afgraph

A Framework for Graph Algorithms

Aiguo Fei

Public Release 1.0

May 2012

Contents

1 afgraph: A Framework for Graph Algorithms

1.1 Introduction

My Ph.D. research (1997-2001) was mostly on graph algorithm, in some sense. Most of my work was to come out with some algorithms to solve some particular problems and wrote some code to simulate the algorithms (in a particular computer network setting) and compare them with some other algorithms on certain metrics. Most of the code that I wrote were specific to certain problems and it is mostly useless to others; the problems that I worked on were no longer relevant anyway. However, a small core portion of the code is generic in nature -- it served as the basis for me to write implementations of various graph-related algorithms. A few months after I graduated, I packaged that portion of code and gave it to a friend of mine upon his request. I have never asked him if he found the code useful or not. However, it is always my belief that it can be useful to certain people who happen to work in this domain. So I decided to release this code to the public, a good 11-years after my graduation. Better late than never.

The code has been basically the same as 11 years ago, except some small updates that have been made to make it compile with newer versions of compilers -- C++ the language and compilers have evolved quite a bit over the time. As of now (March 2012), it compiles with the following: g++ 4.6.1 on Linux, g++ 4.3.4 under Cygwin on Windows, Microsoft Visual C++ 2010 (Express Edition). Because of the age of the code, it may look a little old-fashioned or out of date, or may not. Either way, I hope that some people will find it useful.

1.1.1 What's Inside

Under the top directory, you will find the following:

```
docs/: documentation source
docs-gen/: generated documentations, in pdf, chm, qch format
    html/: documentations in html
include/: header files
    afl/: some template utilties
    afgraph/: the main stuff
    news/: a couple of files for an unfinished network simulator
    research/: code for research purpose (e.g., code to import "asconnect" file,
        may be useful to others)
net_topology/: data files of actual network topology
    mbone/: (very old) mbone topology
test/: code for sanity test
    afgraph/
    news/
    research/
src/: .cpp file, just one
    news/
examples/: examples, juse one for now.
```

1.1.2 Documentations

Code has been documented extensively with doxygen tags. Documentations in pdf, chm, qch and html format are generated from the source and included in this release. You may run doxygen to re-generate them if you add/modify source code documentations: go to docs/, run "doxygen afgraphdoc_doxygen-chm.conf" and "doxygen afgraphdoc_-doxygen-html.conf".

You need the following tools to produce documentations other than html: (1) pdflatex to generate the pdf version (go to docs-gen/latex, run "pdflatex" twice); (2) Microsoft HTML Help Workshop to produce chm file (open docs-gen/chm/index.hhp in HTML Help Workshop); (3) qhelpgenerator from Qt SDK to create qch file (edit docs/afgraphdoc_-doxygen-chm.conf to point to where qhelpgenerator is installed), which can be opened with Qt assistant.

1.2 Get Started

1.2.1 An Example

Under examples/, you will find one example: e_graph.cpp. This file demonstrates how to create a graph and run a couple of algorithms included in this package.

```
// mem_fun should give us this; but couldn't get it pass the compiler.
// Too much templating, I guess.
char get_vertex_d( const CiVertex<char, int> &v )
{
    return v.vertex_d();
}

int main()
{
    // Create a graph with char vertext type (i.e., each vertex is identified
    // by a char) and integer edge type (i.e., each edge has an integer value).
    // It has 10 vertices and 'z' will not be used as a vertex id.
    // An unused vertex id is needed to populate empty spots (if any) in the
    // vertex array.
    CGraph<char, int> gra( 6, 'z' );

    // insert all the nodes; their IDs are ['a'..'f'].
    for( int i=0; i<6; ++i ){
        gra.insert_v( i+97 ); // 97 is ASCII 'a'
    }

    // Now insert all the edges. It is a non-directed graph, each edge is
    // represented by 2 directed edges.
    gra.insert_2e( 'a', 'b', 3 ); gra.insert_2e( 'a', 'd', 4 );
    gra.insert_2e( 'b', 'c', 2 ); gra.insert_2e( 'b', 'd', 3 );
    gra.insert_2e( 'c', 'f', 6 ); gra.insert_2e( 'd', 'e', 2 );
    gra.insert_2e( 'e', 'f', 3 );

    // print out the graph; note the difference between vertex ID and vertex index.
```

```
    std::cout << "graph now: " << std::endl << gra << std::endl;

    // let's check if this graph is connected.
    std::cout << "connected?: " << (is_connected( gra )?"yes":"no") << std::endl;

    // All graph algorithms operate on vertex indices; let's find the index
    // for node 'c' and 'e'.
    int idxC = gra.find_index( 'c' );
    int idxE = gra.find_index( 'e' );
    cout << "node 'c' is at "<< idxC << ", and node 'e' is at " << idxE << endl;

    /* Run dijkstra algorithm to find the shorted path from 'c' to 'e', using
    each edge's integer value as the weight -- so we pass functor
    afl::pointer2value<int>() for that purpose. The edge weight function passed
    to dijkstra_t() will be called with an pointer to an edge.
    */
    CPath lp;
    cout << "shortest path by edge weight from node 'c' to 'e' result: "
         << dijkstra_t<CGraph<char, int> >(
             gra, idxC, idxE, afl::pointer2value<int>(), 1000, lp )
         << endl << "path found (by indices): " << lp << endl;
    cout << "path found (by ID): " ;
    lp.output( cout, gra, get_vertex_d ) << endl;
    //lp.output( cout, gra, mem_fun( &ciVertex_t::vertex_d ) ) << endl;

    /* Run dijkstra algorithm to find the shorted path from 'c' to 'e', each
    edge having weight 1 -- equivalent to finding minimum hop-count path.
    */
    CPath lp2;
    cout << "shortest path by hop-count from node 'c' to 'e' result: "
         << dijkstra_t<CGraph<char, int> >(
             gra, idxC, idxE, afl::runit_p<int>(), 1000, lp2 )
         << endl << "path found (by indices): " << lp2 << endl;
    cout << "path found (by ID): " ;
    lp2.output( cout, gra, get_vertex_d ) << endl;

    return 0;
}
```

Under test/afgraph, there are some test code that I used for sanity test -- they can serve as examples of how to use the framework. To learn how to use the framework to implement your own graph algorithms, you may check out the following files under include/afgraph as examples: shortest_path.h, graph_alg.h, kthsp.h, tree_alg.h.

1.2.2 Compile and Run

For the example and test programs, a build tool called Scons is used -- instead of Make-file, the build script file is named Scontruct. Compared with conventional make, Scons has some advantages; for small profject, it is fairly easy and straight-forward. If you don't have Scons installed already on your system, you need to install it.

On a terminal under Linux/BSD/∗nix/Windows, go to examples/, type the command

"scons" or "scons.bat", it will build the example program with the output under examples/output_opt/; type the command "./output_opt/e_graph" to run it ("./output_opt/e_graph.exe" under Windows). To build a debug version of the program with debug symbols, do "scons debug=1" and the output will be under output_dbg/.

Go to test/afgraph, do "scons" or "scons debug=1" to build all the test programs.

Note 1: some test programs use some data files which are located in the same directory as the source file; e.g., under test/afgraph/, t_graph_input.cpp uses gra1.vve and a few others. You need to run the compiled program from the directory where the data files are located so the test program can find them; e.g., under test/afgraph/, you run "./output_opt/t_graph_input".

Note 2: "-Wall" is enabled in Sconstruct files. With gcc, all code compiles without any warning. However, with MS Visual C++, it generates a lot of warnings which don't really have anything to do with the code here. You may disable "-Wall" on for Visual C++.

1.3 Concept and Design

The framework implements an adjacency-list representation of graphs: vertices are stored as an vecrtor with each elements accessible via indices, and all edges of a vertex is stored in a list accessible via iterators. Vertices can be accessed via iterators as well. Each vertex should have some uniquely identifiable data (identifier), while each edge may have some arbitrary data.

An algorithm would use indices or iterators to access the vertices and use iterators to access the edges and operate on them. For an algorithm that takes a vertex and vertices as input, usually the input can ben in the form of index or indices -- there is really no need to use vertex "identifiers" as input since index of a vertex can be found from a vertex identifier.

The implementation supports (fairly) efficient insertion/deletion of vertices. A few helpers funtions are there to facilitate that: (1) because deletion can leave hole in the vertex vector, function is_in_use() can be used to test if the given index has a valid vertex or just empty; (2) vertex vector will dynamically grow when new ones are inserted, controlled by a growth factor. (3) vertex vector doesn't shrink by itself -- call function pack() to remove all holes in the vertex vector and store all vertices consecutively and thus shrink the vertex vector; (4) if you know the graph size beforehand, you can use reserve() to reserve the vertex vector.

1.3.1 Framework as An Interface

Adjacency-list is one possible representation of graphs, and implementation here is one possible implementation. For some algorithms, maybe a different representation or implementation works better. Algorithms implemented based on the framework is "portable" in the sense that you can easily use a different graph implementation with-

out needing to re-write the algorithms as long as the new implementation provides the same interface as specified in graph_intf.h -- which basically specifies index and iterator access to vertices and iterator access to edges.

1.4 Stuff of Interest

export_gdl.h: utility to export gdl (graph description language) file that can be imported to graph visualization tools to visualize graphs.

graph_gen.h: includes a graph generator based on B.M. Waxman's method; can be used to generate graphs to test graph algorithms.

import_gitalt.h: utility to import graph from the alt file by GeorgiaTech graph generator.

import_asconnect.h: utility to import graph from an "asconnect" files. Asconnect file is a list of AS's (Autonomous System in BGP routing) and their interconnections, ∗used to∗ be available for download from: http://www.antc.uoregon.edu/route-views/; however, when I checked recently, I didn't see them there, nor could I find them at http://www.routeviews.org/. You may need to contact them to see if those files are still available.

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Namespace Index

3.1 Namespace List

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

4 Class Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

5 Class Index

5.1 Class List

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

6 File Index

6.1 File List

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

7 Module Documentation

7.1 Exception classes

Classes

- class afl::general_except< str_type >

  General exception.
- class afl::unknown_except< str_type >
- class afl::init_failed< str_type >

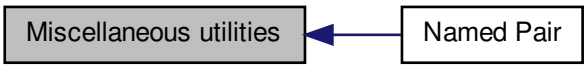
  Initialization failure exception.
- class afl::mem_alloc_failed< str_type >

  Memory allocation failure exception.
- class afg::invalid_path

  invalid path.

7.2 Miscellaneous utilities

Collaboration diagram for Miscellaneous utilities:



Classes

- struct afl::in_convt2t< T1, T2 >
- struct afl::runit< T >

  Operator( ) always returns 1 for any reference passed.
- struct afl::runit_p< T >

  Operator( ) always returns 1 for any pointer passed.
- struct afl::ref2value< T >

Operator ( ) returns the value of a reference.

- struct afl::ref2ref< T >

  Operator ( ) returns a const reference of a reference.

- struct afl::pointer2value< T >

  Operator ( ) returns the value of a pointer,.

Modules

- Named Pair

Functions

- template<class T1 , class T2 >
  std::istream & **afl::operator>>** (std::istream &is, in_convt2t< T1, T2 > i)
- template<class T >
  const T & afl::tmin (const T &a, const T &b)

  return the smaller one of two elements.
- template<class T >
  const T & afl::tmax (const T &a, const T &b)

  return the larger one of two elements.
- template<class T , class Cmp >
  const T & afl::tmin (const T &a, const T &b, Cmp cmp)

  return the smaller one of two elements, given comparison function object.
- template<class T , class Cmp >
  const T & afl::tmax (const T &a, const T &b, Cmp cmp)

  return the larger one of two elements, given comparison function object.

7.3 Named Pair

Collaboration diagram for Named Pair:

Classes

- struct afl::named_pair< NameT, KeyT >

Functions

- template<class NameT , class KeyT >
  bool afl::operator== (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs, equal if their names are the same.
- template<class NameT , class KeyT >
  bool afl::operator!= (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs, not equal if their names are different.
- template<class NameT , class KeyT >
  bool afl::operator== (const named_pair< NameT, KeyT > &lhs, const NameT &n)

  compare a named pair with a name
- template<class NameT , class KeyT >
  bool afl::operator!= (const named_pair< NameT, KeyT > &lhs, const NameT &n)

  compare a named pair with a name
- template<class NameT , class KeyT >
  bool afl::operator< (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs by their keys.
- template<class NameT , class KeyT >
  bool afl::operator> (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs by their keys.
- template<class NameT , class KeyT >
  bool afl::operator<= (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs by their keys.
- template<class NameT , class KeyT >
  bool afl::operator>= (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs by their keys.
- template<class NameT , class KeyT >
  std::ostream & afl::operator<< (std::ostream &os, const named_pair< NameT, KeyT > &np)

  Output two values within parenthese, separated by ",".

- template<class NameT , class KeyT >
  named_pair< NameT, KeyT > afl::make_npair (const NameT &lhs, const KeyT &rhs)

  produce a named_pair from two values

7.4 Random number related utilities

Classes

- struct afl::rand::CConst
- struct afl::rand::CUniform
- struct afl::rand::CExp

Functions

- void afl::rand::init (int n=111193)
- double afl::rand::gen (double d1, double d2)
- double afl::rand::gen (double d=1.0)

7.4.1 Function Documentation

7.4.1.1 double afl::rand::gen (double *d1*, double *d2*) [inline]

Generate a random number within the given range.

Returns

a double random number between d1 and d2

7.4.1.2 double afl::rand::gen (double *d* = 1.0) [inline]

Generate a random number within the given range.

Returns

a double random number between 0 and d

7.4.1.3 void afl::rand::init (int *n* = 111193) [inline]

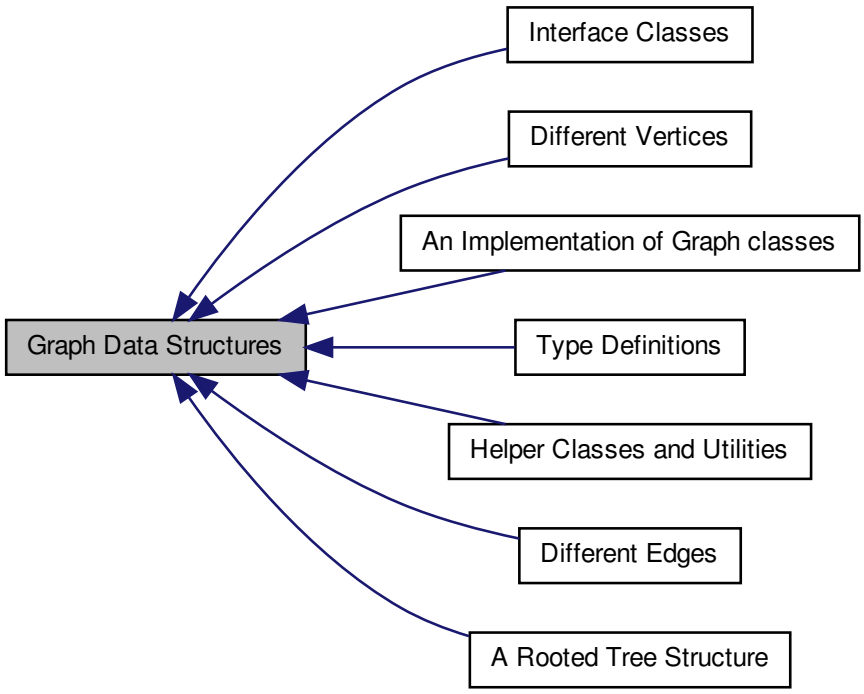
Initialize the random seed with system time. Use system time divided by a specific integer (default is 111193) as the seed. Thus makes it possible for a program to initialize the seed with a different number every time it runs though the integer passed to init_-rand() is the same.

Returns

none

7.5 Graph Data Structures

Collaboration diagram for Graph Data Structures:



Modules

- Interface Classes
- Different Vertices
- Different Edges
- An Implementation of Graph classes
- Helper Classes and Utilities

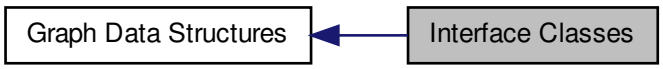
- A Rooted Tree Structure
- Type Definitions

7.5.1 Detailed Description

Data structures to support graph operations and algorithms.

7.6 Interface Classes

Collaboration diagram for Interface Classes:



Classes

- class afg::IEdge< EdgeDT >
- struct afg::equal_i< EDT >

  predicate to test if the indices of two edges are the same.
- struct afg::equal_n< EDT >

  predicate to test if the index of an edge is equal to an integer.
- struct afg::Cvoid
- class afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type >
- class afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >

Functions

- template<class EdgeDT >
  bool afg::operator== (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  equal operator for IEdge

- template<class EdgeDT >
  bool afg::operator!= (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  not_equal operator for IEdge
- template<class EdgeDT >
  bool afg::operator< (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  less operator for IEdge
- template<class EdgeDT >
  bool afg::operator> (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  larger operator for IEdge
- template<class EdgeDT >
  std::ostream & afg::operator<< (std::ostream &os, const IEdge< EdgeDT > &edge)

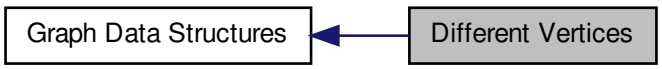
  conventional output operator for IEdge
- template<class VertexDT , class EdgeDT , class iterator_type , class const_iterator_type >
  std::ostream & afg::operator<< (std::ostream &os, const IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type > &rhs)

  output operator for IVertex
- template<class VertexDT , class EdgeDT , class iterator_type , class const_v_iterator_type , class f_eqv , class CVertex >
  std::ostream & afg::operator<< (std::ostream &os, const IGraph< VertexDT, EdgeDT, iterator_type, const_v_iterator_type, f_eqv, CVertex > &rhs)

  conventional output operator for IGraph

7.7 Different Vertices

Collaboration diagram for Different Vertices:



Classes

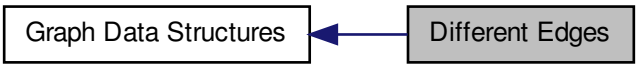
- struct afg::iddVertex< IDT, DT >
- struct afg::idwVertex< IDT >
- struct afg::xyVertex< T >
- struct afg::ixyVertex< IDT, T >
- struct afg::tsVertex

Functions

- template<class IDT , class DT >
  bool **afg::operator==** (const iddVertex< IDT, DT > &lhs, const iddVertex< IDT, DT > &rhs)
- template<class IDT , class DT >
  bool **afg::operator<** (const iddVertex< IDT, DT > &lhs, const iddVertex< IDT, DT > &rhs)
- template<class IDT , class DT >
  bool **afg::operator!=** (const iddVertex< IDT, DT > &lhs, const iddVertex< IDT, DT > &rhs)
- template<class IDT , class DT >
  std::ostream & **afg::operator<<** (std::ostream &os, const iddVertex< IDT, DT > &rhs)
- template<class IDT , class DT >
  std::istream & **afg::operator>>** (std::istream &is, iddVertex< IDT, DT > &rhs)
- template<class T >
  bool **afg::operator==** (const xyVertex< T > &lhs, const xyVertex< T > &rhs)
- template<class T >
  bool **afg::operator!=** (const xyVertex< T > &lhs, const xyVertex< T > &rhs)
- template<class T >
  std::ostream & **afg::operator<<** (std::ostream &os, const xyVertex< T > &v)
- template<class T >
  std::istream & **afg::operator>>** (std::istream &is, xyVertex< T > &v)
- template<class IDT , class T >
  bool **afg::operator==** (const ixyVertex< IDT, T > &lhs, const ixyVertex< IDT, T > &rhs)
- template<class IDT , class T >
  bool **afg::operator!=** (const ixyVertex< IDT, T > &lhs, const ixyVertex< IDT, T > &rhs)
- template<class IDT , class T >
  std::ostream & **afg::operator<<** (std::ostream &os, const ixyVertex< IDT, T > &v)
- std::istream & **afg::operator>>** (std::istream &is, tsVertex &v)
- std::ostream & **afg::operator<<** (std::ostream &os, const tsVertex &v)

7.8 Different Edges

Collaboration diagram for Different Edges:



Classes

- struct afg::Ce_nCare
- struct afg::CEdgeW2< WType1, WType2 >
- struct afg::CEdgeW3< WType1, WType2, WType3 >

stream input/output opertors

- std::ostream & **afg::operator**<< (std::ostream &os, const Ce_nCare &edge)
- template<class WType1 , class WType2 >
  std::ostream & afg::operator<< (std::ostream &os, const CEdgeW2< WType1, WType2 > &edge)
- template<class WType1 , class WType2 >
  std::istream & afg::operator>> (std::istream &is, CEdgeW2< WType1, WType2 > &edge)
- template<class WType1 , class WType2 , class WType3 >
  std::ostream & afg::operator<< (std::ostream &os, const CEdgeW3< WType1, WType2, WType3 > &edge)
- template<class WType1 , class WType2 , class WType3 >
  std::istream & afg::operator>> (std::istream &is, CEdgeW3< WType1, WType2, WType3 > &edge)

7.8.1 Function Documentation

7.8.1.1 template<class WType1 , class WType2 > std::ostream& afg::operator<< (std::ostream & *os,* const CEdgeW2< WType1, WType2 > & *edge*) [inline]

conventional stream output operator for CEdgeW2. weights are separated by space.

7.8.1.2 template<class WType1 , class WType2 , class WType3 > std::ostream& afg::operator<< (std::ostream & *os,* const CEdgeW3< WType1, WType2, WType3 > & *edge*) [inline]

conventional stream output operator for CEdgeW3. weights are separated by space.

7.8.1.3 template<class WType1 , class WType2 > std::istream& afg::operator>> (std::istream & *is,* CEdgeW2< WType1, WType2 > & *edge*) [inline]

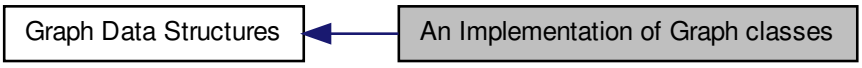
conventional stream input operator for CEdgeW2. Read two weights separated by space, may not work with some weight type.

7.8.1.4 template<class WType1 , class WType2 , class WType3 > std::istream& afg::operator>> (std::istream & *is,* CEdgeW3< WType1, WType2, WType3 > & *edge*) [inline]

conventional stream input operator for CEdgeW3. Read three weights separated by space, may not work with some weight type.

7.9 An Implementation of Graph classes

Collaboration diagram for An Implementation of Graph classes:



Classes

- class afg::CiEdge< EdgeDT >
- class afg::CiVertex< VertexDT, EdgeDT >
- class afg::CGraph< VertexDT, EdgeDT, f_eqv >

Defines

- #define graph_type afg::CGraph

  type of the graph provided by this file
- #define **_GRAPHDEFINED**

stream input/output operators

- template<class EdgeDT >
  std::istream & afg::operator>> (std::istream &is, CiEdge< EdgeDT > &edge)

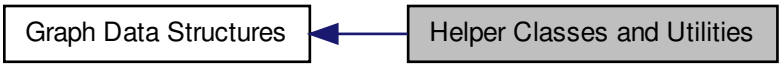
7.9.1 Function Documentation

7.9.1.1 template<class EdgeDT > std::istream& afg::operator>> (std::istream & *is*, CiEdge< EdgeDT > & *edge*) [inline]

operator to input CiEdge. Read an integer as index then a character (':') then edge_data

7.10 Helper Classes and Utilities

Collaboration diagram for Helper Classes and Utilities:



Classes

- class afg::CPath

path related utility functions

- std::ostream & afg::operator<< (std::ostream &os, CPath &lp)

  output operator to output path as a list of indices separated by space.
- bool afg::pred2path (int n, int *pnpred, int ns, int nt, CPath &lp)
- bool afg::pred2path (const std::vector< int > &vpred, int ns, int nt, CPath &lp)
- bool afg::allpred2path (int n, const std::vector< int > &vpred, int ns, int nt, CPath &lp)
- template<class GraphT , class Fun >
  Fun::result_type afg::path_length (const GraphT &gra, const CPath &lp, Fun fw)

7.10.1 Function Documentation

7.10.1.1 bool afg::allpred2path (int *n,* const std::vector< int > & *vpred,* int *ns,* int *nt,* CPath & *lp*)

get path from a n∗n predcessor vector.

Parameters

vpred	predcessor vector, size at least n∗n. vpred[i∗n+j] is the node (index) closest to j on the shortest path from source node i to j.

See also

pred2path

7.10.1.2 template<class GraphT , class Fun > Fun::result_type afg::path_length (const GraphT & *gra,* const CPath & *lp,* Fun *fw*)

get path length. GraphT is the type of graph passed.

Parameters

gra	graph to work on, to map the edges specified in lp.
fw	function object to get the length or "weight" of an edge. It takes a GraphT::EdgeDT∗ and returns a weight.

Exceptions

invalid_path,with	message "in path_length( ): edge from->to not found".

7.10.1.3 bool afg::pred2path (const std::vector< int > & *vpred,* int *ns,* int *nt,* CPath & *lp*)

get path from a predcessor vector.

Parameters

vpred	predcessor vector. vpred[i] is the node (index) closest to i on the shortest path from source node ns to i.
ns	source node
nt	destination node
lp	result path

Returns

true if path is successfully constructed, false otherwise.

7.10.1.4 bool afg::pred2path (int *n*, int * *pnpred*, int *ns*, int *nt*, CPath & *lp*)

get path from a predcessor index array.

Parameters

n	number of nodes
pnpred	predcessor array, at least of size n. If array is smaller than that, out of range error may happen. pnpred[i] is the node (index) closest to i on the shortest path from source node ns to i.
ns	source node
nt	destination node
lp	result path

Returns

true if path is successfully constructed, false otherwise.

7.11 A Rooted Tree Structure

Collaboration diagram for A Rooted Tree Structure:

Classes

- class afg::CrTree< VDT, EDT, f_eqv >

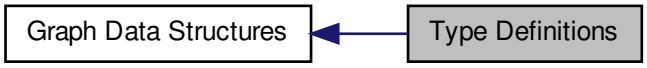
Defines

- #define rtree_type afg::CrTree

  type of rooted tree provided by rtree.h
- #define **_RTREEDEFINED**

7.12 Type Definitions

Collaboration diagram for Type Definitions:



Typedefs

- typedef CEdgeW2< double, double > afg::T_d2Edge
  edge type with 2 double weights
- typedef CEdgeW3< double, double, double > afg::T_d3Edge
  edge type with 3 double weights
- typedef graph_type< int, Ce_nCare > afg::T_nGraph
  graph with an integer id only
- typedef graph_type< int, double > afg::T_ndGraph
  graph with an integer id and a double weight
- typedef graph_type< xyVertex< int >, double > afg::T_xydGraph
  graph with integer xyVertex and a double weight
- typedef graph_type< xyVertex< int >, int > afg::T_xynGraph
  graph with integer xyVertex and an integer weight
- typedef graph_type< int, T_d2Edge > afg::T_nd2Graph
  graph with an integer index and an edge of two double numbers
- typedef graph_type< xyVertex< int >, T_d2Edge > afg::T_xyd2Graph
  graph with integer xyVertex and an edge of two double numbers
- typedef graph_type< tsVertex, int > afg::T_tsGraph
  transit-stub graph
- typedef rtree_type< int, Ce_nCare > afg::T_nrTree
  rtree with an integer id only
- typedef rtree_type< int, double > afg::T_ndrTree
  rtree with an integer id and a double weight
- typedef rtree_type< xyVertex< int >, double > afg::T_xydrTree
  rtree with integer xyVertex and a double weight
- typedef rtree_type< xyVertex< int >, int > afg::T_xynrTree

rtree with integer xyVertex and an integer weight

- typedef rtree_type< int, T_d2Edge > afg::T_nd2rTree

  rtree with an integer index and an edge of two double numbers

- typedef rtree_type< xyVertex< int >, T_d2Edge > afg::T_xyd2rTree

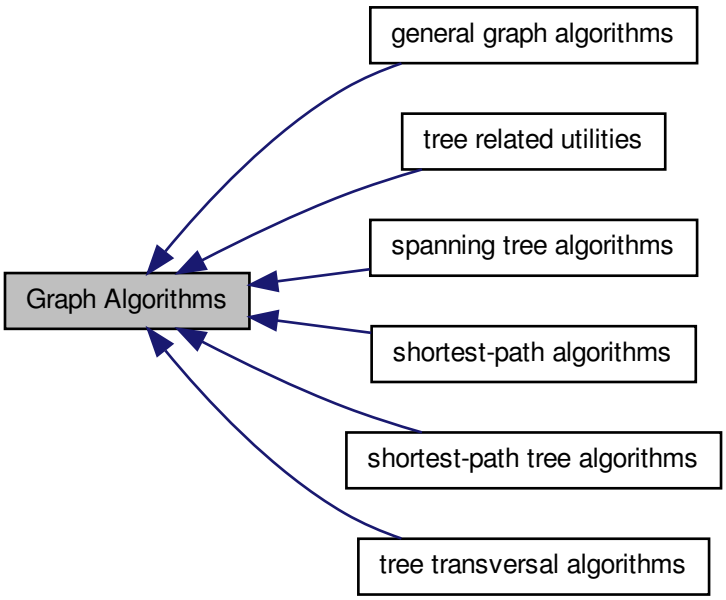
  rtree with integer xyVertex and an edge of two double numbers

7.12.1 Detailed Description

Type definitions to save some typing.

7.13 Graph Algorithms

Collaboration diagram for Graph Algorithms:

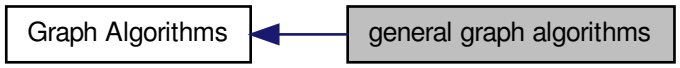


Modules

- general graph algorithms
- shortest-path algorithms
- tree related utilities
- shortest-path tree algorithms
- spanning tree algorithms
- tree transversal algorithms

7.14 general graph algorithms

Collaboration diagram for general graph algorithms:



Functions

- template<class GT >
  bool afg::is_connected (const GT &grf)
- template<class GraphT , class Fun >
  Fun::result_type afg::graph_cost (const GraphT &g, Fun f_w)

7.14.1 Function Documentation

7.14.1.1 template<class GraphT , class Fun > Fun::result_type afg::graph_cost (const GraphT & *g*, Fun *f_w*)

Compute the total cost of all edges of the graph.

Parameters

g	the given graph
f_w	function object that returns the cost of an edge

7.14.1.2 template<class GT > bool afg::is_connected (const GT & *grf*)

Determine if a graph is connected. GT: a graph type provides interface defined in graph_intf.h.

Parameters

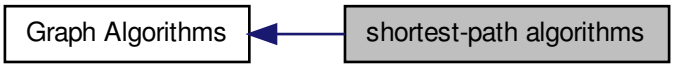
grf	the graph, doesn't need to be "packed" beforehand.

Returns

true if grf is connected, false otherwise

7.15 shortest-path algorithms

Collaboration diagram for shortest-path algorithms:



Classes

- class afg::CkthSP< GraphT, Fun >

Functions

- template<class GraphT , class Fun >
  bool afg::dijkstra (const GraphT &graph, int nsource, Fun f_weight, typename Fun::result_type w_infty, std::vector< int > &pred, std::vector< typename Fun::result_type > &dist)
- template<class GraphT , class Fun >
  Fun::result_type afg::dijkstra_t (const GraphT &graph, int nsource, int ndst, Fun f_weight, typename Fun::result_type w_infty, CPath &lp)
- template<class GraphT , class Fun >
  bool afg::floyd_warshall_allsp (const GraphT &graph, Fun f_weight, typename Fun::result_type w_infty, std::vector< int > &pred, std::vector< typename Fun::result_type > &dist)

7.15.1 Function Documentation

7.15.1.1 template<class GraphT , class Fun > bool afg::dijkstra (const GraphT & *graph,* int *nsource,* Fun *f_weight,* typename Fun::result_type *w_infty,* std::vector< int > & *pred,* std::vector< typename Fun::result_type > & *dist*)

Dijkstra algorithm to compute single source shortest paths. GraphT: type of the graph.

Parameters

graph	graph on which to run the algorithm, no "negative" weighted edge. Graph is not required to be "packed".
nsource	source node, should be between 0 and graph.szie()-1
f_weight	function object to get weight of an edge (passed by pointer), e.g.: pointer2value<int>( ), runit_p<..>( ).
w_infty	an "upper bound" value of weight which should be larger than the path length of any valid path; used to initialize distance to all nodes.
pred	a vector to store predecessor information, must have size()>=graph.range().
dist	a vector to store path length information (from nsource to a node), must have size()>=graph.range(); if there is no path from nsource to node i, then dist[i]=w_infty.

Exceptions

unknown_error	

7.15.1.2 template<class GraphT , class Fun > Fun::result_type afg::dijkstra_t (const GraphT & *graph,* int *nsource,* int *ndst,* Fun *f_weight,* typename Fun::result_type *w_infty,* CPath & *lp*)

run Dijkstra's algorithm for a single destination. A wrapper for the above more general dijkstra().

Parameters

graph	on which to find the path, no "negative" weighted edge. Graph is not required to be "packed".
nsource	source nodes
ndst	destination node
f_weight	edge weight function, pass pointer
w_infty	an upper bound value of weight which should be larger than the path length of any valid path
lp	result path

Returns

path length, w_infty if no path found.

Exceptions

unknown_error,from	dijkstra( ) function call.

7.15.1.3 template<class GraphT , class Fun > bool afg::floyd_warshall_allsp (const GraphT & *graph,* Fun *f_weight,* typename Fun::result_type *w_infty,* std::vector< int > & *pred,* std::vector< typename Fun::result_type > & *dist*)

Floyd-Warshall algorithm to compute all pair shortest paths.

Parameters

graph	graph on which to run the algorithm, no "negative" weighted edge.
pred[i∗n+j],:	predcessor of j on the shortest path from i to j
dist[i∗n+j],:	distance of shortest path from i to j

7.16 tree related utilities

Collaboration diagram for tree related utilities:

Functions

- template<class TreeT >
  void afg::prune_tree (TreeT &tree, const set< int > &in_set)
- template<class GT , class TT >
  bool afg::pred2tree (const GT &gra, int n, const int ∗pred, TT &tree)
- template<class GT , class TT >
  bool afg::pred2tree (const GT &gra, const vector< int > &vpred, TT &tree)
- template<class TT , class EDT >
  bool afg::pred2tree_s (const vector< int > &vpred, const vector< EDT > &vdist, TT &tree)
- template<class GT , class TT >
  bool afg::all_pred2tree (const GT &gra, const vector< int > &vpred, int ns, TT &tree)

- template<class TT , class EDT >
  bool afg::all_pred2tree_s (const vector< int > &vpred, const vector< EDT > &vdist, int n, int ns, TT &tree)

7.16.1 Function Documentation

7.16.1.1 template<class GT , class TT > bool afg::all_pred2tree (const GT & *gra,* const vector< int > & *vpred,* int *ns,* TT & *tree*)

get a "full-blown" tree from an all-pair-shortest-pathpredecessor vector and the corresponding graph. The tree has all the vertices and all the vertex and edge information from the original graph.

