实验4、《回溯法实验》

**一、实验目的**

1. 掌握回溯算法思想

2. 掌握回溯递归原理

3. 了解回溯法典型问题

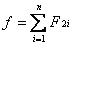
**二、实验内容**

1. 编写一个简单的程序，解决8皇后问题。

2. 批处理作业调度问题

[问题描述]给定n个作业的集合J=(J1, J2, … , Jn)。每一个作业Ji都有两项任务需要分别在2台机器上完成。每一个作业必须先由机器1处理，然后再由机器2处理。作业Ji需要机器i的处理时间为tji，i=1,2, … ,n; j=1,2。

对于一个确定的作业调度，设Fji是作业i在机器i上完成处理的时间。则所有作业在机器2上完成处理的时间和成为该作业调度的完成时间和。



批处理作业调度问题要求对于给定的n个作业，制定一个最佳的作业调度方案，使其完成时间和达到最小。

要求输入：

1）作业数 2）每个作业完成时间表：

|  |  |  |
| --- | --- | --- |
| 作业完成时间 | 机器1 | 机器2 |
| 作业1 | 2 | 1 |
| 作业2 | 3 | 1 |
| 作业3 | 2 | 3 |

要求输出： 1）最佳完成时间 2）最佳调度方案

提示：算法复杂度为O(n!),建议在测试的时候n值不要太大，可以考虑不要超过12。

3. 数字全排列问题

任意给出从1到N的N个连续的自然数，求出这N个自然数的各种全排列。如N=3时，共有以下6种排列方式：123，132，213，231，312，321。

注意：数字不能重复，N由键盘输入（N<=9）。

**三、算法思想分析**

1.八皇后问题是典型的回溯问题，先从空格子起逐行放皇后，如果符合要求即安全则放置，否则返回上一行下一个位置继续，直至最后一行安全放置则为一种放置方式。

2.批处理作业调度的解空间为排列数，不断利用递归函数直至叶节点，剪枝函数为当前用时与最佳用时的比较。关于时间的计算，每次选择作业后先将机器1用时累加，机器2上总用时需要先比较上一个作业完成时间与此时机器1上的总用时，如果机器1上总用时大于上一作业用时，那么机器2上用时则加上机器1上用时与此作业在机器2上的单独用时，反之，则代表此时机器2仍然在处理上一任务，那么机器2上用时则加上上一作业用时与此作业在机器2上的单独用时。

3.数字全排列问题的解空间为排列树，依次向下排列，判断是否继续的条件为该数字是否已经使用，通过一个flag与for循环判断即可，直至到了最后一个数字则输出结果。

**四、实验过程分析**

1.八皇后问题的难点在于安全条件的设计，即同一列，对角线都不能同时放置，对角线的设计比较巧妙，根据PPT设计向右斜为i-j+N,向左斜为i+j，利用二维数组很好的避免了代码的重复与累赘。

2.批处理作业调度的回溯条件即当前时间与最佳时间的比较，本题的难点在于时间的计算，最开始对于机器2上的总用时是用一个整型数据记录，但没办法与上一个作业用时比较，或者说不方便进行下一次回溯条件的改变，后来参考课本PPT采用数组记录就解决了。计算时间的另一个问题即作业进入机器2上时，上一个作业是否已经完成，需不需要等待，这里要加判断条件，否则时间计算会出问题。

3.从数字全排列问题的实验中，我更加熟悉了回溯的基本套路，判断数字是否使用的flag和for循环设计很典型也很实用。

**五、算法源代码及用户屏幕**

1.（1）算法源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

八皇后问题。

codeblocks C++

2018.11.3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <iomanip>

using namespace std;

#define N 8 //定义皇后数量

int countN = 0;

int tryN(int n, int a[N][N], int m[], int right[], int left[]); //检测皇后摆放方式

void printN(int a[N][N]); //打印皇后摆放位置

int main()

{

int a[N][N] = {0}; //摆放皇后的数组

int m[N] = {0}; //同一竖排

int right[N\*2] = {0}; //右斜对角线

int left[N\*2] = {0}; //左斜对角线

int countN = tryN(0, a, m, right, left);

cout<<"count = "<<countN<<endl;

return 0;

}

int tryN(int n, int a[N][N], int m[], int right[], int left[]){

for(int j=0; j<N; j++){

if(!m[j] && !right[n-j+N] && !left[n+j]){ //位置安全

//放皇后

a[n][j] = n+1;

m[j] = right[n-j+N] = left[n+j] = 1;

if(n == N-1){ //已经到最后一行

printN(a);

countN++;

}

else{ //检测下一行

tryN(n+1, a, m, right, left);

