Algoritmos e Estruturas de Dados: Lista Simplesmente Encadeada

Rômulo Silva de Oliveira
Departamento de Automação e Sistemas – DAS – UFSC

romulo@das.ufsc.br
http://www.das.ufsc.br/~romulo
Maio/2011

Referências

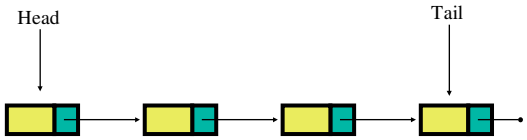
- Mastering Algorithms with C
  Kyle Loudon
  O´Reilly, 1999
- Livros de algoritmos e estruturas de dados em geral

Lista Simplesmente Encadeada – Introdução 1/2

- É a estrutura de dados mais fundamental
- Nela os elementos são encadeados em alguma ordem
- Facilita inserção e remoção em relação aos arrays
- Usa alocação dinâmica de memória
- Aplicações são as mais variadas

Lista Simplesmente Encadeada – Introdução 2/2

- Cada elemento aponta para o próximo da lista
- Último elemento aponta para NULL
- Head aponta para o primeiro elemento
- Tail aponta para o último elemento
- Pesquisa exige caminhar pela lista do início ao fim



Lista Simplesmente Encadeada – Interface 1/11

- ***list_init***
- void list_init(List **list*, void (**destroy*)(void **data*));
- **Retorno**
  - Nenhum.
- **Descrição**
  - Inicializa a lista encadeada especificada por *list*. Esta operação deve ser chamada para uma lista encadeada antes da lista poder ser usada com qualquer outra operação. O argumento *destroy* fornece uma forma para liberar dinamicamente dados alocados quando *list_destroy* é chamada. Por exemplo, se a lista contem dados alocados dinamicamente usando *malloc*, *destroy* deve ser definido como *free* para liberar os dados quando a lista encadeada for destruida. Para dados estruturados contendo vários membros alocados dinamicamente, *destroy* deveria ser associada com uma função do usuário que chama *free* para cada membro alocado dinamicamente assim como para a própria estrutura. Para uma lista encadeada contendo dados que não deveriam ser liberados, *destroy* deverá ser feita igual a NULL.

Lista Simplesmente Encadeada – Interface 2/11

- ***list_destroy***
- void list_destroy(List **list*);
- **Retorno**
  - Nenhum.
- **Descrição**
  - Destrói a lista encadeada especificada por *list*. Nenhuma outra operação é permitida após a chamada de *list_destroy* a não ser que *list_init* seja chamada novamente. A operação *list_destroy* remove todos os elementos de uma lista encadeada e chama a função passada como *destroy* para *list_init* uma vez para cada elemento quando o mesmo é removido, desde que *destroy* não seja NULL.

Lista Simplesmente Encadeada – Interface 3/11

- ***list_ins_next***
- int list_ins_next(List **list*, ListElmt **element*, const void **data*);
- **Retorno**
  - 0 se a inserção tiver sucesso, ou -1 caso contrário.
- **Descrição**
  - Insere um elemento logo após *element* na lista encadeada especificada por *list*. Se *element* é NULL, o novo elemento é inserido na cabeça da lista. O novo elemento contem um pointer para *data*, logo a memória referenciada por *data* deveria manter-se válida enquanto o elemento permanece na lista. É responsabilidade do chamador gerenciar a memória associada com *data*.

Lista Simplesmente Encadeada – Interface 4/11

- ***list_rem_next***
- int list_rem_next(List **list*, ListElmt **element*, void ***data*);
- **Retorno**
  - 0 se a remoção do elemento teve sucesso, ou -1 caso contrário.
- **Descrição**
  - Remove o elemento logo após *element* da lista encadeada especificada por *list*. Se *element* é NULL, o elemento na cabeça da lista é removido. Após o retorno, *data* aponta para os dados armazenados no elemento que foi removido. É responsabilidade do chamador gerenciar a memória associada com os dados.

Lista Simplesmente Encadeada – Interface 5/11

- ***list_size***
- int list_size(const List **list*);
- **Retorno**
  - Número de elementos na lista.
- **Descrição**
  - Macro que avalia o número de elementos na lista encadeada especificada por *list*.

Lista Simplesmente Encadeada – Interface 6/11

- ***list_head***
- ListElmt *list_head(const List **list*);
- **Retorno**
  - Elemento na cabeça da lista.
- **Descrição**
  - Macro que obtem o elemento na cabeça da lista encadeada especificada por *list*.

Lista Simplesmente Encadeada – Interface 7/11

- ***list_tail***
- ListElmt *list_tail(const List **list*);
- **Retorno**
  - Elemento na cauda da lista.
- **Descrição**
  - Macro que obtem o elemento na cauda da lista encadeada especificada por *list*.

