projet_ranking_groupe_3

1.0

Generated by Doxygen 1.8.13

Contents

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Class Documentation

3.1 s_edge Struct Reference

An edge of the spare matrix.

```
#include <matrix.h>
```

Public Attributes

- u32 y

  Destination vertex.
- f64 w

  Weight.

3.1.1 Detailed Description

An edge of the spare matrix.

3.1.2 Member Data Documentation

3.1.2.1 w

```
f64 s_edge::w
```

Weight.

3.1.2.2 y

```
u32 s_edge::y
```

Destination vertex.

The documentation for this struct was generated from the following file:

- src/matrix.h

3.2 s_frange Struct Reference

A range datatype to iterate over a range of floating point numbers.

```
#include <frange.h>
```

Public Attributes

- f64 begin

    The first value in the range.
- f64 end

    The last value in the range (inclusive).
- f64 step

    The step size.
- u32 count

    The number of values in the range.

3.2.1 Detailed Description

A range datatype to iterate over a range of floating point numbers.

3.2.2 Member Data Documentation

3.2.2.1 begin

```
f64 s_frange::begin
```

The first value in the range.

3.2.2.2 count

```
u32 s_frange::count
```

The number of values in the range.

3.2.2.3 end

```
f64 s_frange::end
```

The last value in the range (inclusive).

3.2.2.4 step

```
f64 s_frange::step
```

The step size.

The documentation for this struct was generated from the following file:

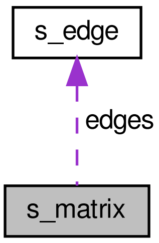
- src/frange.h

3.3 s_matrix Struct Reference

A spare matrix representation.

```
#include <matrix.h>
```

Collaboration diagram for s_matrix:



Public Attributes

- u32 vertices_count

    Number of vertices.
- u32 edges_count

    Number of edges.
- edge ∗ edges

    Edges stored in a contiguous array.
- u32 ∗ row_start

    Row start indices for each vertex.

3.3.1 Detailed Description

A spare matrix representation.

3.3.2 Member Data Documentation

3.3.2.1 edges

```
edge* s_matrix::edges
```

Edges stored in a contiguous array.

3.3.2.2 edges_count

```
u32 s_matrix::edges_count
```

Number of edges.

3.3.2.3 row_start

```
u32* s_matrix::row_start
```

Row start indices for each vertex.

3.3.2.4 vertices_count

```
u32 s_matrix::vertices_count
```

Number of vertices.

The documentation for this struct was generated from the following file:

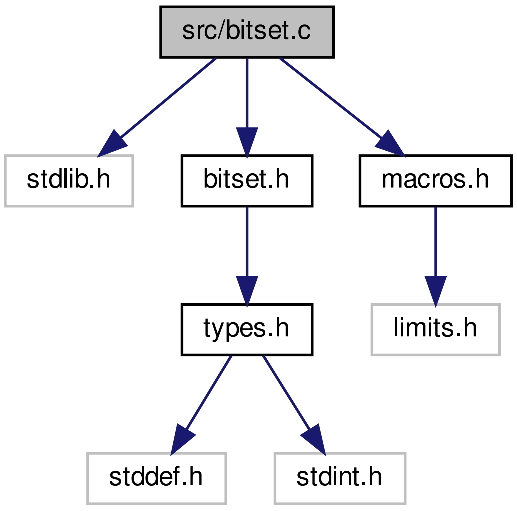
- src/matrix.h

4 File Documentation

4.1 src/bitset.c File Reference

```
#include <stdlib.h>
#include "bitset.h"
#include "macros.h"
```

Include dependency graph for bitset.c:



Functions

- static usize bitset_size (usize n)

  Returns the size of a bitset in bytes to store n bits.
- bitset ∗ bitset_alloc (usize size)

  Allocates memory for a bitset.
- usize bitset_is_set (const bitset ∗bs, usize i)

  Checks if a bit is set in a bitset.
- void bitset_set (bitset ∗bs, usize i)

  Sets a bit in a bitset.
- void bitset_reset (bitset ∗bs, usize n)

  Resets all bits of a bitset of n bits.
- void bitset_unset (bitset ∗bs, usize i)

  Unsets a bit in a bitset.

4.1.1 Function Documentation

4.1.1.1 bitset_alloc()

```
bitset* bitset_alloc (
            usize size )
```

Allocates memory for a bitset.

The bitset is not initialized.

Parameters

size	The number of bits in the bitset.

Returns

The allocated bitset or NULL if the allocation failed.

4.1.1.2 bitset_is_set()

```
usize bitset_is_set (
            const bitset * bs,
            usize i )
```

Checks if a bit is set in a bitset.

Parameters

bs	The bitset.
i	The index of the bit to check.

Returns

A non zero value if the bit at index i is set, 0 otherwise.

4.1.1.3 bitset_reset()

```
void bitset_reset (
            bitset * bs,
            usize n )
```

Resets all bits of a bitset of n bits.

Parameters

bs	The bitset.
n	The number of bits.

4.1.1.4 bitset_set()

```
void bitset_set (
             bitset * bs,
             usize i )
```

Sets a bit in a bitset.

Parameters

bs	The bitset.
i	The index of the bit to set.

4.1.1.5 bitset_size()

```
static usize bitset_size (
             usize n )  [static]
```

Returns the size of a bitset in bytes to store n bits.

The returned value can be used to allocate a bitset.

Parameters

n	The number of bits in the bitset.

Returns

The size needed to store n bits.

4.1.1.6 bitset_unset()

```
void bitset_unset (
             bitset * bs,
             usize i )
```

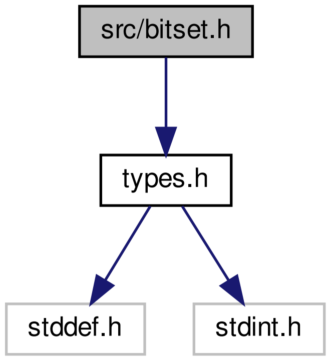
Unsets a bit in a bitset.

Parameters

bs	The bitset.
i	The index of the bit to unset.

4.2 src/bitset.h File Reference

`#include "types.h"`
Include dependency graph for bitset.h:



Typedefs

- typedef usize bitset

  A memory-efficient representation of an array of bits.

Functions

- bitset ∗ bitset_alloc (usize size)

  Allocates memory for a bitset.
- usize bitset_is_set (const bitset ∗bs, usize i)

  Checks if a bit is set in a bitset.
- void bitset_set (bitset ∗bs, usize i)

  Sets a bit in a bitset.
- void bitset_reset (bitset ∗bs, usize n)

  Resets all bits of a bitset of n bits.
- void bitset_unset (bitset ∗bs, usize i)

  Unsets a bit in a bitset.

4.2.1 Typedef Documentation

4.2.1.1 bitset

```
typedef usize bitset
```

A memory-efficient representation of an array of bits.

4.2.2 Function Documentation

4.2.2.1 bitset_alloc()

```
bitset* bitset_alloc (
             usize size )
```

Allocates memory for a bitset.

The bitset is not initialized.

Parameters

size	The number of bits in the bitset.

Returns

The allocated bitset or NULL if the allocation failed.

4.2.2.2 bitset_is_set()

```
usize bitset_is_set (
             const bitset * bs,
             usize i )
```

Checks if a bit is set in a bitset.

Parameters

bs	The bitset.
i	The index of the bit to check.

