*** RELATóRIO TéNICO ***

ALGORITMOS PARA GERAÇÃO DE DíGRAFOS REDUTíVEIS

Lilian Markenzon

Oswaldo Vernet de S. Pires

NCE 04/91

Março/91

Universidade Federal do Rio de Janeiro

Núcleo de Computação Eletrônica

Caixa Postal 2324

20001 - Rio de Janeiro - RJ

BRASIL

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

ALGORITMOS PARA GERAÇÃO
DE DIGRAFOS REDUTIVEIS

Lilian Markenzon
Oswaldo Vernet

RESUMO

Este Relatório Técnico descreve quatro algoritmos para geração de digrafos redutíveis. De início, caracterizamos esta família de digrafos, enunciando algumas de suas principais propriedades e o algoritmo de Tarjan para o seu reconhecimento. Propomos dois métodos para geração randômica: o primeiro deles decorre da imediata aplicação do teste de redutibilidade sobre um digrafo gerado aleatoriamente, no sentido de eliminar arestas que o tornem não redutível; o segundo utiliza um percurso em pós-ordem de uma árvore de particionamento. Caracterizamos, finalmente, uma sub-classe dos digrafos redutíveis, denominados completos ou saturados, e mostramos duas maneiras distintas de gerá-los.

ABSTRACT

This Technical Report describes four algorithms for the reducible flow graphs generation. We characterize at first this class of digraphs, enunciating their main properties and Tarjan's recognition algorithm. We propose two methods for the random generation: the first one is a slight variant of the reducibility test applied on a digraph randomly generated, deleting the edges which violate the reducibility criteria; the second one uses a post-order traversal of a partitioning tree. Finally, the subclass of complete (or saturated) reducible flow graphs is also characterized, including two distinct methods for its generation.

1. INTRODUÇÃO.

Na pesquisa e desenvolvimento de algoritmos eficientes, é fundamental a geração de conjuntos de dados que garantam um tratamento adequado no momento da comparação entre algoritmos de mesma ordem de complexidade. Este trabalho apresenta alguns resultados nesta área, com o exame de algoritmos para geração pseudo-aleatória de digrafos redutíveis. Tais algoritmos foram utilizados para a obtenção de uma massa de dados destinada à comparação entre duas implementações do teste de redutibilidade, conforme pode ser visto em [4].

Na seção 2 revisamos brevemente alguns conceitos no tocante a digrafos redutíveis. Em seguida, apresentamos o teste de redutibilidade proposto em Tarjan [1]. A seção 4 dedica-se ao estudo de dois algoritmos para geração de digrafos redutíveis. Caracterizamos, na seção 5, uma sub-classe desta família de digrafos, denominados digrafos redutíveis completos ou saturados, provando algumas proposições acerca da construção de tais digrafos. Seguem-se dois algoritmos para geração de digrafos redutíveis saturados e, finalmente, a seção 7 apresenta algumas conclusões acerca do presente trabalho.

2. CARACTERIZAÇÃO DOS DIGRAFOS REDUTIVEIS.

Um *digrafo de fluxo F* é uma tripla (V, E, r), sendo (V, E) um digrafo e $r \in V$ tal que todo $w \in V$ é alcançável a partir de r. O vértice r é denominado *origem* do digrafo. Um vértice v *domina* um vértice w em um digrafo de fluxo F = (V, E, r) se v pertence a todo caminho de r a w.

Um digrafo de fluxo T = (V, E, r) é uma *árvore direcionada* se nenhuma aresta chega a r e qualquer outro vértice pertencente a $V - \{ r \}$ tem apenas uma aresta chegando a ele. Se uma árvore direcionada T é subgrafo de um digrafo de fluxo F e T contém todos os vértices de F então T é uma *árvore de espalhamento* ou *árvore geradora* de F. .

Realizando uma busca em profundidade [1] em um digrafo de fluxo F = (V, E, r) a partir do vértice r, separamos as arestas de E em quatro categorias disjuntas: arestas de *árvore*, de *avanço*, de *cruzamento* e de *retorno*. Se $E_T \subseteq E$ for o conjunto de arestas de árvore resultante da busca, o subgrafo T = (V, E_T, r) é uma árvore geradora de F, denominada *árvore de profundidade*.

Um fato notável é que, mesmo fixado o vértice de partida, a classificação de arestas ainda assim é sensível à ordem dos vértices na estrutura de adjacências utilizada para armazenar o digrafo em memória.