Lista Simplesmente Encadeada – Interface 8/11

- ***list_is_head***
- int list_is_head(const ListElmt **element*);
- **Retorno**
  - 1 se o elemento está na cabeça da lista, ou 0 caso contrário.
- **Descrição**
  - Macro que determina se o elemento especificado como *element* está na cabeça da lista encadeada.

Lista Simplesmente Encadeada – Interface 9/11

- ***list_is_tail***
- int list_is_tail(const ListElmt *_element_);
- **Retorno**
  - 1 se o elemento está na cauda da lista, ou 0 caso contrário.
- **Descrição**
  - Macro que determina se o elemento especificado como *element* está na cauda da lista encadeada.

Lista Simplesmente Encadeada – Interface 10/11

- ***list_data***
- void *list_data(const ListElmt *_element_);
- **Retorno**
  - Dados armazenados no elemento.
- **Descrição**
  - Macro que gera os dados armazenados no elemento de uma lista encadeada especificada por *element*.

Lista Simplesmente Encadeada – Interface 11/11

- ***list_next***
- ListElmt *list_next(const ListElmt *_element_);
- **Retorno**
  - Elemento seguinte ao elemento especificado.
- **Descrição**
  - Macro que obtem o elemento de uma lista encadeada imediatamente ap ós o elemento especificado por *element*.

Lista Simplesmente Encadeada – Header 1/4

```
#ifndef LIST_H
#define LIST_H

#include <stdlib.h>

// Define a structure for linked list elements.

typedef struct ListElmt_ {

    void            *data;
    struct ListElmt_   *next;
} ListElmt;
```


Lista Simplesmente Encadeada – Header 2/4

```
// Define a structure for linked lists.

typedef struct List_ {

    int             size;

    int             (*match)(const void *key1, const void *key2);
    void            (*destroy)(void *data);

    ListElmt        *head;
    ListElmt        *tail;

} List;
```


Lista Simplesmente Encadeada – Header 3/4

```
//   Public Interface

void list_init(List *list, void (*destroy)(void *data));

void list_destroy(List *list);

int list_ins_next(List *list, ListElmt *element, const void *data);

int list_rem_next(List *list, ListElmt *element, void **data);
```

Lista Simplesmente Encadeada – Header 4/4

```
#define list_size(list) ((list)->size)

#define list_head(list) ((list)->head)

#define list_tail(list) ((list)->tail)

#define list_is_head(list, element) ((element) == (list)->head ? 1 : 0)

#define list_is_tail(element) ((element)->next == NULL ? 1 : 0)

#define list_data(element) ((element)->data)

#define list_next(element) ((element)->next)

#endif
```


Lista Simplesmente Encadeada – Implementação 1/4

```
#include <stdlib.h>
#include <string.h>

#include "list.h"


void list_init(List *list, void (*destroy)(void *data)) {

    list->size = 0;
    list->destroy = destroy;
    list->head = NULL;
    list->tail = NULL;
    return;
}
```


Lista Simplesmente Encadeada – Implementação 2/4

```
void list_destroy(List *list) {

    void    *data;

    while (list_size(list) > 0) {

        if (list_rem_next(list, NULL, (void **)&data) == 0  &&  list->destroy != NULL) {

                // Call a user-defined function to free dynamically allocated data.
                list->destroy(data);
         }

    }

    // clear the structure as a precaution

    memset(list, 0, sizeof(List));

    return;
}
```


Lista Simplesmente Encadeada – Implementação 3/4

```
int list_ins_next(List *list, ListElmt *element, const void *data) {
    ListElmt    *new_element;

    if ((new_element = (ListElmt *) malloc(sizeof(ListElmt))) == NULL)      return -1;

    new_element->data = (void *) data;

    if (element == NULL) {
        if (list_size(list) == 0)
            list->tail = new_element;
        new_element->next = list->head;
        list->head = new_element;
        }
    else {
        if (element->next == NULL)
            list->tail = new_element;
        new_element->next = element->next;
        element->next = new_element;
        }
    list->size++;
    return 0;
}
```


Lista Simplesmente Encadeada – Implementação 4/4

```
int list_rem_next(List *list, ListElmt *element, void **data) {
    ListElmt    *old_element;
    if (list_size(list) == 0)       return -1;

    if (element == NULL) {
        *data = list->head->data;
        old_element = list->head;
        list->head = list->head->next;
        if (list_size(list) == 1)           list->tail = NULL;
    }
    else {
        if (element->next == NULL)      return -1;
        *data = element->next->data;
        old_element = element->next;
        element->next = element->next->next;
        if (element->next == NULL)      list->tail = element;
    }
    free(old_element);
    list->size--;
    return 0;
}
```


Lista Simplesmente Encadeada – Sumário

- Mais fundamental das estruturas de dados
- Simples de implementar
- Usada para muita coisa

- Mas não resolve todos os problemas