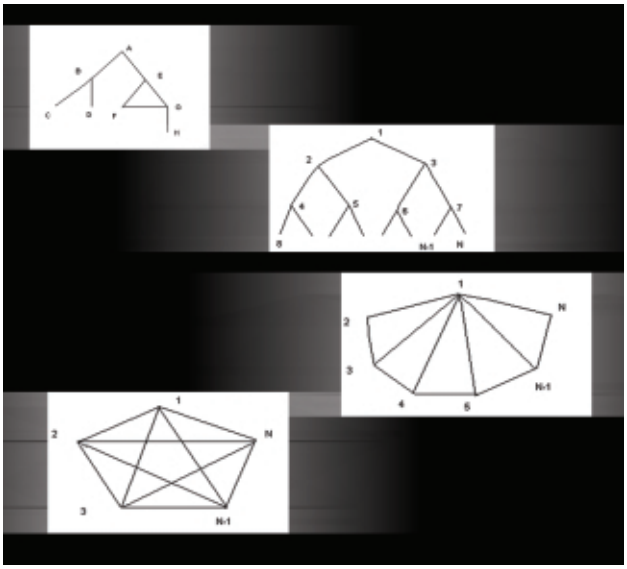
Ministério da Agricultura, Pecuária e Abastecimento

Comunicado Técnico 15

Dezembro, 2001
Campinas, SP



ISSN 1677-8464

Um Algoritmo para Geração de Caminhos em Grafos a Partir de um Vértice

Adauto Luiz Mancini[1]

Este trabalho apresenta um algoritmo para gerar todos os caminhos possíveis que não sejam ou não contenham ciclos a partir de um determinado vértice de um grafo conexo não direcionado. Estabelecendo-se um critério que permita a associação de um peso ao caminho, o algoritmo permite pesar os caminhos à medida em que estes são gerados a partir do vértice de origem, possibilitando um *ranqueamento* de caminhos. Inicialmente são revistos alguns conceitos básicos sobre grafos e depois é apresentado o algoritmo. A motivação para o desenvolvimento do algoritmo é obter-se uma ferramenta para geração de dados que permita o estudo de algumas propriedades estruturais de proteínas.

Proteínas são constituídas por seqüências de aminoácidos. Cada um dos 20 aminoácidos existentes na natureza tem sua composição química definida, bem como as ligações químicas entre seus átomos. A estrutura espacial da proteína depende não somente das forças de atração e repulsão entre átomos internos aos aminoácidos, mas também destas forças envolvendo átomos próximos de aminoácidos distintos, formando uma rede de interações. Podemos representar esta rede como um grafo em que os vértices são os átomos da proteína e as arestas são as forças de interação entre os átomos por ela ligados. A partir da associação de valores às arestas do grafo (distância, energia, etc.) podemos determinar um valor a um dado caminho do grafo fazendo um cálculo que use os valores parciais das arestas que formam o caminho. Como exemplos pode-se considerar o percurso do caminho formado pela soma das distâncias de cada aresta e a energia do caminho formada pela soma das energias das arestas. O valor calculado para o caminho também pode ser dependente de outros elementos que não arestas. Caso se deseje a distância entre o vértice de origem ao vértice de término do caminho é necessário calcular esta distância para cada vértice que compõe o caminho, e este cálculo depende apenas das posições dos vértices, não influenciando os valores associados às arestas. Ponderando os caminhos por critérios podemos montar um *ranking,* selecionando os melhores ou piores para análise. A análise deste tipo de dados envolvendo redes de interações atômicas poderá ser útil na pesquisa de fatores envolvidos para determinação e previsão da estrutura espacial de proteínas.

