# 拼图算法

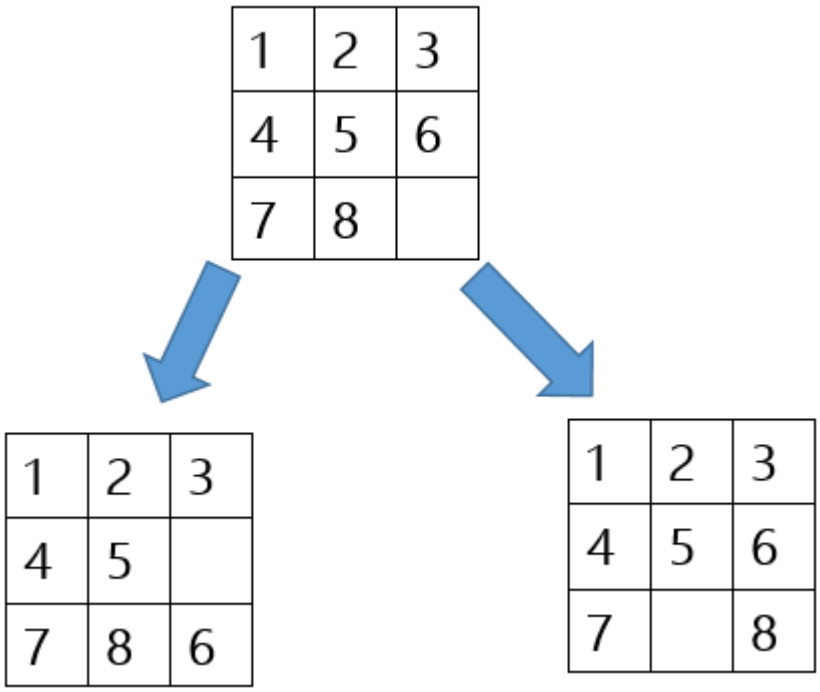
## 问题描述

将一个拼图从原始状态拼到指定状态，一次只能移动一个格子，求最短移动次数，如下图所示：


> **输入：**
>
> 起始状态： [1, 2, 3, 4, 5, 6, 7, 8, 0]
>
> 目标状态：[1, 2, 3, 0, 4, 5, 6, 7, 8]
>
> **输出：**
>
> 14

## 问题分析

1. 每一步的移动都是固定的，但是每一步之后会有上，下，左，右多个选择，如下图所示。因此可以构建一个树的结构，树的节点每一步移动的结果。在这么节点中，肯定会存在一个或多个节点是和目标是匹配的。




2. 每一个分支的数量非常大，并且不是每一个分支都能到目标节点，要排除深度遍历算法DFS。使用广度遍历算法BFS时，由于BFS的特性，因此遇到的第一个目标值的时候，肯定是最短的路径。
3. 由于节点非常多，一共有 $9^9$个，所以不能一次性初始化所有节点，必须在遍历父节点的过程时，将子节点加入到遍历队列中。

## 代码

```
import queue

def init_Graph(root, target):
    if root == target:
        return

    nodes = []

    q = queue.Queue()
    q.put((root, ''))
    while not q.empty():
        node = q.get()
        print(node[0])

        # 目标值
        if node[0] == target:
            print(node[1])
            print('一共需要' + str(node[1].count('\n')) + '步')
            break
        # 判断是否有重复点
        if node[0] in nodes:
            continue

        # 节点集合
```

```
            nodes.append(node[0])
            handle_child(q, node[0], node[1])


# 处理孩子节点
def handle_child(q, root, path):
    root_path = ''.join(str(e) for e in root)
    path = path + '\n' + root_path
    i = root.index(0)
    if i not in (0, 3, 6):
        temp_left = list(root)
        temp_left[i] = temp_left[i - 1]
        temp_left[i - 1] = 0
        q.put((temp_left, path))

    if i not in (6, 7, 8):
        temp_up = list(root)
        temp_up[i] = temp_up[i + 3]
        temp_up[i + 3] = 0
        q.put((temp_up, path))

    if i not in (2, 5, 8):
        temp_right = list(root)
        temp_right[i] = temp_right[i + 1]
        temp_right[i + 1] = 0
        q.put((temp_right, path))

    if i not in (0, 1, 2):
        temp_down = list(root)
        temp_down[i] = temp_down[i - 3]
        temp_down[i - 3] = 0
        q.put((temp_down, path))


if __name__ == '__main__':
    origin = [1, 2, 3, 4, 5, 6, 7, 8, 0]
    target = [1, 2, 3, 4, 0, 5, 6, 7, 8]
    init_Graph(origin, target)
```