WillFall.NET: Um software de visualização de redes de computadores que faz uso de grafos e do algoritmo de Dijkstra

Antony Araujo Oliveira[1]

[1]Universidade Estadual de Feira de Santana (UEFS)
Av. Transnordestina, S/N, Novo Horizonte
44036-900 – Feira de Santana – BA – Brasil

antonyaraujo@protonmail.com

Resumo. *Este relatório descreve a formulação de um software que realiza implementação de grafos que representem as mais diversas redes de computadores da empresa WillFall NET. No TAD utilizado, os equipamentos são vértices e suas conexões são arestas, estes elementos são exibidos visualmente como JButtons e drawLines da biblioteca Graphics, estando associado ainda ao uso de vários componentes gráficos como JFrame, JPanel, JDialogs e outros. O sistema provê ainda diversas funcionalidades, entre elas a visualização do caminho mínimo entre dois equipamentos ou um e todos, ambos calculados através do algoritmo de Dijkstra (1959).*

1. Introdução

Visando a melhora e otimização de seus principais serviços, a WillFall Net, provedora de conexão de redes de computadores, solicitou o desenvolvimento de um software que possibilite a esquematização visual de seus equipamentos e das ligações entre estes, o que deve gerar maior controle e eficiência sobre as atividades realizadas na montagem, configuração e manutenção dos elementos componentes da sua infraestrutura. A solução para o problema solicitado pela empresa requer um produto polido que seja marcadamente gráfico, intuitivo, único e que resulte em destaque frente outros similares.

Computadores, roteadores, hubs e demais dispositivos de redes são alguns dos possíveis responsáveis pela existência de uma conexão, no entanto, a empresa adota uma lógica binária de funcionamento para encaminhamento e recebimento de pacotes de dados, onde os dispositivos que podem enviar e encaminhar são caracterizados como terminal (computador) e os que só encaminham dados (roteador) não são terminais. É preciso atentar-se ainda que cada ligação possui um peso e o roteamento obedece sempre o menor caminho possível entre dois terminais, a partir do peso de cada conexão na rota.

Para implementar os requisitos basilares supracitados foi feito uso da estrutura matemática e computacional de grafos direcionados, que permite representar os equipamentos como vértices e suas conexões como arestas. Enquanto que, para resolução do problema do caminho mais curto foi utilizada a solução do cientista da computação holandês Edsger Dijkstra, em um algoritmo que leva o seu sobrenome.

A solução deve, a partir dos critérios já mencionados, possibilitar: a criação e remoção dos equipamentos de rede (computador e roteador) e suas respectivas conexões; calcular a distância euclidiana entre dois equipamentos distintos; identificar, a partir de um nó, quais os menores caminhos possíveis para todos os outros; identificar o menor

caminho entre dois terminais; e, por fim, a importação e exportação dos arquivos de configurações, que são responsáveis por guardar a topologia de rede.

2. Fundamentação Teórica

A estruturação do sistema perpassou, *a priori*, em definir como os equipamentos seriam representados, o que é responsabilidade da estrutura de dados grafo; o menor caminho foi definido pelo algoritmo de Dijkstra, que provê uma solução sofisticada para o problema do menor caminho entre dois vértices, enquanto que, a entrada e saída de dados, principal foco da solução aqui desenvolvida, está teorizada na biblioteca de interface gráfica *Swing* utilizada no mesmo. Esses conceitos citados são teorizados nos subtópicos a seguir.

2.1. Grafos

Pode-se definir um grafo como sendo um tipo abstrato de dados (TAD), ou seja, uma estrutura de dados, que se caracteriza por conseguir representar relacionamentos que existem entre diversos tipos de objetos. Essa estrutura é composta por um conjunto **não vazio** de nós ou vértices (que seriam os objetos) e um conjunto (ou subconjunto, a depender da literatura) de arestas, que são as ligações existentes entre pares de vértices [Goodrich and Tamassia 2007].

Essas arestas podem ser dirigidas, que é quando quando há um sentido nas mesmas (*e.g.* a aresta A só permite a via do vértice a para o b, b para o a não é possível), ou podem ser não-dirigidas, que é quando não há sentido, logo, tanto faz se vai-se do vértice b para o a, ou do vértice a para o b. Um grafo que tem suas arestas dirigidas é dito grafo dirigido, dígrafo ou direcional, enquanto que um grafo que tem suas arestas não dirigidas é dito grafo não direcional ou não dirigido [de Carvalho 2005].