Returns

A non zero value if the bit at index i is set, 0 otherwise.

4.2.2.3 bitset_reset()

```
void bitset_reset (
             bitset * bs,
             usize n )
```

Resets all bits of a bitset of n bits.

Parameters

bs	The bitset.
n	The number of bits.

4.2.2.4 bitset_set()

```
void bitset_set (
            bitset * bs,
            usize i )
```

Sets a bit in a bitset.

Parameters

bs	The bitset.
i	The index of the bit to set.

4.2.2.5 bitset_unset()

```
void bitset_unset (
            bitset * bs,
            usize i )
```

Unsets a bit in a bitset.

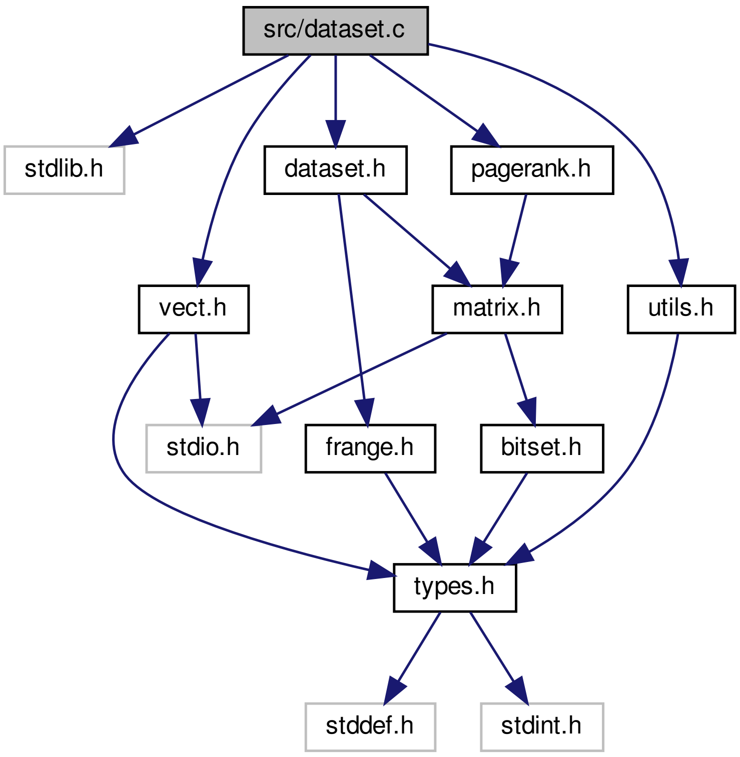
Parameters

bs	The bitset.
i	The index of the bit to unset.

4.3 src/dataset.c File Reference

```
#include <stdlib.h>
#include "dataset.h"
#include "pagerank.h"
#include "utils.h"
#include "vect.h"
```

Include dependency graph for dataset.c:



Macros

- #define ERASE_LINE "\033[1K\r"

Functions

- static void print_progression (frange ∗percent_indicator)

  Prints current progress.
- static f64 ∗ init_custom_pi (const f64 ∗pg_pi)

  Initilizes pi for custom PageRnak.
- void dataset_clear ()

  Clears internal dataset cache.
- int dataset_init (const matrix ∗m, const frange ∗alpha)

  Initialize internal dataset cache.
- void generate_dataset (FILE ∗output_file, u32 n, const frange ∗r)

  Generates the dataset.

Variables

- static const matrix ∗ g_original_graph = NULL
- static f64 ∗ g_pi_cache = NULL
- static u32 ∗ g_iter_cache = NULL
- static f64 ∗ g_pi = NULL
- static matrix ∗ g_subgraph = NULL
- static bitset ∗ g_removed_set = NULL
- static const frange ∗ g_alpha = NULL

4.3.1 Macro Definition Documentation

4.3.1.1 ERASE_LINE

```
#define ERASE_LINE "\033[1K\r"
```

4.3.2 Function Documentation

4.3.2.1 dataset_clear()

```
void dataset_clear ( )
```

Clears internal dataset cache.

4.3.2.2 dataset_init()

```
int dataset_init (
             const matrix ∗ original_graph,
             const frange ∗ alpha )
```

Initialize internal dataset cache.

Parameters

original_graph	The matrix representation of the original graph.
alpha	The range of alpha values.

Returns

0 on success, -1 on error.

4.3.2.3 generate_dataset()

```
void generate_dataset (
            FILE * output_file,
            u32 n,
            const frange * r )
```

Generates the dataset.

Parameters

output_file	The file where the results will be stored.
n	The number of subgraphs to generate.
r	r The range of ratio of removed vertices to test.

4.3.2.4 init_custom_pi()

```
static f64* init_custom_pi (
            const f64 * pg_pi )  [static]
```

Initilizes pi for custom PageRnak.

Parameters

pg↩ _pi	The original PageRank vector.

Returns

The initialized vector.

4.3.2.5 print_progression()

```
static void print_progression (
            frange * percent_indicator )  [static]
```

Prints current progress.

Increments the progress counter.

Parameters

percent_indicator	The percentage indicator as a range.

4.3.3 Variable Documentation

4.3.3.1 g_alpha

```
const frange* g_alpha = NULL  [static]
```

4.3.3.2 g_iter_cache

```
u32* g_iter_cache = NULL  [static]
```

4.3.3.3 g_original_graph

```
const matrix* g_original_graph = NULL  [static]
```

4.3.3.4 g_pi

```
f64* g_pi = NULL  [static]
```

4.3.3.5 g_pi_cache

```
f64* g_pi_cache = NULL  [static]
```

4.3.3.6 g_removed_set

```
bitset* g_removed_set = NULL  [static]
```

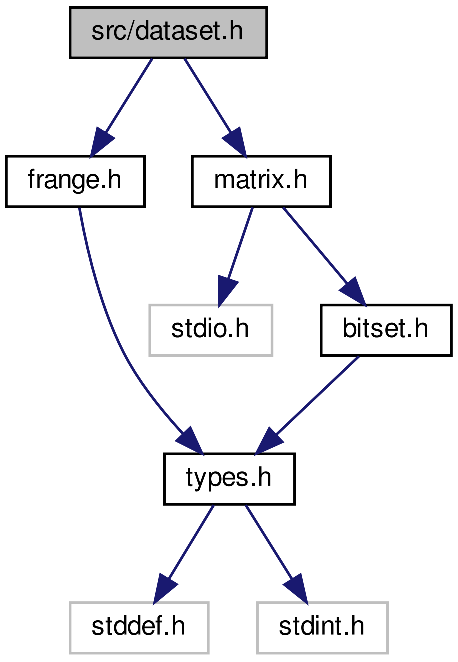
4.3.3.7 g_subgraph

```
matrix* g_subgraph = NULL  [static]
```

4.4 src/dataset.h File Reference

```
#include "frange.h"
#include "matrix.h"
```

Include dependency graph for dataset.h:



Functions

- void dataset_clear ()

  Clears internal dataset cache.
- int dataset_init (const matrix ∗original_graph, const frange ∗alpha)

  Initialize internal dataset cache.
- void generate_dataset (FILE ∗output_file, u32 n, const frange ∗r)

  Generates the dataset.

4.4.1 Function Documentation

4.4.1.1 dataset_clear()

```
void dataset_clear ( )
```

Clears internal dataset cache.

4.4.1.2 dataset_init()

```
int dataset_init (
             const matrix * original_graph,
             const frange * alpha )
```

Initialize internal dataset cache.

