Introdução

Noções, Fundamentos e Problemas Importantes

Aula 1
Alessandro L. Koerich

Pontifícia Universidade Católica do Paraná (PUCPR)

Ciência da Computação – 7º Período
Engenharia de Computação – 5º Período

alekoe@ppgia.pucpr.br

Programa do PA

Introdução
- 1. Resolução de Problemas e Tipos de Problemas

Fundamentos da Análise da Eficiência de Algoritmos
- 2. Fundamentos
- 3. Notação Assintótica e Classe de Eficiência
- 4. Análise Matemática de Algoritmos
- 5. Análise Empírica de Algoritmos

Técnicas de Projeto de Algoritmos
- 6. Força Bruta
- 7. Dividir & Conquistar
- 8. Decrementar & Conquistar
- 9. Transformar & Conquistar
- 10. Compromisso Tempo-Espaço
- 11. Programação Dinâmica
- 12. Estratégia Gulosa
- 13. *Backtracking* & *Branch and Bound*
- 14. Algoritmos Aproximados

Limitações
- 15. Teorema do Limite Inferior
- 16. Árvores de Decisão
- 17. Problemas P, NP e NPC

Plano de Aula

- Noções de Algoritmos
- Fundamentos da Resolução de Problemas usando Algoritmos
- Tipos Importantes de Problemas

Por que Estudar Algoritmos?

- Razões Práticas & Teóricas
  - Devemos conhecer um conjunto de algoritmos de diferentes áreas
  - Devemos ser capazes de projetar novos algoritmos e analisar suas eficiências.
  - O estudo de algoritmos é reconhecidamente a pedra fundamental da ciência da computação.

Por que Estudar Algoritmos?

"Algoritmo é muito mais do que um ramo da ciência da computação. É o núcleo da ciência da computação, e com toda a imparcialidade, pode ser considerado relevante para a maioria das ciências, negócios e tecnologia"

David Harel.

Por que Estudar Algoritmos?

- Mais Razões
  - Programas de computadores não existiriam sem algoritmos
  - Desenvolver habilidades analíticas

O que é um Algoritmo ?

- Algoritmos podem ser vistos como soluções especiais para problemas
- Os algoritmos não são soluções em si para problemas, mas procedimentos precisamente definidos para obter as soluções.

O que é um Algoritmo ?

- A palavra algoritmo vem do nome de um autor *Persa Abu Ja'far Mohammed ibn Musa al Khowarizmi* que escreveu um livro de matemática.
  www.lib.virginia.edu/science/parshall/khwariz.html
- **Def 1**: Em ciência da computação → método que pode ser usado por um computador para a solução de um problema
- **Def 2**: Uma seqüência finita de instruções ou operações básicas que se seguidas, realiza uma tarefa em particular.

O que é um Algoritmo ?

- **Def 3**: Um *algoritmo* é uma seqüência de instruções não ambíguas para resolver um problema, i.e., para obter uma saída desejada para qualquer entrada legítima em um intervalo de tempo finito.

O que é um Algoritmo ?

- Os algoritmos devem satisfazer aos seguintes critérios:
  - Entrada: uma ou mais quantidades são fornecidas externamente
  - Saída: Ao menos uma quantidade é produzida
  - Certeza: Cada instrução é clara e não ambígua
  - Finito: Se seguirmos as instruções de um algoritmo, então, para todos os casos, o algoritmo termina após um número finito de passos.
  - Efetividade: Cada instrução deve ser simples o suficiente de modo a permitir que uma pessoa utilizando somente lápis e papel realize–o.

Por que Estudar Algoritmos?

- Em resumo:

  Técnicas específicas de projeto de algoritmos podem ser interpretadas como estratégias para a resolução de problemas que podem ser úteis, estando ou não um computador envolvido.

- **Atenção!!!** Nem todos os problemas podem ser resolvidos por algoritmos. Ex. Como se tornar rico e famoso?

