Comparação do uso de matriz de adjacência e lista de adjacência para o algoritmo de Dijkstra

Igor A. Engler[1], **Marcos A. C. Mucelini**[1]

[1]Ciência da Computação – Universidade Estadual do Oeste do Paraná
Caixa Postal 801 – 85.814-110 – Cascavel – PR – Brazil

Resumo. *Este trabalho tem como objetivo analisar o algoritmo de de Dijkstra com duas diferentes formas de representar os vértices e arestas. A primeira estratégia utiliza matriz de adjacência, e a segunda utiliza lista de adjacência.*

1. Introdução

Na ciência da computação existe um ramo de estudos relacionados a problemas de otimização, cujo objetivo é encontrar a solução ótima de um problema utilizando a menor quantidade de recursos possível, sejam eles de qualquer natureza (tempo, energia, distância percorrida, etc). Um problema clássico desta área, é a otimização de rotas. Neste problema, o objetivo é encontrar a rota de menor distância de um ponto *A*, para um ponto *B* qualquer. Uma solução para esse problema é um algoritmo definido por Edsger Dijkstra [Dijkstra 1959].

Neste trabalho serão realizadas comparações entre duas estratégias distintas para a implementação do algoritmo de Dijkstra. Na primeira estratégia, é utilizada uma matriz de adjacência para representar vértices e arestas, na segunda estratégia, uma lista de adjacência é utilizada. A análise realizada neste trabalho é referente à: (A) Tempo de execução do algoritmo; (B) Número de comparações necessárias para encontrar o caminho mais curto do vértice de origem até todos os outros vértices.

2. Grafos

Em Ciência da Computação, um grafo é uma estrutura de dados que contém um número finito de vértices, e um conjunto de pares de vértices chamados de arestas. Essa estrutura é utilizada para representar uma rede de conexão, com custo, capacidade ou tamanho.

Um grafo pode ser direcionado ou não-direcionado. Para o primeiro caso, o conjunto de arestas possui um sentido/ordem a ser seguido. No segundo caso, a aresta possui duas vias, ou seja, quando um vértice *A* possui uma aresta que aponta para um vértice *B*, esta aresta também aponta de *B* para *A*.

Duas das estratégias mais populares para representar um grafo em um computador consistem da utilização de matrizes ou listas de adjacências. A Figura 1 ilustra a abstração da estrutura de um grafo não-direcionado.

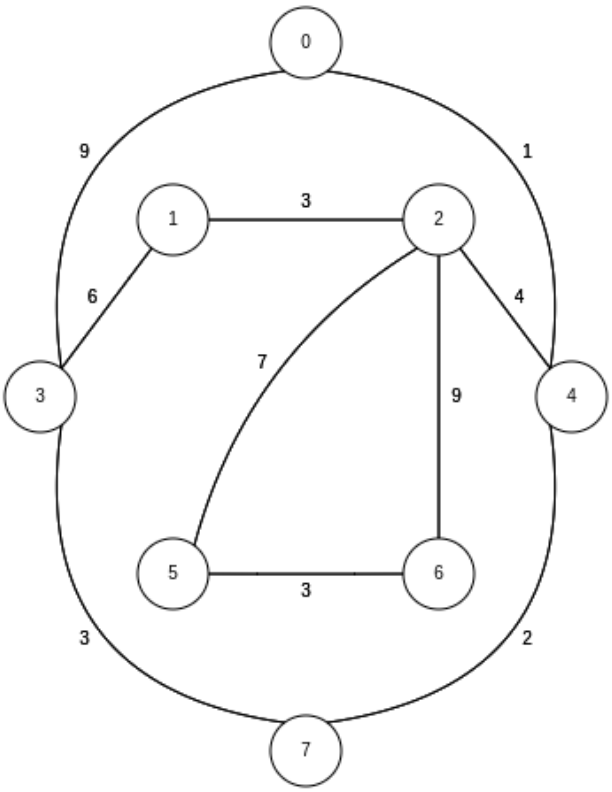
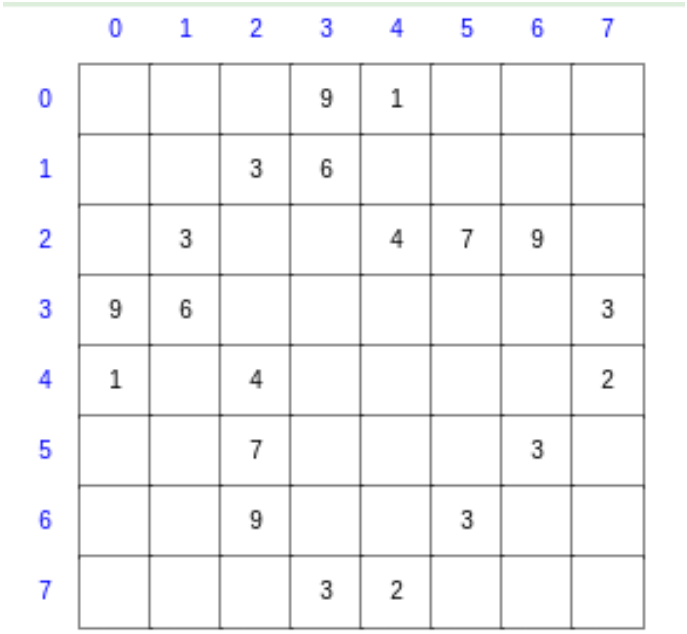