Parameters

original_graph	The matrix representation of the original graph.
alpha	The range of alpha values.

Returns

0 on success, -1 on error.

4.4.1.3 generate_dataset()

```
void generate_dataset (
             FILE * output_file,
             u32 n,
             const frange * r )
```

Generates the dataset.

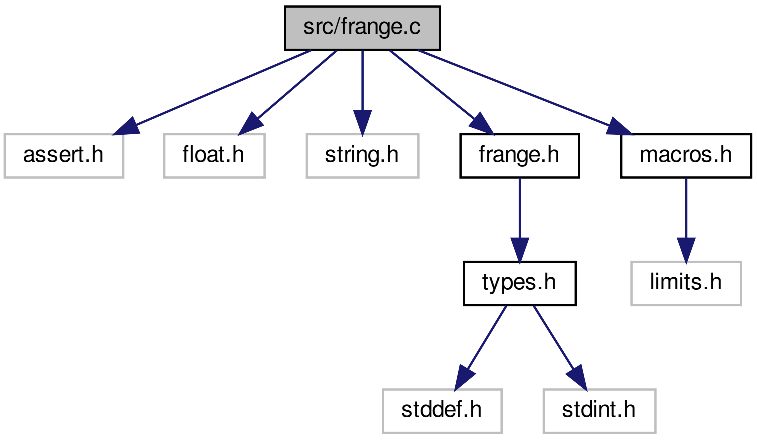
Parameters

output_file	The file where the results will be stored.
n	The number of subgraphs to generate.
r	r The range of ratio of removed vertices to test.

4.5 src/frange.c File Reference

```
#include <assert.h>
#include <float.h>
#include <string.h>
#include "frange.h"
#include "macros.h"
```

Include dependency graph for frange.c:



Functions

- frange ∗ frange_copy (frange ∗dst, const frange ∗src)

  Copys a range.
- void frange_init (frange ∗r, f64 begin, f64 end, f64 step)

  Initializes a range.
- int frange_has_next (const frange ∗r)

  Checks if a range is still valid.
- f64 frange_next (frange ∗r)

  Iterates over a range.

4.5.1 Function Documentation

4.5.1.1 frange_copy()

```
frange* frange_copy (
            frange ∗ dst,
            const frange ∗ src )
```

Copys a range.

Parameters

dst	The destination range.
src	The range to copy.

Returns

The destination range.

4.5.1.2 frange_has_next()

```
int frange_has_next (
            const frange * r )
```

Checks if a range is still valid.

Parameters

r	The range to check.

Returns

The number of steps remaining in the range.

4.5.1.3 frange_init()

```
void frange_init (
            frange * r,
            f64 begin,
            f64 end,
            f64 step )
```

Initializes a range.

Parameters

r	The range to initialize.
begin	The beginning of the range.
end	The end of the range.
step	The step size.

4.5.1.4 frange_next()

```
f64 frange_next (
            frange * r )
```

Iterates over a range.

The range is modified in-place.

Parameters

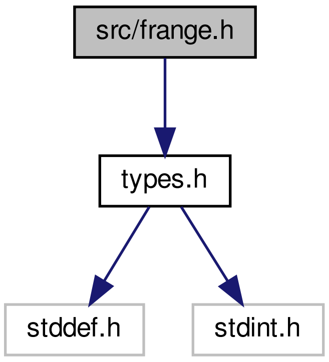
r	The range to iterate over.

Returns

The next value in the range.

4.6 src/frange.h File Reference

`#include "types.h"`
Include dependency graph for frange.h:



Classes

- struct s_frange

  A range datatype to iterate over a range of floating point numbers.

Typedefs

- typedef struct s_frange frange

  A range datatype to iterate over a range of floating point numbers.

Functions

- frange ∗ frange_copy (frange ∗dst, const frange ∗src)

  Copys a range.
- void frange_init (frange ∗r, f64 begin, f64 end, f64 step)

  Initializes a range.
- int frange_has_next (const frange ∗r)

  Checks if a range is still valid.
- f64 frange_next (frange ∗r)

  Iterates over a range.

4.6.1 Typedef Documentation

4.6.1.1 frange

```
typedef struct s_frange frange
```

A range datatype to iterate over a range of floating point numbers.

4.6.2 Function Documentation

4.6.2.1 frange_copy()

```
frange* frange_copy (
            frange * dst,
            const frange * src )
```

Copys a range.

Parameters

dst	The destination range.
src	The range to copy.

Returns

The destination range.

4.6.2.2 frange_has_next()

```
int frange_has_next (
            const frange * r )
```

Checks if a range is still valid.

Parameters

r	The range to check.

Returns

The number of steps remaining in the range.

4.6.2.3 frange_init()

```
void frange_init (
            frange * r,
            f64 begin,
            f64 end,
            f64 step )
```

Initializes a range.

Parameters

r	The range to initialize.
begin	The beginning of the range.
end	The end of the range.
step	The step size.

4.6.2.4 frange_next()

```
f64 frange_next (
            frange * r )
```

Iterates over a range.

The range is modified in-place.

Parameters

r	The range to iterate over.

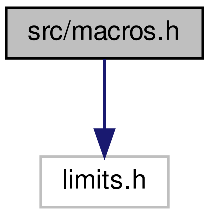
Returns

The next value in the range.

4.7 src/macros.h File Reference

```
#include <limits.h>
```

Include dependency graph for macros.h:



Macros

- #define BIT_SIZEOF(N) (sizeof(N) ∗ CHAR_BIT)

  Gets the size in bits instead of bytes.
- #define IN_BOUNDS(MIN, X, MAX) ((X) >= (MIN) && (X) <= (MAX))

  Checks if a number is in a range.
- #define SWAP(A, B)

  Swap the content of two variables.

4.7.1 Macro Definition Documentation

4.7.1.1 BIT_SIZEOF

```
#define BIT_SIZEOF(
              N ) (sizeof(N) * CHAR_BIT)
```

Gets the size in bits instead of bytes.

Parameters

N	The number of bytes.

Returns

The size in bits of N bytes.

4.7.1.2 IN_BOUNDS

```
#define IN_BOUNDS(
              MIN,
              X,
              MAX ) ((X) >= (MIN) && (X) <= (MAX))
```

Checks if a number is in a range.

Parameters

MIN	The minimum value of the range.
MAX	The maximum value of the range.
X	The number to check.

Returns

1 if X is in the range [MIN, MAX], 0 otherwise.

4.7.1.3 SWAP

```
#define SWAP(
                A,
                B )
```

Value:

```
do { \
        typeof(A) C = (A); \
        (A) = (B); \
        (B) = C; \
    } while (0)
```

Swap the content of two variables.

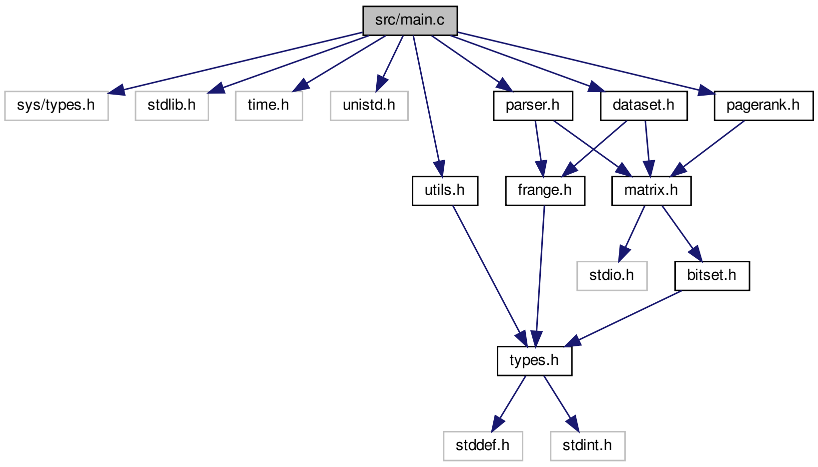
Parameters

A	The first variable.
B	The second variable.

