Universidade Federal do Ceará - Campus Quixadá
QXD0010 – Estruturas de Dados – Turma 03A – 2021.1
Prof. Atílio Gomes

Listas Sequenciais (Vetores)

1. Faça uma função que encontre o valor mínimo em uma lista sequencial.
2. Escreva uma função que remova de uma lista sequencial o elemento que tem o valor máximo.
3. Escreva uma função que inclui um elemento em uma lista sequencial.
4. Escreva uma função que remove um elemento de uma lista sequencial.
5. Escreva uma função que busca a primeira ocorrência de um elemento x em uma lista sequencial e retorna a posição do elemento na lista caso ele ocorra, ou retorne -1 caso contrário.
6. Escreva algoritmos de inserção e remoção em uma lista sequencial cujos elementos estão ordenados em ordem crescente e devem permanecer ordenados após estas operações. Qual a complexidade dos seus algoritmos? Justifique.
7. Faça uma função para remover de uma lista sequencial todos os elementos com valor x, dado como entrada.
8. Escreva uma função que retorna a quantidade de vezes que o elemento x aparece na lista sequencial.

Listas Simplesmente Encadeadas

Para as questões a seguir, considere que a lista simplesmente encadeada é implementada por meio de uma classe chamada `List` e que o único atributo da classe `List` seja um ponteiro para um nó auxiliar, chamado nó cabeça, tal como foi implementado em sala. Esse ponteiro é definido da seguinte maneira:

```
1 Node *head;
```

Além disso, considere que um nó de uma lista encadeada é definido como uma estrutura (`struct Node`) da seguinte maneira:

```
1 struct Node {
2     int key;
3     Node *next;
4 };
```

Modifique a classe `List` de modo a acrescentar nela as funcionalidades descritas nas questões a seguir:

9. Implemente uma função que tenha como valor de retorno o comprimento da lista encadeada, isto é, que calcule o número de nós da lista. Essa função deve obedecer ao protótipo:
`int comprimento();`
Observação: Note que, como estamos implementando a lista como uma classe, a função é autocontida e não precisa de nenhum parâmetro de entrada, já que ela tem acesso direto ao único atributo da classe, que é o ponteiro `head`.

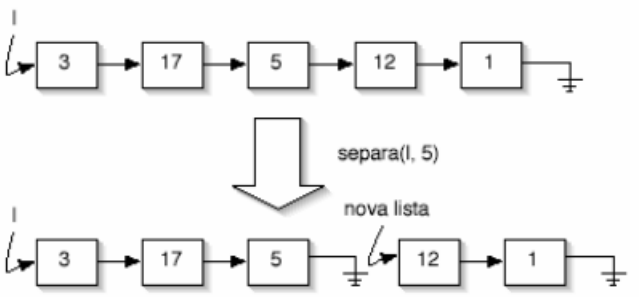
10. Implemente uma função para retornar o número de nós da lista que possuem o campo *key* com valores maiores do que n. Essa função deve obedecer ao protótipo:
`int maiores(int n);`

11. Implemente uma função que tenha como valor de retorno o ponteiro para o último nó de uma lista encadeada. Essa função deve obedecer ao protótipo: `Node* ultimo();`

12. Implemente uma função que receba uma lista encadeada do tipo `List` e retorne a lista resultante da concatenação da lista atual com a lista recebida como parâmetro, isto é, após a concatenação, o último elemento da lista a qual a função pertence deve apontar para o primeiro elemento da lista recebida por parâmetro. Além disso, a lista recebida por parâmetro deve ficar vazia. Essa função deve obedecer ao protótipo:
`void concatena(List* lista2);`

13. Implemente uma função que receba como parâmetro um valor inteiro n e retire da lista todas as ocorrências de n. Essa função deve obedecer ao protótipo:
`void retira_n(int n);`

14. Implemente uma função que receba como parâmetro um valor inteiro n e divida a lista em duas, de forma à segunda lista começar no primeiro nó logo após a primeira ocorrência de n na lista original. A figura a seguir ilustra essa separação:



Essa função deve obedecer ao protótipo: `List* separa(int n);`

A função deve retornar um ponteiro para a segunda subdivisão da lista original, enquanto a cabeça da lista original deve continuar apontando para o primeiro elemento da primeira lista, caso ele não tenha sido o primeiro a ter valor n.

15. Implemente uma função que receba uma `List` como parâmetro e construa uma nova lista com a intercalação dos nós da lista original com os nós da lista passada por parâmetro. Essa função deve retornar a lista resultante, conforme ilustrado na Figura 1.

