Desenvolvimento de um aplicativo para auxiliar na busca simples e rápida de mecânicos próximos.

*Análise e Desenvolvimento de Sistemas*

## Rafael Figueiredo Junior ∗ Matheus Henrique de Mello†

Luiz Fernando Feriato Pereira dos Santos ‡ Elaine Pasqualini§

Resumo

Este trabalho tem como objetivo o desenvolvimento de um aplicativo para auxiliar usuários na busca de mecânicos próximos de seu local , visando ajudar quando o veiculo começa a falhar ou se encontra em estado de emergência em um município que não é de conhecimento do usuário .Para tal, será usada a metodologia de desenvolvimento *Scrum* e as tecnologias *Typescript*, *HTML*, *CSS*, biblioteca React, Firebase e API do Google Maps. O trabalho resultou na ferramenta PWA “getMech” que permite que usuários possam buscar mecânicos próximos com base na sua localização atual .Concluiu-se que a ferramenta pode ajudar na agilização da busca rápida em situações de emergência do veículo ou em municípios desconhecidos .

**Palavras-chave**: Lógica de Programação. Ferramenta computacional. Estruturas de Dados.

# Abstract

This work aims to develop an application to assist users in the search for mechanics close to their location , in order to help when the vehicle starts to fail or is in a state of emergency in a municipality that is not known to the user . , the Scrum development methodology and Typescript, HTML, CSS, React Native framework, Firebase and Google Maps API technologies will be used. The work resulted in the “getMech” tool that allows users to search for nearby mechanics based on their current location. It was concluded that the tool can help speed up the quick search in vehicle emergency situations or in unknown cities.

**Keywords**: Programming logic. Computational Tool. Data Structures.

**Data de submissão e aprovação**: elemento obrigatório. Indicar dia, mês e ano

∗ graduando em Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologia de Ourinhos, rafael.figueiredo7@fatec.sp.gov.br

† graduando em Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologia de Ourinhos, matheus.mello4@fatec.sp.gov.br

‡ graduan fatec do em Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologia de Ourinhos, luiz.santos158@fatec.sp.gov.br

§ doutora em Educação pela Universidade Estadual Paulista, UNESP e atualmente é docente na Faculdade de Tecnologia de Ourinhos, elaine.pasqualini@fatecourinhos.edu.br

# Introdução

Segundo o motorista Gabriel Bialli(2018), que ficou com seu carro parado na Marginal Pinheiros em horário de pico e sem nenhum contato de guincho ou mecânico , teve que passar o sufoco de horas e horas tentando arrumar algum jeito de conseguir números de oficinas por perto .

Situações como essa podem colocar vidas em risco , reforça Clayton Henrique dos Santos , sargento da Polícia Militar , acostamento é só pra situação emergencial. Se parar no acostamento, tem que desembarcar do veículo de imediato e posicionar o triângulo a cerca de 30 metros do veículo.

Motoristas enfrentam em seu dia a dia , diversos problemas com seu veículo , uma situação de viagem por exemplo, em um munícipio em que o motorista não conhece , ou já esteve ali mas não se recorda direito sobre o local , seu carro começa a apresentar falhas e de repente, para no meio da avenida, desesperado ele tenta encontrar em seu smartphone ,contatos ou números de oficinas mecânicas para lhe socorrer , mesmo que ele encontre , ele vai levar um bom tempo para encontrar e vai ter que ficar esperando ali com seu veículo.

Dessa forma, esse projeto tem como objetivo o desenvolvimento de uma ferramenta mobile para auxiliar motoristas que se encontram nessas situações. Esta ferramenta permitirá que os usuários com base em sua localização , possa efetuar a busca simples e rápida de oficinas mecânicas ou mecânicos , próximos de seu local atual , fornecendo informações, como dados para entrar em contato endereço , fotos e a avaliação da oficina.

# Revisão Bibliográﬁca

Este capítulo descreve sobre reprovação em lógica e programação, abstração, lógica, estrutura de dados, trabalhos correlatos, entre outros assuntos.

## 2.1 Reprovação e desistência em cursos da área da computação

Os dados disponibilizados pelo INEP de 2018, mostram que foram matriculados 333.996 alunos em 2018 em cursos relacionados a Computação e Tecnologias da Informação e Comunicação (TIC) e vê-se que Sistemas da Informação é o curso mais procurado, tendo 172.954 matriculados, representando 51,78% do total (INEP, 2019).

Em relação aos cursos da área da computação, observando o número de alunos concluintes (43.562), matrículas trancadas (69.898), desvinculadas (107.247) e transferências (6.939), percebe-se que a maioria não completou, sendo essa característica ainda mais expressiva no curso de Ciência da Computação, com apenas 6.657 concluintes, contrastando com 11.566 matrículas trancadas, 15.995 matrículas desvinculadas e 1.997 transferências (INEP, 2019).