Os algoritmos tradicionalmente encontrados na literatura para travessia de grafos, busca em largura (Even, 1979; Sedgewick, 1984) e busca em profundidade (Even, 1979; Swamy & Thulasiraman, 1981; Sedgewick, 1984), têm por objetivo percorrer todos os vértices e arestas do grafo a partir de um vértice

[1] *Bacharel em Ciência da Computação, Pesquisador da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo – 13083-970 – Campinas, SP. (E-mail: adaulto@cnptia.embrapa.br)*

Embrapa

raiz. Desta forma, estes algoritmos não são diretamente aplicáveis para o problema de geração de todos os caminhos a partir de um vértice, uma vez que encerram a execução a partir do momento em que todos vértices e arestas do grafo tenham sido percorridos. Para grafos com valores associados às arestas existem os algoritmos de busca do caminho mais curto (Even, 1979; Berge, 1985; Swamy & Thulasiraman, 1981; Gondran & Minoux, 1984; Sedgewick, 1984) e mais longo (Gondran & Minoux, 1984) entre dois vértices. Para que estes algoritmos possam ser executados de forma eficiente e convergir para uma solução sem entrar em *loop* é necessário impor restrições ao grafo. A não existência de ciclos negativos é requerida para o caminho mais curto (Even, 1979; Berge, 1985; Swamy & Thulasiraman, 1981; Gondran & Minoux, 1984; Sedgewick, 1984). O caso geral do caminho mais curto envolvendo ciclos negativos pertence à classe dos problemas NP_Completos, para a qual ainda não existe solução conhecida com complexidade polinomial (Gondran & Thulasiraman, 1984). A busca do caminho mais longo através do uso de álgebra de caminhos requer ciclos de valor zero (Gondran & Thulasiraman, 1984). Estes algoritmos são eficientes, mas devido às suas restrições não podem ser aplicados ao caso geral com garantia de resultados corretos. Assim, torna-se necessário o uso de um algoritmo que gere todos os caminhos a partir de um vértice usando busca exaustiva. Os caminhos poderão então ser classificados em um *ranking* através do uso de algum critério de ponderação. Algoritmos de busca exaustiva consistem na geração de todas as combinações possíveis para a solução de um problema necessitando de altíssimo custo de processamento. Esta classe de algoritmo não é de grande interesse para a pesquisa de algoritmos que em geral busca soluções otimizadas para problemas com mínimo custo de processamento.

No momento o algoritmo proposto encontra-se em fase de implementação e ainda não foram gerados dados concretos para posterior análise de sua importância no estudo de estruturas de proteínas.

Revisão de Conceitos Sobre Grafos

Um grafo G = (V,E) consiste de um conjunto V de vértices e um conjunto E de arestas. Cada aresta corresponde a um par de vértices. As arestas de um grafo direcionado apresentam uma relação de ordem (v_i,v_j) entre os 2 vértices que conecta, enquanto que no grafo não direcionado é indiferente a ordem em que escolhemos qualquer um dos vértices conectados pela aresta.

Um caminho de v_1 a v_k é uma seqüência de vértices $v_1,v_2,...,v_k$ conectados pelas arestas:

$$\{(v_1,v_2),(v_2,v_3),...,(v_{k-1},v_k)\}.$$

Um ciclo é um caminho em que o primeiro e o último vértices são os mesmos e nenhum outro vértice do caminho se repete.

Um grafo é conexo quando existe ao menos um caminho de qualquer vértice para qualquer outro vértice.

Algoritmo

O algoritmo recursivo apresentado a seguir gera todos os caminhos existentes que não são ou não contêm ciclos e começam em um vértice *X* de um grafo não direcionado conexo. Este algoritmo é uma adaptação da idéia do algoritmo DFS (Depth-First Search) facilmente encontrado na literatura sobre grafos (Even, 1979; Swamy & Thulasiraman, 1981; Sedgewick, 1984) que consiste em varrer todos os nós de um grafo a partir de um nó inicial.

