MC102 – Aula 22
Recursão II

Instituto de Computação – Unicamp

5 de Junho de 2012

Roteiro

1. Recursão – Relembrando
2. Cálculo de Potências
3. Soma de um Vetor
4. Exercício

Recursão - Relembrando

- Definições recursivas de funções são baseadas no *princípio matemático da indução* que vimos anteriormente.
- A idéia é que a solução de um problema pode ser expressa da seguinte forma:
  - Definimos a solução para os casos básicos;
  - Definimos como resolver o problema geral utilizando soluções do mesmo problema só que para casos menores.

Cálculo de Potências

Suponha que queiramos calcular x^n para n inteiro positivo. Como calcular de forma recursiva?

Cálculo de Potências

Suponha que queiramos calcular x^n para n inteiro positivo. Como calcular de forma recursiva?

- Vamos olhar um exemplo: 2^4
- $2^4 = 2 * 2^3 \Leftarrow$ subdividimos o problema
- $2^3 = 2 * 2^2 \Leftarrow$ subdividimos o problema
- $2^2 = 2 * 2^1 \Leftarrow$ subdividimos o problema
- $2^1 = 2 * 2^0 \Leftarrow$ subdividimos o problema
- $2^0 = 1 \Leftarrow$ Agora sabemos dar uma resposta para o problema!!!! Temos a base!

Cálculo de Potências

Suponha que queiramos calcular x^n para n inteiro positivo. Como calcular de forma recursiva?

- Vamos olhar um exemplo: 2^4
- $2^4 = 2 * 2^3 \Leftarrow$ subdividimos o problema
- $2^3 = 2 * 2^2 \Leftarrow$ subdividimos o problema
- $2^2 = 2 * 2^1 \Leftarrow$ subdividimos o problema
- $2^1 = 2 * 2^0 \Leftarrow$ subdividimos o problema
- $2^0 = 1 \Leftarrow$ Agora sabemos dar uma resposta para o problema!!!! Temos a base!

(Instituto de Computação – Unicamp) MC-102 5 de Junho de 2012 5 / 39

Cálculo de Potências

Suponha que queiramos calcular x^n para n inteiro positivo. Como calcular de forma recursiva?

- Vamos olhar um exemplo: 2^4
- $2^4 = 2 * 2^3 \Leftarrow$ subdividimos o problema
- $2^3 = 2 * 2^2 \Leftarrow$ subdividimos o problema
- $2^2 = 2 * 2^1 \Leftarrow$ subdividimos o problema
- $2^1 = 2 * 2^0 \Leftarrow$ subdividimos o problema
- $2^0 = 1 \Leftarrow$ Agora sabemos dar uma resposta para o problema!!!! Temos a base!

(Instituto de Computação – Unicamp) MC-102 5 de Junho de 2012 5 / 39

Cálculo de Potências

Suponha que queiramos calcular x^n para n inteiro positivo. Como calcular de forma recursiva?

- Vamos olhar um exemplo: 2^4
- $2^4 = 2 * 2^3 \Leftarrow$ subdividimos o problema
- $2^3 = 2 * 2^2 \Leftarrow$ subdividimos o problema
- $2^2 = 2 * 2^1 \Leftarrow$ subdividimos o problema
- $2^1 = 2 * 2^0 \Leftarrow$ subdividimos o problema
- $2^0 = 1 \Leftarrow$ Agora sabemos dar uma resposta para o problema!!!! Temos a base!

(Instituto de Computação – Unicamp) MC-102 5 de Junho de 2012 5 / 39

Cálculo de Potências

Suponha que queiramos calcular x^n para n inteiro positivo. Como calcular de forma recursiva?

- Vamos olhar um exemplo: 2^4
- $2^4 = 2 * 2^3 \Leftarrow$ subdividimos o problema
- $2^3 = 2 * 2^2 \Leftarrow$ subdividimos o problema
- $2^2 = 2 * 2^1 \Leftarrow$ subdividimos o problema
- $2^1 = 2 * 2^0 \Leftarrow$ subdividimos o problema
- $2^0 = 1 \Leftarrow$ Agora sabemos dar uma resposta para o problema!!!! Temos a base!

Cálculo de Potências

Agora vamos pensar a potência x^n, para n inteiro positivo, de forma genérica. Como calcular de forma recursiva?

x^n é:

- 1 se $n = 0$.
- xx^{n-1} caso contrário.

Cálculo de Potências

Suponha que queiramos calcular x^n para n inteiro positivo. Como calcular de forma recursiva?

