Teoria sobre Algoritmos

Complexidade

Sumário

- Introdução
- Recursos Computacionais
- Análise de Algoritmo
- Problemas Difíceis
- Considerações Finais
- Referências

Introdução

- Antes de executar um programa, pela primeira vez, queremos saber:

  1) O programa vai entrar em loop?
     - indecidível
  2) Quanto tempo o programa vai demandar?
  3) Há memória suficiente para executar o programa?
     - A resposta às perguntas 2 e 3 depende do conhecimento sobre o algoritmo do programa.

Recursos Computacionais

- Recursos Computacionais
  - Espaço (memória)
  - Tempo (processador)

- Estimar o uso dos recursos de um programa ANTES de executá-lo depende da análise do algoritmo.

Análise de Algoritmo

- Área de pesquisa da CC.
- Tempo ou Espaço.
- Teórica ou Experimental.
- O resultado da análise pode variar conforme:
  - Os dados;
  - A necessidade (melhor caso, caso médio, pior caso).

Análise Experimental

- Demanda a existência de um programa.
  - O algoritmo precisa estar implementado.
- O resultado varia conforme a implementação do algoritmo.
- Útil para validar a Análise Teórica.

Análise Teórica

- Demanda estudo do algoritmo.
  - As operações e as estruturas de dados mais relevantes são consideradas.
- Determina o custo dos recursos computacionais.
  - Independente do hardware.
- O custo é expresso em termos de funções de complexidade.

Exemplo de Análise Teórica

- Considere o programa seguinte

```
int v[MAX]; // v keeps natural numbers in order
int free;   // keep the free position in v


// return d, if d can be put in v; otherwise return -1
int insert (int d)
{
   if (free == MAX) return -1;
   for (i = free++; i > 0 && d < v[i-1]; i--)
      v[i] = v[i-1];
   return v[i] = d;
}
```

Quanto tempo?

Quanto espaço?

Tempo: f(N) = N, tq N = num.itens em v
Espaço: f(M) = M, tq M = tam. v

Função de Complexidade

- Expressa o comportamento do algoritmo quanto ao consumo de um recurso.
- Algumas funções:
  - log N
  - N
  - NlogN
  - N^i i > 1
  - 2^N

Complexidade Polinomial

Complexidade Exponencial

N representa o tamanho do problema a ser pelo algoritmo.

Por que se preocupar com complexidade do algoritmo?

- Porque um algoritmos eficiente minimiza o custo com um novo hardware.
- Um novo hardware é solução momentânea.

Tam.problema = 10^6

OPS	N	NlgN	N^2
10^6	s	s	S
10^9	!	!	h
10^{12}	!	!	s

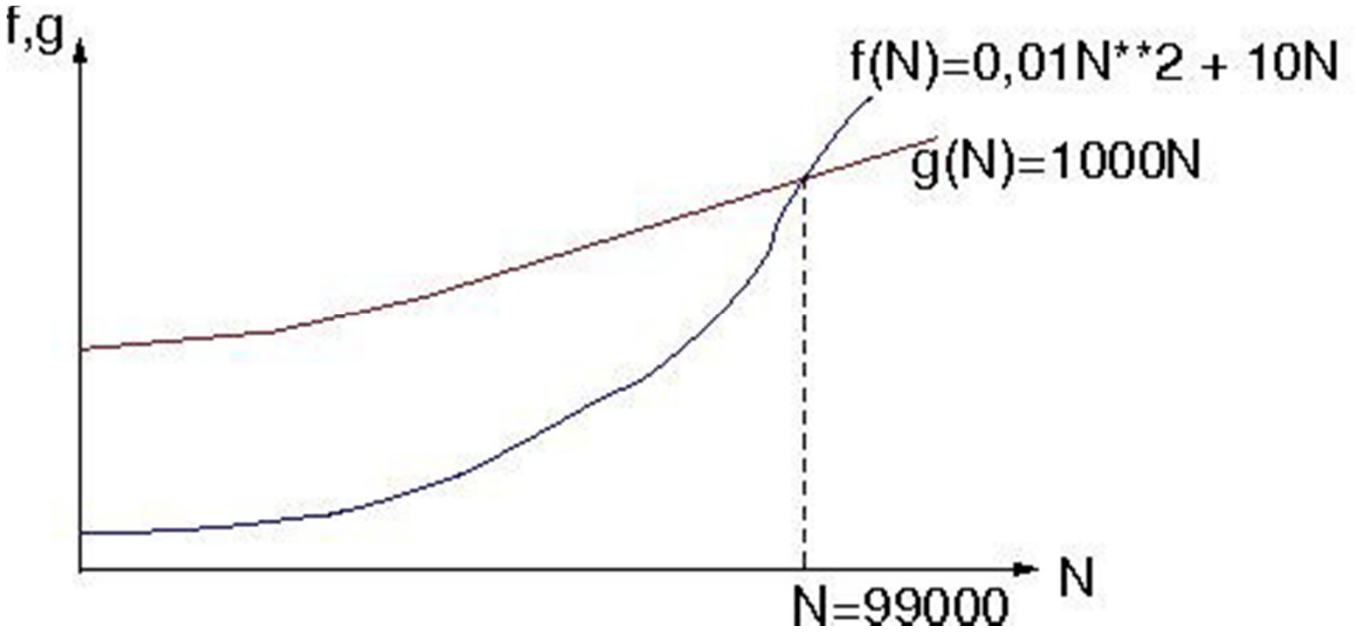
Tam.problema = 10^9

OPS	N	NlgN	N^2
10^6	h	h	?
10^9	s	s	D
10^{12}	!	!	s

s(egundos); S(emanas); h(oras); D(écadas); ! (imediato); ? (indefinido)

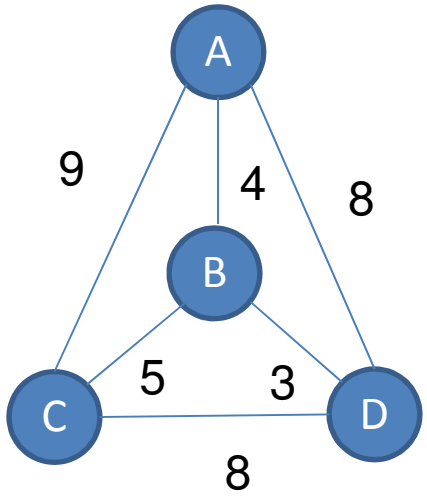
Comparando Algoritmos

- Qual dos algoritmos é o mais rápido?
  - f(N) = 0,01N2 + 10N
  - g(N) = 1000N + 10



Problemas Difíceis

- Problemas “difíceis” são aqueles cujas soluções algorítmicas possuem funções de complexidade da ordem $O(c^N)$, $c > 1$.
- Exemplo:
  - problema do caixeiro viajante
  - O(N!) adições
    - N = 50; 50! ≈ 10^{64}
    - 10^{45} séculos; 10^9 adições p/s



Considerações Finais

- A função de complexidade (custo) de um algoritmo está relacionado com gasto de um recurso computacional (tempo ou espaço).

- Os problemas computáveis têm diferentes custos (funções de complexidade).

- Os problemas “difíceis” são mais numerosos do que os “fáceis”.

Referências

- Brookshear - Cap.11
- Teoria da Computação - Tiaraju Asmuz Diverio & Paulo Blauth Menezes - Cap.3 (3.4) e Cap.5
- Goldschlager - Cap.3 (3.1)
- Projeto de Algoritmos – Nívio Ziviani – Cap.9 – 3ª ed.