ALGORITMO DE DIJKSTRA: APOIO DIDÁTICO E MULTIDISCIPLINAR NA IMPLEMENTAÇÃO, SIMULAÇÃO E UTILIZAÇÃO COMPUTACIONAL

Edson A. R. Barros [1], Sergio V. D. Pamboukian [2], Lincoln C. Zamboni [3]

Abstract *— Since the decade of 70, the programming have showing itself as an essential and indispensable tool for students, engineers and researchers. It is fact that the programming languages developed from such a way to allow the conception of classes of objects and the modeling of the techno-scientific problems in a simple way. In the Escola de Engenharia of the Universidade Presbiteriana Mackenzie, that it is commemorating their 110 years of existence, grows several interconnected projects that use the programming as tool not only of search of the solution, but also didacticism and practice for the student. This is the case of use of the Algorithm of Dijkstra that, with its simplicity, reveals a didactic support in the implementation of classes, simulation and reuse in the most several engineering applications.*

Index Terms — programming languages, Dijkstra, programming with classes, simulation, reuse.

Resumo *— Desde a década de 70, a programação vem se mostrando como uma ferramenta essencial e indispensável para estudantes, engenheiros e pesquisadores. É fato que as linguagens de programação evoluíram de tal forma a permitir a concepção de classes de objetos e a modelagem dos problemas técnico-científicos de forma simples, clara e objetiva. Na Escola de Engenharia da Universidade Presbiteriana Mackenzie, em seus 110 anos de existência, desenvolve-se diversos projetos multidisciplinares que utilizam a programação como ferramenta não só de busca da solução, mas também no reúso desta solução em outros projetos. Este é o caso de uso do Algoritmo de Dijkstra que, com sua simplicidade, revela seu apoio didático e multidisciplinar na implementação de classes, simulação e reúso nas mais diversas aplicações de engenharia.*

Palavras chaves — linguagens de programação, Dijkstra, programação com classes, simulação, reúso.

INTRODUÇÃO

O Algoritmo de Dijkstra foi desenvolvido em 1959 por Edsger Wybe Dijkstra, de onde vem a origem do nome. Este algoritmo se aplica em grafos simples para determinar o caminho mínimo entre dois vértices. Ele é desenvolvido em uma árvore de caminhos mínimos. Quando o grafo não for simples, então se procura transformá-lo em um grafo simples eliminando seus laços e eventuais arestas múltiplas deixando apenas a com o peso menor. A complexidade do Algoritmo de Dijkstra é $O(n^2)$, onde n é a quantidade de vértices do grafo [1].

O ALGORITMO

Para entender os passos do Algoritmo de Dijkstra se deve seguir minuciosamente o Fluxograma apresentado na Figura 3. Inicialmente os vértices do grafo em estudo devem ser numerados, o que resulta em uma matriz de vínculos e pesos que pode ser usada como base para se desenvolver a lógica do programa.

Na Figura 1 se apresenta um grafo ilustrativo para melhor compreensão do algoritmo e na Figura 2 sua respectiva matriz de vínculos e pesos [1][2].

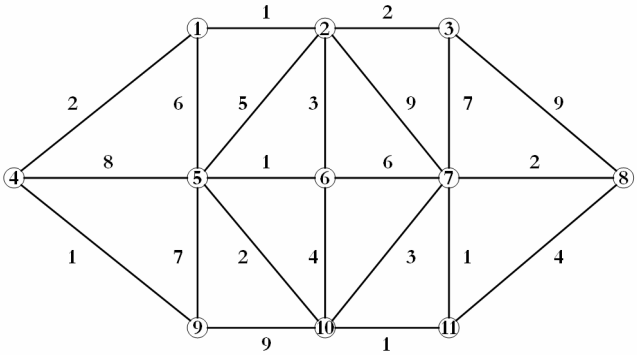

FIGURA. 1
GRAFO ILUSTRATIVO COM 11 VÉRTICES.

O passo seguinte é definir respectivamente o vértice origem (u0) e o vértice destino (uf) na matriz de dados. Para a atual demonstração, a origem é o vértice 4 (u0=4) e o destino é o vértice 8 (uf=8).

