EDGRAFO - Um Sistema Computacional para Usuários de Grafos

Werner Adler
Lilian Markenzon

NCE-02/89

Julho/89

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

ABSTRACT

A computacional system for visual graph simulations is presented. It is composed of three modules that can be used individually or not. The first module implements an Abstract Data Type Graph that supports an edge-oriented way of handling graphs. The second module is a graphic editor specific for graphs and the third is a visualization-routine set.
The main advantage of the system is the easy interface between the modules. Therefore a graph previously drawn with the editor can be used as an input for a program. It also permits the visual follow-up of an implemented algorithm.

RESUMO

Este trabalho apresenta um sistema computacional para visualização dinâmica de programas em grafos. O sistema se compõe basicamente de três módulos que podem ser usados interligados ou não. O primeiro módulo é a implementação do Tipo Abstrato de Dados Grafo, com uma abordagem voltada para manipulação de arestas. O segundo módulo é um editor de uso específico para a construção de grafos e o terceiro módulo é um conjunto de rotinas que visa a exibição de resultados.
A vantagem do istema é justamente a fácil comunicação entre os módulos existentes. Desta forma, um grafo desenhado via editor pode ser utilizado como entrada para um programa que utiliza o Tipo Abstrato de Dados Grafo, bem como ter sua imagem modificada pelas rotinas do terceiro módulo.

1 - Introdução

A partir da década dos setenta, a evolução e disseminação dos computadores estimulou a pesquisa de algoritmos eficientes para solucionar problemas em grafos. Muitas vezes, entretanto, a convivência usuário de grafos x computador não é totalmente pacífica. O presente trabalho trata de diagnosticar alguns pontos sensíveis nesta relação e apresentar tentativas para solucioná-los através da implementação de ferramentas computacionais. O projeto de pesquisa e desenvolvimento destas ferramentas é chamado EDGRAFO.

2 - Motivação

Estão relacionados abaixo tres aspectos básicos nos quais se concentram as maiores dificuldades surgidas na elaboração e implementação de algoritmos em grafos. Pela sua diversidade, estas podem, ou não, coexistir. Assim sendo, as soluções apresentadas, na medida do possível, são independentes entre si.

1º) Implementação de algoritmos

A representação geométrica de um grafo e suas representações para uso no computador são muito diferentes. Para cada problema existe uma estrutura de armazenamento conveniente, como listas encadeadas e matrizes de adjacência ou incidência. Assim, pode ser penoso para o usuário a escolha e manipulação destas estruturas.

É claro que devemos levar em consideração a existência de diferentes tipos de usuários em grafos. Pesquisadores que desenvolvem algoritmos devem conhecer profundamente a implementação que será utilizada, uma vez que esta pode, inclusive, alterar a complexidade do algoritmo tratado. Para estes, obviamente, a implementação de algoritmos não é considerado um assunto que apresente dificuldades.

Nossa atenção se volta para outro grupo, composto por aqueles que utilizam estes algoritmos, isto é, usuários que programam algoritmos, ou mesmo os desenvolvem sem maiores preocupações de natureza teórica. As dificuldades encontradas na escolha e manipulação da estrutura adequada lhes acarretam um distanciamento cada vez maior do objetivo primitivo, simplesmente obter resultados para um problema modelado por grafos, ou mesmo o reconhecimento de propriedades estruturais em um dado problema...

Um grupo especial de usuários deve ainda ser mencionado. Em cursos introdutórios de algoritmos em grafos os problemas apresentados no parágrafo anterior também se apresentam. Se bem que com finalidades diferentes, a solução empregada pode se apresentar também como de utilidade no ensino.

2º) Representação gráfica.

Como já foi mencionado, é grande a distância existente entre a representação geométrica do grafo e as estruturas utilizadas para armazená-lo em computador. Sendo assim, é impossível utilizar a representação tradicional de um problema como entrada para programas. Cada massa de dados deve ser convertida à representação empregada, para que possa ser processada.

Um aspecto diverso do problema é a impressão de um grafo quando é conhecida sua representação geométrica. é óbvio que qualquer pacote gráfico, mesmo com poucos recursos, permite o traçado de um grafo. Deve-se porém levar em consideração o esforço dispendido no aprendizado de um novo pacote, que nem sempre satisfaz totalmente o usuário porque não é específico para sua aplicação.

Alguns trabalhos utilizam outra abordagem para o problema da impressão do grafo, tratando seu traçado a partir da estrutura abstrata do mesmo. Como referências citamos Rowe et al. [1987] e Kamada e Kawai [1989].

3º) Exibição de resultados

Na realidade, este problema se enquadra na tarefa mais ampla de visualizar a execução de um algoritmo e apresentar, também visualmente, seus resultados. Ao receber respostas de um programa o usuário é, muitas vezes, obrigado a "reconverter" a solução fornecida para que a representação gráfica habitual, importante muitas vezes, possa ser analisada. Na realidade este esforço é equivalente à construção (manual) de um novo grafo. Obviamente, qualquer que seja o problema proposto, cada vez que o programa é executado para um novo grafo, todo este trabalho de "conversão" deve ser realizado novamente.

