Notas de aula – AED 1 – listas estáticas
Prof. Jefferson T. Oliva

Programas operam sobre dados, sejam eles fornecidos pelo usuário ou já inclusos dentro dos programas (variáveis, arquivos, etc).

Estrutura de dados armazena dados na memória do computador a fim de permitir o acesso eficiente dos mesmos.

Uma estrutura de dados abstrai as características principais de uma atividade que envolve armazenamento de informações.

Uma das formas de organizar informações é por meio do uso de conjuntos de dados.

Conjuntos são tão fundamentais para a Computação quanto para a Matemática.

Abstração de conjuntos de dados pode ser por alocação estática ou alocação dinâmica.

Hoje veremos uma das estruturas fundamentais na computação: listas

Listas

Lista é uma estrutura de dados que implementa uma coleção de elementos (como structs) organizados em sequência. Essa estrutura agrupa informações cujos elementos são organizados de alguma forma.

Listas estão presentes no nosso cotidiano, como lista de compras no site da Amazon, lista de presença, lista de contatos, etc. Como eles são organizados?

Na Computação, listas são usadas em aplicações em que não é possível prever a demanda de memória, como manipulação simbólica, gerenciamento de memória, simulações, processamento de imagens, compiladores, etc.

Uma lista pode ser linear ou não linear (generalizada), sendo essa última veremos se houver tempo.

Existem dois tipos especiais de listas que veremos nas próximas aulas: listas e pilhas.

Formalmente, uma lista linear é uma sequência de 0 ou mais itens: {x[1], x[2], x[3], …, x[n]}, onde n é o tamanho da lista, x[i] é o i-ésimo elemento, x[1] é o primeiro elemento e x[n], o último, x[i] sucede x[i – 1] e precede x[i + 1]. Por sim, se n = 0, então a lista está vazia.

Operações básicas em listas (TAD de lista):
- Criar uma lista vazia
- Inserir um elemento
- Remover um elemento
- Verificar se a lista está vazia
- Procurar um elemento
- Concatenar duas ou mais listas
- Dividir uma lista em duas ou mais listas
- Ordenar a lista
- Imprimir todos os elementos
- Retornar referência (e.g. primeiro ou próximo item da lista)

Em listas, os elementos podem ser alocados de forma sequencial (estática) ou de forma encadeada (dinâmica)

Listas Estáticas

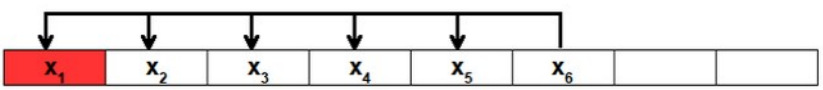
Os itens dessa estrutura são armazenados em posições contíguas da memória (vetores/alocação estática): lista.item[i]

A lista estática pode ser precorrida, linearmente, em qualquer direção.

Uma forma de implementação de listas estáticas, além de considerar um vetor, também podem ser definidas: Uma variável para indicar a posição do primeiro elemento da lista (útil em filas estáticas) e uma variável para indicar a posição do último elemento

Um novo item pode ser inserido no final da lista com custo constante

No entanto, a remoção do item requer o deslocamento de itens para preencher o espaço vazio



Vantagens: simplicidade para implementação e “economia de memória”

Desvantagens: custo para a remoção de itens e impossibilidade de realocar memória

TAD Listas Estáticas (alternativa à apresentada em aula, para vocês verem um outro exemplo de TAD de listas estáticas)

- Criar uma lista vazia
- Inserir um item
- Remover um item
- Acessar um item
- Verificar se a lista está vazia
- Verificar se a lista está cheia
- Imprimir a lista
- Liberar lista

Estrutura de dados:

```
#define MAX_SIZE 100 // tamanho máximo da lista
typedef struct {
        int key;

        /*aqui podem haver outros campos, mas para fins didáticos e para irmos diretos ao assunto, usei a chave, que é o campo mais utilizado em operações de busca (conteúdo da segunda prova)*/
}Item;

typedef struct{
        Item item[MAX_SIZE];
        int pos_last;
}List;
```

Arquivo .h

```
typedef struct Item Item;
typedef struct List List;

#define MAX_SIZE 100 // tamanho máximo da lista

List* criar();

int vazio(List *l);

int cheio(List *l);

int buscar(List *l, int key);
```

```
int inserir(List *l, int key);

int remover(List *l, int key);
void imprimir(List *l);

void liberar(List *l);
```

Arquivo .c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "list.h"

struct Item{
   int key;
};

struct List{
   Item item[MAX_SIZE];
   int pos_last;
};

List* criar(){
   List *l = malloc(sizeof(List));
   l->pos_last = -1;
   return l;
}


int vazio(List *l){
   return l->pos_last < 0;
}


int cheio(List *l){
   return l->pos_last >= (MAX_SIZE - 1);
}


int buscar(List *l, int key){
   int i;

   for (i = 0; i <= l->pos_last; i++)
      if (key == l->item[i].key)
         return i;
```

```
    return -1;
}

int inserir(List *l, int key){
    Item new_item;

    if (!cheio(l)){
        new_item.key = key;
        l->pos_last++;
        l->item[l->pos_last] = new_item;

        return 1;
    }

    return 0;
}


int remover(List *l, int key){
    int i;
    int p = buscar(l, key);

    if (p >= 0){
        for (i = p; i < l->pos_last; i++)
            l->item[i] = l->item[i + 1];

        l->pos_last--;

        return 1;
    }

    return 0;
}


void imprimir(List *l){
    int i;

    for (i = 0; i <= l->pos_last; i++)
        printf("%d\n", l->item[i].key);

}


void liberar(List *l){
    free(l);
}
```

Exercício

Aproveitando o TAD anterior, faça:
- Implemente uma função que concatena duas listas.
- Altere o TAD de forma que os itens devam estar ordenados
- Implemente uma função que intercale duas listas em uma terceira de forma ordenada

Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. Third edition, The MIT Press, 2009.

Pereira, S. L. Estrutura de Dados e em C: uma abordagem didática. Saraiva, 2016.

Szwarcfiter, J.; Markenzon, L. Estruturas de Dados e Seus Algoritmos. LTC, 2010.

Tenenbaum, A.; Langsam, Y. Estruturas de Dados usando C. Pearson, 1995.

Ziviani, M. Projetos de Algoritmos: com implementações em Pascal e C. Thomson, 2004.