Week 6 — Sheet 5

Algorithms and Data Structures

30.10.2023 — Georg Hasebe

Debriefing of Submissions

$$\sum_{i=1}^{\log_2(n)} \log_2(n) - i + 1$$

Notice that if $i = 1$ then $\log_2(n) - i + 1 = \log_2(n)$, and if $i = \log_2(n)$ then $\log_2(n) - i + 1 = 1$.

$$\sum_{i=1}^{\log_2(n)} \log_2(n) - i + 1$$

Notice that if $i = 1$ then $\log_2(n) - i + 1 = \log_2(n)$, and if $i = \log_2(n)$ then $\log_2(n) - i + 1 = 1$.

We sum from 1 to $\log_2(n)$, using Gauss' sum formula we get:

$$\sum_{i=1}^{\log_2(n)} \log_2(n) - i + 1 = \frac{\log_2(n)(\log_2(n) + 1)}{2}$$

(b) Describe an algorithm that determines the *smallest* integer $T \in \mathbb{N}$ such that $f(T) \geq N$, making $O(\log T)$ function calls to f. Prove that your algorithm is correct, and uses at most the desired number of function calls.

Hint: *Consider using a two-step approach. In the first step, apply the algorithm of part (a). For the second step, modify the binary search algorithm and apply it to the array* $\{1, 2, \ldots, T_{\text{ub}}\}$. *Use helper variables* $i_{\text{low}}, i_{\text{high}} \in \mathbb{N}$, *that satisfy* $i_{\text{low}} \leq T \leq i_{\text{high}}$ *at all times during the algorithm. In each iteration, update* i_{low} *and/or* i_{high} *so that the number of remaining options for* T *is halved.*

Solutions didn't really use the array they described in the hint. Careful because of array initialisation.

Exercise Sheet 5

Debriefing of Exercise Sheet 5

Theory Recap

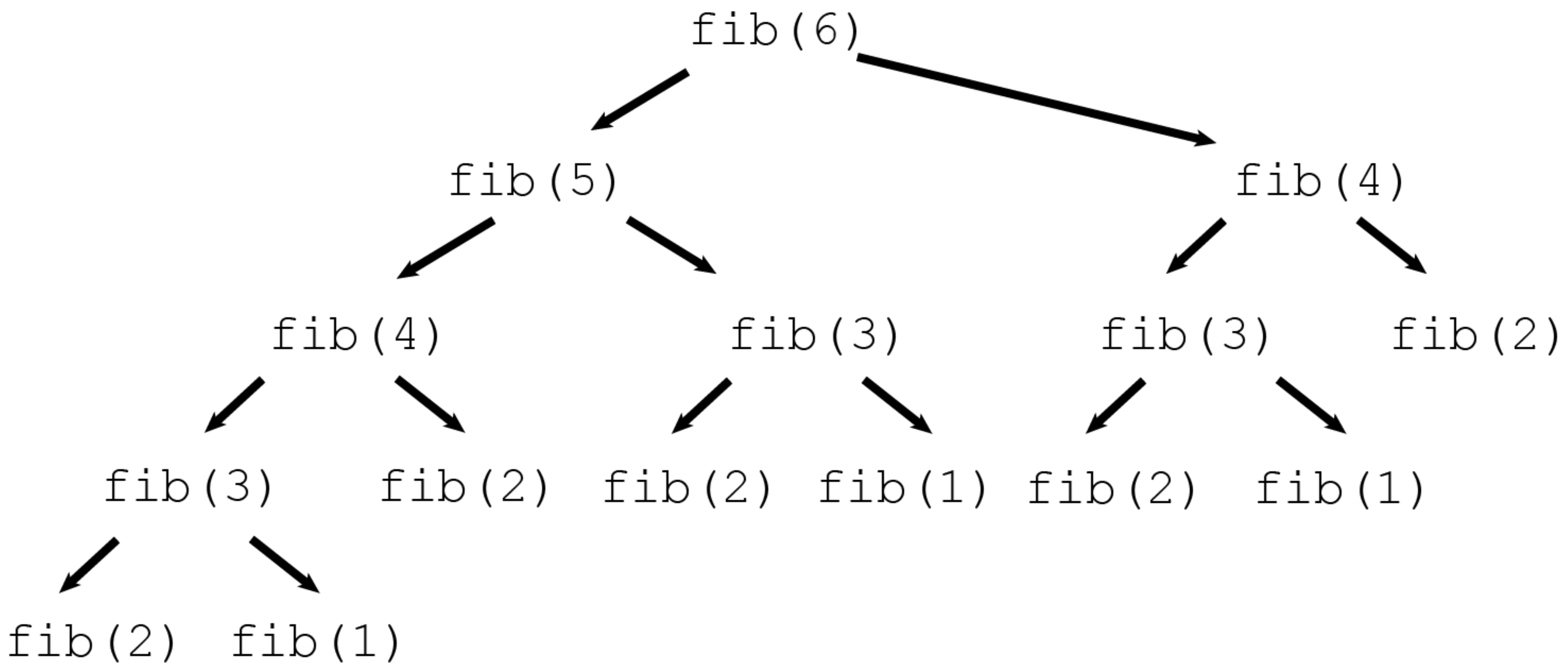
Dynamic Programming (DP)

