中 国 矿 业 大 学

2018 级《数据结构与算法分析》课程作业

学生姓名 王茂凯

学　　号 04181425

中国矿业大学信控学院

1. 线性表可用顺序存储结构或链式存储结构。那么，（1）两种存储表示各有哪些主要优缺点？（2）如果有 n 个表同时并存，并且在处理过程中各表的长度会动态发生变化，表的总数也可能自动改变，在此情况下，应选用哪种存储表示？为什么？（3）若表的总数基本稳定，并且很少进行插入和删除，但要求以最快的速度存取表中的元素，这时，应采用哪种存储表示？为什么？

(1)

顺序存储结构的底层是数组,存储空间是一次性分配的且是连续的存储密度等于 1,在增加元素和删除元素方面,每次增删都需要移动其它元素,时间复杂度为 O(n),但是顺序表可以随机存取,随机查询的时间复杂度为 O(1)

链式存储结构的底层是指针域和数据域的结构体,存储空间是多次分配的,存储密度小于 1,在增删元素方面,不需要移动元素,只需要修改指针,所以时间复杂度为 O(1),但在查询元素时,只能从头节点开始顺序存取,时间复杂度为 O(n)

(2)链式存储,因为表的长度会动态发生变化,而链表时通过指针连接的,空间为多次分配,且增删时复杂度较低,性能更好

(3)顺序存储,表的总数基本稳定,可以提前一次性分配空间,很少进行复杂度较高的增删操作,而顺序结构存取时复杂度为 O(1),性能更好

2. 分别对顺序表和单链表，完成以下操作（函数）：

（1）插入操作（函数）insert；

（2）删除操作（函数）delete；

（3）返回某元素位置操作（函数）locate；

（4）统计给定值 x 的所有元素操作（函数）number；

3. 写一个交换单向链表中位置 P 和 Next(P)的元素的算法。

```
//顺序表
//元素位置从 0 开始
#include <iostream>
using namespace std;
//顺序表模板
template <typename T>
class sqlist
{
public:
    sqlist(int size = 2) : _first(new T[size]()),
                           _last(_first),
                           _end(_first + size)
    {
    }
    ~sqlist() //析构函数
    {
        delete[] _first;
    }
    void push_back(const T &val)
    {
        if (full())
            expand();
        *_last++ = val;
    }
    void insert(const int &pos, const T &val) //在 pos 位置插入元素 val
    {
        if (pos < 0 || pos > size()) //判断插入是否合理
        {
            throw "the pos error";
        }
        //从最后一个元素到 pos 位置的元素后移一位
        for (int i = size() - 1; i >= pos; --i)
            _first[i + 1] = _first[i];
        _first[pos] = val; //插入元素
        ++_last;
        if (full())
            expand();
    }
```

```
    void del(const int &pos) //删除 pos 位置的元素
    {
        if (pos < 0 || pos > size())
        {
            throw "the pos error";
        }
        //从 pos 之后的元素向前移动一位
        for (int i = pos; i < num(); ++i)
            _first[i] = _first[i + 1];
        _first[num() - 1] = 0;
        --_last;
    }
    void locate(const T &val) //返回元素 val 的位置
    {
        bool flag = true;
        for (int i = 0; i < size(); ++i)
        {
            if (_first[i] == val)
            {
                cout << i << endl;
                flag = false;
            }
        }
        if (flag)
            cout << "no val" << endl;
    }
    void number(const T &val) //返回元素 val 的个数
    {
        int count = 0;
        for (int i = 0; i < num(); ++i)
            if (_first[i] == val)
                ++count;
        cout << count << endl;
    }
    void print()
    {
        if (empty())
            cout << "the sqlist is empty" << endl;
        else
        {
            for (int i = 0; i < num(); ++i)
                cout << _first[i] << " ";
            cout << endl;
        }
```

```
    }
    bool full() { return _last == _end; }    //判断是否满
    bool empty() { return _first == _last; } //判断是否为空
    int size() { return _end - _first; }     //返回数组大小
    int num() { return _last - _first; }     //返回元素个数
private:
    T *_first;    //指向第一个元素
    T *_last;     //指向最后一个元素的后继位置
    T *_end;      //指向数组空间的后继位置
    void expand() //扩容
    {
        int _size = _end - _first;
        T *temp = new T[2 * _size](); //二倍动态扩容
        for (int i = 0; i < _size; ++i)
            temp[i] = _first[i];
        delete[] _first;
        _first = temp;
        _last = _first + _size;
        _end = _first + 2 * _size;
    }
};
int main()
{
    sqlist<int> sq; //默认开辟 2 个空间
    sq.push_back(5);
    sq.push_back(5);
    sq.push_back(6);
    sq.push_back(7);
    sq.print(); //5 5 6 7
    sq.del(0);
    sq.print();   //5 6 7
    sq.locate(5); //0
    sq.insert(0, 5);
    sq.print(); //5 5 6 7
    sq.insert(1, 4);
    sq.print();   //5 4 5 6 7
    sq.number(5); //2
    return 0;
}
```

```
5 5 6 7
5 6 7
0
5 5 6 7
5 4 5 6 7
2
```

```cpp
//单向链表
