Tutorial Cgraph

Stephen C. North
scnorth@gmail.com

Emden R. Gansner
erg@graphviz.com

7 de fevereiro de 2014

Conteúdo

1. Introdução

Cgraph é uma biblioteca C para programação gráfica. Ele define tipos de dados e operações para gráficos compostos por nós, arestas e subgrafos atribuídos. Os atributos podem ser pares de nome-valor de string para E/S de arquivo conveniente ou estruturas de dados C internas para implementação de algoritmo eficiente.

Cgraph visa a representação gráfica; não é uma biblioteca de algoritmos de nível superior, como caminho mais curto ou fluxo de rede. Nós as visualizamos como bibliotecas de nível superior escritas em cima do Cgraph. Esforços foram feitos no projeto do Cgraph para buscar eficiência de tempo e espaço. A representação gráfica básica (não atribuída) leva 104 bytes por nó e 64 bytes por aresta, portanto, o armazenamento de gráficos com milhões de objetos é razoável. Para gráficos atribuídos, o Cgraph também mantém um pool de strings compartilhado interno, portanto, se todos os nós de um gráfico tiverem color=red, apenas uma cópia de "color" e "red" será feita. Existem outros truques que os especialistas podem explorar para obter eficiência de codificação total. Por exemplo, existem maneiras de inserir as instruções para a travessia da lista de bordas e o acesso à estrutura de dados interna.

O Cgraph usa a biblioteca de dicionários do Phong Vo, libcdt, para armazenar conjuntos de nós e bordas. Esta biblioteca fornece uma interface uniforme para tabelas de hash e árvores de splay, e sua API é utilizável para programação geral (como armazenar multisets, tabelas de hash, listas e filas) em programas Cgraph.

Notação A seguir, usamos TRUE para denotar um valor diferente de zero e FALSE para denotar zero.

2 Objetos de Gráfico

Quase toda a programação Cgraph pode ser feita com ponteiros para esses tipos de dados:

- Agraph_t: um gráfico ou subgráfico
- Agnode_t: um nó de um gráfico ou subgráfico específico
- Agedge_t: uma aresta de um gráfico ou subgráfico específico
- Agsym_t: um descritor para um atributo de par string-valor
- Agrec_t: um atributo de registro de dados C interno de um objeto gráfico

O Cgraph é responsável por seu próprio gerenciamento de memória; a alocação e desalocação de estruturas de dados Cgraph é sempre feita por meio de chamadas Cgraph.

3 Gráficos

Um gráfico de nível superior (também chamado de gráfico raiz) define um universo de nós, arestas, subgrafos, dicionários de dados e outras informações. Um grafo tem um nome e duas propriedades: se é direcionado ou não e se é estrito (multi-arestas são proibidas). [1]

[1] É possível especificar que um grafo é simples (nem multi-arestas nem laços), ou pode ter múltiplas arestas mas não laços.

Observe que nós, arestas e subgrafos existem em exatamente um grafo raiz. Eles não podem ser usados independentemente desse gráfico, ou anexado a outro gráfico raiz.

Os exemplos a seguir usam a convenção de que G e g são Agraph_t* (ponteiros de gráfico), n, u, v, w são Agnode_t* (ponteiros de nó) e e, f são Agedge_t* (ponteiros de aresta).

Para criar um novo gráfico direcionado de nível superior vazio:

```
Agraph_t g =          *g;
agopen("G", direcionado, NULL);
```

O primeiro argumento para agopen é qualquer string e não é interpretado pelo Cgraph, exceto que é gravado e preservado quando o gráfico é escrito como um arquivo.2 O segundo argumento indica o tipo de gráfico e deve ser Agdirected, Agstrictdirected, Agundirigida, ou Agstrictundirected. O terceiro argumento é um ponteiro opcional para uma coleção de métodos para substituir certos comportamentos padrão do Cgraph e, na maioria das situações, pode ser apenas NULL.

Você pode obter o nome de um gráfico por agnameof(g), e você pode obter suas propriedades pelas funções de predicado agisdirected(g) e agisstrict(g).

Você também pode construir um novo gráfico lendo um arquivo:

```
g = agread(stdin,NULL);
```

Aqui, o nome, o tipo e o conteúdo do gráfico, incluindo os atributos, dependem do conteúdo do arquivo. (O segundo argumento é o mesmo ponteiro de método opcional mencionado acima para agopen).

Às vezes é conveniente ter o gráfico representado concretamente como uma cadeia de caracteres str. Nesse caso, o gráfico pode ser criado usando:

```
g = agmemread(str);
```

Você pode escrever uma representação de um gráfico em um arquivo:

```
g = agwrite(g,stdout);
```

agwrite cria uma representação externa do conteúdo e atributos de um gráfico (exceto para tributos internos), que podem ser posteriormente reconstruídos chamando agread no mesmo arquivo.3 agnnodes(g), agnedges(g) e agnsubg(g) retornam a contagem de nós, arestas e (imediato)
subgráficos em um gráfico (ou subgráfico).

Para deletar um grafo e suas estruturas de dados associadas, liberando sua memória, utiliza-se:

```
agclose(g);
```

Finalmente, há uma função interessante, embora obscura, para concatenar o conteúdo de um arquivo gráfico em um arquivo existente. gráfico, como mostrado aqui.

```
g = agconcat(g,stdin,NULL);
```

2Um aplicativo poderia, é claro, manter seu próprio catálogo de grafos usando nomes de grafos.

[3]É trabalho do programador de aplicativos converter entre atributos internos em strings externas quando os gráficos são lidos e escritos, se desejado. Isso parecia melhor do que inventar uma maneira complicada de automatizar essa conversão.

4 nós

No Cgraph, um nó geralmente é identificado por um nome de string exclusivo e um ID de inteiro interno exclusivo atribuído pelo Cgraph. (Por conveniência, você também pode criar nós "anônimos" dando NULL como o nome do nó.) Um nó também possui conjuntos de bordas de entrada e saída, mesmo em grafos não direcionados.

Depois de ter um gráfico, você pode criar ou pesquisar nós desta maneira:

```
Agnode_t *n; n =
agnode(g,"no28",TRUE);
```

O primeiro argumento é um gráfico ou subgráfico no qual o nó deve ser criado. O segundo é o nome (ou NULL para nós anônimos). Quando o terceiro argumento é TRUE, o nó é criado se ainda não existir. Quando é FALSE, como mostrado abaixo, o Cgraph procura localizar um nó existente com o nome fornecido, retornando NULL se nenhum for encontrado.

```
n = agnode(g,"no28",FALSE);
```

A função agdegree(g, n, in, out) dá o grau de um nó no (sub)grafo g, onde in e selecione os conjuntos de arestas.

- agdegree(g,n,TRUE,FALSE) retorna o grau de entrada.
- agdegree(g,n,FALSE,TRUE) retorna o grau de saída.
- agdegree(g,n,TRUE,TRUE) retorna sua soma.

A função agcountuniqedges é idêntica a agdegree, exceto quando os dois últimos argumentos são TRUE. Nesse caso, um loop é contado apenas uma vez.

agnameof(n) retorna o nome da string imprimível de um nó. Observe que, por vários motivos, essa string pode ser um buffer temporário que pode ser substituído por chamadas subsequentes. Assim, o uso

```
printf("%s %s\n",agnameof(agtail(e)),agnameof(aghead(e)));
```

não é seguro porque o buffer pode ser substituído quando os argumentos para printf estão sendo calculados.

Um nó pode ser excluído de um gráfico ou subgráfico por agdelnode(g,n).

5 Bordas

Uma aresta é um par de nós: um par ordenado em um grafo direcionado, não ordenado em um grafo não direcionado. Por conveniência, há uma estrutura de dados de borda comum para ambos os tipos e os pontos de extremidade são os campos tail e head. [4]

Como uma aresta é implementada como um par de arestas, existem dois ponteiros válidos para a mesma aresta, portanto, a comparação simples de ponteiros não funciona para igualdade de arestas. A função ageqedge(Agedge_t *e0, Age_t *e1)

4Quando uma aresta é criada, o primeiro nó será usado como nó final e o segundo nó como cabeça.

