Exercícios - LISTAS

1. Construa o TDA-LDSE com as funcionalidades já discutidas sem sala:

```
int cria(ppLDSE pp, int tamInfo);
void reinicia(pLDSE p);
void destroi(ppLDSE pp);
int tamanho(pLDSE p);
int testaVazia(pLDSE p);
int insereNovoPrimeiro(pLDSE p, void *novo);
int insereNovoUltimo(pLDSE p, void *novo);
int insereNaPosLog (pLDSE p, void *novo,unsigned int posLog);
int buscaNaPosLog(pLDSE p, void *reg, unsigned int posLog);
int buscaUltimo(pLDSE p, void *reg);
int buscaPrimeiro(pLDSE p, void *reg);
int removeDaPosLog(pLDSE p, void *reg, unsigned int posLog);
int removeUltimo(pLDSE p, void *reg);
int removePrimeiro(pLDSE p, void *reg);
```

2. Construa a LDDE com as funcionalidades solicitadas na questão anterior.

3. Complete a implementação das operações da LESE (remoções, buscas, inserções, reiniciação, destruição, etc...).
   Interface Pública:

```
int cria(ppLista pp, int tamanho);
void destroi(ppLista pp);
void purga(pLista p);
int testaVazia(pLista p);
int testaCheia(pLista p);
int buscaPrimeiro(pLista p, void * reg);
int buscaUltimo(pLista p, void * reg);
int buscaNaPosLog(pLista p, void * reg, int posLog);
int removePrimeiro(pLista p);
int removeUltimo(pLista p);
int removeDaPosLog(pLista p, int posLog);
int inserePrimeiro(pLista p, void * reg);
int insereUltimo(pLista p, void * reg);
int insereNaPosLog(pLista p, void * reg, int posLog);
int tamanho(pLista);
```

   Privativo:

```
int obtemNoh(pLista *p );
int devolveNoh(pLista *p, int posicao);
```

4. Implemente uma LEDE - Lista Estática Duplamente Encadeada, ela é semelhante a uma LESE onde cada nó possui um campo extra de ligação com o antecessor.

5. Implemente uma estrutura de matriz dinâmica utilizando um TDA Multi-Lista. Abstraia as operações necessárias para esta estrutura do conceito matemático de matriz.

6. Construa uma LDDE com descritor que possui referência para o início e final da lista, as operações realizadas em posição especificada na sequência da lista devem ser aproveitar essa característica. Especialmente a busca, inserção e remoção em posição específica, devem acessar o nó alvo pelo menor percurso possível, a partir do início ou do final da lista.

7. Construa uma LDDE com descritor que possui referência móvel para a lista (ponteiro não fixo no início da lista). As operações devem ser realizadas de maneira que o referido ponteiro se desloque o mínimo possível.

8. Refaça a questão anterior de forma que a lista seja circular.

9. Construa uma operação de inserção em ordem como uma aplicação de uma LDDE.

10. Reimplemente a LDDE disponibilizando a opção e inserção em ordem como operação interna da LDDE. Você precisará de uma função de *callback*, semelhante ao caso da fila de prioridade.

11. Implemente um TDA Pilha e um TDA Fila, utilizando/aplicando um TDA Lista. Tanto estático quanto dinâmico.

12. Uma fila convencional permite alterações apenas pelas suas extremidades, de maneira que as inserções ocorrem pela cauda e remoções pela frente. Uma *Double-Ended Queue* é similar a uma fila comum porém permite inserções e remoções tanto pela cauda quanto pela frente. Implemente uma *Double-Ended Queue* como uma aplicação de uma LDDE.

13. A Busca-Binária é um método clássico de pesquisa realizável sobre uma coleção de dados ordenados. Diferentemente da Busca Sequencial, onde a pesquisa é realizada como ocorreria em uma fita de vídeo, a Busca Binária é similar a uma pesquisa em um catálogo ordenado. Na Busca Binária define-se um caminho de pesquisa não seqüencial, de forma que a coleção ordenada é repetidamente "quebrada" em partes cada vez menores, até que seja encontrado o item procurado ou esgotem-se as partições, sem a detecção do item.

