**[CURSOS ONLINE:](http://www.cursos24horas.com.br/parceiro.asp?cod=promocao84273&url=cursos.asp" \t "_blank)**   [Inglês](http://www.cursos24horas.com.br/parceiro.asp?cod=promocao84273&url=cursos/curso-de-ingles-online)   [Excel](http://www.cursos24horas.com.br/parceiro.asp?cod=promocao84273&url=cursos/excel-2010)   [Fotografia Digital](http://www.cursos24horas.com.br/parceiro.asp?cod=promocao84273&url=cursos/curso-de-fotografia)   [Turismo](http://www.cursos24horas.com.br/parceiro.asp?cod=promocao84273&url=cursos/guia-de-turismo)   [Maquiagem](http://www.cursos24horas.com.br/parceiro.asp?cod=promocao84273&url=cursos/maquiagem)   [Enfermagem](http://www.cursos24horas.com.br/parceiro.asp?cod=promocao84273&url=cursos/curso-de-enfermagem)   [Informática](http://www.cursos24horas.com.br/parceiro.asp?cod=promocao84273&url=cursos/informatica)   [AutoCAD](http://www.cursos24horas.com.br/parceiro.asp?cod=promocao84273&url=cursos/autocad-3D)   [Decoração](http://www.cursos24horas.com.br/parceiro.asp?cod=promocao84273&url=cursos/decoracao)

[**Dijkstra - Algoritmo de Cálculo de Caminho Mais Curto em Grafos**](http://www.mabesi.com/artigos/redes-seguranca/internet/23-dijkstra-algoritmo-de-calculo-de-caminho-mais-curto-em-grafos.html)

Estrela ativaEstrela ativaEstrela ativaEstrela ativaEstrela ativa

Parte superior do formulário

Parte inferior do formulário

 Categoria Pai: [Redes e Segurança](http://www.mabesi.com/artigos/redes-seguranca.html)

 Categoria: [Internet](http://www.mabesi.com/artigos/redes-seguranca/internet.html)

 Publicado: Domingo, 15 Maio 2011 13:09

 Escrito por Plinio Mabesi

* [Dijkstra - Algoritmo de Cálculo de Caminho Mais Curto em Grafos](http://www.mabesi.com/artigos/redes-seguranca/internet/23-dijkstra-algoritmo-de-calculo-de-caminho-mais-curto-em-grafos.html?showall=)
* [Pag 2 - Implementando o Algoritmo](http://www.mabesi.com/artigos/redes-seguranca/internet/23-dijkstra-algoritmo-de-calculo-de-caminho-mais-curto-em-grafos.html?showall=&start=1)
* [Todas as Páginas](http://www.mabesi.com/artigos/redes-seguranca/internet/23-dijkstra-algoritmo-de-calculo-de-caminho-mais-curto-em-grafos.html?showall=1)

**INTRODUÇÃO**

O algoritmo de Dijkstra é utilizado para se calcular o caminho mais curto em um grafo. Para isso deve ser informada a matriz de custo da rede, o nó origem, a partir do qual as distâncias serão calculadas, e um valor que represente o infinito, ou seja, o custo da aresta entre dois nós que não tem ligação.

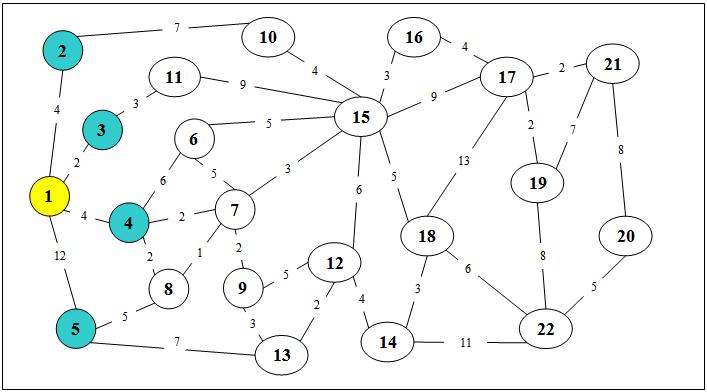
Para o cálculo de caminho mais curto em redes de computadores é utilizada uma versão ligeiramente modificada do algoritmo, na qual será mantido um conjunto com todos os nós da rede, exceto o nó origem. Conforme seja calculada a distância mínima para um nó, este será retirado do conjunto, e assim por diante.

**PROGRAMA**

Esta implementação do algoritmo oferece a possibilidade de se utilizar uma rede qualquer, com o custo de cada aresta sendo informado pelo usuário, ou verificar o funcionamento a partir do exemplo dado, descrito adiante.

Ao final da execução o resultado dos cálculos é apresentado, no qual são exibidos a distância mínima, o próximo hop e a pista para cada nó, a partir da origem.