é avaliado como verdadeiro se os dois ponteiros representarem a mesma aresta abstrata e geralmente deve ser usado para comparações de arestas.

Uma aresta é feita usando

```
Agnode_t    *u, *v;
Idade_t     *e;

/* suponha que u e v já estejam definidos */ e =
agege(g,u,v,"e28",TRUE);
```

uev devem pertencer ao mesmo gráfico ou subgráfico para que a operação seja bem-sucedida. O “nome” de uma aresta (mais corretamente, identificador) é tratado como um identificador exclusivo para arestas entre um determinado par de nós. Ou seja, só pode haver no máximo uma aresta com nome e28 entre quaisquer u e v, mas pode haver muitas outras arestas e28 entre outros nós.

agtail(e) e aghead(e) retornam os pontos finais de e. Se e for criado como na chamada para idade acima, u será o nó final e v será o nó principal. Isso vale mesmo para grafos não direcionados.

O valor e->node é o “outro” ponto final em relação ao nó do qual e foi obtido. Isto é, se e é uma extremidade do nó n (equivalentemente, n é a cauda de e), então e->nó é a cabeça de e. Um idioma comum é:

```
for (e = agfstout(g,n); e; e = agnxtout(g,e))
    /* faça algo com e->node */
```

agege também pode procurar por arestas:

```
/* encontra qualquer aresta u,v */ e =
idade(g,u,v,NULL,FALSE); /* encontra au,v
aresta com o nome "e8" */ e = agege(g,u,v,"e8",FALSE);
```

Em um grafo não direcionado, uma busca por aresta considerará os vértices dados como nós de cauda e de cabeça.

Uma aresta pode ser excluída de um gráfico ou subgrafo por agdeledge(g,e). A função agnameof pode ser usada para obter o “nome” de uma aresta. Observe que esta será a string identificadora fornecida na criação da borda. Os nomes dos nós head e tail não farão parte da string. Além disso, retorna NULL para arestas anônimas.

6 Travessias

Cgraph tem funções para iterar sobre objetos de gráfico. Por exemplo, podemos escanear todas as arestas de um grafo (direcionadas ou não direcionadas) da seguinte forma:

```
for (n = agfstnode(g); n; n = agnxtnode(g,n)) {
    for (e = agfstout(g,n); e; e = agnxtout(g,e)) {
        /* faz algo com e */
    }
}
```

As funções agfstin(g,n) e afnxtin(g,e) são fornecidas para percorrer listas de borda.

No caso de uma borda direcionada, os significados de “dentro” e “fora” são óbvios. Para gráficos não direcionados, Cgraph atribui uma orientação baseada na ordem análoga dos dois nós quando a aresta é criada.

Para visitar todas as arestas de um nó em um grafo não direcionado:

```
for (e = agfstedge(g,n); e; e = agnxtedge(g,e,n))
    /* faz algo com e */
```

Tenha cuidado se o seu código excluir uma aresta ou nó durante a travessia, pois o objeto não será mais válido para obter o próximo objeto. Isso geralmente é tratado por código como:

```
for (e = agfstedge(g,n); e; e = f) {
    f = agnxtedge(g,e,n)) /* delete e */

}
```

As travessias são garantidas para visitar os nós de um grafo, ou arestas de um nó, em sua ordem de criação no grafo raiz (a menos que permitamos que os programadores substituam a ordenação dos objetos, conforme mencionado na seção 16).

7 Atributos Externos

Objetos de gráfico podem ter pares de nome-valor de string associados. Quando um arquivo de gráfico é lido, o analisador do Cgraph cuida dos detalhes disso, de modo que os atributos podem ser adicionados em qualquer lugar do arquivo. Em programas C, os valores devem ser declarados antes do uso.

O Cgraph assume que todos os objetos de um determinado tipo (gráficos/subgráficos, nós ou arestas) têm os mesmos atributos - não há noção de subtipagem dentro dos atributos. As informações sobre os atributos são armazenadas em dicionários de dados. Cada gráfico tem três (para gráficos/subgráficos, nós e arestas) para os quais você precisará das constantes predefinidas AGRAPH, AGNODE e AGEDGE nas chamadas para criar, pesquisar e percorrer esses dicionários.

Assim, para criar um atributo para nós, utiliza-se:

```
Agsym_t *sym; sym
= agattr(g,AGNODE,"forma","caixa");
```

Se isso for bem-sucedido, sym aponta para um descritor para o atributo recém-criado (ou atualizado). (Assim, mesmo que a forma tenha sido declarada anteriormente e tivesse algum outro valor padrão, ela seria definida como caixa pelo acima.)

Usando um ponteiro NULL como o valor, você pode usar a mesma função para pesquisar as definições de atributo de um gráfico.

```
sym = agattr(g,AGNODE,"forma",0); if (sym) printf("A
forma padrão é %s.\n",sym->defval);
```

Se você tiver o ponteiro para algum objeto gráfico, também poderá usar a função agattrsym.

```
Agnode_t* n;
Agsym_t* sym = agattrsym(n,"forma"); if (sym) printf("A forma
padrão é %s.\n",sym->defval);
```

Ambas as funções retornam NULL se o atributo não estiver definido.

Em vez de procurar um atributo específico, é possível iterar sobre todos eles:

```
(sym = agnxtattr(g,AGNODE,sym)) para o primeiro */ sym = 0; while
    printf("%s = %s\n",sim->nome,sim->defval);
```

Supondo que um atributo já exista para algum objeto, seu valor pode ser obtido ou definido usando seu nome de string ou seu descritor Agsym_t. Para usar o nome da string, temos:

```
str = agget(n,"forma");
agset(n,"forma","hexágono");
```

Se um atributo for referenciado com frequência, é mais rápido usar seu descritor como índice, conforme mostrado aqui:

```
Agsym_t *sym = agattr(g,AGNODE,"forma","caixa"); str = agxget(n,sym);
agxset(n,sym,"hexágono");
```

O Cgraph fornece duas funções auxiliares para lidar com atributos. A função agsafeset(void *obj, char *na primeiro verifica se o atributo foi definido, definindo-o com o valor padrão def se não estiver. Em seguida, ele usa valor como o valor específico atribuído a obj.

Às vezes é útil copiar todos os valores de um objeto para outro. Isso pode ser feito facilmente usando agcopyattr(void *src, void* tgt). Isso pressupõe que a origem e o destino são do mesmo tipo de objetos de gráfico e que os atributos de src já foram definidos para tgt. Se src e tgt pertencerem ao mesmo gráfico raiz, isso será automaticamente verdadeiro.

8 Atributos Internos

Seria possível fazer tudo usando apenas atributos com valor de string. Em geral, porém, isso será muito ineficiente. Para lidar com isso, cada objeto gráfico (grafo, nó ou aresta) pode ter uma lista de registros de dados internos associados. O layout de cada registro é definido pelo programador, exceto que cada um deve ter um cabeçalho Agrec_t.
Os registros são alocados através do Cgraph. Por exemplo:

```
typedef struct mynode_s {
    Agregado h;
    int contar;
} mynode_t;

mynode_t *dados;
Agnode_t n *n;
= agnode(g,"mynodename",TRUE); data =
(mynode_t*)agbindrec(n,"mynode_t",
    sizeof(mynode_t),FALSE); dados-
>contagem = 1;
```

De forma semelhante, aggetrec procura um registro, retornando um ponteiro para o registro se existir e NULL caso contrário; agdelrec remove um registro de um objeto.