Para a resolução do problema em questão foi adotado um grafo direcional, pois, entende-se que este abrange tanto os casos de arestas direcionais quanto não-direcionais, tal possibilidade não propiciada por grafos não-direcionais, que, obrigatoriamente, possuirão ligação em ambas as direções ou sentidos.

Um outro conceito importante, e que foi utilizado no sistema em questão, é o de Grafo ponderado ou valorado. Segundo Carvalho (2005), um grafo é dito ponderado se um peso, ou um conjunto de pesos, é associado as suas arestas, ou seja, se cada conexão entre dois vértices possui um valor intrínseco dessa relação.

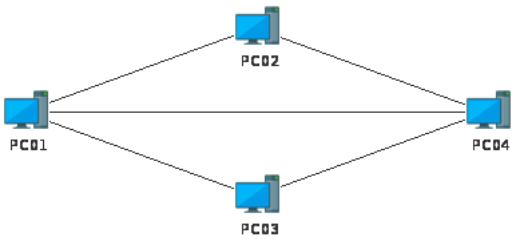


Figura 1. Exemplo de Grafo

Tendo as características básicas de grafos definidas, é preciso representá-lo de alguma forma, para tal, existem estruturas matemáticas e computacionais utilizadas, como

Tabela 1. Listas dos componentes das listas de adjacência da Figura 1.

Vértice	Vértices da Lista de Adjacência		
PC01 $->$	PC02	PC03	PC04
PC02 $->$	PC01	PC04	
PC03 $->$	PC01	PC04	
PC04 $->$	PC01	PC02	PC04

matrizes ou listas de adjacências [Lafore 2003]. Como na figura 1 a adjacência entre dois vértices é denotada se, e somente se, existe uma aresta que os ligue, caso exista essa ligação, isso pode ser representado através de uma lista, a exemplo da tabela 1.

O exemplo utilizado na tabela 1 é de lista de adjacências, pois, estas foram utilizadas na aplicação, onde, cada equipamento (PC01, PC02, PC03 e PC04), que são vértices, possuem uma lista (de adjacência, vez que esta é para as suas conexões com outros vértices) onde estão os vértices ligados ao mesmo. A constituição básica do grafo é, portanto, de uma lista de vértices, uma lista de arestas, sendo que, cada vértice possui uma lista de vértices adjacentes [Nascimento 2011].

2.2. Algoritmo de Dijkstra

Uma lista ou sequência de arestas existentes entre dois vértices, e que os conecte, é chamada de caminho. Em um grafo ponderado, o caminho mínimo é aquele cujo custo (ou seja, a soma dos pesos de todas as arestas de um dado caminho) é o menor possível [de Carvalho 2005], em grafos ponderados, este é um dos problemas mais comuns. Exibir o menor caminho entre os equipamentos (vértices) é um dos serviços a serem oferecidos, e para desenvolver tal funcionalidade, fez-se uso do algoritmo de Dijkstra.

O algoritmo de Dijkstra foi criado em 1956 e publicado em 1959, pelo notável ganhador do prêmio Turing e cientista da computação, o holandês, Edsger W. Dijkstra, no artigo intitulado *"A Note on Two Problems in Connexion with Graphs"* (do inglês: Uma nota sobre dois problemas em conexões com grafos [Dijkstra 1959]), onde o mesmo descreve, em linguagem natural, os elementos necessários para a criação de um algoritmo que encontre o menor caminho entre dois vértices.

Na solução apresentada por Dijkstra, deve-se primeiro definir a distância do vértice inicial como zero, vez que, a distância da origem até si mesma é zero, e depois define-se que a distância para todos os outros vértices é infinita. Para abranger tal conceito, de infinito, foi utilizado o maior valor constante de tipo double (`Double.MAX_VALUE`). Então, cria-se um laço, que deve percorrer todos os vértices, e nesse laço é verificado, entre o conjunto de vértices v, qual das estimativas de valor (A distância atual + a distância de um dado vértice v) é a menor.

