Parallelize-Tarjan-MPI-CUDA

Generated by Doxygen 1.9.1

1 Class Index 1

1.1 Class List . . . 1

2 File Index 3

2.1 File List . . . 3

3 Class Documentation 5

3.1 linkedlist_int Struct Reference . . . 5

3.1.1 Detailed Description . . . 5

3.2 lnode_int_t Struct Reference . . . 6

3.2.1 Detailed Description . . . 6

3.3 scc_set_t Struct Reference . . . 6

4 File Documentation 7

4.1 common/src/cuda_graph.c File Reference . . . 7

4.1.1 Detailed Description . . . 8

4.1.2 Function Documentation . . . 8

4.1.2.1 cuda_graph_load_from_file() . . . 8

4.1.2.2 cuda_graph_to_graph() . . . 8

4.2 common/src/graph.c File Reference . . . 8

4.2.1 Detailed Description . . . 11

4.2.2 Function Documentation . . . 11

4.2.2.1 graph_copy() . . . 11

4.2.2.2 graph_delete_edge() . . . 11

4.2.2.3 graph_delete_vertex() . . . 12

4.2.2.4 graph_deserialize() . . . 12

4.2.2.5 graph_free() . . . 12

4.2.2.6 graph_fully_connected_disconnected() . . . 13

4.2.2.7 graph_get_num_vertex() . . . 13

4.2.2.8 graph_init() . . . 13

4.2.2.9 graph_insert_edge() . . . 14

4.2.2.10 graph_insert_vertex() . . . 14

4.2.2.11 graph_load_from_file() . . . 14

4.2.2.12 graph_merge() . . . 15

4.2.2.13 graph_merge_vertices() . . . 15

4.2.2.14 graph_print_debug() . . . 15

4.2.2.15 graph_random() . . . 16

4.2.2.16 graph_save_to_file() . . . 16

4.2.2.17 graph_serialize() . . . 16

4.2.2.18 graph_tarjan() . . . 17

4.2.2.19 graph_tarjan_foreach() . . . 17

4.2.2.20 graph_tarjan_foreach_helper() . . . 18

4.2.2.21 graph_tarjan_helper() . . . 18

4.2.2.22 min() . . . 19
4.2.2.23 scc_set_add() . . . 19
4.2.2.24 scc_set_contains() . . . 20
4.2.2.25 scc_set_free() . . . 20
4.2.2.26 scc_set_init() . . . 20
4.2.2.27 scc_set_load_from_file() . . . 20
4.2.2.28 scc_set_merge() . . . 21
4.2.2.29 scc_set_print_debug() . . . 21
4.2.2.30 scc_set_save_to_file() . . . 21
4.3 common/src/linkedlist.c File Reference . . . 22
4.3.1 Detailed Description . . . 23
4.3.2 Function Documentation . . . 23
4.3.2.1 linkedlist_int_delete() . . . 23
4.3.2.2 linkedlist_int_dequeue() . . . 24
4.3.2.3 linkedlist_int_enqueue() . . . 24
4.3.2.4 linkedlist_int_free() . . . 24
4.3.2.5 linkedlist_int_init() . . . 26
4.3.2.6 linkedlist_int_insert() . . . 26
4.3.2.7 linkedlist_int_length() . . . 26
4.3.2.8 linkedlist_int_pop() . . . 27
4.3.2.9 linkedlist_int_print() . . . 27
4.3.2.10 linkedlist_int_push() . . . 27
4.3.2.11 linkedlist_int_top() . . . 28
4.3.2.12 nodeCreate() . . . 28
4.3.2.13 nodeDestroy() . . . 28
4.4 common/src/random.c File Reference . . . 29
4.4.1 Detailed Description . . . 29
4.4.2 Function Documentation . . . 29
4.4.2.1 rand_bernoulli() . . . 29
4.4.2.2 rand_binomial() . . . 30
4.4.2.3 rand_binomial_2() . . . 30
4.5 common_mpi/src/mpi_logic.c File Reference . . . 31
4.5.1 Detailed Description . . . 32
4.5.2 Function Documentation . . . 32
4.5.2.1 callback() . . . 32
4.5.2.2 master_schedule() . . . 32
4.5.2.3 master_work() . . . 33
4.5.2.4 master_work2() . . . 33
4.5.2.5 slave_work() . . . 34
4.6 tools/rsg/src/main.c File Reference . . . 34
4.6.1 Detailed Description . . . 34
4.7 tools/rgg/src/main.c File Reference . . . 35

4.7.1 Detailed Description 35
4.8 tools/sgg/src/main.c File Reference 36
4.8.1 Detailed Description 36
4.9 tools/print-SCC/src/main.c File Reference 36
4.9.1 Detailed Description 37
4.10 tools/print-graph/src/main.c File Reference 37
4.10.1 Detailed Description 38
4.11 tools/compare/src/main.c File Reference 38
4.11.1 Detailed Description 38

Index **39**

Chapter 1

Class Index

1.1 Class List

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

linkedlist_int
: This struct represents the linkedlist data structure 5

lnode_int_t
: This struct represents the node of the list data structure 6

scc_set_t 6

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

common/src/cuda_graph.c
: This file contains the implementation for the operations involving the data structure cuda_graph: loading from a file, cuda_graph to graph conversion and cuda_graph deallocation 7

common/src/graph.c
: This file implements the abstract data type graph with adjacency maps. It also contains basic operation on the graph (vertices and edges insert and deletion) as well as specific operations needed by the ∗ ∗ MPI and CUDA algorithms. The file also defines the scc_set data structure which is used to store SCCs found by Tarjan's algorithm as well as operations on this data structure 8

common/src/linkedlist.c
: This file implements the abstract data type linkedlist 22

common/src/random.c
: This file contains the implementations for utility functions that are useful for randomly generating graphs 29

common_mpi/src/mpi_logic.c
: This file implements a version of a parallelization of Tarjan's algorithm 31

tools/compare/src/main.c
: This tool compare two different SCC discovered file 38

tools/print-graph/src/main.c
: This tool print to standard output the graph from a file in input 37

tools/print-SCC/src/main.c
: This tool print to standard output the SCC discovered from a file 36

tools/rgg/src/main.c
: This tool generates a graph starting from a seed 35

tools/rsg/src/main.c
: This tool generate a random graph with max_n_node node and each node have mean number of edge with a variance_edge, the number of nodes follows the Bernoulli distribution 34

tools/sgg/src/main.c
: This tool generate graph fully connected, graph fully disconnected or graph bipartite 36

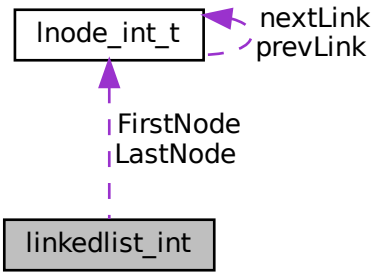
Chapter 3

Class Documentation

3.1 linkedlist_int Struct Reference

This struct represents the linkedlist data structure.

