FURB GRAPHS: UMA FERRAMENTA DE APOIO AO APRENDIZADO PARA A DISCIPLINA DE TEORIA DOS GRAFOS

Gustavo Bittencourt, Aurélio Faustino Hoppe – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

guto_bitta@hotmail.com, aureliof@furb.br

Resumo: *Este trabalho descreve a reestruturação do FURB Graphs, desenvolvido por Bernardes (2016), tendo como objetivo torná-la uma ferramenta de apoio ao ensino da disciplina de Teoria dos Grafos. A implementação consiste em realizar a troca de linguagem de programação, mantendo as funcionalidades do protótipo atual, como propriedades do grafo e acompanhamento da execução de algoritmos passo a passo. Após revisão do estado da arte, observou-se que as ferramentas atuais para auxílio no ensino de grafos não estão completas, tornando esta área de pesquisa relevante. Os resultados obtidos a partir de testes realizados com usuários apontam que a ferramenta desenvolvida favorece o entendimento da área, através de manipulação de grafos, acompanhamento passo a passo e coloração das execuções dos algoritmos.*

Palavras-chave*: Buscas. Dijkstra. Árvores geradoras. Teoria dos grafos. Ferramenta de ensino.*

1 INTRODUÇÃO

Atualmente, conforme Ferreira e Lozano (2009, p. 2), "a sociedade contemporânea está sujeita a rápidas transformações, isso ocorre, principalmente, devido à velocidade da informação promovida pelo avanço tecnológico". Assim, considerando o mundo globalizado em que vive-se, as escolas são um dos diversos pontos afetados pelas mudanças nas formas de pensamentos (FERREIRA; LOZANO, 2009).

Segundo Moreira e Kramer (2007, p. 2), "a globalização tem afetado o modo de estruturar a educação escolar e de desenvolver o trabalho docente", notando-se a presença de diferentes recursos tecnológicos em salas de aula. Ainda segundo Moreira e Kramer (2007, p. 2), "é como se os objetos técnicos pudessem, por um passe de mágica, garantir qualidade na educação". Em contrapartida, Farinha (2005) garante que a utilização de tecnologias no meio educacional é inevitável e que já está ocorrendo no Brasil e no mundo. Além disso, ela aponta que o ambiente de ensino deve estar preparado e conscientizado para o uso de tecnologias digitais, pois elas são mais do que uma importante ferramenta para o processo de ensino, elas são indispensáveis para o profissional da educação.

As várias tecnologias, como softwares educativos e laboratórios informatizados, estão cada vez mais presentes nas instituições de ensino superior, proporcionando novas perspectivas para o desenvolvimento do currículo escolar (SILVA, 2010). A utilização de tecnologias no ensino superior, segundo Caldas (2012, p. 12), "[...] denotou uma forte ferramenta como fonte de auxílio, tanto dos professores como dos alunos, no que tange a representação do conhecimento". Assim, no curso de Ciência da Computação, devido ao contexto em que está presente, são utilizadas diversas tecnologias para auxiliar no ensino acadêmico. Porém, algumas das disciplinas, como Teoria dos Grafos, carecem de tecnologias ou as que existem não são completas, tornando o processo de aprendizagem mais lento (SILVEIRA, 2007).

Segundo Silveira (2007), a grande quantidade de conteúdo teórico que é apresentado na disciplina de Teoria dos Grafos requer um nível elevado de abstração por parte dos alunos. O autor ainda ressalta que a quantidade de elementos que compõem um grafo é diretamente proporcional à dificuldade de representá-lo, visualizá-lo e, principalmente, analisá-lo. Para auxiliar no ensino de teoria dos grafos, nos últimos anos foram desenvolvidas algumas ferramentas tais como o TGrafos (SILVEIRA, 2007), o Editor Visual de Grafos (BRAUN, 2009) e o FURB Graphs (BERNARDES, 2016). Contudo, as aplicações carecem de diversas funcionalidades como materiais de apoio ao estudo, apresentação das matrizes de adjacência e custos, atalhos para navegações e melhorias na criação dos grafos.

Diante do exposto, este trabalho apresenta a reestruturação da ferramenta FURB Graphs, desenvolvida por Bernardes (2016), permitindo que ela possa ser utilizada como facilitadora do aprendizado do conteúdo da disciplina de Teoria dos Grafos. Para alcançar o objetivo proposto, os seguintes objetivos específicos foram realizados: disponibilização de uma interface para manipulação dos grafos, através da criação de vértices e arestas e a visualização das matrizes de adjacência e custo; disponibilização de um mecanismo de acompanhamento passo a passo da execução dos algoritmos de busca, caminhamento mínimo e árvore geradora.

Este artigo está estruturado em cinco capítulos. A fundamentação teórica necessária para a compreensão deste trabalho, como conceitos e técnicas, encontra-se no segundo capítulo. No terceiro capítulo é descrita a estruturação da

ferramenta através de diagramas, especificações e detalhamentos da implementação. Os resultados obtidos a partir dos testes realizados estão no quarto capítulo. Por fim, no quinto capítulo, são apresentadas as conclusões, limitações e extensões do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está organizado em três seções. A seção 2.1 aborda os conceitos de grafos e os seus principais algoritmos de busca, caminhamento mínimo e árvore geradora mínima. Na seção 2.2 são descritos os estudos realizados por Bernardes (2016). Por fim, na seção 2.3 são apresentados os trabalhos correlatos.

2.1 TEORIA DOS GRAFOS

Segundo Melo (2014), a teoria dos grafos tem sua origem no século XVIII, que para a história da matemática, é considerada uma origem recente. O autor ainda comenta que no século XX, a teoria dos grafos já estava desenvolvida, apresentando a sua importância devido as ligações e aplicações com outras ciências. Melo (2014) complementa que mesmo o termo grafo sendo utilizado pela primeira vez por J. J. Sylvester em 1877, a resolução de Euler do Problema das Pontes de Königsberg, publicada em 1736, é referida como primeira publicação da área.

A teoria dos grafos, conforme Feofiloffm, Kohayakama e Wakabayashi (2011, p. 5) "[...] estuda objetos combinatórios – grafos – que são um bom modelo para muitos problemas para vários ramos da matemática, da informática, da engenharia e da indústria". Rabuske (1992) comenta que a teoria dos grafos está relacionada com muitos ramos da matemática, como teoria dos grupos, matrizes, probabilidade, análise numérica, topologia e combinatória.

Os algoritmos em teoria dos grafos têm como objetivo a resolução de problemas que podem ser estruturados utilizando um grafo. Dentre os algoritmos mais comuns estão o Depth-First Search (DFS) *e* Breadth-First Search *(BFS*) (busca), Dijkstra (caminhamento mínimo) e Prim e Kruskal (árvore geradora mínima).

Segundo Carvalho (2018, p. 4), "a busca em grafos (ou percurso em grafos) é a examinação de vértices e arestas de um grafo". O autor ainda comenta que os algoritmos de busca utilizam da técnica de marcar os vértices como não explorados e explorados durante as suas execuções. O algoritmo de busca em profundidade (DFS) tem como característica básica, segundo Pantuza (2017a, p. 1), "percorrer todos os nós filhos ao nó raiz o mais profundo possível para somente depois retroceder". Enquanto que o algoritmo de busca em largura (BFS) tem como característica "que a busca sempre ocorre nos filhos ou nós próximos ao nó pelo qual a busca foi iniciada" (PANTUZA, 2017b, p. 1).

As implementações dos algoritmos de DFS e BFS utilizam da coloração dos vértices para identificar quais já foram visitados e a marcação dos tempos de abertura e fechamento. A principal diferença está na forma de determinar qual a é sequência de vértices a serem processados. O algoritmo de DFS utiliza o conceito de pilha, processando o vértice do topo enquanto houver elementos. O processamento consiste em atribuir seu tempo de abertura e adicionar na pilha um vértice adjacente ainda não visitado. Caso todos os vértices adjacentes tenham sido visitados, é atribuído seu tempo de fechamento e removido da pilha. Já o algoritmo de BFS utiliza o conceito de fila, processando o primeiro vértice enquanto houver elementos. O processamento consiste em adicionar ao final da fila todos os vértices adjacentes ainda não visitados, atribuindo para cada um seu tempo de abertura. O vértice em processamento recebe seu tempo de fechamento e é removido da fila. O Quadro 1 apresenta os pseudocódigos para os algoritmos DFS e BFS.

Quadro 1 – Pseudocódigos para os algoritmos de DFS e BFS

```
01. BuscaEmProfundidade(G)
02.     for each u ∈ V[G]
03.         c[u] ← white
04.         π[u] ← NULL
05.     time ← 0
06.     for each u ∈ V[G]
07.         if c[u] = white
08.             visita(u)
09. visita(u)
10.     c[u] ← gray
11.     d[u] ← time ← time + 1
12.     for each v ∈ Adj[u]
13.         if c[v] = white
14.             π[v] ← u
15.             visita(v)
16.     c[u] ← black
17.     f[u] ← time ← time + 1
```

```
01. BuscaEmLargura(G, s)
02.     for each u ∈ V[G]
03.         c[u] ← white
04.         d[u] ← ∞
05.         π[u] ← NULL
06.     c[s] ← gray
07.     d[s] ← 0
08.     Q ← {s} //Queue
09.     while Q ≠ ∅
10.         u ← head[Q]
11.         for each v ∈ Adj[u]
12.             if c[v] = white
13.                 c[v] ← gray
14.                 d[v] ← d[u] + 1
15.                 π[v] ← u
16.                 enqueue(Q,v)
17.         dequeue(Q)
18.         c[u] ← black
```

Fonte: adaptado de Lima (2015).

O algoritmo de Dijkstra foi publicado em 1959 por Edsger Dijkstra e, segundo Méndez e Guardia (2008, p. 7), "encontra o menor caminho de um nó fonte ou origem até os outros nós de uma rede orientada para o caso em que todos os pesos dos arcos sejam não negativos". A execução do algoritmo parte do princípio de que através de um vértice inicial, seja encontrado o menor caminho para os demais vértices. Para isso, é utilizado um vetor com predecessores e a rotulação dos vértices com distâncias permanentes e temporárias. Os vértices com distâncias permanentes significam que já foram processados e possuem o menor caminho em relação ao vértice inicial, enquanto que os vértices com distâncias temporárias ainda precisam ser processados. O próximo vértice a ser processado é determinado buscando o vértice pendente que possui a menor distância temporária em relação ao vértice inicial. Após escolher o vértice, a sua distância passa de temporária para permanente e é aplicada uma função de relaxamento para os vértices adjacentes com distâncias temporárias. A função de relaxamento consiste em alterar a distância e o predecessor do vértice adjacente, caso a distância temporária do vértice adjacente seja maior que a distância do vértice em processamento somada ao custo da aresta que os relaciona. O Quadro 2 apresenta o pseudocódigo do algoritmo de Dijkstra.

