图

1.1 完成图的邻接矩阵表示方法，以及图的深度优先搜索与广度优先搜索算法。要求报告给出简单的实验思路（程序的运行方法、简单的分析）、代码与运行结果图

graph.py

```python
from collections import deque

import matplotlib.pyplot as plt
import networkx as nx


class Graph:
    def __init__(self, vertices):
        """
        构建图，初始化全 0 邻接矩阵
        """
        self.vertices = vertices
        self.adj_matrix = [[0] * vertices for _ in range(vertices)]

    def add_edge(self, v1, v2):
        """
        邻接矩阵加边
        """
        self.adj_matrix[v1][v2] = 1
        self.adj_matrix[v2][v1] = 1

    def add_edges(self, edges: list):
        """
        邻接矩阵批量加边
        """
        for edge in edges:
            self.add_edge(edge[0], edge[1])


def dfs(graph, start, visited):
    """
    递归实现 dfs，优先走编号小的边。
    """
    print(start, end=" ")
    visited[start] = True

    for i in range(graph.vertices):
        if graph.adj_matrix[start][i] == 1 and not visited[i]:
            dfs(graph, i, visited)


def bfs(graph, start):
    """
    使用队列实现 bfs。首先将初始节点设为访问并加入队列。
    接下来每轮循环都出队一个节点，将未访问过的其邻居节点加入队列。
    """
    visited = [False] * graph.vertices
    queue = deque([start])
    visited[start] = True
    while queue:
        current = queue.popleft()
        print(current, end=" ")

        for i in range(graph.vertices):
            if graph.adj_matrix[current][i] == 1 and not visited[i]:
                queue.append(i)
                visited[i] = True


def visualize_graph(graph):
    G = nx.Graph()

    G.add_nodes_from(range(graph.vertices))

    for i in range(graph.vertices):
        for j in range(graph.vertices):
            if graph.adj_matrix[i][j] == 1:
                G.add_edge(i, j)

    pos = nx.spring_layout(G)
    nx.draw(
        G,
        pos,
        with_labels=True,
        font_weight="bold",
        node_size=700,
        node_color="skyblue",
        font_color="black",
        font_size=10,
    )
    plt.show()


g = Graph(6)
g.add_edges([(0, 1), (0, 2), (1, 3), (2, 4), (0, 5), (3, 4), (3, 5)])


print("DFS traversal:")
dfs(g, 0, [False] * g.vertices)
print("\n")

print("BFS traversal:")
bfs(g, 0)

# visualize_graph(g)

# DFS traversal:
# 0 1 3 4 2 5

# BFS traversal:
# 0 1 2 5 3 4

print()
for j in range(len(g.adj_matrix)):
    print("[{}]".format(j), end=",")
print()
for ind, i in enumerate(g.adj_matrix):
    print("[{}]".format(ind), end=",")
    for j in i:
        print("[{}]".format(j), end=",")
    print()
```

邻接矩阵：

\	**0**	**1**	**2**	**3**	**4**	**5**
0	0	1	1	0	0	1
1	1	0	0	1	0	0
2	1	0	0	0	1	0
3	0	1	0	0	1	1
4	0	0	1	1	0	0
5	1	0	0	1	0	0

