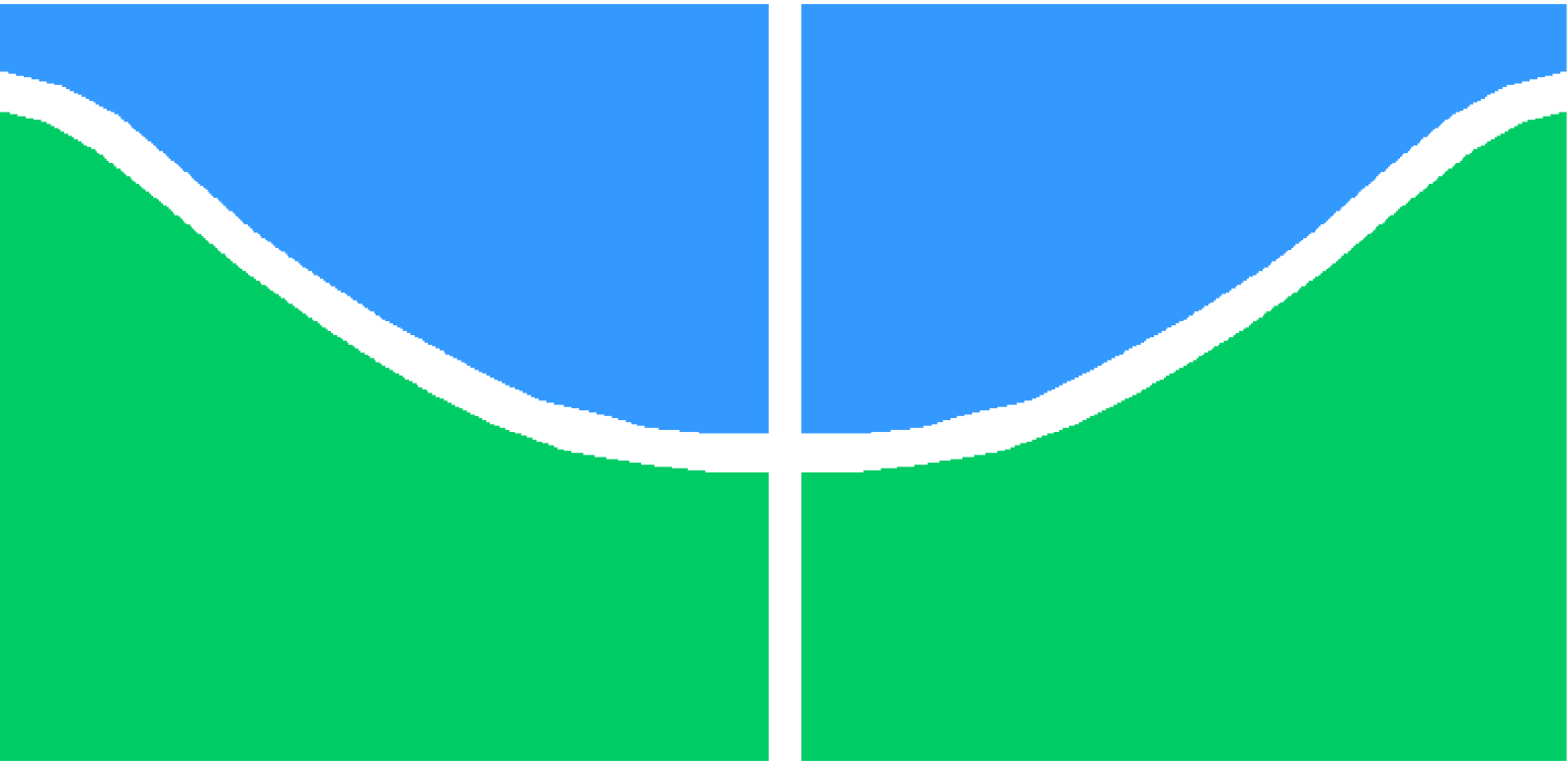

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Estrutura de Dados e Algorítmos

Algorítmo *Best-First Search*

Autores: Dylan Jefferson Maurício Guimarães Guedes, Filipe Ribeiro de Morais, Omar Faria dos Santos Junior
Orientador: Nilton Correia da Silva