Para todo o vértice do conjunto j de vértices, que são os antecessores ao vértice escolhido do conjunto v, é somado o valor do peso da aresta que possui ambos, caso esse valor for menor que a soma de outro conjunto anterior, já calculado, então, substitui-se pelo novo menor valor, este processo descrito é, comumente, chamado de relaxamento das arestas, e está exemplificado na figura 2. No software desenvolvido, o conjunto j é representado como uma fila de prioridade (`PriorityQueue`, da biblioteca `Collections`), por esta manter a ordem dos vértices no caminho provável a ser o menor caminho entre

dois vértices distintos [de Carvalho 2003].

```
double peso = aresta.getPeso();
double minDistance = vertex.getDistanciaMinima() + peso;
if (minDistance < v.getDistanciaMinima()) {
    filaPrioridade.remove(vertex);
    v.setAnterior(vertex);
    v.setDistanciaMinima(minDistance);
    filaPrioridade.add(v);
}
```

Figura 2. Código que exemplifica o relaxamento de aresta dentro de um loop

2.3. Interface Gráfica

Para propiciar uma melhor comunicação com os usuários, e um entendimento mais imediato e intuitivo do funcionamento de um programa, bem como, uma melhora da interação humano-computacional (IHC) é que foram desenvolvidas as interfaces gráficas do usuário (do inglês, *Graphical User Interface* - GUI). A visualização gráfica propicia um melhor entendimento das ações realizadas no programa e é formada por componentes, contêineres, janelas gráficas e periféricos, como mouse e teclado [Bernardini 2012].

Em POO (Programação Orientada a Objetos), os componentes visuais são objetos, a linguagem Java possui suas bibliotecas padrões de componentes visuais, que são a AWT (*Absract Window Toolkit*) e a Swing. A biblioteca AWT, primeiro conjunto padrão da linguagem, é mais pesada porque abriga sistemas de janelas de diferentes plataformas, contrastando com o Swing que possui uma única forma para lidar com os componentes visuais em sistemas operacionais distintos, bem como, tem uma maior variedade de componentes, por essas razões, o Swing é a biblioteca gráfica mais utilizada [Deitel and Deitel 2017].

A base de construção de uma janela é o `JFrame`, um contêiner (componente) de alto nível, todo programa visual que usa Swing deve ter pelo menos um contêiner de alto nível, pois são eles os responsáveis por ocupar um espaço, de facto, na tela e por abrigarem todos os outros componentes, permitindo, através disso, o desenho desses na tela. Os componentes são adicionados ao *frame* através do método `add()`, e.g.: "`getContentPane().add(painel);`", onde getContentPane() retorna o painel padrão de um dado JFrame, e painel é um JPanel que está sendo adicionado ao frame.

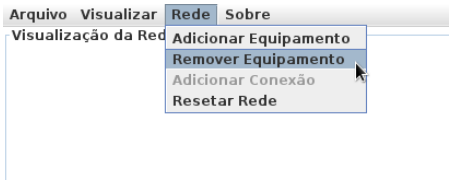


Figura 3. Exemplo de um menu superior, devidamente implementado

Na composição de uma tela também podem existir componentes como uma barra de menu (`JMenuBar`), a exemplo do que consta na figura 3 que possuí diversos itens (`JMenuItem`), sendo que, tais itens, são capazes de evocar ações (em Java, eventos).