Essa função deve obedecer ao protótipo: `void combina_filas(List* lista2);`

16. Faça uma função que encontre uma nó de conteúdo mínimo em uma lista encadeada.

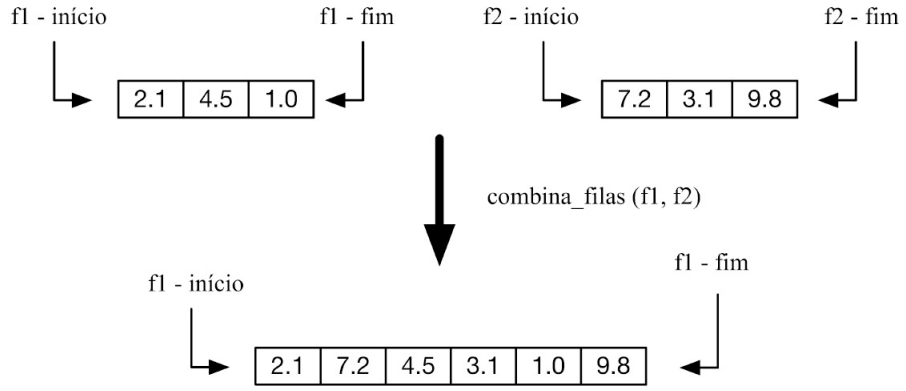


Figura 1: Intercalação de listas

17. Escreva uma função que remova de uma lista encadeada um nó cujo conteúdo tem o valor máximo.

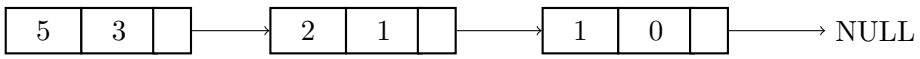
18. Escreva uma função para remover elementos repetidos de uma lista encadeada.

19. Escreva uma função que retorna a quantidade de vezes que o elemento x aparece na lista l.

20. Escreva uma função que conta e retorna a quantidade de números primos em uma lista l de números inteiros.

21. Escreva uma função que decida se duas listas dadas tem o mesmo conteúdo (Os elementos de uma lista devem ser iguais aos da outra lista, na mesma ordem).

22. Polinômios podem ser representados por meio de listas, cujos nós são registros com 3 campos: coeficiente, expoente e referência ao seguinte. Por exemplo, o polinômio $5x^3 + 2x - 1$ seria representado por:



    (a) Criar a lista de polinômios, inserindo os elementos em ordem decrescente pelo expoente do polinômio.

    (b) Somar polinômios. A função recebe os ponteiros para o polinômio P1 e P2 e cria a lista S, a qual representa a soma dos polinômios P1 e P2. Exemplo de soma entre polinômios:

$$(4x^2 - 10x - 5) + (6x + 12) = 4x^2 - 4x + 7.$$

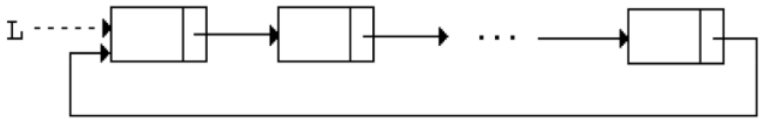
    (c) Escrever um programa principal que leia os polinômios, crie as correspondentes listas representando-os e faça as chamadas à função soma e imprima o resultado.

23. Escreva uma função que inverta a ordem dos nós de uma lista encadeada (o primeiro passe a ser o último, o segundo passe a ser o penúltimo etc.) Faça isso sem criar novos nós; apenas altere os ponteiros.

24. Implemente uma função para criar uma cópia de uma lista simplesmente encadeada. Essa função deve obedecer ao protótipo: `List* copy();`

Listas Encadeadas Circulares e Duplamente Encadeadas

25. Uma *lista encadeada circular* é uma lista encadeada cujo último elemento aponta para o primeiro:



Implemente métodos para:

(a) Contar o número de elementos numa lista circular;

(b) Inserir um elemento à esquerda da cabeça da lista;

(c) Concatenar duas listas circulares;

(d) Eliminar o elemento de valor x;

(e) Intercalar duas listas ordenadas;

(f) Fazer uma cópia da lista.

