搜索有以下几种算法：

枚举算法：

也即列举问题的所有状态从而寻找符合问题的解的方法。

适合用于状态较少，比较简单的问题上。

广度优先搜索：

从初始点开始，根据规则展开第一层节点，并检查目标节点是否在这些节点上，若没有，再将所有的第一层的节点逐一展开，得到第二层节点，如没有，则扩展下去，直到发现目标节点为止。

比较适合求最少步骤或最短解序列的题目。

一般设置一个队列queue，将起始节点放入队列中，然后从队列头取出一个节点，检查是否是目标节点，如不是则进行扩展，将扩展出的所有节点放到队尾，然后再从队列头取出一个节点，直至找到目标节点。

深度优先搜索：

一般设置一个栈stack，将起始节点放入栈中，然后从栈中弹出一个节点，检查是否是目标节点，如不是则进行扩展，将扩展出的所有节点入栈，然后再从栈顶弹出一个节点，直到找到目标节点。

深度优先搜索得到的第一个解，不一定是最优解。

双向广度优先搜索：

双向搜索：从起始节点向目标节点方向搜索，同时从目标节点向起始节点方向搜索。

双向搜索只能用于广度优先搜索中。

双向搜索扩展的节点数量要比单向少的多。

A\*算法

利用问题的规则和特点来制定一些启发规则，由此来改变节点的扩展顺序，将最有希望扩展出最优解的节点优先扩展，使得尽快找到最优解。

对每一个节点，有一个估价函数F来估算起始节点经过该节点到达目标节点的最佳路径的代价。

每个节点扩展的时候，总是选择具有最小的F的节点。

F=G+B×H：G为从起始节点到当前节点的实际代价，已经算出，H为从该节点到目标节点的最优路径的估计代价。F要单调递增。

B最好随着搜索深度成反比变化，在搜索深度浅的地方，主要让搜索依靠启发信息，尽快的逼近目标，而当搜索深的时候，逐渐变成广度优先搜索。

回溯算法：

和深度优先相似，不同之处在于对一个节点扩展的时候，并不将所有的子节点扩展出来，而只扩展其中的一个。因而具有盲目性，但内存占用少。

搜索中的优化：

在搜索前，根据条件降低搜索规模。

广度优先搜索中，被处理过的节点，充分释放空间。

给据问题的约束条件进行剪枝。

利用搜索过程中的中间解，避免重复计算。

一、马的走法

题目：

在一个4X5的棋盘上，马的起始坐标由用户输入，求马返回初始位置的所有不同的走法的总数和步骤。马走过的位置不能重复。