Na seqüência criam-se dois conjuntos de vértices, o M dos vértices marcados, e o D dos vértices desmarcados. A marcação pode ser feita por meio de um vetor com dados do tipo booleano (bool), verdadeiro (true) ou falso (false), dessa

1 Edson de Almeida Rego Barros, Escola de Engenharia, Universidade Presbiteriana Mackenzie, Rua da Consolação, 930, 01302-907, Consolação, São Paulo, SP, Brazil, prof_edson@mackenzie.com.br
2 Sergio Vicente Denser Pamboukian, Escola de Engenharia, Universidade Presbiteriana Mackenzie, Rua da Consolação, 930, 01302-907, Consolação, São Paulo, SP, Brazil, sergiop@mackenzie.com.br
3 Lincoln César Zamboni, Escola de Engenharia, Universidade Presbiteriana Mackenzie, Rua da Consolação, 930, 01302-907, Consolação, São Paulo, SP, Brazil, lincoln.zamboni @mackenzie.com.br

forma os marcados poderiam ter valor verdadeiro e os desmarcados o valor falso.

A próxima providência é a criação de dois vetores para definir os comprimentos acumulados (L) e o vértice anterior (A), ambos vetores guardam informações relativas ao nó em estudo. O vetor de comprimento (L) deve ter inicialmente o valor máximo (INFINITY em C++), enquanto o vetor vértices anteriores (A) deve apontar inicialmente para nulo.

	1	2	3	4	5	6	7	8	9	10	11
1	0	1	0	2	6	0	0	0	0	0	0
2	1	0	2	0	5	3	9	0	0	0	0
3	0	2	0	0	0	0	7	9	0	0	0
4	2	0	0	0	8	0	0	0	1	0	0
5	6	5	0	8	0	1	0	0	7	2	0
6	0	3	0	0	1	0	6	0	0	4	0
7	0	9	7	0	0	6	0	2	0	3	1
8	0	0	9	0	0	0	2	0	0	0	4
9	0	0	0	1	7	0	0	0	0	9	0
10	0	0	0	0	2	4	3	0	9	0	1
11	0	0	0	0	0	0	1	4	0	1	0

FIGURA. 2
MATRIZ DE VÍNCULOS E PESOS.

Uma vez preparadas as etapas anteriores, as estruturas de dados necessárias para o desenvolvimento do algoritmo estão prontas. Pode-se, então, dar inicio a uma rotina de verificação, vértice a vértice. Partindo-se do vértice inicial (u0) executa-se uma interação até atingir o vértice de destino (uf).

A rotina de interação consiste basicamente dos seguintes passos: a) verificar as conexões do vértice da vez (ui) que não foram usadas (desmarcadas); b) calcular as distâncias acumuladas das conexões disponíveis; c) identificar entre as disponíveis qual apresenta a menor distância acumulada e marcá-la; d) repetir a rotina até o vértice da vez ser o vértice de destino (ui = uf).

Depois de definidas as distâncias mínimas, ainda se faz necessária uma rotina para varrer os vértices anteriores se partindo do vértice destino (uf), a seqüência em ordem inversa dos caminhos mínimos. O algoritmo ainda contempla a verificação do caso de não existir a conexão entre os vértices inicial (u0) e final (uf).

APLICAÇÕES DO ALGORITMO

O algoritmo de Dijkstra pode ser empregado em vários contextos, desde o estudo de uma cadeia de produção até o clássico problema do carteiro que não pode passar duas vezes na mesma rua.