Quadro 2 – Pseudocódigo para o algoritmo de Dijkstra

```
01. Relax(u, v, w)
02.     if v.d > u.d + w(u, v)
03.        v.d = u.d + w(u, v)
04.        v.π = u
05. Initialize-Single-Source(G, s)
06.     for each v ∈ G.V
07.         v.d = ∞
08.         v.π = nil
09.     s.d = 0
10. Dijkstra(G, w, s)
11.     Initialize-Single-Source(G, s)
12.     S = ∅
13.     Q = G.V
14.     while Q != ∅
15.         u = Extract-Min(Q)
16.         S = S ∪ {u}
17.         for each v ∈ G.Adj[u]
18.             Relax(u, v, w)
```

Fonte: adaptado de Lobo (2017).

Segundo Rocha (2007, p. 1), uma árvore é considerada geradora se ela interliga todos os vértices do grafo. O autor ainda comenta que elas "são as menores estruturas que conectam todos os nós do grafo". Algoritmos como DFS e BFS podem auxiliar na obtenção de uma árvore geradora. Porém, em determinados casos, pode ser necessário que a árvore geradora seja mínima. De acordo com Feofiloff (2017c, p. 1), "uma árvore geradora T de G é mínima se nenhuma outra árvore geradora tem custo estritamente menor que o de T". Os algoritmos mais comuns para obter uma árvore geradora mínima são Prim e Kruskal.

O algoritmo de Prim têm como característica crescer uma subárvore até que ela se torne geradora, satisfazendo o critério de minimalidade baseado em cortes (FEOFILOFF, 2017a, p. 1). Esta subárvore, inicialmente vazia, é preenchida com um vértice por vez, até que tenha todos os vértices do grafo inicial. A escolha do vértice a ser adicionado tem como critério escolher o menor custo entre um vértice pendente adjacente a um vértice já adicionado. Caso seja a escolha do primeiro vértice, o algoritmo permite selecionar qualquer vértice como inicial. Um pseudocódigo para o algoritmo de Prim pode ser visualizado no Quadro 3.

Quadro 3 – Pseudocódigo para o algoritmo de Prim

```
01. Prim(G, o)
02.     Para cada vértice v
03.        custo[v] = infinito
04.     Define conjunto S = 0
05.     custo[o] = 0
06.     Enquanto S != V
07.        Selecione u em V-S, tal que custo[u] é mínimo
08.        Adicione u em S
09.        Para cada vizinho v de u faça
10.           Se custo[v] > w((u,v)) então
11.              custo[v] = w((u,v))
```

Fonte: adaptado de Quitzau (2003).

Conforme Feofiloff (2017b, p. 1), o algoritmo de Kruskal têm como característica crescer uma floresta geradora até que ela se torne conexa, satisfazendo o critério de minimalidade baseado em circuitos. Inicialmente, as arestas são ordenadas de forma crescente utilizando o seu custo e são criadas florestas contendo cada vértice do grafo inicial. Em seguida, percorre-se a lista ordenada de arestas, realizando um processamento em cada uma, até que todas as

arestas tenham sido processadas. O processamento consiste em verificar se o vértice inicial e final estão em florestas distintas e caso estejam, estas florestas são agrupadas. O pseudocódigo para o algoritmo de Kruskal pode ser observado no Quadro 4.

Quadro 4 – Pseudocódigo para o algoritmo de Kruskal

```
01. KRUSKAL(G,w)
02.     para cada vértice v em V[G]
03.         criarFloresta(v)
04.     Ordenar E em ordem crescente de peso (w)
05.     para cada aresta (u,v) em E
06.         se floresta(u) != floresta(v)
07.             UNION(floresta(u), floresta(v))
```

Fonte: adaptado de Quitzau (2003).

Os algoritmos apresentados objetivam solucionar os mais diversos problemas, cada um conforme suas características de implementação. Os algoritmos de busca visam encontrar um determinado elemento contido em um conjunto, como verificar se um valor está presente em uma árvore de valores. Já o algoritmo de Dijkstra encontra o menor caminho a partir de um vértice inicial em relação aos demais vértices do grafo. Assim, este algoritmo é facilmente encontrado em aplicações que envolvem mapas. Por fim, os algoritmos de árvore geradora mínima consistem em gerar um novo grafo que possui um custo total mínimo, sendo encontrados em redes de transmissão de energia.

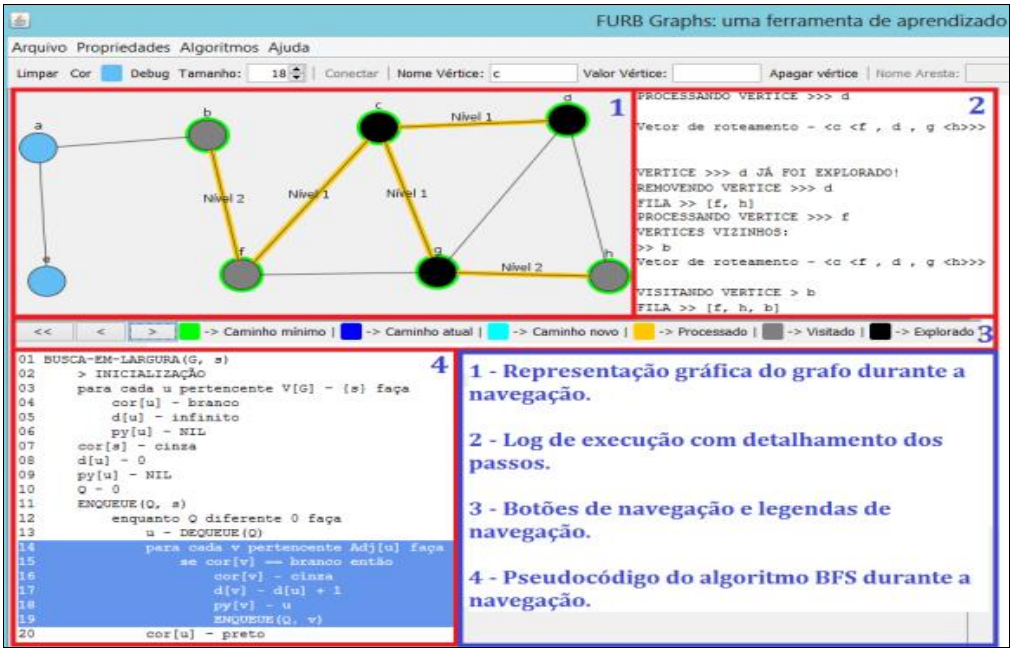
2.2 VERSÃO ANTERIOR DA FERRAMENTA

O protótipo FURB Graphs desenvolvido por Bernardes (2016) teve por objetivo adaptar o trabalho desenvolvido por Borba (2014), de forma a obter uma ferramenta de apoio ao ensino de Teoria dos Grafos. O foco do desenvolvimento foi alterar as rotinas criadas por Borba (2014) melhorando a visualização e acompanhamento dos algoritmos de BFS, DFS e Dijkstra. Para conseguir o objetivo proposto, Bernardes (2016) adicionou novas funcionalidades e opções visuais, sendo elas:

a) cores para representação da navegação: utiliza cores para indicar o que acontece no grafo ao realizar um determinado passo (Figura 1 item 1);
b) log de execução: apresenta o detalhamento do que foi executado (atributos, estados e vetores) (Figura 1 item 2);
c) acompanhamento da execução do algoritmo na forma de passo a passo: através de botões de navegação, o usuário pode avançar ou retroceder cada passo da execução do algoritmo selecionado (Figura 1 item 3);
d) pseudocódigo do algoritmo: apresenta um pseudocódigo do algoritmo em execução, destacando a linha referente ao passo atual (Figura 1 item 4).

Para habilitar/desabilitar as funcionalidades mencionadas acima, foi disponibilizado o botão `Debug` na barra superior. Esta opção só está disponível para os algoritmos de BFS, DFS e Dijkstra.

Figura 1 – Mosaico da representação gráfica durante a navegação



Fonte: Bernardes (2016, p. 48).

O protótipo foi desenvolvido utilizando o ambiente de desenvolvimento Eclipse Mars com a linguagem de programação Java 7. Para a criação de telas e menus, foi utilizada a biblioteca gráfica Swing que é nativa do JDK Java. Para o campo do pseudocódigo foi utilizado Hypertext Markup Language (HTML). Utilizou-se do JavaScript Object Notation (JSON) para serialização e leitura dos grafos, com auxílio da biblioteca Gson, desenvolvida pela Google.

Bernardes (2016) aponta que os resultados foram satisfatórios e que a ferramenta atingiu o objetivo de fazer com que o usuário compreenda o funcionamento dos algoritmos de BFS, DFS e Dijkstra, de tal forma que ele consiga relacioná-los com possíveis aplicações reais. Ele destaca que os recursos para avançar e retroceder os passos dos algoritmos, quando necessário, permite que o usuário tenha o tempo necessário para compreender a execução, podendo também, analisar os valores gerados no campo de log. Entretanto, Bernardes (2016) também destaca que o protótipo possui uma limitação em relação ao processo de retroceder passo a passo durante a navegação no algoritmo de Dijkstra. Foi adotada inicialmente uma estratégia complexa que acabou consumindo mais tempo do que o esperado, fazendo com que esse processo não fosse concluído. Bernardes (2016) sugere que sejam incluídas funcionalidades como atalhos para navegação na execução dos algoritmos, permitir ocultar o log de execução e pseudocódigo, melhorias no desempenho dos algoritmos e de usabilidade.

2.3 TRABALHOS CORRELATOS

Neste capítulo são apresentados três trabalhos correlatos, todos trabalhos de conclusão de curso, que possuem características semelhantes ao tema proposto. O Quadro 5 descreve o trabalho de Braun (2009) que desenvolveu uma ferramenta para criação e execução de algoritmos em grafos. Logo após, o Quadro 6 descreve a ferramenta desenvolvida por Hackbarth (2008), que serve para representar graficamente estruturas de grafos. Por fim, o Quadro 7 apresenta o software educacional desenvolvido por Silveira (2007) que utiliza mecanismos de apoio ao estudo.

