

实验三 回溯法

实验目的

学习编程实现深度优先搜索状态空间树求解实际问题的方法,着重体会求解第一个可行解和求解所有可行解之间的差别。加深理解回溯法通过搜索状态空间树、同时用约束函数剪去不含答案状态子树的算法思想,会用蒙特卡罗方法估计算法实际生成的状态空间树的结点数。

实验内容

一、要求用回溯法求解 **8-皇后问题**,使放置在 8×8 棋盘上的 8 个皇后彼此不受攻击,即:任何两个皇后都不在同一行、同一列或同一斜线上。请输出 8 皇后问题的所有可行解。

二、用回溯法编写一个**递归程序**解决如下**装载问题**:有 n 个集装箱要装上 2 艘载重分别为 c_1 和 c_2 的轮船,其中集装箱 i 的重量为 w_i ($1 \leq i \leq n$),且 $\sum_{i=1}^n w_i \leq c_1 + c_2$ 。问是否有一个合理的装载方案可以将这 n 个集装箱装上这 2 艘轮船?如果有,请给出装载方案。

提示:参考子集和数问题的求解方法。

举例:当 $n=3$, $c_1=c_2=50$, 且 $w=[10,40,40]$ 时,可以将集装箱 1 和 2 装到第一艘轮船上,集装箱 3 装到第二艘轮船上;如果 $w=[20,40,40]$ 时,无法将这 3 个集装箱都装上轮船。

实验步骤

一、8 皇后问题

通过求解 n -皇后问题,体会回溯法深度优先遍历状态空间树,并利用约束函数进行剪枝的算法思想。

```
#include <iostream>
using namespace std;
#include <math.h>
bool Place(int k,int i,int *x)           //判定两个皇后是否在同一列或同一斜线上
{
    for (int j=0;j<k;j++)
        if ((x[j]==i)||((abs(x[j]-i)==abs(j-k)))) return false;
    return true;
}
void NQueens(int k,int n,int *x)         //递归函数(求解n皇后问题)
{
    for (int i=0;i<n;i++)
    {
        if(Place(k,i,x))
        {
            x[k]=i;
```

```

        if (k==n-1)
        {
            for (i=0;i<n;i++) cout<<x[i]<<" ";
            cout<<endl;
        }
        else
        {   NQueens(k+1,n,x);
        }
    }
}

void NQueens(int n,int *x)
{
    NQueens(0,n,x);
}

void main()
{
    int x[8];
    for (int i=0;i<8;i++) x[i]=-1;
    NQueens(8,x);
}

```

二、装载问题

1、如果给定的装载问题有解，则采用下面的策略可以得到一个最优装载方案：

- (1) 首先将第一艘轮船尽可能装满；
- (2) 然后将剩余的集装箱装到第二艘轮船上。

将第一艘轮船尽可能装满，等价于选取全体集装箱的一个子集，使该子集中集装箱重量之和最接近 c_1 。由此可知，装载问题等价于以下特殊的0-1背包问题：

$$\max \sum_{i=1}^n w_i x_i$$

$$\sum_{i=1}^n w_i x_i \leq c_1$$

$$x_i \in \{0,1\}, 1 \leq i \leq n$$

2、求最优解值时，用变量 cw 表示当前实际载重量， $bestw$ 表示当前最优载重量，

$r = \sum_{j=i+1}^n w_j$ 是剩余集装箱的总重量。

该问题解空间可用子集树表示。子集树中可行结点的左儿子结点表示 $x[i]=1$ 的情形，右儿子结点表示 $x[i]=0$ 的情形。引入约束函数用于剪去不含可行解和最优解的子树：

- ◆ （约束函数）用来剪去不含可行解的分枝：仅当 $cw+w[i] \leq c$ 时进入可行结点的左子树，否则剪去左子树。而可行结点的右儿子结点总是可行的，故进入右子树时不需检查可行性。
- ◆ （限界函数）用来剪去不含最优解的分枝：由于当前结点的左孩子 $cw+r$ 值保持不变，

