A*算法的实现

`A*算法`是一种启发式的算法，通过预估与目标点的距离选择最优路径

算法实现如下，`get_best`函数从`openset`中获取优先度最高，即估算距离最多的点

`find`函数通过传入的起点和终点找出最优路径

`get_neighboor`函数寻找指定点的相邻结点，并放入`openset`中

```
class A_star:
    def __init__(self, graph, h_cost):
        self.graph = graph
        self.opened = {}
        self.closed = set()
        self.path = []
        self.search_path = []
        self.h_cost = h_cost

    def HeuristicCost(self):
        pass

    def find(self, start, end):
        self.opened[start] = Node(start, None, 0, 0)
        while len(self.opened):
            best = self.get_best()
            self.search_path.append(best)
            #最优点为终点
            if best.name == end:
                self.print_path(best)
                return

            self.closed.add(best)
            self.opened.pop(best.name)

            self.get_neighboor(best)

    def get_neighboor(self, node):
        neighboors = self.graph[node.name]
        for key, value in neighboors.items():
            if key in self.closed:
                continue

            self.opened[key] = Node(key, node, value+node.g_cost,
self.h_cost[key])


    def get_best(self):
        best = None
        f_cost = sys.maxsize
        for key, value in self.opened.items():
            if value.f_cost < f_cost:
                best = value
                f_cost = value.f_cost
```

```
        return best

    def print_path(self, node):
        self.path.append(node)
        while node:
            self.path.append(node.parent)
            node = node.parent
        self.path.reverse()
        del self.path[0] #去掉None

        for p in self.path:
            print(p.name)

    def print_search_path(self):
        for p in self.search_path:
            print(p.name)


class Node:
    def __init__(self, name, parent, g_cost, h_cost):
        self.name = name
        self.parent = parent
        self.g_cost = g_cost
        self.h_cost = h_cost
        self.f_cost = g_cost + h_cost
```

Dijkstra和A*算法的对比

当`A*算法`中的`h(n)`都等于0时，`A*算法`实际退化为`Dijkstra算法`，因此利用同样的代码我们可以探寻`Dijkstra`和`A*算法`的异同

搜索的路径顺序

`A*算法`的搜索路径如下

```
Arad
Sibiu
Rimnicu
Fagaras
Pitesti
Bucharest
```

`Dijkstra算法`的搜索路径如下

```
Arad
Zerind
Timisoara
Sibiu
Rimnicu
Lugoj
Fagaras
Arad
Oradea
```

```
Mehadia
Pitesti
Zerind
Lugoj
Drobeta
Rimnicu
Bucharest
```

运行速率

`a_stat` 搜索用时为

```
27.5 µs ± 929 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

`Dijkstra` 搜索用时为

```
73.2 µs ± 1.5 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

个人理解

`A*算法` 作为一种启发式算法，利用 `h(n)` 这一信息达到了缩短搜索次数的目的，而 `Dijkstra` 则是一种贪婪的算法，从算法层面上看 `A*算法` 是要优于 `Dijkstra` 算法的。但 `A*算法` 的效率依赖与 `h(n)` 的设计，当 `h(n)` 设计不好时，`A*算法` 的效率也会随之下降，甚至退化为 `Dijkstra` 算法，