算法设计与分析 回溯与分支限界法

班级:2017211314 学号:2017213508 学生:蒋雪枫

一、综述：

回溯法是一种较为通用的算法，从根节点出发，深度优先搜索所有可能含有问题的解，适用于求解搜索可行解和优化问题，而这颗搜索树很多情况下都是排列树或者组合树。搜索过程中，要求算法能看到每一个“叶节点”，并且通过每个节点分支判定减少不必要的搜索，如有必要还要记录当前的求解路径。而分支限界法则以广度优先的方式加剪枝使问题逐渐向有最优解的分支推进。

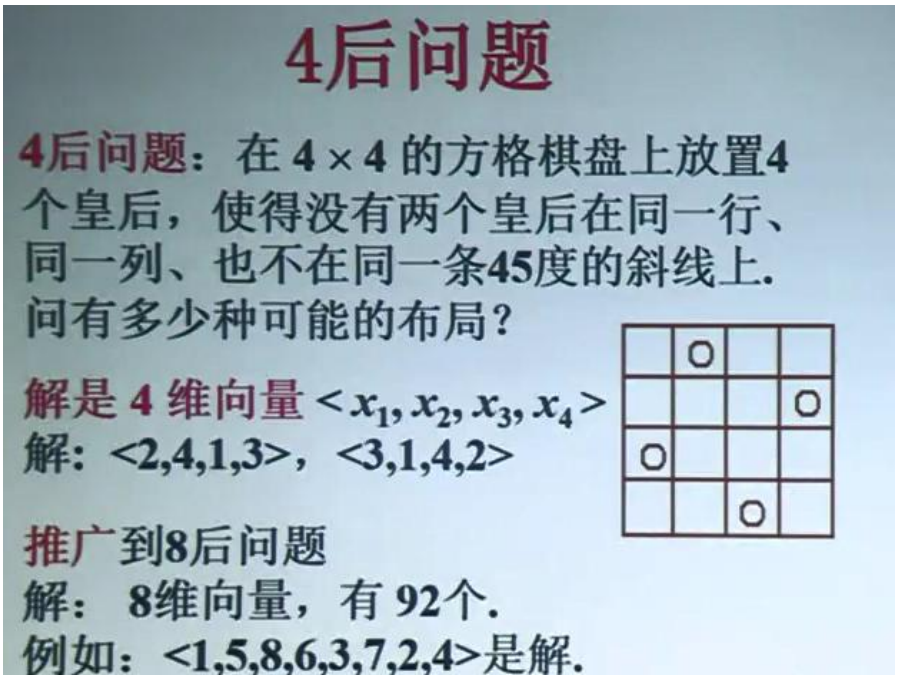
回溯法解决的问题一般具有”Domino”特性,即对 k+1 成立的 S(n)问题,同时对 k 成立，其逆否命题也平凡成立。同时，有时候可以用数学的一些特性，不用整体遍历解空间树，通过如”对称“的方式，来获取问题的所有解，比如着色问题，只是这里不过多说明了。本次实验，我把重点放在算法思想本身上，同时在完成报告的同时进行一定程度的复习。

介于期末邻近，本次实验完成地较为匆促，也略为粗糙，不过算法的思想与计算过程学生自己也已经基本掌握了。

二、N 皇后问题：

在课堂上，我们学习了 N 皇后问题。这里我会先总结一下 N 皇后问题的传统解法，然后分析老师给出的局部快速搜索算法。当然，N 皇后问题作为一种非常典型的搜索问题，历史上对它的研究也有很多，我们也知道有一种“Dancing Links”的优化算法。另外，我也在暑假看到过有通过位运算来加速该问题的一种解法，不过记忆有些久远了。

先说传统的算法:



在搜索这个 n=4 的 N-queen 问题时，每一个内部节点都有四个儿子，表示当前行上放的皇后到底处于哪一列，即该解空间树是一颗完全四叉树。虽然说是说“四叉树”，但我们是隐含地遍历整个解空间树，一般来说并没有必要去构造一颗树出来。

