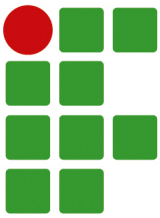

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Goiás

PEDRO HENRIQUE SILVA RODRIGUES

Análise de Complexidade: Algoritmo de Dijkstra

Anápolis - GO
Julho de 2017

PEDRO HENRIQUE SILVA RODRIGUES

Análise de Complexidade: Algoritmo de Dijkstra

Relatório apresentado como requisito parcial para obtenção de nota na disciplina de Teoria dos Grafos do curso Bacharelado em Ciência da Computação do Instituto Federal de Goiás, Câmpus Anápolis

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS

CAMPUS ANÁPOLIS

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Anápolis - GO

Julho de 2017

Resumo

A análise de complexidade de um algoritmo tem como objetivo verificar a eficiência desse com base na quantidade de operações que realiza. Embora a cada dia os computadores estejam mais rápidos, desenvolver softwares otimizados é indispensável no contexto da computação. Algoritmos que implementam grafos geralmente são utilizados com grandes volumes de dados, dessa forma é importante analisar o seu desempenho.

Palavras-chave: algoritmo, complexidade, grafos, desempenho.

Sumário

1 Introdução

Um algoritmo é qualquer procedimento computacional bem definido que toma algum valor, ou conjunto de valores, como entrada e produz algum valor, ou conjunto de valores, como saída. Um algoritmo é, portanto, uma sequência computacional de etapas que transformam a entrada na saída (CLRS, 2001).

Em Análise de Algoritmos conta-se o número de operações consideradas relevantes realizadas pelo algoritmo e expressa-se esse número como uma função de n. Essas operações podem ser comparações, operações aritméticas, movimento de dados, etc (SEDGEWICK, 1998). Embora o tempo de execução não dependa somente do algoritmo, mas do conjunto de instruções do computador, a qualidade do compilador, e a habilidade do programador, realizar a análise da comlexidade é importante pois assim temos uma previsão do desempenho que esse terá.

Um algoritmo só se tornará custoso se sua entrada for uma grande quantidade de dados, logo é interessante avaliar os algoritmos com base em grandes valores dos parâmetros. Grafos são usados para modelagens de circuitos elétricos, diagramas moleculares, aplicações em bancos de dados não relacionais, e diversas áreas da ciência, ou seja, aplicações com grandes quantidades de informações.

2 Fundamentação teórica

2.1 Algoritmo

Um algoritmo é qualquer procedimento computacional bem definido que toma algum valor, ou conjunto de valores, como entrada e produz algum valor, ou conjunto de valores, como saída. Um algoritmo é, portanto, uma sequência computacional de etapas que transformam a entrada na saída(CLRS, 2001).

2.2 Grafos

Um grafo é uma que representação abstrata de um conjunto de objetos e das relações existentes entre eles. É definido por um conjunto de nós ou vértices, e pelas ligações ou arestas, que ligam pares de nós. Uma grande variedade de estruturas do mundo real podem ser representadas abstratamente através de grafos.

2.3 Análise de compexidade

A complexidade de um algoritmo não necessariamente condiz com o tempo de processamento de algoritmo, de uma forma mais genérica está relacionado a quantidade de operações executadas com base em algum parâmetro específico do problema que o algoritmo implementa (SEDGEWICK, 1998).

A complexidade de um algoritmo pode variar dependendo das entradas que são fornecidas, haverão empre o melhor caso, o pior caso e o caso médio.

2.4 Algoritmo de Dijkstra

O algoritmo de Dijkstra foi criado pelo cientista da computação holandês Edsger Dijkstra em 1956 e publicado em 1959. Esse algoritmo soluciona o problema do caminho mais curto num grafo dirigido ou não dirigido com arestas de peso não negativo.

