**算法分析与设计实验报告**

**第 8 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 周思宇 | 学号 | 201608030201 | | 班级 | 计科1601 |
| 时间 | 6.8 | 地点 | 028 | | | |
| 实验名称 | **拉斯维加斯算法结合回溯法求解n后问题** | | | | | |
| 实验目的 | 通过上机实验，要求掌握n后问题的问题描述、拉斯维加斯算法结合回溯法求解n后问题的算法设计思想、程序设计、效果和时间复杂度分析。 | | | | | |
| 实验原理 | 实验过程中，先随机放置 stopVegas 个皇后，然后采用拉斯维加斯算法对皇后位置进行求解; 余下的8-stopVegas个皇后采用回溯的办法进行放置。由此可见，8皇后放置得正确与否 完全取决于拉斯维加斯算法能否确定前 stopVegas个皇后的位置。由于拉斯维加斯算法的一个显著特征是它所作的随机决策有可能导致算法找不到所需的解,因此，实验用一个bool型函数表示拉斯维加斯型算法。当算法找到一个解时返回true，否则返回false。  void Obstinate(InputType x, OutputType &y){  bool success= false;  while (!success)  success = LV(x,y);  }  特别需要说明的是，拉斯维加斯型概率算法的选择的随机性也许是无法得到问题的解的一个重要原因，也就是说算法运行一次，可能获得一个完整无误的解，也可能得不到正确的解.只要出现失败的概率不占多数，当算法运行失败时，在相同的输入实例上再次运行拉斯维加斯型概率算法，就可能获得正确的解.  设p(x)是对输入x调用拉斯维加斯算法获得问题的一个解的概率。一个正确的拉斯维加斯算法应该对所有输入x均有p(x)>0。设t(x)是算法obstinate找到具体实例x的一个解所需的平均时间 ,s(x)和e(x)分别是算法对于具体实例x求解成功或求解失败所需的平均时间，则有。解此方程得： | | | | | |
| 实验步骤 | ① 参考课件、教材、其它资料，将伪代码改成正式程序代码。  ② 用一组小数据，手工验证程序正确性，发现可能的错误并修复。  ③ 自己设计代码，生成小、中、大规模数据，分别存到三个.txt文件。  ④ 对三种规模的数据，检测程序运行时间，观察并记录结果和发现。 | | | | | |
| 关键代码 | 1. **randomnumber cpp文件**   //  // RandomNumber.cpp  // 拉斯维加斯n后  //  // Created by zsy on 2018/6/7.  // Copyright © 2018年 zsy. All rights reserved.  //  #include "RandomNumber.hpp"  // RandomNumber.cpp  #include <iostream>  #include <stdlib.h>  #include <time.h>  using namespace std;  // 产生种子  RandomNumber::RandomNumber(unsigned long s)  {    if(s == 0)    randSeed = time(0); //用系统时间产生种子    else    randSeed = s;    }  // 产生0 ~ n-1 之间的随机整数  unsigned short RandomNumber::Random(unsigned long n)  {    randSeed = multiplier \* randSeed + adder;    return (unsigned short)((randSeed >> 16) % n);    }  // 产生[0, 1)之间的随机实数  double RandomNumber::fRandom()  {    return Random(maxshort) / double(maxshort);    }   1. **randomnumber hpp文件**   //  // RandomNumber.hpp  // 拉斯维加斯n后  //  // Created by zsy on 2018/6/7.  // Copyright © 2018年 zsy. All rights reserved.  //  // RandomNumber.h  const unsigned long maxshort = 65535L;  const unsigned long multiplier = 1194211693L;  const unsigned long adder = 12345L;  #ifndef RANDOMNUMBER\_hpp  #define RANDOMNUMBER\_hpp  #include <stdio.h>  class RandomNumber{    private:    // 当前种子    unsigned long randSeed;    public:    // 构造函数,默认值0表示由系统自动产生种子    RandomNumber(unsigned long s = 0);    // 产生0 ~ n-1之间的随机整数    unsigned short Random(unsigned long n);    // 产生[0, 1) 之间的随机实数    double fRandom();    };  #endif   1. **queen cpp文件**   //  // Queen.cpp  // 拉斯维加斯n后  //  // Created by zsy on 2018/6/7.  // Copyright © 2018年 zsy. All rights reserved.  //  #include <iostream>  #include "Queen.hpp"  #include "RandomNumber.hpp"  using namespace std;  bool Queen::Place(int k)  {  // 测试皇后k置于第x[k]列的合法性  for(int j = 1; j < k; ++ j)  if((abs(k-j) == abs(x[j]-x[k])) || (x[j]==x[k]))//p剪枝函数  return false;  return true;  }  bool Queen::Backtrack(int t)  {  // 解n后问题的回溯法  if(t > n)  {  for(int i=1; i<=n; ++i)  y[i] = x[i];  return true;  }  else  for(int i=1; i<=n; ++i)  {  x[t] = i;  if(Place(t) && Backtrack(t+1))  return true;  }  return false;  }  // k是第k行，x[k]表示第k行的皇后放在x[k]列  bool Queen::QueensLV(int stopVegas)  {  // 随机放置n个皇后的拉斯维加斯算法  RandomNumber rnd; // 随机数产生器  int k = 1; // 下一个放置的皇后编号  int count = 1;  // 1 <= stopVegas <= n 表示允许随机放置的皇后数  while((k <= stopVegas) && (count > 0))//放置一个皇后（第k个）  {  count = 0;  for(int i = 1; i<=n; ++i) //这里是遍历第k行所有可以放置的列号，用y保存下来，并用count记录有多少个位置可以放置  {//一次for循环把第k个皇后能放的位置都保存下来。  x[k] = i;  if(Place(k))//随机选一个  y[count++] = i;  }  if(count > 0) //这里利用上面保存的可以放置的列，然后随机取其中一列来放置第k行的皇后。这就是Las Vegas的精髓！  x[k++] = y[rnd.Random(count)]; // 随机位置count内挑  }  return (count > 0); // count > 0表示放置位置成功  }   1. **queen hpp文件**   //  // Queen.hpp  // 拉斯维加斯n后  //  // Created by zsy on 2018/6/7.  // Copyright © 2018年 zsy. All rights reserved.  //  #ifndef Queen\_hpp  #define Queen\_hpp  #include <stdio.h>  class Queen  {  friend bool nQueen(int);  private:  bool Place(int k); // 测试皇后k置于第x[k]列的合法性  bool Backtrack(int t); // 解n后问题的回溯法  bool QueensLV(int stopVegas); // 随机放置n个皇后拉斯维加斯算法  int n, \*x, \*y;  };  #endif   1. **main cpp文件**   //  // main.cpp  // 拉斯维加斯n后  //  // Created by zsy on 2018/6/7.  // Copyright © 2018年 zsy. All rights reserved.  //  #include "Queen.hpp"  #include "RandomNumber.hpp"  #include <iostream>  using namespace std;  // 与回溯法结合的解n后问题的拉斯维加斯算法😤  bool nQueen(int n)  {  Queen X;  // 初始化X  X.n = n;  int \*p = new int[n+1];  int \*q = new int[n+1];  for(int i=0; i<=n; ++i)  {  p[i] = 0;  q[i] = 0;  }  X.y = q;  X.x = p;  int stop = 4;//设置随机放置皇后的个数🙄  if(n > 15)  stop = n-15;  bool found = false;  while(! X.QueensLV(stop));  //算法的回溯搜索部分🤭  if(X.Backtrack(stop+1))  {  for(int i=1; i<=n; ++i)  cout << p[i] << " ";  found = true;  }  cout << endl;  delete [] p;  delete [] q;  return found;  }  int main()  {  int u;  cout<<"请问棋盘大小为多少？"<<endl;  cin>>u;  nQueen(u);  } | | | | | |