Collaboration diagram for linkedlist_int:



Public Attributes

- lnode_int_t * **FirstNode**
- lnode_int_t * **LastNode**
- int **length**

3.1.1 Detailed Description

This struct represents the linkedlist data structure.

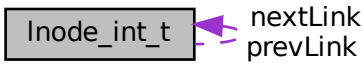
The documentation for this struct was generated from the following file:

- common/src/linkedlist.c

3.2 lnode_int_t Struct Reference

This struct represents the node of the list data structure.

Collaboration diagram for lnode_int_t:



Public Attributes

- int **info**
- lnode_int_t * **nextLink**
- lnode_int_t * **prevLink**

3.2.1 Detailed Description

This struct represents the node of the list data structure.

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

- common/src/linkedlist.c

3.3 scc_set_t Struct Reference

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

- common/src/graph.c

Chapter 4

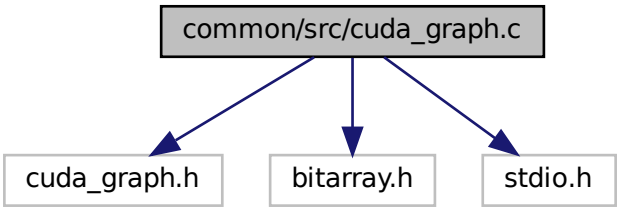
File Documentation

4.1 common/src/cuda_graph.c File Reference

This file contains the implementation for the operations involving the data structure cuda_graph: loading from a file, cuda_graph to graph conversion and cuda_graph deallocation.

```
#include <cuda_graph.h>
#include <bitarray.h>
#include <stdio.h>
```

Include dependency graph for cuda_graph.c:



Functions

- cuda_graph_t * cuda_graph_load_from_file (char *filename)

  This function loads a cuda_graph from a file.
- graph_t * cuda_graph_to_graph (cuda_graph_t *G, int *deleted_bitarray)

  This function converts a cuda_graph in a graph, ignoring the nodes marked in the bitmask.
- void cuda_graph_free (cuda_graph_t *G)

  This function deallocates a cuda_graph.
- void cuda_graph_print_debug (cuda_graph_t *G)

  This function prints the content of a cuda_graph on stdout.

4.1.1 Detailed Description

This file contains the implementation for the operations involving the data structure cuda_graph: loading from a file, cuda_graph to graph conversion and cuda_graph deallocation.

4.1.2 Function Documentation

4.1.2.1 cuda_graph_load_from_file()

```
cuda_graph_t* cuda_graph_load_from_file (
            char * filename )
```

This function loads a cuda_graph from a file.

Parameters

filename	the name of the file to be loaded.

Returns

The loaded cuda_graph

4.1.2.2 cuda_graph_to_graph()

```
graph_t* cuda_graph_to_graph (
            cuda_graph_t * G,
            int * deleted_bitarray )
```

This function converts a cuda_graph in a graph, ignoring the nodes marked in the bitmask.

Parameters

G	the cuda_graph that must be converted
deleted_bitarray	the bitmask containing the nodes that must be ignored during the converion

Returns

the resulting graph

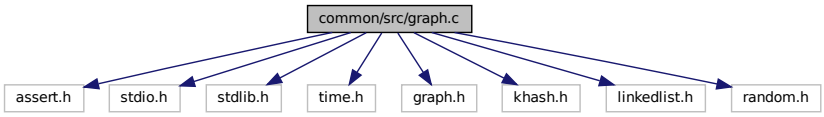
4.2 common/src/graph.c File Reference

This file implements the abstract data type graph with adjacency maps. It also contains basic operation on the graph (vertices and edges insert and deletion) as well as specific operations needed by the ∗ ∗ MPI and CUDA algorithms.

The file also defines the scc_set data structure which is used to store SCCs found by Tarjan's algorithm as well as operations on this data structure.