O uso do algoritmo proposto supõe que o grafo é transformado em um grafo direcionado da seguinte maneira: cada aresta (v_i,v_j) do grafo não direcionado é transformada em duas arestas (v_i,v_j) e (v_j,v_i) no grafo direcionado. Também é suposto que antes da execução do algoritmo cada vértice v_i contém duas listas: *ArestasDesmarcadas* contendo todas as arestas que tenham v_i como primeiro vértice e *ArestasMarcadas* que é vazia. *CaminhoCorrente* é uma variável global que é uma lista de arestas inicialmente vazia que armazena o caminho sendo expandido pelo algoritmo. A função *Classifica(Caminho)* recebe como parâmetro a lista de arestas *Caminho* e encarrega-se de atribuir um peso ao caminho segundo algum critério preestabelecido de ponderação e de determinar sua ordem e armazená-lo em um *ranking*. Vamos imaginar a existência de um caminho

$$\{(X,v_2),(v_2,v_3),...,(v_{k-1},v_k)\}$$

tal que não seja possível adicionar mais nenhuma aresta (v_k,v_{k+1}) sem formar um ciclo no caminho. No caso geral o algoritmo pressupõe que o critério de ponderação será aplicado a caminhos deste tipo. No entanto, caso se deseje que o critério de ponderação seja aplicado também aos caminhos que são subconjuntos deste caminho tendo como origem o vértice X, ou seja,

$$\{(X,v_2)\}, \{(X,v_2),(v_2,v_3)\},..., \{(X,v_2),(v_2,v_3),...,(v_{k-2},v_{k-1})\}$$

isto será indicado no algoritmo pelo termo "critério usa caminhos incompletos", significando que o caminho ainda pode ser expandido pela adição de uma ou mais arestas sem formar um ciclo.

```
procedimento caminhos(X)
{//exclui arestas que podem formar um ciclo
 para todas (X.ArestasDesmarcadas)
        {se (aresta.VerticeDestino pertence a CaminhoCorrente)
              {X.ArestasDesmarcadas.Remove(aresta);
               X.ArestasMarcadas.Insere(aresta);
              }
        }
 se (CaminhoCorrente ≠ vazio)
       {//não há como acrescentar uma nova aresta ao caminho se X não tem arestas desmarcadas
        se (X.ArestasDesmarcadas = vazio)
              {Classifica(CaminhoCorrente);
              }
       }
 enquanto (X.ArestasDesmarcadas ≠ vazio)
       {//Aresta recebe uma aresta removida de arestas desmarcadas
        Aresta := X.ArestasDesmarcadas.Remove;
         //insere a aresta removida em ArestasMarcadas
         X.ArestasMarcadas.Insere(Aresta);
         //acrescenta aresta ao caminho sendo gerado
         CaminhoCorrente.Insere(Aresta);
          se (critério de ponderação usa caminhos incompletos)
                {Classifica(CaminhoCorrente);
                }
          caminhos(Aresta.VerticeDestino);
          //remove a aresta do caminho corrente, para que uma nova aresta
          //possa ser acrescida ao vértice X
          CaminhoCorrente.Remove(Aresta);
       }
 enquanto (X.ArestasMarcadas ≠ vazio)
       {Aresta := X.ArestasMarcadas.Remove;
        X.ArestasDesmarcadas.Insere(Aresta);
        }
}
```

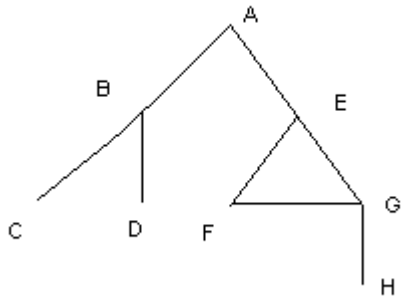
Complexidade

Para simplificar o cálculo da complexidade, vamos fazer as considerações a partir do pior caso: um grafo fortemente conexo. Neste tipo de grafo para qualquer vértice *X* existe uma aresta ligando-o a qualquer outro vértice *Y*. Assim, se o número de vértices do grafo for *nv*, cada vértice terá *nv – 1* arestas ligando-o aos demais *nv – 1* vértices. É fácil observar que neste tipo de grafo o caminho mais longo (em número de arestas) é composto de *nv* vértices e *nv – 1* arestas. Assim o algoritmo será executado em *nv – 1* passos, correspondentes as *nv – 1* arestas do caminho mais longo. Em cada passo são verificadas *nv – 1* arestas incidentes no vértice sendo expandido pelo passo. Desta forma, a quantidade de arestas verificadas em função do número de vértices *nv* é a função exponencial:

$$F(nv) = (nv - 1)^{(nv - 1)}$$

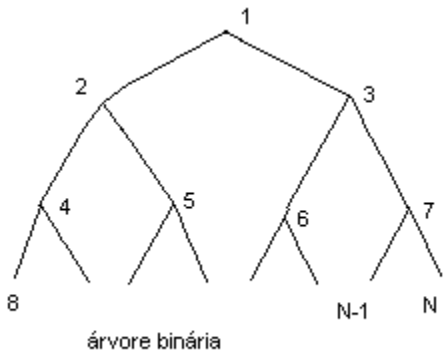
Exemplos de execução

A execução do algoritmo para o grafo ilustrado a seguir, usando o vértice A como raiz, é mostrada a seguir em forma de tabela, para compreensão de como o algoritmo trabalha. Na tabela as colunas passos *n* indicam as arestas sendo inseridas ou removidas no caminho apresentando uma distância de *n* arestas com relação ao vértice raiz A. A coluna caminho encontrado é preenchida sempre que se chega a um vértice que não consegue mais ser expandido. Cada vez que uma nova aresta é inserida no caminho é mostrada a chamada do algoritmo para expandir o vértice destino da nova aresta.

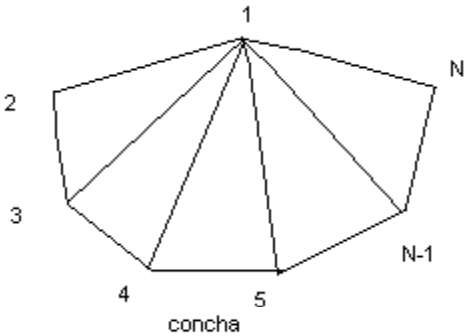


Passo 1	Passo 2	Passo 3	Passo 4	Caminho encontrado
Caminhos(A)				
AB Caminhos(B)				
AB	BC Caminhos(C)			(AB)(BC)
AB				
AB	BD Caminhos(D)			(AB)(BD)
AB				
AE Caminhos(E)				
AE	EF Caminhos(F)			
AE	EF	FG Caminhos(G)		
AE	EF	FG	GH Caminhos(H)	(AE)(EF)(FG)(GH)
AE	EF	FG		
AE	EF			
AE				
AE	EG Caminhos(G)			
AE	EG	GF Caminhos(F)		(AE)(EG)(GF)
AE	EG			
AE	EG	GH Caminhos(H)		(AE)(EG)(GH)
AE	EG			
AE				

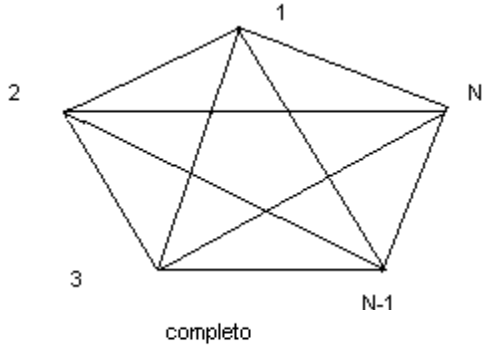
As três estruturas de grafo ilustradas a seguir foram utilizadas para análise de desempenho.



