Algoritmos e Estruturas de Dados: Lista Duplamente Encadeada

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

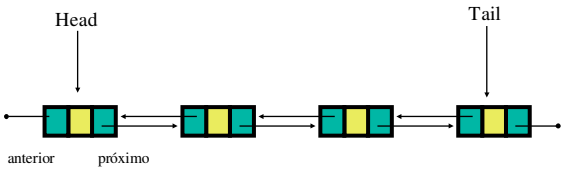
romulo@das.ufsc.br
http://www.das.ufsc.br/~romulo
Maio/2011

Referências

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

Lista Duplamente Encadeada – Introdução

- Cada elemento possui 3 partes:
  - Dados, pointer para o próximo, pointer para o anterior
- Último elemento tem NULL como próximo
- Primeiro elemento tem NULL como anterior
- É possível caminhar pela lista nos dois sentidos



Lista Duplamente Encadeada – Interface 1/13

- ***dlist_init***
- void dlist_init(DList **list*, void (**destroy*)(void **data*));
- **Retorno**
  - Nenhum.
- **Descrição**
  - Inicializa a lista duplamente encadeada especificada por *list*. Esta operação deve ser chamada para uma lista duplamente encadeada antes que a lista possa ser usada com qualquer outra operação. O argumento *destroy* fornece uma maneira para liberar dinamicamente dados alocados quando *dlist_destroy* for chamada. Ele funciona de uma maneira similar aquela descrita para *list_destroy*. Para uma lista duplamente encadeada contendo dados que não devem ser liberados, *destroy* deve receber NULL.

Lista Duplamente Encadeada – Interface 2/13

- ***dlist_destroy***
- void dlist_destroy(DList **list*);
- **Retorno**
  - Nenhum.
- **Descrição**
  - Destrói a lista duplamente encadeada especificada por *list*. Nenhuma outra operação é permitida após chamar *dlist_destroy* a não ser que *dlist_init* seja chamada novamente. A operação *dlist_destroy* remove todos os elementos de uma lista duplamente encadeada e chama a função passada como *destroy* para *dlist_init* uma vez para cada elemento quando ele é removido, desde que *destroy* não tenha sido feito NULL.

Lista Duplamente Encadeada – Interface 3/13

- ***dlist_ins_next***
- int dlist_ins_next(DList **list*, DListElmt **element*, const void **data*);
- **Retorno**
  - 0 se a inserção do elemento teve sucesso, ou -1 caso contrário.
- **Descrição**
  - Insere um elemento imediatamente após *element* em uma lista duplamente encadeada especificada por *list*. Quando inserindo em uma lista vazia, *element* pode apontar para qualquer lugar, mas deveria ser NULL para evitar confusão. O novo elemento contém um pointer para *data*, logo a memória referenciada por *data* deveria permanecer válida pelo tempo que o elemento permanecer na lista. É responsabilidade do chamador gerenciar a memória associada com *data*.

Lista Duplamente Encadeada – Interface 4/13

- ***dlist_ins_prev***
- int dlist_ins_prev(DList *_list_, DListElmt *_element_, const void *_data_);
- **Retorno**
  - 0 se a inserção do elemento tiver sucesso, ou -1 caso contrário.
- **Descrição**
  - Insere um elemento imediatamente antes de *element* em uma lista duplamente encadeada especificada por *list*. Quando inserindo em uma lista vazia, *element* pode apontar para qualquer lugar, mas deveria ser NULL para evitar confus ão. O novo elemento contem um pointer para *data*, assim a memória referenciada por *data* deveria permanecer válida pelo tempo que o elemento permanece na lista. É responsabilidade do chamador gerenciar a mem ória associada com *data*.

Lista Duplamente Encadeada – Interface 5/13

- ***dlist_remove***
- int dlist_remove(DList *_list_, DListElmt *_element_, void **_data_);
- **Retorno**
  - 0 se a remoção do elemento tiver sucesso, ou -1 caso contrário.
- **Descrição**
  - Remove o elemento especificado por *element* de uma lista duplamente encadeada especificada por *list*. Após o retorno, *data* aponta para os dados armazenados no elemento que foi removido. É responsabilidade do chamador gerenciar a mem ória associada com os dados.

Lista Duplamente Encadeada – Interface 6/13

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

Lista Duplamente Encadeada – Interface 7/13

- ***dlist_head***
- DListElmt *dlist_head(const DList *_list_);
- **Retorno**
  - O elemento na cabeça da lista.
- **Descrição**
  - Macro que obtem o elemento na cabeça da lista duplamente encadeada especificada por *list*.

Lista Duplamente Encadeada – Interface 8/13

- ***dlist_tail***
- DListElmt *dlist_tail(const DList *_list_);
- **Retorno**
  - Elemento na cauda da lista.
- **Descrição**
  - Macro que obtem o elemento na cauda da lista duplamente encadeada especificada por *list.*

Lista Duplamente Encadeada – Interface 9/13

- ***dlist_is_head***
- int dlist_is_head(const DListElmt *_element_);
- **Retorno**
  - 1 se o elemento está na cabeça da lista, ou 0 caso contrário.
- **Descrição**
  - Macro que determina se o elemento especificado por *element* está na cabeça da lista duplamente encadeada.