}

//回溯，去掉该皇后

a[n][j] = 0;

m[j] = right[n-j+N] = left[n+j] = 0;

}

}

return countN;

}

void printN(int a[N][N]){

for(int i=0; i<N; i++){

for(int j=0; j<N; j++){

cout<<setw(2)<<a[i][j];

}

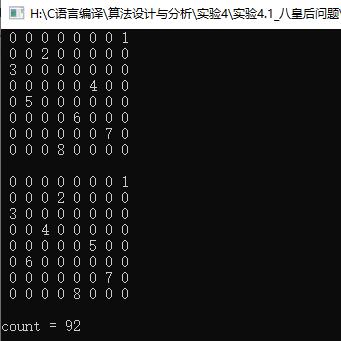
cout<<endl;

}

cout<<endl;

}

（2）用户屏幕



2.（1）算法源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

批处理作业调度问题

codeblocks C++

2018.11.4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <iomanip>

using namespace std;

const int num = 100;

int m1Time[num]; //作业在机器1上花费的时间

int m2Time[num]; //作业在机器2上花费的时间

int workBest[num]; //记录最有作业调度方法

int work[num]; //记录当前作业调度方法

int timeCurrent = 0;

int timeBest = 9999;

int timeM1 = 0; //记录机器1上用时

int timeM2[num] = {0}; //记录每个作业总用时，从下标1开始

void backTrack(int deep, int workNumber);

int main()

{

//用时输入作业数量与在机器1、2上的用时

int workNumber;

cout<<"Enter work numbers : ";

cin>>workNumber;

cout<<"work m1 m2"<<endl;

for(int i=0; i<workNumber; i++){

cout<<" "<<i+1<<" ";

cin>>m1Time[i];

cin>>m2Time[i];

}

for(int i=0; i<workNumber; i++){

workBest[i] = i; //顺序调度,编号-1

work[i] = i;

}

backTrack(0,workNumber);

cout<<"BestTime : "<<timeBest<<endl;

cout<<"Scheduling sort : "<<endl;

for(int i=0; i<workNumber; i++)

cout<<workBest[i]+1<<" ";

cout<<endl;

return 0;

}

void backTrack(int deep, int workNumber){

if(deep >= workNumber){ //到底叶节点

if(timeCurrent < timeBest){

//该调度方法为最佳方法，更新最佳方法的数据

for(int i=0; i<workNumber; i++){

workBest[i] = work[i];

}

timeBest = timeCurrent;

}

}

else{

for(int i=deep; i<workNumber; i++){

//从剩余的作业中选择一个与当前作业交换位置

int temp = work[deep];

work[deep] = work[i];

work[i] = temp;

timeM1 += m1Time[work[deep]]; //在机器1上总用时

//作业deep+1在机器2上的完成时间和

//在机器1完成时间如果大于上一个作业时间，则直接进入机器2继续，

//反之，如果小于上一个作业完成时间，则需要继续等待，直到上一个作业完成

timeM2[deep+1] = (timeM2[deep] > timeM1 ? timeM2[deep]:timeM1 ) + m2Time[work[deep]];

//该调度当前总用时

timeCurrent += timeM2[deep+1];

if(timeCurrent < timeBest){

backTrack(deep+1,workNumber);

}

timeM1 -= m1Time[work[deep]];

timeCurrent -= timeM2[deep+1];

temp = work[deep];

work[deep] = work[i];

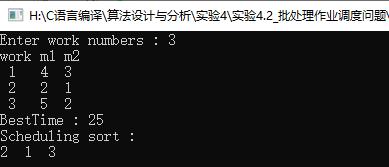
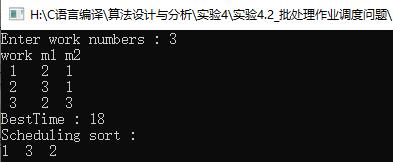
work[i] = temp;

}

}

}

（2）用户屏幕



3.（1）算法源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

N个自然数全排列问题。

codeblocks C++

2018.11.3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

using namespace std;

void backtrack(int deep, int number, int a[]);

int main(){

int number;

int a[9];

cout<<"Enter a number : ";

cin>>number;

backtrack(0,number,a);

return 0;

}

void backtrack(int deep, int number, int a[]){

if(deep >= number){ //到底叶节点，输出结果

for(int i=0; i<number; i++)

cout<<a[i]+1<<" ";

cout<<endl;

}

for(int i=0; i<number; i++){ //继续选择数字排列

bool flag = true;

for(int j=0; j<deep; j++){ //判断该数字是否已使用

if(a[j] == i)

flag = false;

}

if(flag){

a[deep] = i;

backtrack(deep+1,number,a);

}

}

}

（2）用户屏幕