Parameters

vpred[i∗n+j],:	predcessor of j on the shortest path from i to j
ns	source node

7.16.1.2 template<class TT , class EDT > bool afg::all_pred2tree_s (const vector< int > & *vpred,* const vector< EDT > & *vdist,* int *n,* int *ns,* TT & *tree*)

get a "simple" tree from an all-shortest-path predecessor vector and the corresponding distance vector. The tree has an integer vertex (its index in the original graph) and a single weight edge (distance from distance vector). Note: if the original graph from which vpred and vdist are generated is not packed, then resulting tree may have "extra" node(s).

Parameters

vpred[i∗n+j],:	predcessor of j on the shortest path from i to j
vdist[i∗n+j],:	distance of shortest path from i to j

7.16.1.3 template<class GT , class TT > bool afg::pred2tree (const GT & *gra,* int *n,* const int ∗ *pred,* TT & *tree*)

get a "full-blown" tree from a predecessor array and the corresponding graph. The tree has all the vertices and all the vertex and edge information from the original graph.

7.16.1.4 template<class GT , class TT > bool afg::pred2tree (const GT & *gra,* const vector< int > & *vpred,* TT & *tree*)

get a "full-blown" tree from a predecessor vector and the corresponding graph. The tree has all the vertices and all the vertex and edge information from the original graph.

7.16.1.5 template<class TT , class EDT > bool afg::pred2tree_s (const vector< int > & *vpred,* const vector< EDT > & *vdist,* TT & *tree*)

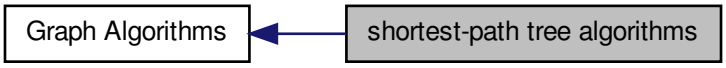
get a "simple" tree from a predecessor vector and the corresponding distance vector. The tree has integer vertecies (their indecies in the original graph) and edges of a single weight (distance from distance vector), or edges of type that can initialize from the distance type. Note: if the original graph from which vpred and vdist are generated using Dijkstra and the graph is not packed, then resulting tree may have "extra" node(s).

7.16.1.6 template<class TreeT > void afg::prune_tree (TreeT & *tree,* const set< int > & *in_set*)

prune a tree such that a vertex can be a leaf node only if it is in in_set .

7.17 shortest-path tree algorithms

Collaboration diagram for shortest-path tree algorithms:



Functions

- template<class GraphT , class TreeT , class Fun >
  bool afg::sptree_all (const GraphT &gra, int ns, TreeT &tree, Fun f_weight, typename Fun::result_type w_infty)
- template<class GraphT , class TreeT , class Fun >
  bool afg::sptree (const GraphT &gra, int ns, TreeT &tree, const std::set< int > &smem, Fun f_weight, typename Fun::result_type w_infty)
- template<class GraphT , class TreeT , class Fun >
  bool afg::sptree_all_s (const GraphT &gra, int ns, TreeT &tree, Fun f_weight, typename Fun::result_type w_infty)
- template<class GraphT , class TreeT , class Fun >
  bool afg::sptree_s (const GraphT &gra, int ns, TreeT &tree, const std::set< int > &smem, Fun f_weight, typename Fun::result_type w_infty)

7.17.1 Function Documentation

7.17.1.1 template<class GraphT , class TreeT , class Fun > bool afg::sptree (const GraphT & *gra,* int *ns,* TreeT & *tree,* const std::set< int > & *smem,* Fun *f_weight,* typename Fun::result_type *w_infty*)

build a single-source shortest-path tree for a subset of nodes. Nodes are given by index. It is apparent that only nodes from the given set can be tree leaf nodes. The tree is "full-blown", having all the vertices and all vertex and edge information from the original graph.

See also

sptree_all

7.17.1.2 template<class GraphT , class TreeT , class Fun > bool afg::sptree_all (const GraphT & *gra,* int *ns,* TreeT & *tree,* Fun *f_weight,* typename Fun::result_type *w_infty*)

build a single-source shortest-path tree. The tree is "full-blown", having all the vertices and all vertex and edge information from the original graph.

Parameters

gra	the graph
ns	index of source node
tree	result tree
f_weight	function to retrieve the weight(distance) of an edge, an edge pointer will be passed.
w_infty	a weight that is large enough to be considered infinity.

Returns

true is successful, false if any exception happens when it calls dijkstra() to compute the shortest paths from the source.

7.17.1.3 template<class GraphT , class TreeT , class Fun > bool afg::sptree_all_s (const GraphT & *gra,* int *ns,* TreeT & *tree,* Fun *f_weight,* typename Fun::result_type *w_infty*)

build a "simple" single-source shortest-path tree. The tree has integer vertecies (their indices in the original graph) and a single weight edge.

See also

sptree_all pred2tree_s

7.17.1.4 template<class GraphT , class TreeT , class Fun > bool afg::sptree_s (const GraphT & *gra,* int *ns,* TreeT & *tree,* const std::set< int > & *smem,* Fun *f_weight,* typename Fun::result_type *w_infty*)

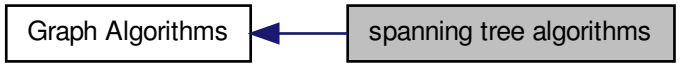
build a "simple" single-source shortest-path tree for a subset of nodes. The tree has integer vertecies (their indices in the original graph) and a single weight edge.

See also

sptree pred2tree_s

7.18 spanning tree algorithms

Collaboration diagram for spanning tree algorithms:



Functions

- template<class GT , class TT , class EDT >
  bool afg::extend_greedy (TT &tree, const GT &gra, const std::set< int > &snodes, const std::vector< int > &vpred, const std::vector< EDT > &vdist, EDT d_infty)
- template<class TT , class EDT >
  bool afg::extend_greedy_s (TT &tree, const std::set< int > &snodes, int n, const std::vector< int > &vpred, const std::vector< EDT > &vdist, EDT d_infty)

7.18.1 Function Documentation

7.18.1.1 template<class GT , class TT , class EDT > bool afg::extend_greedy (TT & *tree,* const GT & *gra,* const std::set< int > & *snodes,* const std::vector< int > & *vpred,* const std::vector< EDT > & *vdist,* EDT *d_infty*)

extend a tree to cover more nodes by greedy strategy. The tree is a "full-blown" one.

Parameters

tree	the tree to be extended.
gra	the original graph.
snodes	set of nodes that to be covered (index)
vpred	all pair shortest path predcessors (n∗n)
vdist	all pari shortest path distances (n∗n)

7.18.1.2 template<class TT , class EDT > bool afg::extend_greedy_s (TT & *tree,* const std::set< int > & *snodes,* int *n,* const std::vector< int > & *vpred,* const std::vector< EDT > & *vdist,* EDT *d_infty*)

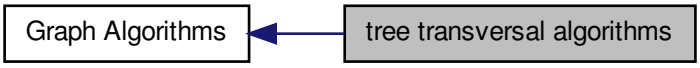
extend a tree to cover more nodes by greedy strategy. The tree is a "simple" tree: vertex data type is int and edge has single weight (from vdist). Vertex data is index from the original graph given in snodes.

Parameters

snodes	set of nodes that to be covered (index)
vpred	all pair shortest path predcessors (n∗n)
vdist	all pari shortest path distances (n∗n)

7.19 tree transversal algorithms

Collaboration diagram for tree transversal algorithms:



Classes

- struct afg::tree_dfs_c< TreeT >
- struct afg::tree_dfs< TreeT >

7.20 graph convertors

Collaboration diagram for graph convertors:

Classes

- struct afg::lvConverter< VDT1, VDT2 >
- struct afg::lviConverter< VDT1, VDT2 >
- struct afg::leConverter< EDT1, EDT2 >
- struct afg::leiConverter< EDT1, EDT2 >
- struct afg::vConverter< VDT1, VDT2 >
- struct afg::vCopier< VDT >
- struct afg::viConverter< VDT1, VDT2 >
- struct afg::viCopier< VDT >
- struct afg::eConverter< EDT1, EDT2 >
- struct afg::eCopier< EDT >
- struct afg::eiConverter< EDT1, EDT2 >
- struct afg::eiCopier< EDT >
- struct afg::v2index< VDT1 >
- struct afg::e_add_dist< GT, DT >
- struct afg::e_add_rand< EDT1, DT >
- struct afg::e_add_rand_s< EDT1, DT >

Functions

- template<class GT1 , class GT2 , class Vconvert , class Econvert >
  void afg::graph_convert (const GT1 &g1, GT2 &g2, Vconvert f_vc, Econvert f_ec)
- template<class VDT , class WT1 >
  void afg::grfw2grfw2 (const graph_type< VDT, WT1 > &gra0, graph_type< VDT, CEdgeW2< WT1, double > > &ngra)
- template<class VDT , class WT1 >
  void afg::grfw2grfw2_s (const graph_type< VDT, WT1 > &gra0, graph_type< VDT, CEdgeW2< WT1, double > > &ngra)

7.20.1 Function Documentation

7.20.1.1 template<class GT1 , class GT2 , class Vconvert , class Econvert > void afg::graph_convert (const GT1 & *g1,* GT2 & *g2,* Vconvert *f_vc,* Econvert *f_ec*)

convert a graph of tyep GT1 to type GT2.

Parameters

g1	original graph
g2	converted graph, should be empty before being passed.
f_vc	a vertex converter with index, providing interface IviConverter
f_ec	an edge converter with indices, providing interface IeiConverter

7.20.1.2 template<class VDT , class WT1 > void afg::grfw2grfw2 (const graph_type< VDT, WT1 > & *gra0,* graph_type< VDT, CEdgeW2< WT1, double > > & *ngra*)

graph of single weight -> two double weights. In new graph, each edge has two weights: one is the original, one is a random number between 0 and 1. ngra should be cleared fist (or newly created without inserting any vertex) before passed to this function. VT is the type of vertex data and WT1 is the type of original weight.

Parameters

gra0	original graph
ngra	new graph

Note: obsolete.

7.20.1.3 template<class VDT , class WT1 > void afg::grfw2grfw2_s (const graph_type< VDT, WT1 > & *gra0,* graph_type< VDT, CEdgeW2< WT1, double > > & *ngra*)

graph of single weight -> two double weights. Edges of original graph are assumed to be symmetric (edge i->j has same "edge data" as edge j->i), added new random weight

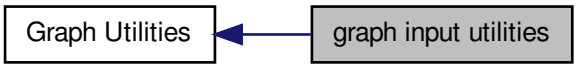
See also

grfw2grfw2

Note: obsolete.

7.21 graph input utilities

Collaboration diagram for graph input utilities:



Functions

- void **afg::input_gra_header** (std::istream &is, std::string &sformat, int &nn, int &nl, std::string &sdir)
- template<class GT >
  void afg::input_gra_die (GT &gra, std::istream &is)
- template<class GT >
  void afg::input_gra_vie (GT &gra, std::istream &is)
- template<class GT , class VIDT >
  void afg::input_gra_vve (GT &gra, std::istream &is)
- template<class GT >
  bool **afg::input_gra_i_2e** (GT &gra, std::istream &is)

7.21.1 Function Documentation

7.21.1.1 template<class GT > void afg::input_gra_die (GT & *gra,* std::istream & *is*)

input graph for type die: default integer ids (from 0 to n-1) and links with certain type of edge data, nodes are not explicitly listed in the file

Exceptions

general_except	with a specific error message

7.21.1.2 template<class GT > void afg::input_gra_vie (GT & *gra,* std::istream & *is*)

input graph for type vie

Exceptions

general_except	with a specific error message

7.21.1.3 template<class GT , class VIDT > void afg::input_gra_vve (GT & *gra,* std::istream & *is*)

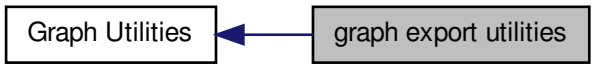
input graph for type vve

Exceptions

general_except	with a specific error message

7.22 graph export utilities

Collaboration diagram for graph export utilities:

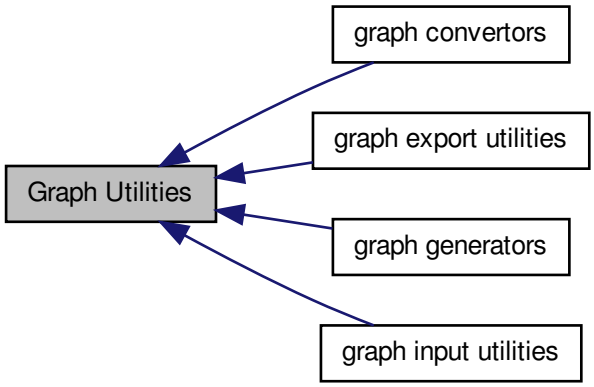


Classes

- struct afg::title_ind< VertexT >
- struct afg::title_vd< VertexT >
- struct afg::title_ivd< VertexT >
- struct afg::node_empty< VertexT >
- struct afg::node_combine< VertexT, F1, F2 >
- struct afg::node_xyloc< VertexT >
- struct afg::node_color< VertexT >
- struct afg::edge_empty< EdgeT >
- struct afg::edge_label_ed< EdgeT >
- struct afg::edge_combine< EdgeT, F1, F2 >
- class afg::export_gdl< GraphT, Ftitle, Fv_optional, Fe_optional >

7.23 Graph Utilities

Collaboration diagram for Graph Utilities:



Modules

- graph convertors
- graph input utilities
- graph export utilities
- graph generators

7.23.1 Detailed Description

graph generators, convertors, input/export utilities

7.24 graph generators

Collaboration diagram for graph generators:

Functions

- void afg::grid_graph_gen (T_xynGraph &grf, int n, int m)
- int afg::bmw_graph_gen (T_xydGraph &dgrf, int ngrid, int n, int ntry=10, double lamda=0.3, double rou=0.3)

7.24.1 Function Documentation

7.24.1.1 int afg::bmw_graph_gen (T_xydGraph & *dgrf,* int *ngrid,* int *n,* int *ntry* = 10, double *lamda* = 0.3, double *rou* = 0.3)

a graph generator based on B.M. Waxman's method. Generate a connected directed graph where if there is an edge (i->j) then there is an edge (j->i). Graph generated has vertex data type of xyVertex<int> (integer x/y coordinates) and edge data type of double, which is the geometric distance between two nodes.

Parameters

dgrf	result graph, each vertex has integer x,y coordinates, each edge has a double weight which is geometric distance between nodes. dgrf should be empty before passed to this function; dgrf.clear( ) will make it emtpy. A newly created graph is empty as well.
ngrid	grid size, graph nodes are picked from an ngrid∗ngrid area
n	number of nodes
ntry	number of trials; if no connected graph can be generated after n trials, then gives up.

Returns

number of links in graph generated; since every link is bidirectional, # of edges=2∗(# of links).

7.24.1.2 void afg::grid_graph_gen (T_xynGraph & *grf*, int *n*, int *m*)

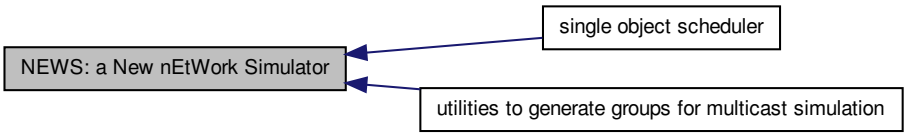
generate bi-directional mesh of n column, m rows. Graph generated has vertex data type of xyVertex<int> (integer x/y coordinates) and edge data type of int (every edge has weight 1).

Parameters

grf	result graph. grf should be empty before passed to this function; grf.clear( ) will make it emtpy. A newly created graph is empty as well.

7.25 NEWS: a New nEtWork Simulator

Collaboration diagram for NEWS: a New nEtWork Simulator:



Modules

- single object scheduler
- utilities to generate groups for multicast simulation

7.26 single object scheduler

Collaboration diagram for single object scheduler:

Classes

- class news::PktObj
- class news::SSEvent
- class news::SScheduler
- class news::SSNetObj
- class news::SSimulator

Typedefs

- typedef double news::TimeS

  timestamp
- typedef double news::TimeE

  elapsed time

7.27 utilities to generate groups for multicast simulation

Collaboration diagram for utilities to generate groups for multicast simulation:

Classes

- struct news::CConstGrpSz
- struct news::CUniformGrpSz
- struct news::CGrpGenerator
- class news::CRandGrpX< GrpSzG >
- class news::CRandGrpG
- class news::CRandGrpG2< GraphT >
- class news::CWeightedGrpG< GraphT, GrpSzG >

Functions

- template<class BT >
  void news::rand_seq_gen (const BT &base, double alpha, int ns, int nt, std::list< std::pair< int, int > > &lact, std::set< int > &sdest)

7.27.1 Function Documentation

7.27.1.1 template<class BT > void news::rand_seq_gen (const BT & *base,* double *alpha,* int *ns,* int *nt,* std::list< std::pair< int, int > > & *lact,* std::set< int > & *sdest*)

generate a sequence of node addition/deletion actions for a multicast group. BT: a class that provides at least two interfaces "int size()" and "bool can_be_member(int i)". The 1st one returns the number of nodes in a graph, and the 2nd determines if a node of index i can be a multicast group member. For example, CnmcBase<...> (research/nmcbase.h) or something derived from it. Reference (paper):

Parameters

base	an instance of BT that provides size() function and can_be_member(int i) function that determines if node i can be a multicast group member.
alpha	percentage of nodes that a group can have in average; i.e., at steady state, average number of group members(including source node ns) is base.size()∗alpha.
ns	source node of a group, it will never be deleted from the group.
nt	number of actions to generate after an initial sequence of additions that brings the number of members to the average number.
lact	a list of actions; for every pair in the list, the 1st integer number represents the action, 1: add, -1: delete; the 2nd integer (>=0, <base.size()) is the index of the node for the action. First a sequence of additions will be generated until the number of group members reaches the average, then addition/deletion
sdest	set of destination nodes in the group (not including ns); initially it is the set of current destination nodes in the group; when this procedure returns, it is the set of destination nodes that are in the group after the list of actions.

7.28 Stuff for Research Work

Classes

- class afg::CnmcBase< GT, WT1, WT2, WT3 >

Functions

- template<class GT >
  bool afg::import_gitalt_ts (GT &gra, std::istream &is)
- template<class GT >
  void afg::import_asconnect (GT &gra, std::istream &is)

7.28.1 Function Documentation

7.28.1.1 template<class GT > void afg::import_asconnect (GT & *gra,* std::istream & *is*)

import graph from an "asconnect" file. Asconnect file is a list of AS's and their interconnections, available for download from: http://www.antc.uoregon.edu/route-views/.

File format per line: as_index -> #_of_neighbors : a list of AS neighbors separated by ":". The imported graph will have vertex data of int type (AS index), each edge will have weight 1(type int). All links are assumed to be bi-directional.

7.28.1.2 template<class GT > bool afg::import_gitalt_ts (GT & *gra,* std::istream & *is*)

import graph from the alt file by GeorgiaTech graph generator. File format (transit stub graph): first line is some explaination

2nd line: number_of_nodes number_of_edges keyword"transtub"

3rd line: empty line

4th line: vertex line

list of nodes (as many as number_nodes)

empty line

edge line

list of links (as many as half of number_edges -- bidirectional links)

GT is the graph type, it must have vertex data type tsVertex and edge data type int (or convertions from them provided).

8 Namespace Documentation

8.1 afg Namespace Reference

Classes

- class IEdge
- struct equal_i

  predicate to test if the indices of two edges are the same.
- struct equal_n

  predicate to test if the index of an edge is equal to an integer.
- struct Cvoid
- class IVertex
- class IGraph
- struct iddVertex
- struct idwVertex
- struct xyVertex
- struct ixyVertex
- struct tsVertex
- struct Ce_nCare
- struct CEdgeW2
- struct CEdgeW3
- class CiEdge
- class CiVertex
- class CGraph
- class CPath
- class CrTree
- class CkthSP
- struct tree_dfs_c
- struct tree_dfs
- struct IvConverter
- struct IviConverter
- struct IeConverter
- struct IeiConverter
- struct vConverter
- struct vCopier
- struct viConverter
- struct viCopier
- struct eConverter
- struct eCopier
- struct eiConverter
- struct eiCopier
- struct v2index
- struct e_add_dist
- struct e_add_rand

- struct e_add_rand_s
- struct title_ind
- struct title_vd
- struct title_ivd
- struct node_empty
- struct node_combine
- struct node_xyloc
- struct node_color
- struct edge_empty
- struct edge_label_ed
- struct edge_combine
- class export_gdl
- class invalid_path

  invalid path.
- class CnmcBase

Typedefs

- typedef CEdgeW2< double, double > T_d2Edge

  edge type with 2 double weights
- typedef CEdgeW3< double, double, double > T_d3Edge

  edge type with 3 double weights
- typedef graph_type< int, Ce_nCare > T_nGraph

  graph with an integer id only
- typedef graph_type< int, double > T_ndGraph

  graph with an integer id and a double weight
- typedef graph_type< xyVertex< int >, double > T_xydGraph

  graph with integer xyVertex and a double weight
- typedef graph_type< xyVertex< int >, int > T_xynGraph

  graph with integer xyVertex and an integer weight
- typedef graph_type< int, T_d2Edge > T_nd2Graph

  graph with an integer index and an edge of two double numbers
- typedef graph_type< xyVertex< int >, T_d2Edge > T_xyd2Graph

  graph with integer xyVertex and an edge of two double numbers
- typedef graph_type< tsVertex, int > T_tsGraph

  transit-stub graph
- typedef rtree_type< int, Ce_nCare > T_nrTree

  rtree with an integer id only
- typedef rtree_type< int, double > T_ndrTree

  rtree with an integer id and a double weight

- typedef rtree_type< xyVertex< int >, double > T_xydrTree

  rtree with integer xyVertex and a double weight
- typedef rtree_type< xyVertex< int >, int > T_xynrTree

  rtree with integer xyVertex and an integer weight
- typedef rtree_type< int, T_d2Edge > T_nd2rTree

  rtree with an integer index and an edge of two double numbers
- typedef rtree_type< xyVertex< int >, T_d2Edge > T_xyd2rTree

  rtree with integer xyVertex and an edge of two double numbers

Functions

- template<class EdgeDT >
  bool operator== (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  equal operator for IEdge
- template<class EdgeDT >
  bool operator!= (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  not_equal operator for IEdge
- template<class EdgeDT >
  bool operator< (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  less operator for IEdge
- template<class EdgeDT >
  bool operator> (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  larger operator for IEdge
- template<class EdgeDT >
  std::ostream & operator<< (std::ostream &os, const IEdge< EdgeDT > &edge)

  conventional output operator for IEdge
- template<class VertexDT , class EdgeDT , class iterator_type , class const_iterator_type >
  std::ostream & operator<< (std::ostream &os, const IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type > &rhs)

  output operator for IVertex
- template<class VertexDT , class EdgeDT , class iterator_type , class const_v_iterator_type , class f_eqv , class CVertex >
  std::ostream & operator<< (std::ostream &os, const IGraph< VertexDT, EdgeDT, iterator_type, const_v_iterator_type, f_eqv, CVertex > &rhs)

  conventional output operator for IGraph
- template<class IDT , class DT >
  bool **operator==** (const iddVertex< IDT, DT > &lhs, const iddVertex< IDT, DT > &rhs)
- template<class IDT , class DT >
  bool **operator**< (const iddVertex< IDT, DT > &lhs, const iddVertex< IDT, DT > &rhs)

- template<class IDT , class DT >
  bool **operator!=** (const iddVertex< IDT, DT > &lhs, const iddVertex< IDT, DT > &rhs)
- template<class IDT , class DT >
  std::ostream & **operator**<< (std::ostream &os, const iddVertex< IDT, DT > &rhs)
- template<class IDT , class DT >
  std::istream & **operator**>> (std::istream &is, iddVertex< IDT, DT > &rhs)
- template<class T >
  bool **operator==** (const xyVertex< T > &lhs, const xyVertex< T > &rhs)
- template<class T >
  bool **operator!=** (const xyVertex< T > &lhs, const xyVertex< T > &rhs)
- template<class T >
  std::ostream & **operator**<< (std::ostream &os, const xyVertex< T > &v)
- template<class T >
  std::istream & **operator**>> (std::istream &is, xyVertex< T > &v)
- template<class IDT , class T >
  bool **operator==** (const ixyVertex< IDT, T > &lhs, const ixyVertex< IDT, T > &rhs)
- template<class IDT , class T >
  bool **operator!=** (const ixyVertex< IDT, T > &lhs, const ixyVertex< IDT, T > &rhs)
- template<class IDT , class T >
  std::ostream & **operator**<< (std::ostream &os, const ixyVertex< IDT, T > &v)
- std::istream & **operator**>> (std::istream &is, tsVertex &v)
- std::ostream & **operator**<< (std::ostream &os, const tsVertex &v)
- template<class GT >
  bool is_connected (const GT &grf)
- template<class GraphT , class Fun >
  Fun::result_type graph_cost (const GraphT &g, Fun f_w)
- template<class GraphT , class Fun >
  bool dijkstra (const GraphT &graph, int nsource, Fun f_weight, typename Fun::result_type w_infty, std::vector< int > &pred, std::vector< typename Fun::result_type > &dist)
- template<class GraphT , class Fun >
  Fun::result_type dijkstra_t (const GraphT &graph, int nsource, int ndst, Fun f_weight, typename Fun::result_type w_infty, CPath &lp)
- template<class GraphT , class Fun >
  bool floyd_warshall_allsp (const GraphT &graph, Fun f_weight, typename Fun::result_type w_infty, std::vector< int > &pred, std::vector< typename Fun::result_type > &dist)
- template<class TreeT >
  void prune_tree (TreeT &tree, const set< int > &in_set)
- template<class GT , class TT >
  bool pred2tree (const GT &gra, int n, const int *pred, TT &tree)

- template<class GT , class TT >
  bool pred2tree (const GT &gra, const vector< int > &vpred, TT &tree)
- template<class TT , class EDT >
  bool pred2tree_s (const vector< int > &vpred, const vector< EDT > &vdist, TT &tree)
- template<class GT , class TT >
  bool all_pred2tree (const GT &gra, const vector< int > &vpred, int ns, TT &tree)
- template<class TT , class EDT >
  bool all_pred2tree_s (const vector< int > &vpred, const vector< EDT > &vdist, int n, int ns, TT &tree)
- template<class GraphT , class TreeT , class Fun >
  bool sptree_all (const GraphT &gra, int ns, TreeT &tree, Fun f_weight, typename Fun::result_type w_infty)
- template<class GraphT , class TreeT , class Fun >
  bool sptree (const GraphT &gra, int ns, TreeT &tree, const std::set< int > &smem, Fun f_weight, typename Fun::result_type w_infty)
- template<class GraphT , class TreeT , class Fun >
  bool sptree_all_s (const GraphT &gra, int ns, TreeT &tree, Fun f_weight, typename Fun::result_type w_infty)
- template<class GraphT , class TreeT , class Fun >
  bool sptree_s (const GraphT &gra, int ns, TreeT &tree, const std::set< int > &smem, Fun f_weight, typename Fun::result_type w_infty)
- template<class GT , class TT , class EDT >
  bool extend_greedy (TT &tree, const GT &gra, const std::set< int > &snodes, const std::vector< int > &vpred, const std::vector< EDT > &vdist, EDT d_infty)
- template<class TT , class EDT >
  bool extend_greedy_s (TT &tree, const std::set< int > &snodes, int n, const std::vector< int > &vpred, const std::vector< EDT > &vdist, EDT d_infty)
- template<class GT1 , class GT2 , class Vconvert , class Econvert >
  void graph_convert (const GT1 &g1, GT2 &g2, Vconvert f_vc, Econvert f_ec)
- void **input_gra_header** (std::istream &is, std::string &sformat, int &nn, int &nl, std::string &sdir)
- template<class GT >
  void input_gra_die (GT &gra, std::istream &is)
- template<class GT >
  void input_gra_vie (GT &gra, std::istream &is)
- template<class GT , class VIDT >
  void input_gra_vve (GT &gra, std::istream &is)
- template<class GT >
  bool **input_gra_i_2e** (GT &gra, std::istream &is)
- void grid_graph_gen (T_xynGraph &grf, int n, int m)
- int bmw_graph_gen (T_xydGraph &dgrf, int ngrid, int n, int ntry=10, double lamda=0.3, double rou=0.3)

- template<class VDT , class WT1 >
  void grfw2grfw2 (const graph_type< VDT, WT1 > &gra0, graph_type< VDT, CEdgeW2< WT1, double > > &ngra)
- template<class VDT , class WT1 >
  void grfw2grfw2_s (const graph_type< VDT, WT1 > &gra0, graph_type< VDT, CEdgeW2< WT1, double > > &ngra)
- template<class GT >
  bool import_gitalt_ts (GT &gra, std::istream &is)
- template<class GT >
  void import_asconnect (GT &gra, std::istream &is)

stream input/output opertors

- std::ostream & **operator**<< (std::ostream &os, const Ce_nCare &edge)
- template<class WType1 , class WType2 >
  std::ostream & operator<< (std::ostream &os, const CEdgeW2< WType1, WType2 > &edge)
- template<class WType1 , class WType2 >
  std::istream & operator>> (std::istream &is, CEdgeW2< WType1, WType2 > &edge)
- template<class WType1 , class WType2 , class WType3 >
  std::ostream & operator<< (std::ostream &os, const CEdgeW3< WType1, WType2, WType3 > &edge)
- template<class WType1 , class WType2 , class WType3 >
  std::istream & operator>> (std::istream &is, CEdgeW3< WType1, WType2, WType3 > &edge)

stream input/output operators

- template<class EdgeDT >
  std::istream & operator>> (std::istream &is, CiEdge< EdgeDT > &edge)

path related utility functions

- std::ostream & operator<< (std::ostream &os, CPath &lp)
  output operator to output path as a list of indices separated by space.
- bool pred2path (int n, int *pnpred, int ns, int nt, CPath &lp)
- bool pred2path (const std::vector< int > &vpred, int ns, int nt, CPath &lp)
- bool allpred2path (int n, const std::vector< int > &vpred, int ns, int nt, CPath &lp)
- template<class GraphT , class Fun >
  Fun::result_type path_length (const GraphT &gra, const CPath &lp, Fun fw)

8.1.1 Detailed Description

A Framework for Graph (network) data structures and algorithms.

This file provides a number of type definitions to save some typing time. Before including any file that uses types defined in this file, make sure the header file(s) in which graph_type and/or rtree_type are defined is(are) included first. For example, graph.h provides graph_type and rtree.h provides rtree_type. This way, types defined here and other functions/clsses that use these tyepes will not be tied to a specific graph or rtree type.

8.2 news Namespace Reference

Classes

- class PktObj
- class SSEvent
- class SScheduler
- class SSNetObj
- class SSimulator
- struct CConstGrpSz
- struct CUniformGrpSz
- struct CGrpGenerator
- class CRandGrpX
- class CRandGrpG
- class CRandGrpG2
- class CWeightedGrpG

Typedefs

- typedef double TimeS

  timestamp
- typedef double TimeE

  elapsed time

Functions

- template<class BT >
  void rand_seq_gen (const BT &base, double alpha, int ns, int nt, std::list< std::pair< int, int > > &lact, std::set< int > &sdest)

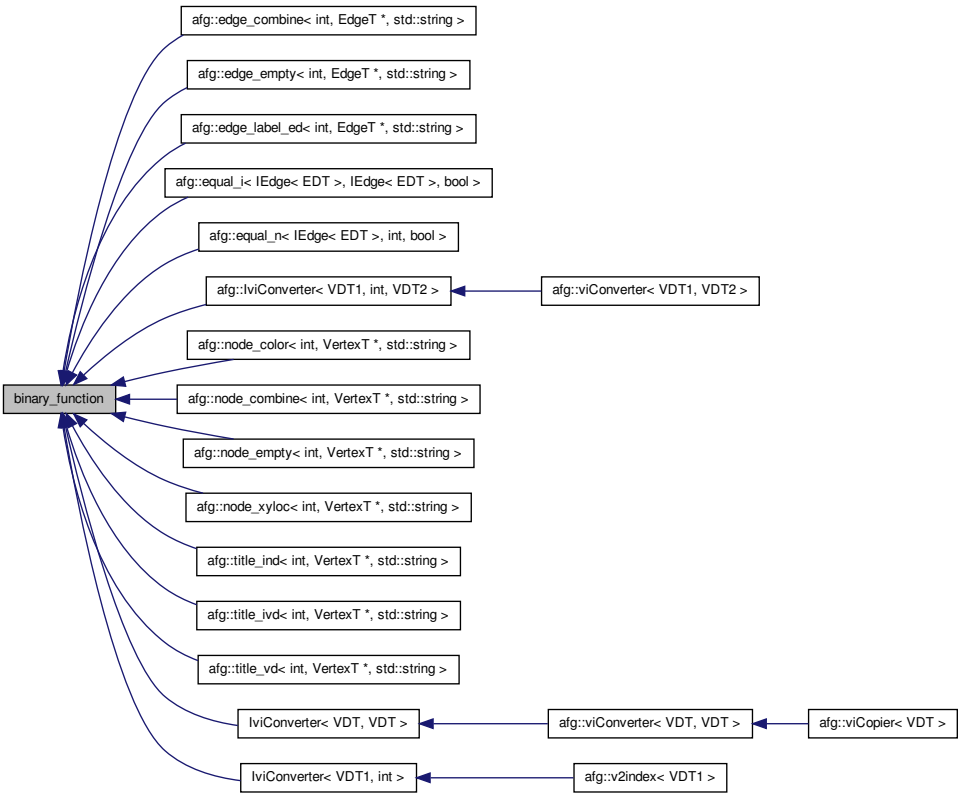
8.2.1 Detailed Description

a New nEtWork Simulator.