A Tabela 2 exibe uma “struct” que representa um registro de dados. Crie um vetor REG, de tamanho igual a 500 (utilize o arquivo txt disponível no moodle), contendo instâncias desta “struct” ordenadas pelo campo “chave”, a qual identifica unicamente cada registro de dados.

Crie um vetor auxiliar AUX contendo 200 valores inteiros entre 0 e 499, aleatórios e sem repetição.

```
#include “stdlib.h”
#incluide “time.h”

#define RAND_MAX ...

void srand(time(NULL));

x = rand()%(RAND_MAX);
```

Tabela 1: Gerando numeros aleatórios no intervalo definido entre 0 e RAND_MAX-1.

a) Realize uma busca sequencial sobre REG para cada chave contida em AUX, contabilizando o número de comparações realizadas, ao final calcule a média dessas comparações.

b) Realize uma Busca Binária sobre REG para cada chave contida em AUX, contabilizando o número de comparações realizadas, ao final calcule a média dessas comparações.

c) Após isso, adapte o código abaixo (Tabela 2) para realizar uma Busca Binária sobre uma lista encadeada, faça esta adaptação para:

c.1) LDDE-Referência-Móvel e
c.2) LDDE-Circular.

Contabilize o número de comparações realizadas, ao final calcule a média dessas comparações para cada caso.

Compare as médias obtidas para as buscas sequenciais e Buscas Binárias. Qual o método de busca foi mais rápido? Para qual estrutura de dados?

```
typedef struct
{
 int matricula;
 char nome[tamString];
 int telefone;
 float salario;
 int idade;
 char departamento[tamString];
} Registro;
```

```
reg *BuscaBin(reg A[ ], int chave, int inicioPart, int fimPart)
{  int metade=0, ret=0;
   reg *pt=NULL;

   if (inicioPart > fimPart)
          return NULL;
   metade = (inicioPart + fimPart)/2;
   if(chave == A[metade].chave)
   {     pt=(reg *)malloc (sizeof(reg));
         memcpy(pt, &A[metade], sizeof(reg))
         return pt;
   }
   else
   { if(x < A[metade])
         ret = BuscaBin(A, chave,inicioPart, metade -1);
     else
         ret = BuscaBin(A, chave, metade + 1, fimPart);
     return ret;
   }
}
```

Tabela 2: Exemplo de implementação da Busca-Binária.

14. Discuta a implementação de uma busca sequencial sobre: a) uma lista encadeada cujo descritor aponta para o primeiro item da sequência e b) uma lista encadeada com referencial móvel para um item na lista. Tente determinar o custo computacional para uma busca no pior caso (o item procurado não existe na lista).

15. Discuta a implementação do dicionário (estrutura de dados voltada à operação de busca) utilizando o conceito de Hashing aplicado sobre uma lista encadeada [1]. Faça isso para: a) uma lista encadeada cujo descritor aponta para o primeiro item da lista e b) uma lista encadeada com referencial móvel para um item na lista. Tente determinar o custo computacional para uma busca no pior caso (o item procurado não existe na lista).

[1] ZIVIANI, Nivio. **Projeto de algoritmos:** com implementações em Pascal e C. São Paulo: Pioneira, c1993. 267 p. :

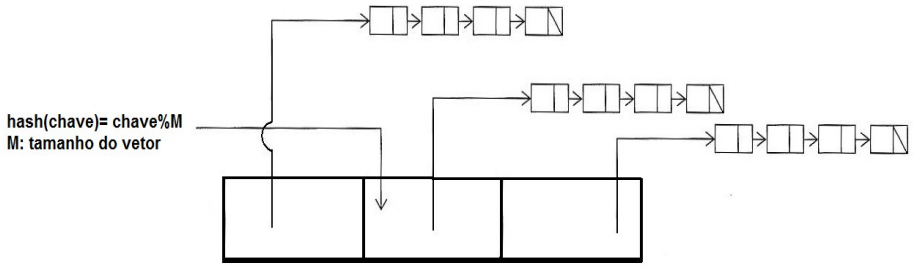

Figura 1: dicionário Hashing usando lista encadeada.