Embora cada objeto gráfico possa ter sua própria coleção individual de registros, por conveniência, existem funções que atualizam um gráfico inteiro alocando ou removendo o mesmo registro de todos os nós, arestas ou subgrafos ao mesmo tempo. Essas funções são:

```
void aginit(Agraph_t *g, int kind, char *rec_name, int rec_size, int move_to_front);

void agclean(Agraph_t *g, int type, char *rec_name);
```

Observe que, além de agdelrec e agclean, os registros são removidos e seu armazenamento liberado quando o objeto gráfico associado é excluído. Apenas a estrutura de dados do registro é liberada. Se o aplicativo tiver anexado qualquer memória heap adicional a um registro, é responsabilidade do aplicativo lidar com isso antes que o registro real seja excluído.

Para maior eficiência, existe uma maneira de “bloquear” o ponteiro de dados de um objeto gráfico para apontar para um determinado registro. Isso pode ser feito usando TRUE como o último argumento em agbindrec, aginit ou aggetrec. Se isso for feito, no exemplo acima, poderíamos simplesmente converter esse ponteiro no tipo apropriado para acesso direto (não tipificado) aos dados.

```
(meusdados_t*) (n->base.dados)->contagem = 1;
```

Normalmente, é conveniente encapsular esse acesso usando macros. Por exemplo, podemos ter:

```
#define ND_count(n) (((mydata_t*)(AGDATA(n)))->count)
ND_count(n) = 1;
```

Como isso não é seguro se o registro não foi alocado para algum objeto, é aconselhável duas versões da macro:

```
#ifdef DEBUG
#define ND_count(n) \
    (assert(aggetrec(n,"mynode_t",1)),((mynode_t*)(AGDATA(n)))->count) #else #define ND_count(n) (((mydata_t*)
(AGDATA(n) ))->contar) #endif
```

9 Subgráficos

Os subgrafos são uma construção importante no Cgraph. Eles são destinados a organizar subconjuntos de objetos gráficos e podem ser usados alternadamente com gráficos de nível superior em quase todas as funções Cgraph.

Um subgrafo pode conter quaisquer nós ou arestas de seu pai. (Quando uma aresta é inserida em um subgrafo, seus nós também são inseridos implicitamente, se necessário. Da mesma forma, a inserção de um nó ou aresta implica automaticamente a inserção em todos os subgrafos contidos até a raiz.) Os subgrafos de um grafo formam uma hierarquia (uma árvore). Cgraph tem funções para criar, pesquisar e iterar sobre subgrafos.

Por exemplo,

```
Agraph_t *g, *h;

    /* busca por subgráfico pelo nome */
h = agsubg(g,"meusubgrafo",FALSE); se (!h)

        /* cria subgrafo pelo nome */ h =
    agsubg(g,"meusubgrafo",TRUE);

for (h = agfstsubg(g); h; h = agnxtsubg(h)) { /* agparent está um nível
        acima */
    assert(g == aparent(h));
    /* Usa o subgrafo h */
}
```

A função aparent retorna o (sub)grafo imediatamente contendo o subgrafo do argumento. A iteração feita usando agfstsubg e agnxtsubg retorna apenas subgrafos imediatos. Encontrar subgráficos mais abaixo na hierarquia requer uma pesquisa recursiva.

Não é incomum querer preencher um subgrafo com nós e arestas que já foram criadas.
Isso pode ser feito usando as funções agsubnode e agsubedge,

```
Agnode_t *agsubnode(Agraph_t *g, Agnode_t *n, int create);
Age_t *agsubedge(Agraph_t *g, Age_t *e, int create);
```

que pegam um subgrafo e um objeto de outro subgrafo do mesmo gráfico (ou possivelmente um objeto de nível superior) e o adicionam ao subgrafo de argumento se o sinalizador de criação for TRUE. Também é adicionado a todos os subgráficos anexos, se necessário. Se o sinalizador de criação for FALSE, a solicitação será tratada apenas como uma pesquisa e retornará NULL em caso de falha.

Um subgrafo pode ser removido por agdelsubg(g,subg) ou por agclose(subg).

10 Funções Utilitárias e Macros

Por conveniência, Cgraph fornece algumas funções polimórficas e macros que se aplicam a todos os objetos Cgraph. (A maioria dessas funções pode ser implementada em termos de outras já descritas, ou acessando campos no objeto base Agobj_t.

- AGTYPE(obj): tipo de objeto - AGRAPH, AGNODE ou AGEDGE
- AGID(obj): ID de objeto interno (um comprimento não assinado)
- AGSEQ(obj): timestamp de criação do objeto (um inteiro)
- AGDATA(obj): ponteiro de registro de dados (um Agrec_t*)

Outras funções úteis incluem:

```
    /* Retorna o gráfico raiz de obj */
Agraph_t *agroot(void* obj);
    /* Retorna o gráfico raiz de obj ou obj se um (sub)gráfico */
Agraph_t *agraphof(void* obj);
    /* True de obj pertence a g */ int agcontains(Agraph_t
*g, void *obj);
    /* Excluir obj do (sub)gráfico */
int agdelete(Agraph_t *g, void *obj);
    /* Sinônimo de AGTYPE */ int
agobjkind(void *obj);
```

Um grafo raiz g sempre terá

```
g == agroot(g) == agraphof(g)
```

11 Tratamento de erros

O Cgraph fornece algumas funções básicas de tratamento de erros, prejudicadas pela falta de exceções em C. Atualmente, existem basicamente dois tipos de anomalias: avisos e erros.

Para reportar uma anomalia, utiliza-se:

```
typedef enum { AGWARN, AGERR, AGMAX, AGPREV } agerrlevel_t;


int agerr(agerrlevel_t level, const char *fmt, ...);
```

A função agerr tem uma interface printf, com o primeiro argumento indicando a gravidade do problema. Uma mensagem só é gravada se sua gravidade for maior que um mínimo controlado pelo programador, que é AGWARN por padrão. O programador pode definir esse valor usando agseterr, que retorna o valor anterior. Chamar agseterr(AGMAX) desativa a escrita de mensagens.

Às vezes, informações de contexto adicionais só estão disponíveis em funções que chamam a função em que o erro é realmente detectado. Nesse caso, a função de chamada pode indicar que está continuando o erro atual usando AGPREV como o primeiro argumento.

A função agwarningf é um atalho para agerr(AGWARN,...); da mesma forma, agerrorf é uma abreviação para agerr(AGERR,...).

Alguns aplicativos desejam controlar diretamente a escrita de mensagens. Tal aplicativo pode usar a função tion agseterrf para registrar uma função que a biblioteca deve chamar para realmente escrever a mensagem:

```
typedef int (*agusererrf) (char*);


agusererrf agseterrf(agusererrf);
```

A função de erro anterior é retornada. Por padrão, as mensagens são gravadas em stderr. Erros não gravados são armazenados em um arquivo de log. O último erro registrado pode ser recuperado chamando aglasterr. A função agerr retorna a gravidade máxima relatada para agerr. A função agreseterrors é idêntica, exceto que também redefine o nível de erro como se nenhum erro tivesse sido relatado.

12 Cordas

Conforme mencionado, o Cgraph mantém um pool de strings compartilhados com contagem de referência para cada gráfico. Como muitas vezes há muitas strings idênticas usadas em um gráfico, isso ajuda a reduzir a sobrecarga de memória.

```
char *agstrdup(Agraph_t *, char *); char *agstrbind(Agraph_t
*, char*); int agstrfree(Agraph_t *, char *);
```

```
char *agstrdup_html(Agraph_t *, char *); int aghtmlstr(char *);


char *agcanonStr(char *str); char
*agstrcanon(char *, char *); char *agcanon(char *, int);
```

Cgraph tem funções para criar e destruir diretamente referências a strings compartilhadas. Como acontece com qualquer objeto contado por referência, você pode usá-los como strings comuns, embora os dados não devam ser modificados. Se você precisar armazenar o ponteiro em alguma estrutura de dados, você deve chamar agstrdup(g,s). Isso criará uma nova cópia de s, se necessário, e incrementará a contagem de referências, garantindo que a string não seja liberada por alguma outra parte do programa enquanto você ainda a estiver usando. Como o agstrdup faz uma cópia da string, a string do argumento pode usar armazenamento temporário.

