计算机科学与技术学院

算法分析与设计

实验报告

目录

实验报告一　排序问题求解……2

实验报告二　背包问题求解……8

实验报告三　最短路径求解……12

实验报告四　N-皇后问题求解……18

姓名：　XXX　　学号：XXXXXXXX　班级：XXXXXXX

实验报告一　排序问题求解

实验目的

1）以排序（分类）问题为例，掌握分治法的基本设计策略。

2）熟练掌握一般插入排序算法的实现；

3）熟练掌握快速排序算法的实现；

4) 理解常见的算法经验分析方法；

实验内容与步骤

1. 生成实验数据.

要求：编写一个函数 datagenetare，生成 2000 个在区间[1,10000]上的随机整数，并将这些数输出到外部文件 data.txt 中。这些数作为后面算法的实验数据。

2. 实现直接插入排序算法.

要求：实现 insertionsort 算法。

算法的输入是 data.txt；

输出记录为文件：resultsIS.txt；同时记录运行时间为 TimeIS。

3. 实现快速排序算法.

要求：实现 QuickSort 算法。

算法的输入是 data.txt；

输出记录为文件：resultsQS.txt；同时记录运行时间为 TimeQS。

算法的基本思想

直接插入排序：

假设待排序的 n 个记录{R0，R1,…，Rn}顺序存放在数组中，直接插入法在插入记录 Ri(i=1，2,…，n-1)时，记录被划分为两个区间[R0,Ri-1]和[Ri+1,Rn-1],其中，前一个子区间已经排好序，后一个子区间是当前未排序的部分，将关键码 Ki 与 Ki-1Ki-2,…，K0 依次比较，找出应该插入的位置，将记录 Ri 插，然后将剩下的 i-1 个元素按关键词大小依次插入该有序序列，没插入一个元素后依然保持该序列有序，经过 i-1 趟排序后即成为有序序列。每次将一个待排序的记录，按其关键字大小插入到前面已经排好序的子文件中的适当位置，直到全部记录插入完成为止。

快速排序：

快速排序是基于分治模式的。假设待排序数组 A[p..r]。①分解：数组 A[p..r]被划分成两个（可能空）子数组 A[p..q-1]和 A[q+1..r],使得 A[p..q-1]中的每个元素都小于等于 A(q)，而且，小于等于 A[q+1..r]中的元素。小标 q 也在这个划分过程中进行计算。②解决：通过递归调用快速排序，对子数组 A[p..q-1]和 A[q+1..r]排序。③合并：因为两个子数组是就地排序的，将它们合并不需要操作：整个数组 A[p..r]已排序。

算法实现

（1）直接插入排序：

```
void insert_sort(ElemType a[],int n)
//待排序元素用一个数组 a 表示，数组有 n 个元素
{ int i,j;
ElemType t;
for ( i=1; i<n; i++) //i 表示插入次数，共进行 n-1 次插入
{ t=a[i]; //把待排序元素赋给 t
j=i-1;
while ((j>=0)&& (t<a[j]))
{ a[j+1]=a[j]; j--; } //  顺序比较和移动
a[j+1]=t;}
}
```