O Teorema 1, enunciado a seguir, propõe três caracterizações equivalentes para a família dos digrafos *redutíveis*.

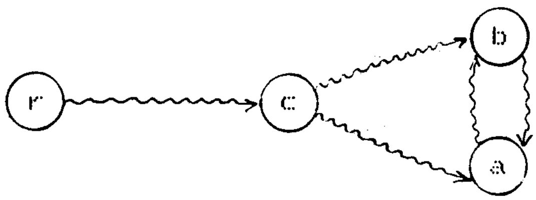
Teorema 1. Seja $F = (V, E, r)$ um digrafo de fluxo. As seguintes proposições são equivalentes:

(i) Para qualquer aresta de retorno (v, w) relativa a uma árvore de profundidade $T = (V, E_T, r)$, w domina v.

(ii) Para qualquer árvore de profundidade com raiz r, o conjunto de arestas de retorno oriundo da busca é invariante.

(iii) Não há vértices $a, b \in V$ tais que existem caminhos: de r a a, de r a b, de a a b e de b a a, concomitantemente. Em outras palavras, o subgrafo mostrado na figura 1 é proibido.

figura 1



Prova. Ver [2].

Definição 1. Um digrafo de fluxo F que satisfaça a um dos itens do Teorema 1 é dito *redutível*.

3. RECONHECIMENTO DE DIGRAFOS REDUTIVEIS.

Utilizando o item *(ii)* do Teorema 1, o algoritmo proposto em [1] constrói, para cada vértice $w \in V$, dois conjuntos:

$\mathrm{Ret}\ (w) = \{\ v \in V \mid (v, w)$ é aresta de retorno $\}$

$\mathrm{Kernel}\ (w) = \{\ v \in V \mid \exists z \in \mathrm{Ret}\ (w)$ tal que há um caminho de v a z que evita $w\ \}$

Observe que $\mathrm{Ret}\ (w) \subseteq \mathrm{Kernel}\ (w)$.

O Teorema 2 a seguir, proposto por Tarjan [1], caracteriza os digrafos redutíveis em função dos conjuntos Ret e Kernel calculados para todos os vértices.

Teorema 2. F é redutível se e somente se, para todo $w \in V$ e para todo $v \in \mathrm{Kernel}\ (w)$, w alcança v por arestas de árvore.

Com efeito, se existe $w \in V$ e $v \in \mathrm{Kernel}\ (w)$ tais que v não é descendente de w, então existe um caminho de r a algum vértice em Kernel (w) que evita w; logo F não é redutível pelo Teorema 1. Reciprocamente, se F não é redutível existe uma aresta de retorno (v, w) e um caminho de r a v que evita w. Então r ∈ Kernel (w), mas r não é descendente de w.

Este resultado leva ao algoritmo de reconhecimento. Inicialmente, realizamos sobre F uma busca em profundidade, calculando, para cada vértice w ∈ V, as profundidades PE (w) de entrada e PS (w) de saída, bem como o conjunto Ret (w). No passo geral, seja w ∈ V o vértice sendo processado. Calculamos Kernel (w). Havendo algum v ∈ Kernel (w) que não seja descendente de w, o digrafo não é redutível. Caso contrário, os vértices em Kernel (w) são condensados com w, formando um novo digrafo. Quanto à ordem em que os vértices devem ser considerados, Tarjan [1] demonstra que o processamento deve ser feito em ordem decrescente das profundidades de entrada dos vértices: o primeiro a ser processado é o último alcançado na busca, decrescendo até a origem do digrafo.

O cálculo de Kernel (w) é simples. Inicialmente fazemos Aux = Kernel (w) = Ret (w). No passo geral, escolhemos x ∈ Aux e retiramos x de Aux. Analisamos todos os vértices z, predecessores de x, tais que (z, x) não seja aresta de retorno. Se z ainda não tiver sido incluído em Kernel (w) e z ≠ w, incluímos z em Aux e em Kernel (w). A cada z considerado, aproveitamos para testar se w alcança z por arestas de árvore, ou seja, PE (w) ≤ PE (z) e PS (w) ≥ PS (z). Se isto não acontecer, o teste pode ser interrompido, pois o digrafo não será redutível. Repetimos este passo geral até que Aux se esvazie.