**REDE DE EXEMPLO**

****

* **1** – Origem
* **2, 3, 4 e 5**– Nós que tem ligação com a origem
* **6 a 22**– Nós que não tem ligação com a origem
* **infinito**– 500 (Valor superior à soma de todas as arestas, convencionado, representando o valor infinito, que é a distância entre dois nós que não possuem ligação. Em uma rede sem limites poderia ser por exemplo nulo, ou algo que não pudesse ser considerado um valor válido)

**Estruturas de dados necessárias para a implementação:**

**Vetor Custo[i][j]: inteiro**

Armazena o custo da distância entre os nós i e j. Caso i e j sejam o mesmo vértice o custo será zero. Caso não haja aresta entre os dois nós o custo será infinito (infinito vale 500, conforme convencionado).

            Ex:      Custo[2][10] = custo da distância entre os nós 2 e 10 = 7;

                        Custo[15][18] = custo da distância entre os nós 15 e 18 = 5;

                        Custo[19][19] = custo da distância entre os nós 19 e 19 = 0;

                        Custo[6][17] = custo da distância entre os nós 6 e 17 = 500 (infinito);

**Vetor D[i]: inteiro**

Armazena a distância corrente da origem até o nó i.

            Ex:      D[2] = distância (ou peso) da origem até o nó 2 = 4;

                        D[5] = distância (ou peso) da origem até o nó 5 = 12;

**Vetor R[i]: inteiro**

Armazena o número do nó que é o próximo hop para o nó i, a partir da origem. Deve ser um dos nós de 1 a 4, pois são os únicos que tem ligação com a origem.

            Ex:      R[2] = próximo hop da origem para o nó 2 = 2;

                        R[3] = próximo hop da origem para o nó 3 = 3;

                        R[8] = próximo hop da origem para o nó 8 = 4;

**Conjunto S: Lista encadeada de inteiros**

Armazena cada nó exceto o nó da origem. Pode ser implementado utilizando-se ponteiros duplamente encadeados para que seja possível a retirada de um nó ou o passeio pelos nós do conjunto. O conjunto deve prover funções de inclusão, retirada e pesquisa de nós.

**Funções de S:**

* **incluir(u: inteiro)**: booleano - inclui um nó no conjunto. O número do nó a ser incluído deve ser passado como argumento para a função;
* **retirar(u: inteiro)**: booleano - retira um nó do conjunto. O número do nó a ser retirado deve ser passado como argumento para a função;
* **primeiro()**: inteiro - move o cursor para o primeiro nó do conjunto, retornando o seu número, ou zero caso não haja nenhum nó no conjunto;
* **ultimo()**: inteiro - move o cursor para o último nó do conjunto, retornando o seu número, ou zero caso não haja nenhum nó no conjunto;
* **proximo()**: inteiro - move o cursor para o próximo nó, a partir da posição atual do cursor, retornando o seu número, ou zero caso não haja mais nenhum nó no conjunto;
* **existe(u: inteiro)**: booleano – indica se um nó existe atualmente no conjunto, retornando verdadeiro em caso afirmativo, ou falso caso contrário.

**Variável C: inteiro**

Armazena temporariamente o valor da distância entre a origem e um nó qualquer, somado com o peso da aresta entre o nó e um vizinho seu. É utilizado para a comparação entre a distância encontrada atualmente e a distância armazenada no vetor D(i).

**Variável Origem: inteiro**

Armazena o número do nó de origem. Deve ser informado pelo usuário no início do programa.

**Variável MaisProximo: inteiro**

Utilizada para guardar temporariamente o número do nó que ainda esteja no conjunto S e que é atualmente o mais próximo da origem. O mais próximo é o nó que será apagado para que sejam calculadas as distâncias até os seus vizinhos.

**IMPLEMENTAÇÃO DO ALGORITMO**

A implementação do algoritmo para cálculo do caminho mais curto em uma rede está representada abaixo na linguagem simbólica de programação, mais conhecida como "Portugol".