- Vamos olhar um exemplo: 2^4
- $2^4 = 2 * 2^3 \Leftarrow$ subdividimos o problema
- $2^3 = 2 * 2^2 \Leftarrow$ subdividimos o problema
- $2^2 = 2 * 2^1 \Leftarrow$ subdividimos o problema
- $2^1 = 2 * 2^0 \Leftarrow$ subdividimos o problema
- $2^0 = 1 \Leftarrow$ Agora sabemos dar uma resposta para o problema!!!! Temos a base!

Cálculo de Potências

Agora vamos pensar a potência x^n, para n inteiro positivo, de forma genérica. Como calcular de forma recursiva?

x^n é:

- 1 se $n = 0$.
- xx^{n-1} caso contrário.

Cálculo de Potências

Agora vamos pensar a potência x^n, para n inteiro positivo, de forma genérica. Como calcular de forma recursiva?
x^n é:

- 1 se $n = 0$.
- xx^{n-1} caso contrário.

Cálculo de Potências

```
long pot(long x, long n){
  if(n == 0)
    return 1;
  else
    return x*pot(x,n-1);
}
```


Cálculo de Potências

```
long pot(long x, long n){
  if(n == 0)
    return 1;
  else
    return x*pot(x,n-1);
}
```


Cálculo de Potências

```
long pot(long x, long n){
  if(n == 0)
    return 1;
  else
    return x*pot(x,n-1);
}
```

Cálculo de Potências

```
long pot(long x, long n){

  if(n == 0)

    return 1;

  else

    return x*pot(x,n-1);
}
```


Cálculo de Potências

Neste caso a solução iterativa é mais eficiente.

```
long pot(long x, long n){
    long p = 1, i;
    for( i=1; i<=n; i++)
       p = p * x;
    return p;
}
```

- O laço é executado n vezes.
- Na solução recursiva são feitas n chamadas, mas tem-se o custo adicional para criação/remoção de variáveis locais na pilha.

Cálculo de Potências

```
long pot(long x, long n){

  if(n == 0)

    return 1;

  else

    return x*pot(x,n-1);
}
```


Cálculo de Potências

Mas e se definirmos a potência de forma diferente:
x^n é:

- Caso básico:
  - Se $n = 0$ então $x^n = 1$.
- Caso Geral:
  - Se $n > 0$ e é par, então $x^n = (x^{n/2})^2$.
  - Se $n > 0$ e é ímpar, então $x^n = x(x^{(n-1)/2})^2$.

Note como no caso geral definimos a solução do caso maior em termos de casos menores.

Cálculo de Potências

Mas e se definirmos a potência de forma diferente:
x^n é:

- Caso básico:
  - Se $n = 0$ então $x^n = 1$.
- Caso Geral:
  - Se $n > 0$ e é par, então $x^n = (x^{n/2})^2$.
  - Se $n > 0$ e é ímpar, então $x^n = x(x^{(n-1)/2})^2$.

Note como no caso geral definimos a solução do caso maior em termos de casos menores.

Cálculo de Potências

Este algoritmo é mais eficiente do que o iterativo. Por que? Quantas chamadas recursivas o algoritmo pode fazer?

```
long pot(long x, long n){
    double aux;
    if(n == 0)
       return 1;

    else if(n%2 == 0){ //se n é par
      aux = pot(x, n/2);
      return aux * aux;
    }

    else{ //se n é impar
      aux = pot(x, (n-1)/2);
      return x*aux*aux;
    }
}
```


Cálculo de Potências

Este algoritmo é mais eficiente do que o iterativo. Por que? Quantas chamadas recursivas o algoritmo pode fazer?

```
long pot(long x, long n){
    double aux;
    if(n == 0)
       return 1;

    else if(n%2 == 0){ //se n é par
      aux = pot(x, n/2);
      return aux * aux;
    }

    else{ //se n é impar
      aux = pot(x, (n-1)/2);
      return x*aux*aux;
    }
}
```

(Instituto de Computação – Unicamp) MC-102 5 de Junho de 2012 10 / 39

Cálculo de Potências

Este algoritmo é mais eficiente do que o iterativo. Por que? Quantas chamadas recursivas o algoritmo pode fazer?

```
long pot(long x, long n){
    double aux;
    if(n == 0)
       return 1;

    else if(n%2 == 0){ //se n é par
      aux = pot(x, n/2);
      return aux * aux;
    }

    else{ //se n é impar
      aux = pot(x, (n-1)/2);
      return x*aux*aux;
    }
}
```