搜索约束条件：

1. 显约束

1≤xi ≤ n，即 xi =1,2, … ,n，每个皇后只能位于 1 ~ n 列

2. 隐约束

1) 任意 2 个皇后不同列： xi !=xj

2)任意 2 个皇后不处于同一正、反对角线：|i-j| !=|xi-xj|

那么我们需要判定当前在 k 行上，我们所放的 x【k】列是否满足条件，并且当 k 的情况成立，k+1 的情况才可能成立

```
bool Place(x[], int k) //x[]为包含皇后1,2,3,k-1,k位置的1个部分解
                       <x1, x2, …, xk-1, xk, ?, ?>
{
 for (int i=1; i<k; i++)          //判断第k个与第i=1,2,3,
                                  k-1个皇后之间位置是否冲突
 if ((abs(k-i)==abs(x[i]-x[k]))||(x[i]==x[k]))
    return false;          //k与第i<k个皇后存在对角线或列冲
                                  突，返回假
 return true;
}
```

而具体实现的迭代算法是：

```
void nQueen(int n,int X[])
{
  int k=1;
  x[1]=0;
  //外层循环 依次安排皇后 123...k...n
  while(k>0)
  {
    x[k]=x[k]+1;
    //内层循环,为第 k 个皇后选择位置,并判断是否合适
    while((x[k]<=n)&&(!place(X,k)))
      x[k]=x[k]+1;//当前列会引起冲突,考虑下一列

    //是否为第 k 个皇后搜索到了合适的位置
    if(x[k]<=n)
    {
      if(k==n) break; //搜索结束
      else
      {
        k=k+1;//扩展下一个皇后
        x[k]=0;
      }
    }
    //考虑了每一列,还是没有找到合适的位置存放第 k 个皇后,说明需要往上回溯
    else
```

```
    {
      x[k]=0; //第 k 号复原
      k=k-1; //往上回溯
    }

  }
}
```

但以上算法实际运行时间较长，于是老师给了我们基于局部快速搜索的 N 后求解算法。

- 解向量表示：

  对N个皇后，解向量为整数(1, 2, 3, ..., N)上的一个**排列**：
  pi(1), pi(2), pi(3),...,pi(N)，
  ，表示在第 i 行上面的皇后的列号是 pi(i)。

- 这种表示方法下，每个解能确保在每行、每列上恰好只有一个皇后，因此算法求解时，只需考虑**对角线冲突**

- 在快速局部搜索算法中，仅需考虑如何避免在正对角线、负对角线方向上出现2个或者2个以上的皇后

伪代码:

算法 1 Local Search for N-Queens
输入: N-皇后实例
输出: 解
Begin
    Do
    (1) 产生一个随机解 pi
    (2) for all i, j: where 皇后 pi(i) 或 pi(j) 有冲突 do
        (2.1) If 交换 pi(i) 与 pi(j) 能减少冲突
            (2.2) Then 交换 pi(i) 与 pi(j)
    Until 解无冲突
End

生成随机解 pi 没有考虑他们不同列，只是简单地随机初始化。避免了在这个环节的复杂度:

```
void GeneratePermutation()
{
    for (long i = 0; i < N; i++) pSolution[i] = i;

    int high, low;
    long rand_index;
```

```
            for (i = 0; i < N; i++)
            {
                // 考虑到 N 的数值可能很大，随机数由两段拼接而成
                high = rand() % sqrt_N;
                low = rand()% sqrt_N;
                rand_index = high * sqrt_N + low;

                Swap(pSolution[i],pSolution[rand_index]);
            }
        }
```

交换在程序中经常用到，采取内联方式：

```
inline void Swap(long &x, long &y)
{
    long temp;

    temp = y;
    y = x;
    x = temp;
}
```