What is DP and why is it useful?

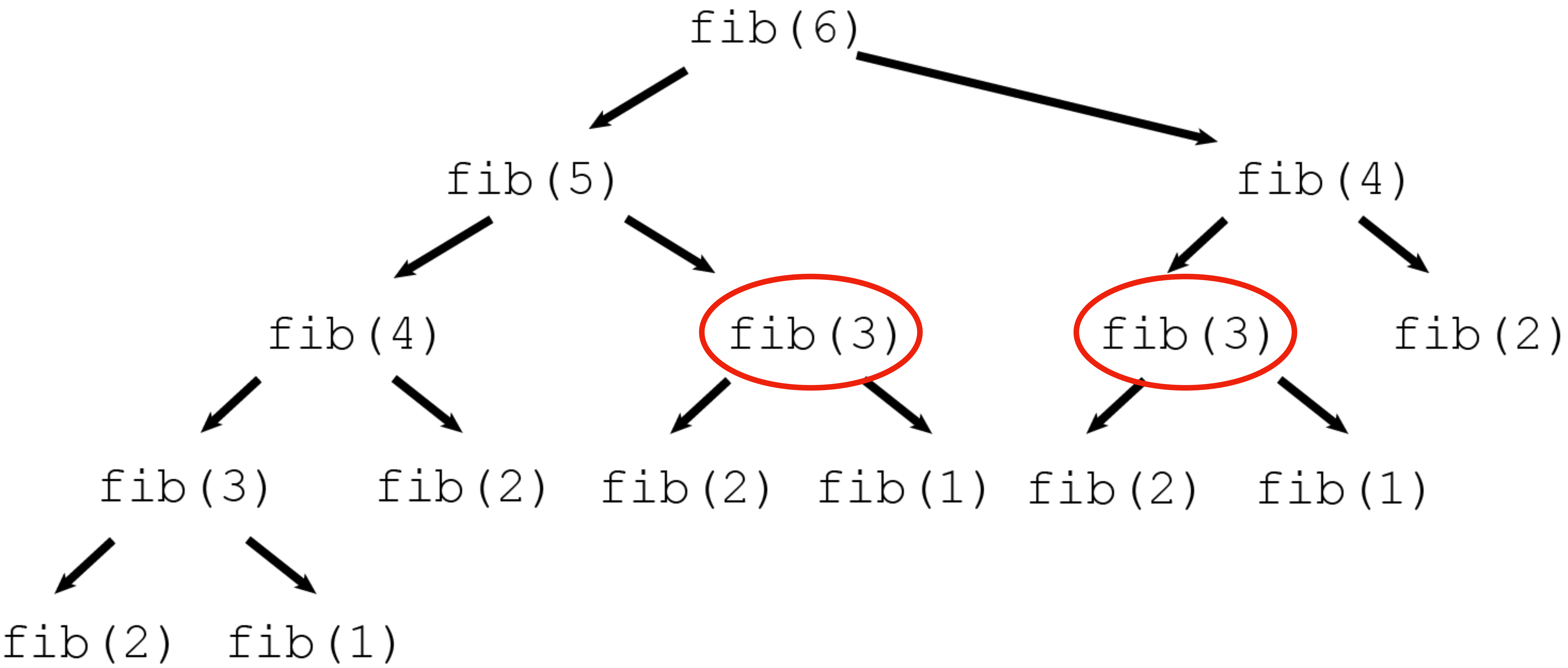
In Dynamic Programming, we try to **simplify** a complex problem by breaking it down to simpler **sub-problems** in a **recursive manner.**

