

MAC122 Princípios de Desenvolvimento de Algoritmos - BM,BMA Segundo Semestre de 2011

Exercício-Programa 1, Peso 1

Data de entrega: veja no paca.ime.usp.br.

Observações

- Este exercício é para ser feito individualmente.
- Entregue no sistema PACA UM ÚNICO arquivo contendo os arquivos seguintes, eventualmente comprimidos:
 - um arquivo chamado LEIA.ME (em formato .txt) com:
 - * seu nome completo, e número USP,
 - * os nomes dos arquivos inclusos com uma breve descrição de cada arquivo,
 - * uma descrição sucinta de como usar o programa executável, necessariamente na linha-de-comando, i.e., SEM interface gráfica,
 - * qual computador (Intel, AMD, ou outro) e qual sistema operacional (LINUX, UNIX, iOS, ou outro) foi usado,
 - * instruções de como compilar o(s) arquivo(s) fonte(s).
 - o arquivo MAKE, se for o caso,
 - os arquivos do programa-fonte necessariamente em linguagem ANSI-C, compatível com o compilador GCC.
 - o programa compilado, i.e., **incluir o código executável (se não incluir, a nota será zero!)**. Programa com algum erro de compilação receberá **nota zero!**
 - se for o caso, alguns arquivos de entrada e saída usados nos testes: arquivos com os dados de entrada chamados ENT1, ENT2, etc., e arquivos com os dados de saída correspondentes, chamados SAI1, SAI2, etc.
- O seu programa deve ser compilado pelo compilador GCC. Você deve compilá-lo com as opções (flags) `-ansi` e `-pedantic` (veja URL para documentação correspondente no paca.ime.usp.br)
- Coloque comentários em seu programa explicando o que cada etapa do programa significa! Isso será levado em conta na sua nota.

- Faça uma saída clara! Isso será levado em conta na sua nota.
- Não deixe para a última hora. Planeje investir 70 por cento do tempo total de dedicação em escrever o seu programa todo e simular o programa SEM computador (eliminando erros de lógica) ANTES de digitar e compilar no computador. Isso economiza muito tempo e energia.
- A nota será diminuída de um ponto a cada dia “corrido” de atraso na entrega.

1 Torres de Hanói

Este EP consiste em implementar uma versão NÃO-recursiva do jogo das Torres de Hanói utilizando “stacks” (pilhas) em vetor, como visto em aula. O seu programa deve ler o número de argolas do teclado; prever no máximo 20 argolas.

Recursivamente, a solução é executar a função $TH(n,A,B,C)$ (conforme definição abaixo) onde n é o número de argolas, e A,B,C representam os três pinos, sendo que as n argolas estão no pino A . As restrições são: (1) mover só uma argola de cada vez, e (2) nunca ocorrer argola maior em cima de uma menor. No final o pino C deverá estar com as n argolas. A função $TH()$ é:

$TH(n,A,B,C)$	// n,A,B,C s o inteiros	Linha
se $n==1$ {	mover argola de A para C ;} L1	
sen o se $n>=2$ {		L2
	$TH(n-1,A,C,B)$;	L3
	mover argola de A para C ;	L4
	$TH(n-1,B,A,C)$;	L5
	} // fim se	L6

Por exemplo, para $n=3$, se representamos a maior argola por 3, e a menor por 1, a execução de $TH(3,A,B,C)$ resulta os seguintes passos: (o índice a no pino X , como em X_a , só indica que o pino X é o primeiro pino na execução de $TH(n, X_a, Y_b, Z_c)$)

nível 0	nível 1	nível 2	nível 3
TH(3, A_a, B_b, C_c)	L3: TH(2, A_a, C_b, B_c)	L3: TH(1, A_a, B_b, C_c)	L1: $A_a \rightarrow C_c$
$[A, B, C] = [\begin{smallmatrix} 1 \\ 2 \\ 3 \end{smallmatrix}, 0, 0]$			$[A, B, C] = [\begin{smallmatrix} 2 \\ 3 \end{smallmatrix}, 0, 1]$
		L4: $A_a \rightarrow B_c$	
		$[A, B, C] = [3, 2, 1]$	
		L5: TH(1, C_a, A_b, B_c)	L1: $C_a \rightarrow B_c$
			$[A, B, C] = [3, \begin{smallmatrix} 1 \\ 2 \end{smallmatrix}, 0]$
	L4: $A_a \rightarrow C_c$		
	$[A, B, C] = [0, \begin{smallmatrix} 1 \\ 2 \end{smallmatrix}, 3]$		
	L5: TH(2, B_a, A_b, C_c)	L3: TH(1, B_a, C_b, A_c)	L1: $B_a \rightarrow A_c$
			$[A, B, C] = [1, 2, 3]$
		L4: $B_a \rightarrow C_c$	
		$[A, B, C] = [1, 0, \begin{smallmatrix} 2 \\ 3 \end{smallmatrix}]$	
		L5: TH(1, A_a, B_b, C_c)	L1: $A_a \rightarrow C_c$
			$[A, B, C] = [0, 0, \begin{smallmatrix} 1 \\ 2 \\ 3 \end{smallmatrix}]$

Vamos convencionar que a variável topo == -1 significa pilha vazia.

