EP1 - Recursão, recorrências e séries

Fábio Nakano e Luciano Digiampietri

Objetivos Específicos

Treinar codificação e depuração de métodos recursivos
Comparar o resultado de fórmulas de recorrência e fórmulas fechadas.

Introdução

Em computação, método recursivo é o método que direta ou indiretamente invoca a si mesmo. Em teoria, este método poderia ficar se invocando indefinidamente, parecendo ao usuário que o programa travou. Em realidade, ele está processando algo, não responde, nem termina.

O computador (ou sua abstração, a Máquina Virtual Java) controla todas as invocações de método, de forma que quando um método retorna, o programa volte ao ponto imediatamente posterior a sua invocação e com o escopo ajustado corretamente.

A principal estrutura de controle de invocação é chamada pilha de execução. Uma pilha (em computação) funciona como uma pilha no mundo real, ou seja, vamos amontoando objetos (por exemplo papéis) um sobre o outro. Quando quisermos desempilhar os papeis, o que conseguimos desempilhar é o que empilhamos por último. Dizemos que é uma estrutura último que entra, primeiro que sai (inglês LIFO - last in first out).

Por exemplo, no código abaixo:

```
1: class Exemplo {
2:         public static void main (String[] args) {
3:                 System.out.println (teste(5));
4:         }
5:         static int teste (int t) {
6:                 System.out.println ();
7:                 return t;
8:         }
9: }
```

Quando o programa é executado, o primeiro método invocado é main, logo ele é o primeiro a entrar na pilha de execução. Na linha 3 o método println é invocado, e, antes que este retorne, o método teste é invocado. Se imprimíssemos os nomes dos métodos na pilha de execução entre as linhas 5 e 6, obteríamos “teste”, “println” e “main”.

Fazendo de teste um método recursivo, obtemos:

```
1: class Exemplo {
2:         public static void main (String[] args) {
3:                 System.out.println (teste(5));
4:         }
5:         static int teste (int t) {
6:                 System.out.println ();
```

```
6a:                 teste (t);
7:                  return t;
8:          }
9: }
```

A pilha de execução, como qualquer outro objeto, tem tamanho limitado (a quantidade de memória do computador é limitada). A cada invocação de teste ela ganha um elemento, logo, o método teste vai ser invocado até que a pilha “transborde”. Quando isso ocorrer, será lançada a exceção *stack overflow*, e a máquina virtual vai abortar o programa.

Métodos recursivos necessitam de uma condição de parada. Quando essa condição for verdadeira, a invocação do método não é feita, interrompendo a recursão. Por exemplo no cálculo do produto por somas sucessivas:

```
1: class Produto {
2:          public static void main (String[] args) {
3:                  System.out.println (produto(5,7));
4:          }
5:          public static int produto (int m, int n) {
6:                  if (n==0) return 0;   // condição de parada
7:                  return m+produto(m,n-1);
8:          }
9: }
```

A primeira invocação de produto é feita com os parâmetros 5 e 7. A segunda é feita com os parâmetros 5 e 7-1=6. A terceira é feita com os parâmetros 5 e 7-2=5 e assim por diante, até que o parâmetro n valha zero. Neste caso não há nova invocação e o método retorna. Neste exemplo, as invocações retornam sucessivamente, até que se retorne para main - quando é impresso o valor final.

Se quiser ver como funciona, acrescente “prints”:

```
1: class Produto {
2:          public static void main (String[] args) {
3:                  System.out.println (produto(5,7));
4:          }
5:          public static int produto (int m, int n) {
6:                  if (n==0) return 0;   // condição de parada
7a:                 System.out.println (“invocando”);
7b:                 int r=produto(m,n-1);
7c:                 System.out.println (“retornando”);
7d:                 return m+r;
8:          }
9: }
```

Java permite visualizar a pilha de execução na forma de um arranjo (array, vetor) de objetos da classe StackTraceElement. Esse array é extraído do programa através do método getStackTrace() [1]. Desta forma é possível verificar quantas invocações recursivas foram feitas.

Métodos recursivos têm relação estreita com fórmulas de recorrência. Estas fórmulas descrevem um termo de uma sequência em função dos termos anteriores. O método para calcular o produto corresponde à seguinte recorrência:

$$P(a,n)=\begin{cases}0 & n=0\\ a+P(a,n-1) & n>0\end{cases}\quad \text{(I)}$$

Já conhecemos várias fórmulas como essa:

Fatorial

$$F(n)=\begin{cases}1 & n=0\\ n*F(n-1) & n>0\end{cases}\quad \text{(II)}$$

Fibonacci

(a)

$$F(n)=\begin{cases}0 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

Há outras recorrências que serão importantes para nós pois são a função de complexidade de tempo de algoritmos conhecidos, por exemplo:

$$F(n)=\begin{cases}1 & n=0\\ a*F(n-1)+1 & n>0\end{cases}\quad \text{(b)}$$

a fórmula geral é $\sum_{i=0}^{n} a^i$ que

para $a=2$ é equivalente a $F(n)=a^{n+1}-1$

$$F(n)=\begin{cases}1 & n=0\\ a*F(\frac{n}{2})+1 & n>0\end{cases}\quad \text{(c)}$$

a fórmula fechada equivalente para $a=1 e n=2^k$ é $F(n)=k+2$. Note que $\log_2(n)=k$.