3 - Projeto EDGRAFO

O equipamento utilizado em nosso projeto é um microcomputador IBM-PC compatível com placa gráfica CGA, vídeo monocromático ou colorido, com memória de 640K, conectado a uma impressora padrão EPSON.

A solução aqui proposta é formada por três módulos, interligados entre si. Cada um deles é apresentado com detalhes a seguir.

Módulo I: Tipo Abstrato de Dados Grafo

Pode-se pensar num tipo abstrato de dados (TAD) como um modelo matemático com uma coleção de operações definida sobre este modelo. O TAD encapsula a estrutura de dados no sentido que todas as operações podem estar localizadas numa única seção do programa, fora da qual o TAD pode ser tratado como uma estrutura primitiva (Aho, Hopcroft e Ullman [1983]). A implementação do TAD Grafo permite então ao usuário se abstrair completamente de detalhes de programação, inclusive da própria estrutura de armazenamento que suporta o tipo. Este fato minimiza erros de programação e facilita mudanças na implementação.

A importância da utilização de tipos abstratos de dados como recurso educacional é amplamente discutida em Moffat [1986]. Liss e McMillan [1987] apresentam um projeto de programação específico para o TAD Árvore Binária.

A abordagem escolhida para a implementação do TAD Grafo é a apresentada em Ebert [1987], que enfatiza a descrição das arestas. Esta abordagem é particularmente atraente por tratar grafos e digrafos indiferentemente, bem como permitir arestas valoradas.

A notação utilizada para grafos e digrafos é a tradicional, vista, por exemplo, em Szwarcfiter [1984]. Apenas alguns conceitos devem ser acrescentados. Seja um grafo, ou digrafo, G = (V,E) com n vértices e m arestas. Seja uma aresta e = (v,w), pertencente ao conjunto E. Denotamos *alfa(e)* o vértice v, vértice de partida da aresta e, e *omega(e)* o vértice w, vértice de chegada da aresta e. Nomeia-se *estrela* do vértice v ao conjunto de arestas que tem extremidade em v; *estrela exterior* do vértice v ao conjunto de arestas tais que v = alfa(e) e *estrela interior* do vértice v ao conjunto de arestas tais que v = omega(e). Constata-se então que, na realidade, mesmo no caso de grafos, as arestas tem uma orientação que pode, ou não, ser utilizada. Por exemplo,se é necessário percorrer arestas de um grafo somente uma vez, este conceito pode ser utilizado.

Cada aresta e, e∈E, pode ser encarada como um objeto com dois estados. Quando a aresta e pertence à estrela exterior de um vértice ela tem sinal positivo; quando pertence à estrela interior, seu sinal é negativo. Obviamente isto não interfere na conceituação de vértice de partida e de chegada; a aresta e, ou -e, é única. Nesta abordagem um digrafo e seu grafo subjacente têm a mesma representação; algoritmos elaborados para grafos podem ser executados em digrafos sem maiores modificações.

O conjunto de operações básicas associadas ao TAD, apresentadas a seguir acompanhadas por uma descrição

sucinta, se baseiam nos conceitos que foram apresentados. Deste modo, alem de procedimentos para criação e operação de grafos e digrafos são também fornecidos procedimentos para manipulação de arestas "com sinal".

Procedimentos para criação do grafo e alterações posteriores:

create-vertex : cria um vértice e retorna seu número
delete-vertex(v) : destrói o vértice v
create-edge(v,w) : cria uma aresta e retorna seu número
delete-edge(e) : destrói a aresta e

Procedimentos para manipulação de arestas:

alfa(e) : retorna o vértice de partida da aresta e
omega(e) : retorna o vértice de chegada da aresta e
this(e) : retorna o vértice de partida da aresta e, se esta é positiva, e seu vértice de chegada em caso contrário
that(e) : retorna o vértice de chegada da aresta e se esta é positiva, e seu vértice de partida em caso contrário
normal(e) : retorna a aresta e com direção positiva
reverse(e) : retorna a aresta e com direção invertida

Procedimentos para operação no grafo:

first-out(v) : retorna a primeira aresta da estrela exterior do vértice v
next-out(e) : retorna a aresta consecutiva a e na estrela exterior do vértice v = alfa(e)
first-in(v) : retorna a primeira aresta da estrela interior do vértice v
next-in(e) : retorna a resta consecutiva a e na estrela interior do vértice v = omega(e)
first(v) : retorna a primeira aresta da estrela de v
next(e) : retorna a aresta seguinte a aresta e na estrela de v = this(e)
first-vertex : retorna o primeiro vértice do grafo
next-vertex(v) : retorna o vértice consecutivo a v no grafo
first-edge : retorna a primeira aresta do grafo
next-edge(e) : retorna a aresta consecutiva a aresta e no grafo

Foram acrescentadas rotinas de tratamento de erros já mencionadas como necessárias na referência utilizada. Além destas foram incluidos também os procedimentos *input(grafo)*, que executa a leitura de um grafo já processado anteriormente, e *output(grafo)*, que arquiva um grafo para posterior reaproveitamento.

