第5章 走不下去就回退——回溯法

5.1 单项选择题及其参考答案

5.1.1 单项选择题

1. 回溯法是在问题的解空间中按_____策略从根结点出发搜索的。

A. 广度优先　　B. 活结点优先

C. 扩展结点优先　　D. 深度优先

2. 下列算法中_____通常以深度优先方式搜索问题的解。

A. 回溯法　　B. 动态规划　　C. 贪心法　　D. 分支限界法

3. 关于回溯法,以下叙述中不正确的是_____。

A. 回溯法有通用解题法之称,可以系统地搜索一个问题的所有解或任意解

B. 回溯法是一种既带系统性又带有跳跃性的搜索算法

C. 回溯算法需要借助队列来保存从根结点到当前扩展结点的路径

D. 回溯算法在生成解空间的任一结点时,先判断该结点是否可能包含问题的解,如果肯定不包含,则跳过对该结点为根的子树的搜索,逐层向祖先结点回溯

4. 回溯法的效率不依赖于下列因素_____。

A. 确定解空间的时间　　B. 满足显式约束的值的个数

C. 计算约束函数的时间　　D. 计算限界函数的时间

5. 下面_____是回溯法中为避免无效搜索采取的策略。

A. 递归函数　　B. 剪枝函数　　C. 随机数函数　　D. 搜索函数

6. 对于含有 n 个元素的子集树问题(每个元素二选一),最坏情况下解空间树的叶子结点个数是_____。

A. $n!$　　B. 2^n　　C. $2^{n+1}-1$　　D. 2^{n-1}

7. 用回溯法求解 0/1 背包问题时的解空间是_____。

A. 子集树　　B. 排列树

C. 深度优先生成树　　D. 广度优先生成树

8. 用回溯法求解 0/1 背包问题时的最坏时间复杂度是_____。

A. $O(n)$　　B. $O(n\log_2 n)$　　C. $O(n\times 2^n)$　　D. $O(n^2)$

9. 用回溯法求解 TSP 问题时的解空间是_____。

A. 子集树　　B. 排列树

C. 深度优先生成树　　D. 广度优先生成树

10. 有 n 个学生,每个人有一个分数,求最高分的学生的姓名,最简单的方法是_____。

A. 回溯法　　B. 归纳法　　C. 迭代法　　D. 以上都不对

11. 求中国象棋中马从一个位置到另外一个位置的所有走法,采用回溯法求解时对应的解空间是_____。

A. 子集树　　B. 排列树

C. 深度优先生成树　　D. 广度优先生成树

12. n 个人排队在一台机器上做某个任务，每个人的等待时间不同，完成他的任务的时间不同，求完成这 n 个任务的最小时间，采用回溯法求解时对应的解空间是_____。

