Listas Duplamente Encadeadas

Listas Circulares

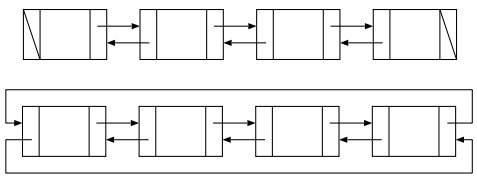
- de qualquer nó é possível atingir qualquer outro nó
- não é possível percorrer a lista no sentido contrário
- não é possível eliminar um nó com o uso de apenas um ponteiro para esse nó

Lista Duplamente Encadeada

- cada nó possui duas referências: o nó predecessor e o nó sucessor
- quando a busca ocorre pelo valor do campo info, ao encontrá-lo podemos inserir/eliminar sem a necessidade de ponteiro auxiliar
- no encadeamento duplo pode ocorrer todo tipo de variação: circular, com nó cabeça, sem nó cabeça, circular com nó cabeça, circular sem nó cabeça, etc

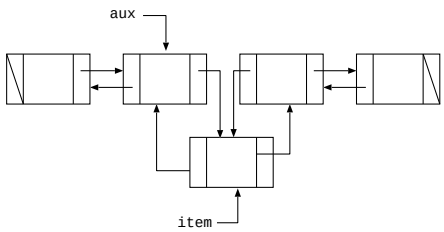
Lista Duplamente Encadeada

```
typedef struct no {
  elem_t info;
  struct no *prox, *ant;
} No_lista;
```



Inserção

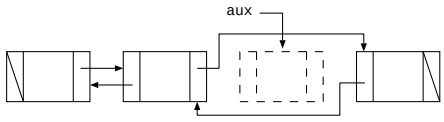
Após percorrer a lista (*item), em que ordem devemos atualizar os ponteiros `ant` e `prox` de cada nó?




```
No_lista *item;
item = (No_lista *)malloc(sizeof(No_lista));

// percorre lista com aux

item->info = e;
item->ant = aux;
item->dir = aux->prox;
aux->prox->ant = item;
aux->prox = item;
```

Remoção

Após percorrer a lista (*item), em que ordem devemos atualizar os ponteiros `ant` e `prox` de cada nó?




```
No_lista *aux;

// percorre lista com aux

aux->ant->prox = aux->prox;
aux->prox->ant = aux->ant;
free(aux);
```

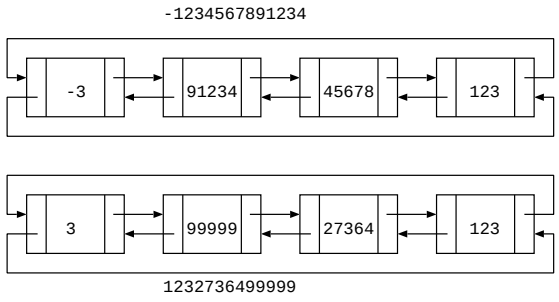
Soma de inteiros longos

- soma de inteiros positivo e negativo
- cabeçalho que indica a quantidade de nós e se é positivo ou não
- subtrai o maior valor absoluto do menor: se ambos tem a mesma quantidade de nós?
- percorrer do dígito mais significativo até o menos significativo para determinar o maior número $\Rightarrow$ lista duplamente encadeada

Exercício

Implemente a soma de dois números grandes com a possibilidade de somar números negativos. Utilize listas duplamente encadeadas.

Exercícios



Listas Genéricas

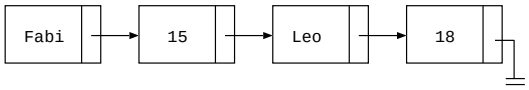
Listas Não-Lineares

Sistema para controle de reservas:

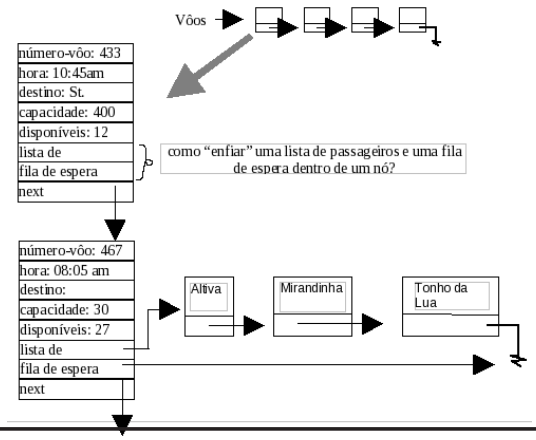
- uma lista de vôos
- para cada vôo, número do vôo, hora, destino, capacidade, lugares disponíveis, lista de passageiros, fila de espera

Listas Genéricas

Um única estrutura para armazenar diferentes informações:



- Como inserir diferentes tipos em uma única lista?