A implementação do TAD Grafo, realizada em Turbo Pascal 4.0, é muito simples, utilizando a representação interna também vista em Ebert [1987], totalmente baseada em vetores. A única inconveniência encontrada na implementação é a restrição de dimensionamento do grafo tratado; nesta implementação optamos por 560 vértices e 2500 arestas, dimensões confortáveis para a maioria das aplicações.

Módulo II: Editor de Grafos

Os problemas da visualização dos dados de entrada e a impressão de grafos podem ter uma solução comum; propomos um editor específico para grafos, que é também capaz de gerar a entrada de um programa. A meta é a comunicação entre os módulos I e II, que permite a rápida modificação de dados de entrada, mantendo sempre presente o conceito da "visão" de um grafo.

O módulo é interativo, permitindo ao usuário se comunicar facilmente com esta ferramenta para desenhar grafos. A tela é dividida em duas regiões: cardápio e região para exibição do grafo. A região do cardápio contém cardápio principal ou secundários. A figura 1 mostra a hierarquia dos mesmos. A região de exibição exibe sempre uma grade onde o grafo é desenhado. A dimensão da grade decorre do tamanho desejado para os vértices. Formas diferentes para arestas (retas ou curvas) e vértices (triângulo, quadrado ou círculo) podem coexistir num desenho. Estas possibilidades são apresentadas em figuras do trabalho. A entrada dos movimentos do cursor é executa via teclado.

Se bem que muito simples, a implementação do editor gráfico apresenta alguns problemas interessantes. O primeiro, e mais

importante, é a impressão, com boa qualidade, do grafo visível na tela do editor. Como a impressão diretamente da tela deixa a desejar, é necessário a implementação de uma tela virtual onde o grafo é redesenhado com maior precisão. Nesta tela as deformações da futura impressão são compensadas e esta então é a tela impressa. A figura 2 apresenta uma imagem da tela e a figura 3 a impressão resultante do editor. Outro problema é causado pela possibilidade de dimensões diversas para a grade (e consequentemente para os vértices). A grade de tamanho pequeno oferece 40x28 células; ao se considerar o tamanho médio se passa a ter quatro diferentes telas e no tamanho grande dezesseis. O corte de arestas que resulta foi resolvido pelo algoritmo introduzido em Nicholl, Lee e Nicholl [1987]. As figuras 4 e 5 mostram a impressão da tela de um grafo nos tamanhos pequeno e médio (a tela escolhida foi a superior esquerda).

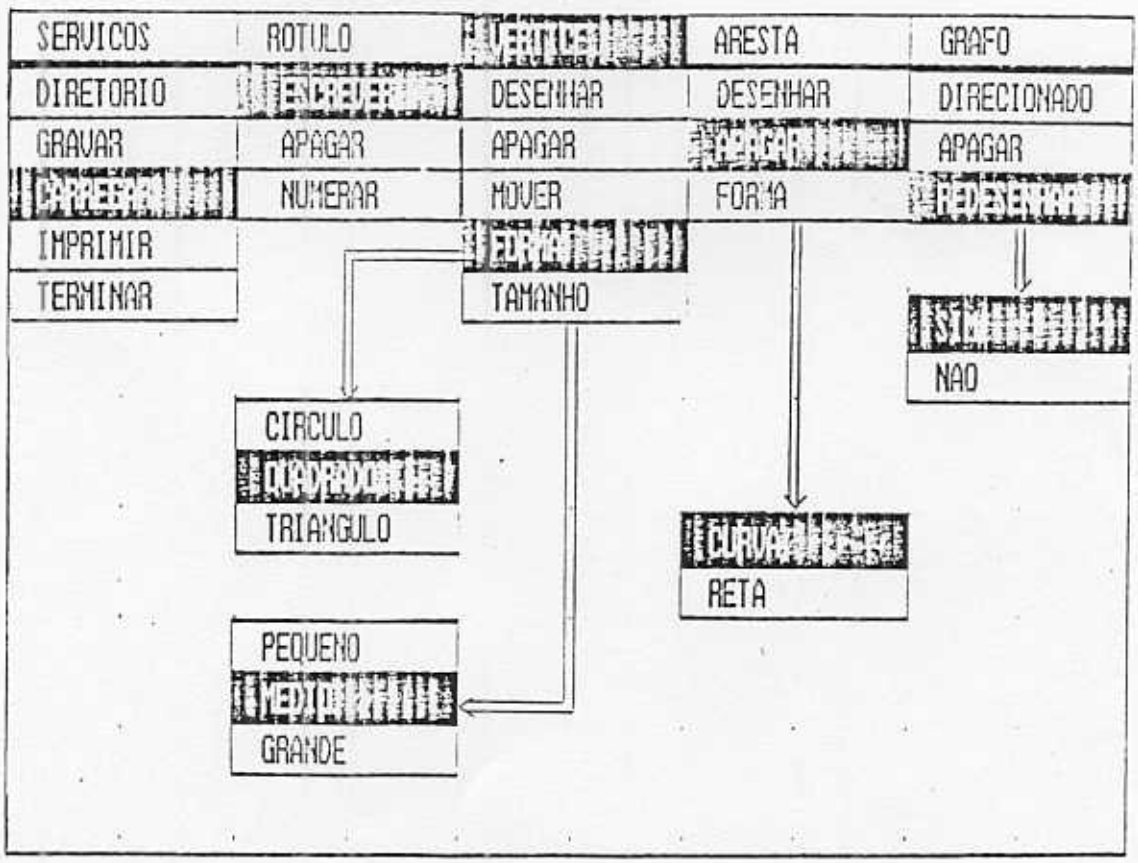

figura 1

A saída do editor é composta por dois arquivos. O primeiro, armazena a estrutura abstrata do grafo utilizando a estrutura de armazenamento também empregada no módulo I. O