Figura 1. Grafo não-direcionado representado com vértices e arestas

2.1. Matriz de Adjacência

Uma matriz de adjacência é uma matriz quadrada *V x V*, onde *V* é o número de vértices no grafo. Os índices da matriz representam os vértices do grafo, e as arestas são representadas pelos elementos contidos na matriz. O valor definido em *matriz[i][j]* representa a distância entre os vértices *i* e *j*, caso esse valor seja igual a zero, não existe uma aresta entre os dois vértices.

A Figura 2 representa a matriz de adjacência do grafo definido na Figura 1. Para facilitar a visualização, os elementos da matriz com valores iguais a zero não são representados.



	0	1	2	3	4	5	6	7
0				9	1			
1			3	6				
2		3			4	7	9	
3	9	6						3
4	1		4					2
5			7				3	
6			9			3		
7				3	2			

Figura 2. Representação do grafo da Figura 1 através de uma matriz de adjacência

A representação por matriz de adjacência é considerada fácil de implementar e entender, porém, ocupa muito espaço, tendo complexidade espacial $O(n^2)$, mesmo quando se trata de uma matriz esparsa.

2.2. Lista de Adjacência

Uma lista de adjacência consiste de uma coleção de listas utilizadas para representar um grafo, onde cada lista dentro da lista de adjacência representa os vértices vizinhos de um certo vértice no grafo.

Uma das formas mais comuns de representar uma lista de adjacência é utilizando listas encadeadas, mas outras estruturas, como arrays, também podem ser utilizados. Existem diversas variações para implementações de lista de adjacência. A implementação utilizada nesse trabalho armazena um conjunto formado pelo vértice vizinho e pela distância do vértice A implementação utilizada neste trabalho trata a lista da seguinte forma: Para V vértices do grafo, V listas são criadas dentro da lista de adjacência. Cada uma dessas listas aponta para os vizinhos de seus respectivos vértices. Cada posição da lista guarda qual vértice é vizinho, e qual a distância até ele.

A Figura 3 representa a lista de adjacência do grafo definido na Figura 1.

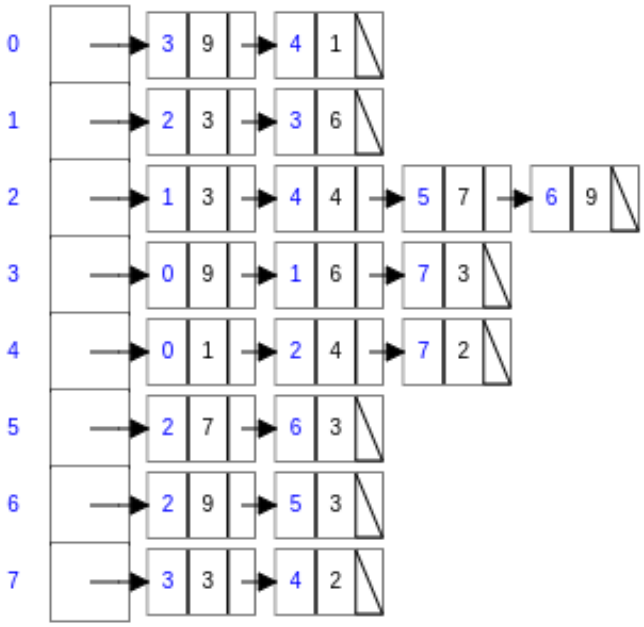


Figura 3. Representação do grafo da Figura 1 através de uma lista de adjacência

Em uma lista de adjacência, o número total de listas é igual a V. Apesar de que cada lista pode ter até $V - 1$ vértices, um grafo não direcionado contém no total $2 * E$ elementos, onde E representa o número de arestas, já que cada aresta aparece duas vezes na lista de adjacência. Para um grafo direcionado, a lista de adjacência contém um total de E elementos.