Uma chamada para agstrfree(g,s) diminui a contagem de referência e, se ela se tornar zero, libera a string.
A função agstrbind(g,s) verifica se existe uma string compartilhada idêntica a s e, em caso afirmativo, a retorna. Caso contrário, retorna NULL.

A linguagem DOT suporta um tipo especial de string, contendo marcações do tipo HTML. Para garantir que a semântica HTML seja usada, o aplicativo deve chamar agstrdup_html(g,s) em vez de agstrdup(g,s).
O código a seguir garante que o rótulo do nó seja interpretado como uma string semelhante a HTML e apareça como uma tabela. Se agstrdup fosse usado, o rótulo apareceria como a string literal s.

```
Agraph_t* g;
Agnode_t* n; char*
s = "<TABLE><TR><TD>uma célula</TD></TR></TABLE>"; agset (n, "rótulo",
agsgtrdup_html (g,s));
```

O predicado aghtmlstr retornará TRUE se a string do argumento estiver marcada como uma string do tipo HTML. **NB** Isso só é válido se a string do argumento for uma string compartilhada. Strings semelhantes a HTML também são liberadas usando agstrfree.

A linguagem DOT usa vários caracteres especiais e sequências de escape. Ao escrever strings como texto concreto, é importante que esses recursos lexicais sejam usados para que a string possa ser lida novamente como DOT e interpretada corretamente. O Cgraph fornece três funções para lidar com isso. O mais simples é agcanonStr(s).
Ele retorna um ponteiro para uma versão canônica da string de entrada. Ele usa um buffer interno, portanto, a string retornada deve ser gravada ou copiada antes de outra chamada para agcanonStr. agstrcanon(s,buf) é idêntico, exceto que a função de chamada também fornece um buffer onde a versão canônica pode ser gravada. Um aplicativo deve usar apenas o ponteiro retornado, pois é possível que o buffer não seja usado. O buffer precisa ser grande o suficiente para conter a versão canônica. Normalmente, uma expansão de 2*strlen(s)+2 é suficiente.

Tanto agcanonStr quanto agstrcanon assumem que o argumento string é uma string compartilhada. Por conveniência, a biblioteca também fornece a função agcanon(s,h). Isso é idêntico a agcanonStr(s), exceto que s pode ser qualquer string. Se h for TRUE, a canonização assume que s é uma string do tipo HTML.

13 ajustes de nível de especialista

13.1 Retornos de chamada

Existe uma maneira de registrar funções cliente a serem chamadas sempre que objetos de gráfico são inseridos ou excluídos de um gráfico ou subgrafo, ou têm seus atributos de string modificados. Os argumentos para as funções de retorno de chamada para inserção e exclusão (um agobjfn_t) são o (sub)grafo que contém, o objeto e um ponteiro para uma parte dos dados de estado fornecidos pelo aplicativo. O retorno de chamada de atualização do objeto (um agobjupdfn_t) também recebe o ponteiro de entrada do dicionário de dados para o par nome-valor que foi alterado. O argumento gráfico será o gráfico raiz.

```
typedef void (*agobjfn_t)(Agraph_t*, Agobj_t*, void*); typedef void (*agobjupdfn_t)
(Agraph_t*, Agobj_t*,
                          void*, Agsym_t*);

struct Agcbdisc_s { struct
    { agobjfn_t agobjupdfn_t
        agobjfn_t                ins;
                                 mod;
                                 del; }
     gráfico, nó, aresta;
};
```

As funções de retorno de chamada são instaladas pelo agpushdisc, que também leva um ponteiro para a estrutura de dados do cliente estado que é passado posteriormente para a função de retorno de chamada quando é invocado.

```
agpushdisc(Agraph_t *g, Agcbdisc_t *disco, void *estado);
```

Os retornos de chamada são removidos pelo agpopdisc, que exclui um conjunto de retornos de chamada instalado anteriormente em qualquer lugar da pilha. Esta função retorna zero para sucesso. (Na vida real, essa função não é muito usada; geralmente os retornos de chamada são configurados e deixados sozinhos durante a vida útil de um gráfico.)

```
int agpopdisc(Agraph_t *g, Agcbdisc_t *disco);
```

O padrão é que os retornos de chamada sejam emitidos de forma síncrona, mas é possível mantê-los em uma fila de pendências retornos de chamada a serem entregues sob demanda. Este recurso é controlado pela interface:

```
    /* retorna o valor anterior */ int
agcallbacks(Agraph_t *g, int flag);
```

Se o sinalizador for zero, os retornos de chamada serão mantidos pendentes. Se o sinalizador for um, os retornos de chamada pendentes serão emitidos imediatamente e o gráfico será colocado no modo de retorno de chamada imediato. (Portanto, o estado deve ser redefinido por meio de agcallbacks se eles forem mantidos pendentes novamente.)

NB: é uma pequena inconsistência que o Cgraph dependa do cliente para manter o armazenamento da estrutura da função de retorno de chamada. (Assim, provavelmente não deve ser alocado na pilha dinâmica.) A semântica de agpopdisc atualmente identifica retornos de chamada pelo endereço dessa estrutura, portanto, seria necessário um pouco de retrabalho para corrigir isso. Na prática, as funções de retorno de chamada geralmente são passadas em uma estrutura estática.

13.2 Disciplinas

Um gráfico tem um conjunto associado de métodos (“disciplinas”) para E/S de arquivo, gerenciamento de memória e atribuição de ID de objeto gráfico.

```
estrutura typedef {
    Agmemdisc_t     *me;
    Agidisc_t       *Eu
    Agiodisc_t      iria; *io;
} Agdisc_t;
```

Um ponteiro para uma estrutura Agdisc_t é usado como argumento quando um gráfico é criado ou lido usando agopen, agread e agconcat. Se o ponteiro for NULL, as disciplinas Cgraph padrão serão usadas. Um aplicativo pode passar em suas próprias disciplinas para substituir os padrões. Observe que ele não precisa fornecer disciplinas para todos os três campos. Se algum campo for o ponteiro NULL, o Cgraph usará a disciplina padrão para essa tarefa.

As disciplinas padrão também são acessíveis por nome.

```
Agmemdisc_t AgMemDisc;
Agiddisc_t AgIdDisc;
Agiodisc_t AgIoDisc;
Agdisc_t   AgDefaultDisc;
```

Isso é útil porque, diferentemente do Agdisc_t, todos os campos de disciplinas específicas devem ser não NULL.

Cgraph copia os três ponteiros individuais. Assim, essas três estruturas devem permanecer alocadas para a vida do gráfico, embora o Agdisc_t possa ser temporário.

13.2.1 Gerenciamento de memória

A disciplina de gerenciamento de memória permite chamar versões alternativas de malloc, particularmente, Vmalloc de Vo, que oferece alocação de memória em arenas ou pools. O benefício é que o Cgraph pode alocar um gráfico e seus objetos dentro de um pool compartilhado, para fornecer rastreamento refinado de seus recursos de memória e a opção de liberar todo o gráfico e objetos associados fechando a área em tempo constante, se finalização de objetos de gráfico não são necessários.5

5Isso pode ser corrigido.

```
typedef struct Agmemdisc_s { void *(*open)
    (void); void *(*alloc)(void *state, size_t
    req); void *(*resize)(void *state, void *ptr, size_t old,

                                    size_t req); void
    (*free)(void *state, void *ptr); void (*fechar)(void *estado);

} Agmemdisc_t;
```

Quando um gráfico é criado, mas antes que qualquer memória seja alocada, o Cgraph chama a função aberta da disciplina. O aplicativo deve, então, executar as inicializações necessárias de seu sistema de memória e retornar um ponteiro para qualquer estrutura de dados de estado necessária para a alocação de memória subsequente. Quando o gráfico for fechado, o Cgraph chamará a função close da disciplina, passando-lhe o estado associado, para que a aplicação possa finalizar e liberar quaisquer recursos envolvidos.

