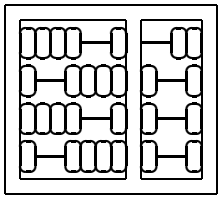

MC-102 Algoritmos e Programação de Computadores
IC-UNICAMP
Aula 28 - Listas

1 Objetivos

Apresentar a estrutura de listas dinâmicas.

2 Motivação

Listas podem representar qualquer seqüência de elementos, permitindo operações mais genéricas que vetores. Além disso, elas possuem tamanho dinâmico, necessitando apenas alocar ou desalocar o espaço para cada elemento da lista.

3 Listas Encadeadas

Uma lista encadeada é uma representação de uma seqüência de objetos na memória do computador. Cada elemento da seqüência é armazenado em um nodo (*node*) da lista: o primeiro elemento no primeiro nó, o segundo no segundo e assim por diante. Ao contrário de vetores, para acessar o i-ésimo elemento é necessário percorrer todos os elementos anteriores.

3.1 Estrutura de uma lista encadeada

Uma lista encadeada (*inked list* ou lista ligada) é uma seqüência de nodos; cada nodo contém um objeto de algum tipo e o endereço do nodo seguinte. Suporemos nesta página que os objetos armazenados nos nodos são do tipo `int`. A estrutura de cada nodo de uma lista pode ser definida assim:

```
struct _nodo {
   int          info;
   struct _nodo *prox;
};
```

Figura 1: Estrutura do Nodo

É conveniente tratar os nodos como um novo tipo-de-dados e atribuir um nome a esse novo tipo:

`typedef struct _nodo Nodo;`

Um nodo `c` e um ponteiro `p` para um nodo podem ser declarados assim:

```
Nodo  c;
Nodo *p;
```

Se `c` é um nodo então `c.info` é o conteúdo do nodo e `c.prox` é o endereço do próximo nodo. Se `p` é o endereço de um nodo, então `p->info` é o conteúdo do nodo e `p->prox` é o endereço do próximo nodo. Se `p` aponta para o último nodo da lista então

`p->prox == NULL.`

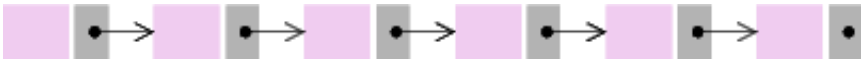

Figura 2: Exemplo de lista

3.2 Endereço de uma lista encadeada

O endereço de uma lista encadeada é o endereço de seu primeiro nodo. Se `p` é o endereço de uma lista, convém, às vezes, dizer simplesmente **"p é uma lista"**.

Listas são estruturas eminentemente recursivas. Para tornar isso evidente, basta fazer a seguinte observação: se `p` é uma lista então vale uma das seguintes alternativas:

* `p == NULL` ou

* `p->prox` é uma lista.

A lista é dita **vazia** se o endereço de seu primeiro nodo é `NULL`. Para criar uma lista vazia basta fazer

```
Nodo *ini;
ini = NULL;
```

Exemplos

Eis como se imprime o conteúdo de uma lista encadeada:

```
// Imprime o conteudo de uma lista encadeada.
// O endereco do primeiro nodo e ini.

void imprima (Nodo *ini)
{
    Nodo *p=ini;
    while(p!=NULL){
        printf ("%d\n", p->info);
        p=p->prox;
    }
}
```

Ou posso usar o `for` também:

```
void imprima_for (Nodo *ini)
{
   Nodo *p;
   for (p = ini; p != NULL; p = p->prox)
      printf ("%d\n", p->info);
}
```

3.3 Busca em uma lista encadeada

Veja como é fácil verificar se um inteiro x pertence a uma lista encadeada, ou seja, se é igual ao conteúdo de alguma nodo da lista:

```
 /* Esta funcao recebe um inteiro x e uma lista
    encadeada de inteiros. O endereco da lista e
    ini. A çãfuno devolve o endereco de um Nodo
    que contem x. Se tal Nodo nao existe,
    a funcao devolve NULL. */

Nodo *busca (int x, Nodo *ini)
{
   Nodo *p;
   p = ini;
   while (p != NULL && p->info != x)
      p = p->prox;
   return p;
}
```

Que beleza! Nada de variáveis booleanas! A função se comporta bem até mesmo quando a lista está vazia.

