第5章

回　溯　法

5.1　概　　述

在现实世界中，很多问题没有（至少目前没有）有效的算法，这些问题的求解只能通过蛮力穷举搜索来实现。但蛮力法需要生成并评估所有的解，执行效率低下。

回溯法（Back Tracking Method）是一种深度优先搜索算法，该方法试图穷举所有可能的解，因此，回溯法能够确保获得最优解。在搜索过程中根据问题约束等规则对搜索树进行剪枝操作，跳过部分搜索区域，提高算法性能。回溯法按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再选择另外一个方向试探的技术即为回溯法，满足回溯条件的某个状态的点称为“回溯点”。回溯法把问题的解空间转化成了图或者树的结构表示，然后使用深度优先搜索策略进行遍历，遍历的过程中记录和寻找所有可行解或者最优解。

5.2　回溯法设计思路

复杂问题常常有很多可能解，这些可能解构成问题的解空间，也就是在穷举法中提到的所有可能解的搜索空间。一般而言，解空间中应该包括所有的可能解。

回溯法按深度优先策略搜索问题的解空间树。首先从根结点出发搜索解空间树，当算法搜索至解空间树的某一结点时，先利用剪枝函数判断该结点是否可行（即能得到问题的解）。如果不可行，则跳过对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，继续按深度优先策略搜索。

5.3　回溯法示例与过程分析

5.3.1　n 皇后问题

例 5.1　在 $n \times n$ 格的国际象棋棋盘上摆放 n 个皇后（见图 5.1，此时 $n=8$），

使其不能互相攻击,即任意两个皇后都不能处于同一行、同一列或同一斜线上,有多少种摆法?

n 皇后是由八皇后问题演变而来的。该问题由国际象棋棋手马克斯·贝瑟尔于 1848 年提出:在 8×8 格的国际象棋棋盘上摆放 8 个皇后,使其不能互相攻击,即任意两个皇后都不能处于同一行、同一列或同一斜线上,求有多少种摆法。高斯认为有 76 种方案。1854 年在柏林的象棋杂志上不同的作者发表了 40 种不同的解,后来有人用图论的方法解出 92 种结果。

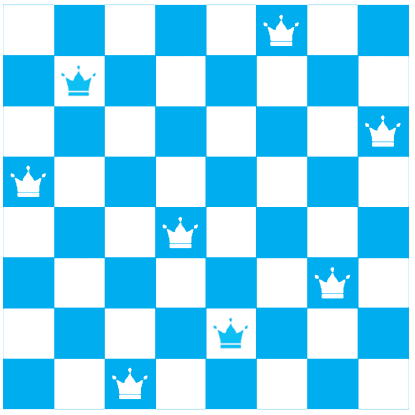


图 5.1 八皇后问题示意图

下面用数组模拟棋盘,从第一行开始,依次选择位置,如果当前位置满足条件,则向下选位置,如果不满足条件,那么当前位置后移一位。以四皇后为例,从第一行开始,依次选择位置,如果当前位置满足条件,则向下选位置,如果不满足条件,那么返回上一步。