Why is it useful?
Example

Consider the Fibonacci Example from the lecture.



https://willrosenbaum.com/teaching/2021s-cosc-112/notes/recursive-fibonacci/



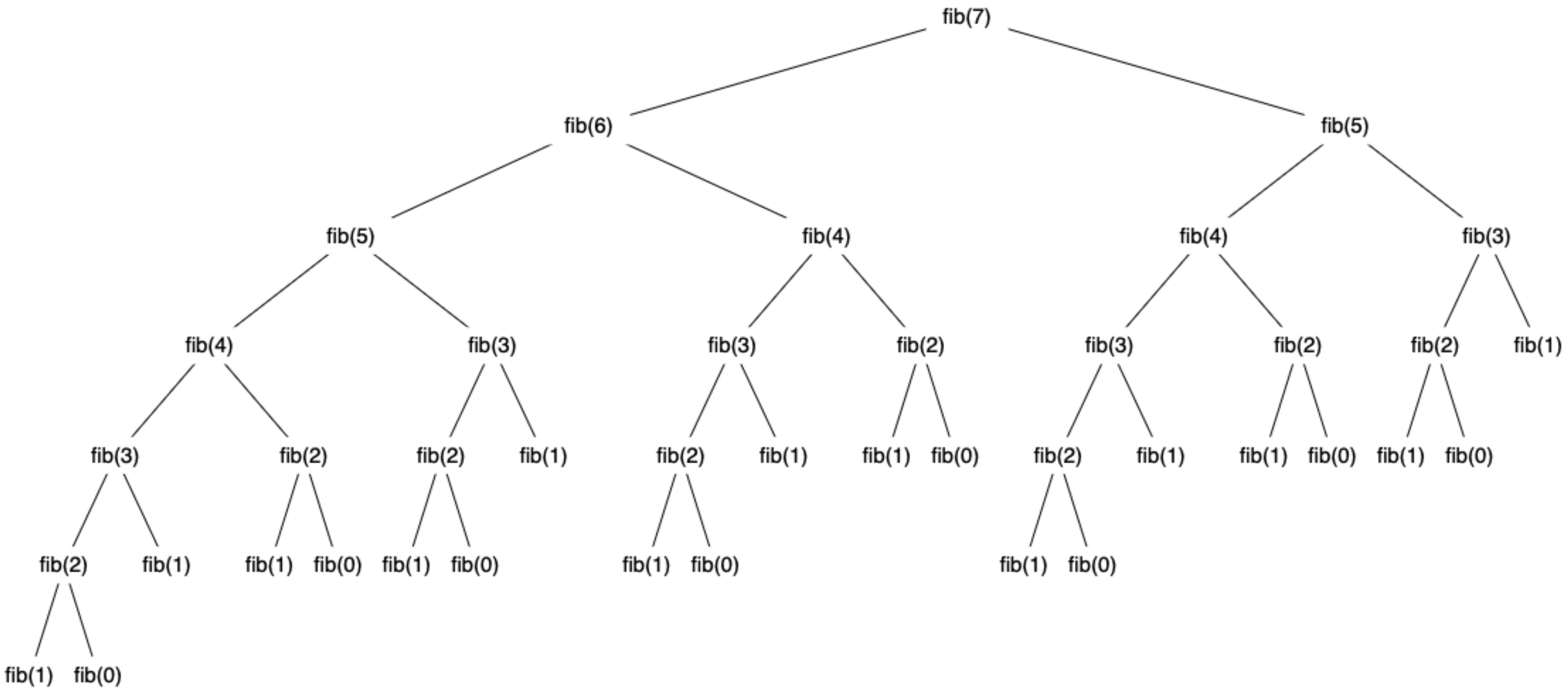
https://willrosenbaum.com/teaching/2021s-cosc-112/notes/recursive-fibonacci/