4.8 src/main.c File Reference

```
#include <sys/types.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "dataset.h"
#include "pagerank.h"
#include "parser.h"
#include "utils.h"
```

Include dependency graph for main.c:



Functions

- static int show_usage (const char ∗binary_name)

  Prints a usage message.
- int main (int ac, char ∗∗av)

4.8.1 Function Documentation

4.8.1.1 main()

```
int main (
              int ac,
              char ** av )
```

4.8.1.2 show_usage()

```
static int show_usage (
              const char * binary_name )  [static]
```

Prints a usage message.

Parameters

binary_name	The name of the binary.

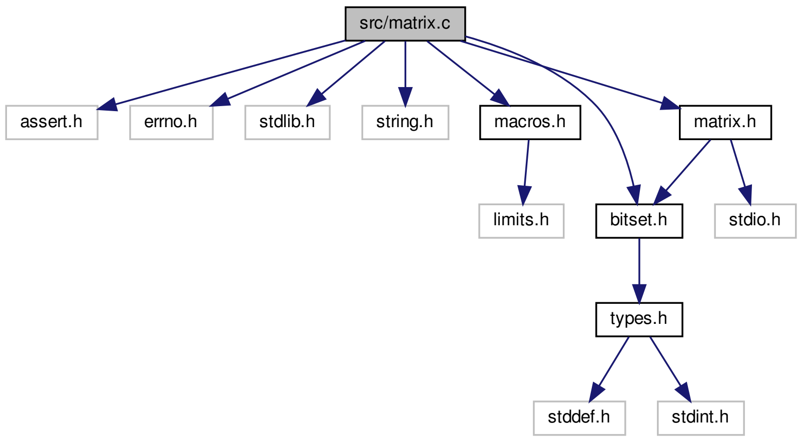
Returns

Always EXIT_FAILURE.

4.9 src/matrix.c File Reference

```
#include <assert.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include "bitset.h"
#include "macros.h"
#include "matrix.h"
```

Include dependency graph for matrix.c:



Functions

- static int edge_init_from_file (edge ∗e, FILE ∗f)

  Initiliazes an edge from a stream.
- static void edge_init (edge ∗e, u32 y, f64 w)

  Initializes an edge.
- void matrix_cache_clear ()

  Clears internal matrix cache.
- int matrix_cache_init (const matrix ∗m)

  Initialize internal matrix cache for subgraph generation.
- void matrix_destroy (matrix ∗m)

  Frees a matrix.
- void matrix_generate_subgraph (matrix ∗dst, const matrix ∗src, f64 r, bitset ∗removed_set)

  Generate a subgraph based on the given matrix with n random vertices removed.
- matrix ∗ matrix_init_from_file (FILE ∗f)

  Initializes a new matrix from a file.
- matrix ∗ matrix_init (u32 vertices_count, u32 edges_count)

  Initializes a new matrix.
- int matrix_print (const matrix ∗m, FILE ∗f)

Prints a matrix on a file stream.

- const edge ∗ matrix_row (const matrix ∗m, usize i)

  Returns the begining of the row at the given index.

- usize matrix_row_count (const matrix ∗m, usize i)

  Returns the size of the row at the given index.

Variables

- static u32 ∗ g_vertices_set = NULL
- static u32 ∗ g_edges_count_set = NULL

4.9.1 Function Documentation

4.9.1.1 edge_init()

```
static void edge_init (
             edge ∗ e,
             u32 y,
             f64 w )  [static]
```

Initializes an edge.

Parameters

e	The edge to initialize.
y	The vertex index.
w	The weight.

4.9.1.2 edge_init_from_file()

```
static int edge_init_from_file (
             edge ∗ e,
             FILE ∗ f )  [static]
```

Initiliazes an edge from a stream.

Parameters

e	The edge to initialize.
f	The file to read the row from.

Returns

0 on success, -1 on error.

4.9.1.3 matrix_cache_clear()

```
void matrix_cache_clear ( )
```

Clears internal matrix cache.

4.9.1.4 matrix_cache_init()

```
int matrix_cache_init (
             const matrix * m )
```

Initialize internal matrix cache for subgraph generation.

Parameters

m	The original matrix.

Returns

0 on success, -1 on error.

4.9.1.5 matrix_destroy()

```
void matrix_destroy (
             matrix * m )
```

Frees a matrix.

If the matrix is NULL, nothing happens.

Parameters

m	The matrix.

4.9.1.6 matrix_generate_subgraph()

```
void matrix_generate_subgraph (
             matrix * dst,
             const matrix * src,
             f64 r,
             bitset * removed_set )
```

Generate a subgraph based on the given matrix with n random vertices removed.

The dst matrix must be allocated and have enough space to store the subgraph. The removed vertices are stored in the given bitset. The bitset must have enough space to store the number of vertices of the src matrix.

Parameters

dst	The destination matrix.
src	The source matrix.
r	The ratio of vertices to remove.
removed_set	The set of removed vertices.

4.9.1.7 matrix_init()

```
matrix* matrix_init (
             u32 vertices_count,
             u32 edges_count )
```

Initializes a new matrix.

Parameters

vertices_count	The number of vertices.
edges_count	The number of edges.

Returns

The new matrix or NULL if the allocation failed.

4.9.1.8 matrix_init_from_file()

```
matrix* matrix_init_from_file (
             FILE * f )
```

Initializes a new matrix from a file.

Parameters

f	The file stream.

Returns

The new matrix or NULL if the allocation failed.

4.9.1.9 matrix_print()

```
int matrix_print (
             const matrix * m,
             FILE * f )
```

Prints a matrix on a file stream.

Parameters

m	The matrix.
f	The file stream.

Returns

EOF if an error occured, 0 otherwise.

4.9.1.10 matrix_row()

```
const edge* matrix_row (
            const matrix * m,
            usize i )
```

Returns the begining of the row at the given index.

Parameters

m	The matrix.
i	The index of the row.

Returns

The first edge of the row.

4.9.1.11 matrix_row_count()

```
usize matrix_row_count (
            const matrix * m,
            usize i )
```

Returns the size of the row at the given index.

Parameters

m	The matrix.
i	The index of the row.

Returns

The number of elements in the row.

4.9.2 Variable Documentation

4.9.2.1 g_edges_count_set

```
u32* g_edges_count_set = NULL  [static]
```

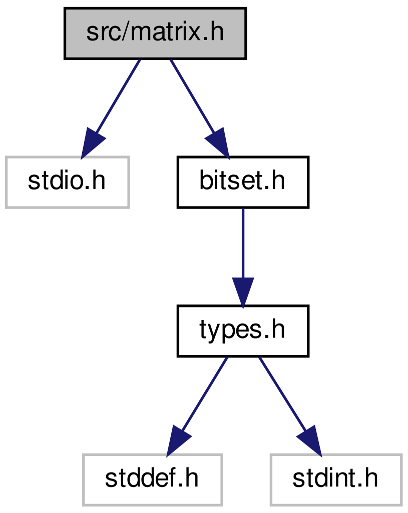
4.9.2.2 g_vertices_set

```
u32* g_vertices_set = NULL  [static]
```

4.10 src/matrix.h File Reference

```
#include <stdio.h>
#include "bitset.h"
```

Include dependency graph for matrix.h:



Classes

- struct s_edge

  An edge of the spare matrix.
- struct s_matrix

  A spare matrix representation.