As outras três funções de disciplina alloc, resize e free devem ter semântica aproximadamente idêntica às funções padrão da biblioteca C malloc, realloc e free. A principal diferença é que qualquer nova memória retornada por alocação e redimensionamento deve ser zerada e esse redimensionamento recebe o tamanho do buffer antigo além do novo tamanho solicitado. Além disso, todos eles tomam o estado da memória como o primeiro argumento.

Para simplificar o uso da disciplina de memória, o Cgraph fornece três funções wrapper que ocultam a tarefa de obtenção do estado de memória. Estas são as mesmas funções que o Cgraph usa para lidar com a memória de um gráfico.

```
void *agalloc(Agraph_t *g, size_t size); void *agrealloc(Agraph_t
*g, void *ptr, size_t oldsize,
                                tamanho_t tamanho);
void agfree(Agraph_t *g, void *ptr);
```

13.2.2 Gerenciamento de E/S

A disciplina de E/S é provavelmente a mais usada das três, pois os requisitos de E/S dos aplicativos variam muito.

```
typedef struct Agiodisc_s { int (*afread)
    (void *chan, char *buf, int bufsize); int (*putstr)(void *chan, char *str); int (*flush)
    (void *chan);

} Agiodisc_t ;
```

A disciplina de E/S padrão usa stdio e a estrutura FILE para leitura e escrita. As funções afread, putstr e flush devem ter uma semântica aproximadamente equivalente a fread, fputs e fflush, com a óbvia permutação de argumentos.

A implementação da função agmemread do Cgraph fornece um exemplo típico de uso de uma disciplina de E/S personalizada. A ideia é ler um gráfico a partir de uma determinada sequência de caracteres. A implementação da função

é dado abaixo. O rdr_t fornece uma versão em miniatura do FILE, fornecendo as informações de estado necessárias. A função memiofread preenche o papel de afread usando o estado fornecido por rdr_t. O agmemread junta tudo, criando o estado necessário usando a string de argumento e construindo uma estrutura de disciplina usando memiofread e os padrões. Em seguida, ele chama agread com o estado e a disciplina para realmente criar o gráfico.

```
typedef struct
    { const char
    *data; int len; int
    cur; } rdrt;
     -

static int memiofread(void *chan, char *buf, int bufsize) {

    const char *ptr;
    caractere *optr;                                                    10
    caractere c; intl ;
    rdrt*s;
       -

    if (bufsize == 0) return 0; s
    = (rdr t *) chan; if (sÿ>cur >=
    sÿ>len) retorna 0; l = 0; ptr
          = sÿ>dados + sÿ>cur;
    optr = buf; do { *optr++ = c
    = *ptr++; l++;                                                      20




    } while (c && (c != '\n') && (l < bufsize)); sÿ>cur
    += l; retornar l;

}

estático Agiodisc t memIoDisc = {memiofread, 0, 0};                     30

Agraph t *agmemread(const char *cp) {

    rd t rdr;
    disco Agdisc t;
    Agiodisc t memIoDisc;

    memIoDisc.putstr = AgIoDisc.putstr;
    memIoDisc.flush = AgIoDisc.flush;
    rdr.data = cp; rdr.len = strlen(cp);                                40
    rdr.cur = 0;
```

```
    disco.mem = NULL;
    disco.id = NULL;
    disco.io = &memIoDisc;
    return agread (&rdr, &disco);
}
```

13.2.3 Gerenciamento de ID

Objetos de gráfico (nós, arestas, subgrafos) usam um valor inteiro longo não interpretado como chaves. A disciplina de ID dá ao aplicativo controle sobre como essas chaves são alocadas e como elas são mapeadas de e para strings. A disciplina ID possibilita que um cliente Cgraph controle esse mapeamento. Por exemplo, em um aplicativo, o cliente pode criar IDs que são ponteiros para outro espaço de objeto definido por um interpretador front-end. Em geral, a disciplina de ID deve fornecer um mapa entre IDs internos e strings externas.

```
typedef unsigned long ulong;

typedef struct Agidisc_s {
    void *(*open)(Agraph_t *g, Agdisc_t*); long (*map)(void
    *state, int objtype, char *name,
                              ulong *id, int createflag); long (*alloc)(void
    *state, int objtype, ulong id); void (*free)(void *state, int objtype, ulong id); char
    *(*print)(void *state, int objtype, ulong id); void (*fechar)(void *estado); void
    (*idregister) (void *state, int objtype, void *obj);


} Agidisc_t;
```

A função open permite que a disciplina de ID inicialize quaisquer estruturas de dados que mantenha por gráfico individual. Seu valor de retorno é então passado como o primeiro argumento (void *state) para todas as chamadas subsequentes do gerenciador de ID. Quando o gráfico é fechado, a função close da disciplina é chamada para permitir a finalização e liberação de quaisquer recursos utilizados.

As funções alloc e free criam e destroem explicitamente IDs. O primeiro é usado pelo Cgraph para ver se o ID fornecido está disponível para uso. Se estiver disponível, a função deve alocá-lo e retornar TRUE; caso contrário, deve retornar FALSE. Se não estiver disponível, a função de chamada abortará a operação. free é chamado para informar ao gerenciador de ID que o objeto rotulado com o ID fornecido está prestes a ser excluído.

Se houver suporte, compre a disciplina ID, a função map é chamada para converter um nome de string em um ID para um determinado tipo de objeto (AGRAPH, AGNODE ou AGEDGE), com um sinalizador opcional que informa se o ID deve ser alocado, caso não seja já existe.

Existem quatro casos:

name **&&** createflag Mapeie a string (por exemplo, um nome em um arquivo gráfico) em um ID. Se o gerenciador de ID puder cumprir, ele armazenará o resultado no parâmetro id e retornará TRUE. Ele também é responsável por poder imprimir o ID novamente como uma string. Caso contrário, o gerenciador de ID pode retornar FALSE, mas deve implementar o seguinte caso, pelo menos para que a leitura e a gravação de arquivos gráficos funcionem.

!name **&&** createflag Cria um novo ID exclusivo. Ele pode retornar FALSE se não suportar objetos anônimos, mas isso é fortemente desencorajado para que o Cgraph suporte “nomes locais” em arquivos de gráfico.

name **&& !**createflag Testa se o nome já foi mapeado e alocado. Em caso afirmativo, o ID deve ser retornado no parâmetro id. Caso contrário, o gerenciador de ID pode retornar FALSE ou armazenar qualquer ID não alocado no resultado. Isso é conveniente, por exemplo, se os nomes forem conhecidos como cadeias de dígitos que são convertidas diretamente em valores inteiros.

!name **&& !**createflag Nunca usado.

A função de impressão é chamada para converter um ID interno de volta em uma string. É permitido retornar um ponteiro para um buffer estático; um chamador deve copiar seu valor, se necessário, após as chamadas subsequentes. NULL deve ser retornado por gerenciadores de ID que não mapeiam nomes.

Observe que as funções alloc e map não recebem ponteiros para nenhum objeto recém-criado. Se um cliente precisa instalar ponteiros de objeto em uma tabela de tratamento, ele pode obtê-los por meio de novos retornos de chamada de objeto (Seção 13.1).

Para tornar este mecanismo acessível, Cgraph fornece funções para criar objetos por ID em vez de externo nome:

```
Agnode_t *agidnode(Agraph_t *g, unsigned long id, int create);
Agedge_t *agidedge(Agraph_t *g, Agnode_t *t, Agnode_t *h,
                                    id longo não assinado, int create);
Agraph_t *agidsubg(Agraph_t *g, unsigned long id, int create);
```

Observe que, com a disciplina de ID padrão, essas funções retornam NULL.