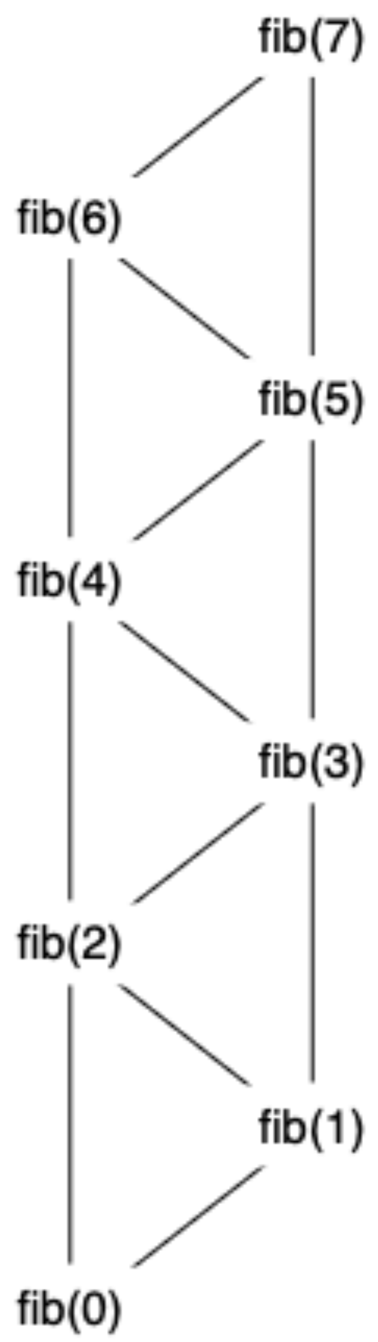
Using DP we can make improvements

https://programming.guide/dynamic-programming-vs-memoization-vs-tabulation.html

Doesn’t seem very useful?

$O(2^n)$ $O(n)$





https://programming.guide/dynamic-programming-vs-memoization-vs-tabulation.html

Top-Down vs Bottom-Up

```
fib_mem(n) {
    if (mem[n] is not set)
        if (n < 2) result = n
        else result = fib_mem(n-2) + fib_mem(n-1)
        mem[n] = result
    return mem[n]
}
```

Top-Down

(Memoization)

```
fib_tab(n) {
    mem[0] = 0
    mem[1] = 1
    for i = 2...n
        mem[i] = mem[i-2] + mem[i-1]
    return mem[n]
}
```

Bottom-Up

(Tabulation)

https://programming.guide/dynamic-programming-vs-memoization-vs-tabulation.html

DP is not easy.

But it can be mastered through practice!

How to solve DP problems?

1. Understand the problem
2. Find/Design recurrence relation (i.e. how the smaller sub-problems relate to each other)
3. Translate it to code (CodeExpert)

Hardest part?

