AED1
Estruturas Lineares - Listas

Hebert Coelho e Nádia Félix

Instituto de Informática
Universidade Federal de Goiás

Roteiro

- Tipo Abstrato de Dados (TAD)
- Listas Lineares
- Exercícios

Tipo Abstrato de Dados

Tipo Abstrato de Dado (TAD) é uma especificação de um **conjunto de dados** e **operações** que podem ser executadas sobre esses dados.

Para ser considerado uma TAD devemos aplicar as operações definidas independente de saber como foram implementadas ou como os dados foram armazenados. O importante é saber o que as operações fazem, quais seus parâmetros de entrada e o que produzem como resultado, sem saber como foram construídas.

Listas Lineares

Conjunto de elementos (de informações) cuja propriedade estrutural envolve as posições relativas de seus elementos. Supondo $n > 0$, uma lista linear é representada por $(x_1, x_2, \ldots, x_n)$, onde:

1. x_1 é o primeiro elemento da lista;
2. x_k é precedido por x_{k-1} e seguido por x_{k+1}, para $1 < k < n$; e
3. x_n é o último elemento da lista.

Listas Lineares

Algumas operações que podem ser realizadas sobre listas lineares:

1. **ter acesso ao elemento x_k, para um k qualquer, com o objetivo de consultar ou alterar o seu conteúdo;**
2. **inserir um elemento novo;**
3. **remover o elemento x_k;**
4. combinar 2 ou mais listas lineares em uma só lista linear;
5. quebrar uma lista linear em 2 ou mais listas lineares;
6. copiar uma lista linear em outro espaço; ou
7. determinar o número de elementos de uma lista linear.

Listas Lineares

Casos particulares de listas são de especial interesse:

1. **Deque** quando as inserções e remoções são permitidas nas extremidades da lista;
2. **Pilha** quando as inserções e remoções são realizadas apenas em um extremo;
3. **Fila** quando as inserções são realizadas em um extremo e as remoções em outro extremo;

Fonte: SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

Alocação Sequencial (uso de vetores)

A maneira mais simples de manter uma lista linear é usando um vetor (alocação sequencial).

- Como a alocação sequencial da maioria das linguagens geralmente é realizada com a reserva prévia de memória para cada estrutura, a inserção e a remoção de nós não ocorrem de fato.
- Normalmente é feita uma simulação das operações de inserção e remoção. É comum delimitar os limites da estrutura utilizando variáveis.

Listas Lineares em Alocação Sequencial

```
1  struct lista{
2    int chave;
3    int dado;
4  };
```

Busca do elemento x na lista L com n elementos:

```
1  int busca1 ( struct lista* L, int x, int n)
2  {
3    int i = 0;
4    while (i <= n-1){
5       if (L[i].chave == x){
6            return i; // chave encontrada
7       }
8       else i = i + 1; // pesquisa prossegue
9    }
10   return -1;
11 }
```

Fonte: SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

Listas Lineares em Alocação Sequencial

```
1  struct lista{
2    int chave;
3    int dado;
4  };
```

Busca do elemento x na lista L, conhecendo sua chave:

```
1  int busca2 ( struct lista* L, int x, int n)
2  {
3    int i = 0;
4    L[n].chave = x;
5    while (L[i].chave != x){
6         i = i + 1;
7    }
8    if (i != n)
9         return i;
10   else
11        return -1;
12 }
```

Fonte: SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

Exercícios

1. Qual função é executada mais rápida, busca1 ou busca2? Porque?
2. Qual função de complexidade para a busca1 ou busca2?
3. Qual a notação O (big O) para a busca1 ou busca2? (ex: $O(1)$, $O(log(n))$, $O(n^2)$).
4. Qual a complexidade no melhor caso, pior caso e caso médio para a busca1 e busca2 do elemento x na lista L?

Verifique as respostas nos slides da última aula e no livro texto abaixo.

