UNIVERSIDADE FEDERAL DO AGRESTE DE PERNAMBUCO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME PAES CAVALCANTI

MENOR CAMINHO EM GRAFOS: IMPLEMENTAÇÃO INSPIRADA NO MAPA DE PACIFIC DRIVE

Professor: Igor Medeiros Almeida Vanderlei
Disciplina: Algoritmos e Estrutura de Dados II (2024.1)

Garanhuns
2024

1 Introdução

Este projeto propõe otimizar a seleção de rotas contidas no jogo Pacific Drive, este possui um mapa composto por diversos locais e rotas, sendo que cada uma desta possui um certo valor associado. Analisando o contexto apresentado, é notável o uso de grafos para a representação do mapa, pois cada local seria um vértice e cada rota (entre dois vértices) seria uma aresta.

Sobre o método para obter o menor caminho, tenhamos em vista que o grafo gerado possui apenas arestas de peso positivo pois, no contexto do jogo, os pesos a serem possivelmente considerados são ou o tempo a decorrer, ou o nível de perigo ou até mesmo a riqueza de recursos na rota. Sendo assim, o algoritmo de Dijkstra se fez muito atrativo por sua eficiência e praticidade de implementação visto que a linguagem de programação Java disponibiliza muitos recursos já preparados para a resolução, também sendo importante ressaltar que o mesmo foi fortemente mencionado em *[2] Algoritmos: Teoria e Prática*.

2 Conceitos Teóricos

Definição de Grafo

Um grafo é uma estrutura matemática usada para modelar a relação entre pares de elementos. Um grafo $G = (V, E)$ é composto por dois conjuntos: V, o conjunto de vértices, que representam os objetos ou pontos de interesse; E, o conjunto de arestas, que são pares de vértices e representam as conexões entre esses objetos.

Um grafo é considerado ponderado quando suas arestas possuem pesos, direcionado quando as arestas têm uma direção definida, e não direcionado quando as conexões entre vértices são bidirecionais. Para o projeto em questão, consideraremos um grafo ponderado e possivelmente direcionado, de acordo com as características das rotas do jogo.

Quando no contexto da computação e, assim como apresentado em *[2] Algoritmos: Teoria e Prática*, podemos escolher dois modos padrões para representar um grafo: uma coleção de listas de adjacências ou como uma matriz de adjacências, ambas servindo para grafos dirigidos ou não, a escolha entre um ou outro se resume a identificar se o grafo é denso ou esparso, isto é, aqueles para os quais o número de arestas é muito menor que o quadrado do número de vértices ou aqueles em que o número de arestas está próximo do quadrado do número de vértices. Uma **lista de adjacências** se resume a armazenar todos os vizinhos de um vértice numa lista, e fazer isto para cada vértice pertencente ao grafo, ao fim teremos uma coleção de listas de adjacência. Uma **matriz de adjacência** é uma forma de representar um grafo utilizando uma matriz quadrada. Para um grafo com n vértices, a matriz é de tamanho $n \times n$, onde a entrada a_{ij} indica a presença e o peso da aresta que conecta o vértice i ao vértice j. Se $a_{ij} = 0$, não existe aresta entre os vértices i e j; caso contrário, a_{ij} representa o peso da aresta. Para grafos não ponderados, o valor de a_{ij} é 1 se existir uma conexão e 0 caso contrário.

Algoritmo de Dijkstra

O algoritmo de Dijkstra é um método clássico para encontrar o caminho mais curto em um grafo a partir de um vértice de origem para todos os outros vértices. Este algoritmo é adequado para grafos com arestas de peso positivo, como no caso do mapa do jogo. O funcionamento se dá seguinte forma:

1. Inicializa-se uma lista de distâncias mínimas de todos os vértices a partir da origem, sendo a distância da origem para si mesma igual a 0 e para os demais vértices como infinito (∞).

2. Marca-se o vértice de origem como visitado e os demais como não visitados.

3. Em cada iteração, seleciona-se o vértice não visitado com a menor distância conhecida e o define como o vértice atual.

4. Para cada vizinho do vértice atual, calcula-se a distância total a partir da origem. Se a nova distância calculada for menor do que a anteriormente conhecida, atualiza-se o valor.

5. Marca-se o vértice atual como visitado. O processo se repete até que todos os vértices sejam visitados ou até que o menor caminho para o destino seja encontrado.

Para a implementação no contexto do projeto, o algoritmo de Dijkstra é capaz de processar o grafo do mapa de Pacific Drive, identificando a rota mais eficiente (com o menor custo) de um ponto a outro, facilitando assim a escolha da melhor trajetória para o jogador.