Typedefs

- typedef struct s_edge edge

  An edge of the spare matrix.
- typedef struct s_matrix matrix

  A spare matrix representation.

Functions

- void matrix_cache_clear ()

    Clears internal matrix cache.
- int matrix_cache_init (const matrix ∗m)

    Initialize internal matrix cache for subgraph generation.
- void matrix_destroy (matrix ∗m)

    Frees a matrix.
- void matrix_generate_subgraph (matrix ∗dst, const matrix ∗src, f64 r, bitset ∗removed_set)

    Generate a subgraph based on the given matrix with n random vertices removed.
- matrix ∗ matrix_init_from_file (FILE ∗f)

    Initializes a new matrix from a file.
- matrix ∗ matrix_init (u32 vertices_count, u32 edges_count)

    Initializes a new matrix.
- int matrix_print (const matrix ∗m, FILE ∗f)

    Prints a matrix on a file stream.
- const edge ∗ matrix_row (const matrix ∗m, usize i)

    Returns the begining of the row at the given index.
- usize matrix_row_count (const matrix ∗m, usize i)

    Returns the size of the row at the given index.

4.10.1 Typedef Documentation

4.10.1.1 edge

```
typedef struct s_edge edge
```

An edge of the spare matrix.

4.10.1.2 matrix

```
typedef struct s_matrix matrix
```

A spare matrix representation.

4.10.2 Function Documentation

4.10.2.1 matrix_cache_clear()

```
void matrix_cache_clear ( )
```

Clears internal matrix cache.

4.10.2.2 matrix_cache_init()

```
int matrix_cache_init (
             const matrix * m )
```

Initialize internal matrix cache for subgraph generation.

Parameters

m	The original matrix.

Returns

0 on success, -1 on error.

4.10.2.3 matrix_destroy()

```
void matrix_destroy (
            matrix * m )
```

Frees a matrix.

If the matrix is NULL, nothing happens.

Parameters

m	The matrix.

4.10.2.4 matrix_generate_subgraph()

```
void matrix_generate_subgraph (
            matrix * dst,
            const matrix * src,
            f64 r,
            bitset * removed_set )
```

Generate a subgraph based on the given matrix with n random vertices removed.

The dst matrix must be allocated and have enough space to store the subgraph. The removed vertices are stored in the given bitset. The bitset must have enough space to store the number of vertices of the src matrix.

Parameters

dst	The destination matrix.
src	The source matrix.
r	The ratio of vertices to remove.
removed_set	The set of removed vertices.

4.10.2.5 matrix_init()

```
matrix* matrix_init (
            u32 vertices_count,
            u32 edges_count )
```

Initializes a new matrix.

Parameters

vertices_count	The number of vertices.
edges_count	The number of edges.

Returns

The new matrix or NULL if the allocation failed.

4.10.2.6 matrix_init_from_file()

```
matrix* matrix_init_from_file (
              FILE * f )
```

Initializes a new matrix from a file.

Parameters

f	The file stream.

Returns

The new matrix or NULL if the allocation failed.

4.10.2.7 matrix_print()

```
int matrix_print (
              const matrix * m,
              FILE * f )
```

Prints a matrix on a file stream.

Parameters

m	The matrix.
f	The file stream.

Returns

EOF if an error occured, 0 otherwise.

4.10.2.8 matrix_row()

```
const edge* matrix_row (
             const matrix * m,
             usize i )
```

Returns the begining of the row at the given index.

Parameters

m	The matrix.
i	The index of the row.

Returns

The first edge of the row.

4.10.2.9 matrix_row_count()

```
usize matrix_row_count (
             const matrix * m,
             usize i )
```

Returns the size of the row at the given index.

Parameters

m	The matrix.
i	The index of the row.

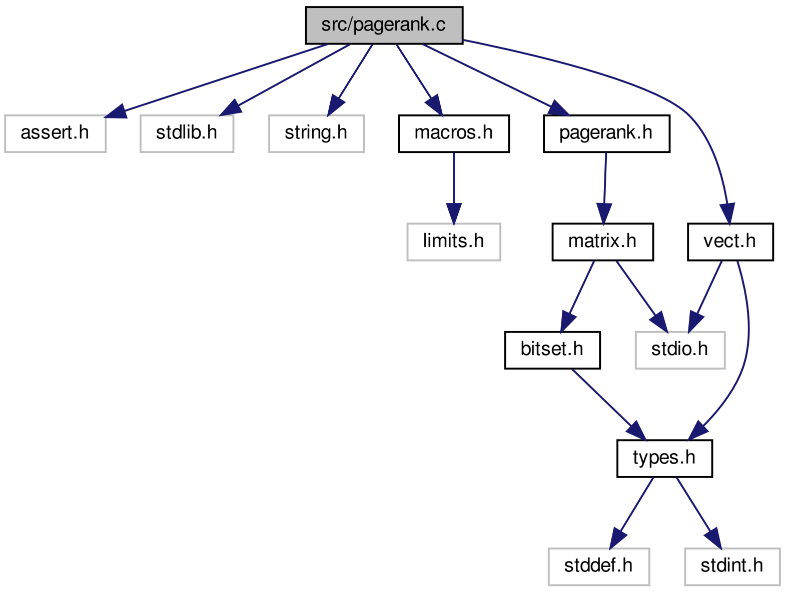
Returns

The number of elements in the row.

4.11 src/pagerank.c File Reference

```
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include "macros.h"
#include "pagerank.h"
#include "vect.h"
```

Include dependency graph for pagerank.c:



Functions

- static void init_f (const matrix ∗m, bitset ∗f)
- static f64 vect_mul_cf (const f64 ∗x, const bitset ∗f, usize n)

  Computes r = x ∗ f'.
- static void vect_mul_p (f64 ∗r, const f64 ∗x, const matrix ∗p)

  Computes r = x ∗ p.
- void pagerank_clear ()

  Clears internal PageRank cache.
- int pagerank_init (u64 n)

  Initialize internal PageRank cache.
- u32 pagerank (const matrix ∗m, f64 ∗init_vect, f64 alpha)

  PageRank algorithm.

Variables

- static f64 ∗ g_pi_cache = NULL
- static bitset ∗ g_f_cache = NULL
- static usize g_pi_cache_size = 0

4.11.1 Function Documentation

4.11.1.1 init_f()

```
static void init_f (
             const matrix * m,
             bitset * f )  [static]
```

4.11.1.2 pagerank()

```
u32 pagerank (
             const matrix * m,
             f64 * init_vect,
             f64 alpha )
```

PageRank algorithm.

pagerank_init() must be called before this function. Stores the result in the given vector.

Parameters

m	The matrix to use.
init_vect	The initial vector to use.
alpha	The damping factor.

Returns

The number of iterations needed to converge.

4.11.1.3 pagerank_clear()

```
void pagerank_clear ( )
```

Clears internal PageRank cache.

4.11.1.4 pagerank_init()

```
int pagerank_init (
             u64 n )
```

Initialize internal PageRank cache.

