Algoritmo de Dijkstra

Patricia Castro, Thiago Leite, Vanessa Soares,
Orientador: Leandro G. M. Alvim

[1]Departamento de Ciência da Computação –
Universidade Federal Rural do Rio de Janeiro (UFRRJ)
R. Governador Roberto Silveira S/N – Nova Iguaçu –
Rio de Janeiro – RJ – Brasil

Abstract. *The report shows the operation of the Dijkstra algorithm, used to solve the problem of shortest paths in a graph with non-negative weights at the edges, showing that this can also be used in real-world examples such as the example shown below, the Barcelona subway. By resemble a graph, where the statios are vertex and the lines represent the edges, this example becomes almost perfect and seeks to find the best route for which stations the passenger should move to go the station A to B, using the shortest distance. The objective comes down to explain and elaborate on the problem of shortest paths, present the Dijkstra's algorithm as a possible solution, prove it and show it's complexity.*

Resumo. *O trabalho demonstra o funcionamento do algoritmo de Dijkstra, utilizado para resolver o problema de caminhos mínimos em um grafo com pesos não negativos nas arestas, mostrando que este também pode ser usado em exemplos do mundo real, tal como o exemplo mostrado a seguir do metrô de Barcelona. Por se assemelhar a um grafo, onde as estações são vértices e as linhas representam as arestas, este exemplo torna-se praticamente perfeito e procura encontrar o melhor caminho, por quais estações o passageiro deverá passar para seguir da estação A até B, percorrendo-se a menor distância. O objetivo do trabalho se resume a explicar e dissertar sobre o problema de caminhos mínimos, apresentar o algoritmo do Dijkstra como possível solução, prová-lo e mostrar sua complexidade.*

1. Introdução

Criado pelo cientista da computação Edsger Dijkstra em 1956, o algoritmo de Dijkstra soluciona o problema do caminho mínimo num grafo dirigido com arestas de peso não negativos. Assim, um problema prático seria aplicar esse algoritmo no metrô de Barcelona, para ajudar um passageiro a encontrar o caminho mais rápido ou com menor custo de uma estação A até B.

Sabendo que uma rede de metrô de uma cidade como Barcelona, na Espanha. Esta rede cobre boa parte da cidade, sendo composta por várias linhas que se cruzam em estações específicas. Nestes pontos de cruzamento um usuário pode livremente sair de uma composição e passar para uma composição de outra linha. Sendo assim, em geral o usuário tem mais de uma opção de rota quando deseja deslocar-se de uma parte a outra da cidade. Escolher a melhor rota passa, então, a ser fundamental para que o deslocamento seja o mais rápido possível.

Figura 1. Mapa do metrô de Barcelona[1]

As distâncias entre estações vizinhas do metrô não são iguais e, consequentemente, o tempo de deslocamento entre estações vizinhas não é constante.

2. Problema e Motivação

Em redes de metrô de várias cidades do mundo há sistemas computacionais que auxiliam os usuários a escolher a melhor rota para um deslocamento particular. Mais do que simplesmente indicar uma rota possível, estes sistemas procuram identificar aquela rota que conduz o usuário o mais rapidamente ao seu destino.

No trabalho proposto, podemos representar o metrô de Barcelona através de um grafo G (V, E), onde as estações são os vértices e as linhas do metro são as arestas, contendo as baldeações nas trocas das linhas.

3. Objetivo

O objetivo é dissertar sobre o problema de caminhos mínimos, o algoritmo Dijkstra, suas provas de corretude, complexidade e uma análise empírica.

4. Proposta

4.1. O Algoritmo de Dijkstra

O algoritmo Dijkstra, assim chamado devido ao seu criador Edsger Dijkstra, surgiu para solucionar o problema do caminho mínimo entre vértices de um grafo dirigido ou não dirigido. Esse algoritmo não consegue encontrar o menor caminho em um grafo com arestas de pesos negativos, para esse propósito, pode-se usar o algoritmo de Bellman-Ford, que possui uma maior complexidade de tempo de execução do que o Dijkstra.