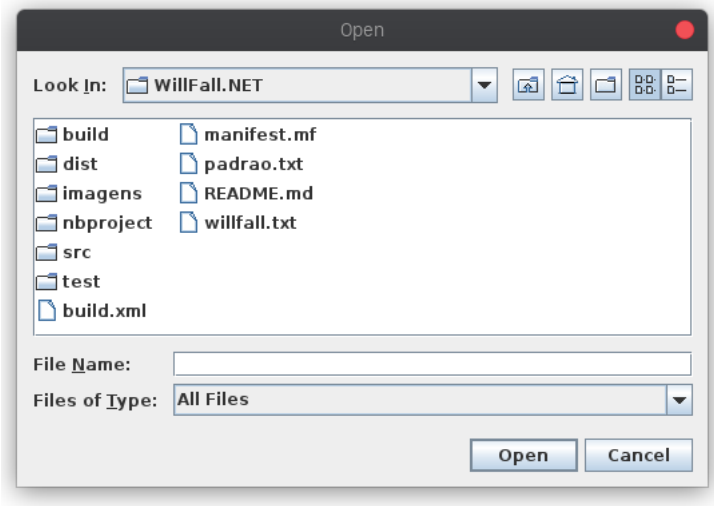


Figura 4. Exemplo de um JFileChooser

A biblioteca swing também possibilita, através da classe `JFileChooser`, uma alternativa visual e mais interativa para trabalhar com arquivos. A forma para salvamento e carregamento é a mesma, mas agora, ao invés de digitar o caminho, o usuário pode simplesmente escolhê-lo, através do seu mouse, conforme a figura 4. O arquivo (do tipo `File`) escolhido, no carregamento, pode ser extraído através do método `getSelectedFile()` que todos os objetos do tipo `JFileChooser` possuem, ou para o salvamento, pode-se ainda, aliado a este método, utilizar o `getPath()`, que retorna, juntamente ao caminho do sistema operacional, o nome do dado arquivo.

3. Metodologia

A referida aplicação foi desenvolvida a partir da linguagem de programação imperativa e orientada a objetos Java, através de colaboração em dupla, com uso da plataforma *on-line* Github e da IDE Apache NetBeans 11.2, versionando na plataforma *OpenJDK* versão 11.0.6 em um sistema Debian GNU/Linux 10 (Buster) x64 e a plataforma *JDK* 13.0 em um sistema Windows 10 x64.

A elaboração da solução aqui apresentada perpassou, a *priori*, por implementar o grafo e o algoritmo do menor caminho de Dijkstra, vez que, estes são essenciais para o funcionamento de todo o sistema. O grafo, em si, é estabelecido através das seguintes classes: Grafo, Vertice e Aresta. A classe Aresta contém dois objetos do tipo Vertice, um para indicar origem e outro destino, e uma variável peso, do tipo `double`.

O Vertice, cerne da teoria dos grafos, possui um nome (*String*) que o identifica, um booleano, para identificar se este é terminal ou não (terminais podem enviar e encaminhar dados, enquanto que não-terminais não o podem), um `double` com a distância mínima (que é o valor calculado no relaxamento de arestas até aquele vértice), duas variáveis inteiras (x e y) que são suas coordenadas na tela e, por fim, uma `ArrayList` de Arestas que guarda todos seus vértices adjacentes, representando o que seria uma lista de adjacência com arestas.

Tendo a Aresta e Vértices definidos, a constituição da classe Grafo tornou-se mais simples, pois, esta contém, basicamente, uma `ArrayList` de objetos do tipo Vertice e

outra do tipo Aresta. No entanto, dentro do grafo, ainda fora implementado os métodos para funcionamento do algoritmo de Dijkstra, onde há um primeiro método, responsável por calcular os menores caminhos de um dado vértice para todos os outros, conforme descrito no tópico anterior.

```
public List<Vertice> getCaminhoMaisCurtoPara(Vertice destino) {
    List<Vertice> path = new ArrayList<>();
    for (Vertice vertex = destino; vertex != null; vertex = vertex.getAnterior()) {
        path.add(vertex);
    }
    for (Vertice v : vertices) {
        v.reset();
    }
    Collections.reverse(path);
    return path;
}
```

Figura 5. Método que gera uma lista com o menor caminho para um dado vértice

Ainda, para gerar o menor caminho, há outro método, conforme figura 5, que gera o caminho, propriamente dito, em uma *list*, do vértice cujos caminhos foram calculados no método anterior, para o vértice de destino especifcado como parâmetro. Como o Dijkstra trabalha com a ideia de anteriores, ao percorrê-los tem-se uma lista onde o começo (posição 0 na lista) é o destino e o fim é a origem, para resolver tal questão fez-se uso do método `reverse()` da biblioteca `Collections` para inverter o caminho, partindo da origem para o destino, de forma sequencial e não inversa.

A posteriori da implementação completa do grafo, preocupou-se com a representação visual dos vértices e arestas na interface. Onde utilizou-se JButtons para representar os equipamentos (vértices) e para as arestas fez-se uso de linhas desenhadas pela biblioteca `Graphics`, através do método `drawLine()`, onde as coordenadas passadas em sua construção foram as dos vértices que são origem e destino da mesma.