[?](http://www.mabesi.com/artigos/redes-seguranca/internet/23-dijkstra-algoritmo-de-calculo-de-caminho-mais-curto-em-grafos.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227 | Programa Dijkstra;    /\*Conjunto S: Lista encadeada de inteiros, contendo os nós  da rede exceto o nó origem, implementado através de uma classe\*/  Tipo Conjunto {      /\*Variável que indica o nó atual, para o qual o    cursor da lista está apontando\*/    noAtual: inteiro      /\*Variável que indica a quantidade de nós atualmente    existentes no conjunto\*/    qtdNos: inteiro      /\*Função que inclui um nó no conjunto\*/    Função incluir(u: inteiro): booleano         /\*Código que implementa a função... \*/      Fim Função      /\*Função que retira um nó do conjunto\*/    Função retirar(u: inteiro): booleano        /\*Código que implementa a função... \*/      Fim Função      /\*Função que aponta para o primeiro nó e retorna seu número \*/    Função primeiro(): inteiro        /\*Código que implementa a função... \*/      Fim Função      /\*Função que aponta para o último nó e retorna seu número \*/    Função ultimo(): inteiro        /\*Código que implementa a função... \*/      Fim Função      /\*Função que aponta para o próximo na, a partir do atual,    e retorna seu número \*/    Função próximo(): inteiro        /\*Código que implementa a função... \*/      Fim Função      /\*Função que indica se um nó existe no conjunto \*/    Função existe(u: inteiro): booleano        /\*Código que implementa a função... \*/      Fim Função    }    Variáveis      /\*Contadores auxiliares para loop.\*/    i, j: inteiro;      /\*Valor que representa infinito. Soma de todas as arestas mais 1.\*/    infinito: inteiro;      /\*Número de nós da rede. Informado pelo usuário.\*/    n: inteiro;      /\*Distância desde a origem até o nó u, apagado, somada com    o peso da aresta   de u até v, o seu vizinho, cuja distância    está sendo atualmente calculada.\*/    C: inteiro;      /\*Nó de origem. Informado pelo usuário.\*/    origem: inteiro;      /\*Nó mais próximo da origem, ainda presente no conjunto S.\*/    maisProximo: inteiro;      /\*Lista encadeada do tipo Conjunto, contendo os nós    da rede, exceto o nó origem.\*/    S: Conjunto;      /\*Matriz contendo o custo entre cada par de nós.\*/    Custo[][]: vetor bidimensional de inteiros;      /\*Matriz contendo o próximo hop para cada nó.\*/    R[]: vetor unidimensional de inteiros;      /\*Matriz contendo a distância para cada nó. \*/    D[]: vetor unidimensional de inteiros;      /\*Retorna o nó que ainda se encontra em S e que possui o menor  custo desde a origem, ou zero caso não existam mais nós.\*/  Função mínimo(): inteiro ;      /\*Variável que guarda o número do nó do conjunto S,    cuja distância seja a mínima atualmente.\*/    minimoAtual: inteiro;      /\*Variável que guarda o número do próximo nó do    conjunto S, para comparação com o mínimoAtual\*/    proximoNo: inteiro;      /\*Inicializando o mínimoAtual com o primeiro    do conjunto S\*/    minimoAtual ← S.primeiro;      /\*Inicializando o proximoNo com o próximo nó    do conjunto S\*/    proximoNo ← S.proximo;      Repita        /\*Comparando o próximo e o atual\*/      Se D[proximoNo] ‹ D[minimoAtual] então          /\*Atualizando o mínimo atual\*/        minimoAtual ← proximoNo        Fim Se        /\*Buscando o próximo nó\*/      proximoNo ← S.proximo      /\*Terminando o loop quando não houver mais nós\*/    Até proximoNo = 0    Fim Função    /\*Início da execução do programa.\*/  Início      /\*Usuário informa o número de nós da rede\*/    Leia n;      /\*Usuário informa o nó de origem\*/    Leia origem;      /\*Usuário informa o valor que representa o infinito.    Deve ser maior que o maior caminho do grafo\*/    Leia infinito:      /\*Loop para inicialização da matriz de custo\*/    Para i de 1 até n        Para j de 1 até n        /\*Usuário informa o custo entre cada par de nós\*/        Leia Custo[i][j];      Fim Para      Fim Para      /\*Loop para inicialização do conjunto S\*/    Para i de 1 até n        Se i origem então        /\*Caso o nó não seja a origem será incluído em S\*/        S.incluir(i);      Fim Se      Fim Para      /\*Loop para inicialização da matriz de distâncias    a partir da origem\*/    Para i de 1 até n        /\*Distância até o nó i recebe o custo entre a      origem e i\*/      D[i] ← Custo[origem][i]      Fim Para      /\*Loop para inicialização da matriz de roteamento\*/    Para i de 1 até n        /\*Caso haja uma aresta entre origem e o nó i o próximo hop      para i recebe i. Caso contrário recebe 0\*/      Se Custo[origem][i] > 0 e Custo[origem][i] ‹ infinito então        R[i] ← i      Senão        R[i] ← 0      Fim Se      Fim Para      /\*Início do cálculo da rota mínima\*/    Repita        /\*A variável maisProximo recebe o nó que está no conjunto S,      cuja distância seja a menor\*/      maisProximo ← mínimo()        /\*Caso ainda haja nós em S, maisProximo será diferente de 0\*/      Se maisProximo > 0 então          /\*Caso o custo do maisProximo seja infinito não há mais arestas        a partir da origem até os nós restantes\*/        Se D[maisProximo] = infinito então            Msg: “Não há mais sucessores da origem em S!”;          /\*Fim do cálculo\*/          Fim;          Fim Se          /\*Agora vamos retirar do encadeamento o nó escolhido, caso ele não        seja infinito, pois se chegou até aqui é porque ainda há nós em S\*/        S.retirar(maisProximo);          Para i de 1 até n            /\*Caso exista uma aresta entre o maisProximo e algum nó, este nó é          vizinho dele então a distância será verificada. Para saber se há          uma aresta basta saber se o custo é maior que 0 e menor que infinito\*/          Se Custo[maisProximo][i] > 0 e Custo[maisProximo][i] ‹ infinito então              /\*Caso o vizinho ainda esteja em S será calculada a distância\*/            Se S.existe(i) então                /\*A variável temporária C recebe o custo até o vizinho\*/              C ← D[maisProximo] + Custo[maisProximo][i]                /\*Se for menor que o custo atual do vizinho, atualiza os dados\*/              Se C ‹ D[i] então                  /\*Próximo hop para i, o vizinho do maisProximo, recebe                o mesmo hop do maisProximo\*/                R[i] ← R[maisProximo]                  /\*Distância mínima da origem até o nó i recebe o valor de C \*/                D [i] ← C                Fim Se              Fim Se            Fim Se          Fim Para        Fim Se      Até maisProximo = 0      /\*Apresentação do Resultado\*/    /\*Título do relatório\*/    /\*Faça os ajustes de espaço para linguagem que utilizar\*/    Escreva “Nó        Distância da Origem    Próximo Hop” + [quebra de linha]    Para i de 1 até n       \*Distância e próximo hop para cada nó, a partir da origem\*/      Escreva i + “:   ” + D[i] + “              ” + R[i] + [quebra de linha]    Fim Para    Fim Programa |