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "graph.h"
#include "khash.h"
#include "linkedlist.h"
#include "random.h"
```

Include dependency graph for graph.c:



Classes

- struct scc_set_t

Functions

- int min (int a, int b)

  This function takes two integers as input and returns the minimum integer.
- graph_t ∗ graph_init ()

  This function declares and initializes an empty graph_t data structure.
- void graph_free (graph_t ∗G)

  This function takes as input a graph_t data structure and takes care of deallocating the entire memory occupied by the data structure.
- int graph_get_num_vertex (graph_t ∗G)

  This function takes as input a graph_t data structure and returns the number of vertices in the graph.
- void graph_insert_vertex (graph_t ∗G, int v)

  This function takes as input a graph_t data structure and a vertex v; it takes care of inserting the vertex v within the graph.
- void graph_insert_edge (graph_t ∗G, int u, int v)

  This function takes as input a graph_t data structure and two vertices u and v. It takes care of inserting an edge from vertex u to vertex v of the graph.
  Throws an error is the edge already exists.
- void graph_delete_edge (graph_t ∗G, int u, int v)

  This function takes as input a graph_t data structure and two vertices u and v. It deletes the edge from the node u to the node v if it exists. Otherwise does nothing.
- void graph_delete_vertex (graph_t ∗G, int v)

  This function takes as input a graph_t data structure and a vertex v. It deletes the vertex v and every edge incident on the vertex.
- void graph_tarjan_helper (graph_t ∗G, int node, khash_t(m32) ∗disc, khash_t(m32) ∗low, linkedlist_int ∗stack, khash_t(m32) ∗stackMember, int ∗time, array_int ∗result)

  This function is an helper function for graph_tarjan(). For more info see documentation for graph_tarjan().

- array_int * graph_tarjan (graph_t *G)

  This function is a Tarjan's algorithm implementation using recursion.
  This is a modified version of the algorithm on the geeksforgeeks.com website.
  The main differences are:
  - disc, low and stackMember are now hash tables because we remove the hypotesis that vertex ids go from 0 to N-1: when working on a subgraph (as a slave process), there are no guarantees on the order nor continuity of the vertex ids.
  Using hash tables instead of arrays we save a lot of memory.
  - we remove the hypotesis that every vertex in an adjacency map exists in the graph. This is also caused by executions on subgraphs of a given graph.
- void graph_tarjan_foreach_helper (graph_t *G, int node, khash_t(m32) *disc, khash_t(m32) *low, linkedlist_int *stack, khash_t(m32) *stackMember, int *time, array_int *scc, void(*f)(array_int *))

  This function is an helper function for graph_tarjan_foreach(). For more info see documentation for graph_tarjan_foreach().
- void graph_tarjan_foreach (graph_t *G, void(*f)(array_int *))

  This function takes as input a graph and a callback function f. It finds all the SCCs in the graph and each time it finds one it calls the callback function f on the SCC.
- array_int * graph_serialize (graph_t *G, int n, khint_t *bucket)

  This function takes as input a graph, the number of vertices to be serialized n, and a variable to store the reference to the adicence map.
  The function returns an array of integers representing the serialization of the first n vertices of the graph.
  In other words, we go from a representation using pointers to a representation that uses integers only.
- void graph_deserialize (graph_t *G, array_int *buff)

  This function takes as input an array representing a deserialized graph and a reference to a graph.
  The function transforms the serialized representation of the graph to a graph_t representation via pointers on which all library operations are defined.
- void graph_save_to_file (graph_t *G, char *filename)

  This function takes as input a graph and a string. It serializes the input graph and stores it on a binary file.
- graph_t * graph_load_from_file (char *filename)

  This function takes a string as input. It extracts a serialized graph from a file and returns a graph in the graph_t format.
- void graph_merge_vertices (graph_t *G, int dest, array_int *src)

  This function takes as input graph, a vertex identifier 'dest' and an array of vertices. It merges all vertices in the array into the vertex 'dest'.
- void graph_merge (graph_t *to, graph_t *from, double p)

  *Merge 2 graph and the merged graph is in graph_t * to.*
- graph_t * graph_random (int max_n_node, int mean_edges, double variance_edges)

  create a random graph with max_n_node node and each node have mean number of edge with a variance_edge
- graph_t * graph_fully_connected_disconnected (int max_n_node, int isFullyConnected)

  create a graph fully connected or fully disconnected with max_n_node
- graph_t * graph_copy (graph_t *from)

  geneate a copy of a graph
- void graph_print_debug (graph_t *G)

  print on standard output graph in input: number of edged, for earch line node -> adjacency list and node-> inverted adjacency list
- scc_set_t * scc_set_init ()

  Initialize a new scc_set.
- void scc_set_free (scc_set_t *S)

  Destroy an scc_set.
- void scc_set_add (scc_set_t *S, int scc_id, array_int *nodes)

  Add a new SCC to the set handling merges if needed.
- void scc_set_print_debug (scc_set_t *S)

  print an scc_set
- void scc_set_merge (scc_set_t *dest, scc_set_t *src)

  Merge src scc_set into dest.

- bool scc_set_contains (scc_set_t *b, scc_set_t *a)

  Check if scc set b contains all of scc set a's content.
- array_int * **scc_set_serialize** (scc_set_t *S)
- void **scc_set_deserialize** (scc_set_t *S, array_int *buff)
- void scc_set_save_to_file (scc_set_t *S, char *filename)

  Write scc set to file.
- scc_set_t * scc_set_load_from_file (char *filename)

  Load scc set from file.

4.2.1 Detailed Description

This file implements the abstract data type graph with adjacency maps. It also contains basic operation on the graph (vertices and edges insert and deletion) as well as specific operations needed by the * * MPI and CUDA algorithms. The file also defines the scc_set data structure which is used to store SCCs found by Tarjan's algorithm as well as operations on this data structure.

4.2.2 Function Documentation

4.2.2.1 graph_copy()

```
graph_t* graph_copy (
            graph_t * from )
```

geneate a copy of a graph

Parameters

from	graph to be copied

Returns

graph_t* graph generated

4.2.2.2 graph_delete_edge()

```
void graph_delete_edge (
            graph_t * G,
            int u,
            int v )
```

This function takes as input a graph_t data structure and two vertices u and v. It deletes the edge from the node u to the node v if it exists. Otherwise does nothing.

Parameters

G	graph data structure.
u	vertex of the graph.
v	vertex of the graph.

4.2.2.3 graph_delete_vertex()

```
void graph_delete_vertex (
            graph_t * G,
            int v )
```

This function takes as input a graph_t data structure and a vertex v. It deletes the vertex v and every edge incident on the vertex.

Parameters

G	input graph
v	vertex to be deleted

4.2.2.4 graph_deserialize()

```
void graph_deserialize (
            graph_t * G,
            array_int * buff )
```

This function takes as input an array representing a deserialized graph and a reference to a graph.
The function transforms the serialized representation of the graph to a graph_t representation via pointers on which all library operations are defined.

Parameters

G	graph data structure.
buff	array representing a deserialized graph.

4.2.2.5 graph_free()

```
void graph_free (
            graph_t * G )
```

This function takes as input a graph_t data structure and takes care of deallocating the entire memory occupied by the data structure.

Parameters

G	graph data structure to be deallocated.

4.2.2.6 graph_fully_connected_disconnected()

```
graph_t* graph_fully_connected_disconnected (
            int max_n_node,
            int isFullyConnected )
```

create a graph fully connected or fully disconnected with max_n_node

Parameters

max_n_node	number of node of graph
isFullyConnected	0 create a fully disconneted graph 1 create a fully connected graph

Returns

graph_t∗ graph generated

4.2.2.7 graph_get_num_vertex()

```
int graph_get_num_vertex (
            graph_t * G )
```

This function takes as input a graph_t data structure and returns the number of vertices in the graph.

Parameters

G	graph data structure of which we want to know the number of vertices.

Returns

int.

4.2.2.8 graph_init()

```
graph_t* graph_init ( )
```

This function declares and initializes an empty graph_t data structure.

Returns

graph_t*

4.2.2.9 graph_insert_edge()

```
void graph_insert_edge (
            graph_t * G,
            int u,
            int v )
```

This function takes as input a graph_t data structure and two vertices u and v. It takes care of inserting an edge from vertex u to vertex v of the graph.
Throws an error is the edge already exists.

Parameters

G	graph data structure.
u	vertex of the graph.
v	vertex of the graph.

4.2.2.10 graph_insert_vertex()

```
void graph_insert_vertex (
            graph_t * G,
            int v )
```

This function takes as input a graph_t data structure and a vertex v; it takes care of inserting the vertex v within the graph.

Parameters

G	graph data structure.
v	vertex to be inserted.

4.2.2.11 graph_load_from_file()

```
graph_t* graph_load_from_file (
            char * filename )
```

This function takes a string as input. It extracts a serialized graph from a file and returns a graph in the graph_t format.

Parameters

filename	string representing the filename of the input file.

4.2.2.12 graph_merge()