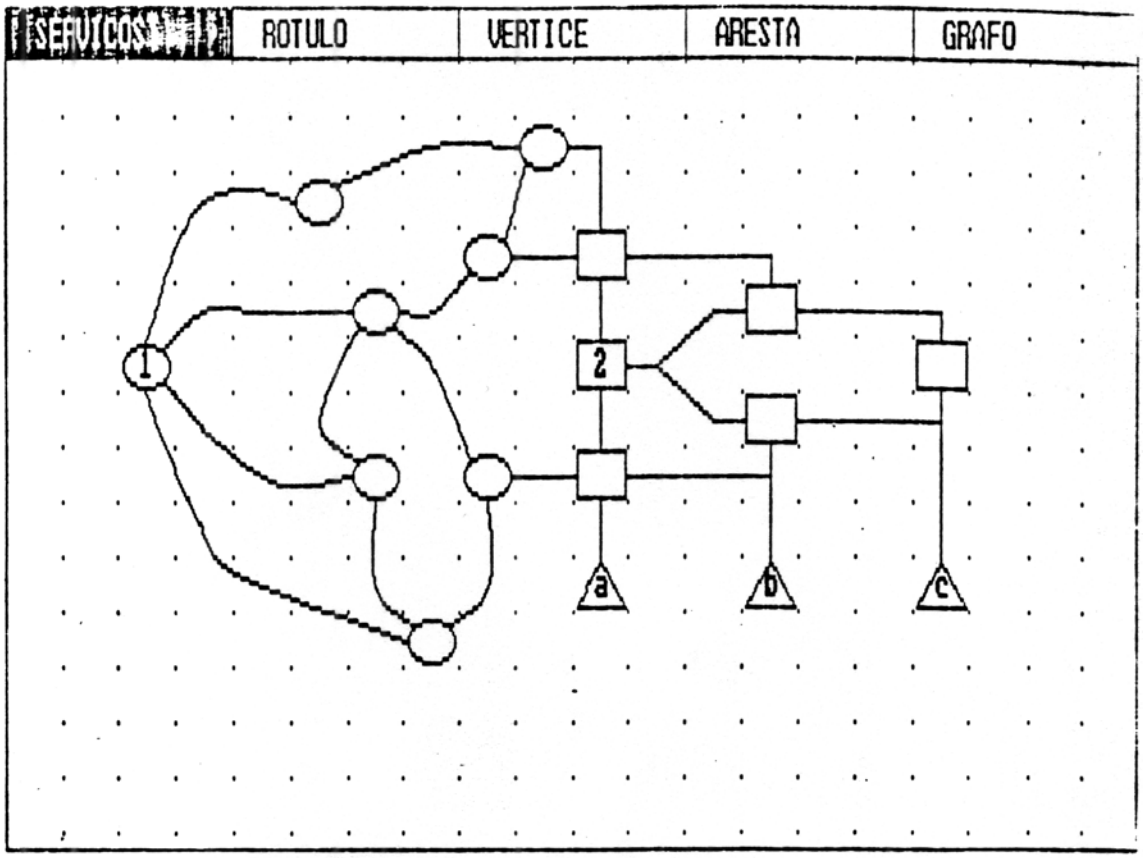

Figura 2

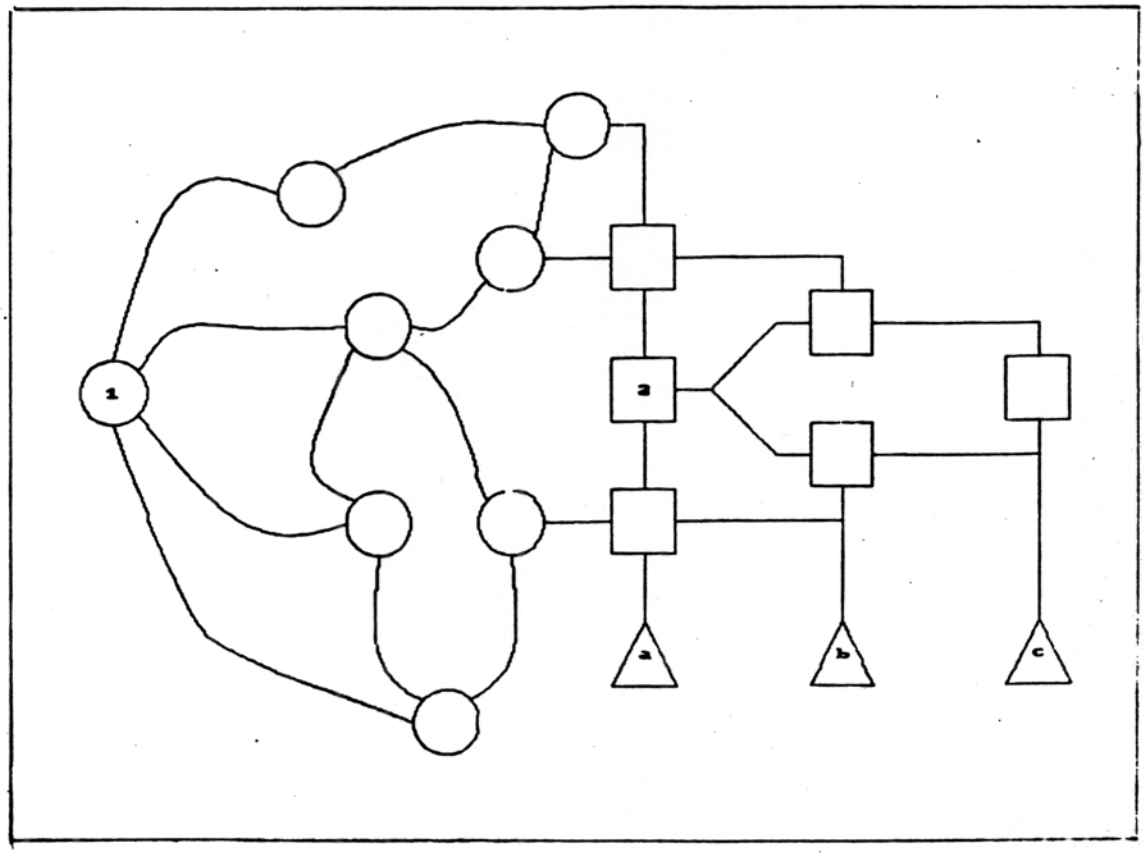

Figura 3

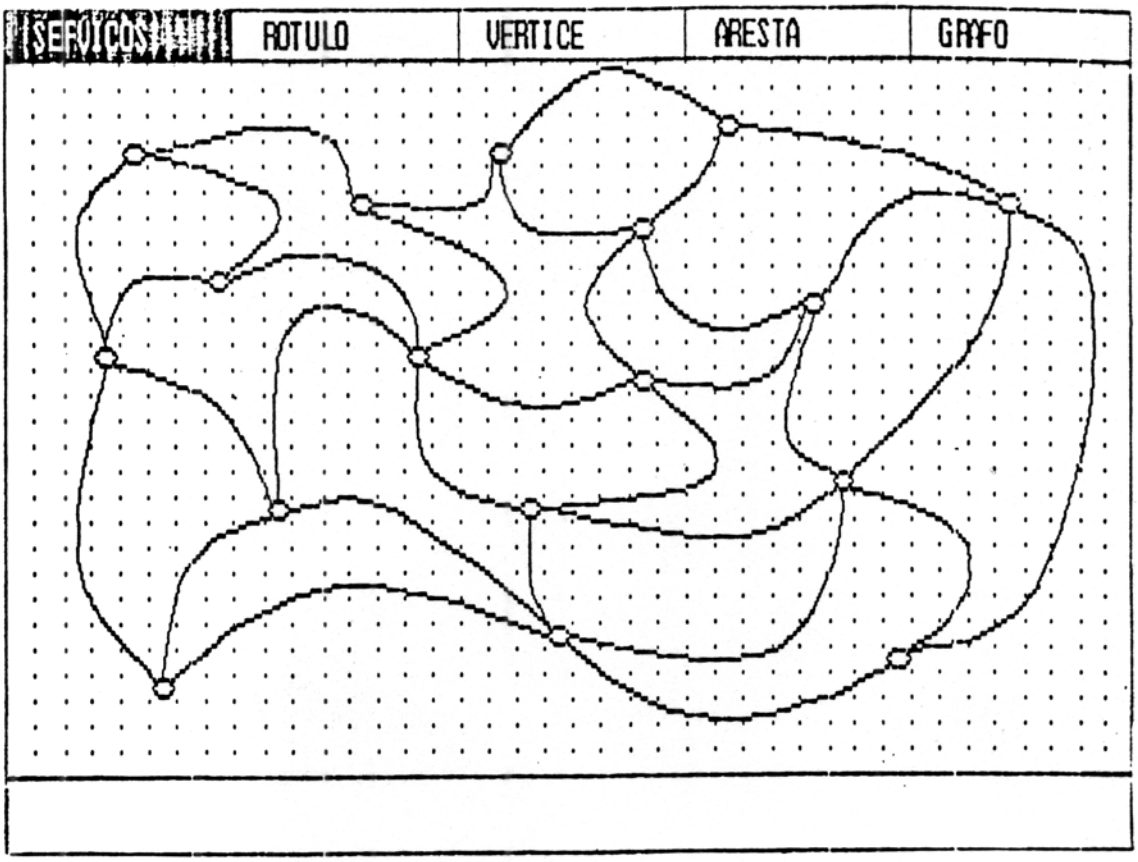

Figura 4

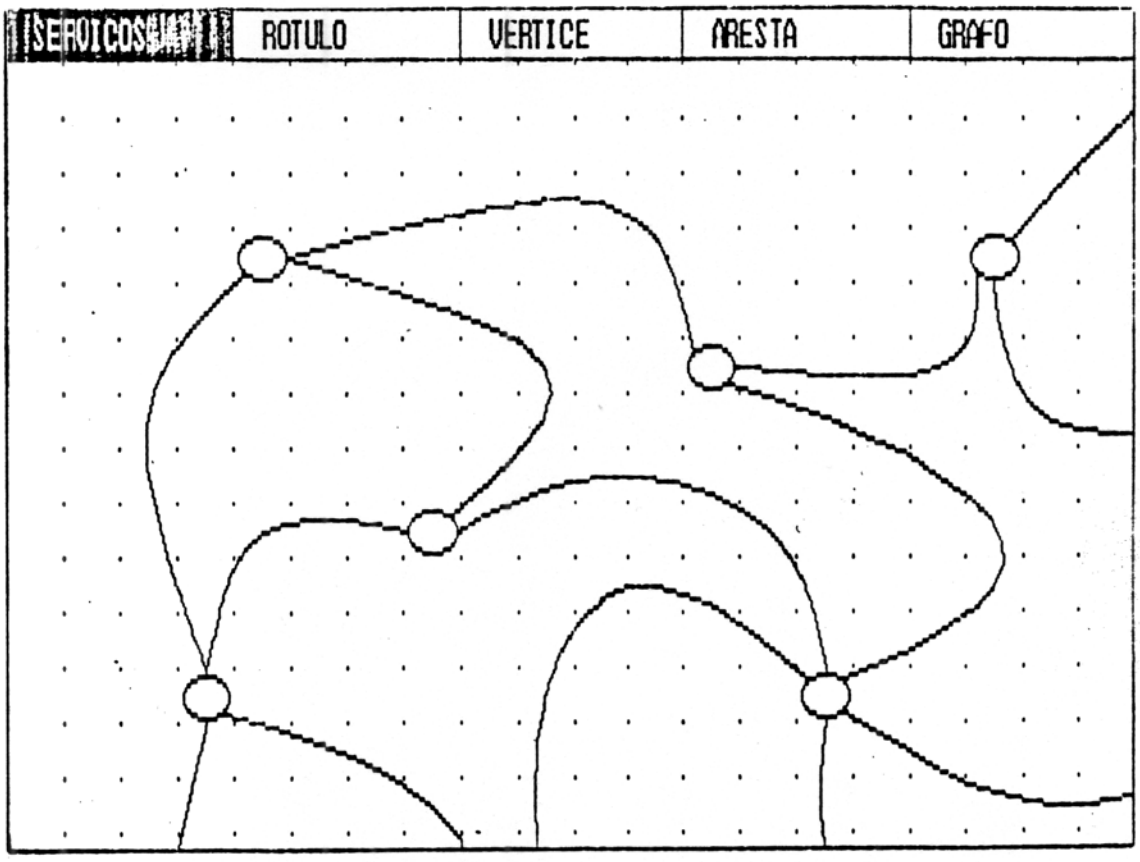

Figura 5

segundo codifica as informações geométricas, necessárias para a reutilização do desenho. Note-se que o grafo armazenado pelo editor pode ser utilizado por qualquer programa. É óbvio que, pelas características do armazenamento, esta tarefa é muito simples para quem utiliza o módulo I.

Módulo III - Rotinas de Visualização

Visualizar respostas, ou mesmo acompanhar com efeitos visuais sobre a estrutura de dados a execução de um algoritmo, é tarefa árdua, para a qual já existem sugestões na literatura. Estruturas de