Relatório Latim Airlines

Feito por: Allan José

Objetivo do Projeto

Este projeto tem como objetivo escolher a rota de viagens cujo preço de passagem seja o mais barato ou o tempo de viagem seja o mais curto possivel.

Para fazer isso eu escolhi usar uma implementação simples de grafos em Java com um algoritmo de Dijkstra para calcular os pesos das arestas e encontrar o menor caminho. Uma outra opção era usar o algoritmo de Bellman Ford, mas o algoritmo de Dijkstra é um pouco mais leve se comparado a ele.

Grafos: Conceitos e Algoritmos utilizados

Grafos são estruturas usadas para representar conexões entre diversos objetos. São compostos por vértices, que representam um objeto, e arestas, que representam o caminho que conecta um objeto com outro, no caso deste projeto a vértices representam as cidades que é possível viajar e as arestas representam as rotas de voo que ligam cada cidade. Existem grafos Não Direcionados, Direcionados e Ponderados, no caso deste projeto foi utilizado de um grafo não direcionado e ponderado.

O Principal algoritmo usado neste projeto foi o Algoritmo de Dijkstra, ele é utilizado para encontrar o menor caminho entre cidades seja pelo preço ou pelo tempo de viagem, este algoritmo se mantém bem firme em estruturas mais fundamentais do Java como ArrayLists e Listas que são usados para armazenar os vértices e arestas, mas em certo ponto é necessário usar uma lista de prioridade que garante que a menor distância conhecida seja sempre usada primeiro, o que garante que o algoritmo funcione corretamente e de maneira mais eficiente.

Requisitos do Sistema

Este projeto visa resolver o problema de quando há um destino muito distante que queira ser visitado mas não existe um caminho direto para este destino, e então é necessário fazer múltiplas viagens que podem se estender por várias rotas diferentes, e então, dentre as opções, escolher a qual tem o menor custo ou tempo de viagem.

O sistema precisa receber uma lista com todas cidades e todas as rotas de voo que ligam umas com as outras e então ser capaz de determinar qual o melhor caminho a se seguir para ter o menor preço ou tempo de viagem

O projeto foi desenvolvido em Java, usando algumas bibliotecas padrões do JDK, como a ArrayList, List, PriorityQueue e Collections, e para a IDE foi utilizado o Eclipse.

Estrutura do Grafo

O grafo foi feito usando lista de adjacência, utilizando ArrayList para representar o conjunto de Vértices e Arestas, Cada Vértice pode possuir parâmetros básicos como Strings e int para guardar informações de cada cidade, além de um double para a distância. Este parâmetro de distância representa a soma do peso das arestas percorridas, logo ela significa tanto o preço da passagem quanto o tempo de viagem. Além disso possui também um ponteiro para o vértice anterior, que utilizado no algoritmo de Dijkstra para guardar o caminho de vértices percorrido. Durante a criação de um grafo apenas os parâmetros básicos como Strings e int são preenchidos. Cada aresta possui dois Vértices que representam o vértice que se inicia e o que se encerra a aresta, e possui também dois parâmetros de peso: o tempo de viagem e o preço da passagem.

A leitura dos grafos é feito por um arquivo .txt, ele começa a leitura dos dados quando encontra a palavra “Vertices:” e então ele começa a preencher um ArrayList para os vértices., dividindo os parâmetros sempre que ler um ponto e vírgula, e assim que ele encontrar a palavra “Arestas:” ele começa a preencher um ArrayList para as arestas, também seguindo a mesma lógica de dividir os parâmetros em após cada ponto e vírgula.

Para a classe do algoritmo de Dijkstra, a estrutura dela recebe um grafo, um vértice inicial e destino, uma ArrayList que guardará os vértices que ainda faltam ser percorridos e um ArrayList que guardará os vértices que representam o caminho mínimo. Sempre que for necessário encontrar o menor caminho o grafo precisa ser preparado para a operação, isso é feito de forma que ele define a distância de todos os vértices como sendo infinito e o vértice inicial sendo 0.

Após a etapa de preparação é iniciado o processo para encontrar o menor caminho, primeiro é criada a lista de prioridade e o vértice inicial é adicionado a ela, Enquanto houver vértices na fila, o algoritmo retira o vértice com a menor distância conhecida, e verifica todos os seus vizinhos (vértices adjacentes).

