# Problem 1 : sum

Program received N as input, Return summation value of index 1 to N

| index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| value | 1 | 2 | 6 | 4 | 15 | 12 | 7 | 8 | 18 | 30 | 11 | 24 | 13 | 14 | 150 | 16 | 17 | 36 |

N = 5 , output = 1+2+6+4+15 = 28
N = 15, output = 1+2+6+4+15+12+7+8+18+30+11+24+13+14+150 = 315

| Example Input | Example |
|---|---|
| 5 | 28 |
| 15 | 315 |

# Problem 2 : coin

Given two type of coin, which have value A baht and B baht. Determine whether the target value T can be summed by the given types of coin.

6,8,10  -> A=6, B=8, target value=10; Coins 6 and 8 cannot sum up to 10

2,3,10  -> A=2, B=3, target value=10; Coins 2 and 3 can sum up to 10

If there are many possible answers, print any of them

| Example Input | Example |
| --- | --- |
| 6, 8, 10 | IMPOSSIBLE |
| 2, 3, 10 | na: 2, nb: 2 |

# Problem 2 : coin

find such a case that T = a*na + b*nb

Proposed algorithm

Check GCD(a,b)
https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm

loop find na
    if b | T - (a * na) then
        nb = ( T - (a*na) )/ b

# Problem 3: anagram

A word is an **anagram** of another if the second is simply a re-arrangement of the first.

| Example Input | Example |
| --- | --- |
| APPLE | ['APPEL', 'APPLE', 'PEPLA'] |
| PIE | ['EPI', 'IPE', 'PEI', 'PIE'] |
| noon | ['NONO’, 'NOON'] |

# Problem 3: anagram

**Proposed algorithm: input list of all words**

1. We gonna count number of alphabet as shown below,
   get_key(PIE) -> 0_0_0_0_1_0_0_0_1_0_0_0_0_0_0_1_0_0_0_0_0_0_0_0_0_0

| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |

2.Then, we will store “PIE” into the dictionary, using our precompute key

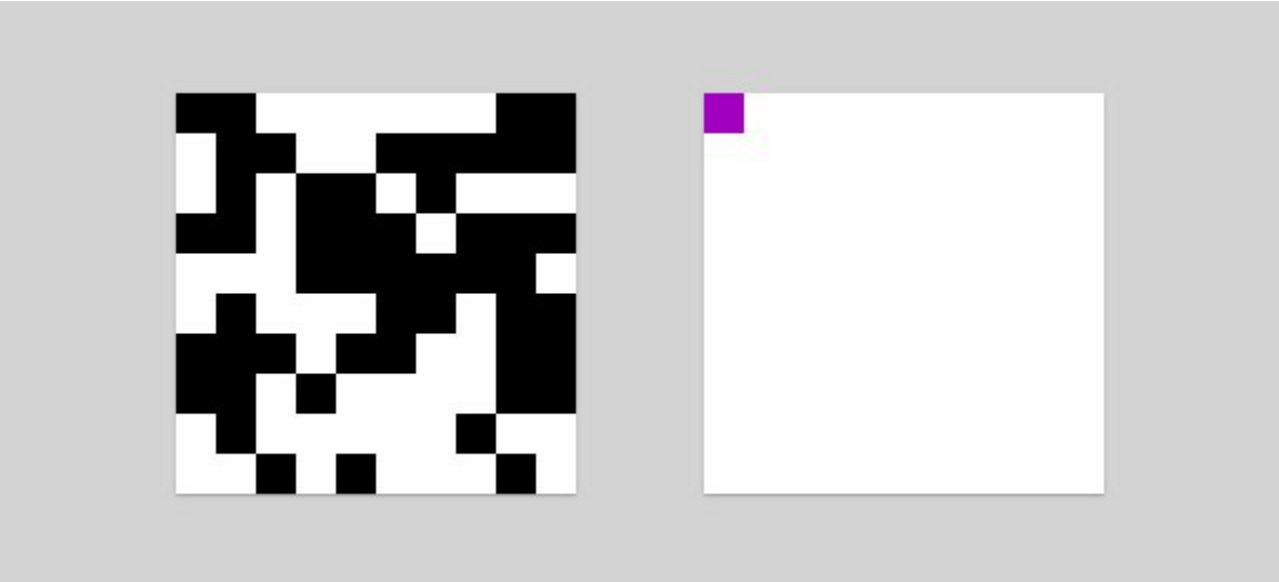
dict[get_key(PIE)].append(“PIE”)

3.In the end, dictionary should contains all words with the same alphabet count

dict[get_key(PIE)] = ['EPI', 'IPE', 'PEI', 'PIE']