Quadro 5 – Ferramenta visual para criação e execução de algoritmos aplicados sobre teoria dos grafos

Referência	Braun (2009)
Objetivos	Desenvolver uma ferramenta para criação e execução de algoritmos em grafos, utilizando de reflexão computacional para visualizar a execução e os resultados obtidos do algoritmo implementado.
Principais funcionalidades	A ferramenta foi organizada em três partes. A primeira está relacionada aos algoritmos, que possui uma interface com as funcionalidades para editar, salvar e compilar códigos de algoritmos em Java. A segunda parte refere-se aos grafos ou dígrafos, com opções para editar, salvar, alterar atributos dos vértices ou arestas, visualizar propriedades e executar os algoritmos disponíveis. Por fim, a terceira parte permite a visualização da execução do algoritmo selecionado, que ocorre através da coloração dos vértices.
Ferramentas de desenvolvimento	Para o desenvolvimento foi utilizada a linguagem de programação Java, juntamente com o ambiente de programação NetBeans 6.7.1. Para instanciar os algoritmos em tempo de execução, foi utilizada a tecnologia de Reflexão Computacional. Foram utilizadas a bibliotecas Java Open Graphics Library (JOGL) para construção das interfaces gráficas, Document Object Model (DOM) para criação dos arquivos Extensible Markup Language (XML) e Another Neat Tool (ANT) para compilação em tempo de execução. Para salvar os grafos criados foi utilizado o padrão GraphML.
Resultados e conclusões	Segundo a autora, os resultados obtidos foram satisfatórios e todos os requisitos propostos foram cumpridos. Como limitação, a autora aponta que só é possível criar um grafo por vez, logo, o EVG não permite a verificação de grafos isomorfos. Durante o desenvolvimento, Braun (2009) teve dificuldades para carregar os algoritmos na memória do computador. Ela atribui esta limitação à linguagem de programação Java. A autora sugere como extensões: a possibilidade de edição de vários grafos ao mesmo tempo e a disponibilização de uma opção para pausar e retroceder, permitindo a execução passo a passo dos algoritmos.

Fonte: elaborado pelo autor.

Quadro 6 – Ferramenta para representação gráfica do funcionamento de algoritmos aplicados em grafos

Referência	Hackbarth (2008)
Objetivos	Desenvolver uma ferramenta para representar graficamente estruturas de grafos.
Principais funcionalidades	É possível realizar a criação de vértices e arestas, permitindo ao usuário organizá-lo de acordo com sua necessidade. Uma vez desenhado o grafo, o usuário poderá selecionar um algoritmo para acompanhar a execução através da coloração dos vértices e arestas.
Ferramentas de desenvolvimento	A ferramenta foi desenvolvida na linguagem de programação C++, juntamente com o ambiente de desenvolvimento Microsoft Visual Studio 2005. Foi utilizada a biblioteca GraphObj para realizar a camada lógica da ferramenta, enquanto que para a interface gráfica, foram utilizadas as bibliotecas IUP e OpenGL.
Resultados e conclusões	Segundo o autor, a ferramenta permite a criação de um grafo de forma interativa. Entretanto, o autor destaca como limitação a falta de alguns recursos gráficos e a impossibilidade de salvar e

	carregar um grafo. Ele relata que teve dificuldades para controlar os eventos de interface durante a execução dos algoritmos. Como extensões, o autor sugere diversas complementações, dentre elas: permitir o carregamento e armazenamento dos grafos, visualizar os custos das arestas, alterar as cores e gerar o reposicionamento automático do grafo.

Fonte: elaborado pelo autor.

Quadro 7 – Desenvolvimento de um aplicativo educacional para o estudo de teoria dos grafos

Referência	Silveira (2007).
Objetivos	Visa facilitar, inicialmente, aos alunos do curso de Ciência da Computação, o processo de criação, manipulação, análise e entendimento de determinados conceitos da Teoria dos Grafos.
Principais funcionalidades	Permite ao usuário criar os vértices e arestas, adicionar valores as arestas, informar se é um grafo ou dígrafo e realizar análises sobre os grafos criados. Segundo Silveira (2007), todas as telas do software possuem mensagens explicativas que abordam os conceitos sobre o que está sendo visualizado na tela, garantindo assim, uma introdução gradual para o entendimento do conteúdo da disciplina.
Ferramentas de desenvolvimento	Para o desenvolvimento do software foi utilizado o ambiente de desenvolvimento Delphi Enterprise 7.0.
Resultados e conclusões	O autor descreve que o grande diferencial do seu trabalho em relação aos existentes, são os mecanismos de apoio ao estudo, como mensagens explicativas e tutoriais sobre a disciplina de Teoria dos Grafos. Porém, conforme o autor, o software não foi testado em sala de aula, pois a disciplina não foi lecionada durante o período de desenvolvimento. Silveira (2007) sugere como extensões a realização de testes e a implementação de novos algoritmos abordando os conteúdos referentes à coloração e árvore geradora mínima.

Fonte: elaborado pelo autor.

3 DESCRIÇÃO DA FERRAMENTA

Este capítulo está organizado em quatro seções. A seção 3.1 descreve a especificação da ferramenta, através dos requisitos e diagrama de casos de uso. Na seção 3.2 apresenta a disposição dos componentes na ferramenta e as opções disponibilizadas no menu. A seção 3.3 aborda a implementação da representação gráfica e manipulação dos grafos. Por fim, a seção 3.4 descreve a forma que os algoritmos foram implementados e como acompanhar suas execuções. Os diagramas apresentados foram realizados utilizando a ferramenta online Draw.io.

3.1 ESPECIFICAÇÃO

Esta seção tem como objetivo apresentar os Requisitos Funcionais (RF) e Não Funcionais (RNF) definidos para o desenvolvimento da ferramenta. Eles podem ser observados no Quadro 8 e Quadro 9 juntamente com a matriz de rastreabilidade dos casos de uso.

Quadro 8 – Requisitos funcionais da ferramenta

Requisito Funcional	Caso de Uso
RF01: permitir manter grafos	UC01
RF02: permitir exportar e importar um grafo	UC02, UC03
RF03: permitir manter vértices e arestas	UC04, UC05
RF04: permitir mover vértices	UC06
RF05: permitir indicar se o grafo é dirigido	UC07
RF06: visualizar propriedades do grafo, matriz de adjacência e matriz de custos	UC08, UC09, UC10
RF07: permitir indicar vértice de início e destino nos algoritmos	UC11, UC12
RF08: permitir controlar a execução de algoritmos	UC14
RF09: permitir alterar custo das arestas	UC13

Fonte: elaborado pelo autor.

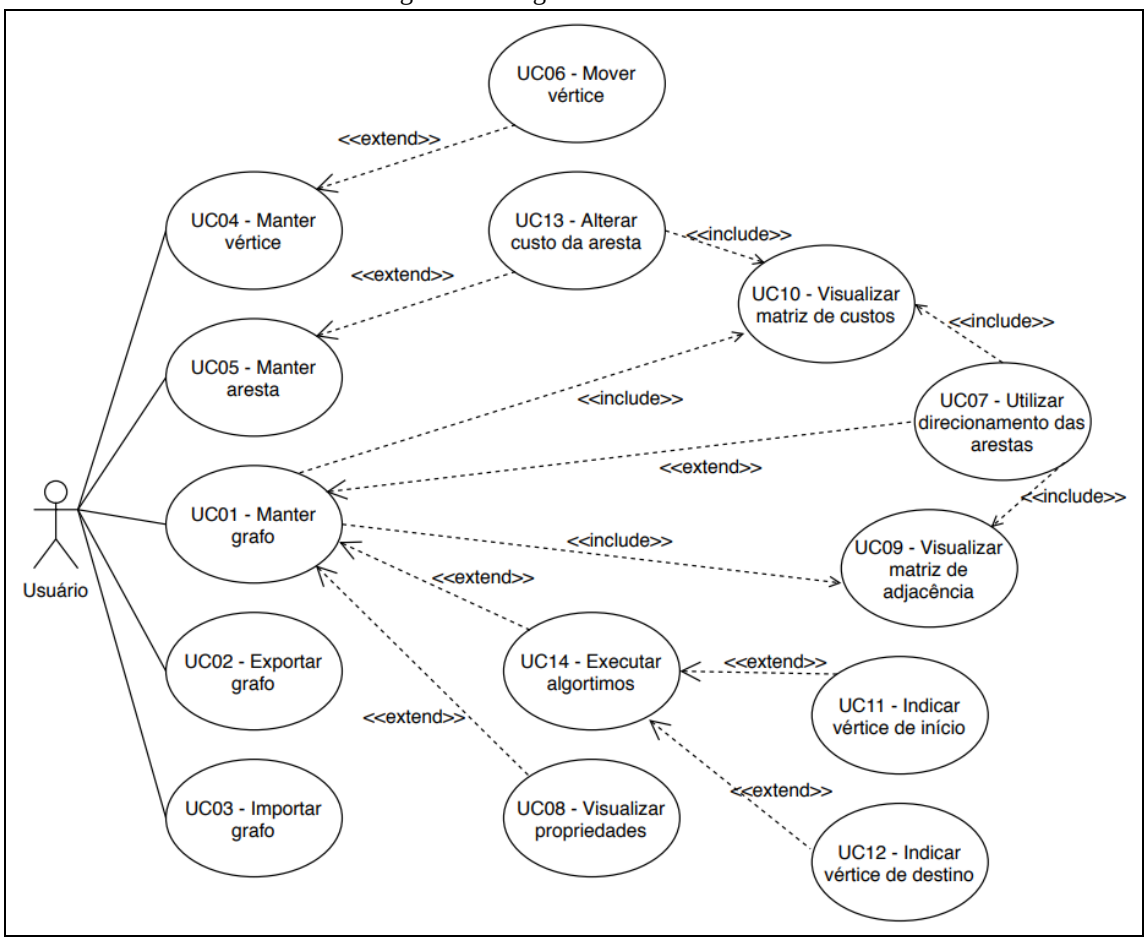
Quadro 9 – Requisitos não funcionais da ferramenta

Requisito Não Funcional
RNF01: facilitar identificações de vértices através de colorações
RNF02: sugerir nome aos vértices de forma sequencial
RNF03: ser desenvolvida na linguagem de programação C#
RNF04: utilizar o ambiente de desenvolvimento Microsoft Visual Studio 2015
RNF05: utilizar a biblioteca gráfica Windows Presentation Foundation (WPF)

Fonte: elaborado pelo autor.