Brasília, DF
Brasília, Julho de 2015

Dylan Jefferson Maurício Guimarães Guedes, Filipe Ribeiro de Morais, Omar Faria dos Santos Junior

Algorítmo *Best-First Search*

Documento submetido a disciplina de graduação Estrutura de Dados e Algorítmos da Universidade de Brasília.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Nilton Correia da Silva

Brasília, DF

Brasília, Julho de 2015

Dylan Jefferson Maurício Guimarães Guedes, Filipe Ribeiro de Morais, Omar Faria dos Santos Junior

Algorítmo *Best-First Search*/ Dylan Jefferson Maurício Guimarães Guedes, Filipe Ribeiro de Morais, Omar Faria dos Santos Junior. – Brasília, DF, Brasília, Julho de 2015-

15 p. : il. (algumas color.) ; 30 cm.

Orientador: Nilton Correia da Silva

Tutorial Instalação nxcEditor – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , Brasília, Julho de 2015.

I. Nilton Correia da Silva. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Algorítmo *Best-First Search*

CDU 02:141:005.6

Dylan Jefferson Maurício Guimarães Guedes, Filipe Ribeiro de Morais, Omar Faria dos Santos Junior

Algorítmo *Best-First Search*

Documento submetido a disciplina de graduação Estrutura de Dados e Algorítmos da Universidade de Brasília.

Trabalho aprovado. Brasília, DF, Brasília, Julho de 2015:

Nilton Correia da Silva
Orientador

Brasília, DF
Brasília, Julho de 2015

Sumário

1 Introdução

Independente do ramo (redes, *machine learning*, inteligência artificial, detecção de padrões, labirintos, etc), um dos problemas mais comuns no contexto de grafos é em relação a localização de melhores caminhos para determinados problemas. Um exemplo comum seria o da necessidade de um GPS: o usuário se encontra em determinado lugar, que chamaremos de Origem, e deseja chegar em outro lugar, que chamaremos de Destino. O usuário deseja que o GPS seja capaz de fornecer o melhor caminho possível, levando em consideração as limitações que o próprio usuário teria a respeito do percurso (por exemplo, somente deveria ser levado em conta caminhos terrestres e não aéreos, por motivos lógicos).

Deste tipo de problema surge a necessidade de utilização de um algoritmo que seja capaz de fornecer a solução do melhor caminho. Entre os vários disponíveis (Dijkstra, Tremaux, Bellman-Ford, etc (JOYNER; NGUYEN; COHEN, 2011)), existe o BFS - Best-First Search, que foi implementado e será explicado durante este relatório. No caso, o algoritmo deveria ser aplicado num contexto de capitais brasileiras, levando em consideração somente rotas rodoviárias (dados cedidos pelo DNIT).

No capítulo a seguir será dada uma explanação a respeito do algoritmo, e, no capítulo seguinte, as principais dificuldades encontradas serão levantadas. Uma observação é que, como será mostrado, o grupo utilizou o paradigma de Orientação a Objetos para apoiar o desenvolvimento do algoritmo. A escolha mostrou-se interessante por vários motivos, e será detalhada durante o relatório.

O grupo teve um enorme aprendizado no que tange Estruturas de Dados, Metaprogramação, Orientação a Objetos, Técnicas de Programação entre outros assuntos relativos a programação durante o desenvolvimento do algoritmo, e tentará passar essa ideia.

2 Execução - BFS

Este capítulo têm como objetivo relatar o processo de desenvolvimento do algoritmo, trazendo a abordagem utilizada pelo grupo, as principais técnicas, e uma descrição do código como um todo.

2.1 Descrição do Algorítmo

Contextualizando inicialmente, têm-se duas filas: **Prioridades** e **Soluções**. Antes de iniciar o algoritmo, é feita uma preparação, onde define-se o atributo *visited* da cidade origem como *true*, e é inserido na fila **Prioridades** os caminhos possíveis da origem (ou seja, as fronteiras da origem). O algoritmo inicia-se com a premissa: enquanto **Prioridades** não estiver vazia, execute o bloco de código do algoritmo.