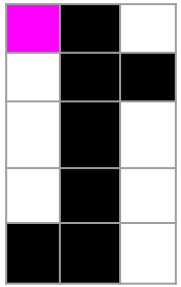
# Problem 4: connected component

Visualized coloring algorithm



Connected Component -- Step 1

1



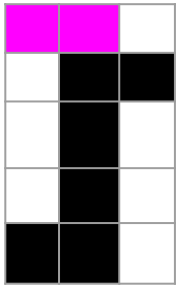
top

bottom

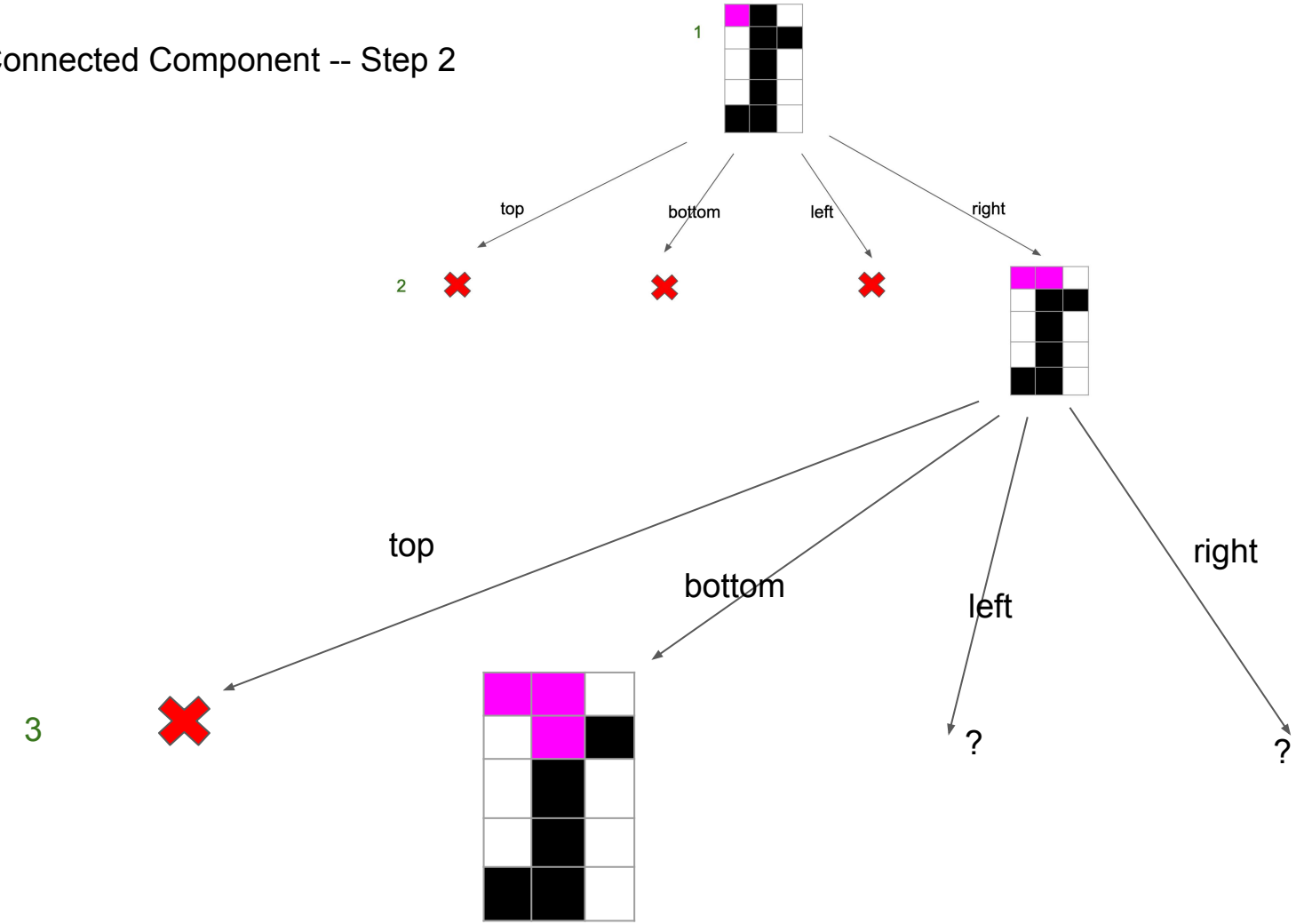
