UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

NÍCOLAS ERCIRO ZANATTO

GRAPHEDU: SISTEMA WEB PARA VISUALIZAÇÃO DE ALGORITMOS DE GRAFOS

CAXIAS DO SUL

2021

NÍCOLAS ERCIRO ZANATTO

GRAPHEDU: SISTEMA WEB PARA VISUALIZAÇÃO DE ALGORITMOS DE GRAFOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dr. Ricardo Vargas Dorneles

CAXIAS DO SUL

2021

NÍCOLAS ERCIRO ZANATTO

GRAPHEDU: SISTEMA WEB PARA VISUALIZAÇÃO DE ALGORITMOS DE GRAFOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 30/11/2021

BANCA EXAMINADORA

Prof. Dr. Ricardo Vargas Dorneles
Universidade de Caxias do Sul - UCS

Prof. Dra. Helena Graziottin Ribeiro
Universidade de Caxias do Sul - UCS

Prof. Me. Alexandre Erasmo Krohn Nascimento
Universidade de Caxias do Sul - UCS

AGRADECIMENTOS

Gostaria de agradecer e dedicar esta monografia a minha família, por todo o carinho e apoio durante esse caminho tão importante na minha vida. Meu pai, Erciro e a minha mãe, Márcia, que não me deixaram faltar nada. Meu irmão, Henrique, pelas dicas. E a minha namorada, Teiffny, que me incentivou e cuidou de tudo em momentos difíceis do desenvolvimento. Este trabalho foi por vocês!

Agradeço ao meu grande amigo Carlos, pela amizade de longa data e momentos de alegria.

Agradeço ao Professor Ricardo Vargas Dorneles, por ter me ensinado o que era uma variável, pelas aulas de reforços aos sábados para a disciplina de Algoritmos e por toda a atenção durante a orientação deste projeto. Também sou grato ao professor Alexandre Erasmo Krohn Nascimento e a professora Helena Graziottin Ribeiro pelas dicas nesse projeto e pelas demais disciplinas ministradas.

Muito obrigado a todos os amigos e familiares que me incentivaram e me ajudaram para conclusão deste trabalho. Por fim, agradeço a Deus que tem olhado por mim em todos os momentos e colocou tantas pessoas especiais na minha vida.

"Imagination will often carry us to worlds that never were, but without it we go nowhere."

Carl Sagan

RESUMO

A teoria dos grafos é um ramo da matemática em que os conceitos e algoritmos abordados são utilizados em diversas áreas do conhecimento. Este trabalho teve como objetivo realizar o desenvolvimento de uma aplicação capaz de simular a visualização de alguns dos principais algoritmos de grafos (Dfs, Bfs, Dijkstra, Floyd-Warshall, Bellman-Ford, Prim e Kruskal), a fim de contribuir com o aprendizado dos alunos que estão estudando essa área. Para tal, foi implementada uma plataforma que possibilitasse a criação de grafos e a visualização da execução dos algoritmos através de seu pseudocódigo. Para realizar o desenvolvimento do sistema foi utilizada a biblioteca *D3.js*, usada na criação dos grafos, a biblioteca *React Js*, utilizada no *front-end* da aplicação e a linguagem de programação *TypeScript* para a implementação dos algoritmos. A implementação baseou-se no sistema *python-tutor*, uma plataforma para visualização linha a linha da execução de um código, deste modo toda execução do algoritmo é armazenada e o resultado é apresentado ao usuário após o fim da execução. O sistema foi validado pelos alunos do curso de Ciência da Computação da Universidade de Caxias do Sul e teve aprovação dos estudantes para a utilização em sala de aula com o acompanhamento de um professor.

Palavras-chave: Teoria dos Grafos. Visualização. Simulação. Algoritmos. Aprendizado.

ABSTRACT

Graph theory is a branch of mathematics that the concepts and algorithms discussed are used in different areas of knowledge. This paper aimed the development of an application capable of simulating the visualization of some of the main graph algorithms (Dfs, Bfs, Dijkstra, Floyd-Warshall, Bellman-Ford, Prim and Kruskal), to contribute to the learning of students who are studying this area. An application that allows the creation of graphs and the visualization of the execution of the algorithms through its pseudocode has been implemented. To develop the software the D3.js library was used, used in implemention of the graph creations, React Js library, used in applications's front-end and the programming language TypeScript, used to implement the algorithms. The implemention was based on the application Python Tutor, a platform for line-by-line visualization of code execution, this way all execution is stored and the result is presented to the user by the end of execution. The application has been validated by the students of Computer Science of Universidade de Caxias do Sul and had the approval of the students to use it with the attendance of a professor.

Keywords: Graph Theory. Visualization. Simulation. Algorithms. Learning.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

DFS *Depth-first search*
BFS *Breadth First Search*
XML *Extensible Markup Language*
SVG *Scalable Vector Graphics*
DOM *Document Object Model*
HTML *HyperText Markup Language*
CSS *Cascading Style Sheets*
JSON *JavaScript Object Notation*

SUMÁRIO

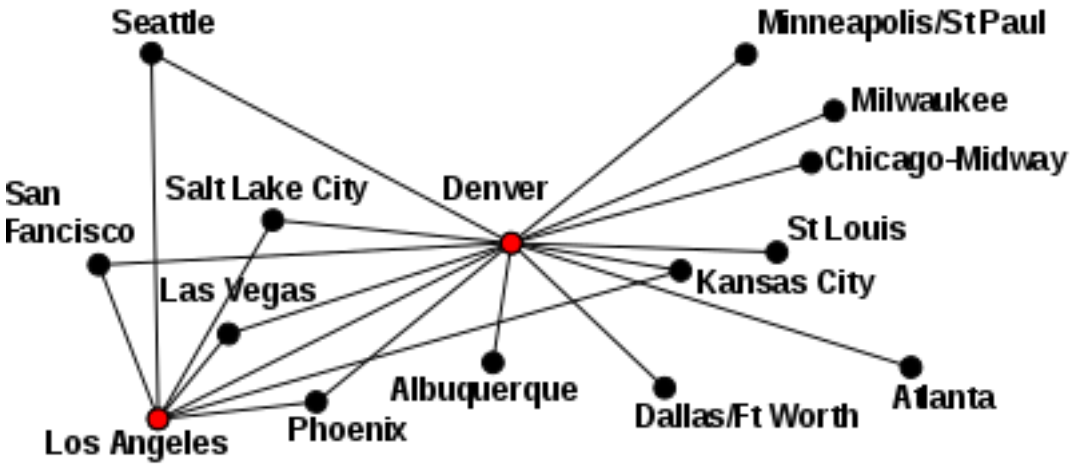
1 INTRODUÇÃO

A Teoria dos Grafos é uma área da matemática direcionada ao estudo e análise das relações entre os objetos e relacionamentos de um determinado conjunto de dados, esse agrupamento de objetos e relacionamentos é usualmente reconhecido como grafos (NICOLETTI, 2018).

Um grafo é definido como $G = (V(G),E(G))$. Sendo V o conjunto de objetos chamados de vértices, obrigatoriamente não vazio, e E o conjunto de relacionamentos chamados de arestas. O modo mais habitual de retratar um grafo é com a criação de um diagrama, em que os vértices são os pontos e as arestas são as ligações que unem um ponto ao outro (NICOLETTI, 2018).

No mundo real, há diversas situações que podem ser modeladas com base em um grafo. Por exemplo, em um diagrama de sistema de transporte aéreo de pessoas e mercadorias, os pontos são as cidades e os relacionamentos são os voos comerciais feitos de uma cidade a outra, conforme ilustrado na Figura 1. (BONDY, 1976)

Figura 1 – Diagrama de Transporte Aéreo

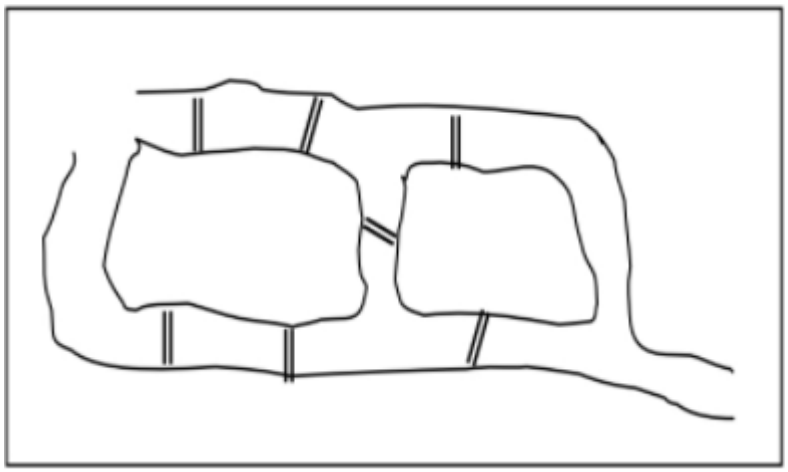


Fonte: (SPOKE–HUB. . . , 2021)

O marco inicial que colaborou para a criação do ramo da matemática da teoria dos grafos foi o problema das pontes de Königsberg. Existia na cidade de Königsberg, atual Kaliningrado, um conjunto de 7 pontes que cruzavam o rio Pregel, Figura 2. O problema consistia em descobrir uma rota que possibilitasse alguém a passar todas as 7 pontes e retornar ao seu ponto de partida, sem atravessar nenhuma das pontes mais de uma vez (CARLSON, 2010).

Leonhard Euler apresentou, em 26 de agosto de 1735, para os membros da Petersburg Academy a sua solução para o problema (ALEXANDERSON, 2006). A forma

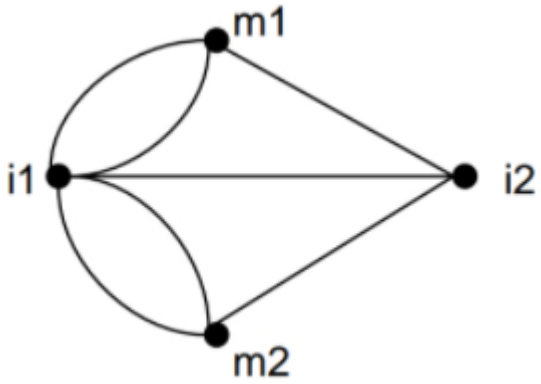
Figura 2 – As Sete pontes de Königsberg



Fonte: (NICOLETTI, 2018)

como Euler demonstrou as pontes de Königsberg, serve de exemplo em como associar um problema real a um problema de grafos. Para simplificar, Euler escreveu o problema em forma de diagrama, Figura 3. Ele usou 4 pontos para representar as áreas acessadas pelas pontes e 7 arcos para simbolizar as pontes (NRICH, 2011)

Figura 3 – Diagrama das Pontes de Königsberg



Fonte: (NICOLETTI, 2018)

Em sua primeira afirmação, Euler diz que se houver mais de duas áreas, relacionadas com *N* número de pontes, sendo N um número ímpar, esse caminho é inviável. A segunda afirmação foi de que, se para somente duas áreas, o número de pontes relacionadas for ímpar, o caminho é possível caso ele comece em alguma das duas áreas com o número de pontes ímpares. Em sua última afirmação, Euler declara que se não existir áreas relacionadas com um número ímpar de pontes para outras áreas, o caminho pode ser feito a partir de qualquer região (LEONARD. . . , 2011).

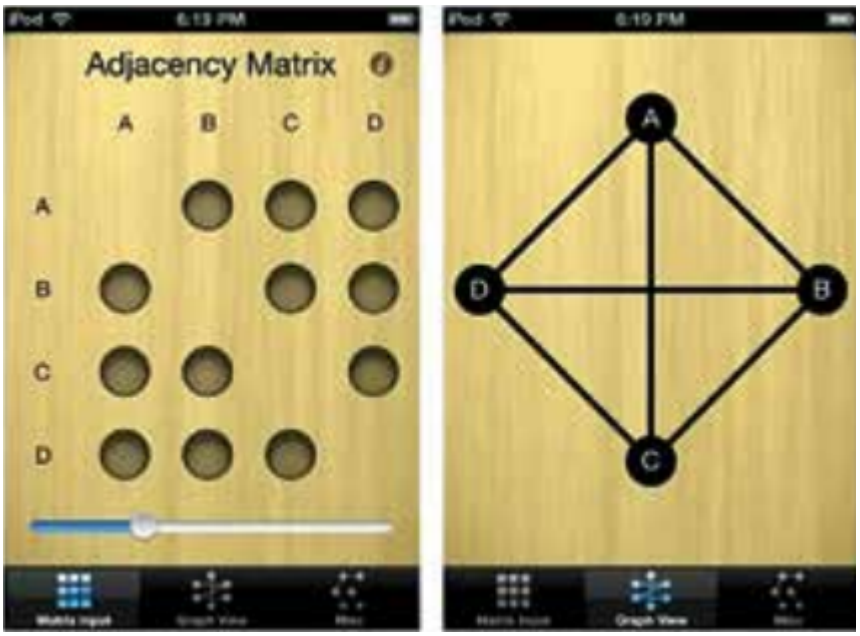
O estudo de Euler sobre as pontes de Königsberg solucionou não só o problema, mas também ajudou a criar a área da matemática de teoria dos grafos. A sua publicação

com o nome de "*Solutio Problematis ad Geometriam Situs Pertinentis* (A solução para um problema relacionado a Geometria de Posição)" e outros trabalhos futuros, serviram de base para as ideias essenciais da topologia combinatória (CARLSON, 2010).

A utilização de ferramentas de software como auxílio no ensino da Teoria dos Grafos já foi estudada por alguns autores. Por exemplo, no trabalho de Quinn (QUINN, 2014), a autora utiliza um aplicativo para celulares chamado Graphing para ajudar os alunos a visualizarem os grafos na sala de aula. Quinn exalta a natureza interativa que aplicativos como esse podem ser usados. Por exemplo, ao invés de permitir a criação manualmente dos grafos em uma folha de papel, o ensino com o Graphing obriga o aluno a criar corretamente a matriz de adjacências para visualizar o grafo resultante, conforme Figura 4.

Ferramentas de visualização de Grafos ajudam os professores na apresentação de um conteúdo mais profissional. E também ajudam os alunos a obterem um maior entendimento dos conceitos, através da visualização e experimentação (QUINN, 2014).

Figura 4 – Aplicativo Graphing



Fonte: (QUINN, 2014)

1.1 OBJETIVOS

Dentro deste contexto, para contribuir com o aprendizado da Teoria dos Grafos, este trabalho teve como objetivo criar uma ferramenta de visualização de algoritmos de grafos, ferramenta essa na qual fosse possível simular um pseudocódigo e visualizar no grafo o passo a passo do algoritmo. Com base no objetivo geral, foram elaborados os seguintes objetivos específicos:

1. Identificação dos principais algoritmos para visualização.
2. Detalhamento dos softwares existentes sobre visualização de algoritmos de grafos.
3. Desenvolvimento de um sistema para visualização de algoritmos de grafos, descrição dos recursos, da implementação e do funcionamento do programa.

1.2 ESTRUTURA DO TRABALHO

O trabalho está estruturado em cinco capítulos: introdução, importância e conceituação, ensino e ferramentas, desenvolvimento e considerações finais. Na introdução é feita uma contextualização sobre o que são grafos, sendo abordado um histórico sobre a origem do tema, acompanhado de exemplos de sua utilização. A introdução também explica e enumera os objetivos do trabalho.

No Capítulo 2, é explicada a relevância da teoria dos grafos na resolução de problemas em diferentes áreas do conhecimento, neste capítulo também é apresentado o funcionamento dos algoritmos escolhidos para desenvolvimento da aplicação.

No Capítulo 3, é feita uma análise sobre as ferramentas existentes que auxiliam o ensino da teoria dos grafos e são apresentados os softwares pesquisados que serviram como base para a ideia da aplicação.

O Capítulo 4 apresenta o desenvolvimento do sistema. Este capítulo é composto pelas telas do sistema, explicação das ferramentas escolhidas para o desenvolvimento do trabalho e pela descrição da implementação do software.

No Capítulo 5, são descritas as considerações finais deste trabalho, juntamente com as ideias para os projetos futuros.

2 IMPORTÂNCIA E CONCEITUAÇÃO

Este capítulo demonstra a importância e a utilidade da teoria dos grafos em situações do dia a dia através de exemplificações de seu uso em aplicações reais. A seção 2.1 cita exemplos de sua utilização em algumas áreas do conhecimento. Os algoritmos utilizados no desenvolvimento do trabalho também são apresentados nesse capítulo. A seção 2.3, demonstra o funcionamento dos algoritmos *Depth-first search* (DFS) e *Breadth First Search* (BFS) para buscas em grafos. A seção 2.4 mostra os algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall usados para calcular o caminho mínimo de um grafo. A seção 2.5, descreve os algoritmos de Prim e Kruskal responsáveis por descobrir a árvore geradora mínima de um grafo.

2.1 IMPORTÂNCIA E EXEMPLOS DE USO DA TEORIA DOS GRAFOS

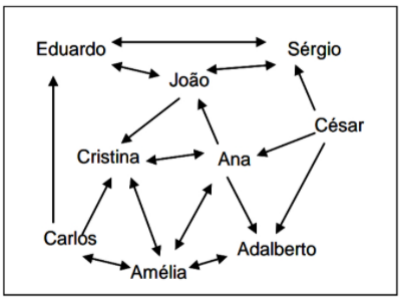
Atualmente a Teoria dos Grafos, é uma área de interesse constante, das quais tem finalidades em diversas áreas do conhecimento, desde problemas de localizações e de traçados de caminhos para distintas classes de serviços, até o planejamento de processadores eletrônicos, movendo-se pelo projeto de horários, ensino estrutural do DNA e pela elaboração de códigos. (NETTO; JURKIEWICZ, 2017). O campo de estudo da sociologia aplica os conceitos e representações de grafos nos estudos das relações interpessoais, no campo da engenharia elétrica grafos são usados na demonstração de circuitos elétricos, na área da linguística são frequentemente utilizados em analisadores sintáticos e em redes de computadores podemos citar redes conectadas de computadores.

Segundo (ROBERTS, 1978), ocasionalmente, a simples abstração de um assunto em grafos é válido para um bom entendimento do mesmo. No momento em que essa abstração é gerada através da linguagem teórica de grafos, é viável usar todos os conceitos dessa teoria no seu estudo. Euler utilizou essa abordagem para resolver o problema das Pontes de Königsberg. Portanto, a teoria dos grafos demonstra ser um elemento de ensino, onde através dele poderemos utilizá-lo como uma ferramenta de suporte para indicar novos caminhos e ações.

Um exemplo de utilização da teoria grafos é demonstrado em (GOLDBARG; GOLDBARG, 2017) para descrever um modelo de relacionamento de uma turma de alunos. Com o intuito de descobrir os relacionamentos existentes entre os estudantes da classe, uma professora solicitou aos educandos para que anotassem em um pedaço de papel, os colegas de aula por quem mais sentiam empatia. A partir dos papeis coletados, a

professora juntou todas as informações em um único diagrama, conforme Figura 5. Com base na análise do diagrama, a professora pode descobrir informações úteis sobre a turma de alunos, por exemplo, descobriu que o aluno César não foi citado por nenhum colega, e que existem dois grupos de alunos bastante unidos na classe, um de meninas, outro de meninos.

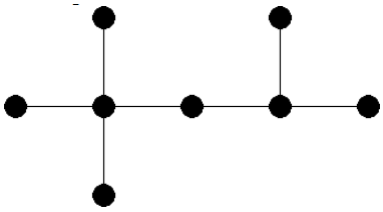
Figura 5 – Modelo de Relacionamentos de Alunos



Fonte: (NETTO; JURKIEWICZ, 2017)

A teoria dos grafos não é exclusivamente utilizada para resolver problemas inteiros, por vezes um fragmento de um vasto problema é abstraído e descobre-se que pode ser representado como um diagrama de grafo (ROBERTS, 1978). A Chemical Graph Theory é um ramo matemático utilizado na química matemática. Ramo que une a química com a teoria dos grafos, ela é empregada na representação de moléculas em modelos matemáticos, com o objetivo de coletar dados sobre as propriedades físicas de compostos químicos. A Figura 6 demonstra a utilização da teoria dos grafos na representação da árvore de carbono do trimetilpentano, (BURCH, 2006).