26. Implemente uma nova versão da lista encadeada circular com um "nó cabeça" como mostrado na Figura 2.

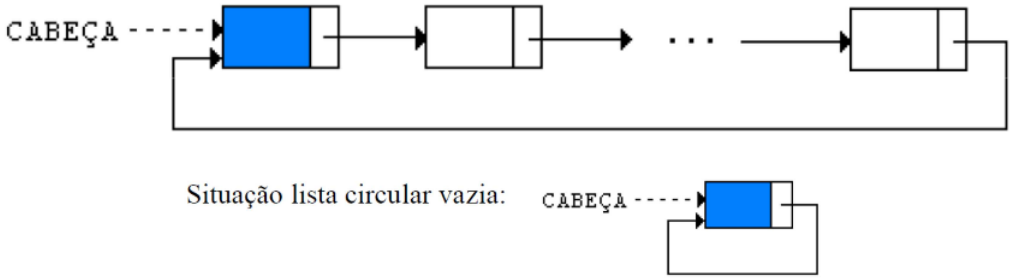


Figura 2: Lista encadeada circular com nó cabeça.

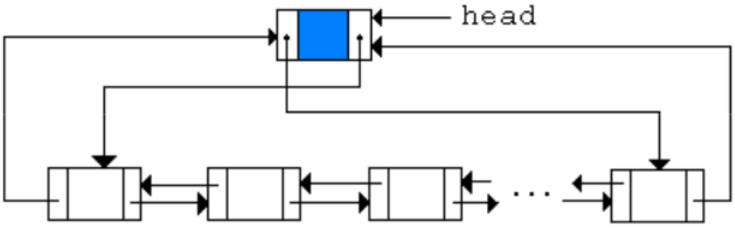
27. Numa *lista duplamente encadeada*, cada nó contém o endereço do nó anterior e o do nó seguinte. O campo `ant` do primeiro nó aponta para `NULL` e o campo `prox` do último nó aponta para `NULL`. Com relação às **listas duplamente encadeadas** implemente:

(a) Um método para concatenar duas listas.

(b) Um método que separa uma lista em duas novas listas.

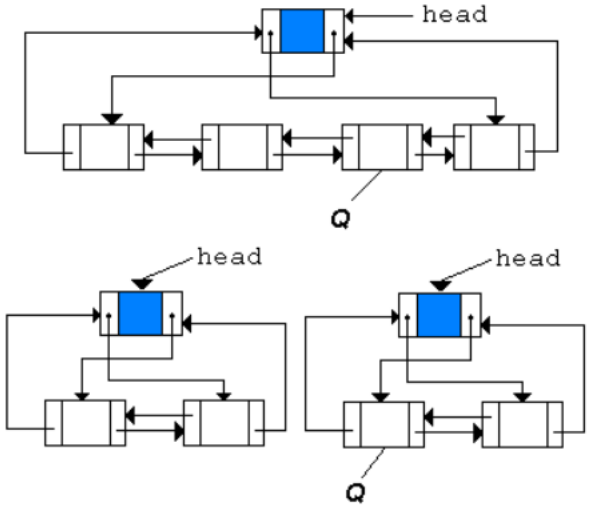
(c) Um método para intercalar duas listas ordenadas em uma lista ordenada.

28. Uma *lista circular duplamente encadeada* é uma lista encadeada circular em que cada nó aponta para o seu predecessor e seu sucessor na lista (o "último" elemento aponta para o "primeiro" e vice versa). Além disso, ainda pode existir um nó auxiliar chamado *nó cabeça*, que aponta para o "primeiro" e o "último" registro da lista e é apontado por eles. Implemente métodos para busca, inserção e eliminação de elementos para este tipo de lista duplamente encadeada.



29. Generalize a lista circular do exercício 17 para **Lista Circular Duplamente Encadeada**, e repita o itens (a) até (f).

30. Escreva um método `quebra` do TAD lista circular duplamente encadeada, que recebe o endereço de um nó `Q` da lista e devolve uma lista duplamente encadeada circular com os elementos a partir de `Q`, como mostrado na figura abaixo.



31. Faça o exercício anterior considerando uma lista circular encadeada.

32. (Problema de Josephus) Imagine n pessoas dispostas em círculo. Suponha que as pessoas estão numeradas de 1 a n no sentido horário. Começando com a pessoa de número 1, percorra o círculo no sentido horário e elimine cada m-ésima pessoa enquanto o círculo tiver duas ou mais pessoas. (Veja *Josephus problem* na Wikipedia.) Qual o número do sobrevivente? Escreva e teste uma função que resolva o problema.

33. Discuta vantagens e desvantagens de vetores em relação a listas encadeadas (implementadas num vetor ou dinamicamente). Dê atenção especial as questões de quantidade de memória, velocidade de inserção, remoção e acesso.