Um estudo realizado por Souza, Petró e Gessinger (2017) que analisou as produções acadêmicas entre os anos 2000 a 2011 revela que as maiores causas para a desistência de um curso são: falta de condições financeiras, influência familiar, questão vocacional, reprovações em disciplinas que requerem conhecimento matemático, qualidade do curso, localização da instituição, trabalho, idade do aluno, entre outros fatores. Além disso, verifica-se que a questão vinculada à aprendizagem em matemática está entre as principais causas de evasão em cursos superiores.

Em relação aos cursos superiores na área de computação, a maioria possui em sua grade diversas disciplinas voltadas para cálculos matemáticos e para desenvolvimento de algoritmos, sendo que ambos requerem uma bagagem de conhecimento para o sucesso nessas matérias. Como a matéria de algoritmos é determinante para a área de computação, o insucesso do aluno em matérias com essa finalidade acaba dificultando ou até mesmo impedindo a continuidade dos alunos no curso (HOED, 2019).

Para Valaski e Paraiso (2012) as disciplinas ligadas à programação de computadores tem um alto índice de reprovação, pois é geralmente o primeiro contato que o aluno ingressante tem com instruções computacionais e há a necessidade de desenvolver um raciocínio lógico para a resolução de um problema.

Outro agravante observado por Júnior e Rapkiewicz (2004) é em relação a dificuldade de abstração algorítmica por parte do aluno, em que o discente apresenta problemas em assimilar as abstrações envolvidas nos processos de ensino e aprendizagem de computação.

## 2.2 Abstração

Conforme Brookshear (2013), o papel da abstração é permitir que o usuário compreenda um dispositivo complexo sem a necessidade de entender os detalhes que o compõe, ou seja, transformar algo difícil em fácil de ser utilizado, como é o caso de um computador, um automóvel e até mesmo um forno de micro-ondas. Todos esses exemplos são formados por componentes, que por sua vez, são formados por componentes menores ainda e assim sucessivamente que no final sempre formam uma entidade única. Cada componente representa um nível de abstração no qual o uso do componente e isolado de seus detalhes internos.

Por meio da abstração é possível analisar, construir e gerenciar sistemas computacionais grandes e complexos, que seriam confusos caso fossem vistos em sua totalidade com um alto nível de detalhamento de seus componentes. Para cada nível de abstração, o sistema é visto em termos de componentes, chamados de ferramentas de abstratas, cujos detalhes internos são ignorados, permitindo entender como cada componente interage com outros de mesmo nível e como o conjunto forma um componente de nível mais alto, facilitando seu entendimento (BROOKSHEAR, 2013).

Vale ressaltar que a abstração tem um papel importante no avanço da ciência da computação, pois permite que pesquisadores foquem sua atenção em áreas particulares dentro de uma área complexa. Além disso, permite que equipamentos computacionais sejam construídos em níveis de ferramentas abstratas, assim como no desenvolvimento de sistemas de software que são realizados de forma modular, em que cada módulo é visto como uma ferramenta abstrata em módulos maiores (BROOKSHEAR, 2013).

## 2.3 Algoritmos

O algoritmo é o conceito mais fundamental da ciência da computação. Basicamente é um conjunto de passos ordenados que define como uma tarefa é executada. Antes que uma máquina execute uma determinada tarefa, é necessário que um algoritmo seja pensado e traduzido para a linguagem de máquina para que assim ela a execute (BROOKSHEAR, 2013).

O estudo de algoritmos iniciou-se muito antes do desenvolvimento de qualquer linguagem de programação ou de computadores, começando como um objeto da matemática. O objetivo dos matemáticos era descobrir um conjunto único de diretrizes que descrevessem como todos os problemas de um determinado tipo poderiam ser resolvidos. Um exemplo é o Algoritmo de Euclides que permite calcular o máximo divisor comum entre dois números inteiros positivos. Uma vez descoberto um algoritmo que execute dada tarefa, o próximo passo é traduzi-lo para uma linguagem de programação para que assim a máquina execute tais instruções (BROOKSHEAR, 2013).

## 2.4 Lógica Computacional

A palavra lógica está normalmente associada apenas à matemática, deixando um pouco de lado sua aplicabilidade e sua relação com as demais ciências. A lógica ensina a usar corretamente as leis do pensamento por meio da correção do pensamento, pois uma de suas preocupações é determinar quais operações são válidas (FORBELLONE; EBERSPÄCHER, 2005).