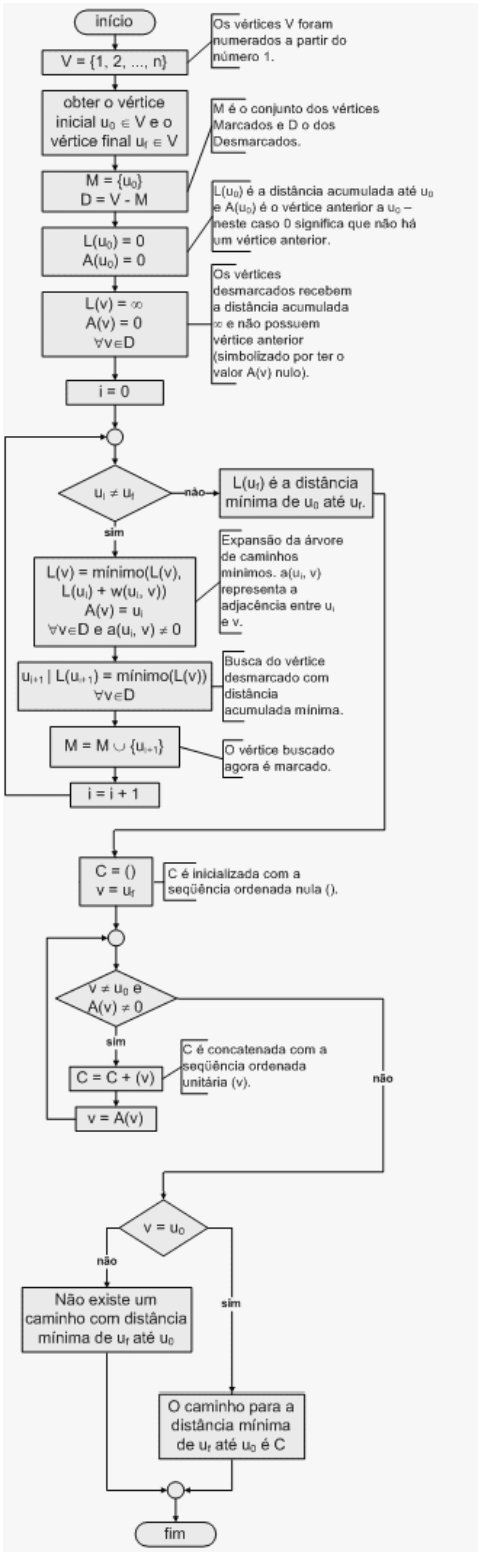

FIGURA. 3
FLUXOGRAMA DO ALGORITMO DE DIJKSTRA.

A matriz de vínculos e pesos pode ser composta por informações que representam distância entre localidades, custos operacionais, quantidade de recursos, ou qualquer outro elemento passível de ser ponderado entre os vértices [2].

Por sua vez, os vértices também podem assumir uma infinidade de possibilidades. Os vértices podem representar cidades ou qualquer outra localidade, podem ser os estados em um sistema dinâmico complexo, ou mesmo abstrações de momentos em um contexto.

Qualquer situação que possa ser representada por um grafo simples, que possuas sua matriz de pesos definida é passível de ser submetido a proposta de Dijkstra.

FUNÇÕES E CLASSES NA ESCOLA DE ENGENHARIA

Conforme narrado em [3], nos cursos de engenharia da universidade, procura-se fornecer aos alunos alguns conceitos fundamentais de linguagem de programação para que a mesma, em conjunto com outras disciplinas, possam auxiliá-lo na solução de problemas específicos de sua área. Estes conceitos trazidos para o âmbito da linguagem C++, e que podem ser facilmente migrados para qualquer outra linguagem orientada a objetos como Java, C#, Object Pascal, etc, são:

- tipos básicos de dados: int, double, char, bool, float, long int, long double, etc;
- instruções de controle do fluxo: seqüência, condição (if/else, switch) e repetição (for, do/while e while);
- reúso de classes de objetos de interface gráfica: formulários, rótulos, botões, caixas de edição, de checagem, de listagem, combinadas, grades, botões de rádio, etc;
- reúso de funções matemáticas e de tratamento de cadeias de caracteres: +, -, *, /, %, sqrt, sin, cos, atan, exp, log, etc;
- arranjos e estruturas: ponteiros, operadores [], new e delete para alocação dinâmica;
- funções especificadas pelo programador:
- declaração;
- definição;
- classes definidas pelo programador:
- construtor padrão, construtor de cópia, destruidor, operador de atribuição, encapsulamento e métodos de acesso;
- polimorfismo, métodos virtuais e virtuais puros;
- hereditariedade;
- fôrmas de classes e fôrmas de funções da Standard Template Library (STL): string, vector, list, etc.

Os conceitos anteriores são abordados em duas metodologias de programação:

- Estruturada: onde se trata, em essência, o conceito de função e suas aplicações;
- Orientada a Objetos: onde se trata, em essência, o conceito de classe e suas aplicações.

É interessante que tanto o conceito de função, como o de classe, são tratados de forma didática em duas fases crescentes de dificuldade: a primeira reusando funções e classes presentes no ambiente integrado da linguagem e a segunda desenvolvendo funções e classes para reuso.

DESAFIO PARA OS ALUNOS DA ESCOLA DE ENGENHARIA

Com a lógica do algoritmo apresentada aos alunos, sugere-se o desenvolvimento do mesmo tanto em forma de função como em forma de classe.

Para a parte destinada aos dados recomenda-se um fragmento de código como se segue:

```
const int MAX = 100; //Ordem máxima para arranjos.
double W[MAX][MAX];  //Matriz de pesos W.
double L[MAX];       //Distâncias acumuladas L.
int A[MAX];          //Vértices anteriores A.
bool M[MAX];         //Matriz de marcação M.
int n, i, j, Origem, Destino, Vert;
double NovaDist, Min;
```

A rotina da leitura da matriz de pesos sugere-se programação como o código seguinte:

```
// Leitura da matriz de pesos W
for(int i = 0; i < n; i++)
 for(int j = 0; j < n; j++)
    // W[i][j]= ...
```