Finding the recurrence relation

- Go through examples
- Construct your own examples
- Once you have an idea, try to simulate it

Minimal Editing Distance

Step 1: Understand the Problem

Minimal Editing Distance

(Edit Distance, Levenshtein Distance)

Given two strings $A[1 \ldots n]$ and $B[1 \ldots m]$, what is the minimum number of single-character edits (insertion, deletion, substitution) required to turn one word into the other word?

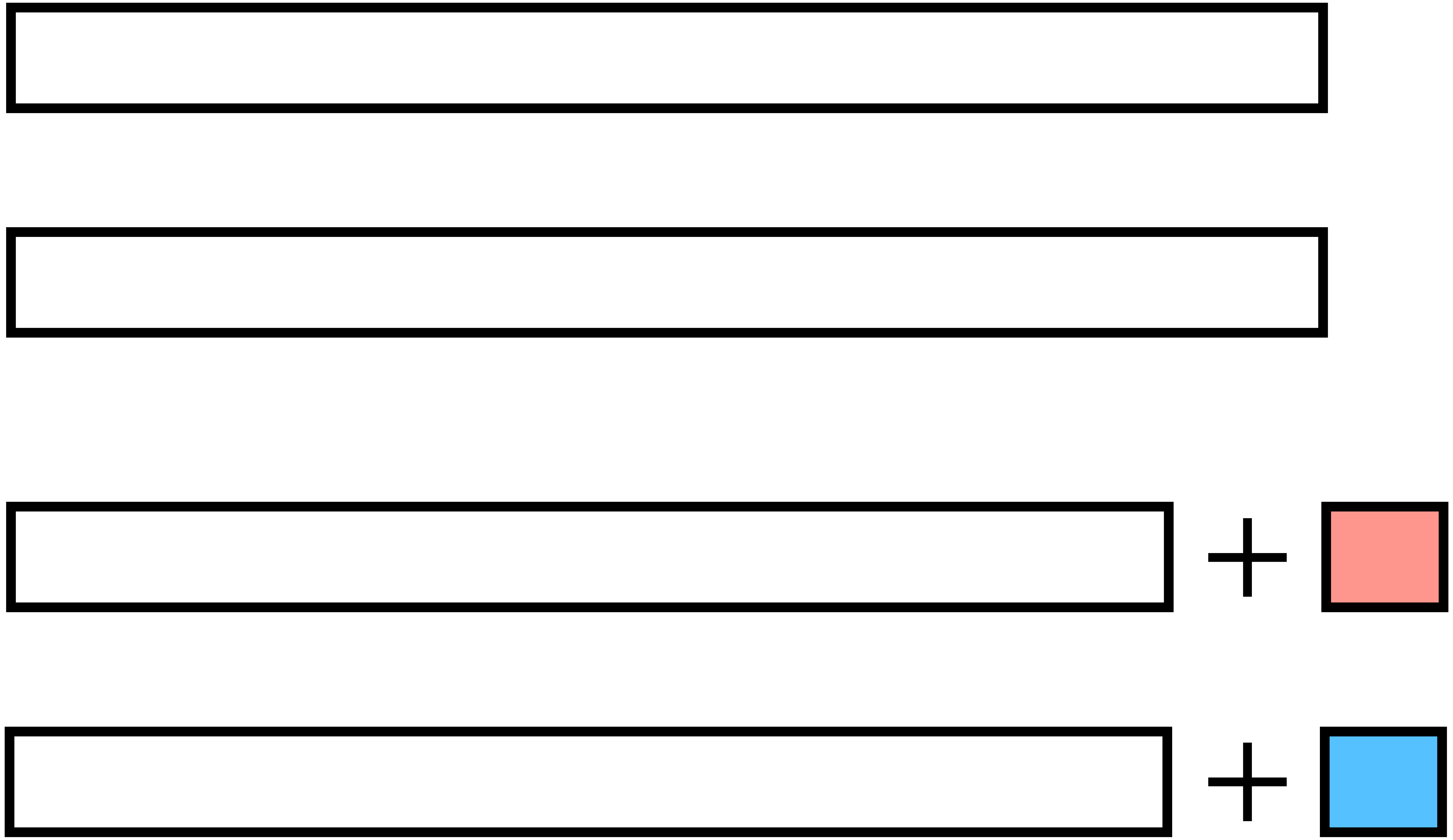
Step 2: Design Recurrence Relation

$$A[1 \ldots n] =$$

$$B[1 \ldots m] =$$

$$A[1 \ldots n-1] + A[n] =$$

$$B[1 \ldots m-1] + B[m] =$$



What if

?