3 Descrição do Projeto

Problema a Ser Resolvido

O principal desafio deste projeto é otimizar a seleção de rotas no jogo Pacific Drive, onde os jogadores navegam por um mapa que contém diversos locais interconectados. Cada rota entre esses locais possui um tempo de deslocamento. O objetivo é encontrar a rota mais eficiente entre os pontos, utilizando para isso o algoritmo de Dijkstra, que se mostrará uma solução eficaz para o problema de caminho mais curto.

Figura 1: Mapa do Pacific Drive.

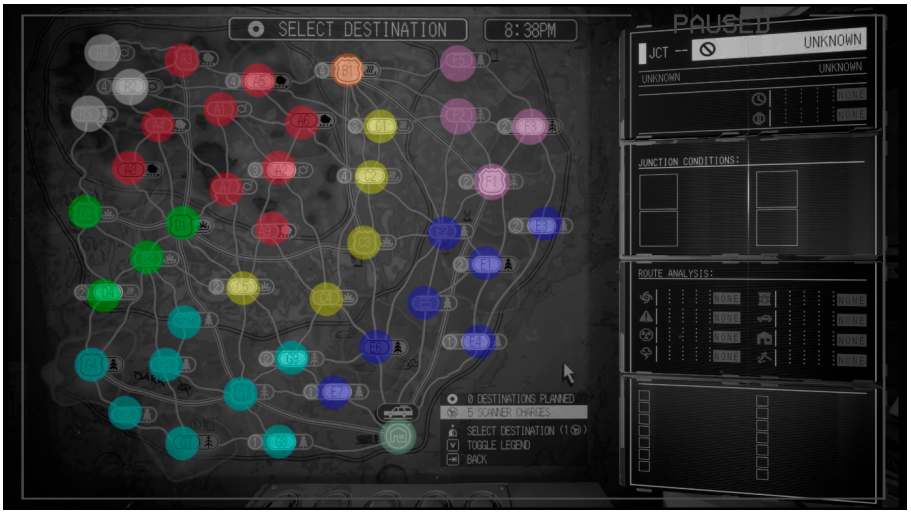

Figura 2: Mapa editado pelo autor

Com base na Figura 2, foi feita a contagem de vértices e arestas para teste posterior.

Requisitos do Sistema

- O sistema deve permitir a representação do mapa do jogo como um grafo.
- O sistema deve implementar o algoritmo de Dijkstra para calcular o caminho mais curto.
- O sistema deve informar ao usuário as condições do grafo através do terminal do computador.
- O sistema deve ser implementado em Java, garantindo portabilidade e eficiência.

Ferramentas e Tecnologias Utilizadas

Para o desenvolvimento do projeto, foram utilizadas as seguintes ferramentas e tecnologias:

- **Linguagem de Programação:** Java - escolhida pela sua gama de bibliotecas disponíveis e praticidade.
- **IDE:** Visual Studio Code (VSCode) - utilizada para codificação, com extensões necessárias para suporte a Java, como:
  - `Java Extension Pack` - um conjunto de ferramentas para desenvolvimento em Java.
  - `Debugger for Java` - para depuração do código.

4 Implementação

Classes principais

- **Vertex**: Representa um vértice no grafo, contendo um atributo `String name` que identifica o vértice.
- **Edge**: Representa uma aresta no grafo, com dois vértices (`origin` e `destiny`) e um peso (`weight`) que indica a "distância"ou custo para atravessar a aresta.

- **Graph**: Contém um conjunto de vértices (`Set<Vertex> vertices`) e um conjunto de arestas (`Set<Edge> edges`). Possui métodos para adicionar vértices e arestas, verificar a existência de vértices, e imprimir a lista de adjacência.
- **Dijkstra**: Contém a função de obter o menor caminho, com base no algoritmo de Dijkstra. Além de também conter uma função para imprimir o menor caminho.

Estrutura de Dados

O grafo é representado usando conjuntos (sets) para armazenar vértices e arestas, garantindo que não haja duplicatas. Uma lista de adjacência é utilizada para representar as conexões entre os vértices.

5 Algoritmos Implementados

Algoritmo de Dijkstra

Implementado na classe **Dijkstra**. O algoritmo encontra o caminho mais curto entre dois vértices em um grafo ponderado, utilizando uma fila de prioridade para explorar os vértices de menor distância. Armazena as distâncias dos vértices a partir do vértice inicial e também a referência ao vértice anterior, permitindo a reconstrução do caminho mais curto ao final.

