Ackermann function

From Wikipedia, the free encyclopedia

In computability theory, the **Ackermann function** or **Ackermann-Péter function** is a simple example of a computable function that is not primitive recursive. The set of primitive recursive functions is a subset of the set of general recursive functions. Ackermann's function is an example that shows that the former is a strict subset of the latter.

The Ackerman function takes two natural numbers as arguments and yields another natural number, using the notation A (m,n). Its value grows rapidly; even for small inputs, for example A(*4,2*)[1] and A(*4,3*), the results are large numbers. These large numbers in the m=4 row can also be expressed using tetrations.

Contents

History

In the late 1920s, the mathematicians Gabriel Sudan and Wilhelm Ackermann, students of David Hilbert, were studying the foundations of computation. Sudan is credited with inventing the lesser-known Sudan function, the first published function that is recursive but not primitive recursive. Shortly afterwards and independently, in 1928, Ackermann published his own recursive but not primitive recursive function.[2]

Ackermann originally considered a function $A(m, n, p)$ of three variables, the p-fold iterated exponentiation of m with n, or $m \to n \to p$ as expressed using the Conway chained arrow notation. When $p = 1$, this is m^n, which is m multiplied by itself n times. When $p = 2$, it is a tower of exponents $m^{m^{\cdot^{\cdot^{m}}}}$ with n levels, or m raised n times to the power m also written as ${}^{n}m$, the tetration of m with n. This can be generalized indefinitely as p becomes larger.

Ackermann proved that A is a recursive function, a function that a computer with unbounded memory can calculate, but it is not a primitive recursive function, a class of functions including almost all familiar functions such as addition and factorial.

In *On the Infinite*, David Hilbert hypothesized that the Ackermann function was not primitively recursive, but it was Ackermann, Hilbert's personal secretary and former student, who actually proved the hypothesis in his paper *On Hilbert's Construction of the Real Numbers*. *On the Infinite* was Hilbert's most important paper on the foundations of mathematics, serving as the heart of Hilbert's program to secure the foundation of transfinite numbers by basing them on finite methods.[3] [4]

A similar function of only two variables was later defined by Rózsa Péter and Raphael Robinson; its definition is given below. The numbers, except in the first few rows, are three less than powers of two. For the exact relation between the two functions, see below.[5]

Definition and properties

The Ackermann function is defined recursively for non-negative integers m and n as follows (this presentation is due to Rózsa Péter):

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

It may not be immediately obvious that the evaluation of these functions always terminates. The recursion is bounded because in each recursive application either *m* decreases, or *m* remains the same and *n* decreases. Each time that *n* reaches zero, *m* decreases, so *m* eventually reaches zero as well. (Expressed more technically, in each case the pair (*m*, *n*) decreases in the lexicographic order, which preserves the well-ordering of the non-negative integers.) However, when *m* decreases there is no upper bound on how much *n* can increase — and it will often increase greatly.

The Ackermann function can also be expressed nonrecursively using:

- Conway chained arrow notation:

  $A(m, n) = (2 \rightarrow (n+3) \rightarrow (m-2)) - 3$ for $m > 2$
  hence
  $2 \rightarrow n \rightarrow m = A(m+2, n-3) + 3$ for $n>2$
  (n=1 and n=2 would correspond with $A(m,-2) = -1$ and $A(m,-1) = 1$, which could logically be added.)

- hyper operators:

  $A(m, n) = \text{hyper}(2, m, n + 3) - 3$

- the indexed version of Knuth's up-arrow notation:

  $A(m, n) = 2 \uparrow^{m-2} (n + 3) - 3$
  The part of the definition $A(m, 0) = A(m-1, 1)$ corresponds to $2 \uparrow^{m+1} 3 = 2 \uparrow^{m} 4$.

For small values of *m* like 1, 2, or 3, the Ackermann function grows relatively slowly with respect to *n* (at most exponentially). For $m \geq 4$, however, it grows much more quickly; even $A(4, 2)$ is about 2×10^{19728}, and the decimal expansion of $A(4, 3)$ is very large by any typical measure.

If we define the function $f(n) = A(n, n)$, which increases both *m* and *n* at the same time, we have a function of one variable that dwarfs every primitive recursive function, including very fast-growing functions such as the exponential function, the factorial function, multi- and superfactorial functions, and even functions defined using Knuth's up-arrow notation (except when the indexed up-arrow is used).