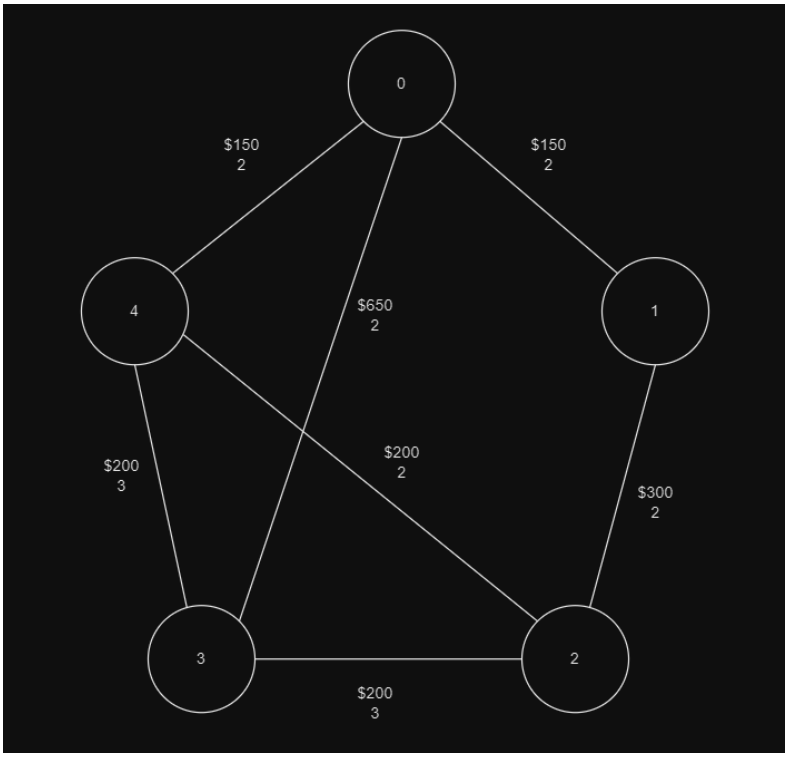
Para cada vizinho será feito a soma da distância do vértice atual com o peso da aresta que conecta este vizinho. Se essa nova distância somada for menor que a distância armazenada anterior no vizinho, então significa que há um caminho mais rápido/barato para chegar neste vértice, então agora atualiza a distância deste vizinho para esta nova rota mais curta, e então adicione este vizinho de volta para a lista de prioridade. Eventualmente na lista de prioridades não haverá mais vértices a serem percorridas, indicando que o grafo foi completamente percorrido.

Após o fim da execução do algoritmo, a partir do vértice destino a fila de vértices é percorrida inversamente usando o ponteiro anterior até chegar no vértice inicial, esse é o menor caminho do vértice inicial até o vértice destino, agora só resta inverter essa lista, pois ela foi obtida percorrendo inversamente o grafo.

Testes e Resultados

```
22
23          ArrayList<Aresta> arestas = new ArrayList<Aresta>();
24          ArrayList<Vertice> vertices = new ArrayList<Vertice>();
25
26          Vertice v0 = new Vertice(0, "cidade0", "cidade0porto");
27          Vertice v1 = new Vertice(1, "cidade1", "cidade1porto");
28          Vertice v2 = new Vertice(2, "cidade2", "cidade2porto");
29          Vertice v3 = new Vertice(3, "cidade3", "cidade3porto");
30          Vertice v4 = new Vertice(4, "cidade4", "cidade4porto");
31          Vertice v5 = new Vertice(5, "cidade5", "cidade5porto");
32          Vertice v6 = new Vertice(6, "cidade6", "cidade6porto");
33          Vertice v7 = new Vertice(7, "cidade7", "cidade7porto");
34
35          vertices.add(v0);
36          vertices.add(v1);
37          vertices.add(v2);
38          vertices.add(v3);
39          vertices.add(v4);
40          vertices.add(v5);
41          vertices.add(v6);
42          vertices.add(v7);
43
44          Aresta a0 = new Aresta(v0, v1, 4.0, 150.0);
45          Aresta a1 = new Aresta(v1, v2, 2.0, 300.0);
46          Aresta a2 = new Aresta(v2, v3, 3.0, 200.0);
47          Aresta a3 = new Aresta(v2, v4, 2.0, 200.0);
48          Aresta a4 = new Aresta(v3, v0, 2.0, 650.0);
49          Aresta a5 = new Aresta(v3, v4, 3.0, 200.0);
50          Aresta a6 = new Aresta(v4, v0, 2.0, 150.0);
51
52          arestas.add(a0);
53          arestas.add(a1);
54          arestas.add(a2);
55          arestas.add(a3);
56          arestas.add(a4);
57          arestas.add(a5);
58          arestas.add(a6);
59
60          Grafo grafo = new Grafo(arestas, vertices);
61
```

