Linguagens e Ambientes de Programação (Aula Teórica 6)

LEI - Licenciatura em Engenharia Informática

João Costa Seco (joao.seco@fct.unl.pt)

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Agenda

- Funções recursivas sobre números naturais.
- Pensamento indutivo vs. pensamento iterativo.
- Tipos estruturados: produtos e registos.

Funções recursivas

- A recursão é um mecanismo que permite definir uma entidade (função, tipo, classe, etc.), usando na sua definição o seu próprio nome.

```
let rec x = e1 in e2
```

```
class Node<T> {
    T value;
    Node<T> next;
}
```

- A recursão é um mecanismo que permite instanciar o mesmo código (função) mais que uma vez no mesmo traço de execução, com valores potencialmente diferentes para os parâmetros.

Funções recursivas

- As linguagens funcionais utilizam a recursão como meio básico para iterar.

```
let rec loop () = read_int () |> print_int; print_endline ""; loop ();;
```

- A utilização recursiva de um nome tem que estar sempre "guardada" pela definição de uma função.

Sintaxe Hint: Operadores

```
let (|>) x f = f x
```

```
let rec loop () = read_int () |> print_int; print_endline ""; loop ();;
```

```
let ( ^^ ) x y = max x y
```

```
( + )
```

```
- : int -> int -> int = <fun>
```

Funções recursivas (exemplo: somatório)

$$\sum_{i=l}^{u} f(i)$$

```
let rec sum f l u =
  if l > u then 0
  else f l + sum f (l+1) u
```

```
utop # sum (fun x -> 2*x) 1 10
;;
- : int = 110
```

Execução baseada numa pilha

- A chamada de funções é baseada numa pilha, onde são guardados os valores dos parâmetros para cada chamada e as variáveis locais correspondentes.
- Cada chamada ocupa espaço em memória.

```ocaml
let rec sum n =
  if n = 0 then 0
  else n + sum (n-1)
```

```
utop # sum 10;;
sum <-- 10
sum <-- 9
sum <-- 8
sum <-- 7
sum <-- 6
sum <-- 5
sum <-- 4
sum <-- 3
sum <-- 2
sum <-- 1
sum <-- 0
sum --> 0
sum --> 1
sum --> 3
sum --> 6
sum --> 10
sum --> 15
sum --> 21
sum --> 28
sum --> 36
sum --> 45
sum --> 55
- : int = 55
```

Execução baseada numa pilha

- A chamada de funções é baseada numa pilha, onde são guardados os valores dos parâmetros para cada chamada e as variáveis locais correspondentes.
- Cada chamada ocupa espaço em memória.

```
let rec count n =
  if n = 0 then 0
  else 1 + count (n-1)
```

```
utop # count 100000;;
- : int = 100000
```

```
utop # count 1000000;;
Stack overflow during evaluation (looping recursion?).
```

Execução baseada numa pilha

- A chamada de funções é baseada numa pilha, onde são guardados os valores dos parâmetros para cada chamada e as variáveis locais correspondentes.
- Cada chamada ocupa espaço em memória.

```java
class Main {
    static int count(int n) {
        if (n == 0) {
            return 0;
        }
        return 1+count(n-1);
    }

    public static void main(String[] args) {
        System.out.println(count(20000));
    }
}
```

```
jcs@joaos-imac lap2024 % java Main | more
Exception in thread "main" java.lang.StackOverflowError
        at Main.count(A.java:8)
        at Main.count(A.java:8)
        at Main.count(A.java:8)
        at Main.count(A.java:8)
        at Main.count(A.java:8)
        at Main.count(A.java:8)
        at Main.count(A.java:8)
        at Main.count(A.java:8)
        at Main.count(A.java:8)
        at Main.count(A.java:8)
        at Main.count(A.java:8)
```

A pilha de execução e as chamadas de função

- Quando uma função tem alguma computação para fazer entre a chamada recursiva e o final, e devolver o resultado, então precisa de manter todos os registos de ativação.

“Tail recursion”

- A mesma função pode ser escrita de outra forma de modo a que, a seguir à chamada recursiva, não haja mais computação nenhuma.

```
let rec sum n =
    if n = 0
        then 0
        else n + sum(n-1)
```

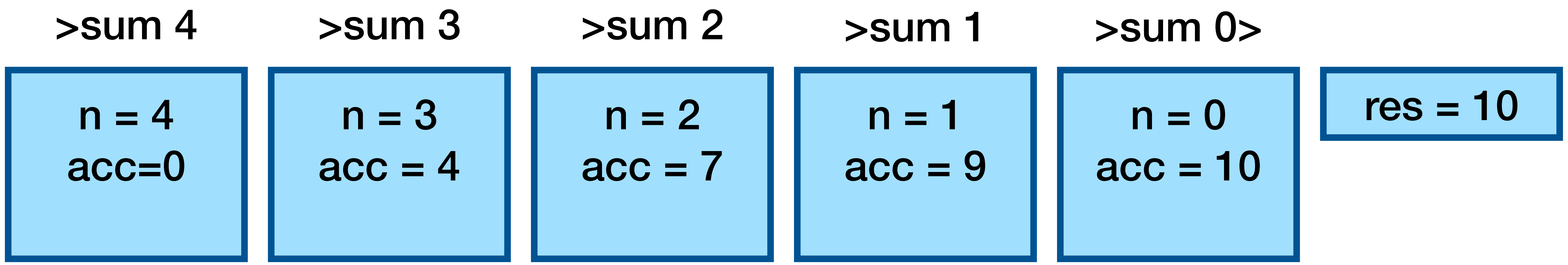
```
let sum n =
    let rec sum' n acc =
        if n = 0
            then acc
            else sum' (n-1) (n+acc)
    in sum' n 0
```

```
utop # count 1000000;;
Stack overflow during evaluation (looping recursion?).
```

```
utop # sum 1000000;;
- : int = 500000500000
```

“Tail recursion” e a pilha de execução

- Se a chamada recursiva é a ultima coisa a fazer na função, o registo de ativação pode ser reutilizado porque as variáveis locais não vão ser precisas depois de retornar e o resultado já está no sítio (topo da pilha).



