Prova 1 – Estruturas de Dados (INE5408) – 14dez2021

Ciências da Computação – Universidade Federal de Santa Catarina

Atenção: Esta prova deve ser respondida individualmente até amanhã (quarta) às 12h (meio-dia). A submissão consiste em um único documento em "pdf" ou "odt" ou "docx" com todas as respostas digitadas ou digitalizadas (em caso de utilização de fotografias, verifique se há iluminação suficiente e resolução apropriada).

Importante: Para a resolução das questões, as constantes α e β são extraídas de seu número de matrícula da UFSC, conforme as equações a seguir ("**mod**" é o operador de "resto da divisão inteira"):

Entre com sua matrícula:

$\square\square\square\square\square\square\boxed{A}\boxed{B}\boxed{C}$

$$\begin{cases} R = (A + B + C) \ \textbf{mod} \ 8 & \\ \alpha = 9 - R & \Rightarrow \alpha = \boxed{} \\ \beta = R + 2 & \Rightarrow \beta = \boxed{} \end{cases}$$

1. Considere o seguinte processamento de um vetor de N inteiros:

```
ArrayStack<int> *questao1(int *vet, int N, int α, int β) {
    ArrayStack<int> *pilha = new ArrayStack<int>((α + β)*N);
    for (int i = 0; i < N; i++) {
        if (vet[i] == α) {
            for (int j = 0; j < (α + β); j++) {
                pilha->push(i);
            }
        } else if ( ! pilha->empty() && vet[i] == β ) {
            pilha->pop();
        }
    }
    return pilha;
}
```

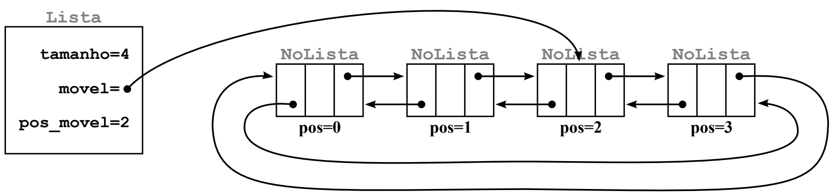
   (a) *(1,0pt)* Desenhe a pilha construída para a seguinte chamada:

```
const int N = 20;
int vet[N] = {0,1,2,3,4,5,6,7,8,9,9,8,7,6,5,4,3,2,1,0};
ArrayStack<int> *pilha = questao1(vet, N, α, β);
```

   (b) *(1,0pt)* Qual(is) a(s) vantagem(ns) em se trocar `ArrayStack` por `ArrayList` nessa implementação? Justifique.

   (c) *(1,0pt)* Expresse a complexidade (assintótica) desse algoritmo em termos de N, α e β.

2. *(2,0pt)* Considere a classe `Lista` do desenho a seguir para uma lista dinâmica duplamente encadeada e circular, contendo um inteiro para a quantidade atual de elementos (`tamanho`), um ponteiro para um nó qualquer (`movel`) e um inteiro para indicar a posição desse nó (`pos_movel`), de forma que o único ponteiro (`movel`): • em uma operação de busca, aponte para o nó procurado (a figura abaixo refere-se à busca pelo dado na posição 2); • em uma inserção, aponte para o nó recém inserido; e, • em uma remoção, aponte para o nó à direita (se houver) do nó removido. A estrutura `NoLista` de nó da lista possui um tipo `T` (`dado`), ponteiros para anterior (`ant`) e próximo (`prox`).



Considerando que o ponteiro `movel` pode se encontrar em qualquer lugar e **não se pode acrescentar nenhum outro ponteiro à classe** `Lista`, escreva o pseudocódigo para implementar a operação de:

- **Para alunos com $\beta = 2$ ou $\beta = 6$:**
  Inserção no início: $\Rightarrow$ `void insereNoInicio(T dado);`
- **Para alunos com $\beta = 3$ ou $\beta = 7$:**
  Remoção do fim: $\Rightarrow$ `T removeDoFim();`
- **Para alunos com $\beta = 4$ ou $\beta = 8$:**
  Inserção no fim: $\Rightarrow$ `void insereNoFim(T dado);`
- **Para alunos com $\beta = 5$ ou $\beta = 9$:**
  Remoção do início: $\Rightarrow$ `T removeDoInicio();`

3. Um método foi desenvolvido para inserir uma **lista encadeada** secundária com n elementos na posição p de uma lista encadeada principal, sem utilização de nenhuma lista auxiliar. Por exemplo:

   - Lista secundária com $n = 3$ elementos: [X, Y, Z]
   - Lista principal: [A, B, C, D, E, F, G, H, I, J]
   - Para $p = 7$:
     - **Lista resultante**: [A, B, C, D, E, F, G, X, Y, Z, H, I, J]
   - Para qualquer $p \geq 10$ (*append*):
     - **Lista resultante**: [A, B, C, D, E, F, G, H, I, J, X, Y, Z]

   Considerando que, em sendo p igual ou superior ao tamanho da lista principal, a lista secundária deve ser simplesmente acrescentada ao final da primeira (conforme o exemplo *append*), e:

   - $n = 100\alpha$
   - $p = 100\beta$

   Analise a **quantidade de atribuições de um ponteiro para nó de lista** (tipicamente: `it = it->next();`) ao longo da execução dessa inserção para finalizar o processamento de geração da lista resultante, se for utilizada uma:

   (a) *(1,0pt)* Lista dinâmica simplesmente encadeada com um único ponteiro para o início.
   (b) *(1,0pt)* Lista dinâmica duplamente encadeada com um ponteiro para o início e outro para o fim.
   (c) *(1,0pt)* Lista dinâmica duplamente encadeada com um único ponteiro móvel, conforme o exercício anterior.

4. O algoritmo a seguir processa um objeto x de tamanho n, gerando y como solução:

   $\text{ALGORITMO}(x)$
   1: **se** x é suficientemente pequeno **então** // Custo: 1
   2: **devolve** $\text{SOLUÇÃO}(x)$ // Custo: 1
   3: **senão** // Custo: 1
   4: $x_1, x_2, \ldots, x_k \leftarrow \text{DIVIDIR}(x)$ // Custo: n
   5: **para** $i \leftarrow 1$ **até** k **faça** // Custo: k
   6: $y_i \leftarrow \text{ALGORITMO}(x_i)$ // Custo: $T\left(\frac{n}{p}\right)$
   7: $y \leftarrow \text{COMBINAR}(y_i)$ // Custo: n
   8: **devolve** y // Custo: 1

   Considerando que o custo da linha 6 é: $T\left(\frac{n}{p}\right)$ (custo de cada chamada recursiva); o custo das linhas 4 e 7 é: n (tempo linear); o custo da linha 5 é: k (tempo constante); o custo das demais linhas é: 1 (tempo constante), pede-se:

(a) *(1,0pt)* Escreva a relação de recorrência que representa o custo total $T(n)$ de ALGORITMO dada pela soma de custos de todas as linhas (não é preciso resolvê-la, ou seja, não é preciso encontrar a complexidade).

(b) *(1,0pt)* Compare duas versões da implementação em termos de tempo computacional:

- Implementação 1: $k = \alpha$; $p = \alpha$
- Implementação 2: $k = \alpha$; $p = \beta$

Boa prova!