3. Algoritmo de Dijkstra

O algoritmo de Dijsktra, criado pelo cientista da computação Edsger Dijkstra em seu trabalho [Dijkstra 1959], soluciona o problema do caminho mais curto em grafos orientados com pesos, e em grafo não orientados com pesos não negativos. O passo-a-passo do algoritmo pode ser visto abaixo.

1. Crie um conjunto $visited$ que registra quais vértices tiveram suas distâncias mínimas calculadas e finalizadas. Inicialmente, esse conjunto é vazio.
2. Inicialize a distância para todos os nós como sendo infinito, em seguida, atribua o valor 0 para a distância do vértice inicial para que ele seja escolhido primeiro.
3. Enquanto o conjunto $visited$ não conter todos os vértices:
   (a) Escolha um vértice u que não esteja no conjunto $visited$ e possua o menor valor de distância.
   (b) Inclua u no conjunto $visited$.
   (c) Atualize a distância de todos os vértices adjacentes de u, para isso:
      i. Itere sobre os vértices adjacentes.
      ii. Para cada vértice v adjacente, se a soma da distância de u (do vértice inicial) e o peso da aresta (u, v) é menor do que a distância de v, então atualize a distância de v.

Para percorrer os vértices a fim de encontrar o vértice não visitado com o menor caminho, a complexidade é $O(V)$. Para cada vértice selecionado, é necessário aplicar

relaxamento em seus vizinhos, o tempo necessário para isso é $O(1)$. Para cada vértice, é necessário relaxar todos seus vizinhos, e como um vértice pode ter até $V - 1$ vizinhos, o tempo necessário para atualizar todos os vizinhos de um vértice é $O(V) * O(1) = O(V)$. Portanto, como é necessário tempo $O(V)$ para visitar todos os vértices e tempo $O(V)$ para processar cada vértice, a complexidade do algoritmo de Dijkstra é igual a $O(V) * O(V) = O(V^2)$.

4. Metodologia

A linguagem de programação C++ foi escolhida para a implementação dos algoritmos. Para realizar a análise, quinze arquivos contendo representação de grafos não direcionados utilizando matrizes de adjacência com número de vértices variando entre 10 e 1500 foram utilizados para testar o tempo de execução do algoritmo de Dijkstra utilizando matriz de adjacência e lista de adjacência, assim como o número de comparações necessárias para encontrar a distância mínima do vértice inicial até todos os outros. No total, foram realizados 11 testes, os valores do primeiro foram descartados.

4.1. Ambiente de Testes

Os testes foram realizados em uma máquina com as seguintes especificações:

- Processador Intel Core i5-4200m @2.5GHz
- 8GB de Memória RAM DDR3
- SSD SATA 120GB
- Linux Mint 20.3 Una

5. Resultados e Discussões

A Tabela 1 mostra a média de tempo de 10 execuções do algoritmo de Dijkstra com lista de adjacência e matriz de adjacência, assim como o número de comparações necessárias para encontrar as distâncias entre o vértice inicial e os outros vértices.

Vertices	Tempo de execução em lista (ns)	Comparações em lista	Tempo de execução em matriz (ns)	Comparações em matriz
10	5889,5	156	4360,5	180
25	39918,2	1043	30287,9	1200
50	90587,1	4178	82999,4	4900
75	92520	9473	88932,8	11100
100	155971,2	16880	153113,4	19800
150	354057,4	37950	354703,3	44700
200	627705,4	67768	614168,7	79600
250	965414,2	106074	947340,3	124500
300	1369084,1	152498	1350804	179400
400	2432937,2	271616	2373266,8	319200
500	3834894,6	424416	3735416,3	499000
650	6426380,8	716678	6370622	843700
800	9804731	1087036	10017467,8	1278400
1000	15281058,3	1699172	15203147,4	1998000
1500	34680879,7	3823912	34850965,7	4497000

Tabela 1. Média de tempo de execução e número de comparações para o algoritmo de Dijkstra em lista e matriz de adjacência

A Figura 4 ilustra o tempo de execução necessário para rodar o algoritmo. É possível ver que o tempo da lista de adjacência e da matriz de adjacência são extremamente semelhantes. Nesses testes, o tempo necessário para executar o algoritmo na matriz de adjacência acabou sendo um pouco menor do que para listas.

