NAME

libgraph − abstract graph library

SYNOPSIS

```
#include <graphviz/graph.h>
void       aginit();
Agraph_t   *agread(FILE*);
int        agwrite(Agraph_t*, FILE*);
int        agerrors();
Agraph_t   *agopen(char *name, int kind);
void       agclose(Agraph_t *g);
Agraph_t   *agsubg(Agraph_t *g, char *name);
Agraph_t   *agfindsubg(Agraph_t *g, char *name);
Agnode_t   *agmetanode(Agraph_t *g);
Agraph_t   *agusergraph(Agnode_t *metanode);
int        agnnodes(Agraph_t *g), agnedges(Agraph_t *g);
int        agcontains(Agraph_t *g, void *obj);
int        aginsert(Agraph_t *g, void *obj);
int        agdelete(Agraph_t *g, void *obj);

Agnode_t   *agnode(Agraph_t *g, char *name);
Agnode_t   *agfindnode(Agraph_t *g, char *name);
Agnode_t   *agfstnode(Agraph_t *g);
Agnode_t   *agnxtnode(Agraph_t *g, Agnode_t *n);
Agnode_t   *aglstnode(Agraph_t *g);
Agnode_t   *agprvnode(Agraph_t *g, Agnode_t *n);

Agedge_t   *agedge(Agraph_t *g, Agnode_t *tail, Agnode_t *head);
Agedge_t   *agfindedge(Agraph_t *g, Agnode_t *tail, Agnode_t *head);
Agedge_t   *agfstedge(Agraph_t *g, Agnode_t *n);
Agedge_t   *agnxtedge(Agraph_t *g, Agedge_t *e, Agnode_t *n);
Agedge_t   *agfstin(Agraph_t *g, Agnode_t *n);
Agedge_t   *agnxtin(Agraph_t *g, Agedge_t *e);
Agedge_t   *agfstout(Agraph_t *g, Agnode_t *n);
Agedge_t   *agnxtout(Agraph_t *g, Agedge_t *e);

char       *agget(void *obj, char *name);
char       *agxget(void *obj, int index);
void       agset(void *obj, char *name, char *value);
void       agxset(void *obj, int index, char *value);
int        agindex(void *obj, char *name);

Agsym_t*   agraphattr(Agraph_t *g,char *name,char *value);
Agsym_t*   agnodeattr(Agraph_t *g,char *name,char *value);
Agsym_t*   agedgeattr(Agraph_t *g,char *name,char *value);
Agsym_t*   agfindattr(void *obj,char *name);
```

DESCRIPTION

libgraph maintains directed and undirected attributed graphs in memory and reads and writes graph files. Graphs are composed of nodes, edges, and nested subgraphs. A subgraph may contain any nodes and edges of its parents, and may be passed to any *libgraph* function taking a graph pointer, except the three that create new attributes (where a main graph is required).

Attributes are internal or external. Internal attributes are fields in the graph, node and edge structs defined at compile time. These allow efficient representation and direct access to values such as marks, weights, and pointers for writing graph algorithms. External attributes, on the other hand, are character strings (name-value pairs) dynamically allocated at runtime and accessed through *libgraph* calls. External

attributes are used in graph file I/O; internal attributes are not. Conversion between internal and external attributes must be explicitly programmed.

The subgraphs in a main graph are represented by an auxiliary directed graph (a meta-graph). Meta-nodes correspond to subgraphs, and meta-edges signify containment of one subgraph in another. agmetanode and agusergraph map between subgraphs and meta-nodes. The nodes and edges of the meta-graph may be traversed by the usual *libgraph* functions for this purpose.

USE

1. Define types Agraphinfo_t, Agnodeinfo_t, and Agedgeinfo_t (usually in a header file) before including <graphviz/graph.h>.

2. Call aginit() before any other *libgraph* functions. (This is a macro that calls aginitlib() to define the sizes of Agraphinfo_t, Agnodeinfo_t, and Agedgeinfo_t.)

3. Compile with −lgraph −lcdt.

Except for the **u** fields, *libgraph* data structures must be considered read-only. Corrupting their contents by direct updates can cause catastrophic errors.

GRAPHS

```
typedef struct Agraph_t {
   char             kind;
   char            *name;
   Agraph_t          *root;
   char            **attr;
   graphdata_t       *univ;
   Dict_t           *nodes,*inedges,*outedges;
   proto_t          *proto;
   Agraphinfo_t       u;
} Agraph_t;

typedef struct graphdata_t {
   Dict_t           *node_dict;
   attrdict_t       *nodeattr, *edgeattr, *globattr;
} graphdata_t;

typedef struct proto_t {
   Agnode_t          *n;
   Agedge_t          *e;
   proto_t           *prev;
} proto_t;
```

