Estruturas de Dados

Listas

Baseado no material do prof.
Dr. Ronaldo A. L. Gonçalves

Porf: Nilton Luiz Queiroz Jr.

Listas

- Listas são tipos abstratos de dados;
- São uma das formas mais simples de interligar elementos de um conjunto;
- Armazenam dados de forma alinhada;
    - Um dado após o outro;

Listas

- Uma lista linear é uma sequência de zero ou mais itens:

$$x_1 , x_2 , x_3 , \dots , x_n$$

- Cada elemento x_i é de um determinado tipo;
- O valor n representa o tamanho da lista;

Listas

- É muito comum que esse tipo seja um agregado heterogêneo contendo um campo chave e as informações que se deseja armazenar;
  - É comum que o campo chave não se repita em uma lista, porém em alguns casos isso pode acontecer;
  - Questões como essa devem ser tratadas na implementação dos algoritmos que realizam as operações da lista;

```
struct tipo_item{
    int chave
    /*outros campos*/
};
```

Listas

- É necessário que sejam definidas algumas operações sobre listas;
- O conjunto de operações é dependente da aplicação;
- Operações necessárias na maior parte das aplicações são:
  - Inicializar lista;
  - Inserir um elemento;
  - Remover um elemento;
  - Localizar um elemento;
  - Imprimir a lista;
  - Combinar duas ou mais listas em uma única lista;
  - Pesquisar a ocorrência de um item em algum local particular;

Listas

- Listas podem ser representadas por várias estruturas de dados;
- As formas mais comuns de representar listas são:
  - Utilizando arranjos
    - Listas estáticas;
  - Utilizando ponteiros;
    - Listas dinâmicas;

Listas Estáticas

- São listas que tem seu tamanho limitado;
  - Podem crescer até um certo ponto;
  - Memória alocada estaticamente;
    - Vetor;
  - É importante verificar se a lista não está cheia antes de inserir qualquer elemento
- Em geral usam-se meios para verificar se a lista está cheia, e qual a quantidade de elementos que ela possui;

Lista Estática com Remanejamento

- A lista estática com remanejamento aloca os dados de maneira sequencial:
  - O primeiro elemento da lista está na primeira posição do vetor;
  - O segundo elemento da lista está na segunda posição do vetor;
  - O N-ésimo elemento da lista está na N-ésima posição do vetor;
- Essas listas em geral são um agregado heterogêneo composto por um vetor e uma variável para controlar o tamanho da lista;

Lista Estática com Remanejamento

- A busca na lista estática com remanejamento pode ser feita da seguinte maneira:
  - Inicia-se a busca da primeira posição do vetor;
  - Faz a comparação do conteúdo buscado com a posição atual;
    - Se for o conteúdo buscado, retorna-se a posição;
    - Se não for, segue para o próximo;
    - Caso não seja encontrado o elemento, retorne um valor de índice inválido;

Lista Estática com Remanejamento

- A inserção nesse tipo de lista é feita da seguinte maneira:
  - Remaneja-se todos os elementos à partir da posição que o novo elemento será inserido;
  - Insere o novo elemento em sua posição;
    - De maneira semelhante ao algoritmo de ordenação insertion sort;
  - Incrementa-se o tamanho da lista
- A remoção de elementos nesse tipo de lista é feita da seguinte maneira:
  - Remaneja-se todos elementos a frente do que será excluído em uma posição para trás;
  - Decrementa-se o tamanho da lista;

Implementação

- Para a implementação da lista assuma o seguinte agregado heterogêneos:

```
struct tipo_lista{
    int tam;
    struct tipo_item dados[TAM];
};
```

- Além disso, assuma a definição da seguinte constante:

```
#define TAM 1000
```

Implementação

- A implementação de uma lista estática com remanejamento usa funções para as seguintes finalidades:
  - Inicializar lista;
  - Apagar lista;
  - Lista vazia;
  - Lista cheia;
  - Inserir elemento no fim;
  - Inserir elemento em uma posição
  - Remover elemento;
  - Procurar elemento;

Implementação

- A Sempre que uma lista é criada ela deve ser inicializada;
  - Ao criar uma lista ela está vazia, assim seu tamanho é 0;

```
void inicia_lista(struct tipo_lista*l){
    l->tam=0;
}
```

- Algumas vezes queremos apagar todos elementos da lista;
  - A menor maneira de fazer isso é colocando seu tamanho em 0;
    - Ou seja, é feita uma exclusão lógica de todos elementos.

Implementação

- Duas funções muito importantes para manipular os elementos da lista são as funções que perguntam se a lista está cheia ou se ela está vazia;
  - Quando a lista está cheia não podemos inserir novos elementos;
  - Quando a lista está vazia não podemos excluir elementos;

```
int lista_cheia(struct tipo_lista*l){
    return l->tam==TAM;
}
```

```
int lista_vazia(struct tipo_lista*l){
    return l->tam == 0;
}
```

Implementação

- Podemos inserir elementos na lista de diversas maneiras, dentre elas temos:
  - No final da lista
  - Em uma posição desejada
- Toda inserção deve ser feita somente quando a lista não está cheia;
- Toda inserção incrementa o tamanho da lista;

Implementação

- Inserção no final da lista;
  - É a maneira mais simples para inserir elementos na lista;
  - Não precisa de nenhum remanejamento das informações;

```
int insere_fim(struct tipo_lista *l, struct tipo_item x){
    if(!lista_cheia(l)){
        l->dados[l->tam]=x;
        l->tam++;
        return 1;
    }
    return 0;
}
```

Implementação

- Inserção na posição desejada:
  - Para inserir na posição desejada é necessário remanejar os dados, fazendo com que todos os dados a frente da posição desejada ocupem uma posição à frente da atual;

```
int insere_pos(struct tipo_lista *l, struct tipo_item x,int p){
    int i;
    if((!lista_cheia(l)) && (p <= l->tam)){
        for(i=l->tam-1;i>=p;i--){
            l->dados[i+1]=l->dados[i];
        }
        l->dados[p]=x;
        l->tam++;
        return 1;
    }
    return 0;
}
```

Implementação

- A exclusão também pode ser feita de mais de uma maneira, dentre elas temos:
  - Exclusão em uma posição determinada:
  - Exclusão por chave;
- Não se pode excluir elementos em listas vazias
- Toda exclusão decrementa o tamanho da lista

Implementação

- Exclusão em uma posição determinada:
  - Move-se todos elementos que estão a frente da posição que deseja ser excluída uma “casa” para trás;

```
int exclui_pos(struct tipo_lista *l, int p){
    int i;
    if((!lista_vazia(l)) && (p < l->tam)){
        for(i=p+1;i<l->tam;i++){
            l->dados[i-1]=l->dados[i];
        }
        l->tam--;
        return 1;
    }
    return 0;
}
```

Exercícios

1. Faça um procedimento que receba uma lista e mostre todos os elementos da lista.
2. Implemente a função de busca por chave na lista. Assuma que não existem chaves repetidas.
3. Altere as implementações de inserção para que não sejam permitidos elementos com a mesma chave na lista.
4. Implemente a função de remoção por chave na lista. Assuma que não existem chaves repetidas.
5. Crie uma função que permita a inserção ordenada na lista.

Referências

ZIVIANI, N.; Projeto de Algoritmos com Implementações em Pascal e C, 3ª Edição, Livraria Pioneira Editora, 1996.

TANENBAUM, A. M., Langsam, Y., Augenstein, M.- Estruturas de Dados Usando C. Editora Makron Books, 1995.