**实习报告**

题目：车厢调度

班级：电信1809

姓名：叶杰栋

学号：U201813497

完成日期：2019/4/10

1. 需求分析

假设停在铁路调度站入口处的车厢序列编号依次为1,2,3…,n。设计一个程序，求出所有可能由此输出的长度为n的车厢序列。

1. 输入应为车厢的最大长度n。
2. 输出为可能的所有车厢序列。
3. 用多个栈储存车厢序列编号，通过控制在不同栈之间数据的入栈和出栈，得出可能的所有车厢序列。
4. 测试数据为n=1,2,3,4,5
5. 概要设计

（1）本程序中主要的抽象数据类型为栈。

ADT stack{

数据对象：D={ai|i=1,2,3,…,n，n>0}

数据关系：R1={<ai-1,ai>|ai-1,ai∈D，i=2,3…,n}

基本操作：

Innit（&S）

操作结果：构造一个空栈S。

Isempty（S）

初始条件：栈S已存在。

操作结果：判断栈是否为空，空则返回0，非空则返回1。

Isfull（S）

初始条件：栈S已存在。

操作结果：判断栈是否已满，满则返回1，不满则返回0。

Push（&S，e）

初始条件：栈S已存在。

操作结果：将数据e推入栈顶。

Pop（&S，&e）

初始条件：栈S已存在且非空。

操作结果：将栈S的栈顶元素出栈，并用e返回其值。

}ADT stack

1. 主程序的流程：本程序主要有三个模块①主函数模块②栈模块③车厢调度模块，调用关系为①→②→③。
2. 详细设计
3. 栈相关操作的函数：

typedef struct \_stack

{

int\* top;

int\* base;

int size;

}stack;

stack init(stack sta){

sta.base=(int\*)malloc(Maxsize\*(sizeof(int)));

sta.top=sta.base;

sta.size=Maxsize;

printf("初始化成功！\n");

return sta;

}//初始化栈

stack push(stack sta,int a){

if(sta.top-sta.base>=sta.size)

{

printf("栈满，正在扩充栈空间\n");

sta.base=(int\*)realloc(sta.base,(sta.size+Extra\_space)\*sizeof(int));

sta.top=sta.base+sta.size;

sta.size+=Extra\_space;

printf("扩充成功\n");

}

\*sta.top++=a;

return sta;

}//入栈操作

stack pop(stack sta,int\* pa){

if(sta.base==sta.top)

{

printf("栈空，无法再弹出元素\n");

return sta;

}

\*pa=\*(sta.top-1);

sta.top--;

return sta;

}//出栈操作

int isempty(stack sta){

if(sta.base==sta.top)

{

return 0;

}

else

return 1;

}//判断栈是否为空，空返回0，非空返回1

int isfull(stack sta,int n)

{

if(sta.top-sta.base==N)

return 1;

else

return 0;

}//判断栈是否满，即栈内元素个数是否等于n，满则返回1，不满则返回0。

1. 核心函数，即dispatch函数：

逻辑描述：在已建立的三个栈中（初始车厢数据的栈①、过渡栈②、输出栈③），按顺序从初始栈中将数据逐个从过渡站推到输出栈（注：数据只能按初始栈↔过渡栈↔输出栈顺序传递，不能直接从初始栈到输出栈），输出栈满后输出序列，得到一种排序方式。然后进行回溯，将数据从输出栈逐个推入过渡栈，再从过渡栈推出部分元素到初始栈，回到之前的某一状态，再从该状态进行数据从过渡栈到输出栈，再从初始栈到输出栈的操作，输出栈满后再输出新的序列。然后再次回溯到某一状态，重复上述操作，直到输出所有可能的车厢排序为止。

该函数的代码如下：

void dispatch(stack s1,stack s2,stack s3,int \*pm)

{

if(isempty(s1))

{

s1=pop(s1,pm);

s2=push(s2,\*pm);

dispatch(s1,s2,s3,pm);

s2=pop(s2,pm);

s1=push(s1,\*pm);