Figura 6 – Modelo matemático de árvore de carbono trimetilpentano



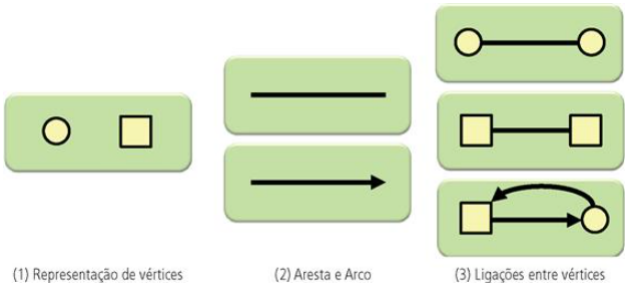
Fonte: (BURCH, 2006)

2.2 CONCEITOS INICIAIS

Um grafo é uma estrutura abstrata que contém dois conjuntos finitos de elementos denominados informalmente de vértices e arestas. Matematicamente, um grafo é usado para retratar as relações entre os elementos de um conjunto de objetos. Nessa modelagem os vértices são apresentados por pontos ou círculos. As arestas, relacionamentos entre um par de vértices, são usualmente ilustradas por linhas ou setas. A

Figura 7 exibe exemplos de representações de vértices, arestas e ligações entre vértices (GOLDBARG; GOLDBARG, 2017).

Figura 7 – Elementos de um grafo



Fonte: (GOLDBARG; GOLDBARG, 2017)

2.3 BUSCAS EM GRAFOS

Questões pertinentes a buscas em grafos são problemas que ocorrem com frequência na teoria dos grafos. O percurso em grafos pode ser utilizado para resolver totalmente ou apenas parte de um problema, com o objetivo de utilizar uma metologia correspondente ao navegar de um vértice ao outro. Deste modo, este capítulo apresenta dois métodos de busca: DFS e BFS.

> De modo geral, pode-se dizer que o projeto de bons algoritmos para a determinação de estruturas ou propriedades dos grafos depende do domínio de técnicas que permitam examinar com eficiência vértices e arestas. A esse tipo de procedimento denomina-se, genericamente, "busca em grafos" ou "percurso em grafos" (GOLDBARG; GOLDBARG, 2017, pp. 59).

Os algoritmos de busca em profundidade e busca em amplitude são parcialmente parecidos. Ambos algoritmos seguem a mesma lógica em suas execuções. Do mesmo modo, partilham dos mesmos objetivos, alcançar todos os vértices do grafo sem que seja feito caminhos com ciclos e visitas a vértices mais de uma vez.

2.3.1 Algoritmo de Busca em Profundidade

A ideia da DFS é a partir de um vértice inicial, escolher uma aresta e a partir dela percorrer o mais "fundo"possível todos os vértices acessíveis antes de verificar a próxima aresta (GOLDBARG; GOLDBARG, 2017).

Um pseudocódigo do algoritmo de busca em profundidade é mostrado no Algoritmo 1. O algoritmo inicia atribuindo como não visitado a todos os vértices do grafo, a seguir é escolhido como início da busca o vértice inicial do grafo e então

é executada a função DFS responsável pela busca. A função DFS recebe o vértice e o marca como visitado. Após isso, executa para todos os vértices não visitados e adjacentes a ele o método DFS.

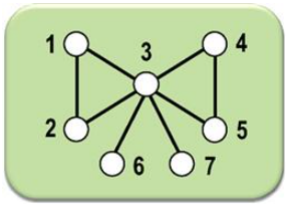
Algoritmo 1 – Pseudocódigo do algoritmo de Busca em Profundidade

```
1  DFS(s){
2      s.visitado = true
3      para cada v em G
4          se s.adj(v) && v.visitado == false
5              DFS(v)
6  }
7
8  main(){
9      para cada v em G
10         v.visitado = false
11     s = G(0)
12     DFS(s)
13 }
```

Fonte: O Autor (2021)

Em (GOLDBARG; GOLDBARG, 2017), Goldbarg cria um exemplo da aplicação do algoritmo de busca em amplitude no grafo da Figura 8. A Figura 9, Figura 10 e Figura 11 demonstram o funcionamento do algoritmo.

Figura 8 – Grafo exemplo para a busca em profundidade



Fonte: (GOLDBARG; GOLDBARG, 2017)

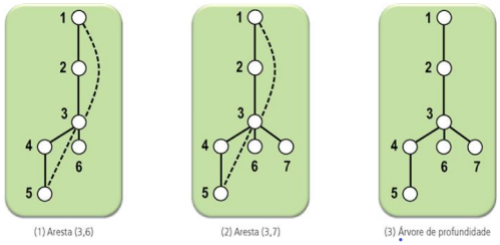
Figura 9 – Busca em profundidade em grafo - parte 1

Fonte: (GOLDBARG; GOLDBARG, 2017)

Figura 10 – Busca em profundidade em grafo - parte 2

Fonte: (GOLDBARG; GOLDBARG, 2017)

Figura 11 – Busca em profundidade em grafo - parte 3

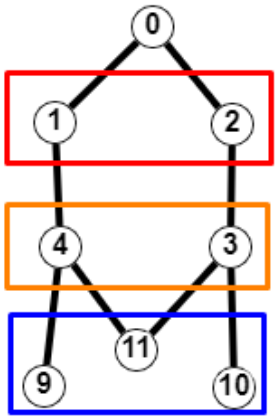


Fonte: (GOLDBARG; GOLDBARG, 2017)

2.3.2 Algoritmo de Busca em Amplitude

No algoritmo de busca em amplitude é realizada a busca de todos os elementos adjacentes ao vértice inicial antes de buscar os outros elementos. (GOLDBARG; GOLDBARG, 2017). A busca pode ser representada como uma busca em níveis, no grafo da Figura 12 pode ser vista a ordem em que os vértices são visitados, primeiramente são percorridos os vértices dentro do quadrado vermelho, após isso são encontrados os vértices do quadrado laranja e por fim os vértices do quadrado azul são marcados como visitado.

Figura 12 – Pseudocódigo do algoritmo de Busca em Amplitude



Fonte: O Autor (2021)

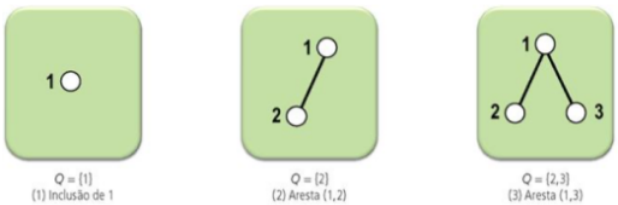
Algoritmo 2 – Pseudocódigo do algoritmo de Busca em Amplitude

```
1  BFS(s){
2      Q = nova fila
3      s.visitado = true
4      Q.coloca(s)
5      enquanto Q != vazio
6          v = Q.retirar_primeiro_elemento()
7          para cada e adjacente a v
8              se e.visitado == false
9                  Q.coloca(e)
10                 e.visitado = true
11             fimse
12         fimpara
13     fimenquanto
14 }
15
16 main(){
17     para cada v em G
18         v.visitado = false
19     fimpara
20     s = G(0)
21     BFS(s)
22 }
```

Fonte: O Autor (2021)

No Algoritmo 2 é mostrado um pseudocódigo do algoritmo de busca em amplitude. O pseudocódigo utiliza uma fila para realizar o percurso em níveis, de modo que a ordem em que os vértices são inseridos é a mesma em que são retirados. Na Figura 13, Figura 14 e Figura 15 é demonstrada a execução do algoritmo de busca em amplitude no grafo da Figura 8.

Figura 13 – Busca em amplitude em grafo - parte 1

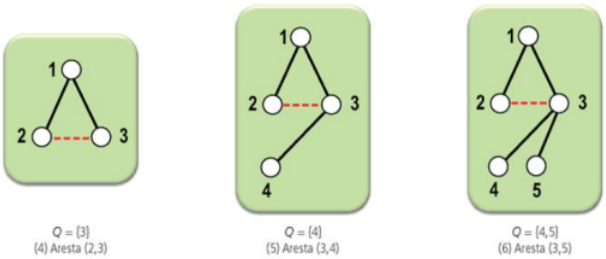


Fonte: (GOLDBARG; GOLDBARG, 2017)

2.4 CAMINHOS MÍNIMOS

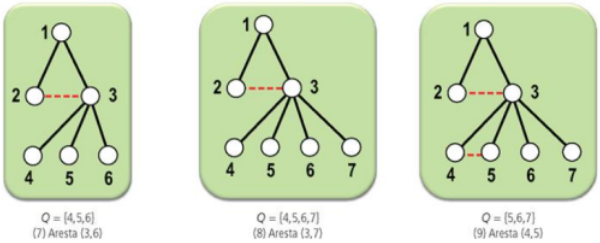
Na teoria dos grafos, problemas de caminhos mínimos são utilizados para descobrir o percurso de menor distância entre dois vértices, o cálculo do caminho mais

Figura 14 – Busca em amplitude em grafo - parte 2



Fonte: (GOLDBARG; GOLDBARG, 2017)

Figura 15 – Busca em amplitude em grafo - parte 3



Fonte: (GOLDBARG; GOLDBARG, 2017)

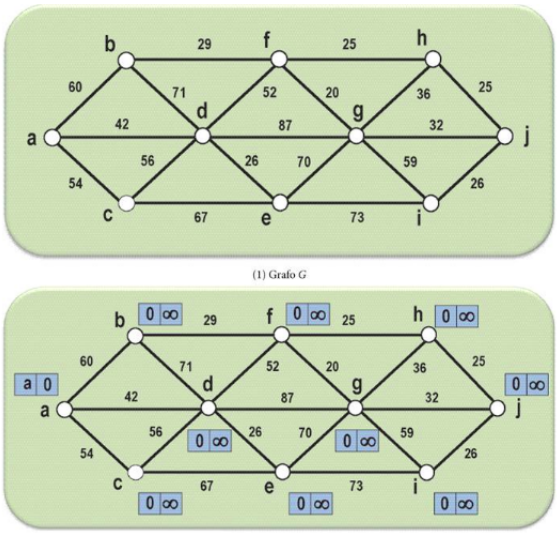
curto é realizado com base nos pesos das arestas de um grafo, se a soma das arestas de um caminho de i até j é a menor possível entre todos os caminhos possíveis, então esse percurso é considerado o caminho mínimo (GOLDBARG; GOLDBARG, 2017). A seguir são apresentados os algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshal aplicados para o cálculo do caminho mínimo.

2.4.1 Algoritmo de Dijkstra

Criado por Edsger Dijkstra em 1959, o algoritmo de Dijkstra é um dos principais algoritmos utilizados para resolver problemas de caminhos em grafos. Sendo apenas possível ser aplicado em grafos ponderados em que as arestas são positivas, um grafo é chamado ponderado se todas as suas arestas são representadas por um peso. Este algoritmo tem como objetivo encontrar o caminho mais econômico entre tais dois vértices (BALTAZAR; PEREIRA, 2018).

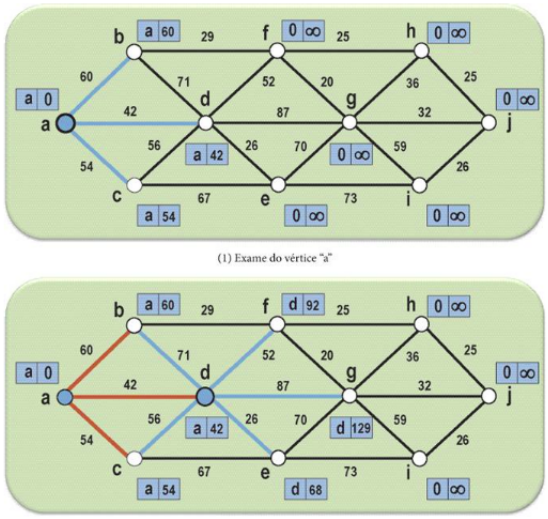
No algoritmo de Dijkstra enquanto houver algum vértice para o qual a distância mínima ainda não foi calculada, é escolhido dentre os disponíveis um vértice em que a distância calculada até o momento é mínima. Deste modo, a cada execução da busca, se for encontrado um caminho em que a soma das arestas for menor que a distância atual até o vértice buscado, o caminho até este vértice é atualizado com o novo valor. Na Figura 16 e na Figura 17 é apresentado alguns passos da execução do algoritmo de Dijkstra em um grafo de exemplo.

Figura 16 – Exemplo de execução do algoritmo de Dijkstra em grafo - Parte 1



Fonte: (GOLDBARG; GOLDBARG, 2017)

Figura 17 – Exemplo de execução do algoritmo de Dijkstra em grafo - Parte 2



Fonte: (GOLDBARG; GOLDBARG, 2017)

O Algoritmo 3 demonstra um pseudocódigo do algoritmo de Dijkstra. Seja G o grafo a ser percorrido, d é utilizado para armazenar a distância do vértice inicial até todos os outros, l é responsável pelo peso das arestas, s é o vértice inicial e t o vértice destino. O algoritmo inicia atribuindo infinito para todas as distâncias e 0 para a distância do vértice inicial. Q recebe o conjunto dos vértices em que a distância ainda não foi calculada, inicialmente são atribuídos todos os vértices do grafo. Enquanto Q for diferente de vazio, são realizados os passos a seguir:

1. u recebe um vértice em Q, no qual $d[u]$ é a distância mínima de d.
2. Se u for igual ao vértice destino t o algoritmo é finalizado.

3. Seja v o vértice adjacente a u, se v está em Q é realizado o passo seguinte, se não o passo 5 é executado.

4. Se a distância de v é maior que a distância de u somado ao peso da aresta atual, a distância de v é atualizada.

5. u é removido de Q.

Algoritmo 3 – Pseudocódigo do algoritmo de Dijkstra

```
1  Dijkstra(s){
2      para cada v em G
3          d[v] = infinito
4      fimpara
5      d[s] = 0
6      Q = V -- Conjunto de vértices que a distância ainda não foi calculada
7
8      enquanto Q != vazio
9          u = um vértice em Q, no qual d[u] é a distancia mínima
10         se u == t
11             return
12
13         para cada v adjacente a u
14             se v está em Q
15                 e = aresta(v,u)
16                 se d[v] > d[u] + l[e]
17                     d[v] = d[u] + l[e]
18                 fimse
19             fimse
20         fimpara
21
22         Q.remove(u)
23     fimenquanto
24 }
```

Fonte: O Autor (2021)

2.4.2 Algoritmo de Bellman-Ford

Semelhante ao algoritmo de Dijkstra, o algoritmo de Bellman-Ford tem como objetivo encontrar o menor caminho de um vértice s até todos os outros vértices de um grafo. Bellman-Ford ou Ford-Moore-Bellman foi o nome reconhecido ao algoritmo devido às contribuições dos pesquisadores Lester Ford, Edward Moore e Richard Bellman. Em seus trabalhos publicados, os pesquisadores diferem do algoritmo de Dijkstra que após selecionado um vértice, não calcula novamente a sua distância, por outro lado o algoritmo de Bellman-Ford nunca determina que a menor distância foi atingida. Com

essa abordagem, é possível calcular caminhos mínimos em grafos que tenham arestas com pesos negativos (GOLDBARG; GOLDBARG, 2017).

O Algoritmo 4 apresenta um pseudocódigo do algoritmo, a estrutura é semelhante ao algoritmo de Dijkstra. O algoritmo inicia atribuindo infinito para as distâncias d de todos os vértices, e nulo para p, vetor responsável por guardar a informação do vértice pai, a seguir na linha 6 a distância do vértice inicial é setada para 0.

No primeiro laço "para"a partir da linha 8, é feita uma quantidade de iterações igual ao número de vértices-1. A seguir, para cada aresta (u, v), se a distância de v ($d[v]$) é menor que a distância de u ($d[u]$) somado ao peso da aresta (u, v), então a distância de v é atualizada e u é atribuído como pai de v. O caminho mínimo do grafo é possível obter a partir do vetor p, pois p guarda a informação de onde cada vértice foi alcançado.

Algoritmo 4 – Pseudocódigo do algoritmo de Bellman-Ford

```
1  BelmannFord(s){
2      para cada v em G
3          d[v] = infinito
4          p[v] = null
5      fimpara
6      d[s] = 0
7
8      para i de 1 ate V-1
9          para cada aresta (u,v) em G
10             tempDistancia = d[u] + peso_aresta(u,v)
11             se tempDistancia < d[v]
12                 d[v] = tempDistancia
13                 p[v] = u
14             fimse
15         fimpara
16     fimpara
17
18     para cada aresta (u,v) em G
19         se d[u] + peso_aresta(u,v) < d[v]
20             return "Há ciclos negativos"
21         fimse
22     fimpara
23 }
```

Fonte: O Autor (2021)

2.4.3 Algoritmo de Floyd-Warshall

Publicado por Robert Floyd em 1962 o algoritmo de Floyd-Warshall, foi fundamentado no algoritmo de Stephen Warshall utilizado para calcular fechos transitivos em grafos, o algoritmo de Floyd-Warshall busca encontrar o caminho mínimo entre

todos os pares de vértices em um grafo não esparso (GOLDBARG; GOLDBARG, 2017). O pseudocódigo do algoritmo demonstrado no Algoritmo 5 utiliza a matriz chamada *dist* para guardar as informações dos menores caminhos, inicialmente *dist* recebe a matriz de adjacências do grafo e atribui infinito para as posições em que a aresta (i, j) não existe no grafo. O cálculo do caminho mínimo acontece a partir da linha 13, a cada passo é calculado para cada par de vértices (i, j) o custo do menor caminho que passa por vértices com numeração igual ou menor a k. Para identificar o caminho mínimo é utilizada a matriz p responsável por armazenar em cada posição (i, j) o último vértice acessado antes de j.

Algoritmo 5 – Pseudocódigo do algoritmo de Floyd-Warshall

```
1  FloydWarshall(){
2      para i de 1 até N
3          para j de 1 até N
4              se existe aresta(i,j)
5                  dist(i,j) = peso_aresta(i,j)
6                  p(i,j) = i
7              senao
8                  dist(i,j) = infinito
9              fimse
10         fimpara
11     fimpara
12
13     para k de 1 ate N
14         para i de 1 ate N
15             para j de 1 ate N
16                 se dist(i,j) > dist(i,k) + dist(k,j)
17                     dist(i,j) = dist(i,k) + dist(k,j)
18                     p(i,j) = p(k,j)
19                 fimse
20             fimpara
21         fimpara
22     fimpara
23 }
```

Fonte: O Autor (2021)

2.5 ÁRVORE GERADORA MÍNIMA

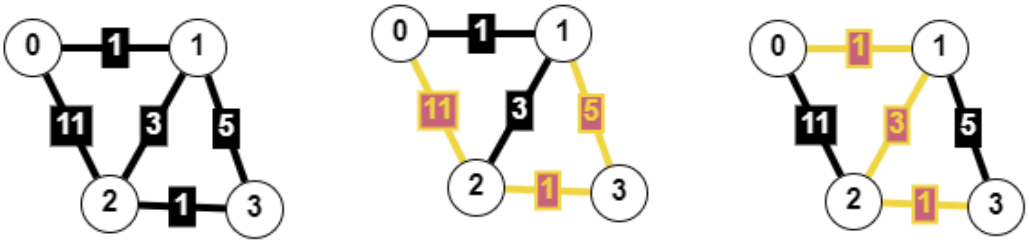
Segundo GOLDBARG & GOLDBARG (2017), uma árvore é um grafo conexo[1], acíclico[2] e com somente um trajeto para quaisquer dois vértices de um grafo.

[1] Grafo no qual existe um caminho para qualquer par de vértices.
[2] Grafo sem ciclos.

> As árvores são uma das mais importantes estruturas da Teoria dos Grafos, encontrando aplicações na formação das estruturas de dados e na solução de grande número de problemas reais. Os modelos vão desde as estruturas de dados na Ciência da Computação até problemas de comunicação (GOLDBARG; GOLDBARG, 2017, pp. 128).