9 Class Documentation

9.1 binary_function Class Reference

Inheritance diagram for binary_function:



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

- graph_convert.h

9.2 afl::rand::CConst Struct Reference

```
#include <rand.hpp>
```

Public Member Functions

- CConst (double d)
- double operator() (void) const
- double gen_rand (void) const

Public Attributes

- double **m_dconst**

9.2.1 Detailed Description

function object to return a constant number.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 afl::rand::CConst::CConst (double *d*) [inline]

Constructor to init object with a double number that will be returned every time member function gen_rand() is called or operator() is called.

9.2.3 Member Function Documentation

9.2.3.1 double afl::rand::CConst::gen_rand (void) const [inline]

Function to return the constant number given at construction.

9.2.3.2 double afl::rand::CConst::operator() (void) const [inline]

Operator to return the constant number given at construction.

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

- rand.hpp

9.3 news::CConstGrpSz Struct Reference

`#include <grp_gen.h>`

Public Member Functions

- **CConstGrpSz** (int nsize)

- int **operator()** (void) const
- int **gen_size** (void) const

Public Attributes

- int **m_nSize**

9.3.1 Detailed Description

generate a constant group size.

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

- grp_gen.h

9.4 afg::Ce_nCare Struct Reference

```
#include <edge.h>
```

Public Member Functions

- template<class T >
  Ce_nCare (const T &)

9.4.1 Detailed Description

edge with nothing that we care about.

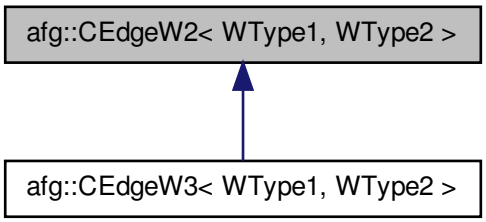
The documentation for this struct was generated from the following file:

- edge.h

9.5 afg::CEdgeW2< WType1, WType2 > Struct Template Reference

```
#include <edge.h>
```

Inheritance diagram for afg::CEdgeW2< WType1, WType2 >:



Public Types

- typedef WType1 WT1

  type of the 1st weight
- typedef WType2 WT2

  type of the 2nd weight

Public Member Functions

- CEdgeW2 (WType1 w1=WType1(), WType2 w2=WType2())

  constructor, both weights need to have default constructors.
- WType1 weight1 (void) const

  return the 1st weight
- WType2 weight2 (void) const

  return the 2nd weight

Public Attributes

- WType1 m_Weight1

  data member, first weight
- WType2 m_Weight2

  data member, second weight

9.5.1 Detailed Description

template<class WType1, class WType2>struct afg::CEdgeW2< WType1, WType2 >

edge with two weights

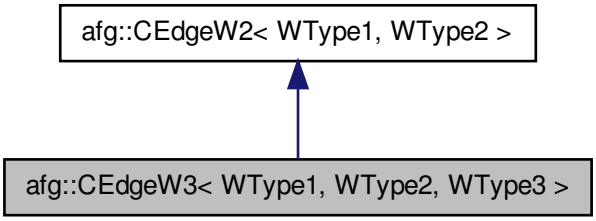
The documentation for this struct was generated from the following file:

- edge.h

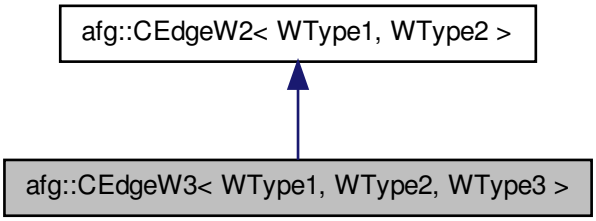
9.6 afg::CEdgeW3< WType1, WType2, WType3 > Struct Template Reference

`#include <edge.h>`

Inheritance diagram for afg::CEdgeW3< WType1, WType2, WType3 >:



Collaboration diagram for afg::CEdgeW3< WType1, WType2, WType3 >:



Public Types

- typedef WType3 WT3
  type of the 3rd weight

Public Member Functions

- CEdgeW3 (WType1 w1=WType1(), WType2 w2=WType2(), WType3 w3=WType3())
  constructor, all weights need to have default constructors.
- WType3 weight3 (void) const
  return the 3rd weight

Public Attributes

- WType3 m_Weight3
  data member, the 3rd weight

9.6.1 Detailed Description

template<class WType1, class WType2, class WType3>struct afg::CEdgeW3< WType1, WType2, WType3 >

edge with three weights

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

- edge.h

9.7 afl::rand::CExp Struct Reference

`#include <rand.hpp>`

Public Member Functions

- CExp (double davg)
- double operator() (void) const
- double gen_rand (void) const

Public Attributes

- double **m_avg**

9.7.1 Detailed Description

function object to generate exponentially distributed random numbers with a given average.

9.7.2 Constructor & Destructor Documentation

9.7.2.1 afl::rand::CExp::CExp (double *davg*) [inline]

Constructor to init object with the average.

Parameters

davg	the average; it has to be >0, otherwise it will be set to 1.0.

9.7.3 Member Function Documentation

9.7.3.1 double afl::rand::CExp::gen_rand (void) const [inline]

Function to return a random number

9.7.3.2 double afl::rand::CExp::operator() (void) const [inline]

Operator to return a random number

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

- rand.hpp

9.8 afg::CGraph< VertexDT, EdgeDT, f_eqv > Class Template Reference

`#include <graph.h>`

Inheritance diagram for afg::CGraph< VertexDT, EdgeDT, f_eqv >:

Collaboration diagram for afg::CGraph< VertexDT, EdgeDT, f_eqv >:

Public Types

type definitions.

- typedef EdgeDT EDT
  edge data type
- typedef VertexDT VDT
  vertex data type
- typedef CiEdge< EdgeDT > iET
  edge type
- typedef CiVertex< VertexDT, EdgeDT > iVT
  vertex type,
- typedef CGraph< VertexDT, EdgeDT, f_eqv > GT
  graph type, this class itself
- typedef f_eqv COMPT
  type compare function object to determine if two vertex data are equal
- typedef std::vector< CiVertex< VertexDT, EdgeDT > >::iterator iterator
- typedef std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator const_iterator
- typedef iVT::iterator e_iterator
  edge iterator
- typedef iVT::const_iterator const_e_iterator

edge iterator, const

- typedef IGraph< VertexDT, EdgeDT, f_eqv, typename std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, typename std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > > **igraph_base_type**

Public Member Functions

- CGraph (int size=1, const VertexDT &v=VertexDT(), double dg=0.25, const f_eqv &eqv=f_eqv())
- **CGraph** (const CGraph< VertexDT, EdgeDT, f_eqv > &rhs)
- const CGraph< VertexDT, EdgeDT, f_eqv > & **operator=** (const CGraph< VertexDT, EdgeDT, f_eqv > &rhs)
- virtual void copy_vertices (const GT &gra)

  copy all vertices from a graph
- virtual void reserve (int n)
- virtual int pack (int ns=0)

element access and helpers

- const VDT & v_default (void) const

  return the default vertex value
- virtual iterator begin (void)

  return iterator pointing to the first vertex
- virtual iterator end (void)

  return iterator pointing to the end (not pointing to any vertex).
- virtual const_iterator begin (void) const

  return iterator pointing to the first vertex
- virtual const_iterator end (void) const

  return iterator pointing to the end (not pointing to any vertex).
- virtual iVT & at (int i)
- virtual iVT & operator[] (int i)
- virtual const iVT & at (int i) const
- virtual const iVT & operator[] (int i) const
- EDT * get_edge (const VDT &u, const VDT &v)
- EDT * get_edge_byi (int nfrom, int nto)
- const EDT * get_edge (const VDT &u, const VDT &v) const
- const EDT * get_edge_byi (int nfrom, int nto) const
- virtual int find_index (const VDT &v) const

  find the index of vertex v
- virtual int size (void) const

  return graph size: number of vertices in the graph
- virtual int range (void) const
- virtual bool is_in_use (int i) const

  return if i (range checked) is an index used for a vertex.

insertion/deletion

- virtual int insert_v_qik (const VDT &v)
- virtual int insert_v (const VDT &v)
- virtual bool insert_v_ati (const VDT &v, int i)
- virtual void remove_v (const VDT &v)
- virtual void remove_v_byi (int vi)
- virtual bool insert_e (const VDT &u, const VDT &v, const EDT &e)
- virtual bool insert_2e (const VDT &u, const VDT &v, const EDT &e)
- virtual bool insert_e_byi (int i, int j, const EDT &e)
- virtual bool insert_2e_byi (int i, int j, const EDT &e)
- virtual void remove_e (const VDT &u, const VDT &v)
- virtual void remove_2e (const VDT &u, const VDT &v)
- virtual void remove_alle (const VDT &u)
- virtual void remove_e_byi (int nfrom, int nto)
- virtual void remove_2e_byi (int n1, int n2)
- virtual void remove_alle_byi (int nfrom)

  remove all edges of vertex of index nfrom
- virtual void remove_all_edges (void)

  remove all edges of all vertices
- virtual void clear (void)

  clear all nodes and all their edges

Protected Member Functions

- virtual int **grow** (int ns=0)

Protected Attributes

data members

- std::vector< CiVertex< VertexDT, EdgeDT > > m_Vertices

  data member, vertices
- VertexDT m_vDefault

  default vertex value
- int m_nSize

  size of the graph, which is the number of vertices
- double m_dGrow

  growth percentage when vertex vector is full
- afl::sorted_list< int > m_iUnused

  list of unused indices in m_Vertices

9.8.1 Detailed Description

template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>>class afg::CGraph< VertexDT, EdgeDT, f_eqv >

The graph class. A graph is represented as an adjacency list, naturally a directed graph. To store an un-directed graph, each edge will be kept as two copies. A vertex has "vertex data" of type VertexDT, which could be simple type (e.g., int, string) or a compound type (e.g., including x,y coordinates). An edge has "edge data" of type EdgeDT, which could be "simple" data type (int, double representing weight), or compound (two double number, or a weight plust a string as a "label"). See edge.h

In many comments, the words of vertex and node are used interchangeablly.

Internally vertices (adjacency list) are stored in a vector, for fast access. Many algorithms will use [index] to access vertices (with their edges) by index.

VertexDT: data type of vertex data, must have default constructor defined; a default value of VertexDT type must be provided for the constructor, which should be some value a "normal" vertex never has. EdgeDT: data type of edge data. f_eqv: a function object which compares two vertex data to see they are equal; by default, std::equal_to is used.

Author

Aiguo Fei

Version

0.5a, November/December 2000

9.8.2 Member Typedef Documentation

9.8.2.1 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> typedef std::vector<CiVertex<VertexDT, EdgeDT> >::const_iterator afg::CGraph< VertexDT, EdgeDT, f_eqv >::const_iterator

iterator type to access vertices. Only read access is allowed.

Reimplemented from afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.2.2 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> typedef std::vector<CiVertex<VertexDT, EdgeDT> >::iterator afg::CGraph< VertexDT, EdgeDT, f_eqv >::iterator

iterator type to access vertices.

Reimplemented from afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.3 Constructor & Destructor Documentation

9.8.3.1 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> afg::CGraph< VertexDT, EdgeDT, f_eqv >::CGraph (int *size* = 1, const VertexDT & *v* = VertexDT(), double *dg* = 0.25, const f_eqv & *eqv* = f_eqv()) [inline]

constructor

Parameters

v	a vertex data value that won't be taken by any vertex, internally used by CGraph to fill unused vertex table entries.
dg	growth percentage to grow a graph when graph needs to grow due to insertion of new vertices.
eqv	predicate to determine if two vertices are equivalent (having the same key or id); default std::equal_to is used.

9.8.4 Member Function Documentation

9.8.4.1 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual iVT& afg::CGraph< VertexDT, EdgeDT, f_eqv >::at (int *i*) [inline, virtual]

return vertex at position i

Exceptions

std::out_of_range.	

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.2 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual const iVT& afg::CGraph< VertexDT, EdgeDT, f_eqv >::at (int *i*) const [inline, virtual]

return vertex at position i, const version.

Exceptions

std::out_of_range.	

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.3 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> EDT* afg::CGraph< VertexDT, EdgeDT, f_eqv >::get_edge (const VDT & *u,* const VDT & *v*) [inline, virtual]

Get edge data (from u to v).

Returns

a pointer to that edge (data), NULL if not found.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.4 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> const EDT* afg::CGraph< VertexDT, EdgeDT, f_eqv >::get_edge (const VDT & *u,* const VDT & *v*) const [inline, virtual]

Get edge data (from u to v), const version.

Returns

a pointer to that edge (data), NULL if not found.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.5 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> EDT* afg::CGraph< VertexDT, EdgeDT, f_eqv >::get_edge_byi (int *nfrom,* int *nto*) [inline, virtual]

get edge data by index.

Returns

a pointer to that edge (data), NULL if not found.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.6 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> const EDT* afg::CGraph< VertexDT, EdgeDT, f_eqv >::get_edge_byi (int *nfrom,* int *nto*) const [inline, virtual]

get edge data by index.

Returns

a pointer to that edge (data), NULL if not found.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.7 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual bool afg::CGraph< VertexDT, EdgeDT, f_eqv >::insert_2e (const VDT & *u,* const VDT & *v,* const EDT & *e*) [inline, virtual]

insert both (u->v) and (v->u). The same edge data is used for both edges. Equivalent to "insert_e(u,v,e); insert_e(v,u,e);".

See also

insert_e( )

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.8 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual bool afg::CGraph< VertexDT, EdgeDT, f_eqv >::insert_2e_byi (int *i,* int *j,* const EDT & *e*) [inline, virtual]

insert both (i->j) and (j->i). The same edge data is used for both edges. Equivalent to "insert_e(i,j,e); insert_e(j,i,e);".

See also

insert_e_byi( )

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.9 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual bool afg::CGraph< VertexDT, EdgeDT, f_eqv >::insert_e (const VDT & *u,* const VDT & *v,* const EDT & *e*) [inline, virtual]

insert an edge for vertex u to v. Insertion fails if any of the vertex is not found. If edge already exists then edge data will be replaced by e.

Returns

true if succeeds, false otherwise.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

Reimplemented in afg::CrTree< VDT, EDT, f_eqv >.

9.8.4.10 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual bool afg::CGraph< VertexDT, EdgeDT, f_eqv >::insert_e_byi (int *i*, int *j*, const EDT & *e*)** `[inline, virtual]`

insert an edge (i->j). Insertion fails if any of the two indices is unused (not vertext at that position).

Returns

true if succeeds, otherwise false.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

Reimplemented in afg::CrTree< VDT, EDT, f_eqv >.

9.8.4.11 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual int afg::CGraph< VertexDT, EdgeDT, f_eqv >::insert_v (const VDT & *v*)** `[inline, virtual]`

insert a vertex of value v. Vertex data is updated if vertex (v) already exists.

Returns

index of the vertex in vertex table, -1 (an invalid index) to indicate failure.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.12 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual bool afg::CGraph< VertexDT, EdgeDT, f_eqv >::insert_v_ati (const VDT & *v*, int *i*)** `[inline, virtual]`

insert a vertex of value v at position i. This allows a graph user to control vertex of each position. Existing vertex at position i (if exist) will be replaced by v, any existing edge will be lost. However, any other edge pointing to this vertex will still be there.

Returns

true if insertion succeeds; false if fails (table is smaller than i and fails to grow).

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.13 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>>**
virtual int afg::CGraph< VertexDT, EdgeDT, f_eqv >::insert_v_qik (const VDT & *v*)
[inline, virtual]

insert a vertex of value v. Quick version that doesn't search for existing vertex of the same value -- use this if it is known that the new vertex doesn't exist.

Returns

index of the vertex in vertex table, -1 (an invalid index) to indicate failure.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.14 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>>**
virtual iVT& afg::CGraph< VertexDT, EdgeDT, f_eqv >::operator[] (int *i*)
[inline, virtual]

return vertex at position i

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.15 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>>**
virtual const iVT& afg::CGraph< VertexDT, EdgeDT, f_eqv >::operator[] (int *i*) const [inline, virtual]

return vertex at position i, const version.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.16 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>>**
virtual int afg::CGraph< VertexDT, EdgeDT, f_eqv >::pack (int *ns* = 0)
[inline, virtual]

rearrange vertices to make them stored consecutively. After pack( ), all iterators and indices may be invalidated. All vertices will be stored within [0, size()-1].

Parameters

ns	new capacity; if ns<size( ), size() will be used.

Returns

number of index changes that have been made.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

Reimplemented in afg::CrTree< VDT, EDT, f_eqv >.

9.8.4.17 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual int afg::CGraph< VertexDT, EdgeDT, f_eqv >::range (void) const** [inline, virtual]

return current upper bound on vertex index range. Vertex index will be within [0, range()-1], will always have range()>=size()

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.18 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual void afg::CGraph< VertexDT, EdgeDT, f_eqv >::remove_2e (const VDT & *u*, const VDT & *v*)** [inline, virtual]

remove both (u->v) and (v->u). Equivalent to "remove_e(u,v); remove_e(v,u);". Has no effect if edge not found.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.19 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual void afg::CGraph< VertexDT, EdgeDT, f_eqv >::remove_2e_byi (int *n1*, int *n2*)** [inline, virtual]

remove both edges by index. Has no effect if edge not found.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.20 **template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual void afg::CGraph< VertexDT, EdgeDT, f_eqv >::remove_alle (const VDT & *u*)** [inline, virtual]

remove all edges of vertex v. Has no effect if vertex not found.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.21 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual void afg::CGraph< VertexDT, EdgeDT, f_eqv >::remove_e (const VDT & *u*, const VDT & *v*) [inline, virtual]

remove edge (u->v). Has no effect if edge not found.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.22 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual void afg::CGraph< VertexDT, EdgeDT, f_eqv >::remove_e_byi (int *nfrom*, int *nto*) [inline, virtual]

remove an edge by index. Has no effect if edge not found.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

Reimplemented in afg::CrTree< VDT, EDT, f_eqv >.

9.8.4.23 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual void afg::CGraph< VertexDT, EdgeDT, f_eqv >::remove_v (const VDT & *v*) [inline, virtual]

remove a vertex of value v. Has no effect if vertex not found; otherwise this vertex and all related edges (from/to it) will be removed as well.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

9.8.4.24 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>> virtual void afg::CGraph< VertexDT, EdgeDT, f_eqv >::remove_v_byi (int *vi*) [inline, virtual]

remove a vertex of index vi. Has no effect if vertex not found; otherwise this vertex and all related edges (from/to it) will be removed as well.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

Reimplemented in afg::CrTree< VDT, EDT, f_eqv >.

9.8.4.25 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>>
virtual void afg::CGraph< VertexDT, EdgeDT, f_eqv >::reserve (int *n*)
[inline, virtual]

reserve a certain size for the graph. Has no effect if n is less than or equal to range( ); otherwise capacity will be increased and all iteroators may be invalidated, but an index should still pointing to the same vertex.

Implements afg::IGraph< VertexDT, EdgeDT, f_eqv, std::vector< CiVertex< VertexDT, EdgeDT > >::iterator, std::vector< CiVertex< VertexDT, EdgeDT > >::const_iterator, CiVertex< VertexDT, EdgeDT > >.

Reimplemented in afg::CrTree< VDT, EDT, f_eqv >.

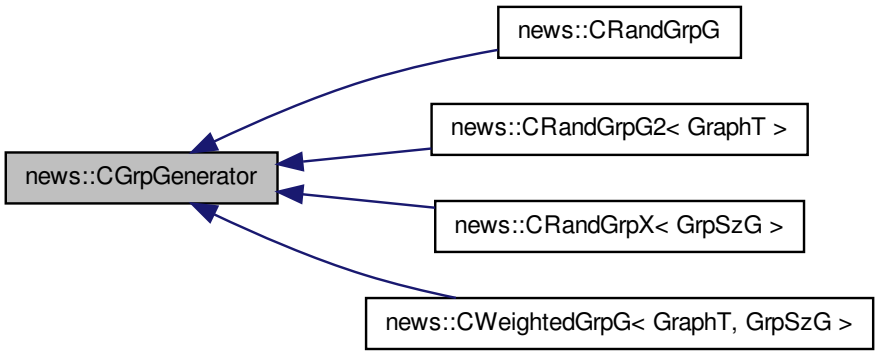
The documentation for this class was generated from the following file:

- graph.h

9.9 news::CGrpGenerator Struct Reference

`#include <grp_gen.h>`

Inheritance diagram for news::CGrpGenerator:



Public Member Functions

- virtual int gen_grp (int &ns, std::set< int > &smem)=0

- int operator() (int &ns, std::set< int > &smem)

9.9.1 Detailed Description

interface of group generator.

9.9.2 Member Function Documentation

9.9.2.1 virtual int news::CGrpGenerator::gen_grp (int & *ns*, std::set< int > & *smem*) [pure virtual]

generate a group.

Parameters

ns	source node of generated group
smem	member nodes (indices) of generated group

Returns

number of nodes in group, 0 means failed.

Implemented in news::CRandGrpX< GrpSzG >, news::CRandGrpG, news::CRandGrpG2< GraphT >, and news::CWeightedGrpG< GraphT, GrpSzG >.

9.9.2.2 int news::CGrpGenerator::operator() (int & *ns*, std::set< int > & *smem*) [inline]

operator () -- so this class can be used as a function object.

See also

gen_grp

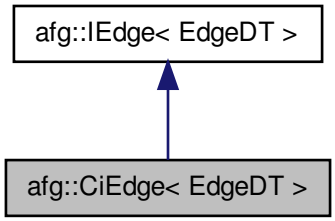
The documentation for this struct was generated from the following file:

- grp_gen.h

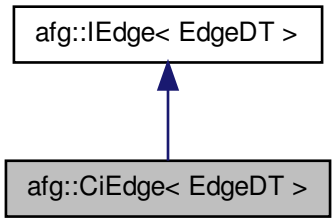
9.10 afg::CiEdge< EdgeDT > Class Template Reference

```
#include <graph.h>
```

Inheritance diagram for afg::CiEdge< EdgeDT >:



Collaboration diagram for afg::CiEdge< EdgeDT >:



Public Types

- typedef EdgeDT EDT
  type of the edge data

Public Member Functions

- CiEdge (int nto=-1, const EdgeDT &edg=EdgeDT())
  constructor

- int to (void) const

  return destination node (index)
- int & to (void)

  return reference of destination node (index)
- EdgeDT & edge_d (void)

  return reference of edge data
- const EdgeDT & edge_d (void) const

  return reference of edge data, const version

Protected Attributes

- int m_nTo

  data member, destination node (index)
- EdgeDT m_EdgeD

  data member, edge data

9.10.1 Detailed Description

template<class EdgeDT>class afg::CiEdge< EdgeDT >

edge

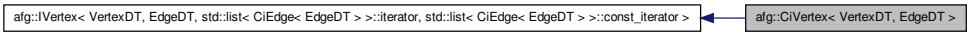
The documentation for this class was generated from the following file:

- graph.h

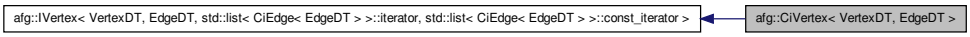
9.11 afg::CiVertex< VertexDT, EdgeDT > Class Template Reference

`#include <graph.h>`

Inheritance diagram for afg::CiVertex< VertexDT, EdgeDT >:



Collaboration diagram for afg::CiVertex< VertexDT, EdgeDT >:



Public Types

- typedef CiEdge< EdgeDT > iET

  edge type
- typedef CiVertex< VertexDT, EdgeDT > iVT

  vertex type, this class itself
- typedef EdgeDT EDT

  edge data type
- typedef VertexDT VDT

  vertex data type
- typedef std::list< CiEdge< EdgeDT > >::iterator iterator
- typedef std::list< CiEdge< EdgeDT > >::const_iterator const_iterator

Public Member Functions

- CiVertex (const VertexDT &v=VertexDT())

  constructor.
- **CiVertex** (const CiVertex< VertexDT, EdgeDT > &rhs)
- const CiVertex< VertexDT, EdgeDT > & operator= (const CiVertex< VertexDT, EdgeDT > &rhs)

  assignment operator
- virtual VertexDT & vertex_d (void)

  return a reference of the vertex data
- virtual const VertexDT & vertex_d (void) const

  return a reference of the vertex data, const version
- virtual iterator begin (void)

  return iterator pointing to the first edge
- virtual iterator end (void)

  return iterator pointing to the end (not pointting to any endge).
- virtual const_iterator begin (void) const

  return iterator pointing to the first edge
- virtual const_iterator end (void) const

return iterator pointing to the end (not pointting to any endge).

- virtual EdgeDT * get_edge (int nto)
- virtual const EdgeDT * get_edge (int nto) const
- virtual int out_degree (void) const
- void insert_edge (int nto, EdgeDT edge)
- void remove_edge (int nto)
- void remove_all (void)

Protected Attributes

- VertexDT m_VertexD

  data member, vertex data
- std::list< CiEdge< EdgeDT > > m_lEdges

  data member, a list of edges.

9.11.1 Detailed Description

template<class VertexDT, class EdgeDT>class afg::CiVertex< VertexDT, EdgeDT >

internal vertex in our graph representation.

Author

Aiguo Fei

Version

0.8a, revision November 2000

9.11.2 Member Typedef Documentation

9.11.2.1 template<class VertexDT, class EdgeDT> typedef std::list<CiEdge<EdgeDT> >::const_iterator afg::CiVertex< VertexDT, EdgeDT >::const_iterator

const iterator type to access edges (read-only).

Reimplemented from afg::IVertex< VertexDT, EdgeDT, std::list< CiEdge< EdgeDT > >::iterator, std::list< CiEdge< EdgeDT > >::const_iterator >.

9.11.2.2 template<class VertexDT, class EdgeDT> typedef std::list<CiEdge<EdgeDT> >::iterator afg::CiVertex< VertexDT, EdgeDT >::iterator

iterator type to access edges.

Reimplemented from afg::IVertex< VertexDT, EdgeDT, std::list< CiEdge< EdgeDT > >::iterator, std::list< CiEdge< EdgeDT > >::const_iterator >.

9.11.3 Member Function Documentation

9.11.3.1 template<class VertexDT, class EdgeDT> virtual EdgeDT* afg::CiVertex< VertexDT, EdgeDT >::get_edge (int *nto*) [inline, virtual]

get an edge of this vertex

Parameters

nto	destination node (index)

Returns

a pointer to the edge data, NULL if edge doesn't exist

Implements afg::IVertex< VertexDT, EdgeDT, std::list< CiEdge< EdgeDT > >::iterator, std::list< CiEdge< EdgeDT > >::const_iterator >.

9.11.3.2 template<class VertexDT, class EdgeDT> virtual const EdgeDT* afg::CiVertex< VertexDT, EdgeDT >::get_edge (int *nto*) const [inline, virtual]

get an edge of this vertex, const version

Parameters

nto	destination node (index)

Returns

a pointer to the edge data, NULL if edge doesn't exist

Implements afg::IVertex< VertexDT, EdgeDT, std::list< CiEdge< EdgeDT > >::iterator, std::list< CiEdge< EdgeDT > >::const_iterator >.

9.11.3.3 template<class VertexDT, class EdgeDT> void afg::CiVertex< VertexDT, EdgeDT >::insert_edge (int *nto*, EdgeDT *edge*) [inline]

insert an edge for this vertex. If to the given destination node already exists, old edge data will be replaced.

Parameters

nto	destination node (index)
edge	edge data

9.11.3.4 template<class VertexDT, class EdgeDT> virtual int afg::CiVertex< VertexDT, EdgeDT >::out_degree (void) const [inline, virtual]

get the out degree of this vertex.

Returns

out degree

Implements afg::IVertex< VertexDT, EdgeDT, std::list< CiEdge< EdgeDT > >::iterator, std::list< CiEdge< EdgeDT > >::const_iterator >.

9.11.3.5 template<class VertexDT, class EdgeDT> void afg::CiVertex< VertexDT, EdgeDT >::remove_all (void) [inline]

remove all edges for this vertex.

9.11.3.6 template<class VertexDT, class EdgeDT> void afg::CiVertex< VertexDT, EdgeDT >::remove_edge (int *nto*) [inline]

remove an edge for this vertex. Has no effect if the given edge doesn't exist.

Parameters

nto	destination node (index)

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

- graph.h

9.12 afg::CkthSP< GraphT, Fun > Class Template Reference

`#include <kthsp.h>`

Public Types

- typedef Fun::result_type **WT**

Public Member Functions

- CkthSP (const GraphT &thegra, int source, int k, Fun fw, typename Fun::result_type winfty)
- void get_first ()
- void get_next ()

  call this one to compute the next shortest path (from ith to (i+1)th).

- WT get_path (int nt, int nk, CPath &lp)
- bool increase_k (int newk)

  increase k value to newk, no effect if newk<=old k

9.12.1 Detailed Description

template<class GraphT, class Fun>class afg::CkthSP< GraphT, Fun >

the kth shortest-path algorithm. This class is used to compute the shortest path, the 2nd shortest path, the 3rd shortest path, until the kth shortest path, from a given source to all destinations.

Algorithm description: see "Graphs and Algorithms" by Michel Gondran and Michel Minoux, pp.63, John Wiley & Sons, 1984

How to use: after declare an instance, call get_first( ) to compute the shortest path and it will do some initialization at the same time; now the 1st shorest path is available for retrival by calling get_path(, 1,). To get the ith shortest path, have to call get_next( ) (i-1) times (after calling get_first( )). If want to go beyond the initial k, call increase_k( ) to specify a new k value.

9.12.2 Constructor & Destructor Documentation

9.12.2.1 template<class GraphT , class Fun > afg::CkthSP< GraphT, Fun >::CkthSP (const GraphT & *thegra,* int *source,* int *k,* Fun *fw,* typename Fun::result_type *winfty*) [inline]

Parameters

thegra	graph on which to run the algorithm; a reference to that graph is kept internally, reference graph can't change before running to the ith path wanted (i<=k).
source	source node
k	desired kth
fw	function object used to get edge weight, pass pointer
winfty	an "upper bound" value of weight which should be larger than the path length of any valid path; used to initialize distance to all nodes.

9.12.3 Member Function Documentation

9.12.3.1 template<class GraphT , class Fun > void afg::CkthSP< GraphT, Fun >::get_first ( ) [inline]

get the first shortest path. This should be the first function to call after constructor.

9.12.3.2 template<class GraphT , class Fun > WT afg::CkthSP< GraphT, Fun >::get_path (int *nt,* int *nk,* CPath & *lp*) [inline]

return the ith shortest path to node nt. get_next( ) should have been already called (nk-1) times.

Parameters

nt	destination node
nk	the nkth shortest-path to nt
lp	the path retrieved

Returns

the total "weight" (distance) of the path

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

- kthsp.h

9.13 afg::CnmcBase< GT, WT1, WT2, WT3 > Class Template Reference

`#include <nmcbase.h>`

Public Member Functions

- CnmcBase (const GT &gra, double dmax=10000)
- **CnmcBase** (const CnmcBase< GT, WT1, WT2, WT3 > &rhs)
- const CnmcBase & **operator=** (const CnmcBase< GT, WT1, WT2, WT3 > &rhs)

- int **size** (void) const
- WT1 **w1** (int i, int j) const
- WT2 **w2** (int i, int j) const
- WT3 **w3** (int i, int j) const
- bool **can_be_member** (int i) const
- bool **mark** (int i) const
- void **set_mark** (int i, bool b=false)
- bool **get_path** (int ns, int nt, afg::CPath &lp) const
- template<class Fun1 , class Fun2 , class Fun3 >
  void init (Fun1 fw1, Fun2 fw2, Fun3 fw3)
- bool check (void) const

Public Attributes

- const GT & **m_gra**
- int **m_nSize**
- std::vector< WT1 > **m_vW1**
- std::vector< WT2 > **m_vW2**
- std::vector< WT3 > **m_vW3**
- std::vector< int > **m_vPred**
- std::vector< bool > **m_vMarks**
- double **m_dMax**

Friends

- std::ostream & **operator**<< (std::ostream &os, const CnmcBase< GT, WT1, WT2, WT3 > &tmg)

9.13.1 Detailed Description

template<class GT, class WT1 = int, class WT2 = double, class WT3 = int>class afg::CnmcBase< GT, WT1, WT2, WT3 >

base class for network multicast. Essentially provides a shortest-path "routing" table that can be used to build various multicast trees, along with two additional characteristics on the shortest-paths. 1st weight is the underlying unicast routing weight (shortest path); 2nd weight is an end-to-end measure that can be used for TM routing; 3rd weight is another end-to-end measure. 1st one most likely to be assigned link cost, 2nd/3rd one can be: end-to-end delay, hop-count.

After declare a CnmcBase, first call init() to initialize the distance and predecessor tables, then call check() to check if there is any unreachable nodes. Call set_mark() to mark any nodes that shouldn't be in a multicast group if necessary.

9.13.2 Constructor & Destructor Documentation

9.13.2.1 template<class GT, class WT1 = int, class WT2 = double, class WT3 = int> afg::CnmcBase< GT, WT1, WT2, WT3 >::CnmcBase (const GT & *gra,* double *dmax* = 10000) [inline]

Parameters

gra	the graph, should be packed before passed.

9.13.3 Member Function Documentation

9.13.3.1 template<class GT, class WT1 = int, class WT2 = double, class WT3 = int> bool afg::CnmcBase< GT, WT1, WT2, WT3 >::check (void) const [inline]

check to see if there is any unreachable nodes or inter-node distances that are irregular.

Returns

true if no such node(s), false otherwise

9.13.3.2 template<class GT, class WT1 = int, class WT2 = double, class WT3 = int> template<class Fun1 , class Fun2 , class Fun3 > void afg::CnmcBase< GT, WT1, WT2, WT3 >::init (Fun1 *fw1,* Fun2 *fw2,* Fun3 *fw3*) [inline]

initialization. Initialize the inter-member distance(weight) and predecessor tables. 1st weight is the underlying unicast routing weight (shortest path), 2nd weight is an end-to-end measure that can be used for TM routing, 3rd weight is another end-to-end measure.