|  | **手动测试代码：**  **测试样例：8**  **1．随机放置0颗棋子**    成功：    **2．随机放置3颗棋子**  第一次：  失败  第二次：  失败  第三次：    成功  **3．随机放置8颗棋子**    成功    **测试样例：12**  **1．随机放置0颗棋子**  **成功**      **2．随机放置5颗棋子**  第一次  成功    第二次  失败    第三次  失败    **3．随机放置12颗棋子**  成功    用电脑测试总概率：    时间复杂度分析：  以八皇后问题举例，经过测试实验能够明显的看出，利用拉斯维加斯型概率算法放置两个皇后，然后再采用回朔法比完全采用回朔法快大约3倍，而随机的放置三个皇后再采用回朔法比完全采用回朔法快大约1.5倍，扩展到全部皇后都随机放置比完全采用回朔法慢大约5倍.这种情况解释起来比较容易: 即产生随机数也需要时间，当皇后数达到一定程度时，八皇后问题的求解在产生随机数这个问题上大概花费了70%的时间.  关于时间复杂度，一开始认为是  但是实际测试的结果并没有这么快,没注意拉丝维加斯可能要调用多次  通过多次跑出来的数据用Excel拟合画图，推测时间复杂度是 | | | | | |
| 实验心得 | stop=0表示完全使用回溯法解决问题。因此**一定**可以得到一组解。  stop=8表示完全实用随机法解决问题。因此**一定**可以得到一组解。  首先，这个程序为何会造成有失败的情况，那就是因为在随机求出前stop行成立时，不代表后面N-stopVegas行用回溯就可以成立，所以不一定能求出可行解。  而我们在完全随机时，那么它就是反复调用随机位置放置n个皇后的Las Vegas算法，直至放置成功。所以当stopVegas=8时，他的成功率也应该是100%。    实验对于stop的各种取值在开始分别运行 20 次，然后统计成功情况，设 P 为算法成功的频率stopVegas 越大，成功的概率便越小，实验是以频率代替概率。因此，对出现与猜想不符的频率值，反复地增加了实验次数，然而最终得出的数据结果如上图所示，仍然与猜想的数据不吻合。 （左实验，右猜想）  后来弄清楚，书上的概率是按照一次回溯便成功来算的。  由于在实际中，概率是无法直接求出的，所以以频率代替概率。由于频率在实验次数充分多的时候才接近于概率，因此数据精确度有待提高，但数据反映的概率变化趋势却是不能否认的。  从得到的实验数据可以看出，当随机置的皇后数为7个(实验次数为150次)的时候，所有皇后被成功放置的频率突然增大了，当随机放置的皇后数为8个(实验次数为 150 次)的时候，所有皇后被成功放置的频率恒为1。  实际上，当 stopVegas为8的时候，本文算法退化为一个纯概率算法，而本文引用的算法是通过循环语句调用拉斯维加斯算法，仅仅当拉斯维加斯算法搜索成功时方才退出循环，因此最终概率算法搜索的成功概率肯定为1，在这种情况下，搜索皇后位置成功的概率也就为1了。  回溯法和拉斯维加斯算法的结合与和回溯法的对比：  //  // main.cpp  // n后问题回溯法优化  //  // Created by zsy on 2018/5/30.  // Copyright © 2018年 zsy. All rights reserved.  //  #include <iostream>  #include <algorithm>  #include<cstdio>  using namespace std;  int n;  int x[16]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};  void Backtrack(int t,int x[],int n,int left,int right)  {  int pos=0,feasible=0;  if(t==n)  {  for(int i=0;i<n;i++)  printf("%d%c",x[i],i==n-1?'\n':' ');  }  else  for(int i=t;i<n;i++)  {  pos=1<<(x[i]-1);  feasible=!((left&pos)|(right&pos));  if(feasible)  {  swap(x[i],x[t]);  Backtrack(t+1,x,n,(left|pos)>>1,(right|pos)<<1);  swap(x[i],x[t]);  }  }  }  int main() {  scanf("%d",&n);  Backtrack(0,x,n,0,0);  return 0;  }  /\*  #include<iostream>  #include<cstdio>  using namespace std;  int n,a[16];  void dfs(int x,int y,int z,int a[],int l){  if(l==n){  for(int i=0;i<n;i++)  printf("%d%c",a[i]+1,i==n-1?'\n':' ');  return ;  }  for(int i=0;i<n;i++){  if(((1<<i)&x)==0&&((1<<(i+l))&y)==0&&((1<<(i-l+15))&z)==0){  a[l]=i;  dfs((1<<i)|x,(1<<(i+l))|y,(1<<(i-l+15))|z,a,l+1);  }  }  }  int main(){  scanf("%d",&n);  dfs(0,0,0,a,0);  return 0;  }\*/  回溯法解n后问题时, 实际上是在系统地搜索整个解空间树的过程中找出满足要求的解, 这忽略了一个重要的事实:对于n后问题的任何一个解而言, 每一个皇后在棋盘上的位置无任何规律,不具有系统性,  而更像是随机放置的,由此容易想到随机性极强的概率算法.  在stackoverflow看到的构造法，（答者在维基百科找到的）    会在超小的时间复杂度内获得一个正确解。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**