Metodologia

- Passo 1: Análise do Problema
- Passo 2: Projeto do Programa
- Passo 3: Implementação
- Passo 4: Verificação

Passo 1: Análise do Problema

- Compreensão correta do problema
- Entendimento das entradas e saídas

Passo 2: Projeto do Programa

- Desenvolver um ALGORITMO
- Definir uma estrutura de dados
- Otimizar o binômio tempo – espaço

Passo 3: Implementação

- Codificação do ALGORITMO em uma linguagem de programação

Passo 4: Verificação

- Demonstrar que o algoritmo realmente resolve o problema proposto, qualquer que seja sua instância

Exemplo Prático

- Três métodos para resolver um mesmo problema: computar o Maior Divisor Comum de dois números inteiros.
- O Maior Divisor Comum (MDC) ou *Greatest Common Divisor* (GCD) de dois números inteiros não negativos m e n, indicado como $mdc\,(m,n)$ é definido como o maior inteiro que divide tanto m quanto n exatamente, ou seja, com resto = 0.

Exemplo Prático

Algoritmo Euclideano

- Aplicar repetidamente a igualdade:

$$mdc\,(m,n) = mdc\,(n,\ m \bmod n)$$

até que $m \bmod n$ seja igual a zero (0). Como o maior divisor de $(m,0) = m$, o último valor de m é também o mdc de m e n iniciais.

- Obs: $m \bmod n$ é o resto da divisão de m por n.

Exemplo Prático

- Exemplo: $mdc\,(60,24)$

$$mdc\,(60,24) =$$
$$mdc\,(24,12) =$$
$$mdc\,(12,0) =$$
$$12$$

Exemplo Prático

- Uma descrição mais estruturada do algoritmo:
  - **Passo 1**: Se n=0, retorna o valor de m como resposta e para. Caso contrário executar o Passo 2.
  - **Passo 2**: Dividir m por n e atribuir o valor do resto para r.
  - **Passo 3**: Atribuir o valor de n para m e o valor de r para n. Ir para o Passo 1.

Exemplo

- Pseudocódigo do Algoritmo Euclidiano

Algoritmo Euclideano (m, n)
// Calcula mdc (m,n) usando o algoritmo euclideano
// Entrada: dois inteiros m e n não negativos, ambos não podem ser zero.
// Saída: maior divisor comum de m e n
enquanto $n \neq o$ ***faça***
$r \leftarrow m \ mod \ n$
$m \leftarrow n$
$n \leftarrow r$
retorna m

Exemplo

- Pseudocódigo do Algoritmo Euclidiano

ALGORITHM *Euclid*(m, n)
//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while $n \neq 0$ **do**
$r \leftarrow m \bmod n$
$m \leftarrow n$
$n \leftarrow r$
return m

Exemplo

- Como sabemos que o algoritmo Euclidiano para?
  - O segundo número do par diminui em cada iteração e ele não pode ser negativo.
  - Portanto o valor do segundo número eventualmente será 0 e o algoritmo para.

Exemplo

- A exemplo de vários outro problemas, existem outros algoritmos que resolvem o problema do maior divisor comum.
- Outro método...

Exemplo

Algoritmo **Verificação do Inteiro Consecutivo**

- **Passo 1**: Atribua o valor: $min\{m, n\}$ para t.
- **Passo 2**: Dividir m por t. Se o resto da divisão for 0, vá para o Passo 3. Caso contrário, vá para o Passo 4.
- **Passo 3**: Dividir n por t. Se o resto da divisão for 0, retorne o valor de t como resposta e pare. Caso contrário, vá para o Passo 4.
- **Passo 4**: Decrementar em 1 o valor de t. Ir para o Passo 2.

Exemplo

- Este algoritmo funciona corretamente?
  - Sim?
- E quando uma das entradas for zero?
- Por isso é importante especificar a faixa das entradas de um algoritmo explicitamente e cuidadosamente.

Exemplo

Algoritmo dos tempos de escola....

- **Passo 1**: Encontre os fatores primos de m
- **Passo 2**: Encontre os fatores primos de n
- **Passo 3**: Identificar todos os fatores comuns nas duas expansões encontradas nos Passos 1 e 2.
- **Passo 4**: Computar o produto de todos os fatores comuns e retorná–lo como o maior divisor comum

Exemplo

- O último algoritmo é muito mais complexo e lento que o algoritmo Euclidiano.
- Além disso, ele não é um algoritmo legítimo, pois:
  - Os passos de fatorização prima não estão definidos sem ambigüidades; É necessário uma lista de números primos.
  - O Passo 3 também não está definido claramente.