(Instituto de Computação – Unicamp) MC-102 5 de Junho de 2012 10 / 39

Cálculo de Potências - executando um exemplo na pilha: 2^4

- $2^4 \Leftarrow$ if par

Cálculo de Potências - executando um exemplo na pilha: 2^4

- $2^2 \Leftarrow$ if par
- ————————-
- $2^4 \Leftarrow$ if par

Cálculo de Potências - executando um exemplo na pilha: 2^4

- $2^1 \Leftarrow$ if impar
- ————————-
- $2^2 \Leftarrow$ if par
- ————————-
- $2^4 \Leftarrow$ if par

Cálculo de Potências - executando um exemplo na pilha: 2^4

- $2^0 \Leftarrow$ resolve: retorna 1
- ————————-
- $2^1 \Leftarrow$ if impar
- ————————-
- $2^2 \Leftarrow$ if par
- ————————-
- $2^4 \Leftarrow$ if par

Cálculo de Potências - executando um exemplo na pilha: 2^4

-
-
- $2^1 \Leftarrow$ if impar (aux=1 da solução anterior; resolve $(x * aux * aux = 2 * 1 * 1)$ e retorna 2)
- ————————-
- $2^2 \Leftarrow$ if par
- ————————-
- $2^4 \Leftarrow$ if par

Cálculo de Potências - executando um exemplo na pilha: 2^4

-
-
-
-
-
-
-
- $2^4 \Leftarrow$ if par (aux=4 da solução anterior; resolve $(aux * aux = 4 * 4)$ e retorna 16 que é a solução final!)

Cálculo de Potências - executando um exemplo na pilha: 2^4

-
-
-
-
- $2^2 \Leftarrow$ if par (aux=2 da solução anterior; resolve $(aux * aux = 2 * 2)$ e retorna 4)
- ————————-
- $2^4 \Leftarrow$ if par

Cálculo de Potências

- No algoritmo anterior a cada chamada recursiva, o valor de n é dividido por 2. Ou seja, a cada chamada recursiva, o valor de n decai para pelo menos a metade.
- Usando divisões inteiras faremos no máximo $\lceil(\log_2 n)\rceil + 1$ chamadas recursivas.
  - Lembram da curva log?
- Enquanto isso, o algoritmo iterativo executa o laço n vezes.

Vamos verificar isso! Alg. versão 1 para 2^9

```
long pot(long x, long n){
  if(n == 0)
    return 1;
  else
    return x*pot(x,n-1);
}
```

$2^9 \rightarrow 2^8 \rightarrow 2^7 \rightarrow 2^6 \rightarrow 2^5 \rightarrow 2^4 \rightarrow 2^3 \rightarrow 2^2 \rightarrow 2^1 \rightarrow 2^0$

- Fizemos 10 chamadas

Vamos verificar isso! Alg. versão 2 para 2^9

```
long pot(long x, long n){
    if(n == 0)
       return 1;
    else if(n%2 == 0){ //se n é par
      double aux = pot(x, n/2);
      return aux * aux;
    }
    else{ //se n é impar
      p = pot(x, (n-1)/2);
      return x*aux*aux;
    }
}
```

$2^9 \rightarrow 2^4 \rightarrow 2^2 \rightarrow 2^1 \rightarrow 2^0$

- Fizemos 5 chamadas $= \lceil(\log_2 9)\rceil + 1 = \lceil(2^k = 9)\rceil + 1$
- $\lceil(\log_2 n)\rceil + 1$

Recursão com várias chamadas

- Não há necessidade da função recursiva ter apenas uma chamada para si própria.
- A função pode fazer várias chamadas para si própria.
- A função pode ainda fazer chamadas recursivas indiretas. Neste caso a função 1, por exemplo, chama uma outra função 2 que por sua vez chama a função 1.
- Como assim? O algoritmo não vai "se perder" com tantas chamadas?
  - Não!!!!!!!!! Por que?
  - O que é usado para "controlar" a sequência de execução de um programa?
  - Pilha!!!!!!!

Recursão com várias chamadas

- Não há necessidade da função recursiva ter apenas uma chamada para si própria.
- A função pode fazer várias chamadas para si própria.
- A função pode ainda fazer chamadas recursivas indiretas. Neste caso a função 1, por exemplo, chama uma outra função 2 que por sua vez chama a função 1.
- Como assim? O algoritmo não vai "se perder" com tantas chamadas?
  - Não!!!!!!!!! Por que?
  - O que é usado para "controlar" a sequência de execução de um programa?
  - Pilha!!!!!!!