```
void graph_merge (
            graph_t * to,
            graph_t * from,
            double p )
```

Merge 2 graph and the merged graph is in graph_t * to.

Parameters

to	graph with vertex index from 0 to graph_get_num_vertex(to)
from	graph with vertex index from 0 to graph_get_num_vertex(from)
p	probability of create an edge between a node of graph from and a node of graph to and viceversa

4.2.2.13 graph_merge_vertices()

```
void graph_merge_vertices (
            graph_t * G,
            int dest,
            array_int * src )
```

This function takes as input graph, a vertex identifier 'dest' and an array of vertices. It merges all vertices in the array into the vertex 'dest'.

Parameters

G	graph data structure.
dest	a vertex identifier.
src	an array of vertices.

4.2.2.14 graph_print_debug()

```
void graph_print_debug (
            graph_t * G )
```

print on standard output graph in input: number of edged, for earch line node -> adjacency list and node-> inverted adjacency list

Parameters

G	graph to be printed

4.2.2.15 graph_random()

```
graph_t* graph_random (
            int max_n_node,
            int mean_edges,
            double variance_edges )
```

create a random graph with max_n_node node and each node have mean number of edge with a variance_edge

Parameters

max_n_node	number of node of graph
mean_edges	mean edge for each node
variance_edges	variance of number of edge for each node

Returns

graph_t∗ graph generated

4.2.2.16 graph_save_to_file()

```
void graph_save_to_file (
            graph_t * G,
            char * filename )
```

This function takes as input a graph and a string. It serializes the input graph and stores it on a binary file.

Parameters

G	graph data structure.
filename	string representing the filename of the output file.

4.2.2.17 graph_serialize()

```
array_int* graph_serialize (
            graph_t * G,
```

```
            int n,
            khint_t * bucket )
```

This function takes as input a graph, the number of vertices to be serialized n, and a variable to store the reference to the adicence map.
The function returns an array of integers representing the serialization of the first n vertices of the graph.
In other words, we go from a representation using pointers to a representation that uses integers only.

Parameters

G	graph data structure.
n	number of vertices to be serialized.
bucket	variable to store the reference to the adicence map.

Returns

array_int* array of integers representing the serialization of the first n vertices.

4.2.2.18 graph_tarjan()

```
array_int* graph_tarjan (
            graph_t * G )
```

This function is a Tarjan's algorithm implementation using recursion.
This is a modified version of the algorithm on the geeksforgeeks.com website.
The main differences are:
- disc, low and stackMember are now hash tables because we remove the hypotesis that vertex ids go from 0 to N-1:
when working on a subgraph (as a slave process), there are no guarantees on the order nor continuity of the vertex ids.
Using hash tables instead of arrays we save a lot of memory.
- we remove the hypotesis that every vertex in an adjacency map exists in the graph. This is also caused by executions
on subgraphs of a given graph.

Parameters

G	graph data structure.

Returns

array_int* array containing all the found SCCs.

4.2.2.19 graph_tarjan_foreach()

```
void graph_tarjan_foreach (
            graph_t * G,
            void(*)(array_int *) f )
```

This function takes as input a graph and a callback function f. It finds all the SCCs in the graph and each time it finds one it calls the callback function f on the SCC.

Parameters

G	graph data structure.
f	callback function.

4.2.2.20 graph_tarjan_foreach_helper()

```
void graph_tarjan_foreach_helper (
            graph_t * G,
            int node,
            khash_t(m32) * disc,
            khash_t(m32) * low,
            linkedlist_int * stack,
            khash_t(m32) * stackMember,
            int * time,
            array_int * scc,
            void(*)(array_int *) f )
```

This function is an helper function for graph_tarjan_foreach(). For more info see documentation for graph_tarjan_foreach().

Parameters

G	graph data structure.
node	integer which represents the node.
stack	stack data structure.
time	pointer to an integer.
scc	array containing the SCCs.
f	callback function.

4.2.2.21 graph_tarjan_helper()

```
void graph_tarjan_helper (
            graph_t * G,
            int node,
            khash_t(m32) * disc,
            khash_t(m32) * low,
            linkedlist_int * stack,
            khash_t(m32) * stackMember,
            int * time,
            array_int * result )
```

This function is an helper function for graph_tarjan(). For more info see documentation for graph_tarjan().

Parameters

G	graph data structure.
node	pointer to an integer.
stack	stack data structure.
time	pointer to an integer.
result	array data structure.

4.2.2.22 min()

```
int min (
            int a,
            int b )
```

This function takes two integers as input and returns the minimum integer.

Parameters

a	first integer.
b	second integer.

Returns

minimum.

4.2.2.23 scc_set_add()

```
void scc_set_add (
            scc_set_t * S,
            int scc_id,
            array_int * nodes )
```

Add a new SCC to the set handling merges if needed.

Parameters

S	the reference to the scc_set.
scc↩ _id	the id of the SCC to be added. By convention, it is the lowest among the ids of the nodes in the SCC.
nodes	the nodes of the SCC.

4.2.2.24 scc_set_contains()

```
bool scc_set_contains (
            scc_set_t * b,
            scc_set_t * a )
```

Check if scc set b contains all of scc set a's content.

Parameters

b	the first scc_set.
a	the second scc_set.

Returns

true scc set b contains all of scc set a's content.

false scc set b not contains all of scc set a's content.

4.2.2.25 scc_set_free()

```
void scc_set_free (
            scc_set_t * S )
```

Destroy an scc_set.

Parameters

S	The scc_set to be destroyed.

4.2.2.26 scc_set_init()

```
scc_set_t* scc_set_init ( )
```

Initialize a new scc_set.

Returns

scc_set_t* The scc_set

4.2.2.27 scc_set_load_from_file()

```
scc_set_t* scc_set_load_from_file (
            char * filename )
```

Load scc set from file.

Parameters

filename	the file to load the set from.

Returns

scc_set_t* ssc set readed

4.2.2.28 scc_set_merge()

```
void scc_set_merge (
            scc_set_t * dest,
            scc_set_t * src )
```

Merge src scc_set into dest.

Parameters

dest	the reference of the destination scc_set.
src	the reference of the source scc_set.

4.2.2.29 scc_set_print_debug()

```
void scc_set_print_debug (
            scc_set_t * S )
```

print an scc_set

Parameters

S	ssc_set to be printed

4.2.2.30 scc_set_save_to_file()

```
void scc_set_save_to_file (
            scc_set_t * S,
            char * filename )
```

Write scc set to file.

Parameters

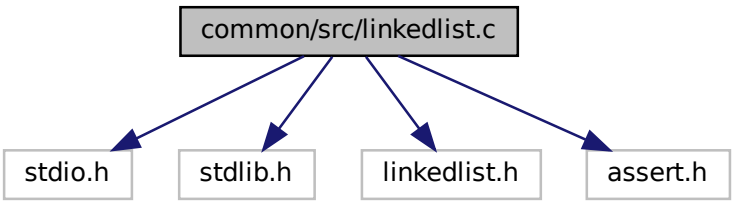
S	the scc_set to be saved.
filename	the file to be saved to.

4.3 common/src/linkedlist.c File Reference

This file implements the abstract data type linkedlist.