Lista Duplamente Encadeada – Interface 10/13

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

Lista Duplamente Encadeada – Interface 11/13

- ***dlist_data***
- void *dlist_data(const DListElmt *_element_);
- **Retorno**
  - Dados armazenados no elemento.
- **Descrição**
  - Macro que obtem os dados armazenados no elemento de uma lista duplamente encadeada especificado por *element.*

Lista Duplamente Encadeada – Interface 12/13

- ***dlist_next***
- DListElmt *dlist_next(const DListElmt *_element_);
- **Retorno**
  - Elemento imediatamente após o elemento especificado.
- **Descrição**
  - Macro que obtem o elemento de uma lista duplamente encadeada imediatamente após o elemento especificado por *element.*

Lista Duplamente Encadeada – Interface 13/13

- ***dlist_prev***
- DListElmt *dlist_prev(const DListElmt *_element_);
- **Retorno**
  - Elemento que precede o elemento especificado.
- **Descrição**
  - Macro que obtem o elemento de uma lista duplamente encadeada que precede o elemento especificado por *element*.

Lista Duplamente Encadeada – Header 1/3

```
#ifndef DLIST_H
#define DLIST_H

#include <stdlib.h>

// Define a structure for doubly-linked list elements.
typedef struct DListElmt_ {
    void            *data;
    struct DListElmt_  *prev;
    struct DListElmt_  *next;
} DListElmt;


// Define a structure for doubly-linked lists.
typedef struct DList_ {
    int             size;
    int             (*match)(const void *key1, const void *key2);
    void            (*destroy)(void *data);

    DListElmt       *head;
    DListElmt       *tail;
} DList;
```


Lista Duplamente Encadeada – Header 2/3

```
void dlist_init(DList *list, void (*destroy)(void *data));

void dlist_destroy(DList *list);

int dlist_ins_next(DList *list, DListElmt *element, const void *data);

int dlist_ins_prev(DList *list, DListElmt *element, const void *data);

int dlist_remove(DList *list, DListElmt *element, void **data);
```

Lista Duplamente Encadeada – Header 3/3

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

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

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

#define dlist_is_head(element) ((element)->prev == NULL ? 1 : 0)

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

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

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

#define dlist_prev(element) ((element)->prev)

#endif
```


Lista Duplamente Encadeada – Implementação 1/8

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

#include "dlist.h"

void dlist_init(DList *list, void (*destroy)(void *data)) {

list->size = 0;
list->destroy = destroy;
list->head = NULL;
list->tail = NULL;

return;
}
```


Lista Duplamente Encadeada – Implementação 2/8

```
void dlist_destroy(DList *list) {

    void            *data;

    while (dlist_size(list) > 0) {

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

            list->destroy(data);
        }

    }

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

    return;

}
```


Lista Duplamente Encadeada – Implementação 3/8

```
int dlist_ins_next(DList *list, DListElmt *element, const void *data) {

    DListElmt       *new_element;

    if (element == NULL && dlist_size(list) != 0)
        return -1;

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

    new_element->data = (void *)data;

    if (dlist_size(list) == 0) {

        list->head = new_element;
        list->head->prev = NULL;
        list->head->next = NULL;
        list->tail = new_element;
    }
```


Lista Duplamente Encadeada – Implementação 4/8

```
    else {

        // Handle insertion when the list is not empty.

        new_element->next = element->next;
        new_element->prev = element;

        if (element->next == NULL)
            list->tail = new_element;
        else
            element->next->prev = new_element;

        element->next = new_element;
    }

    list->size++;

    return 0;

}
```


Lista Duplamente Encadeada – Implementação 5/8

```
int dlist_ins_prev(DList *list, DListElmt *element, const void *data) {

    DListElmt       *new_element;

    if (element == NULL && dlist_size(list) != 0)
        return -1;

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

    new_element->data = (void *)data;

    if (dlist_size(list) == 0) {

        list->head = new_element;
        list->head->prev = NULL;
        list->head->next = NULL;
        list->tail = new_element;
    }
```

Lista Duplamente Encadeada – Implementação 6/8

```
    else {
        // Handle insertion when the list is not empty.

        new_element->next = element;
        new_element->prev = element->prev;

        if (element->prev == NULL)
            list->head = new_element;
        else
            element->prev->next = new_element;

        element->prev = new_element;

        }
    list->size++;

    return 0;

}
```


Lista Duplamente Encadeada – Implementação 7/8

```
int dlist_remove(DList *list, DListElmt *element, void **data) {

    if (element == NULL || dlist_size(list) == 0)
        return -1;

    *data = element->data;

    if (element == list->head) {

        list->head = element->next;

        if (list->head == NULL)
            list->tail = NULL;
        else
            element->next->prev = NULL;
  }
```


Lista Duplamente Encadeada – Implementação 8/8

```
    else {

        // Handle removal from other than the head of the list.

        element->prev->next = element->next;

        if (element->next == NULL)
            list->tail = element->prev;
        else
            element->next->prev = element->prev;
    }

    free(element);
    list->size--;

    return 0;
}
```


Lista Duplamente Encadeada – Sumário

- Simples de implementar
- Facilita manipulação no meio da lista

- Gasta 1 pointer a mais por elemento