Tarefa

- Escreva um algoritmo para buscar uma chave *K* em um vetor de inteiros não ordenado *A*[0...*n*-1].
  - Entrada: Um vetor *A* de *n* números inteiros
  - Saída: A posição da chave K no vetor se presente ou -1 se a chave não estiver presente no vetor.
  - Pseudocódigo ou Descrição Estruturada

Problemas Clássicos

- Ordenação (*Sorting*)
- Busca (*Searching*)
- Processamento de cadeias (*String Processing*)
- Caminho mais curto (*Shortest paths in a graph*)
- Árvore de amplitude mínima (*Minimum spanning tree*)
- Caixeiro viajante (*Traveling salesman problem*)
- Mochila (*Knapsack problem*)
- Jogos (Chess)
- Torres de Hanoi (*Towers of Hanoi*)
- Otimização (*Optimization problems*)

Algoritmos são Importantes?

- Suponha que os computadores fossem infinitamente rápidos e que a memória do computador fosse ilimitada.
- Seria necessário estudar algoritmos ?
  - É obtida uma solução?
  - A resposta é correta?
- Porém ...
  - Em condições reais é importante considerar tempo e espaço

Eficiência de Algoritmos

- Algoritmos criados para resolver um mesmo problema diferem de forma drástica quanto a sua eficiência.
  - Exemplo: *Quicksort* versus *Bubblesort*
- Acredite!! Esta diferença é mais significativa que as diferença de *hardware* e *software* !!

Eficiência de Algoritmos: Exemplo

- Dados dois algoritmos de ordenação:
  - Algoritmo **ordenação por inserção**: assumimos que o "tempo de execução*" é $t=c_1n^2$
  - Algoritmo **ordenação por intercalação**: assumimos que o "tempo de execução*" $t=c_2n\ log_2\ n$
  - Normalmente as constantes $c_1 < c_2$
  - Normalmente c_1 e c_2 são menos significativos que n

* Apesar de chamarmos de "tempo de execução", normalmente não há uma relação direta deste tempo com o "tempo físico"

Eficiência de Algoritmos

- Exemplo: Comparar um computador rápido (*A*) que executa a ordenação por inserção com um computador mais lento (*B*) que executa a ordenação por intercalação.
- Cada um deles deve ordenar um arranjo de um milhão de números

Computador A
(1 bilhão op/sec)

Computador B
(10 milhões op/sec)

Eficiência de Algoritmos

Computador A
2000 segundos

Computador B
100 segundos

- Conclusão: Usando um algoritmo cujo tempo de execução cresce mais lentamente, até mesmo com um computador mais fraco, B funciona 20x mais rápido que A.
- E para 10 milhões de números ?

Eficiência de Algoritmos

- Conclusão: Na verdade, o desempenho total de um sistema computacional depende da combinação de :

Algoritmos Eficientes + Hardware Rápido

Fundamentos P&A Algoritmos

- Podemos considerar algoritmos como soluções procedurais para os problemas
- Estas soluções não são respostas, mas *instruções específicas* para obter respostas.
- A seguir é apresentada uma seqüência de passos normalmente seguida para projetar e analisar um algoritmo

Fundamentos P&A Algoritmos

Entender o Problema

- A primeira coisa antes de projetar um algoritmo é entender completamente o problema em questão.
  - Ler o problema
  - Formular questões
  - Fazer alguns exemplos a mão
  - Pensar a respeito de casos especiais
  - Formular questões novamente se necessário

Fundamentos P&A Algoritmos

Entender o Problema

- É importante especificar “exatamente” uma faixa de instâncias que o algoritmo deve tratar.
- **Lembre–se**: Um algoritmo correto não é aquele que funciona na maioria do casos, mas aquele que funciona corretamente para todas as entradas legítimas !!!

Fundamentos P&A Algoritmos

Verificar a Capacidade do Computador

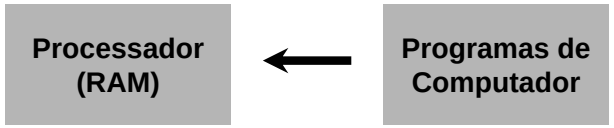
- A maioria dos algoritmos foi concebida para ser programado em RAM, isto é, uma instrução é executada após a outra.
- Porém, esta suposição (RAM) não se mantém para o novos computadores . . .
- Ainda assim, o estudo de técnicas de projeto e análise de algoritmos sob o modelo RAM continuarão por um longo tempo.