```
#include <stdio.h>
#include <stdlib.h>
#include "linkedlist.h"
#include <assert.h>
```

Include dependency graph for linkedlist.c:



Classes

- struct lnode_int_t

  This struct represents the node of the list data structure.
- struct linkedlist_int

  This struct represents the linkedlist data structure.

Typedefs

- typedef struct linkedlist_ptr **linkedlist_ptr**

Functions

- lnode_int_t * nodeCreate (int info)

  This function takes an integer as input and creates a new lnode_int_t node.
- void nodeDestroy (lnode_int_t *n)

  This function takes a lnode_int_t node as input and deallocates it.
- linkedlist_int * linkedlist_int_init ()

  This function generates and initializes a linkedlist_int data structure.

- void linkedlist_int_free (linkedlist_int *a)

  This function takes as input a linkedlist_int data structure and deallocates the occupied memory.
- int linkedlist_int_length (linkedlist_int *a)

  This function takes as input a linkedlist_int data structure and returns the number of nodes contained in it.
- void linkedlist_int_push (linkedlist_int *a, int elem)

  This function takes as input a linkedlist_int data structure and an integer. It inserts the element passed as input at the last position in the data structure.
  The linkedlist_int can work like a Stack data structure if the linkedlist_int_pop() and linkedlist_int_push() functions are used.
- int linkedlist_int_pop (linkedlist_int *a)

  This function takes as input a linkedlist_int data structure. It removes the element at the last position in the data structure.
  The linkedlist_int can work like a Stack data structure if the linkedlist_int_pop() and linkedlist_int_push() functions are used.
- void linkedlist_int_insert (linkedlist_int *a, int elem)

  This function takes as input a linkedlist_int data structure and an integer to insert. It inserts the element passed as input into the data structure.
  The linkedlist_int works as an ordered list data structure if the insert and remove functions are used.
- void linkedlist_int_delete (linkedlist_int *a, int elem)

  This function takes as input a linkedlist_int data structure and an integer. It removes the element passed as input from the data structure.
  The linkedlist_int works like a ordered list data structure if the insert and remove functions are used.
- void linkedlist_int_enqueue (linkedlist_int *a, int elem)

  This function takes as input a linkedlist_int data structure and an integer to insert. It inserts the element passed as input into the data structure.
  The linkedlist_int works like a queue data structure if the linkedlist_int_enqueue() and linkedlist_int_dequeue() functions are used.
- int linkedlist_int_dequeue (linkedlist_int *a)

  This function takes as input a linkedlist_int data structure. It removes the element at the top of the data structure.
  The linkedlist_int works like a queue data structure if the linkedlist_int_enqueue() and linkedlist_int_dequeue() functions are used.
- void linkedlist_int_print (linkedlist_int *a)

  This function prints all nodes in the linkedlist_int.
- int linkedlist_int_top (linkedlist_int *a)

  This function returns the element at the top of the linkedlist_int data structure.

4.3.1 Detailed Description

This file implements the abstract data type linkedlist.

4.3.2 Function Documentation

4.3.2.1 linkedlist_int_delete()

```
void linkedlist_int_delete (
            linkedlist_int * a,
            int elem )
```

This function takes as input a linkedlist_int data structure and an integer. It removes the element passed as input from the data structure.
The linkedlist_int works like a ordered list data structure if the insert and remove functions are used.

Parameters

a	is linkedlist_int data structure.
elem	is an integer which represents the elem to be deleted.

4.3.2.2 linkedlist_int_dequeue()

```
int linkedlist_int_dequeue (
            linkedlist_int * a )
```

This function takes as input a linkedlist_int data structure. It removes the element at the top of the data structure.
The linkedlist_int works like a queue data structure if the linkedlist_int_enqueue() and linkedlist_int_dequeue() functions are used.

Parameters

a	is linkedlist_int data structure.

Returns

int

4.3.2.3 linkedlist_int_enqueue()

```
void linkedlist_int_enqueue (
            linkedlist_int * a,
            int elem )
```

This function takes as input a linkedlist_int data structure and an integer to insert. It inserts the element passed as input into the data structure.
The linkedlist_int works like a queue data structure if the linkedlist_int_enqueue() and linkedlist_int_dequeue() functions are used.

Parameters

a	is linkedlist_int data structure.
elem	is an integer which represents the elem to be inserted.

4.3.2.4 linkedlist_int_free()

```
void linkedlist_int_free (
            linkedlist_int * a )
```

This function takes as input a linkedlist_int data structure and deallocates the occupied memory.

Parameters

a	is linkedlist_int data structure.

4.3.2.5 linkedlist_int_init()

```
linkedlist_int* linkedlist_int_init ( )
```

This function generates and initializes a linkedlist_int data structure.

Returns

linkedlist_int*

4.3.2.6 linkedlist_int_insert()

```
void linkedlist_int_insert (
            linkedlist_int * a,
            int elem )
```

This function takes as input a linkedlist_int data structure and an integer to insert. It inserts the element passed as input into the data structure.
The linkedlist_int works as an ordered list data structure if the insert and remove functions are used.

Parameters

a	is linkedlist_int data structure.
elem	is an integer which represents the elem to be inserted.

4.3.2.7 linkedlist_int_length()

```
int linkedlist_int_length (
            linkedlist_int * a )
```

This function takes as input a linkedlist_int data structure and returns the number of nodes contained in it.

Parameters

a	is linkedlist_int data structure.

Returns

int

4.3.2.8 linkedlist_int_pop()

```
int linkedlist_int_pop (
            linkedlist_int * a )
```

This function takes as input a linkedlist_int data structure. It removes the element at the last position in the data structure.
The linkedlist_int can work like a Stack data structure if the linkedlist_int_pop() and linkedlist_int_push() functions are used.

Parameters

a	is linkedlist_int data structure.

Returns

int

4.3.2.9 linkedlist_int_print()

