数据结构之线性链表

数组实现使得PrintList和Find是于线性时间执行，而FindKth则花费常数时间。然而，插入和删除的花费是昂贵的。

为了避免插入和删除的线性开销，需要允许表不是连续存储的，否则表的部分或全部需要整体移动。

1. 1.0、线性链表之makefile

make工具是一个相当不错的工具，只对有改动的文件进行编译，只需一条make指令即可完成编译。

线性链表的makefiel如下：

test.out:test.c List.h List.o

gcc test.c List.o -o test.out

List.o:List.c

gcc -c List.c -o List.o

1. 2.0、线性链表之List.h文件

List文件如下：

#ifndef \_LIST\_H

struct Node;

typedef struct Node \*pNode;

typedef pNode List;

typedef pNode Position;

List CreateEmptyList();

int IsEmpty(List L);

int IsLast(Position P);

Position Find(int x,List L);

int Delete(int x,List L);

Position FindPrevious(int x,List L); //x的前驱元

int Insert(int x,Position P);

void DeleteList(List L);

Position Header(List L);

Position First(List L);

Position Advance(List L); //使position指针增一

int Retrieve(Position P); //返回data元素

#endif

1. 3.0、线性链表之List.c

List.h的实现

#include <stdio.h>

#include <malloc.h>

#include "List.h"

//结构体

struct Node

{

int data;

struct Node \*next;

};

//init List

List CreateEmptyList()

{

List L;

L = malloc(sizeof(struct Node));

L->next = NULL;

return L;

}

//return true if Node is empty

int IsEmpty(List L)

{

return L->next == NULL;//if list is empty and it will return 1

}

//

int IsLast(Position P)

{

return P->next == NULL;//if it is last it will return 1

}

//Inster Element

int Insert(int x,Position P)

{

Position tmpCell;

tmpCell = malloc(sizeof(struct Node));

if(tmpCell == NULL)

{

return 0;

}

tmpCell->data = x;

tmpCell->next = P->next;

P->next = tmpCell;

return 1;

}

//find list

Position Find(int x,List L)

{

Position P;

P=L->next;

while(P!=NULL && P->data != x)

P=P->next;

return P;

}

//delete List

int Delete(int x,List L)

{

Position P,tmpCell;

P=FindPrevious(x,L);

if(!IsLast(P))

{

tmpCell=P->next;

P->next = tmpCell->next;

free(tmpCell);

return 1;

}

return 0;

}

Position FindPrevious(int x,List L)

{

Position P;

P=L;

while(P->next != NULL && P->next->data !=x)

P=P->next;

return P;

}

//Delete List

void DeleteList(List L)

{

Position P,tmp;

P=L->next;

L->next = NULL;

while(P!=NULL)

{

tmp = P->next;

free(P);

P=tmp;

}

}

//Header

Position Header(List L)

{

return L;

}

//Fist

Position First(List L)

{

return L->next;

}

//advance

Position Advance(List L)

{

return L->next;

}

//Retrive

int Retrieve(Position P)

{

return P->data;

}

1. 4.0、线性链表之测试代码：test.c

test.c的代码如下：

#include <stdio.h>

#include "List.h"

int main()

{

List list = CreateEmptyList();

int ifempty = IsEmpty(list);

printf("the empty of List ( 1 is empty) : %d \n",ifempty);

int a =Insert(5,list);

printf("the insert return is (1 is suc): %d \n",a);

Insert(35,list);

Insert(34,list);

Insert(84,list);

Position pis=Find(34,list);

printf("-------find 34-----------\n");

printf("find the data is %d\n",Retrieve(pis));

printf("-------Delete 34---------\n");

printf("Delete 34 ,the return number is (0 is fale): %d \n",Delete(34,list));

printf("-------find 34 againt----\n");

Position p2=Find(34,list);

if(p2==NULL)

printf("No find 34!!\n");

printf(">>>>>>> print all element <<<<<<<\n");

Position p3=Advance(list);

while(p3!=NULL)

{

printf("print all data: %d \n",Retrieve(p3));

p3=Advance(p3);

}

printf("\n");

printf(">>>>>>>>delete list<<<<<<<<<\n");

DeleteList(list);

printf("list is empty? return number is :%d \n",IsEmpty(list));

return 0;

}

1. 5.0、输出结果

the empty of List ( 1 is empty) : 1

the insert return is (1 is suc): 1

-------find 34-----------

find the data is 34

-------Delete 34---------

Delete 34 ,the return number is (0 is fale): 1

-------find 34 againt----

No find 34!!

>>>>>>> print all element <<<<<<<

print all data: 84

print all data: 35

print all data: 5

>>>>>>>>delete list<<<<<<<<<

list is empty? return number is :1

基本实现了线性链表的初始化，插入，删除，等功能。

参考资料：

《数据结构与算法分析-c语言描述》 ----Mark Allen Weiss

<http://blog.csdn.net/nethanhan/article/details/9670577>