直接插入排序的时间复杂度为 O（n2）

（2）快速排序：

```
void QuickSort(SqList *list){
    //数组从下表为 1 开始有效，下表为 0 的单元做哨兵 初值为 0
    Qsort(list,1,list->length-1);
}
void Qsort(SqList *list,int low,int hight){

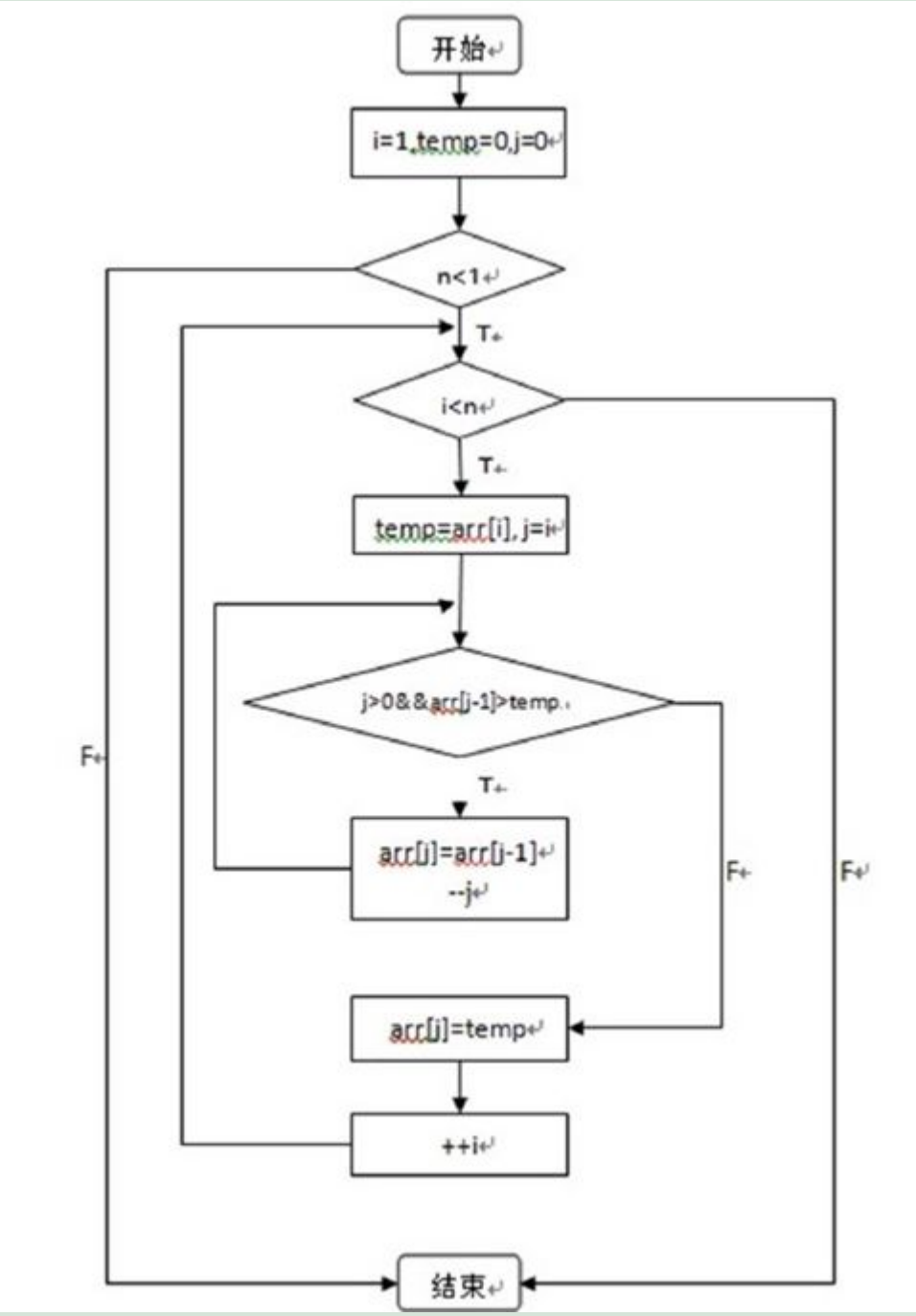
    if( low< hight){
        int mid =0;
        mid = partition(list,low,hight);
       Qsort(list,low,mid-1);                    //对左边的无序数列进行排序
       Qsort(list,mid+1,hight);                  //对右边的无序数列进行排序
```

```
    }//end if
}
int    partition(SqList *list,int low,int hight){
        ElemType mid = list->array[low];
       while(low< hight){
           while(low< hight && list->array[hight] >= mid)
         {
              hight--;
         }
         swap(list,low,hight);
          while(low< hight && list->array[low] <= mid)
         {
              low++;
         }
         swap(list,low,hight);
       }//end out while
        return low;
}
void swap( SqList *list,int low,int hight){
    ElemType temp = 0;
    temp = list->array[low];
    list->array[low] = list->array[hight];
    list->array[hight] = temp;
}
```