4. Resultados e Discussões

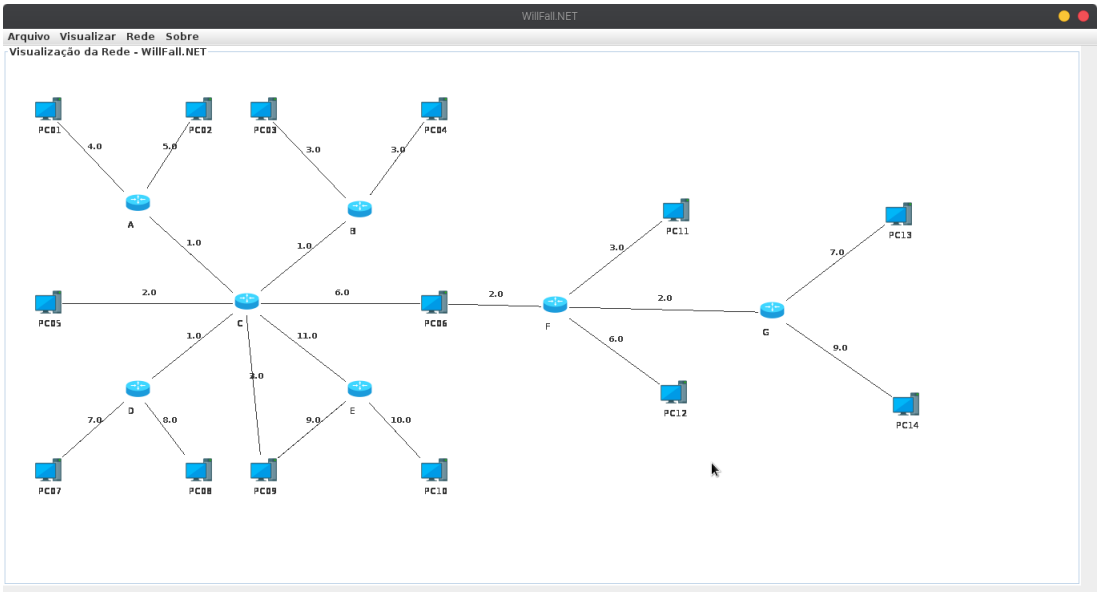


Figura 6. Tela da aplicação desenvolvida com uma rede desenhada.

Como disposto na figura 6, a tela da aplicação tem uma apresentação simples, na qual, a maioria das operações podem ser realizadas através de um menu superior composto pelos seguintes itens de menu:

1. Arquivo – Ofereces opções para o carregamento e salvamento de uma topologia de rede (contendo os equipamentos e suas conexões), em um arquivo;
2. Visualizar – Oferece opções, a partir de um grafo com 30 vértices ou mais, visualizar itens de menor rota entre dois terminais, a distância euclidiana entre dois equipamentos ou a distância de um equipamento para todos os outros, contando ainda com a possibilidade de exibir ou não os pesos das arestas;
3. Rede – Oferece as operações de controle do grafo (rede), onde estão as opções de adicionar e remover equipamentos, adicionar conexão ou de excluir todos os elementos da rede.
4. Sobre – Mostra informações detalhadas da aplicação;

A opção de identificar o menor caminho entre dois terminais, quando escolhida pelo menu superior, pede ao usuário que informe o nome de dois terminais, a partir de caixas de diálogo (`JOptionPane.showInputDialog()`), no entanto, a opção também pode ser ativada ao clicar com o botão direito sobre algum terminal e escolhendo a opção de menor caminho, a partir disso, é pedido que se escolha o destino, clicando em algum outro terminal. Tendo a origem e destino escolhidos, as arestas que pertencem ao caminho mínimo são destacadas com seu tamanho aumentado e tem as cores trocadas para vermelho, como mostra a figura 7.