Uma árvore geradora mínima é o caminho que apresenta a menor soma de pesos associados às arestas de um grafo, de modo que o percurso seja feito passando por todos os vértices do grafo. A Figura 18 demonstra um exemplo de árvore geradora mínima. A árvore geradora apresentada na Figura 18b não é mínima, tem custo 17, enquanto que a árvore geradora da Figura 18c é mínima, pois tem custo 5 e não existe outra árvore geradora com soma menor que ela.

Figura 18 – Exemplo de árvore geradora mínima



(a) Grafo de exemplo

(b) Árvore geradora não mínima

(c) Árvore geradora mínima

Fonte: O Autor (2021)

2.5.1 Algoritmo de Prim

Criado por Robert C. Prim em 1957, o algoritmo de Prim se caracteriza pela construção iterativa da solução, a cada passo um vértice é escolhido para ser incluído na árvore geradora (GOLDBARG; GOLDBARG, 2017). O pseudocódigo do algoritmo de Prim é demonstrado no Algoritmo 6. Seja V o conjunto dos vértices do grafo, U o conjunto de vértices selecionados que fazem parte da árvore geradora e T o conjunto de arestas da árvore geradora. Enquanto o conjunto V for diferente do conjunto U, ou seja, enquanto todos os vértices ainda não foram selecionados, são realizados os seguintes passos:

1. É escolhida a aresta (u, v) de menor valor, no qual u está em U e v está em V.
2. É adicionado o vértice v ao conjunto U.
3. A aresta (u, v) é adicionada a T, conjunto da árvore geradora mínima.

Algoritmo 6 – Pseudocódigo do algoritmo de Prim

```
1  Prim(s){
2      para cada v em V
3          d[v] = infinito
4      fimpara
5      d[s] = 0
6      para i de 0 até V.tamanho
7          u = um vértice em V, no qual d[u] é a distância mínima
8          U.coloca(u)
9          para cada v adjacente a u
10             se U.nao_existe(v) && peso_aresta(u,v) < d[v]
11                 d[v] = peso_aresta(u,v); pai[v] = u;
12             fimse
13         fimpara
14     fimpara
15 }
```

Fonte: O Autor (2021)

2.5.2 Algoritmo de Kruskal

O algoritmo de Kruskal, criado por Joseph B. Kruskal em 1956, ao contrário do algoritmo de Prim, tem como objetivo encontrar a árvore geradora mínima através da inserção de arestas de um grafo. Enquanto a árvore não é finalizada, ela é de fato uma floresta[3], visto que para cada aresta inserida não é obrigatória estar conectada a outra aresta da árvore (GOLDBARG; GOLDBARG, 2017).

O Algoritmo 7 apresenta um pseudocódigo do algoritmo de Kruskal, seja E as arestas do grafo, V os vértices do grafo e T o conjunto de arestas da árvore geradora mínima, o primeiro passo do algoritmo é ordenar de forma ascendente todas as arestas do grafo pelo seu peso. A seguir para cada aresta na lista de arestas ordenadas, se a adição da aresta em T não formar um ciclo, a aresta é adicionada em T. Caso o tamanho de T for igual a $V - 1$, o algoritmo é encerrado pois a árvore geradora mínima foi encontrada.

Algoritmo 7 – Pseudocódigo do algoritmo de Kruskal

```
1  Kruskal(){
2      listaAresta = OrdenarArestasPorPeso(E)
3      T = {}
4      para i de 0 ate listaAresta.tamanho
5          e = listaAresta[i]
6          se nao_forma_ciclo(T,e)
7              T.Adiciona(e)
```

[3] Conjunto de árvores sem vértice compartilhado

```
8                se T.tamanho == V.tamanho-1
9                    return
10               fimse
11           fimse
12       fimpara
13   }
```

Fonte: O Autor (2021)

3 ENSINO E FERRAMENTAS DE VISUALIZAÇÃO

As atividades educacionais em sala de aula demandam que o docente desenvolva cenários nos quais os estudantes consigam passar pelo processo de ensino e aprendizagem, tendo como missão, proporcionar aos alunos a capacidade de definir conceitos matemáticos, que colaboram para a formação do pensamento crítico diante dos conteúdos abordados na escola. (SILVA; RODRIGUES, 2015). Além disso, segundo Silva & Rodrigues (2015), os professores devem incluir em suas aulas a junção de divergentes temas e conceitos que favoreçam a construção do raciocínio matemático. O uso de ferramentas de visualização tem o intuito de favorecer a compreensão de conceitos e métodos abstratos, de modo a tornar mais "concreto"o que é abstrato.

3.1 FERRAMENTAS

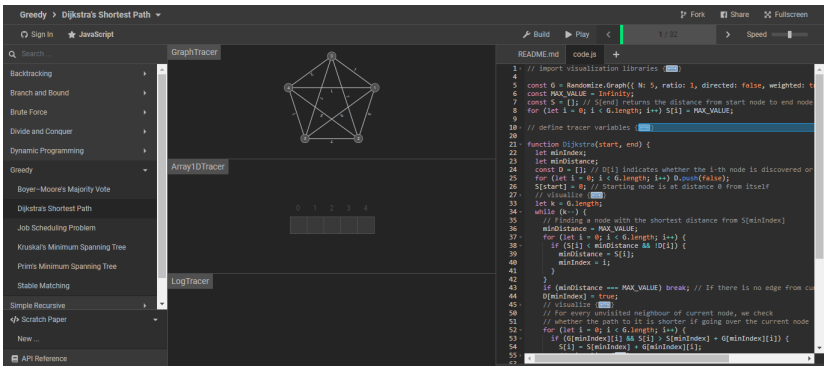
Com objetivo de auxiliar professores e alunos na passagem de conhecimento, existe atualmente uma notável quantidade de ferramentas para a visualização de algoritmos de grafos. Neste capítulo veremos algumas ferramentas existentes sobre o assunto, no fim do capítulo é disponibilizado um quadro comparativo sobre os recursos disponíveis em cada programa.

3.1.1 Algorithm Visualizer

Algorithm Visualizer é um software *Open Source*[1] que apresenta uma diversa gama de algoritmos para simulação e dentre eles estão certos algoritmos de grafos. Na aplicação o algoritmo é executado e o seu estado pode ser visto através de um grafo disponibilizado no centro da tela, como observado na Figura 19. Em alguns algoritmos outras funcionalidades também são apresentadas, por exemplo, na simulação do algoritmo de Dijkstra um vetor com as distâncias é mostrado e atualizado a cada passo da execução. Entretanto, se o software for usado como uma ferramenta educacional, ele apresenta algumas deficiências. O sistema não permite o usuário criar o próprio grafo ou alterar o grafo apresentado, o usuário é então obrigado a assistir a simulação no grafo desenvolvido pela aplicação. Além disso, não é possível acompanhar no código o passo a passo do algoritmo, a aplicação não destaca qual parte está sendo executada. Por fim, o código fornecido é escrito em uma linguagem específica, podendo ser considerada limitada segundo Mocinecová & Steingartner (2020, pp.15).

[1] Software de Código Aberto

Figura 19 – Impressão da execução do algoritmo de Dijkstra na ferramenta *Algorithm Visualizer*



Fonte: O Autor (2021)

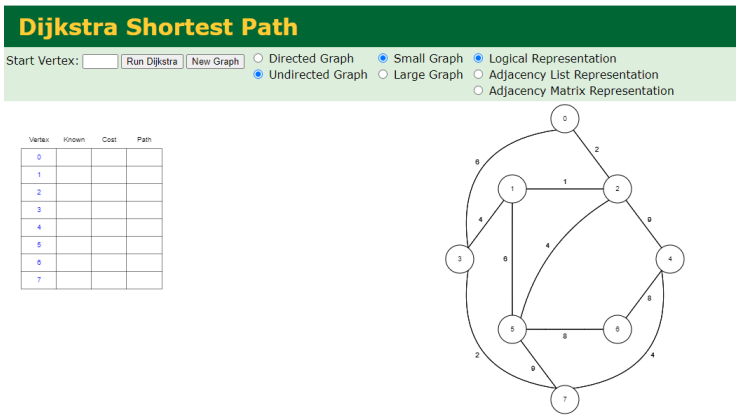
3.1.2 Data Structure Visualization

Outro sistema que abrange o ramo da teoria dos grafos é o *Data Structure Visualization*, uma ferramenta gratuita que possibilita aos usuários a visualização de diferentes estruturas de dados e algoritmos. O sistema apresenta uma lista de possibilidades, desde a representação de vetores, pilhas e filas, até algoritmos de grafos. O software apresenta a simulação dos algoritmos, BFS, DFS, Dijkstra, Floyd-Warshall, Prim e Kruskal, estudados nesse projeto. A execução depende do tipo de algoritmo ou estrutura de dados escolhida, porém algumas ações aparecem em sua grande maioria: O sistema permite controlar o estado do algoritmo, através dos botões voltar, avançar, pausar e até escolher a velocidade em que a simulação é demonstrada (FADZILAH; NURKALIZA; FARHANA, 2016, pp. 29). Na Figura 20 pode ser visto um exemplo da animação feita para o algoritmo de Dijkstra. Diferente do *Algorithm Visualizer* é possível escolher até dois grafos diferentes para serem visualizados, porém semelhante ao *Algorithm Visualizer* não é possível criar o seu próprio exemplo, podendo ocorrer o caso de nenhum dos dois grafos disponibilizados suprirem as necessidades do usuário (MOCINECOVá; STEINGARTNER, 2020).

3.1.3 VisuAlgo

Criado em 2011, VisuAlgo é uma ferramenta que tem um intuito semelhante à *Data Structure Visualization*, auxiliar estudantes e professores no ensino de estruturas de dados e algoritmos, permitindo aos estudantes aprenderem no seu próprio ritmo (FADZILAH; NURKALIZA; FARHANA, 2016, pp.29). Segundo o criador Dr. Steven Halim (SCIENCE, 2020), a ideia do sistema surgiu a partir de uma pesquisa sobre os softwares para animações de algoritmos existentes na época, existiam alguns sites que disponibilizavam alguns algoritmos, porém era improvável encontrar páginas da internet que apresentavam algoritmos menos conhecidos, além de ser inconveniente para os alunos

Figura 20 – Impressão da execução do algoritmo de Dijkstra na ferramenta *Data Structure Visualization*



Fonte: O Autor (2021)

de terem que procurar em diversas fontes. Alguns outros problemas foram citados pelo Dr.Halim: A dificuldade dos seus alunos para conseguir anotar os exemplos apresentados no quadro durante a aula e também a falta de tempo dele para apresentar aos seus educandos exemplos fora dos descritos em livros ou slides, visto que o tempo para apresentação do conteúdo em sala de aula era limitado (SCIENCE, 2020),.

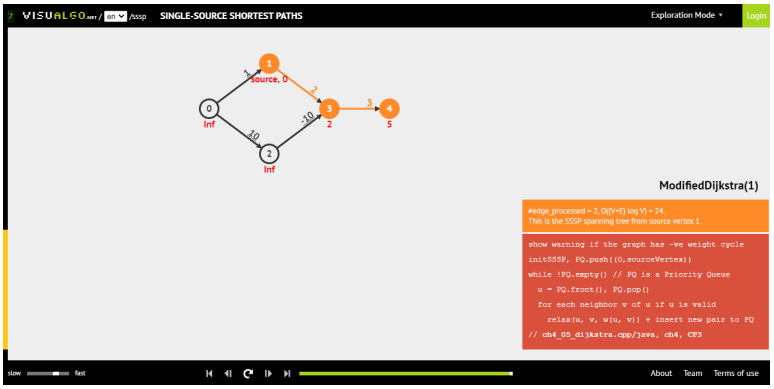
O VisuAlgo disponibiliza para visualização 23 tipos diferentes de algoritmos, dentre eles alguns algoritmos de grafos (SCIENCE, 2020), a Figura 21 apresenta um *screenshot* do algoritmo de Dijkstra sendo executado. O diferencial do sistema são suas funcionalidades, pois além de realizar as animações, o sistema permite ao usuário acompanhar o passo a passo de um pseudocódigo junto a sua execução, e uma pequena descrição do passo atual é mostrada. A aplicação também permite a criação dos próprios exemplos e oferece opções prontas para simulação do algoritmo. Entretanto, segundo Fadzilah, Nurkaliza & Farhana (2016, pp.29) o lado negativo é que o VisuAlgo não tem um bom funcionamento em telas *touchscreen* pequenas, pois para o desenvolvimento das animações foi utilizado um grande número de pixels.

3.1.4 D3 Graph Theory

D3 Graph Theory é um projeto *front-end*, criado em 2017 por Avinash Pandey, que tem como objetivo ensinar conceitos introdutórios da teoria dos grafos de uma forma interativa e com um visual agradável ao leitor (PANDEY, 2017). Em cada tópico o sistema apresenta uma breve descrição do conteúdo, juntamente com uma área em que é possível realizar a simulação do assunto abordado através de um grafo. Os temas disponibilizados pelo autor são citados no Quadro 1.

Na Figura 22 pode ser visto um exemplo da apresentação do conteúdo sobre

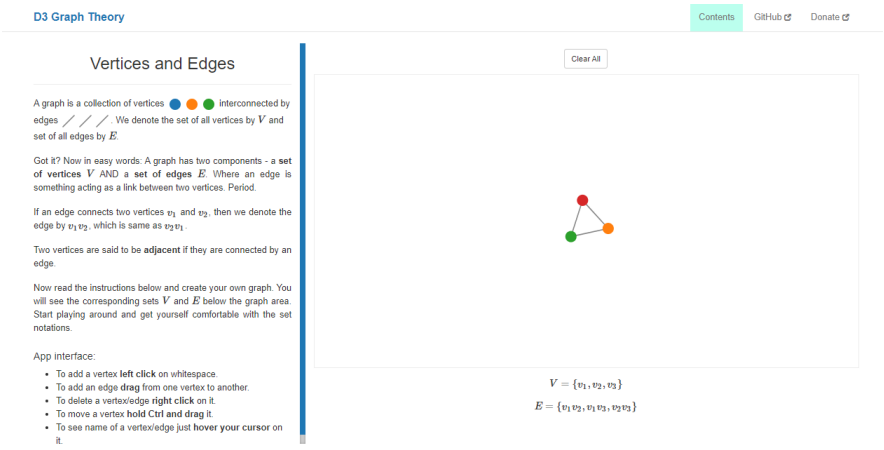
Figura 21 – Impressão da execução do algoritmo de Dijkstra na ferramenta VisuAlgo



Fonte: O Autor (2021)

graus em vértices[2]. Neste exemplo o grafo inicial é constituído de 5 vértices e em cada um é possível ver o grau referente a ele. O diferencial da aplicação é a possibilidade de atualizar o grafo de uma forma dinâmica, sendo possível adicionar vértices através de um click do mouse, enquanto que as arestas são formadas selecionando um vértice inicial e arrastando até o vértice destino. Toda vez que uma aresta é adicionada o grau dos vértices adjacentes[3] é atualizado. O programa também permite a exclusão de um vértice com um click do botão direito do mouse, caso este tenha alguma aresta associada a aresta também é removida. Segundo o autor, novos temas serão adicionados, porém até o momento nenhum algoritmo estudado nesse trabalho foi implementado pelo software.

Figura 22 – Explicação do conteúdo de graus em vértices no software D3 Graph Theory



Fonte: O Autor (2021)

[2] O grau de um vértice é calculado a partir da quantidade de vértices adjacentes a ele
[3] Dois vértices são considerados adjacentes ou vizinhos se estão conectados por uma aresta.

Quadro 1 – Quadro com os conteúdos abordados no sistema D3 Graph Theory

Índice	Conteúdos Abordados
1	*Vertices and Edges*
2	*Order and Size of a Graph*
3	*Degree of a Vertex*
4	*Degree Sequence of a Graph*
5	*Graphic Sequence*
6	*Havel-Hakimi Algorithm*
7	*Pigeonhole Principle*
8	*Regular Graph*
9	*Complete Graph*
10	*Bipartite Graph*
11	*Complete Bipartite Graph*
12	*Walk*
13	*Open vs Closed Walks*
14	*Connectivity*
15	*Eulerian Circuit*
16	*Eulerian Trail*
17	*Graph Coloring*
18	*k-Colorable Graph*
19	*Chromatic Number*
20	*Trees*
21	*Rooted Trees*
22	*Spanning Tree of a Graph*

Fonte: Adaptado de (PANDEY, 2017)

3.1.5 GraphViz

O software criado por Mocinecová & Steingartner (2020) tem o intuito de ser uma ferramenta de aprendizagem, segundo os autores o projeto tem a finalidade de proporcionar aos usuários uma forma simples e fácil de entender os algoritmos de grafos abordados, isso ocorre pois o programa possibilita a visualização da execução do algoritmo.

Mediante um contato feito com o autor William Steingartner, foi disponibilizado pelo mesmo o software GraphViz para testes. O ambiente do sistema é dividido em 3 seções, sendo elas:

- Área de Desenho
- Algoritmos
- Exemplos de Grafos

A área de desenho é responsável pela criação e visualização do estado do algoritmo. É possível criar vértices clicando com o botão esquerdo do mouse em qualquer lugar da tela, já as arestas são criadas após clicar duplamente em um vértice e clicar novamente duas vezes no vértice destino. Ao clicar com o botão direito em cima de uma aresta, algumas dessas opções podem ser feitas: A adição de um valor à aresta, troca de direção da aresta, mudança de aresta para uma aresta não dirigida ou dirigida, ou remoção da aresta. A remoção de um vértice implica na remoção de todas suas arestas adjacentes. Na área de desenho também é possível informar se o grafo é dirigido[4] ou não dirigido e se ele é ponderado[5]. A Figura 23 apresenta a seção.

Figura 23 – Seção *Drawing Area* do software GraphViz

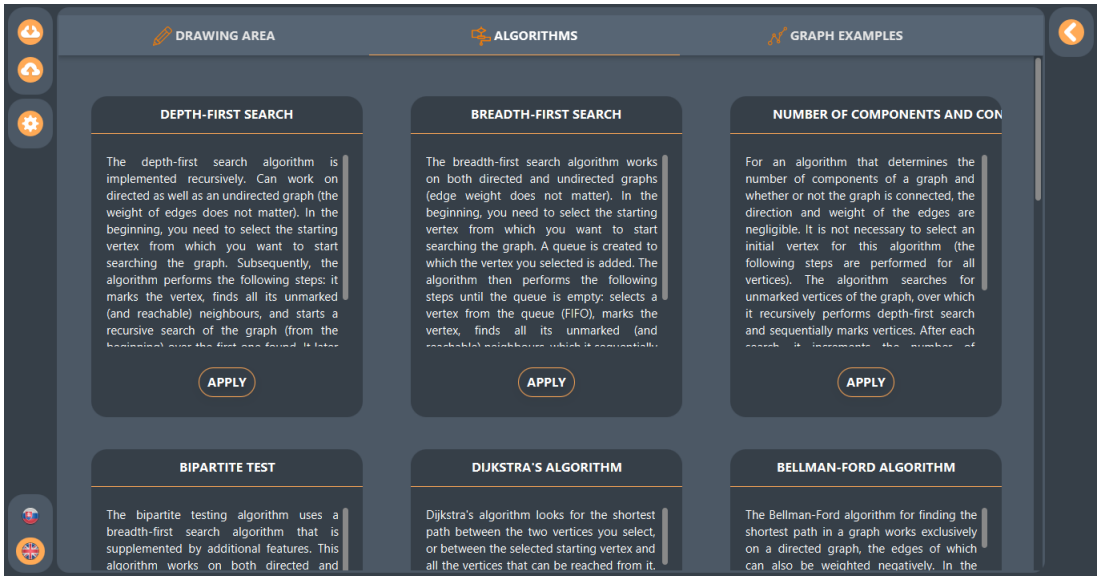
Fonte: O Autor (2021)

Toda a configuração de visualização do algoritmo é disponibilizada nessa área, é possível iniciar, pausar e pular etapas. O tempo em segundos entre cada etapa também pode ser configurado. Porém, a execução só pode ser iniciada após a criação ou seleção de um grafo e o algoritmo ser escolhido na seção seguinte.