Como assegura Forbellone e Eberspächer (2005), pode-se dizer que a lógica de programação consiste em desenvolver uma sequência lógica que tem como finalidade atingir um determinado objetivo. Nesse contexto, pode-se dizer que uma sequência lógica é determinada por um algoritmo, sendo assim, ela tem como objetivo auxiliar o programador a desenvolver uma solução para um problema. Com essa definição fica claro que o programador deve aprender lógica de programação, pois ela garante ao iniciante conhecimentos para aprender qualquer linguagem de programação posteriormente.

Logo, após definir um algoritmo que resolva um problema, a segunda etapa é sua codificação, no entanto, existe uma técnica que tem como foco estabelecer uma sequência de operações a serem efetuadas em um programa que é preparar um diagrama de blocos para demonstrar sua linha de raciocínio lógico e como consequência conseguir codificá-la em qualquer linguagem de programação a princípio sem problemas (MANZANO; OLIVEIRA, 1997).

## 2.5 Estruturas de Dados

Estrutura de dados é o ramo da computação que estuda os vários tipos de organização de dados visando atender os diferentes requisitos de processamento, definindo sua organização e métodos de acesso. Por definição, estrutura de dado é um conjunto básico de dados primitivos (inteiro, real, caracter e lógico) que quando agrupados formam estruturas que dão origem a vetores (matrizes unidimensionais), matrizes (com mais de uma dimensão) ou registros (BEZERRA, 2018).

Segundo Szwarcfiter e Markezon (2010), a escolha da estrutura correta para a resolução de um problema depende diretamente de algoritmos para manipulá-la da maneira mais eficiente, pois para resolver um problema é necessário decidir qual estrutura é mais adequada para sua solução. Além disso, essa decisão é tomada de acordo com as operações que serão realizadas sobre esses dados, já que o que difere uma estrutura da outra é a forma como ela trata a informação.

Logo, seu eo stud é necessário para que se possa identificar e desenvolver modelos matemáticos abstratos, assim como a resolução de problemas que envolvam estruturas de dados (ZIVIANI, 1999).

## 2.6 *Softwares* Educativos

Segundo Valente (1997), *software* educativo é aquele que transmite conhecimento por meio eletrônico, sendo para isso mais usado o computador, seguindo a abordagem de auxiliar o processo pedagógico tradicional. Esse método de transmissão se baseia em um modelo que diz que o conhecimento é retido por meio da experiência, ou seja, memorização. Os *softwares* que se baseiam nesse modelo para transmitir conhecimento são, em geral, categorizados em tutoriais, *software* de exercício-e-prática e jogos.

Valente (1997) afirma que um computador pode substituir um professor na tarefa de transmitir conhecimento, no entanto, vê-se que o computador não pode instruir um profissional para o mundo, pois este não tem as habilidades interpessoais exigidas de um pedagogo.

## 2.7 Ferramentas de Visualização

Soares et al. (2004) dizem que os alunos procuram abstrações por meio de desenhos e visualizações mentais para entender algoritmos complexos, sendo interessante aproveitar essa técnica para algoritmos mais gerais. É visto que a visualização do algoritmo em execução permite além de sua compreensão, a obtenção de detalhes não vistos em sala de aula.

McGrath e Brown (2005) descrevem que ferramentas visuais se tornaram essenciais para demonstrar assuntos complexos e comunicar conceitos sutis. A visualização permite que pessoas experientes consigam explicar com clareza suas ideias e pessoas leigas possam entendê-las com menos dificuldade. McCormick, DeFanti e Brown (1988) ainda dizem que a visualização pode ajudar na análise de dados, percebendo padrões que uma máquina não perceberia por si só.

## 2.8 Trabalhos Correlatos

Na literatura existem diversos trabalhos em relação à visualização de estruturas de dados e algoritmos, sendo os mais importantes:

### 2.8.1 *Incense*

*Incense* é um sistema que permite a geração de imagens de estruturas de dados em tempo de execução, sendo antes necessário especificar a variável que deseja visualizar e o formato da figura (MYERS, 1983). A geração de imagens é feita em tempo real e tem como o objetivo facilitar a depuração e explicação de programas. Ele é, no entanto, muito antigo e só permite a visualização de programas escritos na linguagem de programação *Mesa*.

### 2.8.2 *Balsa*

*Balsa* foi uma ferramenta usada nos laboratórios da Universidade *Brown* para auxiliar no ensino de disciplinas como introdução à programação e estruturas de dados, na pesquisa de desenho e análise de algoritmos e depuração de programas. Brown e Sedgewick (1984) narram que a ferramenta faz simulações gráficas em tempo de execução. Os usuários da ferramenta não escrevem os programas diretamente, mas passam estes a um *scriptwriter* que os traduz e coloca e em execução e usa as animações já construídas.