如图 5.2(a)所示,第一个皇后放置于第一行第一列(1-1),则第二个皇后可放置于(2-3),此时第三个皇后所有位置均不可放置,返回上一步。

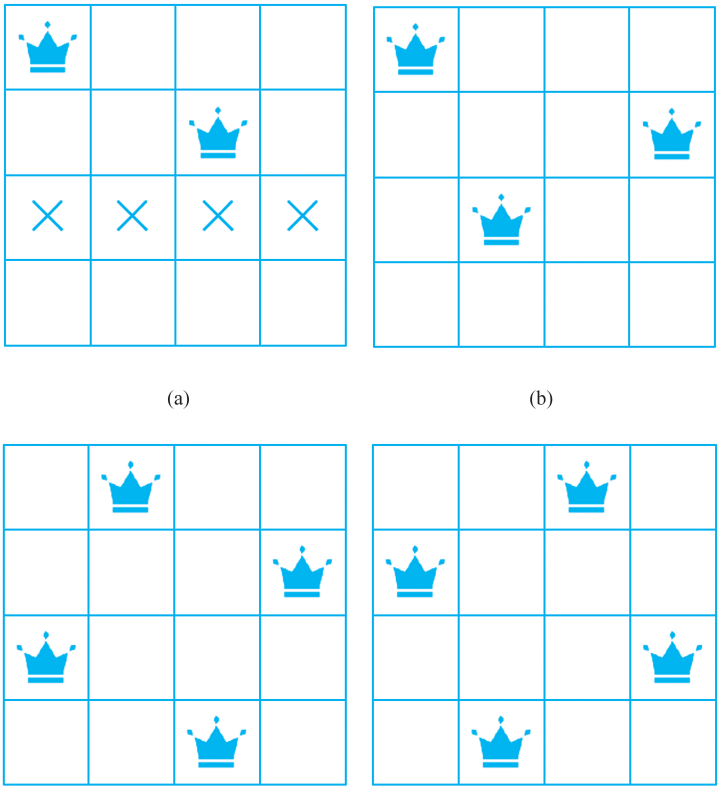


图 5.2 四皇后问题回溯模拟图

如图 5.2(b)所示，第二个皇后放置于(2-4)，则第三个皇后可放置于(3-2)，此时第四个皇后所有位置均不可放置，返回上一步。

如图 5.2(c)所示，第一个皇后放置于(1-2)，则第二、三、四个皇后可顺利放置于(2-4)、(3-1)、(4-3)中，得到一个可行解。

同理，如图 5.2(d)所示，第一个皇后放置于(1-3)，则第二、三、四个皇后可顺利放置于(2-1)、(3-4)、(4-2)中，得到第二个可行解。

值得注意的是，问题的求解并不需要用 $n \times n$ 的数组来表示结果，而只需要一个 n 长度的数组来表示每一行的皇后位置即可，即 arr[i]=k，表示第 i 行的皇后位置为 k。此时使用一个 arr[n]的数组就可以表示一个解，通过回溯可以使我们得到所有可行解。

n 皇后问题的回溯法算法如下所示。

```
//输入：n,表示皇后个数
//输出：n 皇后问题的所有可行解
1. 初始化解向量 arr[i]={-1}
2. i=1
3. while(i>=1)
3.1 把皇后 i 摆放在下一列，即 arr[i]++
3.2 从 arr[i]位置开始依次考察每一列，如果皇后 i 摆放在 arr[i]位置不发生冲突，则转 3.3
    步骤；否则 arr[i]++试探下一列
3.3 若 n 个皇后已全部摆放，则输出一个解，算法结束
3.4 若尚有皇后未摆放，则转步骤 3 摆放下一个皇后
3.5 若 arr[i]出界，则回溯，即 arr[i]=-1,i--,转步骤 3 重新摆放皇后 i
4. 退出循环，说明 n 皇后问题无解
```

5.3.2　0-1 背包问题


0-1 背包问题

例 5.2　有一个背包，最多能承载 10kg，现在有 3 个物品，质量分别为[4,8,5]、价值分别为[24,40,20]，详情如表 5.1 所示。那么应该如何选择才能使得你的背包背走最多价值的物品？

表 5.1　0-1 背包问题示例

物品	质量(w)/kg	价值(v)/元	价值/质量(v/w)
1	4	24	6
2	8	40	5
3	5	20	4

0-1 背包问题属于最优解问题，用回溯法需要构造解的子集树。对于每一个物品 i，该物品只有选与不选 2 个决策，总共有 n 个物品，可以顺序依次考虑每个物品，这样就形成了一棵解空间树。求解的基本思想就是遍历这棵树，以枚举所有情况，最后进行判断，如果质量不超过背包容量，且价值最大的话，该方案就是最后的答案。