A seção Algoritmos apresenta uma lista dos algoritmos possíveis de visualizar através do software, sendo eles: DFS, BFS, teste de grafos bipartidos, identificação de componentes, algoritmo de Dijkstra, algoritmo de Bellman-Ford, algoritmo de Floyd-Warshall, teste de isomorfismo em grafos, verificação de grafos eulerianos e hamiltonianos, algoritmo de Kruskal, algoritmo de Jarník, e identificação de caminho crítico (MOCINECOVá; STEINGARTNER, 2020). Todos são apresentados com uma breve descrição sobre o algoritmo e seu funcionamento. Na Figura 24 pode ser visto um *screenshot* da seção do programa.

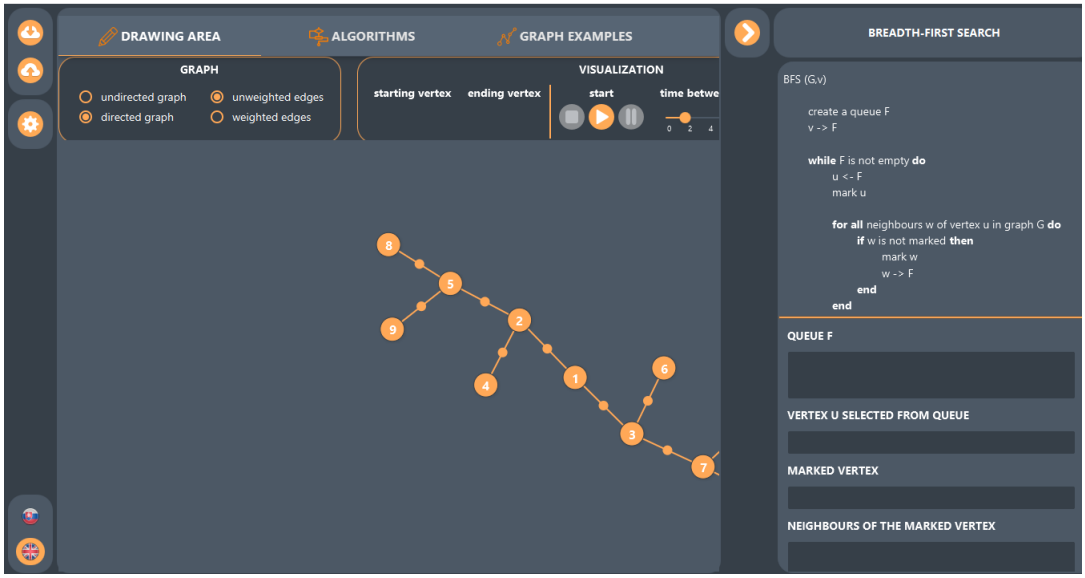
[4] Quando as arestas têm direção.
[5] Quando suas arestas possuem um valor.

Figura 24 – Seção *Algorithms* do software GraphViz



Fonte: O Autor (2021)

Figura 25 – Aba do algoritmo selecionado no software GraphViz

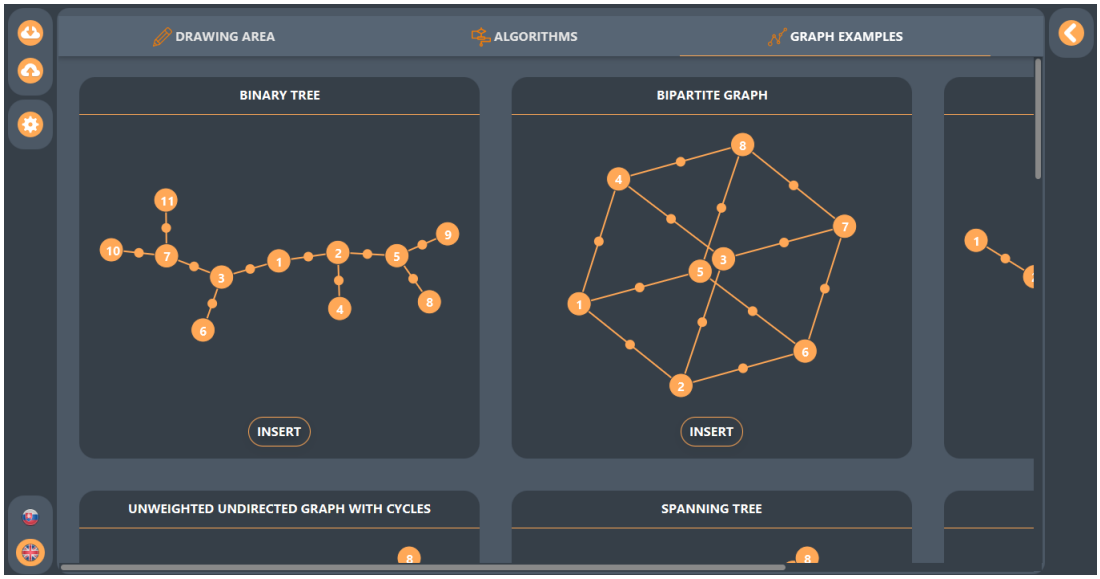


Fonte: O Autor (2021)

Na seção Exemplos de Grafos uma lista de grafos é disponibilizada, sendo possível o usuário escolher algum deles para a visualização do algoritmo, a Figura 26 demonstra essa seção do programa.

Após o algoritmo ser selecionado, uma aba contendo um pseudocódigo do algoritmo é aberta (Figura 25) e informações referente ao tipo do algoritmo são exibidas e então pode ser acompanhado tanto no pseudocódigo, quanto no grafo o passo a passo da execução. O programa possibilita algumas outras opções: O idioma da aplicação pode ser trocado para eslovaco ou inglês, além disso é permitido importar e exportar

Figura 26 – Seção *Graph Examples* do software GraphViz



Fonte: O Autor (2021)

grafos no sistema sendo esses armazenados no formato *Extensible Markup Language* (XML).

3.2 CONSIDERAÇÕES FINAIS

Com base nas ferramentas pesquisadas e apresentadas neste capítulo, a relação com os recursos disponíveis em cada ferramenta é apresentado no Quadro 2. A coluna Criação apresenta as ferramentas que possibilitam a criação manual do grafo. As colunas Simulação e Visualização são respectivamente referentes à simulação do algoritmo e à visualização do código ou pseudocódigo. A coluna Descrição aponta as ferramentas que detalham o algoritmo. A coluna Campos Adicionais retrata se a ferramenta apresenta outras informações sobre o algoritmo.

Quadro 2 – Quadro com os recursos disponíveis em cada ferramenta pesquisada

Ferramenta	Criação	Simulação	Visualização	Descrição	Campos Adicionais
Algorithm Visualizer		X	X		
Data Structure Visualization		X			X
VisuAlgo	X	X	X	X	
D3 Graph Theory	X			X	
GraphViz	X	X	X	X	X

Fonte: O Autor (2021)

4 DESENVOLVIMENTO DO SISTEMA

Por meio do estudo realizado e apresentado no capítulo anterior, identificou-se algumas ferramentas existentes sobre o tema de visualização de algoritmos de grafos. Este trabalho então teve como objetivo inspirar-se nos recursos apresentados no Quadro 2. Neste capítulo é apresentado como foi realizado o desenvolvimento do projeto.

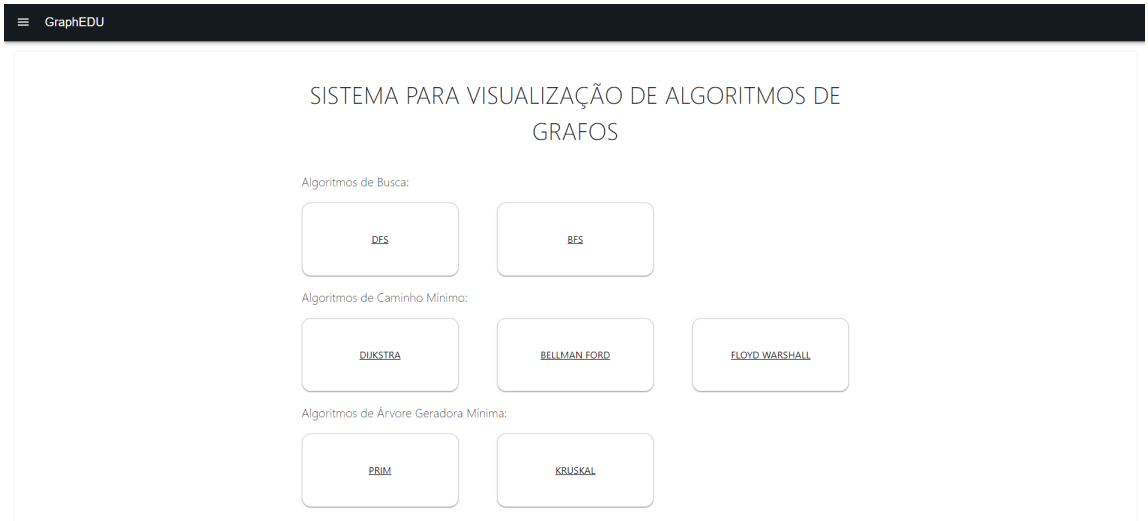
4.1 FUNCIONALIDADES DA FERRAMENTA

Neste trabalho foi realizado o desenvolvimento de uma aplicação no formato web, que tem o intuito de facilitar a utilização do sistema pelos os usuários, a aplicação permite o usuário construir o grafo graficamente, visualizar um pseudocódigo do algoritmo selecionado, visualizar a execução do algoritmo passo a passo através da simulação no grafo e pelos dados do algoritmo, e também a possibilidade de importar e exportar o grafo através de um arquivo *JavaScript Object Notation* (JSON).

4.2 TELAS DO SISTEMA

A tela inicial do sistema contém as opções dos algoritmos descritos no Capítulo 2, ela apresenta opções de seleção dos algoritmos de busca, de caminho mínimo e de árvore geradora mínima, visto na Figura 27. O *template* foi baseado no menu inicial do sistema VisuAlgo apresentado na Seção 3.1.3, que utiliza um layout retangular para apresentar cada algoritmo do seu sistema.

Figura 27 – Página Inicial do GraphEDU

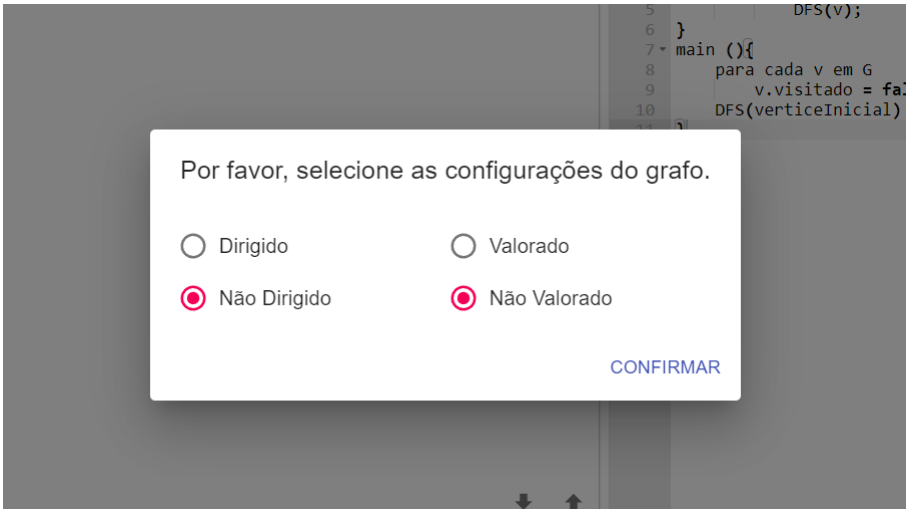


Fonte: O Autor (2021)

Após o algoritmo ser escolhido, a tela de visualização é apresentada. No exemplo atual, o algoritmo escolhido foi o DFS, após o carregamento da página é apresentado ao usuário uma modal para a escolha das configurações do grafo, mostrado na Figura 28. O usuário poderá escolher entre Dirigido/Não Dirigido e Valorado/Não Valorado.

Para fim de facilitar a explicação, a tela será dividida em 5 partes e cada uma será detalhada, a Figura 29 apresenta a página do algoritmo DFS. As seguintes áreas compõem a tela de visualização:

Figura 28 – Opções do Algoritmo DFS



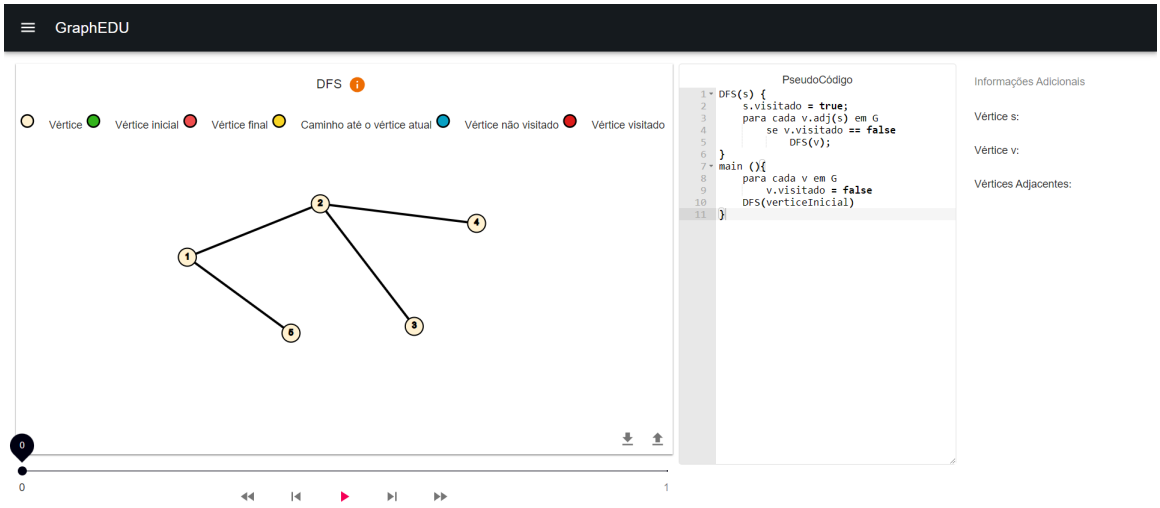
Fonte: O Autor (2021)

1. Descrição do Algoritmo.
2. *Canvas*.
3. Apresentação do Pseudocódigo.
4. Campos Relacionados ao Algoritmo.
5. Componente de Simulação.

O campo descrição do algoritmo tem como intuito apresentar uma breve definição do algoritmo, nele é descrito como o algoritmo é implementado e o passo a passo da execução. O objetivo é centralizar as informações para o usuário, de modo que não seja preciso consultá-las em outros lugares. Além disso, é apresentada uma descrição da utilização do sistema, na qual é descrito como é feita a inserção dos vértices e das arestas, e como a execução é iniciada. A descrição do algoritmo pode ser visualizada através do clique no botão, em laranja, ao lado do nome do algoritmo, Figura 30.

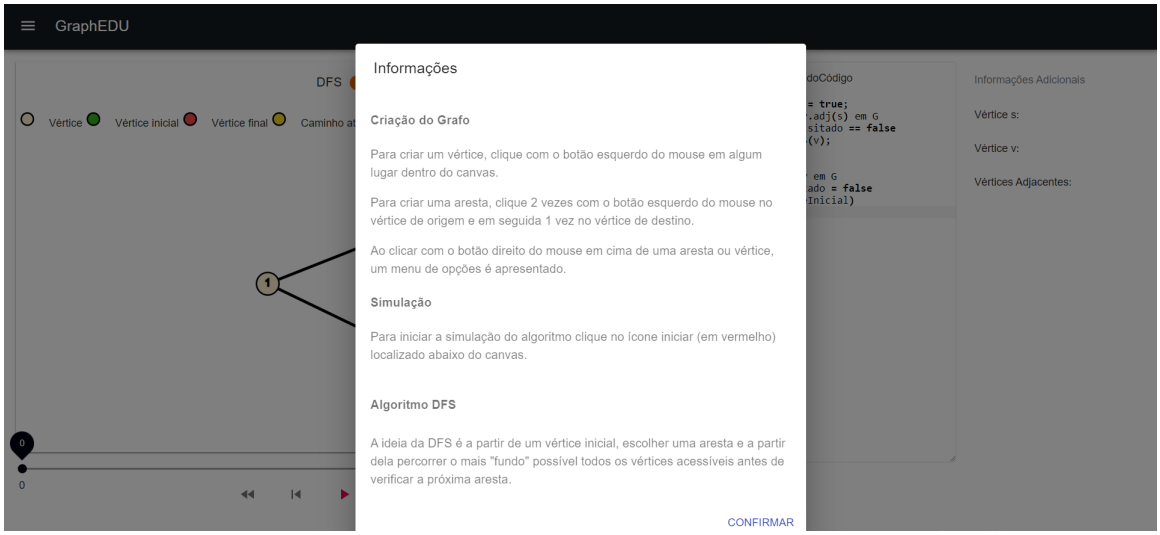
O *Canvas* é a área principal da tela de visualização, na qual será criado o grafo do algoritmo. Os vértices e as arestas serão criados a partir da interação do usuário

Figura 29 – Página do Algoritmo DFS do GraphEDU



Fonte: O Autor (2021)

Figura 30 – Informações do Algoritmo DFS do GraphEDU

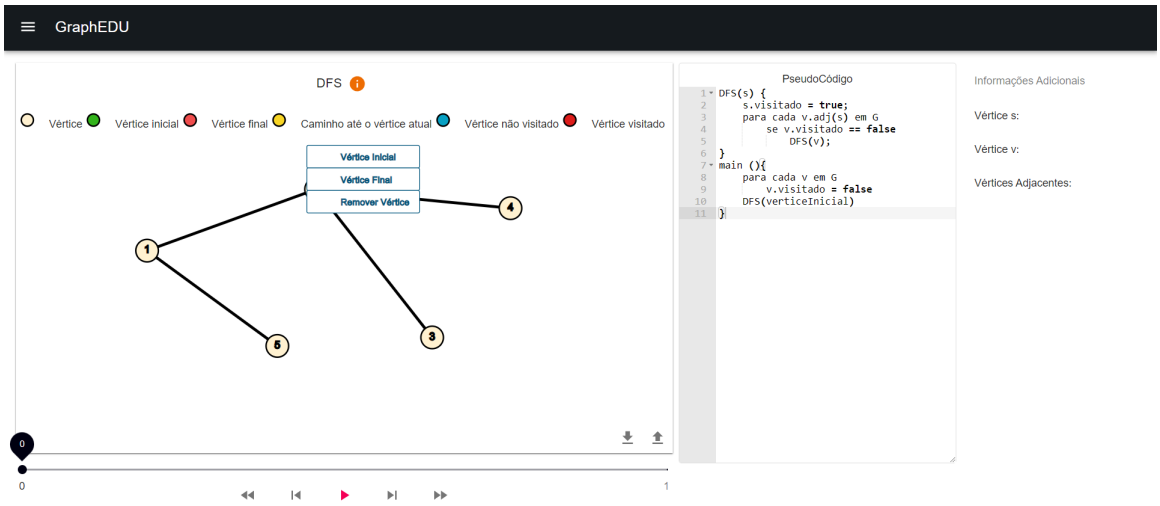


Fonte: O Autor (2021)

com essa área central, em todos os sistemas estudados a criação dos vértices são feitas de forma semelhante, após um *click* com o botão esquerdo do mouse em algum local dentro do *Canvas*, este sistema seguiu o mesmo padrão. Já para as arestas foi constatada a geração de maneiras diferentes, selecionando o vértice inicial e arrastando até o vértice destino, observado nos sistemas VisuAlgo (Seção 3.1.3) e D3 Graph Theory (Seção 3.1.4) apresentados no Capítulo 3, ou pressionando duas vezes o botão esquerdo do mouse no vértice inicial e em seguida pressionando novamente duas vezes no vértice de destino, como visto no sistema GraphViz (Seção 3.1.5). Neste sistema para realizar a criação das arestas é necessário pressionar uma vez, o botão esquerdo do mouse no vértice de origem e em seguida pressionar uma vez no vértice de destino.