}

if(isempty(s2))

{

s2=pop(s2,pm);

s3=push(s3,\*pm);

dispatch(s1,s2,s3,pm);

s3=pop(s3,pm);

s2=push(s2,\*pm);

}

if(isfull(s3,N))

{

printstack(s3);

}

}

函数调用关系：

Push

Init pop

主函数 dispatch dispatch

Push isempty

Isfull

1. 调试分析
2. 本程序最大的难点是如何在输出一种序列之后，回溯到之前的某一状态再次进行排序操作。在查阅网络资料和对网上的部分代码进行分析理解后，最终我采用的了递归算法进行数据的回溯，以及以三个栈的形式来储存数据。
3. 本程序的时间复杂度O(n3)。
4. 经过本次程序设计，我对递归也有了更深的理解，了解到递归也有回溯的作用。像之前所接触到的递归算法大多都是用来逐步缩小问题的规模，本次程序中的递归和它们很不一样。
5. 用户使用说明

直接输入最大车厢数n即可，本程序会直接输出所有可能的车厢排序。

1. 测试结果

输入：1

输出：1

输入：3

输出：3 2 1

2 3 1

2 1 3

1 3 2

1 2 3

输入：4

输出：4 3 2 1

3 4 2 1

3 2 4 1

3 2 1 4

2 4 3 1

2 3 4 1

2 3 1 4

2 1 4 3

2 1 3 4

1 4 3 2

1 3 4 2

1 3 2 4

1 2 4 3

1 2 3 4

1. 附录

完整代码：

#include <stdio.h>

#include <stdlib.h>

#define Maxsize 100

#define Extra\_space 10

#define N 3//重新定义N值时需要修改

typedef struct \_stack

{

int\* top;

int\* base;

int size;

}stack;

stack init(stack sta)

{

sta.base=(int\*)malloc(Maxsize\*(sizeof(int)));

sta.top=sta.base;

sta.size=Maxsize;

printf("初始化成功！\n");

return sta;

}

stack push(stack sta,int a)

{

if(sta.top-sta.base>=sta.size)

{

printf("栈满，正在扩充栈空间\n");

sta.base=(int\*)realloc(sta.base,(sta.size+Extra\_space)\*sizeof(int));

sta.top=sta.base+sta.size;

sta.size+=Extra\_space;

printf("扩充成功\n");

}

\*sta.top++=a;

return sta;

}

stack pop(stack sta,int\* pa)

{

if(sta.base==sta.top)

{

printf("栈空，无法再弹出元素\n");

return sta;

}

\*pa=\*(sta.top-1);

sta.top--;

return sta;

}

int isempty(stack sta)

{

if(sta.base==sta.top)

{

return 0;

}

else

return 1;

}

int isfull(stack sta,int n)

{

if(sta.top-sta.base==N)

return 1;

else

return 0;

}

void printstack(stack sta)

{

int\* pa=sta.base;

while(pa!=sta.top)

{

printf("%d ",\*(pa++));

}

printf("\n");

}

void dispatch(stack s1,stack s2,stack s3,int \*pm)

{

if(isempty(s1))

{

s1=pop(s1,pm);

s2=push(s2,\*pm);

dispatch(s1,s2,s3,pm);

s2=pop(s2,pm);

s1=push(s1,\*pm);

}

if(isempty(s2))

{

s2=pop(s2,pm);

s3=push(s3,\*pm);

dispatch(s1,s2,s3,pm);

s3=pop(s3,pm);

s2=push(s2,\*pm);

}

if(isfull(s3,N))

{

printstack(s3);

}

}

void main()

{

stack s1,s2,s3;

s1=init(s1);

s2=init(s2);

s3=init(s3);

int first=1;

int last=4;

int m=0;

int i;

int a[N]={1,2,3};//重新定义N值时需要修改

for(i=2;i>=0;i--)//重新定义N值时需要修改i

{

s1=push(s1,a[i]);

}

dispatch(s1,s2,s3,&m);

}