实验报告

BFS

```python
from collections import deque
#bfs
def bfs_search(graph, start, end):
    if start not in graph or end not in graph:
        return [] , -1#无效起始
    path = []
    #请在以下区域完成BFS算法的代码，求解路径path
    # parent[node] 表示节点的父节点
    parent = {}
    # visited 用于记录已经访问过的节点
    visited = {}
    for node in graph.keys():
        visited[node] = False
    # 初始化起始节点
    parent[start] = None
    visited[start] = True
    # BFS
    queue = deque([start])
    while queue:
        node = queue.popleft()
        for neighbor in graph[node]:
            if not visited[neighbor]:
                visited[neighbor] = True
                parent[neighbor] = node
                queue.append(neighbor)
    # 从终点回溯到起点
    node = end
    while node:
        path.insert(0, node)
        node = parent[node]

    return path, len(path)-1

# 含环图的邻接表
cyclic_graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D'],  # A-B-A 形成环
    'C': ['A', 'D', 'F'],
    'D': ['B', 'C', 'E'],
    'E': ['D', 'F'],
    'F': ['C', 'E']
}

# 测试 BFS
path, dist = bfs_search(cyclic_graph, 'A', 'F')
print(f"路径: {path}, 距离: {dist}")
```

思路

边权值都是1，没必要更新权重，直接visited标注得到就是最优解。搜索过程中记录搜索路径，搜索到结果后回溯得到路径

输出结果如下，**符合预期**

```
路径: ['A', 'C', 'F'], 距离: 2
```

DFS

```python
from collections import deque
#dfs
def dfs_search(graph, start, end):
    if start not in graph or end not in graph:
        return [], -1#无效起始
    path = []
    #请在以下区域完成DFS算法的代码，求解路径path

    path.append(start)
    visited = {}
    for node in graph.keys():
        visited[node] = False
    # DFS
    def dfs(node, visited, path):
        if node == end:
            return [path[:]]
        paths = []
        visited.add(node)
        for neighbor in graph[node]:
            if neighbor not in visited:
                path.append(neighbor)
                paths.extend(dfs(neighbor, visited, path))
                path.pop()

        visited.remove(node)
        return paths

    best_paths = dfs(start, set(), [start])
    # print(best_paths)
    path = min(best_paths, key=len)

    return path, len(path)-1

# 含环图的邻接表
cyclic_graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D'],  # A-B-A 形成环
    'C': ['A', 'D', 'F'],
    'D': ['B', 'C', 'E'],
    'E': ['D', 'F'],
    'F': ['C', 'E']
}

```

```
# 测试 DFS
path, dist = dfs_search(cyclic_graph, 'A', 'F')
print(f"路径: {path}, 距离: {dist}")
```

思路

如果只是找到可行解，那么用栈记录路径，迭代搜索，遇到结尾中断并输出即可。

以上代码是找到最优解的方案，定义了一个局部函数（匿名函数？不知道python管这个叫什么），递归找到所有可行解。
具体来说，对于每一次搜索，到达终点或无路可走后回溯，取消标记visited状态。每一次dfs都持有自己的visited状态，（空间复杂度较高）。

其中line29

```
[['A', 'B', 'D', 'C', 'F'], ['A', 'B', 'D', 'E', 'F'], ['A', 'C', 'D', 'E', 'F'], ['A', 'C', 'F']]
```

对所有路径按长度排序，得到最优解（不唯一）

```
路径: ['A', 'C', 'F'], 距离: 2
```