Exercise 4.1

Give the name of the algorithm that results from each of the following special cases:

1. Local beam search with k=1.
2. Local beam search with one initial state and no limit on the number of states retained.
3. Simulated annealing with T=0 at all times (and omitting the termination test).
4. Simulated annealing with T= ∞ at all times.
5. Genetic algorithm with population size N=1.

1. **爬山法。**因为只选取 1 个最优的后继节点。

2. **BFS 广度优先搜索。**总是将每层的结点扩展后再扩展下一层结点。

3. **爬山法。**温度 T 始终为 0，选择更坏移动的概率始终为 0，故总是选取更优移动。

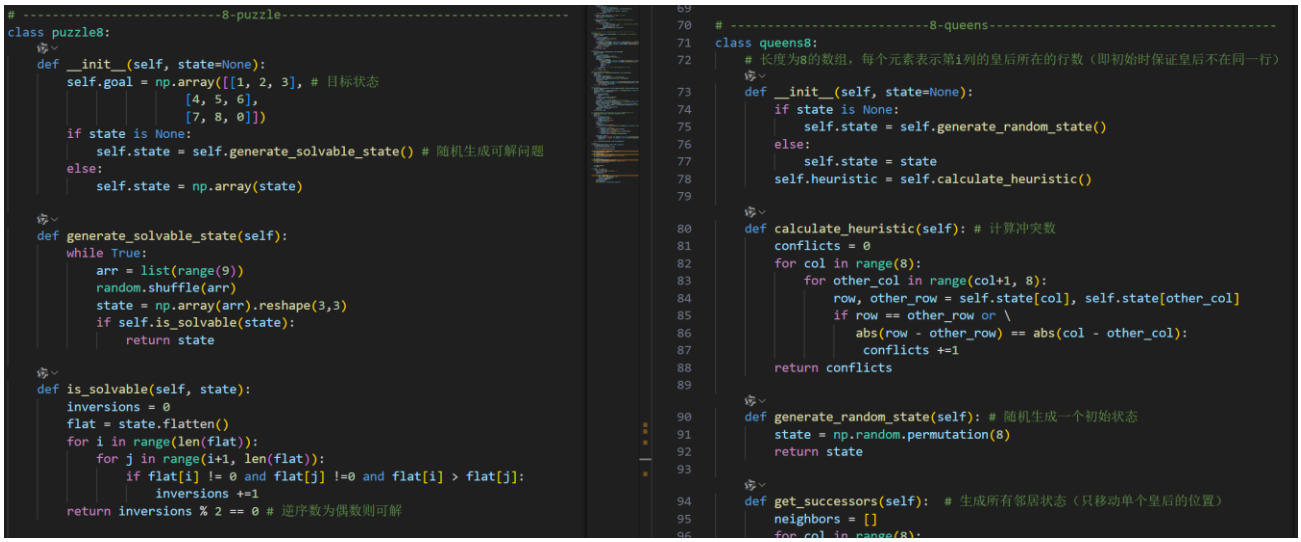
4. **随机游走。**T=∞时无条件接受所有移动，相当于随机移动。

5. **随机游走。**遗传算法个体数为 1，则交叉无法产生新解，仅能通过变异产生，相当于随机游走。

Exercise 4.4

+

问题生成器：



```python
# ----------8-puzzle----------
class puzzle8:
    def __init__(self, state=None):
        self.goal = np.array([[1, 2, 3], # 目标状态
                              [4, 5, 6],
                              [7, 8, 0]])
        if state is None:
            self.state = self.generate_solvable_state() # 随机生成可解问题
        else:
            self.state = np.array(state)

    def generate_solvable_state(self):
        while True:
            arr = list(range(9))
            random.shuffle(arr)
            state = np.array(arr).reshape(3,3)
            if self.is_solvable(state):
                return state

    def is_solvable(self, state):
        inversions = 0
        flat = state.flatten()
        for i in range(len(flat)):
            for j in range(i+1, len(flat)):
                if flat[i] != 0 and flat[j] !=0 and flat[i] > flat[j]:
                    inversions +=1
        return inversions % 2 == 0 # 逆序数为偶数则可解
```

```python
# ----------8-queens----------
class queens8:
    # 长度为8的数组，每个元素表示第i列的皇后所在的行数（即初始时保证皇后不在同一行）
    def __init__(self, state=None):
        if state is None:
            self.state = self.generate_random_state()
        else:
            self.state = state
        self.heuristic = self.calculate_heuristic()

    def calculate_heuristic(self): # 计算冲突数
        conflicts = 0
        for col in range(8):
            for other_col in range(col+1, 8):
                row, other_row = self.state[col], self.state[other_col]
                if row == other_row or \
                   abs(row - other_row) == abs(col - other_col):
                    conflicts +=1
        return conflicts

    def generate_random_state(self): # 随机生成一个初始状态
        state = np.random.permutation(8)
        return state

    def get_successors(self):  # 生成所有邻居状态（只移动单个皇后的位置）
        neighbors = []
```