Para representar as principais funcionalidades definidas pelos requisitos da ferramenta, foi elaborado um diagrama de casos de uso, que pode ser observado na Figura 2.

Figura 2 – Diagrama de casos de uso



Fonte: elaborado pelo autor.

As opções de adicionar e remover vértices e arestas são representadas pelos casos de uso `UC04 – Manter vértice` e `UC05 – Manter aresta`. Alterar a posição dos vértices pode ocorrer através do `UC06 – Mover vértice`. As arestas criadas podem ter seu custo alterado pelo caso `UC13 – Alterar custo da aresta`. Um grafo pode ser salvo, carregado, exportado e importado através dos casos: `UC01 – Manter grafo`, `UC02 – Exportar grafo` e `UC03 – Importar grafo`.

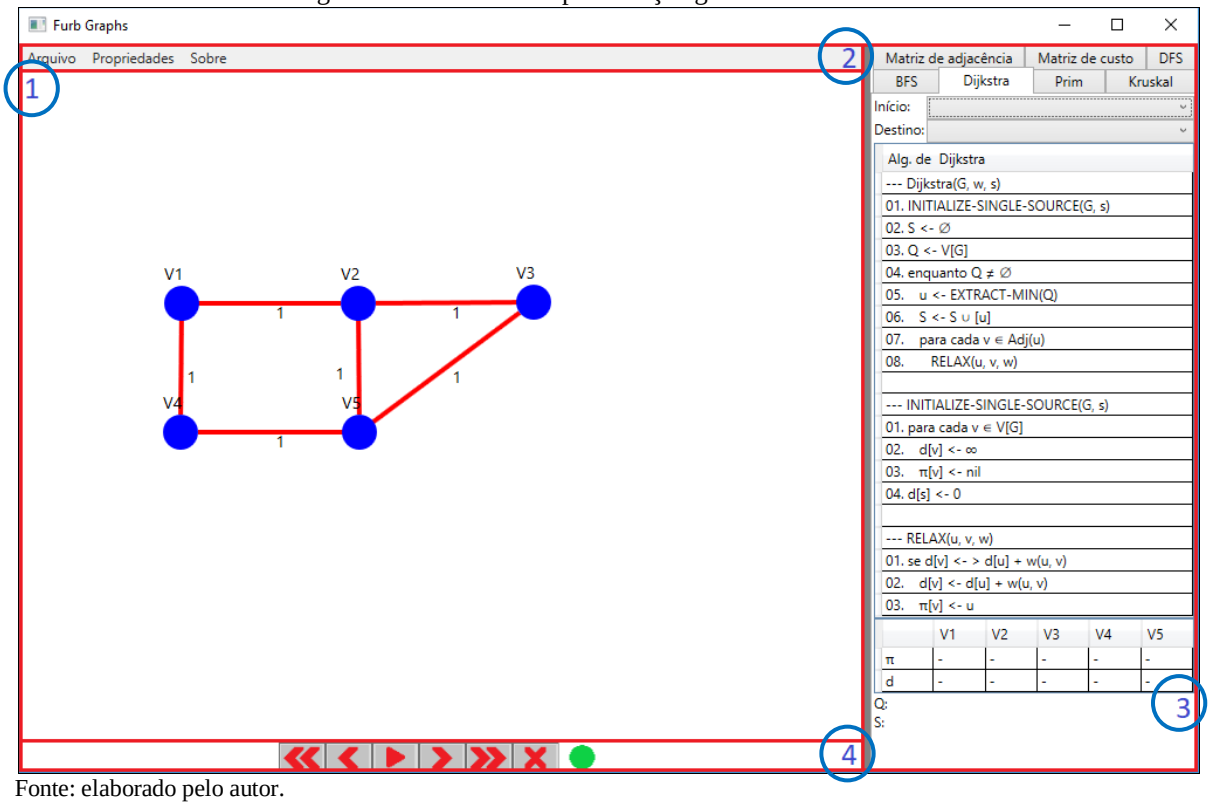
Informações sobre o grafo podem ser visualizadas utilizando os casos `UC08 – Visualizar propriedades`, `UC09 – Visualizar matriz de adjacência` e `UC10 – Visualizar matriz de custos`. A execução dos algoritmos disponibilizados é representada pelo caso `UC14 – Executar algoritmos`. Para obter um resultado específico, os vértices de início e destino podem ser informados utilizando os casos `UC11 – Indicar vértice de início` e `UC12 – Indicar vértice de destino`. O caso `UC07 – Utilizar direcionamento das aresta` permite que a direção das arestas seja considerada nas execuções dos algoritmos.

Nas próximas seções, são descritas as implementações realizadas na ferramenta. Inicialmente é apresentada a tela principal, identificando como os componentes disponibilizados estão dispostos e suas funcionalidades. Em seguida, descreve-se a criação do grafo e a representação visual. Por fim, apresenta-se como está organizada a execução e acompanhamento dos algoritmos DFS, BFS, Dijkstra, Prim e Kruskal.

3.2 DISPOSIÇÃO DOS COMPONENTES

Esta seção tem como objetivo contextualizar como os componentes presentes na ferramenta estão organizados, sendo representados na Figura 3.

Figura 3 – Mosaico da representação gráfica da ferramenta



Fonte: elaborado pelo autor.

A ferramenta está dividida em quatro áreas, sendo elas:

1. representação gráfica dos grafos: área designada para permitir inserir vértices e arestas, manipular os objetos inseridos e acompanhar graficamente a execução dos algoritmos (Figura 3 item 1);
2. menu: local em que é possível salvar, carregar, importar e exportar grafos, visualizar propriedades dos grafos e informações sobre a ferramenta (Figura 3 item 2);
3. algoritmos: apresenta os algoritmos disponibilizados e informações para acompanhar as suas execuções, como as matrizes de adjacência e custos (Figura 3 item 3);
4. botões de navegação: componentes responsáveis pelas execuções dos algoritmos (Figura 3 item 4).

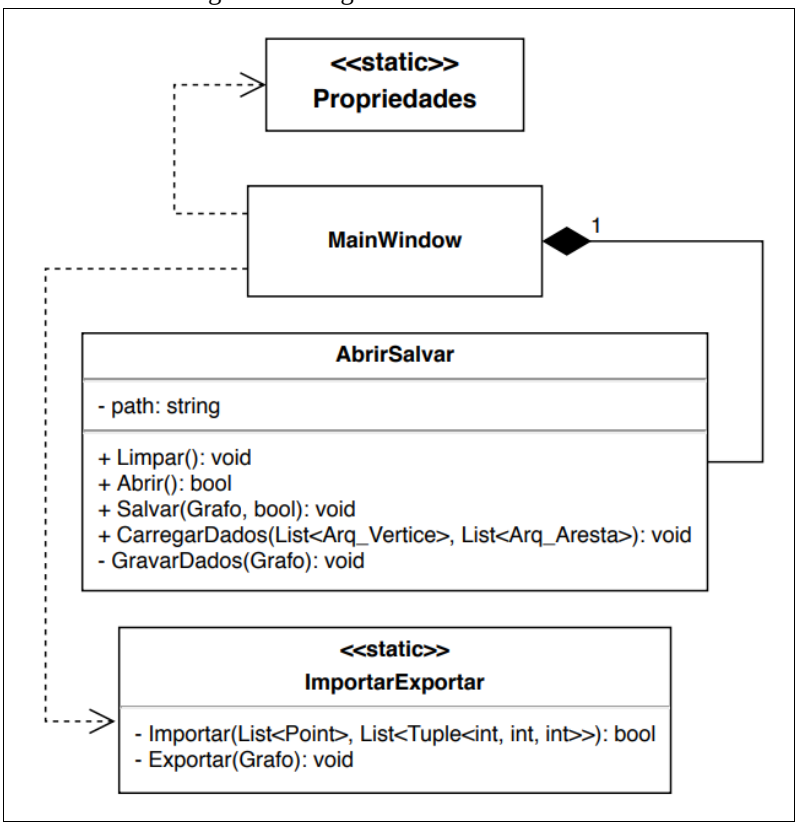
Um projeto WPF tem como principal elemento o arquivo XAML criado e é nele que os componentes são definidos. Junto à criação deste arquivo, o WPF cria a classe `MainWindow`, que será responsável pela inicialização da tela e tratamento das operações realizadas nos componentes. Cada uma das áreas apresentadas possui um item correspondente no arquivo XAML, sendo objetos do tipo `Canvas`, `Menu`, `TabControl` e `StackPanel` respectivamente, disponíveis no *framework* utilizado.

O objeto `Canvas` tem como objetivo armazenar os objetos gráficos criados e apresenta-los na tela, como vértices, arestas e textos. Já o `TabControl` é utilizado como um conjunto de abas. Cada aba dos algoritmos é representada por um `TabItem`, que será adicionado no `TabControl`. O `StackPanel` é um agrupamento de elementos, que neste caso são objetos do tipo `Button`. Por fim, o `Menu` é responsável por apresentar e consistir as ações realizadas no menu, que está dividido em três partes, sendo elas: `Arquivo`, `Propriedades` e `Sobre`. As classes responsáveis por realizar as ações do menu podem ser observadas na Figura 4.

As opções disponibilizadas no menu `Arquivo` são:

a) `novo`: limpa todo o conteúdo apresentado;
b) `abrir`: permite carregar um grafo previamente salvo;
c) `salvar`: salva o grafo no mesmo local caso tenha sido carregado, senão terá o mesmo funcionamento da opção salvar como;
d) `salvar como`: permite selecionar um local para salvar o grafo;
e) `importar`: permite selecionar um arquivo de importação para carregar um grafo;
f) `exportar`: permite selecionar um local para exportar o grafo;
g) `sair`: fecha a ferramenta.

Figura 4 – Diagrama de classes do menu



Fonte: elaborado pelo autor.

A classe `AbrirSalvar` permite armazenar e carregar o grafo. Para isso, ela transforma os objetos de `Vertice` e `Aresta` presentes no `Grafo` em objetos que permitem serialização. Assim é possível gravar e ler um arquivo com as informações necessárias como posições e custos.