left

right

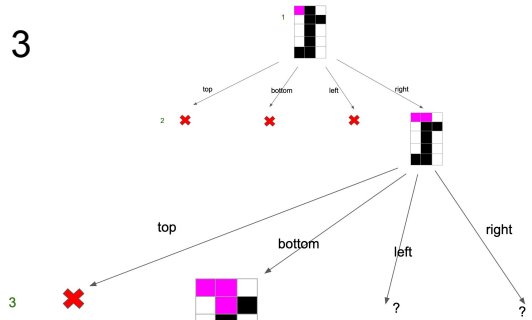
2



# Connected Component -- Step 2



# Connected Component -- Step 3




top

bottom

left

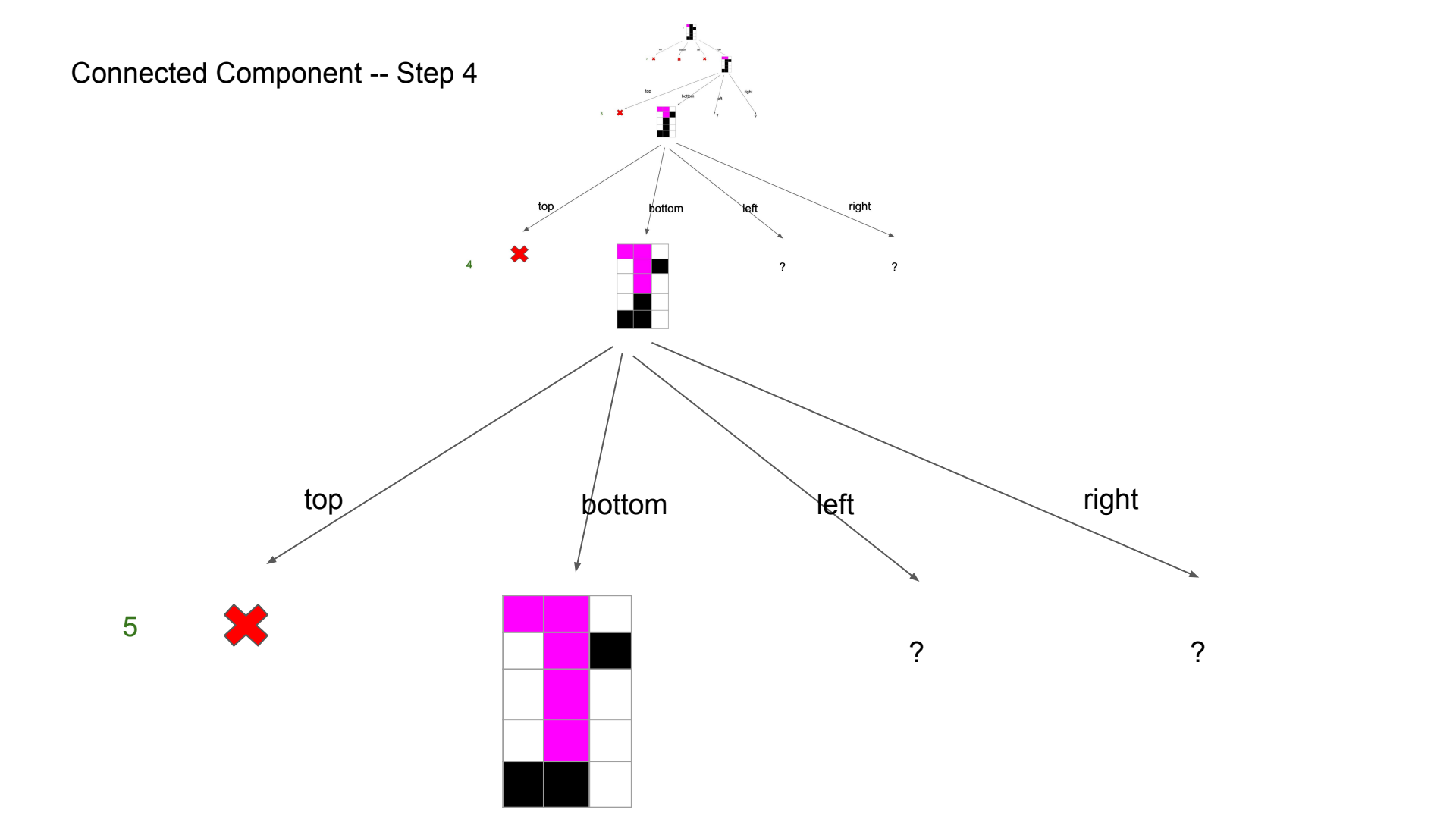
right

4

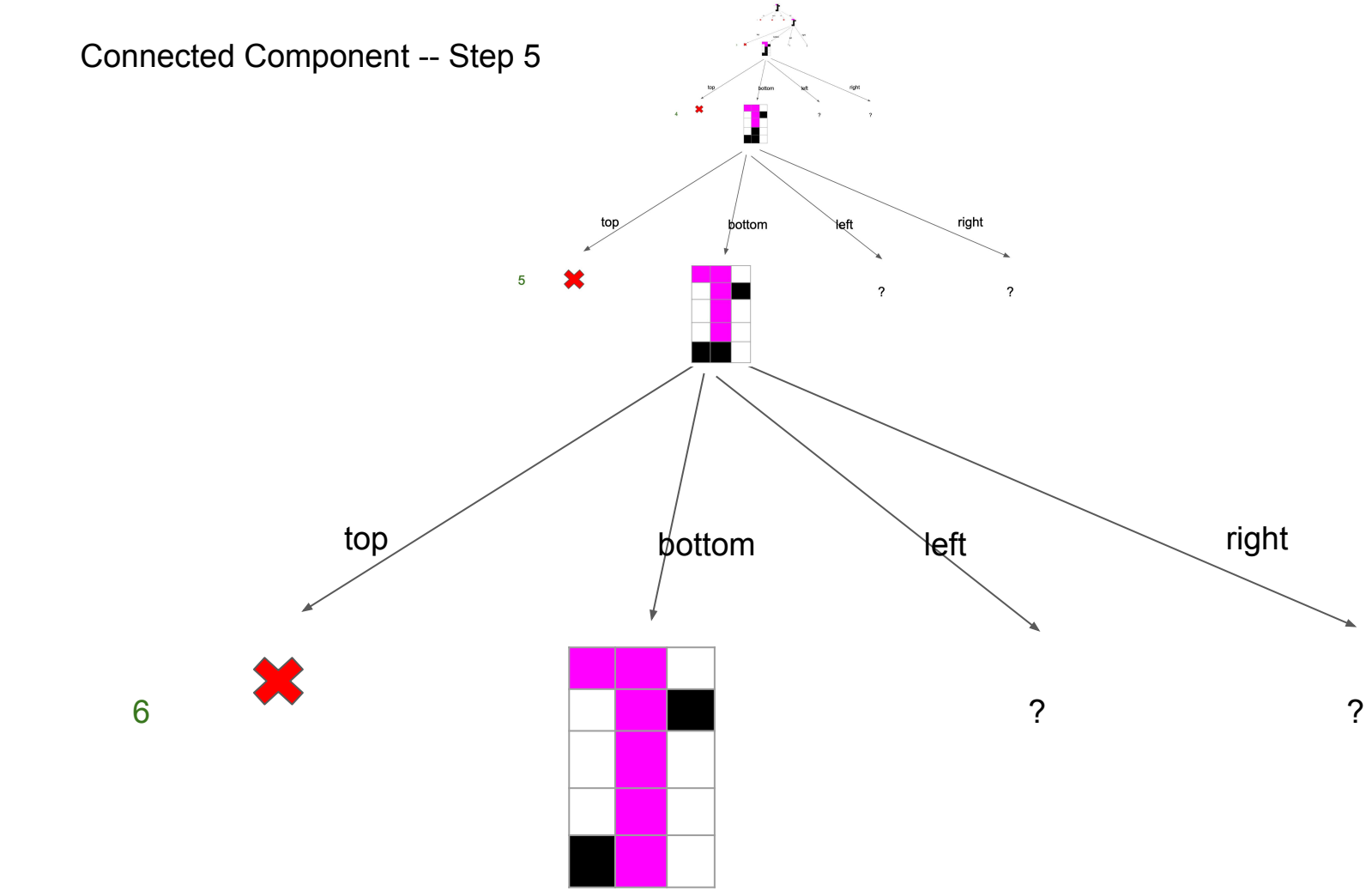
?

