



**UNIVERSIDADE FEDERAL DO AGreste DE PERNAMBUCO  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**GIUDICELLI ELIAS DA SILVA  
JOÃO BATISTA NETO  
JOÃO GUILHERME BENJAMIN ALVES DE REZENDE**

**PROJETO DE GRAFOS: COMO ENCONTRAR AS MENORES ROTAS  
POSSÍVEIS NO MAPA DE SKYRIM.**

Professor: Igor Medeiros Almeida Vanderlei  
Algoritmos e Estrutura de Dados II, turma 2022.1

**GARANHUNS  
2023**

## INTRODUÇÃO

O projeto ”Skyrim Mapper” é um algoritmo que busca otimizar as viagens no mapa do jogo *The Elder Scrolls V: Skyrim*. Este algoritmo implementa um grafo ponderado bidirecional através de uma GUI (Graphical User Interface).

Buscando uma interpretação da realidade, analisamos o algoritmo CPT (*Cheapest Paths-Tree*) de Dijkstra, e percebemos que é o mais adequado para a nossa situação problema pois: o mapeamento de uma área ou região de maneira a localizar os possíveis caminhos na mesma e por fim localizar o menor caminho entre dois pontos do mapa além de não precisarmos lidar com pesos negativos nos caminhos. Deste modo optamos por utilizar o algoritmo de Dijkstra por ser altamente adequado para essa situação.

Além disso, a utilização dos algoritmo de grafos e Dijkstra para resolver problemas de otimização de rotas pode ter aplicações em diversas áreas, como: logística, transporte, telecomunicações, e entre outras. Portanto, esse projeto pode abranger significativamente situações reais, e trazer maior ênfase e desenvolvimento para a comunidade científica e tecnológica em geral, já que a criação científica colabora de forma gradual para o progresso da sociedade, a tornando mais desenvolvida e completa.

Por fim vale ressaltar que a construção desse projeto passou por uma evolução criativa, onde em seu começo buscava ser principalmente didático e ilustrativo a respeito do funcionamento do algoritmo de Dijkstra. Porém com o seu desenvolvimento tornou-se perceptível a possibilidade de uma representação visual adequada, aprimorada ao seu uso e interativa ao usuário. Assim, permitindo não só um fim educativo mas também a resolução da problemática a que este projeto se dispõe a resolver: achar o menor caminho entre dois pontos no mapa do jogo.

## OBJETIVO

O projeto ”Skyrim Mapper” tem como objetivo calcular a menor distância entre as cidades, vilas ou outros pontos de interesse que podem ser medidos em coordenadas em um mapa. O principal objetivo é garantir a eficiência e o melhor desempenho durante viagens pela província de Skyrim, buscando minimizar o custo das rotas.

Utilizando grafos e o algoritmo de Dijkstra, será possível encontrar a menor distância entre as cidades e vilas em Skyrim, considerando os caminhos mais seguros e eficientes. Com isso, será possível planejar rotas mais otimizadas para viagens, economizando tempo, energia e recursos, além de evitar perigos como monstros e áreas de difícil acesso. Isso permitirá aos usuários planejar suas viagens com maior precisão e segurança.

## METODOLOGIA

Para realizarmos este projeto, seguimos diversas etapas em sua criação, como:

1. Os requisitos necessários para o projeto, como a ideia de escolhermos o mapeamento do jogo *The Elder Scrolls V: Skyrim*, mas que além disso, este projeto também é adequado para situações reais de cálculo da menor distância entre cidades, vilas e quaisquer pontos de interesse que estão mapeados.
2. O escopo do projeto inclui as ferramentas necessárias para seu desenvolvimento, tais como a IDE *IntelliJ IDEA* da Jetbrains, a linguagem de programação Java e outras bibliotecas, como *AWT* e *Swing*.
3. Na implementação foram desenvolvidas as classes e métodos necessários para a construção do mapa interativo de Skyrim, utilizando a estrutura de grafos, um identificador de posição do mouse para o uso da interface gráfica, e o algoritmo de Dijkstra para calcular as rotas mais eficientes e seguras entre os pontos de interesse.