O Dijkstra é considerado um algoritmo guloso, pois toma a decisão que parece ótima no momento, apesar disso ele é considerado ótimo pois encontra sempre o caminho de menor custo da raiz para todos os demais nós do grafo. A estratégia gulosa é conveniente já que em um menor caminho C entre dois vértices do grafo, todo sub-caminho entre esses vértices é um menor caminho entre dois vértices pertencentes ao caminho C,

[1]Fonte: Site World Insider

```
DIJKSTRA-
1  para u ← 1 até n faça
2      cor[u] ← branco
3      dist[u] ← ∞
4  cor[r] ← cinza
5  dist[r] ← 0
6  Q ← CRIA-FILA-VAZIA ( )
7  INSERE-NA-FILA (r, Q)
8  enquanto  Q  não está vazia faça
9      u ← EXTRAI-MIN (Q)
10     para cada v em Adj[u] faça
11         se cor[v] = branco
12             então cor[v] ← cinza
13                   dist[v] ← dist[u]+f(uv)
14                   INSERE-NA-FILA (v, Q)
15             senão  se cor[v] = cinza e dist[u]+f(uv) < dist[v]
16                       então DIMINUI-CHAVE (v, Q, dist[u]+f(uv))
17     cor[u] ← preto
18 devolva  dist[1..n]
```

Figura 2. Pseudo código de Dijkstra[2]

desta forma são obtidos menores custos nos vértices alcançáveis pelo vértice inicial determinando todos os menores custos intermediários, caso existam dois menores caminhos apenas um será descoberto.

4.2. Prova de Corretude

Após a finalização do algoritmo Dijkstra sobre um grafo orientado,conforme visto no livro Algoritmos [1], com pesos não negativos nas arestas $w : E \longrightarrow \Re^+$ e vértice de início *s*, temos $d[u] = \delta\ (s, u)$, para todo vértice $u \in V$. Nota:$d[u] = \delta\ (s, u)$ denota o valor dos pesos das somas da arestas de *u* até *v* sendo ∞ quando não há um caminho de *u* a *v*.
Prova: Mostrar que no instante que *u* é inserido em *S*, $d[u] = \delta\ (s,u)$. Mostraremos isso por contradição.
Suponha que exista um vértice *u*, tal que ao inserir *u* em *S*, $d[u] \neq \delta\ (s, u)$. Escolha *u* como primeiro vértice em que isso ocorre. Logo, $u \neq s$, já que $d[s] == \delta\ (s,s)$;
Se o caminho associado a *d[u]*, não é de peso mínimo, então deve existir outro caminho *P* de *s* até *u* de peso δ ¡ *d[u]*. Seja *(x, y)* a primeira aresta de *P* que liga um vértice $x \in S$ com um vértice $y \in$ a *(V-S)*. Como $x \in S$ e $y \in Adj[x]$, temos a Relaxação.
Como todos os pesos de *P* são não negativos e $(x,y) \in P$, temos que
(1)

$$d[y] \leq d[x] + w(x, y) \leq \delta(s, u)$$

Como *x* e $y \in$ *(V-S)* e *u* é escolhido antes de *y*, temos que
(2)

$$d[u] \leq d[y]$$

[2]Fonte: Site IME-USP

Das inequações (1) e (2) temos
(3)

$$d[u] \leq d[y] \leq d[x] + w(w, y) \leq \delta(s, u)$$

Logo, a inequação (3) contraria a escolha de *u*.

4.3. Análise de sua Complexidade

A rapidez do algoritmo de Dijkstra consiste em manter a fila de prioridade mínima Q chamando três operações de filas de prioridade: inserção, extração de minimo e decrescimento de chave. Inserção é invocado uma vez por vértice, como é extração minima. Pelo fato de cada vértice v pertencente à V ser adicionado ao conjunto S exatamente uma vez durante o curso do algoritmo. Tendo em vista que o numero total de arestas em todas as linhas de adjacências é $|E|$ iterações desse loop for, e há portanto um total de no máximo $|E|$ operações do tipo decremento de chave.