dados mais simples do que grafos têm sido privilegiadas. Entre outros, Barnes e Kind [1987] apresentam um pacote para a simulação de algoritmos em árvores binárias, que mostra simultâneamente a representação gráfica, o algoritmo que está sendo analisado em pseudo-código e ainda outros recursos disponíveis. Trabalho semelhante, dedicado principalmente a listas, é o de Rambally [1985]. Em grafos, para o caso particular de máquinas de estados finitos, Jagielski [1988] apresenta uma simulação gráfica.

Um mesmo grafo pode modelar problemas diversos e diferentes maneiras de visualizar suas respostas podem ser necessárias. Neste trabalho, a abordagem escolhida foi então a de suprir o usuário com procedimentos de alteração. Isto significa poder modificar um grafo já desenhado anteriormente, realçando e suavisando vértices e arestas; pode-se mesmo gerar novos grafos, escolhendo subgrafos formados por vértices e arestas do grafo original com determinados atributos.

As rotinas propostas são:

Rotinas de tranferência

- *ingraf* e *outgraf* : ler e gravar em arquivo as informações geométricas relativas ao grafo, no mesmo padrão usado pelo editor apresentado no módulo I.

- *print* : imprime o grafo, com as mesmas alternativas permitidas no editor (tamanho de vértices e especificação de parte da grade)e com as opções de modificação produzidas pelas rotinas apresentadas abaixo.

- *display* : similar ao procedimento print, dedicada ao vídeo.