0-1 背包问题的回溯法解空间树如图 5.3 所示。

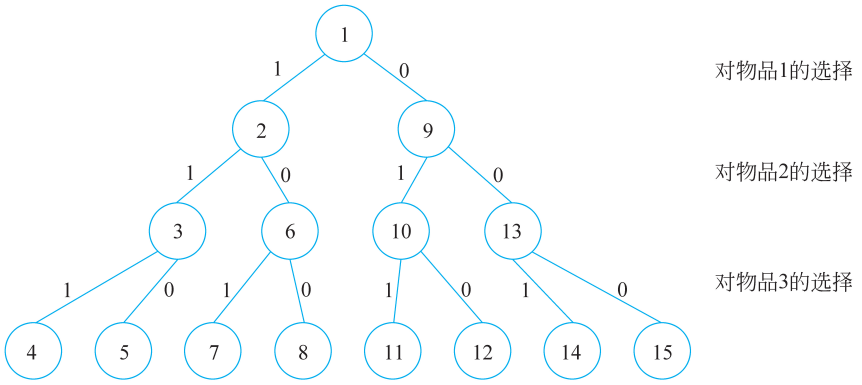


图 5.3 0-1 背包问题的回溯法解空间树

0-1 背包问题的回溯法求解过程如下。

(1) 选择物品 1,此时背包的质量为 4,价值为 24。

(2) 选择物品 2,此时背包的质量为 12,超出背包容量,因此物品 2 不可选择,回溯至结点 2。

(3) 不选择物品 2,行至结点 6;选择物品 3,行至结点 7,此时背包的质量为 9,价值为 44,由于已达叶子结点,则得到可行解(1,0,1),即该解选择物品 1 与 3。

(4) 回溯至结点 6,不选择物品 3,则得到可行解(1,0,0),即该解选择物品 1,总价值为 24,小于先前最优解 44。

(5) 回溯至结点 1,不选择物品 1,行至结点 9。

(6) 选择物品 2,此时背包的质量为 8,价值为 40,行至结点 10。

(7) 选择物品 3,行至结点 11,超出背包容量,回溯至结点 10。

(8) 不选择物品 3,行至结点 12,则得到可行解(0,1,0),即该解选择物品 2,总价值为 40,小于先前最优解 44。

(9) 回溯至结点 9,不选择物品 2,行至结点 13。

(10) 选择物品 3,此时背包的质量为 5,价值为 20,行至结点 14,则得到可行解(0,0,1),即该解选择物品 3,总价值为 20,小于先前最优解 44。

(11) 回溯至结点 13,不选择物品 3,行至结点 15,则得到可行解(0,0,0),即该解不选择任何物品,总价值为 0,小于先前最优解 44。

(12) 解空间树遍历完毕,输出最优解(1,0,1),总价值为 44。

0-1 背包问题的回溯法算法如下所示。

```
//输入: 背包的最大容量 C,物品个数 n,每一个物品的质量 wi,价值 vi
//输出: 问题的最优解选择的物品 x0[],总价值 max
rv=0;rw=0;                    //未考虑的物品总价值与总质量初始化
knapsack(k,r,cv,rc,rw)        //k 为解空间树的层数,r 为背包当前剩余容量,cv 为当前结
                              //点已装入背包物品的总价值
```

```
    if r>=0 and cv>max then                                    //找到并更新最优解
      max = cv; x0[] = x[1..k]; x0[k+1..n] = 0;
    end if
    if rw<=r then                                              //剪枝
      if cv+rv>max then
        max=cv+rv; x0[] = x[1..k]; x0[k+1..n] = 0;
      end if
    else
      if r>0 and cv+rv>max then
        rv=rv-v[k+1]; rw=rw-w[k+1];
        x[k+1]=0;
        knapsack(k+1,r,cv,rc,rw);                              //搜索左子树
        x[k+1]=1;
        knapsack(k+1,r-w[k+1],cv+v[k+1];,rc,rw);               //搜索右子树
      end if
    end if
  end knapsack
```