Recursão com várias chamadas

- Não há necessidade da função recursiva ter apenas uma chamada para si própria.
- A função pode fazer várias chamadas para si própria.
- A função pode ainda fazer chamadas recursivas indiretas. Neste caso a função 1, por exemplo, chama uma outra função 2 que por sua vez chama a função 1.
- Como assim? O algoritmo não vai "se perder" com tantas chamadas?
  - Não!!!!!!!!! Por que?
  - O que é usado para "controlar" a sequência de execução de um programa?
  - Pilha!!!!!!!!

Soma de um vetor

- Dado um vetor, vamos definir $S(i, n)$ como a soma de n elementos a partir da posição i.
- Com isso temos a seguinte definição recursiva para a soma dos elementos de um vetor:
  - Se $n = 1$ então $S(i, n) = v[i]$.
  - Se $n > 1$ então $S(i, n) = S(i, \lceil n/2 \rceil) + S(i + \lceil n/2 \rceil, \lfloor n/2 \rfloor)$.
- Observações
  - $n = \lceil n/2 \rceil + \lfloor n/2 \rfloor$.
  - Dada uma posição i e quantidade $x = \lceil n/2 \rceil$ de elementos incluindo x, a próxima posição a ser considerada será $(i + x)$.
- Para computarmos a soma de todos os elementos de um vetor com n elementos, devemos calcular $S(0, n)$.

Soma de um vetor

- Dado um vetor, vamos definir $S(i, n)$ como a soma de n elementos a partir da posição i.
- Com isso temos a seguinte definição recursiva para a soma dos elementos de um vetor:
  - Se $n = 1$ então $S(i, n) = v[i]$.
  - Se $n > 1$ então $S(i, n) = S(i, \lceil n/2 \rceil) + S(i + \lceil n/2 \rceil, \lfloor n/2 \rfloor)$.
- Observações
  - $n = \lceil n/2 \rceil + \lfloor n/2 \rfloor$.
  - Dada uma posição i e quantidade $x = \lceil n/2 \rceil$ de elementos incluindo x, a próxima posição a ser considerada será $(i + x)$.
- Para computarmos a soma de todos os elementos de um vetor com n elementos, devemos calcular $S(0, n)$.

(Instituto de Computação – Unicamp) MC-102 5 de Junho de 2012 22 / 39

Soma de um vetor

- Dado um vetor, vamos definir $S(i, n)$ como a soma de n elementos a partir da posição i.
- Com isso temos a seguinte definição recursiva para a soma dos elementos de um vetor:
  - Se $n = 1$ então $S(i, n) = v[i]$.
  - Se $n > 1$ então $S(i, n) = S(i, \lceil n/2 \rceil) + S(i + \lceil n/2 \rceil, \lfloor n/2 \rfloor)$.
- Observações
  - $n = \lceil n/2 \rceil + \lfloor n/2 \rfloor$.
  - Dada uma posição i e quantidade $x = \lceil n/2 \rceil$ de elementos incluindo x, a próxima posição a ser considerada será $(i + x)$.
- Para computarmos a soma de todos os elementos de um vetor com n elementos, devemos calcular $S(0, n)$.

(Instituto de Computação – Unicamp) MC-102 5 de Junho de 2012 22 / 39

Soma de um vetor

O código recursivo segue abaixo. Basta implementarmos funções para calcular o teto e o chão da divisão de dois números.

```
int soma(int v[], int i, int n){
  if(n == 1)
    return v[i];
  else{
    return soma(v, i, teto(n,2)) +
      soma(v, i+teto(n,2), chao(n,2));
  }
}
```