```
void linkedlist_int_print (
            linkedlist_int * a )
```

This function prints all nodes in the linkedlist_int.

Parameters

a	is linkedlist_int data structure.

4.3.2.10 linkedlist_int_push()

```
void linkedlist_int_push (
            linkedlist_int * a,
            int elem )
```

This function takes as input a linkedlist_int data structure and an integer. It inserts the element passed as input at the last position in the data structure.
The linkedlist_int can work like a Stack data structure if the linkedlist_int_pop() and linkedlist_int_push() functions are used.

Parameters

a	is linkedlist_int data structure.
elem	is an integer which represents the elem to be inserted.

4.3.2.11 linkedlist_int_top()

```
int linkedlist_int_top (
            linkedlist_int * a )
```

This function returns the element at the top of the linkedlist_int data structure.

Parameters

a	is linkedlist_int data structure.

Returns

int

4.3.2.12 nodeCreate()

```
lnode_int_t* nodeCreate (
            int info )
```

This function takes an integer as input and creates a new lnode_int_t node.

Parameters

info	an integer representing the information stored by the node.

Returns

lnode_int_t*.

4.3.2.13 nodeDestroy()

```
void nodeDestroy (
            lnode_int_t * n )
```

This function takes a lnode_int_t node as input and deallocates it.

Parameters

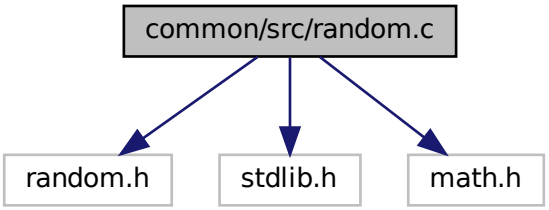
n	pointer to lnode_int_t node.

4.4 common/src/random.c File Reference

This file contains the implementations for utility functions that are useful for randomly generating graphs.

```
#include "random.h"
#include <stdlib.h>
#include <math.h>
```

Include dependency graph for random.c:



Functions

- int rand_bernoulli (double p)

  Extract a random variable distributed with Bernoulli distribution. The variable takes the value 1 with probability p and 0 with probability 1-p.
- int rand_binomial (long n, double p)

  Extract a random variable distributed with Binomial distribution. A binomial variable is generated by exectuing n Bernoulli experiments with a fixed probability p and counting the numbersuccesses.
- int rand_binomial_2 (int mean, double variance)

  Extract a random variable distributed with Binomial distribution with a given mean and variance.

4.4.1 Detailed Description

This file contains the implementations for utility functions that are useful for randomly generating graphs.

4.4.2 Function Documentation

4.4.2.1 rand_bernoulli()

```
int rand_bernoulli (
            double p )
```

Extract a random variable distributed with Bernoulli distribution. The variable takes the value 1 with probability p and 0 with probability 1-p.

Parameters

p	probability of the variable assuming the value 1

Returns

the random variable

4.4.2.2 rand_binomial()

```
int rand_binomial (
            long n,
            double p )
```

Extract a random variable distributed with Binomial distribution. A binomial variable is generated by exectuing n Bernoulli experiments with a fixed probability p and counting the numbersuccesses.

Parameters

n	number of experiments
p	probability of the variable assuming the value 1

Returns

the random variable

4.4.2.3 rand_binomial_2()

```
int rand_binomial_2 (
            int mean,
            double variance )
```

Extract a random variable distributed with Binomial distribution with a given mean and variance.

Parameters

mean	the mean of the Binomial distibution
variance	the variance of the Binomial distibution

Returns

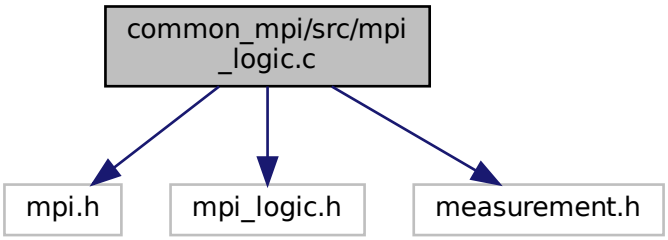
the random variable

4.5 common_mpi/src/mpi_logic.c File Reference

This file implements a version of a parallelization of Tarjan's algorithm.

```
#include <mpi.h>
#include "mpi_logic.h"
#include "measurement.h"
```

Include dependency graph for mpi_logic.c:



Functions

- void callback (array_int *scc)

  This is a callback function. It is called every time the Tarjan'algorithm, run by a slave process on a portion of the graph, finds an scc.
  The slave process sends the found scc to the master node along with its size.

- void master_schedule (graph_t *graph, int N, int n_slaves, scc_set_t *SCCs)
- void master_work (int rank, int size, char *filename, char *outputfilename)

  The master node calls this function.
  The function takes as input the name of a binary file that contains a graph represented by an adjacency map.
  The master_work function takes care of extracting the contents of the binary file and converting it into a graph, then calls the master_work2() function to execute the MPI algorithm.
- void master_work2 (int rank, int size, graph_t *graph, scc_set_t *SCCs, char *outputfilename, double time_init)

  This function is called by the master node.
  The function takes as input the graph and an empty set, which will be filled with the scc found by Tarjan's algorithm.
  The function divides the graph into chunks of fixed size and will delegate the work to be done on the chunks to the master_schedule() function.
  In addition, the function is responsible of sending the termination message to all slave processes. This happens when all the work is done.
- void slave_work (int rank)

  This function is called by the slave nodes. The function receives messages containing the portion of the graph on which the slave node must find the scc through Tarjan's algorithm.
  The function ends when a master node sends a special termination message. The termination message is a message specifying that the size of the next message is 0.

Variables

- double **time_split_graph** = 0.0
- double **time_merge_graph** = 0.0

4.5.1 Detailed Description

This file implements a version of a parallelization of Tarjan's algorithm.

4.5.2 Function Documentation

4.5.2.1 callback()

```
void callback (
            array_int * scc )
```

This is a callback function. It is called every time the Tarjan'algorithm, run by a slave process on a portion of the graph, finds an scc.
The slave process sends the found scc to the master node along with its size.

Parameters

scc	It is the scc discovered from the Tarjan's algorithm.

4.5.2.2 master_schedule()

```
void master_schedule (
            graph_t * graph,
            int N,
            int n_slaves,
            scc_set_t * SCCs )
```

The function takes as input the graph, the size of the chunk, the number of slave processes and the data structure where to save all the scc found.
The function sends a chunk of the graph to each slave node. Then, it waits for the slave nodes to find the scc by applying Tarjan's algorithm on their chunk of the graph. As soon as a slave node finishes execution, the master node assigns it another chunk of the graph.
The iterations terminate as soon as the whole graph for the fixed chunk size has been completed by the slave nodes.

Parameters

graph	graph on which compute all the scc.
N	represents the chunk size.
n_slaves	number of slave precesses.
SCCs	data structure where to save all the scc found.

4.5.2.3 master_work()

