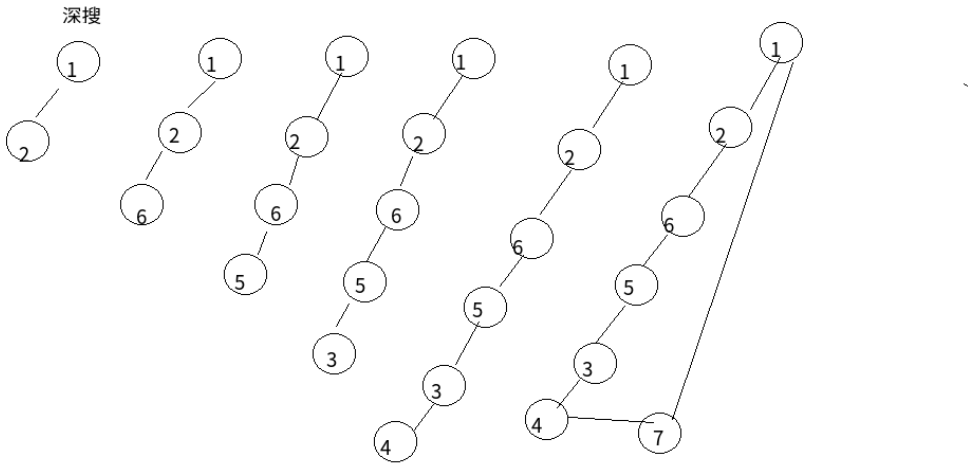
1)

深搜：维护一个栈，栈中的元素是节点的编号队列，该序号队列的特点是不允许出现重复节点编号。

while（栈不空）{

操作是，每次出栈，将得到一个编号队列，以该编号队列的最后一个元素进行拓展，查看该元素所有邻接节点编号，对于每个邻接编号与当前的编号队列进行判重，如果该编号不在当前队列中则将其加入当前队列，成为一个新的堆栈元素，然后入栈。否则，判断编号队列的长度，如果等于 N 则表示找到了，然后退出循环即可，再否则什么都不做。

}

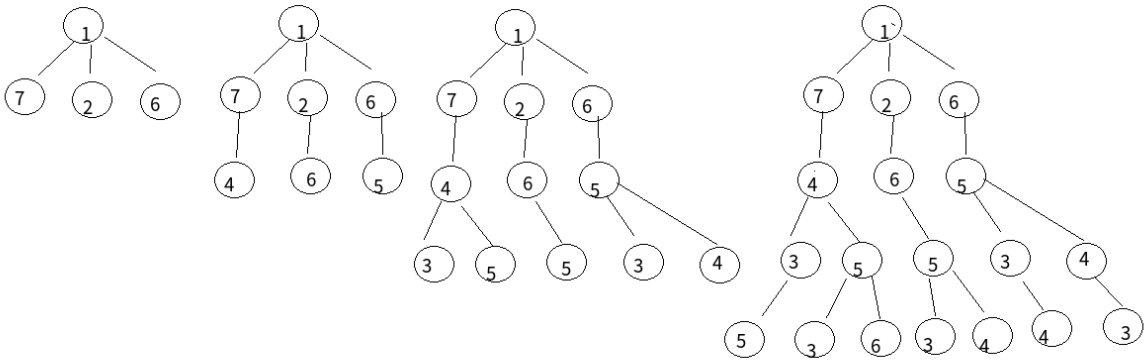


广搜：维护一个队列，队列中的元素是节点的编号队列，该序号队列的特点是不允许出现重复节点编号。

while（队列不空）{

操作是，每次出队，将得到一个编号队列，以该编号队列的最后一个元素进行拓展，查看该元素所有邻接节点编号，对于每个邻接编号与当前的编号队列进行判重，如果该编号不在当前队列中则将其加入当前队列，成为一个新的堆栈元素，然后入队。否则，判断编号队列的长度，如果等于 N 则表示找到了，然后退出循环即可，再否则什么都不做。}

广搜



2）

预处理：对 9 个格子编号，第一行，0,1,2;第二行，3,4,5;第三行，6,7,8

利用求和公式 sum(x) = xi * 9 ^ i，i 是格子的编号，xi 是该格子对应的数值。

借由这个 sum 和表示状态。

深搜： 创建栈 A，将起始格局存入栈 A 中，创建队列 B（或者 hash 表）；

while（栈不为空）

{

A 栈顶出栈，计算该出栈格局的 sum 值，存入队列 B（或者 hash 表）；

如果该 sum 值等于目标格局的 sum 值，直接结束;

考虑该状态下所有可能移动的格局，对于每一个格局计算 sum 值，并在 B（或者 hash 表）中进行判定是否存在。如果存在则放弃该格局，如果不存在，则将该格局入栈。

}

广搜：创建队列 A，将起始格局存入队列 A 中，创建队列 B（或者 hash 表）；

while（队列不为空）

{

A 队列出队，计算该出队格局的 sum 值，存入队列 B（或者 hash 表）；

如果该 sum 值等于目标格局的 sum 值，直接结束;

考虑该状态下所有可能移动的格局，对于每一个格局计算 sum 值，并在 B（或者 hash 表）中进行判定是否存在。如果存在则放弃该格局，如果不存在，则将该格局入队。

}

爬山法：

预处理：

设定启发函数 h（x）{1,设定 h（x）= 当前格局中与目标格局一致的数字的数目，2,设定 h（x）= 当前格局中所有数字移动到应该抵达位置所需步数的和的相反值}