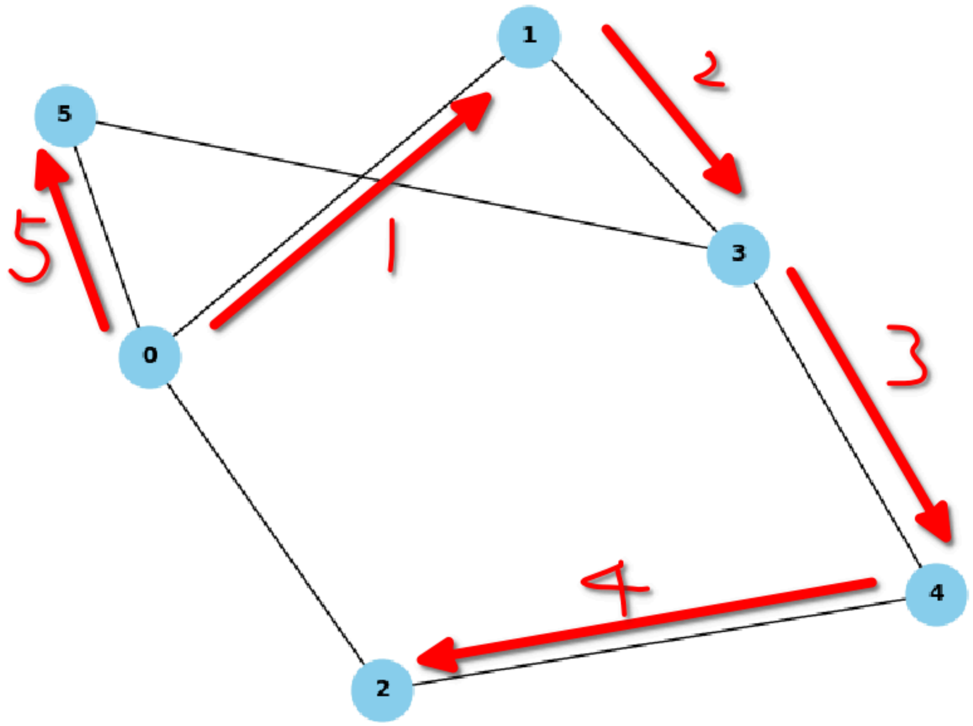

图 1　dfs

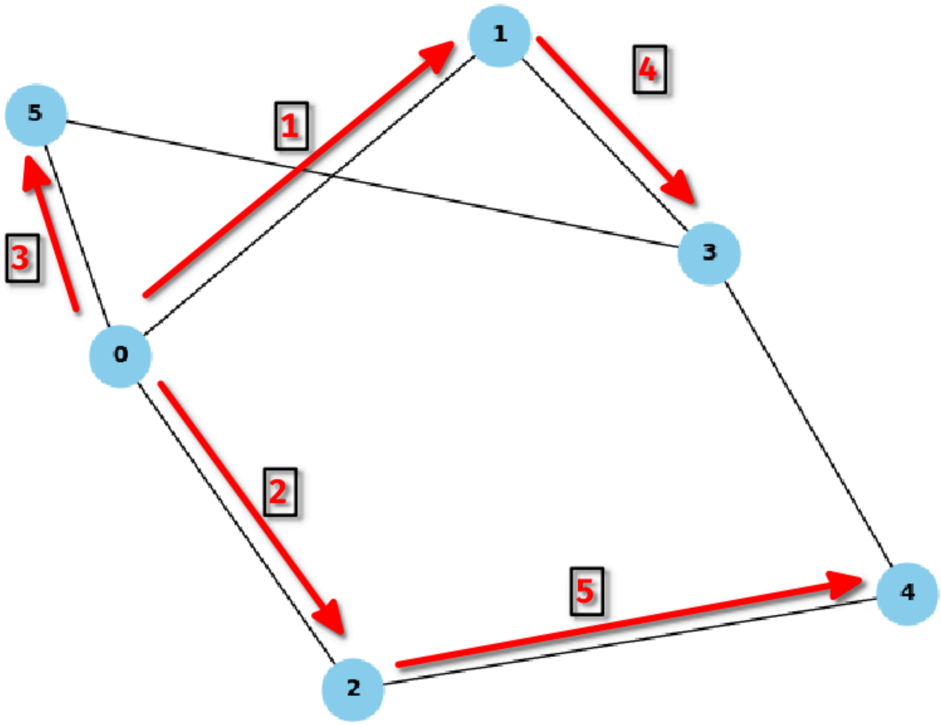

图 2　**bfs**