Algoritmos e Programação Estruturada

Listas

Você sabia que seu material didático é interativo e multimídia? Isso significa que você pode interagir com o conteúdo de diversas formas, a qualquer hora e lugar. Na versão impressa, porém, alguns conteúdos interativos ficam desabilitados. Por essa razão, fique atento: sempre que possível, opte pela versão digital. Bons estudos!

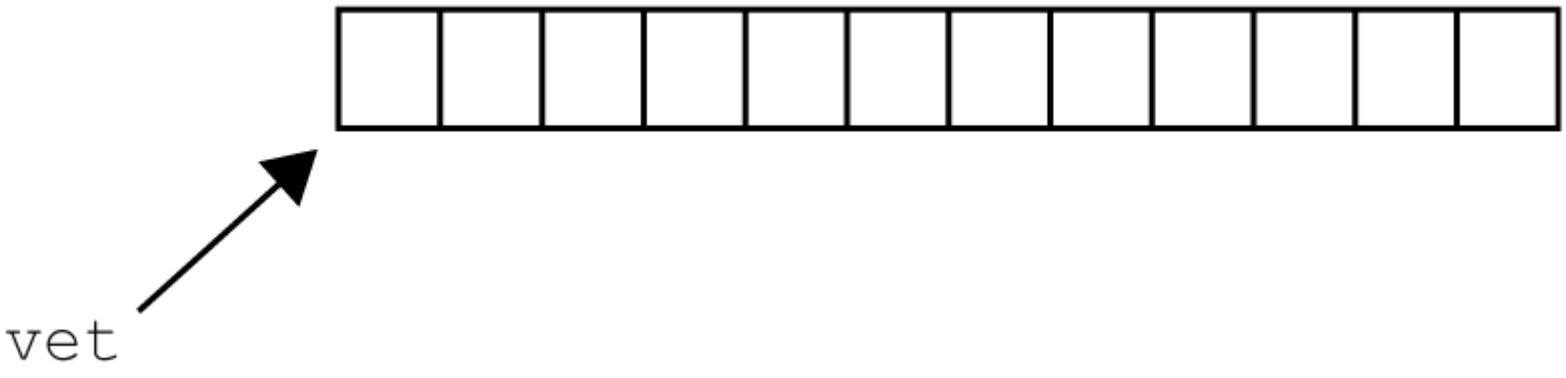
Nesta webaula vamos conhecer o funcionamento das listas ligadas.

Estruturas de dados

As estruturas de dados são formas de organização e distribuição de dados para tornar mais eficientes a busca e manipulação dos dados por algoritmos. As listas são estruturas de dados dinâmicas, uma vez que o número de elementos de uma lista é variável conforme eles são inseridos ou removidos.

Para a implementação de listas, os vetores são muito utilizados para representar um conjunto de dados, permitindo definir um tamanho máximo de elementos a ser utilizados neste vetor.

Exemplo de vetor



Fonte: Celes, Cerqueira, Rangel, (2004, p. 134).

O uso do vetor, ao ser declarado, reserva um espaço contíguo na memória para armazenar seus elementos. Assim, é possível acessar qualquer um dos seus elementos a partir do primeiro elemento, por meio de um ponteiro (CELES, CERQUEIRA e RANGEL, 2004).

Listas Ligadas

Uma Lista Ligada, também conhecida como Lista Encadeada, é um conjunto de dados dispostos por uma sequência de nós, onde a relação de sucessão desses elementos é determinada por um ponteiro que indica a posição do próximo elemento, podendo estar ordenado ou não (SILVA, 2007).

Uma lista é uma coleção

```
L:[a1, a2, ..., an]
```

onde `n > 0`

Sua propriedade estrutural baseia-se apenas na posição relativa dos elementos, que são dispostos linearmente.

O nó de lista é composto por duas informações, uma informação armazenada e um ponteiro que indica o próximo elemento da lista.

Modelo de lista ligada

Fonte: Celes, Cerqueira, Rangel, (2004, p. 135).

Diferente dos vetores, onde o armazenamento é realizado de forma contígua, a Lista Ligada estabelece a sequência de forma lógica.

Na lista encadeada é definido um ponto inicial ou um ponteiro para o começo da lista, e a partir daí pode se inserir elementos, remover ou realizar buscas na lista.

Quando uma lista está sem nós, definimos como lista vazia ou lista nula, assim, o valor do ponteiro para o próximo nó da lista, é considerado como ponteiro nulo.

Criação ou definição da estrutura de uma lista	Inicialização da lista	Inserção com base em um endereço como referência
Alocação de um endereço de nó para inserção na lista	Remoção do nó com base em um endereço como referência	Deslocamento do nó removido da lista

Elementos de dados em listas ligadas

Os elementos de uma lista são armazenados em posições sequenciais de memória, sempre que possível e de forma dinâmica, e permitem que a lista seja percorrida em qualquer direção.

Os elementos de informação de uma lista podem ser do tipo `int`, `char` e/ou `float` .

Ao criar uma estrutura de uma lista, definimos também o tipo de dados que será utilizado em sua implementação.

```
struct lista {
          int info;
          struct lista * prox;
};
typedef struct lista Lista;
```

Exemplo de declaração para criar uma lista em C

```
/*Cria a estrutura da lista*/
struct alunos
   char nome[25];
   struct alunos*prox;
};

Typedef struct alunos Classe;
```

- Criado uma `struct` (registro) alunos;
- Na `struct`, há uma variável nome do tipo char, que será nossa informação;
- Outra `struct prox` com ponteiro para a própria `struct` alunos para receber o endereço de apontamento da próxima informação.

