Estrutura de Dados
Algoritmos Recursivos

Alessandro Ferreira Leite

01/2012

Recursão

- Um objeto é dito recursivo se ele consistir parcialmente ou for definido em termos de si próprio. Recursões não são encontradas apenas em matemática mas também no dia a dia.
- Recursão é uma técnica particularmente poderosa em definições matemáticas. Alguns exemplos: números naturais, estrutura de árvore e certas funções:
  1. Números naturais:
     1. 0 é um número natural.
     2. O sucessor de um número natural é um número natural.
  2. Estruturas de árvores
     1. 0 é uma árvore (chamada árvore vazia).
     2. Se t_1 e t_2 são árvores, então a estrutura que consiste de um nó com dois ramos t_1 *e* t_2 também é uma arvore.
  3. A função fatorial $n!$
     1. $0! = 1$
     2. $n > 0, n! = n * (n - 1)$

Recursão

- Se uma função f possuir uma referência explícita a si próprio, então a função é dita *diretamente recursiva*. Se f contiver uma referência a outra função g, que por sua vez contém uma referência direta ou indireta a f, então f é dita *indiretamente recursiva*.
- Em termos matemáticos, a recursão é uma técnica que através de substituições sucessivas reduz o problema a ser resolvido a um caso de solução mais simples (Dividir para conquistar).

```
/**
 * Calcula a soma dos números inteiros
 * existentes entre in e n inclusive.
 */
int somatorio(int in, int n){
 int s = in;
 if (s < n)
 {
   return s + somatorio(s + 1, n);
 }
 return s;
}

public static void main(String args)
{
   print(somatorio(1, 100));
}
```

Recursão

1. Há dois requisitos-chave para garantir que a recursão tenha sucesso:
    1. Toda chamada recursiva tem de simplificar os cálculos de alguma maneira.
    2. Tem de haver casos especiais para tratar os cálculos mais simples diretamente.
2. Muitas recursões podem ser calculadas com laços. Entretanto, as soluções iterativas para problemas recursivos podem ser mais complexas.
3. Por exemplo, a permutação de uma palavra.

Recursão

- A permutação é um exemplo de recursão que seria difícil de programar utilizando laços simples.
- Uma permutação de uma palavra é simplesmente um rearranjo das letras. Por exemplo, a palavra "eat" tem seis permutações ($n!$, onde n é o número de letras que formam a palavra).
- Como gerar essas permutações?
- Simples, primeiro, gere todas as permutações que iniciam com a letra "e", depois as que iniciam com a letra "a" e finalmente as que iniciam com a letra "t".
- Mas, como gerar as permutações que iniciam com a letra "e"?
- Gere as permutações da sub-palavra "at". Porém, esse é o mesmo problema, mas com uma entrada mais simples, ou seja, uma palavra menor.
- Logo, podemos usar a recursão nesse caso.

Recursão

Como pensar recursivo

1. Combine várias maneiras de simplificar as entradas.
2. Combine as soluções de entradas mais simples para uma solução do problema original.
3. Encontre soluções para as entradas mais simples.
4. Implemente a solução combinando os casos simples e o passo de redução.

Eficiência da Recursão

1. A recursão pode ser uma ferramenta poderosa para implementar algoritmos complexos.
2. No entanto, a recursão pode levar a algoritmos que tem um desempenho fraco.
3. Vejamos quando a recursão é benéfica e quando é ineficiente.

Eficiência da Recursão

1. Considere a sequência de Fibonacci, uma sequência de números inteiros definidos pela equação:

$$f_1 = 1$$

$$f_2 = 1$$

$$f_n = f_{n-1} + f_{n-2}$$

2. Exemplo: $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \; \cdots.$
3. Vejamos uma implementação recursiva que calcule qualquer valor de n.

