Algoritmos Recursivos

Rômulo César Silva

Unioeste

Abril de 2016

unioeste
Universidade Estadual do Oeste do Paraná

Sumário

Caracterização de Problemas

Problema

Um **problema** pode ser descrito através das informações essenciais que o caracterizam, geralmente designados como sendo os seus parâmetros de entrada.

Instância

Uma **instância** de um problema refere-se a um caso particular do problema, com dados específicos.

Exemplo: O problema de calcular a média aritmética de N números inteiros, tem como parâmetros de entrada os N números.
Uma instância desse problema é o cálculo da média aritmética dos números 2, 7, 11, 8 e 6. Outra instância seria para os números 25, 12, 17, 5, 9, 4, 2, 50 e 23.

Problemas

Estrutura Recursiva

Muitos problemas tem a seguinte propriedade: cada instância do problema contém uma instância menor do mesmo problema. Esse tipo de problema é dito ter **estrutura recursiva**.

Estrutura geral de um algoritmo recursivo

Algoritmo Recursivo

1: **if** instância é pequena **then**
2: resolva-a diretamente
3: **else**
4: reduza-a a uma instância menor do mesmo problema
5: resolva o problema para a instância menor recursivamente
6: use o resultado da instância menor para encontrar a solução da instância maior
7: **end if**

Algoritmos Recursivos

- **instância pequena** significa que o problema não pode mais ser subdividido ou sua solução são definidas explicitamente
- as instâncias pequenas são chamados **casos base** em analogia à técnica matemática de demonstração por indução

Algoritmos Recursivos

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

- $n == 0$ é o caso base
- calcular o fatorial de $n - 1$ é exatamente ter uma instância menor do problema. Isto é, $n - 1$ está mais próximo do caso base. O cálculo do fatorial de $n - 1$ corresponde à chamada recursiva.
- o resultado do fatorial de $n - 1$ será usado no cáculo do fatorial de n.
- em seguida a partir da multiplicação entre n e $fatorial(n-1)$ obtém-se o fatorial de n.

Algoritmos Recursivos

```
int fibonacci(int n) {
   if(n == 0)
     return 0;
   if(n == 1)
     return 1;
   else
     return fibonacci(n-1) + fibonacci(n-2);
   }
```

- $n == 0$ e $n == 1$ são casos base
- 2 chamadas recursivas recursivas são necessárias para calcular *fibonacci*(n). Isto tipo de situação é chamado de *recursão múltipla*.

Algoritmos Recursivos

```
// função para retornar o maior elem.
// de um vetor de n inteiros
int maiorElem(int v[], int n) {
   if (n == 1)
     return v[0];
   else {
     int x = maiorElem(v,n-1);
     if(x > v[n-1])
       return x;
     else
       return v[n-1];
}
```