- *new* : gera novos arquivos (o que significa novos grafos) para as definições abstrata e geométrica, definidas de acordo com um subconjunto do grafo corrente. A especificação que define este subconjunto é produzida pelas rotinas de visualização, apresentadas abaixo.

Rotinas de visualização
- *soft-vertex(v)* e *soft-edge(e)* : o traçado do vértice ou da aresta é suavizado na impressão ou exibição na tela.

- *mask-vertex(v)* e *mask-edge(e)* : o vértice ou aresta são apagados na saída (impressa ou tela) escolhida.

- *normal-vertex(v)* e *normal-edge(e)* : a impressão do vértice ou aresta é executada normalmente.

- *soft-grafo*, *mask-grafo* e *normal-grafo* : todos os vértices e arestas do grafo tomam o estado indicado.

É importante frisar que os estados decorrentes de rotinas de visualização (soft e mask) são temporários, durando apenas o tempo de execução do programa. Por esta razão o procedimento new é de grande valia na fixação de resultados visuais. Por exemplo, podemos gerar um novo grafo formado pelos vértices e arestas suaves do grafo original.

Finalmente, é interessante ressaltar as múltiplas possibilidades de aplicação destes procedimentos para a visualização de algoritmos em grafos. Uma sequência de modificações pode gerar o acompanhamento visual dos efeitos do algoritmo sôbre o grafo, recurso de grande valia para o ensino. Como exemplos, a figura 6 apresenta, em realce, as

articulações de um grafo; a figura 7, as arestas de árvore de uma busca em profundidade aplicada ao grafo da figura anterior e a figura 8, o realce de um componente fortemente conexo de um digrafo.