Funções indutivas nos números naturais (correção)

- Funções recursivas podem seguir um raciocínio indutivo para chegar ao resultado. É possível provar a sua correção através de uma hipótese de indução.

```
(** [sum n] is the sum of the first [n] positive integers
    requires: [n >= 0] *)
let rec sum n =
    if n = 0
        then 0                  (* base case: sum 0 = 0 *)
        else n + sum(n-1)       (* inductive case: sum n = n + sum(n-1) *)
```

Funções indutivas nos números naturais (tail recursion)

- Funções recursivas podem seguir um raciocínio indutivo para chegar ao resultado. É possível provar a sua correção através de uma hipótese de indução.

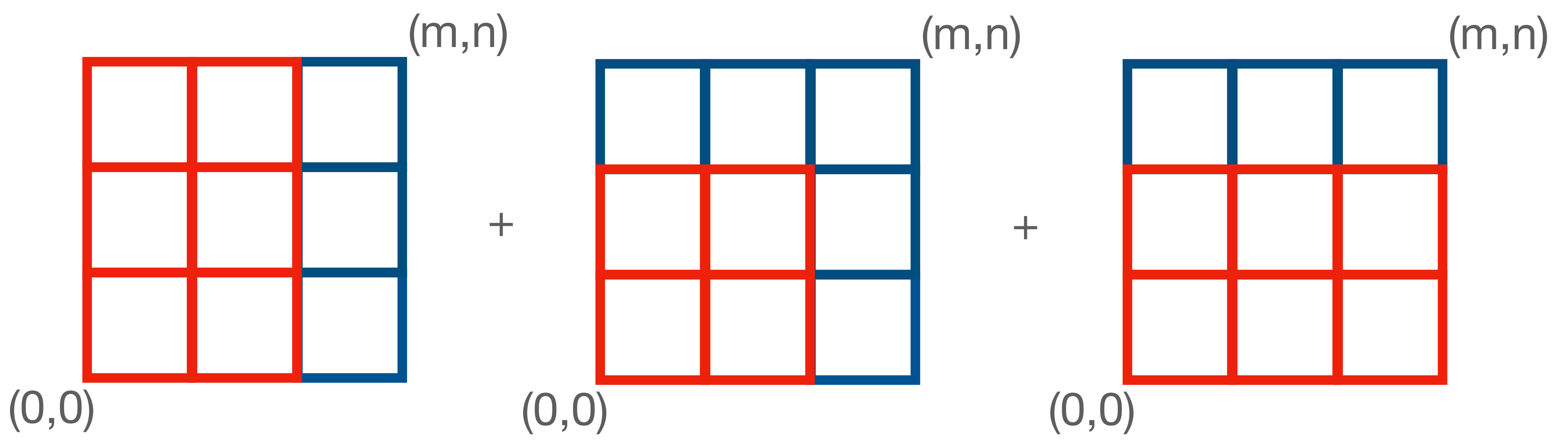
```
let sum m =
    let rec sum' n acc =           (* post-condition: sum' n acc = sum m && acc = (sum m) - (sum n) *)
        if n = 0
            then acc               (* base case: sum' 0 acc = sum m && acc = (sum m) - (sum 0) *)
            else sum' (n-1) (n+acc) (* inductive case: sum' (n-1) (n+acc) = sum m
                                                    && n+acc = (sum m) - (sum (n-1))
                                       ==>             acc = (sum m) - (n + sum (n-1))
                                       ==>             acc = (sum m) - (sum n). qed.           *)
    in sum' m 0                    (* conclusion: sum' m 0 = sum m && acc = (sum m) - (sum m) = 0 *)
```

Funções indutivas: o número de Delannoy

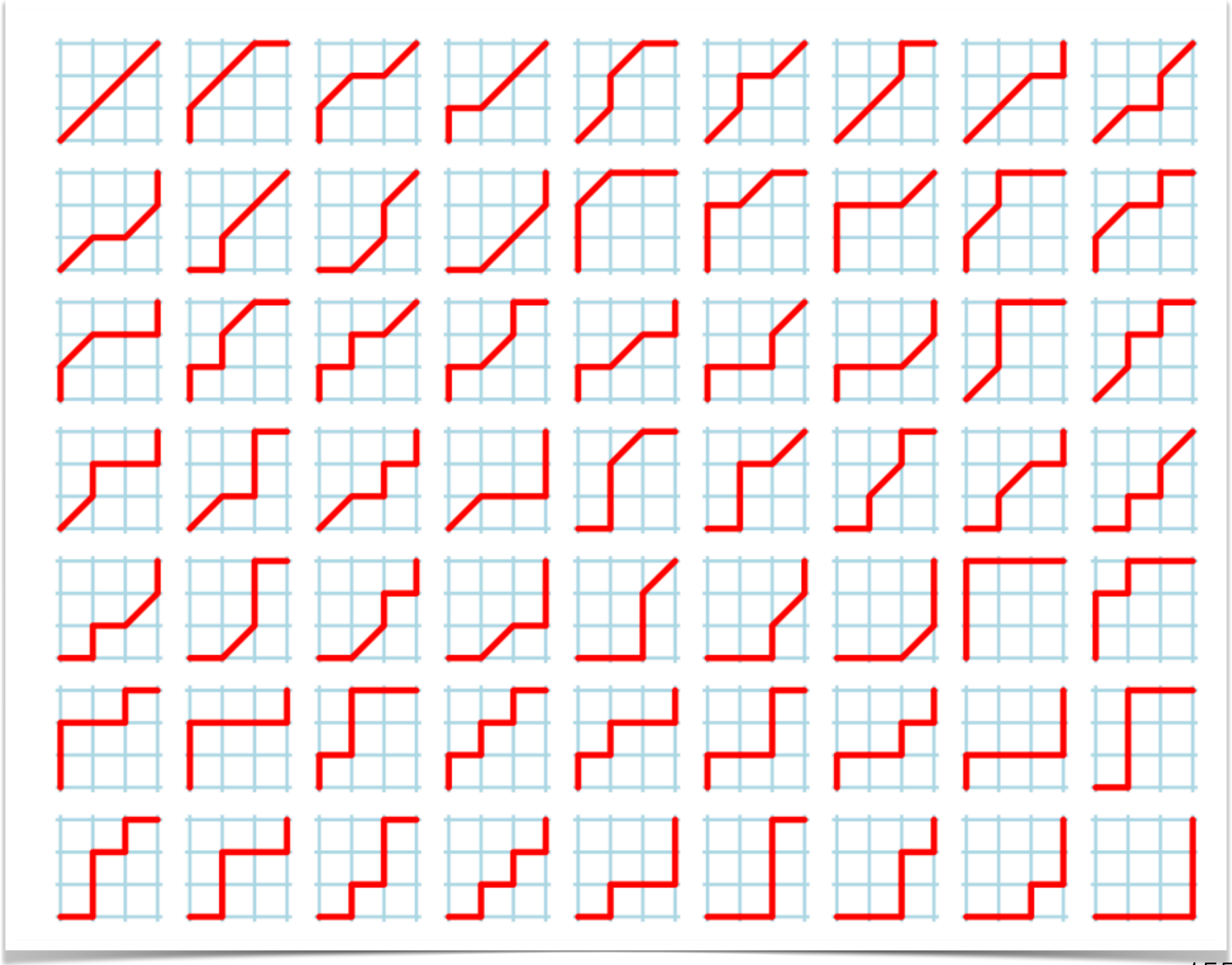
- Determine o número de caminhos que existem numa grelha de n por m, entre o ponto (0,0) e o ponto (m,n) usando passos de unidade 1 no sentido "norte", "nordeste", e "este".