16. Um polinômio inteiro é uma função da forma abaixo, onde todos os expoentes são naturais e os coeficientes (a_i) são inteiros.

$$p(x) = \sum_{i=0}^{m} a_i \ x^i = a_m \ x^m + a_{m-1} \ x^{am-1} + \ldots + a_1 x^1 \ 1 + a_0 \ x^0$$

Em termos de estruturas de dados, um polinômio pode ser apresentado por uma lista de itens onde cada nó possui (como informação) o coeficiente (a_i) **não nulo** e o respectivo expoente.

Sabendo disso, pede-se uma função *int polinômio (...)* para um módulo *cliente* do TDA, que cria a representação computacional de um polinômio P(x).

Considere o TDA já implementado com as operações discutidas em sala. Operações adicionais deverão ser implementadas.

17. Considere as seguintes regras de derivação aplicáveis a polinômios:
(I) A derivada de um polinômio corresponde à soma das derivadas dos seus termos. Portanto, se $p(x) = f_m(x) + f_{m-1}(x) + ... \ f_1(x) + f_0(x)$ e:

$$f_m(x) = a_m x^m \qquad f_{m-1}(x) = a_{m-1} x^{m-1} \qquad ... \qquad f_1(x) = a_1 x^1 \quad f_0(x) = a_0 x^0$$

Então $p'(x) = f'_m(x) + f'_{m-1}(x) + ... + f'_1(x) + f'_0(x)$, onde $p'(x)$ corresponde à derivada do polinômio $p(x)$. Perceba que $p'(x)$ é também um polinômio.

(II) Se $f(x) = b \ x^m$, então $f'(x)$ será definida conforme abaixo:
(i) Se $m \ \ 0, f'(x) = m.b \ x^{m-1}$
(ii) Se $m = 0, f'(x) = 0$.
Onde f '(x) é a derivada de f(x).
Para um cliente de um TDA-lista, pede-se a função que recebe um polinômio e retorna a sua derivada.

Considerando o P(x) representado conforme estratégia já descrita, implemente uma operação de aplicação que soma dois polinômios (sem destruí-los) e fornece o polinômio resultante como uma nova lista. Faça o mesmo para o produto entre 2 polinômios. Faça o mesmo para a divisão entre 2 polinômios.

Considere o TDA já implementado com as operações discutidas em sala. Operações adicionais deverão ser implementadas.

18. Construa uma representação para um grafo (Figura 2) a partir de uma ML cujo encadeamento entre as listas ("espinha dorsal") seja também uma lista dinâmica. Reaproveite ao máximo os TDA's já implementados.

19. Construa uma representação para uma matriz esparsa, para isso utilize uma ML cuja "espinha dorsal" seja um vetor. Reaproveite ao máximo os TDA's já implementados.

20. Refaça as questões 18 e 19 como novos TDAs sem reaproveitar o código já desenvolvido.

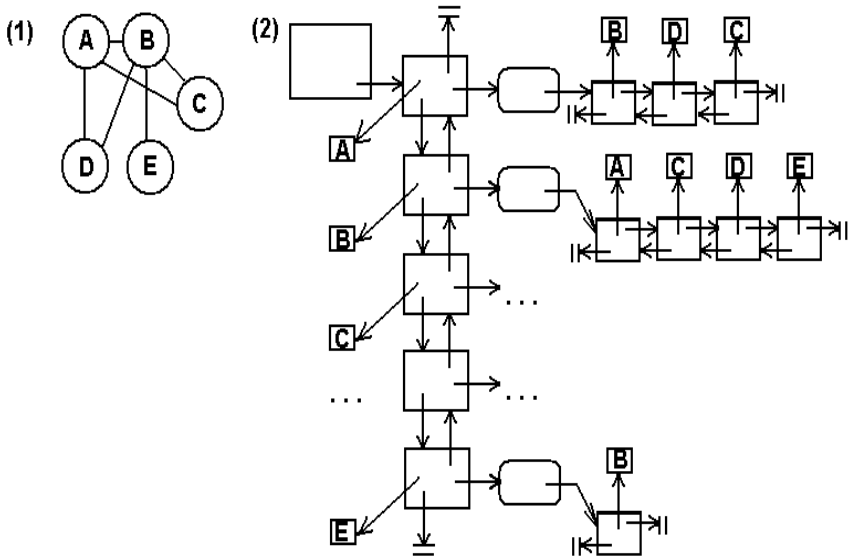

Figura 2: Um Grafo e sua representação através da ML.