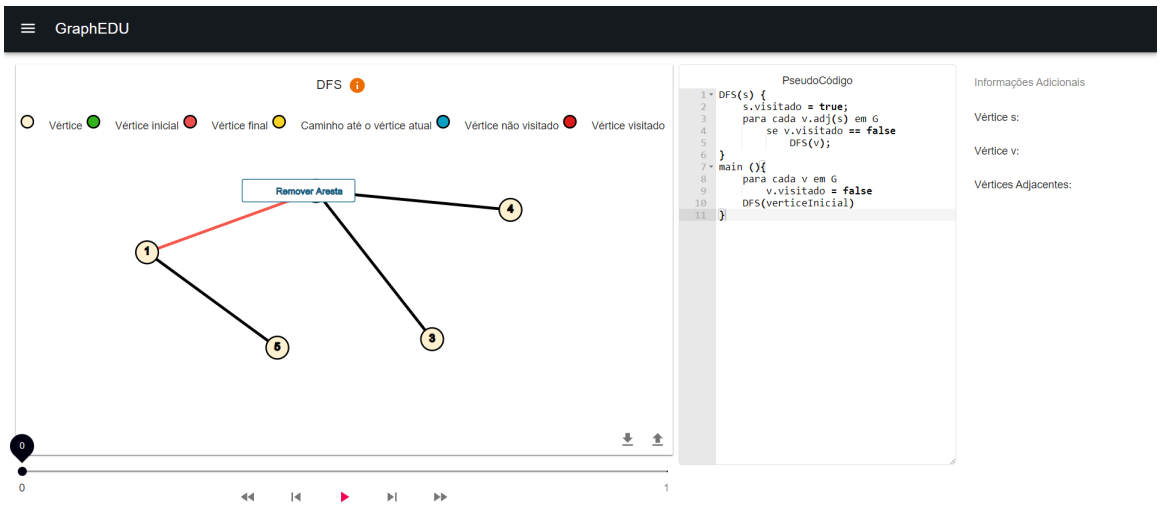
Ao pressionar o botão direito do mouse em algum vértice já criado, as opções "Vértice Inicial", "Vértice Final"e "Remover Vértice"apresentadas na Figura 31 poderão ser realizadas. As arestas também poderão ser atualizadas, as opções "Trocar Direção", "Editar Peso", e "Remover Aresta", são disponibilizadas. Para os algoritmos não dirigidos e não valorados somente o "Remover Aresta"é apresentado, conforme pode ser visto na Figura 32.

Figura 31 – Menu Vértices do Algoritmo DFS do GraphEDU



Fonte: O Autor (2021)

Figura 32 – Menu Arestas do Algoritmo DFS do GraphEDU



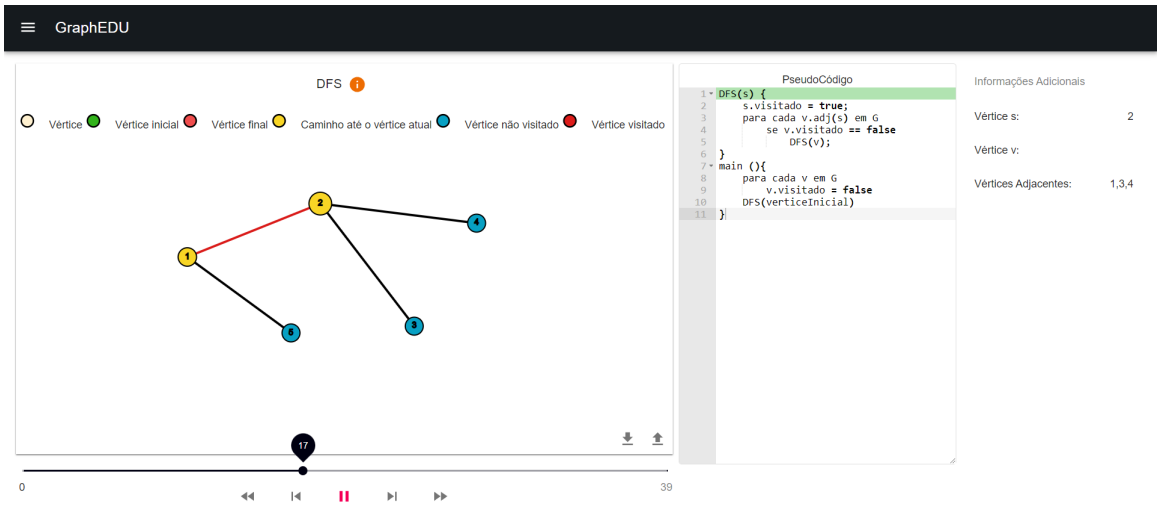
Fonte: O Autor (2021)

Além disso, o *Canvas* permite realizar a importação e a exportação dos grafos criados, para isso deve ser clicado nos botões localizados no canto inferior direito, visto na Figura 29. Os dados são exportados em formato de arquivo JSON, e a importação

somente aceita o mesmo formato. Na Figura 45 é apresentado a representação do grafo no formato JSON.

A terceira área é a área de apresentação do pseudocódigo, na qual é apresentado o pseudocódigo do algoritmo escolhido. Após a execução ser iniciada é realizada uma marcação na linha em que o passo do algoritmo está. Na Figura 33 pode ser visto um exemplo, a linha que está sendo simulada é a linha destacada em verde.

Figura 33 – Printscreen do Algoritmo DFS sendo executado



Fonte: O Autor (2021)

Os campos relacionados ao algoritmo, localizado ao lado do pseudocódigo, são informações úteis que serão apresentadas junto à simulação. Cada algoritmo tem dados importantes para o seu entendimento, como por exemplo o vértice s e o vértice v do algoritmo DFS, o intuito dessa área é apresentar e atualizar esses dados a cada passo da simulação e disponibilizar uma forma na qual o usuário tenha as informações necessárias para facilitar seu aprendizado.

A quinta e última área é referente ao componente de simulação, nesta área é onde toda a execução do algoritmo e atualizações de outras áreas são controladas. Para isso o sistema fornece 5 botões, sendo eles apresentados nessa ordem: para voltar totalmente a execução; voltar um passo, iniciar, pausar ou continuar; avançar um passo, e por fim finalizar a execução. As ações geradas por este componente atualizam as áreas *Canvas*, Apresentação do Pseudocódigo e Campos Relacionados ao Algoritmo.

4.3 FERRAMENTAS DE DESENVOLVIMENTO

Considerando os objetivos apresentados, a proposta de solução para o desenvolvimento de um sistema para visualização de algoritmos de grafos buscou ferramentas facilitadoras para o desenvolvimento do software. Desta forma, foram feitas buscas

de bibliotecas ou ferramentas existentes, utilizadas para a visualização de dados. As bibliotecas e ferramentas encontradas foram analisadas em 4 categorias: facilidade de uso, adaptabilidade, quantidade de exemplos e tamanho da comunidade. Dentre as opções descobertas, as bibliotecas D3.js, Vis.js e P5.js foram as mais relevantes. A partir da análise das três bibliotecas, foram identificadas vantagens e desvantagens de cada uma. Embora a facilidade de uso ser um dos requisitos procurados, para este trabalho, a adaptabilidade, quantidade de exemplos e o tamanho da comunidade foram os itens que mais se destacaram para a escolha da ferramenta. Isto posto, foi escolhida a biblioteca D3.js para o desenvolvimento da área *canvas*, apresentada na Seção 4.2, visto que é a ferramenta com a maior comunidade e foi a ferramenta com o maior número de dados encontrados sobre visualização em forma de grafos.

Com o intuito de facilitar a criação do design de interface de usuário buscou-se linguagens de programação capazes de auxiliar no desenvolvimento das demais áreas do sistema. Em virtude de as telas do sistema serem semelhantes, optou-se por uma linguagem na qual seja possível aproveitar trechos de códigos de uma forma simples. Na Seção 4.3.2 é descrita a biblioteca javascript React.js escolhida para o desenvolvimento do projeto.

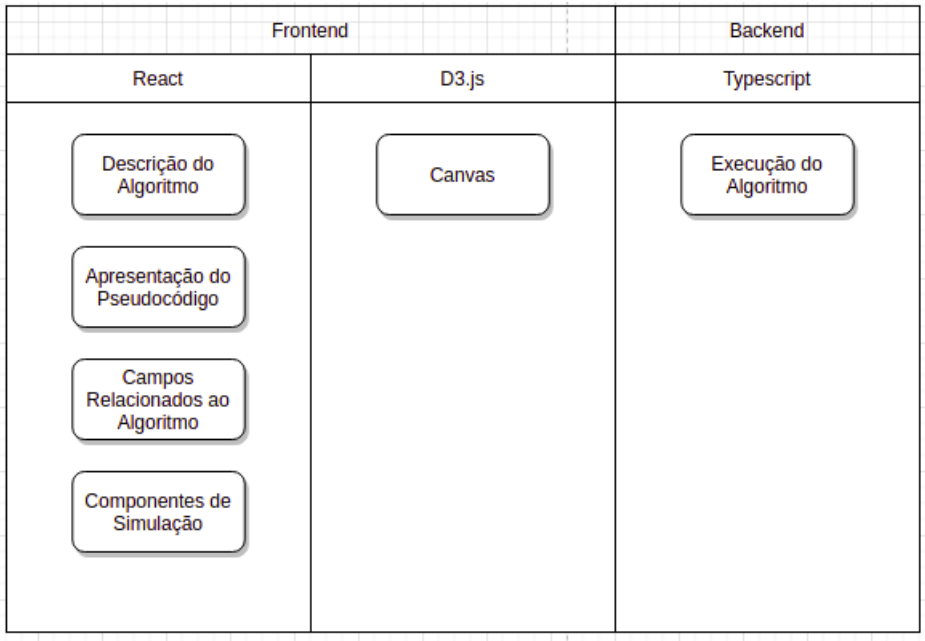
A ideia principal do sistema foi dividir o desenvolvimento em duas partes. A primeira foi o desenvolvimento do *front end* da aplicação, realizado através das ferramentas D3.js e React.js, apresentadas anteriormente. A segunda foi a implementação dos algoritmos, essas foram realizadas com a linguagem Typescript, uma vez que é uma linguagem de fácil utilização junto às demais escolhidas. A Figura 34 apresenta a relação entre as ferramentas de desenvolvimentos e as respectivas áreas que serão implementadas.

4.3.1 D3.js

D3.js é uma biblioteca javascript que tem como objetivo manipular os objetos em documentos *HyperText Markup Language* (HTML). Isso ocorre pois a biblioteca D3.js utiliza o formato de arquivo *Scalable Vector Graphics* (SVG) para apresentar os dados ao usuário, o SVG pode ser utilizado junto com o javascript para realizar essa manipulação e pode ser representado em três tipos de objetos gráficos, imagens, textos ou formas geométricas vetoriais. Tanto a ferramenta VisuAlgo, quanto a D3 Graph Theory apresentadas no Capítulo 3 utilizam esse formato de arquivo para representar os grafos criados através delas, a última também utiliza a biblioteca D3.js na criação dos grafos. O diferencial da D3.js são os recursos oferecidos pela própria biblioteca, não é preciso combinar a ela algum framework ou outra biblioteca. Utilizando apenas a D3.js é possível manipular o *Document Object Model* (DOM)[1] de um site e utilizar os componentes

[1] Interface que representa os elementos HTML de uma página web.

Figura 34 – Relação das ferramentas de desenvolvimento e as áreas utilizadas



Fonte: O Autor (2021)

de visualização disponibilizados por ela (BOSTOCK, 2021).

4.3.2 React

Criado pelo Facebook em 2011, React é uma biblioteca desenvolvida em javascript, utilizada para criar interfaces de usuários interativas, ela permite criar interfaces complexas a partir da criação de pequenos trechos de códigos isolados chamados de componentes. Os componentes criados pelo React são atualizados e renderizados à medida que os dados mudam, se o componente utilizar um dado que foi alterado, ele é renderizado (FACEBOOK, 2021a). O React foi escolhido pois através da criação de componentes é possível dividir a interface do usuário em partes independentes, podendo assim reutilizá-los em diversos lugares. Outras ferramentas como Angular.js e Vue.js também foram analisadas para o desenvolvimento deste projeto, porém o React foi escolhido devido à maior disponibilidade de artigos e tutoriais encontrados sobre a utilização dele com a biblioteca D3.js

Como o React não é um framework, é fácil de combinar outras bibliotecas com a ferramenta. Neste trabalho porém, as renderizações do React não foram utilizadas no componente *canvas* que foi desenvolvido com a biblioteca D3.js, em consequência de ambas bibliotecas utilizarem o DOM da página da web para realizar manipulações de dados, deste modo a aplicação do React foi utilizada nas demais áreas do sistema.

4.3.3 TypeScript

A linguagem TypeScript é comumente conhecida como um *Static Type Checker* do JavaScript, uma forma de utilizar JavaScript e corrigir erros de programação em tempo de compilação. A linguagem facilita o desenvolvimento, uma vez que não é possível utilizar um atributo que não existe em um objeto. A Figura 35 apresenta um exemplo em que não seria informado o erro se a linguagem utilizada fosse JavaScript.

Além disso, a linguagem permite realizar a tipagem dos objetos[2], característica que foi útil no desenvolvimento do sistema, visto que foi necessário compartilhar as informações do grafo para diversos componentes da tela. Desta forma, ao receber o objeto já era conhecido quais os atributos que ele teria.

Figura 35 – Erro de propriedade inválida - TypeScript

```
const obj = { width: 10, height: 15 };
const area = obj.width * obj.heigth;
```

```
Property 'heigth' does not exist on type '{ width: number; height: number; }'. Did you mean
'height'?
```

Fonte: (.ORG, 2021)

4.4 FUNCIONAMENTO DO SOFTWARE

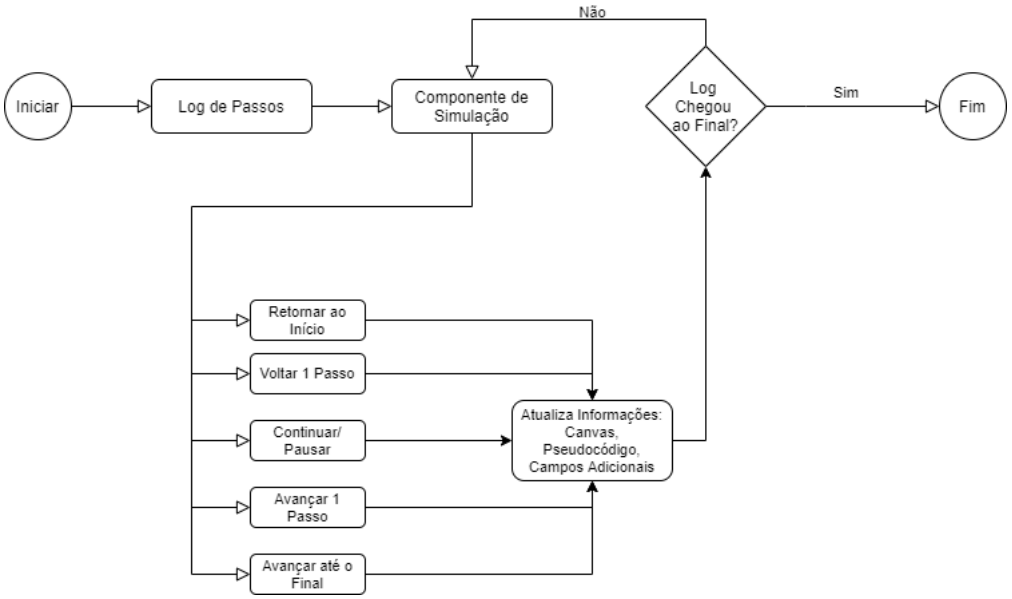
Neste trabalho o desenvolvimento da execução dos algoritmos foi baseada no software Python Tutor, uma ferramenta educacional que permite aos usuários visualizar o que acontece em cada linha do código, o software apresenta diagramas para representar a estrutura de dados do programa. Para realizar a representação dos dados, o software utiliza uma abordagem que executa o código do usuário antes de demonstrar os diagramas, o código é executado e durante o andamento um *log* das variáveis de cada linha do programa é feito. Após o fim da execução, uma lista com o *log* das linhas é retornado para o *frontend* da aplicação para serem demonstradas através dos diagramas (GUO; ALI, 2019).

A Figura 36 apresenta um diagrama de funcionamento do software proposto neste trabalho, a execução do algoritmo inicia após coleta das informações do usuário: arestas e vértices informados e configurações do grafo (dirigido ou não dirigido, valorado ou não valorado). Após o fim da execução, o componente de execução irá receber um *log* de passos do algoritmo, e então poderá gerenciar o estado da aplicação, através dos 5 botões disponibilizados no componente: Retornar ao início, voltar 1 passo, continuar/pausar, avançar 1 passo e avançar até o final. Os botões são responsáveis por

[2] Declarar quais propriedades estão no objeto

controlar qual passo está sendo apresentado, eles implicam na atualização das informações para o usuário, nesta etapa as áreas relacionadas ao passo são atualizadas. Por fim, será verificado se a execução chegou ao fim, caso contrário um novo passo será escolhido e os seus dados serão apresentados.

Figura 36 – Diagrama de Funcionamento do Software



Fonte: O Autor (2021)

4.4.1 Coleta de Informações

Como visto na Seção 4.3.1 a biblioteca D3.js possibilita a representação de modelos gráficos em objetos JavaScript, deste modo para a coleta das informações, o grafo criado pelo usuário é abstraído em duas listas de objetos: A primeira é uma lista de vértices e ela é composta dos seguintes atributos:

- id: Número do vértice.
- x: Coordenada x no Canvas.
- y: Coordenada y no Canvas.
- fx: Atributo fx no Canvas (Contém a coordenada x do Canvas enquanto o vértice está sendo movido).
- fy: Atributo fy no Canvas (Contém a coordenada y do Canvas enquanto o vértice está sendo movido).
- vx: Atributo vx no Canvas (Velocidade do eixo x enquanto o vértice está sendo movido)

- vy: Atributo vy no Canvas (Velocidade do eixo y enquanto o vértice está sendo movido)

O atributo id é inserido sequencialmente, os demais atributos são calculados pela própria biblioteca. Já a segunda lista é uma lista de arestas, com os atributos a seguir:

- id: Número da aresta.
- source: Contém o vértice de origem da aresta.
- target: Contém o vértice de destino da aresta .
- value: Contém o peso da aresta.

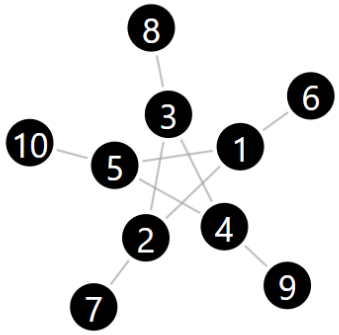
Após o algoritmo ser escolhido, é apresentado ao usuário um menu com as opções do grafo: valorado/não valorado e dirigido/não dirigido. As escolhas feitas pelo usuário são armazenadas no objeto que representa o grafo. Além disso, para compor as informações necessárias para a execução dos algoritmos, são guardados os vértices escolhidos como inicial e final. Visto que, são informações essenciais para a execução de alguns dos algoritmos. A Figura 37 apresenta um teste realizado com a biblioteca D3.js, na Figura 37b pode ser visto a lista de arestas do grafo, no quadrado superior em vermelho. Abaixo dele está a lista de vértices. O grafo da Figura 37a é o grafo resultante da união dessas duas listas de objetos.