?

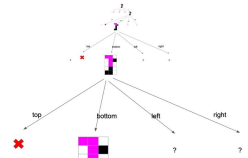
# Connected Component -- Step 4



# Connected Component -- Step 5

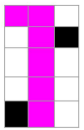


# Connected Component -- Step 6



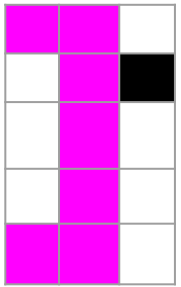

top bottom left right

6 ? ?

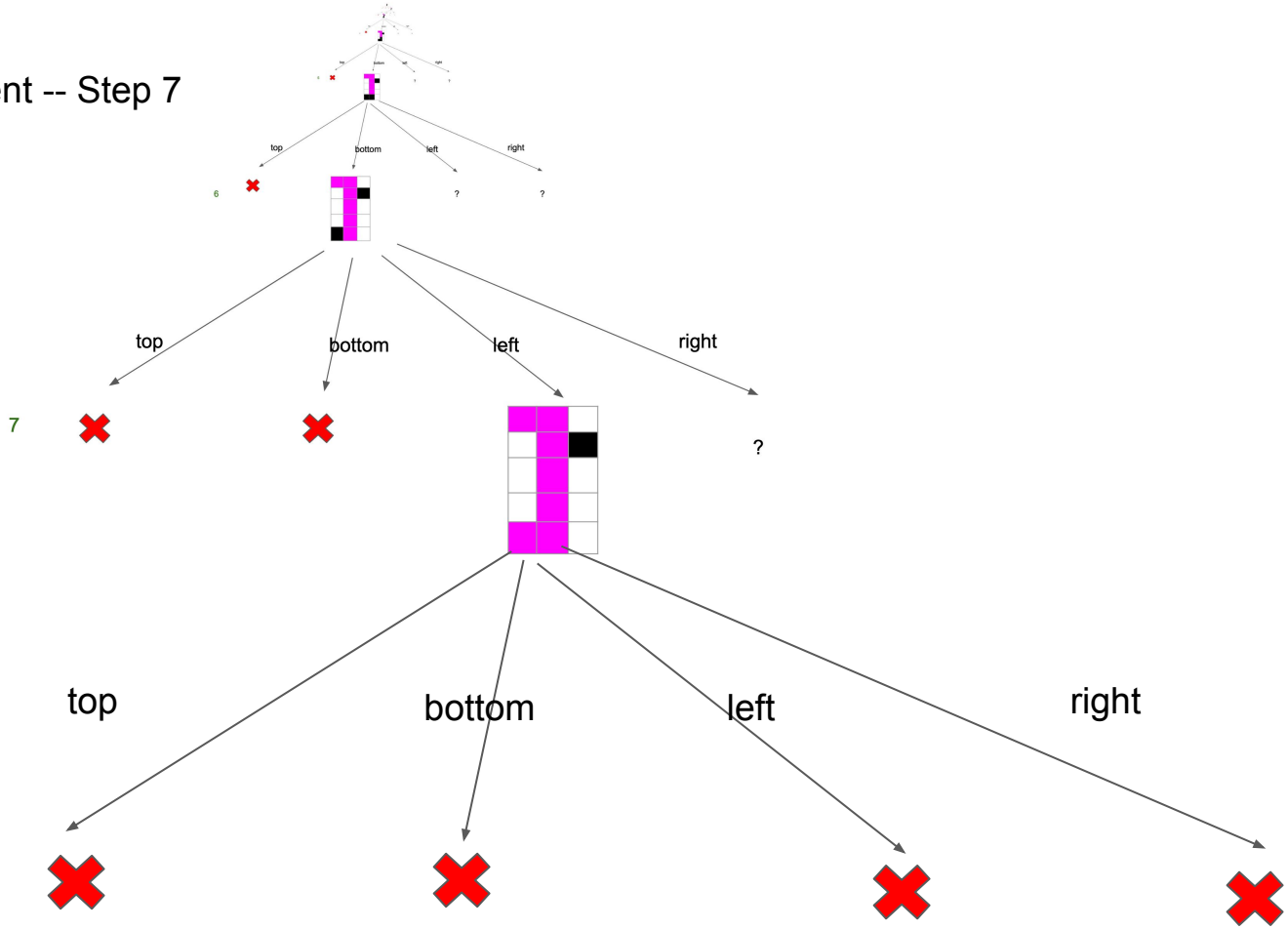


top bottom left right

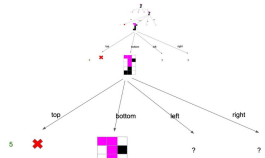
7 ?



# Connected Component -- Step 7

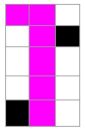


# Connected Component -- Step 8



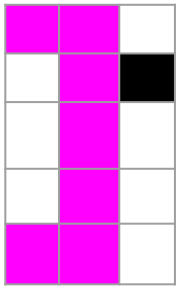

top bottom left right

6 ? ?