A graph *kind* is one of: AGRAPH, AGRAPHSTRICT, AGDIGRAPH, or AGDIGRAPHSTRICT. There are related macros for testing the properties of a graph: AG_IS_DIRECTED(g) and AG_IS_STRICT(g). Strict graphs cannot have self-arcs or multi-edges. **attr** is the array of external attribute values. **univ** points to values shared by all subgraphs of a main graph. **nodes**, **inedges**, and **outedges** are sets maintained by **cdt(3)**. Normally you don't access these dictionaries directly, though the edge dictionaries may be reordered to support programmer-defined ordered edges (see dtreorder in *cdt(3)*). **proto** is a stack of templates for node and edge initialization. The attributes of these nodes and edges are set in the usual way (agget, agset, etc.) to set defaults.

agread reads a file and returns a new graph if one was successfully parsed, otherwise returns NULL if EOF or a syntax error was encountered. Errors are reported on stderr and a count is returned from agerrors(). write_graph prints a graph on a file. agopen and agsubg create new empty graph and subgraphs.

agfindsubg searches for a subgraph by name, returning NULL when the search fails.

ALL OBJECTS

agcontains, aginsert, agdelete are generic functions for nodes, edges, and graphs. gcontains is a predicate that tests if an object belongs to the given graph. aginsert inserts an object in a graph and agdelete undoes this operation. A node or edge is destroyed (and its storage freed) at the time it is deleted from the main graph. Likewise a subgraph is destroyed when it is deleted from its last parent or when its last parent is deleted.

NODES

```
typedef struct Agnode_t {
   char            *name;
   Agraph_t        *graph;
   char            **attr;
   Agnodeinfo_t    u;
} Agnode_t;
```

agnode attempts to create a node. If one with the requested name already exists, the old node is returned unmodified. Otherwise a new node is created, with attributed copied from g−>proto−>n. agfstnode (agnxtnode) return the first (next) element in the node set of a graph, respectively, or NULL. aglstnode (agprvnode) return the last (previous) element in the node set of a graph, respectively, or NULL.

EDGES

```
typedef struct Agedge_t {
   Agnode_t        *head,*tail;
   char            **attr;
   Agedgeinfo_t    u;
} Agedge_t;
```

agedge creates a new edge with the attributes of g−>proto−>e including its key if not empty. agfindedge finds the first (u,v) edge in g. agfstedge (agnxtedge) return the first (next) element in the edge set of a graph, respectively, or NULL. agfstin, agnxtin, agfstout, agnxtout refer to in- or out-edge sets. The idiomatic usage in a directed graph is:

```
for (e = agfstout(g,n); e; e = agnextout(g,e)) your_fun(e);
```

An edge is uniquely identified by its endpoints and its key attribute (if there are multiple edges). If the key of g−>proto−>e is empty, new edges are assigned an internal value. Edges also have tailport and headport values. These have special syntax in the graph file language but are not otherwise interpreted.

ATTRIBUTES

```
typedef struct attrsym_t {
   char            *name,*value;
   int             index;
   unsigned char   printed;
} attrsym_t;
```

```
typedef struct attrdict_t  {
   char              *name;
   Dict_t             *dict;
   attrsym_t          **list;
} attrdict_t;
```

agraphattr, agnodeattr, and agedgeattr make new attributes. g should be a main graph, or NULL for declarations applying to all graphs subsequently read or created. agfindattr searches for an existing attribute.

External attributes are accessed by agget and agset These take a pointer to any graph, node, or edge, and an attribute name. Also, each attribute has an integer index. For efficiency this index may be passed instead of the name, by calling agxget and agxset. The printed flag of an attribute may be set to 0 to skip it when writing a graph file.

The list in an attribute dictionary is maintained in order of creation and is NULL terminated. Here is a program fragment to print node attribute names:

```
   attrsym_t *aptr;
   for (i = 0; aptr = g−>univ−>nodedict−>list[i]; i++) puts(aptr−>name);
```

EXAMPLE GRAPH FILES

```
graph any_name {           /* an undirected graph */
   a −− b;              /* a simple edge */
   a −− x1 −− x2 −− x3;    /* a chain of edges */
   "x3.a!" −− a;          /* quotes protect special characters */
   b −− {q r s t};        /* edges that fan out */
   b [color="red",size=".5,.5"];   /* set various node attributes */
   node [color=blue];      /* set default attributes */
   b −− c [weight=25];     /* set edge attributes */
   subgraph sink_nodes {a b c};    /* make a subgraph */
}

digraph G {
   size="8.5,11";          /* sets a graph attribute */
   a −> b;              /* makes a directed edge */
   chip12.pin1 −> chip28.pin3; /* uses named node "ports" */
}
```

SEE ALSO

dot(1), **neato**(1), **libdict**(3)

S. C. North and K. P. Vo, "Dictionary and Graph Libraries'' 1993 Winter USENIX Conference Proceedings, pp. 1-11.

AUTHOR

Stephen North (north@ulysses.att.com), AT&T Bell Laboratories.