Se um digrafo F´ resulta de F após uma seqüência de condensações de vértices, vértices de F´ correspondem a conjuntos de vértices de F. Para manipular esta coleção de conjuntos disjuntos, fazemos uso das operações UNION e FIND, definidas da seguinte maneira:

UNION (A, B, C) Promove a união dos conjuntos A e B, removendo-os da coleção e chamando o resultado de C

FIND (x) Descobre a que conjunto da coleção o elemento x pertence

Os conjuntos da coleção são identificados por nomes; em nosso caso, o nome de um conjunto é dado por um de seus elementos e necessariamente será distinto dos nomes dos demais, uma vez que os conjuntos são disjuntos. Ao condensar os vértices de Kernel (w) com w, denominamos w o conjunto daí resultante. Inicialmente, os conjuntos da coleção são unitários e constituídos, cada um, por um vértice de F.

O Algoritmo 1 mostra a implementação do teste de redutibilidade. Observe que o procedimento de busca em profundidade separa as arestas que chegam a cada vértice em arestas de retorno (lista Ret) e as demais (lista Pred).

ALGORITMO 1: TESTE DE REDUTIBILIDADE

```
Var
    PE, PS, Vértice : arranjos [1 .. n] de inteiros;
    Nv, Origem, pent, psai : inteiros;
    Succ, Pred,
    Ret             : arranjos [1 .. n] de listas de inteiros;

Proc Busca-em-Profundidade (v : inteiro);
Var
    w : inteiro;
Início
    PE [v] := pent := pent + 1;
    Vértice [pent] := v;
    Para w ∈ Succ [v] faça
    início
        Se PE [w] = 0 então
            Inclua v em Pred [w];
            Busca-em-Profundidade (w)
        Caso contrário
            Se PS [w] = 0 então
                Inclua v em Ret [w]
            Caso contrário
                Inclua v em Pred [w]
            fim
        fim
    fim;
    PS [v] := psai := psai + 1
fim;

Func Redutível : lógica;
Var
    Aux, Kernel    : conjuntos de inteiros;
    u, w, x, y, z : inteiros;
Início
    Para w := 1 a Nv faça
    início
        Crie um conjunto unitário { w } cujo nome é w;
        PE [w] := PS [w] := Vértice [w] := 0;
        Ret [w] := Pred [w] := lista_vazia
    fim;
    pent := psai := 0;
    Busca-em-Profundidade (Origem);
    Para u := Nv descendo a 1 faça
    início
        w := Vértice [u];
        Kernel := ø;
        Para x ∈ Ret [w] faça
            Kernel := Kernel U { FIND (x) };
        Aux := Kernel;
        Enquanto Aux ≠ {} faça
        início
            Escolha x ∈ Aux;
            Aux := Aux - { x };
            Para y ∈ Pred [x] faça
            início
```

```
                    z := FIND (y);
                    Se PE [w] ≤ PE [z] e
                        PS [w] ≥ PS [z] então
                         Se z ≠ w e z ∉ Kernel então
                              Kernel := Kernel U { z };
                              Aux := Aux U { z }
                         fim
                    Caso contrário
                         Retorne (FALSO)                    (*)
                    fim
               fim
          fim;
          Para x ∈ Kernel faça UNION (x, w, w);
     fim;
     Retorne (VERDADEIRO)
fim;
```

A complexidade de tempo é fácil de ser estimada: a busca em profundidade requer tempo O (n + m). Cada aresta de retorno é examinada exatamente uma vez, construindo o conjunto Kernel inicial. Quando um vértice se torna um elemento de Kernel, ele será condensado com algum outro vértice e seus predecessores jamais serão reexaminados; portanto, cada aresta que não seja de retorno também é examinada exatamente uma vez. Isto significa que a malha principal do algoritmo leva tempo O (n + m) mais o tempo para executar O (n) UNIONs e O (m) FINDs. Como m ≥ n, temos o tempo total O ($m\alpha(m, n)$), se utilizarmos um *algoritmo alpha* [4].

4. GERAÇÃO RANDOMICA DE DIGRAFOS REDUTIVEIS.

O primeiro algoritmo utilizado para a geração baseia-se no teste de redutibilidade apresentado na seção anterior. A idéia consiste em gerar um digrafo qualquer tendo como parâmetros o número de vértices n e uma estimativa para o número de arestas ne. Em seguida, submetemos este digrafo ao teste de redutibilidade, removendo cada aresta (y, x) que violar a condição de alcançabilidade; para isto, basta substituir no Algoritmo 1 a linha assinalada com (*) por Retire x de Succ [y]. Evidentemente o digrafo resultante será redutível.