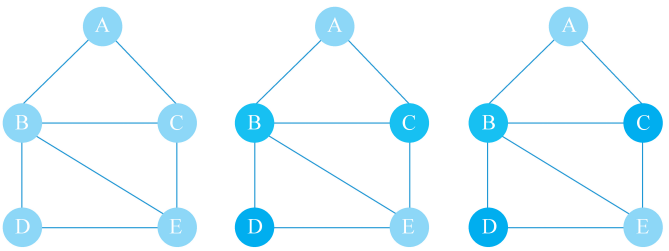
5.3.3　图的 m 着色问题

图的 m 着色问题

例 5.3　给定无向连通图 $G=(V,E)$和正整数 m，求最小的整数 m，使得用 m 种颜色对 G 中的顶点着色后任意两个相邻顶点着色不同。

用 m 种颜色为无向图 $G=(V,E)$着色，其中，V 的顶点个数为 n，可以用一个 n 元组 $C=(c_1,c_2,\cdots,c_n)$来描述图的一种可能着色，其中，$c_i\in\{1,2,\cdots,m\}(1\leqslant i\leqslant n)$表示赋予顶点 i 的颜色。

如图 5.4(a)所示，有 A、B、C、D、E 共 5 个顶点组成的无向图 G，如果采用图 5.4(b)的着色方法，可以看到相邻顶点 B、C 着色相同，因此该解为错误解；如果采用图 5.4(c)的着色方法，则满足题目条件，因此该解为可行解。



(a)　(b)　(c)

图 5.4　图的 m 着色问题示例

以图 5.4(a)为例，图的 m 着色问题的解空间树搜索过程如下。

(1) A=1，即顶点 A 着色 1。

(2) B=1，即顶点 B 着色 1，而 A、B 两顶点相邻，不满足题意，因此进行回溯。

(3) B=2,即顶点 B 着色 2。

(4) C=1,即顶点 C 着色 1,而 A、C 两顶点相邻,不满足题意,因此进行回溯。

(5) C=2,即顶点 C 着色 2,而 B、C 两顶点相邻,不满足题意,因此进行回溯。

(6) C=3,即顶点 C 着色 3。

(7) D=1,即顶点 D 着色 1。

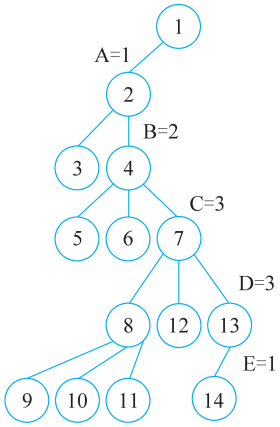


图 5.5 图的 m 着色问题解空间树搜索过程

(8) E=1,即顶点 E 着色 1,而 D、E 两顶点相邻,不满足题意,因此进行回溯。

(9) E=2,即顶点 E 着色 2,而 B、E 两顶点相邻,不满足题意,因此进行回溯。

(10) E=3,即顶点 E 着色 3,而 C、E 两顶点相邻,不满足题意,因此进行回溯。

(11) E=3,即顶点 E 着色 3,而 C、E 两顶点相邻,不满足题意,因此进行回溯。

(12) D=2,即顶点 D 着色 2,而 B、D 两顶点相邻,不满足题意,因此进行回溯。

(13) D=3,即顶点 D 着色 3。

(14) E=1,即顶点 D 着色 1,此时所有顶点均完成着色,且满足题意,得到可行解 5 元组(1,2,3,3,1)。

图的 m 着色问题的解空间树搜索过程如图 5.5 所示。

图的 m 着色问题的回溯法算法如下所示。

```
//输入:无向图点与边之间的关系(邻接矩阵),m 种颜色
//输出:无向图各顶点的着色记录 color[n]
1 将数组 color[n]初始化为 0
2 k=0;        //第一个顶点
3 while(k>=0)
3.1 依次考察每一种颜色,若顶点 k 的着色与其他顶点的着色不发生冲突,则转步骤 3.2;否则,
    搜索下一个颜色
3.2 若顶点已全部着色,则输出数组 color[n],返回
3.3 否则
3.3.1 若顶点 k 是合法着色,则 k=k+1,转步骤 3 处理下一顶点
3.3.2 否则,重置顶点 k 的着色情况,k=k-1,转步骤 3 回溯
```

5.3.4 批处理作业调度问题

例 5.4 设有 n 个作业{1,2,…,n}要在两台机器上处理,每个作业要先在机器 1 上处理,再在机器 2 上处理,请给出一种作业调度方案,使得所有任务完成的总时间最短。