启发函数设定成两种中的任意一种皆可。

算法：

创建队列 B（或者 hash 表）

当前节点 = 起始格局;

while（当前节点不为空）{

考虑当前格局所能达成的所有格局，计算每个所能达成格局的 sum 值，在 B 中查看是否已经存在，对于已经存在的则直接抛弃;

对于留下的所有格局考虑启发函数值。取其中最大的那个作为新的节点值。继续循环。

}

最佳优先方法：

预处理：

设定启发函数 h（x）{1,设定 h（x）= 当前格局中与目标格局一致的数字的数目，2,设定 h（x）= 当前格局中所有数字移动到应该抵达位置所需步数的和的相反值}

创建最大堆 A，创建队列 B（或者 hash 表）

while（堆不为空）

{

A 堆顶出堆，计算该出队格局的 sum 值，存入队列 B（或者 hash 表）；

如果该 sum 值等于目标格局的 sum 值，直接结束;

考虑该状态下所有可能移动的格局，对于每一个格局计算 sum 值，并在 B（或者 hash 表）中进行判定是否存在。如果存在则放弃该格局，如果不存在，则将该格局入堆。

利用每个格局的启发函数 h（x）来维护最大堆。

}

3)

预处理：我们对序列排个序，这样就能进行深搜的剪枝。

深搜：

对于集合中的每一个元素，都有两种选择，选或者不选，所以所有的集合将构成一个完全二叉树。

找出满足要求的集合就只需要我们遍历这棵树就行，好处在于我们不用每次都遍历到叶子节点，我们只需要遍历到某一节点之后，发现满足要求，则直接就可以返回找到的结果。

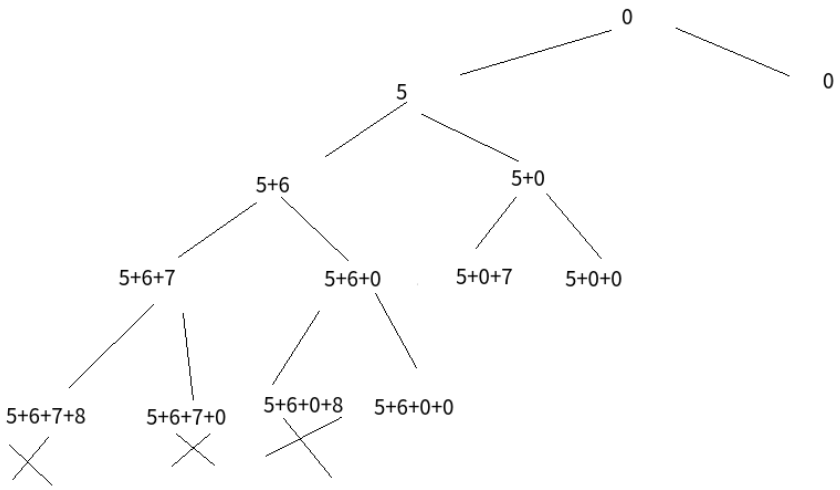
算法：

建立二叉树，第 1 层的根节点随便都行，第 i 层的左节点表示选择第 i 个元素，第 i 层的右节点表示不选择第 i 个元素加入集合，然后深搜先序遍历该树就行，剪枝策略：对于当前节点，如果该节点的左节点和已经大于目标值 K，则无需再往下继续拓展分支。找到符合结果的集合就存储下该集合。

以实例举例：

先排序：

5　6　7　8　13　21



例子太麻烦了，图已经形象地体现了剪枝策略。

分支界限法

思想与深搜不同，不再采用二叉树，采用多叉树，第 i + 1 层表示共选择了 i 个元素作为子集。

每个节点不仅存当前集合的和 sum，也要存一个元素，该元素是选择集合中，按总队列顺序的最后一个元素。设当前节点的记录元素是 X，总队列是 S ，同样还要存下界就是 sum+所有在 X 之后值是负数元素的和。存上界是 sum+所有在 X 之后值是正数元素的和。

扩展分支节点的步骤：凡是当前节点的下界 > K 或者，上界 <K，则不需要继续扩展，否则，对 X 元素之后的所有元素都要添加进集合一次（添加一次后还要取出来），然后计算每次添加进集合后的 sum，记录当前添加的元素 Y，计算新的上下界，然后成为新的儿子节点。

遍历：最后层序遍历该树，找到符合要求的集合则记录下来。

4）

预处理：

对 9 个格子编号，第一行，0,1,2;第二行，3,4,5;第三行，6,7,8
利用求和公式 sum(x) = xi * 9 ^ i，i 是格子的编号，xi 是该格子对应的数值。借由这个 sum 和表示状态。

设定启发函数 h（x）{h（x）= 当前格局中所有数字移动到应该抵达位置所需步数的和}

算法：

创建最小堆 A（维护时通过每个节点当前的已用步骤 step+启发函数 h(x)的和来维护），创建队列 B（或者 hash 表）

while（堆不为空）{

A 堆顶出堆，计算该出队格局的 sum 值，存入队列 B（或者 hash 表）；

如果该 sum 值等于目标格局的 sum 值，直接结束;

考虑该状态下所有可能移动的格局，对于每一个格局计算 sum 值，并在 B（或者 hash 表）中进行判定是否存在。如果存在则放弃该格局，如果不存在，则将该格局的 step + 1,然后入堆，。

利用每个格局的（启发函数 h（x）+ 达到该格局已用步数 step）来维护最小堆。

}