A. 子集树　　B. 排列树

C. 深度优先生成树　　D. 广度优先生成树

5.1.2 单项选择题参考答案

1. **答**：回溯法采用深度优先搜索在解空间中搜索问题的解。答案为 D。

2. **答**：回溯法采用深度优先搜索在解空间中搜索问题的解，分支限界法采用广度优先搜索在解空间中搜索问题的解。答案为 A。

3. **答**：回溯算法是采用深度优先遍历的，需要借助栈保存从根结点到当前扩展结点的路径。答案为 C。

4. **答**：回溯法的解空间是虚拟的，不必事先确定整个解空间。答案为 A。

5. **答**：剪支函数包括约束函数(在扩展结点处剪去不满足约束条件的路径)和限界函数(剪去得不到问题的解或最优解的路径)。答案为 B。

6. **答**：这样的解空间树是一棵高度为 $n+1$ 的满二叉树，叶子结点恰好有 2^n 个。答案为 B。

7. **答**：在 0/1 背包问题中每个物品是二选一(要么选中，要么不选中)，与物品的顺序无关，对应的解空间为子集树类型。答案为 A。

8. **答**：0/1 背包问题的解空间是一棵高度为 $n+1$ 的满二叉树，结点个数为 $2^{n+1}-1$，最坏情况下搜索全部结点。答案为 C。

9. **答**：TSP 问题的解空间属于典型的排列树，因为路径与顶点的顺序有关。答案为 B。

10. **答**：最简单的方法是依次迭代比较求最高分数。答案为 C。

11. **答**：每一步马从相邻可走的位置中选择一个位置走下去。答案为 A。

12. **答**：该问题是求 1～n 的某个排列，对应 n 个任务完成的最小时间。答案为 B。

5.2 问答题及其参考答案

5.2.1 问答题

1. 回溯法的搜索特点是什么？

2. 有这样一个数学问题，x 和 y 是两个正实数，求 $x+y=3$ 的所有解，请问能否采用回溯法求解？如果 x 和 y 是两个均小于或等于 10 的正整数，又能否采用回溯法求解？如果能够，请采用解空间画出求解结果。

3. 对于 $n=4$，$a=(11,13,24,7)$，$t=31$ 的子集和问题，利用左、右剪支的回溯法算法求解，求出所有解并且画出解空间中的搜索过程。

4. 对于 n 皇后问题，通过解空间说明 $n=3$ 时是无解的。

5. 对于 n 皇后问题，有人认为当 n 为偶数时其解具有对称性，即 n 皇后问题的解个数

恰好为 $n/2$ 皇后问题的解个数的两倍，这个结论正确吗？

6. 请问能否采用解空间为排列树的回溯框架求解 n 皇后问题？如果能，请给出剪支操作，说明最坏情况下的时间复杂度，按照最坏情况下的时间复杂度比较，哪个算法更好？

7. 对于如图 5.1 所示的无向连通图，假设颜色数 $m=2$，给出 m 着色的所有着色方案，并且画出对应的解空间。

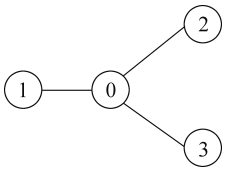

图 5.1　一个无向连通图

8. 有一个 0/1 背包问题，物品个数 $n=4$，物品编号分别为 0～3，它们的重量分别是 3、1、2 和 2，价值分别是 9、2、8 和 6，背包容量 $W=3$。利用左、右剪支的回溯法算法求解，并且画出解空间中的搜索过程。

9. 以下算法用于求 n 个不同元素 a 的全排列，当 $a=(1,2,3)$ 时，请给出算法输出的全排列的顺序。

```
int cnt=0;                                  //累计排列的个数
void disp(int a[]){                         //输出一个解
    System.out.printf(" 排列%2d: (",++cnt);
    for(int i=0;i<a.length-1;i++)
        System.out.printf("%d,",a[i]);
    System.out.printf("%d)",a[a.length-1]);
    System.out.println();
}
void swap(int a[],int i,int j) {            //交换 a[i]与 a[j]
    int tmp=a[i];
    a[i]=a[j]; a[j]=tmp;
}
void dfs(int a[],int i) {                   //递归算法
    int n=a.length;
    if(i>=n-1)                              //递归出口
        disp(a);
    else {
        for(int j=n-1;j>=i;j--) {
            swap(a,i,j);                    //交换 a[i]与 a[j]
            dfs(a,i+1);
            swap(a,i,j);                    //交换 a[i]与 a[j]:恢复
        }
    }
}
void perm(int a[]) {                        //求 a 的全排列
    dfs(a,0);
}
```

10. 假设问题的解空间为 $(x_0,x_1,\cdots,x_{n-1})$，每个 x_i 有 m 种不同的取值，所有 x_i 取不同的值，该问题既可以采用子集树递归回溯框架求解，也可以采用排列树递归回溯框架求解，考虑最坏时间性能应该选择哪种方法？

11. 以下两个算法都是采用排列树递归回溯框架求解任务分配问题，判断其正确性，如果不正确，请指出其中的错误（其中，swap(x,i,j)用于交换 $x[i]$和 $x[j]$）。

(1) 算法 1：

```
void dfs(int x[],int cost,int i) {          //回溯算法
    if(i>n) {                               //到达叶子结点
        if(cost<bestc) {                    //比较求最优解
            bestc=cost;
            bestx=x;
        }
    }
    else {                                  //没有到达叶子结点
        for(int j=1;j<=n;j++) {             //为人员 i 试探任务 x[j]
            if(task[x[j]]) continue;        //若任务 x[j]已经分配,则跳过
            task[x[j]]=true;
            cost+=c[i][x[j]];
            swap(x,i,j);                    //为人员 i 分配任务 x[j]
            if(bound(x,cost,i)<bestc)       //剪支
                dfs(x,cost,i+1);            //继续为人员 i+1 分配任务
            swap(x,i,j);
            cost-=c[i][x[j]];               //cost 回溯
            task[x[j]]=false;               //task 回溯
        }
    }
}
```