Espero que tenham gostado.

Obrigado e até o próximo artigo...

{jcomments on}

**Artigos Populares**

* [Manipulação de Registros em Caixas de Listagem](http://www.mabesi.com/artigos/aplicativos-office/access/38-manipulacao-de-registros-em-caixas-de-listagem.html)
* [Sistema de Login e Senha no Access/VBA - Parte 1/5](http://www.mabesi.com/artigos/aplicativos-office/access/33-sistema-de-login-e-senha-no-accessvba-parte-1-5.html)
* [Sistema de Login e Senha no Access/VBA - Parte 2/5](http://www.mabesi.com/artigos/aplicativos-office/access/40-sistema-de-login-e-senha-no-accessvba-parte-2-5.html)
* [Conexão ODBC via VBA entre Access e MySql](http://www.mabesi.com/artigos/aplicativos-office/access/37-conexao-odbc-vba-entre-access-e-mysql.html)
* [Sistema de Login e Senha no Access/VBA - Parte 5/5](http://www.mabesi.com/artigos/aplicativos-office/access/43-sistema-de-login-e-senha-no-accessvba-parte-5-5.html)
* [Criando um Aplicativo Básico com o Android SDK da Google](http://www.mabesi.com/artigos/sistemas-operacionais/android/16-aplicativo-basico-android-sdk-google.html)
* [Sistema de Login e Senha no Access/VBA - Parte 3/5](http://www.mabesi.com/artigos/aplicativos-office/access/42-sistema-de-login-e-senha-no-accessvba-parte-3-5.html)
* [Filtros em Cascata - Caixas de Combinação e de Listagem](http://www.mabesi.com/artigos/aplicativos-office/access/62-filtros-em-cascata-caixas-de-combinacao-e-listagem.html)
* [Programação em Java para Iniciantes com BlueJ](http://www.mabesi.com/artigos/programacao-desktop/java/15-programacao-em-java-para-iniciantes-com-bluej.html)
* [Sistema de Login e Senha no Access/VBA - Parte 4/5](http://www.mabesi.com/artigos/aplicativos-office/access/41-sistema-de-login-e-senha-no-accessvba-parte-4-5.html)
* [Dijkstra - Algoritmo de Cálculo de Caminho Mais Curto em Grafos](http://www.mabesi.com/artigos/redes-seguranca/internet/23-dijkstra-algoritmo-de-calculo-de-caminho-mais-curto-em-grafos.html)
* [Comparação: Frameworks Java para Desenvolvimento WEB 2.0](http://www.mabesi.com/artigos/programacao-web/jsp/1-comparacao-frameworks-java-para-desenvolvimento-web-20.html)

**Seja social. Compartilhe!**