Parameters

n	The maximum number of vertices.

Returns

0 on success, -1 on error.

4.11.1.5 vect_mul_cf()

```
static f64 vect_mul_cf (
             const f64 * x,
             const bitset * f,
             usize n )  [static]
```

Computes r = x * f'.

Parameters

x	The row vector.
f	The bitset as a column vector.
n	The size of both vectors.

Returns

The result of the multiplication as a scalar.

4.11.1.6 vect_mul_p()

```
static void vect_mul_p (
             f64 * r,
             const f64 * x,
             const matrix * p )  [static]
```

Computes r = x * p.

Parameters

r	The result vector.
x	The row vector.
p	The matrix.

4.11.2 Variable Documentation

4.11.2.1 g_f_cache

```
bitset* g_f_cache = NULL  [static]
```

4.11.2.2 g_pi_cache

```
f64* g_pi_cache = NULL  [static]
```

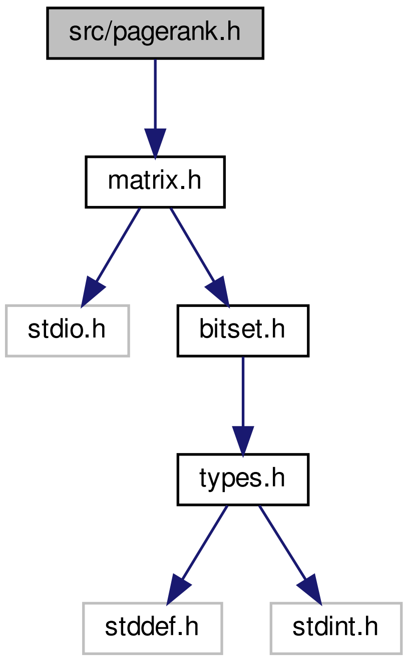
4.11.2.3 g_pi_cache_size

```
usize g_pi_cache_size = 0  [static]
```

4.12 src/pagerank.h File Reference

```
#include "matrix.h"
```

Include dependency graph for pagerank.h:



Macros

- #define PAGERANK_EPSILON 1e-6

  The convergence threshold for the PageRank algorithm.

Functions

- void pagerank_clear ()

  Clears internal PageRank cache.
- int pagerank_init (u64 n)

  Initialize internal PageRank cache.
- u32 pagerank (const matrix ∗m, f64 ∗init_vect, f64 alpha)

  PageRank algorithm.

4.12.1 Macro Definition Documentation

4.12.1.1 PAGERANK_EPSILON

```
#define PAGERANK_EPSILON 1e-6
```

The convergence threshold for the PageRank algorithm.

4.12.2 Function Documentation

4.12.2.1 pagerank()

```
u32 pagerank (
             const matrix * m,
             f64 * init_vect,
             f64 alpha )
```

PageRank algorithm.

pagerank_init() must be called before this function. Stores the result in the given vector.

Parameters

m	The matrix to use.
init_vect	The initial vector to use.
alpha	The damping factor.

Returns

The number of iterations needed to converge.

4.12.2.2 pagerank_clear()

```
void pagerank_clear ( )
```

Clears internal PageRank cache.

4.12.2.3 pagerank_init()

```
int pagerank_init (
             u64 n )
```

Initialize internal PageRank cache.

Parameters

n	The maximum number of vertices.

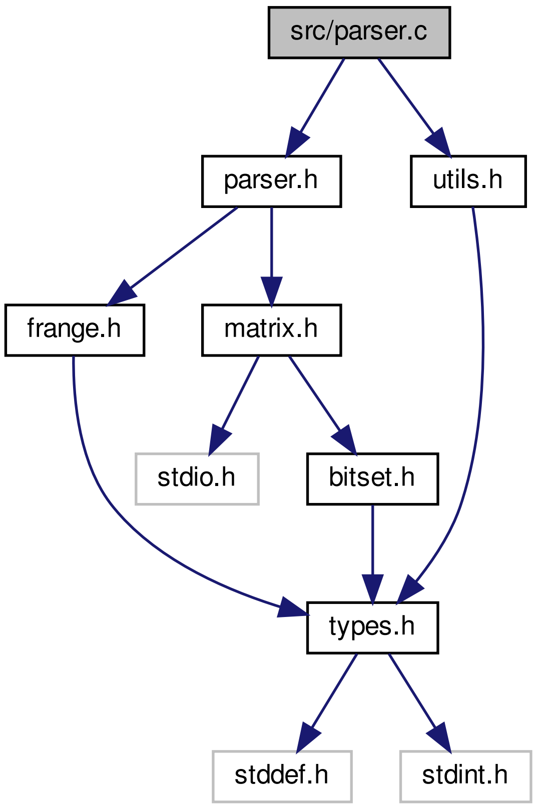
Returns

0 on success, -1 on error.

4.13 src/parser.c File Reference

```
#include "parser.h"
#include "utils.h"
```

Include dependency graph for parser.c:



Functions

- void check_range (const char ∗arg, f64 min_value, f64 max_value, int ∗ec)
- FILE ∗ parse_file (const char ∗path, const char ∗mode, int ∗ec)

  Try to open a file.
- matrix ∗ parse_matrix (const char ∗path, FILE ∗file, int ∗ec)

Parse the graph file.
- u32 parse_non_negative (const char ∗arg, int ∗ec)

  Parse a non negative integer.
- frange parse_range (const char ∗arg1, const char ∗arg2, const char ∗arg3, int ∗ec)

  Parse a range of floating point numbers.
- f64 parse_ratio (const char ∗arg, int ∗ec)

  Parse the ratio of the number of vertices to remove.

4.13.1 Function Documentation

4.13.1.1 check_range()

```
void check_range (
             const char * arg,
             f64 min_value,
             f64 max_value,
             int * ec )
```

4.13.1.2 parse_file()

```
FILE* parse_file (
             const char * path,
             const char * mode,
             int * ec )
```

Try to open a file.

Increments the errors counter if an error occured.

Parameters

path	The path of the file.
mode	The mode to open the file.
ec	The errors counter.

Returns

The opened file or NULL if an error occured.

4.13.1.3 parse_matrix()

```
matrix* parse_matrix (
             const char * path,
             FILE * file,
             int * ec )
```

Parse the graph file.

Increments the errors counter if an error occured.

Parameters

path	The original path of the file.
file	The file to parse.
ec	The errors counter.

Returns

The loaded graph or NULL if an error occured.

4.13.1.4 parse_non_negative()

```
u32 parse_non_negative (
             const char * arg,
             int * ec )
```

Parse a non negative integer.

Increments the errors counter if an error occured.

Parameters

arg	The argument to parse.
ec	The errors counter.

Returns

The parsed integer.

4.13.1.5 parse_range()

```
frange parse_range (
             const char * arg1,
             const char * arg2,
             const char * arg3,
             int * ec )
```

Parse a range of floating point numbers.

Increments the errors counter if an error occured.

Parameters

arg1	The beginning of the range.
arg2	The end of the range.
arg3	The number of steps in the range.
ec	The errors counter.

Returns

The parsed range.

4.13.1.6 parse_ratio()

```
f64 parse_ratio (
             const char * arg,
             int * ec )
```

Parse the ratio of the number of vertices to remove.

Increments the errors counter if an error occured.

Parameters

arg	The argument to parse.
ec	The errors counter.

