Exploring the Ackermann Function

CSCI 4365

Hugh Coleman

Definition

This is the common two argument Ackermann–Péter function

It is defined for non-negative integers m and n

Ackermann Function Definition

$$
\begin{aligned}
A(0,n) &= n + 1 \\
A(m{+}1,0) &= A(m,1) \\
A(m{+}1,n{+}1) &= A(m,A(m{+}1,n))
\end{aligned}
$$

[J. Paul, 2023]

Fast Growing

Examples

Number of quarks in the universe is 3.28×10^{80}

$$\begin{aligned} A(1,1) &= 3 \\ A(2,2) &= 7 \\ A(3,3) &= 61 \\ A(4,4) &= 2^{2^{2^{65536}}} - 3 \end{aligned}$$

[Bennett, 2017]

Notes

Halting

The evaluation of $A(m, n)$ always terminates.

$$
\begin{aligned}
A(0,n) &= n + 1 \\
A(m{+}1,0) &= A(m,1) \\
A(m{+}1,n{+}1) &= A(m,A(m{+}1,n))
\end{aligned}
$$

Notes

Halting

The evaluation of $A(m,\ n)$ always terminates.

PR

A(m,n) is not primitive recursive

$$
\begin{aligned}
A(0,n) &= n+1 \\
A(m+1,0) &= A(m,1) \\
A(m+1,n+1) &= A(m,A(m+1,n))
\end{aligned}
$$

Notes

Halting

The evaluation of $A(m,\ n)$ always terminates.

PR

A(m,n) is not primitive recursive

Majorization

Given $f, g : \mathbb{N} \to \mathbb{N}$, then f majorize's g if for all x, $f(x) \geq g(x)$

$$
\begin{array}{lcl}
A(0,n) & = & n + 1 \\
A(m{+}1,0) & = & A(m,1) \\
A(m{+}1,n{+}1) & = & A(m,A(m{+}1,n))
\end{array}
$$

Termination

The Ackermann function, denoted as A(m, n), always terminates.

- Decreasing m: In each recursive application, either m decreases or remains the same while n decreases.
- n reaching zero: When n reaches zero, m decreases in the next recursive call.
- m reaching zero: Since m is always decreasing or remaining the same, eventually m will hit zero.

[Paulson, 2022]

Iterative Calculation

The Ackermann function can be computed using a stack, which initially contains elements, $[m, n]$

[Grossman and Zeitman, 1988, J. Paul, 2023]

Iterative Calculation

The Ackermann function can be computed using a stack, which initially contains elements, $[m, n]$

0	, n	$\rightarrow$	(n+1)		
(m+1)	, 0	$\rightarrow$	m,	1	
(m+1)	, (n+1)	$\rightarrow$	m,	(m+1),	n

[Grossman and Zeitman, 1988, J. Paul, 2023]

Iterative Calculation

The Ackermann function can be computed using a stack, which initially contains elements, $[m, n]$

0	,	n	$\rightarrow$	(n+1)		
(m+1)	,	0	$\rightarrow$	m,	1	
(m+1)	,	(n+1)	$\rightarrow$	m,	(m+1),	n

Ackermann Function Definition

$$
\begin{aligned}
A(0,n) &= n + 1 \\
A(m+1,0) &= A(m,1) \\
A(m+1,n+1) &= A(m,A(m+1,n))
\end{aligned}
$$

[Grossman and Zeitman, 1988, J. Paul, 2023]

Iterative Computation

$A(2,1) = 5$

```
[2] 1
[1, 2] 0
[1, 1] 1
[1, 0, 1] 0
[1, 0, 0] 1
[1, 0] 2
[1] 3
[0, 1] 2
[0, 0, 1] 1
[0, 0, 0, 1] 0
[0, 0, 0, 0] 1
[0, 0, 0] 2
[0, 0] 3
[0] 4
[] 5
```

$A(1,3) = 5$

```
[1] 3
[0, 1] 2
[0, 0, 1] 1
[0, 0, 0, 1] 0
[0, 0, 0, 0] 1
[0, 0, 0] 2
[0, 0] 3
[0] 4
[] 5
```

$A(2,1) = 5$

```
[2] 1
[1, 2] 0
[1, 1] 1
[1, 0, 1] 0
[1, 0, 0] 1
[1, 0] 2
[1] 3
[0, 1] 2
[0, 0, 1] 1
[0, 0, 0, 1] 0
[0, 0, 0, 0] 1
[0, 0, 0] 2
[0, 0] 3
[0] 4
[] 5
```

```python
def ackermann(m, n):
    stack = []
    stack.append(m)
    while stack:
        m = stack.pop()
        if m == 0:
            n += 1
        elif n == 0:
            n = 1
            stack.append(m-1)
        else:
            n = n - 1
            stack.append(m-1)
            stack.append(m)
    return n
```