(0,0)

- 1

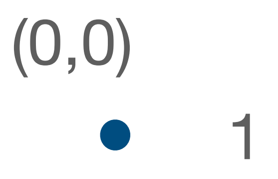


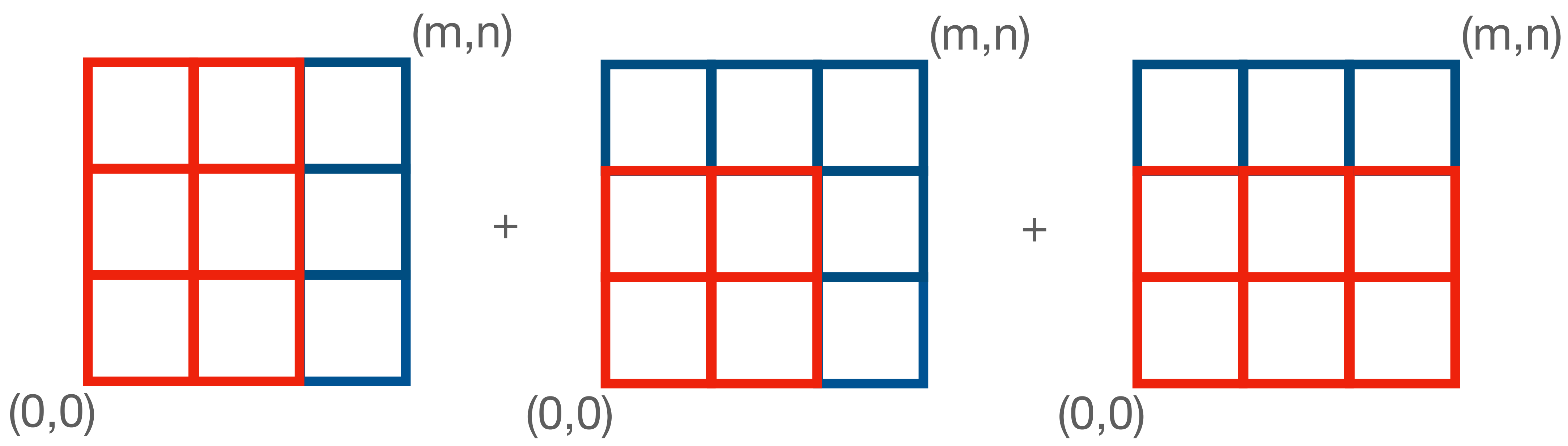
https://en.wikipedia.org/wiki/Delannoy_number



Funções indutivas: o número de Delannoy

- Determine o número de caminhos que existem numa grelha de n por m, entre o ponto (0,0) e o ponto (m,n) usando passos de unidade 1 no sentido “norte”, “nordeste”, e “este”.

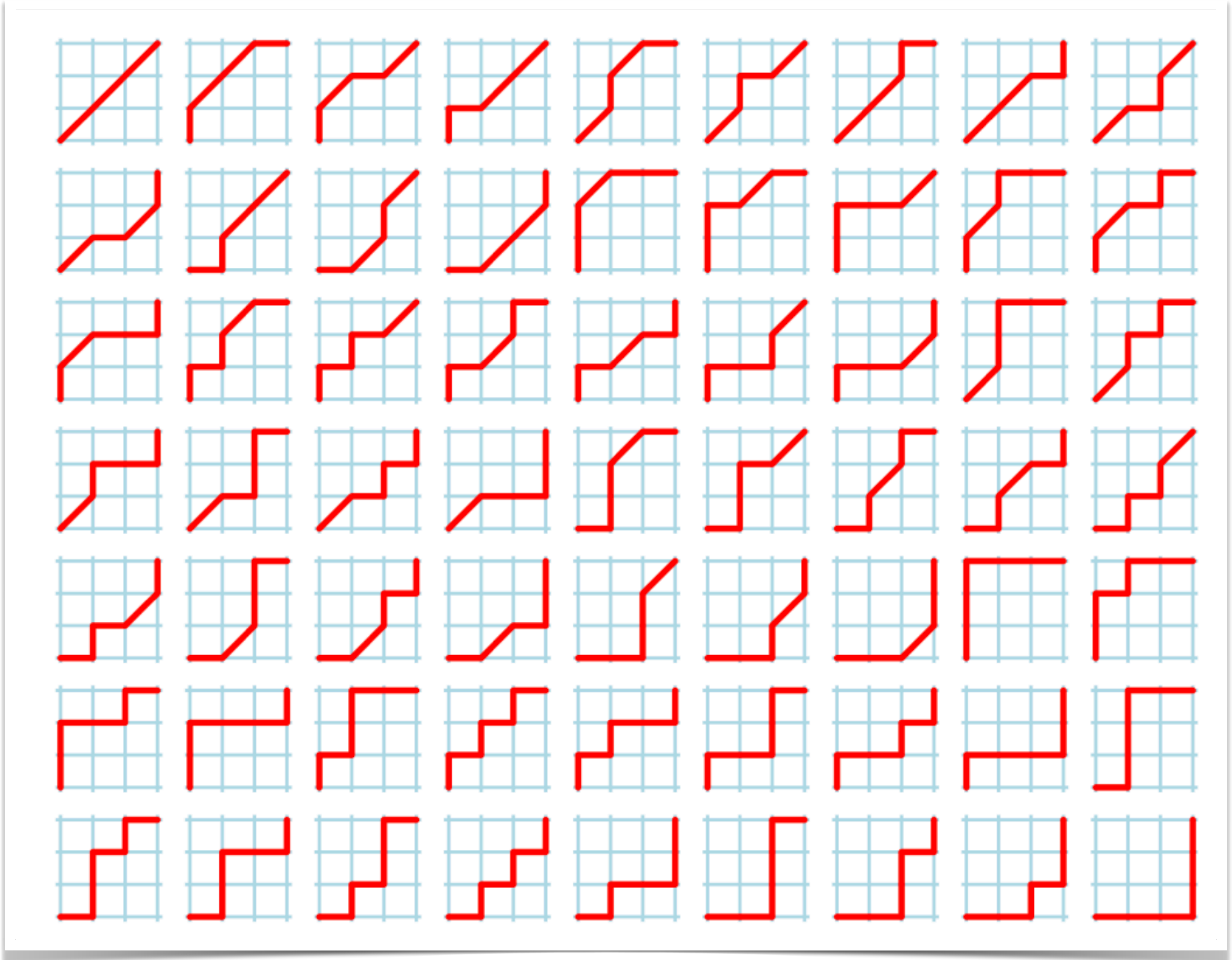





```
let rec delannoy m n =
  if m = 0 || n = 0 then 1
  else delannoy (m-1) n + delannoy m (n-1) + delannoy (m-1) (n-1)
```