A geração de um digrafo qualquer é mostrada no Algoritmo 2. O mesmo vetor PE que, após a busca, guarda as profundidades de entrada de cada vértice, é utilizado para armazenar os graus de entrada, garantindo que chegue pelo menos uma aresta a cada um, exceto porventura ao primeiro. Observe que o número de arestas final do digrafo pode exceder a estimativa dada em, no máximo, n - 1 unidades.

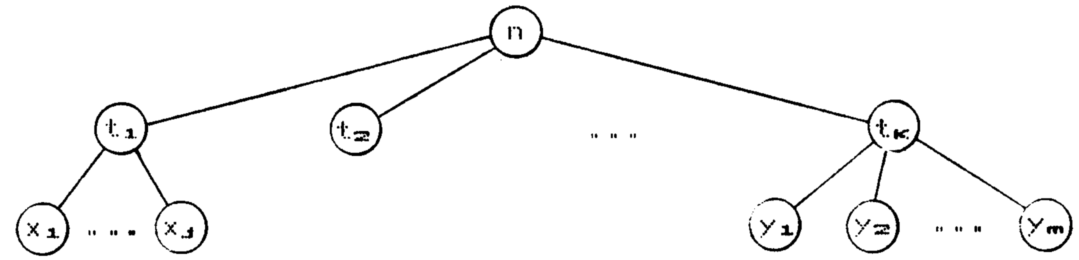
ALGORITMO 2: GERAÇÃO DE UM DIGRAFO QUALQUER

```
Proc Gera-Digrafo (n, ne : inteiros);
Var
    y, w, e : inteiros;
Início
    Para w := 1 a n faça
    início
        PE [w] := 0;
        Succ [w] := lista_vazia
    fim;
    w := 1;
    Para e := 1 a ne faça
    início
        Repita
            Gere y ≠ w entre 1 e n
        Enquanto y ∈ Succ [w];
        Inclua y em Succ [w];
        PE [y] := PE [y] + 1;
        w := y
    fim;
    Para w := 2 a n faça
    início
        Se PE [w] = 0 então
            Repita
                Gere y ≠ w entre 1 e n
            Enquanto w ∈ Succ [y];
            Inclua w em Succ [y]
        fim
    fim;
    Origem := 1; Nv := n
fim;
```

Para o segundo método de geração, devemos fornecer como parâmetros o número de vértices do digrafo n e uma constante $M \geq 2$. Particionamos o conjunto inicial de n vértices da seguinte maneira: determinamos, de início, o tamanho K da partição, que é o número de subconjuntos que a compõem. A partição equânime teria subconjuntos de tamanho $\lfloor n / K \rfloor$ aproximadamente; geramos então um número inteiro aleatório t_1 no entorno de $\lfloor n / K \rfloor$ (mais precisamente no intervalo $[n / 2K, 3n / 2K]$) que será o tamanho do primeiro subconjunto. O problema agora se resume em dividir $n - t_1$ vértices em $K - 1$ subconjuntos, que é resolvido de maneira análoga. Ao fim, teremos K subconjuntos de tamanhos $t_1, \ldots, t_K$. Para cada um deles que possua tamanho superior a M, repetimos o particionamento. O aspecto final é o de uma árvore, que denominamos *árvore de particionamento*, onde as folhas representam subconjuntos com até M vértices (figura 2).

A idéia básica do algoritmo apóia-se na obtenção de *Núcleos Fortemente Conexos*. Nestes núcleos existe uma aresta de retorno fundamental, ligando o último vértice ao primeiro; além disso, existe um caminho entre o primeiro vértice e cada vértice intermediário e um caminho entre cada vértice intermediário e o último. Portanto, os núcleos satisfazem aos requisitos do Teorema 2.

+— figura 2 ———+



O algoritmo de geração consiste então em um percurso em pós-ordem da árvore de particionamento: gerados os núcleos para todos os descendentes de um determinado nó da árvore, consideramos cada um deles como um único vértice condensado e tornamos a gerar um núcleo, até obtermos o digrafo completo, que corresponde à raiz da árvore.