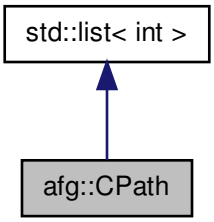
The documentation for this class was generated from the following file:

- nmcbase.h

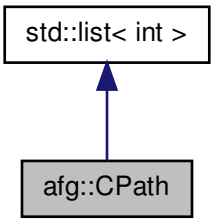
9.14 afg::CPath Class Reference

`#include <path.h>`

Inheritance diagram for afg::CPath:



Collaboration diagram for afg::CPath:



Public Member Functions

- **CPath** (int i)
- CPath (int nsource, int ndest, const std::vector< int > &vpred)
- int source (void) const

  get the source node of the path
- int dest (void) const

  get the destination node of the path
- bool insert_before (int i, int n)
- bool insert_after (int i, int n)
- template<class GraphT , class Fun >
  std::ostream & output (std::ostream &os, GraphT &gra, Fun fv, const std::string &connector=" -> ")

  output path as a list of vertex (value) connected by a given connector.

9.14.1 Detailed Description

helper class for path manipulation. A path is list of integers representing indices in a graph.

See also

IGraph

9.14.2 Constructor & Destructor Documentation

9.14.2.1 afg::CPath::CPath (int *nsource,* int *ndest,* const std::vector< int > & *vpred*) [inline]

construct a path from nsource to ndest from a predecessor vector.

Parameters

nsource	source node
ndest	destination node
vpred	predecessor index vector. vpred[i] is the node that comes just before i on the shortest path from nsource to ndest. vpred should be generated from some shortest-path algorithm such as Dijkstras's.

9.14.3 Member Function Documentation

9.14.3.1 bool afg::CPath::insert_after (int *i,* int *n*) [inline]

insert i after n

Returns

ture if n found, false otherwise

9.14.3.2 bool afg::CPath::insert_before (int *i,* int *n*) [inline]

insert i before n

Returns

ture if n found, false otherwise

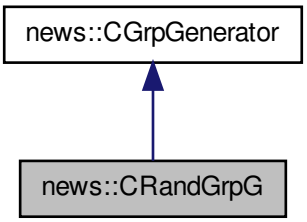
The documentation for this class was generated from the following file:

- path.h

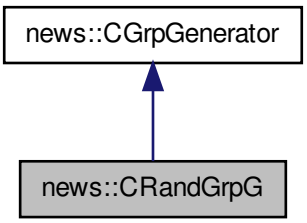
9.15 news::CRandGrpG Class Reference

```
#include <grp_gen.h>
```

Inheritance diagram for news::CRandGrpG:



Collaboration diagram for news::CRandGrpG:



Public Member Functions

- CRandGrpG (double dp, int nmax, int nmin=2)
- int gen_grp (int &ns, std::set< int > &smem)

9.15.1 Detailed Description

generate random groups. A group is generated by randomly picking up nodes of indices between 0 and a given upper bound according to certain probability. Group size is

enforced to be larger than a minimum.

9.15.2 Constructor & Destructor Documentation

9.15.2.1 news::CRandGrpG::CRandGrpG (double *dp*, int *nmax*, int *nmin* = 2) [inline]

Parameters

dp	probability according to which to pick up nodes
nmax	upper bound on node index allowed, inclusive
nmin	minimum group size

9.15.3 Member Function Documentation

9.15.3.1 int news::CRandGrpG::gen_grp (int & *ns*, std::set< int > & *smem*) [inline, virtual]

generate a group.

Parameters

ns	source node of generated group
smem	member nodes (indices) of generated group

Returns

number of nodes in group, 0 means failed.

Implements news::CGrpGenerator.

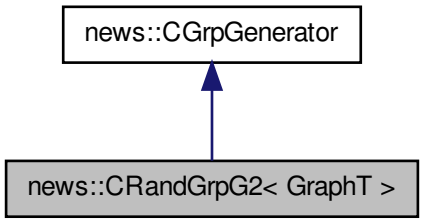
The documentation for this class was generated from the following file:

- grp_gen.h

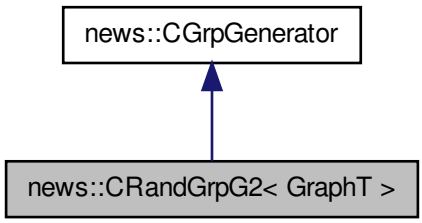
9.16 news::CRandGrpG2< GraphT > Class Template Reference

```
#include <grp_gen.h>
```

Inheritance diagram for news::CRandGrpG2< GraphT >:



Collaboration diagram for news::CRandGrpG2< GraphT >:



Public Member Functions

- **CRandGrpG2** (const GraphT &gra, double dp, int n1=2)
- int gen_grp (int &ns, std::set< int > &smem)

9.16.1 Detailed Description

template<class GraphT>class news::CRandGrpG2< GraphT >

generate random groups. A group is generated by randomly picking up nodes of indices from a given graph according a certain probability. Group size is enforced to be larger than a minimum. An index is checked to make sure it is a valid one in the graph before it is picked-- this makes it possible to generate groups for a graph having "holes" in its index range; for graph without such "holes", use CRandGrpG instead (specifying group size as the upper bound on index).

9.16.2 Member Function Documentation

9.16.2.1 template<class GraphT > int news::CRandGrpG2< GraphT >::gen_grp (int & *ns*, std::set< int > & *smem*) [inline, virtual]

generate a group.

Parameters

ns	source node of generated group
smem	member nodes (indices) of generated group

Returns

number of nodes in group, 0 means failed.

Implements news::CGrpGenerator.

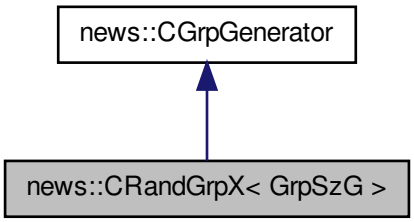
The documentation for this class was generated from the following file:

- grp_gen.h

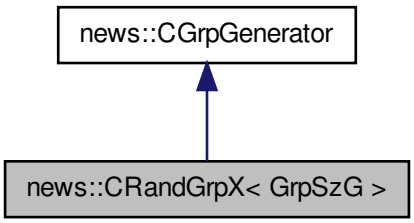
9.17 news::CRandGrpX< GrpSzG > Class Template Reference

```
#include <grp_gen.h>
```

Inheritance diagram for news::CRandGrpX< GrpSzG >:



Collaboration diagram for news::CRandGrpX< GrpSzG >:



Public Member Functions

- CRandGrpX (int nmax, const GrpSzG &gsg)
- int gen_grp (int &ns, std::set< int > &smem)

9.17.1 Detailed Description

template<class GrpSzG>class news::CRandGrpX< GrpSzG >

generate a random group of size generated by a group size generator. GrpSzG: group size generator. A group is generated by randomly picking up certain number (size) of nodes of indices between 0 and the upper bound. The size is generated by the group size generator.

9.17.2 Constructor & Destructor Documentation

9.17.2.1 template<class GrpSzG > news::CRandGrpX< GrpSzG >::CRandGrpX (int *nmax,* const GrpSzG & *gsg*) [inline]

constructor.

Parameters

nmax	upper bound of node index allowed in the group, i.e, any node in the group is going to have an index from 0 to nmax (inclusive).
gsg	group size generator

9.17.3 Member Function Documentation

9.17.3.1 template<class GrpSzG > int news::CRandGrpX< GrpSzG >::gen_grp (int & *ns,* std::set< int > & *smem*) [inline, virtual]

generate a group.

Parameters

ns	source node of generated group
smem	member nodes (indices) of generated group

Returns

number of nodes in group, 0 means failed.

Implements news::CGrpGenerator.

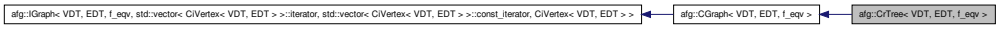
The documentation for this class was generated from the following file:

- grp_gen.h

9.18 afg::CrTree< VDT, EDT, f_eqv > Class Template Reference

```
#include <rtree.h>
```

Inheritance diagram for afg::CrTree< VDT, EDT, f_eqv >:



Collaboration diagram for afg::CrTree< VDT, EDT, f_eqv >:

Public Types

- typedef CrTree< VDT, EDT, f_eqv > TT

  tree type, this class itself
- typedef CGraph< VDT, EDT, f_eqv > **base_graph_t**

Public Member Functions

- CrTree (int size=1, VDT v=VDT(), double dg=0.25, const f_eqv &eqv=f_eqv())

  constructor.
- **CrTree** (const TT &rhs)
- const TT & **operator=** (const TT &rhs)
- virtual void copy_vertices (const base_graph_t &gra)

  copy all vertices from a graph
- int parent (const VDT &v) const

  return parent node of node v
- int parent_byi (int i) const throw (std::out_of_range)
- int root (void) const

  return the index of the root node
- int & root (void)

  return the index of the root node, by reference
- bool set_root (const VDT &v)

  set the root node
- bool set_root_byi (int n)

  set the root node by index
- int first_root (void) const

  find the first node which doesn't have a parent

- bool **is_member** (int i) const
- bool **is_leaf** (int i) const
- virtual std::ostream & output (std::ostream &os) const

  output graph to a stream
- virtual void reserve (int n)
- virtual int pack (int ns=0)
- int get_all_children (int inode, std::set< int > &sch)

insertion/deletion

- virtual void remove_v_byi (int vi)
- virtual int clean (void)
- virtual bool insert_e (const VDT &u, const VDT &v, const EDT &e)
- virtual bool insert_e_byi (int i, int j, const EDT &e)
- virtual void remove_e_byi (int nfrom, int nto)
- virtual void remove_alle_byi (int nfrom)

  remove all edges of vertex of index nfrom
- virtual void remove_all_edges (void)

  remove all edges of all vertices
- virtual void clear (void)

  clear all nodes and all their edges

Protected Member Functions

- virtual int **grow** (int ns=0)

Protected Attributes

- int **m_nRoot**
- std::vector< int > **m_Parents**

9.18.1 Detailed Description

template<class VDT, class EDT, class f_eqv = std::equal_to<VDT>>class afg::CrTree< VDT, EDT, f_eqv >

rooted tree. Tree structure that has a specific root.

9.18.2 Member Function Documentation

9.18.2.1 template<class VDT , class EDT , class f_eqv = std::equal_to<VDT>> virtual int afg::CrTree< VDT, EDT, f_eqv >::clean (void) [inline, virtual]

clean-up the "tree" by removing "isolated" nodes. An "isolated" is a node that is not any node's parent or child; in other words, has no neighbors.

Returns

number of nodes that are removed.

9.18.2.2 template<class VDT , class EDT , class f_eqv = std::equal_to<VDT>> int afg::CrTree< VDT, EDT, f_eqv >::get_all_children (int *inode*, std::set< int > & *sch*) [inline]

get all child nodes of a specific node given by index.

Parameters

inode	index the specific node
sch	set to hold indices of the children

Returns

number of children found.

9.18.2.3 template<class VDT , class EDT , class f_eqv = std::equal_to<VDT>> virtual bool afg::CrTree< VDT, EDT, f_eqv >::insert_e (const VDT & *u*, const VDT & *v*, const EDT & *e*) [inline, virtual]

insert an edge for vertex u to v. Insertion fails if any of the vertex is not found. If edge already exists then edge data will be replaced by e.

Returns

true if succeeds, false otherwise.

Reimplemented from afg::CGraph< VDT, EDT, f_eqv >.

9.18.2.4 template<class VDT , class EDT , class f_eqv = std::equal_to<VDT>> virtual bool afg::CrTree< VDT, EDT, f_eqv >::insert_e_byi (int *i*, int *j*, const EDT & *e*) [inline, virtual]

insert an edge (i->j). Insertion fails if any of the two indices is unused (not vertext at that position).

Returns

true if succeeds, otherwise false.

Reimplemented from afg::CGraph< VDT, EDT, f_eqv >.

9.18.2.5 template<class VDT , class EDT , class f_eqv = std::equal_to<VDT>> virtual int afg::CrTree< VDT, EDT, f_eqv >::pack (int *ns* = 0) [inline, virtual]

rearrange vertices to make them stored consecutively. After pack( ), all iterators and indices may be invalidated. All vertices will be stored within [0, size()-1].

Parameters

ns	new range; if ns<size( ), size() will be used.

Returns

number of index changes that have been made.

Reimplemented from afg::CGraph< VDT, EDT, f_eqv >.

9.18.2.6 template<class VDT , class EDT , class f_eqv = std::equal_to<VDT>> int afg::CrTree< VDT, EDT, f_eqv >::parent_byi (int *i*) const throw (std::out_of_range) [inline]

return parent node of node of index i

Exceptions

std::out_of_range,with	message "parent node access in tree".

9.18.2.7 template<class VDT , class EDT , class f_eqv = std::equal_to<VDT>> virtual void afg::CrTree< VDT, EDT, f_eqv >::remove_e_byi (int *nfrom,* int *nto*) [inline, virtual]

remove an edge by index. Has no effect if edge not found.

Reimplemented from afg::CGraph< VDT, EDT, f_eqv >.

9.18.2.8 template<class VDT , class EDT , class f_eqv = std::equal_to<VDT>> virtual void afg::CrTree< VDT, EDT, f_eqv >::remove_v_byi (int *vi*) [inline, virtual]

remove a vertex of index vi. Has no effect if vertex not found; otherwise this vertex and all related edges (from/to it) will be removed as well.

Reimplemented from afg::CGraph< VDT, EDT, f_eqv >.

9.18.2.9 template<class VDT , class EDT , class f_eqv = std::equal_to<VDT>> virtual void afg::CrTree< VDT, EDT, f_eqv >::reserve (int *n*) [inline, virtual]

reserve a certain size for the tree. Has no effect if n is less than or equal to range( ); otherwise capacity will be increased and all iteroators may be invalidated, but an index should still pointing to the same vertex.

Reimplemented from afg::CGraph< VDT, EDT, f_eqv >.

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

- rtree.h

9.19 afl::rand::CUniform Struct Reference

`#include <rand.hpp>`

Public Member Functions

- CUniform (double dmax, double dmin=0)
- double operator() (void) const
- double gen_rand (void) const

Public Attributes

- double **m_maxt**
- double **m_mint**

9.19.1 Detailed Description

function object to generate uniformly distributed random numbers within a range.

9.19.2 Constructor & Destructor Documentation

9.19.2.1 afl::rand::CUniform::CUniform (double *dmax*, double *dmin* = 0) [inline]

Constructor to init object with a range within which to generate random numbers.

9.19.3 Member Function Documentation

9.19.3.1 double afl::rand::CUniform::gen_rand (void) const [inline]

Function to return a random number within the given range.

9.19.3.2 double afl::rand::CUniform::operator() (void) const [inline]

Operator to return a random number within the given range.

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

- rand.hpp

9.20 news::CUniformGrpSz Struct Reference

`#include <grp_gen.h>`

Public Member Functions

- **CUniformGrpSz** (int nmax, int nmin=2)
- int **operator()** (void) const
- int **gen_size** (void) const

Public Attributes

- int **m_nmin**
- int **m_nmax**

9.20.1 Detailed Description

generate uniformly distributed group size, from nmin to nmax.

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

- grp_gen.h

9.21 afg::Cvoid Struct Reference

`#include <graph_intf.h>`

9.21.1 Detailed Description

a struct that has nothing.

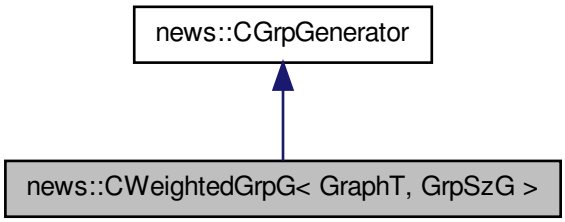
The documentation for this struct was generated from the following file:

- graph_intf.h

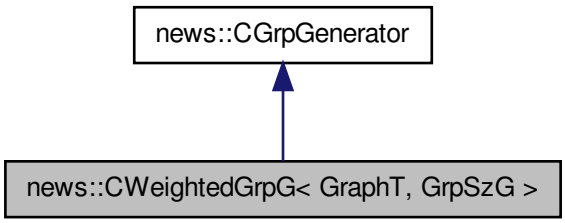
9.22 news::CWeightedGrpG< GraphT, GrpSzG > Class Template Reference

`#include <grp_gen.h>`

Inheritance diagram for news::CWeightedGrpG< GraphT, GrpSzG >:



Collaboration diagram for news::CWeightedGrpG< GraphT, GrpSzG >:



Public Member Functions

- **CWeightedGrpG** (const GraphT &gra, const GrpSzG &gsg)
- int gen_grp (int &ns, std::set< int > &smem)

9.22.1 Detailed Description

template<class GraphT, class GrpSzG>class news::CWeightedGrpG< GraphT, GrpSzG >

generate group from a "node-weighted" graph. In this graph, each vertex has a weight within [0,1] and provides a weight() function (as idwVertex) to access that weight. GraphT: graph type. GrpSzG: group size generator.

9.22.2 Member Function Documentation

9.22.2.1 template<class GraphT , class GrpSzG > int news::CWeightedGrpG< GraphT, GrpSzG >::gen_grp (int & *ns*, std::set< int > & *smem*) [inline, virtual]

generate a group.

Parameters

ns	source node of generated group
smem	member nodes (indices) of generated group

Returns

number of nodes in group, 0 means failed.

Implements news::CGrpGenerator.

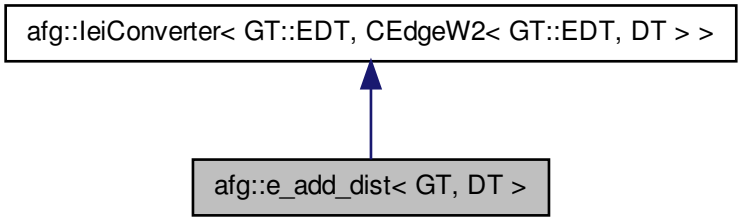
The documentation for this class was generated from the following file:

- grp_gen.h

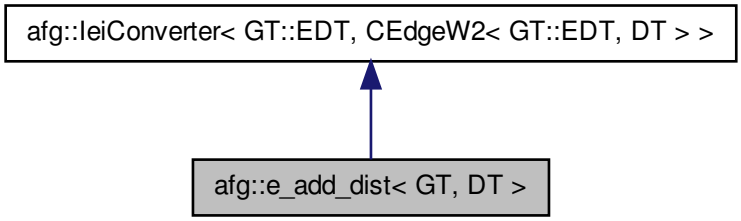
9.23 afg::e_add_dist< GT, DT > Struct Template Reference

```
#include <graph_convert.h>
```

Inheritance diagram for afg::e_add_dist< GT, DT >:



Collaboration diagram for afg::e_add_dist< GT, DT >:



Public Types

- typedef IeiConverter< typename GT::EDT, CEdgeW2< typename GT::EDT, DT > > **converter_base_type**

Public Member Functions

- **e_add_dist** (const GT &gra)

- converter_base_type::result_type **convert** (const typename converter_base_type::original_type &e1, int i, int j)

Public Attributes

- const GT & **m_gra**

9.23.1 Detailed Description

template<class GT, class DT>struct afg::e_add_dist< GT, DT >

add geometric distance as a new edge weight. The original graph is passed to the constructor, and the graph is assumed to have vertex which contains x,y coordinates accessible through x() and y(). New edge has two weights: the original one and the new distance. GT: original graph type, DT: distance type

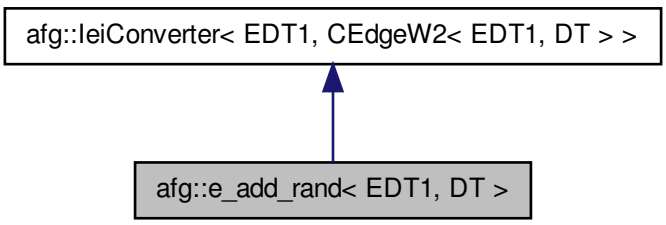
The documentation for this struct was generated from the following file:

- graph_convert.h

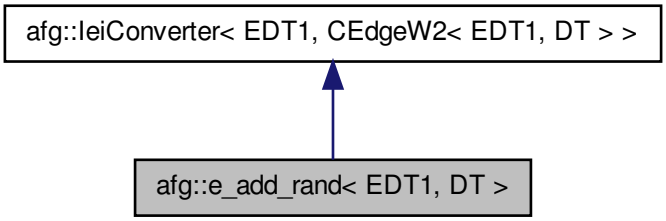
9.24 afg::e_add_rand< EDT1, DT > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::e_add_rand< EDT1, DT >:



Collaboration diagram for afg::e_add_rand< EDT1, DT >:



Public Types

- typedef IeiConverter< EDT1, CEdgeW2< EDT1, DT > > **converter_base_type**

Public Member Functions

- **e_add_rand** (double max=1, double min=0)
- converter_base_type::result_type **convert** (const EDT1 &e1, int, int)

Public Attributes

- double **m_max**
- double **m_min**

9.24.1 Detailed Description

template<class EDT1, class DT>struct afg::e_add_rand< EDT1, DT >

add random number as a new edge weight. New edge has two weights: the original one and a random number. EDT1: original edge type; DT: random number type (e.g., double or int).

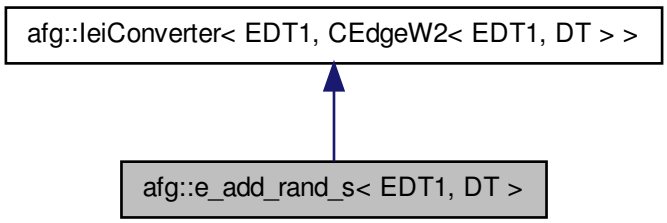
The documentation for this struct was generated from the following file:

- graph_convert.h

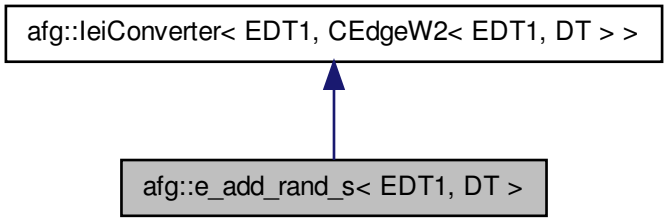
9.25 afg::e_add_rand_s< EDT1, DT > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::e_add_rand_s< EDT1, DT >:



Collaboration diagram for afg::e_add_rand_s< EDT1, DT >:



Public Types

- typedef IeiConverter< EDT1, CEdgeW2< EDT1, DT > > **converter_base_type**

Public Member Functions

- void clear_cache (void)

  clear previously stored random numbers
- **e_add_rand_s** (double max=1.0, double min=0.0)
- converter_base_type::result_type **convert** (const typename converter_base_type::original_type &e1, int i, int j)

Public Attributes

- double **m_max**
- double **m_min**
- std::map< std::pair< int, int >, DT > **m_cache**

9.25.1 Detailed Description

template<class EDT1, class DT>struct afg::e_add_rand_s< EDT1, DT >

add symmetrical random number as a new edge weight. New edge has two weights: the original one and a random number. New weight is symmetrical: w(i->j)=w(j->i) EDT1: original edge type; DT: random number type (e.g., double or int).

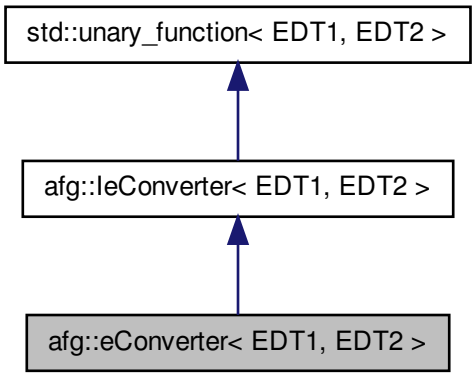
The documentation for this struct was generated from the following file:

- graph_convert.h

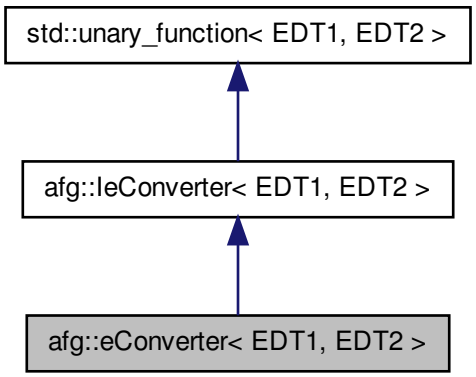
9.26 afg::eConverter< EDT1, EDT2 > Struct Template Reference

```
#include <graph_convert.h>
```

Inheritance diagram for afg::eConverter< EDT1, EDT2 >:



Collaboration diagram for afg::eConverter< EDT1, EDT2 >:



Public Member Functions

- EDT2 **convert** (const EDT1 &e1)

9.26.1 Detailed Description

template<class EDT1, class EDT2>struct afg::eConverter< EDT1, EDT2 >

edge converter that just does an explicit type conversion.

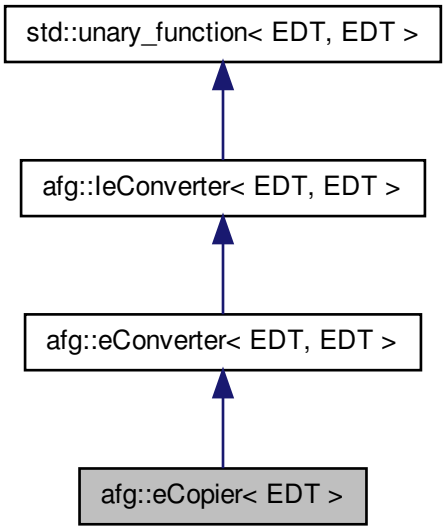
The documentation for this struct was generated from the following file:

- graph_convert.h

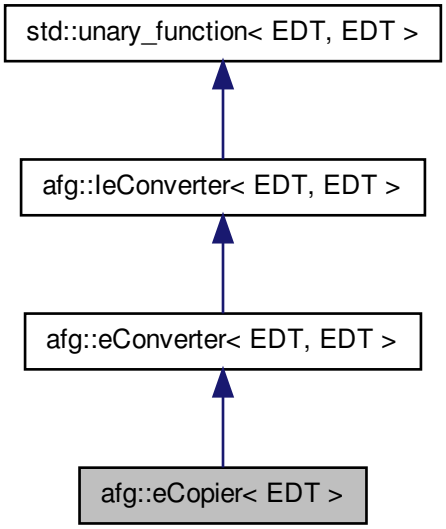
9.27 afg::eCopier< EDT > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::eCopier< EDT >:



Collaboration diagram for afg::eCopier< EDT >:



Public Member Functions

- EDT **convert** (const EDT &e1)

9.27.1 Detailed Description

template<class EDT>struct afg::eCopier< EDT >

edge converter that just makes a copy.

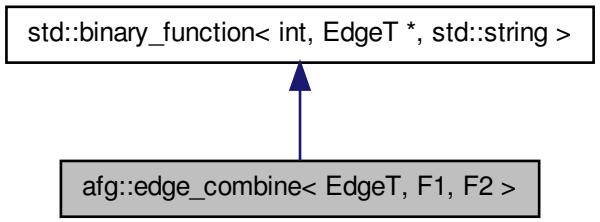
The documentation for this struct was generated from the following file:

- graph_convert.h

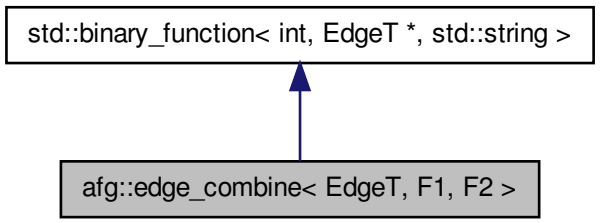
9.28 afg::edge_combine< EdgeT, F1, F2 > Struct Template Reference

```
#include <export_gdl.h>
```

Inheritance diagram for afg::edge_combine< EdgeT, F1, F2 >:



Collaboration diagram for afg::edge_combine< EdgeT, F1, F2 >:



Public Types

- typedef edge_combine< EdgeT, F1, F2 > **my_type**

Public Member Functions

- **edge_combine** (F1 &f1, F2 &f2)
- std::string **operator()** (int i, const EdgeT *pe) const

Public Attributes

- F1 **m_f1**
- F2 **m_f2**

9.28.1 Detailed Description

template<class EdgeT, class F1, class F2>struct afg::edge_combine< EdgeT, F1, F2 >

combine two edge attribute function objects.

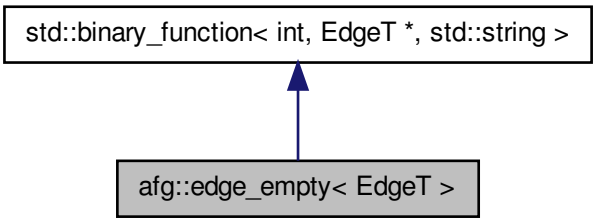
The documentation for this struct was generated from the following file:

- export_gdl.h

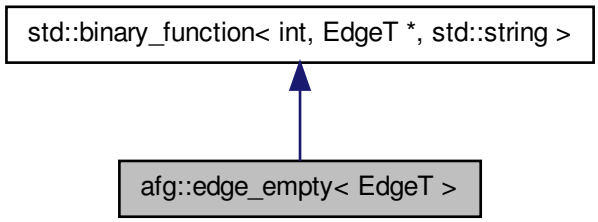
9.29 afg::edge_empty< EdgeT > Struct Template Reference

`#include <export_gdl.h>`

Inheritance diagram for afg::edge_empty< EdgeT >:



Collaboration diagram for afg::edge_empty< EdgeT >:



Public Member Functions

- std::string **operator()** (int, const EdgeT ∗) const

9.29.1 Detailed Description

template<class EdgeT>struct afg::edge_empty< EdgeT >

return an empty attribute for an edge.

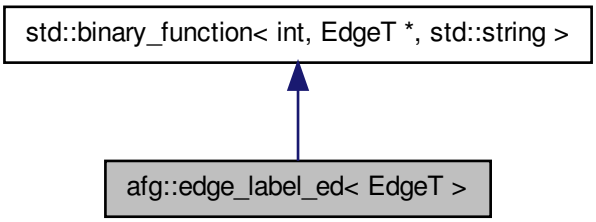
The documentation for this struct was generated from the following file:

- export_gdl.h

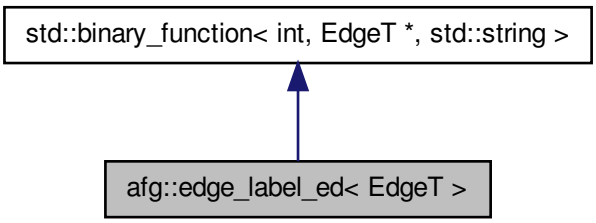
9.30 afg::edge_label_ed< EdgeT > Struct Template Reference

```
#include <export_gdl.h>
```

Inheritance diagram for afg::edge_label_ed< EdgeT >:



Collaboration diagram for afg::edge_label_ed< EdgeT >:



Public Member Functions

- std::string **operator()** (int, const EdgeT ∗pe) const

9.30.1 Detailed Description

template<class EdgeT>struct afg::edge_label_ed< EdgeT >

return an edge data as its label.

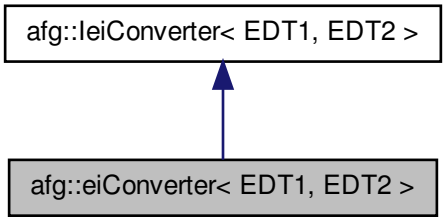
The documentation for this struct was generated from the following file:

- export_gdl.h

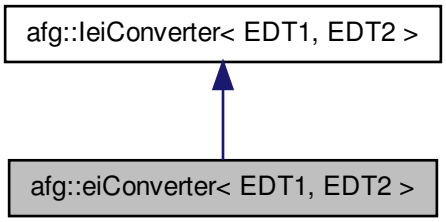
9.31 afg::eiConverter< EDT1, EDT2 > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::eiConverter< EDT1, EDT2 >:



Collaboration diagram for afg::eiConverter< EDT1, EDT2 >:



Public Member Functions

- EDT2 **convert** (const EDT1 &e1, int, int)

9.31.1 Detailed Description

template<class EDT1, class EDT2>struct afg::eiConverter< EDT1, EDT2 >

edge converter with indices, just do an explicit type conversion.

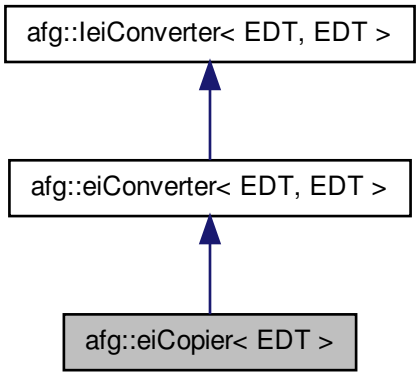
The documentation for this struct was generated from the following file:

- graph_convert.h

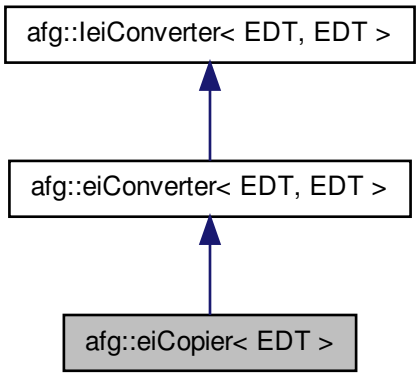
9.32 afg::eiCopier< EDT > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::eiCopier< EDT >:



Collaboration diagram for afg::eiCopier< EDT >:



Public Member Functions

- EDT **convert** (const EDT &e1, int, int)

9.32.1 Detailed Description

template<class EDT>struct afg::eiCopier< EDT >

edge converter with indices, that just makes a copy.

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

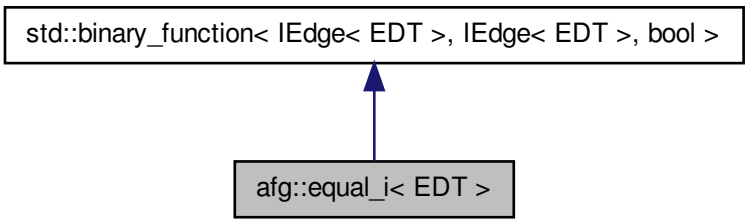
- graph_convert.h

9.33 afg::equal_i< EDT > Struct Template Reference

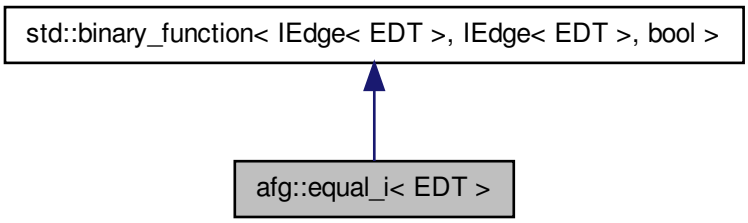
predicate to test if the indices of two edges are the same.