Como sugestão, você pode implementar as funções listadas a seguir. Você NÃO precisa aceitar estas sugestões.

1. Uma função chamada Vazia(S, topo) que retorna -1 se a pilha S estiver vazia, e 1 no caso contrário;
2. Uma função chamada Empilha(S, topo, Elem, Nmax) que empilha o elemento Elem na pilha S, se houver memória suficiente, e retorna o novo índice do topo; se não houver memória para o novo elemento, deve retornar -2 (sinal de "overflow");
se topo+1 < Nmax {
topo = topo + 1;
S[topo] = Elem;
retorna topo;
} // fim se
senão retorna -2
3. Uma função Desempilha(S, topo, Elem) que desempilha o elemento Elem do topo da pilha, se não estiver vazio, e retorna o novo índice do topo; se estiver vazia, deve retornar -2 (sinal de "underflow").

```

se      topo!=-1{
        Elem=S[topo];
        topo=topo-1;
        retorna topo;
    }// fim se
senão   retorna -2

```

A definição não-recursiva da função TH() é como segue:

```

L1  Inicializar pilha S como sendo vazio;
L2  Empilha (n,A,B,C) em S;
L3  enquanto (pilha não vazia) {
L4
L5      Desempilha de S, obtém (n,A,B,C);
L6      se(n==1) mover argola de A para C
L7      senão {
L8          Empilha (n-1,B,A,C) em S;
L9          Empilha (1,A,B,C) em S;
L10         Empilha (n-1,A,C,B) em S;
L11     } // fim senão
    } // fim enqto

```

Os elementos na pilha na execução para n=3 serão:

1. no início:

n			
3	A	B	C

índice
topo 0

2.

n			
2	A	C	B
1	A	B	C
2	B	A	C

índice
topo 2
1
0

3.

n			
1	A	B	C
1	A	C	B
1	C	A	B
1	A	B	C
2	B	A	C

índice mover
topo 4 A→C
3 A→B
2 C→B
1 A→C
0

4.

n			
2	B	A	C

índice
topo 0

5.

n			
1	B	C	A
1	B	A	C
1	A	B	C

índice mover
topo 2 B→A
1 B→C
0 A→C

As argolas serão representadas por inteiros 1,2,3,..., n.

O seu programa deve:

1. Representar a pilha S, ilustrado acima, por um vetor (array) S de 4 colunas contendo inteiros. Assim,
 - (a) S[topo][0] conteria o valor de n,
 - (b) S[topo][1] conteria o inteiro que representa um dos 3 pinos,
 - (c) S[topo][2] conteria o inteiro que representa um outro pino, e
 - (d) S[topo][3] conteria o inteiro que representa um terceiro pino.
2. Representar os 3 pinos por um array PINO de 3 linhas, cada linha representando um pino.
 - (a) A linha 0 armazenaria as argolas do pino A
 - (b) A linha 1 armazenaria as argolas do pino B
 - (c) A linha 2 armazenaria as argolas do pino C
3. Armazenar os índices dos topos dos 3 pinos em um vetor chamado TopoPino. Assim,
 - (a) TopoPino[0] seria o índice do topo do pino A. E a argola no topo do pino A estaria em PINO[A][TopoPino[A]], onde A vale 0.
 - (b) TopoPino[1] seria o índice do topo do pino B. E a argola no topo do pino B estaria em PINO[B][TopoPino[B]], onde B vale 1.
 - (c) TopoPino[2] seria o índice do topo do pino C. E a argola no topo do pino C estaria em PINO[C][TopoPino[C]], onde C vale 2.
4. Escrever 3 funções para os pinos: PinoVazio, EmpilhaNoPino, DesempilhaDoPino, pois cada pino é um pilha também!

O seu programa deve mostrar o conteúdo do topo da pilha S, cada vez que este for desempilhado ou empilhado, e os topos dos 3 pinos A,B,C logo depois de cada movimento de argola.