### 2.8.3 *JAWAA*

Segundo Pierson e Rodger (1998), *JAWAA* é um sistema escrito em *Java* que permite a criação de animações de estruturas de dados no navegador. A ferramenta permite a criação e movimento de objetos e estruturas de dados simples. Para usá-la é necessário gerar um *script* e exportá-lo para a *applet* do sistema. O *script* é simples, precisando de apenas uma linha para criar, exibir e fazer operações, não sendo necessário experiência em *Java* para escrevê-los.

### 2.8.4 *Data Structure Visualizations*

*Data Structure Visualizations* é uma ferramenta web, desenvolvida por *Galles* (2011), para a visualização de estruturas de dados e algoritmos. A ferramenta tem algoritmos e estruturas de dados para serem estudados e seu uso é bem intuitivo, sendo apenas necessária uma breve introdução para começar a usar. A ferramenta, no entanto, dá pouca liberdade ao usuário (principalmente em algoritmos de grafo) e tem uma interface antiga.

### 2.8.5 *Visualgo*

*Visualgo* é uma ferramenta web criada por Halim (2015) para a visualização de estruturas de dados. A ferramenta tem uma interface moderna e grande quantidade de algoritmos, sendo a maioria deles focados em competições de programação. Ele ainda permite acompanhar a execução dos algoritmos linha por linha. O problema dessa ferramenta é sua complexidade, já que embora exista um guia de utilização ao selecionar cada estrutura e algoritmo, este é longo e todo em inglês.

### 2.8.6 *DataViz* (*Golang*)

Segundo Khan (2018), *DataViz* (*Golang*) é uma biblioteca que permite visualizar diversas estruturas de dados e salvar seu estado em imagens. Para usar tal biblioteca, é necessário baixar alguns programas e desenvolver o programa que implementará a imagem em *Golang*.

Embora tenha uma ideia interessante, essa biblioteca é difícil de utilizar no sistema operacional *Windows* e salva imagens estáticas do estado da estrutura, não mostrando sua animação.

### 2.8.7 Quadro comparativo

Após a descrição de alguns trabalhos correlatos foi criado um quadro comparativo das ferramentas computacionais.

Figura 1 - Quadro comparativo das ferramentas computacionais

Fonte: autores

Analisando o quadro comparativo, vê-se que *Visualgo* é a única aplicação que atende os requisitos da Figura 1, elaborada pelos autores. Ainda se percebe que apenas esta ferramenta tem suporte a língua portuguesa, apesar de não ser completo, pois nem todas as instruções são traduzidas.

Outro ponto importante a ser percebido é que apenas três sistemas de visualização de estruturas de dados estão atualmente disponíveis, sendo que dessas, apenas um continua a receber suporte.

Dessa forma, identificou-se a importância da implementação de um sistema de visualização de algoritmos acessíveis aos estudantes ingressantes na área de computação para desenvolver a lógica, sendo este um sistema, embora mais simples e com menos estruturas comparado ao *Visualgo* e *Data Structures Visualization*, intuitivo e suficiente para dar uma base aos estudos de estruturas de dados.

# Materiais e Métodos

Este capítulo descreve os materiais e procedimentos para a conclusão do trabalho.

## 3.1 Ferramentas e Tecnologias

Para o desenvolvimento da aplicação proposta foi usado o *HTML*, *CSS*, *Javascript*, *JQuery* (biblioteca *Javascript* com recursos que facilitam a manipulação de elementos *HTML*, eventos e principalmente animação que foi útil no desenvolvimento de aplicação, além de ser compatível com a maioria dos navegadores da atualidade) e o *Bootstrap*, um *framework* de *CSS* que tem como principal objetivo facilitar a manipulação de elementos de uma tela e tornar a aplicação responsiva. Também foi necessário o uso do *runtime* de *Javascript Node.js* e seu *framework* *Express.js*.

Para a edição dos programas foi usado *Sublime Text*, que é um editor de texto que facilita na edição do código fonte.

Segundo Haughee (2013), *Sublime Text* é um editor de textos para edição de arquivos. Suas funções embutidas mostram seu diferencial, sendo elas o realce de sintaxe, o reconhecimento automático de tipos de arquivos, macros e *plugins* para facilitar a codificação.

*Javascript* é, segundo Haverbeke (2018), uma linguagem de programação que teve como objetivo inicial adicionar páginas web ao navegador *Netscape* *Navigator*. Atualmente *Javascript* é usado para processar eventos em páginas web sem precisar recarregá-la.

Criado por Tim Berners-Lee para comunicação e disseminação de pesquisas entre ele e seus colegas, o *HTML* (*Hypertext Markup Language*) é uma linguagem de marcação que é utilizada para desenvolver *websites*, sendo sua versão mais recente o *HTML5* (*HTML*, 2019).