(2) 算法 2：

```
void dfs(int x[],int cost,int i) {          //回溯算法
    if(i>n) {                               //到达叶子结点
        if(cost<bestc) {                    //比较求最优解
            bestc=cost;
            bestx=x;
        }
    }
    else {                                  //没有到达叶子结点
        for(int j=1;j<=n;j++) {             //为人员 i 试探任务 x[j]
            if(task[x[j]]) continue;        //若任务 x[j]已经分配,则跳过
            swap(x,i,j);                    //为人员 i 分配任务 x[j]
            task[x[j]]=true;
            cost+=c[i][x[j]];
            if(bound(x,cost,i)<bestc)       //剪支
                dfs(x,cost,i+1);            //继续为人员 i+1 分配任务
            cost-=c[i][x[j]];               //cost 回溯
            task[x[j]]=false;               //task 回溯
            swap(x,i,j);
        }
    }
}
```

5.2.2 问答题参考答案

1. **答**：回溯法的搜索特点是深度优先搜索＋剪支。深度优先搜索可以尽快地找到一个解，剪支函数可以终止一些路径的搜索，提高搜索性能。

2. **答**：当 x 和 y 是两个正实数时，理论上讲两个实数之间有无穷个实数，所以无法枚举 x 和 y 的取值，不能采用回溯法求 $x+y=3$ 的所有解。

当 x 和 y 是两个均小于或等于 10 的正整数时，它们的枚举范围是有限的，可以采用回溯法求 $x+y=3$ 的所有解，采用剪支仅扩展 $x,y\in[1,2]$的结点。解向量是(x,y)，对应的解空间如图 5.2 所示，找到的两个解是(1,2)和(2,1)。

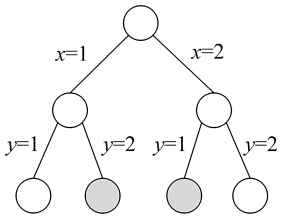

图 5.2 求 $x+y=3$ 的解空间

3. **答**：利用左、右剪支的回溯法算法求出两个解如下。

```
第 1 个解: 选取的数为 11 13 7
第 2 个解: 选取的数为 24 7
```

在解空间中的搜索过程如图 5.3 所示，图中每个结点为(cs,rs)，其中 cs 为考虑第 i 个整数时选取的整数和，rs 为剩余整数和。题中实例搜索的结点个数是 11，如果不剪支，需要搜索 31 个结点。

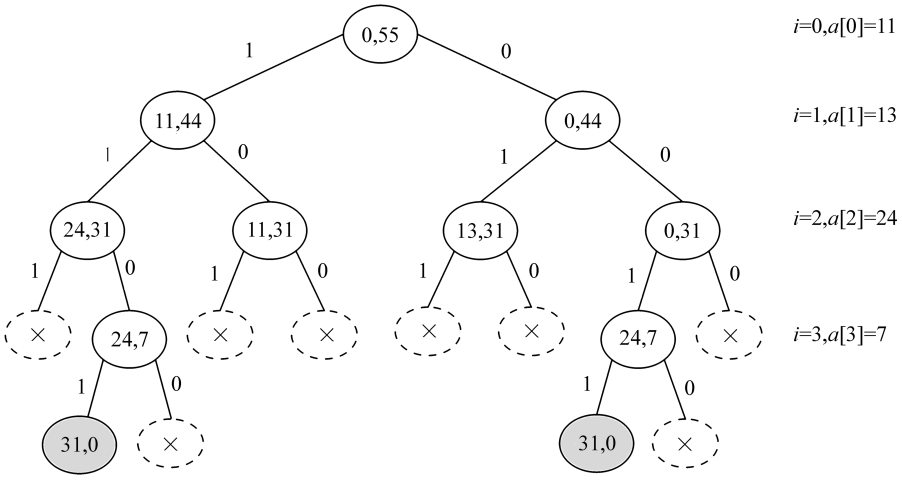

图 5.3 子集和问题的搜索过程

4. **答**：$n=3$ 时的解向量为(x_1,x_2,x_3)，x_i 表示第 i 个皇后的列号，对应的解空间如图 5.4 所示，所有的叶子结点均不满足约束条件，所以无解。