✓ 0.0s

```
val delannoy : int → int → int = <fun>
```



Primeiro trabalho

São Triângulos, Senhor, são Triângulos

O *Triângulo de Pascal* é uma representação matricial dos coeficientes binomiais, com grande utilidade em teoria das probabilidades, combinatória e álgebra. O seguinte diagrama apresenta as 8 primeiras linhas do triângulo de Pascal:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
```

As linhas e colunas do triângulo são numeradas começando ambas em 0 (zero). Assim, para um triângulo com n linhas e k colunas, o número que se encontra na última linha e última coluna terá índices $n-1$ e $k-1$.

Cada elemento do triângulo pode ser construído de forma recursiva, utilizando apenas informação da linha anterior. Seja $\binom{n}{k}$ o elemento da n-ésima linha, k-ésima coluna do triângulo. O valor de tal elemento é dado pela seguinte equação:

$$\binom{n}{k} \triangleq \begin{cases} 1 & \text{se } n = k = 0 \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{se } 0 < n \wedge 0 \leq k \leq n \end{cases}$$

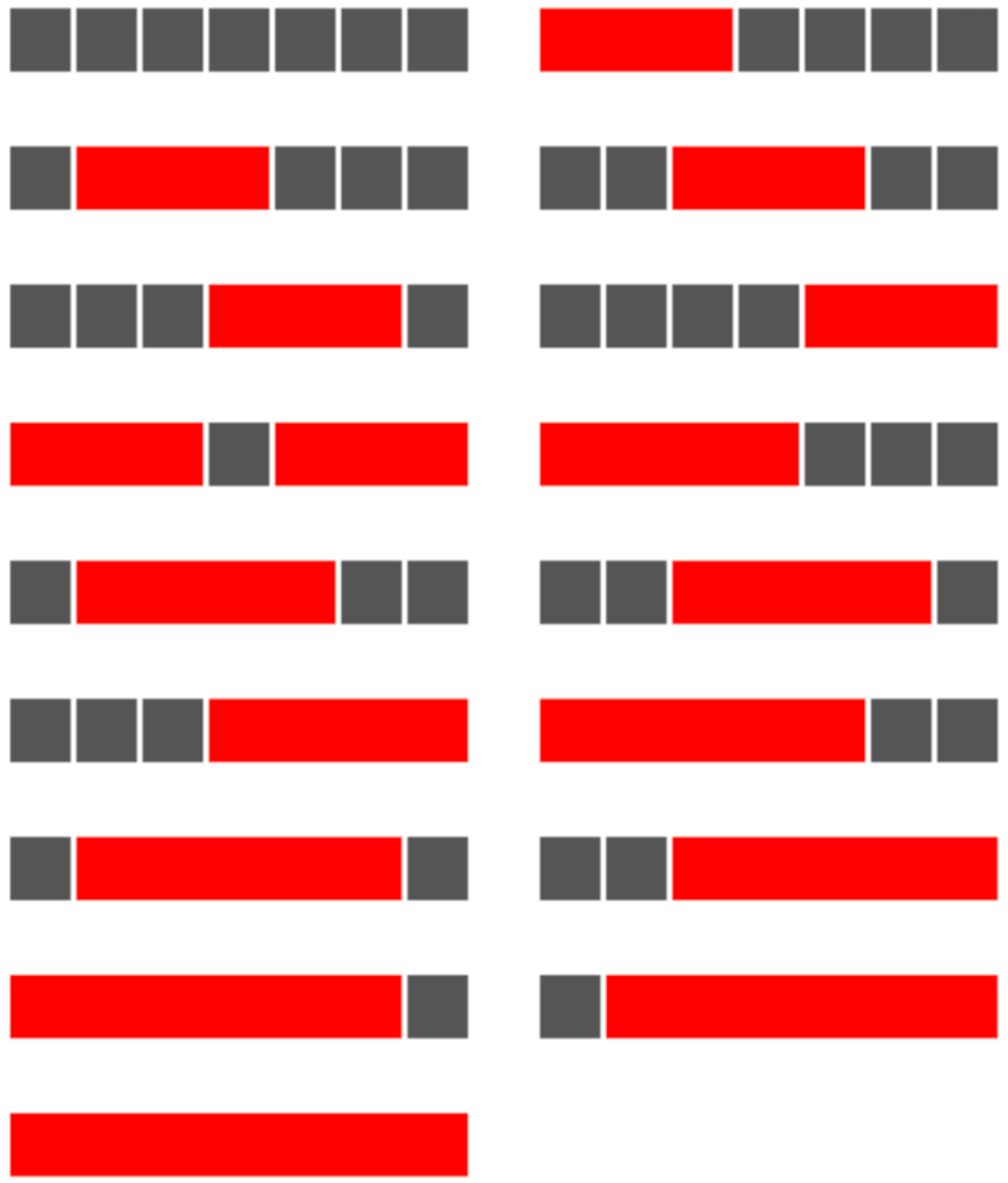
Querido, a Indução Mudou a Casa

A Ana e o Bernardo vão começar obras de remodelação da cozinha. Uma das ideias que gostariam de implementar é colocar um friso de azulejos ao longo de toda a parede. Sendo ambos grandes amantes da decoração de interiores, acordaram nas seguintes regras de estética:

1. os azulejos do friso só deverão ser de cor vermelha ou preta;
2. cada bloco de azulejos vermelhos deve ter pelo menos 3 unidades consecutivas;
3. dois blocos de azulejos vermelhos (que podem ser de tamanhos diferentes) devem estar separados por pelo menos um azulejo preto.

Antes de começarem o trabalho, e conhecendo o comprimento do friso, a Ana e o Bernardo gostariam de saber quantas formas distintas existem de preencher o friso, respeitando as regras que estabeleceram.

Tomando como exemplo um friso de tamanho 7, existem as seguintes 17 formas diferentes de preencher o friso:



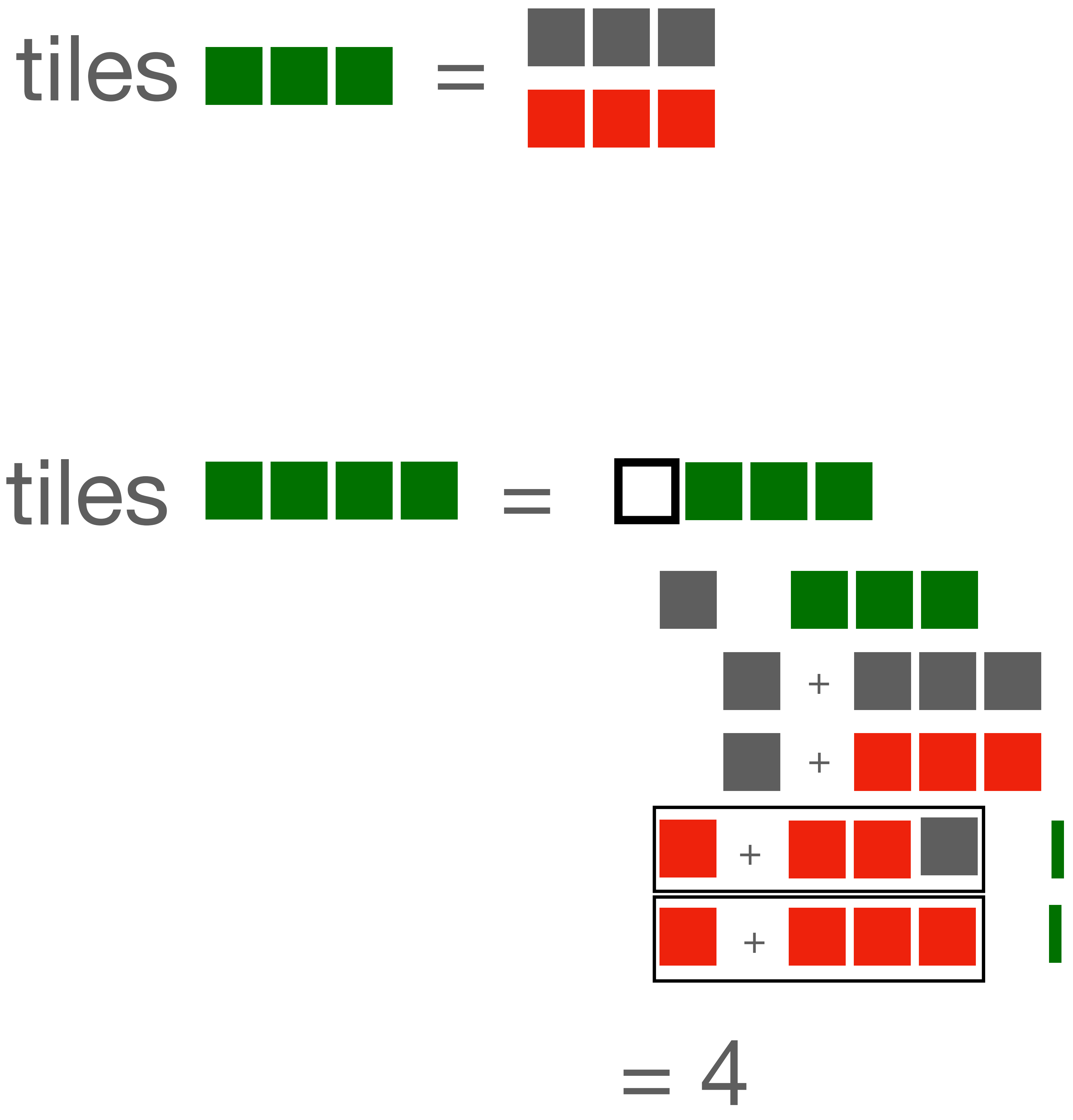
