1 Enumerating vs. Counting

1.1 Permutation of Letters

Example: How many words can we make out of the letters A B C using each letter once?

- ABC
- BAC
- CAB
- ACB
- BCA
- CBA

When we list all objects as above we call it **enumeration**, whereas **counting** is only concerned with the total number of objects. If we consider the example above, how many words would be possible for A B C D?

It's best to find a formula, as using it is a very efficient way to count Objects. For $n = 4$ letters we end up with 24 permutations.

The formula for the amount of different words with n letters is $n!$

1.2 Points in Convex Position

How many crossing-free spanning paths exist for n points on convex position?

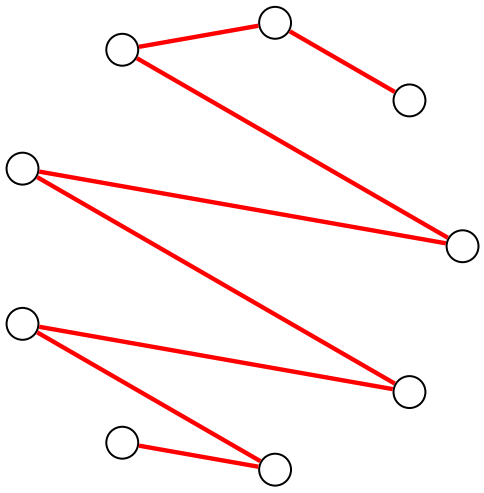


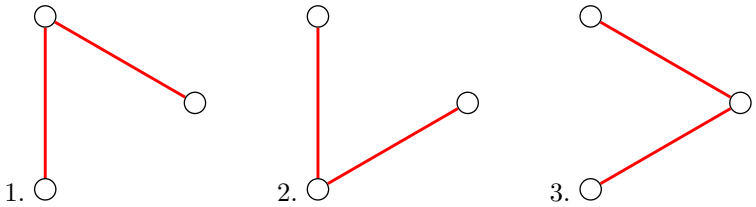
Figure 1: An example illustrating one possibility of a spanning path for $n = 9$ points

For $n = 1$ points the definition of the spanning path is unclear, in some cases it is considered as path with the size 1 and in others with size 0.

Let's look at some examples for $n > 1$ and try to determine a suitable formula.

- **n = 2**

- **n = 3**



- **n = 4**

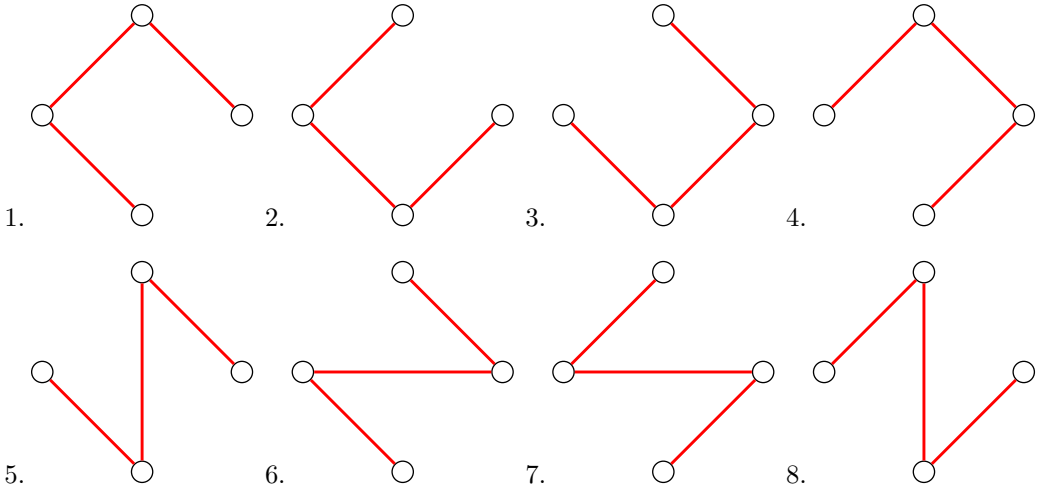


Figure 2: Enumeration of crossing free spanning paths up to $n = 4$

As we can see from this example, enumeration can become a tedious and error prone task very fast. Can you list all paths for $n = 5$?

