Recursão e algoritmos recursivos

"Ao tentar resolver o problema,
encontrei obstáculos dentro de obstáculos.
Por isso, adotei uma solução recursiva."
— aluno S.Y.

"To understand recursion,
we must first understand recursion."
— anônimo

"Para fazer um procedimento recursivo
é preciso ter fé."
— prof. Siang Wun Song

Muitos problemas têm a seguinte propriedade: cada instância do problema contém uma instância menor do mesmo problema. Dizemos que esses problemas têm *estrutura recursiva*. Para resolver uma instância de um problema desse tipo, podemos aplicar o seguinte método:

- se a instância em questão for pequena,
  resolva-a diretamente (use força bruta se necessário);
- senão
  reduza-a a uma instância menor do mesmo problema,
  aplique o método à instância menor,
  volte à instância original.

O programador só precisa mostrar como obter uma solução da instância original a partir de uma solução da instância menor; o computador faz o resto. A aplicação desse método produz um algoritmo *recursivo*.

Um exemplo

Para ilustrar o conceito de algoritmo recursivo, considere o seguinte problema: *Determinar o valor de um elemento máximo de um vetor* `v[0..n-1]`. (O problema já foi mencionado num dos exercícios na página *Vetores*.)

Observe que o problema só faz sentido se o vetor não é vazio, ou seja, se `n ≥ 1`. Eis uma função recursiva que resolve o problema:

```
// Ao receber v e n >= 1, a função devolve
// o valor de um elemento máximo do
// vetor v[0..n-1].

int
maximo (int n, int v[])
{
   if (n == 1)
      return v[0];
   else {
      int x;
      x = maximo (n-1, v);
      // x é o máximo de v[0..n-2]
      if (x > v[n-1]) return x;
      else return v[n-1];
   }
}
```

A análise da correção da função tem a forma de uma prova por indução. Para começar, considere o caso em que `n` vale `1`. Nesse caso, a resposta que a função devolve, `v[0]`, está correta pois `v[0]` é o único elemento relevante do vetor. Agora considere o caso em que `n` é maior que `1`. Nesse caso, o vetor tem duas partes não vazias:

`v[0..n-2]` e `v[n-1]`.

Podemos supor que a função resolve corretamente a instância `v[0..n-2]` do problema, pois essa instância é menor que a original. Portanto, depois da linha "`x = maximo (n-1, v)`", podemos supor que `x` é um máximo correto de `v[0..n-2]`. Logo, a solução da instância original é o maior dos números `x` e `v[n-1]`. E é justamente esse o número que a função calcula e devolve. Isso conclui a prova de que a função `maximo` está correta.

Para que uma função recursiva seja compreensível, é essencial que o autor diga *o que* a função faz. Isso deve ser dito de maneira explicita e completa, como fiz acima no comentário que precede o código da função.

Como o computador executa uma função recursiva? Embora relevante e importante, essa pergunta será ignorada por enquanto. (Mas você pode ver o apêndice *A pilha de execução de um programa* do capítulo *Pilhas*.) Vamos nos limitar a mostra um exemplo de execução. Se `v[0..3]` é o vetor `77 88 66 99` então teremos a seguinte sequência de chamadas da função:

```
maximo(4,v)
│  maximo(3,v)
│  │  maximo(2,v)
│  │  │  maximo(1,v) ... devolve 77
│  │  └──────────── ... devolve 88
│  └─────────────── ... devolve 88
└──────────────────── ... devolve 99
```

Desempenho. Algumas pessoas acreditam que funções recursivas são inerentemente lentas, mas isso não passa de lenda. Talvez a lenda tenha origem em certos usos descuidadas da recursão, como em alguns dos exercícios abaixo. (Nem tudo são flores, entretanto: o espaço de memória que uma função recursiva consome para "rascunho" pode ser grande.)

Exercícios 1

1. Que tipos de problemas podem ser resolvidos por um algoritmo recursivo?
2. Considere a função `maximo` acima. Faz sentido trocar "`return v[0]`" por "`return 0`"? Faz sentido trocar "`return v[0]`" por "`return INT_MIN`"? Faz sentido trocar "`x > v[n-1]`" por "`x >= v[n-1]`"?
3. Considere a função `maximo` acima. Se o vetor `v[0..n-1]` tiver dois os mais elementos máximos, qual deles a função devolve?
4. Considere a função `maximo` acima. Quantas comparações envolvendo elementos do vetor sua função faz no pior caso?
5. Verifique que a seguinte maneira de escrever a função `maximo` é exatamente equivalente à dada acima:

```
int maximo (int n, int v[]) {
   int x;
   if (n == 1) x = v[0];
   else {
      x = maximo (n-1, v);
      if (x < v[n-1]) x = v[n-1];
   }
   return x;
}
```