Algoritmos Recursivos

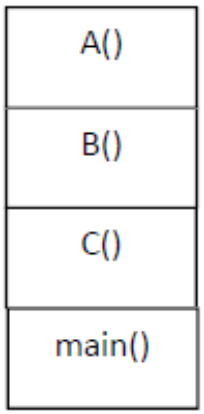
Pilha de chamadas

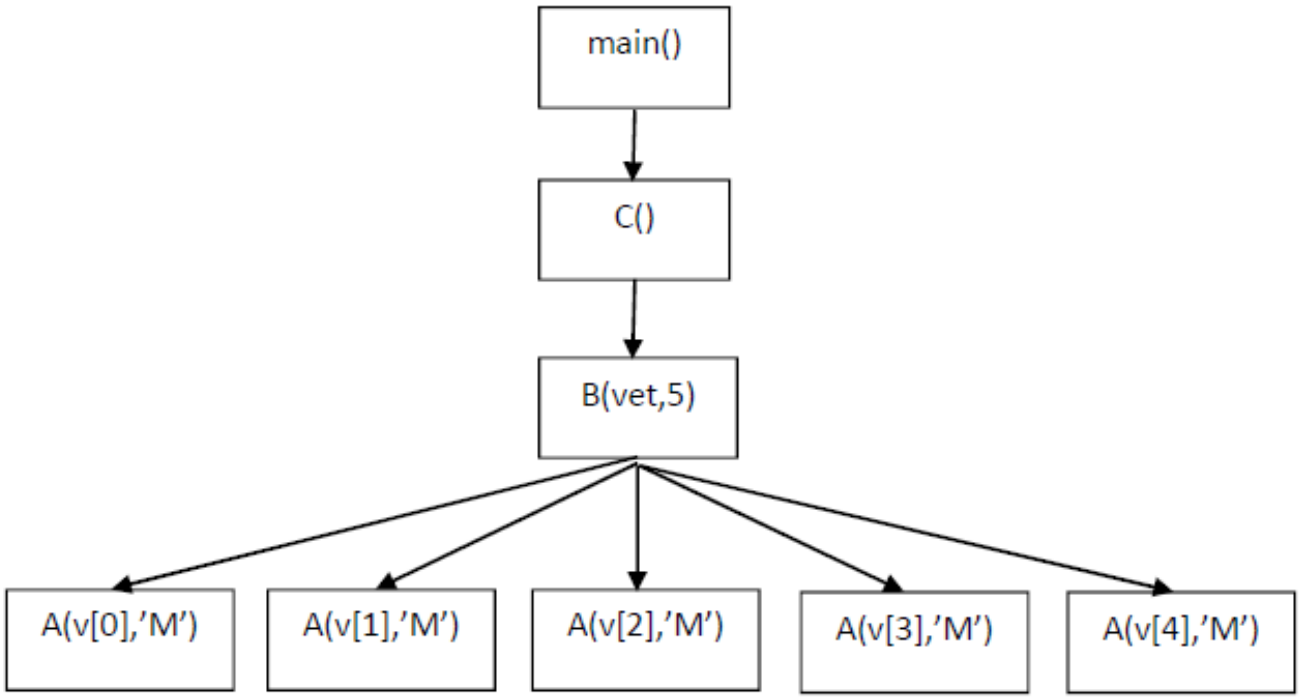
- Existe uma pilha de chamadas associada a cada invocação de procedimento ou função em um programa
- A pilha contém os parâmetros passados para o procedimento ou função invocada e o ponto de retorno
- No caso de uma função recursiva, a pilha irá conter várias vezes informação relacionada à mesma função, sendo uma para cada chamada recursiva.

Pilha de chamadas - exemplo

```
int A(int param1, char param2) {...}
int B(int v[], int n) {
...
for(i = 0; i < n; i++)
   r = r + A(v[i],'M'); // chama função A
return r;
}
int C() {
int vet[5];
...
return B(vet,5); // chama função B
}
int main() { // programa principal
   int x = C();
}
```