If  , then for the minimal edit distance (MED) we know:

$$\text{MED}(A[1\ldots n], B[1\ldots m]) = \text{MED}(A[1\ldots n-1], B[1\ldots m-1])\,.$$

If $\neq$, then what do we know?

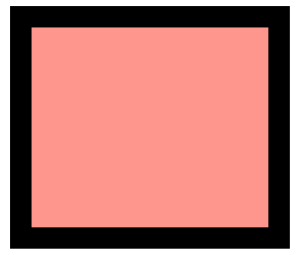
If we replace (**substitution**) with we again have $=$

With the difference being that we made one edit operation, thus we have

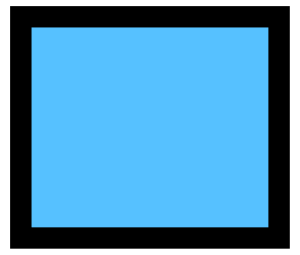
$$\text{MED}(A[1\ldots n], B[1\ldots m]) = 1 + \text{MED}(A[1\ldots n-1], B[1\ldots m-1])\,.$$

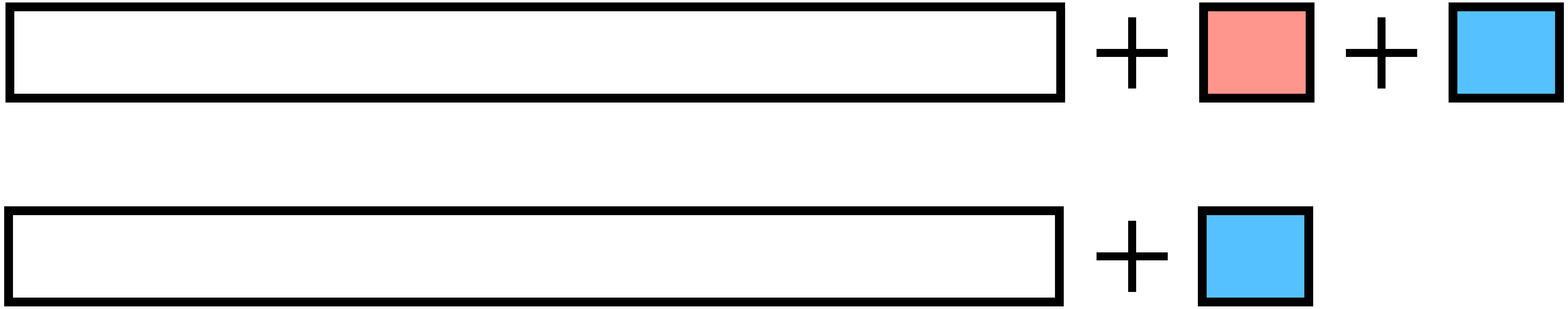
In some cases though, it can be beneficial to either delete or insert characters depending the length of A and B.

For example, if we have $A =$ ABCDX and $B =$ ABCD then we simply delete X from A or insert X into B.

