# The Hofstadter Sequence – Solution

(Cătălin Frâncu)

(problema a fost pregătită pentru BOI 2003, deci citiţi descrierea soluţiei în limba engleză !)

$a$ = { 1, 3, 7, 12, 18, 26, 35, 45, 56, 69, 83, 98, 114, 131, 150, 170, 191, 213, ... }
$b$ = { 2, 4, 5, 6, 8, 9, 10, 11, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 27, 28, ... }

Let $\{b_n\}$ be the first-order differences of $\{a_n\}$, i.e. $b_k = a_{k+1} - a_k$ for all $k \geq 1$. Then $a_{k+1} = a_k + b_k$. To ensure that $\{a_k\} \cup \{b_k\}$ cover all the natural integers and do not overlap, we must enforce that

- $b_k = b_{k-1} + 1$, if $b_{k-1} + 1 \notin \{a_n\}$
- $b_k = b_{k-1} + 2$, if $b_{k-1} + 1 \in \{a_n\}$

The first condition ensures that all the integers are covered. The second condition ensures that $\{a_n\}$ and $\{b_n\}$ are disjoint. Thus we obtain $a = \{1, 3, 7, ...\}$ and $b = \{2, 4, 5, 6, ...\}$. Notice that $b_2 - b_1 = 2$ due to the second condition. We conclude that $\{b_n\}$ grows almost linearly, which implies that $\{a_n\}$ grows almost quadratically. This gives us an idea of which algorithms work in the given time and memory and which don't work.

Here is the straightforward solution that does not work: simply generate $a_k$ and $b_k$ as shown above. To determine if a number is in the sequence, run a binary search on the subsequence $\{a_1, a_2, ..., a_{k-1}\}$, which we have already generated. For example, we generate the terms 1, 3, 7, 12 and 18, after which $a_6$ can be 25 = 18 + 7 or 26 = 18 + 8. Since 7 is already in the sequence, it follows that $a_6 = 26$. The problem with this algorithm is that it needs to store all the terms that it generates, which can be 100,000,000. The memory requirements are prohibitive.

The correct solution is a little more complicated, but it requires quasi-constant memory. We only keep the last terms in memory, $a_k$ and $b_k$. To generate $a_{k+1}$, we set

- $a_{k+1} = a_k + b_k$
- $b_{k+1} = b_k + 1$ if $b_k + 1 \notin \{a_1, a_2, ..., a_{k-1}\}$ or
- $b_{k+1} = b_k + 2$ if $b_k + 1 \in \{a_1, a_2, ..., a_{k-1}\}$

The problem here is how to look up $b_k + 1$, since we do not keep the terms of $a$ in memory. Note, however, that the values we look up are in increasing order ($b_1 + 1, b_2 + 1, b_3 + 1, ...$). The idea is to construct another pair of sequences in parallel, $\{a'_n\}$ and $\{b'_n\}$. These sequences are identical to $\{a_n\}$ and $\{b_n\}$, but we will need to generate much fewer terms. Again, we will only store two numbers in memory, $a'_k$ and $b'_k$. The actual algorithm is:

```
1   a = 1, b = 2
2   for (i = 2; i ≤ n; i++)
3     a = a + b;
4     b = b + 1;
5     // Here we look up b in a'
6     generate terms in a' and b' until a' ≥ b
7     if (a' == b) b = b + 1
```

Because $\{a'_n\}$ grows quadratically, while $\{b'_n\}$ grows linearly, the first $\sqrt{n}$ terms of $\{a'_n\}$ will be enough to look up all the values of $\{b_n\}$.

To complete the algorithm, notice that line 6 of the algorithm contains a recurrence. How do we generate the terms of $\{b'_n\}$ since we do not keep the terms of $\{a'_n\}$ in memory? The answer is to generate yet another pair of sequences, $\{a''_n\}$ and $\{b''_n\}$, that will grow even slower. We will only need $\sqrt[4]{n}$ terms from these sequences. Thus, the final algorithm will generate multiple sequences; each sequence uses a shorter helper sequence with only $\sqrt{n}$ of the number of terms. The last one of this sequence will only contain one term, and therefore it will not need a helper sequence. The running time of this algorithm is linear. More precisely, the total number of terms generated in all the sequences is $n + \sqrt{n} + \sqrt{\sqrt{n}} + \sqrt{\sqrt{\sqrt{\sqrt{n}}}} + ...$ . For $n$ = 100,000,000, we will need 7 sequences.