一、火车车厢重排问题

1、问题描述

一列列车共有 n 节车厢，每节车厢将停放在不同的车站。假定 n 个车站的编号分别为 1~n，

即货运列车按照第 n 站至第 1 站的次序经过这些车站。为了便于从列车上卸掉相应的车厢，

车厢的编号应与车站的编号相同。这样，在每个车站只需卸掉最后一节车厢。所以，给定任

意次序的车厢，必须重新排列它们。

车厢的重排工作可以通过转轨站完成，在转轨站中有一个入轨、一个出轨和 k 个缓冲轨，

缓冲轨位于入轨和出轨之间。假定缓冲轨按先进先出的方式工作，设计算法解决火车车厢重

排问题。

2、基本要求

① 设计存储结构表示 n 个车厢、k 个缓冲轨以及入轨和出轨；

② 设计并实现车厢重排算法；

③ 分析算法的时间复杂度。

3、概要设计

（1）数据结构的设计

此程序采用队列这种带限制的数据结构（用数组模拟队列）模拟缓冲轨，普通的数组模拟入轨和出轨，因为题目要求排序是要从入轨经缓冲轨到入轨，根据日常经验，入轨出轨和单个缓冲轨肯定是线性的，题目有要求缓冲轨要先进先出，很自然想到队列，入轨出轨就可以采用任意的线性结构。

1. 算法设计

分两个模块

第一个、重排函数，初始化各个代表缓冲轨的队列和代表入轨、出轨的队列，从入轨依此取出各个元素（车厢次序），经缓冲轨按照从小到大的次序进入出轨：

#define List 20

#define Last 10

typedef int Elemtype;

void chongpai()

{

Elemtype hc[List][List];int front[List]={0},rear[List]={0};//数组模拟队列，头指针指向头元素，尾指针指向尾元素的下一个位置

printf("输入火车车厢数:");

scanf(&n);//火车车厢数

printf("输出车厢的初始顺序(从1开始，如：3 4 5 6 7)：\n");

for(i=0;i<n;i++)

{

scanf(&a[i]);//代表入轨

b[i]=0;//代表出轨

}

for(i=0;i<List;i++)

for(j=0;j<List;j++)

hc[i][j]=0;

for(i=0;i<n;i++)

{

for(k=0;k<n;k++)

if((rear[k]!=front[k]&&hc[k][rear[k]-1]<a[i])||rear[k]==front[k]){hc[k][rear[k]]=a[i];rear[k]++;break;}

for(j=1;b[j-1]!=0;j++);

for(;j<=n;j++)

{

for(k=0;k<n&&hc[k][front[k]]!=j;k++);

if(k==n)break;

for(k=0;k<n;k++)

{

if(rear[k]!=front[k]&&hc[k][front[k]]==j){b[j-1]=hc[k][front[k]];front[k]++;break;}

}

}

}

printf("输出重排后的火车车厢次序：");

for(i=0;i<n;i++)

printf(b[i]);

}

第二个、主函数，调用重排函数：

int main()

{

chongpai();

return 0;

1. 抽象数据类型的设计

①Elemtype是自定义的类型，

②为了使代码的长度短，我用简单的数组模拟队列，用两个int型的数据front和rear作为队头指针（指向队头元素）和队尾指针（指向队尾元素的下一个位置）

Elemtype hc[List][List];int front[List]={0},rear[List]={0};//数组模拟队列，头指针指向头元素，尾指针指向尾元素的下一个位置

hc中第一维代表第几个缓冲轨，第二维代表单个队列的数据元素，Elemtype是自定义的类型，front[0]、rear[0]代表第1个队列的队头指针，队尾指针,依次类推。为确保空间够用，我让每个下标的最大值为火车车厢数减一。

③判断第k个队列是否为空rear[k]!=front[k]

入队hc[k][rear[k]]=a[i];rear[k]++;

出队b[j-1]=hc[k][front[k]];front[k]++;

1. 详细设计

①主函数

比较简单，就调用了重排函数一下，就return 0；了

②子函数

重排函数 由于想到先从缓冲轨出来进入出轨的要比之后小，因此元素进入第k个缓冲轨的条件（从第一个开始）是若不为空且队尾元素比要进的元素小，就进入，否则为空的话就进入。若条件不满足，就找下一个缓冲轨，直至找到为止，将元素放入。

首先要将所有队列以及队头指针数组，队尾指针数组初始化为零。

车厢序列从1开始，依此加1，因此不必用排序的方法，只要看每个缓冲轨的队头元素是不是出轨的当前第一个为空的空间的下标加一，就可完成判断，是就入队，然后依此类推；如果遇到哪个元素没有任何缓冲轨满足这一条件，就跳出循环，不能继续找下去。因为出轨是线性的，必须放第一个才能放下一个，不能放完第二个就放第四个。因此重新开始从入轨到缓冲轨放元素。

最后用个循环输出就可得到已排好序（从小到大）的火车车厢顺序。

1. 运行与测试

①第4条第②个的第三行一开始没想那么细，所以出了问题，因此就把（每个缓冲轨的队头元素是不是出轨的当前第一个为空的空间的下标）和（依此类推）组合起来，最后输出成功。

一开始就想要尽量节省缓冲轨的数量，不能用跟火车车厢数量一样的缓冲轨，虽然也试过最坏情况，确实不能节省数量，但想不出怎么节省的办法，也是为找好的测试数据，就上网看了一个，它使用三个缓冲轨，没看它的算法，但思路和我的一样，就用三个缓冲轨最后许多数据测试，有些情况下会失败，确实不能节省缓冲轨数量，就只好弄成现在这个样子。

②用九个数，从最好情况（1 2 3 4 5 6 7 8 9）到（9 8 7 6 5 4 3 2 1），再到随便给数，最后都成功了。

1. 总结与心得

通过此次作业，我觉得我应该在编程之前画出流程图，考虑到各个情况。可能是受某些节目的影响，现在只是列出大纲，觉得大致可行，直接就写，在编程过程中添细节，这样可能会遇到无法预知且难以解决的情况。因此以后一定要先将细节弄明白再动手，这样便可减少出错。从上到下，逐步细化。

时间复杂度为：

for(i=0;i<n;i++)

{

for(k=0;k<n;k++)

if((rear[k]!=front[k]&&hc[k][rear[k]-1]<a[i])||rear[k]==front[k]){hc[k][rear[k]]=a[i];rear[k]++;break;}

for(j=1;b[j-1]!=0;j++);

for(;j<=n;j++)

{

for(k=0;k<n&&hc[k][front[k]]!=j;k++);

if(k==n)break;

for(k=0;k<n;k++)

{

if(rear[k]!=front[k]&&hc[k][front[k]]==j){b[j-1]=hc[k][front[k]];front[k]++;break;}

}

}

}

O(n2)

空间复杂度:要用二维数组形式的多个队列O（n2）