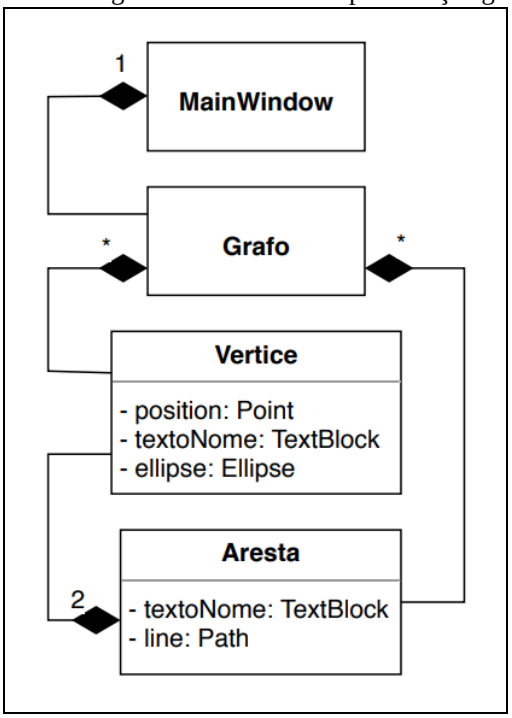
A classe `ImportarExportar` é responsável por realizar a interpretação e geração de um arquivo texto, permitindo assim importar e exportar um grafo. O arquivo texto possui um padrão estabelecido que é: iniciar com a quantidade de vértices do grafo e, nas linhas seguintes, indicar os pares de vértices que possuem arestas, podendo ser informado o custo de cada relacionamento.

O menu `Propriedades` tem como objetivo determinar e verificar propriedades sobre os grafos. Nele é possível indicar se o grafo que está sendo apresentado deve ser dirigido, ou seja, levar em consideração a direção em que a aresta foi criada. Esta opção é utilizada durante a execução dos algoritmos. Também estão disponíveis itens que apresentam características sobre o grafo, sendo eles nulo, regular, simples, multigrafo e completo. A classe `Propriedades` é responsável por tratar essas informações. Ela tem como objetivo encapsular a implementação da apresentação das propriedades do grafo. Para determinar qual ação a classe deve realizar, é utilizado um valor de enumeração que representa a opção selecionada no menu. Após determinar qual rotina deve ser executada, é apresentada uma tela com o resultado obtido.

3.3 REPRESENTAÇÃO GRÁFICA DOS GRAFOS

Nesta seção são descritas as classes utilizadas para a criação dos grafos e como representa-los graficamente. Também são apresentadas as opções disponíveis para manipulação dos grafos criados. As classes podem ser visualizadas na Figura 5.

Figura 5 – Diagrama de classes da representação gráfica



Fonte: elaborado pelo autor.

Os vértices criados são representados pela classe `Vertice`, que possui três atributos principais. O primeiro, representado por uma classe denominada `Point`, é a posição na tela em que o vértice será apresentado. O segundo é o nome do vértice, representado pela classe `TextBlock`, sendo gerado de forma automática para facilitar a sua identificação. Por fim, o terceiro atributo possui as propriedades de visualização, como tamanho e cor, representado pela classe `Ellipse`. Para que seja possível visualizar o vértice na tela, os atributos `TextBlock` e `Ellipse` devem ser adicionados ao `Canvas`.

As arestas criadas são representadas pela classe `Aresta`, que possui quatro atributos principais. Dois dos atributos são da classe `Vertice` e representam os vértices de início e destino. Semelhante ao vértice, a aresta também possui um objeto do tipo `TextBlock` para apresentar o custo da aresta, que inicialmente sempre possui o valor um (1). O quarto atributo, representado pela classe `Path`, é responsável pelas propriedades de visualização, como espessura e cor. Da mesma forma que o vértice, para ser possível visualizar a aresta na tela, os atributos `TextBlock` e `Path` devem ser adicionados ao `Canvas`.

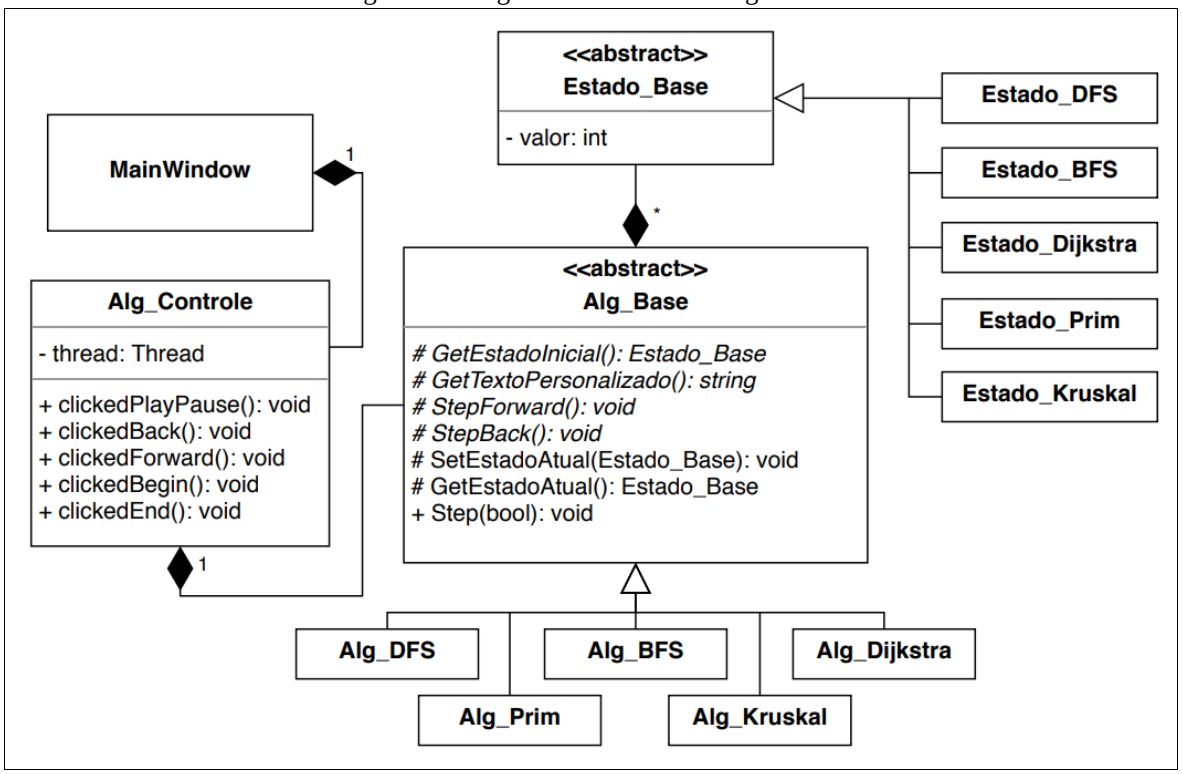
A classe `Grafo`, tem como objetivo encapsular as rotinas relacionadas aos vértices e arestas. Como atributos, possui duas coleções, do tipo `Vertice` e `Aresta`, para armazenar os objetos criados. Para facilitar a criação dos grafos, estão disponíveis opções para manipular os objetos.

A ferramenta permite a seleção de um ou mais vértices, para que assim seja possível move-los pela tela. Quando um vértice é alterado de posição, as arestas são redesenhadas conforme nova disposição. Isso ocorre pois as arestas utilizam os atributos do tipo `Vertice` para determinar suas novas posições de início e destino. Os vértices selecionados também podem ser removidos, juntamente com as arestas que possuem relacionamento com estes vértices. Para as arestas, estão disponíveis as opções de seleção, para que possam ser removidas, e alteração do custo, que é utilizada na execução dos algoritmos que são descritos na próxima seção.

3.4 ALGORITMOS

Os algoritmos estão disponibilizados em um controle de abas, sendo eles DFS, BFS, Dijkstra, Prim e Kruskal. Cada aba possui componentes com as finalidades de configurar e acompanhar a execução dos algoritmos. Estes componentes podem ser personalizados conforme a necessidade de cada algoritmo. Dentre os componentes estão a apresentação do pseudocódigo e as seleções dos vértices de início e destino. Também estão disponíveis as abas contendo as matrizes de adjacência e custos, que são atualizadas sempre que houver uma alteração no grafo. As informações apresentadas nessas abas são requisitadas à classe `Grafo`. A aba matriz de adjacência apresenta a quantidade de relacionamentos entre dois vértices, enquanto que a aba de custos apresenta a lista de custos das arestas. Na Figura 6, podem ser observadas as classes que implementam a execução dos algoritmos.

Figura 6 – Diagrama de classes dos algoritmos



Fonte: elaborado pelo autor.

A execução dos algoritmos pode ser acompanhada passo a passo, para isso, cada algoritmo foi separado em estados. Os estados são representados pela classe `Estado_Base`, que possui um atributo de identificação. Cada algoritmo possui uma classe especializada de `Estado_Base`, em que cada estado recebe um valor único. O algoritmo de BFS, por exemplo, utiliza da classe `Estado_BFS`, que possui uma herança para `Estado_Base`, e cria uma lista de objetos que identificam cada estado, conforme pode ser observado no Quadro 10.

Quadro 10 – Definição de estados do algoritmo de BFS

```
01. public class Estado_BFS : Estado_Base
02. {
03.     public const int e_0 = 0;
04.     public const int e_1 = 1;
05.     public const int e_2 = 2;
06.     [...]
07.     public static Estado_BFS Estado_BFS_0 = new Estado_BFS(e_0);
08.     public static Estado_BFS Estado_BFS_1 = new Estado_BFS(e_1);
09.     public static Estado_BFS Estado_BFS_2 = new Estado_BFS(e_2);
10.     [...]
11.     private Estado_BFS(int valor)
12.         : base(valor) { }
13. }
```

Fonte: elaborado pelo autor.

Cada algoritmo disponibilizado na ferramenta possui uma classe correspondente, com os controles necessários para realizar as execuções. Com isso, foi identificado que determinadas rotinas se repetiam, sendo necessário criar uma classe generalizada para ser utilizada como base. A classe `Alg_Base` tem como objetivo centralizar as rotinas dos algoritmos. Seu principal atributo é uma pilha do tipo `Estado_Base`, que armazena a sequência em que os estados foram processados, sendo o topo dessa pilha o próximo estado a ser processado. Ela também é responsável por realizar a seleção da linha atual no pseudocódigo e requisitar as classes especializadas o que deve ser apresentado nos componentes personalizados. As classes especializadas dos algoritmos processam o próximo estado conforme suas necessidades e atualizam a pilha de estados.

