计算机算法设计与分析

回溯法

易凯

2017 年 4 月 18 日

班　　级 软件 53 班

学　　号 2151601053

邮　　箱 williamyi96@gmail.com

联系电话 13772103675

个人网站 https://williamyi96.github.io

williamyi.tech

实验日期 2017 年 5 月 18 日

提交日期 2017 年 6 月 6 日

目录

插图

1 回溯法基本框架

1.1 问题的解空间

问题的解空间就是至少包含问题的一个可能最优解的向量空间。

其中，问题的解一定要注意包含有显约束和隐约束两个层面的内容，其中显约束是指对每个分量 xi 满足的取值限定，而隐约束是为满足问题的解而对不同的分量之间施加的约束。

1.2 回溯法基本思想

回溯法是一种“通用的题解法”，其在求解问题的解空间找到最优解时，使用的是深度优先策略，从根节点出发搜索解空间树。对于任意一个结点，判断该结点是否包含问题的解，如果肯定不包含，则跳过对以该结点为根的子树的搜索，逐层向其祖先结点回溯。否则，进入该子树，继续使用深度优先策略。

扩展结点： 一个正在产生儿子的结点称之为扩展结点；

活结点： 一个自身已生成但其儿子还没有全部生成的结点；

死结点： 一个所有儿子已经产生的结点称之为死结点。

回溯法搜索解空间时，通常采用两种策略避免无效搜索。一种是用**约束函数**在扩展结点处剪去不满足约束的子树，另一种是用**限界函数**剪去得不到最优解的子树。

回溯法解题基本步骤 1. 针对所给问题，定义问题的解空间；

2. 确定易于搜索的解空间结构；

3. 以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索

1.3 递归回溯与迭代回溯

掌握递归回溯以及迭代回溯的基本使用框架。

1.4 子集树和排列树

子集树： 所给的问题就是从 n 个元素的集合 S 中找出满足某种性质的子集，得到相应的解空间树为子集树；

排列树： 所给的问题就是确定 n 个元素满足某种性质的排列，相应得到的解空间树称为排列树。

注意在用回溯法求解排列树的时候具有两次 swap 操作。

2 回溯法效率的依赖要素

1. 产生 x[k] 的时间；
2. 满足显约束的 x[k] 值的个数；
3. 计算约束函数 constraint 的时间；
4. 计算上界函数 Bound 的时间；
5. 满足约束函数和上界函数约束的所有 x[k] 的个数

3 装载问题回溯法改进

3.1 题目描述

用教材中的改进策略 1 重写装载问题回溯法，使得改进后算法的计算时间复杂性为 $O(2^n)$。

3.2 代码实现

该算法的改进为首先运行只计算最优解的算法，计算出最优装载量 W。由于该算法不记录最优解，故所需的计算时间为 $O(2^n)$。然后再运行改进后的算法 Backtrack，并在算法中将 bestw 置为 W。在首次达到的叶节点处(即首次遇到 i>n 时)，终止算法。由此返回的 bestx 即为最优解。

```
1    template<class T>
2    void Loading<T>::maxLoading(int i) {
3        if(i > n) {bestw = cw; return; }
4        r = r - w[i];
5        if(cw + w[i] <= c) {cw = cw + w[i]; maxLoading(i+1);}
6        if(cw + r > bestw) maxLoading(i+1);
```

```
7           r = r + w[i];
8       }
```

4 有向图回溯方法

4.1 题目描述

设 G 是一个有 n 个顶点的有向图，从顶点 i 发出的边的最小费用记为 min(i)。

1. 证明图 G 的所有前缀为 x[1:i] 的旅行售货员回路的费用至少为 $\sum_{j=2}^{i} a(x_{j-1}, x_j) + \sum_{j=i}^{n} min(x_j)$, 式子中，a(u,v) 是边 (u,v) 的费用。