Casos Base (0, 1, 2, 3)

Se não houver espaço para pelo menos 3 azulejos, o número de hipóteses é 1.

Se houver exactamente 3, há duas hipóteses.

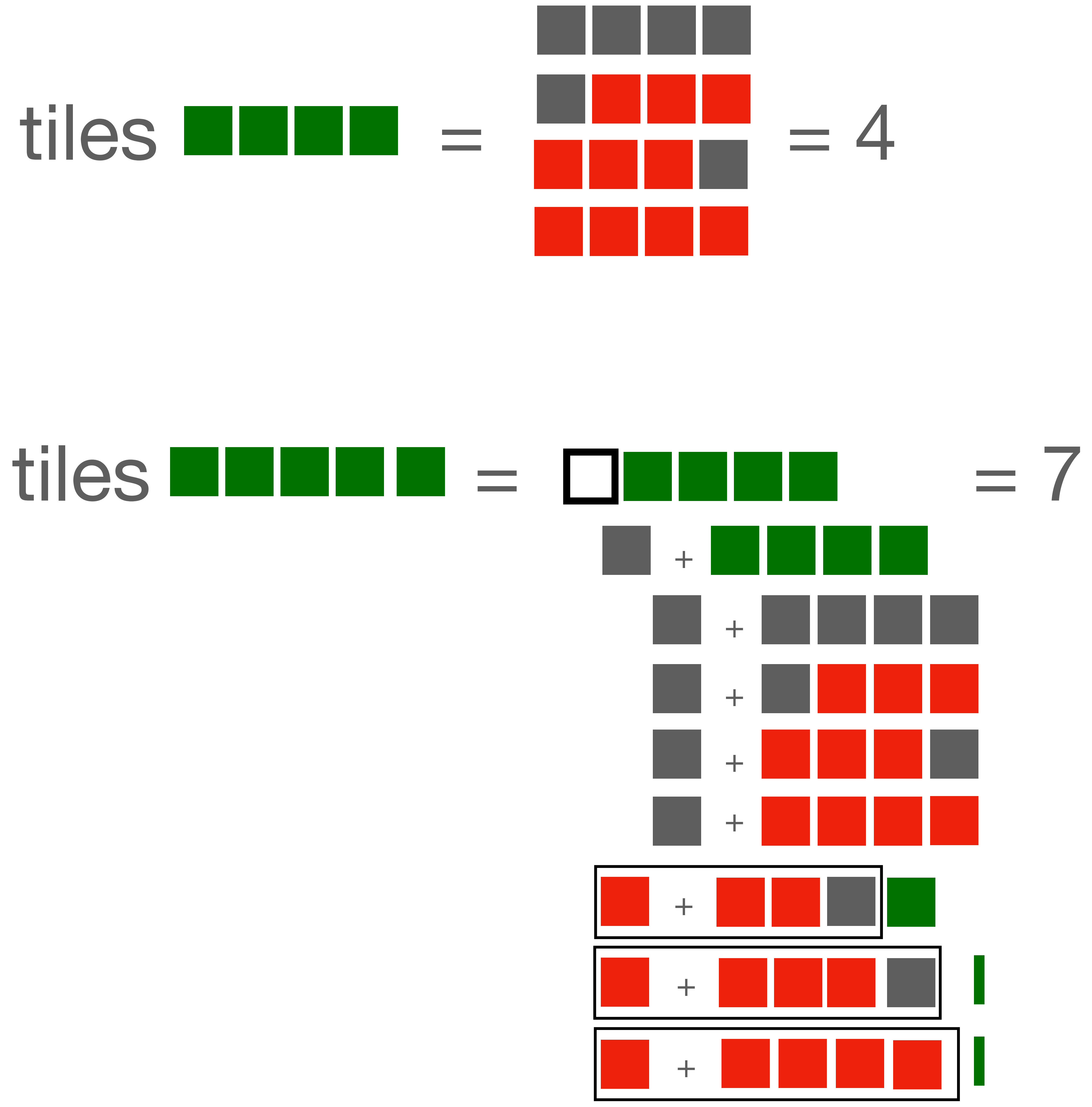
Caso n = 4, se tivermos a solução para n = 3?

Para um numero de azulejos (n) maior que 3, podemos assumir que sabemos resolver para (n-1).



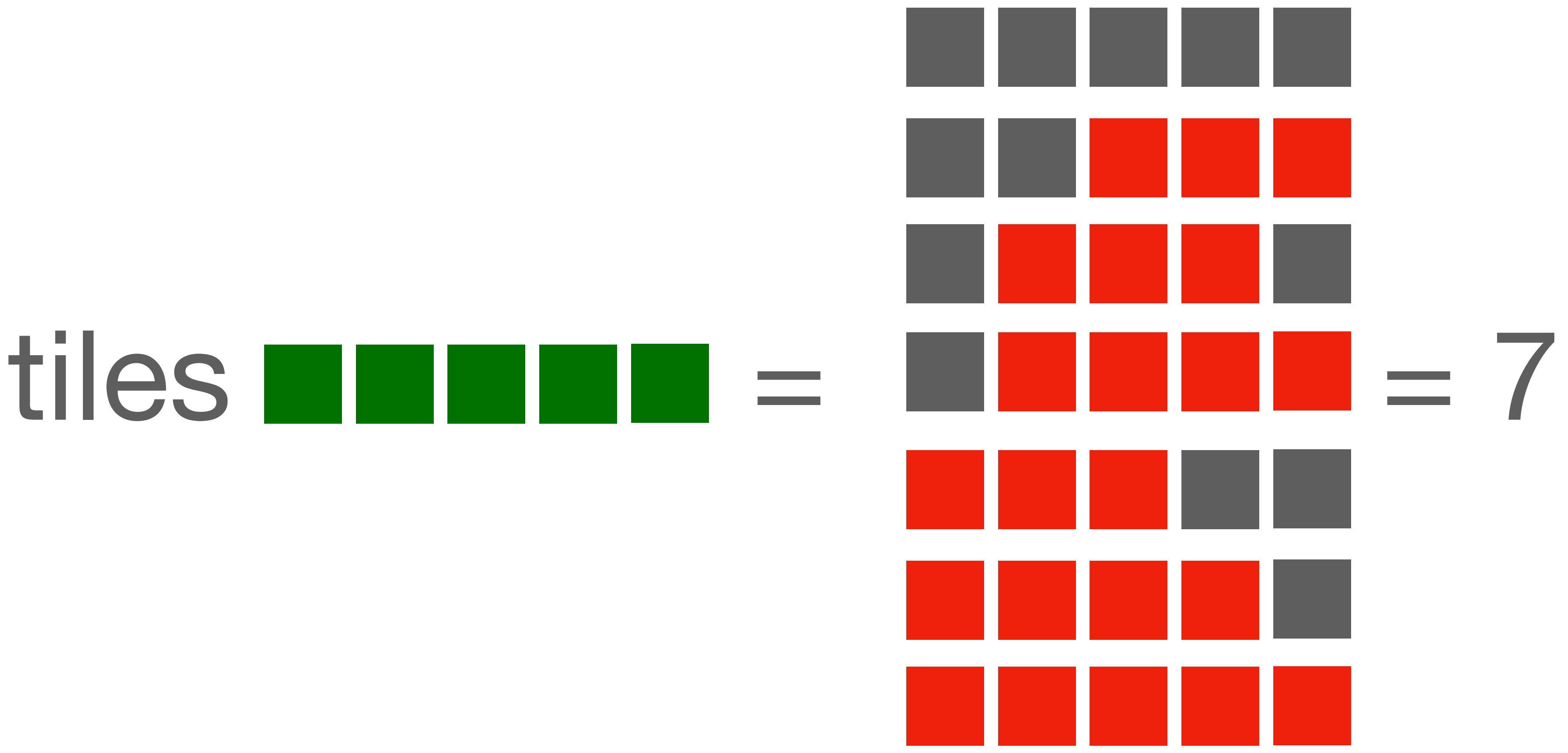
O primeiro azulejo ou é cinzento, ou é encarnado.
Se for cinzento, as soluções são as de um azulejo cinzento combinado com as soluções de (n-1).
Se for encarnado, então existem sequências de pelo menos 3, até ao limite dos azulejos, combinado com as possibilidades dos restantes azulejos, sendo que há um de intervalo.