Soma de um vetor

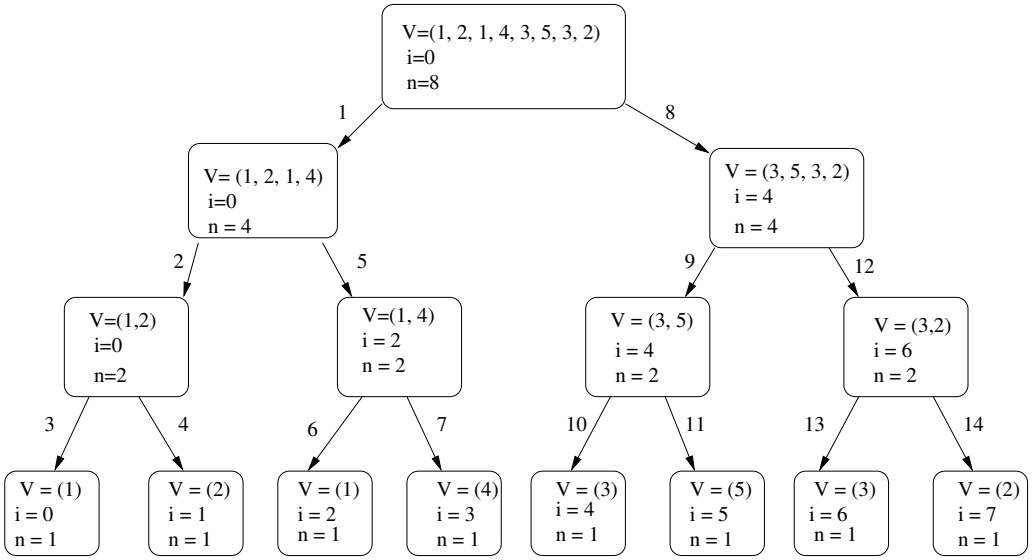
```
int teto(int numerador, int denominador){
  if(numerador % denominador == 0) //se divisão for inteira
    return (numerador/denominador);
  else
    return (numerador/denominador + 1);
}


int chao(int numerador, int denominador){
    return (numerador/denominador);
}
```


Soma de um vetor

- Abaixo temos um exemplo de execução da função para o vetor v=[1, 2, 1, 4, 3, 5, 3, 2].
- Há uma indicação da ordem em que ocorrem as sucessivas chamadas recursivas.
- Em cada balão é apresentado apenas a parte do vetor que está sendo considerada pela função naquele momento.
- A chamada da função deve ser: **somar(v, 0, 8)**.

Soma de um vetor



E na pilha?

- v(1,2,1,4); $i = 0$ e $n = 4$

E na pilha?

- v(1,2); $i = 0$ e $n = 2$ + !!
- v(1,2,1,4); $i = 0$ e $n = 4$ + !!

E na pilha?

- v(1); $i = 0$ e $n = 1 \Leftarrow$ retorna v[0]=1
- v(1,2); $i = 0$ e $n = 2$ + !!
- v(1,2,1,4); $i = 0$ e $n = 4$ + !!

E na pilha?

- v(1,2); $i = 0$ e $n = 2$ retorno v(1)= 1 + chamada para v(2)
- v(1,2,1,4); $i = 0$ e $n = 4$!!

E na pilha?

- v(2); $i = 1$ e $n = 1$ $\Leftarrow$ retorna v[1]=2
- v(1,2); $i = 0$ e $n = 2$ retorno v(1)= 1 + chamada para v(2)
- v(1,2,1,4); $i = 0$ e $n = 4$!!

E na pilha?

- v(1,2); $i = 0$ e $n = 2$ retorno v(1)= 1 + retorno v(2) = 2 $\rightarrow$ 1+2 = 3 retornado
- v(1,2,1,4); $i = 0$ e $n = 4$!!

E na pilha?

- retorno v(1,2) = 3 + chamada v(1,4) $i = 2$ e $n = 2$
- v(1,2,1,4); $i = 0$ e $n = 4$!!

E na pilha?

- v(1) $i = 2$ e $n = 1$ retorna v[2]=1
- retorno v(1,2) = 3 + v(1,4) $i = 2$ e $n = 2$
- v(1,2,1,4); $i = 0$ e $n = 4$!!

E na pilha?

- retorno v(1,2) = 3 + v(1,2,1,4); $i = 0$ e $n = 4$
- retorno v(1) = 1 + chamada v(4) !!

E na pilha?

- v(4) $i = 3$ e $n = 1$ retorna v[3]=4
- retorno v(1,2) = 3 + v(1,2,1,4); $i = 0$ e $n = 4$
- retorno v(1) = 1 + chamada v(4) !!

E na pilha?

- v(1,2)= 3 + v(1,4) = 1 + 4 = 5 → 3 + 5 = 8 → retorna 8
- v(1,2,1,4); $i = 0$ e $n = 4$!!

E na pilha?

- v(1,2,1,4); $i = 0$ e $n = 4$ 8

Exercício

- Defina de forma recursiva a busca binária.
- Escreva um algoritmo recursivo para a busca binária.