//第 0 个位置为头结点
#include <iostream>
using namespace std;

template <typename T>
struct listnode
{
    T data;
    listnode *next;
};
template <typename T>
class linklist
{
public:
    linklist() : _head(new listnode<T>()),
                 _ptr(_head),
                 _size(0)
    {
    }
    ~linklist()
    {
        listnode<T> *p = _head->next;
        for (int i = 0; i < _size; ++i)
        {
            if (p != nullptr)
                delete p;
            p = p->next;
        }
    }
    void push_back(const T &val) //尾插
    {
        listnode<T> *temp = new listnode<T>();
        temp->data = val;
        temp->next = nullptr;
        _ptr->next = temp;
        _ptr = temp;
        _size++;
    }
    listnode<T> *findnode(int pos) //返回 pos 位置的结点
    {
        listnode<T> *p = _head->next;
        if (pos < 0)
            throw "pos is error";
```

```
        if (empty())
            throw "list is empty";
        if (pos == 0)
            return _head;
        while (--pos)
            p = p->next;
        return p;
    }
    void insert(const int &pos, const T &val) //在 pos 位置插入元素 val
    {
        if (pos <= 0 || pos > _size + 1)
            throw "pos id error";
        listnode<T> *p = findnode(pos - 1);
        listnode<T> *temp = new listnode<T>();
        temp->data = val;
        temp->next = p->next;
        p->next = temp;
        _size++;
    }
    void del(const int &pos) //删除 pos 位置的结点
    {
        if (pos <= 0 || pos > _size)
            throw "pos id error";
        listnode<T> *p = findnode(pos - 1);
        listnode<T> *temp = findnode(pos);
        p->next = temp->next;
        delete temp;
        --_size;
    }
    void locate(const T &val) //返回某元素的位置
    {
        listnode<T> *p = _head->next;
        bool flag = true;
        for (int i = 1; i <= _size; ++i)
        {
            if (p->data == val)
            {
                cout << i << endl;
                flag = false;
            }
            p = p->next;
        }
        if (flag) //没有找到
            cout << "no val" << endl;
```

```
    }
    void number(const T &val) //返回元素 val 的个数
    {
        listnode<T> *p = _head->next;
        int count = 0;
        for (int i = 0; i < _size; ++i)
        {
            if (p->data == val)
                count++;
            p = p->next;
        }
        cout << count << endl;
    }
    void exchange(const int &pos) //交换 pos 位置和 next 的位置
    {
        if (pos <= 0 || pos > _size)
            throw "pos id error";
        listnode<T> *p1 = findnode(pos - 1); //pos 前一个结点
        listnode<T> *p2 = findnode(pos);     //pos 位置的结点
        listnode<T> *p3 = findnode(pos + 1); //pos 后一个结点
        p1->next = p2->next;
        p2->next = p3->next;
        p3->next = p2;
    }
    void print()
    {
        if (!empty())
        {
            listnode<T> *p = _head->next;
            for (int i = 0; i < _size; ++i)
            {
                cout << p->data << " ";
                p = p->next;
            }
            cout << endl;
        }
        else
        {
            cout << "the linklist is empty" << endl;
        }
    }
    bool empty() { return _head->next == nullptr; }

private:
```

```
    listnode<T> *_head; //头结点
    listnode<T> *_ptr;  //指向最后一个结点
    int _size;          //链表大小
};
int main()
{
    linklist<int> list;
    list.push_back(1);
    list.push_back(2);
    list.push_back(3);
    list.push_back(2);
    list.print(); //1 2 3 2
    list.insert(1, 5);
    list.print(); //5 1 2 3 2
    list.del(1);
    list.print();   //1 2 3 2
    list.locate(1); //1
    list.number(2); //2
    list.exchange(2);
    list.print(); //1 3 2 2
    list.locate(4);
    return 0;
}
```

```
1 2 3 2
5 1 2 3 2
1 2 3 2
1
2
1 3 2 2
no val
```