Para gerar um núcleo com $u \leq M$ vértices, numeramo-los de 1 a u e geramos um digrafo acíclico com apenas uma fonte (o vértice 1) e um único sumidouro (vértice u), acrescentando arestas (v, w), tais que $1 \leq v < w \leq u$ e também garantindo que todo vértice (exceto o primeiro) tenha grau de entrada 1 pelo menos. Em seguida, introduzimos alguns ciclos (eventualmente nenhum) neste digrafo, mediante a adição de arestas de retorno da forma $(v, 1)$, sendo $1 < v < u$. No passo final, colocamos a aresta de retorno fundamental $(u, 1)$.

O problema de considerar um núcleo como um só vértice sugere a seguinte questão: arestas destinadas ao (oriundas do) vértice condensado destinam-se a (originam-se de) que vértice do núcleo original? Pelo Teorema 1, cada núcleo correspondente a uma aresta de retorno só deve ter um ponto de entrada, podendo ter vários de saída; portanto, arestas destinadas ao vértice condensado destinam-se, na verdade, ao primeiro vértice do núcleo original e arestas oriundas do vértice condensado podem ser oriundas de qualquer vértice do núcleo.

Em termos algorítmicos, associamos para cada vértice de um núcleo as informações *inicial* e *final*, que indicam respectivamente os vértices inicial e final do núcleo que lhe deu origem. A geração de um núcleo recai em um dos casos abaixo:

a. Quando o núcleo a ser gerado corresponde a uma folha da árvore de particionamento, seus vértices são os próprios vértices do digrafo (e não núcleos condensados). Nesta situação, para cada vértice do núcleo, os ponteiros inicial e final contêm a mesma informação: o número de um vértice do digrafo final.

b. Quando o núcleo corresponde a um vértice interior da árvore, seus vértices são outros núcleos condensados e, para cada um deles, as informações inicial e final indicam respectivamente o primeiro e o último vértice do núcleo original.

ALGORITMO 3: GERAÇÃO DE UM DIGRAFO REDUTIVEL

```
Constantes
     M = ...............;

Tipos
     NUCLEO = Estrutura
                       inicial : inteiro;
                       final   : inteiro
                  fim;
     VETNUC = arranjo [1 .. ?] de NUCLEO;

Var
     Succ : arranjo [1 .. ?] de lista de inteiros;
     Grau : arranjo [1 .. ?] de inteiros;
     Nv   : inteiro;

Programa Principal (n : inteiro);
Var
     Nuc  : NUCLEO;
Inicio
     Alocar n células para o vetor Succ;
     Alocar n células para o vetor Grau;
     Para Nv := 1 a n faça
     inicio
          Succ [Nv] := lista_vazia;
          Grau [Nv] := 0
     fim;
     Nv := 0;
     Particiona (Nuc, n);
     Liberar as células alocadas para Grau
fim;

Proc Particiona (Ref núcleo : NUCLEO; u : inteiro);
Var
     Nuc             : VETNUC;
     i, x, K, tam : inteiros;
Inicio
     Se u ≤ M então
          {*
           *     O núcleo tem menos que M vértices:
           *     gera diretamente.
           *}
          Alocar u células para o vetor Nuc;
          Para i := 1 a u faça
          inicio
               Nv := Nv + 1;
               Nuc [i].inicial := Nv;
               Nuc [i].final   := Nv
          fim;
```

```
            K := u
        Caso contrário
            {*
             *     O núcleo tem mais de M vértices: particiona
             *     em K subconjuntos.
             *}
            Estabeleça o tamanho K ≤ M da partição;
            Alocar K células para o vetor Nuc;
            Para i := K descendo a 2 faça
            início
                {*
                 *     Gera tam no intervalo [ n/2K .. 3n/2K ].
                 *}
                tam := u / i;
                Gere x entre 0 e tam;
                tam := tam / 2 + x;
                Particiona (Nuc [K - i + 1], tam);
                u := u - tam
            fim;
            Particiona (Nuc [K], u)
        fim;

        {*
         *     Gera o núcleo e realiza a "condensação".
         *}
        Gera-Núcleo (Nuc, K);
        núcleo.inicial := Nuc [1].inicial;
        núcleo.final   := Nuc [K].final;
        Liberar as células alocadas para o vetor Nuc
fim;

Proc Gera-Núcleo (Ref Nuc : VETNUC; u : inteiro);
Var
        i, x, e, s : inteiros;
Início
        Se u < 2 então
            Retorne;

        {*
         *     Gera arestas da forma (x, y), 1 ≤ x < y ≤ u.
         *}
        Para i := 1 a u - 1 faça
        início
            Gere x entre 1 e u - i;
            Repita x vezes
                Gere s entre Nuc [i].inicial e Nuc [i].final;
                Repita
                    Gere e entre i + 1 e u;
                    e := Nuc [e].inicial
                Enquanto e ∈ Succ [s];
                Inclua e em Succ [s];
                Grau [e] := Grau [e] + 1
            fim
        fim;
```

```
{*
 *     Verifica se algum vértice ficou com
 *     grau de entrada nulo.
 *}
Para i := 2 a u faça
início
     e := Nuc [i].inicial;
     Se Grau [e] = 0 então
          Repita
               Gere s entre 1 e i - 1;
               Gere s entre Nuc [s].inicial e
                                Nuc [s].final
          Enquanto e ∈ Succ [s];
          Inclua e em Succ [s];
          Grau [e] := Grau [e] + 1
     fim
fim;

{*
 *     Gera algumas arestas de retorno.
 *}
e := Nuc [1].inicial;
Gere x entre 0 e u - 2;
Repita x vezes
     Gere s entre 2 e u - 1;
     Gere s entre Nuc [s].inicial e Nuc [s].final;
     Se e ∉ Succ [s] então
          Inclua e em Succ [s]
fim;

{*
 *     Gera a aresta de retorno fundamental (u, 1).
 *}
s := Nuc [u].final;
Inclua e em Succ [s]
fim;
```