6. Verifique que a seguinte maneira de escrever a função `maximo` é exatamente equivalente à dada acima:

```
int maximo (int n, int v[]) {
   if (n == 1) return v[0];
   int x = maximo (n-1, v);
   if (x > v[n-1]) return x;
   else return v[n-1];
}
```

7. Critique a seguinte função recursiva que promete encontrar o valor de um elemento máximo de `v[0..n-1]`:

```
int maxim1 (int n, int v[]) {
   if (n == 1) return v[0];
   if (n == 2) {
      if (v[0] < v[1]) return v[1];
      else return v[0];
   }
   int x = maxim1 (n-1, v);
   if (x < v[n-1]) return v[n-1];
```

```
   else return x;
}
```

8. Critique a seguinte função recursiva que promete encontrar o valor de um elemento máximo de `v[0..n-1]`:

```
int maxim2 (int n, int v[]) {
   if (n == 1) return v[0];
   if (maxim2 (n-1, v) < v[n-1])
      return v[n-1];
   else
      return maxim2 (n-1, v);
}
```

9. PROGRAMA DE TESTE. Escreva um pequeno programa para testar a função recursiva `maximo` dada acima. O seu programa deve gerar um vetor aleatório e encontrar um elemento máximo desse vetor. Acrescente ao seu programa uma função que *confira a resposta* dada por `maximo`. [Veja uma solução]

10. Escreva uma função recursiva `maxmin` que calcule o valor de um elemento máximo e o valor de um elemento mínimo de um vetor `v[0..n-1]`. Quantas comparações envolvendo os elementos do vetor sua função faz?

Outra solução

A função `maximo` discutida acima reduz a instância `v[0..n-1]` do problema à instância `v[0..n-2]`. Podemos escrever uma função que reduza `v[0..n-1]` a `v[1..n-1]`, ou seja, "empurre pra direita" o início do vetor:

```
// Ao receber v e n >= 1, esta função devolve
// o valor de um elemento máximo do vetor
// v[0..n-1].

int
maximo2 (int n, int v[])
{
   return max (0, n, v);
}
```

A função `maximo2` é apenas uma "embalagem" ou "invólucro" (= *wrapper-function*); o serviço propriamente dito é executado pela função recursiva `max`:

```
// Recebe v e i < n e devolve o valor de
// um elemento máximo do vetor v[i..n-1].

int
max (int i, int n, int v[])
{
   if (i == n-1) return v[i];
   else {
      int x;
      x = max (i + 1, n, v);
      if (x > v[i]) return x;
      else return v[i];
   }
}
```

A função `max` resolve um problema mais geral que o original (veja o parâmetro `i`). A necessidade de generalizar o problema original ocorre com frequência no projeto de algoritmos recursivos.

A título de curiosidade, eis outra maneira, talvez surpreendente, de aplicar recursão ao segmento `v[1..n-1]`. Ela usa aritmética de endereços:

```
int maximo2b (int n, int v[]) {
   if (n == 1) return v[0];
   int x = maximo2b (n - 1, v + 1);
   if (x > v[0]) return x;
   return v[0];
}
```

Exercícios 2

1. Considere a função `maximo2` acima. Se o vetor `v[0..n-1]` tiver dois os mais elementos máximos, qual deles a função devolve? E se trocarmos `if (x > v[i])` por `if (x >= v[i])`?

2. A seguinte função recursiva pretende encontrar o valor de um elemento máximo de `v[p..u]`, supondo `p ≤ u`. (O índice `p` aponta o primeiro elemento do vetor e o índice `u` aponta o último.) A função está correta? Suponha que `p` e `u` valem `0` e `6` respectivamente e mostre a sequência, devidamente indentada, de chamadas de `maxx`.

```
int maxx (int p, int u, int v[]) {
   if (p == u) return v[u];
   else {
      int m, x, y;
      m = (p + u)/2; // p ≤ m < u
      x = maxx (p, m, v);
      y = maxx (m+1, u, v);
      if (x >= y) return x;
      else return y;
   }
}
```

Exercícios 3

1. Escreva uma função recursiva que calcule a soma dos elementos positivos do vetor de inteiros `v[0..n-1]`. O problema faz sentido quando `n` é igual a `0`? Quanto deve valer a soma nesse caso?