Inicialização da lista

Precisamos inicializar a lista para utilizarmos após sua criação.

Para isso, basta criarmos uma função onde inicializamos a lista como nula, como uma das possíveis formas de inicialização.

```
/* Função para inicialização: retorna uma lista vazia */
Lista* inicializa (void)
{
   return NULL;
}
```

Ponteiros – elementos de ligação

A utilização de ponteiros é indicada nos casos onde é preciso conhecer o endereço que está armazenada a variável.

```
int *ptr; /* sendo um ponteiro do tipo inteiro*/
float *ptr; /* sendo um ponteiro do tipo ponto flutuante*/
char *ptr; /* sendo um ponteiro do tipo caracteres*/
```

A seguir, veja uma exemplo de uma estrutura de lista declarada para armazenar dados de uma agenda.

```
typedef struct lista {
        char *nome;     /*Declaração de um ponteiro do tipo char
        int telefone;
        struct lista *proximo;
} Dados;
```

Para sabermos o endereço da memória reservada para a variável, utiliza se o operador & juntamente ao nome de uma variável.

```
   int x = 10;  /*variável
   int *p;        /*ponteiro
   p = &x;      /*ponteiro p aponta para o endereço da variável x
```

Função `malloc()`

Em listas, além do uso de ponteiros, utilizamos também as alocações dinâmicas de memória, que são porções de memórias para utilização das listas. Para isso são utilizadas a função `malloc()`, Memory Allocation ou Alocação de Memória. Ela é a responsável pela reserva de espaços na memória principal. Sua finalidade é alocar uma faixa de bytes consecutivos na memória do computador e retornar seu endereço ao sistema.

Exemplo de utilização da função `malloc()` e de ponteiro:

```
    char *pnt;
    pnt = malloc (2); /* Aloca 2 bytes na memória */
    scanf ("%c", pnt);
```

O endereço retornado é o da posição inicial onde se localiza os bytes alocados.

Em uma lista, precisamos alocar o tipo de dado no qual foram declarados os elementos da lista, e por este tipo de dados ocupar vários bytes na memória, precisaremos utilizar a função sizeof, que permite nos informar quantos bytes o tipo de elemento criado terá.

Exemplo de programa em C com `malloc()` e `sizeof`

```c
#include <stdio.h>
#include <stdlib.h>

int main () {

    int *p;
    p=(int *) malloc(sizeof(int));

    if (!p) {
        printf("Erro de memoria insuficiente");
    }else{
        printf("Memoria alocada com sucesso");
    }

    return 0;
}
```

Podemos classificar as listas de duas formas.

Lista com cabeçalho

Onde o primeiro elemento permanece sempre como ponto inicial na memória, independente se a lista está com valores ou não. Assim, o start é o endereço inicial da nossa lista e para determinar que a lista está vazia.

```c
celula *start;
start = malloc (sizeof (celula));
start -> prox = NULL;
```

Lista sem cabeçalho

Onde o conteúdo do primeiro elemento tem a mesma importância que os demais elementos. Assim, a lista é considerada vazia se o primeiro elemento é NULL. A criação deste tipo de lista vazia pode ser definida por `start = NULL`.

Exemplo de um trecho de uma função do tipo inteiro, para inserir pacientes com prioridades na lista.

```c
int insere_com_prioridade(Fila *F , Paciente *P){
        Lista *ptnodo, *aux , *ant;
        ptnodo = (Lista*)malloc(sizeof(Lista));
```

Aplicação de listas ligadas no nosso dia a dia

Lista de compra de supermercado

Ao anotar tudo que você precisa comprar, você automaticamente está criando uma lista, e esta lista pode ser alocada com novos produtos ou remover produtos dela conforme a compra está sendo realizado.

```
Dados *inicia_listaMerc(char *prod, int numpro) {

        Dados *novo;

        novo = (Dados *)malloc(sizeof(Dados));

        novo -> prod = (char *)malloc(strlen(prod) +1);
        strncpy (novo -> prod, prod, strlen(prod) +1);
        novo -> numpro = numpro;
        novo -> proximo = NULL;

        return novo;
}
```

Desenvolvimento de um sistema para o site de uma empresa de casamentos

Os usuários criam a lista de convidados e a lista de presentes, os noivos podem adicionar convidados ou presentes e remover ou alterar da lista quando necessário.

```
void convidados inserirConvid(tipoitem elemento,int &cont)
{
    festa *novo, *aux, *aux1;
    aux = inicio;
    aux1 = inicio -> prox;

    while (aux1 != NULL) {
        if(strcmp(aux1 -> item.nome, elemento.nome) > 0)
            break;

        aux = aux -> prox;
        aux1 = aux1 -> prox;
    }

    if((novo = new(festa) == NULL)
        printf("\nMemoria insuficiente");
    else {
        novo -> prox = aux -> prox;
        aux - > prox = novo;
        novo -> item = elemento;
        cont++;
        printf("\nConvidado inserido com sucesso!");
    }

    if (aux1 == NULL)
        fim = novo;

    return;
}
```