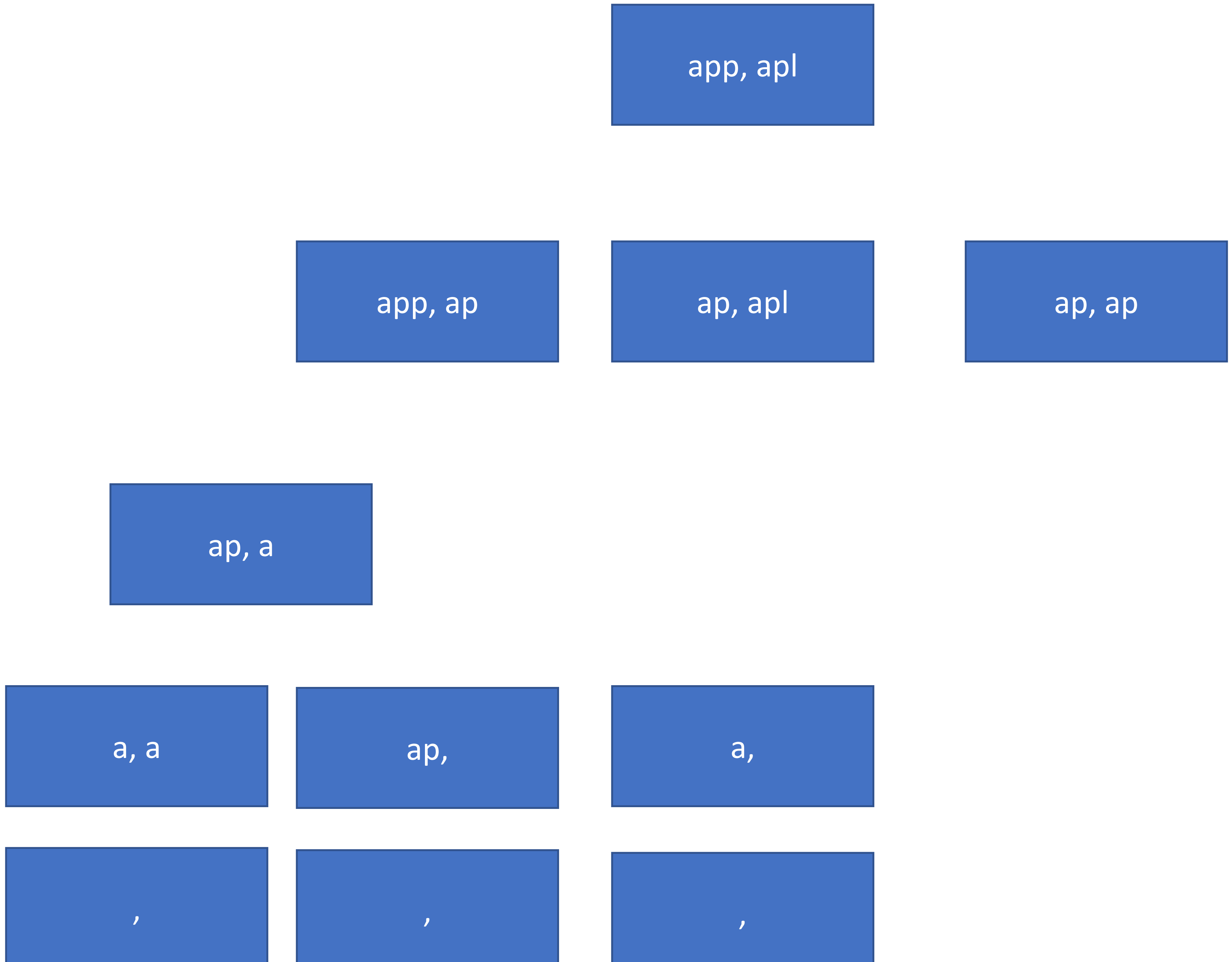
edit distance +1




edit distance ของ cat กับ dog เกิดจากedit distanceที่ดีที่สุดจากกรณีที่ล่างลงไป


app, apl
app, ap
ap, apl
ap, ap


app, apl
app, ap
ap, apl
ap, ap
ap, a


app, apl
app, ap
ap, apl
ap, ap
ap, a
a, a
ap,
a,
,
,
