UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação - ICMC
SCC0202 – Algoritmos e Estrutura de Dados I - 2º Sem /2023
Prof. Rudinei Goularte, Marcelo Manzato

Lista de Exercícios - Listas

1. Uma lista com inserção ordenada é capaz de inserir um elemento com complexidade O(n), mas em compensação, a busca de um elemento nessa lista pode ter complexidade O(log n). Pensando assim, qual é a vantagem de possuir uma lista com inserção ordenada em relação a possuir uma lista comum, porém ordená-la em seguida?

2. Suponhamos uma aplicação de lista de tarefas, ou *To do lists*, em que um usuário pode criar diferentes tarefas e adicioná-las na sua lista de afazeres. Essa aplicação específica pretende impedir que o usuário deixe muitas tarefas incompletas para evitar sobrecarga. Assim, um usuário pode ter apenas 10 itens na sua *to do list*.

   Considerando que vamos implementar essa aplicação para rodar em uma máquina convencional - um computador pessoal, por exemplo - que tipo de lista pode ser mais adequada para essa aplicação?

3. Suponha que cadeias de caracteres são armazenadas em listas encadeadas com um caractere por célula. Escreva as funções análogas a strlen, strcmp, strcpy e strcat. Considere que a lista pode ser simplesmente ligada ou duplamente ligada, conforme preferir.

4. Um tipo de dado abstrato muito comum é o chamado *Big Int*, que consiste em uma estrutura capaz de armazenar e representar números inteiros maiores que 8 bytes. Ele é muito usado em programas científicos que trabalham com grandezas grandes (trocadilho ye ye) e uma das suas implementações envolve armazenar cada dígito do número como um nó em uma lista encadeada (cada nó também pode ser um inteiro de 4 bytes, mas vamos nos ater à implementação mais simples).

   Implemente as operações de soma e subtração de *Big Int*s. Considere o TAD abaixo. Use a implementação de lista encadeada dupla ou simples, como preferir.

```
struct bigint_ {
  LISTA * numeros;
  int positivo;
}
```

5. Construa um método que recebe uma lista encadeada de números inteiros e retorna uma lista sem repetições, ou seja, uma lista onde cada número aparece apenas uma vez.

Exemplo:
12 5 -7 8 5 9 12 1 8 → 12 5 -7 8 9 1

6. Imagine uma máquina que possui recursos de memória muito escassos e, por isso, a sua implementação de lista não contará com o atributo *tamanho*. Essa máquina entregará a lista para uma outra máquina - convencional, com memória e processamentos mais adequados. Essa nova máquina pretende calcular o tamanho da lista. Considerando a implementação encadeada, implemente uma função que calcula o tamanho da lista encadeada.

7. Agora imagine que temos uma lista circular de inteiros sem elementos repetidos e queremos calcular o número de elementos daquela lista. Também não temos o atributo *tamanho* no TAD. Implemente uma função para calcular o tamanho da lista usando apenas os métodos definidos no TAD, isto é, sem usar os atributos internos da lista. O seu TAD possui, além dos métodos convencionais, o seguinte método:

   int proximo(LISTA * lista)

   Quando esse método é chamado pela primeira vez, ele retorna o primeiro elemento da lista. Nas vezes subsequentes, ele retorna o valor do elemento seguinte em relação ao que foi chamado pela última vez. Como a lista é circular, após chamar o método para o último elemento, chamá-lo novamente retornará o valor do primeiro elemento.

8. Agora imagine que temos uma lista circular e encadeada que pode ter elementos repetidos. Sem usar os atributos internos da lista, pode ser difícil implementar um método para calcular o tamanho da lista. Agora, se tivermos uma sentinela, isto é, um nó cabeçalho, a implementação fica mais fácil, usando ou não os atributos internos da lista. Implemente uma função para calcular o tamanho dessa lista, dado que ela usa o nó cabeçalho.

9. A linguagem C entende que dois dados são iguais quando o valor deles é igual. Acontece que, para ponteiros, eles são considerados iguais apenas quando eles apontam para o mesmo endereço de memória. Nesse sentido, queremos um método para comparar duas listas encadeadas e verificar se elas possuem os mesmo elementos, na mesma ordem, com o mesmo tamanho. Implemente uma função **não recursiva** para verificar se duas listas encadeadas e dinâmicas L1 e L2 são iguais.

10. Agora implemente uma função **recursiva** para fazer a mesma verificação da questão anterior.

Fun fact (não vai ser cobrado na prova e nem é assunto da matéria, mas é curioso)

Quando acessamos um endereço de memória, o sistema operacional internamente salva um pedaço “vizinho” da memória em uma parte chamada *cache*. A *cache* é mais rápida para ser acessada, e, por isso, quando tentamos acessar um dado que está na *cache*, ele é acessado

mais rápido do que um dado que não está na *cache*.

Assim, em uma lista linear, como os dados estão todos muito próximos, percorrê-la costuma ser consideravelmente mais rápido do que percorrer uma lista encadeada - em que os dados estão dispersos pela memória *heap* -, mesmo que a complexidade para percorrer ambas as listas seja O(n).🤯🤯