## DESCRIÇÃO DO PROJETO

O projeto "Skyrim Mapper" consiste em um software desenvolvido em Java, utilizando majoritariamente as seguintes classes e objetos:

- **skyrim\_map.png**: representa o mundo de Skyrim, com os pontos mais importantes de locais e encontro.
- **Point**: representa um vértice no mapa, contendo um *id* e coordenadas *x,y* para que através deste, identifiquemos a posição dos vértices no mapa.
- **Route**: representa uma rota entre dois locais, com uma distância e um custo de passagem.
- **Skyrim**: representa o grafo ponderado e bidirecionado que é utilizado para armazenar as rotas entre as localidades.
- **Dijkstra**: implementa o algoritmo de Dijkstra para calcular a menor distância entre dois pontos (Point) no grafo.
- **MapPositionsDefiner**: classe responsável por definir as posições dos locais no mapa através dos clicks do mouse.
- **Main**: classe principal que utiliza as classes acima para criar o mundo de Skyrim, adicionar localidades e rotas, e calcular a menor distância entre elas.

## DESENVOLVIMENTO

O projeto ”Skyrim Mapper” consiste em uma aplicação em Java para o cálculo de caminhos mínimos em grafos, independente da quantidade de aresta, sendo o conjunto de aresta uma quantidade  $n$  finita  $\in \mathbb{N}^*$ . Pode-se representar a notação de grafo como:  $G(V, E)$ , onde  $V$  é um conjunto finito de vértices e  $E$  é um conjunto finito de arestas. Assim, adaptando a notação podemos representar o grafo de Skyrim da seguinte forma:  $S(P, R)$ , sendo  $S$  o grafo de Skyrim,  $P$  o conjunto de vértices como pontos, e  $R$  o conjunto de aresta como rotas. Para a resolução do problema, utiliza-se o algoritmo de Dijkstra, que é capaz de encontrar o caminho mínimo em um grafo ponderado. O peso das arestas é automaticamente calculado com base na distância euclidiana entre 2 pontos, já que a aplicação se trata de um mapeamento de rotas.

Assim, ao utilizar o mapa de Skyrim e o algoritmo de Dijkstra para a otimização de rotas teve sua origem na ideia de aplicar conceitos de jogos em situações da vida real. O jogo Skyrim é conhecido por sua vasta região, com diversos pontos que o jogador precisa explorar e navegar. Assim, o mapa do jogo pode ser visto como um grafo, onde os pontos importantes de localidade são os vértices e as rotas entre os pontos são as arestas. O objetivo do jogador é encontrar a melhor rota possível para chegar a determinado local.

A partir dessa ideia, é possível relacionar o jogo com situações da vida real, como a logística, o transporte e as telecomunicações. Por exemplo, empresas de logística podem usar o algoritmo de Dijkstra para otimizar rotas de entrega e reduzir custos. Empresas de transporte podem usá-lo para otimizar rotas de meios de transporte, como: ônibus, trens e entre outros. Tornando o transporte mais eficiente. Empresas de telecomunicações também podem usá-lo para otimizar a rota de dados entre servidores, reduzindo o tempo de latência e melhorando a velocidade de transmissão de dados.

Vimos que a ideia era adequada após consultar em aula com o professor, depois de estudarmos a teoria de grafos e Dijkstra, começa-se a implementação do código: a primeira etapa foi a parte mais básica, onde o mapa de Skyrim foi modelado como um grafo, onde cada ponto no mapa é uma representação de uma vértice e cada rota era uma aresta. O algoritmo de Dijkstra foi usado para calcular todas as distâncias de um ponto a outro, e este imprimia a distância de um ponto de origem a um ponto de destino (podendo esse ponto ser uma vila no mapa ou não) e o peso de cada caminho (incluindo a soma entre caminhos menores que resultem em um maior) para qual pudesse desejar ir. Posteriormente, a interface gráfica (GUI) foi desenvolvida, e a classe básica ”Village” foi substituída pela classe ”Point” da biblioteca *AXT* que atendia de maneira mais completa o problema, sendo assim possível o uso desta mesma classe em ambas as situações, assim foi visto que essa classe era a mais adequada para as etapas diferentes da construção do código, e essa mudança foi promissora e eficiente para a conclusão final do projeto.