4.4.2 Implementação dos Algoritmos

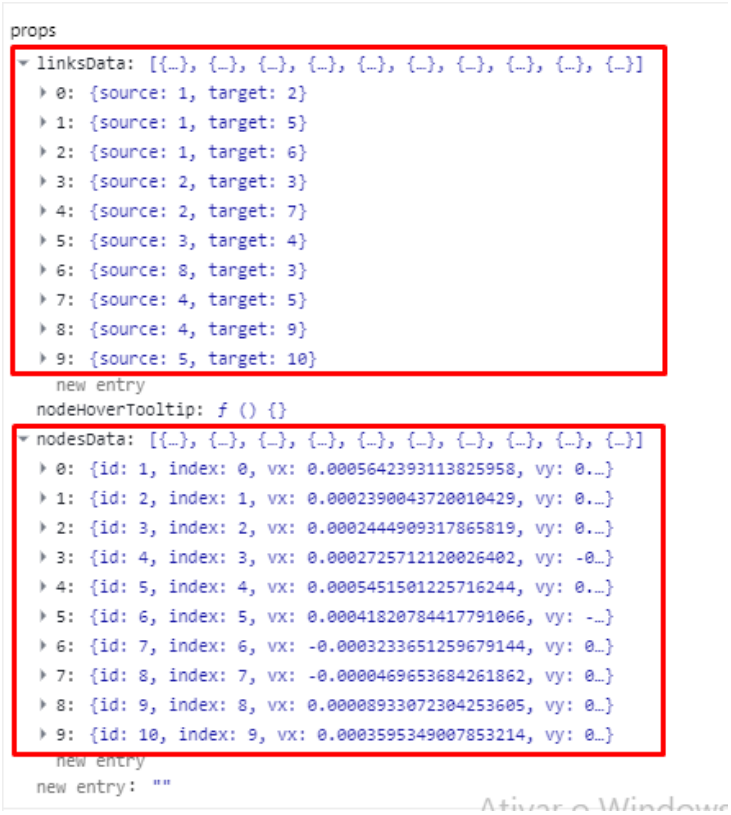
Está subseção descreverá como foi realizada a implementação dos algoritmos e como o *log* de passos de cada algoritmo é criado. Para cada algoritmo um conjunto diferente de dados são armazenados e ao fim da execução uma lista com as informações desse conjunto é retornada para dar início à visualização do algoritmo, como descrito na Seção 4.4. Este retorno contém uma lista com o conjunto de dados a cada linha do algoritmo. A seguir, é descrito os dados que são armazenados para cada tipo de algoritmo:

- DFS: Linha do pseudocódigo, vértice S, vértice V, vértices adjacentes, lista de vértices visitados, caminho até o vértice atual e o caminho das arestas.
- BFS: Linha do pseudocódigo, vértice V, vértice E, fila Q, vértices adjacentes, lista de vértices visitados e o caminho das arestas.
- Dijkstra: Linha do pseudocódigo, vértice S, vértice U, vértice V, distância do vértice U, distância do vértice V, aresta E, conjunto Q, vértices adjacentes e a lista com as distâncias dos vértices.

Figura 37 – Grafo criado com a biblioteca D3.js



(a) Grafo Resultante



(b) Objeto grafo

Fonte: O Autor

- Bellman-Ford: Linha do pseudocódigo, vértice S, vértice U, vértice V, distância do vértice U, distância do vértice V, aresta E, conjunto Q, lista de adjacências e tempDistancia (soma da Distância U com a aresta (u,v)).
- Floyd-Warshall: Linha do pseudocódigo, vértice I, vértice J, vértice K, distância (i,j), distância (i,k), distância (k,j) e a matriz com as distâncias do grafo.
- Prim: Linha do pseudocódigo, vértice U, vértice V, distância do vértice U, distância do vértice V, aresta E, conjunto Q, vértices adjacentes, lista com as distâncias dos vértices e a lista com as arestas do caminho.
- Kruskal: Linha do pseudocódigo, aresta E, vetor pai (informando qual é o pai de cada vértice) e a lista com as arestas do caminho.

4.4.3 Implementação do Algoritmo DFS

Esta subseção tem o objetivo de demonstrar a implementação do algoritmo DFS, que servirá como um exemplo de implementação. Será apresentado o código

fonte do algoritmo, contendo os métodos criados para armazenar os dados utilizados na demonstração do algoritmo.

4.4.3.1 Atributos e Métodos Utilizados

A Figura 38 apresenta o diagrama de classes do arquivo **DFS.ts**, nele são descritos os atributos e métodos utilizados no desenvolvimento do algoritmo. No Apêndice B é apresentado os demais diagramas de classe do sistema. Pode ser visto que o arquivo **DFS.ts** contém o atributo **grafo**, que é do tipo **IGrafo**. Esse objeto recebe as informações do grafo criado pelo usuário, a interface **IGrafo** contém as informações dos vértices, arestas e opções do grafo, como descrito na Seção 4.4.1. O atributo **Retorno** é do tipo **IRetornoDFS**, nele é armazenada uma lista contendo o conjunto de dados do algoritmo DFS, citado na Seção 4.4.2. Ele é o objeto que é retornado ao fim da execução.

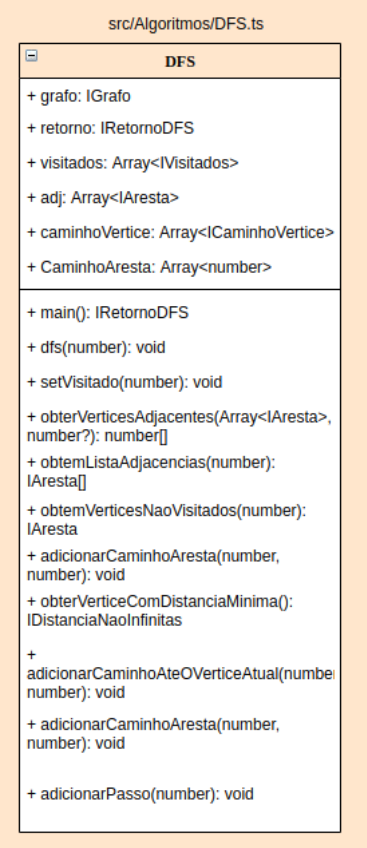
Os demais atributos foram utilizados para facilitar a implementação, o atributo **Visitados**, guarda em um *array* do tipo **IVisitados**, a informação dos vértices que foram visitados. A interface **IVisitados** é formada pelo atributo **idVertice**, do tipo *number* e do atributo **visitado**, do tipo *boolean*. Durante a execução do algoritmo o *array* é alterado após ser invocado o método *setVisitado*, que recebe como parâmetro um id de vértice, e altera para *true* o atributo visitado.

O atributo **adj** é um *array* do tipo **IAresta**, que representa a lista de adjacências do vértice que está sendo iterado no método **dfs**. A interface **IAresta** contém os atributos das arestas criadas pelo usuário, estes atributos são descritos na Seção 4.4.1. No início de cada nova iteração, a lista de adjacências é atualizada pelo método *obtemListaAdjacencias*, essa informação é apresentada ao usuário na área Campos Relacionados ao Algoritmo, conforme descrito na Seção 4.2.

Durante a demonstração da execução do algoritmo DFS, é destacado, no canvas, o vértice da iteração e os vértices que compõem o caminho até ele. Para realizar essa animação, é utilizado o atributo **caminhoVertice**, um *array* do tipo **ICaminhoVertice**. A interface **ICaminhoVertice** contém o atributo **vértice**, do tipo *number* e o atributo **caminho**, um *array* do tipo *number*. Durante a execução essa informação é atualizada pelo método *obterCaminhoAteVerticeAtual*, o método recebe como parâmetro o vértice anterior e o vértice atual, e atualiza o caminho do vértice atual com o caminho do vértice anterior mais o vértice atual.

Por fim, o atributo **caminhoAresta**, utilizado para apresentar as arestas percorridas durante a execução do algoritmo, é um *array* do tipo *number*, ele é utilizado para armazenar as arestas percorridas. O método *adicionarCaminhoAresta* recebe como parâmetro o vértice anterior e o vértice atual, e acrescenta no *array* a aresta corresponde a esses vértices.

Figura 38 – Diagrama de Classe DFS



O autor (2021)

4.4.3.2 Implementação

A Figura 39 apresenta um trecho da implementação do algoritmo DFS, na imagem podemos observar o método *DFS*, que recebe como parâmetro a variável *s* do tipo *number*, essa variável representa o vértice inicial do grafo.

O desenvolvimento do algoritmo é dividido em duas partes, inicialmente é feito a sua implementação e depois é realizado o mapeamento do algoritmo criado, com o pseudocódigo definido previamente. Para isso, é utilizado o método *adicionarPasso*, ele recebe como parâmetro inicial, o número da linha do pseudocódigo, além disso, o método ainda armazena os demais dados utilizados na demonstração. Esses dados são atribuídos ao objeto *Retorno* e ao fim da execução, esse objeto é retornado para dar início a simulação.

4.4.4 Atualização das Informações

Esta subseção tem como objetivo explicar como as informações obtidas através da execução do algoritmo são apresentadas. Como detalhado na Seção 4.4, as áreas Canvas, Apresentação do Pseudocódigo e Campos Relacionados ao Algoritmo são atualizadas a cada passo do algoritmo.

Figura 39 – Trecho da implementação do algoritmo DFS

```
143     dfs(s: number) {
144         this.adj = this.obtemListaAdjacencias(s);
145         this.adicionarPasso(1, s, undefined);
146         this.adicionarPasso(2, s, undefined);
147         this.setVisitado(s);
148         this.adicionarPasso(3, s, undefined);
149         this.adj.forEach(aresta => {
150             this.obtemVerticesNaoVisitados(aresta)
151                 .forEach(vertice => {
152                     this.obterCaminhoAteVerticeAtual(s, vertice.idVertice);
153                     this.adicionarPasso(4, s, vertice.idVertice);
154                     this.adicionarPasso(5, s, vertice.idVertice);
155                     this.adicionarCaminhoAresta(s, vertice.idVertice)
156                     this.dfs(vertice.idVertice);
157                     this.adj = this.obtemListaAdjacencias(s);
158                     this.adicionarPasso(3, s, undefined);
159 
160                 })
161         })
162     }
```

O autor (2021)

Visto que, a representação do grafo criado pelo usuário é feita no formato de arquivo SVG, cada vértice e aresta é representado por uma tag dentro do HTML da página. A atualização do grafo, é feita diretamente através de eventos javascript, adicionando classes *Cascading Style Sheets* (CSS) com estilos de cores diferentes nas arestas ou vértices percorridos. Deste modo, simulando no grafo a execução do algoritmo.

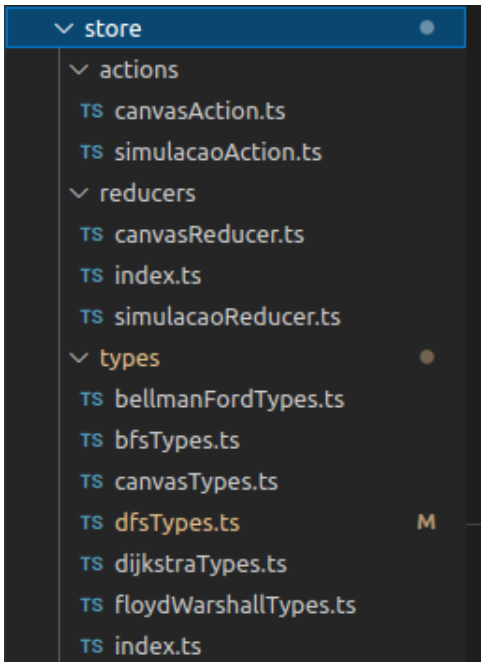
Nas demais áreas, a utilização da biblioteca React.js, descrita na subseção Seção 4.3.2, teve o intuito de facilitar a atualização. A proposta foi criar um componente para cada área e passar para eles os dados necessários do passo atual para o componente ser atualizado. O componente Apresentação do Pseudocódigo, necessita somente da linha do pseudocódigo, já o componente Campos Relacionados ao Algoritmo, tem os dados específicos de cada algoritmo. A atualização dos componentes foi realizada utilizando a função especial *state hook* do React, a explicação de seu uso pode ser visto na sua documentação em Facebook (2021b).

4.4.4.1 React Redux

Para gerenciar esse fluxo de dados foi escolhido utilizar uma biblioteca facilitadora, criada especialmente para isso. A biblioteca React Redux, foi criada para manipular e concentrar as trocas de estados da aplicação em um só lugar, como visto em Abramov (2021). A Figura 40 apresenta a estrutura de pastas e arquivos criada para o gerenciamento da aplicação, a store. A pasta *actions*, define todas as ações que podem ser geradas no sistema, para cada ação existe um *reducer* correspondente, que por sua vez atualiza os componentes que utilizam o dado atualizado. As *actions* funcionam como funções, e são chamadas quando deve ter alguma atualização dos dados. Os seus parâmetros e o seu retorno são definidos pelo o seu *type* correspondente, dentro da pasta *types*. Na Figura 41 pode ser observado um diagrama com um exemplo do

funcionamento do redux. No exemplo, o botão *Deposit* foi acionado pelo usuário, uma *action* foi disparada e o seu *reducer* corresponde atualizou os dados, está ação fez com que os componentes que utilizam esse dado fossem renderizados.

Figura 40 – Store do Sistema



Fonte: O Autor (2021)

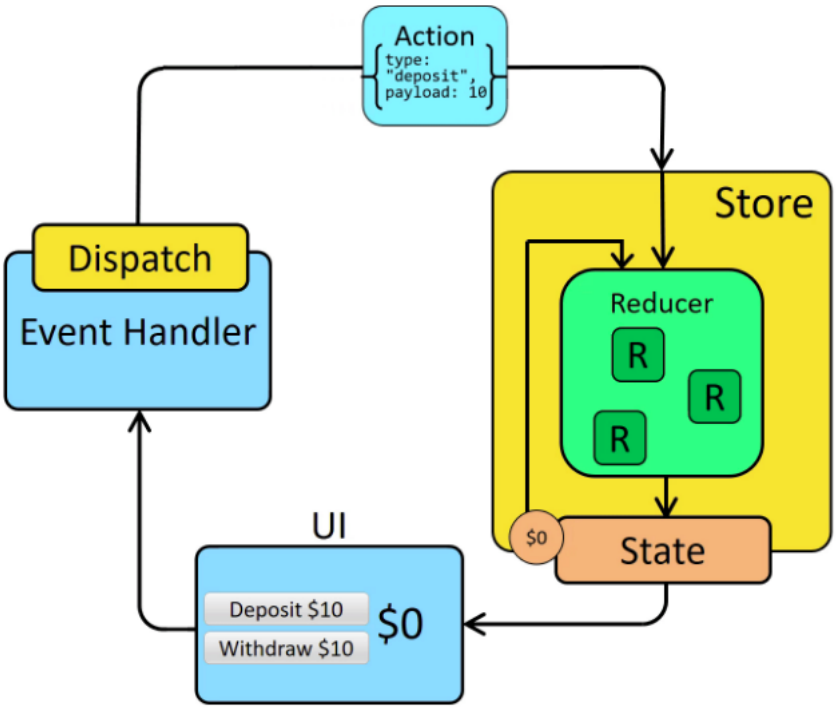
4.4.4.2 Atualização das Informações no Algoritmo DFS

Na Figura 42 é apresentado um trecho do código do arquivo *ComponenteSimulacao.tsx*, no qual pode ser visto o início da execução do algoritmo. Na linha 55 é realizada a instância da classe *DFS*, descrita na Seção 4.4.3 e na linha 56 a variável *retornoDFS* recebe o resultado da execução. Na linha 58 é chamado o método *setQndtPassos*, utilizado para atualizar a quantidade de passos do algoritmo, que resulta na renderização do próprio componente.

As linhas 57 e 59 disparam as *actions* com os dados atualizados, a primeira envia o retorno do algoritmo, já a segunda envia a quantidade de passos. Como descrito na Seção 4.4.4.1, esses eventos geram a renderização dos componentes que utilizam esses dados.

Após o fim da execução do algoritmo é iniciado a simulação, para isso existe no arquivo *ComponenteSimulacao.tsx* um trecho de código que contém o método *setTimeOut* do JavaScript, o método recebe como parâmetro uma função para ser executada a cada determinado período de tempo. A função em questão é a atualização do passo do algoritmo, e foi escolhido o período de tempo de 1 segundo.

Figura 41 – Diagrama da Store



Fonte: (ABRAMOV, 2021)

Conforme descrito na Seção 4.4, a cada atualização do passo os componentes Canvas, Pseudocódigo e Campos Relacionados ao Algoritmo são atualizados. A Figura 43 apresenta um trecho de código do arquivo *atualizarGrafoDFS.js*, um exemplo da atualização do Canvas a partir das informações obtidas. Neste código é utilizada a biblioteca D3.js para destacar as arestas que foram percorridas até o vértice que está sendo testado. Para cada aresta do grafo é testado se a aresta está no caminho. Na linha 56, a variável caminho recebe o conjunto de dados do DFS, e então a partir do passo atual se o algoritmo não foi executado ou se a aresta não está no caminho, então a cor atribuída é preta, se não, para a cor da aresta é atribuído a cor definida como visitada.

Nos demais componentes, Pseudocódigo e Campos Relacionados ao algoritmo, para obter o conjunto de dados do passo atual, é realizada a mesma abordagem da linha 56 da Figura 43, o componente recebe o *array* com o log de passos e o número do passo, após isso as informações do componente são atualizadas.

4.5 PUBLICAÇÃO DA APLICAÇÃO

Para disponibilizar a aplicação para o uso dos usuários foi utilizado a plataforma *Netlify*, responsável por hospedar aplicações web. O procedimento é bem simples de ser feito, primeiramente é necessário associar a conta do *GitHub* e selecionar o repositório

Figura 42 – Componente de Simulação DFS

```
53      switch (props.simulacao.tipoAlgoritmo) {
54        case EAlgoritmos.DFS:
55          const dfs = new DFS(props.canvas);
56          const retornoDFS = dfs.main();
57          props.updateDFSAction(retornoDFS)
58          setQntdPassos(retornoDFS.caminho.length);
59          props.setQntdPassosAction(retornoDFS.caminho.length);
60          break;
```

Fonte: O Autor (2021)

Figura 43 – Trecho de código do arquivo *atualizarGrafoDFS.js*

```
50          d3.selectAll(`.edge`).data(links, function (d) {
51              return "v" + d.source.id + "-v" + d.target.id;
52              })
53              .transition()
54              .duration(500)
55              .attr("stroke", aresta => {
56                  const caminho = simulacao.dfs.caminho[simulacao.passo];
57                  if(caminho === undefined) return "black";
58
59                  if(caminho.caminhoAresta.includes(aresta.id))
60                      return CorVerticeVisitado
61
62                  return "black";
63
64              })
65      }
```

Fonte: O Autor (2021)

da aplicação, ou pode ser feito o *upload* de uma pasta com o código fonte do sistema. Após isso a publicação é feita automaticamente. A cada *commit* no repositório ou alteração da pasta com o código fonte, o sistema é atualizado. A versão de conta grátis do *Netlify* permite somente 100Gb de largura de banda por mês a ser utilizada, porém durante os testes de uso, o máximo que o sistema utilizou foi 30Mb no mês, portanto a plataforma deve servir como hospedagem para o sistema.

4.6 AVALIAÇÃO

Avaliar a qualidade de um software é uma tarefa complexa, mas caso seja feita corretamente é um ótimo método para descobrir se o software supriu as necessidades do cliente. No âmbito de desenvolvimento de software, a qualidade de um sistema pode ser medida através de uma lista de aspectos que devem ser cumpridos para garantir a satisfação do cliente. Uma avaliação totalmente positiva nem sempre é concebida nas primeiras versões do sistema. Por isso, a qualidade do software é melhorada à medida que o desenvolvimento progride (MORAES, 2010).