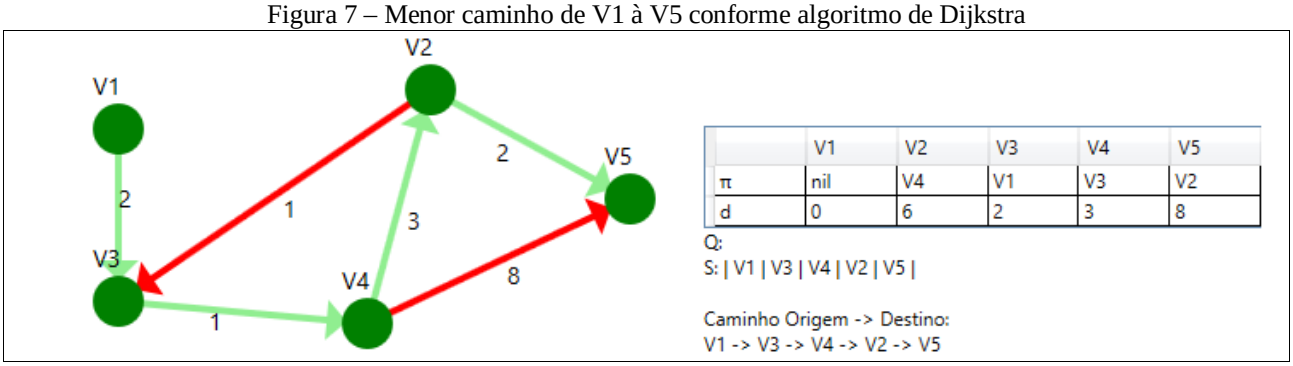
A execução e acompanhamento dos algoritmos possuem diversos componentes envolvidos. Assim, a classe `Alg_Controle` tem como objetivo encapsular estes componentes. O principal atributo desta classe é um objeto do tipo `Alg_Base`, que é responsável por armazenar qual algoritmo deve ser executado. Este atributo é atualizado sempre que uma aba diferente é selecionada. Para realizar o controle de execução, a classe `Alg_Controle` utiliza-se dos botões de navegação.

Os botões de navegação são responsáveis por determinar qual ação a classe `Alg_Controle` deve realizar e repassar ao algoritmo selecionado. Dentre estas ações, existem três características entre os botões. A primeira permite que os algoritmos executem apenas uma troca de estado por vez, ou seja, realizar o acompanhamento na forma de passo a passo. A segunda característica permite que a troca de estados seja feita de forma contínua, com um intervalo entre as trocas. Para isso, a classe `Alg_Controle` utiliza uma `Thread` para manter a troca de estados, simulando uma execução constante da primeira característica. Por fim, a terceira é semelhante a segunda, porém não existe um intervalo de tempo a ser aguardado entre as trocas. Com isso, é possível realizar uma execução direta do algoritmo, obtendo os resultados necessários sem realizar o acompanhamento da execução.

Junto aos botões de navegação, pode ser observada uma imagem que representa a situação de execução do algoritmo. Quando a imagem estiver na cor verde, significa que o grafo pode ser alterado, pois não existe algoritmo em execução. Porém, quando a imagem estiver na cor laranja, o processo de execução do algoritmo foi iniciado e não podem ser realizadas alterações no grafo até que o algoritmo seja finalizado ou cancelada sua execução.

Os vértices de início e destino, que podem ser selecionados, permitem fornecer uma flexibilidade ao executar os algoritmos, pois determinadas execuções podem apresentar resultados diferentes. O algoritmo de Dijkstra é um exemplo em que o vértice inicial é importante para obtenção dos resultados, pois o seu objetivo é apresentar o menor caminho de um vértice em relação aos demais do grafo. Nas execuções em que estes vértices foram informados, ao finalizar o algoritmo, a ferramenta altera cor dos vértices e arestas para destacar qual o caminho a ser realizado, caso ele exista. Esta funcionalidade pode ser observada na Figura 7.

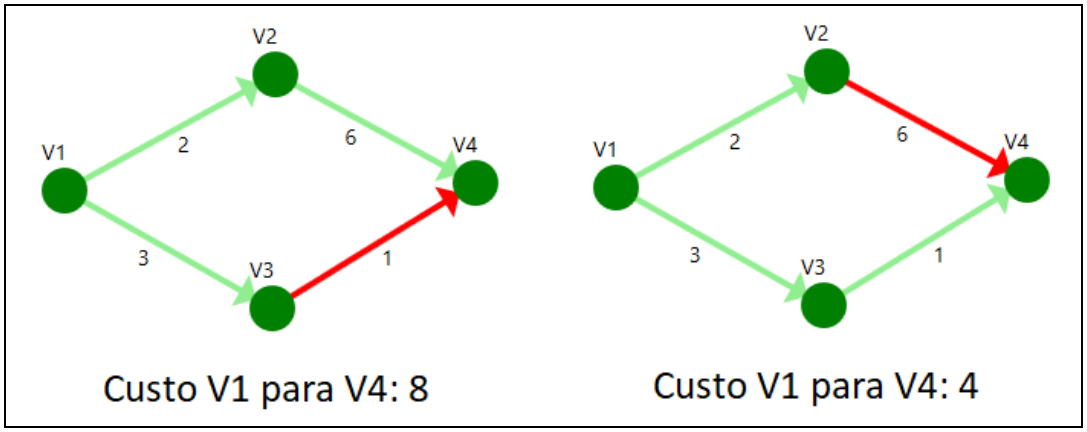
Figura 7 – Menor caminho de V1 à V5 conforme algoritmo de Dijkstra



Fonte: elaborado pelo autor.

Com a disponibilização da execução passo a passo, procura-se permitir ao usuário que ele possa avançar o processamento dos algoritmos conforme julgar necessário para compreender as ações realizadas pela troca de estado. Dessa forma, é possível visualizar e interpretar as informações apresentadas, através dos componentes disponibilizados. Retornando ao algoritmo de Dijkstra como exemplo, é possível acompanhar a comparação de custos das arestas do vértice em processamento com seus adjacentes. Isso pode ser observado na Figura 8, em que o caminho a ser realizado é alterado, pois é encontrado um outro caminho com menor custo. Também está disponível a lista de vértices pendentes de processamento e a visualização, através de um quadro, da relação de predecessores e custos entre os vértices. Ao final, é apresentado o caminho a ser percorrido do vértice inicial ao destino e destacado na representação do grafo, como forma de facilitar a interpretação.

Figura 8 – Função de relaxamento de Dijkstra para determinar o menor caminho



Fonte: elaborado pelo autor.

4 RESULTADOS E DISCUSSÕES

Este capítulo tem como objetivo apresentar o experimento realizado com a ferramenta proposta e seus resultados obtidos.

O experimento foi realizado com estudantes do curso de Ciência da Computação no mês de junho de 2018, em que foi disponibilizado um executável da ferramenta e um formulário a ser respondido. O formulário foi elaborado utilizando a ferramenta online Google Forms, sendo dividido em três etapas. A primeira etapa representa um questionário sobre o perfil do usuário, enquanto que a segunda, consiste de questionário de tarefas a serem realizadas utilizando a ferramenta proposta. Por fim, na terceira etapa é realizado um questionário de avaliação da ferramenta. Para avaliar se a ferramenta possui uma interface de fácil interação, os estudantes realizaram os testes sem um acompanhamento de uma pessoa que já conhecia seu uso. Este formulário foi disponibilizado e preenchido por 31 estudantes da disciplina de Teoria dos Grafos. No entanto, o questionário foi elaborado para ser aplicado considerando qualquer tipo de usuário. As principais informações obtidas com o questionário de perfil podem ser observadas no Quadro 11.

Quadro 11 – Perfis dos usuários envolvidos no experimento

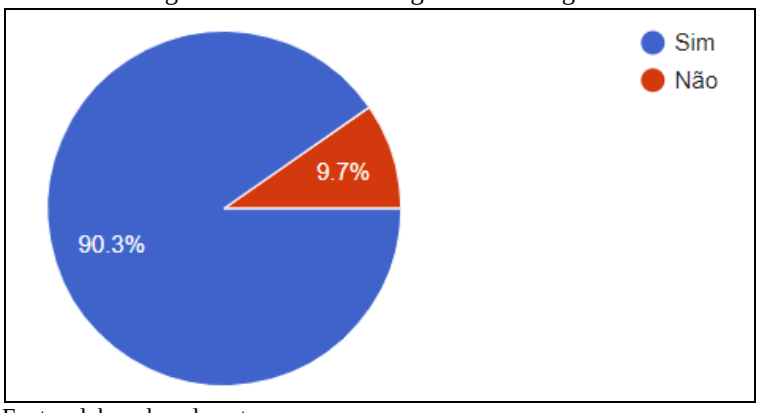
Sexo	87,1% masculino 12,9% feminino
Idade	64,5% tem entre 20 e 25 anos 25,8% tem entre 25 e 30 anos 6,5% tem entre 30 e 35 anos 3,2% tem mais de 35 anos
Atua na área de TI?	93,5% sim 6,5% não
Compreende o que é um grafo?	100% sim
Estudou alguma disciplina que envolvesse Teoria dos Grafo?	100% sim

Fonte: elaborado pelo autor.

A partir do questionário de perfil, pode-se observar que a maioria dos estudantes é do sexo masculino e que possuem variação considerável entre as idades. Também é notável que quase todos os estudantes já atuam na área de tecnologia. Em relação ao conteúdo abordado neste trabalho, todos os estudantes informaram que já estudaram alguma disciplina que envolvesse Teoria dos Grafos. Para realizar essa afirmação, há um pré-requisito de conhecimento na área de grafos, que reflete na resposta que todos compreendem o que é um grafo.

Em relação ao questionário de tarefas a serem realizadas, duas atividades são consideradas como principais na avaliação. A primeira é utilizar vértices e arestas para a criação de um grafo. Já a segunda atividade é conseguir realizar o acompanhamento e entendimento de um algoritmo na forma de passo a passo. Em cada atividade, o estudante pode além de responder, adicionar uma observação na forma de texto, caso julgue necessário adicionar alguma informação extra a ser analisada. Para avaliar a criação de um grafo, foi requisitado que ele tivesse 5 vértices e 7 arestas, sendo uma em laço e outra em paralelo. Os resultados obtidos podem ser observados na Figura 9.