O *CSS* (*Cascading Style Sheets*) é uma linguagem de estilo usada para adicionar estilos a um documento web, sendo sua versão mais recente o *CSS3*. Assim como o *HTML5*, o *CSS3* é compatível com grande parte dos navegadores da atualidade (*CSS*, 2019).

*Bootstrap* é um *framework* que permite o desenvolvimento de telas responsivas, ou seja, um *website* que se adapta a diferentes tamanhos de tela, permitindo uma experiência agradável a diversos dispositivos (*W3SCHOOLS.COM*, 2020). Além de facilitar o desenvolvimento de aplicações destinadas a dispositivos móveis, facilita a organização de componentes de uma tela (*BOOTSTRAP*, 2020).

Segundo Young et al. (2017), *Node.js* é um runtime de *Javascript* que permite que a linguagem deixasse de ser apenas uma linguagem auxiliar de *browser*. Com o uso de *Node.js* e outros pacotes escritos para a plataforma, foi possível usar a mesma linguagem em servidores e navegadores. Um importante pacote de *Node.js* é o *Express.js* que, segundo Hahn (2016), é um *framework* voltado a servidores que permite, como o *JQuery*, facilitar o desenvolvimento, usando para isso sua sintaxe simples e sua fácil extensibilidade.

## 3.2 Procedimentos

O projeto foi realizado com base em metodologias ágeis, focado no *framework* *Scrum*.

Segundo Sommerville (2011), ambientes em que mudanças de requisitos são comuns e entregas rápidas são necessárias, é essencial o uso de uma metodologia de desenvolvimento que permita essas características, sendo a partir disso que surgiram as metodologias ágeis.

De acordo com Sommerville (2011), os modelos de desenvolvimento tradicionais, como o método cascata, passam por fases bem definidas e, caso haja uma mudança no requisito ou descoberta de problemas, se torna custoso alterar documentos e artefatos já terminados.

Embora existam diversas metodologias de desenvolvimento ágil, elas compartilham algumas características em comum, como documentação mínima, desenvolvimento e entregas incrementais, uso de ferramentas para facilitar e agilizar o desenvolvimento e *feedback* constante do cliente (SOMMERVILLE, 2011).

### 3.1.1 *Scrum*

Segundo Sutherland e Schwaber (2017), *Scrum* é um *framework* desenvolvido para auxiliar no gerenciamento de projetos ágeis e tem uma característica importante que é o compartilhamento de responsabilidades por todos os membros do projeto.

Sutherland e Schwaber (2017) dizem que o *Scrum* é sustentado por três pilares:

1. Transparência: os responsáveis pelos resultados visualizam aspectos importantes do processo;
2. Inspeção: validação frequente de artefatos e progresso das *sprints* a fim de verificar possíveis erros e desvios;
3. Adaptação: suporte às mudanças nos artefatos, facilitando o desenvolvimento de projetos em que requisitos mudam constantemente.

Sutherland e Schwaber (2017) particionam o time *Scrum* em três papeis:

1. *Product Owner*: profissional responsável por levantar os requisitos, gerenciar o *backlog* do produto e garantir o entendimento do produto ao time de desenvolvimento;
2. Time de desenvolvimento: responsável por liberar um produto parcial ao final de cada *sprint*. Esse time é autogerenciável, multifuncional e geralmente pequeno a fim de manter a integração e sinergia do time. Projetos grandes têm vários times de desenvolvimento, sendo cada um responsável por um requisito específico;
3. *Scrum Master*: profissional responsável por criar uma ponte entre o *Product Owner* e o time de desenvolvimento. Este deve entender os requisitos e passar de maneira clara para o time de desenvolvimento. O *Scrum Master* tem diversas tarefas, como garantir que o *Product Owner* saiba organizar o *backlog* e garantir que o time de desenvolvimento saiba se autogerenciar.

Segundo Sommerville (2011), o *Scrum* é dividido em três fases, o planejamento geral do sistema, a série de *sprints* e o encerramento do projeto, sendo que destas a série de *sprints* é a mais importante.

Para Sutherland e Schwaber (2017), *sprint* é um evento que dura de 2 a 4 semanas, sendo este dividido em cinco fases:

1. planejamento da *sprint*: é um trabalho colaborativo entre todos os membros do time *Scrum* em que é definido o objetivo da *sprint* e como será realizado o trabalho;
2. reuniões diárias: é um evento de 15 minutos no qual o time de desenvolvimento planeja o dia. Nesse evento são feitas perguntas que ajudam o time a entender o progresso da *sprint* e resolver impedimentos no desenvolvimento;
3. trabalho de desenvolvimento: o desenvolvimento em si;
4. revisão da *sprint*: é um evento realizado no final de cada *sprint* e tem como objetivo obter o *feedback* do time *Scrum* e dos *Stakeholders*;
5. retrospectiva da *sprint*: é o momento em que o time *Scrum* visualiza o que foi feito na *sprint* e discute o que pode ser feito para melhorar o funcionamento das próximas *sprints*.