Computational Length

Ackermann

```
def ackermann(m, n):
    stack = []
    stack.append(m)
    while stack:
        m = stack.pop()
        if m == 0:
            n += 1
        elif n == 0:
            n = 1
            stack.append(m-1)
        else:
            n = n - 1
            stack.append(m-1)
            stack.append(m)
    return n
```

Ackermann PR

```
def ackermannPR(m, n):
    stack = []
    stack.append(m)
    x = comp(m,n)
    do x times:
        m = stack.pop()
        if m == 0:
            n += 1
        elif n == 0:
            n = 1
            stack.append(m-1)
        else:
            n = n - 1
            stack.append(m-1)
            stack.append(m)
    return n
```

Ackermann Compute Time

No primitive recursive function exists that can compute the number of times $A(m, n)$ will loop.

Computation

```
def comp(m, n):
    ...
```

Stack Model Complexity

- For all m, n the computation of $A(m, n)$ takes no more than $(A(m, n) + 1)^m$ steps
- the maximum length of the stack is $A(m, n)$, as long as $m > 0$

$A(1, 3) = 5$

```
[1] 3
[0, 1] 2
[0, 0, 1] 1
[0, 0, 0, 1] 0
[0, 0, 0, 0] 1
[0, 0, 0] 2
[0, 0] 3
[0] 4
[] 5
```

[Cohen, 1987]

Grossman & Zeitman Algorithm

Their algorithm computes $A(m, n)$ within $\mathcal{O}(m\,\mathrm{A}(m, n))$ time and within $\mathcal{O}(m)$ space

$A(2, 1) = 5$

```
[2] 1
[1, 2] 0
[1, 1] 1
[1, 0, 1] 0
[1, 0, 0] 1
[1, 0] 2
[1] 3
[0, 1] 2
[0, 0, 1] 1
[0, 0, 0, 1] 0
[0, 0, 0, 0] 1
[0, 0, 0] 2
[0, 0] 3
[0] 4
[] 5
```

$A(1, 3) = 5$

```
[1] 3
[0, 1] 2
[0, 0, 1] 1
[0, 0, 0, 1] 0
[0, 0, 0, 0] 1
[0, 0, 0] 2
[0, 0] 3
[0] 4
[] 5
```

[Grossman and Zeitman, 1988]

Inverse Ackermann

Single Argument Version

$f(n) = A(n, n)$

[Armando B, 2014]

Inverse Ackermann

Single Argument Version

$f(n) = A(n, n)$

Inverse

The inverse f^{-1} is commonly known as α(n)

[Armando B, 2014]

Inverse Ackermann

Single Argument Version

$f(n) = A(n, n)$

Inverse

The inverse f^{-1} is commonly known as $\alpha(\mathrm{n})$

Primitive Recursive

The inverse of the Ackermann function is primitive recursive

[Armando B, 2014]

Inverse Ackermann - Examples

Examples

$$
\begin{aligned}
\alpha(3) &= 1 \\
\alpha(7) &= 2 \\
\alpha(61) &= 3 \\
\alpha(2^{2^{2^{65536}}} - 3) &= 4
\end{aligned}
$$

[Armando B, 2014]

Inverse Ackermann - Examples

Examples

$$
\begin{aligned}
\alpha(3) &= 1 \\
\alpha(7) &= 2 \\
\alpha(61) &= 3 \\
\alpha(2^{2^{2^{65536}}} - 3) &= 4
\end{aligned}
$$

Domain

As the Ackermann function is not onto, its inverse is not total.

[Armando B, 2014]

Inverse Ackermann - Examples

Examples

$$\begin{aligned} \alpha(3) &= 1 \\ \alpha(7) &= 2 \\ \alpha(61) &= 3 \\ \alpha(2^{2^{2^{65536}}} - 3) &= 4 \end{aligned}$$

Domain

As the Ackermann function is not onto, its inverse is not total.

Domain

The issue of the function being partial is to floor the input.

[Armando B, 2014]

Inverse Ackermann Graph

Lemma 1

For all $m, n \in \mathbb{N}$

- $A(m, n) > m + n$
- $A(m, n) < A(m, n + 1)$
- $A(m, n) < A(m + 1, n)$

[Armando B, 2014]

Graph Primitive Recursive

Note

If $A(x, y) = z$, we must have $x \leq z$ and $y \leq z$.

Given x, y, and z to test if $A(x, y) = z$, we can ignore the arguments m and n outside the rectangle $0 \leq m \leq z$, $0 \leq n \leq z$, as well as the values $A(m, n)$ greater than z because, using Lemma 1.