13.3 Listas de nós e arestas achatadas

Para acesso aleatório, nós e arestas geralmente são armazenados em árvores de splay. Isso adiciona uma sobrecarga pequena, mas perceptível, ao percorrer as “listas”. Para eficiência total, existe uma maneira de linearizar as árvores de splay nas quais os conjuntos de nós e arestas são armazenados, convertendo-os em listas simples. Depois disso, eles podem ser percorridos muito rapidamente. A função agflatten(Agraph_t *g, int flag) irá achatar as árvores se flag for TRUE, ou reconstruir as árvores se flag for FALSE. Observe que, se qualquer chamada adicionar ou remover um objeto de gráfico, a lista correspondente será automaticamente retornada ao seu formato de árvore.

A biblioteca fornece várias macros para automatizar o nivelamento e simplificar os percursos padrão. Por Por exemplo, o código a seguir executa a travessia usual em todas as bordas de um gráfico:

```
Agnode_t* n;
Idade_t* e;
```

```
Agnoderef_t* nr;
Agegeref_t* er; for (nr =
FIRSTNREF(g); nr; nr = NEXTNREF(g,nr)) { n = NODEOF(nr); /* fazer algo com
    o nó n */ for (er = FIRSTOUTREF(g,nr); er; er = NEXTEREF(g,er)) {


        e = EDGEOF(er); /* faz
        algo com edge e */
    }
}
```

Compare isso com o código mostrado na página 7.

14 Bibliotecas Relacionadas

Libgraph é um antecessor do Cgraph e agora é considerado obsoleto. Todo o código Graphviz agora é escrito usando Cgraph. Como alguns aplicativos mais antigos usando libgraph podem precisar ser convertidos para Cgraph, observamos algumas das principais diferenças.

Uma diferença fundamental entre as duas bibliotecas é a manipulação dos atributos da estrutura de dados C. Libgraph conecta-os ao final das estruturas do grafo, nó e aresta. Ou seja, um programador de aplicativos define as estruturas graphinfo, nodeinfo e edgeinfo antes de incluir graph.h, e a biblioteca consulta o tamanho dessas estruturas em tempo de execução para que possa alocar objetos de gráfico do tamanho adequado. Como há apenas uma chance de definir atributos, esse design cria um impedimento para escrever bibliotecas de algoritmos separadas. As estruturas dinâmicas Agrec_t, descritas na Seção 8, permitem que cada algoritmo anexe sua própria estrutura de dados necessária.

Conforme observado na Seção 5, as arestas são implementadas de forma ligeiramente diferente nas duas bibliotecas, portanto, a comparação de ponteiros de aresta para igualdade deve ser substituída por ageqedge, a menos que você tenha certeza de que os dois ponteiros têm tipos consistentes. Como exemplo onde podem surgir problemas, temos o seguinte código:

```
void transversal(Agraph_t *g, Age_t *e0) {

    Agnode_t *n = aghead (e0);
    Idade_t *e;

    for (e = agfstout(g, n); n; n = agnxtout(g, e)) { if (e == e0) continue; /* evitar
        borda de entrada */ /* fazer algo com e */

    }
```

Se e0 for um in-edge (AGTYPE(e) == AGINEDGE), a comparação com e sempre falhará, pois este último é um out-edge.

Em Cgraph, o aninhamento de subgrafos forma uma árvore. Em libgraph, um subgrafo pode pertencer a mais de um pai, então eles formam um DAG (grafo acíclico direcionado). Libgraph na verdade representa este DAG como um meta-grafo especial que é navegado por chamadas de libgraph. Depois de ganhar experiência com libgraph, decidimos que essa complexidade não valia seu custo. Em libgraph, o código para percorrer subgrafos teria uma forma algo como:

```
Idade_t* mim;
Agnode_t* mn;
Agraph_t* mg = g->meta_node->graph;
Agraph_t* subg; for (me
= agfstout(mg, g->meta_node); me; me = agnxtout(mg, me)) {
    mn = eu->cabeça;
    subg = agusergrafo(mn);
        /* usa subgrafo subg */
}
```

A travessia semelhante usando Cgraph teria a forma

```
Agraph_t* subg; for
(subg = agfstsubg(g); subg; subg = agnxtsubg(subg)) {
        /* usa subgrafo subg */
}
```

Finalmente, existem algumas pequenas diferenças sintáticas nas APIs. Por exemplo, em libgraph, o nome de um nó ou gráfico e os nós cabeça e cauda de uma aresta são diretamente acessíveis por meio de ponteiros, enquanto em Cgraph, é necessário usar as funções agnameof, agtail e aghead. Libgraph tende a ter funções separadas para criar e encontrar um objeto, ou para lidar com diferentes tipos de objetos, por exemplo, agnode e agfindnode.
Em vez disso, o Cgraph usará uma única função com um parâmetro adicional. No geral, as duas bibliotecas variam muito, tanto sintaticamente quanto semanticamente, portanto, a conversão é bastante direta.

As Tabelas 1–3 listam as constantes e operações comuns em libgraph, e o valor ou procedimento correspondente em Cgraph, se houver.

Lgraph é um sucessor do Cgraph, escrito em C++ por Gordon Woodhull. Ele segue o modelo geral de grafos do Cgraph (particularmente, seus subgrafos e ênfase em operações dinâmicas eficientes de grafos), mas usa o sistema de templates e herança de tipos C++ para fornecer atributos internos typesafe, modulares e eficientes. (LGraph depende de cdt para conjuntos de dicionários, com uma camada de interface C++ semelhante a STL.) Um protótipo bastante maduro do sistema Dynagraph (um sucessor do dot e do cleano para lidar com a manutenção online de diagramas dinâmicos) foi prototipado no LGraph. Consulte a página dgwin (Dynagraph for Windows) http://www.dynagraph.org/dgwin/ para obter mais detalhes.

15 Interfaces para outros idiomas

Se ativado, o pacote Graphviz contém ligações para Cgraph em várias linguagens, incluindo Java, Perl, PHP, Tcl, Python e Ruby.

libgraph	Gráfico
UM GRÁFICO	Agundirigida
AGRÁFICO	Agrícola não dirigida
AGDÍGRAFO	Direcionado
AGDIGRAFESTRICO	Direcionado por agremiações
aginit	não é necessário 6
agopen(nome,tipo) agopen(nome,tipo,NULL)	
agread(filep) agread(filep,NULL)	
agread_usergets(filep,gets) Agmemdisc_t mem	= AgMemDisc; Agiddisc_t id = AgIdDisc; Agiodisc_t io = AgIoDisc; io.afread = gets;7 agread(arquivo,disco);
agsetiodisc obj->	nenhum análogo direto; Veja acima
nome gráfico-> nó	agnamof(obj);
raiz-> borda do	agroot(gráfico);
gráfico-> borda da	aggraphof(nó);
cabeça-> cauda	aghead(borda);
	agtail(borda);
AG_IS_DIRECTED(gráfico)	agisdirected(gráfico)
AG_IS_STRICT(gráfico)	agisstrict(gráfico)
agobjkind(obj)	AGTYPE(obj)
agsubg(parent,name)	agsubg(pai,nome,1)
agfindsubg(parent,name) g-	agsubg(pai,nome,0)
>meta_node_graph agusergraph(graph) aginsert(graph, obj)	agfstsubg/agnxtsubg Veja os exemplos na página 21
	agsubnó(grafo,obj); se obj é um nó agsubedge(grafo,obj); se obj é uma aresta não permitido se obj for um gráfico