# 4 Resultados

A ferramenta “Visualiza Estruturas” é uma aplicação gratuita e direcionada aos alunos de computação e informática que permite acompanhar, passo a passo, o funcionamento das estruturas de dados.

No entanto, esse programa possuirá, diferentemente dos outros ambientes de ensino discutidos no capítulo 2, algumas características, tais como:

* + Interface amigável;
  + caixa de texto explicativa de cada estrutura de dados;
  + controle da velocidade da animação gráfica;
  + um *highlight* que destaca a linha que está em execução enquanto ocorre a animação;
  + vantagem de ser acessível a qualquer momento para atender alunos que estudam algoritmos e estruturas de dados básicas.

­A ferramenta possui em sua página inicial uma barra de menu com 4 tipos de estruturas de dados (vetor, lista encadeada, fila e pilha) que direcionam o usuário para a página escolhida (Figura 2).

Figura 2 - Menu principal

Interface gráfica do usuário

Descrição gerada automaticamente

Fonte: autores

A Figura 3 mostra o funcionamento da estrutura vetor. O usuário poderá indicar um índice e um valor e o sistema demonstrará uma animação simulando a inserção desse valor no determinado índice. Esse módulo mostra uma das características mais importante dos vetores, inserção e recuperação de valores com complexidade .

A animação ocorrerá quando o índice e o número forem inseridos pelo usuário. É colocado esse valor em seu determinado índice, sobrescrevendo o elemento que estava anteriormente neste índice. Junto à animação é possível o acompanhamento da execução do código fonte em pseudocódigo.

Figura 3 - Funcionamento da estrutura Vetor

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Fonte: autores

A Figura 4 mostra o funcionamento da estrutura de dados lista encadeada, sendo esta uma estrutura que armazena um valor e uma referência ao próximo item da lista. As características mais importantes da lista são as possibilidades de inserir elementos em qualquer posição da lista e sua alocação de memória dinâmica. Inserir e recuperar elemento de uma lista tem complexidade .

A animação é similar à animação do vetor, no entanto deixa explícita a possibilidade de colocar elementos em uma posição já ocupada sem sobrescrever este.

Figura 4 - Funcionamento da estrutura Lista Encadeada

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Fonte: autores

A Figura 5 exibe o funcionamento da estrutura de dados pilha. Esta estrutura tem como principal característica a maneira de inserção e recuperação, onde o elemento inserido ocupa o último item da pilha, sendo este comportamento similar a uma pilha de pratos. Esta estrutura é usada para implementar a pilha de execução das linguagens de programação e pode ser implementada com memória fixa, usando como base um vetor, ou com memória dinamicamente alocada, usando como base uma lista encadeada.

Figura 5 - Funcionamento da estrutura Pilha

Interface gráfica do usuário, Texto, chat ou mensagem de texto, Site

Descrição gerada automaticamente

Fonte: autores

A Figura 6 mostra o funcionamento da estrutura de dados fila. Esta estrutura é similar a uma fila de supermercado, onde o primeiro elemento a entrar na fila será o primeiro elemento a ser atendido (recuperado). Similar à pilha, esta estrutura pode ser implementada com memória fixa ou memória dinamicamente alocada.

As pilhas e filas implementadas no sistema tem memória dinamicamente alocadas e, esta escolha deve-se ao fato destas serem úteis em situações em que o tamanho da estrutura não é conhecido previamente e permitem a visualização do deslocamento das referências ao próximo item da estrutura.

Figura 6 - Funcionamento da estrutura Fila

Interface gráfica do usuário, Texto, Aplicativo, Site

Descrição gerada automaticamente

Fonte: autores

# 5 Considerações finais

O *software* desenvolvido permitiu identificar o funcionamento das estruturas de dados, bem como deu a possibilidade de analisar e copiar suas implementações nas linguagens de programação *C, Golang, Kotlin, Java, Javascript* e *Python*. É necessário alertas que essas implementações não são recomendadas a fim de desenvolver um *software*, apenas para estudo e comparações entre linguagens. Linguagens como Golang, Kotlin e Java já possuem uma biblioteca padrão bem completa que abrange estas estruturas.

O módulo para copiar o código tem como principal objetivo comparar as linguagens e suas implementações, levando em consideração seus paradigmas e idiomas.

