SME0230 - Introdução à Programação de Computadores

Prova de recuperação - 03/08/2020

Instruções e observações

Esta prova é **individual** e deve ser feita pela(o) própria(o) aluna(o). Suponho que somos todos adultos com um senso ético construído, que permite dissernir o certo do errado. Apenas para reforçar: colar na prova é errado! Provas com questões iguais ou muito similares receberão nota 0. E, neste caso, a média final será 0 também.

A entrega da prova será feita por e-mail. Cada questão deverá ser resolvida em um arquivo .c diferente e os 3 arquivos deverão ser enviados para andretta@icmc.usp.br, até o dia 10 de agosto de 2020, às 23h59min.

Alguns cuidados com os códigos que serão entregues são muito importantes. Isso já foi falado e reforçado durante todo o semestre, mas não custa lembrar:

- Comente seu código! Não é para escrever um tratado ou descrever literalmente seu código em português. Os comentários devem ajudar a entender a ideia usada para resolver o exercício e apontar o que trechos de código fazem.
- Deixe seu código claro e organizado. O uso de funções para executar sub-tarefas pode ajudar bastante nesta parte. E não complique seu código à toa (ou para mostrar que sabe alguma coisa). Clareza e boas ideias são muito importantes!
- Seu código precisa resolver os problemas que foram propostos e atender a todos os requisitos pedidos. Não adianta fazer um código que resolve outro problema, caso você não saiba como resolver algum problema proposto. E se alguma coisa for pedida explicitamente, ela precisa ser atendida.
- Não custa lembrar mais uma vez: **NÃO** é permitido usar variáveis globais ou "goto". Códigos com essas coisas receberão nota 0.
- Indentação é **essencial**. Códigos não (ou mal) indentados terão um desconto **considerável** na nota.

Apesar de todas as "pré-broncas", espero que todas(os) se divirtam com a (resolução da) prova. Não entrem em pânico!

Boa prova!

Marina

Questões

1. (3.0) [Adaptado da prova da OBI 2018 - Fase 1] Dizemos que uma sequência de pelo menos dois números é uma *escadinha* se a diferença entre números consecutivos é sempre a mesma.

Por exemplo, a sequência $2, 3, 4, 5$ é uma escadinha, já que um número sempre é igual ao anterior somado de 1. A sequência $10, 7, 4$ também é uma escadinha, já que um número sempre é igual ao anterior somado de -3. Já a sequência $2, 6, -1$ não é uma escadinha, porque do primeiro número para o segundo foi somado o valor 4, mas do segundo para o terceiro foi somado o valor -7.

Em uma sequência maior de números, podemos encontrar algumas escadinhas. Estamos sempre interessados nas maiores escadinhas possíveis, já que um pedaço de uma escadinha sempre é uma escadinha também.

Por exemplo, na sequência $1, 1, 1, 3, 5, 4, 8, 12$ temos 4 escadinhas diferentes: $1, 1, 1$ (soma 0 de um número para o próximo), $1, 3, 5$ (soma 2 de um elemento para o próximo), $5, 4$ (soma -1 de um elemento para o próximo) e $4, 8, 12$ (soma 4 de um elemento para o próximo).

Escreva um programa, em linguagem C, que leia um número inteiro $n > 0$ correspondente ao tamanho da sequência, seguido de n números inteiros que compõem a sequência. Seu programa deve mostrar **quantas** e **quais** são as escadinhas da sequência digitada. Seu programa deve cuidar do caso em que o valor de n é inválido.

2. (3.0) [Adaptado da prova da OBI 2017 - Fase 1] O sistema de segredo para abrir esse cofre é bastante complexo. Ao invés de girar um botão várias vezes, como a gente vê normalmente nos filmes, o dono do cofre tem que deslizar um controle para a esquerda e para a direita, em cima de uma barra, várias vezes, parando em determinadas posições. A barra possui n posições e cada posição contém um número inteiro de 0 a 9.

No exemplo abaixo, a barra tem 14 posições e o controle está na posição 1.

