20220504-算法

1.过程描述

1.1 算法

1）双向链表

> 仅包含一些通用函数的实现

llgen.h C++ 复制代码

```
#ifndef  LLGEN_H
#define LLGEN_H

struct Node
{
    struct Node* prev;
    struct Node* next;
    void* pdata;
};

typedef struct Node* Link;

/* a linked list data structure */
struct List
{
    Link LHead;
    Link LTail;
    unsigned int LCount;
    void* (*LCreateData)(void*);
    int(*LDeleteData)(void*);
    int(*LDuplicatedNode)(Link, Link);
    int(*LNodeDataCmp)(void*, void*);
};

int AddNodeAscend(struct List*, void*);
int AddNodeAtHead(struct List*, void*);
struct List* CreateList(
    void* (*)(void*),
    int(*)(void*),
    int(*)(Link,Link),
    int(*)(void*,void*)
    );
Link CreateNode(struct List*, void*);
int DeleteNode(struct List*, Link);
Link FindNode(struct List*, void*);
Link FindNodeAscend(struct List*, void*);
Link GotoNext(struct List*, Link);
Link GotoPrev(struct List*, Link);

#endif //  LLGEN_H
```

llgen.cpp C++ 复制代码

```cpp
#include "llgen.h"

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

#define IN_LL_LIB 1
/* --Aliases to make the code more readable-- */
#define LLHead (L->LHead)
#define LLTail (L->LTail)
#define NodeCount (L->LCount)

#define CreateData (*(L->LCreateData))
#define DeleteData (*(L->LDeleteData))
#define DuplicatedNode (*(L->LDuplicatedNode))
#define NodeDataCmp (*(L->LNodeDataCmp))

int AddNodeAtHead(struct List* L, void* nd)
{
    Link pn;
    pn = CreateNode(L, nd);
    if (pn == NULL)
        return (0);
    if (LLHead == NULL)
    {
        LLHead = LLTail = pn;
    }
    else
    {
        LLHead->prev = pn;
        pn->next = LLHead;
        LLHead = pn;
    }
    NodeCount += 1;
    return(1);
}

int AddNodeAscend(struct List* L, void* nd)
{
    Link pn;
    Link prev, curr;
    struct Node dummy;
    int compare;
    pn = CreateNode(L, nd);
    if (pn == NULL)
        return (0);
```

```c
    dummy.next = LLHead;
    dummy.prev = NULL;
    if (dummy.next != NULL)
        dummy.next->prev = &dummy;
    prev = &dummy;
    curr = dummy.next;
    for (; curr != NULL; prev = curr, curr = curr->next)
    {
        compare = NodeDataCmp(pn->pdata, curr->pdata);
        if (compare <= 0)
            break;
    }
    if (curr != NULL && compare == 0)
    {
        compare = DuplicatedNode(pn, curr);
        if (compare == 2);
        else
        {
            LLHead = dummy.next;
            LLHead->prev = NULL;
            if (compare == 1)
            {
                DeleteData(pn->pdata);
                free(pn);
            }
            return(1);
        }
    }
    prev->next = pn;
    pn->prev = prev;
    pn->next = curr;
    if (curr != NULL)
        curr->prev = pn;
    else
        LLTail = pn;
    NodeCount += 1;
    LLHead = dummy.next;
    LLHead->prev = NULL;
    return(1);
}

struct List* CreateList(
    void* (*fCreateData)(void*),
    int(*fDelateData)(void*),
    int(*fDuplicatedNode)(Link, Link),
    int(*fNodeDataCmp)(void*, void*)
)
{
    struct List* pL;
    pL = (struct List*)malloc(sizeof(struct List));
    if (pL == NULL)
        return NULL;
    pL ->LHead = NULL;
    pL->LTail = NULL;
    pL->LCount = 0;
    pL->LCreateData = fCreateData;
    pL->LDeleteData = fDelateData;
    pL->LDuplicatedNode = fDuplicatedNode;
    pL->LNodeDataCmp = fNodeDataCmp;
    return (pL);
}

Link CreateNode(struct List* L, void* data)
{
    Link new_node;
    new_node = (Link)malloc(sizeof(struct Node));
    if (new_node == NULL)
        return (NULL);
    new_node->prev = NULL;
    new_node->next = NULL;

    new_node->pdata = CreateData(data);
    if (new_node->pdata == NULL)
    {
        free(new_node);
        return (NULL);
    }
    else
        return(new_node);
}

int DeleteNode(struct List* L, Link to_delete)
{
    Link pn;
    if (to_delete == NULL)
        return(0);
    if (to_delete->prev == NULL)
    {
        LLHead = to_delete->next;
        LLHead->prev = NULL;
    }
    else if (to_delete->next == NULL)
    {
        pn = to_delete->prev;
        pn->next = NULL;
        LLTail = pn;
    }
    else
    {
        pn = to_delete->prev;
        pn->next = to_delete->next;
        pn = to_delete->next;
        pn->prev = to_delete->prev;
    }
    DeleteData(to_delete->pdata);
    free(to_delete);
    NodeCount -= 1;
    return(1);
}

Link FindNode(struct List* L, void* nd)
{
    Link pcurr;
    if (LLHead == NULL)
        return(NULL);
    for (pcurr = LLHead; pcurr != NULL; pcurr = pcurr->next)
    {
        if (NodeDataCmp(nd, pcurr->pdata) == 0)
            return(pcurr);
    }
    return (NULL);
}

Link FindNodeAscend(struct List* L, void* nd)
{
    Link pcurr;
    int cmp_result;
    if (LLHead == NULL)
        return(NULL);
    for (pcurr = LLHead; pcurr != NULL; pcurr = pcurr->next)
    {
        cmp_result = NodeDataCmp(nd, pcurr->pdata);
        if (cmp_result < 0)
            return(NULL);
        if (cmp_result == 0)
            return(pcurr);
    }
    return(NULL);
}
```

2.结果输出

今天看了一点算法的东西，时间利用度很低。最近陷入了一个迷茫期，不知道要往哪个方向使力。明天要花时间好好反思一下。