O ensino de estruturas de dados é difícil, mas essa dificuldade pode ser mitigada usando abstração e ferramentas de visualização.

O programa desenvolvido buscou demonstrar de maneira simples os passos básicos usados nas operações das estruturas de dados vetor, lista, fila e pilha. Esses passos foram animados e sincronizados com o pseudocódigo ao lado. A velocidade dessas animações foi parametrizada.

O desenvolvimento foi realizado por meio de metodologias ágeis, usando o *framework Scrum*.

Concluiu-se que a ferramenta poderá ajudar a entender esse assunto e futuramente poderá ser aplicada a alunos.

# Referências

BEZERRA, C. A. **Introdução às estruturas de dados**. Curitiba: UFPR, 2018.

BOOTSTRAP. Disponível em: <https://getbootstrap.com/>. Acesso em: 09 jul. 2020.

BROOKSHEAR, J. G. **Ciência da Computação**: Uma visão abrangente. 11. ed. Porto Alegre: Bookman, 2013.

BROWN, M. H.; SEDGEWICK, R. A system for algorithm animation. In: **SIGGRAPH**. Minneapolis: ACM Press, 1984. p. 177–186. Disponível em: <https://dl.acm.org/doi/pdf/10.1145/800031.808596>. Acesso em: 04 abr. 2020.

CSS. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/CSS>. Acesso em: 20 nov. 2019.

ENGELBRECHT, A. D. M. **Uma ferramenta para auxiliar no ensino de estruturas de dados como tipo de dado abstrato**. Tese (Doutorado) — Faculdade de Engenharia Elétrica e de Computação, Universidade de Campinas, Campinas, 2003.

FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2005.

GALLES, D. **Data Structure Visualizations**, 2011. Disponível em: ⟨https://www.cs.usfca.edu/∼galles/visualization/about.html⟩. Acesso em: 17 out.2019.

HALIM, S. Visualgo: Visualising data structures and algorithms through animation. **Olympiads in Informatics**, v. 9, p. 243–245, 2015. Disponível em: <https://ioinformatics.org/files/volume9.pdf>. Acesso em: 04 abr. 2020.

HAHN, E. M. **Express in Action**: Writing, building, and testing Node.js applications. Shelter Island: Manning Publications, 2016.

HAUGHEE, E. **Instant Sublime Text Starter**. Birmingham: Packt Publishing, 2013.

HAVERBEKE, M. **Eloquent JavaScript**. 3. ed. San Francisco: No Starch Press, 2018.

HENRIQUES, J.; MENDES, A. J. Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores. Educação, Formação e Tecnologia, v. 1, p. 93–103, 2008.

HOED, R. M. Evasão nos cursos superiores de computação do Brasil: Uma análise usando mineração de dados e técnicas estatísticas. **Educação Brasil**, v. 4, p. 167, 2019. Disponível em: <https://www.academia.edu/40907938/Evas%C3%A3o\_nos\_cursos\_superiores\_de\_Computa%C3%A7%C3%A3o\_do\_Brasil\_uma\_an%C3%A1lise\_usando\_minera%C3%A7%C3%A3o\_de\_dados\_e\_t%C3%A9cnicas\_estat%C3%ADsticas?auto=download>. Acesso em: 04 abr. 2020.

HTML: Linguagem de marcação de hipertexto. Disponível em: ⟨https://developer.mozilla.org/pt-BR/docs/Web/HTML⟩. Acesso em: 20 nov. 2019.

INEP - INSTITUTO NACIONAL DE ESTUDOS E PESQUISAS EDUCACIONAIS ANÍSIO TEIXEIRA. **Sinopse Estatística da Educação Superior 2018**. Brasília: Inep, 2019. Disponível em: <http://portal.inep.gov.br/web/guest/sinopses-estatisticas-da-educacao-superior>. Acesso em: 11 abr. 2020.

JÚNIOR, J. C. R. P.; RAPKIEWICZ, C. E. O Processo de Ensino e Aprendizagem de Algoritmos e Programação: Uma Visão Crítica da Literatura. In: **Workshop de Educação em Computação e Informática do estado de Minas Gerais**. Belo Horizonte, 2004. Disponível em: <https://docplayer.com.br/60837282-O-processo-de-ensino-aprendizagem-de-fundamentos-de-programacao-uma-visao-critica-da-pesquisa-no-brasil.html>. Acesso em: 04 abr. 2020.

KHAN, A. **Introducing DataViz a data-structure visualization library for Golang**. Disponível em: ⟨https://medium.com/@Arafat./introducing-dataviz-a-data-structure-visualization-library-for-golang-f6e60663bc9d⟩. Acesso em: 12 mar. 2020.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. D. **Estudo Dirigido de Algoritmos**. 7. ed. São Paulo: Érica, 1997.