9	4	3	9	1	2	4	5	1	1	9	7	0	5
1	2	3	4	5	6	7	8	9	10	11	12	13	14

O segredo vai depender de quantas vezes cada um dos dez inteiros de 0 a 9 vai aparecer dentro do controle.

Por exemplo, usando a barra acima, suponha que o dono deslize o controle da posição inicial 1 até a posição 9 (percorrendo os números $9, 4, 3, 9, 1, 2, 4, 5, 1$), depois para a posição 4 (percorrendo os números $5, 4, 2, 1, 9$), depois para a posição 11 (percorrendo os números $1, 2, 4, 5, 1, 1, 9$) e por fim até a posição 13 (percorrendo os números $7, 0$). Veja que o inteiro 1, neste exemplo, vai aparecer seis vezes dentro do controle; e o inteiro 9 vai aparecer quatro vezes.

Escreva um programa, em linguagem C, que leia a quantidade $n > 0$ de números na barra, a sequência de n inteiros da barra, a quantidade $m > 0$ de posições que o dono vai percorrer e a sequência de m posições entre as quais o dono desliza o controle, começando da posição inicial 1. Seu programa deve contar quantas vezes cada inteiro de 0 a 9 vai aparecer dentro do controle.

Seu programa deve cuidar do caso em que os valores de n ou m são inválidos, bem como valores inválidos para as sequências de números na barra e de posições para deslizar.

3. (4.0) Um tabuleiro de *Sudoku* é uma matriz de inteiros de dimensão 9×9, composta por 9 submatrizes de dimensão 3×3. Neste tabuleiro, cada linha e coluna deve conter todos os números inteiros de 1 a 9. Além disso, cada submatriz também deve conter todos os números inteiros de 1 a 9.

Por exemplo, o tabuleiro

9	7	6	5	4	3	1	2	8
8	2	1	7	9	6	4	5	3
5	4	3	2	1	8	9	7	6
2	6	4	1	3	9	7	8	5
7	5	8	4	6	2	3	1	9
1	3	9	8	7	5	6	4	2
6	1	2	9	5	4	8	3	7
4	9	5	3	8	7	2	6	1
3	8	7	6	2	1	5	9	4

está preenchido corretamente.

Para jogar Sudoku, o tabuleiro começa parcialmente preenchido e o jogador vai preenchendo as posições vazias com inteiros de 1 a 9, de forma a preencher todo o tabuleiro de maneira válida. Ou seja, não pode haver números repetidos nas linhas, nem nas colunas, nem nas submatrizes. Ao final, todos os números de 1 a 9 devem estar em todas as colunas, linhas e submatrizes.

Escreva um programa, em linguagem C, que leia um inteiro $k > 0$, que indica quantas posições do tabuleiro serão previamente preenchidas, e uma lista com cada uma das posições da matriz do tabuleiro (linha e coluna) que serão preenchidas, bem como os valores a serem usados.

Por exemplo, o tabuleiro que deve ser preenchido inicialmente como

9		6	5					8
8			7			4		
				1		9		
	6				9		8	
			8				4	
		2					3	
	9				7			
				2		5		

deve ter como entrada

20
1 1 9
1 3 6
1 4 5
1 9 8
2 1 8
2 4 7
2 7 4
3 5 1
3 7 9
4 2 6
4 6 9
4 8 8

6 4 8
6 8 4
7 3 2
7 8 3
8 2 9
8 6 7
9 5 2
9 7 5

Seu programa deve verificar se o valor de k é válido, se as posições e valores a serem usados são válídos e se não tornam o tabuleiro inválido.

Depois de preencher o tabuleiro inicial, seu programa deve pedir para o usuário digitar a linha e coluna do tabuleiro que deseja preencher e o valor que deseja usar. Caso a posição ou o valor escolhido sejam inválidos (o que inclui tornar o tabuleiro inválido), o usuário deve jogar novamente; caso contrário, a jogada deve ser feita. Esse processo deve ser repetido até que o tabuleiro esteja completo e válido.

Deve ficar claro para o usuário como está o tabuleiro a cada jogada e o que ele deve digitar.