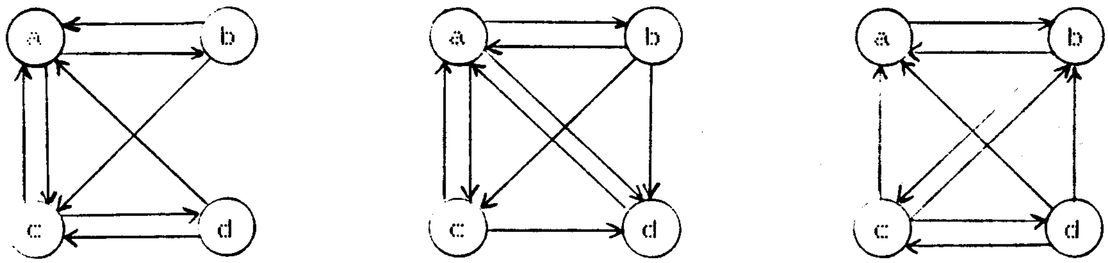
5. DIGRAFOS REDUTIVEIS SATURADOS.

Nesta seção caracterizamos a sub-classe dos Digrafos Redutíveis Saturados (DRS) e enunciamos algumas de suas propriedades. Os Lemas 2 e 3, juntamente com o Corolário 1, ensinam a formar um DRS mediante o acréscimo de um vértice e de algumas arestas a um DRS já existente.

Definição 2. Seja $F = (V, E, r)$ redutível. F é *saturado* (ou *completo*) se o acréscimo de qualquer aresta a E o torna não redutível.

A figura 3 mostra três digrafos redutíveis saturados com 4 vértices. Em todos, o vértice raiz é a. Os digrafos (b) e (c) têm o número máximo de arestas, ao passo que (a) tem uma aresta a menos.

figura 3



(a) (b) (c)

Lema 1. Seja $F = (V, E, r)$ redutível e saturado. Então E contém todas as arestas da forma (v, r), onde $v \in V - \{ r \}$.

Prova. Com efeito, se houver um vértice $v \in V$ tal que $(v, r) \notin E$, F não será saturado, uma vez que arestas desta forma podem sempre ser acrescentadas sem prejuízo dos requisitos de redutibilidade, já que serão classificadas como arestas de retorno em qualquer busca a partir de r.

Lema 2. Seja $F = (V, E, r)$ um digrafo redutível saturado com n vértices. Seja $F' = (V \cup \{x\}, E', r)$ um novo digrafo de fluxo formado a partir de F mediante a adição de um vértice x a V, de algumas arestas a E, mas conservando r como origem. Se $| E' - E | > n + 1$ então F' não é redutível.