O bloco de código do algoritmo se resume a:

1. Pega-se o primeiro elemento de **Prioridades**, que será chamado de **aux** e é do tipo **Way**.

2. Pega-se o destino final de **Way**, irá ser chamado de **destination_shortcut**.

3. Pega-se o caminho mais curto do **destination_shortcut**. Irá ser chamado de **store_path**, e é o candidato a ser expandido.

4. Os caminhos possíveis de **destination_shortcut** precisam ser ordenados, para que o algoritmo já expanda na ordem do melhor caminho para o pior.

5. Será feito um *loop interno* com n iterações, esse n seria a quantidade de fronteiras que **destination_shortcut** dispõe.

   Loop interno:

   a) Dentro do *loop*, é visto inicialmente se o caminho proposto (**store_path**) é uma solução para o problema. Se for, é feita uma cópia de **aux**, dentro dessa cópia é inserido o caminho dandidato, e, finalmente, **aux** é inserido em **Soluções**. Não é necessário mais continuar no *loop* interno, pois uma solução já foi encontrada e todas as outras seguintes para esse destino serão inferiores.

   b) Caso não seja uma solução, é visto se o destino do candidato (**store_path**) já foi visitado (apresenta *visited* como *true*). Se já tiver sido visitado, ignoramos esse candidato

c) Caso o candidato não tenha sido visitado, finalmente o utilizamos. Inicialmente, é feita uma cópia de **aux**, por conta do problema da Persistência de Dados. Armazenamos essa cópia de **aux** nele mesmo.

d) Dentro da nova cópia de **aux**, inserimos ao final o candidato (é dado *push*).

e) É *setado* o *visited* do destino do candidato como *true*.

f) Finalmente, **aux** é inserido na fila **Prioridades**.

g) O caminho candidato agora deixa de ser **store_path**, para ser o caminho anterior ao **store_path**. Novamente, dizemos que **aux** é o primeiro elemento de **Prioridades**.

6. É dado *pop* em **Prioridades**.

2.2 Contextualização

- Cidades tem Caminhos Possíveis (*Paths*);
- Caminhos Possíveis (*Paths*) tem uma cidade Origem e uma cidade Destino;
- Um *Way* é um conjunto de caminhos possíveis (*Paths*);
- Um Grafo contém a lista de Cidades;

Sendo assim, é possível inferir que as cidades nada mais são que nós do grafo, os caminhos possíveis se tratam de simples arestas do grafo, e o *Way* seria o conjunto dessas arestas.

2.3 Matriz de Adjacência - Não Utilizada

Uma abordagem comum nos problemas que envolvem grafos é o da utilização da Matriz de Adjacência. É interessante pois, por conta do índice, se você sabe qual elemento deseja visitar, não é necessário percorrer nada: têm-se acesso direto ao dado.

Contudo, esta abordagem, como tudo na vida, apresenta pontos negativos. O principal é do mal uso do espaço disponível. Para ilustrar: Imagine o mesmo problema solucionado pelo grupo, mas, ao invés das 27 capitais, tendo-se 100000 municípios. Mas, independente do número de nós ter crescido muito, a média de caminhos possíveis por nós ainda continua sendo de 3 a 4. Isso significa que num vetor de 100000 espaços, 99995 não estão sendo utilizados (e isso é feito 100000 vezes). Se fosse utilizado uma abordagem de armazenamento de elo de grafo, como utilizado pelo grupo, só seria necessário armazenar os 100000 nós, o que já seria feito de qualquer jeito.

Dessa forma, é possível notar que a escolha da Matriz de Adjacência é interessante pelo acesso fácil ao dado, mas, a medida que os dados escalam (sendo assim uma grande massa de dados), o desperdício de recursos fica evidente.

2.4 Descrição dos Dados Utilizados

Os dados a serem tratados foram inseridos em arquivos de texto gerando então uma espécie de banco de dados. Cada capital recebeu dois arquivos sendo que o primeiro possui a nomenclatura padrão "*sigla.txt*",onde "sigla"é a sigla do estado. Cada arquivo recebe:

- Nome da capital, onde nomes compostos são separados por letras maiúsculas (padrão *Capitalize*);
- Código identificador;
- Número de fronteiras.

Cada campo é separado por um caracter barra vertical (|).