O tempo de execução do algoritmo de Dijkstra depende de como a fila de prioridade minima e implementada. Considerando um caso em que mantemos a fila de prioridade mínima, tirando proveito do fato de que os vértices são numerados de 1 ao módulo de V. Simplesmente armazenamos $d|v|$ na v-ésima entrada de um arranjo. Cada operação de inserir demora o tempo O(1), e cada operação de extração de minimo demora o tempo de $O(V)$ (pois temos que pesquisar pelo arranjo inteiro), dando um tempo total de $O(V^2 + E) = O(V^2)$.

Se o grafo e suficientemente esparso (em particular, $E = o(v^2/lgV)$) é pratico implementar a fila de prioridade mínima Q com um heap mínimo binário. Cada operação de extração de mínimo demora então o tempo $O(lgV)$. Como antes existem $|V|$ dessas operações. O tempo para construir o heap minimo binário é $O(v^2)$ de implementação direta se $E = o(v^2/lgV)$.

Podemos então alcançar um tempo de execução igual a $O(VlgV + E)$ implementando a fila de prioridade mínima Q com um heap de fibonacci. O custo amortizado de cada uma das $|V|$ operações de extração de minimo é $O(lgV)$, e cada chamada de decrescimento de chave, das quais existe no máximo $|E|$, demora apenas o tempo amortizado $O(1)$.

5. Experimentos

5.1. Trajeto do Metrô

Primeiramente e gerada uma matriz de adjacência com pesos e referências aleatórias, um arquivo e criado aonde são salvos os dados do grafo na seguinte ordem: primeira linha contém quantidade de vértices e arestas, segunda linha contém os vértices, e terceira linha contem as arestas da ordem origem-destino-peso.

O algoritmo lê as informações do arquivo gerado e carrega suas estruturas de vértices e arestas, e o algorítimo de Dijkstra é executado.

Foi criada a fila de prioridades a partir do vértice A (Localização da do passageiro), para cada adjacência do vértice de menor prioridade da fila, se já foi visitado e é avaliada a

distância entre a aresta que os liga, e caso seja menor que a distancia atual dessa adjacência a distância é atualizada e esse vértice é inserido na lista.

5.2. Teste de Desempenho

Mostraremos alguns resultados de experimentos aplicando o algoritmo de Dijkistra em grafos aleatóriamente gerados. Os mesmos utilizados no experimento, possuem quantidades de vértices e arestas em crescimento a cada execução. Os resultados obtidos serão mostrados a seguir:

Tabela 1. Resultados de Execução

	Vértices	Arestas	Tempo(s)
Caso 1	100	1.000	$3{,}10\text{x}10^{-2}$
Caso 2	1.000	10.000	$8{,}60\text{x}10^{-1}$
Caso 3	10.000	100.000	7,34x10
Caso 4	10.000	1.000.000	$9{,}95\text{x}10^{3}$

Como podemos perceber, o número de vértices e, principalmente de arestas cresce consideravelmente, o que interfere bastante no tempo de execução do algoritmo, já que quanto maior o número de vértices e arestas, maior será a quantidade de caminhos possíveis para se chegar ao vértice desejado.

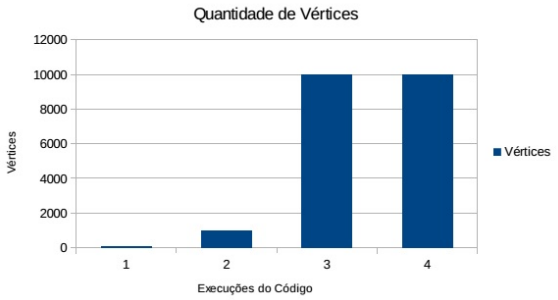

Figura 3. Quantidade de Vértices[3]

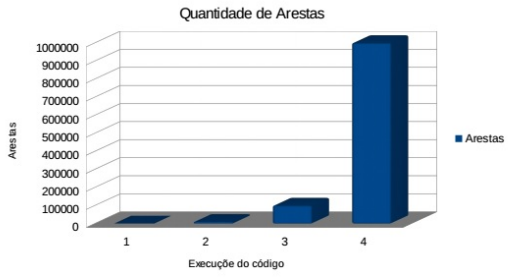

Figura 4. Quantidade de Arestas[3]

