

# 20220504-算法

---

## 1.过程描述

### 1.1 算法

#### 1) 双向链表

## 2.结果输出

## 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

```
1  #ifndef  LLGEN_H
2  #define  LLGEN_H
3
4  struct Node
5  {
6      struct Node* prev;
7      struct Node* next;
8      void* pdata;
9  };
10
11 typedef struct Node* Link;
12
13 /* a linked list data structure */
14 struct List
15 {
16     Link LHead;
17     Link LTail;
18     unsigned int LCount;
19     void* (*LCreateData)(void*);
20     int(*LDeleteData)(void*);
21     int(*LDuplicatedNode)(Link, Link);
22     int(*LNodeDataCmp)(void*, void*);
23 };
24
25 int AddNodeAscend(struct List*, void*);
26 int AddNodeAtHead(struct List*, void*);
27 struct List* CreateList(
28     void* (*)(void*),
29     int(*)(void*),
30     int(*)(Link,Link),
31     int(*)(void*,void*)
32 );
33 Link CreateNode(struct List*, void*);
34 int DeleteNode(struct List*, Link);
35 Link FindNode(struct List*, void*);
36 Link FindNodeAscend(struct List*, void*);
37 Link GotoNext(struct List*, Link);
38 Link GotoPrev(struct List*, Link);
39
40 #endif //  LLGEN_H
```

```
1  ▼ #include "llgen.h"
2
3  ▼ #include <stdlib.h>
4    #include <string.h>
5
6    #define IN_LL_LIB 1
7    /* --Aliases to make the code more readable-- */
8    #define LLHead (L->LHead)
9    #define LLTail (L->LTail)
10   #define NodeCount (L->LCount)
11
12   #define CreateData (*(L->LCreateData))
13   #define DeleteData (*(L->LDeleteData))
14   #define DuplicatedNode (*(L->LDuplicatedNode))
15   #define NodeDataCmp (*(L->LNodeDataCmp))
16
17   int AddNodeAtHead(struct List* L, void* nd)
18   ▼ {
19       Link pn;
20       pn = CreateNode(L, nd);
21       if (pn == NULL)
22           return (0);
23       if (LLHead == NULL)
24   ▼   {
25           LLHead = LLTail = pn;
26       }
27       else
28   ▼   {
29           LLHead->prev = pn;
30           pn->next = LLHead;
31           LLHead = pn;
32       }
33       NodeCount += 1;
34       return(1);
35   }
36
37   int AddNodeAscend(struct List* L, void* nd)
38   ▼ {
39       Link pn;
40       Link prev, curr;
41       struct Node dummy;
42       int compare;
43       pn = CreateNode(L, nd);
44       if (pn == NULL)
45           return (0);
```

```

46     dummy.next = LLHead;
47     dummy.prev = NULL;
48     if (dummy.next != NULL)
49         dummy.next->prev = &dummy;
50     prev = &dummy;
51     curr = dummy.next;
52     for (; curr != NULL; prev = curr, curr = curr->next)
53     {
54         compare = NodeDataCmp(pn->pdata, curr->pdata);
55         if (compare <= 0)
56             break;
57     }
58     if (curr != NULL && compare == 0)
59     {
60         compare = DuplicatedNode(pn, curr);
61         if (compare == 2);
62         else
63         {
64             LLHead = dummy.next;
65             LLHead->prev = NULL;
66             if (compare == 1)
67             {
68                 DeleteData(pn->pdata);
69                 free(pn);
70             }
71             return(1);
72         }
73     }
74     prev->next = pn;
75     pn->prev = prev;
76     pn->next = curr;
77     if (curr != NULL)
78         curr->prev = pn;
79     else
80         LLTail = pn;
81     NodeCount += 1;
82     LLHead = dummy.next;
83     LLHead->prev = NULL;
84     return(1);
85 }
86
87 struct List* CreateList(
88     void* (*fCreateData)(void*),
89     int(*fDeleteData)(void*),
90     int(*fDuplicatedNode)(Link, Link),
91     int(*fNodeDataCmp)(void*, void*)
92 )
93 {

```

```

94     struct List* pL;
95     pL = (struct List*)malloc(sizeof(struct List));
96     if (pL == NULL)
97         return NULL;
98     pL->LHead = NULL;
99     pL->LTail = NULL;
100    pL->LCount = 0;
101    pL->LCreateData = fCreateData;
102    pL->LDeleteData = fDeleteData;
103    pL->LDuplicatedNode = fDuplicatedNode;
104    pL->LNodeDataCmp = fNodeDataCmp;
105    return (pL);
106 }
107
108 Link CreateNode(struct List* L, void* data)
109 {
110     Link new_node;
111     new_node = (Link)malloc(sizeof(struct Node));
112     if (new_node == NULL)
113         return (NULL);
114     new_node->prev = NULL;
115     new_node->next = NULL;
116
117     new_node->pdata = CreateData(data);
118     if (new_node->pdata == NULL)
119     {
120         free(new_node);
121         return (NULL);
122     }
123     else
124         return(new_node);
125 }
126
127 int DeleteNode(struct List* L, Link to_delete)
128 {
129     Link pn;
130     if (to_delete == NULL)
131         return(0);
132     if (to_delete->prev == NULL)
133     {
134         LLHead = to_delete->next;
135         LLHead->prev = NULL;
136     }
137     else if (to_delete->next == NULL)
138     {
139         pn = to_delete->prev;
140         pn->next = NULL;
141         LLTail = pn;

```

```

142     }
143     else
144     {
145         pn = to_delete->prev;
146         pn->next = to_delete->next;
147         pn = to_delete->next;
148         pn->prev = to_delete->prev;
149     }
150     DeleteData(to_delete->pdata);
151     free(to_delete);
152     NodeCount -= 1;
153     return(1);
154 }
155
156 Link FindNode(struct List* L, void* nd)
157 {
158     Link pcurr;
159     if (LLHead == NULL)
160         return(NULL);
161     for (pcurr = LLHead; pcurr != NULL; pcurr = pcurr->next)
162     {
163         if (NodeDataCmp(nd, pcurr->pdata) == 0)
164             return(pcurr);
165     }
166     return (NULL);
167 }
168
169 Link FindNodeAscend(struct List* L, void* nd)
170 {
171     Link pcurr;
172     int cmp_result;
173     if (LLHead == NULL)
174         return(NULL);
175     for (pcurr = LLHead; pcurr != NULL; pcurr = pcurr->next)
176     {
177         cmp_result = NodeDataCmp(nd, pcurr->pdata);
178         if (cmp_result < 0)
179             return(NULL);
180         if (cmp_result == 0)
181             return(pcurr);
182     }
183     return(NULL);
184 }

```

## 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。