```
void master_work (
            int rank,
            int size,
            char * filename,
            char * outputfilename )
```

The master node calls this function.
The function takes as input the name of a binary file that contains a graph represented by an adjacency map.
The master_work function takes care of extracting the contents of the binary file and converting it into a graph, then calls the master_work2() function to execute the MPI algorithm.

Parameters

rank	id of the process within the communicator.
size	size of the communicator.
filename	name of the file that contains a graph represented by an adjacency map.
outputfilename	name of the output binary file that will contain all the scc found.

4.5.2.4 master_work2()

```
void master_work2 (
            int rank,
            int size,
            graph_t * graph,
            scc_set_t * SCCs,
            char * outputfilename,
            double time_init )
```

This function is called by the master node.
The function takes as input the graph and an empty set, which will be filled with the scc found by Tarjan's algorithm.
The function divides the graph into chunks of fixed size and will delegate the work to be done on the chunks to the master_schedule() function.
In addition, the function is responsible of sending the termination message to all slave processes. This happens when all the work is done.

Parameters

rank	id of the process within the communicator.
size	size of the communicator.
graph	graph that will be computed in order to find its sccs.
SCCs	empty set which will be filled with the scc found by Tarjan's algorithm.
outputfilename	name of the output binary file that will contain all the scc found.
time_init	initialization time.

4.5.2.5 slave_work()

```
void slave_work (
            int rank )
```

This function is called by the slave nodes. The function receives messages containing the portion of the graph on which the slave node must find the scc through Tarjan's algorithm.
The function ends when a master node sends a special termination message. The termination message is a message specifying that the size of the next message is 0.

Parameters

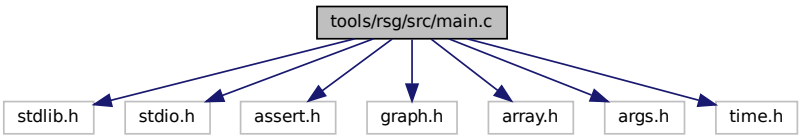
rank	id of the process within the communicator.

4.6 tools/rsg/src/main.c File Reference

This tool generate a random graph with max_n_node node and each node have mean number of edge with a variance_edge, the number of nodes follows the Bernoulli distribution.

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "graph.h"
#include "array.h"
#include "args.h"
#include <time.h>
```

Include dependency graph for main.c:



Functions

- int **main** (int argc, char ∗argv[ ])

4.6.1 Detailed Description

This tool generate a random graph with max_n_node node and each node have mean number of edge with a variance_edge, the number of nodes follows the Bernoulli distribution.

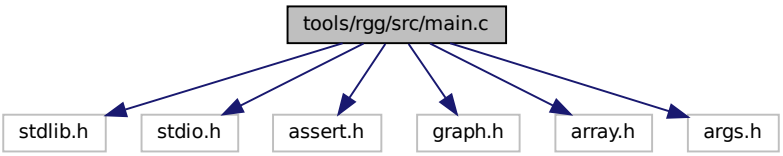
The first parameter is the path of graph generated.
The second parameter is a integer that indicate the number of node of graph.
The thrid parameter is an integer that indicate the the mean of edge for each node.
The fourth parameter is the variance of number of edge for each node.

4.7 tools/rgg/src/main.c File Reference

This tool generates a graph starting from a seed.

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "graph.h"
#include "array.h"
#include "args.h"
```

Include dependency graph for main.c:



Functions

- int **main** (int argc, char ∗argv[ ])

4.7.1 Detailed Description

This tool generates a graph starting from a seed.

This tool uses an interger n to generate a graph with 2^n replicas of the seed keeping all the edges already present in the seed, in addition edges are added between the different seeds of the final graph following the probability passed.
The first parameter is the path of seed graph.
The second parameter is the path of graph generated.
The thrid parameter is an integer n that indicate the 2^n copy to be created.
The fourth parameter is the probability of create an edge between a node of a copy and another and viceversa.
Es: seed 10 edge and second parameter 1 graph generated 20 edge.

Es: seed 10 edge and second parameter 2 graph generated 40 edge.

Es: seed 10 edge and second parameter 3 graph generated 80 edge.

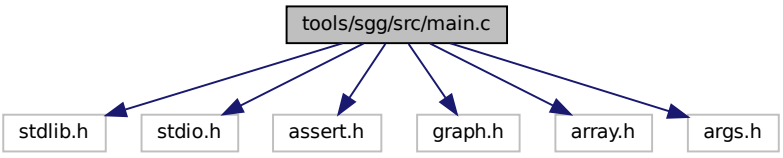
Es: seed 10 edge and second parameter 4 graph generated 160 edge.

4.8 tools/sgg/src/main.c File Reference

This tool generate graph fully connected, graph fully disconnected or graph bipartite.

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "graph.h"
#include "array.h"
#include "args.h"
```

Include dependency graph for main.c:



Functions

- void **generate_bipartite_graph** (long n, char ∗filename)
- int **main** (int argc, char ∗argv[ ])

4.8.1 Detailed Description

This tool generate graph fully connected, graph fully disconnected or graph bipartite.

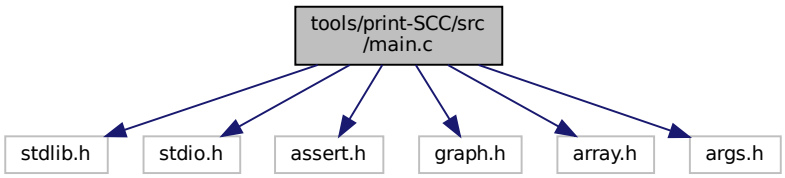
The first parameter is the path of graph generated.
The second parameter is a integer that indicate the number of node of graph.
The thrid parameter is an integer that indicate 0 for graph fully disconnected, 1 for graph fully connected, 2 for grapg bipartite.

4.9 tools/print-SCC/src/main.c File Reference

This tool print to standard output the SCC discovered from a file.