Solução 1

- Vários campos para info
- Usam-se somente os necessários
- Memória alocada desnecessariamente

```
typedef struct no{
  char info1;
  int info2;
  struct no *prox;
} No;
```

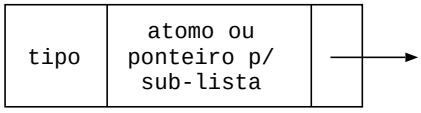
Solução 2

- vários ponteiros
- memória alocada conforme necessidade

```
typedef struct no{
  char *info1;
  int *info2;
  struct no *prox;
} No;
```

Lista Generalizada

Uma lista generalizada A é uma seqüência finita de $n \geq 0$ elementos $\alpha_1, \alpha_2, \ldots, \alpha_n$ em que α_i é átomo ou lista.

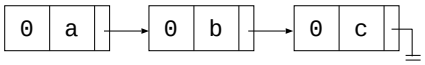


Solução 3

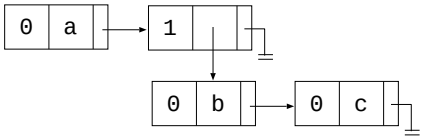
```
enum elem_t {tipo_int, tipo_char};
union info_lista {
  int i;
  char c;
};
typedef struct No {
  enum elem_t tipo;
  union info_lista info;
  struct No* prox;
} No;
```

Exemplos

- $L_1 = (a, b, c)$



- $L_2 = (a, (b, c))$



Lista Generalizada

- pode ter na lista um átomo ou uma lista (sub-lista)
  - átomo (elemento indivisível): inteiro, caracter, string, ponto flutuante, ...
- Exemplo: L = (a,(b,c),d,(e),( ))
  átomos: a, d
  sub-listas: {b,c}, {e}, {}

Exercícios

Desenvolva a representação das listas:

- $L_3 = (a, (b, c), d, (e), ())$
- $L_4 = (a, (b(c, d)), (e))$
- $L_5 = ((a, b), (c, (d, e)), f)$
- $L_6 = (a, (b, c), d, (e, (f, g, (h))))$

Implementação

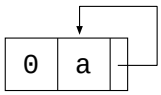
```
enum elem_t {atomo, sublista};

union info_lista {
  int i;
  struct No* sublista;
};

typedef struct No {
  enum elem_t tipo;
  union info_lista info;
  struct No* prox;
} No;
```

Lista generalizada/genérica

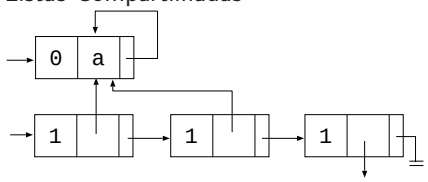
```
enum elem_t {tipo_int, tipo_char, tipo_sublista};
union info_lista {
  int i;
  char c;
  struct No* sublista;
};

typedef struct No {
  enum elem_t tipo;
  union info_lista info;
  struct No* prox;
} No;
```

Variações de Listas Generalizadas

- Listas Recursivas



- Listas Compartilhadas



Compartilhamento pode resultar em grande economia de memória. Entretanto, esse tipo de estrutura cria problemas quando desejamos eliminar ou inserir nós na frente da lista.

Note que, em geral, não se sabe quantos ponteiros estão apontando para a estrutura, e tampouco de onde eles vêm! Ainda que essa informação fosse conhecida, a manutenção seria custosa.

Bibliografia

- Aaron M. Tenenbaum, Yedidyah Langsam, Moshe J. Augenstein *Estruturas de Dados usando C* Macron Books, 2004.
- Thiago A. S. Pardo, material de aula da disciplina *Algoritmos e Estruturas de Dados I* ICMC - USP - São Carlos
- Roberto Ferrari, *Curso de estruturas de dados*, São Carlos, 2006. Apostila disponível em: `http://www2.dc.ufscar.br/~bsi/materiais/ed/`