Estruturas de Dados e Básicas I - DIM0119

Selan R. dos Santos

DIMAp – Departamento de Informática e Matemática Aplicada
Sala 25, ramal 239, selan@dimap.ufrn.br
UFRN

2019.1

Introdução — Conteúdo

Motivação e Objetivos

- **Motivação**
  - Para compreendermos o conteúdo da disciplina precisamos de um conjunto mínimo de conceitos e definições sobre o objeto de estudo da disciplina: estruturas de dados (e algoritmos).
- **Objetivos**
  - Definir formalmente conceitos como algoritmo, problema computacional, tipo abstrato de dados, estruturas de dados, corretude e análise de complexidade.

Algoritmos e problemas computacionais

Definições

- O que é um **algoritmo**?
  - *R: É um processo sistemático para a resolução de um problema computacional.*
- Ok, mas o que é um **problema computacional**?

O problema de ordenação

- **Entrada**: uma sequência, $\langle a_1, \ldots, a_n \rangle$, de n objetos que aceitam uma ordenação total (por exemplo, números inteiros).
- **Saída**: uma permutação, $\langle a_{\pi_1}, \ldots, a_{\pi_n} \rangle$, da sequência de entrada tal que $a_{\pi_1} \leq a_{\pi_2} \leq \cdots \leq a_{\pi_n}$.

Algoritmos e problemas computacionais
Exemplos

- Por exemplo, dada a sequência de entrada $\langle 58, 41, 59, 26, 41, 31\rangle$, para se obter uma ordem não decrescente[1] de elemento a saída correspondente é
  - $\langle 26, 31, 41, 41, 58, 59\rangle$
- A sequência de entrada $\langle 58, 41, 59, 26, 41, 31\rangle$ é uma instância do problema da ordenação.
- Um problema computacional é, portanto, uma coleção (em geral, infinita) de instâncias e suas respectivas soluções (ou seja, as saídas).
- A área de projeto de algoritmos estuda métodos para resolver problemas computacionais de forma eficiente.

[1] *Ordem crescente com possível repetição.*

Algoritmos e problemas computacionais
Corretude e eficiência

- (Novamente) O que é um **algoritmo**?
  - *R (versão 2): É um processo sistemático para a resolução de um problema computacional que especifica uma sequência de ações executáveis que produz (*quando termina*) uma saída (solução) a partir de uma dada instância do problema.*
- Algoritmo: **E**ntrada (dados) $\Rightarrow$ **P**rocessamento $\Rightarrow$ **S**aída (solução)
- Um algoritmo é dito correto se, para qualquer instância dada, ele sempre **termina** e **produz a saída esperada** (ou seja, aquela associada à instância).
- Estudo de algoritmos envolve 2 aspectos básicos: corretude e complexidade.
  - Corretude: o algoritmo está **correto**?
  - Complexidade: o algoritmo é eficiente em termos dos recursos (**tempo de execução** e **uso de memória**) por ele utilizados?

Algoritmos e problemas computacionais
Tipos de problemas computacionais

- É possível categorizar alguns tipos de problemas computacionais.
- Problemas de decisão é uma classe de problemas para os quais a solução para cada instância é simplesmente *sim* ou *não*.

  "Dado um número inteiro n, determinar se n é primo."
- Problemas de busca engloba problemas cuja resposta pode ser uma *string* arbitrária.

  "Dado um inteiro positivo n, achar um fator primo não-trivial de n."

  Um problema de busca é representado por uma relação consistindo de todas os pares instância-solução; por exemplo, para o problema acima, a fatorização pode ser representada pela relação

  $$R = \{(4, 2), (6, 2), (6, 3), (8, 2), (9, 3), (10, 2), (10, 5), \dots\}$$

  que contém todos os pares de números (n, p), onde p é um fator primo não-trivial de n.

Algoritmos e problemas computacionais
Tipos de problemas computacionais

- Problemas de contagem requerem o número de soluções para um dado problema de busca.

  "Dado um inteiro positivo n, determine o número de fatores primo não triviais de n."
- Problemas de otimização procuram achar a "melhor solução" possível dentre todas as soluções possíveis de um problema de busca.

  Dado um grafo G, encontrar um conjunto independente de G de tamanho máximo."
  S é conjunto independente de $G \iff \forall u, v \in S : u \neq v \Rightarrow (u, v) \notin E$.

- Problemas função uma única saída (de uma função total) é esperada para cada entrada, sendo que tal saída é mais complexa do que "sim" ou "não".
  Um exemplo clássico é o problema do Caixeiro Viajante:
  "Dado uma lista de cidades e as distâncias entre cada par de cidades, encontre a rota de menor custo (ex. distância) possível que passa por cada cidade exatamente uma vez e retorna para a cidade original."

