

An implicit graph

- Suppose we represent each node in graph as a state of the puzzle:
 - Specific layout of tiles
 - Rather than generate all nodes, only generate as needed to explore paths
 - Edges then become implicit – create to generate new node

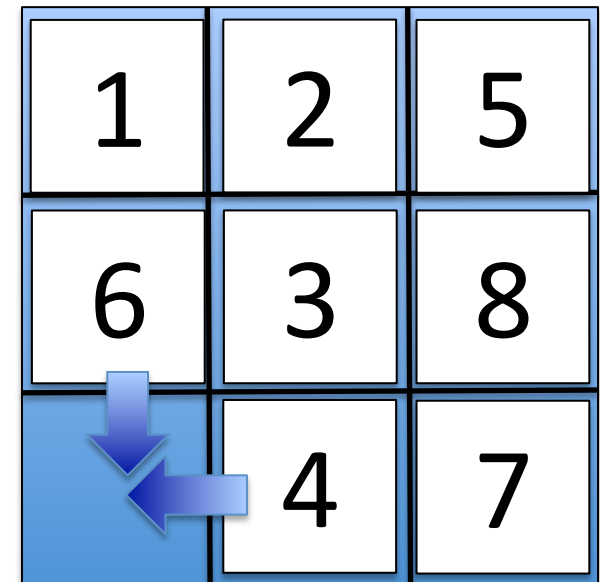
A node

- Convert sequence of tiles into a string of labels – use to represent a node
 - Example: 125638047
- Keep track of blank tile
 - In slot 6 in example
- Need a way to encode edges

1	2	5
6	3	8
	4	7

An “edge”

- Need to encode legal shifts
 - In example, tile in 3rd spot can shift down to 6th spot
 - And tile in 7th spot can shift left to 6th spot
- Could keep track of these legal shifts in a dictionary
 - Index by location in grid
 - Return legal new locations



An implementation: node

```
class puzzle(object):
    def __init__(self, order):
        self.label = order
        for index in range(9):
            if order[index] == '0':
                self.spot = index
        return None
    def transition(self, to):
        label = self.label
        blankLocation = self.spot
        newBlankLabel = str(label[to])
        newLabel = ''
        for i in range(9):
            if i == to:
                newLabel += '0'
            elif i == blankLocation:
                newLabel += newBlankLabel
            else:
                newLabel += str(label[i])
        return puzzle(newLabel)
    def __str__(self):
        return self.label
```

An implementation: implicit edges

```
shiftDict = {}  
shiftDict[0] = [1, 3]  
shiftDict[1] = [0, 2, 4]  
shiftDict[2] = [1, 5]  
shiftDict[3] = [0, 4, 6]  
shiftDict[4] = [1, 3, 5, 7]  
shiftDict[5] = [2, 4, 8]  
shiftDict[6] = [3, 7]  
shiftDict[7] = [4, 6, 8]  
shiftDict[8] = [5, 7]
```

	1	2
3	4	5
6	7	8



An implementation: search

```
def BFSWithGenerator(start, end, q = []):  
    initPath = [start]  
    q.append(initPath)  
    while len(q) != 0:  
        tmpPath = q.pop(0)  
        lastNode = tmpPath[len(tmpPath) - 1]  
        if lastNode.label == end.label:  
            return tmpPath  
        for shift in shiftDict[lastNode.spot]:  
            new = lastNode.transition(shift)  
            if notInPath(new, tmpPath):  
                newPath = tmpPath + [new]  
                q.append(newPath)  
    return None
```

Checking for loops

```
def notInPath(node, path):  
    for elt in path:  
        if node.label == elt.label:  
            return False  
    return True
```

An implementation: search

```
def DFSWithGenerator(start, end, stack = []):  
    initPath = [start]  
    stack.insert(0, initPath)   
    while len(stack) != 0:  
        tmpPath = stack.pop(0)  
        lastNode = tmpPath[len(tmpPath) - 1]  
        if lastNode.label == end.label:  
            return tmpPath  
        for shift in shiftDict[lastNode.spot]:  
            new = lastNode.transition(shift)  
            if notInPath(new, tmpPath): #avoid cycles  
                newPath = tmpPath + [new]  
                stack.insert(0, newPath)   
    return None
```


Searching for a puzzle solution

- Running the breadth first search finds a solution with 8 moves
 - Searches 220 nodes
- Running the depth first search takes a very long time!! (and runs out of memory on my machine!!)