对于这个问题,不难想到一个最优调度应使机器 1 没有空闲时间,且机器 2 的空闲时间最小。

我们给出一个实例,现有 3 个作业{1,2,3},在机器 1 上所需的处理时间为(2,3,2),

在机器 2 上所需的处理时间为(1,1,3)。如果以图 5.6 的调度方案,即作业处理顺序为 1、2、3,则完成时间为 10;如果以图 5.7 的调度方案,即作业处理顺序为 1、3、2,则完成时间为 8。调度方案与处理过程描述如下。

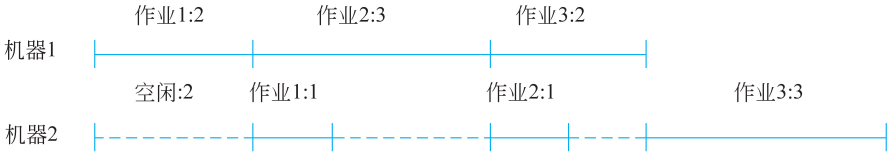


图 5.6　批处理作业调度方案一

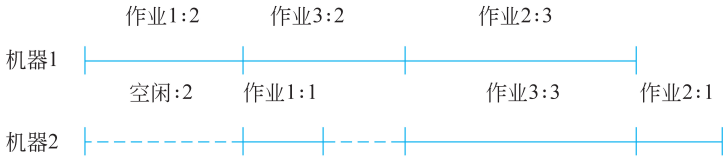


图 5.7　批处理作业调度方案二

1. 调度方案一

(1) 机器 1 处理作业 1,时间为 0～2,此时机器 2 空闲。

(2) 机器 1 处理作业 2,时间为 2～5;机器 2 处理作业 1,时间为 2～3,到时间 3 时,机器 2 空闲。

(3) 机器 1 处理作业 3,时间为 5～7;机器 2 处理作业 2,时间为 5～6,到时间 7 时,机器 2 空闲。

(4) 机器 2 作业 3,时间为 7～10,完成全部作业。

2. 调度方案二

(1) 机器 1 处理作业 1,时间为 0～2,此时机器 2 空闲。

(2) 机器 1 处理作业 3,时间为 2～4;机器 2 处理作业 1,时间为 2～3,到时间 3 时,机器 2 空闲。

(3) 机器 1 处理作业 2,时间为 4～7;机器 2 处理作业 3,时间为 4～7,机器 2 无空闲。

(4) 机器 2 处理作业 2,时间为 7～8,完成全部作业。

由上述两种调度方案可以看出,方案一中机器 2 的总空闲时间为 5,方案二中机器 2 的总空闲时间为 3,因此方案二比方案一节省了时间 2,验证了我们之前的设想。

为了求解该问题,我们进行如下假设。

$x[n]$:表示 n 个作业批处理的调度方案。

$x[k]$:表示第 k 个作业的编号。

sum1$[n]$:机器 1 当前完成时间。

sum2$[n]$:机器 2 当前完成时间。

sum1$[k]$:安排第 k 个作业后,机器 1 完成时间。

sum2[k]：安排第 k 个作业后，机器 2 完成时间。

则：

sum1[k]＝sum1[$k-1$]＋作业 x[k]在机器 1 的处理时间。

sum2[k]＝max{sum1[k],sum2[$k-1$]}＋作业 x[k]在机器 2 的处理时间。

批处理作业调度问题的回溯算法可表示如下。

```
//输入：n 个作业在机器 1 上的处理时间 a[n],在机器 2 上的处理时间 b[n]
//输出：最优调度序列 x[n]
1.初始化解向量 x[n]={-1};最短完成时间 bestTime=MAX;
2.初始化调度方案中机器 1 和机器 2 的完成时间
    sum1[n]=sum2[n]={0}; k=1
3. while(k>=1)
3.1 依次考察每一个作业，如果作业 x[k]尚未处理，则转步骤 3.2，否则尝试下一个作业，即
    x[k]++
    3.2 处理作业 x[k]:
        3.2.1 sum1[k]=sum1[k]+a[x[k]]
        3.2.2 sum2[k]=max{sum1[k],sum2[k-1]}+b[x[k]]
        3.2.3 若 sum2[k]<bestTime,则转步骤 3.3,否则实施剪枝
    3.3 若 n 个作业已全部处理，则输出一个解；
    3.4 若尚有作业没被处理，则 k++,转步骤 3 处理下一个作业
    3.5 回溯,x[k]= -1, k--, 转步骤 3 重新处理第 k 个作业
```

