Tópico 1: Recursividade

Prof. Dr. Juliano Henrique Foleis

Estude com atenção as leituras sugeridas abaixo. Os exercícios servem para ajudar na fixação do conteúdo e foram escolhidos para complementar o material básico apresentado nas leituras e nas aulas. Quando o exercício pede que crie ou modifique algum algoritmo, sugiro que implemente-o em linguagem C para ver funcionando na prática.

Leitura Sugerida

PEREIRA, Silvio Lago. Estruturas de Dados em C - Uma Abordagem Didática. [Minha Biblioteca]. Capítulo 6 (Recursão) Link

SZWARCFITER, Jayme Luiz e MARKENZON, Lilian. Estruturas de Dados e Seus Algoritmos. [Minha Biblioteca]. Capítulo 1, Seção 3 – Recursividade Link

FEOFILOFF, Paulo. Projeto de Algoritmos em C. Recursão e algoritmos recursivos. Link

Exercícios

Exercícios dos materiais de leitura sugerida

Exercícios 6.1 até 6.10 do livro do Pereira: Link

Exercícios 1.1, 1.2 e 1.3 do livro de Szwarcfiter e Markenzon Link

Exercícios 1.2, 1.3, 1.4, 1.5, 1.6, 1.9, 1.10, 2.1, 2.2, 3.1, 3.3, 3.4, 3.5, 3.6, 4.1, 4.2, 4.3, 4.4, 4.5 da página do Prof. Feofiloff Link

Exercícios Complementares

1) Escreva uma função recursiva para computar $a + b$, tal que a e b são inteiros não-negativos usando a função *sucessor* definida como:

```c
int sucessor(int x){
    return x+1;
}
```

2) A sequencia de Fibonacci generalizada é dada por:

$$G(f0, f1, 0) = f0$$
$$G(f0, f1, 1) = f1$$
$$G(f0, f1, n) = G(f0, f1, n-1) + G(f0, f1, n-2) \text{ se } n > 1$$

a) Faça um teste de mesa usando um grafo de chamadas recursivas da execução de $G(2, 3, 4)$.

b) Implemente este algoritmo recursivo em C.

c) Implemente uma versão deste algoritmo com memoização.

d) Escreva um algoritmo iterativo equivalente. Implemente este algoritmo em C.

3) O máximo divisor comum (MDC) de dois inteiros, x e y, é definido recursivamente por:

$$mdc(x, y) = y \text{ se } (y \leq x) \;\&\&\; (x\%y == 0)$$
$$mdc(x, y) = mdc(y, x) \text{ se } x < y$$
$$mdc(x, y) = mdc(y, x\%y), \text{ caso contrário}$$

a) Faça um teste de mesa usando um grafo de chamadas recursivas da execução de $mdc(30, 42)$.

b) Escreva uma função recursiva em C para implementar o MDC, conforme indicado acima.

c) Descubra um método iterativo para calcular essa função.

4) A função $com(n, k)$ deve computar quantos comitês de k pessoas podem ser formados de um grupo com n pessoas. Por exemplo $com(4, 3) = 4$ pois com 4 pessoas (A, B, C e D), existem quatro possíveis comitês de 3 pessoas: ABC, ABD, ACD e BCD. A fórmula tradicional para $com(n, k)$ é dada por:

$$C(n, k) = \frac{n!}{k!(n - k!)}$$

No entanto, há um problema para calcular esta fórmula. $n!$ cresce muito rapidamente, causando *overflow* em um inteiro de 32-bits para valores de n maiores que 8. Entretanto, é possível calcular $C(n, k)$ recursivamente, de forma que não é necessário calcular os fatoriais da fórmula. A formulação recursiva de $C(n, k)$ é dada por:

$$C(n, k) = 0 \text{ se } k > n$$
$$C(n, k) = n \text{ se } k = 1$$
$$C(n, k) = C(n - 1, k) + C(n - 1, k - 1), \text{ caso contrário}$$

a) Faça um teste de mesa usando um grafo de chamadas recursivas da execução de $C(5, 3)$.

b) Implemente este algoritmo recursivo em C.

c) O que você percebeu com o teste de mesa feito no exercício anterior? Implemente uma versão deste algoritmo recursivo para evitar o problema encontrado. (a solução é comumente chamada de Memoização)

Curiosidade: A descrição recursiva deste algoritmo vem da relação do Triângulo de Pascal com o número de subconjuntos possíveis a partir de um conjunto de objetos distintos.

5) Seja A uma matriz de inteiros 10x10 que representa um labirinto. A entrada do labirinto está em $(0, 0)$ e a saída em $(9, 9)$. Considere que esta matriz inicialmente tenha apenas valores 0 e 1, onde 0 indica uma posição vazia e 1 indica uma posição ocupada. Faça um algoritmo para encontrar um caminho da entrada até a saída. Seu algoritmo deve imprimir as posições que estão no caminho, não necessariamente na ordem a partir do início. Caso não haja um caminho, imprima "Não existe um caminho para a saída!". Somente movimentos para as quatro posições adjacentes são permitidos, ou seja, não são permitidos movimentos nas diagonais. Seu algoritmo não precisa encontrar o caminho mais curto. Crie labirintos e avalie o funcionamento do seu algoritmo.

Dica: Faça um algoritmo recursivo $Explorar(A, i, j)$ onde A é a matriz, i é a linha e j é uma coluna da matriz. "Explorar" a posição (i, j) é avaliar se já chegou na saída, caso contrário explorar as posições adjacentes. Para

evitar que muitas chamadas recursivas redundantes sejam realizadas, marque as posições a serem exploradas com um valor diferente de 0 ou 1.

Curiosidade: Este é um algoritmo conhecido no processamento de imagens, chamado *Flood Fill*. Ele é usado para implementar as ferramentas de preencher uma área de pixels de uma mesma cor com uma outra cor qualquer.

BONS ESTUDOS!