Fonte: SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

Listas Lineares em Alocação Sequencial

Busca do elemento x na lista L ordenada, conhecendo sua chave:

```
1  int busca_binaria (struct lista* L, int x, int n)
2  {
3    int inf = 0, sup = n - 1, meio;
4    while (inf <= sup){
5         meio = (int)(inf + sup)/2;
6         if (L[meio].chave == x)
7                return meio;
8         else if(L[meio].chave < x)
9                inf = meio+1;
10             else sup = meio - 1;
11   }
12   return -1;
13 }
```

Fonte: SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

Exercícios

1. Qual função é executada mais rápida busca2 ou busca binária? Porque?
2. Qual função de complexidade para a busca binária?
3. Qual a notação O (big O) para a busca binária? (ex: $O(1)$, $O(log(n))$, $O(n^2)$).
4. Qual a complexidade no pior caso para a busca binária do elemento x na lista L?
5. Compare os algoritmos do livro texto e as implementações apresentadas aqui.

Verifique as respostas nos slides da última aula e no livro texto abaixo.

Fonte: SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

Listas Lineares em Alocação Sequencial

Inserção de um nó na lista L não ordenada:

```
1  int insercao (struct lista* L, int x, int* n, int M, int novo_valor)
2  //M é o número máximo de elementos que a lista L pode armazenar
3  {
4    if (*n < M){
5       if(busca2(L, x, *n) == -1){
6            L[n].chave = x;
7            L[n].dado = novo_valor;
8            (*n)++;
9       }
10      else {
11           printf("valor já existe na lista");
12           return -1;
13      }
14   }
15   else {
16      printf("Overflow"); // Número de elementos ultrapassou o tamanho da lista
17      return -1;
18   }
19 }
```

Exercício

1. Escreva a função de inserção para uma lista ordenada em alocação sequencial. Sua função de inserção deverá utilizar uma função de busca binária externa para verificar se o elemento existe ou não. Caso o elemento não exista a sua busca binária deverá retornar a posição onde o elemento deverá ser inserido.

Alocação Dinâmica (uso de ponteiros)

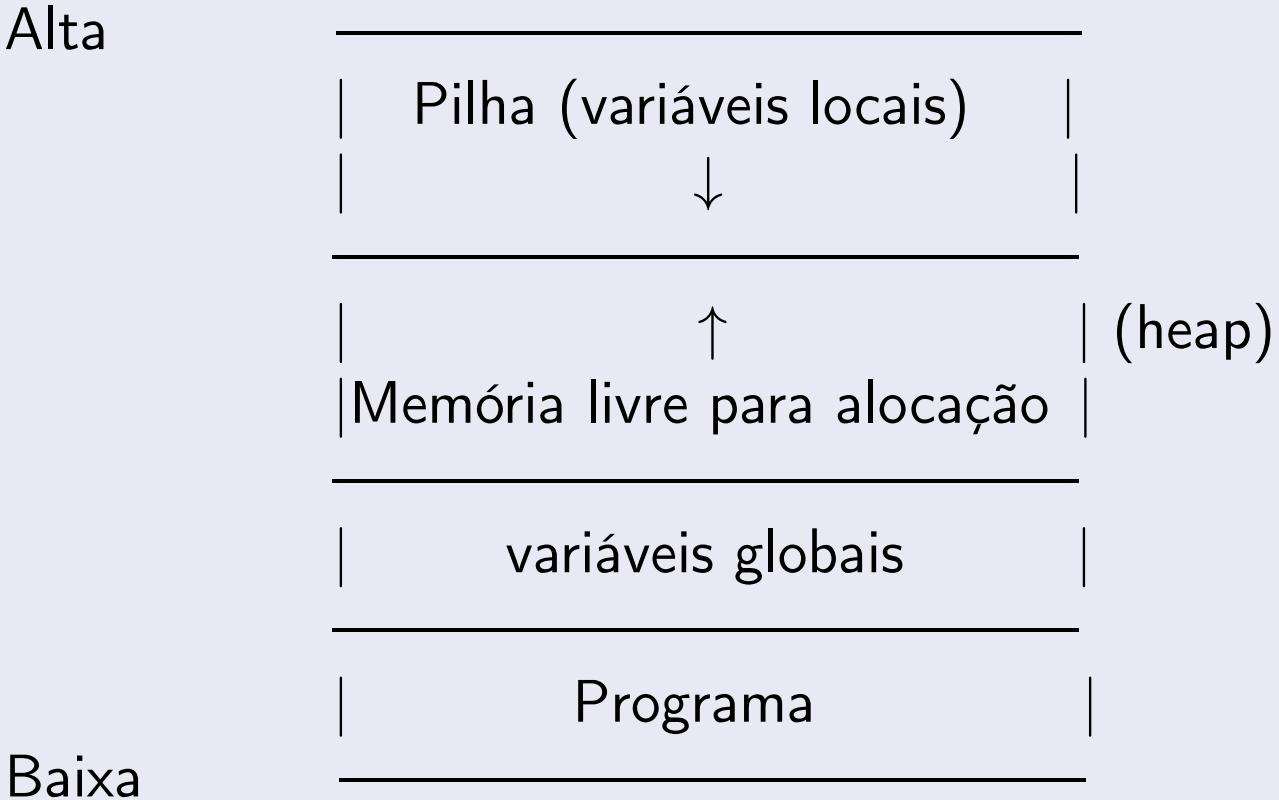
Existem 2 maneiras fundamentais de um programa em C armazenar informações na memória principal do computador.

