Análise e Projeto de Algoritmos

Aula 1

Thiago Cavalcante – thiago.kun@gmail.com

25 de outubro de 2019

Universidade Federal de Alagoas – UFAL
Campus Arapiraca
Unidade de Ensino de Penedo

Informações Gerais da Disciplina

Nome	Análise e Projeto de Algoritmos
Código	SISB014
Semestre	5º
Carga Horária	80h
PPC	03/2018
Turma	2019.2
Horário	Sexta, 18:10 – 21:40
Sala	4
Lista de discussão	Google Groups: sisb014_20192
Repositório	GitHub: theagoliveira/sisb014_20192

Cronograma

Datas importantes	
22/11/2019	*Possível* dia da AB1
23/11/2019	Prazo final para digitação da AB1
14/02/2020	*Possível* dia da AB2
17/02/2020	Prazo final para digitação da AB2
17–22/02/2020	Período de reavaliação
27–29/02/2020	Período de provas finais

Ementa

- Conceitos básicos: motivação e solução de problemas; critérios de análise; correção e eficiência
- Análise de complexidade de algs.: custo; tempo de processamento; operações elementares; função de complexidade; classes de problemas; comparação de algs.
- Comportamento assintótico de funções: dominação assintótica; big O; theta; omega
- Técnicas de análise de algs.: equações de recorrência; teorema mestre
- Paradigmas de projeto de algs.: indução matemática; recursividade; algs. tentativa e erro; divisão e conquista; balanceamento; programação dinâmica; algs. gulosos e aproximados
- Análise de algs. de busca e ordenação; grafos; conhecidos
- Tratamento de problemas NP-completos

Bibliografia

SKIENA, S. S. The Algorithm Design Manual

Algoritmo

Procedimento para realizar uma determinada tarefa

Programa de computador

Algoritmo + estruturas de dados

Um algoritmo resolve um **problema**. O problema é especificado pela **instância**.

problema $\neq$ instância

Exemplo: Ordenação

Entrada: Uma sequência de n elementos $a_1, \ldots, a_n$

Saída: A permutação (reordenação) da sequência de entrada de forma que $a'_1 \leq a'_2 \leq \cdots \leq a'_{n-1} \leq a'_n$

Instâncias

Entrada 1: {“João”, “Thiago”, “Maria”, “Ana”}

Entrada 2: {100, 34, 57, 943, 48}

O algoritmo transforma qualquer possível instância na saída desejada.

Exemplo: *insertion sort*

Três propriedades de um bom algoritmo:

- Correto
- Eficiente
- Fácil de implementar

Três propriedades de um bom algoritmo:

- Correto
- Eficiente
- Fácil de implementar

Correção nem sempre é óbvia!

Correção nem sempre é óbvia!

Correção requer uma prova.

Exemplo: Otimização de caminho de um robô

Entrada: Conjunto de pontos em um plano

Saída: Menor caminho fechado que passa por todos os pontos

Heurística do vizinho mais próximo

VizinhoMaisPróximo(P)
  Escolha um ponto inicial $p0$ de P e o visite
  p = p_0
  i = 0
  Enquanto existirem pontos não visitados
    $i = i + 1$
    Seja p_i o ponto mais próximo não visitado de p_{i-1}
    Visite p_i
  Retorne para p_0 a partir de p_{n-1}

Heurística do par mais próximo

ParMaisPróximo(P)
  Seja n o número de pontos no conjunto P
  Para $i = 1$ até $n - 1$ faça
    $d = \infty$
    Para cada par (s, t) de pontas em cadeias distintas
      Se $dist(s, t) \leq d$, então $s_m = s$ e $t_m = t$ e $d = dist(s, t)$
    Conecte (s_m, t_m)
  Conecte as duas pontas restantes

PCVÓtimo(P)
  $d = \infty$
  Para cada $n!$ permutações P_i do conjunto P
    Se $(custo(P_i) \leq d)$, então $d = custo(P_i)$ e $P_{min} = P_i$
  Retorne P_{min}

PCVÓtimo(P)
  $d = \infty$
  Para cada $n!$ permutações P_i do conjunto P
    Se $(custo(P_i) \leq d)$, então $d = custo(P_i)$ e $P_{min} = P_i$
  Retorne P_{min}

20 pontos: só 2.432.902.008.176.640.000 permutações

PCVÓtimo(P)
  $d = \infty$
  Para cada $n!$ permutações P_i do conjunto P
    Se $(custo(P_i) \leq d)$, então $d = custo(P_i)$ e $P_{min} = P_i$
  Retorne P_{min}

20 pontos: só 2.432.902.008.176.640.000 permutações

1000 pontos: ??????

PCV: Problema do Caixeiro-viajante

Algoritmos sempre fornecem o resultado correto, **heurísticas** fazem um bom trabalho, porém sem garantias de que está correto.

Exemplo: Problema de agendamento

Entrada: Conjunto K de n intervalos em uma linha

Saída: Maior conjunto de intervalos de K que não se sobrepõem

```
PrimeiroTrabalho(K)
  Aceite o primeiro trabalho t de K que não se sobrepõe
  --> a outro trabalho aceito
  Repita até que não sobrem trabalhos
```

```
TrabalhoMaisCurto(K)
  Enquanto K não for vazio faça
    Aceite o trabalho t mais curto em K
    Exclua t e qualquer outro trabalho onde há
    --> sobreposição com t
```

```
AgendamentoExaustivo(K)
  j = 0
  S_max = 0
  Para cada um dos 2^n subconjuntos S_i de intervalos
    Se S_i não tem sobreposições e tam(S_i) > j
      então j = tam(S_i) e S_max = S_i
  Retorne S_i
```

```
AgendamentoÓtimo(K)
  Enquanto K não for vazio faça
    Aceite o trabalho t de K que termina mais cedo
    Exclua t e qualquer outro trabalho onde há
    --> sobreposição com t
```

Algoritmos aparentemente razoáveis podem estar incorretos. A correção de um algoritmo é uma propriedade que precisa ser demonstrada **cuidadosamente**.

Algoritmos podem ser expressados através de:

- Texto
- Pseudocódigo
- Programa real

Algoritmos podem ser expressados através de:

- Texto
- Pseudocódigo
- Programa real

O coração do algoritmo é uma **idéia**.

Especificação de problemas tem duas partes:

1. O conjunto permitido de instâncias de entrada
2. A propriedades exigidas na saída

Outro lado da moeda:
Demonstrando **incorreção** com contra-exemplos

Propriedades importantes de contra-exemplos:

- Verificabilidade
  - Calcular a resposta
  - Mostrar uma resposta melhor
- Simplicidade

Algumas estratégias:

- Pense pequeno
- Pense exaustivamente
- Procure extremos (grande/pequeno, esquerda/direita, poucos/muitos, perto/longe)