Estruturas de Dados

Listas Dinâmicas

Prof: 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

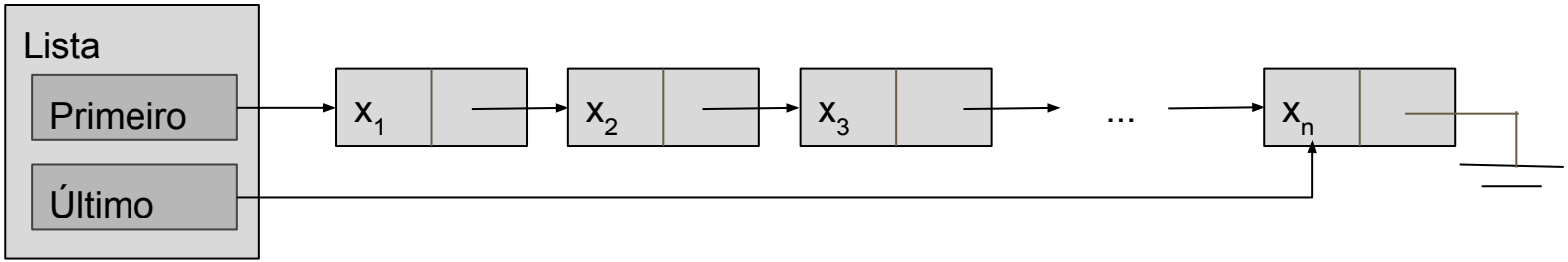
- Listas podem ser representadas por várias estruturas de dados;
- As formas mais comuns de representar listas são:
  - Utilizando arranjos
    - Listas estáticas;
  - Utilizando ponteiros;
    - Listas dinâmicas;

Listas dinâmicas

- Listas dinâmicas fazem uso de ponteiros;
  - Consumo extra de memória;
- As listas dinâmicas podem crescer ou diminuir de tamanho conforme a necessidade;
- São adequadas para aplicações onde não se pode prever a quantidade de dados a serem armazenados;
- São também conhecidas como listas encadeadas;
  - Existem diversas implementações para listas encadeadas;

Listas dinâmicas

- Lista ligada;



Listas dinâmicas

- Em uma lista dinâmica todos elementos são interligados;
  - Seja de maneira direta ou indireta;
- Uma lista dinâmica é composta de células;
- Cada célula possui o item armazenado e um ponteiro para o seguinte;

```
struct tipo_celula{
    struct tipo_item item;
    struct tipo_celula *prox;
};
```

Listas dinâmicas

- Um tipo lista, para listas dinâmicas, pode ser implementado como um agregado heterogêneo contendo:
  - Ponteiro para a primeira posição da lista;
  - Ponteiro para a última posição da lista;

```
struct tipo_lista{
    struct tipo_celula *primeiro;
    struct tipo_celula *ultimo;
};
```

Operações em listas dinâmicas

- Antes de começar a utilizar uma lista é preciso inicializá-la;
- A inicialização da lista é o procedimento de fazer a lista ficar vazia para que ela possa começar a ser operada;

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

Operações em listas dinâmicas

- É muito importante verificar se uma lista está vazia;
  - Não se pode remover itens de uma lista vazia;
  - A inserção e remoção precisam tratar alguns casos especiais que envolvem listas vazias;
- A função para verificar se uma lista está vazia pode ser implementada de duas maneiras:

```
int vazia(struct tipo_lista *l){
    if(l->primeiro == NULL){
        return 1;
    }else{
        return 0;
    }
}
```

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

Operações em listas dinâmicas

- A inserção de um elemento numa lista pode ocorrer em diversas posições:
  - Início da lista;
  - Fim da lista;
  - Após a i-ésima posição;
  - Inserir ordenado;
    - A lista deve estar previamente ordenada;

Operações em listas dinâmicas

- Inserção no início

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

Operações em lista

- Inserção no fim

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

Operações com listas dinâmicas

- A remoção na lista pode ser feita de diversas maneiras:
  - Início da lista;
  - Fim da lista;
  - Em uma posição específica;
  - Por um determinado valor de chave;

Operações com lista dinâmica

- Remoção no início:

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

Operações com listas ligadas

- Remoção no fim

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

Operações com listas dinâmicas

- Para remover um elemento com uma determinada posição:

```
int remove_posicao(struct tipo_lista *l, int pos,struct tipo_item *retorno){
    int i;
    struct tipo_item it;
    struct tipo_celula *e,*ant;
    if(!vazia (l)){
        if(pos==0){
            e=l->primeiro;
            l->primeiro=l->primeiro->prox;
            if (vazia(l)){
                l->ultimo=NULL;
            }
            *retorno=e->item;
            free(e);
            return 1;
        }
```

```
                else{
                 ant=l->primeiro;
                 e=ant->prox;
                 i=1;
                 while((e != NULL) && (i<pos)){
                         e=e->prox;
                         ant=ant->prox;
                         i++;
                 }
                 if(e != NULL){
                         ant->prox = e->prox;
                         *retorno=e->item;
                         free(e);
                         if(ant->prox == NULL){
                                 l->ultimo = ant;
                         }
                         return 1;
                 }
         }
    }
    return 0;
}
```

Operações com listas dinâmicas

- Para buscar um elemento em uma lista, usa-se a seguinte função

```
struct tipo_celula *buscar(struct tipo_lista *l, int c){
        struct tipo_celula *i;
        i=l->primeiro;
        while(i != NULL){
                if(i->item.chave == c){
                        return i;
                }
                i=i->prox;
        }
        return NULL;
}
```

Operações com listas dinâmicas

- Para escrever todos elementos da lista usa-se o seguinte procedimento:

```
void escreve_lista(struct tipo_lista *l){
        struct tipo_celula *i;
        i=l->primeiro;
        while(i!=NULL){
                printf("%d ",i->item.chave);
                i=i->prox;
        }
        printf("\n");
}
```

Exercícios

1. Faça um subprograma que receba uma lista e mostre todos seus elementos, de maneira recursiva. Dica: receba somente um ponteiro para uma célula ao invés de uma lista;
2. Implemente um subprograma para fazer a remoção de um elemento com determinada chave em uma lista. Assuma que não existem chaves repetidas na lista;
3. Implemente um subprograma que receba uma lista e uma posição i e faça a inserção após a i-ésima posição da uma lista;
4. Crie um subprograma que receba duas listas, A e B, e retorne em A a concatenação de A e B, e em B uma lista vazia. Assuma que ambas as listas podem ter elementos com chaves repetidas.

Exercícios

5. Faça um subprograma que insira um elemento ordenado, pela chave, na lista;
6. Faça uma função que receba uma lista e conte a quantidade de elementos na lista;
7. Faça uma função que receba uma lista e um valor para a chave. Sua função deverá retornar todos os elementos da lista cujo a chave tem o valor informado.
8. Faça um subprograma que receba uma lista e exclua todos seus itens.
9. Implemente todas as operações de inserção em uma lista que não permitam inserções com chaves repetidas.

Referências

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

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