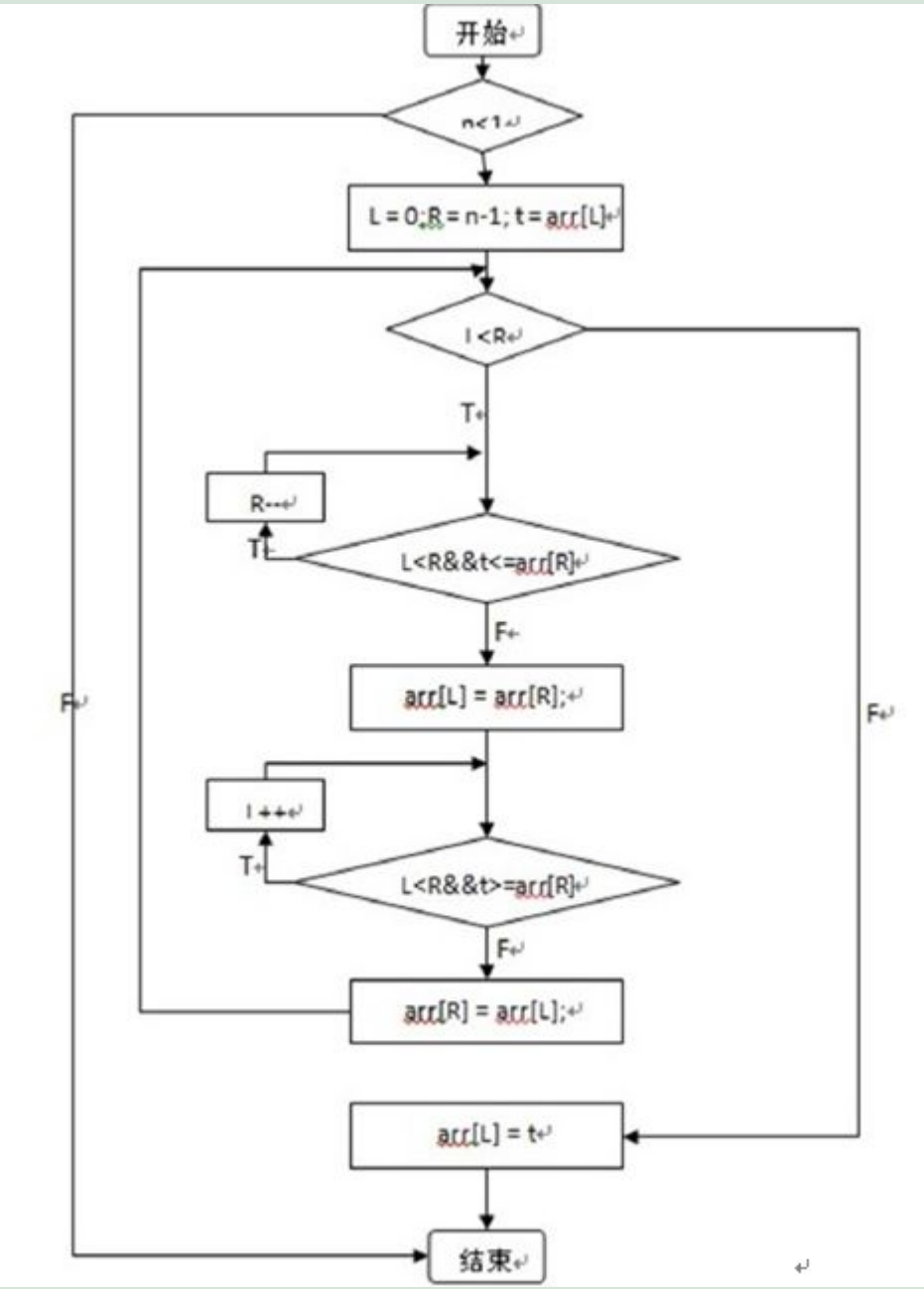
快速排序的时间复杂度是 O(nlog2n)。实验结果表明：就平均计算时间而言，快速排序是我们所讨论的所有内排序方法中最好的一个。

三、程序流程图

1. 直接插入排序程序流程图



2. 快速排序流程图


开始
n<1
L = 0;R = n-1; t = arr[L]
I <R
T
R--
T
L<R&&t<=arr[R]
F
arr[L] = arr[R];
I ++
T
L<R&&t>=arr[R]
F
arr[R] = arr[L];
F
F
arr[L] = t
结束