árvore binária



concha



completo

Os algoritmos Caminhos e DepthFirstSearch (DFS)[2] foram aplicados para percorrer estas estruturas com variações de tamanho, anotando-se o número de caminhos distintos encontrados, a quantidade de arestas percorridas e tempo de execução em milisegundos[3] (o valor zero indica que a execução gastou menos de um milisegundo). Os resultados mostram que os dois algoritmos tem comportamento próximo quando o grafo não apresenta ciclos. À medida que a quantidade de ciclos aumenta o algoritmo proposto precisa percorrer muito mais arestas que o DFS, porém encontra todos os caminhos possíveis.

N = 5				
	Estrutura	Caminhos	Arestas	Tempo
DFS	Árvore	3	8	0
Caminhos	Árvore	3	12	0
DFS	Concha	1	14	0
Caminhos	Concha	6	60	0
DFS	Completo	1	20	0
Caminhos	Completo	24	324	0
N = 10				
	Estrutura	Caminhos	Arestas	Tempo
DFS	Árvore	5	18	0
Caminhos	Árvore	5	27	0
DFS	Concha	1	34	0
Caminhos	Concha	16	315	0
DFS	Completo	1	90	0
Caminhos	Completo	362880	9864099	6000
N = 25				
	Estrutura	Caminhos	Arestas	Tempo
DFS	Árvore	13	48	0
Caminhos	Árvore	13	72	0
DFS	Concha	1	94	0
Caminhos	Concha	46	2280	0
N = 50				
	Estrutura	Caminhos	Arestas	Tempo
DFS	Árvore	25	98	0
Caminhos	Árvore	25	147	0
DFS	Concha	1	194	0
Caminhos	Concha	96	9555	0
N = 100				
	Estrutura	Caminhos	Arestas	Tempo
DFS	Árvore	50	198	0
Caminhos	Árvore	50	297	0
DFS	Concha	1	394	0
Caminhos	Concha	196	39105	0

[2] Ver item Código Fonte.

[3] As execuções do programa foram efetuadas em uma máquina SGI Power Challenge com 4 CPUs de 194 Mhz.