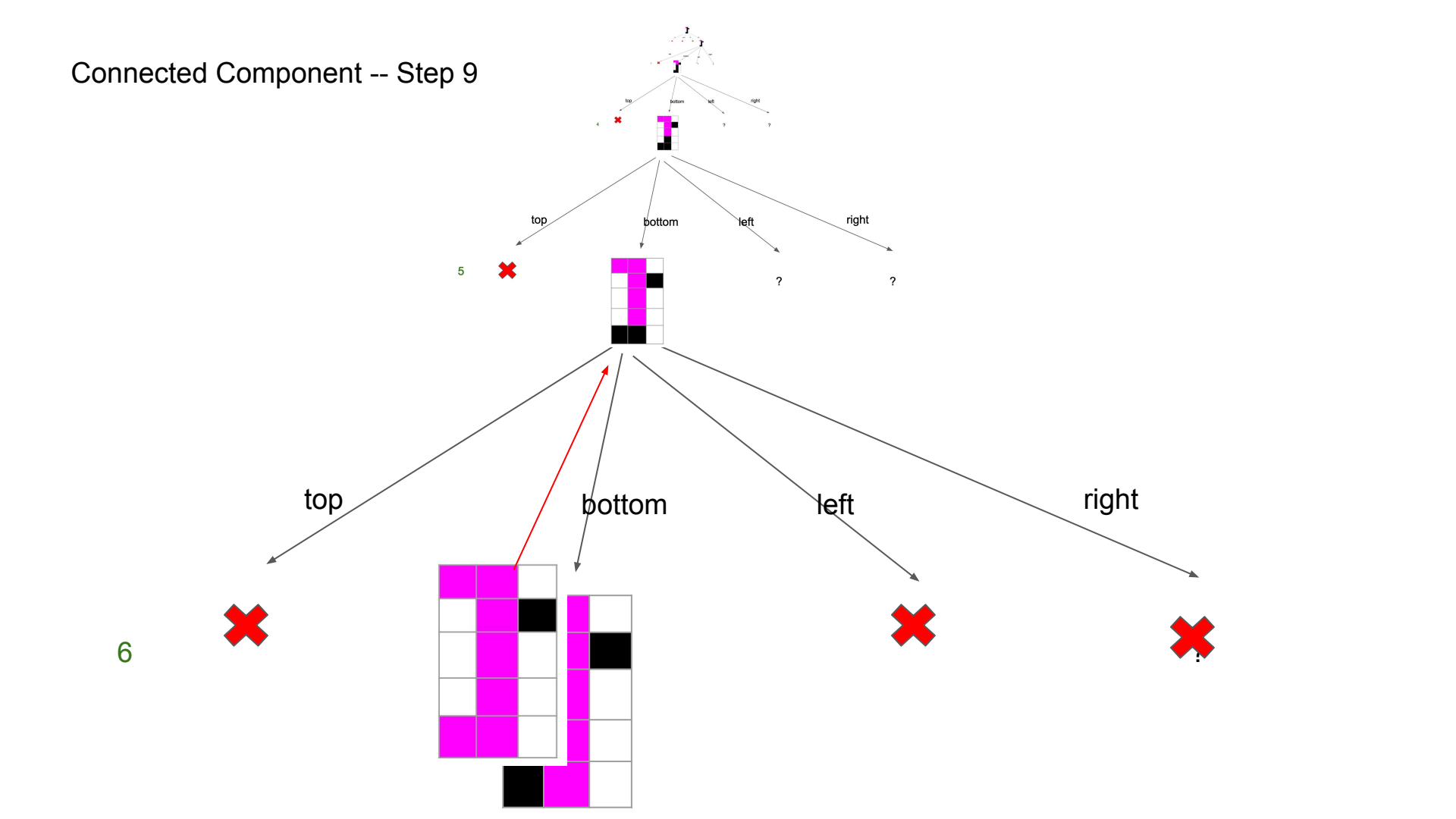


top bottom left right

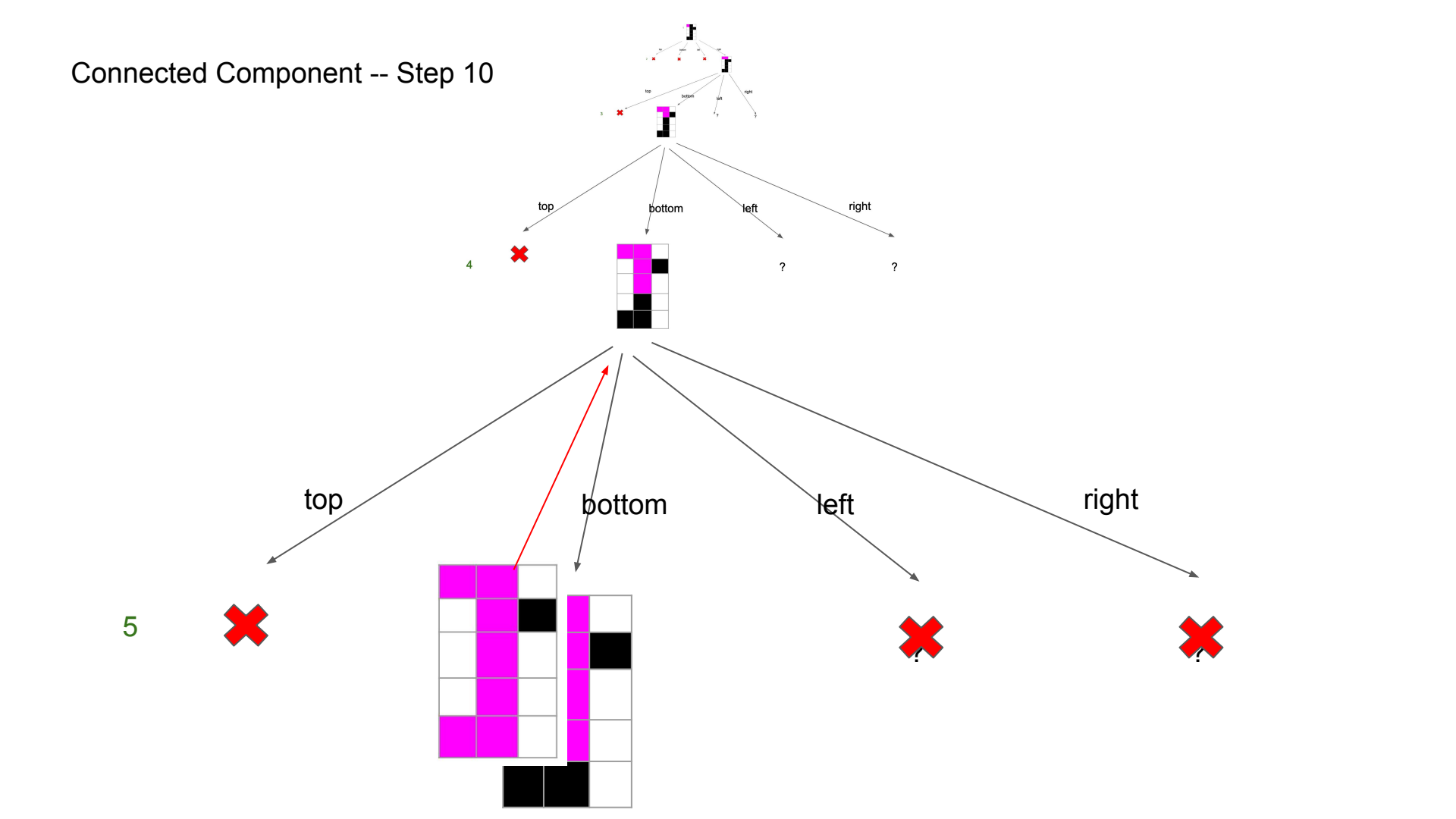
7