最陡上升爬山法：

```python
# 最陡上升爬山法
def steepest_ascent(problem, maxsteps=1000): # 超过最大步数则停止，认为求解失败
    current = problem
    steps = 0
    for _ in range(maxsteps):
        neighbors = current.get_successors()
        if not neighbors:
            break # 没有邻居则停止
        best_neighbor = min(neighbors, key=lambda x: x.get_heuristic() if isinstance(x, puzzle8) else x.calculate_heuristic())
        if (isinstance(best_neighbor, puzzle8) and best_neighbor.get_heuristic() >= current.get_heuristic()) or \
           (isinstance(best_neighbor, queens8) and best_neighbor.calculate_heuristic() >= current.calculate_heuristic()):
            break  # 没有更优解则停止
        current = best_neighbor
        steps +=1
    return current, steps
```

首选爬山法：

```python
# 首选爬山法
def first_choice_hill_climbing(problem, max_steps=1000):
    current = problem
    steps = 0
    for _ in range(max_steps):
        neighbors = current.get_successors()
        random.shuffle(neighbors) # 随机打乱邻居顺序
        improved = False
        for neighbor in neighbors:
            if (isinstance(neighbor, puzzle8) and neighbor.get_heuristic() < current.get_heuristic()) or \
               (isinstance(neighbor, queens8) and neighbor.calculate_heuristic() < current.calculate_heuristic()):
                current = neighbor
                steps +=1
                improved = True
                break
        if not improved: # 在所有邻居中都没有找到更优解，则停止搜索
            break
    return current, steps
```

随机重启爬山法：

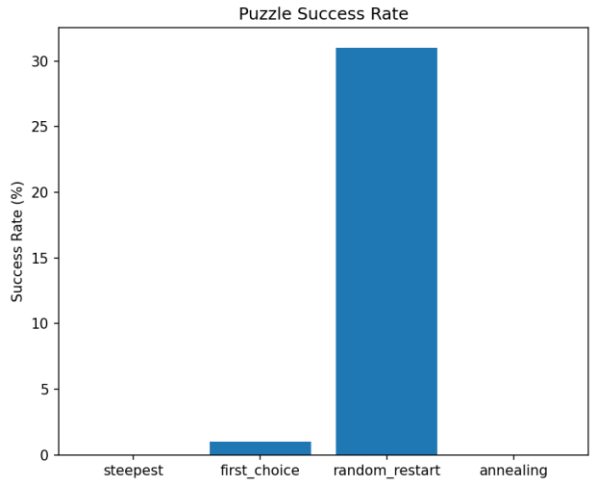
```python
# 重启爬山法（最大重启次数100）
def random_restart_hill_climbing(problem_generator, max_restarts=100):
    restarts = 0
    total_steps = 0
    while restarts < max_restarts:
        current = problem_generator() # 每次重启生成新问题
        solution, steps = steepest_ascent(current, maxsteps=1000)
        total_steps += steps
        restarts +=1
        if (isinstance(solution, puzzle8) and np.array_equal(solution.state, solution.goal)) or \
           (isinstance(solution, queens8) and solution.calculate_heuristic() == 0):
            return solution, total_steps
    return None, total_steps
```

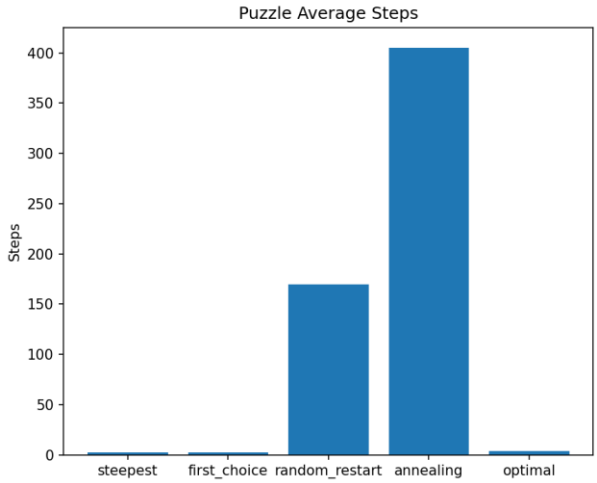
模拟退火：

```python
# 模拟退火算法
def simulated_annealing(problem, initial_temperature=1000, cooling_rate=0.95, max_steps=10000):
    current = problem
    current_heuristic = current.get_heuristic() if isinstance(current, puzzle8) else current.calculate_heuristic()
    temperature = initial_temperature
    steps = 0
    for _ in range(max_steps):
        if temperature <= 1e-6:
            break
        neighbors = current.get_successors()
        if not neighbors: # 没有邻居则停止
            break
        next_state = random.choice(neighbors) # 随机选择一个邻居
        next_heuristic = next_state.get_heuristic() if isinstance(next_state, puzzle8) else next_state.calculate_heuristic()
        delta_E = next_heuristic - current_heuristic
        delta_E = delta_E.item() if isinstance(delta_E, np.ndarray) else delta_E  # 提取单个元素
        if delta_E < 0 or random.random() < math.exp(-delta_E/temperature): # 接受更优解 或 以一定概率接受更差解
            current = next_state
            current_heuristic = next_heuristic
        temperature *= cooling_rate
        steps +=1
    return current, steps
```