,

app, apl

app, ap

ap, apl

ap, ap

ap, a

ใช้ค่าที่น้อยที่สุดของความเป็นไปได้ในแต่ละชั้น

a, a

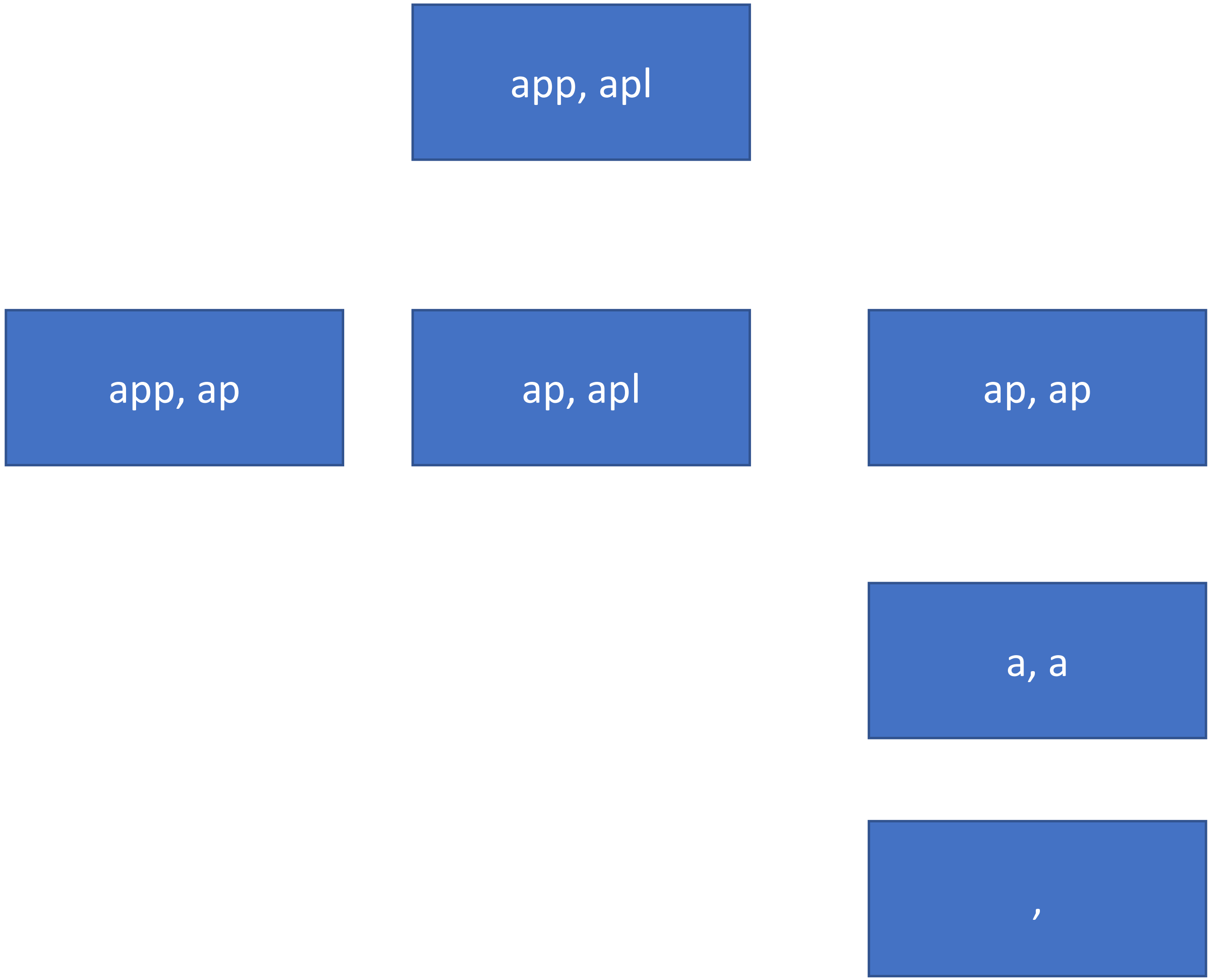
ap,

a,

,

,

,


app, apl
app, ap
ap, apl
ap, ap
a, a
,

# Base case?

ap,

a,

- ไม่เหลือตัวอักษรฝั่งซ้าย
- ไม่เหลือตัวอักษรฝั่งขวา