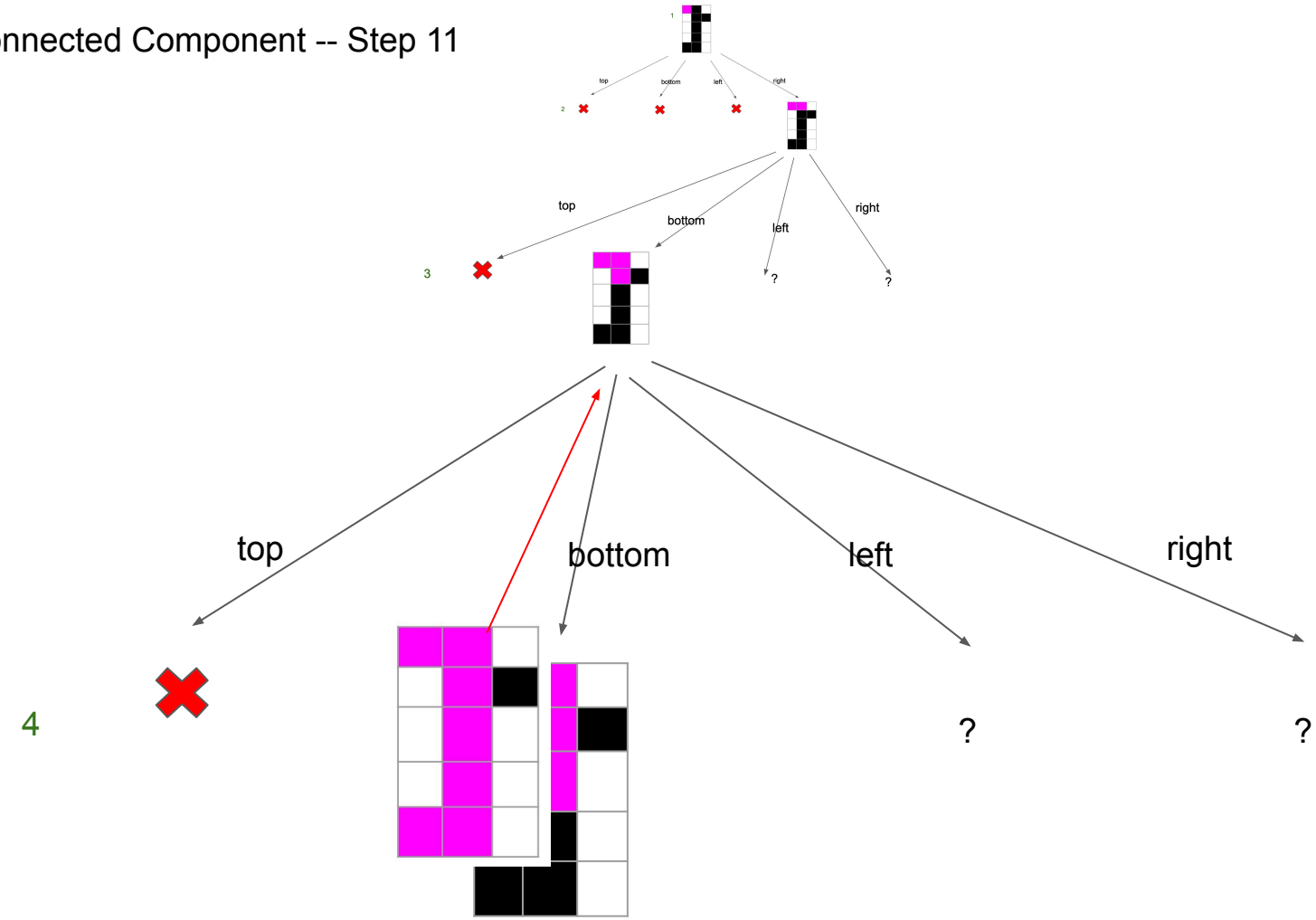
# Connected Component -- Step 9



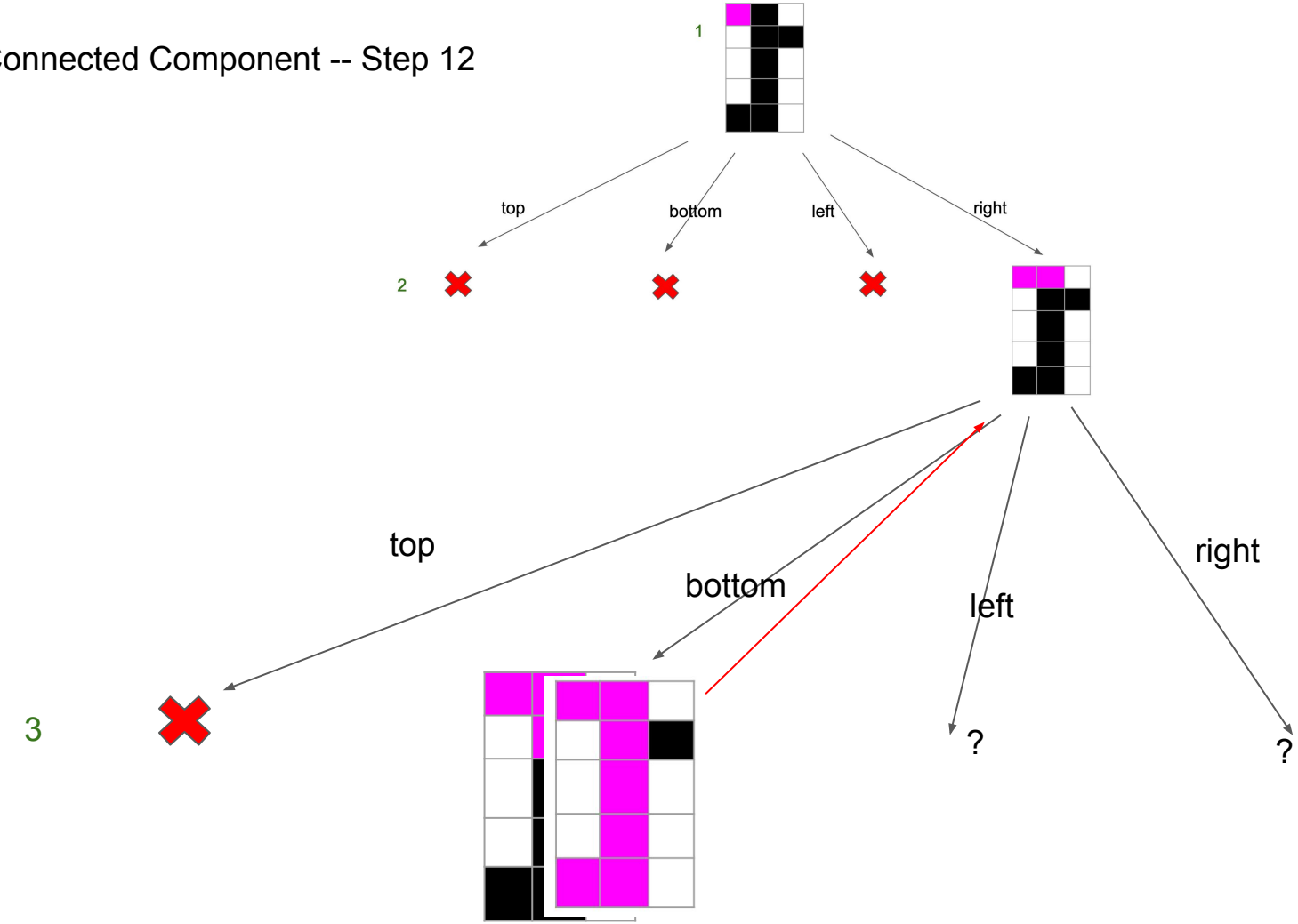
# Connected Component -- Step 