O segundo arquivo possue a nomenclatura "*sigla_paths.txt*" e eles recebem os campos:

- Código da rota, que na verdade é a junção do código da capital de partida com o código da capital de destino;
- Nome da capital de saída (padrão *Capitalize*);
- Nome da capital de destino (padrão *Capitalize*);
- Quantidade de kilômetros do percurso.

Como no primeiro arquivo, cada campo é separado por um caracter barra vertical (|). Tais dados foram retiradas do site do Departamento Nacional de Infraestrutura de Transportes (DNIT, 2015).

2.5 Modelagem - Orientação a Objetos

A seguir, será dado um *overview* a respeito de como as estruturas de dados foram tratadas no trabalho do ponto de vista do paradigma de Orientação a Objetos:

Way É um conjunto de **Paths**. Se trata de uma classe, que tem relação do tipo *n para 1* com a classe **Paths**. Além disso, tem associação com a classe **City**, onde no *Way* são armazenados a cidade Origem, a cidade Destino, o primeiro *Path*, o ultimo *Path*, a quantidade de *Paths* e o tamanho da distância. É uma abstração da estrutura de dados Lista.

Path É uma aresta do grafo, ou a distância entre uma cidade X e uma cidade Y que faz fronteira com X. Armazena uma cidade Origem, uma cidade Destino, uma distância entre as duas e um nome.

PriorityQueue Fila de prioridades, armazena um conjunto de *Ways* (relação *n para 1*). Tem os atributos da cabeça do conjunto de *Ways*, a calda, o total de *Ways*, e uma variável extra chamada de **found**, onde é armazenado o melhor *Way* até um dado momento. É uma abstração da estrutura de dados Fila.

Graph Como dito, armazena todo o conjunto de cidades. No construtor dessa classe é feito o *fetch* dos arquivos *.txt*, onde são carregadas as cidades disponíveis.

City O nó do grafo, tem os atributos nome, *visited*, o primeiro *Path* possível dessa cidade (ou seja, a primeira cidade com que ela faz fronteira), o ultimo *Path* possível, o total de *Paths* disponíveis, e um código. É uma abstração da estrutura de dados Lista.

2.6 Relatório de Execução do Algoritmo

3 Resultados Obtidos

O principal problema encontrado na execução da atividade foi a questão da persistência dos dados. Isso acontecia quando determinado elemento era inserido e, por coincidência, esse mesmo elemento já se encontrava em uma fila diferente. A medida que novos elementos eram inseridos as referências dos elementos enfileirados eram perdidas.

Um fator que gerou uma quantidade considerável de trabalho manual e, por isso, demandou um grande período de tempo, foi a questão da inserção dos dados referentes às distâncias entre cidades dentro dos arquivos de texto. Não se tratava de uma tarefa de difícil abstração, porém trabalhou-se com muitos arquivos com campos a serem estudados.

Outro ponto que causou certa dificuldade foi o fato de se tratar da primeira experiência usando grafos, que é um assunto fora do escopo da disciplina exigindo portanto vários estudos complementares para fundamentação dos conceitos.

A escolha do paradigma Orientado a Objetos na resolução do problema poderia gerar *overhead*, mas essa decisão se mostrou acertada uma vez que essa abordagem abriu um leque de novas funcionalidades e formas de trabalhar que, caso usadas de maneira correta, facilitariam a realização da solução.

Durante a execução do trabalho não foi possível encontrar problemas decorrentes da falta de recursos computacionais e nem de estouro de memória, o que demonstra que se atingiu uma solução de grande qualidade. Outro ponto que deve ser destacado é que as respostas são apresentadas de forma instantânea à inserção das entradas, independente de se tratar de casos muito complexos ou não. Deve-se levar em consideração que o desenvolvimento foi executado e testado em plataformas Linux com mais de 4 GB de *ram*.

Referências

DNIT, D. N. de Infraestrutura de T. *Departamento Nacional de Infraestrutura de Transportes*. 2015. Disponível em: <http://www.dnit.gov.br/>. Citado na página 11.

JOYNER, D.; NGUYEN, M. van; COHEN, N. *Algorithmic Graph Theory*. [S.l.], 2011. Citado na página 7.