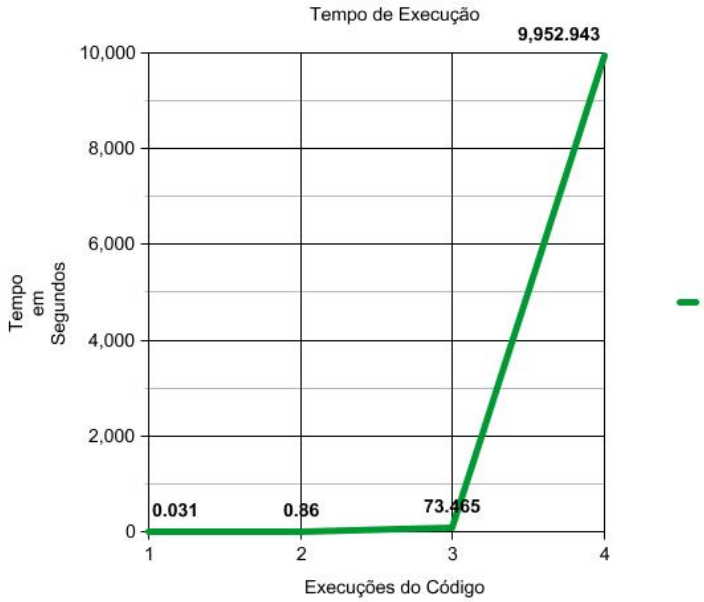

Figura 5. Tempo de Execução[3]

6. Conclusões

O problema do caminho mínimo consiste em encontrar um caminho de menor custo entre dois vértices dados. Podemos, então, apresentar o algoritmo de Dijkstra como uma solução para esse problema em um grafo com arestas de peso não negativo. Usando um problema cotidiano para exemplificar, usamos o do metrô de Barcelona. Tomando as linhas e estações do metro como formando os vértices e arestas de um grafo, podemos aplicar este grafo ao algoritmo e encontrar o melhor caminho, ou seja, o com menor custo, para um passageiro que deseja ir da estação A até a B.

Como esperado, o tempo de execução do algoritmo aumenta de acordo com a densidade do grafo: quanto mais denso, ou seja, quanto maior o número de vértices e arestas, maior o tempo que o algoritmo de Djikstra leva para ser executado. Utilizando grafos que foram gerados aleatoriamente, com vértices de tamanho 100, 1.000, 10.000 e arestas de tamanho 1.000, 10.000, 100.000, 1.000.000, somente na intenção de testar o algortimo.

Podendo ser usado para diversos problemas, esse algoritmo possui um bom desempenho, quando comparado com outros algoritmos gulosos que também podem ser usados na resolução do mesmo problema. Assim, esta mesma resolução, usada no problema do metrô poderia ser usada em outras situações como transmissão de dados em uma rede de computadores, achar o percurso mais rápido entre duas cidades, recnhecimento de voz, segmentação de imagens, entre outros.

[3]Fonte: elaborado(a) pelo(a) autor(a)

7. Referências Bibliográficas

[1] CORMEN, THOMAS, et al. Algoritmos: Teoria e Prática. Editora Campus, pag 470-475.

[2] ROCHA, ANDERSON; DORINI, LEYZA. Algoritmos Gulosos: Definições e Aplicações. 29 de Abril de 2004.

UCHOA, JOEL. Caminhos mínimos com recursos limitados. Dissertação(Mestrado em ciência da computação). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

GUIZZO, A.L, et al. Utilização do Algoritmo de Dijkstra para Cálculo de Rotas no Trabalho Público do Município de Criciúma/SC.

IME-USP. Disponível em: `http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/dijkstra.html`. 25-05-2015

CARVALHO, MARCO. Análise de algoritmo. Fevereiro de 2004.