This extreme growth can be exploited to show that *f*, which is obviously computable on a machine with infinite memory such as a Turing machine and so is a computable function, grows faster than any primitive recursive function and is therefore not primitive recursive. Though the Ackermann function is often used to debunk the hypothesis that all useful or simple functions are primitive recursive, one should not confuse the *primitive recursive* functions with those definable by *primitive recursion* (it is this latter class that is of interest to programming language theorists because programs written using only primitive recursion are guaranteed to terminate). In a category with exponentials, using the isomorphism $A \times B \rightarrow C \cong A \rightarrow (B \rightarrow C)$, the Ackermann function may be defined via primitive recursion over higher-order functionals as follows:

$$\begin{aligned} Ack(0) &= Succ \\ Ack(m+1) &= Iter(Ack(m)) \end{aligned}$$

where *Succ* is the usual successor function and *Iter* is defined by primitive recursion as well:

$$\begin{aligned} Iter(f)(0) &= f(1) \\ Iter(f)(n+1) &= f(Iter(f)(n)). \end{aligned}$$

One interesting aspect of the Ackermann function is that the only arithmetic operations it ever uses are addition and subtraction of 1. Its properties come solely from the power of unlimited recursion. This also implies that its running time is at least proportional to its output, and so is also extremely huge. In actuality, for most cases the running time is far larger than the output; see below.

Table of values

Computing the Ackermann function can be restated in terms of an infinite table. We place the natural numbers along the top row. To determine a number in the table, take the number immediately to the left, then look up the required number in the previous row, at the position given by the number just taken. If there is no number to its left, simply look at column 1 in the previous row. Here is a small upper-left portion of the table:

Values of *A*(*m*, *n*)

*m**n*	0	1	2	3	4	n
0	1	2	3	4	5	$n + 1$
1	2	3	4	5	6	$n + 2 = 2 + (n + 3) - 3$
2	3	5	7	9	11	$2n + 3 = 2 * (n + 3) - 3$
3	5	13	29	61	125	$2^{(n+3)} - 3$
4	13	65533	$2^{65536} - 3$	$2^{2^{65536}} - 3$	$A(3, A(4, 3))$	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3 \text{ twos}} - 3$
5	65533	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{65536 \text{ twos}} - 3$	$A(4, A(5, 1))$	$A(4, A(5, 2))$	$A(4, A(5, 3))$	$A(4, A(5, n-1))$
6	$A(5, 1)$	$A(5, A(6, 0))$	$A(5, A(6, 1))$	$A(5, A(6, 2))$	$A(5, A(6, 3))$	$A(5, A(6, n-1))$

The numbers listed here in a recursive reference are very large and cannot be easily notated in some other form.

Despite the large values occurring in this early section of the table, some even larger numbers have been defined, such as Graham's number, which cannot be written with any small number of Knuth arrows. This number is constructed with a technique similar to applying the Ackermann function to itself recursively.

This is a repeat of the above table, but with the values replaced by the relevant expression from the function definition to show the pattern clearly:

Values of *A*(*m*, *n*)

*m**n*	0	1	2	3	4	n
0	0+1	1+1	2+1	3+1	4+1	$n + 1$
1	A(0,1)	A(0,A(1,0))	A(0,A(1,1))	A(0,A(1,2))	A(0,A(1,3))	$n + 2 = 2 + (n + 3) - 3$
2	A(1,1)	A(1,A(2,0))	A(1,A(2,1))	A(1,A(2,2))	A(1,A(2,3))	$2n + 3 = 2 * (n + 3) - 3$
3	A(2,1)	A(2,A(3,0))	A(2,A(3,1))	A(2,A(3,2))	A(2,A(3,3))	$2^{(n+3)} - 3$
4	A(3,1)	A(3,A(4,0))	A(3,A(4,1))	A(3,A(4,2))	A(3,A(4,3))	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3 \text{ twos}} - 3$
5	A(4,1)	A(4,A(5,0))	A(4,A(5,1))	A(4,A(5,2))	A(4,A(5,3))	$A(4, A(5, n-1))$
6	A(5,1)	A(5,A(6,0))	A(5,A(6,1))	A(5,A(6,2))	A(5,A(6,3))	$A(5, A(6, n-1))$

Extension

The first three of Ackermann functions can be expressed through elementary functions; they allow straightforward analytic extension for complex values of the second argument. In these sense, A (1,z), A(2,z), A(3,z) are analytic functions in the whole z-plane.