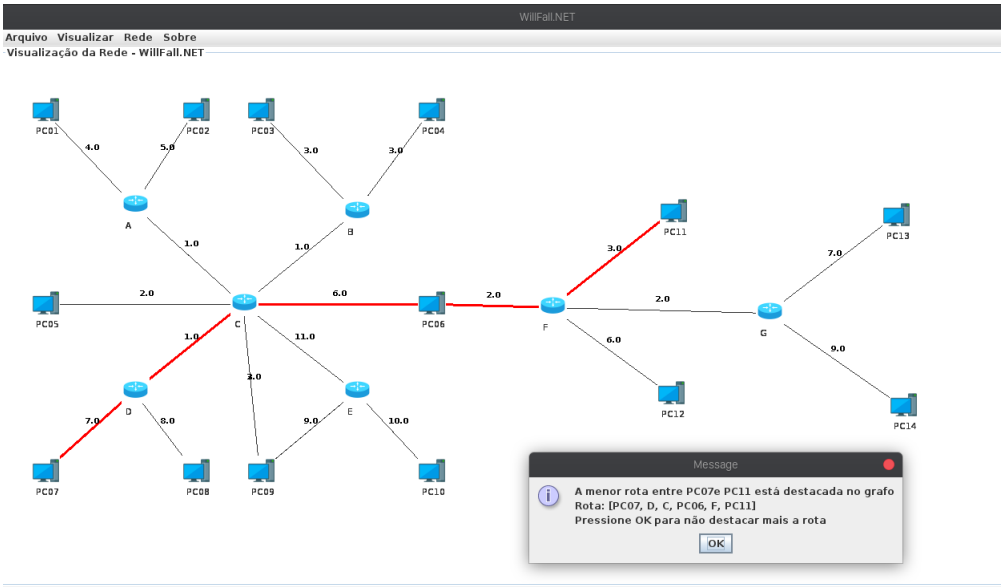
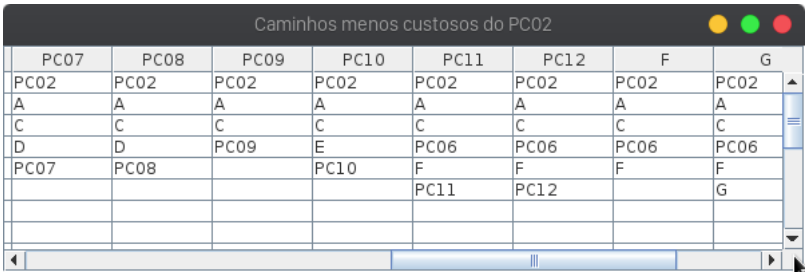


Figura 7. Identificação do caminho mínimo entre os equipamentos PC07 e PC11

A identificação dos melhores caminho de um terminal para todos os outros também pode ser realizado de duas formas, informando o nome do vértice desejado, ou clicando na opção exposta no botão direito do terminal desejado. A partir da escolha, é exibida uma tabela, conforme a figura 8, que exibe os caminhos mínimos do vértice escolhido para todos os outros, organizando-os em colunas, onde o primeiro elemento da linha é a origem e o último elemento da linha é o destino.



PC07	PC08	PC09	PC10	PC11	PC12	F	G
PC02	PC02	PC02	PC02	PC02	PC02	PC02	PC02
A	A	A	A	A	A	A	A
C	C	C	C	C	C	C	C
D	D	PC09	E	PC06	PC06	PC06	PC06
PC07	PC08		PC10	F	F	F	F
				PC11	PC12		G

Figura 8. Tabela com caminhos mínimos do PC02 para todos os outros vértices

Para saber qual a distância euclidiana entre dois equipamentos é necessário ir até o menu, selecionar Visualizar, depois disso, ir em Distância Euclidiana, duas caixas de diálogo serão apresentadas, solicitando o vértice A e B, após isso, é exibido, em um `JOptionPane.showMessageDialog()`, o valor da distância euclidiana, conforme a fórmula: $\sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$, onde x e y são as variáveis inteiras que representam a posição do dado vértice no respectivo eixo, em pixels.

A topologia – forma como os equipamentos e suas conexões estão dispostas na tela e no grafo – deve ser salva e carregada a partir de um arquivo único de texto (cuja extensão é `txt`) e seguindo, estritamente, a ordem `ROTULO;TERMINAL;COORDENADA_X;COORDENADA_Y` para os vértices (equipamentos), e a ordem `ORIGEM;DESTINO;PESO`, para as arestas (conexões), onde rotulo, origem e destino devem ser do tipo textual (*String*), terminal deve ser especificado com "SIM"ou "NAO"(já que será transformado posteriormente para booleano), peso com decimal e as coordenadas como inteiros. A figura 9 mostra um exemplo de arquivo que obedece esta configuração.

```
1  EQUIPAMENTOS
2  ROTULO;TERMINAL;COORDENADA_X;COORDENADA_Y
3  PC01;SIM;239;235
4  PC04;SIM;585;235
5  PC02;SIM;412;172
6  PC03;SIM;412;298
7  CONEXOES
8  ORIGEM;DESTINO;PESO
9  PC01;PC02;1.0
10 PC02;PC01;2.0
11 PC02;PC04;3.0
12 PC04;PC02;3.0
13 PC04;PC03;2.0
14 PC03;PC04;1.0
15 PC03;PC01;4.0
16 PC01;PC03;1.0
```

Figura 9. Exemplo do conteúdo de um arquivo contendo uma topologia de rede.