10



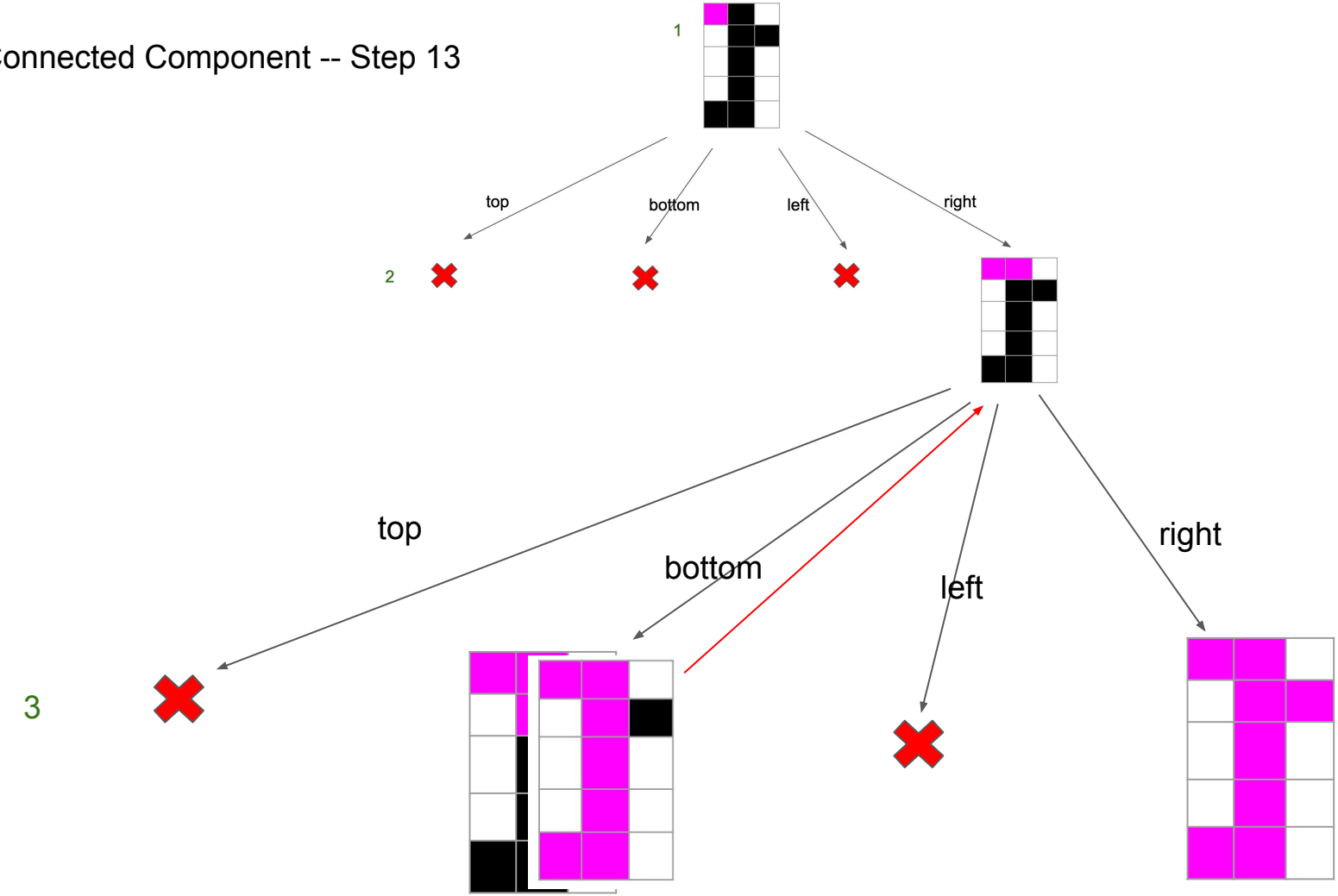
# Connected Component -- Step 11



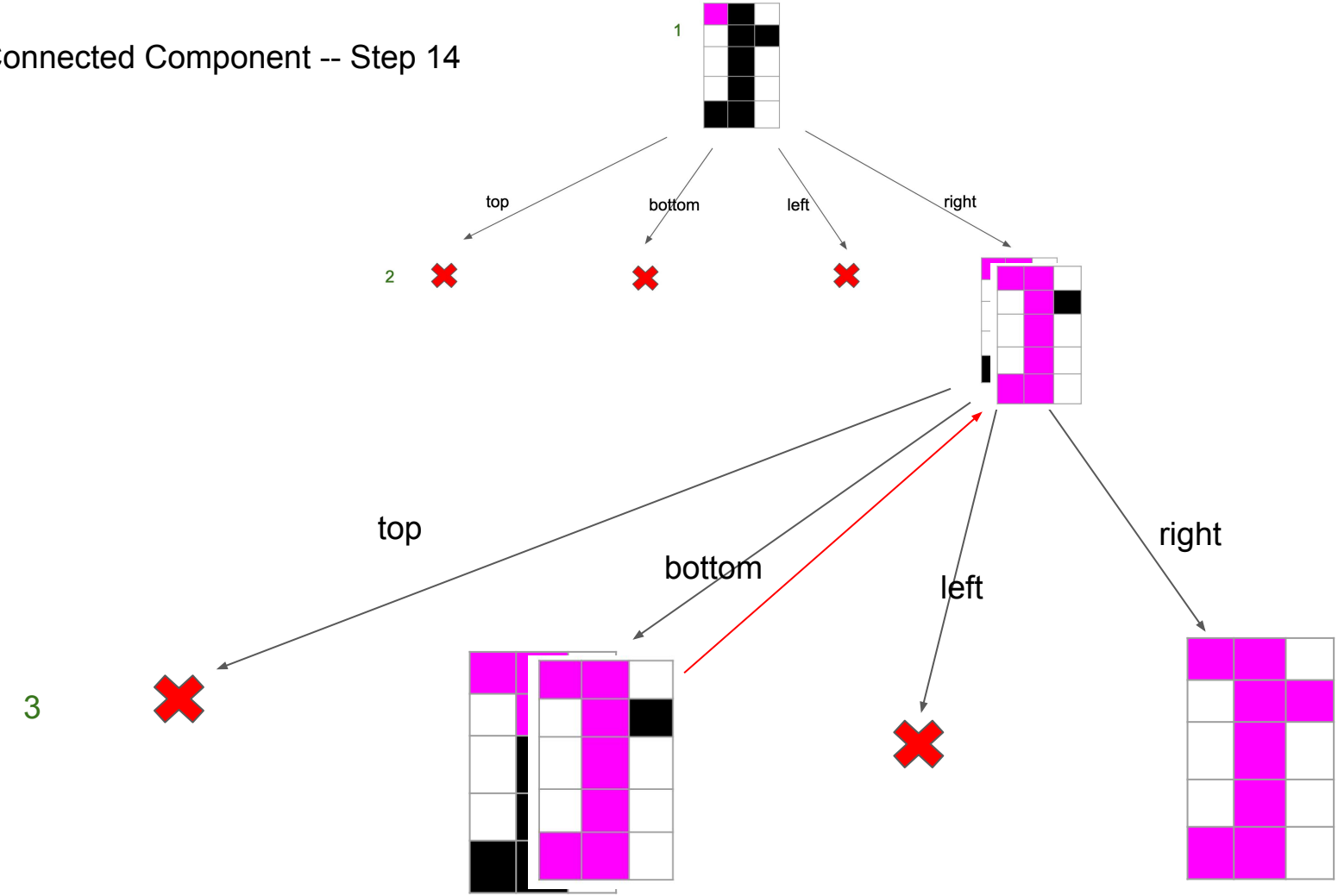
# Connected Component -- Step 12



Connected Component -- Step 13



Connected Component -- Step 14



Connected Component -- Step 15