Tipos de dados

- Em linguagens de programação é importante classificar constantes, variáveis, expressões e funções de acordo com certas características que indicam o que denominamos de **tipo de dados**.
- O tipo de dados caracteriza o conjunto de valores:
  - a que uma constante pertence, ou
  - que podem ser assumidos por uma variável ou expressão, ou
  - que podem ser gerados por uma função.
- Tipos de dados simples são grupos de valores indivisíveis, tais como os tipos básicos `int`, `float`, `double`, `char` e `bool` da linguagem C++.
- Os **tipos estruturados** em geral definem uma coleção de valores simples ou um agregado de valores de tipos de dados diferentes.
- A linguagem C++, por exemplo, oferece uma grande variedade de tipos de dados simples e estruturados ou compostos.

Tipos Abstratos de Dados *vs* Estruturas de Dados

Tipo Abstrato de Dados (TAD)

Pode ser definido como um modelo matemático acompanhado das operações definidas sobre o modelo.

- Ex.: o conjunto dos inteiros munido das operações de adição, subtração e multiplicação é um exemplo de um tipo abstrato de dados.

- São exemplos de TADs:
  - Lista
  - Pilhas, Filas, Deques
  - Conjuntos
  - Listas de prioridade
  - Grafos
  - Árvores
- Em geral, o projeto de algoritmos para problemas computacionais complexos utilizam TAD extensivamente.
- A implementação de um algoritmo em uma linguagem de programação específica requer que encontremos alguma forma de representar TADs em termos dos tipos de dados e operadores suportados pela linguagem.

Tipos Abstratos de Dados *vs* Estruturas de Dados (cont.)

- Portanto, uma **estrutura de dados** é uma representação concreta de um TAD, ou seja, uma implementação de um modelo definido por TAD.
- Por exemplo, um TAD pilha (com operações definidas como push, pop e top) pode ser implementada por meio de um arranjo ou lista encadeada simples (as quais, por sua vez, são estruturas de dados!).
- Outro exemplo: um TAD Dicionário (associa chaves a valores) pode ser implementada como uma estrutura de dados mapa, que internamente utiliza outras estruturas de dados como tabela de dispersão ou lista encadeada simples.

Estruturas de Dados

- Estruturas de dados caracterizam-se pela disposição e manipulação de seus dados.
  - **disposição** está diretamente ligado à maneira usada para organizar os dados na memória — modelo matemático.
  - **manipulação** está diretamente ligado à ideia de algoritmos — operações.
- Em geral, os dados devem ser dispostos de forma a tornar eficientes o acesso e a modificação dos mesmos pelos algoritmos.
- Não há uma estrutura de dados melhor do que todas as demais para todos os algoritmos.

Estruturas de Dados (cont.)

- Cada problema a ser modelado reque uma análise para identificar a estrutura de dados mais adequada, de acordo com possíveis restrições de recursos e imposições do problema.
- Por isso é importante conhecer muito bem várias estrutura de dados para poder saber qual delas utilizar com um algoritmo ou propósito específico.
- Pelo exposto até então é possível perceber que os conceitos de algoritmo, *TAD* e estruturas de dados estão intimamente relacionados.
- **Algoritmos + Estrutura de Dados = Programas!**
- Em EDB1, para cada TAD/ED vamos estudar:
  - Visão lógica;
  - Operações;
  - Custo das operações (complexidade), e;
  - Implementação.

Exemplo #1: verificar pertinência de ponto em retângulos

Ponto em retângulo (problema computacional de decisão)

Denominamos de retângulo xy-alinhado um retângulo R cujos lados são paralelos aos eixos Cartesianos x e y. Tal retângulo é caracterizado por seu canto inferior esquerdo (R_x, R_y), e sua largura R_w e altura R_h.

- **Problema:** Seja R um retângulos xy-alinhados no plano Cartesiano e $p = (x, y)$ um ponto no plano Cartesiano. Escreva uma função que testa se p está contido em R. Considere que as arestas do retângulo fazem parte do mesmo. Retorne verdadeiro se p estiver contido em R ou falso, caso contrário.

Exemplo #2: verificar interseção de retângulos

Interseção de retângulos (problema computacional de decisão)

Denominamos de retângulo xy-alinhado um retângulo R cujos lados são paralelos aos eixos Cartesianos x e y. Tal retângulo é caracterizado por seu canto inferior esquerdo (R_x, R_y), e sua largura R_w e altura R_h.

- **Problema:** Sejam R e S dois retângulos xy-alinhados no plano Cartesiano. Escreva uma função que testa se R e S possui uma interseção não-vazia. Se a interseção for não-vazia, retorne o retângulo formado por sua interseção.

Referências


J. Szwarcfiter and L. Markenzon
Estruturas de Dados e Seus Algoritmos, 2ª edição, **Cap. 1**.
Editora LTC, 1994.


R. Sedgewick
Algorithms in C, Parts 1-4, 3rd edition. **Cap. 2**
Addision Wesley, 2004.


A. Drozdeck
Data Structures and Algorithms in C++, 2nd edition. **Cap. 2**
Brooks/Cole, Thomsom Learning, 2001.


D. Deharbe
Slides de Aula. **aula 2**
DIMAp, UFRN, 2006.


M. Siqueira
Slides de Aula. **aula 1**
DIMAp, UFRN, 2009.