1. Modelo de grafo criado para os testes.



2. Visualização do Grafo utilizado
para os testes a seguir.

```
54
55      Grafo grafo = new Grafo(arestas, vertices);
56
57      String strInicial = "cidade0";
58      String strDestino = "cidade3";
59
60      Vertice inicial = grafo.buscaVertice(grafo, strInicial);
61      Vertice destino = grafo.buscaVertice(grafo, strDestino);
62
63
64      Dijkstra dijkstra = new Dijkstra(grafo, inicial, destino);
65
66      ArrayList<Vertice> caminho = dijkstra.procurarCaminhoTempo(grafo, inicial, destino);
67
68      System.out.println("voce terá de pegar um voo nas seguintes cidades: ");
69      for (Vertice vertice : caminho) {
70          System.out.println(vertice.getNomeCidade());
71
72      }
73      double valorTotal = dijkstra.calcularValorPassagens(caminho, grafo);
74      double tempoTotal = dijkstra.calcularTempoViagem(caminho, grafo);
75      System.out.println("valor total da viagem: " + valorTotal);
76      System.out.println("e ira levar um total de " + tempoTotal +" horas");
77
78  }
79
```

Console

```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe  (24 de set. de 2024 02:23:36 – 02:23:36) [pid:
voce terá de pegar um voo nas seguintes cidades:
cidade0
cidade3
valor total da viagem: 650.0
e ira levar um total de 2.0 horas
```

3. Teste para encontrar o caminho
com o menor tempo.

```java

    Grafo grafo = new Grafo(arestas, vertices);

    String strInicial = "cidade0";
    String strDestino = "cidade3";

    Vertice inicial = grafo.buscaVertice(grafo, strInicial);
    Vertice destino = grafo.buscaVertice(grafo, strDestino);


    Dijkstra dijkstra = new Dijkstra(grafo, inicial, destino);

    ArrayList<Vertice> caminho = dijkstra.procurarCaminhoPassagem(grafo, inicial, destino);

    System.out.println("voce terá de pegar um voo nas seguintes cidades: ");
    for (Vertice vertice : caminho) {
        System.out.println(vertice.getNomeCidade());

    }
    double valorTotal = dijkstra.calcularValorPassagens(caminho, grafo);
    double tempoTotal = dijkstra.calcularTempoViagem(caminho, grafo);
    System.out.println("valor total da viagem: " + valorTotal);
    System.out.println("e ira levar um total de " + tempoTotal +" horas");

}
```

Console

<terminated> Main (1) [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 de set. de 2024 02:24:45 – 02:24:45) [pid: 14396]

```
voce terá de pegar um voo nas seguintes cidades:
cidade0
cidade4
cidade3
valor total da viagem: 350.0
e ira levar um total de 5.0 horas
```

4.Teste para encontrar o caminho com o menor custo de viagem.

Comparando o modelo do grafo e o resultado para cada método, é possível ver que na busca pelo tempo de viagem é retornado o caminho entre a cidade0 e a cidade3, e o tempo dessa viagem demoraria apenas 2 horas, mas o custo de passagem seria de R$ 650,00, já na busca pelo preço da passagem é retornado uma rota diferente, seguindo pela cidade0 depois cidade4 e por fim cidade3 que tem um tempo de viagem de 5 horas, mas o custo da viagem é de apenas R$ 350,00. Com isso é possível ver que há vários caminhos para uma mesma cidade, e que, dependendo do peso que for escolhido, a rota ideal para chegar nela é diferente.

Melhorias

Para melhorias futuras uma interface de usuário mais completa seria ideal, mais tratamentos de erros para lidar com inputs incorretos sem que o código pare de funcionar. Adicionar o horário de partida de cada voo, afinal, um caminho pode até ser o mais rápido, mas pode ser que a data inicial dos voos seja daqui a 1 mês. Fazer um cálculo dos pesos do grafo simultaneamente, assim eu posso buscar o caminho mais rápido, mais barato e que os horários dos voos começam mais cedo, tudo incluso num só cálculo.

Fontes:

https://www.youtube.com/playlist?list=PLxI8Can9yAHf8k8LrUePyj0y3lLpigGcl - 23 á 28
https://www.youtube.com/watch?v=jq0N1LDOTIw

https://youtu.be/fWuqx5JWSQA?si=ePn15AX9VXYnjQ6G
https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
https://www.programiz.com/dsa/dijkstra-algorithm
chatgpt.com