Algoritmos e Estrutura de Dados

Listas dinâmicas com sentinela

Prof. Nilton Luiz Queiroz Jr.

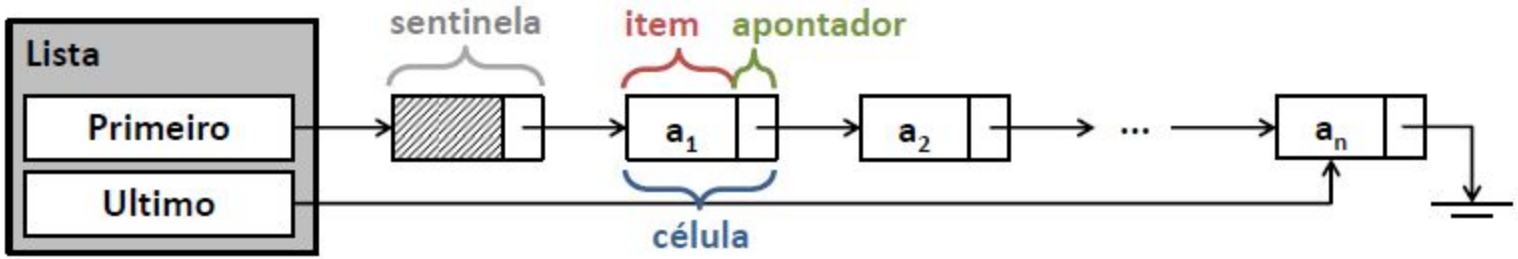
Listas Ligadas

- Listas simplesmente ligadas tem alguns casos específicos;
- Uma alternativa para simplificar alguns desses problemas é usar uma sentinela;
  - Torna genérico o procedimento/função de inserção no fim;

Sentinelas

- Uma sentinela é um objeto fictício que permite simplificar alguns casos de lista
  - Auxilia a deixar o código mais claro;
- Podem ser vistas como um objeto que representa o valor NULL, porém tem todos os outros atributos dos outros objetos da lista;
- Auxiliam na clareza do código;
  - Raramente melhoram o desempenho assintótico;

Listas Ligadas com sentinela



Lista ligada com sentinela

- Assim como na lista ligada os tipos necessários são:

```
struct tipo_item{
    int chave;
    /*outros campos*/
};
struct tipo_celula{
    struct tipo_item item;
    struct tipo_celula *prox;
};
struct tipo_lista{
    struct tipo_celula *primeiro;
    struct tipo_celula *ultimo;
};
```

Lista ligada com sentinela

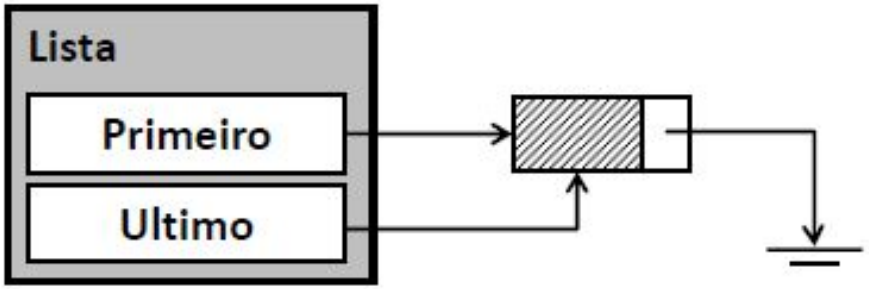
- A inicialização de uma lista com sentinela é diferente da inicialização de uma lista sem sentinela;
  - É necessário criar a sentinela ao inicializar a lista;

```
void inicializa(struct tipo_lista *l){
    l->primeiro=(struct tipo_celula *)malloc(sizeof(struct tipo_celula ));
    l->ultimo=l->primeiro;
    l->ultimo->prox=NULL;
}
```

Lista ligada com sentinela

- Para verificar se a lista está vazia é necessário verificar se o único elemento que compõe a lista é a sentinela;

```
int vazia(struct tipo_lista *l){
    return l->primeiro == l->ultimo;
}
```



Lista ligada com sentinela

- Inserção no início
  - É muito similar a inserção em uma lista sem sentinela;

Lista ligada com sentinela

- Inserção no início
  - É muito similar a inserção em uma lista sem sentinela;

```
void insere_primeiro(struct tipo_lista *l, struct tipo_item x){
    struct tipo_celula *novo;
    novo=(struct tipo_celula *)malloc(sizeof(struct tipo_celula));
    novo->item=x;
    if(vazia(l)){
          l->ultimo=novo;
    }
    novo->prox=l->primeiro->prox;
    l->primeiro->prox=novo;
}
```

Lista ligada com sentinela

- A inserção no fim é simplificada;
  - Não precisa verificar se o elemento inserido é o primeiro;

Lista ligada com sentinela

- A inserção no fim é simplificada;
  - Não precisa verificar se o elemento inserido é o primeiro;

```
void insere_ultimo(struct tipo_lista *l, struct tipo_item x){
    struct tipo_celula *novo;
    novo=(struct tipo_celula *)malloc(sizeof(struct tipo_celula));
    novo->prox=NULL;
    novo->item=x;
    l->ultimo->prox=novo;
    l->ultimo=novo;
}
```

Lista ligada com sentinela

- Inserção após a i-ésima posição

```
void insere_pos(struct tipo_lista *l, struct tipo_item x, int pos){
    struct tipo_celula *novo,*ptr;
    int i=0;
    ptr=l->primeiro->prox;
    while((ptr!=NULL) && (i<pos)){
        ptr=ptr->prox;
        i++;
    }
    if(ptr!=NULL){
        novo=(struct tipo_celula *)malloc(sizeof(struct tipo_celula));
        novo->prox=ptr->prox;
        novo->item=x;
        ptr->prox=novo;
        if(novo->prox==NULL)
            l->ultimo=novo;
    }
}
```

Lista ligada com sentinela

- Remoção no início

```
int remove_primeiro(struct tipo_lista *l, struct tipo_item *x){
    struct tipo_celula *ptr;
    if(!vazia(l)){
          ptr=l->primeiro->prox;
          l->primeiro->prox=ptr->prox;
          *x=ptr->item;
          free(ptr);
          if(l->primeiro->prox==NULL){
                l->ultimo=l->primeiro;
          }
          return 1;
    }else{
         return 0;
    }
}
```

Exercícios

1. Implemente a operação de remover um elemento na i-ésima posição de uma lista com sentinela.
2. Implemente a operação de remoção no fim de uma lista ligada com sentinela;
3. Implemente a operação de inserção ordenada em uma lista ligada com sentinela
4. Implemente a operação de exibir todos itens de uma lista ligada com sentinela
5. Implemente uma operação de busca para uma lista ligada com sentinela;

   Lembre-se que a sentinela não conta como um elemento da lista;

Referências

CORMEN, T.H.; LEISERSON,C.E.; STEIN, C.; RIVEST, R.L. Algoritmos: Teoria e Prática. Terceira Edição. Editora Campus, 2011.