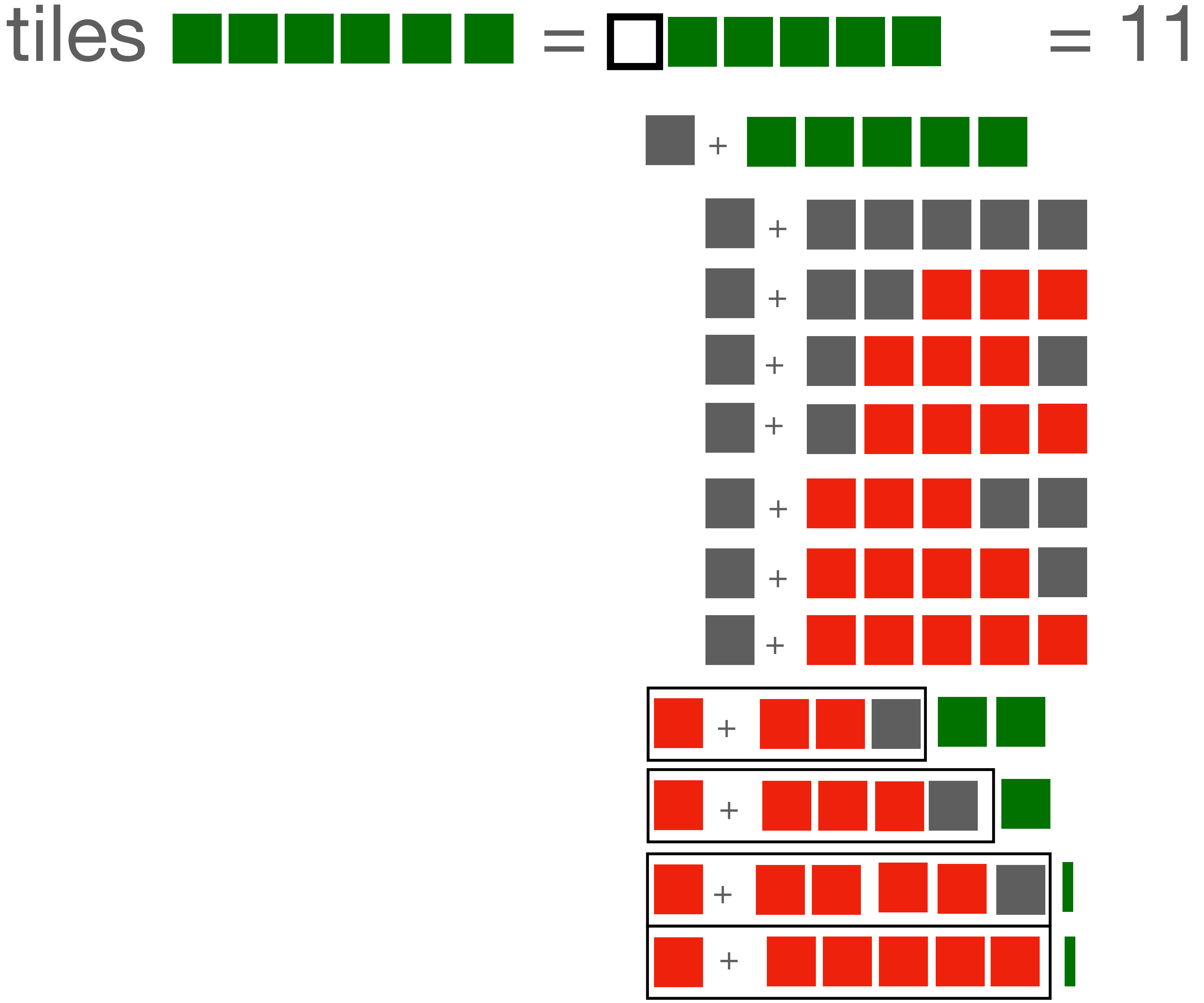
Caso n = 5, se tivermos a solução para n = 4?



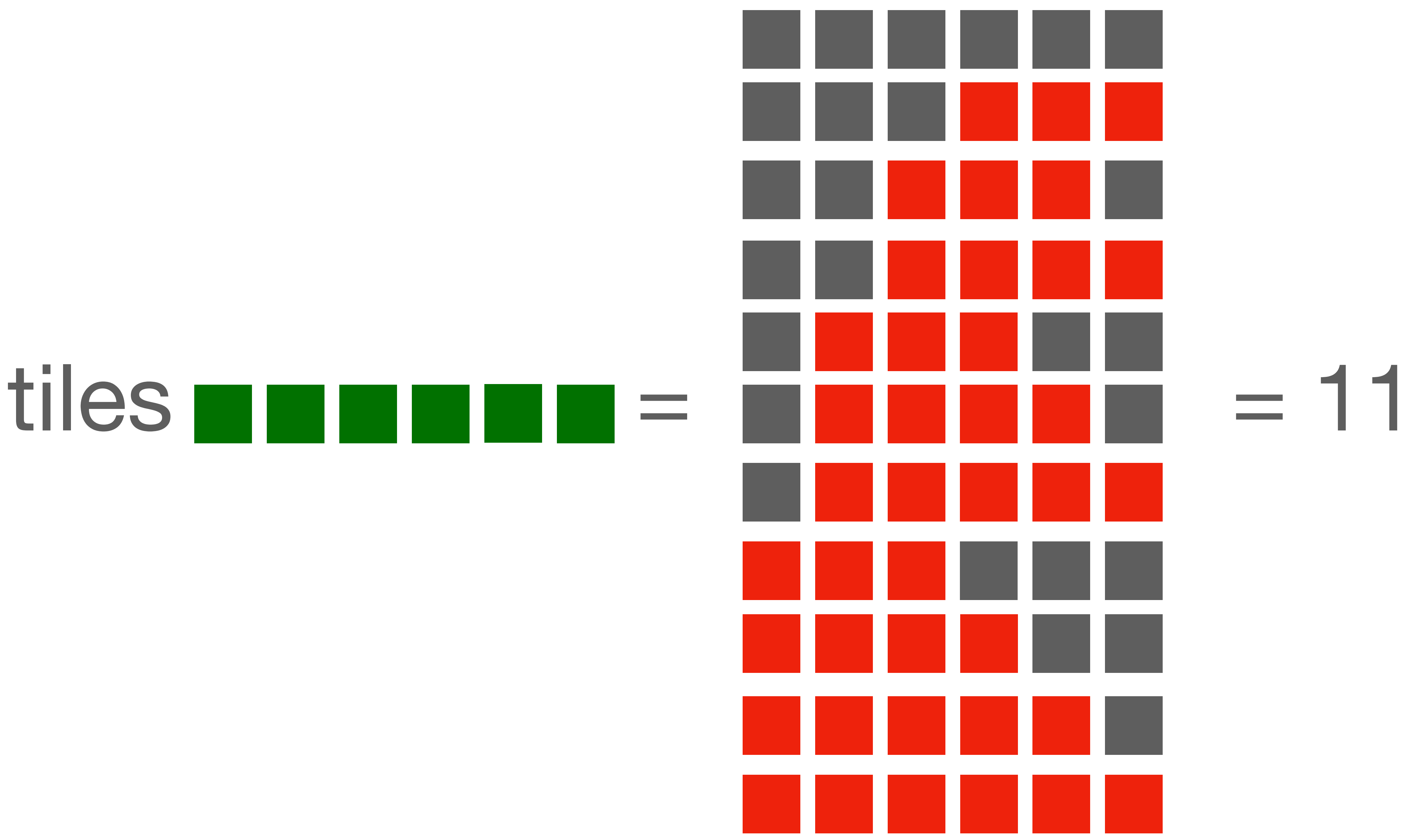
O primeiro azulejo ou é cinzento, ou é encarnado.
Se for cinzento, as soluções são as de um azulejo cinzento combinado com as soluções de (n-1).
Se for encarnado, então existem sequências de pelo menos 3, até ao limite dos azulejos, combinado com as possibilidades dos restantes azulejos, sendo que há um de intervalo.

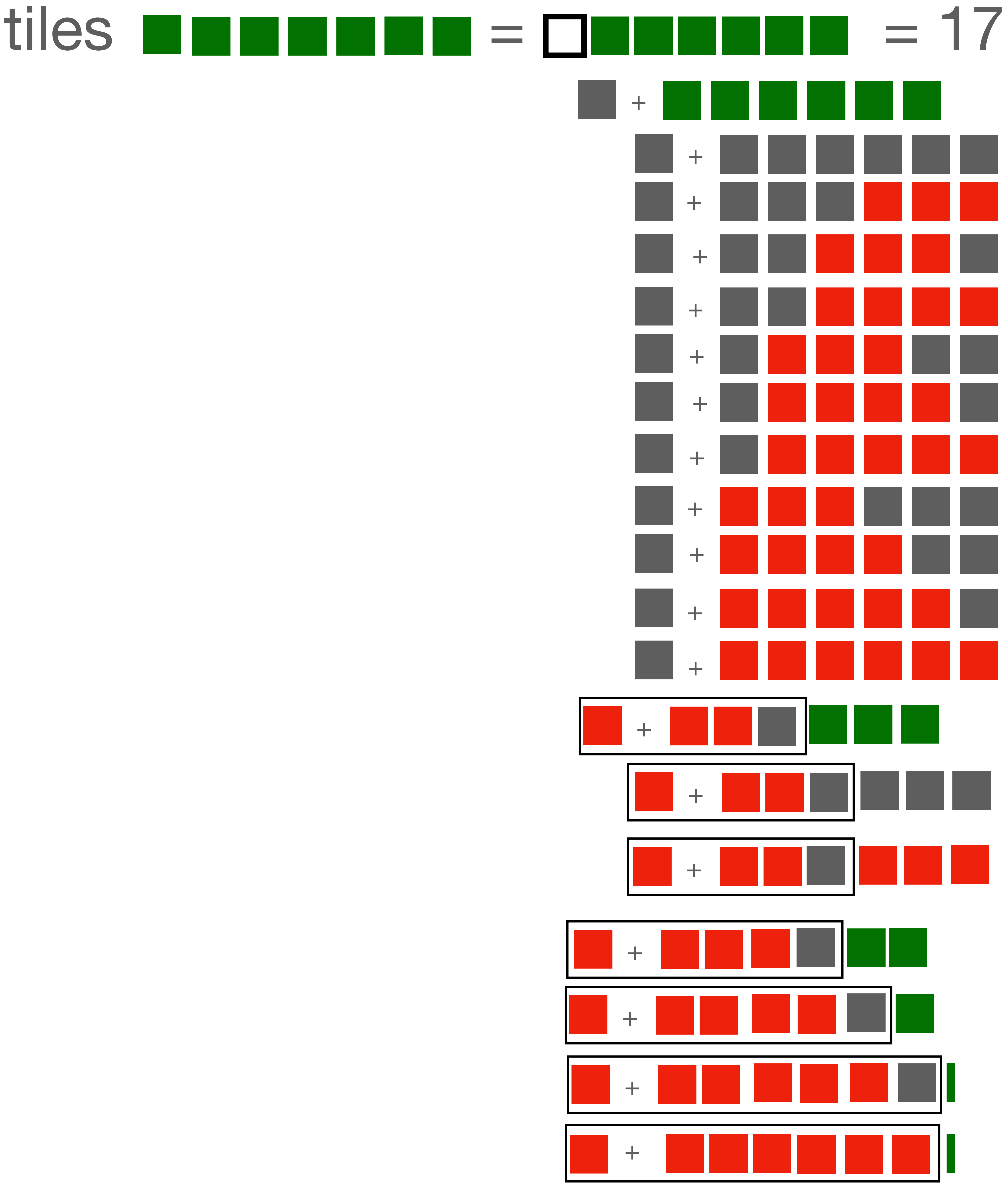
Caso n = 6, se tivermos a solução para n = 5?





Caso n = 7, se tivermos a solução para n = 6?





Intruções de submissão

- Na semana que vem haverá instruções