5.4 能力拓展

5.4.1 全排列问题

例 5.5 规定 n 的全排列是自然数 1 到 n 的所有不重复排列。现在给出 n，要求输出 n 的全排列。

问题分析：首先考虑规模较小情况，当 n＝3 时，全排列为{123,132,213, 231,312, 321}。由于问题要求输出所有不重复的情况，因此很容易就想到可以采用枚举的方法输出所有的情况。

题目明确要求将 n 个数放置在 n 个位置上，所以可以枚举 n 个位置，在每个位置上填入 n 个数放到当前位置上，同时还需要保证不跟之前已填入的数重复，所以在求解过程中还要记录前面已经填入的数来确保没有重复情况。当将 n 个数放完之后，就可以输出一个结果，然后向上回溯遍历其他情况。

以 n＝3 为例，回溯法的解决思路如下(见图 5.8)。

(1) 在位置 1 中填入数字 1，位置 2 中填入数字 2，位置 3 中填入数字 3，此时已填入 n 个数字，到达叶子结点，可输出结果 123。

(2) 回溯至位置 2，在位置 2 中填入数字 3，位置 3 中填入数字 2，此时已填入 n 个数字，可输出结果 132。

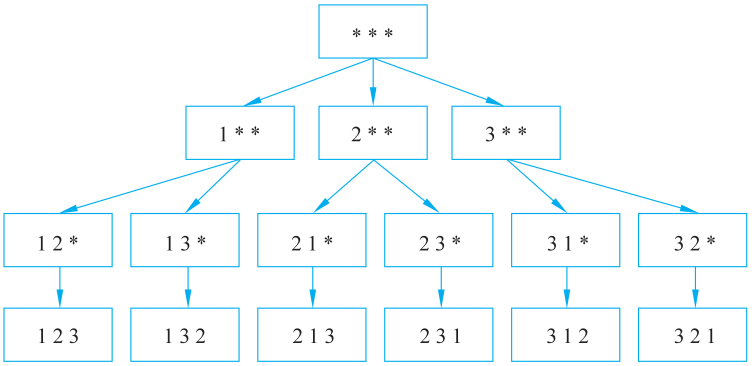


图 5.8　回溯法求解全排列问题示意图

（3）回溯至位置 1，在位置 1 中填入数字 2，位置 2 中填入数字 1，位置 3 中填入数字 3，此时已填入 n 个数字，可输出结果 213。

（4）回溯至位置 2，在位置 2 中填入数字 3，位置 3 中填入数字 1，此时已填入 n 个数字，可输出结果 231。

（5）回溯至位置 1，在位置 1 中填入数字 3，位置 2 中填入数字 1，位置 3 中填入数字 2，此时已填入 n 个数字，可输出结果 312。

（6）回溯至位置 2，在位置 2 中填入数字 2，位置 3 中填入数字 1，此时已填入 n 个数字，可输出结果 321。

5.4.2　存在障碍物的迷宫问题

例 5.6　给定一个 $n \times m$ 的迷宫，迷宫中有些格子存在障碍物，无法移动到这个格子里面，其余格子均为空。求从起点到终点至少需要多少步。规定每次移动只能水平或垂直移动。

输入描述：第一行输入两个整数 n、m，表示迷宫的行数和列数。

接下来 n 行，每行输入 m 个字符，表示整个迷宫。

空的格子用“.”表示，有障碍物的格子用 # 表示。

起点用 S 表示，终点用 T 表示。

首先利用图 5.9 所示问题来进行分析。位置 S 表示迷宫起点，位置 T 表示迷宫终点，# 表示该位置有障碍物，“.”表示该位置可到达。

S	#	#	.
.	#	.	.
.	.	.	.
.	#	.	T
.	#	.	.