6 Exemplos de Execução

Adicionar Vértices

O usuário é solicitado a informar quantos vértices deseja adicionar, e então insere os nomes dos vértices um a um.

```
1
How many?
3
A
B
C
```

Adicionar Arestas

O usuário informa quantas arestas deseja adicionar, e então insere os detalhes de cada aresta no formato `origin_destiny_weight`.

```
2
How many?
2
A_B_5
B_C_3
```

Imprimir Vértices

O comando imprime todos os vértices adicionados ao grafo.

```
3
[A][B][C]
```

Imprimir Arestas

O comando imprime todas as arestas do grafo, incluindo pesos.

```
4
A -> B (5)
B -> C (3)
```

Encontrar Caminho Mais Curto

O usuário insere o vértice inicial e o final, e o algoritmo de Dijkstra é utilizado para encontrar e imprimir o caminho mais curto.

```
5
Initial vertex:
A
Final vertex:
C
# Shortest path between [A] and [C]
[A] [B] [C]
```

Informações do Grafo

O comando imprime o número total de vértices e arestas no grafo.

```
6
# Graph info
· 3 vertices
· 2 edges
```

Parar o Programa

O usuário pode encerrar a aplicação.

```
0
Program stopped!
```

7 Testes e Resultados

Para garantir o funcionamento adequado da aplicação de gerenciamento de grafos, foram realizados testes em diferentes cenários. Esses testes visaram validar as funcionalidades implementadas e interpretar os resultados obtidos.

Cenários de Teste

- **Adicionar Vértices e Arestas**: Testou-se a adição de múltiplos vértices e arestas ao grafo, verificando se a estrutura armazenava as informações corretamente.

- **Imprimir Vértices e Arestas**: Confirmou-se se as listas de vértices e arestas eram impressas corretamente e correspondiam às entradas realizadas.
- **Caminho Mais Curto**: O algoritmo de Dijkstra foi avaliado para confirmar o caminho mais curto entre pares de vértices em diferentes configurações do grafo.

Resultados

- **Adição de Vértices e Arestas**: Todos os vértices e arestas foram adicionados corretamente, com duplicatas sendo evitadas, conforme esperado.
- **Impressão**: As impressões dos vértices e arestas estavam corretas, refletindo fielmente as adições feitas pelo usuário.
- **Caminho Mais Curto**: O algoritmo de Dijkstra retornou caminhos mais curtos corretamente em todos os casos testados. Por exemplo, ao buscar o caminho de A a C em um grafo com arestas pesadas, o resultado foi o esperado.

Interpretação dos Resultados

Os testes demonstraram que a aplicação funciona como projetado.E o algoritmo de Dijkstra operou eficientemente.

A eficácia do algoritmo é notável, pois consegue calcular o caminho mais curto em tempo razoável, mesmo com um número significativo de vértices e arestas.

8 Conclusão

A implementação do gerenciador de grafos provou ser funcional e capaz de atender aos objetivos propostos. Os testes realizados demonstraram que a aplicação consegue realizar as seguintes operações de maneira eficaz:

Apesar dos resultados positivos, o código ainda apresenta algumas inconsistências e áreas que necessitam de melhorias. As principais considerações para desenvolvimento futuro incluem:

- **Tratamento de Erros**: Melhorar a robustez da aplicação, incluindo um tratamento de erros mais adequado, que possa lidar com entradas inválidas e cenários inesperados.
- **Interface do Usuário**: A experiência do usuário poderia ser aprimorada com uma interface gráfica, em vez de depender apenas de comandos de linha de comando.
- **Otimização do Algoritmo**: Analisar e implementar melhorias no algoritmo de Dijkstra, como o uso de estruturas de dados mais eficientes para a fila de prioridade.
- **Extensão da Funcionalidade**: Incluir novas funcionalidades, por exemplo: elaborar novas buscas de menor caminho, pois no momento temos apenas a busca por um número genérico; elaborar funções para maior manipulação do grafo (remoção, edição), refatorar o código no geral para ficar mais fiel ao jogo Pacific Drive.

9 Referências

Livros

[2] Thomas H. Cormen et al. *Algoritmos: Teoria e Prática*. 3ª ed. Acesso em: 27 set. 2024. ELSEVIER, 2012.

[5] Jayme Luiz Szwarcfiter. *Estruturas de Dados e seus Algoritmos*. Rio de Janeiro: LTC - Livros Técnicos e Científicos, 1994.

Referências Adicionais

[1] AULAS DE COMPUTAÇÃO. *Representação de Grafos - Listas e Matriz de Adjacências - Algoritmos em Grafos*. Disponível em: `https://www.youtube.com/watch?v=WItzhg5thjY`. Acesso em: 27 set. 2024. 2021.

[3] AUGUSTO GALEGO. *Explicação do Algoritmo de Dijkstra*. Disponível em `https://www.youtube.com/watch?v=pWrNOjw50As`. Acesso em: 27 set. 2024. 2024.

[4] CODING WITH JOHN. *Set and HashSet in Java - Full Tutorial*. Disponível em: `https://www.youtube.com/watch?v=QvHBHuuddYk`. Acesso em: 27 set. 2024. 2022.

[6] TELUSKO. *#94 Map in Java*. Disponível em: `https://www.youtube.com/watch?v=VcXYlkICcQU`. Acesso em: 27 set. 2024. 2023.