- Variáveis locais e globais, incluindo matrizes e estruturas;
  Armazenamento fixo durante toda a execução do programa.

- Alocação Dinâmica
  Nesta maneira o programa pode obter espaço para armazenamento em tempo de execução.

Memória do sistema

O uso da memória de um programa em C



C ANSI

O padrão C ANSI especifica apenas quatro funções para o sistema de alocação dinâmica:

- calloc();
- **malloc()**;
- **free()**;
- realloc();

As funções de alocação dinâmica definidas pelo padrão C ANSI estão na biblioteca stdlib.h

Fonte:SHILDT, H. C completo e Total, Makron Books, 1996.

malloc()

void *malloc(size_t size);

- A função malloc() devolve um ponteiro para o primeiro byte de uma região de memória de tamanho size que foi alocada do heap;
- Se não houver memória suficiente é devolvido um ponteiro nulo;
- Sempre verificar se o valor devolvido não é um ponteiro nulo antes de utilizá-lo;

malloc - Exemplo de uso do malloc

```
1  #include<stdlib.h>
2  #include<stdio.h>
3
4  struct endereco{
5  char nome[40];
6  char rua[40];
7  char cidade[40];
8  char estado[2];
9  };
10
11 struct endereco *get_struct(void){
12       struct endereco *p;
13
14       if ((p = malloc(sizeof(struct endereco)))==NULL){
15            printf("Erro de alocação");
16            exit(1);
17       }
18       return p;
19 }
```

free()

void free(void *ptr);

- A função free() devolve ao heap a memória apontada por ptr, tornando a memória disponível para alocação futura.
- free() deve ser chamado apenas com um ponteiro que foi previamente alocado com as funções de alocação dinâmica.
- Um ponteiro inválido pode destruir o mecanismo de gerenciamento de memória;

