实验 1：图遍历算法

1. 代码实现

```Python
from collections import defaultdict, deque
import time

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def add_edge(self, src, dest):
        self.graph[src].append(dest)
        self.graph[dest].append(src)

    def read_graph_from_file(self, filename):
        with open(filename, 'r') as file:
            lines = file.readlines()
            self.start_vertex = int(lines[1].strip())
            for line in lines[2:]:
                src, dest = map(int, line.strip().split())
                self.add_edge(src, dest)

    def dfs(self, start, visited=None):
        if visited is None:
            visited = set()
        visited.add(start)
        print(start, end=' ')
        for neighbor in sorted(self.graph[start]):
            if neighbor not in visited:
                self.dfs(neighbor, visited)

    def bfs(self, start):
        visited = set()
        queue = deque([start])
        while queue:
            vertex = queue.popleft()
            if vertex not in visited:
                visited.add(vertex)
```

```
                print(vertex, end=' ')
                for neighbor in sorted(self.graph[vertex]):
                    if neighbor not in visited:
                        queue.append(neighbor)

if __name__ == '__main__':
    start_time = time.time()
    graph = Graph()
    graph.read_graph_from_file('input.txt')
    print("DFS from vertex", graph.start_vertex, ": ", end='')
    graph.dfs(graph.start_vertex)
    print("\nBFS from vertex", graph.start_vertex, ": ", end='')
    graph.bfs(graph.start_vertex)
    end_time = time.time()
    print(f"\nBFS 运行时间: {end_time - start_time:.6f}秒")
```

2. 实现描述

主要变量

名称	类型	描述
graph	defaultdict(list)	这是一个字典，用于存储图的邻接表结构。每个键代表一个顶点，值是一个列表，包含所有与该顶点相连的邻居顶点。使用 `defaultdict` 的好处是，如果尝试访问一个不存在的键，它会自动创建一个空列表，这样可以简化边的添加操作。
start_vertex	int	该变量存储图的起始顶点。在从文件读取图数据时，第二行指定的顶点会被赋值给这个变量，用于后续的搜索操作（DFS 和 BFS）。

主要函数

名称	参数	描述
read_graph_from_file	`filename`（输入文件名）	该方法从指定的文件中读取图的结构。文件的第一行通常包含图的相关信息（如顶点数等），第二行指

		定起始顶点，后续行每行包含一对源和目标顶点。通过调用 `add_edge` 方法，图的邻接表得以构建。
dfs	`start`（起始顶点）， `visited`（已访问的顶点集合）	这个方法实现递归的深度优先搜索（DFS）。它将当前顶点标记为已访问，并打印出来。然后，对所有未访问的邻居顶点进行递归访问。`visited` 是一个集合，用于追踪已访问的顶点，以避免重复访问。
bfs	`start`（起始顶点）	这个方法实现广度优先搜索（BFS）。它使用队列（`deque`）来管理待访问的顶点。首先，将起始顶点加入队列，然后从队列中取出顶点，标记为已访问并打印。接着，将所有未访问的邻居加入队列，以继续遍历。

3. 运行截图

```
PS D:\GitWorkspace\local\python> & D:/Software/anaconda3/python.exe d:/GitWorkspace/local/python/code/dfs_bfs.py
DFS from vertex 1 : 1 2 4 8 5 9 10 6 3 7
BFS from vertex 1 : 1 2 3 4 5 6 7 8 9 10
BFS运行时间: 0.001000秒
```