1. Listas encadeadas

Listas

- Em computação uma lista é uma estrutura de dados linear utilizada para armazenar e organizar dados em um computador;
- É uma sequência de elementos do mesmo tipo;
- Uma lista pode ser ordenada ou não e pode possuir elementos repetidos.
- A estrutura de uma lista pode possuir N (N>0) elementos ou se N for igual a zero, dizemos que a lista está vazia.

Estruturas de dados dinâmicas:

- crescem (ou decrescem) à medida que elementos são inseridos (ou removidos)
- Exemplo:
    - listas encadeadas: amplamente usadas para implementar outras estruturas de dados

Listas encadeadas

Instruções

- sequência encadeada de elementos, chamados de nós da lista
- nó da lista é representado por dois campos:
    - Informação armazenada
    - Ponteiro para o próximo elemento da lista
- a lista é representada por um ponteiro para o primeiro nó
- o ponteiro do último elemento é NULL
- Lista é uma estrutura autorreferenciada, pois o campo prox é um ponteiro para uma próxima estrutura do mesmo tipo.

Aplicações

Estruturas

- struct do tipo lista
    - nessa estrutura armazenamos os atributos da lista;
    - por exemplo: `int numero`
- struct do tipo nó, contendo os dois campos:
    - INFO: `struct tLista info`
        - neste campo é referenciado as informações da estrutura lista
    - PROX: `struct tNo *prox`
        - este é o campo ponteiro

Inclusão

- Inclusão: no início da lista
  - criado um ponteiro auxiliar chamado “novo”; `struct tNo *novo`
    - alocado como struct tNo com a função `malloc(sizeof(struct tNo))`
  - o prox do novo irá apontar para o prox do primeiro nó da lista; `novo->prox = lista`
  - e lista passa a apontar para novo, o atual primeiro. `lista = novo`
- Inclusão: lista vazia
  - lista aponta para NULL
    - já iniciamos ou declamos a lista dessa forma: `struct tNo *lista = NULL`
  - o prox do novo irá apontar para o prox da lista `novo->prox = lista`
  - logo, o prox do novo irá apontar para NULL `lista = novo`

Listagem

- criamos um ponteiro auxiliar chamado “p”; `struct tNo *p`
- p aponta para o primeiro nó da lista; `p = lista`
  - apresenta a informação, enquanto o lugar onde p aponta for diferente de NULL `while (p != NULL)`
- para seguir cada nó da lista, utilizamos: `p = p → prox`
  - Esse exemplo é a mesma ideia que `i = i + 1`, como índice.

---

2. Fila

Introdução

- Não ordenada
- Entrada no fim
- Saída no início
- um novo elemento é inserido no final da fila e um elemento é retirado do início da fila
  - fila = First In First Out (FIFO) “o primeiro que entra é o primeiro que sai”
- fim | … | … | início ou início | … | … | fim

Implementação de fila com lista

- elementos da fila armazenados na lista
- usa dois ponteiros
  - ini aponta para o primeiro elemento da fila
  - fim aponta para o último elemento da fila
- elementos da fila armazenados na lista
- fila representada por um ponteiro para o primeiro nó da lista

Funções

- função cria_fila
    - cria aloca a estrutura da fila
    - inicializa a lista como sendo vazia
- função insere_elemento
    - insere novo elemento n no final da lista
- função remove_elemento
    - retira o elemento do início da lista
    - libera a fila depois de liberar todos os elementos da lista

---

3. Pilha

Introdução

novo elemento é inserido no topo e acesso é apenas ao topo

- o primeiro que sai é o último que entrou (LIFO – last in, first out)
- operações básicas:
    - push - insere novo elemento no topo da pilha
    - pop - remove o elemento do topo da pilha

Implementação de pilha com lista

- elementos da pilha armazenados na lista
- pilha representada por um ponteiro para o primeiro nó da lista

Funções

- função cria_pilha
    - aloca dinamicamente a estrutura da pilha
    - inicializa a lista como sendo vazia
- função push
    - insere novo elemento no início da lista
- função pop
    - retira o elemento do início da lista
- função destruir
    - libera todos os elementos da lista e depois libera a pilha
- img

---

4. Tabela Hash

O método de pesquisa conhecido como hash (tabela de dispersão) é constituído de duas etapas principais:

1. Computar o valor da função de dispersão (ou função hash), a qual transforma a chave de pesquisa em um endereço da tabela.
2. Considerando que duas ou mais chaves podem ser transformadas em um mesmo endereço de tabela, é necessário existir um método para lidar com as colisões.

Tipos de Tratamentos de colisão

- Endereçamento fechado
    - Endereçamento linear
    - Endereçamento quadrático
    - hash duplo
- Endereçamento aberto
    - Encadeamento
        - colisões são colocadas dentro de uma lista encadeada

Definições

- M: tamanho da tabela
- N: quantidade de valores
- alpha: fator de carga N/M
    - alpha = 1, tabela cheia
    - alpha = 0.5, tabela 50% ocupada

Eficiência

- Bem dimensionada 1,5 acessos
- alpha (fator de carga)
    - < 0.75 reduz colisões
    - 0.5 traz bons resultados
    - < 0.25 gasto excessivo
- M ser um valor primo
- Hash perfeito
- Objetivo: Ter eficiência O(1) nas operações de busca, inserção e remoção. Para isso, as inserções e remoções não devem provocar grandes variações na quantidade de registros armazenados.

---

5. Busca Sequencial e Binária

Introdução

- Busca em vetor:
    - elemento chave para busca

    - entrada: vetor com n elementos aleatórios
    - saída: n se o elemento ocorre em vetor[n], -1 se o elemento não se encontra no vetor
    - variáveis v1 e v2 criadas globalmente, para verificar quantidades de acessos/buscas

Busca Sequencial em Vetor

- percorra o vetor, elemento a elemento, verificando se é igual a um dos elementos do vetor

pior caso:

- n comparações, onde n representa o número de elementos do vetor
    - desempenho computacional varia linearmente em relação ao tamanho do problema (algoritmo de busca linear)
- complexidade: O(n)

caso médio:

- n/2 comparações
    - desempenho computacional continua variando linearmente em relação ao tamanho do problema
- complexidade: O(n)

Busca Binária em Vetor

- criado variáveis para início, meio e fim do vetor
    - se a chave for igual ao meio, valor foi encontrado
        - se não, se for maior que o meio, início fica depois de meio
        - se não, se for menor que o meio, fim fica antes de meio
    - até fim ser menor ou igual ao início

prós:

- dados armazenados em vetor, de forma ordenada
- bom desempenho computacional para pesquisa

contra:

- inadequado quando inserções e remoções são frequentes
    - exige rearrumar o vetor para abrir espaço uma inserção
    - exige rearrumar o vetor após uma remoção

Busca binária - O que pode variar?

- Critério de ordenação (primário e desempates)
- A informação retornada:

    - O índice do elemento encontrado ou -1
- Repetição ou não de valores (chaves)

Funções complementares

- gerarVetor: criado vetor com valores aleatórios utilizando a biblioteca time.h
- imprimirVetor: função criada para apresentação do vetor