Prova. Como F é redutível e saturado, as arestas acrescentadas devem ter necessariamente uma extremidade em x, já que bastaria uma só aresta com ambas as extremidades em V ser acrescentada de modo a tornar F não redutível e, conseqüentemente, F'. Como só existem n vértices em V e estamos acrescentando mais de $n + 1$ arestas, existirão pelo menos dois vértices distintos p, $q \in V$ tais que as arestas (x, p), (p, x), (x, q) e (q, x) tenham sido acrescentadas. Como F é um digrafo de fluxo e a origem r foi mantida, certamente há um caminho de r a p e de r a q. A adição das novas arestas cria dois caminhos: um de p a q e outro de q a p. Logo, pelo Teorema 1, F' não é redutível, conforme atesta a figura 4. Observe que r pode coincidir com p ou q.

Lema 3. Seja $F = (V, E, r)$ um digrafo redutível saturado com n vértices. Seja $F' = (V \cup \{x\}, E', x)$ um novo digrafo de fluxo formado a partir de F mediante a adição de um vértice x a V, de algumas arestas a E, sendo x a nova origem. Se $| E' - E | > n + 1$ então F' não é redutível.

figura 4

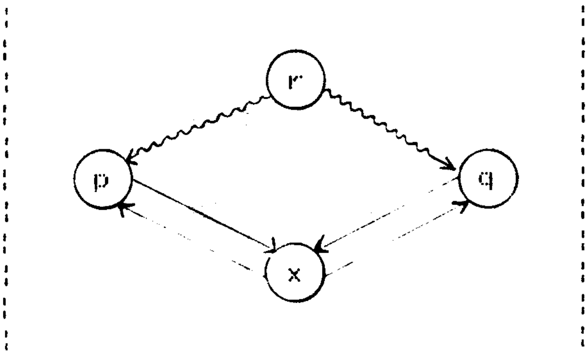
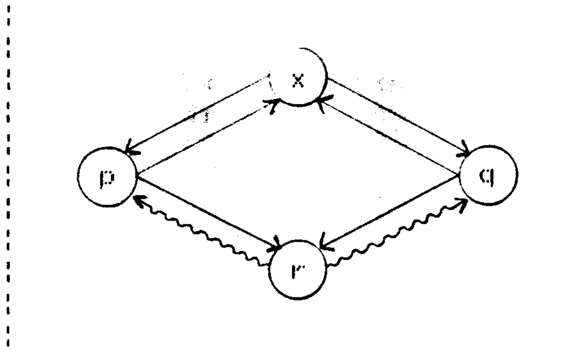


figura 5



Prova. A prova segue a mesma linha do Lema 2: neste caso, existem também pelo menos dois vértices p e q tais que as arestas (x, p), (p, x), (q, x), (x, q) tenham sido acrescentadas. Além disso, existem necessariamente as arestas (p, r) e (q, r) (pelo Lema 1) e caminhos de r a p e de r a q (F é um digrafo de fluxo). Logo, pelo Teorema 1, F′ não será redutível, conforme a figura 5.

Corolário 1. Seja F = (V, E, r) um digrafo redutível saturado com n vértices ao qual se acrescenta um vértice x, que pode ou não ser a nova origem. Então, o número máximo de arestas que podemos acrescentar a E de modo que o novo digrafo seja redutível saturado é n + 1.

Prova. Decorre imediatamente da aplicação dos Lemas 2 e 3. Com efeito, podemos sempre acrescentar n arestas da forma (v, x), onde v ∈ V e a aresta (x, r).

Teorema 3. O número máximo de arestas de um digrafo redutível saturado de n vértices é $(n^2 + n - 2) / 2$.

Prova. Indução em n.

(i) Para n = 1, temos o digrafo trivial, sem arestas;
(ii) Suponha válido para n qualquer (hipótese de indução);
(iii) Acrescentemos um vértice ao digrafo. Pelo Corolário 1, podemos acrescentar no máximo n + 1 arestas sem prejuízo da redutibilidade. Logo, o número de arestas do novo digrafo será

$$(n^2 + n - 2) / 2 + (n + 1) = ((n + 1)^2 + (n + 1) - 2) / 2.$$

6. GERAÇÃO DE DIGRAFOS REDUTIVEIS SATURADOS.

A geração de digrafos redutíveis saturados com número máximo de arestas pode ser feita de duas formas distintas:

a. por avanço: acrescentando todas as arestas da forma (u, v), onde $1 \leq u < v \leq n$ e $n - 1$ arestas de retorno da forma $(v, 1)$, $1 < v \leq n$.

b. por retorno: acrescentando $n - 1$ arestas de árvore da forma $(v, v + 1)$, $1 \leq v < n$ e todas as arestas de retorno possíveis da forma (u, v), $1 \leq v < u \leq n$.