Pilha de Chamadas - exemplo

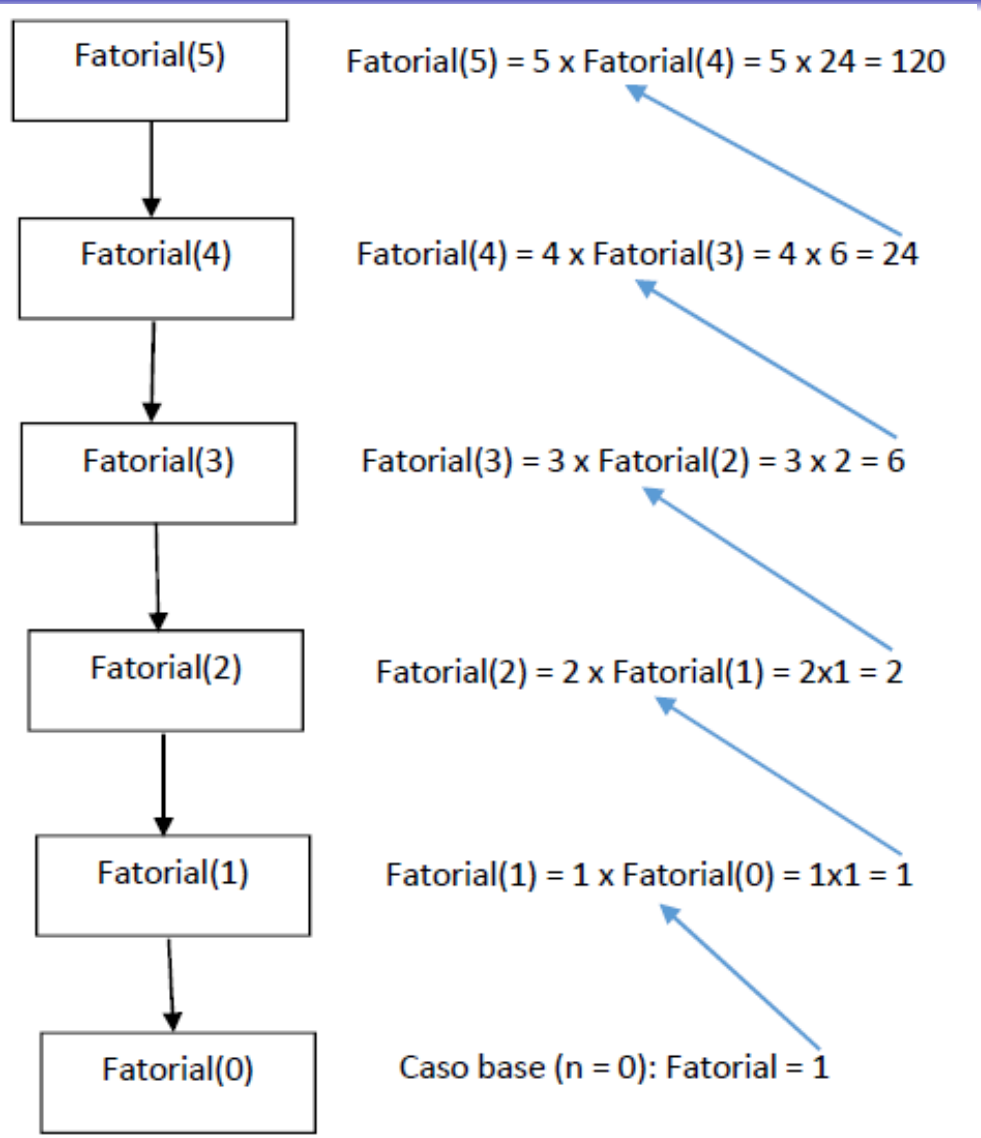




Pilha de Chamadas de algoritmo recursivo - exemplo

```
int fatorial(int n) {
  if(n == 0)
    return 1;
  else
    return n*fatorial(n-1);
}
void main(){
  int fat = fatorial(5);
}
```