5. Conclusão

O presente artigo expõe uma solução para o desenvolvimento de um sistema que permite a manipulação visual de um grafo que representa uma rede, e possibilita diversas operações, a partir do mesmo. A aplicação possui uma interface gráfica extremamente visual, fazendo leitura e escrita de arquivos na memória secundária da máquina, para organizar a topologia da rede.

O resultado apresentado atende a resolução do problema, e o faz através do uso de: estruturas de repetição (`while` e `for`), seleção (`if`), Fila de Espera

(PriorityQueue), ArrayLists, Generics, JFrames, JPanels, JTable, JFileChooser, JOptionPanes e outros. O projeto faz uso do modelo MVC (Modelo, Visão e Controle), onde o Grafo, Vertice e Aresta encontram-se no Modelo, uma classe Sistema coordena a comunicação entre estes e as classes de Interface da visão.

Melhorias são possíveis neste software através de: uma maior possibilidade de manipulação dos componentes visuais, permitindo, por exemplo, que o usuário arraste os equipamentos como bem entender; pode-se acrescer um banco de dados em rede (*on-line*), o que evitaria eventual perca de dados em uma máquina, considerando que seja redundante; pode-se cogitar ainda a implementação de uma hierarquia de acesso às operações realizadas, o que promoveria maior segurança da informação das topologias da empresa.

Os pontos fortes do sistema se dão: na modularização realizada, promovida pelo modelo MVC; na forte utilização de elementos visuais; Os pontos fracos estão: na falta de verificações e tratamentos de erros de forma mais específica e na falta de um diretório destinado somente a alocação dos arquivos de saída do sistema.

Os pontos fortes deste relatório se dão na teorização dos aspectos essenciais para a entrega do produto requisitado, nos exemplos visuais e suas descrições, bem como, na sua organização de tópicos. Os pontos fracos estão na pouca abordagem da estruturação de eventos (poderia ter sido mais amplamente abordado) e na repetição de termos e conceitos já previamente abordados.

Referências

Bernardini, F. C. (2012). Interfaces gráficas (guis) em java usando swing. Disponível em: <https://www.lncc.br/~rogerio/poo/04a\%20-\%20Programacao_GUI.pdf>. Acessado em 24 de Mar de 2020.

de Carvalho, B. M. P. S. (2003). Algoritmo de dijkstra. Disponível em: <https://student.dei.uc.pt/~brunomig/cp/Artigo.pdf>. Acessado em 24 de Mar de 2020.

de Carvalho, M. A. G. (2005). Teoria dos grafos – uma introdução. Disponível em: <https://www.ft.unicamp.br/~magic/ft024/apografos_ceset_magic.pdf>. Acessado em 24 de Mar de 2020.

Deitel, P. and Deitel, H. (2017). *Java: Como programar*. Pearson Education do Brasil, 10th edition. tradução Edson Furmankiewicz.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematlk*, 1:269 – 271. Acessado em 24 de Mar de 2020.

em Grafos (Blog), P. (2014). Dijkstra. Disponível em: <https://passeiemgrafos.wordpress.com/2014/11/21/dijkstra/>. Acessado em 24 de Mar de 2020.

Feofiloff, P., Kohayakawa, Y., and Wakabayashi, Y. (2004). Uma introdução sucinta a teoria dos grafos. *II Bienal*. Acessado em 24 de Mar de 2020.

Goodrich, M. T. and Tamassia, R. (2007). *Estruturas de Dados e Algoritmos em Java*. Bookman, 4th edition. tradução Bernardo Copstein, Leandero Bento Pompermeier.

Lafore, R. (2003). *Data Structures and Algorithms in Java*. Sams Publishing, 2nd edition.

Li, Y. (2018). Dijkstra's algorithm. Disponível em: <http://palantir.swarthmore.edu/cs/courses/S18/cs231/notes/outlines28.pdf>. Acessado em 24 de Mar de 2020.

Nascimento, E. S. (2011). Representações de grafos. Disponível em: <https://erinaldosn.files.wordpress.com/2011/02/aula-2-representac3a7c3a3o.pdf>. Acessado em 24 de Mar de 2020.