free - Exemplo de uso do free

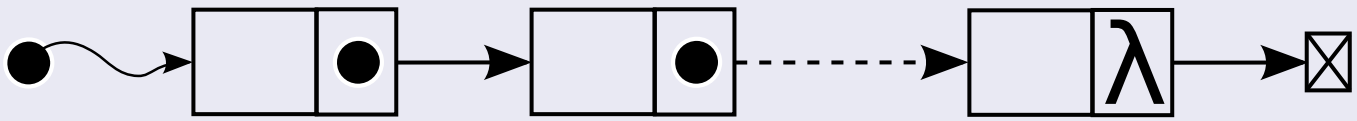
```
1  #include<stdlib.h>
2  #include<stdio.h>
3  #include<string.h>
4  #define tam 3
5  int main (){
6      char *str[tam];
7      int i;
8
9      for (i=0; i<tam; i++){
10         if ((str[i] = malloc(128))==NULL){
11             printf("Erro de alocação");
12             exit(1);
13         }
14          gets(str[i]);
15          puts(str[i]);
16     }
17     for (i=0; i<tam; i++) free(str[i]);
18     return 0;
19 }
```

Listas Lineares em Alocação Dinâmica

Considere a estrutura básica definida abaixo, com ela podemos construir uma lista linear conforme representada na figura a seguir. O λ na figura representa um ponteiro NULL.

Estrutura básica:

```
1 typedef struct elem
2 {
3     int info ;
4     struct elem *link ;
5 } celula;
```



obs.: Iremos usar sem distinção os nomes nó ou célula.

Listas Lineares em Alocação Dinâmica

A estrutura básica definida abaixo também pode ser definida em 2 passos.

Estrutura básica:

```
1 typedef struct elem
2 {
3     int info ;
4     struct elem *link ;
5 } celula;
```

Estrutura básica definida em 2 passos:

```
1 struct elem
2 {
3     int info ;
4     struct elem *link ;
5 };
6 typedef struct elem celula;
```

Fonte: FEOFILOFF, P. Algoritmos em Linguagem C, Campus, 2009.

Listas Lineares em Alocação Dinâmica

Definição de uma célula e de um ponteiro para célula.

```
1 typedef struct elem celula; //celula é um novo tipo de dado
2 celula c; //criou uma celula
3 celula *p; //criou um ponteiro para celula
```

Informação da célula:

```
1 c.info;
2 p->info;
```

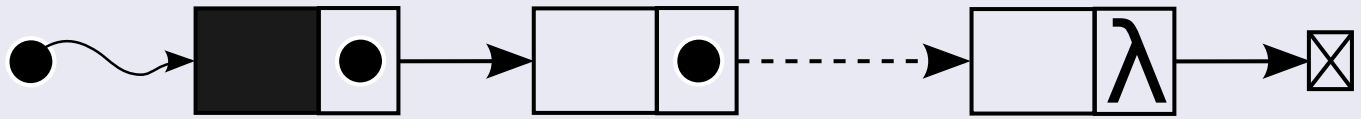
Endereço da célula seguinte:

```
1 c.link;
2 p->link;
```

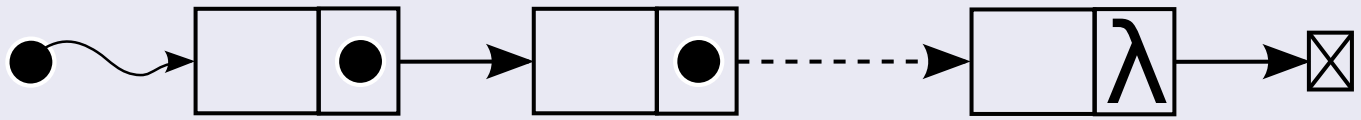
Listas com cabeça

Uma lista encadeada pode ser vista de duas maneiras diferentes, dependendo do papel que sua primeira célula desempenha.

- Na lista com cabeça, a primeira célula (cabeça) serve apenas para marcar o início da lista e portanto seu conteúdo é irrelevante.



- Na lista sem cabeça, o conteúdo da primeira célula é tão relevante quanto o das demais.



obs.:Uma Lista vazia com cabeça terá sempre o elemento cabeça.

