MC-102 — Aula 14
Introdução a Recursão

Instituto de Computação – Unicamp

Segundo Semestre de 2011

Roteiro

Recursividade

Motivação

Programas recursivos são, em geral, mais simples de se escrever, analisar e entender

Recursividade

Recursividade

Um objeto é dito recursivo se pode ser definido em termos de si próprio.

Recursividade

- A maior dificuldade para entender recursão é o fato de querermos, geralmente, mapear a idéia para dentro do computador e ao tentar fazer isso temos que entender a recursão e como o computador se comporta diante da recursão.
- Vamos tentar abstrair a forma como o computador lida com a recursão, e vamos nos ater apenas na técnica de recursão. Dominando isso, então poderemos partir para o entendimento de como a maquina lida com a recursão.

Recursividade

Recursão

É o processo de resolução de um problema, reduzindo-o em um ou mais subproblemas com as seguintes características:
1 - São idênticos aos problemas originais;
2 - São mais simples de resolver.

Recursividade

- Uma vez realizada a primeira subdivisão, a mesma técnica de decomposição é usada para dividir cada subproblema
- Eventualmente, os subproblemas tornam-se tão simples que é possível resolvê-los sem efetuar novas subdivisões
- A solução completa do problema original é obtida através da "montagem"das soluções componentes
- Este processo de resolução de problemas está diretamente ligado ao conceito de indução matemática.

Componentes da Recursão

Toda recursão é composta por

- Um caso base, uma instância do problema que pode ser solucionada facilmente. Por exemplo, é trivial fazer a soma de uma lista com um único elemento.
- Uma ou mais chamadas recursivas, onde o objeto define-se em termos de sí próprio, tentando convergir para o caso base. A soma de uma lista de n elementos pode ser definida a partir da lista da soma de $n - 1$ elementos.

Recursão na matemática

Como definir recursivamente a soma abaixo?

$$\sum_{k=m}^{n} k = m + (m+1) + \cdots + (n-1) + n$$

Recursão na matemática

Primeira definição recursiva

$$\sum_{k=m}^{n} k = \begin{cases} m & \text{se } n = m \\ \sum_{k=m}^{n-1} k + n & \text{se } n > m \end{cases}$$

Recursão na matemática

Segunda definição recursiva

$$\sum_{k=m}^{n} k = \begin{cases} m & \text{se } n = m \\ m + \sum_{k=m+1}^{n} k & \text{se } n > m \end{cases}$$

Recursão na computação

```
int soma(int m, int n) {
  if (m == n)
    return n;
  else
    return m + soma(m+1, n);
}
```

Veja o código: `soma.c`

Pilha de execução

- A cada chamada de função o sistema reserva espaço para parâmetros, variáveis locais e valor de retorno.

main	s
	ret: ??
soma	m: 5
	n: 10
	ret: ??
soma	m: 6
	n: 10
	...

Estouro de pilha de execução

"To understand recursion you must first understand recursion."

- O que acontece se a função não tiver um caso base?
- O sistema de execução não consegue implementar infinitas chamadas. (Lembre-se, somente Chuck Norris conta até o infinito).

Veja o código rec-infinita.c

Fatorial

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n.(n-1)! & \text{se } n > 0 \end{cases}$$

```
int fatorial(int n) {
  if (n == 0)
    return 1;
  else
    return n * fatorial (n-1);
}
```

Potência

$$x^n = \begin{cases} \frac{1}{x^{(-n)}} & \text{se } n < 0 \\ 1 & \text{se } n = 0 \\ x.x^{(n-1)} & \text{se } n > 0 \end{cases}$$

```
double pot(double x, int n) {
  if (n == 0) return 1;
  else if (n < 0)
         return 1/pot(x, -n);
       else
         return x*pot(x, n-1);
}
```

Questões de desempenho

- É sempre mais simples usar recursão?
- É mais eficiente?

Veja pot-iterativa.c

Número de dígitos

```
int num_digitos_rec(int n) {
if (abs(n)<=9)
return(1);
else
return (1+ num_digitos_rec(n/10));
}
```

Fibonacci

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