Package 'RcppBigIntAlgos'

October 14, 2020

Type Package

Title Factor Big Integers with the Parallel Quadratic Sieve

Version 0.3.3

Maintainer Joseph Wood <jwood000@gmail.com>

Description Features the multiple polynomial quadratic sieve (MPQS) algorithm for factoring large integers and a vectorized factoring function that returns the complete factorization of an integer. The MPQS is based off of the seminal work of Carl Pomerance (1984) <doi:10.1007/3-540-39757-4_17> along with the modification of multiple polynomials introduced by Peter Montgomery and J. Davis as outlined by Robert D. Silverman (1987) <doi:10.1090/S0025-5718-1987-0866119-8>. Utilizes the C library GMP (GNU Multiple Precision Arithmetic) and 'RcppThread' for sieving multiple polynomials in parallel. The Pollard's rho algorithm for factoring smaller numbers is the same algorithm used by the factorize function in the 'gmp' package.

License GPL (>= 2)

Encoding UTF-8

SystemRequirements C++11, gmp (>= 4.2.3)

Depends gmp

Imports Rcpp

LinkingTo Rcpp, RcppThread

Suggests testthat, numbers, RcppAlgos

NeedsCompilation yes

URL https://github.com/jwood000/RcppBigIntAlgos, https://gmplib.org/, http://mathworld.wolfram.com/QuadraticSieve.html, http://micsymposium.org/mics_2011_proceedings/mics2011_submission_28.pdf, https://www.math.colostate.edu/~hulpke/lectures/m400c/quadsievex.pdf, https://blogs.msdn.microsoft.com/devdev/2006/06/19/factoring-large-numbers-with-quadratic-sieve/

BugReports https://github.com/jwood000/RcppBigIntAlgos/issues

RoxygenNote 7.1.0

Author Joseph Wood [aut, cre],
Free Software Foundation, Inc. [cph],
Mike Tryczak [ctb]

Repository CRAN

Date/Publication 2020-10-14 00:00:02 UTC

R topics documented:

`divisorsBig` *Vectorized Factorization (Complete) with GMP*

Description

Quickly generates the complete factorization for many (possibly large) numbers.

Usage

```
divisorsBig(v, namedList = FALSE)
```

Arguments

`v`	Vector of integers, numerics, string values, or elements of class bigz.
`namedList`	Logical flag. If `TRUE` and the `length(v) > 1`, a named list is returned. The default is `FALSE`.

Details

Highly optimized algorithm to generate the complete factorization for many numbers. It is built specifically for the data type that is used in the gmp library (i.e. `mpz_t`).

The main part of this algorithm is essentially the same algorithm that is implemented in divisorsRcpp from the RcppAlgos package. A modified merge sort algorithm is implemented to better deal with the `mpz_t` data type. This algorithm avoids directly swapping elements of the main factor array of type `mpz_t` but instead generates a vector of indexing integers for ordering.

See this stack overflow post for examples and benchmarks : R Function for returning ALL factors.

Value

- Returns an unnamed vector of class bigz if `length(v) == 1` regardless of the value of `namedList`.
- If `length(v) > 1`, a named/unnamed list of vectors of class bigz will be returned.

Author(s)

Joseph Wood

References

Divisor

See Also

divisorsRcpp, divisors, factorize

Examples

```
## Get the complete factorization of a single number
divisorsBig(100)

## Or get the complete factorization of many numbers
set.seed(29)
myVec <- sample(-1000000:1000000, 1000)
system.time(myFacs <- divisorsBig(myVec))

## Return named list
myFacsWithNames <- divisorsBig(myVec, namedList = TRUE)
```

---

quadraticSieve *Prime Factorization with the Quadratic Sieve*

Description

Get the prime factorization of a number, n, using the Quadratic Sieve.

Usage

```
quadraticSieve(n, showStats = FALSE, nThreads = NULL, skipExtPolRho = FALSE)
```

Arguments

n	An integer, numeric, string value, or an element of class bigz.
showStats	Logical flag. If TRUE, summary statistics will be displayed.
nThreads	Number of threads to be used. The default is NULL.
skipExtPolRho	Logical flag for skipping extended pollard rho algorithm. The default is FALSE.

Details

First, trial division is carried out to remove small prime numbers, then a modified version of Pollard's rho algorithm that is constrained is called to quickly remove further prime numbers. Next, we check to make sure that we are not passing a perfect power to the main quadratic sieve algorithm. After removing any perfect powers, we finally call the quadratic sieve with multiple polynomials in a recursive fashion until we have completely factored our number.

When `showStats = TRUE`, summary statistics will be shown. The frequency of updates is dynamic as writing to stdout can be expensive. It is determined by how fast smooth numbers (including partially smooth numbers) are found along with the total number of smooth numbers required in order to find a non-trivial factorization. The statistics are:

`MPQS Time` The time measured for the multiple polynomial quadratic sieve section in hours `h`, minutes `m`, seconds `s`, and milliseconds `ms`.

`Complete` The percent of smooth numbers plus partially smooth numbers required to guarantee a non-trivial solution when Gaussian Elimination is performed on the matrix of powers of primes.

`Polynomials` The number of polynomials sieved

`Smooths` The number of Smooth numbers found

`Partials` The number of partially smooth numbers found. These numbers have one large factor, F, that is not reduced by the prime factor base determined in the algorithm. When we encounter another number that is almost smooth with the same large factor, F, we can combine them into one partially smooth number.

Value

Vector of class bigz

Note

Safely interrupt long executing commands by pressing `Ctrl + c` or whatever interruption command offered by the user's GUI.

Author(s)

Joseph Wood

References

- Pomerance, C. (2008). Smooth numbers and the quadratic sieve. In Algorithmic Number Theory Lattices, Number Fields, Curves and Cryptography (pp. 69-81). Cambridge: Cambridge University Press.
- Silverman, R. D. (1987). The Multiple Polynomial Quadratic Sieve. Mathematics of Computation, 48(177), 329-339. doi:10.2307/2007894
- Integer Factorization using the Quadratic Sieve
- From https://codegolf.stackexchange.com/ (Credit to user primo for answer) P., & Chowdhury, S. (2012, October 7). Fastest semiprime factorization. Retrieved October 06, 2017

See Also

factorize

Examples

```
mySemiPrime <- prod(nextprime(urand.bigz(2, 40, 17)))
quadraticSieve(mySemiPrime)
```

---

stdThreadMax *Max Number of Concurrent Threads*

Description

Rcpp wrapper of std::thread::hardware_concurrency(). As stated by cppreference, the returned value should be considered only a hint.

Usage

```
stdThreadMax()
```

Value

An integer representing the number of concurrent threads supported by the user implementation. If the value cannot be determined, 1L is returned.

See Also

detectCores

Examples

```
stdThreadMax()
```

Index