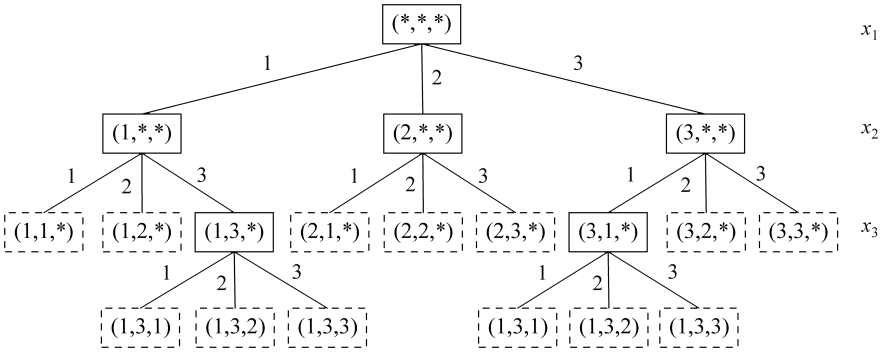

图 5.4 3 皇后问题的解空间

5. **答**：这个结论是错误的，因为两个 $n/2$ 皇后问题的解合并起来不是 n 皇后问题的解。

6. **答**：设 n 皇后问题的解向量为 $(x_1,x_2,\cdots,x_n)$，x_i 表示第 i 个皇后的列号，显然每个解一定是 $1\sim n$ 的某个排列，所以可以采用解空间为排列树的回溯框架求解 n 皇后问题。其剪支操作是任何两个皇后不能同行、同列和同两条对角线。在最坏情况下该算法的时间复杂度为 $O(n\times n!)$，由于 $O(n\times n!)$ 好于 $O(n\times n^n)$，所以按照最坏情况下的时间复杂度比较，解空间为排列树的回溯算法好于解空间为子集树的回溯算法。

7. **答**：这里 $n=4$，顶点编号为 $0\sim3$，$m=2$，颜色编号为 0 和 1，解向量为 (x_0,x_1,x_2,x_3)，x_i 表示顶点 i 的着色，对应的解空间如图 5.5 所示，着色方案有两种，分别是(0,1,1,1)和(1,0,0,0)。