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "graph.h"
#include "array.h"
```

```
#include "args.h"
```
Include dependency graph for main.c:



Functions

- int **main** (int argc, char ∗argv[ ])

4.9.1 Detailed Description

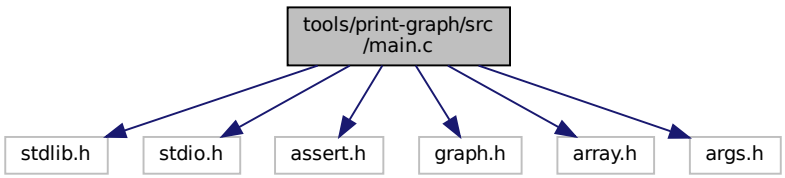
This tool print to standard output the SCC discovered from a file.

The first parameter is the path of the SCC file to be printed.

4.10 tools/print-graph/src/main.c File Reference

This tool print to standard output the graph from a file in input.

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "graph.h"
#include "array.h"
#include "args.h"
```
Include dependency graph for main.c:



Functions

- int **main** (int argc, char ∗argv[ ])

4.10.1 Detailed Description

This tool print to standard output the graph from a file in input.

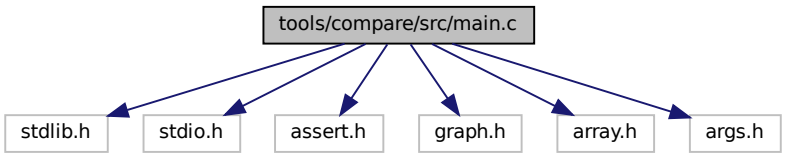
The first parameter is the path of the graph file to be printed.

4.11 tools/compare/src/main.c File Reference

This tool compare two different SCC discovered file.

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "graph.h"
#include "array.h"
#include "args.h"
```

Include dependency graph for main.c:



Functions

- int **main** (int argc, char ∗argv[ ])

4.11.1 Detailed Description

This tool compare two different SCC discovered file.

Checks whether all SSCs in the first file are contained in the second file and vice versa. This is useful to verify the correctness of the parallel algortims by comparing the SSCs found with those found by sequential Tarjan.
The first parameter is the path of the first SCC file to be compered.
The second parameter is the path of the second SCC file to be compered.

Index

callback
    mpi_logic.c, 32
common/src/cuda_graph.c, 7
common/src/graph.c, 8
common/src/linkedlist.c, 22
common/src/random.c, 29
common_mpi/src/mpi_logic.c, 31
cuda_graph.c
    cuda_graph_load_from_file, 8
    cuda_graph_to_graph, 8
cuda_graph_load_from_file
    cuda_graph.c, 8
cuda_graph_to_graph
    cuda_graph.c, 8

graph.c
    graph_copy, 11
    graph_delete_edge, 11
    graph_delete_vertex, 12
    graph_deserialize, 12
    graph_free, 12
    graph_fully_connected_disconnected, 13
    graph_get_num_vertex, 13
    graph_init, 13
    graph_insert_edge, 14
    graph_insert_vertex, 14
    graph_load_from_file, 14
    graph_merge, 15
    graph_merge_vertices, 15
    graph_print_debug, 15
    graph_random, 16
    graph_save_to_file, 16
    graph_serialize, 16
    graph_tarjan, 17
    graph_tarjan_foreach, 17
    graph_tarjan_foreach_helper, 18
    graph_tarjan_helper, 18
    min, 19
    scc_set_add, 19
    scc_set_contains, 19
    scc_set_free, 20
    scc_set_init, 20
    scc_set_load_from_file, 20
    scc_set_merge, 21
    scc_set_print_debug, 21
    scc_set_save_to_file, 21
graph_copy
    graph.c, 11
graph_delete_edge
    graph.c, 11
graph_delete_vertex
    graph.c, 12
graph_deserialize
    graph.c, 12
graph_free
    graph.c, 12
graph_fully_connected_disconnected
    graph.c, 13
graph_get_num_vertex
    graph.c, 13
graph_init
    graph.c, 13
graph_insert_edge
    graph.c, 14
graph_insert_vertex
    graph.c, 14
graph_load_from_file
    graph.c, 14
graph_merge
    graph.c, 15
graph_merge_vertices
    graph.c, 15
graph_print_debug
    graph.c, 15
graph_random
    graph.c, 16
graph_save_to_file
    graph.c, 16
graph_serialize
    graph.c, 16
graph_tarjan
    graph.c, 17
graph_tarjan_foreach
    graph.c, 17
graph_tarjan_foreach_helper
    graph.c, 18
graph_tarjan_helper
    graph.c, 18

linkedlist.c
    linkedlist_int_delete, 23
    linkedlist_int_dequeue, 24
    linkedlist_int_enqueue, 24
    linkedlist_int_free, 24
    linkedlist_int_init, 26
    linkedlist_int_insert, 26
    linkedlist_int_length, 26
    linkedlist_int_pop, 27
    linkedlist_int_print, 27
    linkedlist_int_push, 27

linkedlist_int_top, 28
nodeCreate, 28
nodeDestroy, 28
linkedlist_int, 5
linkedlist_int_delete
linkedlist.c, 23
linkedlist_int_dequeue
linkedlist.c, 24
linkedlist_int_enqueue
linkedlist.c, 24
linkedlist_int_free
linkedlist.c, 24
linkedlist_int_init
linkedlist.c, 26
linkedlist_int_insert
linkedlist.c, 26
linkedlist_int_length
linkedlist.c, 26
linkedlist_int_pop
linkedlist.c, 27
linkedlist_int_print
linkedlist.c, 27
linkedlist_int_push
linkedlist.c, 27
linkedlist_int_top
linkedlist.c, 28
lnode_int_t, 6

master_schedule
mpi_logic.c, 32
master_work
mpi_logic.c, 33
master_work2
mpi_logic.c, 33
min
graph.c, 19
mpi_logic.c
callback, 32
master_schedule, 32
master_work, 33
master_work2, 33
slave_work, 33

nodeCreate
linkedlist.c, 28
nodeDestroy
linkedlist.c, 28

rand_bernoulli
random.c, 29
rand_binomial
random.c, 30
rand_binomial_2
random.c, 30
random.c
rand_bernoulli, 29
rand_binomial, 30
rand_binomial_2, 30

scc_set_add
graph.c, 19
scc_set_contains
graph.c, 19
scc_set_free
graph.c, 20
scc_set_init
graph.c, 20
scc_set_load_from_file
graph.c, 20
scc_set_merge
graph.c, 21
scc_set_print_debug
graph.c, 21
scc_set_save_to_file
graph.c, 21
scc_set_t, 6
slave_work
mpi_logic.c, 33

tools/compare/src/main.c, 38
tools/print-graph/src/main.c, 37
tools/print-SCC/src/main.c, 36
tools/rgg/src/main.c, 35
tools/rsg/src/main.c, 34
tools/sgg/src/main.c, 36