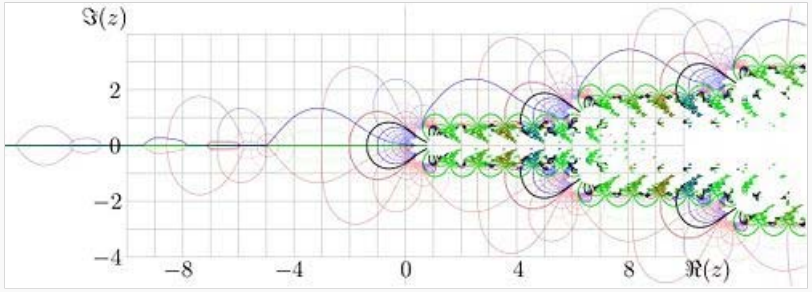

Fig.8. f=A(4,z) in the complex z-plane. Levels, corresponding to integer values of the Re(f) and those for Im(f) are drawn

No such extension is yet established for A (m,z) at integer m>3. The sketch in Fig.8 represents the possible realization of such extension, that remains limited at $\pm i\infty$. The drawing indicates that, perhaps, the extension of A(4,z), analytic in the whole z-plane, is not possible; the analytic extension should have singularity at z=-5 and cut at the real axis for z<-5. Such cut and singularities seem to be typical also for the tetration function.

Expansion

To see how the Ackermann function grows so quickly, it helps to expand out some simple expressions using the rules in the original definition. For example, we can fully evaluate $A(1,2)$ in the following way:

$$\begin{aligned}
A(1,2) &= A(0, A(1,1)) \\
&= A(0, A(0, A(1,0))) \\
&= A(0, A(0, A(0,1))) \\
&= A(0, A(0,2)) \\
&= A(0,3) \\
&= 4.
\end{aligned}$$

To demonstrate how $A(4,3)$'s computation results in many steps and in a large number:

$$\begin{aligned}
A(4,3) &= A(3, A(4,2)) \\
&= A(3, A(3, A(4,1))) \\
&= A(3, A(3, A(3, A(4,0)))) \\
&= A(3, A(3, A(3, A(3,1)))) \\
&= A(3, A(3, A(3, A(2, A(3,0))))) \\
&= A(3, A(3, A(3, A(2, A(2,1))))) \\
&= A(3, A(3, A(3, A(2, A(1, A(2,0)))))) \\
&= A(3, A(3, A(3, A(2, A(1, A(1,1)))))) \\
&= A(3, A(3, A(3, A(2, A(1, A(0, A(1,0))))))) \\
&= A(3, A(3, A(3, A(2, A(1, A(0, A(0,1))))))) \\
&= A(3, A(3, A(3, A(2, A(1, A(0,2)))))) \\
&= A(3, A(3, A(3, A(2, A(1,3))))) \\
&= A(3, A(3, A(3, A(2, A(0, A(1,2)))))) \\
&= A(3, A(3, A(3, A(2, A(0, A(0, A(1,1))))))) \\
&= A(3, A(3, A(3, A(2, A(0, A(0, A(0, A(1,0)))))))) \\
&= A(3, A(3, A(3, A(2, A(0, A(0, A(0, A(0,1)))))))) \\
&= A(3, A(3, A(3, A(2, A(0, A(0, A(0,2)))))) \\
&= A(3, A(3, A(3, A(2, A(0, A(0, A(0,3))))))) \\
&= A(3, A(3, A(3, A(2, A(0, A(0, A(0,4))))))) \\
&= A(3, A(3, A(3, A(2, A(0, A(0, A(0,5))))))) \\
&= \ldots \\
&= A(3, A(3, A(3,13))) \\
&= \ldots \\
&= A(3, A(3,65533)) \\
&= \ldots \\
&= A(3, 2^{65536} - 3) \\
&= \ldots \\
&= 2^{2^{65536}} - 3
\end{aligned}$$

Written as a power of 10, this is roughly equivalent to $10^{10^{19727.78}}$

Inverse

Since the function $f(n) = A(n, n)$ considered above grows very rapidly, its inverse function, f^{-1}, grows very slowly. This

inverse Ackermann function f^{-1} is usually denoted by α. In fact, α(n) is less than 5 for any conceivable input size *n*, since A (4, 4) is on the order of $2^{2^{10^{19729}}}$. "For all practical purposes", α(*n*) can be regarded as being a constant.