Tabela 1: Conversões da função de gráfico

16 Problemas em aberto

Ordenação de nós e arestas. A intenção no design do Cgraph era eventualmente dar suporte a nós e bordas definidos pelo usuário definir a ordenação, substituindo o padrão (que é a ordem do carimbo de data/hora de criação do objeto). Por exemplo, em topologia gráficos incorporados, as listas de bordas podem ser classificadas no sentido horário em torno dos nós. Porque Cgraph assume que todos os conjuntos de arestas no mesmo Agraph_t têm a mesma ordenação, provavelmente deve haver um novo primitiva para comutação de funções de ordenação de nós ou conjuntos de bordas. Entre em contato com o autor se você precisar desse recurso.

libgraph	Gráfico
agnode(gráfico,nome)	agnode(gráfico,nome,1)
agfindnode(gráfico,nome)	agnode(gráfico,nome,0)
agege(gráfico,cauda,cabeça)	agege(gráfico,cauda,cabeça,NULL,1)
agfindedge(gráfico,cauda,cabeça)	agedge(gráfico,cauda,cabeça,NULL,0)

Tabela 2: Conversões de função de nó e borda

libgraph	Gráfico
agprotograph() agprotonode(graph)	sem analógico
agprotoedge(graph) agattr(obj,	não usado
name, default)	não usado
agattr(agroot(obj),AGTYPE(obj),name,default)	
agfindattr(obj, nome) agattrsym(obj,nome)	
agraphattr(gráfico, nome padrão)	agattr(gráfico,AGRAPH,nome,padrão)
agnodeattr(gráfico, nome, padrão)	agattr(gráfico,AGNODE,nome,padrão)
agettr(gráfico, nome, padrão)	agattr(gráfico,AGEDGE,nome,padrão)
agfstattr(obj) agnxtattr(agroot(obj),AGTYPE(obj),NULL)	
agnxtattr(obj,sym)	agnxtattr(agroot(obj),AGTYPE(obj),sym)
aglstattr(obj) nenhum	sem analógico
agprvattr(obj, sym) análogo	

Tabela 3: Conversões de função de atributo

XML. Dialetos XML como GXL e GraphML foram propostos para gráficos. Embora seja simples para codificar nós e arestas em XML, há sutilezas na representação de estruturas mais complicadas, como Subgráficos do Cgraph e aninhamento de padrões de atributo. Por outro lado, GXL e GraphML fornecem noções de gráficos compostos, que não estão disponíveis em DOT. Prototipamos um analisador XML e gostaríamos de concluir e liberar este trabalho se tivéssemos uma aplicação atraente. (A saída XML simples de gráficos não é difícil de programar.)

Visualizações de gráficos; gráficos externos. Às vezes seria conveniente relacionar os objetos de um grafo aos de outro sem transformar um em um subgrafo de outro. Outras vezes, é necessário preencher um gráfico a partir de objetos entregues sob demanda de uma API externa (como um banco de dados relacional que armazena gráficos). Nós estamos agora experimentando ataques em alguns desses problemas.

6A biblioteca Cgraph tem uma função chamada aginit, mas tem uma assinatura e semântica diferentes.

7Observe que a ordem dos parâmetros difere dos gets usados na libgraph.

17 Exemplo

A seguir está um filtro Cgraph simples que lê um gráfico e emite seus componentes fortemente conectados, cada um como um gráfico separado, além de um mapa geral das relações entre os componentes. Para economizar espaço, as funções auxiliares no cabeçalho ingraph.h não são mostradas; o programa inteiro pode ser encontrado na versão do código-fonte do Graphviz em cmd/tools.

Sobre as linhas 32-41 estão as declarações para registros internos para gráficos e nós. As linhas 43-48 definem macros de acesso para campos nestes registros. As linhas 52-83 definem uma estrutura de pilha simples necessária para o algoritmo de componente fortemente conectado e até a linha 97 estão algumas definições globais para o programa.