Quando utiliza-se a GUI, o algoritmo de Dijkstra funciona de forma diferente

da implementação básica, neste o usuário seleciona um botão vermelho para indicar a origem e outro botão vermelho para indicar o destino. O algoritmo então calcula a menor distância, independente do número de arestas entre eles. O que importa é que haja pelo menos uma aresta para se calcular a menor distância, agora com uma aplicação interativa com o usuário, divertida de se usar e útil ao seu propósito.

## FUNCIONALIDADES

O projeto oferece as seguintes funcionalidades:

- Adicionar novos pontos (vértices) ao mapa de Skyrim;
- Adicionar rotas (arestas) entre os pontos existentes;
- Remover rotas (arestas) entre os pontos existentes (essa opção fora da GUI);
- Visualizar todas as vilas e rotas existentes no mapa de Skyrim através da GUI;
- Encontrar a rota mais curta entre duas vilas do mapa de Skyrim, através do algoritmo de Dijkstra;

## COMO UTILIZAR

Para utilizar o software, basta executar a classe Main que se encontra no pacote "SkyrimMapper/Graph/src/" .

Ao executar o programa, uma janela do mapa de Skyrim será exibida para o usuário, seguida de uma série de janelas de tutorial que explicam o uso correto do software. O conteúdo deste tutorial será semelhante ao seguinte:

1. Ao clicar no mapa com o botão esquerdo do mouse, ele criará um botão vermelho, sendo este um vértice do grafo, que pode ser ligado a inúmeros outros vértices.
2. Ao clicar no botão vermelho, ele ficará verde, e assim disponível para criar uma aresta entre dois botões vermelhos. Sendo assim, ao clicar no botão de origem, que ficará verde e posteriormente no botão destino, será criada uma aresta entre eles.

3. Para utilizar o algoritmo de Dijkstra, clique duas vezes no botão vermelho, que ficará verde e, em seguida, azul. Depois disso, escolha o ponto de destino desejado para calcular a menor distância entre os pontos. Então uma linha vermelha será traçada indicando o caminho de menor custo percorrido.
4. Para que as linhas tracejadas em vermelho voltem a ser pretas, é necessário clicar três vezes no mesmo botão, ficando vermelho. E em seguida, basta clicar em qualquer outro botão vermelho para que as rotas voltem a ser exibidas em preto e o algoritmo possa ser utilizado novamente como quiser.

Seguem-se imagens da GUI.

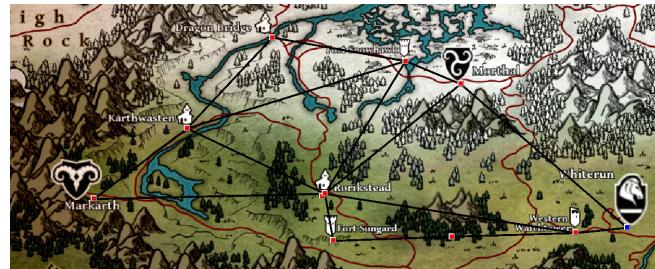


(a) Após clicar em um botão vermelho, ele fica verde, e permite fazer uma aresta a outro ponto.



(b) Após clicar em outro vértice, cria-se a aresta.

Assim, após este breve tutorial de utilização e caso de teste específico funcionando, segue-se adiante uma imagem com uma quantidade maior de pontos, e que de fato está sempre selecionando o menor caminho de um ponto de origem até o seu destino.



(c) Botão azul: apto a receber um vértice destino para calcular o menor caminho.



(d) Menor caminho calculado com sucesso, através do algoritmo de Dijkstra



Figura 1: Mapa vasto com diversos pontos após a aplicação do algoritmo de menor caminho.



Figura 2: Mapa vasto com diversos pontos após a aplicação do algoritmo de menor caminho.

## CONCLUSÃO

Este projeto apresenta uma solução para o problema de encontrar o menor caminho entre dois pontos em um grafo. Utilizando conceitos de Programação Orientada a Objetos, como: encapsulamento, padrões de escrita, classes, objetos e casos de teste unitários. Implementamos uma estrutura de grafo para armazenar os pontos e suas conexões. Além disso, evoluímos o projeto inicialmente simples para uma interface gráfica, utilizando os recursos de *Points* da biblioteca *AWT* para representar os vértices do grafo. Utilizamos o algoritmo de Dijkstra para encontrar o menor caminho entre os pontos e validamos a solução com testes unitários e práticos.

