伪代码：

节点中增加Freq，初始值为0；

写一个RankFreq（L），将链表按Freq大小倒序排序

设置临时指针p，初始值为L的Head；

比较p的Freq和P->Next的Freq大小，如果小就往后移；

判定p为表头的情况；

判定p为表中的情况；

判定p->next为表尾的情况；

循环i次，i为表长。

写一个Locate（L，X），将数值为X的节点的Freq加一，最后RankFreq（L）一下

代码：

#include <iostream>

using namespace std;

class Node {

private:

int Number;

Node\* Next;

Node\* Prior;

int Freq;

public:

Node() {

this->Number = 0;

this->Next = nullptr;

this->Prior = nullptr;

this->Freq = 0;

}

Node(const int& n);

const int& GetNumber() const {

return Number;

}

const Node\* GetNext()const {

return Next;

}

const Node\* GetPrior()const {

return Prior;

}

friend class LinkedList;

};

Node::Node(const int& n) {

this->Number = n;

this->Next = nullptr;

this->Prior = nullptr;

}

class LinkedList {

private:

Node\* Head;

Node\* Tail;

public:

LinkedList() {

Head = nullptr;

Tail = nullptr;

}

LinkedList(LinkedList& L1) {

Node\* Count = L1.Head;

while (Count) {

this->Add(Count->Number);

Count = Count->Next;

}

}

void Clear();//清除堆空间

~LinkedList() { Clear(); }//析构函数

const Node\* GetHead()const { return Head; }

Node\* Add(const int& Data);//插入在尾部

const int GetCount();//计总数

const int GetThisCount(Node\* This);

void RankFreq(LinkedList& L);//按freq倒序排序

void Locate(LinkedList& L, int X);

};

void LinkedList::Clear() {

while (Head) {

Node\* newHead = Head->Next;

delete Head;

Head = newHead;

}

}

Node\* LinkedList::Add(const int& Data) {

Node\* newNode = new Node(Data);

if (!Tail)

Head = Tail = newNode;

else

{

Tail->Next = newNode;

newNode->Prior = Tail;

Tail = newNode;

}

return newNode;

}

const int LinkedList::GetCount() {

int i = 0;

for (Node\* Count = this->Head; Count; Count = Count->Next)i++;

return i;

}

const int LinkedList::GetThisCount(Node\* This) {

int i = 1;

for (Node\* Count = Head; (Count) && (Count != This); Count = Count->Next)i++;

return i;

}

void LinkedList::RankFreq(LinkedList& L) {

for (int i = 1; i <= L.GetCount(); i++) {

for (Node\* p = L.Head; p && p->Next;) {

if (p == L.Head)

if (p->Next->Freq > p->Freq) {

L.Head = p->Next;

if (p->Next->Next) {

p->Next->Next->Prior = p;

p->Prior = p->Next;

p = p->Next;

p->Prior->Next = p->Next;

p->Next = p->Prior;

p->Prior = nullptr;

p = p->Next;

}

else {

L.Head = p->Next;

p->Prior = p->Next;

p = p->Next;

p->Next = p->Prior;

p->Prior->Next = nullptr;

p->Prior = nullptr;

L.Tail = p->Next;

}

}

else p = p->Next;

else {

if (p->Next->Freq > p->Freq) {

if (p->Next->Next) {

p->Prior->Next = p->Next;

p->Next->Prior = p->Prior;

p->Prior = p->Next;

p->Next = p->Next->Next;

p->Next->Prior->Next = p;

p->Next->Prior = p;

p = p->Next;

}

else {

p->Prior->Next = p->Next;

p->Next->Prior = p->Prior;

p->Prior = p->Next;

p->Next->Next = p;

p->Next = nullptr;

L.Tail = p;

}

}

else p = p->Next;

}

}

}

}

void LinkedList::Locate(LinkedList& L, int X) {

for (Node\* p = L.Head; p ;p=p->Next) {

if (p->Number == X)p->Freq++;

}

L.RankFreq(L);

}

ostream& operator << (ostream& o, const LinkedList& list)

{

if (list.GetHead() != nullptr) {

for (const Node\* p = list.GetHead();

p;

p = p->GetNext())

{

o << "[" << p->GetNumber() << "]";

if (p->GetNext())

o << " -> ";

}

}

else {

o << "This list is empty.";

}

return o;

}

int main()

{

LinkedList Mylist1;

Mylist1.Add(1);

Mylist1.Add(2);

Mylist1.Add(7);

Mylist1.Add(16);

Mylist1.Add(22);

cout << "Mylist1 is" << endl;

cout << Mylist1 << endl;

Mylist1.Locate(Mylist1,2);

cout << "Locate(Mylist1,2)" << endl;

cout << Mylist1 << endl;

Mylist1.Locate(Mylist1, 7);

cout << "Locate(Mylist1,7)" << endl;

Mylist1.Locate(Mylist1, 7);

cout << "Locate(Mylist1,7)" << endl;

cout << Mylist1 << endl;

Mylist1.Locate(Mylist1, 16);

cout << "Locate(Mylist1,16)" << endl;

Mylist1.Locate(Mylist1, 16);

cout << "Locate(Mylist1,16)" << endl;

cout << Mylist1 << endl;

}

运行结果：