Observação Importante

- A vasta maioria dos algoritmos em uso atualmente foram desenvolvidas para ser programados como uma máquina de *von Neumann*
- Modelo *Random Access Machine* (RAM): instruções são executadas uma após as outras, uma operação de cada vez.
- Assim, trataremos somente com *algoritmos seqüenciais*.

Observação Importante

- Suposição: Modelo de computação genérico (RAM – Random Acess Machine)



- Neste modelo as instruções são executadas uma após outra, sem operações concorrentes

Fundamentos P&A Algoritmos

- Devemos nos preocupar com a velocidade e quantidade de memória do computador?
  - Não . . .
  - Sim. No caso de problemas muitos complexos por natureza que devam processar grandes quantidades de dados ou aplicações onde o tempo é crítico.

Fundamentos P&A Algoritmos

Solução Exata x Aproximada

- Existem problemas importantes que não podem ser resolvidos com exatidão (raiz quadrada, equações não lineares, etc.)
- Além disso os algoritmos disponíveis para resolver com exatidão alguns problemas podem ser inaceitavelmente lentos. Ex. caixeiro viajante.

Fundamentos P&A Algoritmos

Escolher a Estrutura de Dados Apropriada

- Já foi visto em PA preliminares

Fundamentos P&A Algoritmos

Técnicas de Projeto de Algoritmos

- Tendo agora todos os componentes, como projetar um algoritmo para resolver o problema?
- Esta é a questão que o PA pretende responder, apresentado diferentes técnicas de projeto.

Fundamentos P&A Algoritmos

Métodos para Especificar um Algoritmo

- Pseudocódigo: mistura de linguagem natural e construtores de linguagens de programação (*if*, *for* , *while*, etc.)
- Fluxograma . . .

Fundamentos P&A Algoritmos

Provar a Exatidão do Algoritmo

- Provar que o algoritmo produz um resultado esperado para toda entrada legítima em um intervalo de tempo finito
- Para alguns algoritmos é bem fácil enquanto que para outros pode ser muito complexo.
- Indução matemática

Fundamentos P&A Algoritmos

- Mostrar o funcionamento do algoritmo para algumas entradas pode ser interessante, porém, não prova a exatidão do algoritmo.
- Para algoritmos aproximados: erro produzido não excede um limite pré–definido.

Fundamentos P&A Algoritmos

Analisar um Algoritmo

- Geralmente desejamos que o nosso algoritmo possua diversas qualidades.
- Depois da correção, a mais importante é a eficiência.
  - Na verdade existem dois tipos de eficiência: eficiência temporal e eficiência espacial
- Outras características desejáveis são: simplicidade e generalidade

Fundamentos P&A Algoritmos

Codificar um Algoritmo

- Muitos algoritmos são destinados a implementação como programas de computador.
- Obviamente, a implementação correta de um algoritmo é necessária, mas não suficiente.

Fundamentos P&A Algoritmos

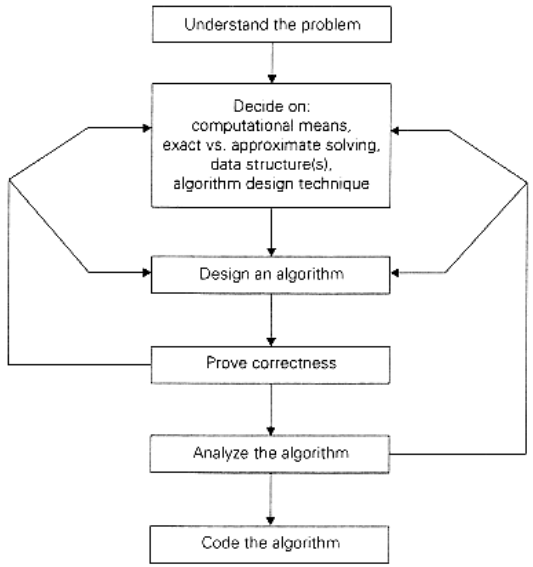


FIGURE 1.2 Algorithm design and analysis process

Tipos Importantes de Problemas

Face ao número ilimitado de problemas que encontramos em computação, alguns tipos de problemas merecem atenção especial, pois, aparecem com muita freqüência.