Listas Lineares em Alocação Dinâmica

Criação de uma lista vazia sem cabeça

```
1 celula *T;
2 T = NULL;
```

Criação de uma lista vazia com cabeça

```
1 celula cabeca;
2 celula *T;
3 cabeca->link = NULL;
4 T = &cabeca;
```

Outra forma de criar uma lista vazia com cabeça

```
1 celula *T;
2 T = malloc (sizeof(celula));
3 T->link = NULL;
```

Listas Lineares em Alocação Dinâmica

Imprime o conteúdo de uma lista T sem cabeça

```
1 void Imprime (celula *T) {
2   celula *p;
3   for (p = T; p != NULL; p = p->link)
4     printf ("%d\n", p->info);
5 }
```

Imprime o conteúdo de uma lista T com cabeça

```
1 void Imprime (celula *T) {
2   celula *p;
3   for (p = T->link; p != NULL; p = p->link)
4     printf ("%d\n", p->info);
5 }
```

Listas Lineares em Alocação Dinâmica

Busca o elemento x em uma lista T com cabeça

```
1 celula *Busca (int x, celula *T) {
2   celula *p;
3   p = T->link;
4   while (p != NULL && p->info != x)
5       p = p->link;
6   return p;
7 }
```

Devolve o endereço de uma célula que contém x ou devolve NULL se tal célula não existe.

Listas Lineares em Alocação Dinâmica

Remoção de uma célula apontada por p

```
1 void Remove (celula *p) {
2     celula *lixo;
3     lixo = p->link;
4     p->link = lixo->link;
5     free (lixo);
6 }
```

Recebe o endereço p de uma célula em uma lista e remove da lista a célula p->link. Supõe que p != NULL e p->link != NULL.

Listas Lineares em Alocação Dinâmica

Insere uma célula em uma lista entre a célula p e a seguinte

```
1 void Insere (int y, célula *p) {
2     celula *nova;
3     nova = malloc (sizeof (celula));
4     nova->info = y;
5     nova->link = p->link;
6     p->link = nova;
7 }
```

Supõe que p != NULL. A nova célula terá conteúdo y.

Listas Lineares em Alocação Dinâmica

Busca seguida de remoção em uma lista T com cabeça

```
1  void Busca_Remove (int x, celula *T) {
2    celula *p, *q;
3    p = T;
4    q = T->link;
5    while (q != NULL && q->info != x) {
6      p = q;
7      q = q->link;
8    }
9    if (q != NULL) {
10     p->link = q->link;
11     free (q);
12   }
13 }
```

Recebe uma lista T com cabeça e remove da lista a primeira célula que contiver x, se tal célula existir.

Listas Lineares em Alocação Dinâmica

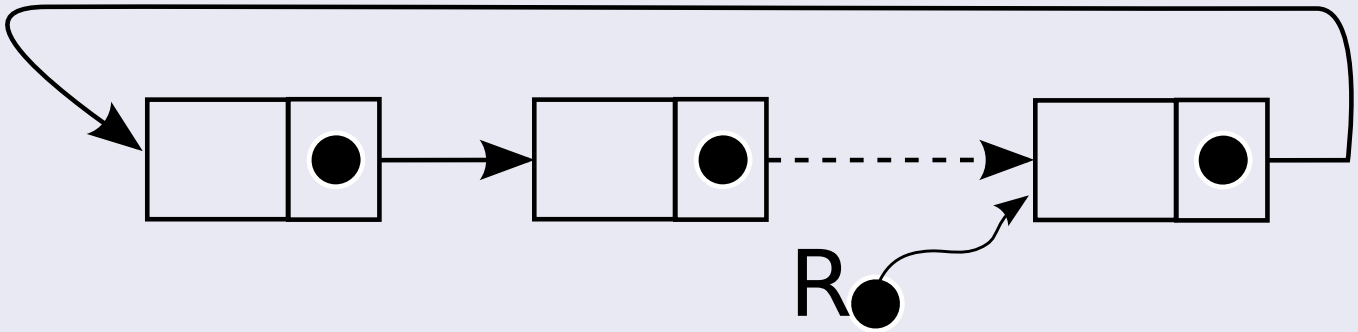
Busca seguida de inserção em uma lista T com cabeça

```
1  void Busca_Insere (int y, int x, celula *T) {
2      celula *p, *q, *nova;
3      nova = malloc (sizeof (celula));
4      nova->info = y;
5      p = T;
6      q = T->link;
7      while (q != NULL && q->info != x) {
8          p = q;
9          q = q->link;
10     }
11     nova->link = q;
12     p->link = nova;
13 }
```