Código Fonte

```
//dados.h

struct aresta
     {int destino;
      struct aresta * next;
     };
typedef struct aresta tipo_aresta;

typedef struct listaAresta
     {tipo_aresta * first;
     }
tipo_listaAresta;

typedef struct vertice
    {int marcado;
     tipo_listaAresta arestasMarcadas,arestasDesmarcadas;
    }
tipo_vertice;

void IniciaListaAresta(tipo_listaAresta * pLista);
tipo_aresta * CriaAresta(int pDestino);
void InsereAresta(tipo_listaAresta * pLista,tipo_aresta * pAresta);
tipo_aresta * RemoveArestaInicio(tipo_listaAresta * pLista);
void RemoveAresta (tipo_listaAresta * pLista,tipo_aresta * pAresta);
void DestroiLista(tipo_listaAresta * pLista);
```

```
//dados.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "dados.h"

void IniciaListaAresta(tipo_listaAresta * pLista)
{pLista->first = NULL;
}

tipo_aresta * CriaAresta(int pDestino)
{tipo_aresta * pointer;

 if (pointer = (tipo_aresta *)malloc(sizeof(tipo_aresta)))
       {pointer->destino = pDestino;
        pointer->next = NULL;
        return pointer;
       }
 else  {printf ("Insufficient memory for allocation - at function CriaAresta\n");
        exit(1);
        return NULL;
       }
};


void InsereAresta(tipo_listaAresta * pLista,tipo_aresta * pAresta)
{pAresta->next = pLista->first;
 pLista->first = pAresta;
};
```

```
tipo_aresta * RemoveArestaInicio(tipo_listaAresta * pLista)
{tipo_aresta * pointer;

 pointer = pLista->first;
 if (pointer != NULL)
     pLista->first = pointer->next;
 return pointer;
};

void RemoveAresta (tipo_listaAresta * pLista,tipo_aresta * pAresta)
{tipo_aresta * prior;

 if (pLista->first == pAresta)
      {pLista->first = pAresta->next;
      }
 else {prior = pLista->first;
       while (prior->next != pAresta)
            prior = prior->next;
       prior->next = pAresta->next;
      }
};

void DestroiLista(tipo_listaAresta * pLista)
{tipo_aresta * pointer, * auxPointer;

 pointer = pLista->first;
 while (pointer != NULL)
     {auxPointer = pointer->next;
      free(pointer);
      pointer = auxPointer;
     }
};
```

```
//principal.c

#include <stdlib.h>
#include <time.h>
#include "dados.h"

#define MAX_VERTICES 101

tipo_vertice vertices[MAX_VERTICES];
long int arestasVistas,   //quantidade de arestas percorridas
         caminhosVistos,  //quantidade de caminhos descobertos
         tamanhoCaminho;  //variavel de controle necessaria para o algoritmo Caminhos

//cria grafo em forma de arvore binaria
void CriaArvore(int pNVertices)
{int i,pai;
 tipo_aresta * pointer;

 for (i = 0;i < MAX_VERTICES;i++)
     {IniciaListaAresta(&(vertices[i].arestasMarcadas));
      IniciaListaAresta(&(vertices[i].arestasDesmarcadas));
      vertices[i].marcado = 0;
     }
 for (i = 2;i <= pNVertices;i++)
     {pai = i/2;
      pointer = CriaAresta(i);
      InsereAresta(&(vertices[pai].arestasDesmarcadas),pointer);
```

```
      pointer = CriaAresta(pai);
      InsereAresta(&(vertices[i].arestasDesmarcadas),pointer);
     }
}

//cria grafo em forma de concha
void CriaConcha(int pNVertices)
{int i;
 tipo_aresta * pointer;

 for (i = 0;i < MAX_VERTICES;i++)
     {IniciaListaAresta(&(vertices[i].arestasMarcadas));
      IniciaListaAresta(&(vertices[i].arestasDesmarcadas));
      vertices[i].marcado = 0;
     }
 for (i = 2;i<pNVertices;i++)
     {pointer = CriaAresta(i + 1);
      InsereAresta(&(vertices[i].arestasDesmarcadas),pointer);
      pointer = CriaAresta(i);
      InsereAresta(&(vertices[i+1].arestasDesmarcadas),pointer);
     }
 for (i = 2;i<=pNVertices;i++)
     {pointer = CriaAresta (i);
      InsereAresta(&(vertices[1].arestasDesmarcadas),pointer);
      pointer = CriaAresta(1);
      InsereAresta (&(vertices[i].arestasDesmarcadas),pointer);
     }
}

//cria grafo completo
void CriaCompleto(int pNVertices)
{int i,j;
 tipo_aresta * pointer;

 for (i = 0;i < MAX_VERTICES;i++)
     {IniciaListaAresta(&(vertices[i].arestasMarcadas));
      IniciaListaAresta(&(vertices[i].arestasDesmarcadas));
      vertices[i].marcado = 0;
     }
 for (i = 1;i<pNVertices;i++)
     {for (j = i+1;j<=pNVertices;j++)
         {pointer = CriaAresta(j);
          InsereAresta(&(vertices[i].arestasDesmarcadas),pointer);
          pointer = CriaAresta(i);
          InsereAresta(&(vertices[j].arestasDesmarcadas),pointer);
         }
     }
}

//algoritmo Caminhos
void Caminhos(int raiz)
{tipo_aresta * ptAresta,* ptAux;

 vertices[raiz].marcado = 1;
 ptAresta = vertices[raiz].arestasDesmarcadas.first;
 while (ptAresta != NULL)
    {arestasVistas++;
     ptAux = ptAresta->next;
     if (vertices[ptAresta->destino].marcado)
          {RemoveAresta(&(vertices[raiz].arestasDesmarcadas),ptAresta);
           InsereAresta(&(vertices[raiz].arestasMarcadas),ptAresta);
          }
     ptAresta = ptAux;
    }
```

```
 if (tamanhoCaminho > 0)
    {if (vertices[raiz].arestasDesmarcadas.first == NULL)
         caminhosVistos++;
    }
 while (vertices[raiz].arestasDesmarcadas.first != NULL)
    {ptAresta = RemoveArestaInicio(&vertices[raiz].arestasDesmarcadas);
     InsereAresta(&vertices[raiz].arestasMarcadas,ptAresta);
     arestasVistas++;
     tamanhoCaminho++;
     Caminhos(ptAresta->destino);
     tamanhoCaminho—;
    }
 while (vertices[raiz].arestasMarcadas.first != NULL)
    {ptAresta = RemoveArestaInicio(&vertices[raiz].arestasMarcadas);
     InsereAresta(&vertices[raiz].arestasDesmarcadas,ptAresta);
    }
 vertices[raiz].marcado = 0;
}

//Algoritmo DepthFirstSearch
void DepthFirstSearch(int raiz)
{int expandiu;
 tipo_aresta * ptAresta;

 vertices[raiz].marcado = 1;
 expandiu = 0;
 ptAresta = vertices[raiz].arestasDesmarcadas.first;
 while (ptAresta != NULL)
     {arestasVistas++;
      if (vertices[ptAresta->destino].marcado == 0)
          {expandiu = 1;
           DepthFirstSearch(ptAresta->destino);
          }
      ptAresta = ptAresta->next;
     }
 if (expandiu == 0)
     caminhosVistos++;
}

//principal
int main()
{long int miliSec = 0;

 caminhosVistos = arestasVistas = tamanhoCaminho = 0;
 CriaCompleto(10);
 clock();
 DepthFirstSearch(1);
 miliSec = clock()/CLOCKS_PER_SEC*1000;
 printf ("caminhos %d arestas %d tempo %d\n",caminhosVistos,arestasVistas,miliSec);

 exit(0);
}
```