ALGORITMO 4: GERAÇÃO POR AVANÇO.

```
Proc Gera-Avanço (n : inteiro);
Var
    i, j : inteiros;
Inicio
    Para i := 1 a n faça
        Succ [i] := lista_vazia;
    Para i := 1 a n - 1 faça
        Para j := i + 1 a n faça
            Inclua j em Succ [i];
    Para i := 2 a n faça
        Inclua 1 em Succ [i]
fim;
```

ALGORITMO 5: GERAÇÃO POR RETORNO.

```
Proc Gera-Retorno (n : inteiro);
Var
    i, j : inteiros;
Inicio
    Para i := 1 a n faça
        Succ [i] := lista_vazia;
    Para i := 1 a n - 1 faça
        Inclua i + 1 em Succ [i];
    Para i := n descendo a 2 faça
        Para j := i - 1 descendo a 1 faça
            Inclua j em Succ [i]
fim;
```

7. CONCLUSÕES.

Neste relatório, propomos quatro algoritmos para geração de digrafos redutíveis. O primeiro deles consiste na geração de um digrafo de fluxo qualquer e posterior remoção de arestas que o tornam não redutível. Para isto, utilizamos o próprio algoritmo de reconhecimento, no sentido de detectar tais arestas e removê-las. O segundo algoritmo constrói randomicamente pequenos núcleos fortemente conexos de até M vértices, onde M é um parâmetro da geração; o processo progride considerando estes núcleos como um único vértice e formando novos núcleos fortemente conexos, tomando a precaução de direcionar as arestas destinadas a um núcleo ao seu primeiro vértice, não violando assim o critério de redutibilidade. Os dois últimos algoritmos geram digrafos redutíveis saturados com número máximo de arestas.

No primeiro algoritmo, fornecemos uma estimativa para o número de arestas e acrescentamos, de início, exatamente esta quantidade ao digrafo. Pode acontecer que, ao fim desta etapa, alguns vértices tenham grau de entrada nulo. Neste caso, acrescentamos mais algumas arestas de modo a garantir que todos os vértices sejam alcançáveis a partir da origem. Logo, a estimativa inicial pode ser ultrapassada em, no máximo, $n - 1$ unidades, onde n é o número de vértices do digrafo. Como algumas arestas serão removidas, o digrafo terá, via de regra, menos arestas do que a previsão dada.

No segundo algoritmo, o número final de arestas será determinado por quatro fatores:

a. o número de vértices n;
b. a constante M (número máximo de vértices em um núcleo);
c. a escolha de K (tamanho de uma partição);
d. o gerador randômico usado.

Na implementação que realizamos, escolhemos $M = 20$, $K = \lfloor \log_2 u \rfloor$, onde u é o número de vértices a particionar, e empregamos o gerador disponível no ambiente. Este algoritmo foi utilizado na obtenção de uma massa de dados que nos proporcionou uma comparação entre duas implementações do teste de redutibilidade [4]. Em todos os digrafos gerados, nunca obtivemos um número final de arestas superior a $4n$, embora a escolha extrema de subconjuntos em uma partição com número máximo de vértices ($K - 1$ subconjuntos com $\lfloor 3n / 2K \rfloor$ vértices e o último com o que sobrar) possa-nos levar a um número de arestas $O(n^2)$.

O terceiro e quarto algoritmos baseiam-se em duas maneiras distintas de acrescentar vértices e arestas a digrafos redutíveis saturados, obtendo digrafos desta espécie com número máximo de arestas.

8. REFERENCIAS BIBLIOGRAFICAS.

[1] TARJAN, R.E. Testing Flow Graph Reducibility. *Journal of Computer and System Sciences*, vol. 9, nº 3, pp. 355-365, Dec. 1974.

[2] HECHT, M.S. & ULLMAN, J.D. Flow Graph Reducibility. *SIAM Journal on Computing*, vol. 1, nº 2, pp. 188-202, June 1972.

[3] HECHT, M.S. & ULLMAN, J.D. Characterizations of Reducible Flow Graphs. *Journal of the ACM*, vol 21, nº 3, pp. 367-375, July 1974.

[4] PIRES, O.V.S. Coleções de Conjuntos Disjuntos: Operações e Algoritmos. *Dissertação de Mestrado*, COPPE/UFRJ, Abril de 1990.