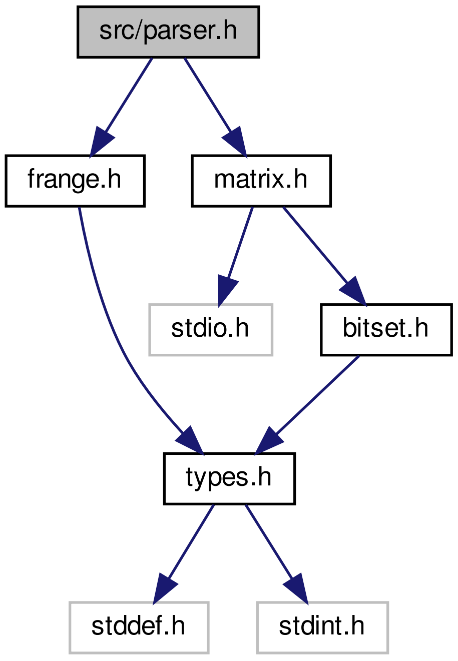
Returns

The ratio of the number of vertices to remove.

4.14 src/parser.h File Reference

```
#include "frange.h"
#include "matrix.h"
```

Include dependency graph for parser.h:



Functions

- FILE ∗ parse_file (const char ∗path, const char ∗mode, int ∗ec)

    Try to open a file.
- matrix ∗ parse_matrix (const char ∗path, FILE ∗file, int ∗ec)

    Parse the graph file.
- u32 parse_non_negative (const char ∗arg, int ∗ec)

    Parse a non negative integer.
- frange parse_range (const char ∗arg1, const char ∗arg2, const char ∗arg3, int ∗ec)

    Parse a range of floating point numbers.
- f64 parse_ratio (const char ∗arg, int ∗ec)

    Parse the ratio of the number of vertices to remove.

4.14.1 Function Documentation

4.14.1.1 parse_file()

```
FILE* parse_file (
             const char * path,
             const char * mode,
             int * ec )
```

Try to open a file.

Increments the errors counter if an error occured.

Parameters

path	The path of the file.
mode	The mode to open the file.
ec	The errors counter.

Returns

The opened file or NULL if an error occured.

4.14.1.2 parse_matrix()

```
matrix* parse_matrix (
             const char * path,
             FILE * file,
             int * ec )
```

Parse the graph file.

Increments the errors counter if an error occured.

Parameters

path	The original path of the file.
file	The file to parse.
ec	The errors counter.

Returns

The loaded graph or NULL if an error occured.

4.14.1.3 parse_non_negative()

```
u32 parse_non_negative (
             const char * arg,
             int * ec )
```

Parse a non negative integer.

Increments the errors counter if an error occured.

Parameters

arg	The argument to parse.
ec	The errors counter.

Returns

The parsed integer.

4.14.1.4 parse_range()

```
frange parse_range (
            const char * arg1,
            const char * arg2,
            const char * arg3,
            int * ec )
```

Parse a range of floating point numbers.

Increments the errors counter if an error occured.

Parameters

arg1	The beginning of the range.
arg2	The end of the range.
arg3	The number of steps in the range.
ec	The errors counter.

Returns

The parsed range.

4.14.1.5 parse_ratio()

```
f64 parse_ratio (
            const char * arg,
            int * ec )
```

Parse the ratio of the number of vertices to remove.

Increments the errors counter if an error occured.

Parameters

arg	The argument to parse.
ec	The errors counter.

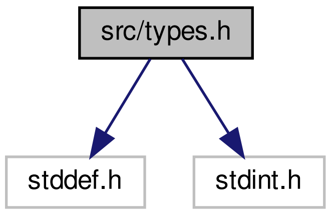
Returns

The ratio of the number of vertices to remove.

4.15 src/types.h File Reference

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for types.h:



Typedefs

- typedef int8_t s8
- typedef int16_t s16
- typedef int32_t s32
- typedef int64_t s64
- typedef uint8_t u8
- typedef uint16_t u16
- typedef uint32_t u32
- typedef uint64_t u64
- typedef float f32
- typedef double f64
- typedef size_t usize

4.15.1 Typedef Documentation

4.15.1.1 f32

```
typedef float f32
```

4.15.1.2 f64

```
typedef double f64
```

4.15.1.3 s16

```
typedef int16_t s16
```

4.15.1.4 s32

```
typedef int32_t s32
```

4.15.1.5 s64

```
typedef int64_t s64
```

4.15.1.6 s8

```
typedef int8_t s8
```

4.15.1.7 u16

```
typedef uint16_t u16
```

4.15.1.8 u32

```
typedef uint32_t u32
```

4.15.1.9 u64

```
typedef uint64_t u64
```

4.15.1.10 u8

```
typedef uint8_t u8
```

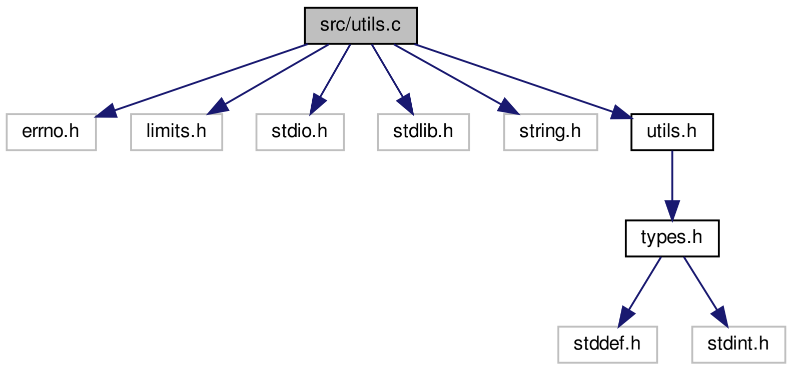
4.15.1.11 usize

```
typedef size_t usize
```

4.16 src/utils.c File Reference

```
#include <errno.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "utils.h"
```

Include dependency graph for utils.c:



Functions

- const char ∗ get_errno_error ()

  Returns the error message of the last error using the errno variable.
- int print_error (const char ∗context, const char ∗message)

  Prints an error message on stderr.
- int stof64 (const char ∗str, f64 ∗f)

  Parse a string to a floating point number.
- int stou32 (const char ∗str, u32 ∗u)

  Parse a string to an unsigned integer in base 10.

4.16.1 Function Documentation

4.16.1.1 get_errno_error()

```
const char* get_errno_error ( )
```

Returns the error message of the last error using the errno variable.

Returns

The error message.

4.16.1.2 print_error()

```
int print_error (
            const char * context,
            const char * message )
```

Prints an error message on stderr.

Parameters

context	The context of the error.
message	The error message. If NULL, the last error from errno is used.

Returns

Always 1.

4.16.1.3 stof64()

```
int stof64 (
            const char * str,
            f64 * f )
```

Parse a string to a floating point number.

Parameters

str	The string to parse.
f	The pointer to the floating point number.

Returns

0 if the string is a number, -1 otherwise (errno is set).

4.16.1.4 stou32()

```
int stou32 (
            const char * str,
            u32 * u )
```

Parse a string to an unsigned integer in base 10.

Parameters

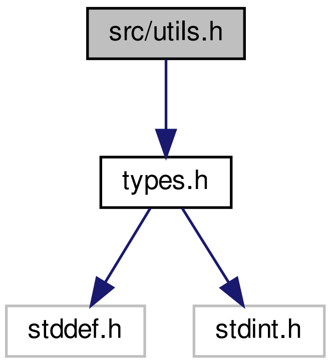
str	The string to parse.
u	The pointer to the unsigned integer.