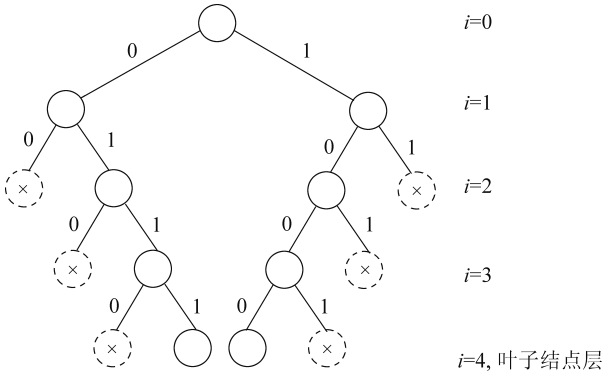

图 5.5　m 着色问题的解空间

8. **答**：求解过程如下。

① 4 个物品按 v/w 递减排序后的结果如表 5.1 所示。

② 从 $i=0$ 开始搜索对应的解空间如图 5.6 所示。

表 5.1　4 个物品按 v/w 递减排序后的结果

序号 i	物品编号 no	重量 w	价值 v	v/w
0	2	2	8	4
1	0	3	9	3
2	3	2	6	3
3	1	1	2	2

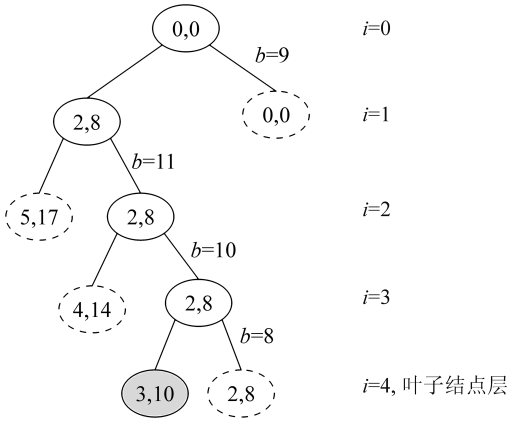

图 5.6　0/1 背包问题的解空间

最后得到的最优解是选择编号为 1 和 2 的物品，总重量为 3，总价值是 10。

9. **答**：当 $a=(1,2,3)$ 时，调用 perm(a)的输出结果及其顺序如下。

```
排列 1: (3,1,2)
排列 2: (3,2,1)
排列 3: (2,3,1)
排列 4: (2,1,3)
排列 5: (1,3,2)
排列 6: (1,2,3)
```

10. **答**：一般情况下，这样的问题采用子集树递归回溯框架求解时最坏时间复杂度为$O(m^n)$，采用排列树递归回溯框架求解时最坏时间复杂度为$O(n!)$，如果$m=2$，由于$O(2^n)<O(n!)$，采用前者较好；如果m接近n，由于$O(n^n)>O(n!)$，采用后者较好。

11. **答**：算法1是正确的。算法2不正确，在执行第一个swap($x[i]$,$x[j]$)后已经为人员i分配了任务$x[j]$，应该置task[$x[i]$]=true，cost+=$c[i][x[i]]$，后面的回溯恢复过程也是如此。

5.3 算法设计题及其参考答案

5.3.1 算法设计题

1. 给定含n个整数的序列a(其中可能包含负整数)，设计一个算法从中选出若干整数，使它们的和恰好为t。例如，$a=(-1,2,4,3,1)$，$t=5$，求解结果是(2,3,1,−1)、(2,3)、(2,4,−1)和(4,1)。

2. 给定含n个正整数的序列a，设计一个算法从中选出若干整数，使它们的和恰好为t并且所选元素个数最少的一个解。

3. 给定一个含n个不同整数的数组a，设计一个算法求其中$m(m\leqslant n)$个元素的组合。例如，$a=\{1,2,3\}$，$m=2$，输出结果是{{1,2},{1,3},{2,3}}。

4. 设计一个算法求1～n中$m(m\leqslant n)$个元素的排列，要求每个元素最多只能取一次。例如，$n=3$，$m=2$时的输出结果是{{1,2},{1,3},{2,1},{2,3},{3,1},{3,2}}。

5. 在求n皇后问题的算法中每次放置第i个皇后时，其列号x_i的试探范围是1～n，实际上前面已经放好的皇后的列号是不必试探的，请根据这个信息设计一个更高效的求解n皇后问题的算法。

6. 请采用基于排列树的回溯框架设计求解n皇后问题的算法。

7. 一棵整数二叉树采用二叉链b存储，设计一个算法求根结点到每个叶子结点的路径。

8. 一棵整数二叉树采用二叉链b存储，设计一个算法求根结点到叶子结点的路径中路径和最小的一条路径，如果这样的路径有多条，求其中的任意一条。

9. 一棵整数二叉树采用二叉链b存储，设计一个算法产生每个叶子结点的编码。假设从根结点到某个叶子结点a有一条路径，从根结点开始，路径走左分支时用0表示，走右分支时用1表示，这样的0/1序列就是a的编码。

10. 假设一个含n个顶点(顶点编号为0～($n-1$))的不带权图采用邻接矩阵$\boldsymbol{A}$存储，设计一个算法判断其中顶点u到顶点v是否有路径。

11. 假设一个含 n 个顶点(顶点编号为 0～($n-1$))的不带权图采用邻接矩阵 $\boldsymbol{A}$ 存储,设计一个算法求其中顶点 u 到顶点 v 的所有路径。

12. 假设一个含 n 个顶点(顶点编号为 0～($n-1$))的带权图采用邻接矩阵 $\boldsymbol{A}$ 存储,设计一个算法求其中顶点 u 到顶点 v 的一条路径长度最短的路径。一条路径的长度是指路径上经过的边的权值和。如果这样的路径有多条,求其中的任意一条。