`#include <graph_intf.h>`

Inheritance diagram for afg::equal_i< EDT >:



Collaboration diagram for afg::equal_i< EDT >:



Public Member Functions

- bool **operator()** (const IEdge< EDT > &lhs, const IEdge< EDT > &rhs) const

9.33.1 Detailed Description

template<class EDT>struct afg::equal_i< EDT >

predicate to test if the indices of two edges are the same.

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

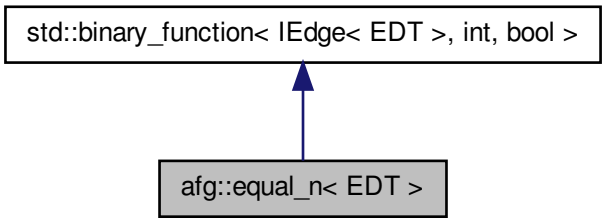
- graph_intf.h

9.34 afg::equal_n< EDT > Struct Template Reference

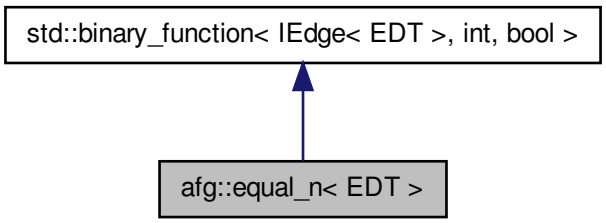
predicate to test if the index of an edge is equal to an integer.

`#include <graph_intf.h>`

Inheritance diagram for afg::equal_n< EDT >:



Collaboration diagram for afg::equal_n< EDT >:



Public Member Functions

- bool **operator()** (const IEdge< EDT > &lhs, int n) const

9.34.1 Detailed Description

template<class EDT>struct afg::equal_n< EDT >

predicate to test if the index of an edge is equal to an integer.

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

- graph_intf.h

9.35 afg::export_gdl< GraphT, Ftitle, Fv_optional, Fe_optional > Class Template Reference

`#include <export_gdl.h>`

Public Member Functions

- **export_gdl** (Ftitle title=Ftitle(), Fv_optional v_optional=Fv_optional(), Fe_optional e_optional=Fe_optional())
- void **set_global** (const std::string &s)
- void export_g (std::ostream &os, const GraphT &gra)

Public Attributes

- std::string **m_sGlobal**
- Ftitle **m_ftitle**
- Fv_optional **m_fvopt**
- Fe_optional **m_feopt**
- std::string **m_sEllipse**
- std::string **m_sBox**
- std::string **m_sNoarrow**

9.35.1 Detailed Description

template<class GraphT, class Ftitle = title_ind<typename GraphT::iVT >, class Fv_optional = node_empty<typename GraphT::iVT>, class Fe_optional = edge_empty<typename GraphT::iET>>class afg::export_gdl< GraphT, Ftitle, Fv_optional, Fe_optional >

Export a grpah in gdl (graph description language) format. The exported gdl file can be imported to other graph visualization tools.

9.35.2 Member Function Documentation

9.35.2.1 template<class GraphT , class Ftitle = title_ind<typename GraphT::iVT >, class Fv_optional = node_empty<typename GraphT::iVT>, class Fe_optional = edge_empty<typename GraphT::iET>> void afg::export_gdl< GraphT, Ftitle, Fv_optional, Fe_optional >::export_g (std::ostream & *os,* const GraphT & *gra*) [inline]

operator ( ) that do the job.

Parameters

os	output stream
gra	the given graph
title	function object to return a title (string type) for a node, it must accept two parameters: an integer (index of node in graph) and a const reference of the node (vertex).
v_optional	function object to return some optional attributes (string type) for a node, it must accept two parameters: an integer (index of node in graph) and a const reference of the node (vertex).
eoptional	function object to return some optional attributes (string type) for an edge, it must accept two parameters: an integer (index of source node in graph) and a const reference of the edge.

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

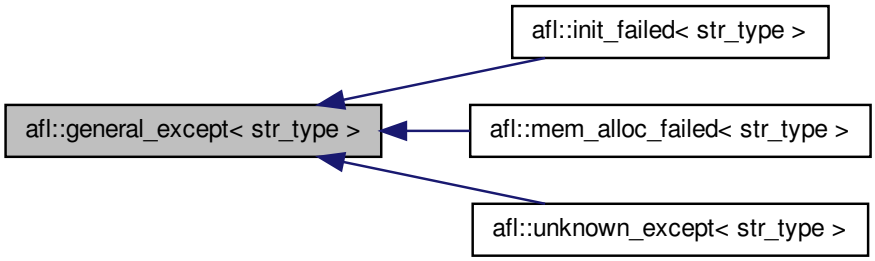
- export_gdl.h

9.36 afl::general_except< str_type > Class Template Reference

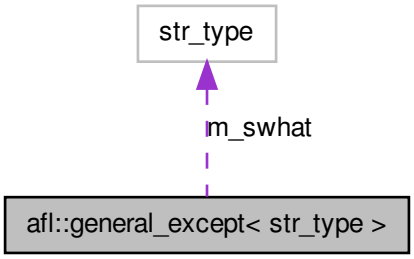
General exception.

`#include <exceptions.hpp>`

Inheritance diagram for afl::general_except< str_type >:



Collaboration diagram for afl::general_except< str_type >:



Public Member Functions

- general_except (const str_type &s=str_type("general exception"))
- virtual const char * what () const
- virtual const str_type & **what_s** () const

9.36.1 Detailed Description

template<class str_type = std::string>class afl::general_except< str_type >

General exception.

9.36.2 Constructor & Destructor Documentation

9.36.2.1 template<class str_type = std::string> afl::general_except< str_type >::general_except (const str_type & *s* = str_type("general exception")) [inline, explicit]

Constructor

Parameters

s	error message for this exception

9.36.3 Member Function Documentation

9.36.3.1 template<class str_type = std::string> virtual const char* afl::general_except< str_type >::what ( ) const [inline, virtual]

Get the error message.

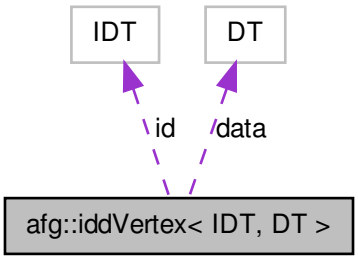
The documentation for this class was generated from the following file:

- exceptions.hpp

9.37 afg::iddVertex< IDT, DT > Struct Template Reference

```
#include <vertex.h>
```

Collaboration diagram for afg::iddVertex< IDT, DT >:



Public Member Functions

- **iddVertex** (const IDT &i=IDT(), const DT &d=DT())

Public Attributes

- IDT **id**
- DT **data**

9.37.1 Detailed Description

template<class IDT, class DT>struct afg::iddVertex< IDT, DT >

vertex with an ID and some other data. The other data doesn't matter in comparison operations.

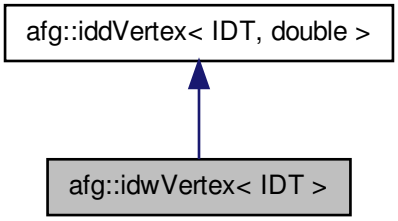
The documentation for this struct was generated from the following file:

- vertex.h

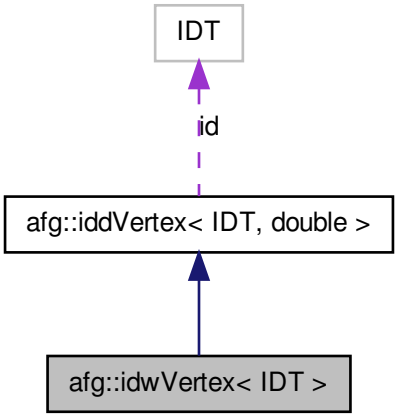
9.38 afg::idwVertex< IDT > Struct Template Reference

```
#include <vertex.h>
```

Inheritance diagram for afg::idwVertex< IDT >:



Collaboration diagram for afg::idwVertex< IDT >:



Public Member Functions

- **idwVertex** (const IDT &i=IDT(), double d=0.0)
- double & **weight** (void)

- double **weight** (void) const

9.38.1 Detailed Description

template<class IDT>struct afg::idwVertex< IDT >

vertex with an id and a weight of type double

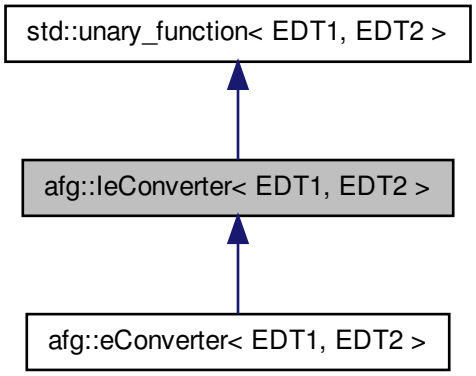
The documentation for this struct was generated from the following file:

- vertex.h

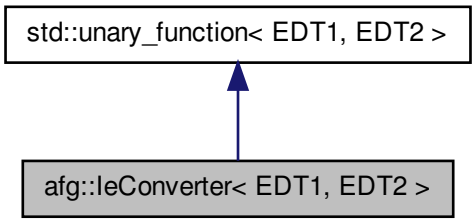
9.39 afg::IeConverter< EDT1, EDT2 > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::IeConverter< EDT1, EDT2 >:



Collaboration diagram for afg::IeConverter< EDT1, EDT2 >:



Public Types

- typedef EDT1 **original_type**
- typedef EDT2 **result_type**

Public Member Functions

- virtual EDT2 **convert** (const EDT1 &e1) const =0
- EDT2 operator() (const EDT1 &e1) const

9.39.1 Detailed Description

template<class EDT1, class EDT2>struct afg::IeConverter< EDT1, EDT2 >

interface of edge converter. Convert an edge from type EDT1 to EDT2.

9.39.2 Member Function Documentation

9.39.2.1 template<class EDT1, class EDT2> EDT2 afg::IeConverter< EDT1, EDT2 >::operator() (const EDT1 & *e1*) const [inline]

() operator.

Parameters

e1	original edge

Returns

converted edge of type EDT2

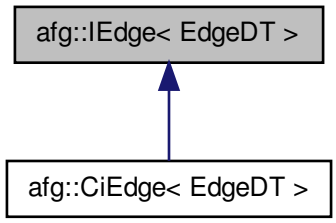
The documentation for this struct was generated from the following file:

- graph_convert.h

9.40 afg::IEdge< EdgeDT > Class Template Reference

`#include <graph_intf.h>`

Inheritance diagram for afg::IEdge< EdgeDT >:



Public Types

- typedef EdgeDT EDT
  type of the edge data

Public Member Functions

- virtual int to (void) const =0
  return destination node (index)
- virtual int & to (void)=0
  return reference of destination node (index)
- virtual EdgeDT & edge_d (void)=0
  return reference of edge data
- virtual const EdgeDT & edge_d (void) const =0

return reference of edge data, const version

- virtual void set (const EdgeDT &e)

  set data of the edge

- virtual std::ostream & output (std::ostream &os) const

  output IEdge to a stream in format of "edge_index: edge_data"

9.40.1 Detailed Description

template<class EdgeDT>class afg::IEdge< EdgeDT >

Edge interface. An edge can return an index of the destination node and edge data of type EdgeDT.

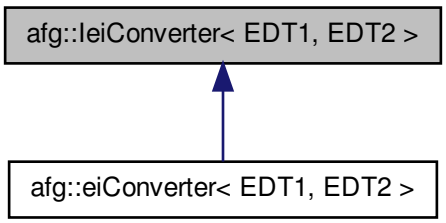
The documentation for this class was generated from the following file:

- graph_intf.h

9.41 afg::IeiConverter< EDT1, EDT2 > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::IeiConverter< EDT1, EDT2 >:



Public Types

- typedef EDT1 **original_type**
- typedef EDT2 **result_type**

Public Member Functions

- virtual EDT2 **convert** (const EDT1 &e1, int i, int j)
- EDT2 operator() (const EDT1 &e1, int i, int j)

9.41.1 Detailed Description

template<class EDT1, class EDT2>struct afg::IeiConverter< EDT1, EDT2 >

interface of edge converter with indices. Convert an edge from type EDT1 to EDT2, indices of source/destination nodes are passed. This can also be used a default converter which just do an explicit type conversion.

9.41.2 Member Function Documentation

9.41.2.1 template<class EDT1, class EDT2> EDT2 afg::IeiConverter< EDT1, EDT2 >::operator() (const EDT1 & *e1*, int *i*, int *j*) [inline]

() operator.

Parameters

e1	original edge
i	originating node
j	destination node

Returns

converted edge

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

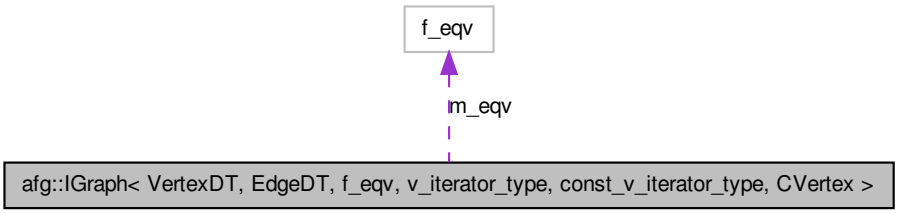
- graph_convert.h

9.42 afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex > Class Template Reference

`#include <graph_intf.h>`

Collaboration diagram for afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_-

v_iterator_type, CVertex >:



Public Types

type definitions.

An implementation should provide all the following type defintions.

- typedef EdgeDT EDT
  edge data type
- typedef VertexDT VDT
  vertex data type
- typedef IEdge< EDT > iET
  edge type, edge destination along with data
- typedef CVertex iVT
  vertex type,
- typedef f_eqv COMPT
  type compare function object to determine if two vertex data are equal
- typedef IGraph< VertexDT, EdgeDT, v_iterator_type, const_v_iterator_type, f_eqv, CVertex > GT
  graph type, this class itself
- typedef v_iterator_type iterator
- typedef const_v_iterator_type const_iterator
- typedef iVT::iterator e_iterator
  edge iterator
- typedef iVT::const_iterator const_e_iterator
  edge iterator, const

Public Member Functions

- **IGraph** (int size=1, const VertexDT &v=VertexDT(), double dg=0.25, const f_eqv &eqv=f_eqv())
- virtual void reserve (int n)=0
- virtual int pack (int ns=0)=0
- virtual std::ostream & output (std::ostream &os) const

  output graph to a stream

element access and helpers

- virtual iterator begin (void)=0

  return iterator pointing to the first vertex
- virtual iterator end (void)=0

  return iterator pointing to the end (not pointing to any vertex).
- virtual const_iterator begin (void) const =0

  return iterator pointing to the first vertex, const version.
- virtual const_iterator end (void) const =0

  return iterator pointing to the end, const version.
- virtual iVT & at (int i)=0
- virtual iVT & operator[] (int i)=0
- virtual const iVT & at (int i) const =0
- virtual const iVT & operator[] (int i) const =0
- virtual VDT & vertex_d (int nindex)
- virtual const VDT & vertex_d (int nindex) const
- virtual e_iterator e_begin (int i)
- virtual e_iterator e_end (int i)
- virtual const_e_iterator e_begin (int i) const
- virtual const_e_iterator e_end (int i) const
- virtual int find_index (const VDT &v) const =0

  find the index of vertex v
- virtual iterator find_vertex (const VDT &v)

  find vertex equal to v, return position (iterator)
- virtual const_iterator find_vertex (const VDT &v) const

  find vertex equal to v, return position (iterator), const version
- virtual EDT * get_edge (const VDT &u, const VDT &v)=0
- virtual EDT * get_edge_byi (int nfrom, int nto)=0
- virtual const EDT * get_edge (const VDT &u, const VDT &v) const =0
- virtual const EDT * get_edge_byi (int nfrom, int nto) const =0
- virtual int size (void) const =0

  return graph size: number of vertices in the graph
- virtual int range (void) const =0
- bool is_in_range (int i) const

  is i within the index access range
- virtual bool is_in_use (int i) const =0

  return if i (range checked) is an index used for a vertex.
- bool is_valid (int i) const

  is i an index pointing to a "valid" vertex

insertion/deletion

- virtual int insert_v_qik (const VDT &v)=0
- virtual int insert_v (const VDT &v)=0
- virtual bool insert_v_ati (const VDT &v, int i)=0
- virtual bool set_v_ati (const VDT &v, int i)
- virtual void remove_v (const VDT &v)=0
- virtual void remove_v_byi (int vi)=0
- virtual bool insert_e (const VDT &u, const VDT &v, const EDT &e)=0
- virtual bool insert_2e (const VDT &u, const VDT &v, const EDT &e)=0
- virtual bool insert_e_byi (int i, int j, const EDT &e)=0
- virtual bool insert_2e_byi (int i, int j, const EDT &e)=0
- virtual void remove_e (const VDT &u, const VDT &v)=0
- virtual void remove_2e (const VDT &u, const VDT &v)=0
- virtual void remove_alle (const VDT &u)=0
- virtual void remove_e_byi (int nfrom, int nto)=0
- virtual void remove_2e_byi (int n1, int n2)=0
- virtual void remove_alle_byi (int nfrom)=0

  remove all edges of vertex of index nfrom
- virtual void remove_all_edges (void)=0

  remove all edges of all vertices
- virtual void clear (void)=0

  clear all nodes and all their edges

Protected Attributes

- f_eqv m_eqv

9.42.1 Detailed Description

template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>>class afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >

Interface of a graph. Requirements of a graph class: (1)Provides an interface defined below. (2)Provides iterator-based access to vertices, an iterator gets a VertexT type which provides an interface defined in IVertex. (3)Provides operator [ ] based random access to vertices, which returns a VertexT type; the index must be within a [0, range( )-1] range; this is to facilitate the implementations of many algorithms. However, an index or iterator may point to a "vertex" which is not part of the graph (removed or empty position); such a "vertex" is isolated without any neighbors, this doesn't affect many algorithms; is_valid(int i) can tell if index i points to a valid vertex; pack( ) can be called to make all vertices appear in consecutive order within [0, size( )-1]. (4)It guarantees that the same index points to the same vertex before a pack( ) function call is made. However, edge removal may lead to poisitioning change and my invalidate iterator pointing to an edge. (5)vertex iterator?

9.42.2 Member Typedef Documentation

9.42.2.1 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> typedef const_v_iterator_type afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::const_iterator

iterator type to access vertices. Only read access is allowed.

Reimplemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.2.2 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> typedef v_iterator_type afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::iterator

iterator type to access vertices.

Reimplemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3 Member Function Documentation

9.42.3.1 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual iVT& afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::at (int *i*) [pure virtual]

return vertex at position i

Exceptions

std::out_of_range.	

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.2 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual const iVT& afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::at (int *i*) const [pure virtual]

return vertex at position i, const version.

Exceptions

std::out_of_range.	

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.3 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual e_iterator afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::e_begin (int *i*) [inline, virtual]

return iterator pointing to the first edge of vertex of index i.

Exceptions

std::out_of_range,with	message "e_begin( ) of IGraph".

9.42.3.4 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual const_e_iterator afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::e_begin (int *i*) const [inline, virtual]

return const iterator pointing to the first edge of vertex of index i.

Exceptions

std::out_of_range,with	message "e_begin( ) of IGraph".

9.42.3.5 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual e_iterator afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::e_end (int *i*) [inline, virtual]

return iterator pointing to the end of edges of vertex of index i.

Exceptions

std::out_of_range,with	message "e_end( ) of IGraph".

9.42.3.6 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual const_e_iterator afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::e_end (int *i*) const [inline, virtual]

return const iterator pointing to the end of edges of vertex of index i.

Exceptions

std::out_of_range,with	message "e_end( ) of IGraph".

9.42.3.7 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual EDT* afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::get_edge (const VDT & *u,* const VDT & *v*) [pure virtual]

Get edge data (from u to v).

Returns

a pointer to that edge (data), NULL if not found.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.8 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual const EDT* afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::get_edge (const VDT & *u,* const VDT & *v*) const [pure virtual]

Get edge data (from u to v), const version.

Returns

a pointer to that edge (data), NULL if not found.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.9 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual EDT* afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::get_edge_byi (int *nfrom,* int *nto*) [pure virtual]

get edge data by index.

Returns

a pointer to that edge (data), NULL if not found.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.10 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual const EDT* afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::get_edge_byi (int *nfrom,* int *nto*) const [pure virtual]

get edge data by index, const version.

Returns

a pointer to that edge (data), NULL if not found.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.11 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual bool afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::insert_2e (const VDT & *u,* const VDT & *v,* const EDT & *e*) [pure virtual]

insert both (u->v) and (v->u). The same edge data is used for both edges. Equivalent to "insert_e(u,v,e); insert_e(v,u,e);".

See also

insert_e( )

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.12 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual bool afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::insert_2e_byi (int *i,* int *j,* const EDT & *e*) [pure virtual]

insert both (i->j) and (j->i). The same edge data is used for both edges. Equivalent to "insert_e(i,j,e); insert_e(j,i,e);".

See also

insert_e_byi( )

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.13 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual bool afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::insert_e (const VDT & *u,* const VDT & *v,* const EDT & *e*) [pure virtual]

insert an edge for vertex u to v. Insertion fails if any of the vertex is not found. If edge already exists then edge data will be replaced by e.

Returns

true if succeeds, false otherwise.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, afg::CrTree< VDT, EDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.14 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual bool afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::insert_e_byi (int *i,* int *j,* const EDT & *e*) [pure virtual]

insert an edge (i->j). Insertion fails if any of the two indices is unused (not vertext at that position).

Returns

true if succeeds, otherwise false.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, afg::CrTree< VDT, EDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.15 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual int afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::insert_v (const VDT & *v*) [pure virtual]

insert a vertex of value v. Vertex data is updated if vertex (v) already exists.

Returns

index of the vertex in vertex table, -1 (an invalid index) to indicate failure.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.16 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual bool afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::insert_v_ati (const VDT & *v,* int *i*) [pure virtual]

insert a vertex of value v at position i. This allows a graph user to control vertex at each position. Vertex data is updated if vertex (v) already exists.

Returns

true if insertion succeeds; false if fails (table is smaller than i and fails to grow).

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.17 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual int afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::insert_v_qik (const VDT & *v*) [pure virtual]

insert a vertex of value v. Quick version that doesn't search for existing vertex of the same value -- use this if it is known that the new vertex doesn't exist.

Returns

index of the vertex in vertex table, -1 (an invalid index) to indicate failure.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.18 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual const iVT& afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::operator[] (int *i*) const [pure virtual]

return vertex at position i, const version. Behavior is undefined if i is out of range, normally results in run-time error.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.19 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual iVT& afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::operator[] (int *i*) [pure virtual]

return vertex at position i. Behavior is undefined if i is out of range, normally results in run-time error.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.20 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual int afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::pack (int *ns* = 0) [pure virtual]

rearrange vertices to make them stored consecutively. After pack( ), all iterators and indices may be invalidated. All vertices will be stored within [0, size()).

Parameters

ns	new capacity; if ns<size( ), size() will be used.

Returns

number of index changes that have been made.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, afg::CrTree< VDT, EDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.21 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual int afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::range (void) const [pure virtual]

return current upper bound on vertex index range. Vertex index will be within [0, range()-1], will always have range()>=size()

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.22 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual void afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::remove_2e (const VDT & *u,* const VDT & *v*) [pure virtual]

remove both (u->v) and (v->u). Equivalent to "remove_e(u,v); remove_e(v,u);". Has no effect if edge not found.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.23 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual void afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::remove_2e_byi (int *n1,* int *n2*) [pure virtual]

remove both edges by index. Has no effect if edge not found.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.24 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual void afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::remove_alle (const VDT & *u*) [pure virtual]

remove all edges of vertex v. Has no effect if vertex not found.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.25 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual void afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::remove_e (const VDT & *u,* const VDT & *v*) [pure virtual]

remove edge (u->v). Has no effect if edge not found.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.26 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual void afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::remove_e_byi (int *nfrom,* int *nto*) [pure virtual]

remove an edge by index. Has no effect if edge not found.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, afg::CrTree< VDT, EDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.27 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual void afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::remove_v (const VDT & *v*) [pure virtual]

remove a vertex of value v. Has no effect if vertex not found; otherwise this vertex and all related edges (from/to it) will be removed as well.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.28 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual void afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::remove_v_byi (int *vi*) [pure virtual]

remove a vertex of index vi. Has no effect if vertex not found; otherwise this vertex and all related edges (from/to it) will be removed as well.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, afg::CrTree< VDT, EDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.29 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual void afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::reserve (int *n*) [pure virtual]

reserve a certain size for the graph. n has to be >=range( ), otherwise it has no effect; Range will will be increased to n, and all iteroators may be invalidated, but an index should still pointing to the same vertex.

Implemented in afg::CGraph< VertexDT, EdgeDT, f_eqv >, afg::CrTree< VDT, EDT, f_eqv >, and afg::CGraph< VDT, EDT, f_eqv >.

9.42.3.30 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual bool afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::set_v_ati (const VDT & *v,* int *i*) [inline, virtual]

?? set the value of vertex at position i. This allows a graph user to control vertex at each position. Value of existing vertex at position i (if exist) will be replaced by v, any existing edge will be kept. Has no effect if there is no vertex at position i.

Returns

true if replacement succeeds; false if there is no vertex at i.

9.42.3.31 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual const VDT& afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::vertex_d (int *nindex*) const [inline, virtual]

return the vertex data of at position index, const version.

See also

vertex_d( ).

9.42.3.32 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> virtual VDT& afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::vertex_d (int *nindex*) [inline, virtual]

return the vertex data of at position index.

Parameters

nindex	position index, should be within [0, size()-1].

Returns

a reference to vertex data at position nindex (type VertexDT).

Exceptions

std::out_of_range,indirectly	through at.

9.42.4 Member Data Documentation

9.42.4.1 template<class VertexDT, class EdgeDT, class f_eqv = std::equal_to<VertexDT>, class v_iterator_type = Cvoid, class const_v_iterator_type = Cvoid, class CVertex = IVertex<VertexDT, EdgeDT>> f_eqv afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >::m_eqv [protected]

predicate to determine if two vertex data (of type VertexDT) are equivalent (e.g., with same key or id)

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

- graph_intf.h

9.43 afl::in_convt2t< T1, T2 > Struct Template Reference

`#include <util_tl.hpp>`

Public Member Functions

- **in_convt2t** (T2 &t)

Public Attributes

- T2 & **t2**

9.43.1 Detailed Description

template<class T1, class T2>struct afl::in_convt2t< T1, T2 >

class used to get input of one type from an input stream and convert it to another type.

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

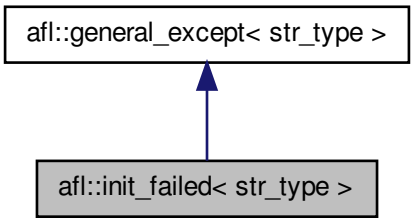
- util_tl.hpp

9.44 afl::init_failed< str_type > Class Template Reference

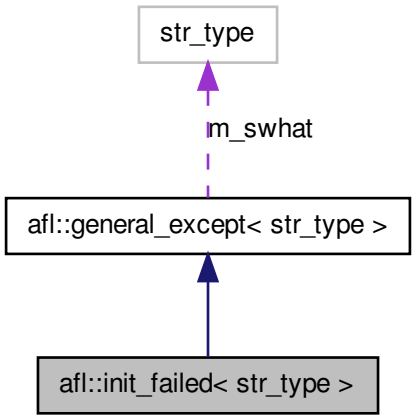
Initialization failure exception.

`#include <exceptions.hpp>`

Inheritance diagram for afl::init_failed< str_type >:



Collaboration diagram for afl::init_failed< str_type >:



Public Member Functions

- init_failed (const str_type &s=str_type(""))

9.44.1 Detailed Description

template<class str_type = std::string>class afl::init_failed< str_type >

Initialization failure exception.

9.44.2 Constructor & Destructor Documentation

9.44.2.1 template<class str_type = std::string> afl::init_failed< str_type >::init_failed (const str_type & *s* = `str_type( "" )`) [inline, explicit]

Parameters

s	specifies where or how the initialization failed (such as " in myclass::myclass( ) (memory allocation)").

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

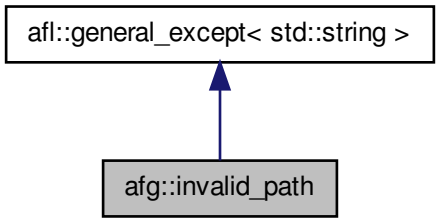
- exceptions.hpp

9.45 afg::invalid_path Class Reference

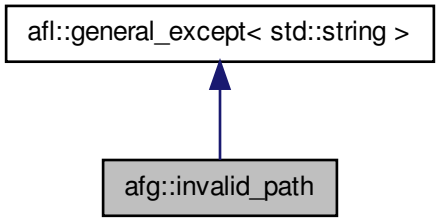
invalid path.

`#include <gexception.h>`

Inheritance diagram for afg::invalid_path:



Collaboration diagram for afg::invalid_path:



Public Member Functions

- **invalid_path** (const std::string &s=std::string("invalid path"))

9.45.1 Detailed Description

invalid path.

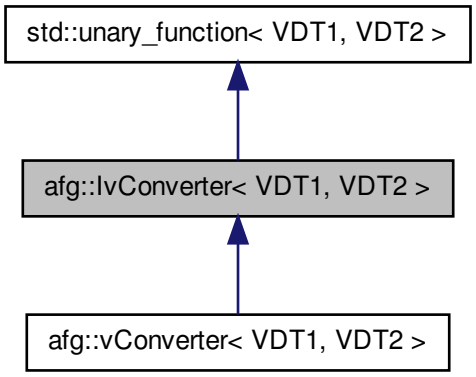
The documentation for this class was generated from the following file:

- gexception.h

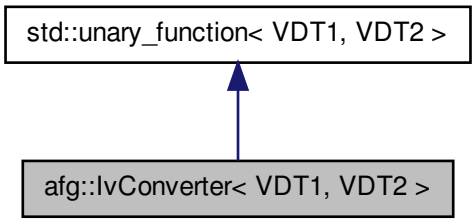
9.46 afg::IvConverter< VDT1, VDT2 > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::IvConverter< VDT1, VDT2 >:



Collaboration diagram for afg::IvConverter< VDT1, VDT2 >:



Public Types

- typedef VDT1 original_type

Public Member Functions

- virtual VDT2 **convert** (const VDT1 &v1) const =0
- VDT2 **operator()** (const VDT1 &v1) const

9.46.1 Detailed Description

template<class VDT1, class VDT2>struct afg::IvConverter< VDT1, VDT2 >

interface of vertex converter. Convert a vertex from type VDT1 to VDT2.

9.46.2 Member Typedef Documentation

9.46.2.1 template<class VDT1, class VDT2> typedef VDT1 afg::IvConverter< VDT1, VDT2 >::original_type

type of the original vertex. New vertex type is result_type as defined in unary_function.

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

- graph_convert.h

9.47 afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type > Class Template Reference

`#include <graph_intf.h>`

Public Types

- typedef EdgeDT EDT
  edge data type
- typedef IEdge< EdgeDT > iET
  edge type, providing the interface defined above
- typedef VertexDT VDT
  vertex data type
- typedef IVertex< VDT, EDT, iterator_type, const_iterator_type > iVT
  vertex type, this class itself
- typedef iterator_type iterator
- typedef const_iterator_type const_iterator

Public Member Functions

- virtual VertexDT & vertex_d (void)=0
  return a reference of the vertex data
- virtual const VertexDT & vertex_d (void) const =0
  return a reference of the vertex data, const version
- virtual void set (const VertexDT &v)
  set vertex data
- virtual iterator begin (void)=0
  return iterator pointing to the first edge
- virtual iterator end (void)=0
  return iterator pointing to the end (not pointting to any endge).
- virtual const_iterator begin (void) const =0
  return iterator pointing to the first edge, const version
- virtual const_iterator end (void) const =0
  return iterator pointing to the end, const version.
- virtual iterator find_edge (int nto)
- virtual const_iterator find_edge (int nto) const
- virtual EDT * get_edge (int nto)=0
- virtual const EDT * get_edge (int nto) const =0
- virtual int out_degree (void) const =0
- virtual std::ostream & output (std::ostream &os) const
  output the vertex to an output stream

9.47.1 Detailed Description

template<class VertexDT, class EdgeDT, class iterator_type = Cvoid, class const_iterator_type = Cvoid>class afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type >

Interface of internal vertex in a graph. VertexDT is the type of data contained in a vertex, simple data type (int, char) or compound type. EdgeT is an edge type which provides an IEdge interface. Requirements of a vertex type: (1)provides all interfaces defined below; (2)provides iterator-based access to edges; assume "iterator_type it;", then *it should return an EdgeT; (3)VertexDT has to be uniquely identifiable.

9.47.2 Member Typedef Documentation

9.47.2.1 template<class VertexDT, class EdgeDT, class iterator_type = Cvoid, class const_iterator_type = Cvoid> typedef const_iterator_type afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type >::const_iterator

const iterator type to access edges (read-only).

Reimplemented in afg::CiVertex< VertexDT, EdgeDT >.

9.47.2.2 template<class VertexDT, class EdgeDT, class iterator_type = Cvoid, class const_iterator_type = Cvoid> typedef iterator_type afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type >::iterator

iterator type to access edges.

Reimplemented in afg::CiVertex< VertexDT, EdgeDT >.

9.47.3 Member Function Documentation

9.47.3.1 template<class VertexDT, class EdgeDT, class iterator_type = Cvoid, class const_iterator_type = Cvoid> virtual iterator afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type >::find_edge (int *nto*) [inline, virtual]

find the edge pointing to node of index nto

Returns

iterator pointing to that edge, equal to end( ) if no edge pointing to nto found.

9.47.3.2 template<class VertexDT, class EdgeDT, class iterator_type = Cvoid, class const_iterator_type = Cvoid> virtual const_iterator afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type >::find_edge (int *nto*) const [inline, virtual]

find the edge pointing to node of index nto

Returns

const_iterator pointing to that edge, equal to end( ) if no edge pointing to nto found.