MCCORMICK, B. H.; DEFANTI, T. A.; BROWN, M. D. Visualization in scientific computing. SIGBIO Newsl., **ACM**, Nove Iorque, v. 10, n. 1, p. 15–21, mar. 1988. Disponível em: <https://dl.acm.org/doi/pdf/10.1145/43965.43966>. Acesso em: 04 abr. 2020.

MCGRATH, M. B.; BROWN, J. R. Visual learning for science and engineering. **IEEE Computer Graphics and Applications**, v. 25, n. 5, p. 56–63, set-out 2005. Disponível em:<http://diglib.eg.org/bitstream/handle/10.2312/14706/WP\_formatted\_final.DL.pdf?sequence=1&isAllowed=y>. Acesso em: 04 abr. 2020.

MYERS, B. A. **Incense:** A system for displaying data structures. In: SIGGRAPH. New York: ACM Press, 1983. Disponível em: <https://dl.acm.org/doi/abs/10.1145/964967.801140>. Acesso em: 04 abr. 2020.

PIERSON, W. C.; **RODGER, S. H. Web-based animation of data structures using jawaa.** In: IGCSE.Atlanta: ACM, 1998. Disponível em: <https://dl.acm.org/doi/pdf/10.1145/274790.274310>. Acesso em: 04 abr. 2020.

SANTOS, R. D.; COSTA, H. Análise de metodologias e ambientes de ensino para algoritmos, estruturas de dados e programação aos iniciantes em computação e informática. **INFOCOMP**, v. 5, n. 1, p. 41–50, 2006. Disponível em: <http://www.dcc.ufla.br/infocomp/index.php/INFOCOMP/article/download/121/106/0>. Acesso em: 04 abr. 2020.

SCANLAN, D. Data-structures students may prefer to learn algorithms using graphical methods. In: **SIGCSE**. St. Louis: ACM, 1987. v. 19, n. 1, p. 302–307. Disponível em: <https://dl.acm.org/doi/10.1145/31726.31776>. Acesso em: 04 abr. 2020.

SOARES, T. C. A. P. et al. Uma proposta metodológica para o aprendizado de algoritmos em grafos via animação não-intrusiva de algoritmos. **III Workshop de Educação em Computação e Informática do Estado de Minas Gerais (WEIMIG 2004)**, Belo Horizonte,2004. Disponível em: <http://www.italost.com/academic/papers/pdf/ring\_weimig2004.pdf>. Acesso em: 04 abr. 2020.

SOMMERVILLE, I. **Engenharia de Software**. 9.ed. São Paulo: Pearson Prentice Hall, 2011.

SOUZA, C. T.; PETRÓ, C. S.; GESSINGER, R. M. Um estudo sobre evasão no Ensino Superior do Brasil nos últimos dez anos. In: II CLABES Pontifícia Universidade Católica Rio Grande do Sul 2012. Madrid, v. 2, p.116-124, 2012. Disponível em: <https://revistas.utp.ac.pa/index.php/clabes/issue/view/64/68>. Acesso em: 22 abr. 2020.

SUTHERLAND, J.; SCHWABER, K. **Guia do Scrum**. Disponível em: <https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 20 mar. 2020.

SZWARCFITER, J. L.; MARKEZON, L. **Estruturas de Dados e seus Algoritmos**. 3. ed. Rio de Janeiro: LTC, 2010.

VALASKI, J; PARAISO, E, C. Limitações da Utilização do Alice no Ensino de Pro-gramação para Alunos de Graduação. In: Simpósio Brasileiro de Informática na Educação, 23., 2012, Rio de Janeiro. **Anais do SBIE**. 2012. p. 26-30. Disponível em: <https://www.researchgate.net/publication/260013117\_Limitacoes\_da\_Utilizacao\_do\_Alice\_no\_Ensino\_de\_Programacao\_para\_Alunos\_de\_Graduacao>. Acesso em: 04 abr. 2020.

VALENTE, J. **O uso inteligente do computador na educação**. Pátio, Porto Alegre, v. 1, n. 1, p.18–21, mai/jun 1997.

YOUNG, A et al. **Node.js in Action**. 2. ed. Shelter Island: Manning Publications, 2017.

ZIVIANI, N. **Projeto de Algoritmos**: Com implementações em pascal e c. 4. ed. São Paulo: Pioneira, 1999.

W3SCHOOLS.COM. **HTML Responsive Web Design**. Disponível em: <https://www.w3schools.com/html/html\_responsive.asp>. Acesso em: 09 jul. 2020.