Para este trabalho foi usado como modelo base de avaliação a norma NBR ISO/IEC 9126-1[3]. As perguntas que foram aplicadas são divididas nas categorias a seguir:

- Funcionalidade
- Confiabilidade
- Usabilidade
- Eficiência

As perguntas fazem parte de um questionário que foi entregue aos alunos que cursaram ou estão cursando a disciplina de Teoria dos Grafos, para os mesmos responderem após a utilização da aplicação. Os dados coletados foram utilizados para descobrir se o sistema colaborou com o aprendizado dos alunos, assim como contribuir para as dicas de trabalhos futuros. As perguntas a seguir compõem o questionário:

a) Funcionalidade
   - O sistema simula corretamente a execução dos algoritmos?
   - O sistema facilita a compreensão dos algoritmos?
   - A pseudo linguagem usada nos algoritmos é apropriada?
   - Você gostaria de utilizar o sistema em uma sala de aula, junto com a explicação de um professor?
   - Indique as melhorias que você gostaria de ver no sistema.

b) Confiabilidade
   - O sistema apresenta falhas?
   - Se você marcou sim na alternativa acima, quais foram as falhas encontradas?

c) Usabilidade
   - É fácil realizar as ações no sistema?
   - O suporte visual da ferramenta pode ser melhorado?
   - Se você respondeu "Sim"ou "Parcialmente"na pergunta acima, indique quais melhorias você gostaria de ver na aplicação.

d) Eficiência
   - O tempo de resposta para a inicialização da execução é rápido?

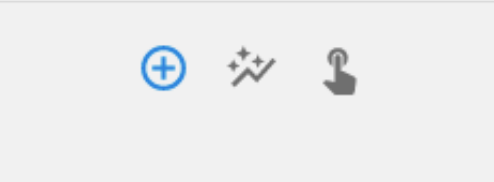
[3] Norma mundial para a qualidade de produtos de software

No Apêndice C é apresentado o resultado do questionário aplicado aos alunos. Foi registrada a participação de 9 alunos do curso, que utilizaram e deram o seu *feedback* sobre a sua experiência com o sistema.

Como resultado da avaliação dos alunos, foram realizadas algumas melhorias na aplicação, a fim de facilitar a interação do usuário com o sistema. Alguns usuários relataram não saber o que fazer ao abrir pela primeira vez as telas dos algoritmos, com base nisso, a janela de informações, vista na Figura 30, passou a ser exibida após o carregamento da página.

Além disso, foi constatada uma dificuldade na criação das arestas, alguns dos nossos *beta-testers* citaram isso como algo a ser melhorado. Em consequência disto, foi adicionado ao sistema os modos de criação, botões responsáveis por determinada ações no grafo, na Figura 44 os botões são apresentados. Essa nova funcionalidade permite a criação de vértices, criação de arestas e a movimentação do grafo, após o seu respectivo botão ser clicado.

Figura 44 – Modos de Criação do Grafo



Fonte: O Autor (2021)

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi realizada uma pesquisa para o desenvolvimento de um sistema para a visualização de algoritmos de grafos. A pesquisa buscou soluções existentes capazes de auxiliar no aprendizados dos alunos, e foram encontradas algumas ferramentas sobre o assunto, que serviram de inspiração para o desenvolvimento do software.

O trabalho teve como objetivo criar uma solução capaz de simular a execução de um algoritmo de grafo e possibilitar a visualização do andamento da sua execução. Foi descrita a necessidade de proporcionar ao usuário do sistema a criação do seu próprio grafo, além disso foi detalhado que para fins de facilitar a compreensão do algoritmo foi preciso apresentar os pseudocódigos dos algoritmos escolhidos e os dados relacionados ao algoritmo. Foi descrito que durante a simulação, o grafo criado pelo o usuário, o pseudocódigo e os campos adicionais seriam atualizados a cada mudança de estado.

Para a construção da aplicação foram utilizadas bibliotecas e linguagens de programação capazes de simplificar o processo de desenvolvimento, assim como facilitar as futuras alterações do software. Para isso, as bibliotecas JavaScript D3.js e React Js foram utilizadas. Junto a elas, utilizada na implementação dos algoritmos, foi aplicada a linguagem de programação TypeScript. Visto que, as telas do sistema partilham da mesma estrutura, o desenvolvimento do sistema teve como objetivo criar componentes capazes de serem utilizados em diversas partes da aplicação, a fim de diminuir o retrabalho e a duplicidade de código.

Para realizar a avaliação do sistema foi solicitado aos alunos do curso que cursaram a disciplina de Teoria dos Grafos relatar como foi a sua experiência com o software, para isso foi disponibilizado um link com o endereço da aplicação e um formulário com algumas perguntas sobre a utilização da aplicação. Os resultados obtidos demonstraram uma aprovação dos alunos em relação ao uso do software, 100% das respostas afirmaram que o sistema simula corretamente a execução dos algoritmos, que o sistema facilita a compreensão dos algoritmos e que a pseudo linguagem usada nos algoritmos foi apropriada. No entanto, pôde ser observado uma dificuldade na utilização de parte dos avaliadores, 33% dos usuários responderam parcialmente para a pergunta "É fácil realizar as ações no sistema?", além disso, mais da metade dos usuários, 55.5%, responderam Sim ou Parcialmente para a pergunta "O suporte visual da ferramenta pode ser melhorado?", com base nisso, para facilitar a utilização do sistema, algumas melhorias foram aplicadas, descritas na Seção 4.6. A partir das respostas da pergunta "Você gostaria de utilizar o sistema em uma sala de aula, junto com a explicação de um professor?", pode ser concluído que mesmo que o sistema facilite a compreensão

dos algoritmos, a sua utilização junto com a explicação de um professor é indispensável, visto que 100% dos alunos responderam sim para a pergunta. O resultado do questionário demonstra que o sistema irá auxiliar o entendimento dos alunos durante a explicação dos algoritmos.

5.1 SUGESTÕES DE TRABALHOS FUTUROS

Como trabalhos futuros sugere-se:

- A possibilidade de o usuário escolher grafos já criados pelo sistema.
- A implementação dos demais algoritmos vistos na disciplina.
- A demonstração dos conceitos vistos na disciplina através dos grafos criados pelo *canvas* do sistema.

REFERÊNCIAS

ABRAMOV, D. **Getting Started with React Redux**. React Redux, 2021. Disponível em: <https://react-redux.js.org/introduction/getting-started>. Acesso em: 15 nov. 2021.

ALEXANDERSON, G. About the cover: Euler and the königsberg bridges: A historical view. **Bulletin (New Series) of the American Mathematical Society**, 10 2006. Disponível em: <https://www.researchgate.net/publication/225035512_About_the_Cover_Euler_and_the_Konigsberg_Bridges_A_Historical_View>. Acesso em: 11 abr. 2021.

BALTAZAR, R.; PEREIRA, L. O estudo de grafos: uma proposta investigativa. **Educação Matemática Pesquisa : Revista do Programa de Estudos Pós-Graduados em Educação Matemática**, v. 20, n. 2, 2018. ISSN 1983-3156. Disponível em: <https://revistas.pucsp.br/index.php/emp/article/view/38665>.

BONDY, J. A. **GRAPH THEORY WITH APPLICATIONS**. University of Waterloo, Ontario, Canada: Department of Combina tories and Optimization, 1976.

BOSTOCK, M. **Data-Driven Documents**. Mike Bostock, 2021. Disponível em: <https://d3js.org/>. Acesso em: 05 jun. 2021.

BURCH, K. J. Chemical graph theory. **Mathematical Association of America**, 10 2006. Acesso em: 21 abr. 2021.

CARLSON, S. C. Königsberg bridge problem. **Encyclopedia Britannica**, Chicago, 30 Jul. 2010. Disponível em: <https://www.britannica.com/science/Konigsberg-bridge-problem>. Acesso em: 11 abr. 2021.

FACEBOOK. **Tutorial: Introdução ao React**. Facebook Inc, 2021. Disponível em: <https://pt-br.reactjs.org/tutorial/tutorial.html>. Acesso em: 05 jun. 2021.

____. **Usando o State do Hook**. Facebook Inc, 2021. Disponível em: <https://pt-br.reactjs.org/docs/hooks-state.html>. Acesso em: 15 jun. 2021.

FADZILAH, A. R. N.; NURKALIZA, K.; FARHANA, A. A. Web-based visualization tools of data structure & algorithm–a review of experience. **International Conference on Information Technology and Multimedia**, p. 27–33, 2016. ISSN 978-967-0850-51-1.

GOLDBARG, M.; GOLDBARG, E. **Grafos: Conceitos, Algoritmos e Aplicações**. Rio de Janeiro: Elsevier, 2017.

GUO, P.; ALI, K. **How Python Tutor Uses Debugger Hooks to Help Novices Learn Programming**. IEE Software Blog, 2019. Disponível em: <http://blog.ieeesoftware.org/2019/02/python-tutor.html/>. Acesso em: 06 jun. 2021.

LEONARD EULER'S SOLUTION TO THE KONIGSBERG BRIDGE PROBLEM - EULER'S CONCLUSIONS. United States: Teo Paoletti, 2011. Semanal. ISSN 1941-9198. Disponível em: <https://www.maa.org/press/periodicals/convergence/leonard-eulers-solution-to-the-konigsberg-bridge-problem>. Acesso em: 11 abr. 2021.

MOCINECOVá, K.; STEINGARTNER, W. Software support for visualizing of the graph algorithms in a novel approach in educating of young it experts. **Transactions on Internet Research**, v. 16, n. 2, p. 14–23, 2020. Disponível em: <http://ipsitransactions.org/journals/papers/tir/2020jul/p3.pdf>.

MORAES, L. **Qualidade de Software - Engenharia de Software 29**. DevMedia, 2010. Disponível em: <https://www.devmedia.com.br/qualidade-de-software-engenharia-de-software-29/18209>. Acesso em: 15 jun. 2021.

NETTO, P. O. B.; JURKIEWICZ, S. **Grafos: Introdução e Prática**. Rio de Janeiro: Blucher, 2017.

NICOLETTI, M. D. C. **Fundamentos da Teoria dos Grafos para Computação**. Rio de Janeiro: LTC - Livros Técnicos e Científicos Editora Ltda., 2018.

NRICH. The königsberg bridge problem. **University of Cambridge**, 02 2011. Disponível em: <https://nrich.maths.org/2484>. Acesso em: 11 abr. 2021.

.ORG, T. L. **TypeScript para o novo programador**. TypeScript Lang, 2021. Disponível em: <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>. Acesso em: 17 out. 2021.

PANDEY, A. **D3 Graph Theory**. GitHub, 2017. Disponível em: <https://github.com/mrpandey/d3graphTheory>. Acesso em: 10 mai. 2021.

QUINN, A. Using apps to visualize graph theory. **The Mathematics Teacher**, v. 108, n. 8, p. 626–631, 2014.

ROBERTS, F. S. **Graph Theory and Its Applications to Problems of Society**. Filadelfia: Rutgers University, 1978.

SCIENCE, D. of C. Visualising algorithms with a click. **National University of Singapore**, 04 2020. Disponível em: <https://www.comp.nus.edu.sg/news/features/2020stevenhalim/>. Acesso em: 08 mai. 2021.

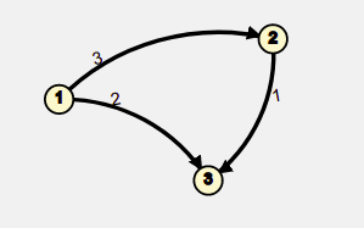
SILVA, L. F. da; RODRIGUES, M. A. R. de. IntroduÇÃo ao estudo da teoria dos grafos: Uma proposta de sequÊncia didÁtica para o ensino mÉdio. **UFRGS**, Novo Hamburgo, 2015. Disponível em: <https://www.lume.ufrgs.br/bitstream/handle/10183/134082/000984432.pdf?sequence=1>. Acesso em: 30 abr. 2021.

SPOKE–HUB Distribution Paradigm. In: WIKIPÉDIA: a enciclopédia livre. Wikipedia, 2021. Disponível em: <https://en.wikipedia.org/wiki/Spoke\T1\textendashhub_distribution_paradigm>. Acesso em: 12 abr. 2021.

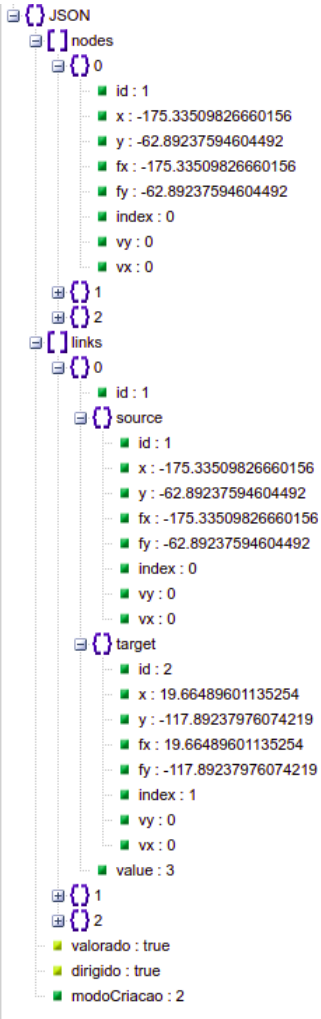
ANEXO A – EXPORTAÇÃO DE GRAFO

As imagens deste anexo apresentam um grafo dirigido e valorado, criado pelo autor, e o arquivo JSON gerado após a exportação do grafo.

Figura 45 – Representação de Grafo para arquivo JSON



(a) Grafo criado pelo autor

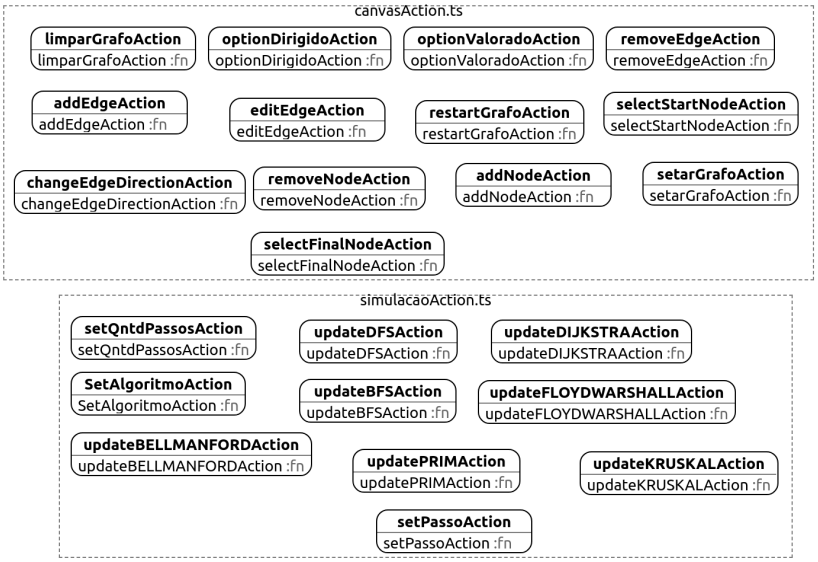


(b) Arquivo gerado pela exportação do grafo

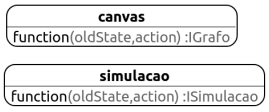
Fonte: O Autor (2021).

ANEXO B – DIAGRAMA DE CLASSES

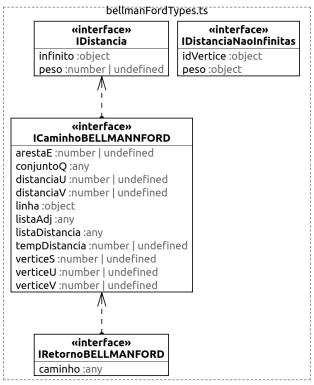
Figura 46 – Diagrama de classes da pasta src/store



(a) Diagrama da pasta src/store/actions

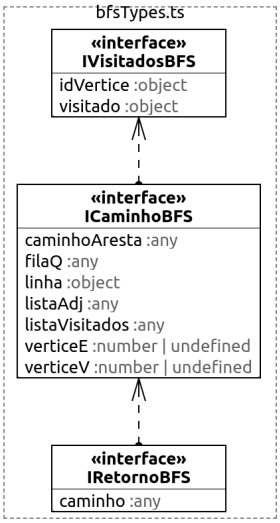


(b) Diagrama da pasta src/store/reducers

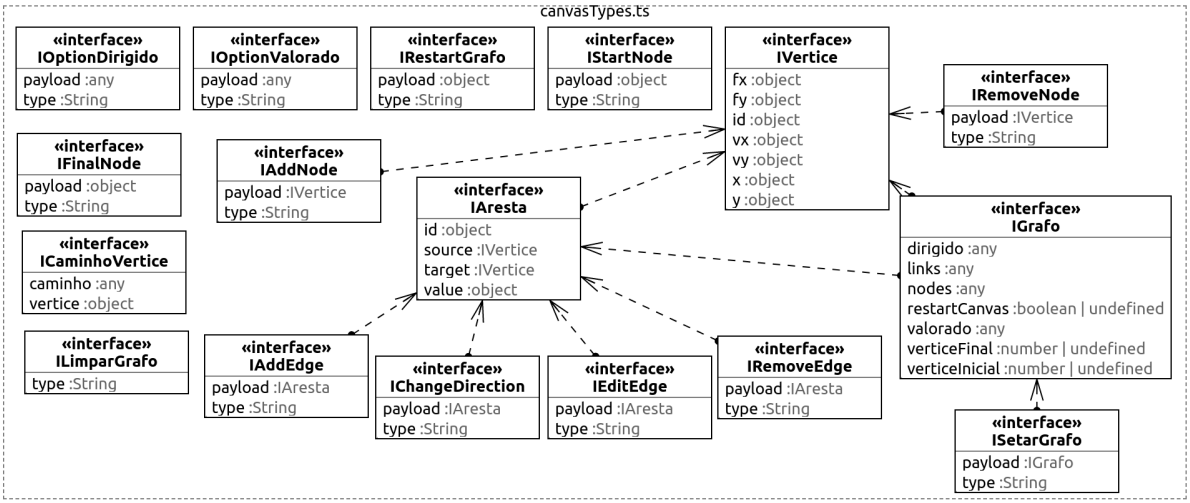


(c) Diagrama de classes do arquivo src/store/types/bellmanfordTypes.ts

Figura 47 – Diagrama de classes da pasta src/store



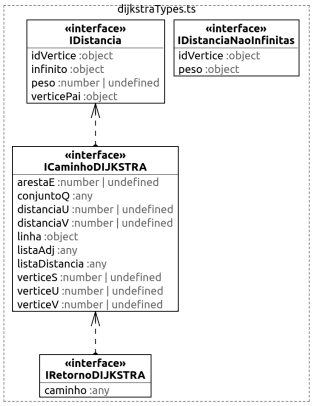
(a) Diagrama de classes do arquivo src/store/types/bfsTypes.ts



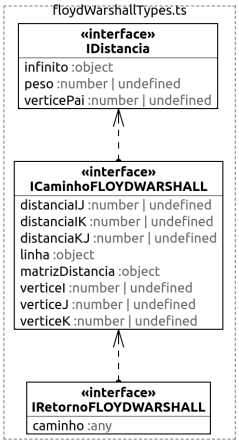
(b) Diagrama de classes do arquivo src/store/types/canvasTypes.ts

Fonte: O Autor (2021).

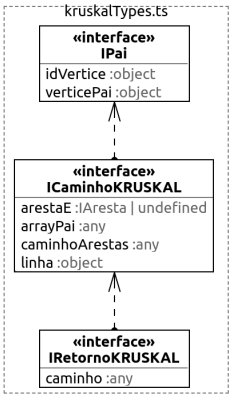
Figura 48 – Diagrama de classes da pasta src/store



(a) Diagrama de classes do arquivo src/store/types/dijkstraTypes.ts



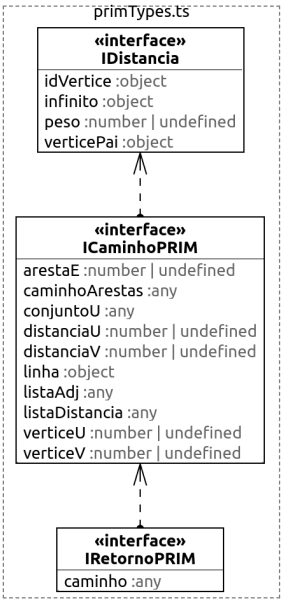
(b) Diagrama de classes do arquivo src/store/types/floydWarshallTypes.ts



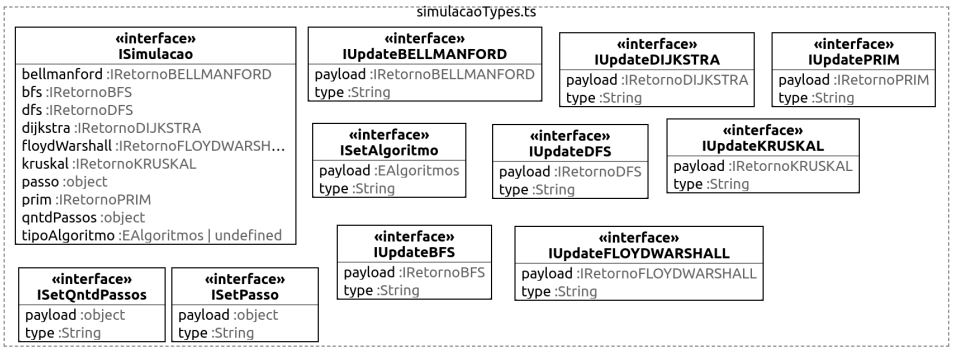
(c) Diagrama de classes do arquivo src/store/types/kruskalTypes.ts

Fonte: O Autor (2021).