Figura 9 – Usuário conseguiu criar um grafo

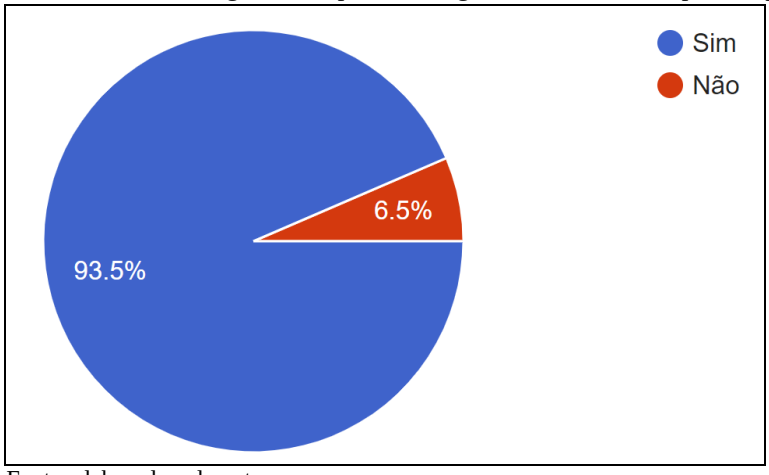


Fonte: elaborado pelo autor.

Conforme pode ser observado, grande parte dos estudantes conseguiu criar o grafo. Analisando as observações adicionadas por alguns estudantes, conclui-se que a criação dos elementos ocorreu de forma simples. Considerando as observações dos estudantes que não conseguiram, nota-se uma dificuldade em lembrar quais eram os comandos necessários a serem utilizados. Esta situação ocorre devido a não familiarização com a ferramenta, podendo ser corrigida utilizando-a por um período maior de tempo.

Para avaliar a execução passo a passo dos algoritmos de DFS e BFS, solicitou-se que o estudante escolhesse o algoritmo de forma livre, executando-o. O objetivo era verificar se os botões de navegação e as informações apresentadas colaboram no entendimento da execução do algoritmo. Os resultados podem ser observados na Figura 10.

Figura 10 – Usuário conseguiu acompanhar o algoritmo na forma de passo a passo



Fonte: elaborado pelo autor.

Semelhante a atividade anterior, grande parte dos estudantes conseguiu acompanhar a execução dos algoritmos na forma de passo a passo. Dentre as observações adicionadas, destacam-se as que apresentam a utilização de cores como auxílio na representação gráfica. Também é informado que ao realizar a troca de aba dos algoritmos, as informações de execução são perdidas. Porém, essa operação é uma característica da ferramenta, pois ao realizar a troca de abas, toda execução em andamento é cancelada.

De modo geral, os demais resultados e observações adicionadas nas demais atividades, apontam que a ferramenta executou de forma adequada, atingindo o resultado esperado. Em determinadas atividades, pode-se observar o relato da ocorrência de erro durante a execução. Estes relatos foram levados em consideração e corrigidos na versão final da ferramenta. Devido a uma característica do WPF em relação ao sistema operacional da máquina, a ferramenta não pode ser executada por um dos estudantes. Ocorreram também sugestões como adicionar uma legenda ao posicionar o cursor sobre os botões de navegação, apresentando a sua funcionalidade.

O questionário de avaliação tem como propósito, julgar se a ferramenta pode ser utilizada como um método auxiliar no estudo de Teoria dos Grafos. As perguntas realizadas podem ser agrupadas conforme o contexto em que estão relacionadas. Assim, o Quadro 12 apresenta as perguntas referentes a interface gráfica e disposição dos componentes.

Quadro 12 – Perguntas e resultados sobre interface gráfica e disposição dos componentes

Perguntas / Critérios de avaliação	**Concordo totalmente**	**Concordo parcialmente**	**Discordo parcialmente**	**Discordo totalmente**
De modo geral, você acredita que a ferramenta foi intuitiva e de fácil utilização?	16,1%	66,7%	9,7%	6,5%
Para você, a interface gráfica e a disposição dos campos facilitaram a utilização da ferramenta?	12,9%	74,2%	6,5%	6,5%
Você consegue se lembrar facilmente de como fazer as operações na ferramenta?	48,4%	32,3%	16,1%	3,2%
Para você a criação/manipulação dos vértices/arestas é de fácil utilização?	29,0%	35,5%	25,8%	9,7%

Fonte: elaborado pelo autor.

A partir dos resultados obtidos, percebe-se que a interface foi bem avaliada, devido ao acesso de forma simples para as opções disponibilizadas, caracterizando-se assim como uma interface intuitiva. Isso permite que o usuário aprenda como utilizar a ferramenta em um curto período de tempo. As poucas avaliações negativas ocorreram devido a utilização da ferramenta sem conhecimento prévio. Utilizando-se do conceito de curva de aprendizagem, há uma grande possibilidade de que, se realizado os mesmos testes após uma apresentação do ambiente, os estudantes teriam um desempenho melhor ao manusear a ferramenta. Percebe-se também que a partir do momento que os usuários estavam

familiarizados com a interface, a utilização tornou-se mais fácil, pois foram disponibilizados comandos e ações com objetivos diretos.

Após realizar a criação dos grafos, foi requisitado que os estudantes utilizassem a execução dos algoritmos. O objetivo era verificar se o acompanhamento passo a passo permite uma melhor compreensão do que está sendo executado pelos algoritmos, se as informações apresentadas são suficientes para explicar cada passo e apresentar como os algoritmos realizam as decisões essenciais para o seu funcionamento. Os resultados obtidos podem ser observados no Quadro 13.

Quadro 13 – Perguntas e resultados sobre execução e acompanhamento dos algoritmos

Perguntas / Critérios de avaliação	**Concordo totalmente**	**Concordo parcialmente**	**Discordo parcialmente**	**Discordo totalmente**
Você teve dificuldades para interpretar cada passo avançado nos algoritmos?	6,5%	32,3%	22,6%	38,7%
O modelo de navegação através dos botões foi prático e interativo?	54,8%	19,4%	22,6%	3,2%
A funcionalidade de retroceder um passo do algoritmo se fez útil para compreensão?	67,7%	19,4%	9,7%	3,2%
A funcionalidade de retrocedes todos os passos do algoritmo se fez útil para compreensão?	67,7%	9,7%	19,4%	3,2%
No algoritmo de busca em largura, você compreendeu o vetor de roteamento?	74,2%	19,4%	3,2%	3,2%
No algoritmo de busca em profundidade, o empilhamento e desempilhamento do algoritmo de busca em profundidade ficou visível e compreensível?	77,4%	16,1%	3,2%	3,2%
No algoritmo de Dijkstra, a representação gráfica contribuiu para o entendimento do algoritmo que é determinar o menor caminho entre uma origem/destino?	64,5%	32,3%	0,0%	3,2%
A compreensão geral dos algoritmos melhorou depois da utilização da ferramenta?	35,5%	54,8%	3,2%	6,5%

Fonte: elaborado pelo autor.

Em relação ao acompanhamento passo a passo, os estudantes responderam que tiveram certa dificuldade em compreender como utilizar os botões de navegação. A partir do momento em que os botões foram compreendidos, nota-se que eles foram avaliados como úteis para realizar a interpretação do algoritmo em execução. Também é possível observar que os componentes personalizados, que apresentam características de cada execução, permitiram um melhor acompanhamento da situação de processamento dos vértices. Segundo os estudantes, após a utilização da ferramenta, a compreensão dos algoritmos teve uma melhora considerável, devido à combinação dos componentes disponibilizados com a execução em etapas.

Para acompanhar as execuções dos algoritmos, é apresentado o pseudocódigo e realizada uma coloração dos vértices. Estas informações apresentam as características dos algoritmos de uma maneira mais simplificada, quando comparadas a outros componentes. Para avaliar estas informações, foram realizadas as perguntas do Quadro 14.

Quadro 14 – Perguntas e resultados sobre pseudocódigo e coloração

Perguntas / Critérios de avaliação	**Concordo totalmente**	**Concordo parcialmente**	**Discordo parcialmente**	**Discordo totalmente**
Na sua opinião, as cores utilizadas na representação gráfica contribuíram para a compreensão dos algoritmos?	77,4%	16,1%	3,2%	3,2%
O pseudocódigo ajudou para a compreensão dos algoritmos?	83,9%	6,5%	3,2%	6,5%
A combinação do pseudocódigo e representação gráfica juntos fizeram sentido para você?	77,4%	16,1%	0,0%	6,5%

Fonte: elaborado pelo autor.

De acordo com as respostas, percebe-se que o pseudocódigo e coloração foram fundamentais para o entendimento da execução dos algoritmos, pois é possível relacionar o passo realizado no pseudocódigo com o seu efeito sobre o grafo apresentado. A coloração permite visualizar o processamento dos vértices e arestas, além de destacar os caminhos requisitados de uma forma simples, sem precisar realizar a interpretação do quadro de relacionamentos. A seleção da linha no pseudocódigo fornece a informação de qual ação foi realizada naquele momento, podendo ser relacionada com a alteração na representação gráfica. Segundo um dos estudantes, a cor da linha selecionada poderia receber um tom diferente, devido a uma dificuldade em identificá-la.

Por fim, as últimas perguntas tinham como objetivo avaliar se a ferramenta pode causar um impacto positivo ao ser utilizada no auxílio do estudo de grafos. Assim, foram realizadas as perguntas disponibilizadas no Quadro 15.

Quadro 15 – Perguntas e resultados sobre impacto da ferramenta no ambiente de estudo

Perguntas / Critérios de avaliação	**Concordo totalmente**	**Concordo parcialmente**	**Discordo parcialmente**	**Discordo totalmente**
Para o estudo de grafos, seria útil utilizar esta ferramenta?	87,1%	6,5%	0,0%	6,5%
Você precisaria de ajuda para operar a ferramenta?	9,7%	22,6%	16,1%	51,6%
A ferramenta em algum momento apresentou algum comportamento inesperado?	41,9%	12,9%	0,0%	45,2%
Seria adequado recomendar esta ferramenta para outras pessoas que estudam grafos?	77,4%	16,1%	0,0%	6,5%

Fonte: elaborado pelo autor.