Conclusões

Algoritmos eficientes para classificação de caminhos ponderados não são sempre aplicáveis a problemas do mundo real em função das restrições impostas sobre a entrada de dados. Nestas situações, o uso de algoritmos de busca exaustiva torna-se viável na busca de resultados confiáveis. Esta viabilidade dependerá do tamanho dos dados de entrada e do poder de processamento das máquinas disponíveis, uma vez que a baixa eficiência é o preço a ser pago.

O algoritmo apresentado pode ser utilizado em aplicações que precisem da determinação dos caminhos oriundos de um vértice. Um exemplo de tal aplicação para o estudo de proteínas é apresentado na introdução. Outro exemplo é a determinação de todas as rotas possíveis para uma transmissão *broadcast* a partir de um nó transmissor.

Referências Bibliográficas

BERGE, C. **Graphs**. 2nd ed. Amsterdam: North-Holland, 1985. 413 p. (North-Holland Mathematical Library, v. 6, pt. 1).

EVEN, S. **Graph algorithms**. Rockville: Computer Science Press, 1979. 249 p. (Computer Software Engineering Series).

GONDRAN, M.; MINOUX, M. **Graphs and algorithms**. Chichester: John Wiley, 1984. 650 p. (Wiley-Interscience Series in Discrete Mathematics).

SEDGEWICK, R. **Algorithms**. Reading: Addison-Wesley, 1984. 551 p. (Addison-Wesley Series in Computer Science).

SWAMY, M. N. S.; THULASIRAMAN, K. **Graphs, networks, and algorithms**. John Wiley, 1981. 592 p.

Comunicado Técnico, 15

MINISTÉRIO DA AGRICULTURA, PECUÁRIA E ABASTECIMENTO

GOVERNO FEDERAL
Trabalhando em todo o Brasil

Embrapa Informática Agropecuária
Área de Comunicação e Negócios
Av. Dr. André Tosello s/nº
Cidade Universitária - "Zeferino Vaz"
Barão Geraldo - Caixa Postal 6041
13083-970 - Campinas, SP
Telefone/Fax: (19) 3789-5743
E-mail: sac@cnptia.embrapa.br

1ª edição