It is better to abstract the problem and find an inductive solution. When constructing the path we start with a point, and from it we only see two immediate choices. After one of those points is added, we have two choices again. This goes on for a while until $n - 2$.

$$\underbrace{2 \cdot 2 \cdot 2 \cdots 2 \cdot 2}_{n-2 \text{ times}} = 2^{n-2}$$

Now in order to construct all paths we need to start at all possible points, when we do that however a double count occurs.

$$n \cdot 2^{n-2} \Rightarrow \frac{n \cdot 2^{n-2}}{2} \Rightarrow n \cdot 2^{n-3} \text{ for } n \geq 2$$

We can use this formula to find the number of crossing-free spanning paths for $n = 5$, which gives us $5 \cdot 2^2 = 20$.

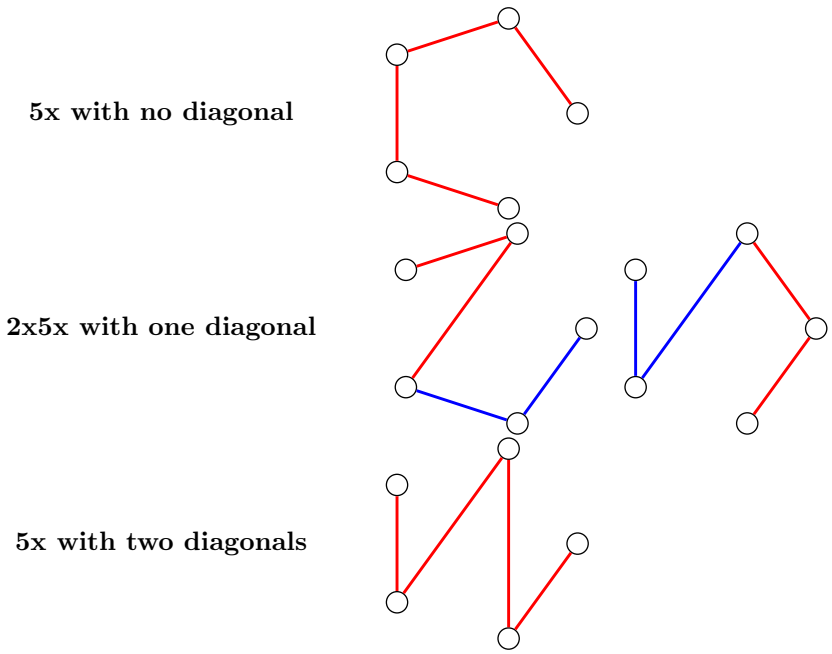


Figure 3: Another method of enumeration, do not explicitly list similar objects

2 Polyominos

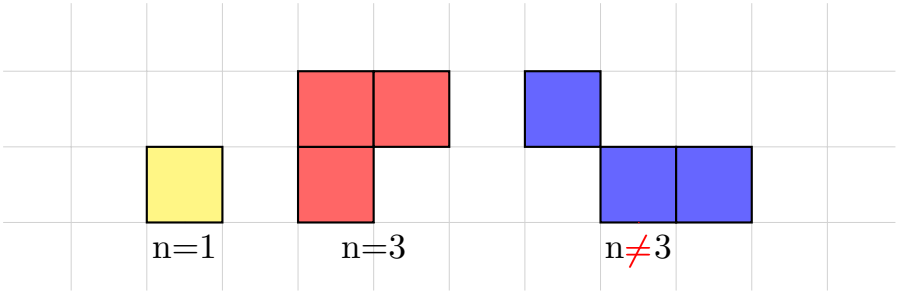


Figure 4: A polyomino of size n consists of n unit squares connected via edges, aligned on a grid