因此无需检查。而仅当右孩子 $cw+r>bestw$ 时才需要进入右子树搜索，否则应对右子树进行剪枝。

- ◆ 引入剪去不含最优解分枝的剪枝函数后，在达到一个叶节点时，就不必再检查该叶节点是否优于当前最优解，因为算法搜索到的每个叶节点都是迄今为止找到的最优解。

3、为了在算法中记录与当前最优值相应的当前最优解，需要增加两个数组数据成员 x 和 $bestx$ 。 x 用于记录从根至当前结点的路径； $bestx$ 记录当前最优解。算法每搜索到一个叶结点处，就修正 $bestx$ 的值。

4、算法实现主体部分如下，请补充完整，并使用下面三个测试案例调试通过。

第一艘船载重60，第二艘船载重40，5个集装箱重量分别为：

(1) 22 35 24 19 4

(2) 22 35 24 15 4

(3) 22 35 24 15 3

```
template <class T>
class Loading
{
private:
    int n, //集装箱数
        *x, //当前解
        *bestx; //当前第一艘船的最优解
    T c1, //第一艘轮船的核定载重量
        c2, //第二艘轮船的核定载重量
        *w, //集装箱重量数组
        total, //所有集装箱重量之和
        cw, //当前第一艘船的载重量
        bestw, //当前第一艘船的最优载重量
        r; //剩余集装箱总重量

public:
    Loading() //构造函数
    {
        ..... //（请自行补充）
    }

    ~Loading() //析构函数
    {
        ..... //（请自行补充）
    }

    void Backtrack(int i); //找到最接近第一艘轮船载重c1的最佳装载方案，
                          //最优载重值bestw，最优解数组bestx。

    void Show(); //输出整个装载方案
};

template <class T>
void Loading<T>::Backtrack(int i)
{
    //搜索第i层结点
    if (i>n)
    {
        //到达叶节点
        if (cw>bestw)
```

```

        {
            for (int j=1;j<=n;j++) bestx[j]=x[j];
            bestw=cw;
        }
        return;
    }
    //搜索子树
    r=w[i];
    if (cw+w[i]<=c1)    //x[i]=1时的可行解约束条件
    { //搜索左子树
        x[i]=1;
        cw+=w[i];
        Backtrack(i+1);
        cw-=w[i];
    }
    if (cw+r>bestw)    //x[i]=0时增加的约束函数，剪去不含最优解的分枝
    { //搜索右子树
        x[i]=0;
        Backtrack(i+1);
    }
    r+=w[i];
}

template <class T>
void Loading<T>::Show()
{
    .....    //（请自行补充）
}

void main()
{
    Loading<int> ld;
    ld.Backtrack(1);
    ld.Show();
    system("pause");
}

```

思考

1、（1）请编程实现从 n-皇后问题的所有 92 种可行解中筛选出 12 种独立解，而其余的解都可以由这些独立解利用对称性或旋转而得到。

（2）若 n-皇后问题要求在求得第一个可行解之后算法即终止，请编程实现。

提示： 可以用 flag 标志是否已经求得第一个可行解。根据 flag 的值决定是否继续进行本层

调用，还是直接返回上层调用点。

2、求上面回溯法求解装载问题的计算时间复杂度？有什么方法可以继续改进算法的时间复杂度？

由于bestx可能被更新 $O(2^n)$ 次，因此该算法的时间复杂度是 $O(n 2^n)$ 。

改进策略可以有下面两种，均可将算法的时间复杂度降为 $O(2^n)$ ：

- (1) 首先运行只计算最优值的算法，计算出最优装载量W，所耗时间 $O(2^n)$ 。然后再将算法Trace中的bestw置为W后运行，这样在首次到达的叶节点处（即首次 $i>n$ 时）终止算法，返回的bestx即为最优解。
- (2) 另一种策略是在算法中动态的更新bestx。在第i层的当前结点处，当前最优解由 $x[j]$ ， $1 \leq j < i$ 和bestx[j]， $i \leq j \leq n$ 所组成。每当算法回溯一层时，将 $x[i]$ 存入bestx[i]。

3、请用非递归的**迭代**回溯方式，重新实现装载问题的求解。

提示：用变量i记录迭代层深度。