Diante disso, a implementação do projeto agregou diversos conhecimentos em sua construção, o quê a tornou desafiadora, mas enriquecedora, possibilitando a aplicação prática de conceitos teóricos estudados em sala de aula e o estímulo de aprendizado para sua conclusão. Além de possibilitar o desenvolvimento criativo de pensar em um tema problemático e a sua resolução, pudemos mesclar situações fictícias com situações reais, através de uma representação como é feita neste software, pode-se buscar uma solução para o cotidiano.

O teorema de Gödel afirma que, em sistemas formais incompletos, é possível adicionar novos axiomas demonstráveis para tornar o sistema mais completo e incluir mais verdades matemáticas. Assim, o sistema pode ser aprimorado gradativamente com a incorporação de cada vez mais verdades matemáticas.

Ao considerarmos um sistema formal incompleto como o meio científico acadêmico, já que não se conhece todas as verdades e soluções possíveis, cada evolução e criação por mais pequena ou grande que seja, é um passo adiante para que tenhamos mais soluções e melhor compreensão dos problemas cotidianos e da compreensão humana, e assim para que se possa evoluir para um sistema formal científico mais desenvolvido e abrangente.

## REFERÊNCIAS

- MEDEIROS VANDERLEI, Igor. AEDII 12 05 [videoaula]. Youtube, 2022. Disponível em: <https://youtu.be/JPI0YAYV4fQ>. Acesso em: 05 abr. 2023.
- MEDEIROS VANDERLEI, Igor. AEDII 05 05 parte1 [videoaula]. Youtube, 2022. Disponível em: <https://youtu.be/tEBdBg9-SXk>. Acesso em: 05 abr. 2023.
- MEDEIROS VANDERLEI, Igor. AEDII 05 05 parte2 [videoaula]. Youtube, 2022. Disponível em: <https://youtu.be/l3x6OzMqWfo>. Acesso em: 05 abr. 2023.
- R. Adriana Mendonça, "Algoritmo de Dijkstra," Faculdade de Computação da Universidade Federal de Uberlândia, 2011. Disponível em: <https://www.facom.ufu.br/madriana/ED/AlgDijkstra.pdf>. Acesso em: 11 abr. 2023.
- OLIVEIRA, Humberto César Brandão de. Aula 13 – Caminho Mínimo: Grafos Acíclicos. UnifalMG. Acesso em: 10 abr. 2023.
- CORMEN, Thomas H. et al. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012. Capítulo 24.
- FEILOLOFF, Paulo. Algoritmos para grafos: Caminhos mais baratos. Disponível em: [https://www.ime.usp.br/pf/algoritmos\\_para\\_grafos/aulas/cheapestpaths.html](https://www.ime.usp.br/pf/algoritmos_para_grafos/aulas/cheapestpaths.html). Acesso em: 11 abr. 2023.
- FEILOLOFF, Paulo. Algoritmo de Dijkstra. Disponível em: [www.ime.usp.br/pf/algoritmos\\_para\\_grafos/aulas/dijkstra.html](http://www.ime.usp.br/pf/algoritmos_para_grafos/aulas/dijkstra.html). Acesso em: 11 abr. 2023.
- FEILOLOFF, Paulo. Algoritmos para Grafos. Disponível em: [https://www.ime.usp.br/pf/algoritmos\\_para\\_grafos/aulas/shortestpath.html](https://www.ime.usp.br/pf/algoritmos_para_grafos/aulas/shortestpath.html). Acesso em: 11 abr. 2023.
- BAELDUNG. Minimum Spanning vs Shortest Path Trees. Disponível em: <https://www.baeldung.com/cs/minimum-spanning-vs-shortest-path-trees#:text=In%20the%20shortest%20path%20tree,%20all%20edges%20are%20selected>. Acesso em: 11 abr. 2023.
- Loureiro, Antonio Alfredo Ferreira. Grafos. Disponível em: [http://www.dcc.ufmg.br/loureiro/md\\_9Grafos.pdf](http://www.dcc.ufmg.br/loureiro/md_9Grafos.pdf). Acesso em: 09 abr. 2023.
- Wikipedia. Shortest-path tree. Disponível em: [https://en.wikipedia.org/wiki/Shortest-path\\_tree](https://en.wikipedia.org/wiki/Shortest-path_tree). Acesso em: 09 abr. 2023.