Recebe uma lista T com cabeça e insere uma nova célula com conteúdo y imediatamente antes da primeira célula que contiver x. Se nenhuma célula contiver x, a nova célula será inserida no fim da lista.

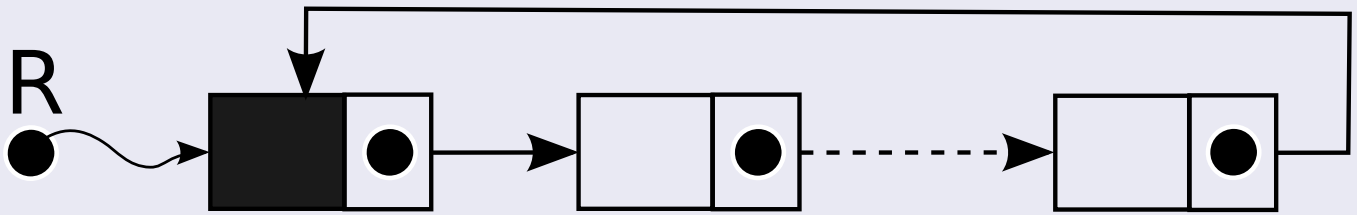
Listas circulares sem cabeça

Em uma lista circular ligada o último elemento da lista referência o primeiro elemento. Neste caso, podemos definir a variável apontadora R apontando para o último elemento, pois $R->link$ dará acesso ao primeiro elemento da lista. Se $R = NULL$, a lista está vazia.

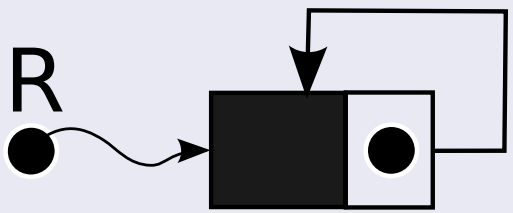


Listas circulares com cabeça

Na busca de um elemento em uma lista circular um modo de verificar se todos os elementos da lista já foram testados é usar a célula *cabeça* da lista. Assim, a única informação disponível a célula *cabeça* é algum valor especial usado somente para tal finalidade, ou seja, a marcação do início da lista:



Uma Lista circular com cabeça vazia terá sempre a célula (*cabeça*):



Listas duplamente encadeadas

Em alguns casos pode ser interessante definir listas com ponteiros que apontam para o próximo elemento e para o elemento anterior, tais listas são denominadas **duplamente encadeadas**.

Estrutura básica de uma lista duplamente encadeada:

```
1 typedef struct elem
2 {
3     int info ;
4     struct elem *prox ;
5     struct elem *ant ;
6 } celula;
```

Exercício

1. Escrever um programa completo para manipulação de uma Lista Circular Encadeada com nó cabeça. (pelo menos a busca, inserção e remoção)
2. Escrever um programa completo para manipulação de uma Lista Circular Encadeada no qual os elementos estão ordenados.
3. Escrever um programa completo para manipulação de uma Lista Circular duplamente encadeada. (pelo menos a busca, inserção e remoção)
4. Ler capítulo 2 do livro texto e resolver os exercícios.
   Livro Texto: SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

Bibliografia

SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

FEOFILOFF, P. Algoritmos em Linguagem C, Campus, 2009.

SHILDT, H. C completo e Total, Makron Books, 1996.

ZIVIANI, N. Projeto de Algoritmos: com implementações em Pascal e C, 2^a edição, Cengage Learning, 2009.