Figura 49 – Diagrama de classes da pasta src/store



(a) Diagrama de classes do arquivo src/store/types/primTypes.ts



(b) Diagrama de classes do arquivo src/store/types/simulacaoTypes.ts

Fonte: O Autor (2021).

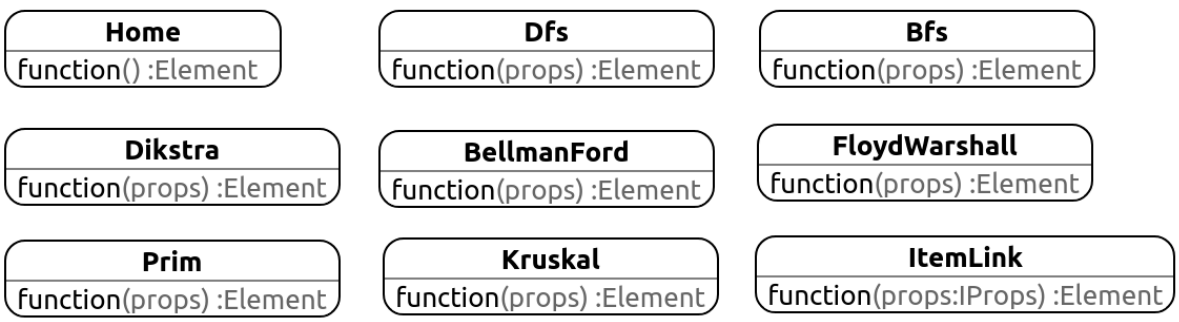


Figura 50 – Diagrama de classes da pasta src/pages

Fonte: O Autor (2021).

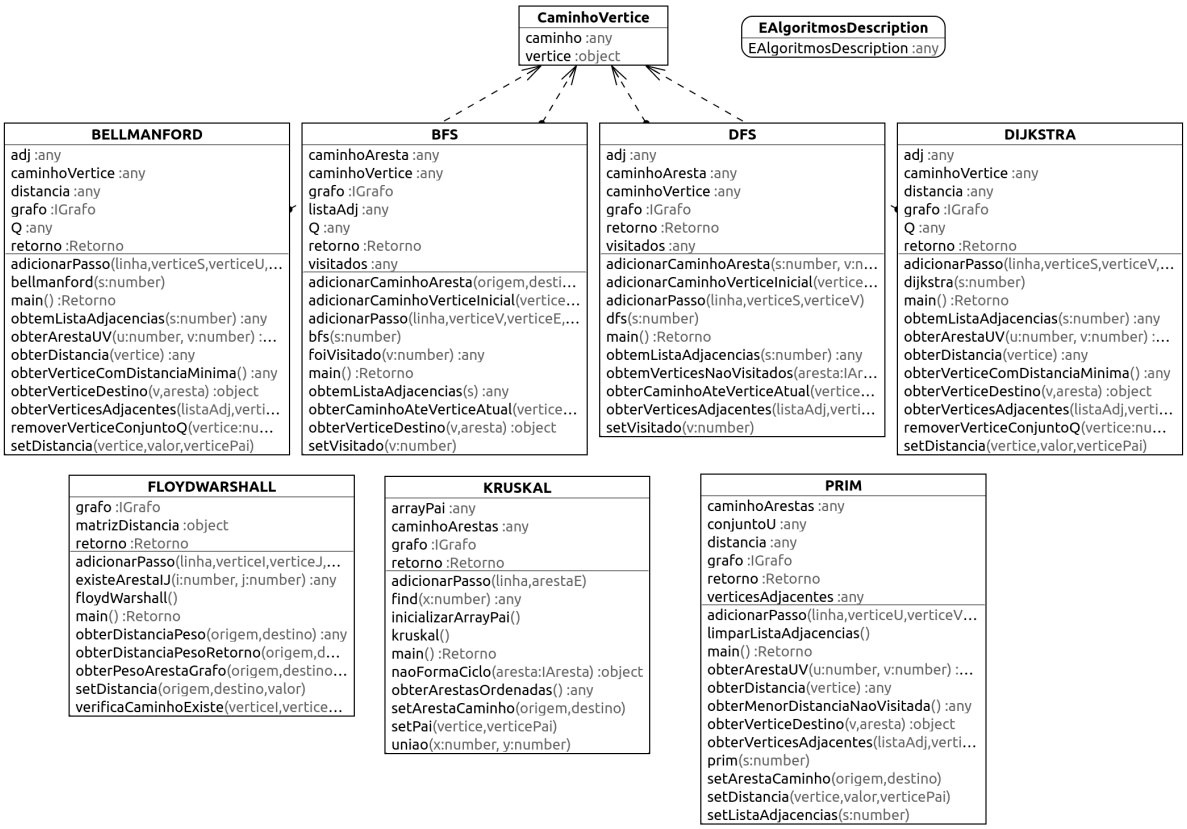


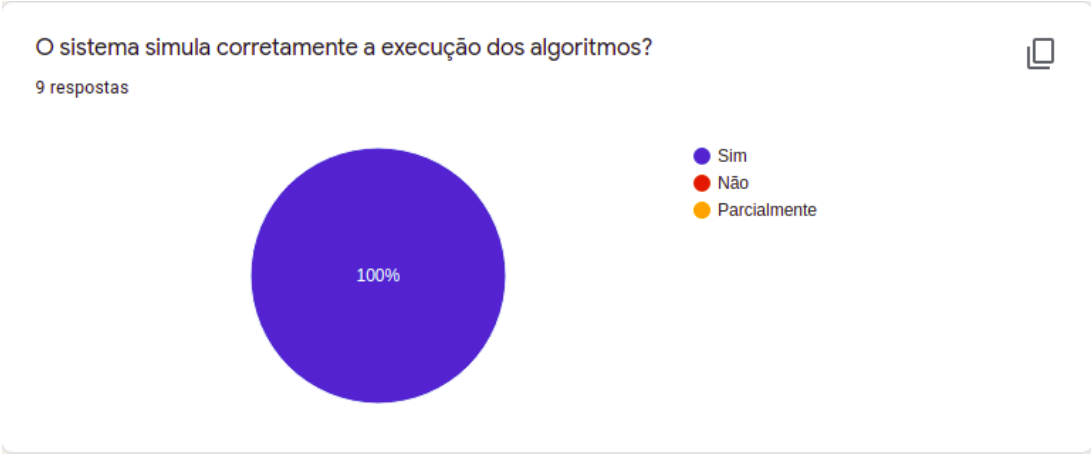
Figura 51 – Diagrama de classes da pasta src/Algoritmos

Fonte: O Autor (2021).

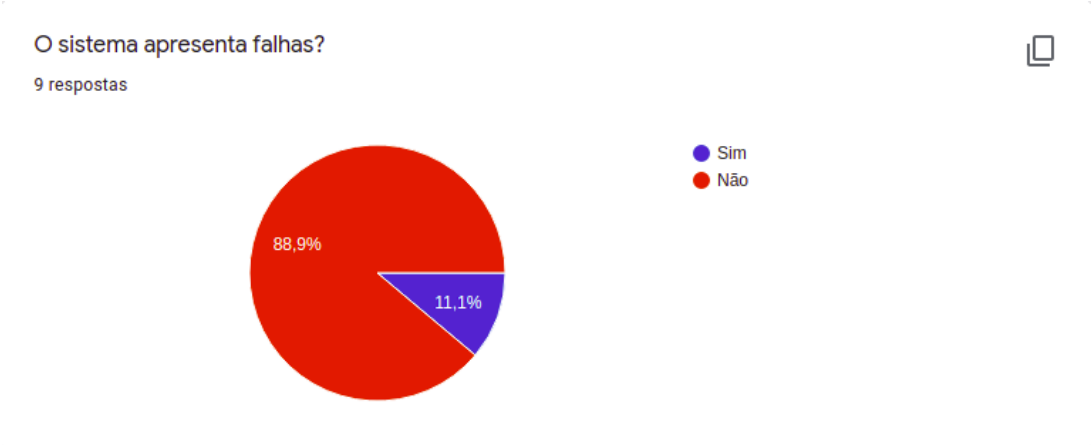
ANEXO C – RESPOSTAS DO QUESTIONÁRIO

As imagens deste anexo apresentam o resultado do questionário.

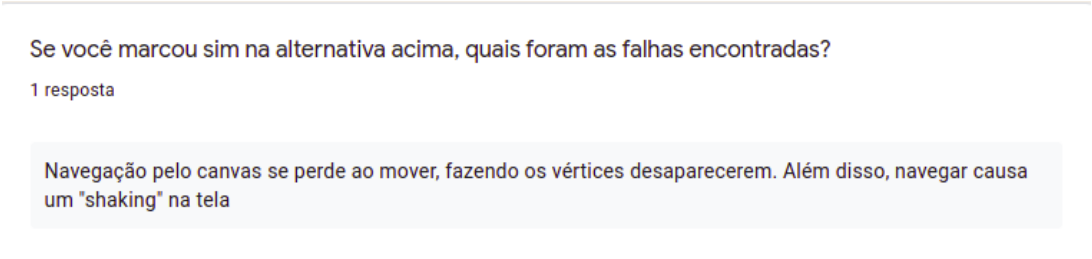
Figura 52 – Perguntas do Questionário - Parte 1



(a) O sistema simula corretamente a execução dos algoritmos?



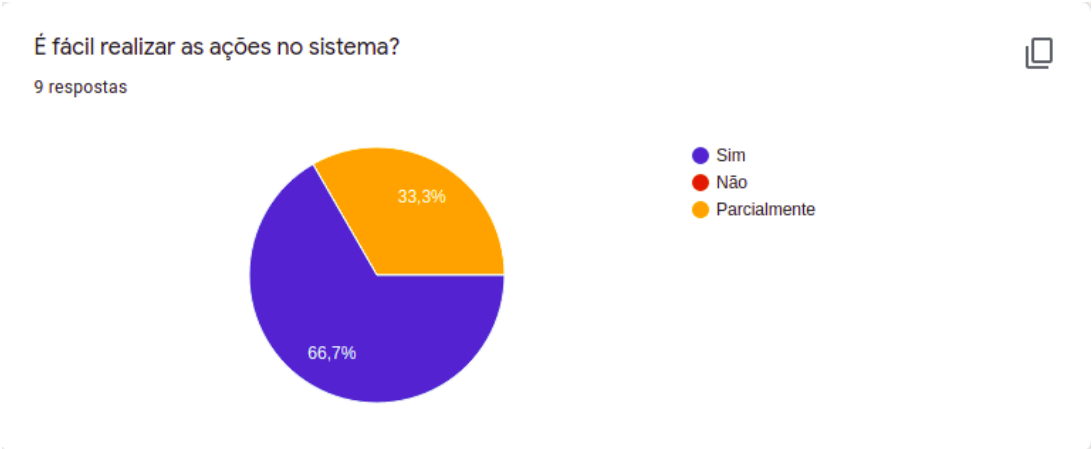
(b) O sistema apresenta falhas?



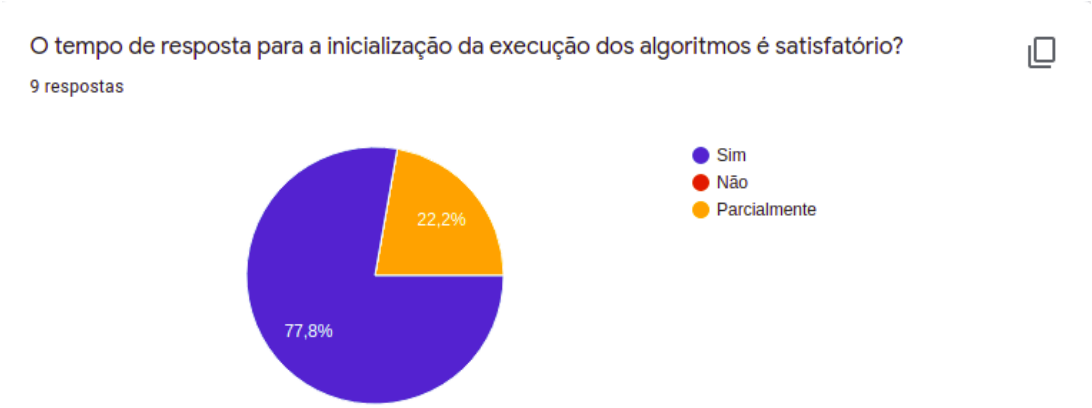
(c) Se você marcou sim na alternativa acima, quais foram as falhas encontradas?

Fonte: O Autor (2021).

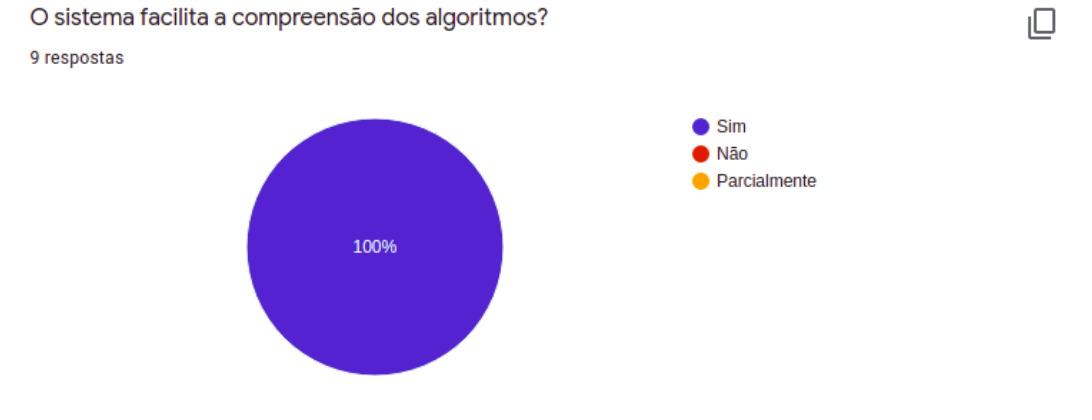
Figura 53 – Perguntas do Questionário - Parte 2



(a) É fácil realizar as ações no sistema?



(b) O tempo de resposta para a inicialização da execução dos algoritmos é satisfatório?



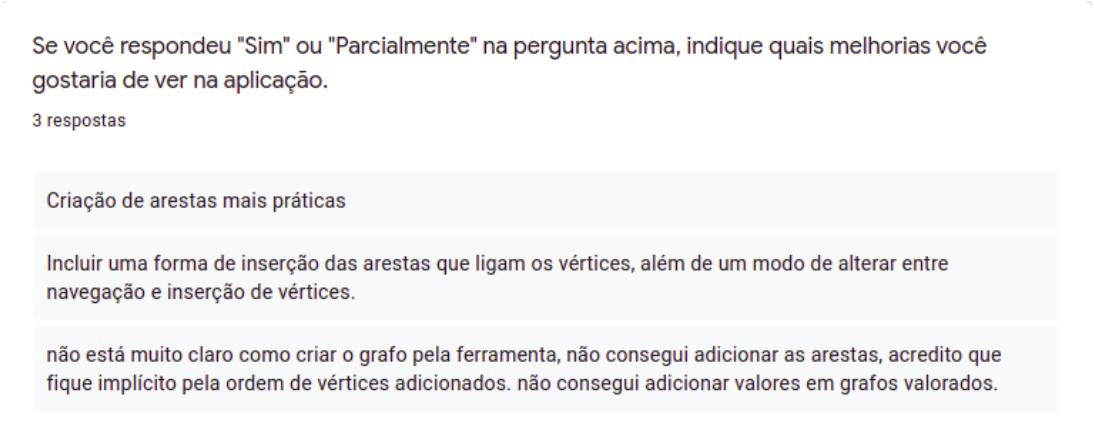
(c) O sistema facilita a compreensão dos algoritmos?

Fonte: O Autor (2021).

Figura 54 – Perguntas do Questionário - Parte 3

(a) A pseudo linguagem usada nos algoritmos é apropriada?

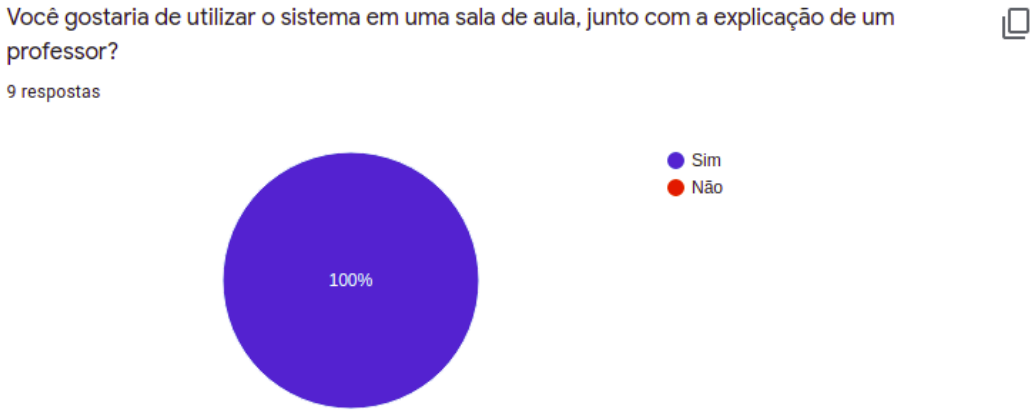
(b) O suporte visual da ferramenta pode ser melhorado?



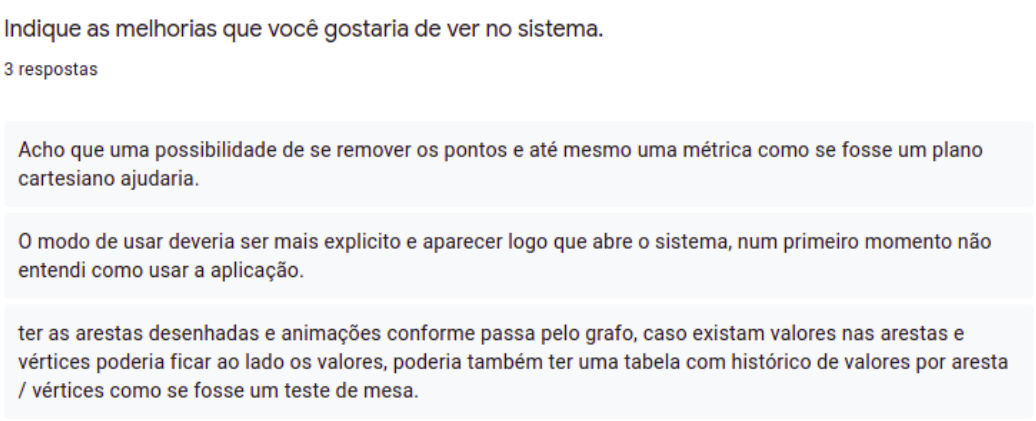
(c) Se você respondeu "Sim"ou "Parcialmente"na pergunta acima, indique quais melhorias você gostaria de ver na aplicação.

Fonte: O Autor (2021).

Figura 55 – Perguntas do Questionário - Parte 4



(a) Você gostaria de utilizar o sistema em uma sala de aula, junto com a explicação de um professor?



(b) Indique as melhorias que você gostaria de ver no sistema.

Fonte: O Autor (2021).