Exercícios - LISTAS

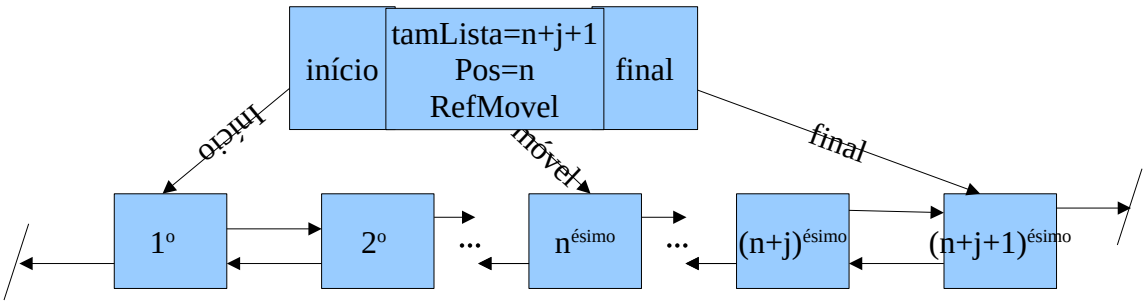
1. Construa o LSE com as funcionalidades já discutidas sem sala:
   ```
   struct descLSE * cria(int tamInfo);
   int tamanhoDaLista(struct descLSE *p);
   int reinicia(struct descLSE *p);
   struct descLSE * destroi(struct descLSE *p);
   int insereNaPoslog(int posLog, info *novo, struct descLSE *p);
   int insereNovoUltimo(info *reg, struct descLSE *p);
   int insereNovoPrimeiro(info *reg, struct descLSE *p);
   int buscaOultimo(info *reg, struct descLSE *p);
   int buscaOprimeiro(info *reg, struct descLSE *p);
   int buscaNaPoslog(int posLog, info *reg, struct descLSE *p);
   int removeDaPoslog(int Poslog, info *reg, struct descLSE  *p);
   int removeOultimo(info *reg, struct descLSE *p);
   int removeOprimeiro(info *reg, struct descLSE *p);
   int testaVazia(struct descLSE *p);
   int inverte(struct descLSE *p);
   struct descLSE * destroi(struct descLSE *p);
   ```

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

3. Complete a implementação das operações da LESE (remoções, buscas, inserções, reiniciação, destruição, etc...).
   Interface Pública:
   ```
   struct descLESE * cria(int tamanhoVetor, int tamInfo);
   struct descLESE * destroi(struct descLESE *p);
   int reinicia(struct descLESE *p);
   int testaVazia(struct descLESE *p);
   int testaCheia(struct descLESE *p);
   int buscaOprimeiro(struct descLESE *p, info *pReg);
   int buscaOultimo(struct descLESE *p, info *pReg);
   int buscaNaPosLog(struct descLESE *p, info *pReg, int posLog);
   int removeOprimeiro(struct descLESE *p, info *pReg);
   int removeOultimo(struct descLESE *p,info *pReg);
   int removeDaPosLog(struct descLESE *p, info *pReg, int posLog);
   int insereNovoPrimeiro(struct descLESE *p,info *pReg);
   int insereNovoUltimo(struct descLESE *p, info *pReg);
   int insereNaPosLog(struct descLESE *p,info *pReg, int posLog);
   int tamanhoDaLista(struct descLESE *p);
   ```
   Privativo:
   ```
   int obtemNoh(pLista *p );
   int devolveNoh(pLista *p, int posicao);
   ```

4. Implemente uma função que calcula o tamanho do vetor que contém uma LESE - Lista Estática Simp. Encadeada. Lembre-se que o tamanho do vetor não consta no descritor dessa est. de dados.
5. 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.

6. Construa uma LDE com descritor que possui:
   i) Referência (ponteiro) para o início da lista;
   ii) Referência (ponteiro) para o final da lista;
   iii) Referência móvel (ponteiro não fixado) para um nó da lista;
   iv) Anotação da posição (pos) atualmente apontada pelo referencial móvel;
   vi) Anotação do tamanho da lista.



Sobre a o referencial móvel:
a) Ele é posicionado sobre um nó recém inserido;
b) Ele é posicionado sobre um nó recém buscado;
c) Ele é posicionado sobre um vizinho adjacente ao item recém removido.

As operações visando uma posição-alvo especificada devem aproveitar essa configuração da lista (ponteiro para o início, final e referencial móvel). Especialmente a busca, inserção e remoção em posição específica, devem acessar o nó alvo pelo menor percurso possível, a partir do início, ou a partir do ref. móvel, ou a partir do final da lista.

O menor percurso é calculado com base nas distâncias:
D1 = módulo da distância entre a posição do referencial móvel em relação ao início (1ª posição) da lista;
D2 = módulo da distância entre a posição do referencial móvel e o final da lista (tamanho da lista).

Exemplo considerando um alvo na posição X:
Se (1 <= X <= pos) então:
    Se (D1/2 for mais próximo do início) percorra a partir do início;
    Senão: percorra a partir do ref. móvel
Se (pos <= X <= tamLista) então:
    Se (pos+(D2/2) for mais próximo do tamLista) percorra a partir do final;
    Senão, percorra a partir do ref. móvel

7. Construa uma LDE 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 LDE.

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

11. Implemente uma Pilha e uma Fila, utilizando/aplicando Listas. 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 LDE.
13. Discuta a implementação do dicionário (estrutura de dados voltada à operação de busca, Figura 1) 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.