$$F(n)=\begin{cases}1 & n=0\\ a*F(\frac{n}{2})+n & n>0\end{cases}\quad \text{(d)}$$

a fórmula fechada equivalente para $a=1 e n=2^k$ é $F(n)=2*n$
extra: veja o que ocorre para $a=2$ ou para $a=4$

Apenas a título de curiosidade, a sequência gerada pelo método de Newton-Raphson também pode ser escrita como uma recorrência:

$$x_n=\begin{cases}x_0 & n=0\\ x_{n-1}-\frac{f(x_{n-1})}{f'(x_{n-1})} & n>0\end{cases}$$ ressaltando que a condição de parada é: $err=\left|\frac{f(x_{n-1})}{f'(x_{n-1})}\right|\leq EPSILON$

Especificação

O método recursivo se chama **function** e tem a assinatura definida na classe Recursivo, dentro do pacote base. Essa classe contém métodos para consulta da pilha de execução. Todas as soluções têm que ser classes derivadas de Recursivo. São dados como exemplo os casos (I) e (II), você deve implementar os casos (a), (b), (c) e (d).

A solução do caso (I) está no pacote produto e tem o nome Produto. Nessa classe são implementados todos os métodos necessários para a instanciação da classe. A solução recursiva deve ser codificada no método **function** e ele DEVE invocar storeStack() na condição de parada, caso haja mais de uma condição de parada, todas devem invocar o método - isso é essencial para a correção do EP. A solução do caso (II) está no pacote fatorial, tem o nome Fatorial e segue o mesmo padrão de Produto.

DemoRecursao.java contém um exemplo de uso do exemplo do produto. Ele imprime na tela:

```
nakano@nakano-Desk:/media/dados/icc2.2/ep1$ java DemoRecursao
20
11
Pilha de execucao:
0:getStackTrace
1:storeStack
2:function
3:function
4:function
5:function
6:function
7:function
8:function
9:function
10:function
11:function
12:function
13:main
```

O primeiro número é o resultado da operação, o segundo número é o máximo número de invocações recursivas e em seguida, a listagem dos nomes dos métodos na pilha de execução.

TestaRecursao.java é um exemplo do corretor automático que será utilizado pelos professores para corrigir o EP.

A solução para o ítem (a) deve estar dentro do pacote a e ter o nome SolA.java, a solução para o ítem (b) deve estar dentro do pacote b e ter o nome SolB.java, a solução para o ítem (c) deve estar dentro do pacote c e ter o nome SolC.java, a solução para o ítem (d) deve estar dentro do pacote d e ter o nome SolD.java.

Exemplos de teste.

Não são apresentados exemplos em tabelas pois na introdução são apresentadas as fórmulas fechadas equivalentes. Você deve usá-las para checar a sua solução.

Avaliação

As soluções serão avaliadas pela clareza e formatação do código (indentação, comentários, etc) e sua capacidade de resolver testes diferentes dos apresentados nos exemplos e que não serão disponibilizados a priori.

Se a solução causar erro ou exceção em determinado teste, receberá zero nesse teste.

Receberão zero as soluções que

- não compilarem;
- estiverem fora da especificação (inclusive formato de entrega);
- tiverem modificado métodos pré-existentes;
- forem copiadas (detecção por sistema automático).

É permitido criar novos métodos dentro das classes que você escrever, lembrando que a resposta tem que ser calculada com uma única invocação de **function**().

Entrega

Os pacotes a, b, c e d, contendo as classes SolA.java, SolB.java, SolC.java e SolD.java devem ser compactados juntos em um único arquivo com formato zip ou rar. O nome de arquivo compactado deve ser seu número USP, sem prefixos ou sufixos. Por exemplo se seu número é 1234567, então seu arquivo deve ser 1234567.zip.

Os arquivos java devem ter o cabeçalho:

```
/*************************************************************/
/** ACH 2002 - Introducao a Ciencia da Computacao II        **/
/** EACH-USP - Segundo Semestre de 2012                     **/
/**                                                         **/
/** <turma> - <nome do professor>                           **/
/**                                                         **/
/** Primeiro Exercicio-Programa                             **/
/**                                                         **/
/** <nome do aluno> <numero USP>                            **/
/**                                                         **/
/*************************************************************/
```

A entrega deve ser feita através do CoL até 22.09.2012. Não serão aceitas entregas após este dia.

Dicas

Para evitar variações na indentação em função da relação entre espaços e TABs, recomenda-se que toda indentação seja feita com TABs e que se for possível no editor de texto escolhido, estes sejam ajustados para exibir recuo equivalente a 4 espaços.

Consulte a API do Java (http://docs.oracle.com/javase/1.6.0/docs/api/).