主函数：

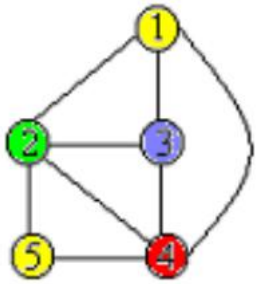
```
int main(int argc, char* argv[])
{
    clock_t start, finish;
    start = clock();
    sscanf(argv[1],"%ld",&N);
    sqrt_N = (int) sqrt(N);
    // 分配解的空间
    pSolution = new long[N]; // 使用下标 0 - (N-1)
    // 分配正对角线上的数组空间，并赋初始值为 0
    pPosDiagonal = new long[2*N-1];
    // 分配负对角线上的数组空间，并赋初始值为 0
    pNegDiagonal = new long[2*N-1];
    do {
        // 产生一个随机解
        GeneratePermutation();
        // 给两个对角线对应的皇后数数组赋值
        SetDiagonals();
        CountCollisions();
        bool flag = true;
        long gain;
        while (flag)
        {
            flag = false;
            for (long i = 0; i < N; i++)
```

```
                for (long j = i; j < N; j++)
                    // 若 pSolution[i] 或 pSolution[j]对应的对角线有冲突
                    if (pPosDiagonal[i - pSolution[i] + N -1] > 1 || pNegDiagonal[i + pSolution[i]] > 1 ||
                        pPosDiagonal[j - pSolution[j] + N -1] > 1 || pNegDiagonal[j + pSolution[j]] > 1)
                    {
                        // 判断一下，如果交换 pSolution[i]和 pSolution[j]，是否可以降低冲突总数
                        gain = SwapEvaluate(i, j);
                        // 若交换会带来冲突的减少，则值得交换
                        if ( gain > 0)
                        {
                            flag = true; // 标记，证明本轮迭代做了交换
                            SwapQueens(i, j, gain); // 交换 2 个皇后，并更新冲突数
                        }
                    }
            }
        } while (NumCollisions > 0);
        finish = clock();
        double duration = (double) (finish-start)/CLOCKS_PER_SEC;
        printf("time:%f seconds\n", duration);
        return 0;
    }
```

三、图着色问题：

采用非递归、迭代回溯算法。
利用可行性约束函数，对搜索过程进行剪枝。

约束函数：相邻结点不能着色相同：
x[i] ≠ x[j]，如果顶点i与顶点j邻接



数据结构：
int n; //顶点个数，顶点1,2,3…,n
int m; //最大颜色数，颜色1,2,..,m
int X[n]; // 各顶点的着色
BOOL a[n][n]: 图中顶点邻接矩阵， a[k][j]=1说明k与j相邻

并且设置 OK(int k)用于检查 1~k 个着色是否出现冲突。
用迭代法实现时，其实和 N 后问题差不多。
编写的源代码如下：

```
#include<iostream>
#include<math.h>
#include<fstream>
```

```
#include<cstdio>
using namespace std;
const int n=22;//vertex
const int bound=n+1;
int m=6; //color
int x[n+1];
bool a[n+1][n+1];
float path[n+1][n+1];

bool OK(int k)
{
    for(int j=1;j<k;j++)
    {
        if((a[k][j]==1)&&(x[j]==x[k]))
            return false;
    }
    return true;
}
void m_coloring(int n, int m, bool a[][bound], int x[])
{
    int i, k;
    for (i=1; i<=n; i++) //解向量 x[]初始化
        x[i]=0;
    k=1; //准备从第一个顶点开始
    while (k>=1) { //外层循环，控制为顶点 1,2,..,k,..n 着色
        x[k]=x[k]+1; //为顶点 k 选择下一颜色
        while ((x[k]<=m)&&(!OK(k))) //如果顶点 k 的当前着色 x[k]不合适
            x[k]=x[k]+1; //为顶点 k 选择下一颜色
        if (x[k]<=m)
            { //前面步骤是否为顶点 k 找到了合适的着色；如果没找到，x[k]>m
                if (k==n) break; //当前顶点是否是最后一个点？如果是，结束
                    else k=k+1; // 如果不是，回到外循环开始处，处理下一顶点 k+1
            }
        else
            { //没有为 k 找到合适着色，搜索失败
                x[k]=0; //置空顶点 k 的颜色
                k=k-1; //回溯到前一个顶点 k-1，搜索 k-1 的其他子节点
        }
} // 外循环结束
}
int main()
{
    ifstream in("station.txt",ios::in);
    int i=1;
```

```
    while(!in.eof())
    {
        in>>path[i][1]>>path[i][2]>>path[i][3]>>path[i][4]>>path[i][5]>>path[i][6]>>path[i][7]
>>path[i][8]>>path[i][9]>>path[i][10]>>path[i][11]>>path[i][12]>>path[i][13]>>path[i][14]
>>path[i][15]>>path[i][16]>>path[i][17]>>path[i][18]>>path[i][19]>>path[i][20]>>path[i][2
1]>>path[i][22];
        for(int ite=1;ite<=n;ite++)
        {
            if(fabs(path[i][ite]-99999)<1e-6)
                a[i][ite]=0;
            else
                a[i][ite]=1;
        }
        printf("i:%d 1:%lf 2:%lf\n",i,path[i][1],path[i][22]);
//      printf("%d %d\n",a[i][1],a[i][22]);
        i+=1;
    }
    m_coloring(n,m,a,x);
    for(int i=1;i<=n;i++)
    {
        cout<<"Vertex "<<i<<":Color index: "<<x[i]<<endl;
    }
}
```