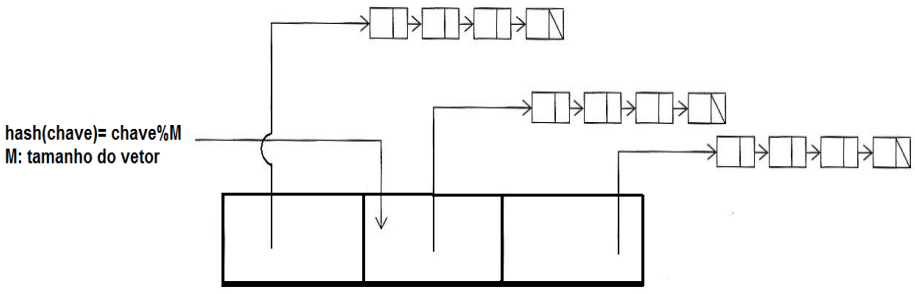


Figura 1: dicionário Hashing usando lista encadeada.

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

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

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) na forma de uma lista.

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

15. 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 $p(x) = f_0(x) + f_1(x) + f_2(x) \cdots + f_{m-1}(x) + f_m(x)$

e $f_0(x)=a_0 x^0;\quad f_1(x)=a_1 x^1;\quad \cdots\quad f_{m-1}(x)=a_{m-1}\quad e\quad f_m(x)=x^{m-1}+a_m x^m$

Então a derivada do polinômio será:

$$p'(x) = f'_0(x) + f'_1(x) + f'_2(x) \cdots + f'_{m-1}(x) + f'_m(x)$$

Perceba que $p'(x)$ é também um polinômio e que $f'(x)$ é definida da seguinte forma: se $f(x) = b\ x^m \rightarrow f'(x) = m.b\ x^{m-1}$

Para um cliente de uma 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 a LDE já implementada com as operações discutidas em sala. Operações adicionais deverão ser implementadas.

16. 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).
17. 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.

O Quadro 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.

```
...
#define RAND_MAX ...
...
void srand(time(NULL));
x = rand()%(RAND_MAX);
```

Quadro 1: Gerando números 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 (Quadro 2) para realizar uma Busca Binária sobre uma lista encadeada, faça esta adaptação para:

c.1) LDE-Referência-Móvel e
c.2) LDE-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;
   }
}
```

Quadro 2: Exemplo de implementação da Busca-Binária sobre um vetor.

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 as est. de dados já implementadas.

19. Construa uma representação para uma matriz esparsa, para isso utilize uma ML cuja “espinha dorsal” seja um vetor. Reaproveite ao máximo as est. de dados já implementadas.

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

21. Refaça as questões 18 e 19 como novas est. de dados sem reaproveitar o código já desenvolvido.

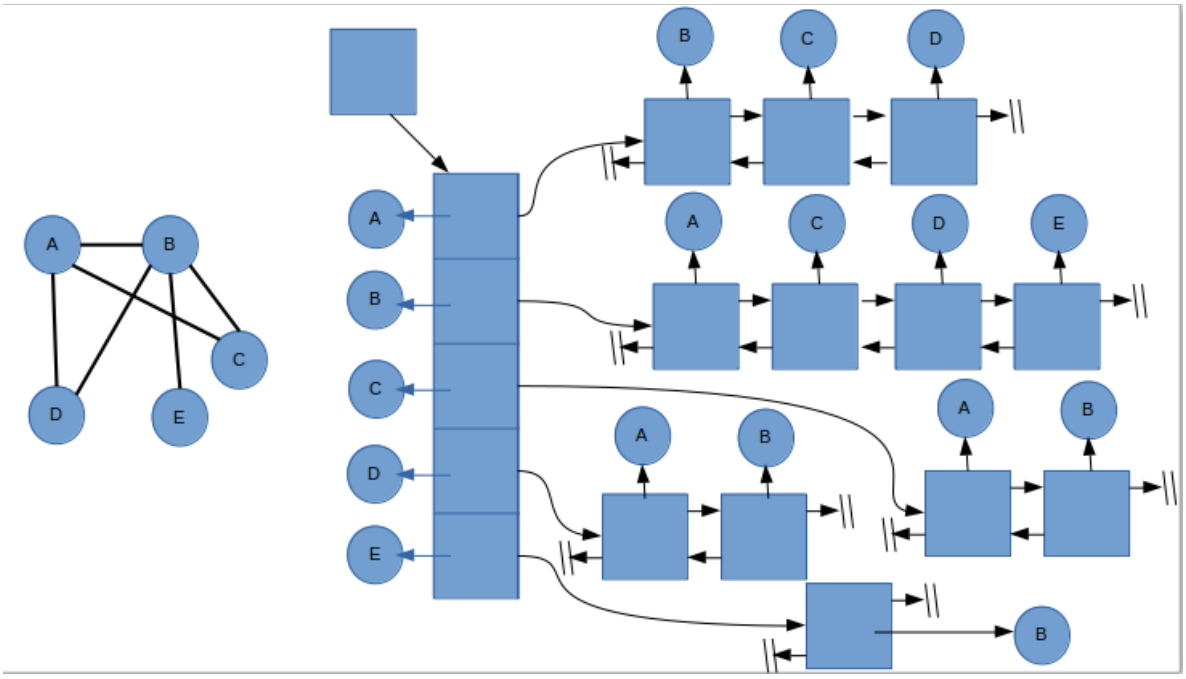


Figura 2: Um Grafo e sua representação através da ML.