If we delete  from A, we reduce the length of A by one and only consider $A[1 \ldots n-1]$ for further inspection. However B doesn't change at all, and therefore we still have to consider $B[1 \ldots m]$. Thus we get:

$$\text{MED}(A[1 \ldots n], B[1 \ldots m]) = 1 + \text{MED}(A[1 \ldots n-1], B[1 \ldots m])\,.$$

If we insert  at the end of A, we get:



Since the last characters now match, we only consider $B[1 \ldots m-1]$ for further inspection. Because we inserted a single character into A, we again consider $A[1 \ldots n]$ (because the initial string, i.e. the long white bar and the small red bar, remains the same). Thus we get:

$$\text{MED}(A[1 \ldots n], B[1 \ldots m]) = 1 + \text{MED}(A[1 \ldots n], B[1 \ldots m-1]) \,.$$

So far we have only considered some modifications at the end of the strings. Since we can delete/insert characters at any position of the string, we need to generalise our observations.

Let $i \in [1 \ldots n]$ and $j \in [1 \ldots m]$, then for the minimal editing distance of $A[1 \ldots i]$ and $B[1 \ldots j]$ we write MED(i, j). It is true that:

$$\text{MED}(i,j) = \min \left\{ \begin{array}{rl} 1 + \text{MED}(i-1,j) & \text{deletion} \\ 1 + \text{MED}(i,j-1) & \text{insertion} \\ \delta + \text{MED}(i-1,j-1) & \text{substitution} \end{array} \right\}$$

where $\delta = 1$ if $A[i] \neq B[j]$ and zero otherwise.

Base Cases

Now that we have established the recurrence relation, it remains to consider the base cases. Often times, these are just very simple cases, that don't require a lot of thinking, but they allow us to generate a solution.

Interactive Example:

$A =$ "TIGER" and $B =$ "ZIEGE"

MED(i,j)	-	T	I	G	E	R
-						
Z						
I						
E						
G						
E						

$$\text{MED}(i,j) = \min \begin{Bmatrix} 1 + \text{MED}(i-1,j) & \text{deletion} \\ 1 + \text{MED}(i,j-1) & \text{insertion} \\ \delta + \text{MED}(i-1,j-1) & \text{substitution} \end{Bmatrix}$$

where $\delta = 1$ if $A[i] \neq B[j]$ and zero otherwise.

MED(i,j)	-	T	I	G	E	R
-	0	1	2	3	4	5
Z	1					
I	2					
E	3					
G	4					
E	5					

$$\text{MED}(i,j) = \min \left\{ \begin{array}{rl} 1 + \text{MED}(i-1,j) & \text{deletion} \\ 1 + \text{MED}(i,j-1) & \text{insertion} \\ \delta + \text{MED}(i-1,j-1) & \text{substitution} \end{array} \right\}$$

where $\delta = 1$ if $A[i] \neq B[j]$ and zero otherwise.

MED(i,j)	-	T	I	G	E	R
-	0	1	2	3	4	5
Z	1	1	2	3	4	5
I	2					
E	3					
G	4					
E	5					

$$\text{MED}(i,j) = \min \left\{ \begin{array}{rl} 1 + \text{MED}(i-1,j) & \text{deletion} \\ 1 + \text{MED}(i,j-1) & \text{insertion} \\ \delta + \text{MED}(i-1,j-1) & \text{substitution} \end{array} \right\}$$

where $\delta = 1$ if $A[i] \neq B[j]$ and zero otherwise.

MED(i,j)	-	T	I	G	E	R
-	0	1	2	3	4	5
Z	1	1	2	3	4	5
I	2	2	1	2	3	4
E	3	3	2	2	2	3
G	4	4	3	2	3	3
E	5	5	4	3	2	3

$$\text{MED}(i,j) = \min \left\{ \begin{array}{rl} 1 + \text{MED}(i-1, j) & \text{deletion} \\ 1 + \text{MED}(i, j-1) & \text{insertion} \\ \delta + \text{MED}(i-1, j-1) & \text{substitution} \end{array} \right\}$$

where $\delta = 1$ if $A[i] \neq B[j]$ and zero otherwise.

Runtime? Space Complexity?

- the table is m by n, thus the space complexity is in $O(nm)$
- each cell can be computed in $O(1)$, we have $m \cdot n$ cells, thus the runtime is in $O(mn)$