运行结果示例(22 个基站):

```
Vertex 1:Color index: 1
Vertex 2:Color index: 1
Vertex 3:Color index: 2
Vertex 4:Color index: 2
Vertex 5:Color index: 1
Vertex 6:Color index: 1
Vertex 7:Color index: 3
Vertex 8:Color index: 3
Vertex 9:Color index: 4
Vertex 10:Color index: 4
Vertex 11:Color index: 2
Vertex 12:Color index: 2
Vertex 13:Color index: 5
Vertex 14:Color index: 3
Vertex 15:Color index: 3
Vertex 16:Color index: 6
Vertex 17:Color index: 5
Vertex 18:Color index: 5
Vertex 19:Color index: 6
Vertex 20:Color index: 2
Vertex 21:Color index: 5
Vertex 22:Color index: 4
```

四、TSP 问题：

源代码：

```
#include<iostream>
#include<algorithm>
#include<fstream>
#define MAX 100
using  namespace std;
int n=15;                          //城市个数
float a[MAX][MAX];                 //城市间距离
int x[MAX];                       //记录路径
int bestx[MAX]  = {0};            //记录最优路径
float bestp = 63355;              //最短路径长
float cp = 0;                     //当前路径长

void backpack(int t){
    if(t>n){
       if((a[x[n]][1])&&(a[x[n]][1]+cp<bestp)){
            bestp = a[x[n]][1]+cp;
            for(int i = 1;i<=n;i++){
              bestx[i] = x[i];
            }
       }
    }else{
        for(int i = t;i<=n;i++){
           /*约束为当前节点到下一节点的长度不为 0,限界为走过的长度+当前要走的长度之和小于最优长度*/
          if((a[x[t-1]][x[i]])&&(cp+a[x[t-1]][x[i]]<bestp)){
             swap(x[t],x[i]);
             cp+=a[x[t-1]][x[t]];
             backpack(t+1);
             cp-=a[x[t-1]][x[t]];
             swap(x[t],x[i]);
          }
        }
   }
}
int main(){
    ifstream in("15vertexs.txt",ios::in);
    int i=1;
    for(int i = 1;i<=n;i++)
```

```
    {
        x[i] = i;
    }
    while(!in.eof())
    {
        in>>a[i][1]>>a[i][2]>>a[i][3]>>a[i][4]>>a[i][5]>>a[i][6]>>a[i][7]>>a[i][8]>>a[i][9]>>a[i]
[10]>>a[i][11]>>a[i][12]>>a[i][13]>>a[i][14]>>a[i][15];
        printf("i:%d 1:%f 2:%f\n",i,a[i][1],a[i][15]);
        i+=1;
    }
    backpack(2);
    cout<<"最少旅行费用为: "<<bestp<<endl;
    cout<<"旅行路径为:"<<endl;
    for(int i = 1;i<=n;i++){
      cout<<bestx[i]<<" ";
    }
    cout<<bestx[1];
    return 0;
}
```

运行示例（n=15）：

```
C:\Users\Administrator\Desktop\Homework56\TSP.exe
i:1 1:0.000000 2:0.000000
i:2 1:0.000000 2:508.014923
i:3 1:586.661255 2:596.130432
i:4 1:0.000000 2:0.000000
i:5 1:417.392212 2:0.000000
i:6 1:0.000000 2:0.000000
i:7 1:0.000000 2:348.634125
i:8 1:0.000000 2:0.000000
i:9 1:240.849518 2:0.000000
i:10 1:203.789322 2:0.000000
i:11 1:655.972839 2:633.725891
i:12 1:0.000000 2:0.000000
i:13 1:0.000000 2:0.000000
i:14 1:0.000000 2:0.000000
i:15 1:0.000000 2:0.000000
最少旅行费用为: 5506.51
旅行路径为:
1 10 3 5 13 11 2 15 7 12 4 6 14 8 9 1
--------------------------------
Process exited after 0.3952 seconds with return value 0
请按任意键继续. . .
```