Eficiência da Recursão

```
  int fibonacci(int n) {
    if (n <= 2)
      return 1;
    else
      return fibonacci(n - 1) + fibonacci(n - 2);
  }

  void main(void) {
    int i;
    for (i = 1; i <= n; i++) {
      int f = fibonacci(i);
      printf("%d", f);
    }
  }
}
```

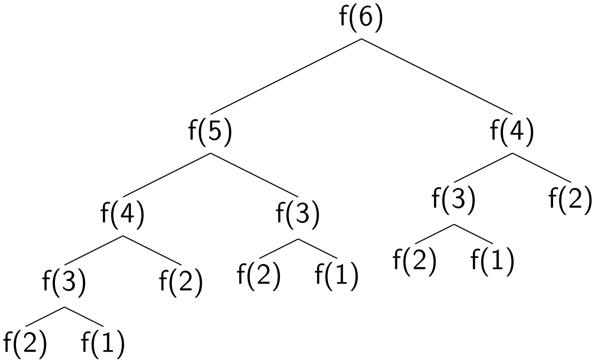
Eficiência da Recursão

1. Ao executarmos o programa de teste podemos notar que as primeiras chamadas à função **fibonacci** são bem rápidas. No entanto, para valores maiores, o programa pausa um tempo considerável entre as saídas.
2. Inicialmente isso não faz sentido, uma vez que podemos calcular de forma rápida com auxílio de uma calculadora esses números, de modo que para o computador não deveria demorar tanto em hipótese alguma.
3. Para descobrir o problema, vamos inserir mensagens de monitoração das funções e verificar a execução para $n = 6$.

Eficiência da Recursão

Início fibonacci n = 6
Início fibonacci n = 5
Início fibonacci n = 4
Início fibonacci n = 3
Início fibonacci n = 2
Término fibonacci n = 2, retorno = 1
Início fibonacci n = 1
Término fibonacci n = 1, retorno = 1
Término fibonacci n = 3, retorno = 2
Início fibonacci n = 2
Término fibonacci n = 2, retorno = 1
Término fibonacci n = 4, retorno = 3
Início fibonacci n = 3
Início fibonacci n = 2
Término fibonacci n = 2, retorno = 1
Início fibonacci n = 1
Término fibonacci n = 1, retorno = 1
Término fibonacci n = 3, retorno = 2
Término fibonacci n = 5, retorno = 5
Início fibonacci n = 4
Início fibonacci n = 3
Início fibonacci n = 2
Término fibonacci n = 2, retorno = 1
Início fibonacci n = 1
Término fibonacci n = 1, retorno = 1
Término fibonacci n = 3, retorno = 2
Início fibonacci n = 2
Término fibonacci n = 2, retorno = 1
Término fibonacci n = 4, retorno = 3
Término fibonacci n = 6, retorno = 8
Fibonacci(6) = 8

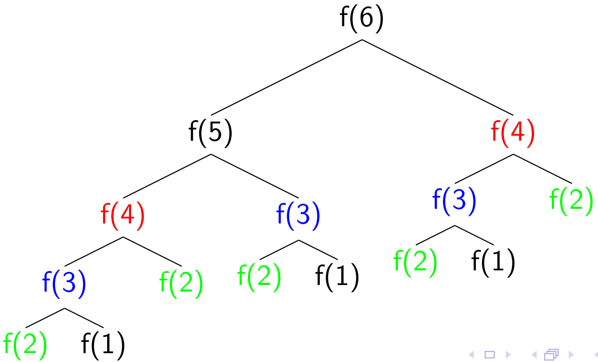
Eficiência da Recursão



Padrão de chamada de função/método recursivo *fibonacci*.

Eficiência da Recursão

1. Analisando o rastro de execução do programa fica claro porque o método leva tanto tempo.
2. Ele calcula os mesmos valores repetidas vezes.
3. Pelo exemplo, o calculo de **fibonacci(6)** chama **fibonacci(4)** duas vezes, **fibonacci(3)** três vezes, **fibonacci(2)** cinco vezes, e **fibonacci(1)** três vezes.
4. Diferente do cálculo que faríamos manualmente.



Em resumo...

Eficiência da Recursão

As vezes acontece de uma solução recursiva ser executada muito mais lentamente do que sua equivalente iterativa. Entretanto, na maioria dos casos, a solução recursiva é apenas levemente mais lenta.

Em muitos casos, uma solução recursiva é mais fácil de entender e implementar corretamente do que uma solução iterativa.