**UNIVERSIDADE DE SÃO PAULO**
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
I.C.M.C.

**SCC-201 Algoritmos e Estruturas de Dados I (Mat. Apl.)**
**Profa. Graça Nunes**
**2º. Semestre de 2010**

**2ª. Prova (Gabarito)**
**22/10/2010**

Nome: _____ Nro USP: _____

1) Considere a definição de tipo de uma lista encadeada dinâmica:

```
struct list_rec {
     tipo_elem info;
     struct list_rec *lig;
};
typedef struct list_rec  Rec;
```

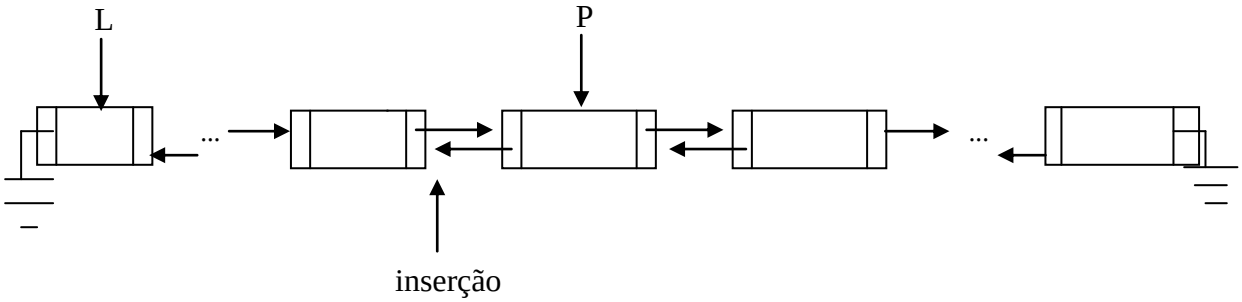
(a) (2.0) Escreva uma função **recursiva** que busca por uma chave **x** numa lista encadeada **ordenada** crescentemente. Ela deve retornar NULL se não encontrar, ou o endereço do registro, se encontrar:

```
Rec* Busca (tipo_elem x, Rec *L)
{ if (L==null) return null;
  if (L->info == x) return L;
  if (L->info > x) return null;
  return Busca (x, L->lig);
}
```

(b) (1.0) Discuta sobre a complexidade da sua função de busca e sobre vantagens ou desvantagens do fato de ela ser recursiva.

Não há vantagens em se fazer uma busca recursiva, pois esta tem o custo do espaço gasto pela pilha de recursão – que é O(tamanho_da_lista) – e tem a mesma complexidade de tempo da versão não recursiva, ou seja, O(tamanho_da_lista), enquanto que a não recursiva não tem gasto adicional de memória.

2) Podemos estender a noção de lista encadeada, aumentando seus registros com um campo de ligação adicional, de maneira que, agora, cada elemento tenha acesso não somente ao seu sucessor na lista, mas também ao seu antecessor. Dessa forma, temos o campo de ligação à direita (liga com seu sucessor) e o campo de ligação à esquerda (liga com seu antecessor). Tais listas são chamadas de listas duplamente encadeadas. Considere o esquema gráfico abaixo, de uma lista duplamente encadeada dinâmica L:




Pede-se:

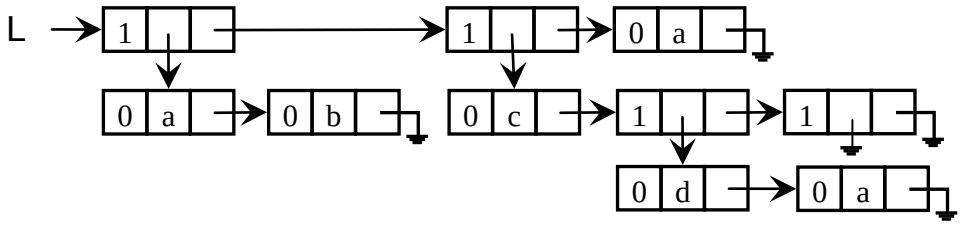
a) (1.0) Faça a declaração de tipos, em C, necessária para a definição de uma lista duplamente encadeada.

```
typedef struct list_rec {
        tipo_elem info;
        struct list_rec *ligesq, *ligdir;
}Rec;
Rec *L;
```

b) (2.0) Escreva os comandos para inserir um registro com chave x à esquerda de um registro cujo endereço é dado por P. Considere que P não é NULL e que, se P coincidir com L, o novo L deve ser o registro inserido.

```
Rec *q = (Rec*) malloc (sizeof (Rec));
q->info = x;
q->ligdir = P;
q->ligesq = P->ligesq;
if (P==L)   L=q;
else P->ligesq->ligdir = q;
P->ligesq->q;
```

3) (1.5) Considere uma lista generalizada. A figura abaixo é um exemplo de representação da lista generalizada L= [[a,b],[c,[d,a],[]],a].



Considere a declaração de tipo abaixo e responda o que faz a função **surpresa** a seguir. A declaração básica do nó já é dada abaixo.

```
typedef struct bloco {
        union {
                char atomo;
                struct bloco *sublista;
        } info;
        int tipo;
        struct bloco *prox;
} no;
no *L;

   int surpresa(char x, no *L) {
   if (L == NULL) return 0;
   if (L->tipo == 0){
        if (L->info.atomo == x)
        return (1 + surpresa(x, L->prox));
        else return (surpresa(x, L->prox);
        }
   return (surpresa(x, L->info.sublista) + surpresa(x, L->prox));
   }
```

Resp.: Retorna o número de vezes que um átomo **x** ocorre na lista. Ele pode ocorrer em qualquer nível da lista. Por exemplo, o átomo "a" ocorre 3 vezes na lista L acima.

**4)** Considere a matriz esparsa abaixo:

$$\begin{pmatrix} 0 & 0 & 2 \\ 3 & -1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

(a) (1.0) Mostre, graficamente, como ela seria representada com listas cruzadas.

(b) (1.0) Quais as vantagens e desvantagens dessa representação com listas cruzadas em relação à matriz bidimensional tradicional? Faça considerações em termos de tempo para manipulação dos dados e espaço de memória utilizado. Lembre-se dos conceitos de complexidade de algoritmos!
Enquanto que o array bidimensional que representa uma matriz n x m ocupa O(n * m) espaço de memória para representar todos os n * m elementos, as listas cruzadas representam apenas os elementos diferentes de zero. Se eles forem em numero bem maior do que os não nulos, haverá economia de espaço. Como desvantagem, entretanto, não há mais acesso direto a cada elemento (i,j) da matriz. Para isso, é necessário percorrer uma lista encadeada (de linha ou coluna). Além disso, operações com matrizes ficam muito mais complexas nas listas cruzadas.

(c) (0.5) Valeria a pena usar a representação com listas cruzadas para a matriz anterior? Justifique.
Se cada elemento não nulo ocupa 5 posições de memória, e há 4 elementos não nulos, então 20 posições são gastas para representá-los (sem contar os arrays de ponteiros para as listas

encadeadas de linha e coluna!). Como na matriz representamos apenas 12 elementos (n=4; m=3), conclui-se que não há vantagens neste caso.