This inverse appears in the time complexity of some algorithms, such as the disjoint-set data structure and Chazelle's algorithm for minimum spanning trees. Sometimes Ackermann's original function or other variations are used in these settings, but they all grow at similarly high rates. In particular, some modified functions simplify the expression by eliminating the -3 and similar terms.

A two-parameter variation of the inverse Ackermann function can be defined as follows:

$$\alpha(m,n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) \geq \log_2 n\}.$$

This function arises in more precise analyses of the algorithms mentioned above, and gives a more refined time bound. In the disjoint-set data structure, *m* represents the number of operations while *n* represents the number of elements; in the minimum spanning tree algorithm, *m* represents the number of edges while *n* represents the number of vertices. Several slightly different definitions of α(*m*, *n*) exist; for example, $\log_2 n$ is sometimes replaced by *n*, and the floor function is sometimes replaced by a ceiling.

Other studies might define an inverse function of one where m is set to a constant, such that the inverse applies to a particular row.[6]

Use as benchmark

The Ackermann function, due to its definition in terms of extremely deep recursion, can be used as a benchmark of a compiler's ability to optimize recursion. The first use of Ackermann's function in this way was by Yngve Sundblad, *The Ackermann function. A Theoretical, computational and formula manipulative study.* (BIT 11 (1971), 107119).

This seminal paper was taken up by Brian Wichmann (co-author of the Whetstone benchmark) in a trilogy of papers written between 1975 and 1982.[7][8][9]

A more recent use of Ackermann's function as a compiler benchmark is in The Computer Language Shootout which compares the time required to evaluate this function for fixed arguments in many different programming language implementations.[10] [11]

For example, a compiler which, in analyzing the computation of *A*(3, 30), is able to save intermediate values like the *A*(3, *n*) and *A*(2, *n*) in that calculation rather than recomputing them, can speed up computation of *A*(3, 30) by a factor of hundreds of thousands. Also, if *A*(2, *n*) is computed directly rather than as a recursive expansion of the form *A*(1, *A*(1, *A*(1,...*A*(1, 0)...))), this will save significant amounts of time. Computing *A*(1, *n*) takes linear time in *n*. Computing *A*(2, *n*) requires quadratic time, since it expands to O(*n*) nested calls to *A*(1, *i*) for various *i*. Computing *A*(3, *n*) requires time proportionate to 4^{n+1}. The computation of *A*(3, 1) in the example above takes 16 (4^2) steps.

A(4, 2), which appears as a decimal expansion in several web pages, cannot possibly be computed by simple recursive application of the Ackermann function in any tractable amount of time. Instead, shortcut formulas such as $A(3, n) = 8\times2^n-3$ are used as an optimization to complete some of the recursive calls.

A practical method of computing functions similar to Ackermann's is to use memoization of intermediate results. A compiler could apply this technique to a function automatically using Donald Michie's "memo functions".[12]

Implementations

The first implementation in a programming language was written in the Fortran programming language in 1964, see H. Gordon Rice[13], *Recursion and iteration*, Commun. ACM, 8(2), 1965, pp. 114--115.)

In the C programming language, it can be implemented like this:

```
unsigned int ackermann(unsigned int m, unsigned int n) {
  if (m == 0)
    return n + 1;
  else if (n == 0) /* m will be > 0 here */
    return ackermann(m-1, 1);
  else /* both m and n will be > 0 */
    return ackermann(m-1, ackermann(m, n - 1));
}
```

In Haskell, it can be implemented like this:

```
ack 0 n = n + 1
ack m 0 = ack (m - 1) 1
ack m n = ack (m - 1) $ ack m (n - 1)
```

Ackermann numbers

Related to the Ackermann function but in fact different are the Ackermann numbers, a sequence where the n^{th} term equals:

$$n \underbrace{\uparrow \ldots \uparrow}_{n} n$$

in the Knuth's up-arrow notation, or

- $n \rightarrow n \rightarrow n$

in the Conway chained arrow notation.[14]

For instance, the first three Ackermann numbers are

- $1\uparrow 1$,
- $2\uparrow\uparrow 2$
- $3\uparrow\uparrow\uparrow 3$

which equal the following:

- $1^1 = 1$
- ${}^2 2 = 4$
- ${}^3 3 = 3 \uparrow\uparrow 3 \uparrow\uparrow 3 = \underbrace{3^{3^{3^{3^{3^{\cdot^{\cdot^{\cdot}}}}}}}}_{3^{27} \text{ or } 7625597484987 \text{ threes}}$

An attempt to express the fourth Ackermann number, $4\uparrow\uparrow\uparrow\uparrow 4$, using iterated exponentiation as above would become extremely complicated. However, it can be expressed using tetration in three nested layers as shown below. Explanation: in the middle layer, there is a tower of tetration whose full length is ${}^{4}{}^{4}4$ and the final result is the top layer of tetrated 4's whose full length equals the calculation of the middle layer. Note that by way of size comparison, the simple expression ${}^{4}4$ already exceeds a googolplex, so the fourth Ackermann number is quite large.

$$\underbrace{{}^{4^{\cdot^{\cdot^{\cdot^{4}}}}}4}_{{}^{4^{\cdot^{\cdot^{\cdot^{4}}}}}4}$$

In popular culture

Randall Munroe has mentioned the Ackermann function in his popular web-comic xkcd.[15] In the comic, Munroe makes reference to the Ackermann function with Graham's number as the arguments. At the time, he considered this to be the largest number ever concisely defined and named it "The xkcd number".[16]

A question involving this function was posed at the International Mathematical Olympiad, the most significant mathematics competition for school age students, in 1981.""International Mathematics Olympiad Problems, Year 1981"". Retrieved on 2008-10-06.

See also

- Computability theory (computer science)
- Charity (programming language)
- Recursion (computer science)

Notes and references

1. ^ Decimal expansion of A(4,2) contains 19729 decimal digits
2. ^ Cristian Calude, Solomon Marcus and Ionel Tevy (November 1979). "The first example of a recursive function which is not primitive recursive". *Historia Math.* **6** (4): 380–84. doi:10.1016/0315-0860(79)90024-7. Summarized in Bill Dubuque (1997-09-12). "*Ackermann vs. Sudan*". sci.logic. (Web link). Retrieved on 2006-06-13.
3. ^ Wilhelm Ackermann (1928). "*Zum Hilbertschen Aufbau der reellen Zahlen*". *Mathematische Annalen* **99**: 118–133. doi:10.1007/BF01459088.
4. ^ von Heijenoort. From Frege To Gödel, 1967.
5. ^ Raphael M. Robinson (1948). "Recursion and Double Recursion". *Bulletin of the American Mathematical Society* **54**: 987–93. doi:10.1090/S0002-9904-1948-09121-2.
6. ^ An inverse-Ackermann style lower bound for the online minimum spanning tree verification problem November 2002
7. ^ "Ackermann's Function: A Study In The Efficiency Of Calling Procedures" (1975).
8. ^ "How to Call Procedures, or Second Thoughts on Ackermann's Function" (1977).
9. ^ "Latest results from the procedure calling test, Ackermann's function" (1982).
10. ^ "Gentoo: Intel Pentium 4 Computer Language Shootout" (2006). Retrieved on 2006-06-13.
11. ^ Benchmarks XGC, May 11, 2005
12. ^ Example: Explicit memo function version of Ackermann's function implemented in PL/SQL
13. ^ The author of Rice's theorem !
14. ^ Ackermann Number
15. ^ ""What xkcd Means"". Retrieved on 2007-06-25.
16. ^ ""The Clarkkkkson vs. the xkcd Number"". Retrieved on 2007-06-25.

External links

- Eric W. Weisstein, *Ackermann function* at MathWorld.
- Paul E. Black, Ackermann's function at the NIST Dictionary of Algorithms and Data Structures.
- Scott Aaronson, *Who can name the biggest number?* (1999)
- Ackermann function's. Includes a table of some values.
- Hyper-operations: Ackermann's Function and New Arithmetical Operation
- Robert Munafo's Large Numbers describes several variations on the definition of A.
- Gabriel Nivasch, Inverse Ackermann without pain on the inverse Ackermann function.
- Raimund Seidel, *Understanding the inverse Ackermann function* (PDF presentation).

Retrieved from "http://en.wikipedia.org/wiki/Ackermann_function"
Categories: Arithmetic | Large numbers | Special functions | Theory of computation | Recursion theory
Hidden categories: All articles with unsourced statements | Articles with unsourced statements since February 2007 | Wikipedia articles incorporating text from public domain works of the United States Government