9.47.3.3 template<class VertexDT, class EdgeDT, class iterator_type = Cvoid, class const_iterator_type = Cvoid> virtual EDT* afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type >::get_edge (int *nto*) [pure virtual]

get an edge of this vertex.

Parameters

nto	destination node (index)

Returns

a pointer to the edge data, NULL if edge doesn't exist

Implemented in afg::CiVertex< VertexDT, EdgeDT >.

9.47.3.4 template<class VertexDT, class EdgeDT, class iterator_type = Cvoid, class const_iterator_type = Cvoid> virtual const EDT* afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type >::get_edge (int *nto*) const [pure virtual]

get an edge of this vertex, const version.

Parameters

nto	destination node (index)

Returns

a pointer to the edge data, NULL if edge doesn't exist

Implemented in afg::CiVertex< VertexDT, EdgeDT >.

9.47.3.5 template<class VertexDT, class EdgeDT, class iterator_type = Cvoid, class const_iterator_type = Cvoid> virtual int afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type >::out_degree (void) const [pure virtual]

get the out degree of this vertex.

Returns

out degree

Implemented in afg::CiVertex< VertexDT, EdgeDT >.

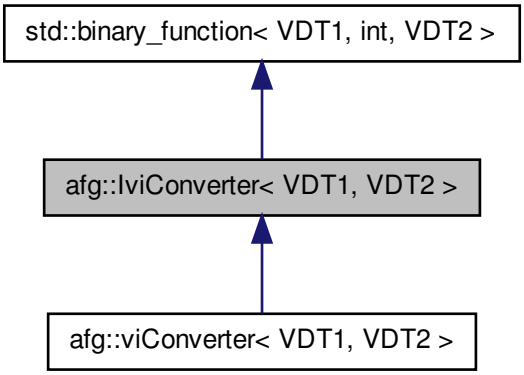
The documentation for this class was generated from the following file:

- graph_intf.h

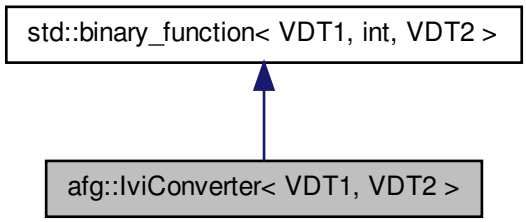
9.48 afg::IviConverter< VDT1, VDT2 > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::IviConverter< VDT1, VDT2 >:



Collaboration diagram for afg::IviConverter< VDT1, VDT2 >:



Public Types

- typedef VDT1 original_type

Public Member Functions

- virtual VDT2 **convert** (const VDT1 &v1, int i) const =0
- VDT2 **operator()** (const VDT1 &v1, int i) const

9.48.1 Detailed Description

template<class VDT1, class VDT2>struct afg::IviConverter< VDT1, VDT2 >

interface of vertex converter with index. Convert a vertex from type VDT1 to VDT2, given index in the original graph.

9.48.2 Member Typedef Documentation

9.48.2.1 template<class VDT1, class VDT2> typedef VDT1 afg::IviConverter< VDT1, VDT2 >::original_type

type of the original vertex. New vertex type (VDT2) is result_type as defined in binary_-function.

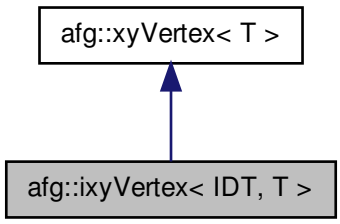
The documentation for this struct was generated from the following file:

- graph_convert.h

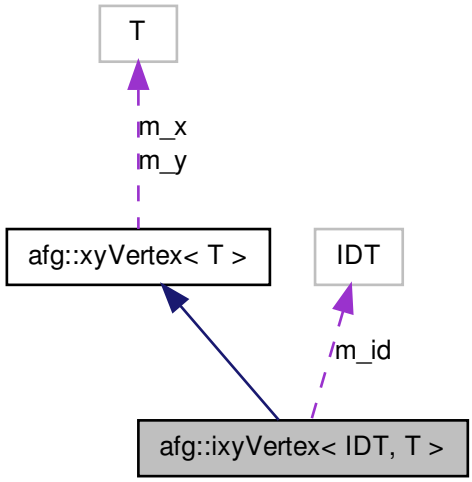
9.49 afg::ixyVertex< IDT, T > Struct Template Reference

`#include <vertex.h>`

Inheritance diagram for afg::ixyVertex< IDT, T >:



Collaboration diagram for afg::ixyVertex< IDT, T >:



Public Member Functions

- **ixyVertex** (IDT i=IDT(), T tx=(T)-1, T ty=(T)-1)
- IDT & **id** (void)
- IDT **id** (void) const

Public Attributes

- IDT **m_id**

9.49.1 Detailed Description

template<class IDT, class T>struct afg::ixyVertex< IDT, T >

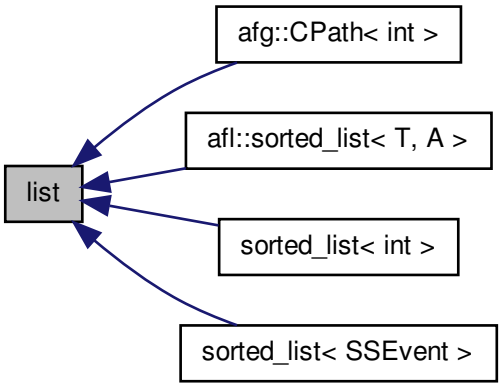
vertex with an id and x,y coordinates. IDT: type the id, T: coordinate type.

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

- vertex.h

9.50 list Class Reference

Inheritance diagram for list:



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

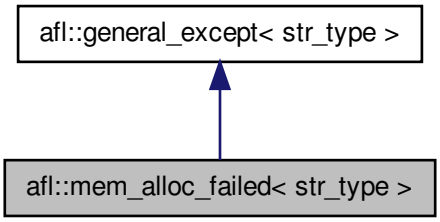
- sorted_list.hpp

9.51 afl::mem_alloc_failed< str_type > Class Template Reference

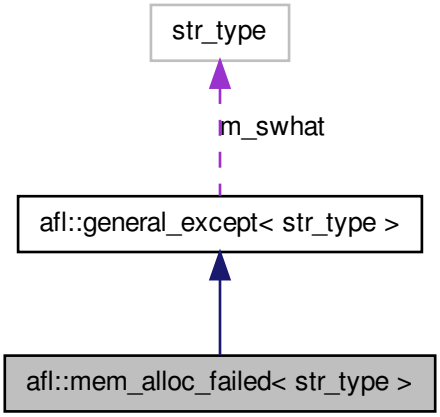
Memory allocation failure exception.

`#include <exceptions.hpp>`

Inheritance diagram for afl::mem_alloc_failed< str_type >:



Collaboration diagram for afl::mem_alloc_failed< str_type >:



Public Member Functions

- mem_alloc_failed (const str_type &s)
- **mem_alloc_failed** (const char *p=NULL)

9.51.1 Detailed Description

template<class str_type = std::string>class afl::mem_alloc_failed< str_type >

Memory allocation failure exception.

9.51.2 Constructor & Destructor Documentation

9.51.2.1 template<class str_type = std::string> afl::mem_alloc_failed< str_type >::mem_alloc_failed (const str_type & *s*) [inline, explicit]

Parameters

s	specifies where or how the initialization failed (such as " in myclass::myclass( ) (memory allocation)").

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

- exceptions.hpp

9.52 afl::named_pair< NameT, KeyT > Struct Template Reference

`#include <util_tl.hpp>`

Public Types

- typedef NameT name_type
  name type
- typedef KeyT key_type
  key type

Public Member Functions

- named_pair ()
- named_pair (const NameT &_V1, const KeyT &_V2)
  constructor
- template<class U , class V >
  named_pair (const named_pair< U, V > &p)
  copy constructor

Public Attributes

- NameT name

  data member: name, used for == and != comparisons.
- KeyT key

  data member: key, used for order comparisons (<, >)

9.52.1 Detailed Description

template<class NameT, class KeyT>struct afl::named_pair< NameT, KeyT >

pair with a name and a key. Equal and not_equal (==, !=) comparisons are based on name only, while order operations (<, >) operations are based on key only.

9.52.2 Constructor & Destructor Documentation

9.52.2.1 template<class NameT, class KeyT> afl::named_pair< NameT, KeyT >::named_pair ( ) [inline]

default constructor. Both NameT and KeyT must have default constructor available to use this one.

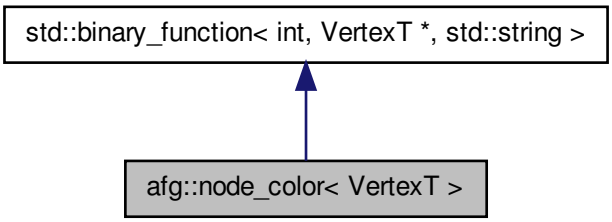
The documentation for this struct was generated from the following file:

- util_tl.hpp

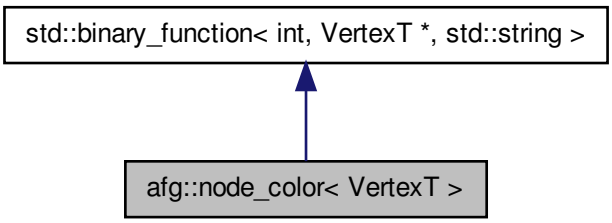
9.53 afg::node_color< VertexT > Struct Template Reference

```
#include <export_gdl.h>
```

Inheritance diagram for afg::node_color< VertexT >:



Collaboration diagram for afg::node_color< VertexT >:



Public Member Functions

- node_color (const std::set< int > &nodes, const std::string &scolor=std::string("red"))
- std::string **operator()** (int i, const VertexT *pe) const

Public Attributes

- const std::set< int > & **m_nodeset**
- const std::string **m_scolor**

9.53.1 Detailed Description

template<class VertexT>struct afg::node_color< VertexT >

return a specific textcolor attribute for nodes in a given set.

9.53.2 Constructor & Destructor Documentation

9.53.2.1 template<class VertexT > afg::node_color< VertexT >::node_color (const std::set< int > & *nodes*, const std::string & *scolor* = `std::string( "red" )`) [inline]

Constructor.

Parameters

nodes	set of nodes that should be colored.
scolor	color for those nodes

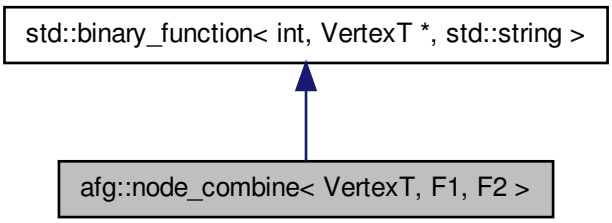
The documentation for this struct was generated from the following file:

- export_gdl.h

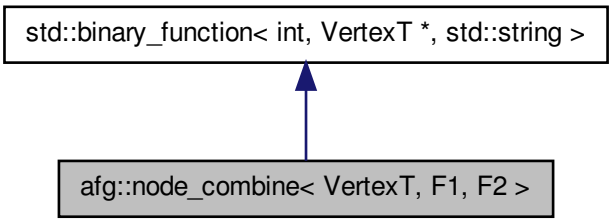
9.54 afg::node_combine< VertexT, F1, F2 > Struct Template Reference

`#include <export_gdl.h>`

Inheritance diagram for afg::node_combine< VertexT, F1, F2 >:



Collaboration diagram for afg::node_combine< VertexT, F1, F2 >:



Public Types

- typedef node_combine< VertexT, F1, F2 > **my_type**

Public Member Functions

- **node_combine** (F1 &f1, F2 &f2)
- std::string **operator()** (int i, const VertexT *pv) const

Public Attributes

- F1 **m_f1**
- F2 **m_f2**

9.54.1 Detailed Description

template<class VertexT, class F1, class F2>struct afg::node_combine< VertexT, F1, F2 >

combine two node attribute function objects.

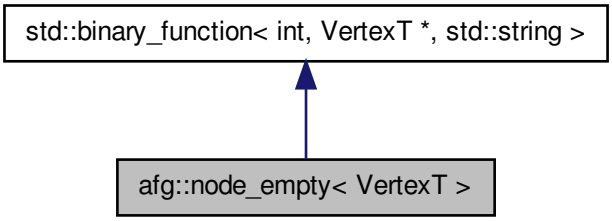
The documentation for this struct was generated from the following file:

- export_gdl.h

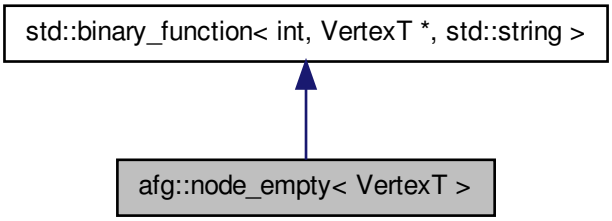
9.55 afg::node_empty< VertexT > Struct Template Reference

`#include <export_gdl.h>`

Inheritance diagram for afg::node_empty< VertexT >:



Collaboration diagram for afg::node_empty< VertexT >:



Public Member Functions

- std::string **operator()** (int, const VertexT ∗) const

9.55.1 Detailed Description

template<class VertexT>struct afg::node_empty< VertexT >

return empty optional attributes for a node.

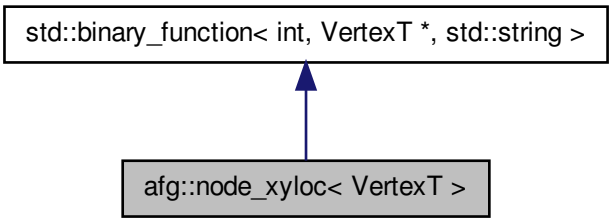
The documentation for this struct was generated from the following file:

- export_gdl.h

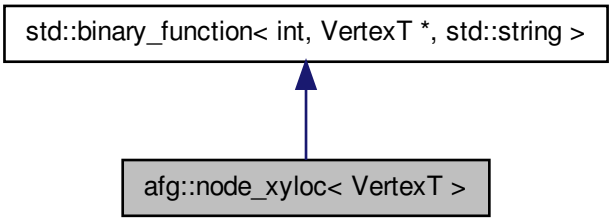
9.56 afg::node_xyloc< VertexT > Struct Template Reference

`#include <export_gdl.h>`

Inheritance diagram for afg::node_xyloc< VertexT >:



Collaboration diagram for afg::node_xyloc< VertexT >:



Public Member Functions

- node_xyloc (int nxs=10, int nys=10)
- std::string **operator()** (int, const VertexT ∗pe) const

Public Attributes

- int **xscale**
- int **yscale**

9.56.1 Detailed Description

template<class VertexT>struct afg::node_xyloc< VertexT >

return x,y location attribute for a node.

9.56.2 Constructor & Destructor Documentation

9.56.2.1 template<class VertexT > afg::node_xyloc< VertexT >::node_xyloc (int *nxs* = 10, int *nys* = 10) [inline]

Constructor. The vertex data of the vertex passed is assumed to have member functions x() and y() to access x/y coordinates. The original coordinates may be double, but will be converted to int (gdl file only support int location).

Parameters

nxs	scale factor for x coordinate; i.e., the x coordinate output to the file will be x∗nxs where x is the x coordinate contained in the vertex
nys	scale factor for y coordinate

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

- export_gdl.h

9.57 news::PktObj Class Reference

`#include <sscheduler.h>`

Public Member Functions

- virtual PktObj ∗ **clone** (void)=0

9.57.1 Detailed Description

basic definition for a packet object. All packet objects should be derived from it and implement the clone() member function.

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

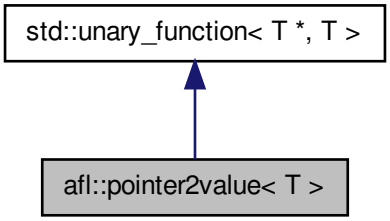
- sscheduler.h

9.58 afl::pointer2value< T > Struct Template Reference

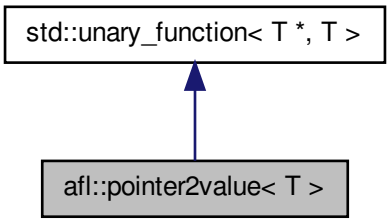
Operator ( ) returns the value of a pointer,.

`#include <util_tl.hpp>`

Inheritance diagram for afl::pointer2value< T >:



Collaboration diagram for afl::pointer2value< T >:



Public Member Functions

- T **operator()** (const T *pt) const

9.58.1 Detailed Description

template<class T>struct afl::pointer2value< T >

Operator ( ) returns the value of a pointer,.

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

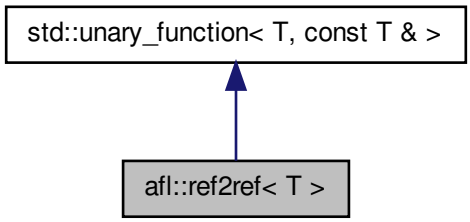
- util_tl.hpp

9.59 afl::ref2ref< T > Struct Template Reference

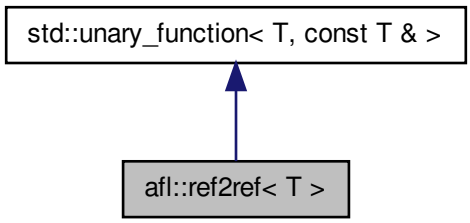
Operator ( ) returns a const reference of a reference.

```
#include <util_tl.hpp>
```

Inheritance diagram for afl::ref2ref< T >:



Collaboration diagram for afl::ref2ref< T >:



Public Member Functions

- const T & **operator()** (const T &x) const

9.59.1 Detailed Description

template<class T>struct afl::ref2ref< T >

Operator ( ) returns a const reference of a reference.

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

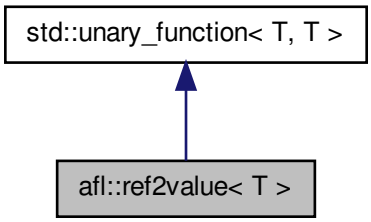
- util_tl.hpp

9.60 afl::ref2value< T > Struct Template Reference

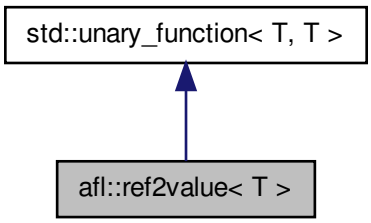
Operator ( ) returns the value of a reference.

`#include <util_tl.hpp>`

Inheritance diagram for afl::ref2value< T >:



Collaboration diagram for afl::ref2value< T >:



Public Member Functions

- T **operator()** (const T &x) const

9.60.1 Detailed Description

template<class T>struct afl::ref2value< T >

Operator ( ) returns the value of a reference.

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

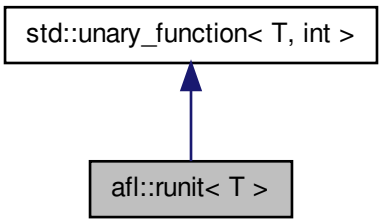
- util_tl.hpp

9.61 afl::runit< T > Struct Template Reference

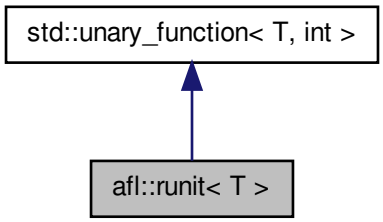
Operator( ) always returns 1 for any reference passed.

`#include <util_tl.hpp>`

Inheritance diagram for afl::runit< T >:



Collaboration diagram for afl::runit< T >:



Public Member Functions

- int **operator()** (const T &) const

9.61.1 Detailed Description

template<class T>struct afl::runit< T >

Operator( ) always returns 1 for any reference passed.

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

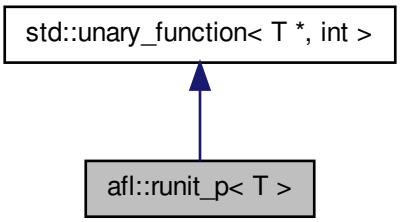
- util_tl.hpp

9.62 afl::runit_p< T > Struct Template Reference

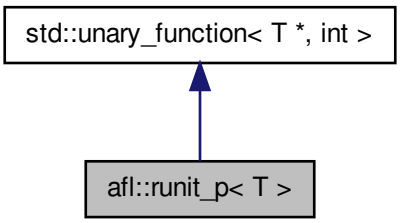
Operator( ) always returns 1 for any pointer passed.

`#include <util_tl.hpp>`

Inheritance diagram for afl::runit_p< T >:



Collaboration diagram for afl::runit_p< T >:



Public Member Functions

- int **operator()** (const T *) const

9.62.1 Detailed Description

template<class T>struct afl::runit_p< T >

Operator( ) always returns 1 for any pointer passed.

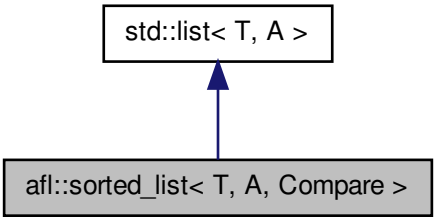
The documentation for this struct was generated from the following file:

- util_tl.hpp

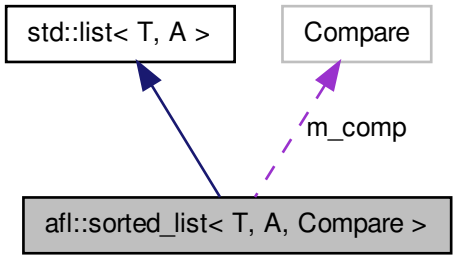
9.63 afl::sorted_list< T, A, Compare > Class Template Reference

`#include <sorted_list.hpp>`

Inheritance diagram for afl::sorted_list< T, A, Compare >:



Collaboration diagram for afl::sorted_list< T, A, Compare >:



Public Member Functions

- sorted_list (const Compare &comp=Compare())

  Constructor, default comparison predicate is less( ).
- void push (const T &x)

  Insert element x into the list, keep ordered property.

Protected Attributes

- Compare **m_comp**

9.63.1 Detailed Description

template<class T, class A = std::allocator<T>, class Compare = std::less<T>>class afl::sorted_list< T, A, Compare >

A list which is always sorted.

T is the data type of element kept in the list.

Provides: (1)ordered element access as provided by priority_queue; (2)access to element one by one through iterator.

By default, std::less( ) is used as comparison predicate when inserting new element, thus elements are sorted from small to large. Use front( ) to access the smallest onea and back( ) to access the largest one. Use pop_front( ) and pop_back( ) to remove the smallest and largest one, respectively.

sorted_list inherites from std::list, most member funcations of list are still available; however, in order to maintain the ordered property, position specific insert operations are not allowed and related functions (including the followings: push_front( ), push_back( ), insert(iterator pos, ...)) are overloaded as "private".

Version

0.1a, November 2000
0.2b, March 2001

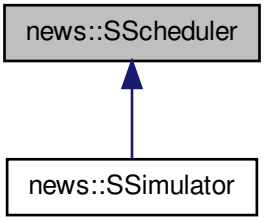
The documentation for this class was generated from the following file:

- sorted_list.hpp

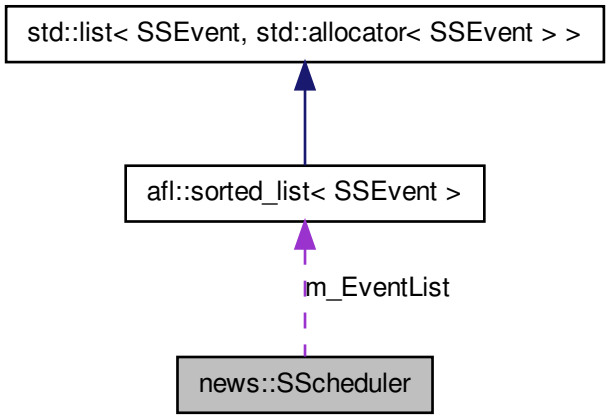
9.64 news::SScheduler Class Reference

```
#include <sscheduler.h>
```

Inheritance diagram for news::SScheduler:



Collaboration diagram for news::SScheduler:



Public Member Functions

- **SScheduler** (TimeS starttime)
- TimeS **get_time** ()

- bool **schedule_event** (SSEvent &event)
- bool **de_schedule_event** (SSEvent &event)

Protected Attributes

- TimeS **m_Time**
- afl::sorted_list< SSEvent > **m_EventList**

9.64.1 Detailed Description

Single object scheduler.

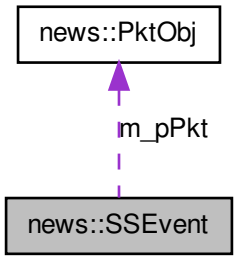
The documentation for this class was generated from the following file:

- sscheduler.h

9.65 news::SSEvent Class Reference

`#include <sscheduler.h>`

Collaboration diagram for news::SSEvent:



Public Member Functions

- **SSEvent** (int ntype=-1, TimeS stime=-1, TimeS etime=-1, PktObj *pp=NULL, int key=0)
- int **get_type** () const

- int **get_key** () const
- TimeS **get_time** () const
- TimeS **get_stime** () const
- PktObj * **get_pkt** ()

Friends

- class **SScheduler**
- bool **operator<** (const SSEvent &lhs, const SSEvent &rhs)
- bool **operator>** (const SSEvent &lhs, const SSEvent &rhs)
- bool **operator!=** (const SSEvent &lhs, const SSEvent &rhs)
- bool **operator==** (const SSEvent &lhs, const SSEvent &rhs)

9.65.1 Detailed Description

Event type for a single object scheduler.

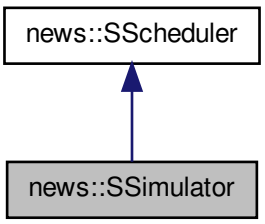
The documentation for this class was generated from the following file:

- sscheduler.h

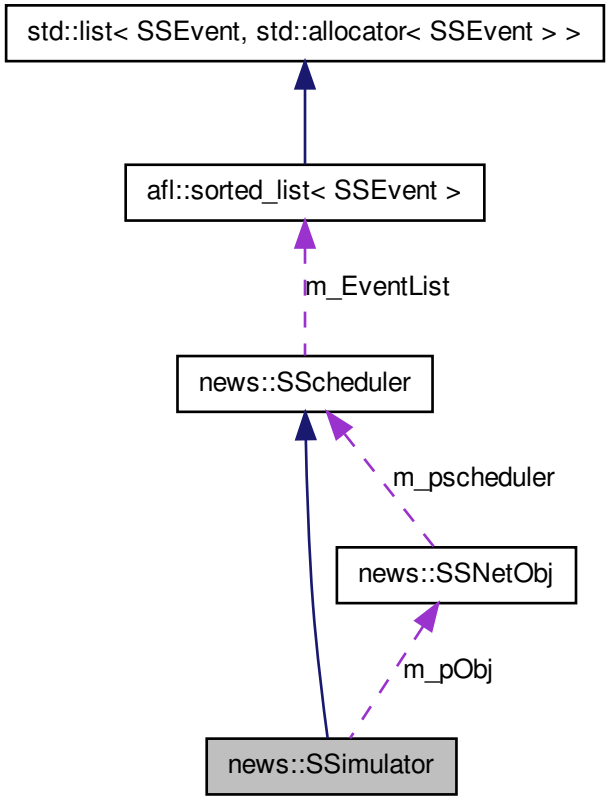
9.66 news::SSimulator Class Reference

`#include <sscheduler.h>`

Inheritance diagram for news::SSimulator:



Collaboration diagram for news::SSimulator:



Public Member Functions

- **SSimulator** (SSNetObj *pobj, TimeS stime, TimeS etime)
- bool **run** (void)

Protected Attributes

- SSNetObj * **m_pObj**
- TimeS **m_EndTime**

9.66.1 Detailed Description

Single object simulator. How to use:

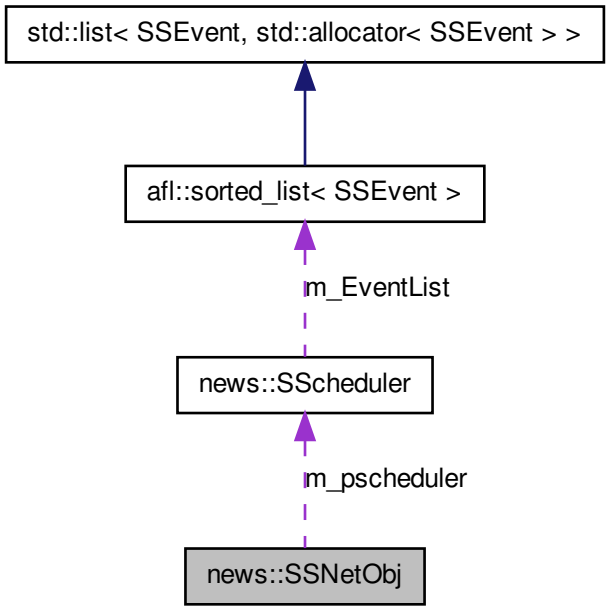
The documentation for this class was generated from the following file:

- sscheduler.h

9.67 news::SSNetObj Class Reference

`#include <sscheduler.h>`

Collaboration diagram for news::SSNetObj:



Public Member Functions

- virtual int **handle_event** (SSEvent &)=0

- virtual bool **init** (SScheduler *psch, TimeS ctime)
- virtual int **stop** (TimeS ctime)

Protected Attributes

- SScheduler * **m_pscheduler**

9.67.1 Detailed Description

Basic network object for the single object scheduler.

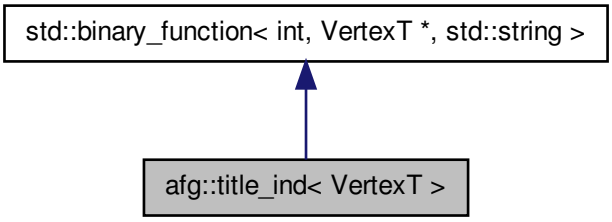
The documentation for this class was generated from the following file:

- sscheduler.h

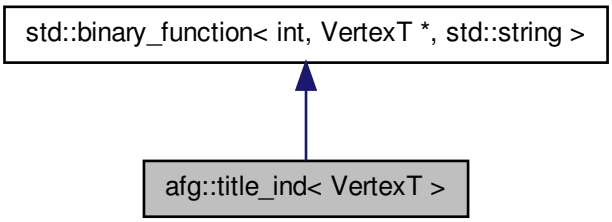
9.68 afg::title_ind< VertexT > Struct Template Reference

`#include <export_gdl.h>`

Inheritance diagram for afg::title_ind< VertexT >:



Collaboration diagram for afg::title_ind< VertexT >:



Public Member Functions

- std::string **operator()** (int i, const VertexT ∗) const

9.68.1 Detailed Description

template<class VertexT>struct afg::title_ind< VertexT >

return the intex of a node as its title.

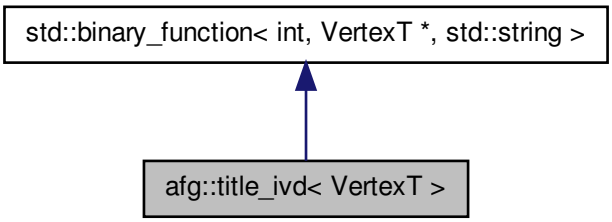
The documentation for this struct was generated from the following file:

- export_gdl.h

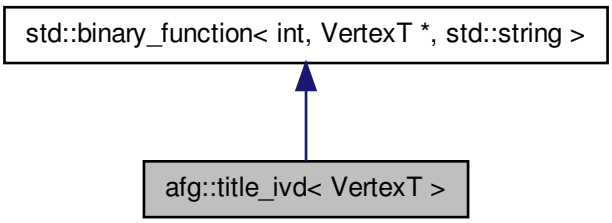
9.69 afg::title_ivd< VertexT > Struct Template Reference

```
#include <export_gdl.h>
```

Inheritance diagram for afg::title_ivd< VertexT >:



Collaboration diagram for afg::title_ivd< VertexT >:



Public Member Functions

- std::string **operator()** (int i, const VertexT *v) const

9.69.1 Detailed Description

template<class VertexT>struct afg::title_ivd< VertexT >

return the combination of index and data of a node as its title.

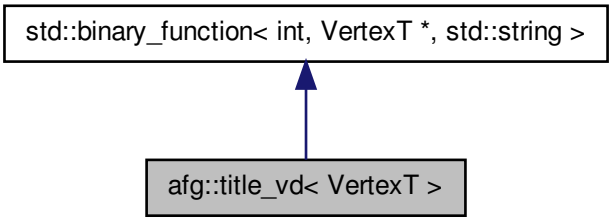
The documentation for this struct was generated from the following file:

- export_gdl.h

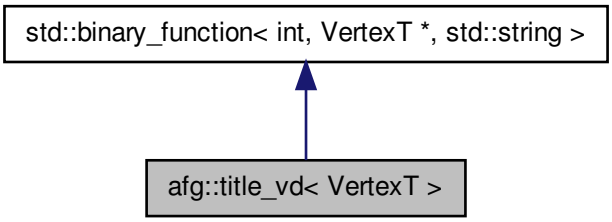
9.70 afg::title_vd< VertexT > Struct Template Reference

`#include <export_gdl.h>`

Inheritance diagram for afg::title_vd< VertexT >:



Collaboration diagram for afg::title_vd< VertexT >:



Public Member Functions

- std::string **operator()** (int, const VertexT *v) const

9.70.1 Detailed Description

template<class VertexT>struct afg::title_vd< VertexT >

return the data of a node as its title.

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

- export_gdl.h

9.71 afg::tree_dfs< TreeT > Struct Template Reference

```
#include <tree_alg.h>
```

Public Types

- typedef TreeT::EDT **EDT**

Public Member Functions

- tree_dfs (TreeT &t)
  constructor
- bool dfs_init (int ns)
- EDT * dfs_next (int &nsrc, int &ndest)

Protected Attributes

- TreeT & **tree**
- std::stack< std::pair< int, int > > **m_sdfsNodes**
- std::stack< EDT * > **m_sdfsEdges**

9.71.1 Detailed Description

template<class TreeT>struct afg::tree_dfs< TreeT >

Depth-first traversal of a tree, non-const version.

9.71.2 Member Function Documentation

9.71.2.1 template<class TreeT > bool afg::tree_dfs< TreeT >::dfs_init (int *ns*) [inline]

call this first before starting a depth-first of node traversal. It actually traverses the subtree rooted at node ns.

Parameters

ns	index of the starting node.