As matrizes precisam ter valores iniciais apropriados para a programação, conforme o segmento de código exemplo abaixo:

```
//Valores iniciais para o algoritmo.
for(int i = 0; i < n; i++){
  // Preencha a matriz de marcação M.
  // M[i] = ... //Pode ser false ou true
  // Preencha as distâncias acumuladas L.
  // L[i] = ... //Um número grande representa ∞.
}
```

A próxima rotina necessária destina-se a definir os caminhos menores:

```
Vert = Origem;
L[Vert] = 0;
while(Vert != Destino && Vert != 0){
 for(int i = 0; i < n; i++)
  if(W[Vert][i] <> 0 && !M[i]){
   NovaDist = L[Vert] + W[Vert][i];
    if(NovaDist < L[i]){
     L[i] = NovaDist;
     A[i] = Vert;}
  }
```

```
  M[Vert] = true;
  // Procura do próximo vértice.
  Min = ...; // Número grande representa ∞
  Vert = 0;
  for(int i = 0; i < n; i++)
   if(!M[i] && L[i] < Min){
      Min = L[i];
      Vert = i;}
  }
```

O procedimento para quando o vértice de destino é encontrado. Na seqüência, a rotina apresenta o caminho mais curto:

```
// Caminho mais curto.
if(Vert == Destino){
// Exiba o primeiro vértice
// ...
while(Vert != Origem){
  Vert = A[Vert];
  // Exiba o valor de Vert novamente.
  // ...
  }
}
else{
// Exibe mensagem dizendo que não existe um
// caminho do vértice Origem ao Destino
// ...
}
```

RESULTADOS

No caso do grafo ilustrativo apresentado na figuras 1 e com a matriz de vínculos e pesos da figura 2 o resultado do processamento pode ser observado na figura 4.

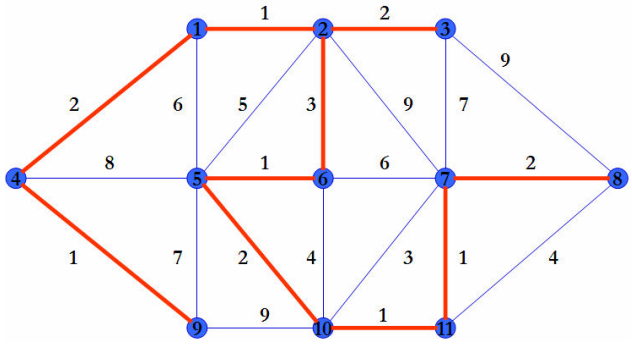

FIGURA. 4
GRAFO ILUSTRATIVO COM CAMINHO MÍNIMO CALCULADO.

CONSIDERAÇÕES FINAIS

Neste trabalho vimos uma aplicação prática de uma linguagem de programação como ferramenta auxiliar para a resolução de problemas de engenharia. Em [1] o leitor encontrará outras boas aplicações de classes na engenharia; em [4] um enfoque didático com objetos de interface gráfica em aplicações de engenharia; em [3] e [5] os fundamentos da programação estruturada com exemplos ligados à engenharia. Em [6] encontra-se excelentes implementações de classes e algoritmos para aplicações numéricas não só para a engenharia. [6] e [7] formam uma referência mundial na programação numérica em C/C++.

REFERÊNCIAS

[1] ZAMBONI, L. C.; PAMBOUKIAN, S. V. D.; BARROS, E. A. R., "C++ para Universitários". São Paulo: Páginas & Letras, 2006.

[2] ZAMBONI, L. C.; MONEZZI JÚNIOR, O., "Cálculo Numérico para Universitários". São Paulo: Páginas & Letras, 2002.

[3] ZAMBONI, L. C.; PAMBOUKIAN, S. V. D.; BARROS, E. A. R., "Simulação Computacional como Apoio para Outras Disciplinas" – WCCSETE: World Congress on Computer Science, Engineering and Technology Education, 2006.

[4] PAMBOUKIAN, S. V. D.; ZAMBONI, L. C.; BARROS, E. A. R., "C++ Builder para Universitários" - 2ª Edição. São Paulo: Páginas & Letras, 2003.

[5] BARROS, E. A. R.; PAMBOUKIAN, S. V. D.; ZAMBONI, L. C., "Ensinando Programação Através de Funções" – WCCSETE: World Congress on Computer Science, Engineering and Technology Education, 2006.

[6] PRESS, W. H.; TEUKOLSKY, S. A.; VETTERLING W. T.; FLANNERY, B. P., "Numerical Recipes in C++: The Art of Scientific Computing". Cambridge University Press, 2002.

[7] PRESS, W. H.; TEUKOLSKY, S. A.; VETTERLING W. T.; FLANNERY, B. P., "Numerical Recipes in C: The Art of Scientific Computing". Cambridge University Press, 1992.