Data Structure

Vine

2022 年 7 月 3 日

目录

1 绪论

2 线性表

3 栈和队列

4 串

5 数组和广义表

6 树和二叉树

7 图

8 动态存储管理

9 查找

10 内部排序

10.1 概述

10.2 插入排序

10.2.1 直接插入

```
void InsertSort(Sqlist &L){
    //对顺序表做直接插入排序
    for(i=2;i<=L.length;i++){
        if(LT(L.r[i].key,L.r[i-1].key)){ //"<",需将L.r[i]插入有序子表
            L.r[0]=L.r[i];                //复制为哨兵
            L.r[i]=L.r[i-1];
            for(j=i-2;LT(L.r[0].key,L.r[j].key);--j)
                L.r[j+1]=L.r[j];          //记录后移
            L.r[j+1]=L.r[0];              //插入正确位置
            printf("vine");
        }
    }
}//InsertSort
```

10.2.2 其他插入

折半插入

```
void BInsertSort(Sqlist &L){
    //对顺序表做折半插入排序
    for(i=2;i<=L.length;i++){
        L.r[0]=L.r[i];                    //将L.r[i]暂存到L.r[0]
        low=1,high=i-1;
        while(low<=high){                 //在L.r[low...high]中折半查找有序插入位置
            m=(low+high)/2;               //折半
            if(LT(L.r[0].key,L.r[m].key)) high=m-1; //插入点在高
            else low =m+1;                            //插入点在低
        }
        for(j=i-1;j>=high+1;--j) L.r[j+1]=L.r[j];    //记录后移
        L.r[high+1]=L.r[0];                           //插入
    }
}//BInsertSort
```

二路插入

表插入

希尔排序

```
void ShellInsort(Sqlist &L, int dk){
    //对顺序表做希尔插入排序
    //1.位置增量dk
    //2.L.r[0]是暂存不是哨兵，j<=0时插入位置已找到
    for(i=dk+1;i<=L.length;i++){
        if(LT(L.r[i].key,L.r[i-dk].key)){ //"<",需将L.r[i]插入有序子表
            L.r[0]=L.r[i];                //暂存L.r[0]
            for(j=i-dk;j>0 && LT(L.r[0].key,L.r[j].key);j-=dk)
                L.r[j+dk]=L.r[j];         //记录后移，查找插入位置
            L.r[j+dk]=L.r[0];             //插入正确位置
        }
    }
}//ShellInsort

void ShellSort(Sqlist &L, int dk[],int t){
    //按增量序列 dk[0...t-1]对顺序表做希尔插入排序
    for(k=0;k<t;k++){
        ShellInsort(L,dk[k]);      //一趟增量为dk[k]的插入排序
    }
}//ShellSort
```

10.3 快速

10.3.1 起泡排序

```
void BubbleSort(int a[],int n){
    for(i=n-1,change=TRUE;i>=1 && change;--i){
        change=FALSE;
        for(j=0;j<i,j++){
            if(a[j]>a[i]){SWAP(a[j],a[i]);change=TRUE;}
        }
    }
}
```

10.3.2 快速排序

```
void Partition(Sqlist &L,int low,int high){
    //交换顺序表L中子序列L.r[low...high]的记录，枢轴记录到位，返回位置此时
    //在枢轴前（后）记录不大于（不小于）它
    L.r[0]=L.r[low];         //第一个记录做枢轴
    pivotkey=L.r[low].key;   //枢轴记录关键字
    while(low<high){         //从表的两端交替向中间扫描
        while(low<high && L.r[high].key>=pivotkey) --high;
        L.r[low]=L.r[high];  //小的左移
        while(low<high && L.r[low]<=pivotkey) ++low;
        L.r[high]=L.r[low];  //大的右移
    }
    L.r[low]=L.r[0];          //枢轴到位
    return low;               //返回枢轴位置
}

void QSort(Sqlist &L,int low,int high){
    //对顺序表L中子序列L.r[low...high]作快速排序
    if(low<high){                          //长度大于1
        pivotkey=Partition(L,low,high); //将L.r[low...high]一分为二
        QSort(L,low,pivotkey-1);         //低子表递归
        QSort(L,pivotkey+1,high);        //高子表递归
    }
}

void QuickSort(Sqlist &L){
    //对顺序表L作快速排序
    QSort(L,1,L.length)
}
```

10.4 选择排序

10.4.1 简单选择排序

```
void SelectSort(Sqlist &L ){
    for(i=1;i<L.length;i++){         //选择第i小的记录，并交换到位
        j=SelectMinKey(L,i);         //在L.r[i...L.length]中选择key最小的记录
        if(i!=j) SWAP(L.r[i],L.r[j]) //与第i个记录交换
    }
}
```

10.4.2 树形排序

10.4.3 堆排序

```
void HeapAdjust(HeapType &H,int s ,int m ){
    //已知H.r[s...m]中记录除H.r[s]外均满足堆的定义
    //调整H.r[s]使得H.r[s...m]称为大顶堆
    rc=H.r[s];
    for(j=2*s;j<=m;j*=2){     //沿key较大的孩子节点向下筛选
        if(j<m && LT(H.r[j].key,H.r[j+1].key)) ++j; //j为key较大的记录的下标
        if(!LT(rc.key,H.r[j].key)) break;           //rc插入s
        H.r[s]=H.r[j];s=j;                           //插入
    }
    H.r[s]=rc;
}

void HeapSort(HeapType &H ){
    for(i=H.length/2;;i>0;--i)            //把H.r[1...H.length]建成大顶堆
        HeapAdjust(H,i,H.length);
    for(i=H.length;i>1;--i){               //堆顶记录和未经排序子序列H.r[1...i]中最后一个记录交换
        SWAP(H.r[1],H.r[i]);                //将[1...i-1]建成大顶堆
        HeapAdjust(H,1,i-1);
    }
}
```

10.5 归并排序

```
void Merge(RcdType SR[],RcdType & TR[],int i,int m,int n ){
    //将有序的SR[i...m],SR[m+1,n]归并为有序的TR[i...n]
    for(j=m+1,k=i;i<=m && j<=n;++k){  //将SR中记录从小到大并入TR
        if(LQ(SR[i].key,SR[j].key))  TR[k]=SR[i++];
        else TR[k]=SR[j++];
    }
    if(i<=m) TR[K...n]=SR[i...m];       //将剩余的SR[i...m]复制到TR[K...n]
    if(j<n)  TR[k...n]=SR[j...n];       //将剩余的SR[j...n]复制到TR[K...n]
}

void Msort(RcdType SR[],RcdType & TR1[],int s,int t){
    //将SR[s...t]归并为TR1[s...t]
    if(s==t) TR1[s]=SR[s];
    else{
        m=(s+t)/2;            //将SR[s...t]平分为SR[s...m], SR[m+1...t]
        Msort(SR,TR2,s,m);   //递归SR[s...m] 为有序 TR2[s...m]
        Msort(SR,TR2,m+1,t);  //递归SR[m+1...t]为有序 TR2[m+1...t]
        Merge[TR2,TR1,s,m.t]; //将TR2[s...m],TR2[m+1...t] 归并到 TR1[s...t]
    }
}

void MergeSort(Sqlist &L){
    Msort(L.r,L.r,1,L.length);
}
```

10.6 基数排序

11 外部排序

12 文件