Estrutura de Dados - MC202D
1º Semestre 2017

Instituto de Computação - UNICAMP

Primeira Lista de Exercícios

Entregue os exercícios 1, 2 e 3 até 08/10/2017 pelo SuSy
Leia com cuidado as instruções abaixo

Instruções:

- Você deverá entregar os três exercícios separados em três tarefas diferentes no SuSy: o exercício 1 deve ser entregue na tarefa `l1ex1`; o exercício 2 deve ser entregue na tarefa `l1ex2`; e o exercício 3 deve ser entregue na tarefa `l1ex3`. Caso não faça algum dos exercícios, basta não submeter na tarefa referente ao exercício.
- A nota da lista será dada em relação aos três exercícios, mesmo considerando o fato que a submissão ocorre em três tarefas diferentes do SuSy.
- Os exercícios podem digitados em um computador (preferencialmente) ou serem escritos a mão e digitalizados (você pode inclusive tirar uma foto). Porém, a entrega precisa ser feita necessariamente através de um arquivo PDF de tamanho no máximo 2MB. Outros formatos não serão aceitos.
- Garanta que o documento tenha qualidade o suficiente para ser lido. Documentos escritos a mão devem ter letras legíveis e feitos preferencialmente a caneta, já que o lápis tem pouco contraste para a digitalização. Assim, se possível, opte por fazer o exercício no computador ao invés de digitalizar.
- Caso tenha problemas com o tamanho do arquivo gerado, procure por um compressor de PDFs online. Existem vários serviços que recebem um PDF e geram o mesmo PDF com um tamanho menor. Porém, tome cuidado para que o arquivo continue legível.
- Lembre-se de indentar corretamente o seu código para facilitar o entendimento.
- Não se esqueça de colocar nome e RA em cada exercício.
- A correção de cada exercício será enviada para o seu email institucional (da DAC). Garanta que há espaço na sua cota para receber os emails[1] e verifique o spam se necessário.

[1] Considere fazer o redirecionamento do seu email da DAC para o seu email pessoal: `https://www.dac.unicamp.br/portal/estudantes/webmail-mais-informacoes`

1. Faça uma implementação em C do TAD Fila (para armazenar números inteiros) utilizando uma lista circular com cabeça, como descrito no slide 8 da unidade 08. Em particular, defina a `struct Fila` e `p_fila` e dê a implementação das funções:

- `p_fila criar_fila()`
- `void destruir_fila(p_fila f)`
- `void enfileira(p_fila f, int x)`
- `int desenfileira(p_fila f)`
- `int fila_vazia(p_fila f)`

2. Mostre como representar um polinômio utilizando uma lista ligada simples sem cabeça. Em particular, defina a `struct No` de maneira apropriada. A partir desta definição, apresente a implementação em C da função `p_no derivada(p_no polinomio)` que, dado um polinômio, devolve a derivada deste polinômio. Suponha que o polinômio dado não tem dois ou mais monômios com o mesmo coeficiente. Por exemplo, $3x^7 + 4x^3 - 15$ e $4x^3 - 15 + 3x^7$ são polinômios válidos como entrada, mas $2x^7 + 1x^7 + 4x^3 - 15$ não.

Esta função deve ser **estritamente recursiva**. Isto é, você não deve usar `for`, `while`, `do while` ou `goto`.

3. Apresente uma forma de implementar duas pilhas em um único **vetor** de n posições de forma que as pilhas juntas possam armazenar no total até n elementos sem qualquer problema. Isto é, se a é a quantidade elementos da primeira pilha e b é a quantidade de elementos da segunda pilha, então temos que $0 \leq a \leq n$, $0 \leq b \leq n$ e $a + b \leq n$. Consideramos que a primeira pilha é a pilha de índice 0 e que a segunda pilha é a pilha de índice 1.

Em particular:

a) defina uma `struct PilhaDupla` com os campos apropriados para a sua implementação e o respectivo tipo `p_pilhadupla` representando um ponteiro para a `struct PilhaDupla`.

b) apresente as seguintes funções em C:

- `p_pilhadupla criar_pilhadupla(int n)` - cria uma nova `PilhaDupla` vazia com espaço para n elementos e devolve um ponteiro para a mesma.
- `void destruir_pilhadupla(p_pilhadupla pd)` - recebe um ponteiro `pd` para uma `PilhaDupla` e a destrói, desalocando toda a memória alocada na função `criar_pilhadupla`.
- `void empilhar(p_pilhadupla pd, int valor, int pilha)` - empilha `valor` na pilha de índice `pilha` da `PilhaDupla` apontada por `pd`.
- `int desempilhar(p_pilhadupla pd, int pilha)` - desempilha o elemento do topo da pilha de índice `pilha` da `PilhaDupla` apontada por `pd` e o retorna.
- `int vazia(p_pilhadupla pd, int pilha)` - devolve `1` se a pilha de índice `pilha` da `PilhaDupla` estiver vazia e `0` caso contrário.
- `int cheia(p_pilhadupla pd, int pilha)` - devolve `1` se a pilha de índice `pilha` da `PilhaDupla` estiver cheia (não é possível inserir novos elementos sem perder dados da primeira ou segunda pilha) e `0` caso contrário.