13. 给定一个 $m \times n$ 的迷宫,每个方格值为 0 时表示空白,为 1 时表示障碍物,在行走时最多只能走到上、下、左、右相邻的方格。设计一个回溯算法求从指定入口 s 到指定出口 t 的所有迷宫路径和其中一条最短路径。

14. 给定一个不带权连通图,由指定的起点前往指定的终点,途经所有其他顶点且只经过一次,称为哈密顿路径,闭合的哈密顿路径称作哈密顿回路。设计一个算法求无向图的所有哈密顿回路。

15. 采用回溯法求解最优调度问题,假设有 n 个任务由 k 个可并行工作的机器来完成,完成任务 $i(0 \leqslant i < n)$需要的时间为 t_i,设计一个算法求完成这 n 个任务的最少时间。例如,$n=10$,$t=\{67,45,80,32,59,95,37,46,28,20\}$,$k=7$,完成这 10 个任务的最少时间是 95。

5.3.2 算法设计题参考答案

1. **解**:由于 a 中可能包含负整数,甚至 t 有可能是负数,无法剪支,采用求 a 的幂集的思路,相当于求出 a 的所有子集并且累计子集和 cs,当到达一个叶子结点时,若满足 cs=t 就是一个解。对应的算法如下:

```
int a[];
int n;
int t;
int x[];                    //解向量
int sum;                    //累计组合个数
void dfs(int cs,int i) {    //回溯算法
    if(i>=n){               //到达一个叶子结点
        if(cs==t){          //找到一个解
            System.out.printf(" (%d): ",++sum);
                for(int j=0;j<n;j++) {
                    if(x[j]==1)
                        System.out.printf("%d ",a[j]);
                }
            System.out.println();
        }
    }
    else {                  //没有到达叶子结点
        x[i]=1; cs+=a[i];
        dfs(cs,i+1);        //选择 a[i]
        cs-=a[i];           //回溯 cs
        x[i]=0;
        dfs(cs,i+1);        //不选择 a[i]
    }
}
```

```
void subs4(int a[],int t) {                    //求解子集和问题
    this.a=a;
    this.t=t;
    n=a.length;
    x=new int[n];                              //指定 x 的长度为 n
    sum=0;
    dfs(0,0);
}
```

说明：有人说一旦找到了 cs=t 就输出一个解(看成不再选择后面的元素)，所以将输出解的条件改为满足 $i<n$ && cs=t。这样是错误的，因为这样做的结果是输出一个解后就停止了该结点的扩展，后面的解就找不到了，例如(2,3) 是一个解，这样修改就找不到(2,3,1,−1) 这个解了。

2. **解**：属于典型的解空间为子集树的问题，采用子集树的回溯算法框架求解。当找到一个解后通过对选取的元素个数进行比较求最优解 bestx 和 bestm，剪支原理见《教程》中5.3.5 节的求解子集和问题的算法。对应的算法如下：

```
int a[];
int n;
int t;
int bestx[];                                   //最优解向量
int bestm;                                     //选择的最少元素个数
void dfs(int cs,int rs,int x[],int m,int i) {  //回溯算法
    if(i>=n){                                  //到达一个叶子结点
        if(cs==t){                             //找到一个解
            if(m<bestm){                       //找更优解
                bestm=m;
                bestx=Arrays.copyOf(x,n);
            }
        }
    }
    else {                                     //没有到达叶子结点
        rs-=a[i];                              //求剩余的整数和
        if(cs+a[i]<=t) {                       //左孩子结点剪支
            x[i]=1;                            //选取整数 a[i]
            dfs(cs+a[i],rs,x,m+1,i+1);
        }
        if(cs+rs>=t){                          //右孩子结点剪支
            x[i]=0;                            //不选取整数 a[i]
            dfs(cs,rs,x,m,i+1);
        }
        rs+=a[i];                              //恢复剩余整数和
    }
}
void subs(int a[],int t) {                     //求解子集和问题
    this.a=a;
    n=a.length;
    this.t=t;
    int x[]=new int[n];                        //解向量
    int rs=0;                                  //表示所有整数和
```