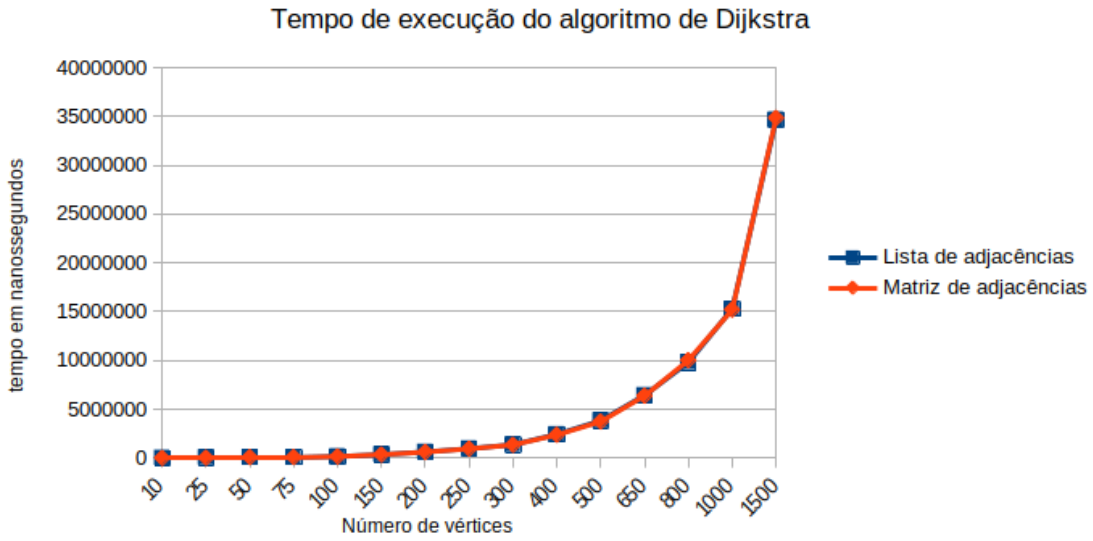


Figura 4. Tempo de execução do algoritmo de Dijkstra

A Figura 5 ilustra o número de comparações necessárias para o funcionamento do algoritmo. Nesses casos de testes, a lista de adjacência levou uma leve vantagem em relação à matriz de adjacência. Os resultados se aproximam devido ao fato de que os grafos utilizados nos testes não são direcionados.

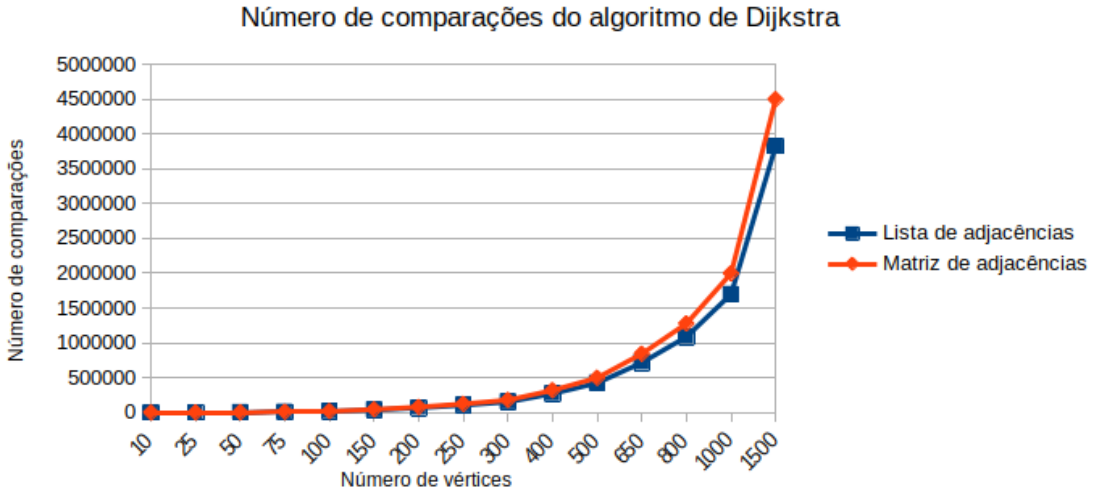


Figura 5. Imagem exemplo caso precise usar

6. Conclusões

As duas representações apresentaram resultados muito semelhantes. A matriz teve uma performance melhor do que a lista, porém a diferença é quase insignificante. A lista levou vantagem ao se tratar do número de comparações, mas poderia ter sido muito melhor se os grafos fossem esparsos ou não direcionados.

7. Referencias

Referências

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.