程序运行结果：



心得体会：

通过对此次试验，让我对不同排序的时间复杂度有了更好的了解，有时候真的需要选择合适的算法来提高效率，在目前计算机硬件水平提高放缓的瓶颈下，通过优质的算法来提高时间效率的方法势在必行，所以掌握优良的算法很重要。

实验报告二　背包问题求解

实验目的

1）以背包问题为例，掌握贪心法的基本设计策略。

2）熟练掌握各种贪心策略情况下的背包问题的算法并实现；其中：量度标准分别取：效益增量 P、物品重量 w、P/w 比值；

3) 分析实验结果来验证理解贪心法中目标函数设计的重要性。

实验内容与步骤

1. 理解需要求解背包问题.

（1）背包问题的描述：已知有 n 种物品和一个可容纳 M 重量的背包，每种物品 i 的重量为 w_i。假定将物品 i 的一部分 x_i 放入背包就会得到 $p_i x_i$ 的效益，这里，$0 \le x_i \le 1$，$p_i > 0$。显然，由于背包容量是 M，因此，要求所有选中要装入背包的物品总重量不得超过 M.。如果这 n 件物品的总重量不超过 M，则把所有物品装入背包自然获得最大效益。现需解决的问题是，在这些物品重量的和大于 M 的情况下，该如何装包，使得得到更大的效益值。由以上叙述，可将这个问题形式表述如下：

极 大 化目标函数　$\sum_{1 \le i \le n} p_i x_i$

约束条件　$\sum_{1 \le i \le n} w_i x_i \le M$

$$0 \le x_i \le 1, p_i > 0, w_i > 0, 1 \le i \le n$$

（2）用贪心策略求解背包问题

首先需确定最优的量度标准。这里考虑三种策略：

策略 1：按物品价值 p 降序装包，

策略 2：按物品重 w 升序装包

策略 3：按物品价值与重量比值 p/w 的降序装包

（3）分别以上面三种策略分别求以下情况背包问题的解：

n=7， M=15，

（$p_1,\cdots,p_7$）=（10，5，15，7，6，18，3）

（$w_1,\cdots,w_7$）=（2，3，5，7，1，4，1）。

部分程序代码

```
void GreedyKnapsack(int weight[])//贪婪装包
{     float m;
      int i;
      m=float(M);
      for(i=0;i<n;i++){
            if(weight[i]>M) break;
            X.x[i]=1;
            m=m-weight[i];
}
      if(i<n){
            X.x[i]=m/weight[i];//只装部分
      }
}
//按照价值/重量(单位重量价值)降序排序
bool compare3(int i,int j){
      if((float)p[i]/w[i]<(float)p[j]/w[j])
            return true;
      else    return false;
}
void swap(int &a,int &b){
      int t=a;
      a=b;
      b=t;
}
void sort(int way){//选择策略
      for(int i=0;i<n;i++){
            int k=i;
            for(int j=i+1;j<n;j++){
```

```
                if(way==1){
                    if(compare1(k,j))
                        k=j;
                }else if(way==2){
                    if(compare2(k,j))
                        k=j;
                }else{
                    if(compare3(k,j))
                        k=j;
                }
            }
            if(k!=i){
                swap(p[i],p[k]);
                swap(w[i],w[k]);
                swap(X.order[i],X.order[k]);
            }
        }
}
void main(){
    for(int j=1;j<=3;j++){
        printf("\t 策略%d:\n",j);
        printf("解向量 X%d:\n",j);
        init();
        sort(j);
        GreedyKnapsack(w);
        float total=0,weight=0;
        for(int i=0;i<n;i++){
            total+=p[i]*X.x[i];//计算背包的价值效益
            weight+=w[i]*X.x[i];//计算背包的重量
printf("装入的物品序号%d    装入的分量：%.6f\n",X.order[i],X.x[i]);
        }
printf("背包的价值效益 fp%d：%.6f,背包的重量 fw%d: %.2f \n",j,total,j,weight);
        printf("\n");
```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：https://d.book118.com/288062023035006023