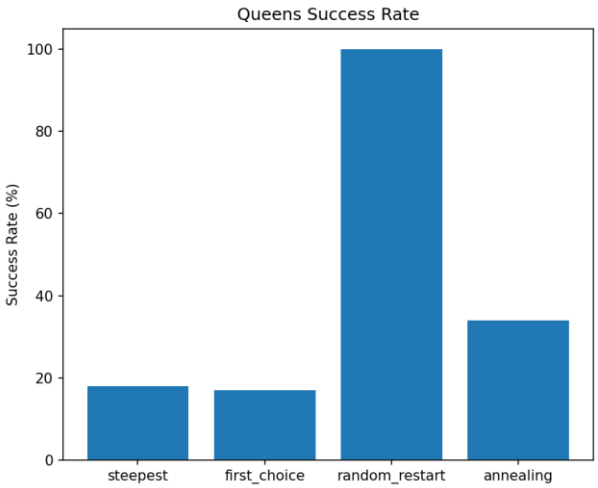
2 种问题 4 种算法各组合实验 100 次，得到以下结果：

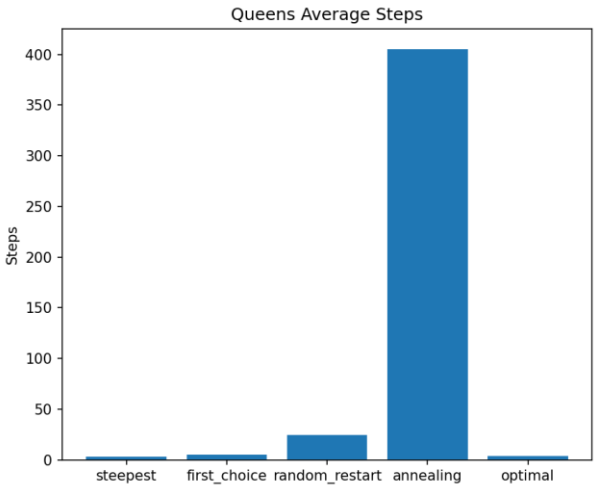
8 puzzle 问题的结果：





8 queens 问题的结果：





结果分析：

对于 8 puzzle 问题，

从成功率来看，最陡上升爬山法、首选爬山法、模拟退火算法均表现很差，几乎不能求出解。随机重启爬山法相比于其他 3 种算法表现较好的原因是：通过随机重启可能恰好达到目标状态，并非它在求解时能做出更高明的行动。至于模拟退火算法，理论上它可以找出解，但可能由于参数设置原因 (如温度不够高，启发式函数取值不合理等)，其几乎未能求解成功。

从平均代价（步数）分析，最陡上升爬山法、首选爬山法的平均步数极低，推测应该是在求解过程中很快落入了局部最优状态而停止，也解释了它们几乎不能求解 8 puzzle 问题的原因。而模拟退火算法步数很高的原因是其停止搜索的条件与其他几种算法不同，若没有添加其他停止条件，它会走完设置的最大步数才停止（为防止这种情况，代码种还添加了温度过低时也会停止)。

对于 8 queens 问题，

从成功率来看，随机重启爬山法的成功率达到了 100%，由于生成的 8 queens 有 8^8=16,777,216 种可能，而合法解仅有 92 种，所以可以排除随机性的原因，说明随机重启爬山法在解决 8 queens 问题上表现良好。模拟退火算法成功率较高于最陡上升爬山法、

首选爬山法，但这三者成功率都不太高。

从平均代价（步数）分析，最陡上升爬山法、首选爬山法的平均步数极低原因同上。模拟退火算法步数很高的原因同上。随机重启爬山法的步数相比于最优解还是较高。

综上，这四种算法都不适合于求解带有曼哈顿距离启发的 8puzzle 问题，因为很容易陷入局部最优状态。对于 8queens 问题，只有随机重启爬山法表现很好。