图 5.9　存在障碍物的迷宫问题示例

由于起点和终点位置未知，因此需要先遍历找出起点 S 和终点 T 的位置。根据题目要求，我们只能水平或垂直移动，所以在移动的过程中只有上、下、左、右 4 个方向。每次移动时枚举这 4 种情况，并判断是否会出界。因为要求找到从起点 S 到终点 T 的最短距离，所以还需要对当前所在位置的步数进行标记，每次移动完毕后判断此时走到该点的步数是否小于上次途径该点的频数，不小于则向上回溯，小于则继续向下走。同时，当走

到一个无法再移动的点时也需要向上回溯(表明迷宫中此处不通)。如果移动至终点,则更新最短路径结果,并向上回溯其他的情况。

以图 5.9 为例,回溯法的求解思路如下。

(1) 遍历整个二维数组(迷宫中所有位置),找出起点 S 和终点 T。

(2) 从起点 S(1,1)出发,假设以上左下右的优先方式进行移动,结合不能越界、不能通过障碍物的约束条件,向下依次经过(2,1)、(3,1)、(4,1)、(5,1),无法再移动,回溯至(4,1),同样无法再移动,回溯至(3,1)。

(3) 从位置(3,1)向右移动至(3,2)、(3,3),并依次向上移动至(2,3),向右移动至(2,4),向上移动至(1,4),此时无法再移动,回溯至(2,4),并向下通过(3,4)、再向下到达终点(4,4),此时路径长度结果为 8。

(4) 回溯至位置(3,3),以上左下右的优先方式依次经过(4,3)、(5,3)、(5,4)并到达终点(4,4),此时路径长度结果为 8。

(5) 回溯至位置(4,3),并直接向右到达终点(4,4),此时路径长度结果为 6,更新最短路径结果。

(6) 再次回溯至位置(3,3),并经过(3,4)到达终点(4,4),此时路径长度结果也为 6。

5.4.3 图的 *m* 着色问题变种

例 5.7 我们规定字母表示一个电台,数字则表示广播电台的无线频谱。如果相邻两个电台之间运用同样的无线频谱就会互相干扰。所以只有所有相邻电台运用的都是不同的无线频谱,所有电台才能都正常运作。最少需要多少个无线频谱才能让所有电台都正常运作?

本题跟图的 *m* 着色问题类似,但是该题要求的是至少需要多少种颜色才能让每个相邻点之间颜色不相同(最少需要多少种无线频谱才能使相邻的电台互不干扰)。

以图 5.10 为例,电台 A 与电台 B、C 相邻,电台 B 与电台 A、C、D 相邻,电台 C 与电台 A、B、D 相邻,电台 D 与电台 B、C 相邻。

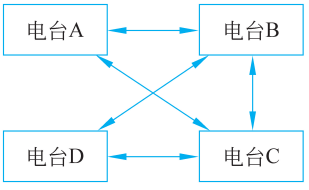


图 5.10 图的 *m* 着色问题变种示例

回溯法的求解思路如下。

(1) 对 A 点(电台 A)进行操作,因为至少需要一种无线频谱,所以 sum=1,此时 A 点相邻的 B、C 两点均未给定无线频谱,因此遍历 B 点。

(2) 将无线频谱 1 给予 B 点,发现存在 A 点与它相邻并且无线频谱相同。所以 B 点不能使用无线频谱 1,我们需要加入一个新的无线频谱才能让 B 点拥有无线频谱并且满足条件。因此,给 B 点无线频谱 2,sum++,此时其他相邻点的无线频谱均与其不同。

(3) 同上将无线频谱 1 给予 C 点,发现存在相邻点 A 与之干扰,将无线频谱 2 给予 C 点,同时发现存在相邻点 B 与之干扰,所以加入一个新的无线频谱 3 给予 C 点,sum++,此时其他相邻点的无线频谱均与其不同。

(4) 此时我们发现对于 D 点来说,使用无线频谱 1 即可满足所有约束条件。

(5) 完成所有电台的无线频谱安排,输出结果“最少需要 3 种无线频谱”。