1、概念

回溯算法实际上一个类似枚举的搜索尝试过程，主要是在搜索尝试过程中寻找问题的解，**当发现已不满足求解条件时，就“回溯”返回，尝试别的路径。**

回溯法是一种选优搜索法，**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走的技术为回溯法**，而满足回溯条件的某个状态的点称为“回溯点”。

许多复杂的，规模较大的问题都可以使用回溯法，有“通用解题方法”的美称。

2、基本思想

在包含问题的所有解的解空间树中，按照**深度优先搜索的策略**，从根结点出发深度探索解空间树。当探索到某一结点时，要先判断该结点是否包含问题的解，如果包含，就从该结点出发继续探索下去，如果该结点不包含问题的解，则逐层向其祖先结点回溯。（其实回溯法就是对隐式图的深度优先搜索算法）。

若用回溯法求问题的所有解时，要回溯到根，且根结点的所有可行的子树都要已被搜索遍才结束。

而若使用回溯法求任一个解时，只要搜索到问题的一个解就可以结束。

3、用回溯法解题的一般步骤：

（1）针对所给问题，确定问题的解空间：

首先应明确定义问题的解空间，问题的解空间应至少包含问题的一个（最优）解。

（2）确定结点的扩展搜索规则

（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。

4、算法框架

（1）问题框架

设问题的解是一个n维向量(a1,a2,……,an),约束条件是ai(i=1,2,3,…..,n)之间满足某种条件，记为f(ai)。

（2）非递归回溯框架

```
1: int a[n],i;
2: 初始化数组a[];
3: i = 1;
4: while (i>0(有路可走) and (未达到目标)) // 还未回溯到头
5: {
6: if(i > n) // 搜索到叶结点
7: {
8: 搜索到一个解，输出;
9: }
10: else // 处理第i个元素
11: {
12: a[i]第一个可能的值;
13: while(a[i]在不满足约束条件且在搜索空间内)
14: {
```

```
15: a[i]下一个可能的值;
16: }
17: if(a[i]在搜索空间内)
18: {
19: 标识占用的资源;
20: i = i+1; // 扩展下一个结点
21: }
22: else
23: {
24: 清理所占的状态空间;  // 回溯
25: i = i -1;
26: }
27: }
```

（3）递归的算法框架

回溯法是对解空间的深度优先搜索，在一般情况下使用递归函数来实现回溯法比较简单，其中i为搜索的深度，框架如下：

```
1: int a[n];
2: try(int i)
3: {
4: if(i>n)
5: 输出结果;
6: else
7: {
8: for(j = 下界; j <= 上界; j=j+1) // 枚举i所有可能的路径
9: {
10: if(fun(j)) // 满足限界函数和约束条件
11: {
12: a[i] = j;
13: ... // 其他操作
14: try(i+1);
15: 回溯前的清理工作（如a[i]置空值等）;
16: }
17: }
18: }
19: }
```