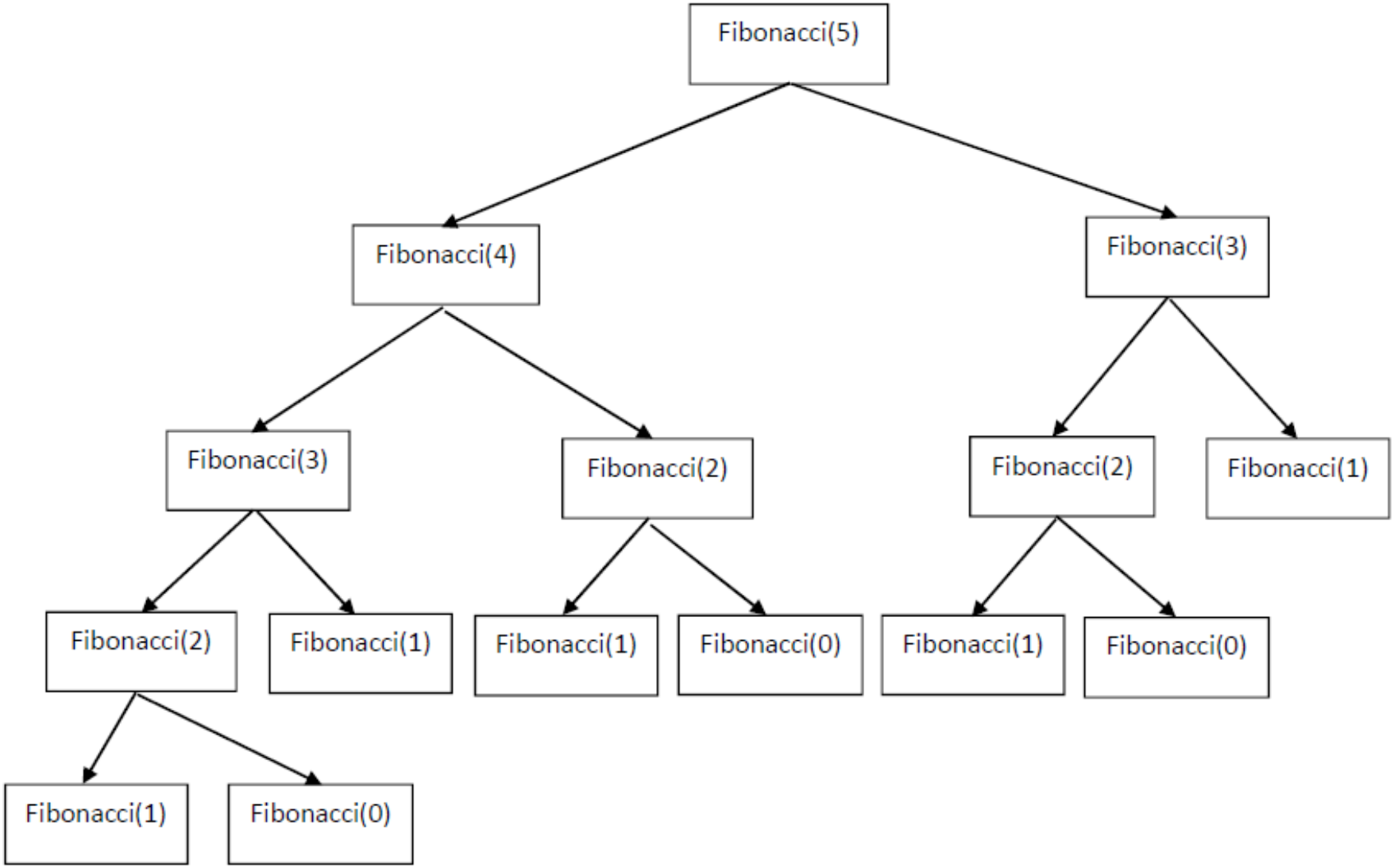
Pilha de Chamadas - fatorial



Pilha de Chamadas de algoritmo recursivo - exemplo

```
 int fibonacci(int n) {
    if(n == 0)
      return 0;
    if(n == 1)
      return 1;
    else
      return fibonacci(n-1) + fibonacci(n-2);
    }
void main(){
  int fat = fatorial(5);
}
```

Pilha de Chamadas - Fibonacci



Tipos de Recursividade

- **Direta**: a função ou procedimento tem uma chamada explícita a si mesmo. Exs: as funções para *fatorial* e *fibonacci* anteriores.
- **Indireta** (ou **Mútua**): duas ou mais funções/procedimento dependem mutuamente um do(s) outro(s). Exemplo: A chama B e B chama A.
- **em Cauda**: a chamada recursiva é a última instrução a ser executada, isto é, sem operações pendentes.

Exemplo de Recursividade Indireta

```
// Retorna 1 se n é par ou 0 se n é ímpar
int par(int n) {
    if(n == 0)
       return 1;
    if(n > 0)
       return impar(n-1);
    else
       return par(-n);
    }
// Retorna 1 se n é ímpar ou 0 se n é par
int impar(int n) {
    if(n == 0)
       return 0;
    if(n > 0)
       return par(n-1);
    else
       return impar(-n);
    }
```

Exemplo de Recursividade em Cauda

```
// versão com recursão em cauda
int fatorial(int n) {
     return fatorial_cauda(n,1);
    }

int fatorial_cauda(int n, int acumulador) {
    if(n == 0)
       return acumulador;
    else // chamada recursiva é a última instrução!!
       return fatorial_cauda(n-1,n*acumulador);
    }
```

Algoritmos Recursivos

Observações

- soluções que utilizam algoritmos recursivos geralmente são pequenas e elegantes. Porém, nem sempre eficientes computacionalmente, o que depende da própria linguagem de programação.
- há linguagens de programação preparadas especificamente para lidar com recursão, sendo este seu processo básico de repetição. (Ex.: Prolog e Scheme)
- se a instância passada inicialmente para o algoritmo é muito grande, pode ocorrer estouro da pilha
- mesmo quando a eficiência seja considerada um aspecto negativo, o algoritmo recursivo pode ser mais fácil de ser desenvolvido, e depois ser convertido em um algoritmo iterativo.

Algoritmos Recursivos

Observações

- Existem diferentes técnicas de projeto de algoritmos. Serão estudadas na disciplina *Projeto e Análise de Algoritmos* do 3º ano.
- A recursão é um mecanismo básico utilizado na técnica de **projeto de algoritmos por indução**:
  - similar a uma demonstração matemática por indução
  - caso base da indução corresponde exatamente ao caso base da recursão
  - a hipótese de indução corresponde à chamada recursiva
  - o passo da indução corresponde à obtenção da solução para a instância maior usando o retorno da chamada recursiva
- a demonstração da correção do algoritmo é direta do próprio mecanismo de indução.

Algoritmos Recursivos

Observações

- A recursão pode ser eliminada substituindo-a por uma pilha explícita. A estrutura de dados *Pilha* será vista mais adiante na disciplina

Algoritmos Recursivos

ATENÇÃO!!

Desenvolver corretamente algoritmos recursivos requer **P R Á T I C A!**

Isso signfica (sugestões!!):

1. **Resolva vários exercícios**, procurando desenvolver os algoritmos primeiro só no papel. Lembre-se: na prova você terá somente o papel e a caneta a sua frente!
2. **Depois programe a solução em alguma linguagem de programação**. Nesta disciplina, usaremos na maioria das vezes a linguagem C.
3. **Teste sua solução**. Dica: não esqueça de testar as situações que representem casos ou condições específicas no algoritmo.
4. Somente olhe a solução do colega ou na Internet após tentar a própria. Devemos estar preparados para lidar com um problema que ninguém ainda resolveu...

Bibliografia I

[Feofiloff 2009] Paulo Feofiloff.
Algoritmos em linguagem C. Elsivier, Rio de Janeiro, 2009.