2. 利用上述结论设计一个高效的上界函数，重写旅行售货员问题的回溯法，并与教材中的算法进行比较。

4.2 问题证明

证明过程后续进行处理。。。

5 排列宝石问题

5.1 题目描述

现有 n 种不同形状的宝石，每种 n 颗，共 n*n 颗。同一形状的 n 颗宝石分别具有 n 种不同的颜色 c1,c2,⋯,cn 中的一种颜色。欲将这 n*n 颗宝石排列成 n 行 n 列的一个方阵，使方阵中每一行和每一列的宝石都有 n 种不同的形状和 n 种不同颜色。是设计一个算法，计算出对于给定的 n，有多少种不同的宝石排列方案。

5.2 算法设计

对于给定的 n，计算出不同的宝石排列方案数。

5.3 数据输入

给定数据的输入，第一行有 1 个正整数 n，0<n<9；

5.4 数据输出

将计算的宝石排列方案数输出到屏幕上。

5.5 问题分析

利用回溯算法 backtrack，当行号（列号）大于 n 时，算法搜索至叶节点，当前找到可行性方案，sum+1；否者当前扩展节点是解空间中的内部节点，找出未排列的宝石，用 place 检验当前宝石是否可以放置，并以深度优先的方式递归的对可行子树搜索。

5.6 代码描述

```
#include<iostream>
#include<fstream>
using namespace std;

class Diamond {
public:
    int color;//颜色编号
    int shape;//形状编号
    int use;//是否已经排列，默认1为未排列
};

 //初始化n*n个宝石
void init(Diamond *a,int n) {
    //分别为n种颜色各具n种形状的宝石赋初值
    for(int i=1;i<=n*n;i++)  {
        a[i].color=(i-1)/n+1;
        a[i].shape=(i-1)%n+1;
        a[i].use=1;
    }
}

//检验宝石是否可放
bool place(Diamond *a,int **s,int x,int y) {
    for(int i=1;i<y;i++)  { //判断行中是否有颜色形状重复
        if(a[s[x][i]].color==a[s[x][y]].color || a[s[x][i]].shape==a[s[x][y]].shape)
            return 0;
```

```
27      }
28      for(int j=1;j<x;j++)  {  //判断列中是否有颜色形状重复
29          if(a[s[j][y]].color==a[s[x][y]].color || a[s[j][y]].shape==a[s[x][y]].shape)
30              return 0;
31      }
32      return 1;
33  }
34
35  //用回溯法递归搜索
36  void backtrack(Diamond *a,int **s,int t,int n,int &sum)  {
37      int x,y;
38      x=(t-1)/n+1;//存放的行号
39      y=(t-1)%n+1;//存放的列号
40      if(x>n)sum++;
41      else
42          for(int i=1;i<=n*n;i++)  {
43              if(a[i].use) { //当前宝石未排列
44                  s[x][y]=i;
45                  if(place(a,s,x,y)) { //当前宝石颜色形状不重复
46                  {
47                      a[i].use=0;
48                      backtrack(a,s,t+1,n,sum);
49                      a[i].use=1;
50                  }
51              }
52          }
53  }
54
55  //计算当前宝石排列的方案数
56  int numDiamond(int n) {
57      Diamond *a=new Diamond[n*n+1];
58      init(a,n);
59      int sum=0;
60      int **s=new int*[n+1];
61      for(int m=1;m<=n;m++)
62          s[m]=new int[n+1];
63      backtrack(a,s,1,n,sum);
64      return sum;
65  }
```

```
66
67 int main()  {
68     //读出输入文件中的数据
69     fstream fin;
70     fin.open("input.txt",ios::in);
71     if(fin.fail()) {
72         cout<<"File does not exist!"<<endl;
73         cout<<"Exit program"<<endl;
74         return 0;
75     }
76
77     int n;
78     fin>>n;
79
80     //调用函数
81     int number=numDiamond(n);
82     cout<<"宝石排列的方案数为: "<<number<<"种"<<endl;
83
84     //将结果数据写入到输出文件
85     fstream fout;
86     fout.open("output.txt",ios::out);
87     fout<<number;
88
89     fin.close();
90     fout.close();
91     system("pause");
92     return 0;
93 }
```

5.7 样例测试

```
宝石排列的方案数为: 1种
请按任意键继续. . .
```

图 1: 排列宝石问题样例测试