Complexidade de Algoritmos

Paulino Ng

2020-04-03

Plano da aula

Esta aula apresenta a análise assintótica de alguns algoritmos simples.

1. Análise do algoritmo de impressão recursiva dos valores de uma lista encadeada.
2. Busca em um vetor aleatório
3. Busca em um vetor ordenado
4. Divisão e conquista: soluções recursivas. Equações de recorrência
5. Teorema Mestre para funções recursivas

Exercício discursivo ENADE-2017

Listas lineares armazenam uma coleção de elementos. A seguir, é apresentada a declaração de uma lista simplesmente encadeada.

```c
struct ListaEncadeada {
  int dado;
  struct ListaEncadeada *proximo;
};
```

Para imprimir os seus elementos da cauda para a cabeça (do final para o início) de forma eficiente, um algoritmo pode ser escrito da seguinte forma:

```c
void mostrar(struct ListaEncadeada *lista) {
  if (lista != NULL) {
    mostrar(lista->proximo);
    printf("%d ", lista->dado);
  }
}
```

Com base no algoritmo apresentado, faça o que se pede nos itens a seguir:

a) Apresente a classe de complexidade do algoritmo, usando a notação Θ.
b) Considerando que já existe implementada uma estrutura de dados do tipo pilha de inteiros - com as operações de empilhar, desempilhar e verificar pilha vazia - reescreva o algoritmo de forma não recursiva, mantendo a mesma complexidade. Seu algoritmo pode ser escrito em português estruturado ou em alguma linguagem de programação, como C, Java ou Pascal.

Solução

a) O algoritmo percorre a lista recursivamente, passando por cada nó uma única vez, logo, o algoritmo é: $\Theta(n)$

b) O "programa" para fazer o que é solicitado, faz um laço onde se desempilha um item, imprime, empilha numa pilha auxiliar. Estas operações são realizadas enquanto a pilha não estiver vazia. Após esvaziar a pilha original, deve-se fazer um novo laço para reempilhar os itens na pilha original.

Busca de um valor num vetor

Seja o código:

```
int busca(int *vi, int n, int val) {
  int i;
  for (i = 0; i < n; i++)
    if (vi[i] == val) return i;
  return -1;
}
```

Se o valor `val` fizer parte do vetor, a função retorna o índice do vetor onde está o valor, se não, a função retorna -1.

No melhor caso, o valor está na primeira posição: $f(n) = 1$

No pior caso, o valor está no fim do vetor ou não está no vetor: $f(n) = n$

No caso médio: $f(n) = \frac{n}{2}$

Busca de um valor num vetor ordenado

```
int busca2(int *vi, int n, int val) {
  int sup = n, inf = 0, meio = n/2;
  while (sup > inf) {
    if (vi[meio] == val) return meio;
    if (val > vi[meio]) inf = meio;
    else                sup = meio;
    meio = (sup + inf + 1)/2;
  }
  if (vi[inf] == val) return inf;
  return -1;
}
```

A cada passagem no laço, o espaço de busca diminui pela metade.

O espaço fica reduzido a 1 elemento em $\log_2 n$ iterações. Logo:

No melhor caso, $f(n) = 1$, o elemento do meio.

No pior caso, $f(n) = lgn$ (o espaço de busca se reduziu a 1, ou o elemento não está no vetor).

O caso médio é próximo do pior caso, $f(n) = lgn$

Divisão e conquista

- A `busca2()` pode ser reescrita como uma busca recursiva:

```
int busca_rec(int *vi, int ini, int fim, int val) {
  if (ini == fim) return (vi[ini] == val) ? ini : -1;
  int meio = (fim + ini)/2;
  if (vi[meio] == val) return meio;
  if (meio == ini) return (vi[fim] == val) ? fim : -1;
  if (val > vi[meio]) return busca_rec(vi, meio, fim, val);
  else return busca_rec(vi, ini, meio, val);
}
```

- O problema é dividido em problemas menores recursivamente até que sejam tão pequenos que possamos calcular a solução. Exemplo: cálculo do fatorial recursivo e cálculo do n-ésimo número da sequência de Fibonacci.

Teorema mestre para funções recursivas

Sejam $a \geq 1$ e $b > 1$ constantes, $f(n)$ uma função assintoticamente positiva e $T(n)$ uma medida de complexidade definida sobre os inteiros. A solução da equação de recorrência:

$T(n) = aT(\frac{n}{b} + f(n))$,

para n uma potência de b é:

1. $T(n) = \Theta(n^{\log_b a})$, se $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$;
2. $T(n) = \Theta(n^{\log_b a} \log n)$, se $f(n) = \Theta(n^{\log_b a})$; e
3. $T(n) = \Theta(f(n))$, se $f(n) = \Omega(n^{\log_b a + \epsilon})$para alguma constante $\epsilon > 0$, e se $a f(\frac{b}{b}) \leq cf(n)$ para alguma constante $c < 1$ e todo n a partir de um valor suficientemente grande.

- A equação de recorrência diz que o problema foi dividido em a subproblemas de tamanho $\frac{n}{b}$ cada um.
- Os problemas são resolvidos recursivamente em tempo $T(\frac{n}{b})$ cada um.
- A função $f(n)$ descreve o custo de dividir o problema em subproblemas e combinar os resultados de cada subproblema.