**Recurrence Relation**

There are two important things that one needs to figure out before implementing a recursive function:

* recurrence relation: the relationship between the result of a problem and the result of its subproblems.
* base case: the case where one can compute the answer directly without any further recursion calls. Sometimes, the base cases are also called *bottom cases*, since they are often the cases where the problem has been reduced to the minimal scale, *i.e.* the bottom, if we consider that dividing the problem into subproblems is in a top-down manner.

Once we figure out the above two elements, to implement a recursive function we simply call the function itself according to the recurrence relation until we reach the base case.

To explain the above points, let's look at a classic problem, Pascal's Triangle:

Pascal's triangle are a series of numbers arranged in the shape of triangle. In Pascal's triangle, the leftmost and the rightmost numbers of each row are always 1. For the rest, each number is the sum of the two numbers directly above it in the previous row.

Here's the illustration of the Pascal's Triangle with 5 rows:

1 / 14

Given the above definition, one is asked to generate the Pascal's Triangle up to a certain number of rows.

A screenshot of a computer

AI-generated content may be incorrect.

Recurrence Relation

Let's start with the recurrence relation within the Pascal's Triangle.

First of all, we define a function f(i,j)*f*(*i*,*j*) which returns the number in the Pascal's Triangle in the i-th row and j-th column.

We then can represent the recurrence relation with the following formula:

f(i,j)=f(i−1,j−1)+f(i−1,j)*f*(*i*,*j*)=*f*(*i*−1,*j*−1)+*f*(*i*−1,*j*)

Base Case

As one can see, the leftmost and rightmost numbers of each row are the base cases in this problem, which are always equal to 1.

As a result, we can define the base case as follows:

f(i,j)=1wherej=1orj=i*f*(*i*,*j*)=1*wherej*=1*orj*=*i*

Demo

As one can see, once we define the recurrence relation and the base case, it becomes much more intuitive to implement the recursive function, especially when we formulate these two elements in terms of mathematical formulas.

Here is an example of how we can apply the formula to recursively calculate f(5,3)*f*(5,3), *i.e.* the 3rd number in the 5th row of the Pascal Triangle:

1 / 17

Starting from f(5,3)*f*(5,3), we can break it down as f(5,3)=f(4,2)+f(4,3)*f*(5,3)=*f*(4,2)+*f*(4,3), we then call f(4,2)*f*(4,2) and f(4,3)*f*(4,3) recursively:

* For the call of f(4,2)*f*(4,2), we could extend it further until we reach the base cases, as follows:

f(4,2)=f(3,1)+f(3,2)=f(3,1)+(f(2,1)+f(2,2))=1+(1+1)=3*f*(4,2)=*f*(3,1)+*f*(3,2)=*f*(3,1)+(*f*(2,1)+*f*(2,2))=1+(1+1)=3

* For the call of f(4,3)*f*(4,3), similarly we break it down as:

f(4,3)=f(3,2)+f(3,3)=(f(2,1)+f(2,2))+f(3,3)=(1+1)+1=3*f*(4,3)=*f*(3,2)+*f*(3,3)=(*f*(2,1)+*f*(2,2))+*f*(3,3)=(1+1)+1=3

* Finally we combine the results of the above subproblems:

f(5,3)=f(4,2)+f(4,3)=3+3=6*f*(5,3)=*f*(4,2)+*f*(4,3)=3+3=6

Next

In the above example, you might have noticed that the recursive solution can incur some duplicate calculations, *i.e.* we compute the same intermediate numbers repeatedly in order to obtain numbers in the last row. For instance, in order to obtain the result for the number f(5,3)*f*(5,3), we calculate the number f(3,2)*f*(3,2) twice both in the calls of f(4,2)*f*(4,2) and f(4,3)*f*(4,3).

We will discuss how to avoid these duplicate calculations in the next chapter of this Explore card.

Following this article, you will find exercises for problems related to Pascal's Triangle.