9.71.2.2 template<class TreeT > EDT* afg::tree_dfs< TreeT >::dfs_next (int & *nsrc*, int & *ndest*) [inline]

get the next edge/node of depth-first traversal.

Parameters

nsrc	reference to hold source node of the edge being visited.
ndest	reference to hold destination node of the edge being visited.

Returns

a pointer to the data of the edge being visited.

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

- tree_alg.h

9.72 afg::tree_dfs_c< TreeT > Struct Template Reference

`#include <tree_alg.h>`

Public Types

- typedef TreeT::EDT **EDT**

Public Member Functions

- tree_dfs_c (TreeT &t)

  constructor
- bool dfs_init (int ns)
- const EDT * dfs_next (int &nsrc, int &ndest)

Protected Attributes

- const TreeT & **tree**
- std::stack< std::pair< int, int > > **m_sdfsNodes**
- std::stack< const EDT * > **m_sdfsEdges**

9.72.1 Detailed Description

template<class TreeT>struct afg::tree_dfs_c< TreeT >

Depth-first traversal of a tree, const version.

9.72.2 Member Function Documentation

9.72.2.1 template<class TreeT > bool afg::tree_dfs_c< TreeT >::dfs_init (int *ns*) [inline]

call this first before starting a depth-first of node traversal. It actually traverses the subtree rooted at node ns.

Parameters

ns	index of the starting node.

9.72.2.2 template<class TreeT > const EDT* afg::tree_dfs_c< TreeT >::dfs_next (int & *nsrc*, int & *ndest*) [inline]

get the next edge/node of depth-first traversal.

Parameters

nsrc	reference to hold source node of the edge being visited.
ndest	reference to hold destination node of the edge being visited.

Returns

a pointer to the data of the edge being visited.

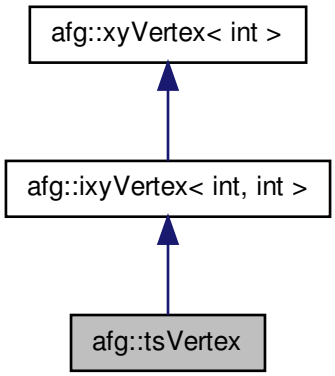
The documentation for this struct was generated from the following file:

- tree_alg.h

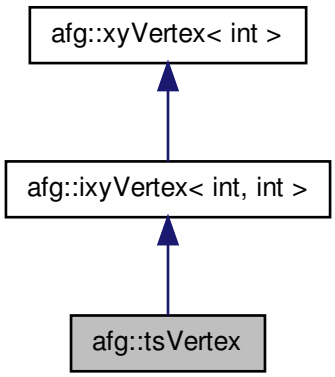
9.73 afg::tsVertex Struct Reference

`#include <vertex.h>`

Inheritance diagram for afg::tsVertex:



Collaboration diagram for afg::tsVertex:



Public Member Functions

- **tsVertex** (int i=-1, int nx=-1, int ny=-1, std::string l=std::string("S"))
- char type (void) const

Public Attributes

- std::string **sLabel**

9.73.1 Detailed Description

vertex for transit-stub graph.

9.73.2 Member Function Documentation

9.73.2.1 char afg::tsVertex::type (void) const [inline]

type of the node.

Returns

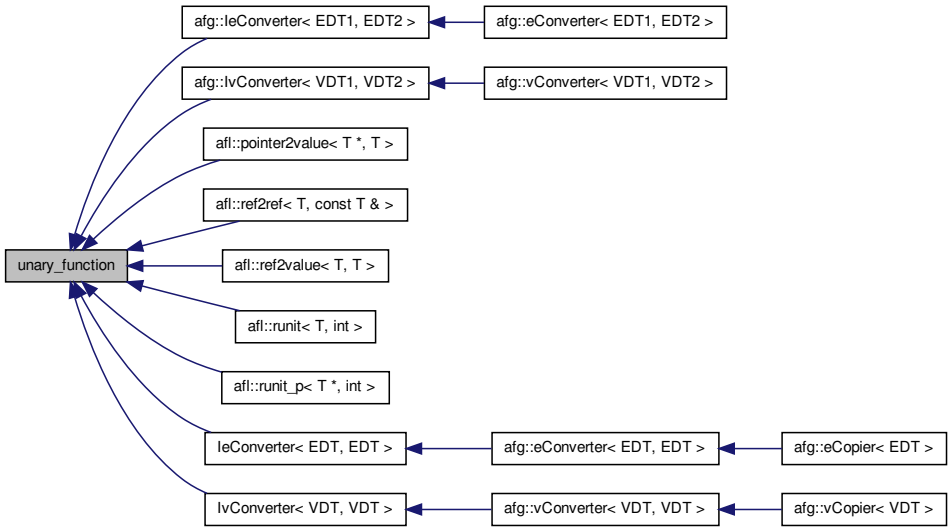
node type, 'T': tranist node, 'S': stub node

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

- vertex.h

9.74 unary_function Class Reference

Inheritance diagram for unary_function:



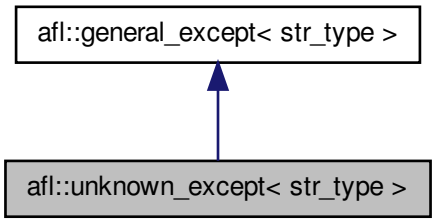
The documentation for this class was generated from the following file:

- graph_convert.h

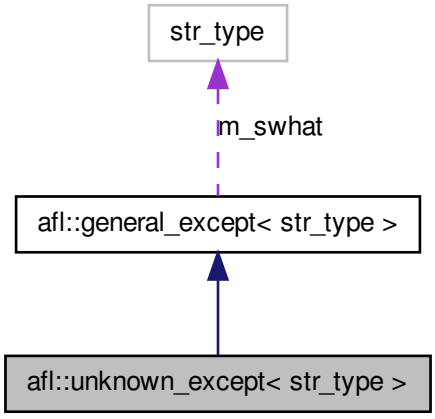
9.75 afl::unknown_except< str_type > Class Template Reference

```
#include <exceptions.hpp>
```

Inheritance diagram for afl::unknown_except< str_type >:



Collaboration diagram for afl::unknown_except< str_type >:



Public Member Functions

- **unknown_except** (const str_type &s=str_type("unknown exception"))

9.75.1 Detailed Description

template<class str_type = std::string>class afl::unknown_except< str_type >

Unknown exception. No specific information will be given for this exception.

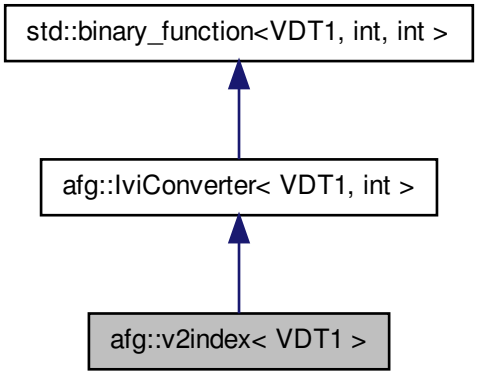
The documentation for this class was generated from the following file:

- exceptions.hpp

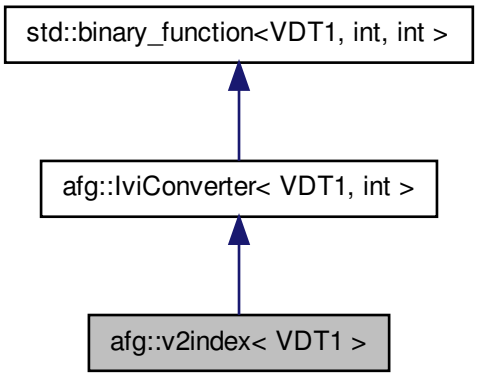
9.76 afg::v2index< VDT1 > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::v2index< VDT1 >:



Collaboration diagram for afg::v2index< VDT1 >:



Public Member Functions

- int **convert** (const VDT1 &, int i) const

9.76.1 Detailed Description

template<class VDT1>struct afg::v2index< VDT1 >

convert a vertex to a new int vertex using its original index.

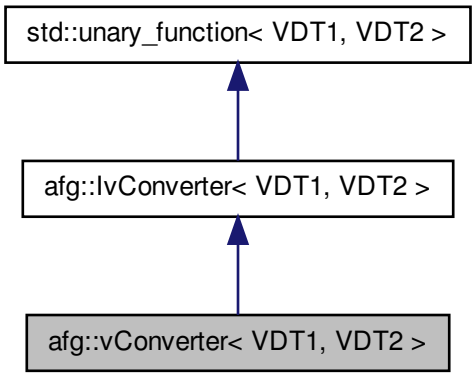
The documentation for this struct was generated from the following file:

- graph_convert.h

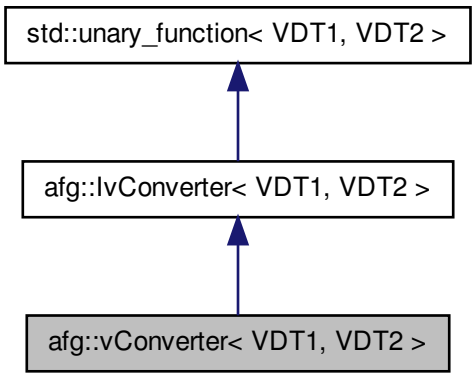
9.77 afg::vConverter< VDT1, VDT2 > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::vConverter< VDT1, VDT2 >:



Collaboration diagram for afg::vConverter< VDT1, VDT2 >:



Public Member Functions

- VDT2 **convert** (const VDT1 &v1) const

9.77.1 Detailed Description

template<class VDT1, class VDT2>struct afg::vConverter< VDT1, VDT2 >

vertex converter that just does an explicit type conversion.

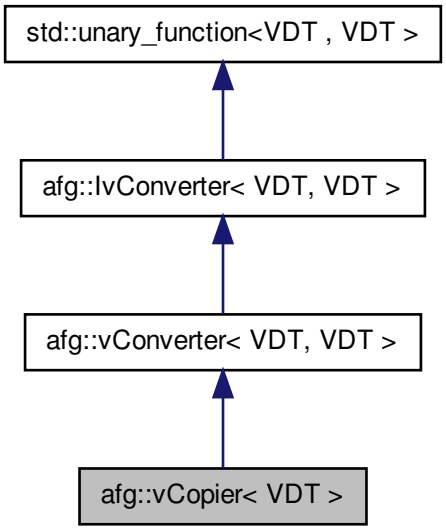
The documentation for this struct was generated from the following file:

- graph_convert.h

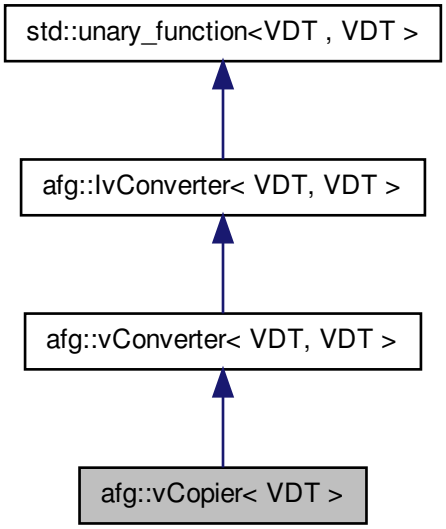
9.78 afg::vCopier< VDT > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::vCopier< VDT >:



Collaboration diagram for afg::vCopier< VDT >:



Public Member Functions

- VDT **convert** (const VDT &v1) const

9.78.1 Detailed Description

template<class VDT>struct afg::vCopier< VDT >

vertex converter that just makes a copy.

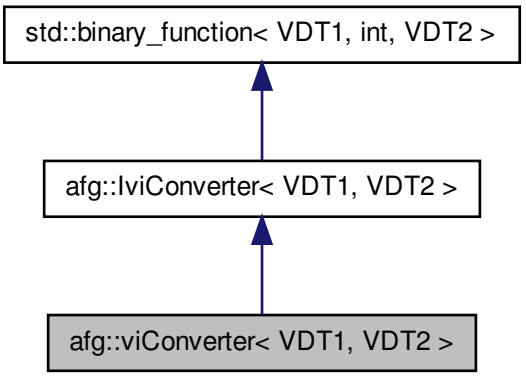
The documentation for this struct was generated from the following file:

- graph_convert.h

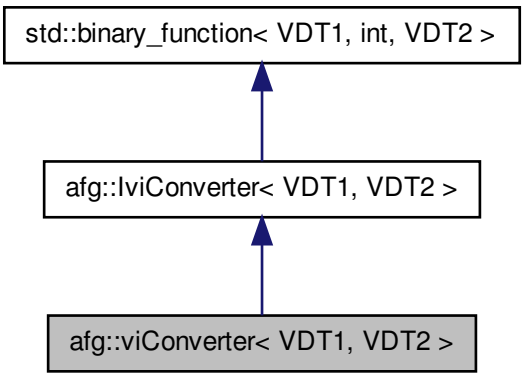
9.79 afg::viConverter< VDT1, VDT2 > Struct Template Reference

```
#include <graph_convert.h>
```

Inheritance diagram for afg::viConverter< VDT1, VDT2 >:



Collaboration diagram for afg::viConverter< VDT1, VDT2 >:



Public Member Functions

- VDT2 **convert** (const VDT1 &v1, int) const

9.79.1 Detailed Description

template<class VDT1, class VDT2>struct afg::viConverter< VDT1, VDT2 >

vertex converter with index, just does an explicit type conversion.

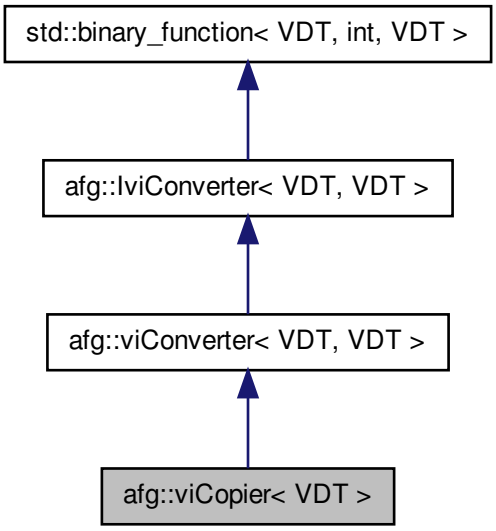
The documentation for this struct was generated from the following file:

- graph_convert.h

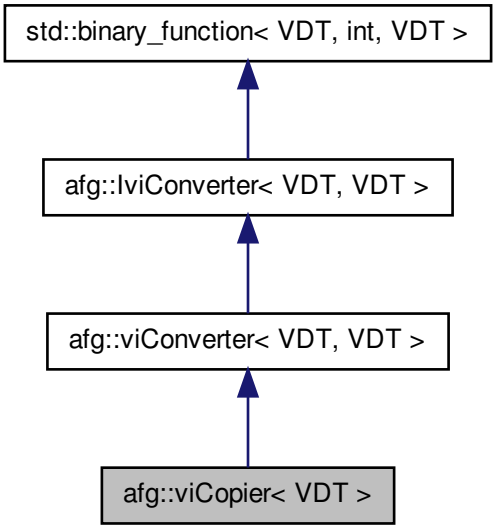
9.80 afg::viCopier< VDT > Struct Template Reference

`#include <graph_convert.h>`

Inheritance diagram for afg::viCopier< VDT >:



Collaboration diagram for afg::viCopier< VDT >:



Public Member Functions

- VDT **convert** (const VDT &v1, int) const

9.80.1 Detailed Description

template<class VDT>struct afg::viCopier< VDT >

vertex converter with index, just makes a copy.

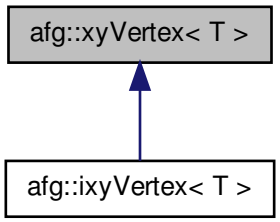
The documentation for this struct was generated from the following file:

- graph_convert.h

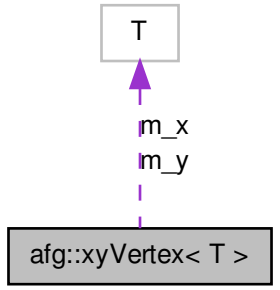
9.81 afg::xyVertex< T > Struct Template Reference

```
#include <vertex.h>
```

Inheritance diagram for afg::xyVertex< T >:



Collaboration diagram for afg::xyVertex< T >:



Public Member Functions

- **xyVertex** (T tx=(T)-1, T ty=(T)-1)
- T & **x** (void)
- T **x** (void) const
- T & **y** (void)
- T **y** (void) const

Public Attributes

- T m_x
  data member, x coordinate
- T m_y
  data member, y coordinate

9.81.1 Detailed Description

template<class T>struct afg::xyVertex< T >

vertex with x,y coordinates only. T is the type of the coordinate, normally integer or double.

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

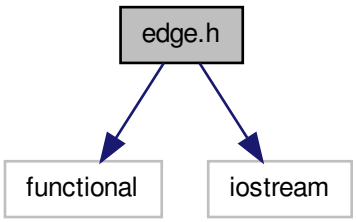
- vertex.h

10 File Documentation

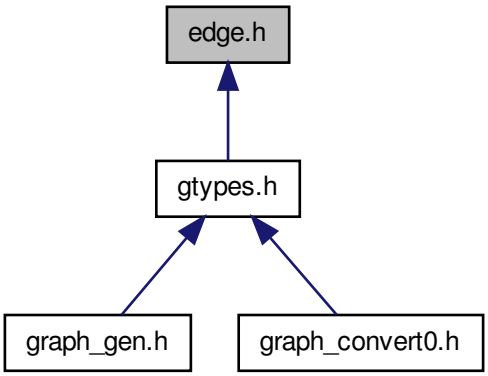
10.1 edge.h File Reference

```
#include <functional>
#include <iostream>
```

Include dependency graph for edge.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct afg::Ce_nCare
- struct afg::CEdgeW2< WType1, WType2 >
- struct afg::CEdgeW3< WType1, WType2, WType3 >

Namespaces

- namespace afg

Functions

stream input/output opertors

- std::ostream & **afg::operator**<< (std::ostream &os, const Ce_nCare &edge)
- template<class WType1 , class WType2 >
  std::ostream & afg::operator<< (std::ostream &os, const CEdgeW2< WType1, WType2 > &edge)
- template<class WType1 , class WType2 >
  std::istream & afg::operator>> (std::istream &is, CEdgeW2< WType1, WType2 > &edge)

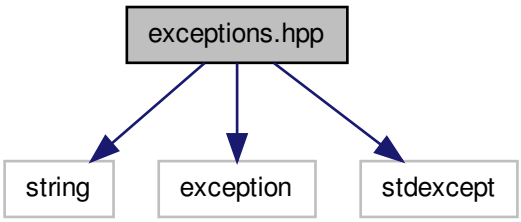
- template<class WType1 , class WType2 , class WType3 >
  std::ostream & afg::operator<< (std::ostream &os, const CEdgeW3< WType1, WType2, WType3 > &edge)
- template<class WType1 , class WType2 , class WType3 >
  std::istream & afg::operator>> (std::istream &is, CEdgeW3< WType1, WType2, WType3 > &edge)

10.1.1 Detailed Description

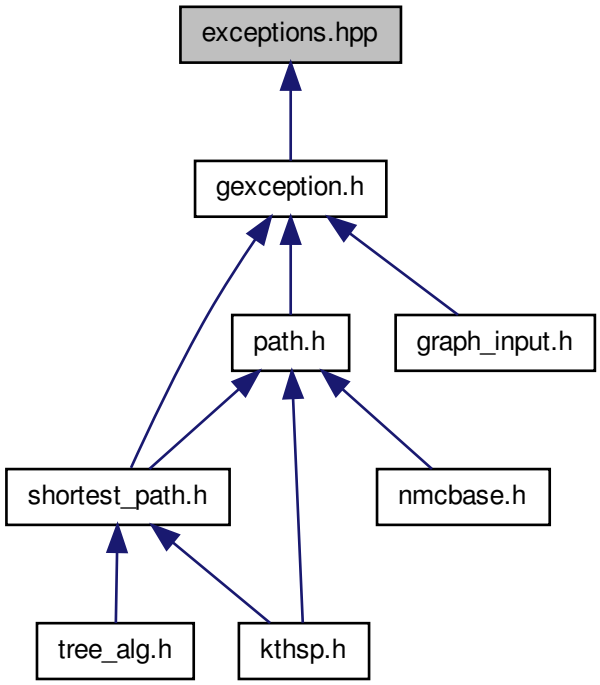
10.2 exceptions.hpp File Reference

```
#include <string>
#include <exception>
#include <stdexcept>
```

Include dependency graph for exceptions.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class afl::general_except< str_type >

  General exception.
- class afl::unknown_except< str_type >
- class afl::init_failed< str_type >

  Initialization failure exception.
- class afl::mem_alloc_failed< str_type >

  Memory allocation failure exception.

10.2.1 Detailed Description

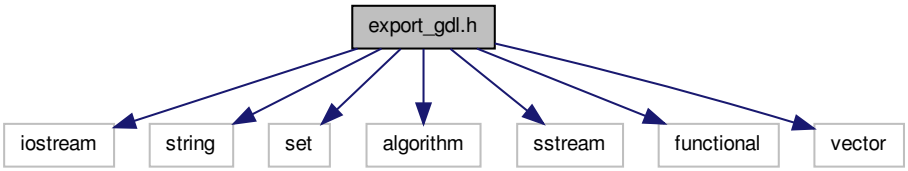
Exception definitions.

Aiguo Fei, 2000-2001

10.3 export_gdl.h File Reference

```
#include <iostream>
#include <string>
#include <set>
#include <algorithm>
#include <sstream>
#include <functional>
#include <vector>
```

Include dependency graph for export_gdl.h:



Classes

- struct afg::title_ind< VertexT >
- struct afg::title_vd< VertexT >
- struct afg::title_ivd< VertexT >
- struct afg::node_empty< VertexT >
- struct afg::node_combine< VertexT, F1, F2 >
- struct afg::node_xyloc< VertexT >
- struct afg::node_color< VertexT >
- struct afg::edge_empty< EdgeT >
- struct afg::edge_label_ed< EdgeT >

- struct afg::edge_combine< EdgeT, F1, F2 >
- class afg::export_gdl< GraphT, Ftitle, Fv_optional, Fe_optional >

Namespaces

- namespace afg

10.3.1 Detailed Description

Export gdl (graph description language) file that can be imported to other graph visualization tools.

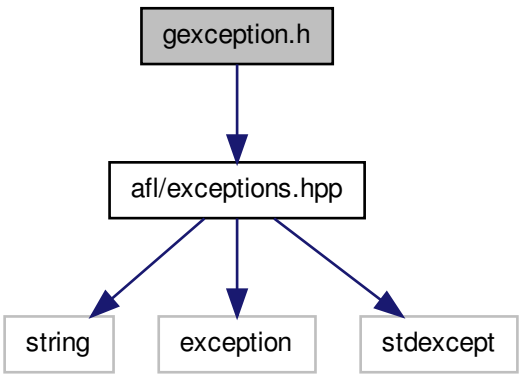
Aiguo Fei

2001/03/02, 03/14

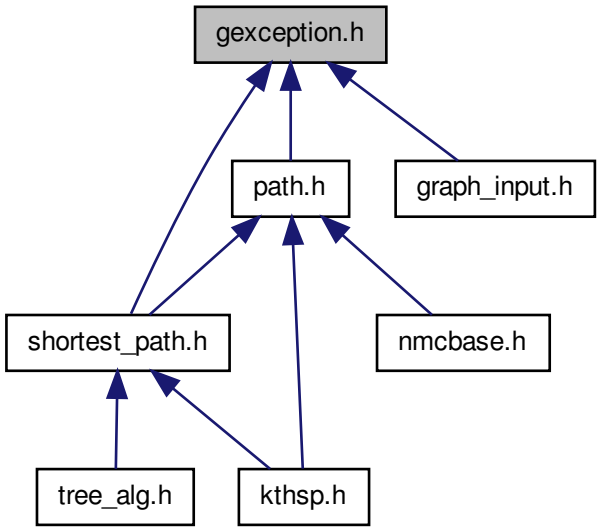
10.4 gexception.h File Reference

```
#include "afl/exceptions.hpp"
```

Include dependency graph for gexception.h:



This graph shows which files directly or indirectly include this file:



Classes

- class afg::invalid_path

  invalid path.

Namespaces

- namespace afg

10.4.1 Detailed Description

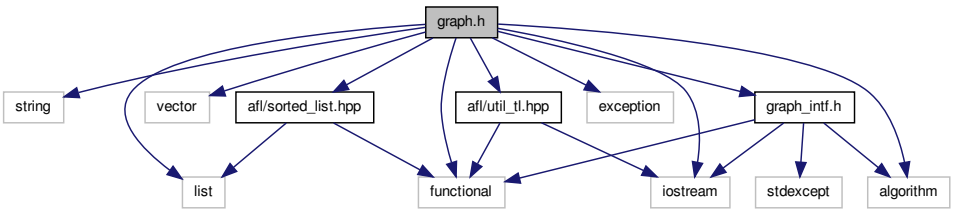
Exception definitions.

Aiguo Fei, 2000-2001

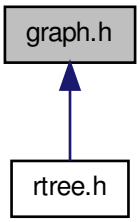
10.5 graph.h File Reference

```
#include <string>
#include <iostream>
#include <vector>
#include <list>
#include <functional>
#include <algorithm>
#include <exception>
#include "afl/util_tl.hpp"
#include "afl/sorted_list.hpp"
#include "graph_intf.h"
```

Include dependency graph for graph.h:



This graph shows which files directly or indirectly include this file:



Classes

- class afg::CiEdge< EdgeDT >
- class afg::CiVertex< VertexDT, EdgeDT >
- class afg::CGraph< VertexDT, EdgeDT, f_eqv >

Namespaces

- namespace afg

Defines

- #define graph_type afg::CGraph

  type of the graph provided by this file
- #define **_GRAPHDEFINED**

Functions

stream input/output operators

- template<class EdgeDT >
  std::istream & afg::operator>> (std::istream &is, CiEdge< EdgeDT > &edge)

10.5.1 Detailed Description

Aiguo Fei

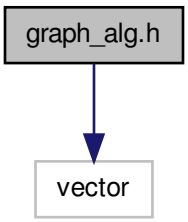
1999-2001

2001/02/19: bug fix, CGraph::pack()

10.6 graph_alg.h File Reference

```
#include <vector>
```

Include dependency graph for graph_alg.h:



Namespaces

- namespace afg

Functions

- template<class GT >
  bool afg::is_connected (const GT &grf)
- template<class GraphT , class Fun >
  Fun::result_type afg::graph_cost (const GraphT &g, Fun f_w)

10.6.1 Detailed Description

some graph algorithms

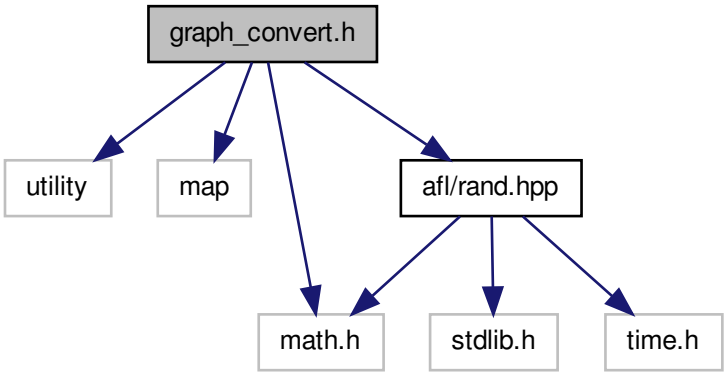
Aiguo Fei

1999-2001

10.7 graph_convert.h File Reference

```
#include <utility>
#include <map>
#include <math.h>
#include "afl/rand.hpp"
```

Include dependency graph for graph_convert.h:



Classes

- struct afg::lvConverter< VDT1, VDT2 >
- struct afg::lviConverter< VDT1, VDT2 >
- struct afg::leConverter< EDT1, EDT2 >
- struct afg::leiConverter< EDT1, EDT2 >
- struct afg::vConverter< VDT1, VDT2 >
- struct afg::vCopier< VDT >
- struct afg::viConverter< VDT1, VDT2 >
- struct afg::viCopier< VDT >

- struct afg::eConverter< EDT1, EDT2 >
- struct afg::eCopier< EDT >
- struct afg::eiConverter< EDT1, EDT2 >
- struct afg::eiCopier< EDT >
- struct afg::v2index< VDT1 >
- struct afg::e_add_dist< GT, DT >
- struct afg::e_add_rand< EDT1, DT >
- struct afg::e_add_rand_s< EDT1, DT >

Namespaces

- namespace afg

Functions

- template<class GT1 , class GT2 , class Vconvert , class Econvert >
  void afg::graph_convert (const GT1 &g1, GT2 &g2, Vconvert f_vc, Econvert f_ec)

10.7.1 Detailed Description

Graph convertors.

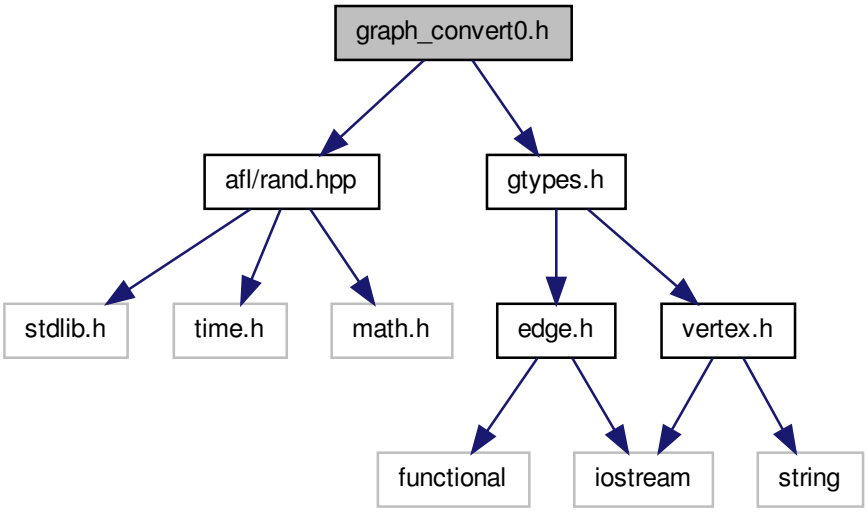
Aiguo Fei

UCLA, 1999-2001

10.8 graph_convert0.h File Reference

```
#include "afl/rand.hpp"
#include "gtypes.h"
```

Include dependency graph for graph_convert0.h:



Namespaces

- namespace afg

Functions

- template<class VDT , class WT1 >
  void afg::grfw2grfw2 (const graph_type< VDT, WT1 > &gra0, graph_type< VDT, CEdgeW2< WT1, double > > &ngra)
- template<class VDT , class WT1 >
  void afg::grfw2grfw2_s (const graph_type< VDT, WT1 > &gra0, graph_type< VDT, CEdgeW2< WT1, double > > &ngra)

10.8.1 Detailed Description

Some old graph convertors.

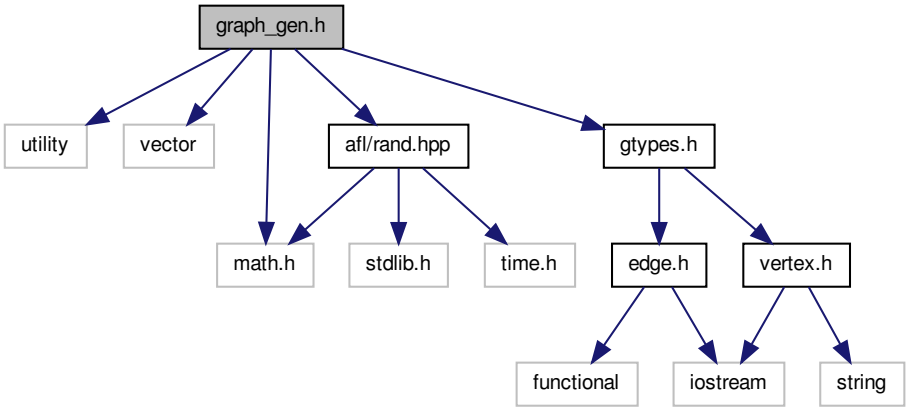
Aiguo Fei

UCLA, 1999-2001

10.9 graph_gen.h File Reference

```
#include <utility>
#include <vector>
#include <math.h>
#include "afl/rand.hpp"
#include "gtypes.h"
```

Include dependency graph for graph_gen.h:



Namespaces

- namespace afg

Functions

- void afg::grid_graph_gen (T_xynGraph &grf, int n, int m)
- int afg::bmw_graph_gen (T_xydGraph &dgrf, int ngrid, int n, int ntry=10, double lamda=0.3, double rou=0.3)

10.9.1 Detailed Description

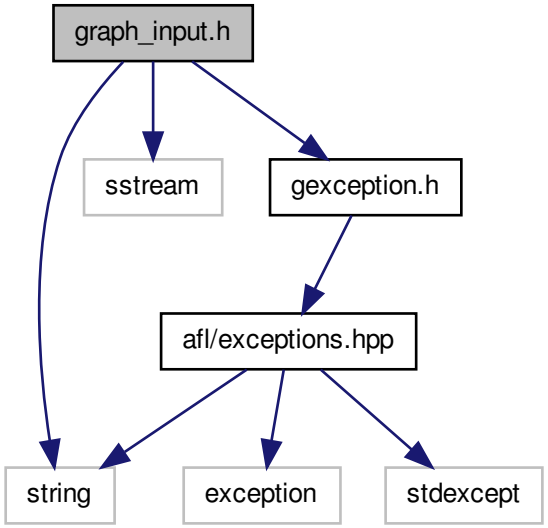
Graph generators.

