Algoritmos e Estruturas de Dados: Lista Circular

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

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

Referências

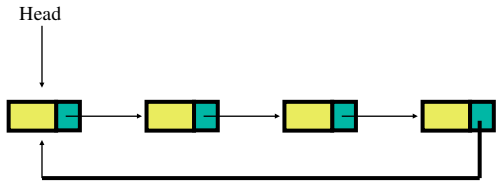
- Mastering Algorithms with C
  Kyle Loudon
  O´Reilly, 1999
- Livros de algoritmos e estruturas de dados em geral

Lista Circular – Introdução 1/2

- Pode ser com encadeamento simples ou duplo
- O último elemento aponta para o primeiro novamente
- O anterior do primeiro é o último
- Caminhar sobre uma lista que agora não tem fim
- Para algumas aplicações isto é útil

Lista Circular – Introdução 2/2

- Head é usado como uma "ponta" para entrar na lista
- Em geral caminhamento inicia do ponto onde o último caminhamento parou



Lista Circular – Interface 1/8

- ***clist_init***
- void clist_init(CList **list*, void (**destroy*)(void **data*));
- **Retorno**
  - Nenhum.
- **Descrição**
  - Inicializa a lista circular especificada por *list*. Esta operação deve ser chamada para uma lista circular antes que a lista possa ser usada com qualquer outra operação. O argumento *destroy* fornece uma maneira para liberar dinamicamente dados alocados quando *clist_destroy* é chamada. Ele funciona de maneira similar àquela descrita para *list_destroy*. No caso de uma lista circular contendo dados que não devem ser liberados, *destroy* deve ser NULL.

Lista Circular – Interface 2/8

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

Lista Circular – Interface 3/8

- ***clist_ins_next***
- int clist_ins_next(CList **list*, CListElmt **element*, const void **data*);
- **Retorno**
  - 0 se a inserção do elemento teve sucesso, ou -1 caso contrário.
- **Descrição**
  - Insere um elemento imediatamente após *element* na lista circular especificada por *list*. Quando inserindo em uma lista vazia, *element* pode apontar qualquer coisa, mas deveria ser NULL para evitar confusão. O novo elemento contem um pointer para *data*, logo a memória referenciada por *data* deve permanecer válida enquanto o elemento permanecer na lista. É responsabilidade do chamador gerenciar a memória associada com *data*.

Lista Circular – Interface 4/8

- ***clist_rem_next***
- int clist_rem_next(CList **list*, CListElmt **element*, void ***data*);
- **Retorno**
  - 0 se a remoção do elemento tiver sucesso, ou -1 caso contrário.
- **Descrição**
  - Remove o elemento imediatamente após *element* da lista circular especificada por *list*. 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 Circular – Interface 5/8

- ***clist_size***
- int clist_size(const CList **list*);
- **Retorno**
  - Número de elementos na lista.
- **Descrição**
  - Macro que avalia o número de elementos na lista circular especificada por *list.*

Lista Circular – Interface 6/8

- ***clist_head***
- CListElmt *clist_head(const CList **list*);
- **Retorno**
  - Elemento na cabeça da lista.
- **Descrição**
  - Macro que obtem o elemento na cabeça da lista circular especificada por *list.*

Lista Circular – Interface 7/8

- ***clist_data***
- void *clist_data(const CListElmt **element*);
- **Retorno**
  - Dados armazenados no elemento.
- **Descrição**
  - Macro que obtem os dados armazenados no elemento de uma lista circular especificado por *element.*

Lista Circular – Interface 8/8

- ***clist_next***
- CListElmt *clist_next(const CListElmt **element*);
- **Retorno**
  - Elemento imediatamente após o elemento especificado.
- **Descrição**
  - Macro que obtem o elemento de uma lista circular imediatamente após o elemento especificado por *element*.

Lista Circular – Header 1/2

```
#ifndef CLIST_H
#define CLIST_H

#include <stdlib.h>

typedef struct CListElmt_ {
    void            *data;
    struct CListElmt_  *next;
} CListElmt;

typedef struct CList_ {

    int             size;

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

    CListElmt       *head;
} CList;
```


Lista Circular – Header 2/2

```
void clist_init(CList *list, void (*destroy)(void *data));

void clist_destroy(CList *list);

int clist_ins_next(CList *list, CListElmt *element, const void *data);

int clist_rem_next(CList *list, CListElmt *element, void **data);

#define clist_size(list) ((list)->size)

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

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

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

#endif
```


Lista Circular – Implementação 1/4

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

#include "clist.h"


void clist_init(CList *list, void (*destroy)(void *data)) {

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

    return;

}
```


Lista Circular – Implementação 2/4

```
void clist_destroy(CList *list) {

    void            *data;

    while (clist_size(list) > 0) {

        if (clist_rem_next(list, list->head, (void **)&data) == 0           &&
                list->destroy != NULL)
        {
            list->destroy(data);
        }
    }

    memset(list, 0, sizeof(CList));
    return;
}
```


Lista Circular – Implementação 3/4

```
int clist_ins_next(CList *list, CListElmt *element, const void *data) {

    CListElmt       *new_element;

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

    new_element->data = (void *)data;

    if (clist_size(list) == 0) {
        new_element->next = new_element;
        list->head = new_element;
    }
else {
        new_element->next = element->next;
        element->next = new_element;
    }

list->size++;
return 0;
}
```


Lista Circular – Implementação 4/4

```
int clist_rem_next(CList *list, CListElmt *element, void **data) {

    CListElmt       *old_element;

    if (clist_size(list) == 0)                  return -1;

    *data = element->next->data;

    if (element->next == element) {
        old_element = element->next;
        list->head = NULL;
    }
    else {
        old_element = element->next;
        element->next = element->next->next;
    }

free(old_element);
list->size--;
return 0;
}
```

Lista Circular – Sumário

- É preciso cuidado quando caminhar na lista
- Inserção e remoção também mais cuidadosa

- Bom negócio para algumas aplicações
  - Elimina o caso particular de chegar no fim da lista