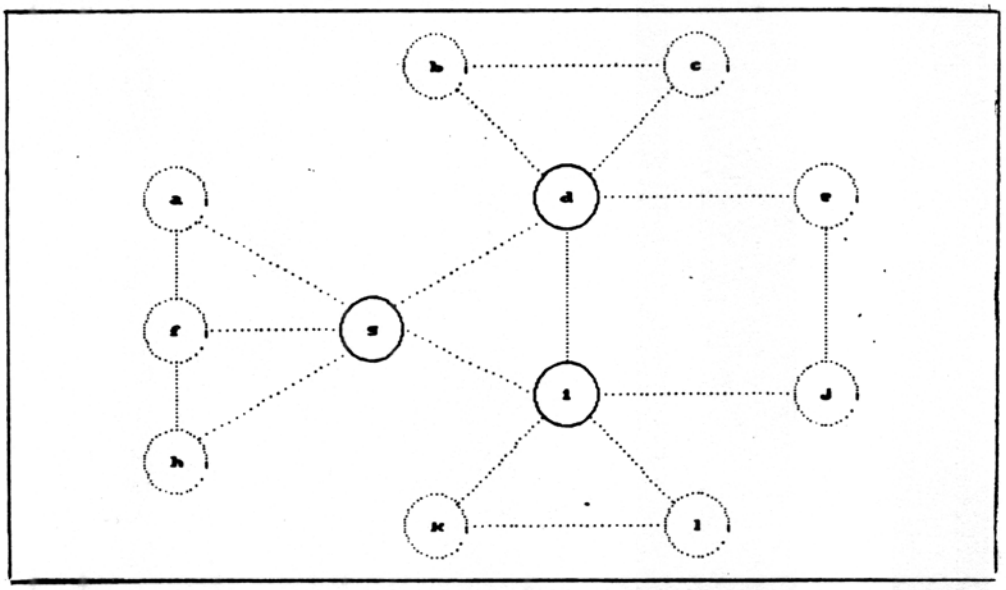

Figura 6

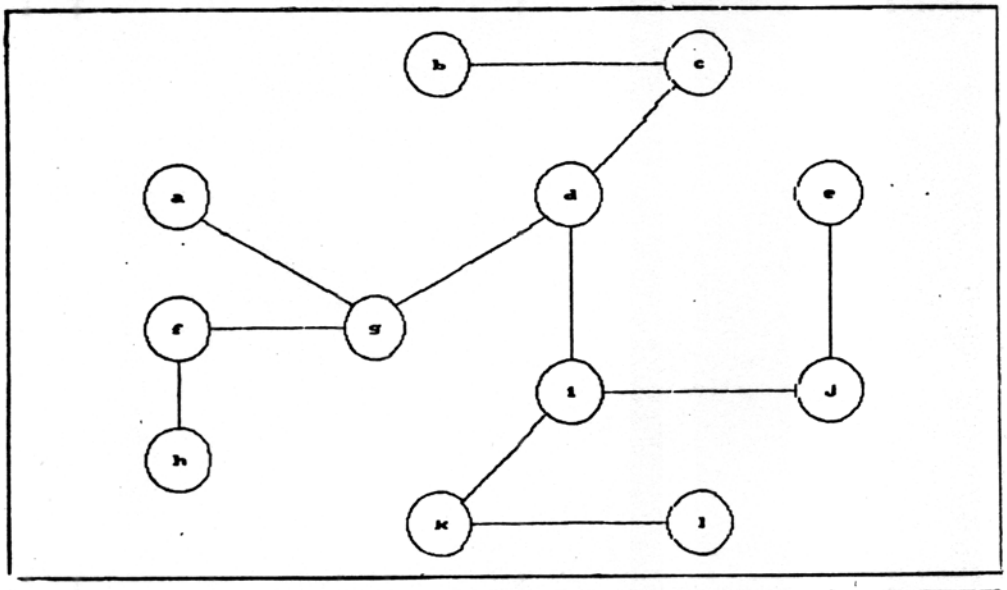

Figura 7

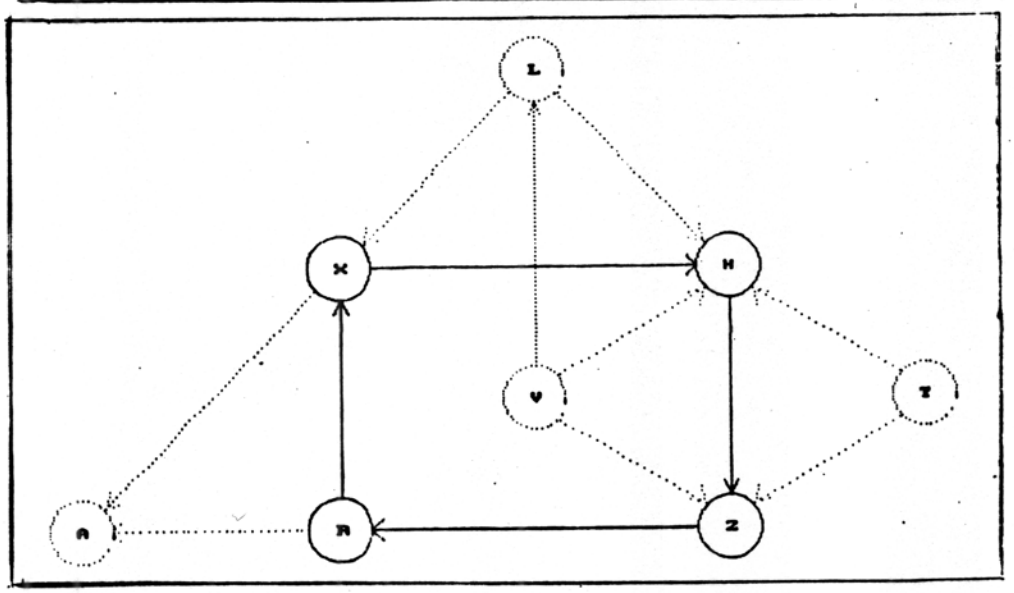

Figura 8

4 - Conclusão

As ferramentas apresentadas constituem-se em auxílio importante para ensino e pesquisa em grafos. A implementação do TAD Grafo é recurso substancial na construção de algoritmos, agilizando o processo de programação. A visualização imediata do grafo, bem como a possibilidade de alterações por algoritmo, justificam integralmente a existência do projeto apresentado.

No presente momento os módulos I e II se encontram implementados e em fase de teste. Detalhes de operação e implementação destes módulos podem ser encontrados em Adler [1989]. O módulo III, ainda sujeito à alterações, se encontra em fase final de desenvolvimento.

Referências Bibliográficas

Adler,W. [1989] - EDGRAFO - Uma Ferramenta para Usuários de Grafos, Trabalho Final de Curso, Instituto de Matemática, UFRJ

Aho,A.V.,Hopcroft,J.E., Ullman,J.D. [1983] - Data Structures and Algorithms, Reading, Mass., Addison-Wesley

Barnes,G.M., Kind,G.A. [1987] - Visual Simulations of Data Structures During Lecture, ACM SIGCSE Bulletin, vol.19, n.1, pp.267-276

Ebert,J. [1987] - A Versatile Data Structure for Edge-oriented Graph, Comm. of ACM, vol.30, n.6 pp.513-519

Jagielski,R. [1988] - Visual Simulation of Finite State Machines, ACM SIGCSE Bulletin, vol.20, n.4, pp.38-40

Kamada,T., Kawai,S. [1989] - An Algorithm for Drawing General Undirected Graphs, Inf. Proc. Lett., vol.31, pp.7-15

Liss,I., McMillan,T.C. [1987] - Trees - A CS2 Programming Project which Introduces a Data Type Using Procedural and Data Abstraction, ACM SIGCSE Bulletin, vol.19, n.1, pp.346-352

Moffat,D.V. [1986] - Teaching a Modern Data Structures Course, ACM SIGCSE Bulletin, vol.18, n.4, pp.57-64

Nicholl,T.M., Lee,D.T., Nicholl,R.A. [1987] - An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis, ACM Computer Graphics, vol.21, n.4, pp.253-262

Rambally,G.K. [1985] - Real-time Graphical Representations of linked Data Structures, ACM SIGCSE Bulletin, vol.17, n.1, pp.41-48

Rowe,L.A., Davis,M., Messinger,E., Meyer,C., Spirakis,C., Tuan,A. [1987] - A Browser for Directed Graphs, Software-Practice and Experience, vol.17, n.1, pp.61-76

Szwarcfiter,J.L.[1984] - Grafos e Algoritmos Computacionais, Rio de Janeiro, Campus