Aiguo Fei UCLA, 1999, 2000, 2001

10.10 graph_input.h File Reference

```
#include <string>
#include <sstream>
#include "gexception.h"
```

Include dependency graph for graph_input.h:



Namespaces

- namespace afg

Functions

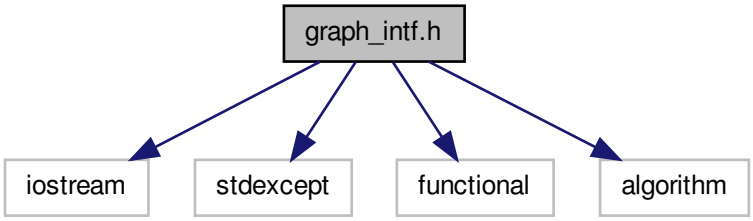
- void **afg::input_gra_header** (std::istream &is, std::string &sformat, int &nn, int &nl, std::string &sdir)
- template<class GT >
  void afg::input_gra_die (GT &gra, std::istream &is)
- template<class GT >
  void afg::input_gra_vie (GT &gra, std::istream &is)
- template<class GT , class VIDT >
  void afg::input_gra_vve (GT &gra, std::istream &is)
- template<class GT >
  bool **afg::input_gra_i_2e** (GT &gra, std::istream &is)

10.10.1 Detailed Description

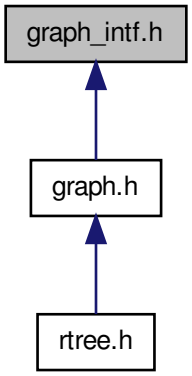
10.11 graph_intf.h File Reference

```
#include <iostream>
#include <stdexcept>
#include <functional>
#include <algorithm>
```

Include dependency graph for graph_intf.h:



This graph shows which files directly or indirectly include this file:



Classes

- class afg::IEdge< EdgeDT >
- struct afg::equal_i< EDT >

  predicate to test if the indices of two edges are the same.
- struct afg::equal_n< EDT >

  predicate to test if the index of an edge is equal to an integer.
- struct afg::Cvoid
- class afg::IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type >
- class afg::IGraph< VertexDT, EdgeDT, f_eqv, v_iterator_type, const_v_iterator_type, CVertex >

Namespaces

- namespace afg

Functions

- template<class EdgeDT >
  bool afg::operator== (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

equal operator for IEdge

- template<class EdgeDT >
  bool afg::operator!= (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  not_equal operator for IEdge
- template<class EdgeDT >
  bool afg::operator< (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  less operator for IEdge
- template<class EdgeDT >
  bool afg::operator> (const IEdge< EdgeDT > &lhs, const IEdge< EdgeDT > &rhs)

  larger operator for IEdge
- template<class EdgeDT >
  std::ostream & afg::operator<< (std::ostream &os, const IEdge< EdgeDT > &edge)

  conventional output operator for IEdge
- template<class VertexDT , class EdgeDT , class iterator_type , class const_iterator_type >
  std::ostream & afg::operator<< (std::ostream &os, const IVertex< VertexDT, EdgeDT, iterator_type, const_iterator_type > &rhs)

  output operator for IVertex
- template<class VertexDT , class EdgeDT , class iterator_type , class const_v_iterator_type , class f_eqv , class CVertex >
  std::ostream & afg::operator<< (std::ostream &os, const IGraph< VertexDT, EdgeDT, iterator_type, const_v_iterator_type, f_eqv, CVertex > &rhs)

  conventional output operator for IGraph

10.11.1 Detailed Description

defintions of interface classes IEdge, IVertex and IGraph

see also graph.h

todo: design documentation, decision on vertex iterator (allow invalid vertex or not) ?add explicit un-directed graph support (by conditional compiling?)

Aiguo Fei

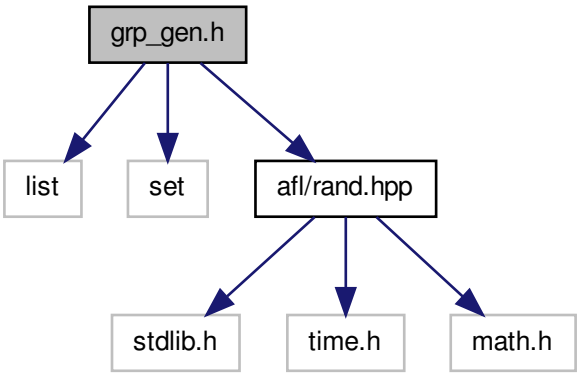
1999, 2000, 2001

10.12 grp_gen.h File Reference

```
#include <list>
#include <set>
```

```
#include "afl/rand.hpp"
```

Include dependency graph for grp_gen.h:



Classes

- struct news::CConstGrpSz
- struct news::CUniformGrpSz
- struct news::CGrpGenerator
- class news::CRandGrpX< GrpSzG >
- class news::CRandGrpG
- class news::CRandGrpG2< GraphT >
- class news::CWeightedGrpG< GraphT, GrpSzG >

Namespaces

- namespace news

Functions

- template<class BT >
  void news::rand_seq_gen (const BT &base, double alpha, int ns, int nt, std::list< std::pair< int, int > > &lact, std::set< int > &sdest)

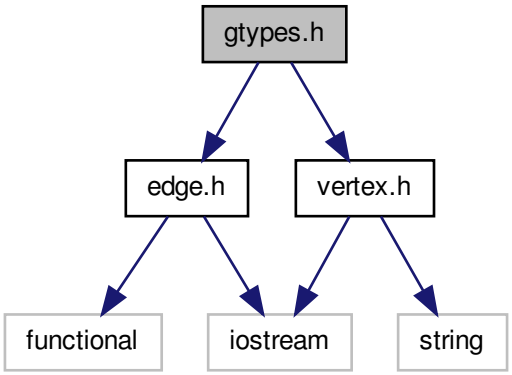
10.12.1 Detailed Description

generate groups

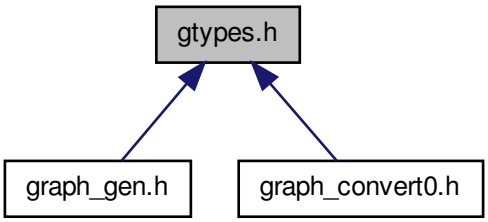
10.13 gtypes.h File Reference

```
#include "edge.h"
#include "vertex.h"
```

Include dependency graph for gtypes.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace afg

Typedefs

- typedef CEdgeW2< double, double > afg::T_d2Edge
  edge type with 2 double weights
- typedef CEdgeW3< double, double, double > afg::T_d3Edge
  edge type with 3 double weights
- typedef graph_type< int, Ce_nCare > afg::T_nGraph
  graph with an integer id only
- typedef graph_type< int, double > afg::T_ndGraph
  graph with an integer id and a double weight
- typedef graph_type< xyVertex< int >, double > afg::T_xydGraph
  graph with integer xyVertex and a double weight
- typedef graph_type< xyVertex< int >, int > afg::T_xynGraph
  graph with integer xyVertex and an integer weight
- typedef graph_type< int, T_d2Edge > afg::T_nd2Graph
  graph with an integer index and an edge of two double numbers
- typedef graph_type< xyVertex< int >, T_d2Edge > afg::T_xyd2Graph
  graph with integer xyVertex and an edge of two double numbers
- typedef graph_type< tsVertex, int > afg::T_tsGraph
  transit-stub graph

- typedef rtree_type< int, Ce_nCare > afg::T_nrTree

  rtree with an integer id only
- typedef rtree_type< int, double > afg::T_ndrTree

  rtree with an integer id and a double weight
- typedef rtree_type< xyVertex< int >, double > afg::T_xydrTree

  rtree with integer xyVertex and a double weight
- typedef rtree_type< xyVertex< int >, int > afg::T_xynrTree

  rtree with integer xyVertex and an integer weight
- typedef rtree_type< int, T_d2Edge > afg::T_nd2rTree

  rtree with an integer index and an edge of two double numbers
- typedef rtree_type< xyVertex< int >, T_d2Edge > afg::T_xyd2rTree

  rtree with integer xyVertex and an edge of two double numbers

10.13.1 Detailed Description

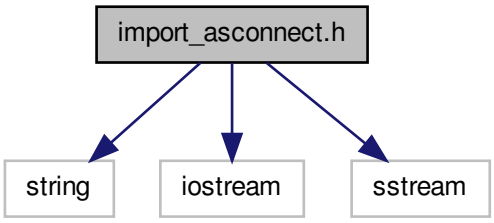
some graph related typde definitions

Aiguo Fei, UCLA 2000

10.14 import_asconnect.h File Reference

```
#include <string>
#include <iostream>
#include <sstream>
```

Include dependency graph for import_asconnect.h:



Namespaces

- namespace afg

Functions

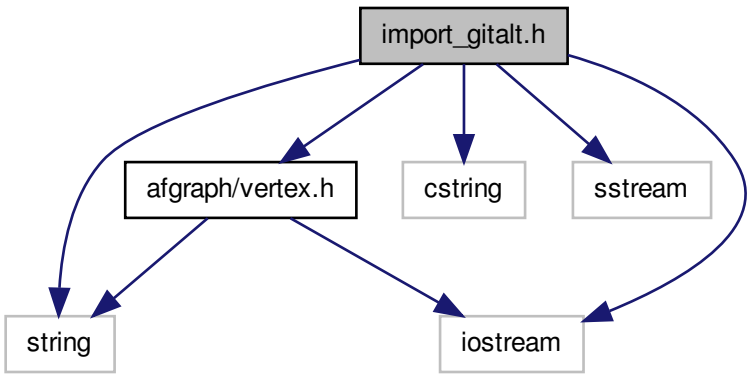
- template<class GT >
  void afg::import_asconnect (GT &gra, std::istream &is)

10.14.1 Detailed Description

10.15 import_gitalt.h File Reference

```
#include <string>
#include <iostream>
#include <cstring>
#include <sstream>
#include "afgraph/vertex.h"
```

Include dependency graph for import_gitalt.h:



Namespaces

- namespace afg

Functions

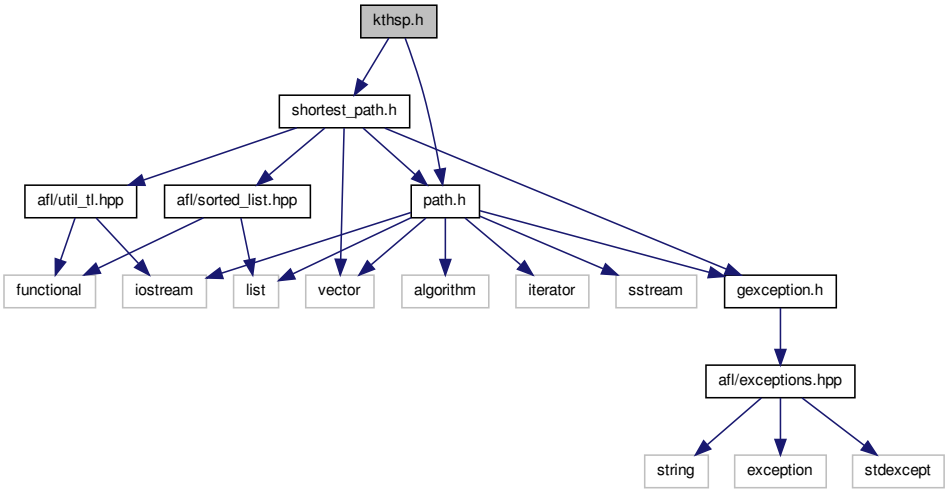
- template<class GT >
  bool afg::import_gitalt_ts (GT &gra, std::istream &is)

10.15.1 Detailed Description

10.16 kthsp.h File Reference

```
#include "path.h"
#include "shortest_path.h"
```

Include dependency graph for kthsp.h:



Classes

- class afg::CkthSP< GraphT, Fun >

Namespaces

- namespace afg

10.16.1 Detailed Description

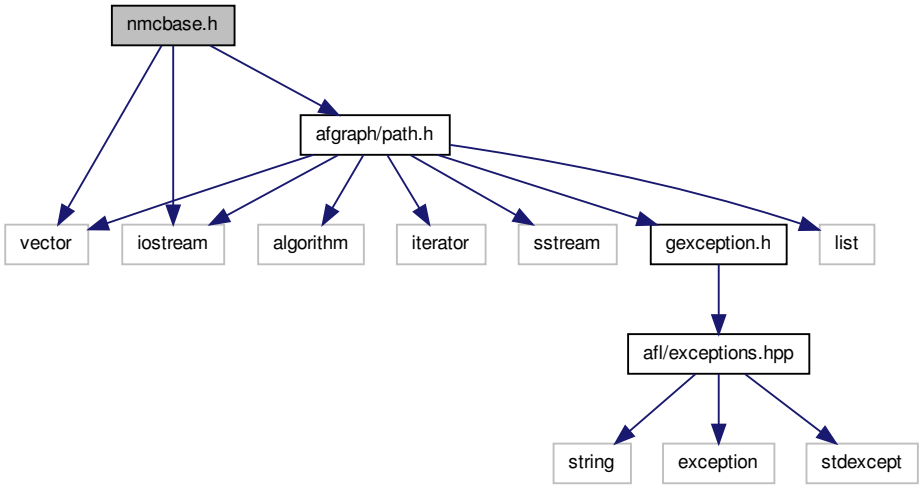
the kth shortest path algorithm

Aiguo Fei, UCLA 2000

10.17 nmcbase.h File Reference

```
#include <vector>
#include <iostream>
#include "afgraph/path.h"
```

Include dependency graph for nmcbase.h:



Classes

- class afg::CnmcBase< GT, WT1, WT2, WT3 >

Namespaces

- namespace afg

10.17.1 Detailed Description

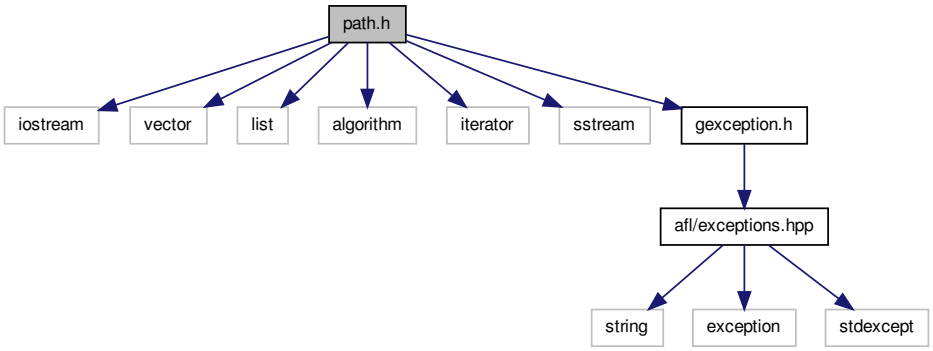
network multicast base class

Aiguo Fei 2000, 2001

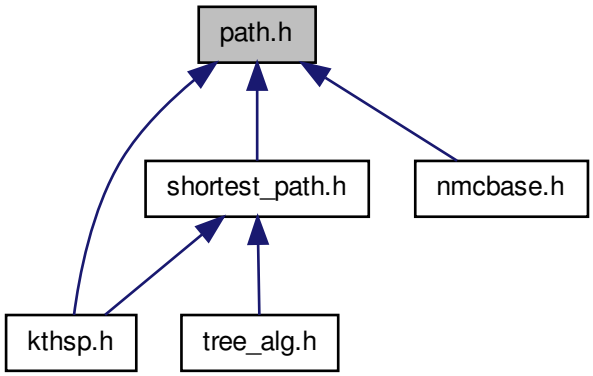
10.18 path.h File Reference

```
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>
#include <iterator>
#include <sstream>
#include "gexception.h"
```

Include dependency graph for path.h:



This graph shows which files directly or indirectly include this file:



Classes

- class afg::CPath

Namespaces

- namespace afg

Functions

path related utility functions

- std::ostream & afg::operator<< (std::ostream &os, CPath &lp)

  output operator to output path as a list of indices separated by space.
- bool afg::pred2path (int n, int *pnpred, int ns, int nt, CPath &lp)
- bool afg::pred2path (const std::vector< int > &vpred, int ns, int nt, CPath &lp)
- bool afg::allpred2path (int n, const std::vector< int > &vpred, int ns, int nt, CPath &lp)
- template<class GraphT , class Fun >
  Fun::result_type afg::path_length (const GraphT &gra, const CPath &lp, Fun fw)

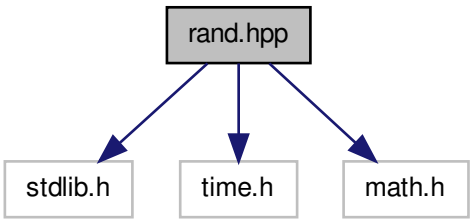
10.18.1 Detailed Description

helper class and functions for path manipulation.

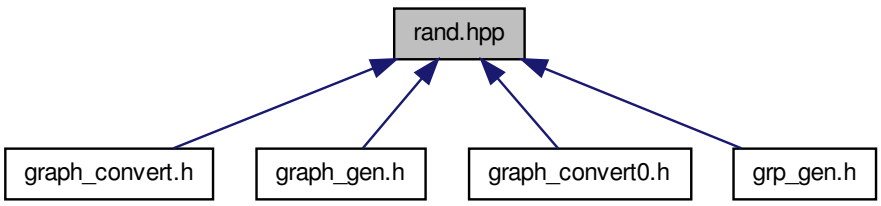
10.19 rand.hpp File Reference

```
#include <stdlib.h>
#include <time.h>
#include <math.h>
```

Include dependency graph for rand.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct afl::rand::CConst
- struct afl::rand::CUniform
- struct afl::rand::CExp

Functions

- void afl::rand::init (int n=111193)
- double afl::rand::gen (double d1, double d2)
- double afl::rand::gen (double d=1.0)

10.19.1 Detailed Description

Author

Aiguo Fei

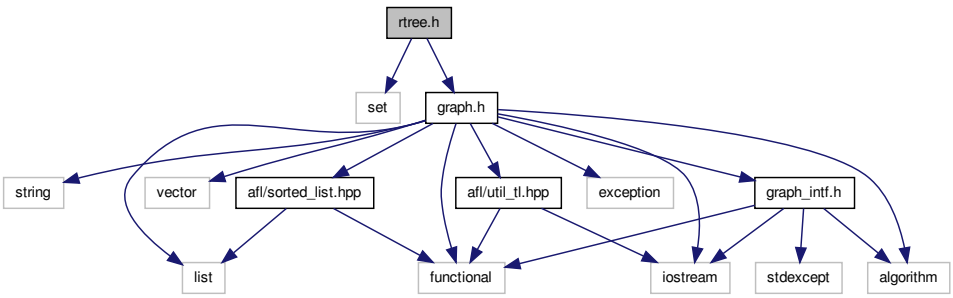
Version

1.0b, 1998
1.1, 2012 Random number related utilities.

10.20 rtree.h File Reference

```
#include <set>
#include "graph.h"
```

Include dependency graph for rtree.h:



Classes

- class afg::CrTree< VDT, EDT, f_eqv >

Namespaces

- namespace afg

Defines

- #define rtree_type afg::CrTree

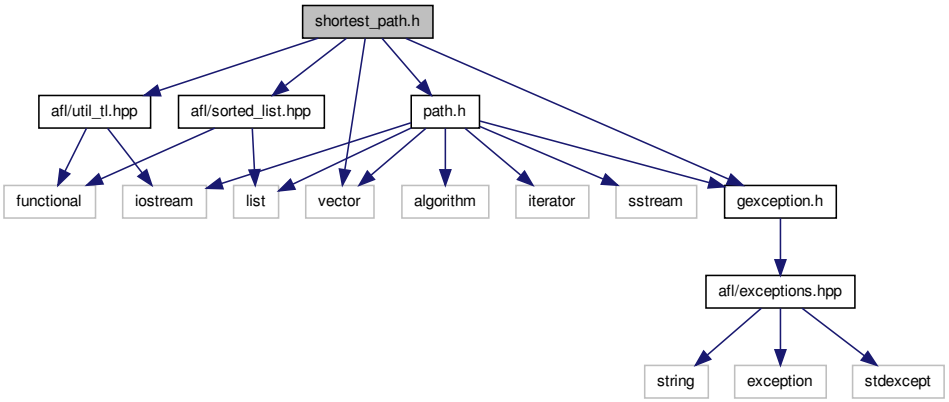
  type of rooted tree provided by rtree.h
- #define **_RTREEDEFINED**

10.20.1 Detailed Description

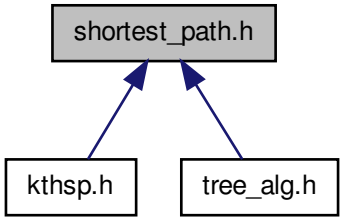
10.21 shortest_path.h File Reference

```
#include <vector>
#include "afl/util_tl.hpp"
#include "afl/sorted_list.hpp"
#include "path.h"
#include "gexception.h"
```

Include dependency graph for shortest_path.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace afg

Functions

- template<class GraphT , class Fun >

bool afg::dijkstra (const GraphT &graph, int nsource, Fun f_weight, typename Fun::result_type w_infty, std::vector< int > &pred, std::vector< typename Fun::result_type > &dist)

- template<class GraphT , class Fun >
  Fun::result_type afg::dijkstra_t (const GraphT &graph, int nsource, int ndst, Fun f_weight, typename Fun::result_type w_infty, CPath &lp)
- template<class GraphT , class Fun >
  bool afg::floyd_warshall_allsp (const GraphT &graph, Fun f_weight, typename Fun::result_type w_infty, std::vector< int > &pred, std::vector< typename Fun::result_type > &dist)

10.21.1 Detailed Description

shortest-path algorithms: dijkstra's shortest path algorithm

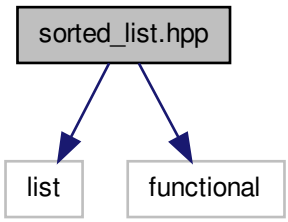
Aiguo Fei

1999, 2000, 2001

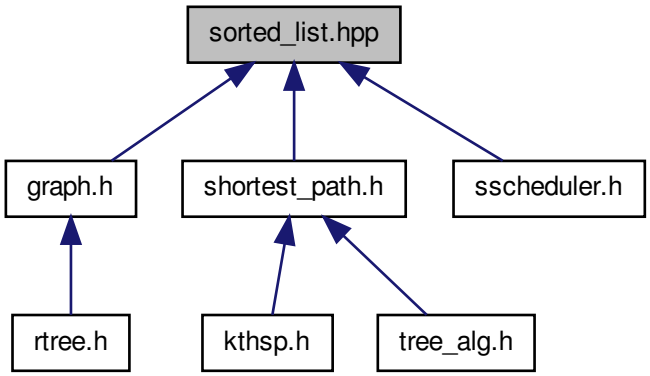
10.22 sorted_list.hpp File Reference

```
#include <list>
#include <functional>
```

Include dependency graph for sorted_list.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class afl::sorted_list< T, A, Compare >

10.22.1 Detailed Description

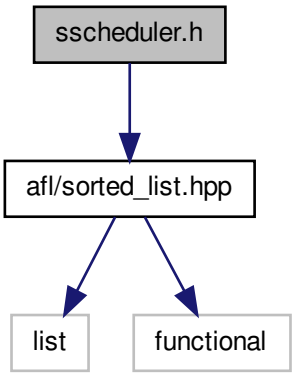
Author

Aiguo Fei

10.23 sscheduler.h File Reference

```
#include <afl/sorted_list.hpp>
```

Include dependency graph for sscheduler.h:



Classes

- class news::PktObj
- class news::SSEvent
- class news::SScheduler
- class news::SSNetObj
- class news::SSimulator

Namespaces

- namespace news

Typedefs

- typedef double news::TimeS

  timestamp
- typedef double news::TimeE

  elapsed time

10.23.1 Detailed Description

single object scheduler

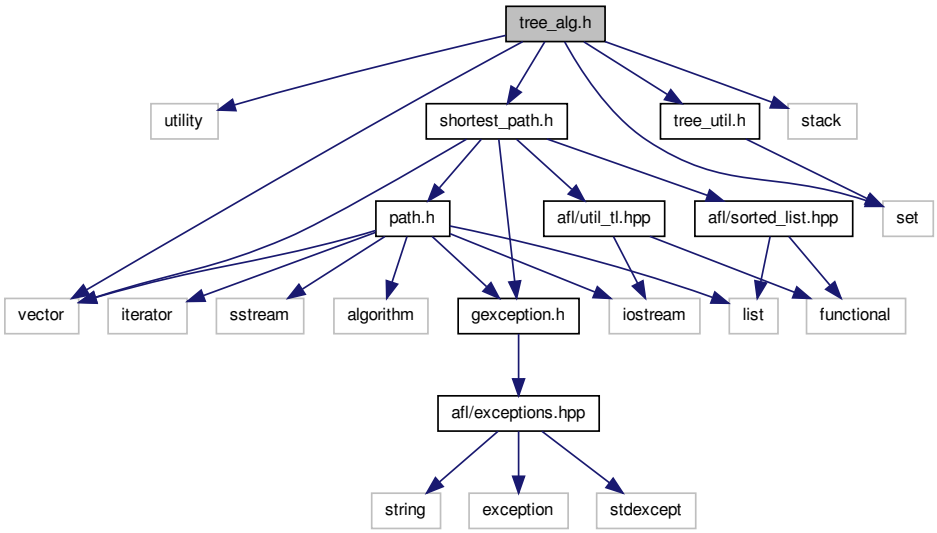
Aiguo Fei, UCLA 2001

derived and simplified from a more general event scheduler done at Bell Labs in 1998

10.24 tree_alg.h File Reference

```
#include <utility>
#include <vector>
#include <set>
#include <stack>
#include "shortest_path.h"
#include "tree_util.h"
```

Include dependency graph for tree_alg.h:



Classes

- struct afg::tree_dfs_c< TreeT >
- struct afg::tree_dfs< TreeT >

Namespaces

- namespace afg

Functions

- template<class GraphT , class TreeT , class Fun >
  bool afg::sptree_all (const GraphT &gra, int ns, TreeT &tree, Fun f_weight, typename Fun::result_type w_infty)
- template<class GraphT , class TreeT , class Fun >
  bool afg::sptree (const GraphT &gra, int ns, TreeT &tree, const std::set< int > &smem, Fun f_weight, typename Fun::result_type w_infty)
- template<class GraphT , class TreeT , class Fun >
  bool afg::sptree_all_s (const GraphT &gra, int ns, TreeT &tree, Fun f_weight, typename Fun::result_type w_infty)
- template<class GraphT , class TreeT , class Fun >
  bool afg::sptree_s (const GraphT &gra, int ns, TreeT &tree, const std::set< int > &smem, Fun f_weight, typename Fun::result_type w_infty)
- template<class GT , class TT , class EDT >
  bool afg::extend_greedy (TT &tree, const GT &gra, const std::set< int > &snodes, const std::vector< int > &vpred, const std::vector< EDT > &vdist, EDT d_infty)
- template<class TT , class EDT >
  bool afg::extend_greedy_s (TT &tree, const std::set< int > &snodes, int n, const std::vector< int > &vpred, const std::vector< EDT > &vdist, EDT d_infty)

10.24.1 Detailed Description

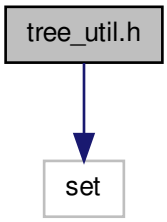
tree algorithms

Aiguo Fei, UCLA 2000

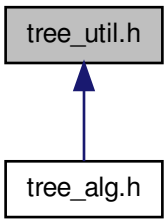
10.25 tree_util.h File Reference

```
#include <set>
```

Include dependency graph for tree_util.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace afg

Functions

- template<class TreeT >
  void afg::prune_tree (TreeT &tree, const set< int > &in_set)

- template<class GT , class TT >
  bool afg::pred2tree (const GT &gra, int n, const int *pred, TT &tree)
- template<class GT , class TT >
  bool afg::pred2tree (const GT &gra, const vector< int > &vpred, TT &tree)
- template<class TT , class EDT >
  bool afg::pred2tree_s (const vector< int > &vpred, const vector< EDT > &vdist, TT &tree)
- template<class GT , class TT >
  bool afg::all_pred2tree (const GT &gra, const vector< int > &vpred, int ns, TT &tree)
- template<class TT , class EDT >
  bool afg::all_pred2tree_s (const vector< int > &vpred, const vector< EDT > &vdist, int n, int ns, TT &tree)

10.25.1 Detailed Description

utility routines for tree

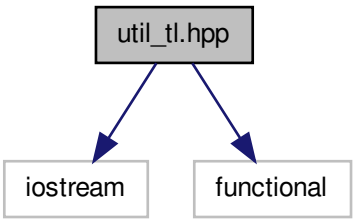
Aiguo Fei, UCLA

1999, 2000

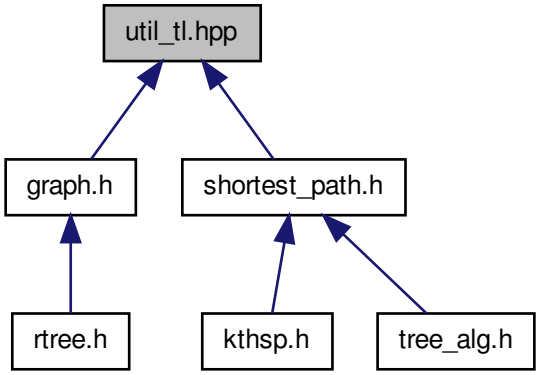
10.26 util_tl.hpp File Reference

```
#include <iostream>
#include <functional>
```

Include dependency graph for util_tl.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct afl::in_convt2t< T1, T2 >
- struct afl::runit< T >

  Operator( ) always returns 1 for any reference passed.
- struct afl::runit_p< T >

  Operator( ) always returns 1 for any pointer passed.
- struct afl::ref2value< T >

  Operator ( ) returns the value of a reference.
- struct afl::ref2ref< T >

  Operator ( ) returns a const reference of a reference.
- struct afl::pointer2value< T >

  Operator ( ) returns the value of a pointer,.
- struct afl::named_pair< NameT, KeyT >

Functions

- template<class T1 , class T2 >
  std::istream & **afl::operator**>> (std::istream &is, in_convt2t< T1, T2 > i)
- template<class T >
  const T & afl::tmin (const T &a, const T &b)

return the smaller one of two elements.

- template<class T >
  const T & afl::tmax (const T &a, const T &b)

  return the larger one of two elements.
- template<class T , class Cmp >
  const T & afl::tmin (const T &a, const T &b, Cmp cmp)

  return the smaller one of two elements, given comparison function object.
- template<class T , class Cmp >
  const T & afl::tmax (const T &a, const T &b, Cmp cmp)

  return the larger one of two elements, given comparison function object.
- template<class NameT , class KeyT >
  bool afl::operator== (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs, equal if their names are the same.
- template<class NameT , class KeyT >
  bool afl::operator!= (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs, not equal if their names are different.
- template<class NameT , class KeyT >
  bool afl::operator== (const named_pair< NameT, KeyT > &lhs, const NameT &n)

  compare a named pair with a name
- template<class NameT , class KeyT >
  bool afl::operator!= (const named_pair< NameT, KeyT > &lhs, const NameT &n)

  compare a named pair with a name
- template<class NameT , class KeyT >
  bool afl::operator< (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs by their keys.
- template<class NameT , class KeyT >
  bool afl::operator> (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs by their keys.
- template<class NameT , class KeyT >
  bool afl::operator<= (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs by their keys.
- template<class NameT , class KeyT >
  bool afl::operator>= (const named_pair< NameT, KeyT > &lhs, const named_pair< NameT, KeyT > &rhs)

  compare two named pairs by their keys.

- template<class NameT , class KeyT >
  std::ostream & afl::operator<< (std::ostream &os, const named_pair< NameT, KeyT > &np)

  Output two values within parenthese, separated by ",".
- template<class NameT , class KeyT >
  named_pair< NameT, KeyT > afl::make_npair (const NameT &lhs, const KeyT &rhs)

  produce a named_pair from two values

10.26.1 Detailed Description

collection of utility template classes

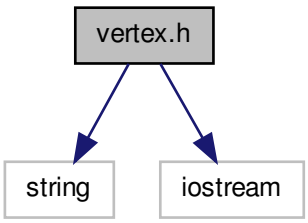
Aiguo Fei

March 1999, December 2000

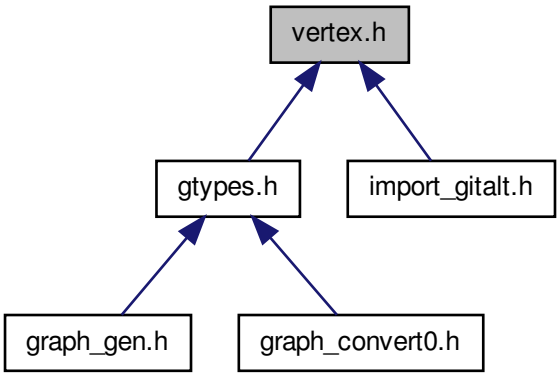
10.27 vertex.h File Reference

```
#include <string>
#include <iostream>
```

Include dependency graph for vertex.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct afg::iddVertex< IDT, DT >
- struct afg::idwVertex< IDT >
- struct afg::xyVertex< T >
- struct afg::ixyVertex< IDT, T >
- struct afg::tsVertex

Namespaces

- namespace afg

Functions

- template<class IDT , class DT >
  bool **afg::operator==** (const iddVertex< IDT, DT > &lhs, const iddVertex< IDT, DT > &rhs)
- template<class IDT , class DT >
  bool **afg::operator<** (const iddVertex< IDT, DT > &lhs, const iddVertex< IDT, DT > &rhs)

- template<class IDT , class DT >
  bool **afg::operator!=** (const iddVertex< IDT, DT > &lhs, const iddVertex< IDT, DT > &rhs)
- template<class IDT , class DT >
  std::ostream & **afg::operator**<< (std::ostream &os, const iddVertex< IDT, DT > &rhs)
- template<class IDT , class DT >
  std::istream & **afg::operator**>> (std::istream &is, iddVertex< IDT, DT > &rhs)
- template<class T >
  bool **afg::operator==** (const xyVertex< T > &lhs, const xyVertex< T > &rhs)
- template<class T >
  bool **afg::operator!=** (const xyVertex< T > &lhs, const xyVertex< T > &rhs)
- template<class T >
  std::ostream & **afg::operator**<< (std::ostream &os, const xyVertex< T > &v)
- template<class T >
  std::istream & **afg::operator**>> (std::istream &is, xyVertex< T > &v)
- template<class IDT , class T >
  bool **afg::operator==** (const ixyVertex< IDT, T > &lhs, const ixyVertex< IDT, T > &rhs)
- template<class IDT , class T >
  bool **afg::operator!=** (const ixyVertex< IDT, T > &lhs, const ixyVertex< IDT, T > &rhs)
- template<class IDT , class T >
  std::ostream & **afg::operator**<< (std::ostream &os, const ixyVertex< IDT, T > &v)
- std::istream & **afg::operator**>> (std::istream &is, tsVertex &v)
- std::ostream & **afg::operator**<< (std::ostream &os, const tsVertex &v)

10.27.1 Detailed Description

pre-defined vertex types