//

// RandomNumber.cpp

// 拉斯维加斯n后

//

// Created by zsy on 2018/6/7.

// Copyright © 2018年 zsy. All rights reserved.

//

#include "RandomNumber.hpp"

// RandomNumber.cpp

#include <iostream>

#include <stdlib.h>

#include <time.h>

using namespace std;

// 产生种子

RandomNumber::RandomNumber(unsigned long s)

{

if(s == 0)

randSeed = time(0); //用系统时间产生种子

else

randSeed = s;

}

// 产生0 ~ n-1 之间的随机整数

unsigned short RandomNumber::Random(unsigned long n)

{

randSeed = multiplier \* randSeed + adder;

return (unsigned short)((randSeed >> 16) % n);

}

// 产生[0, 1)之间的随机实数

double RandomNumber::fRandom()

{

return Random(maxshort) / double(maxshort);

}

//

// RandomNumber.hpp

// 拉斯维加斯n后

//

// Created by zsy on 2018/6/7.

// Copyright © 2018年 zsy. All rights reserved.

//

// RandomNumber.h

const unsigned long maxshort = 65535L;

const unsigned long multiplier = 1194211693L;

const unsigned long adder = 12345L;

#ifndef RANDOMNUMBER\_hpp

#define RANDOMNUMBER\_hpp

#include <stdio.h>

class RandomNumber{

private:

// 当前种子

unsigned long randSeed;

public:

// 构造函数,默认值0表示由系统自动产生种子

RandomNumber(unsigned long s = 0);

// 产生0 ~ n-1之间的随机整数

unsigned short Random(unsigned long n);

// 产生[0, 1) 之间的随机实数

double fRandom();

};

#endif

//

// Queen.cpp

// 拉斯维加斯n后

//

// Created by zsy on 2018/6/7.

// Copyright © 2018年 zsy. All rights reserved.

//

#include <iostream>

#include "Queen.hpp"

#include "RandomNumber.hpp"

using namespace std;

bool Queen::Place(int k)

{

// 测试皇后k置于第x[k]列的合法性

for(int j = 1; j < k; ++ j)

if((abs(k-j) == abs(x[j]-x[k])) || (x[j]==x[k]))//p剪枝函数

return false;

return true;

}

bool Queen::Backtrack(int t)

{

// 解n后问题的回溯法

if(t > n)

{

for(int i=1; i<=n; ++i)

y[i] = x[i];

return true;

}

else

for(int i=1; i<=n; ++i)

{

x[t] = i;

if(Place(t) && Backtrack(t+1))

return true;

}

return false;

}

// k是第k行，x[k]表示第k行的皇后放在x[k]列

bool Queen::QueensLV(int stopVegas)

{

// 随机放置n个皇后的拉斯维加斯算法

RandomNumber rnd; // 随机数产生器

int k = 1; // 下一个放置的皇后编号

int count = 1;

// 1 <= stopVegas <= n 表示允许随机放置的皇后数

while((k <= stopVegas) && (count > 0))//放置一个皇后（第k个）

{

count = 0;

for(int i = 1; i<=n; ++i) //这里是遍历第k行所有可以放置的列号，用y保存下来，并用count记录有多少个位置可以放置

{//一次for循环把第k个皇后能放的位置都保存下来。

x[k] = i;

if(Place(k))//随机选一个

y[count++] = i;

}

if(count > 0) //这里利用上面保存的可以放置的列，然后随机取其中一列来放置第k行的皇后。这就是Las Vegas的精髓！

x[k++] = y[rnd.Random(count)]; // 随机位置count内挑

}

return (count > 0); // count > 0表示放置位置成功

}

//

// Queen.hpp

// 拉斯维加斯n后

//

// Created by zsy on 2018/6/7.

// Copyright © 2018年 zsy. All rights reserved.

//

#ifndef Queen\_hpp

#define Queen\_hpp

#include <stdio.h>

class Queen

{

friend bool nQueen(int);

private:

bool Place(int k); // 测试皇后k置于第x[k]列的合法性

bool Backtrack(int t); // 解n后问题的回溯法

bool QueensLV(int stopVegas); // 随机放置n个皇后拉斯维加斯算法

int n, \*x, \*y;

};

#endif

//

// main.cpp

// 拉斯维加斯n后

//

// Created by zsy on 2018/6/7.

// Copyright © 2018年 zsy. All rights reserved.

//

#include "Queen.hpp"

#include "RandomNumber.hpp"

#include <iostream>

using namespace std;

// 与回溯法结合的解n后问题的拉斯维加斯算法😤

bool nQueen(int n)

{

Queen X;

// 初始化X

X.n = n;

int \*p = new int[n+1];

int \*q = new int[n+1];

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

{

p[i] = 0;

q[i] = 0;

}

X.y = q;

X.x = p;

int stop = 3;//设置随机放置皇后的个数🙄

if(n > 15)

stop = n-15;

bool found = false;

while(! X.QueensLV(stop));

//算法的回溯搜索部分🤭

if(X.Backtrack(stop+1))

{

for(int i=1; i<=n; ++i)

cout << p[i] << " ";

found = true;

}

cout << endl;

delete [] p;

delete [] q;

return found;

}

int main()

{

int u;

cout<<"请问棋盘大小为多少？"<<endl;

cin>>u;

nQueen(u);

}