Questão 4

Questão 4.[4]

Uma função é denominada recursiva quando ela é chamada novamente dentro de seu corpo. Implementações recursivas tendem a ser menos eficientes, porém facilitam a codificação e seu entendimento.

CELES, W.; CERQUEIRA, R.; RANGEL, J. L. *Introdução a estrutura de dados*. Rio de Janeiro, 2004 (com adaptações).

Considere a função recursiva f(), a qual foi escrita em linguagem C:

```
1 int f( int v[], int n){
2     if(n == 0)
3        return 0;
4     else{
5        int s;
6        s = f(v, n-1);
7        if( v[n-1] > 0 ) s = s + v[n-1];
8            return s;
9     }
10 }
```

Suponha que a função f() é acionada com os seguintes parâmetros de entrada:

```
f({2,-4,7,0,-1,4},6);
```

Nesse caso, o valor de retorno da função f() será

A. 8.

B. 10.

C. 13.

D. 15.

E. 18.

1. Introdução teórica

Recursividade

Em um programa computacional, quando temos uma função (ou procedimento) chamando a si mesma, chamamos esse mecanismo de recursividade. À primeira vista, o fato de uma função chamar a si mesma pode parecer estranho e talvez até mesmo errado: se uma função chama a si mesma de forma contínua, quando o processo irá parar?

Ao criar uma função recursiva, o programador deve evitar situações em que o programa nunca termine, com uma função chamando a si mesma sem nunca se estabelecer

[4]Questão 16 – Enade 2014.

um critério de parada. Dessa forma, deve existir uma condição na qual ocorra recursividade e outra condição na qual a função retorne algum valor.

Em um programa bem comportado, sempre deve haver um momento em que o processo de recursividade é interrompido. A função retorna a um valor que vai ser utilizado em cada chamada anterior da função. Normalmente, queremos trabalhar com programas que devem levar um tempo finito para essa execução e, preferencialmente, o menor tempo possível.

Outro cuidado que devemos tomar ao utilizarmos recursividade, mesmo quando não temos uma situação com infinitas chamadas, é que, se o número de chamadas for muito grande, pode ocorrer uma situação chamada de estouro de pilha. Cada chamada para uma função implica a criação de um item a mais em uma região da memória chamada, a pilha de execução.

Se o número de elementos na pilha de execução crescer demasiadamente, a pilha pode estourar, levando ao fim da execução do programa. Esse problema não é exclusivo de programas que utilizam recursividade, mas é mais comum nesses casos, pois um número excessivo de chamadas a uma função pode levar ao estouro da pilha de execução.

Existem algumas técnicas de otimização que podem evitar situações de estouro de pilha. Uma das mais utilizadas é a chamada de recursividade final própria, ou, em inglês, *tail recursion*. Nesse caso, uma chamada pode ser feita sem a necessidade de se adicionar um quadro na pilha de chamadas, o que evita seu crescimento desenfreado.

2. Análise da questão

Para a resolução do problema, podemos construir uma tabela na qual simulamos a execução do programa. Observe que essa tabela será construída na ordem em que a função f(v,n) retorna aos valores, e não na ordem em que ela é chamada. Isso acontece porque essa é uma função recursiva e o primeiro valor a ser retornado é 0, na linha 3, quando n é igual a 0. A função f(v[0], 0) retorna a s=0 na linha 6. Como v[0]=2>0, sabemos que s=s+2=0+2=2. Sendo assim, f(v,1)=2. Esse valor é novamente retornado à linha 6. Com f(v[1],2), mas v[1]=-4<0, portanto, f(v,2)=2. Esse processo é repetido até que se chegue ao valor f(v,6)=13, conforme tabela 1.

Tabela 1. Resultado de f(v,n) para a execução do programa.

N	f(v,n)
0	0
1	2
2	2
3	9
4	9
5	9
6	13

Alternativa correta: C.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J.R. *Introdução à Estrutura de Dados.* Rio de Janeiro: Campus, 2004.
- CORMEN, T.; LEISERSON, C.; RIVEST, R.; STEIN, C. *Introduction to algorithms*. 3. ed. Cambridge: MIT Press, 2009.
- KNUTH, D. E. *The art of computer programming.* Uppers Saddle River: Addison Wesley, 1997, v. I.
- TOSCANI, L. V.; VELOSO, P.A.S. *Complexidade de algoritmos.* 2. ed. Porto Alegre: Bookman, 2008.