Análise da Complexidade do Algoritmo de Floyd-Warshall

César Garcia Daudt
Caio Licks Pires de Miranda
Instituto de Informática
Universidade Federal do Rio Grande do Sul

13/10/2010

Resumo

Este artigo se propõe a fazer uma breve apresentação do algoritmo de Floyd-Warshall, exemplificar algumas de suas aplicações e desenvolver o cálculo de sua complexidade. Mostrará também um exemplo simples de execução do algoritmo.

1 Introdução

O algoritmo de Floyd-Warshall, desenvolvido independentemente três vezes – por Bernard Roy, Stephen Warshall e Robert Floyd –, encontra o menor caminho entre todos os pares de vértices de um grafo valorado[2]. Vale ressaltar que ele apenas encontra os valores de tais caminhos, e não a seqüência de arestas a ser percorrida.

Algumas aplicações deste algoritmo, além da fundamental:

- Calcular o Fecho Transitivo de um grafo[2].
- Verificar se um grafo não-dirigido é bipartido[2].
- Achar um nodo central, i.e., aquele que minimiza a distância máxima ou média entre todos os vértices[1].
- Calcular o diâmetro de um grafo[2].

Em uma situação prática, poderíamos pensar, por exemplo, em como avaliar o melhor local para instalarmos uma loja. De fato, podemos definir como melhor local aquele que diminui a distância entre a loja e locais estratégicos e importantes para ela, e.g.:

- Um bairro onde o consumo dos produtos vendidos por ela é alto
- Estabelicimentos que prestarão serviços para a loja
- Um local onde se tenha uma grande concentração de um público alvo para a loja

É fácil notar que este lugar será um ponto "central", ou seja, um lugar que minimiza a distância máxima ou média de todos os outros pontos considerados até ele.

Algoritmos de função semelhante incluem:

- **Algoritmo de Dijkstra**: Dado um nodo de um grafo, encontra a menor distância entre ele e todos os outros nodos.
- **Algoritmo de Johnson**: Semelhante ao Floyd-Warshall, mas otimizado para se trabalhar com grafos esparsos.

2 Formalização

2.1 Idéia

O algoritmo preenche uma matriz bidimensional, *caminho[][]*, onde *caminho[a][b]* é o tamanho do menor caminho entre os nodos a e b. Além disso, assume-se que esta matriz está inicialmente preenchida com o valor de cada aresta ou infinito, caso não haja uma aresta entre dois vértices.

Começamos fixando um vértice k do grafo; para cada par (i,j) de vértices, então, verificamos se o menor caminho já conhecido entre (i,j) supera a soma do tamanho do caminho de i para k com o de k para j. Caso supere, o tamanho do menor caminho passa a ser essa soma.

2.2 Pseudocódigo

```
1 func floyd_warshall(caminho[][])
2   for k = 1 to n
3       for i = 1 to n
4           for j = 1 to n
5               caminho[i][j] = min(caminho[i][j],
6                                   caminho[i][k]+caminho[k][j])
```

2.3 Dedução da complexidade

Cada laço *for* (linhas 2 a 4) pode ser convertido em um somatório com o mesmo valor inicial e final. Considerando a comparação (decisão do valor mínimo entre dois números) como operação elementar e as atribuições e acessos a matrizes como tempo constante, temos que a complexidade do algoritmo de Floyd-Warshall é dada por:

$$\sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} 1 \tag{1}$$

Pode-se converter o somatório da variável j em n, pois estamos somando n vezes o valor 1, e isolar este termo, que não depende da variável j, resultando no seguinte somatório:

$$n\sum_{k=1}^{n}\sum_{i=1}^{n} 1 \tag{2}$$

Repetindo o mesmo processo de dedução usado em 1, temos como resultado 5.

$$n \sum_{k=1}^{n} n \tag{3}$$

$$n^2 \sum_{k=1}^{n} 1 \tag{4}$$

$$n^3 \tag{5}$$

Logo, é de imediato que notamos que o algoritmo de Floyd-Warshall possui complexidade $\mathbf{O}(\mathrm{n}^3)$, onde n é o número de vértices do grafo fornecido.

3 Exemplo de Execução

No exemplo abaixo, usamos o código disponível em [4] sobre o grafo da Figura em [3] (matriz de adjacência em [3]), mostrando as matrizes resultantes de cada iteração do algoritmo. As iterações nas quais não houve modificações foram omitidas.

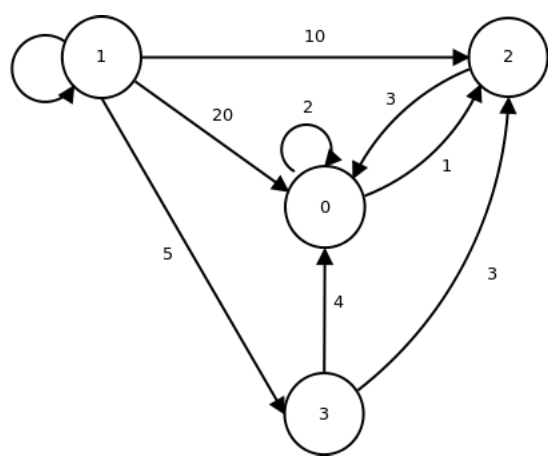

Figura 1: Grafo

	0	**1**	**2**	**3**
0	2	∞	1	∞
1	20	1	10	5
2	3	∞	∞	∞
3	4	∞	3	∞

Tabela 1: Matriz de adjacência do grafo

	0	**1**	**2**	**3**
0	2	∞	1	∞
1	20	1	10	5
2	3	∞	4	∞
3	4	∞	3	∞

Tabela 2: Iteração 11

	0	**1**	**2**	**3**
0	2	∞	1	∞
1	13	1	10	5
2	3	∞	4	∞
3	4	∞	3	∞

Tabela 3: Iteração 37

	0	**1**	**2**	**3**
0	2	∞	1	∞
1	9	1	10	5
2	3	∞	4	∞
3	4	∞	3	∞

Tabela 4: Iteração 53

	0	**1**	**2**	**3**
0	2	∞	1	∞
1	9	1	8	5
2	3	∞	4	∞
3	4	∞	3	∞

Tabela 5: Iteração 55

4 Exemplo de implementação

```python
#!/usr/bin/env python

def print_matriz(matriz):
    for i in range(len(matriz)):
        print matriz[i]
    print

def floyd_warshall(caminho):
    size = len(caminho)
    iteracao = 1

    for k in range(size):
        for i in range(size):
            for j in range(size):
                menor = min(caminho[i][j],
                            caminho[i][k]+caminho[k][j])

                if menor != caminho[i][j]:
                    caminho[i][j] = menor
                    print "Iteracao ", iteracao
                    print_matriz(caminho)

                iteracao += 1

INFINITY = float("Infinity")
grafo = [[2, INFINITY, 1, INFINITY],
         [20, 1, 10, 5],
         [3, INFINITY, INFINITY, INFINITY],
         [4, INFINITY, 3, INFINITY]]

print("Grafo:")
print_matriz(grafo)

floyd_warshall(grafo)
```

5 Referências

Referências

[1] Steven S. Skiena, Miguel A. Revilla .*Programming Challenges - The Programming Contest Training Manual.* Springer 1st edition, 2003.

[2] *Floyd-Warshall Algorithm - Wikipedia, the free encyclopedia*, disponível em `http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm`.