Os estudantes consideram a ferramenta válida como auxílio no ensino de grafos e que recomendariam para outras pessoas. Contudo, como foi observado anteriormente, os estudantes consideram que para facilitar, é interessante ter uma pessoa para ajudar a operar os passos iniciais. Também pode ser observada a existência de comportamentos inesperados durante a operação. Dentre esses comportamentos, faz-se necessário avaliar os casos de forma individual para verificar se é uma falha de interpretação ou usabilidade. Assim, boa parte dos estudantes recomendaria a ferramenta para outras pessoas que estudam na área, pois os benefícios proporcionados compensam o esforço investido em aprender a utiliza-la.

Pode-se concluir que os resultados do questionário de avaliação, refletem os resultados alcançados no questionário de atividades. Pois grande parte dos estudantes conseguiram utilizar a ferramenta de forma simples, compreender as execuções dos algoritmos e considerar a ferramenta um método útil para aprendizado na área de grafos. Também é possível avaliar que os estudantes que tiveram dificuldades, teriam um desempenho melhor se acompanhados de uma pessoa para auxiliar no uso da ferramenta. Avaliando os resultados do acompanhamento passo a passo, percebe-se que as informações apresentadas nos componentes auxiliares e as cores influenciaram diretamente no entendimento dos algoritmos.

5 CONCLUSÕES

Este trabalho propôs uma reestruturação do FURB Graphs, desenvolvido por Bernardes (2016), com o objetivo de permitir que a ferramenta possa ser utilizada para auxiliar o ensino da disciplina de teoria dos grafos. Dessa forma, conceitos presentes no trabalho de Bernardes (2016) foram mantidos, como criação de grafos a partir de vértices e arestas, visualização de propriedades e acompanhamento dos algoritmos na forma de passo a passo. Foram mantidos os algoritmos de DFS, BFS e Dijkstra, além de adicionados os algoritmos de Prim e Kruskal, todos com o acompanhamento completo da execução.

A realização da alteração da linguagem de programação de Java para C# e a utilização do WPF ocorrem sem problemas. Isso traz a ferramenta a possibilidade de permitir utilizar componentes gráficos mais avançados. Contudo, essa alteração trouxe uma limitação em relação ao sistema operacional (OS), na qual o OS deve ser Windows compatível com .Net Framework 4.5.2.

A partir dos resultados obtidos pelo experimento realizado, a ferramenta foi aprovada, por grande parte dos estudantes, como forma de auxiliar na aprendizagem de grafos. Ela possibilita a criação e manipulação de grafos de forma simples, a visualização de propriedades e o acompanhamento e execução dos algoritmos na forma de passo a passo. Também é possível afirmar que a utilização dos componentes nas abas e a utilização de coloração para destacar elementos, facilitou o entendimento geral dos algoritmos. A apresentação das matrizes de adjacência e custos possibilita uma visualização diferenciada das informações, quando comparada as demais ferramentas. Permitir ao usuário, através dos botões de navegação, avançar ou retroceder os passos quando necessário, torna dinâmica a utilização da ferramenta.

Vale ressaltar que alguns ajustes devem ser realizados para diminuir a necessidade de ter uma pessoa para apresentar e ensinar as funcionalidades da ferramenta. Por exemplo, uma melhor apresentação dos atalhos a serem utilizados.

Uma limitação encontrada durante o desenvolvimento foi em relação a *thread* de controle dos algoritmos. A classe utilizada não permitir inicializar uma *thread* no modo suspensa. Assim, durante toda a execução da ferramenta, a *thread* está em execução, consumindo assim, um processamento que é desnecessário. Logo, alterar a classe da *thread* ou encontrar uma outro forma de realizar este tratamento, poderia tomar muito tempo, devido ao pouco conhecimento da linguagem.

Pode-se concluir que o objetivo deste trabalho foi atingido, tendo assim, mais uma ferramenta disponível para auxiliar no aprendizado de teoria dos grafos. As características de criação, manipulação, coloração e execução de algoritmos de forma interativa, traz ao usuário uma experiência positiva no seu momento de estudos. Este trabalho pode servir como base para próximas pesquisas e até uma possível continuação do seu desenvolvimento, de forma a trazer possíveis avanços na utilização de softwares auxiliares no ensino de disciplinas.

Como sugestões de extensões para este trabalhos, têm-se: 1) apresentar descrição ao posicionar cursor sobre os botões de navegação; 2) adicionar a possibilidade de visualizar outras propriedades do grafo; 3) permitir mover os componentes da tela para uma janela separada; 4) melhorar o posicionamento inicial dos vértices quando realizada importação de um grafo; 5) melhorar cálculo da curvatura das arestas paralelas; 6) adicionar material de apoio ao estudo em um menu específico e; 7) permitir para a thread de controle de execução quando não houver algoritmos em processamento.

REFERÊNCIAS

BERNARDES, Luiz H. **FURB Graphs**: uma ferramenta de apoio ao aprendizado para a disciplina de teoria dos grafos. 2016. 91 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BORBA, Anderson de. **FURB Graphs**: uma aplicação para teoria dos grafos. 2014. 78 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BRAUN, Susan. **Ferramenta visual para criação e execução de algoritmos aplicados sobre teoria dos grafos**. 2009. 73 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

CARVALHO, Marco Antonio M. **BCC204 - Teoria dos grafos**. Ouro Preto, [2018]. Disponível em: <http://www.decom.ufop.br/marco/site_media/uploads/bcc204/04_aula_04.pdf>. Acesso em: 15 nov. 2018.

CALDAS, Rosangela F. Análise da integração da tecnologia em instituições de ensino superior através da gestão do conhecimento: projeto Unintera. In: CONGRESSO INTERNACIONAL TIC E EDUCAÇÃO, 2. 2012, Lisboa. **Anais...** Lisboa: Instituto de Educação da Faculdade de Lisboa, 2013. Disponível em: <http://ticeduca.ie.ul.pt/atas/pdf/74.pdf>. Acesso em: 01 out. 2018.

FARINHA, Solange S. **Tecnologia aplicada ao ensino**: perspectivas no âmbito da docência. 2005. 45 f. Trabalho de Conclusão de Curso (Especialização) – Curso de Pós-Graduação em Docência do Ensino Superior, Universidade Candido Mendes, Rio de Janeiro.

FEOFILOFF, Paulo; KOHAYAKAWA, Yoshirau; WAKABAYASHI. Yoshiko. **Uma Introdução Sucinta à Teoria dos Grafos**. São Paulo, [2011]. Disponível em: <https://www.ime.usp.br/~pf/teoriadosgrafos/texto/TeoriaDosGrafos.pdf>. Acesso em: 15 nov. 2018.

FEOFILOFF, Paulo. **Algoritmo de Prim**. São Paulo, [2017a]. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/prim.html>. Acesso em: 2 dez. 2018.

_____. **Algoritmo de Kruskal**. São Paulo, [2017b]. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/kruskal.html>. Acesso em: 2 dez. 2018.

_____. **Árvores geradoras de custo mínimo (MST)**. São Paulo, [2017c]. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/mst.html>. Acesso em: 12 dez. 2018.

FERREIRA, Gessé P.; LOZANO, Abel R. G. A viabilidade do ensino de matemática discreta na educação básica usando modelagem matemática. In: CONGRESSO NACIONAL DE EDUCAÇÃO, 9, 2009. Curitiba. **Anais eletrônicos...** Curitiba: PUCPR, 2009. Disponível em: <http://www.educere.bruc.com.br/arquivo/pdf2009/3092_1808.pdf>. Acesso em: 25 nov. 2018.

HACKBARTH, Rodrigo. **Ferramenta para representação gráfica do funcionamento de algoritmos aplicados em grafos**. 2008. 62 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

LIMA, Edirlei, Soares de. **Projeto e análise de algoritmos**. Rio de Janeiro, [2015]. Disponível em: <http://edirlei.3dgb.com.br/aulas/paa/PAA_Aula_06_Busca_Grafos_2015.pdf>. Acesso em: 25 nov. 2018.

LOBO, Fernando. **Caminho mais curto**. [S.l.], [2017]. Disponível em: <http://www.fernandolobo.info/aed1718/teoricas/a24.pdf>. Acesso em: 25 nov. 2018.

MELO, Gildson Soares de. **Introdução à Teoria dos Grafos.** 2014. 35 f. Dissertação (Mestrado Profissional em Matemática) - Universidade Federal do Paraíba, João Pessoa, 2014.

MÉNDEZ, Yasmín S.; GUARDIA, Luis E. T. **Problema do caminha mais curto**: algoritmo de Dijkstra. Rio de Janeiro, [2008]. Disponível em: <https://www.marinha.mil.br/spolm/sites/www.marinha.mil.br.spolm/files/006_1.pdf>. Acesso em: 15 nov. 2018.

MOREIRA, Antonio F. B.; KRAMER, Sonia. Contemporaneidade, educação e tecnologia. **Educação e Sociedade**, Campinas, v. 28, n. 100, p. 1037-1057, out. 2007.

PANTUZA, Gustavo. **Busca em profundidade**. [S.l.], [2017a]. Disponível em: <https://blog.pantuza.com/artigos/busca-em-profundidade>. Acesso em: 15 nov. 2018.

_____. **Busca em largura**. [S.l.], [2017b]. Disponível em: <https://blog.pantuza.com/artigos/busca-em-largura>. Acesso em: 15 nov. 2018.

QUITZAU, José Augusto Amgarten. **Árvores espalhadas mínimas**. [S. l.], [2003]. Disponível em: <http://www.ic.unicamp.br/~meidanis/courses/mo417/2003s1/aulas/2003-05-16.html>. Acesso em: 2 dez. 2018.

RABUSKE, Márcia A. **Introdução à teoria dos grafos**. 1 ed. Florianópolis: Ed. da UFSC, 1992.

ROCHA, Daniel Amaral de Medeiros. **O problema da árvore geradora mínima (AGM)**. [S.l.], [2007]. Disponível em: <http://danielamaral.wikidot.com/o-problema-da-arvore-geradora-minima-agm>. Acesso em: 15 nov. 2018.

SILVA, Luciana P. A utilização de recursos tecnológicos no ensino superior. **Revista Olhar Científico**, v. 1, n. 2, p. 267-268, 2010.

SILVEIRA, Eduardo B. A. **Desenvolvimento de um aplicativo educacional para o estudo de Teoria dos Grafos**. 2007. 13 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Departamento de Ciência da Computação, Universidade Presidente Antônio Carlos.