Returns

0 if the string is a valid number, -1 otherwise (errno is set).

4.17 src/utils.h File Reference

```
#include "types.h"
```
Include dependency graph for utils.h:



Functions

- const char ∗ get_errno_error ()

  Returns the error message of the last error using the errno variable.
- int print_error (const char ∗context, const char ∗message)

  Prints an error message on stderr.
- int stof64 (const char ∗str, f64 ∗f)

  Parse a string to a floating point number.
- int stou32 (const char ∗str, u32 ∗u)

  Parse a string to an unsigned integer in base 10.

4.17.1 Function Documentation

4.17.1.1 get_errno_error()

```
const char* get_errno_error ( )
```

Returns the error message of the last error using the errno variable.

Returns

The error message.

4.17.1.2 print_error()

```
int print_error (
            const char * context,
            const char * message )
```

Prints an error message on stderr.

Parameters

context	The context of the error.
message	The error message. If NULL, the last error from errno is used.

Returns

Always 1.

4.17.1.3 stof64()

```
int stof64 (
            const char * str,
            f64 * f )
```

Parse a string to a floating point number.

Parameters

str	The string to parse.
f	The pointer to the floating point number.

Returns

0 if the string is a number, -1 otherwise (errno is set).

4.17.1.4 stou32()

```
int stou32 (
            const char * str,
            u32 * u )
```

Parse a string to an unsigned integer in base 10.

Parameters

str	The string to parse.
u	The pointer to the unsigned integer.

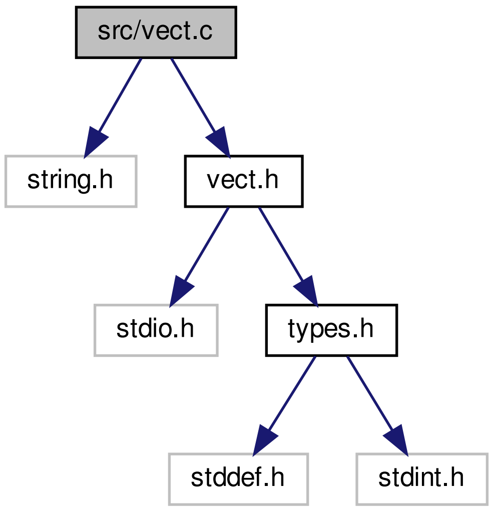
Returns

0 if the string is a valid number, -1 otherwise (errno is set).

4.18 src/vect.c File Reference

```
#include <string.h>
#include "vect.h"
```

Include dependency graph for vect.c:



Functions

- f64 ∗ vect_copy (f64 ∗dst, const f64 ∗src, usize n)

  Copys a vector.
- void vect_mul_add_f64 (f64 ∗v, usize n, f64 f, f64 g)

  Performs the following operation: $v = v * f + g$.
- f64 vect_norm1 (const f64 ∗v, const f64 ∗w, usize n)

  Computes the 1-norm of two vectors.
- int vect_print (const f64 ∗v, usize n, FILE ∗f)

  Prints a vector on a file stream.
- void vect_set (f64 ∗v, usize n, f64 f)

  Sets all elements of a vector to a floating point number.

4.18.1 Function Documentation

4.18.1.1 vect_copy()

```
f64* vect_copy (
             f64 * dst,
             const f64 * src,
             usize n )
```

Copys a vector.

Parameters

dst	The destination vector.
src	The vector to copy.
n	The size of both vectors.

Returns

The destination vector.

4.18.1.2 vect_mul_add_f64()

```
void vect_mul_add_f64 (
             f64 * v,
             usize n,
             f64 f,
             f64 g )
```

Performs the following operation: v = v ∗ f + g.

Parameters

v	The result vector.
n	The size of the vector.
f	The floating point number to multiply with.
g	The floating point number to add.

4.18.1.3 vect_norm1()

```
f64 vect_norm1 (
             const f64 * v,
             const f64 * w,
             usize n )
```

Computes the 1-norm of two vectors.

Parameters

v	The first vector.
w	The second vector.
n	The size of both vectors.

Returns

The 1-norm of the two vectors.

4.18.1.4 vect_print()

```
int vect_print (
            const f64 * v,
            usize n,
            FILE * f )
```

Prints a vector on a file stream.

Parameters

v	The vector.
n	The size of the vector.
f	The file stream.

Returns

EOF if an error occured, 0 otherwise.

4.18.1.5 vect_set()

```
void vect_set (
            f64 * v,
            usize n,
            f64 f )
```

Sets all elements of a vector to a floating point number.

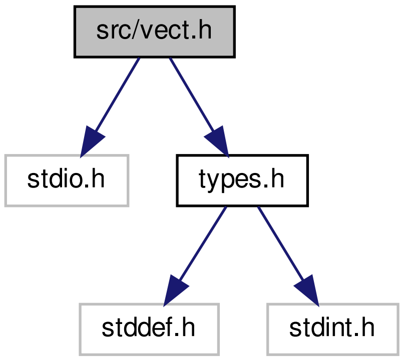
Parameters

v	The vector.
n	The size of the vector.
f	The floating point number to set.

4.19 src/vect.h File Reference

```
#include <stdio.h>
#include "types.h"
```

Include dependency graph for vect.h:



Functions

- f64 ∗ vect_copy (f64 ∗dst, const f64 ∗src, usize n)

  Copys a vector.
- void vect_mul_add_f64 (f64 ∗v, usize n, f64 f, f64 g)

  Performs the following operation: $v = v * f + g$.
- f64 vect_norm1 (const f64 ∗v, const f64 ∗w, usize n)

  Computes the 1-norm of two vectors.
- int vect_print (const f64 ∗v, usize n, FILE ∗f)

  Prints a vector on a file stream.
- void vect_set (f64 ∗v, usize n, f64 f)

  Sets all elements of a vector to a floating point number.

4.19.1 Function Documentation

4.19.1.1 vect_copy()

```
f64* vect_copy (
            f64 * dst,
            const f64 * src,
            usize n )
```

Copys a vector.

Parameters

dst	The destination vector.
src	The vector to copy.
n	The size of both vectors.

Returns

The destination vector.

4.19.1.2 vect_mul_add_f64()

```
void vect_mul_add_f64 (
            f64 * v,
            usize n,
            f64 f,
            f64 g )
```

Performs the following operation: v = v ∗ f + g.

Parameters

v	The result vector.
n	The size of the vector.
f	The floating point number to multiply with.
g	The floating point number to add.

4.19.1.3 vect_norm1()

```
f64 vect_norm1 (
            const f64 * v,
            const f64 * w,
            usize n )
```

Computes the 1-norm of two vectors.

Parameters

v	The first vector.
w	The second vector.
n	The size of both vectors.

Returns

The 1-norm of the two vectors.

4.19.1.4 vect_print()

```
int vect_print (
            const f64 * v,
            usize n,
            FILE * f )
```

Prints a vector on a file stream.

Parameters

v	The vector.
n	The size of the vector.
f	The file stream.

Returns

EOF if an error occured, 0 otherwise.

4.19.1.5 vect_set()

```
void vect_set (
            f64 * v,
            usize n,
            f64 f )
```

Sets all elements of a vector to a floating point number.

Parameters

v	The vector.
n	The size of the vector.
f	The floating point number to set.

Index