1. If $m > z$, $A(m, n) > z$ for every n.
2. If $n > z$, $A(m, n) > z$ for every m.
3. If $A(m, n)$ is used as an argument of another computation of A the "final result" will be greater than z.

[Armando B, 2014]

Ackermann Computation

The computation of A(m,n) is either

- Immediate, when $m = 0 : A(0, n) = n + 1$
- Dependent on $A(m-1, 1)$ when $n = 0 : A(m, 0) = A(m-1, 1)$
- Dependent on $A(m-1, w)$ with $w = A(m, n-1)$, when $m, n \geq 1$

[Armando B, 2014]

Function Outline

The following algorithm computes the Ackermann graph.

Function

1. Input : x, y, z.
2. Output : 1 if $A(x, y) = z$, 0 if $A(x, y) \neq z$.
3. "$\star$" denotes a value greater than z

[Armando B, 2014]

Ackermann Graph Computation

1. Compute A(m,n) inside the rectangle $0 \leq m, n \leq z$
   1. Compute and save $A(0,0), A(0,1), ..., A(0,z)$
   2. Compute and save $A(1,0), ..., A(z,0)$
   3. For $m = 1, 2, ..., z$: Compute and save $A(m,1), A(m,2), ..., A(m,z)$

   In computation above, if $A(m,n) > z$, mark the value of $A(m,n)$ as $\star$
2. Find if $A(x,y) = z$
   1. If $x > z$ or $y > z$ output 0 ((x,y,z) not in the graph)
   2. Otherwise search for a stored triple of the form (x,y,w) (in particular we can have $w = \star$).
      1. If $w = z$ output 1 ((x,y,z) in the graph)
      2. If $w \neq z$ output 0 ((x,y,z) not in the graph; this includes of course the case $w = \star$).

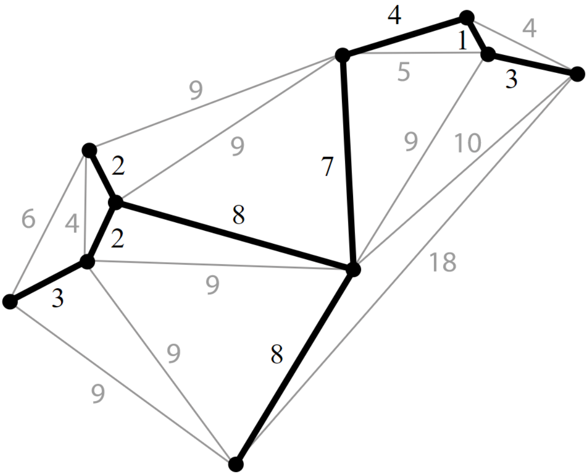
[Armando B, 2014]

Do-Times Program

```
GRAPH = compute (x,y,z) in the rectangle 0 ≤ x,y ≤ z

X:=0
Y:=0

V = Z+1

DO V TIMES[
    DO V TIMES[
        IF((X,Y,Z) is a computed triple)[
            RETURN [ (X,Y) ]
        ]
        INCR(Y)
    ]
    INCR(X)
    Y:= 0
]

RETURN[ NO ]
```

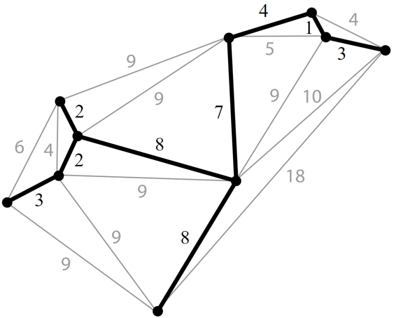
[Armando B, 2014]

For our case m is the number of edges in the graph and n is the number of vertices.



[Commons, 2023]

Chazelle's algorithm has a complexity of $\mathcal{O}(m\alpha(m, n))$



[Chazelle, 2000, Commons, 2023]

Armando B, M. (2014).
The inverse of the ackermann function is primitive recursive.

Bennett, J. (2017).
How many particles are in the observable universe?

Chazelle, B. (2000).
A minimum spanning tree algorithm with inverse-ackermann type complexity.
Journal of the ACM (JACM), 47(6):1028–1047.

Cohen, D. E. (1987).
Computability and Logic.
Mathematics and its Applications. Ellis Horwood Ltd, Publisher, Harlow, England.

Commons, W. (2023).
File:minimum spanning tree.svg — wikimedia commons, the free media repository.
[Online; accessed 10-April-2023].

Grossman, J. W. and Zeitman, R. (1988).
An inherently iterative computation of ackermann's function.
Theoretical Computer Science, 57(2):327–330.

J. Paul, M. (2023).
Csci 4365 lecture notes.

Paulson, L. C. (2022).
Machine logic.