3 Problema

Realizar a análise do algoritmo de Dijkstra, evidenciando os parâmetros e os detalhes dessa análise. O Algoritmo se encontra abaixo:

```
function distanciasAoVertice
(verticeAvaliado, matrizAdjacencia, numeroVertices) {
    setValue(verticesAbertos, numeroVertices, 1)
    setValue(custos, numeroVertices, infinito)
    custos[verticeAvaliado] = 0
    quantidadeDeVerticesFechados = 1
while (quantidadeDeVerticesFechados < numeroVertices) {
    for    (i = 0; i < numeroVertices; i++) {
     if     (i != verticeAvaliado) {
    if (custos[i] > custos[verticeAvaliado] +
    matrizAdjacencia[verticeAvaliado][i]) {
    custos[i] = custos[verticeAvaliado] +
    matrizAdjacencia[verticeAvaliado][i]
    end If
    end If
    end For
    verticesAbertos[verticeAvaliado] = 0
    quantidadeDeVerticesFechados = quantidadeDeVerticesFechados + 1
    menor = infinito
    for    (i = 0; i < numeroVertices; i++) {
     if (menor > custos[i] and verticesAbertos[i]) {
    menor = custos[i]
    verticeAvaliado = i
    end If
    end For
```

end While

return custos

end Function

4 Resultados

Ao realizar a análise de complexidade do algoritmo de Dijkstra tomaremos como parâmetro n, a quantidade de vértices de um dado grafo G. Tal grafo no algoritmo é representado pela sua matriz de adjacência. Essa é uma matriz nxn onde cada elemento aij informa o custo ao ir do vértice i para o j caso haja uma aresta adjacente à eles, ou infinito caso não exista.

Essa matriz possui algumas características importantes: A diagonal principal representa o custo de se ir de um vértice pra ele mesmo, e sempre vale zero. É uma matriz simétrica se o grafo G for não-orientado, e é assimétrica caso seja orientado.

A cada iteração do laço principal do algoritmo um vértice é fechado, e só termina quando todos os vértices estão fechados, logo ocorrerão no máximo n iterações desse laço.

O primeiro laço for dentro do laço while será executado n vezes a cada iteração principall, pois não a break, pois não é feita nenhuma alteração dos contadores dentro de seu domínio.

Consideraremos tanto as operações de comparação quanto as operações aritméticas como operações atômicas. Há então cinco operações atômicas dentro do domínio desse for logo haverão $5n$ operaçõoes.

Ao chegar nessa função que obtém o vértice de menor custo ainda aberto, existe a possibilidade de forçar uma parada no laço principal, que ocorre quando esse menor custo é infinito, pois há vértices desse grafo que são desconexos, ou de forma mais genérica pra cada um dos ne vértices desse grafo que não estão conexos a origem implica que serão executadas n iterações principais, o que siginifica que na hipótese do grafo totalmente desconexo será executado apenas uma iteração principal.

Para se obter o vértice de menor custo ainda aberto é feito uma busca por todos os vértices. Ou seja, outro laço com n iterações de quatro operações atômicas.

$F(n) = n + n + c_1 + c_2 + n[c_3n + (c_4 + c_5)(n + 1) + c_6n + O(n) + O(n) + c_7(n - 1) + c_8(n - 1) + c_9(n - 1) + n + 1 + (c_{10} + c_{11})n + O(n) + O(n)] + c_{12}$

$F(n) = c_{13} + c_{14}n + c_{15}n^2$

Desprezamos os termos de baixa ordem, ignoramos o coeficiente constante. Logo o algoritmo de Dijsktra tem complexidade $O(n^2)$

Conclusão

Com base nessa análise obtivemos que no pior caso a complexidade do algoritmo de Dijkstra é $O(n^2)$, e que no melhor caso é $O(n)$. Embora um algoritmo de complexidade de ordem quadrática seja custoso, para o caso de fazer buscas numa matriz (aqui a matriz de adjacência) essa complexidade é normal uma vez que tem que se avaliar praticamente todos os valores dessa matriz, portanto podemos concluir que esse algoritmo resolve bem o problema da distância mínima entre vértices de um grafo.

Referências

CLRS. *Introduction to Algorithms*. [S.l.: s.n.], 2001. Citado 2 vezes nas páginas 4 e 5.

SEDGEWICK. *Algorithms*. [S.l.: s.n.], 1998. Citado 2 vezes nas páginas 4 e 5.