O resto do código pode ser lido de trás para frente. Da linha 262 até o final está o código clichê que lida com argumentos de linha de comando e abre vários arquivos de gráfico. O trabalho real é feito começando com o processo da função sobre a linha 223, que funciona em um gráfico de cada vez. Depois de inicializar os registros internos do nó e do gráfico usando o aginit, ele cria um novo gráfico para o mapa de visão geral e chama visit nos nós não visitados para encontrar componentes. visit implementa um algoritmo padrão para formar o próximo componente fortemente conectado em uma pilha. Quando um é concluído, um novo subgrafo é criado e os nós do componente são instalados.
(Existe uma opção para pular componentes triviais que contêm apenas um nó.) nodeInduce é chamado para processar as bordas externas dos nós neste subgrafo. Essas arestas pertencem ao componente (e são adicionadas a ele) ou apontam para um nó em outro componente que já deve ter sido processado.

```
/********************************************************* ************************
  * Copyright (c) 2011 AT&T Propriedade Intelectual *
  Todos os direitos reservados. Este programa e os materiais que o acompanham
  *    são disponibilizados sob os termos da Eclipse Public License v1.0
  * que acompanha esta distribuição e está disponível em * http://
  www.eclipse.org/legal/epl-v10.html
  *
  * Contribuintes: Veja os logs do CVS. Detalhes em http://www.graphviz.org/
  ************************************************ ************************************/

#ifdef HAVE_CONFIG_H
#include "config.h" #endif


#include <stdio.h>
#include <stdlib.h>
#ifdef HAVE_UNISTD_H
#include <unistd.h> #endif
#include "cgraph.h" #include
"ingraphs.h"


#ifdef HAVE_GETOPT_H
#include <getopt.h> #else
```

```
#include "compat_getopt.h" #endif


#define INF ((unsigned int)(ÿ1))                                                    30

typedef struct Agraphinfo t { _
    Agregado;
    Agnode t *rep ;
} Agraphinfo t ;_

typedef struct Agnodeinfo t { _
    Agregado;
    valor inteiro não assinado ;
    Gráfico t *scc ;                                                                40
} Agnodeinfo t ;_

#define getrep (g)      (((Agraphinfo t*)( g ÿ >base .data)) ÿ >rep )
#define setrep ( g ,rep) (getrep ( g) = rep ) #define getscc ( n)
                            (((Agnodeinfo t*)(( n ) ÿ >base .data)) ÿ >scc )
#define setscc ( n ,sub) (getscc ( n) = sub ) #define getval ( n)
                            (((Agnodeinfo t*)(( n ) ÿ >base .data)) ÿ >val )
#define setval ( n ,newval)         (getval(n) = newval)

/********* pilha **********/                                                       50
estrutura typedef {
    Agnode t **dados ;
    Agnode t **ptr ;
} Pilha ;

static void initStack (Pilha * {    sp,  int sz)

    sp ÿ >data = (Agnode t **) malloc (sz * sizeof (Agnode t *)); sp ÿ
    >ptr = sp ÿ >data ;
}                                                                                   60

static void freeStack (Stack * sp ) {

    livre (sp ÿ >dados);
}

push vazio estático (Pilha * sp , Agnode t{ *  n)

    *(sp ÿ >ptr++) = n ;
}                                                                                   70

estático Agnode t *top (Pilha * sp) {

    return *(sp ÿ >ptr ÿ 1);
}
```

```
estático Agnodeē t *pop (Pilha * sp) {

    sp ÿ >ptrÿÿ ;
    return *(sp ÿ >ptr);
}

/************ fim da pilha ************/

typedef struct
    { int Comp;
    int ID; int N
    nós emī nontrivī SCC ; } sccēstate;


static int wantDegenerateComp ;
estático int Silencioso ; static int
StatsOnly ; static int Verbose ;
caractere estático *CmdName ;
caractere estático **Arquivos ;
ARQUIVO estático *outfp ;
                                    /* resultado; stdout por padrão */
static void nodeInduce (Agraph t {¯ * g , Gráfico t¯ * mapa)

    Agnode t¯ *n;
    Idade t ¯ *e;
    Agrafo t *rootg = agroot ( g);


    for ( n = agfstnode ( g); n ; n = agnxtnode ( g , n)) {
        for ( e = agfstout (rootg , n); e ; e = agnxtout (rootg , e)) { if
            (agsubnode ( g , aghead ( e), FALSE))
                agsubedge ( g , e, VERDADE);
            senão {
                Gráfico t¯*tscc = getscc (agtail ( e));
                Gráfico t¯*hscc = getscc (aghead ( e)); if
                (tscc && hscc ) agege (map , getrep
                    (tscc), getrep (hscc), NIL (char *),
                        TRUE);
            }
        }
    }
}

visita int estática (Agnodeē * n, Gráfico t¯ * mapa, Pilha * sp, sccstate * st)
t {
    não assinado m , min;
    Agnode t¯ *;
```

```
Gráfico t_*subg ;
Idade t _ * e;

min = ++(st ÿ >ID);
setval (n, min); empurrar (sp,
n);

for ( e = agfstout ( n ÿ >root , n); e ; e = agnxtout ( n ÿ >root , e)) { t =
    aghead ( e); if (getval ( t) == 0) m = visita ( t , map , sp , st); senão

        m = getval(t);
    se (m < min) min =
        m;
}

if (getval ( n) == min ) { if (!
    wantDegenerateComp && (top (sp) == n)) { setval ( n ,
        INF); pop (sp); } else { Agraph_t * G = agroot ( n); sprintf (nome do caractere [32];
        (name , "cluster_%d", (st ÿ >Comp)++); subg =
    agsubg( G , TRUE); agbindrec (subg , "scc_graph" ,
        sizeof (Agraphinfo_t), TRUE); setrep (subg ,
        agnode (map , , TRUE));
                                  , nome
                                                            _
                                        nome

        faça {t = pop (sp);
            agsubnode (subg , t , VERDADE);
            setval ( t , INF);
            setrep ( t , subg); nós em
            nontriv_SCC++; } while ( t != n);
        nodeInduce (subg , map); if (!StatsOnly)
        agwrite ( subg , outfp);

    }

} return min;
}

rótulo int estático (Agnode_t * n , int nodecnt , int * aresta )
t {
    Idade t _ * e;

    setval (n, 1);
    nodecnt++;
```

```
    for ( e = agfstedge ( n ÿ >root , n); e ; e = agnxtedge ( n ÿ >root , e, n)) {
        (*edgecnt) += 1; if ( e ÿ >node == n ) e = agopp ( e); if (!getval
        ( e ÿ >node)) nodecnt = label ( e ÿ >node

                                                , nodecnt edgecnt);
                                                                                    180
    } return nodecnt;
}

int estático
countComponents (Agraph t { * g, int *max degree , float *nontree frac_)

    int nc = 0;
    int soma_arestas = 0;
    int soma_não árvore =
    0; grau int ; int n arestas;                                                    190
    int n nós;

    Agnode t * n;

    for ( n = agfstnode ( g); n ; n = agnxtnode ( g , n)) { if (!
        getval ( n)) { nc++; n arestas = 0; n nós = rótulo (n,
            0, & n arestas); soma arestas += n arestas ;
            soma não-árvore += ( n arestas ÿ n nós + 1);
                                                                                    200
        }

    } if (máximo grau)
        { int maxd = 0;
        for ( n = agfstnode ( g); n ; n = agnxtnode ( g , n)) { deg
            = agdegree ( g , n, VERDADE,
            VERDADE); if (maxd < deg ) maxd = deg ;
                setval (n, 0);
                                                                                    210
        }*máximo_grau = maxd;

    } if (nontree_frac) { if
        (soma_arestas > 0)
            *nontree frac = (float ) soma nontree / (float ) soma arestas;
        else *não-árvore frac = 0,0;

    } return nc;                                                                    220
}
```

```
processo de vazio estático (Gráfico t* G) {

    Agnodo t *n;
    Gráfico t *mapa;
    int nc = 0; float
    não-árvore frac = 0; int
    Grau máximo = 0;
    Pilha de pilha;                                                                    230
    estado sccstate;

    aginit(G, AGRAPH, "scc_graph", sizeof(Agraphinfo t), TRUE); aginit(G, AGNODE,
    "scc_node", sizeof(Agnodeinfo t), TRUE); estado.Comp = estado.ID = 0; state.N
    nós em nontriv SCC = 0;

    if (Verbose)
        nc = countComponents(G, &Maxdegree, &nontree frac);
                                                                                       240
    initStack(&stack, agnodes(G) + 1); map =
    agopen("scc_map", Agdirected, (Agdisc t *) 0); for (n = agfstnode(G);
    n; n = agnxtnode(G, n)) if (getval(n) == 0) visit(n, map, &stack, &state);
        freeStack(&stack); if (!StatsOnly) agwrite(map, outfp);
            agclose(mapa);




                                                                                       250
    if (Verbose)
        fprintf(stderr, "%d %d %d %d %.4f %d %.4f\n", agnnodes(G),
                agnedges(G), nc, state.Comp, state.N nós em nontriv
                SCC / (duplo) agnodes(G), Maxdegree, nontree frac); else if (!
                Silent) fprintf(stderr, "%d nós, %d arestas, %d componentes
    fortes\n", agnnodes(G), agnedges(G), state.Comp);


}                                                                                      260

static FILE *openFile(char *name, char *mode) {

    ARQUIVO
    *fp; char *modestro;

    fp = fopen(nome, modo); if (!
    fp) { if (*mode == 'r') modestr
        = "leitura"; senão
                                                                                       270
```

```
                modesto = "escrevendo" ;
            fprintf (stderr , "gvpack: não foi possível abrir o arquivo %s para %s\n" , modestr);      ,
                    nome
            saída(1);
        }
        return (fp);
}

static char *useString = "Uso: %s [-sdv?] <files>\n\ - somente produz                                    280
    -s          estatísticas\n\ - silent\n\ - permite componentes
    -S          degenerados\n\ -o<outfile> - grava em <outfile > (stdout)\n\
    -d          - verboso\n\

    -v
    -?          - uso de impressão\n\
 Se nenhum arquivo for especificado, stdin será usado\n" ;

uso static void (int v ) {
                                                                                                         290
        printf(useString, CmdName); saída
        (v);
}

static void scanArgs (int argc , char **argv ) {

        intc ;

        CmdName = argv[0];
        opter = 0; while (( c =                                                                          300
        getopt (argc , argv , ":o:sdvS")) != EOF ) { switch ( c ) {

            caso 's:
                Apenas estatísticas
                = 1; quebrar ; caso
            'd':
                wantDegenerateComp = 1;
                quebrar ; caso 'o':

                outfp = openFile(optarg, "w");                                                           310
                quebrar ; caso 'v':

                Verbo = 1;
                quebrar ; caso
            'S':
                Verbo = 0;
                Silencioso = 1;
                quebrar ;
            caso '?' :
                if (optopt == '?')                                                                       320
```

```
                uso(0);
            senão
                fprintf(stderr, "%s: opção -%c não reconhecida - ignorada\n",
                        CmdName, opt);
            parar;
        }

    } argv += optind;
    argc ÿ= optind;

    se (arg)
        Arquivos =
    argv; if (!outfp)
        outfp = stdout;                 /* stdout o padrão */
}

Gráfico estático t *gread(FILE * fp) {

    return agread(fp, (Agdisc t *) 0);
}

int main(int argc, char **argv) {

    Gráfico t*g;
    grafo estado ig;

    scanArgs(argc, argv);
    newIngraph(&ig, Arquivos, gread);

    while ((g = nextGraph(&ig)) != 0) { if
        (agisdirected(g)) process(g); senão


            fprintf(stderr, "Gráfico %s em %s não é direcionado - ignorado\n",
                    agnameof(g), fileName(&ig));
        agclose(g);
    }

    retornar 0;
}
```