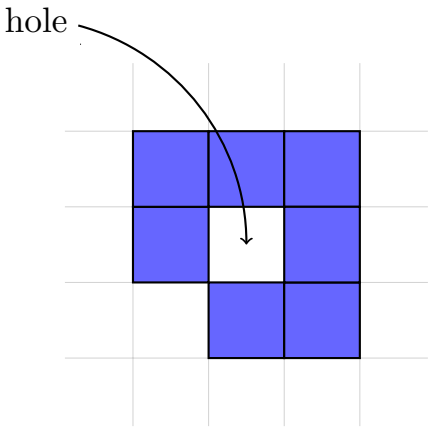


Figure 5: A polyomino with a hole inside. Polyominos without holes are a special case and they are called animals.

How many n-polyominos do exist? We have to define how to count them, that means defining the parameters that decide when two polyominos are regarded as one and the same. Polyominos can be compared using three operations:

1. **Translation**: Move one polyomino on top of another, if they overlap, they are the same.
2. **Rotation**: Additionally rotate one polyomino, if there is one rotation that makes them overlap, they are the same.
3. **Reflection**: Move in the 3^{rd} dimension, mirroring the polyomino.

With these operations polyominos can be classified into these two groups:

- **Fixed polyominos**: Only translation is allowed.
- **Free polyominos**: Translation, rotation and reflection is allowed.

n	# fixed	# free
1	1	1
2	2	1
3	6	2
4	19	5
5	63	12
⋮	⋮	⋮

Figure 6: How many n-polyominos do exist?

What is the formula for generating all n-polyominos? Look at a step from $n \to n+1$, we can add one unit square to all surfaces, in the absolute worst case (a straight polyomino) that means $2n+2$ possibilities. How can we best deal with duplicates?

Approach 1: Generate all new polyominos, then compare them all. For size $n+1$, k polyominos are generated. $\Rightarrow O(\binom{k}{2} \cdot n) = O(k^2 \cdot n)$

Problem: $k >> n$

Approach 2: Fingerprinting. Build a vector of a polyomino that is given by the coordinates of the squares.

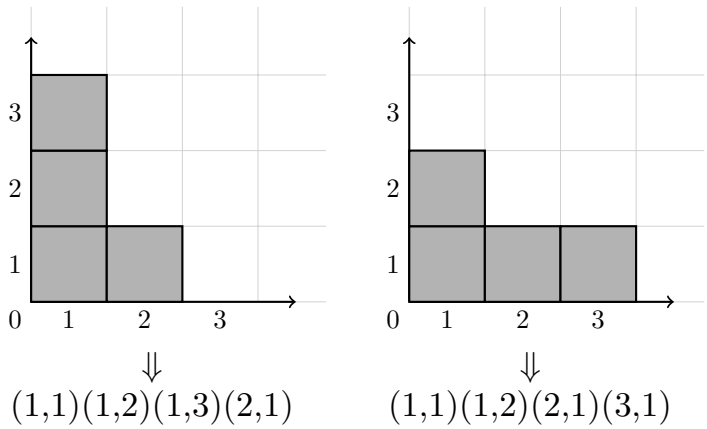


Figure 7: Polynomios and their respective vector fingerprint.

Compute all eight polyonimos (4x rotation & 2x reflection) and take the lexicographical minimum of the vector form!

Time for the fingerprint: $O(n)$

$\Rightarrow$ Compute fingerprints of all generated polyominos: $O(k \cdot n)$

$\Rightarrow$ Sort all fingerprints: $O(k \cdot log(k) \cdot n)$, duplicates are neighboured in the sorting and removing takes $O(k \cdot n) \Rightarrow$ the total runtime is $O(k \cdot log(k) \cdot n)$

No formula is known for the number of fixed/free polyominos.

$$\lim_{n\to\infty} \frac{\#(n+1) \text{ polyominos}}{\#n \text{ polyominos}} = \text{some constant } c$$

The number of polyominos goes to $\Theta(c^n)$ and we know that $4.00253 \leq c \leq 4.65$.

3 Pigeonhole-Principle