Eis uma versão recursiva da mesma função:

```
Nodo *busca2 (int x, Nodo *ini)
{
   if (ini == NULL)
      return NULL;
   if (ini->info == x)
      return ini;
   return busca2 (x, ini->prox);
}
```

3.4 Inserção em uma lista

Quero inserir um novo nodo com contéudo `x` entre a posição apontada por `p` e a posição seguinte (**por que seguinte e não anterior?**) em uma lista encadeada. É claro que isso só faz sentido se `p != NULL`.

```
// Esta funcao insere um novo Nodo em uma
```

```
// lista encadeada. O novo Nodo tem info
// x e e' inserido entre o Nodo apontado por
// p e o seguinte.

void insere (int x, Nodo *p)
{
  Nodo *novo;
  if (p!=NULL){
    novo = (Nodo *)malloc (sizeof (Nodo));
    novo->info = x;
    novo->prox = p->prox;
    p->prox = novo;
  }
}
```

Observe que a função se comporta corretamente mesmo quando quero inserir no fim da lista, isto é, quando `p->prox == NULL`. Infelizmente, a função não é capaz de inserir no início da lista.

Exemplo de função para inserir no início da lista. A função abaixo funciona mesmo que a lista esteja vazia.

```
// Esta funcao insere um novo Nodo no
// inicio de uma lista encadeada.

void insere (int x, Nodo **ini)
{
  Nodo *novo;
  novo = (Nodo *)malloc (sizeof (Nodo));
  novo->info = x;
  novo->prox = (*ini);
  (*ini)=novo;
}
```

3.5 Remoção em uma lista

Suponha que quero remover um certo nodo da lista. Como posso especificar o nodo em questão? A idéia mais óbvia é apontar para o nodo que quero remover. Mas é fácil perceber que essa idéia não é boa. É melhor apontar para a nodo anterior à que quero remover. Infelizmente, isso traz uma nova dificuldade: não há como pedir a remoção da primeira nodo.

Vamos supor que `p` é o endereço de um nodo de uma lista e que desejo remover o nodo apontado por `p->prox`. (Note que a função de remoção não precisa saber onde a lista começa.)

```
// Esta funcao recebe o endereco p de um Nodo
// de uma lista encadeada. A funcao remove da
// lista o Nodo p->prox.

void remove (Nodo *p)
{
```

```
  Nodo *apaga;
  if((p!=NULL)&&(p->prox!=NULL)){
    apaga = p->prox;
    p->prox = apaga->prox;
    free (apaga);
  }
}
```

Não é preciso copiar informações de um lugar para outro, como fizemos para remover um elemento de um vetor: basta mudar o valor de um ponteiro. A função consome sempre o mesmo tempo, quer a nodo a ser removido esteja perto do início da lista quer esteja perto do fim.

Exemplo de função para remover o elemento do início da lista.

```
// Esta funcao remove um novo Nodo no
// inicio de uma lista encadeada.

void remove_ini(int x, Nodo **ini)
{
  Nodo * apaga;
  if ((*ini)!=NULL){
    apaga=(*ini);
    (*ini)=(*ini)->prox;
    free(apaga);
  }
}
```

3.6 Busca-e-remoção

Suponha que `ini` é o endereço de uma lista encadeada. Nosso problema: Dado um inteiro y, remover da lista o primeiro nodo que contém y (se tal nodo não existe, não é preciso fazer nada).

```
// Esta funcao recebe uma lista encadeada ini,
// e remove da lista o primeiro Nodo que
// contiver y, se tal Nodo existir.

void buscaEremove (int y, Nodo *ini)
{
   Nodo *p, *q;
   p = NULL;
   q = ini;
   while (q != NULL && q->info != y) {
      p = q;
      q = q->prox;
   }
   if (q != NULL && p!=NULL) {
      p->prox = q->prox;
      free (q);
   }
```

}

Invariante: no início de cada iteração (imediatamente antes da comparação de q com NULL), temos

`q == p->prox` ,

ou seja, `q` está sempre um passo à frente de `p`.

3.7 Busca-e-inserção

Mais uma vez, suponha que tenho uma lista encadeada `ini`. Nosso problema: Inserir na lista um novo nodo com conteúdo x imediatamente antes da primeira nodo que tiver conteúdo y ; se tal nodo não existe, inserir x no fim da lista.

```
// Esta funcao recebe uma lista encadeada ini
// e insere na lista um novo Nodo imediatamente
// antes do primeiro Nodo que contiver y.
// Se nenhum Nodo écontm y, insere o novo
// Nodo no fim da lista. O info do novo
// Nodoé  x.

void buscaEinsere (int x, int y, Nodo *ini)
{
   Nodo *p, *q, *novo;
   novo = (Nodo *)malloc (sizeof (Nodo));
   novo->info = x;
   p = ini;
   q = ini->prox;
   while (q != NULL && q->info != y) {
      p = q;
      q = q->prox;
   }
   novo->prox = q;
   p->prox = novo;
}
```

4 Bibliografia Utilizada

Paulo Feofiloff - Projeto de Algoritmos. disponível em: http://www.ime.usp.br/ pf/algoritmos