Projeto e Análise de Algoritmos

Prof. Dr. Ednaldo B. Pizzolato

www.dc.ufscar.br/~ednaldo

ednaldo@dc.ufscar.br

ESTRUTURAS DE DADOS

Fundamentos de estruturas de dados

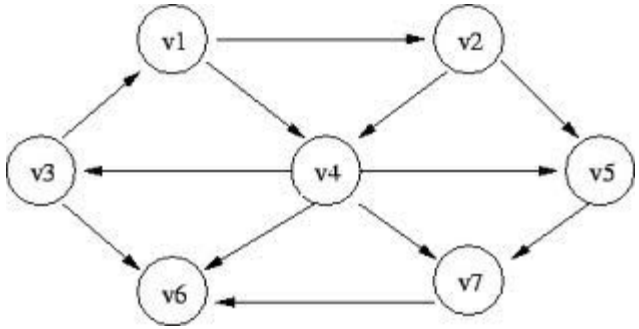
- Estruturas de dados lineares
- Grafos
- Árvores

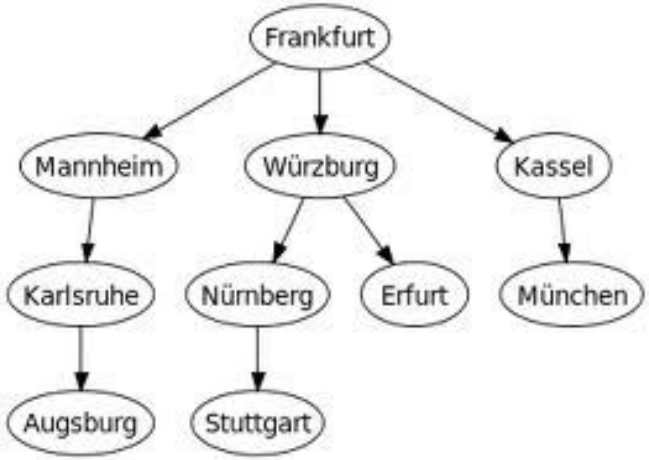
Fundamentos de estruturas de dados

- Estruturas de dados lineares
  - Vetores e matrizes
  - Strings
  - Listas encadeadas (simplesmente e duplamente)
  - Pilhas e filas (simples ou com prioridade)
- Grafos
- Árvores

Fundamentos de estruturas de dados

- Estruturas de dados lineares
- Grafos
  - Direcionamento
  - Representação
  - Grafos com pesos
  - Caminhos e ciclos
- Árvores





Fundamentos de estruturas de dados

- Estruturas de dados lineares
- Grafos
- Árvores
  - Com raiz
  - Ordenadas
  - Binárias
  - De busca
  - Multicaminhos

RESUMO

Resumo

- Um algoritmo é uma sequencia de instruções não ambíguas que devem resolver um problema em um tempo finito. Uma entrada para um algoritmo especifica uma instância do problema que o algoritmo se prontifica a resolver.
- Algoritmos podem ser especificados em linguagem natural ou em pseudocódigo e podem ser implementados na forma de programas de computadores.

Resumo

- Dentro de várias formas de classificar algoritmos, duas principais alternativas se destacam:
  - Agrupa-los de acordo com o tipo de problema que resolvem;
  - Agrupa-los de acordo com a técnica de projeto de algoritmo em que são baseados.

Resumo

- Tipos importantes de problemas são:
  - Ordenação;
  - Busca;
  - Processamento de cadeias de caracteres;
  - Associados a grafos;
  - Combinatoriais;
  - ...

Resumo

- Técnicas (estratégias ou paradigmas) de projeto de algoritmos são abordagens gerais para resolver problemas de forma algorítmica, aplicáveis a uma variedade de problemas de diferentes áreas da computação.

Resumo

- Um bom algoritmo é geralmente o resultado de esforços repetidos e retrabalho.
- Um mesmo problema pode ser resolvido por diversos algoritmos.
- Algoritmos trabalham com dados. Isso faz com que a questão de estruturação dos dados seja crítica na elaboração de algoritmos eficientes.

Exercícios

- Algoritmo de Euclides para MDC
  - MDC(m,n) = MDC(n, m mod n)
  - Lembrando que MDC(m,0) = m
  - MDC(60,24) = MDC(24,12) = MDC(12,0) = 12

Exercícios

- Algoritmo de Euclides

MDC_Euclides(m,n)

//Entrada: 2 números não negativos e não nulos ao mesmo tempo

//Saída: o máximo divisor comum entre eles

enquanto n ≠ 0 faça

r ← m mod n

m ← n

n ← r

retorna m

- Como sabemos que é finito?

Exercícios

- Consecutive integer checking MDC(m,n)
- Passo 1: t ← min(m,n)
- Passo 2: x ← m mod t. Se x = 0, vá para o passo 3; senão para o passo 4.
- Passo 3: x ← n mod t. Se x = 0, retorna t; senão vá para o passo 4.
- Passo 4: t ← t – 1. Vá para o passo 2.

- Funciona sempre?

Exercícios

PROBLEMA 1: Um andarilho está em um lado do rio com um lobo, uma ovelha e uma couve. Ele precisa transportar os 3 para o outro lado do rio em um barco. Entretanto, o barco comporta o andarilho e mais um item. Na ausência do andarilho, o lobo come a ovelha ou ela come a couve. Resolva este problema ou prove que não tem solução.

Exercícios

Passo 1:

Andarilho leva a ovelha

Lobo fica com a couve

Passo 2:

Andarilho retorna só

Lobo e couve no lado A

Ovelha no lado B

Passo 3:

Andarilho leva a couve

Passo 4:

Andarilho deixa a couve e pega a ovelha

Exercícios

Passo 5:

Andarilho deixa a ovelha e pega o lobo

Passo 6:

Andarilho deixa o lobo do outro lado com a couve

Passo 7:

Andarilho retorna só

Passo 8:

Andarilho pega a ovelha e faz a última viagem.

Exercícios

PROBLEMA 2: Há 4 pessoas que querem atravessar uma ponte e estão todos de um lado dela. Todos devem estar do outro lado da ponte dentro de 17 minutos. Mas é noite e há apenas uma lanterna. Assim, só é possível atravessar no máximo 2 pessoas. É claro que quem vai atravessar precisa estar com a lanterna. As pessoas 1, 2, 3 e 4 levam respectivamente 1, 2, 5 e 10 minutos para atravessar. Em duplas, o tempo de travessia será o do mais lento.

Exercícios

Ficou	Atravessou	Tempo acumulado
1 2 5 10		
5 10	1 2	2
5 10	1 (retorno)	3
1 5 10		
1	5 10	13
	2 (retorno)	15
1 2		
	1 2	17

Exercícios

DistanciaMinima(A[0..n-1])

// Entrada: vetor A[0..n-1] de números

// Saída: menor distância entre 2 elementos

```
dmin ← ∞
para i ← 0 até n-1 faça
   para j ← 0 até n-1 faça
      se i ≠ j e |A[i]-A[j]| < dmin
            dmin ← |A[i]-A[j]|
retorna dmin
```

Faça melhorias no algoritmo ou, se quiser ser radical, faça outro do zero melhor que o apresentado.

Exercícios

```
DistanciaMinima(A[0..n-1])
// Entrada: vetor A[0..n-1] de números
// Saída: menor distância entre 2 elementos
dmin ← ∞
para i ← 0 até n-2 faça
   para j ← i+1 até n-1 faça
     temp ← |A[i]-A[j]|
     se temp < dmin
          dmin ← temp
retorna dmin
```

Pelo menos não faz cálculos novamente!

THE END