- **Problemas de Ordenação**: Reorganizar os itens de uma dada lista em ordem crescente.
- **Problemas de Busca**: Encontrar um dado valor chamado de chave de busca em um dado conjunto.
- **Processamento de *Strings***: Buscar uma dada palavra em um texto, avaliar a similaridade entre cadeias de caracteres, etc.

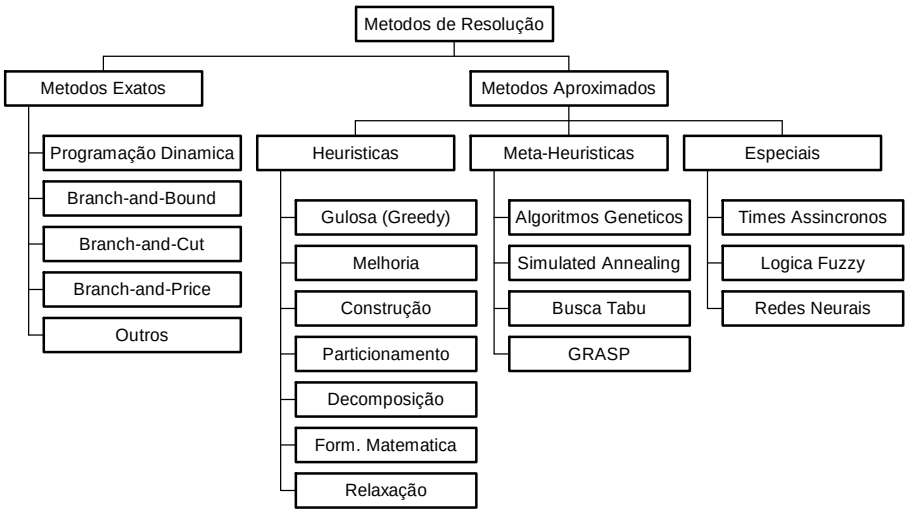
Tipos Importantes de Problemas

- **Problemas de Grafos**: Travessia de grafos (como visitar todos os pontos de uma rede) caminho mais curto (qual a melhor rota entre duas cidades), ordenação topológica.
- **Problemas Combinatoriais**: Problemas onde é necessário encontrar um objeto combinatorial (permutações, combinações ou subconjuntos) que satisfaça certas restrições e tenha certas propriedades (maximizar um valor, minimizar um custo).
- **Problemas Geométricos**: Envolvem objetos geométricos com pontos, linhas e polígonos.
- **Problemas Numéricos**: Envolvem objetos matemáticos de natureza contínua: resolução de equações e sistemas de equações, integrais definidas, etc.

Estratégias de Projeto de Algoritmos

- Força Bruta (*Brute Force*)
- Dividir e Conquistar (*Divide and Conquer*)
- Diminuir e Conquistar (*Decrease and Conquer*)
- Transformar e Conquistar (*Transform and Conquer*)
- Compromisso Tempo–Espaço (*Space and Time Tradeoffs*)
- Estratégia Gulosa (*Greedy*)
- Programação Dinâmica (*Dynamic Programming*)
- *Voltando Atrás* (*Backtracking*)
- Ramificar e Limitar (*Branch and Bound*)
- Algoritmos Aproximados

Estratégias de Projeto de Algoritmos



Resumo

- Com a capacidade cada vez maior dos computadores → resolver problemas mais complexos
- Em problemas maiores → eficiência dos algoritmos é importante
- Uma base sólida de algoritmos é uma característica que separa os programadores verdadeiramente qualificados dos novatos.

Resumo

- Conceitos básicos de análise de algoritmos
- Fundamentos para a resolução de problemas usando algoritmos

Resumo

Pontos Importantes

- Requisito da não ambigüidade para cada passo de um algoritmo
- A faixa de entradas para a qual o algoritmo funciona devem ser especificas cuidadosamente
- Um mesmo algoritmo pode ser representado de diferentes maneiras

Resumo

Pontos Importantes

- Podem existir diferentes algoritmos para resolver um mesmo problema
- Algoritmos para um mesmo problema podem ser baseados em diferentes idéias e podem resolver o problema em velocidades bem diferentes.