2. Critique a documentação do seguinte código:

```
// Recebe um vetor de tamanho n e devolve
// a soma dos elementos do vetor.
int soma (int v[], int n) {
   return sss (v, n+1);
}
int sss (int v[], int n) {
   if (n == 1) return 0;
   return sss (v, n-1) + v[n-1];
}
```

3. Diga *o que* a função abaixo faz. Para quais valores dos parâmetros sua resposta está correta?

```
int ttt (int x[], int n) {
   if (n == 0) return 0;
   if (x[n] > 100)  return x[n] + ttt (x, n-1);
   else  return ttt (x, n-1);
}
```

4. Escreva uma função recursiva que calcule o produto dos elementos estritamente positivos de um vetor de inteiros `v[0..n-1]`. O problema faz sentido quando todos os elementos do vetor são nulos? O problema faz sentido quando `n` vale `0`? Quanto deve valer o produto nesses casos?

5. Escreva uma função recursiva que calcule a soma dos dígitos decimais de um inteiro estritamente positivo `n`. A soma dos dígitos de `132`, por exemplo, é `6`.

6. Escreva uma função recursiva que calcule o piso do logaritmo de N na base 2. (Veja uma versão não-recursiva do exercício.)

Exercícios 4

1. Qual o valor de `X (4)` se `X` é dada pelo seguinte código? [Veja uma solução.]

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

2. Qual é o valor de `f (1,10)`? Escreva uma função equivalente que seja mais simples.

```
double f (double x, double y) {
   if (x >= y) return (x + y)/2;
   else return f (f (x+2, y-1), f (x+1, y-2));
}
```

3. Qual o resultado da execução do seguinte programa?

```
int ff (int n) {
   if (n == 1) return 1;
   if (n % 2 == 0) return ff (n/2);
   return ff ((n-1)/2) + ff ((n+1)/2);
}
int main (void) {
   printf ("%d", ff(7));
   return EXIT_SUCCESS;
}
```

4. Execute `fusc (7,0)`. Mostre a sequência, devidamente indentada, das chamadas a `fusc`.

```
int fusc (int n, int profund) {
  for (int i = 0; i < profund; ++i)
     printf ("  ");
  printf ("fusc(%d,%d)\n", n, profund);
  if (n = 1)
     return 1;
  if (n % 2 == 0)
     return fusc (n/2, profund+1);
  return fusc ((n-1)/2, profund+1) +
         fusc ((n+1)/2, profund+1);
}
```

5. IMPORTANTE. Critique a seguinte função recursiva:

```
int XX (int n) {
   if (n == 0) return 0;
   else return XX (n/3+1) + n;
}
```

6. FIBONACCI. A função de Fibonacci é definida assim: `F(0) = 0`, `F(1) = 1` e `F(n) = F(n-1) + F(n-2)` para `n > 1`. Descreva a função `F` em linguagem C. Faça uma versão recursiva e uma iterativa.

7. Seja `F` a versão recursiva da função de Fibonacci. O cálculo do valor da expressão `F(3)` provocará a seguinte sequência de invocações da função:

```
F(3)
  F(2)
    F(1)
    F(0)
  F(1)
```

   Qual a sequência de invocações da função durante o cálculo de `F(5)`?

8. EUCLIDES. A seguinte função calcula o maior divisor comum dos inteiros estritamente positivos `m` e `n`. Escreva uma função recursiva equivalente.

```
int Euclides (int m, int n) {
   int r;
   do {
      r = m % n;
      m = n;
      n = r;
   } while (r != 0);
   return m;
}
```

9. EXPONENCIAÇÃO. Escreva uma função recursiva eficiente que receba inteiros estritamente positivos k e n e calcule k^n. (Suponha que k^n cabe em um `int`.) Quantas multiplicações sua função executa aproximadamente?

10. RÉGUA INGLESA [Sedgewick 5.8, Roberts 5.11] Escreva uma função recursiva que imprima uma *régua de ordem* n no intervalo $[0..2^n]$. O "traço" no ponto médio da régua deve ter comprimento n, os traços nos pontos médios dos subintervalos superior e inferior devem ter comprimento $n-1$, e assim por diante. A figura mostra uma régua de ordem 4.

```
.
. -
. --
. -
. ---
. -
. --
. -
. ----
. -
. --
. -
. ---
. -
. --
. -
.
```

Perguntas e respostas

- PERGUNTA: Num dos exercícios acima aparece a expressão `if (n == 1 || n == 2)`. Eu não deveria escrever `if ((n == 1) || (n == 2))`?

  RESPOSTA: Não. Os parênteses adicionais são redundantes porque o operador `==` tem precedência sobre `||`.

Veja o verbete *Recursion* na Wikipedia

Veja bons exemplos de recursão no capítulo sobre algoritmos de enumeração

---

Atualizado em 2018-05-21
https://www.ime.usp.br/~pf/algoritmos/
Paulo Feofiloff
DCC-IME-USP