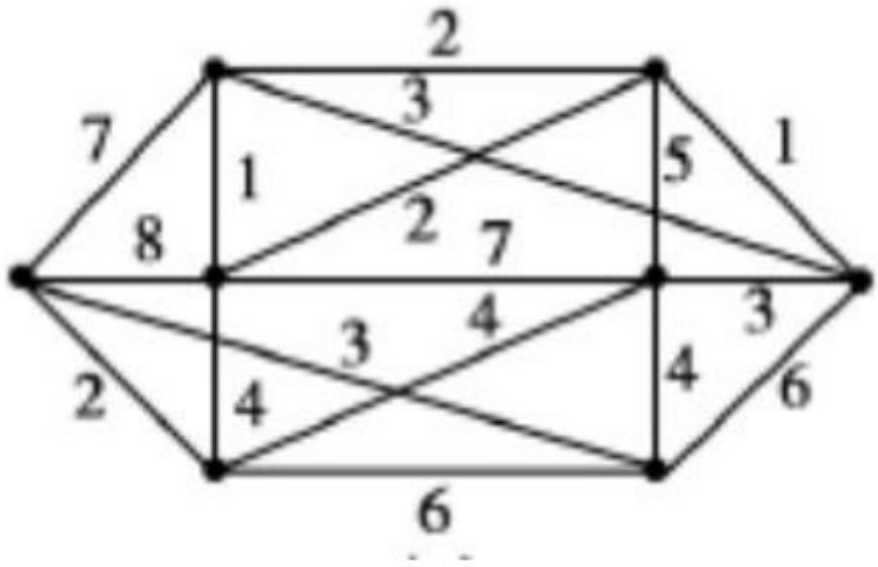
Graph Theory

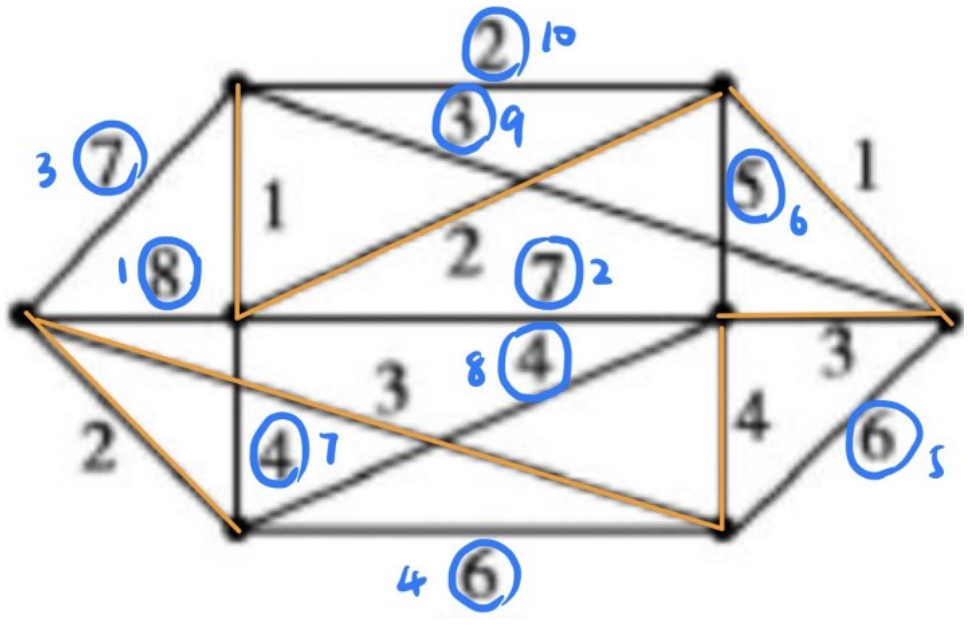
Prob 1

0. 原图



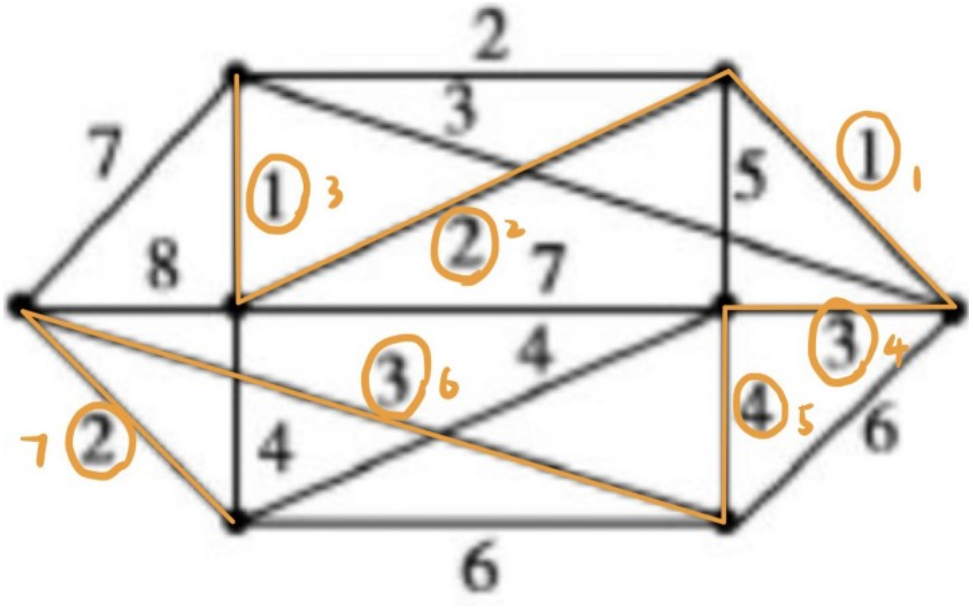
1. 破圈法

- 我先把每条边都描上颜色，然后按边从大到小的顺序逐一破圈
- 蓝色表示的是我的破圈顺序，橙色表示的是最后剩下的最小生成树



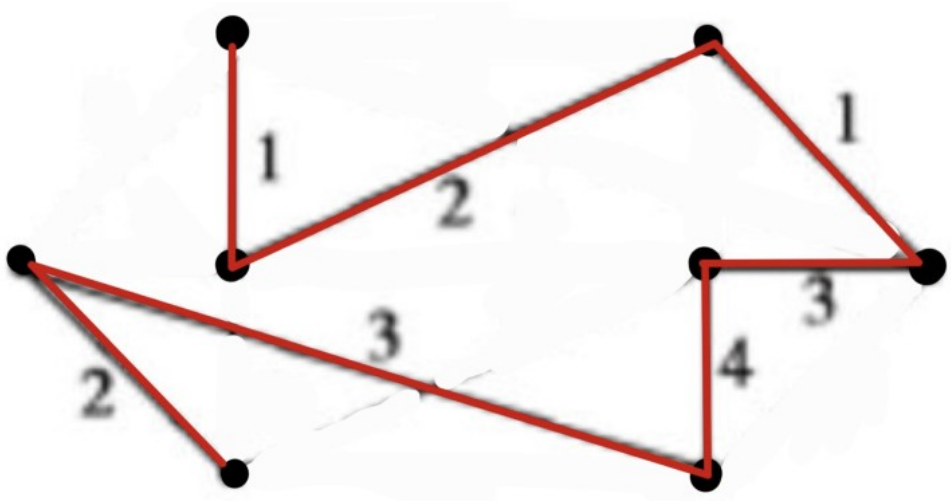
2. 避圈法

- 我按边从小到大的顺序逐一找边，直到所有顶点都被连接为止
- 橙色表示的是最后剩下的最小生成树及其顺序



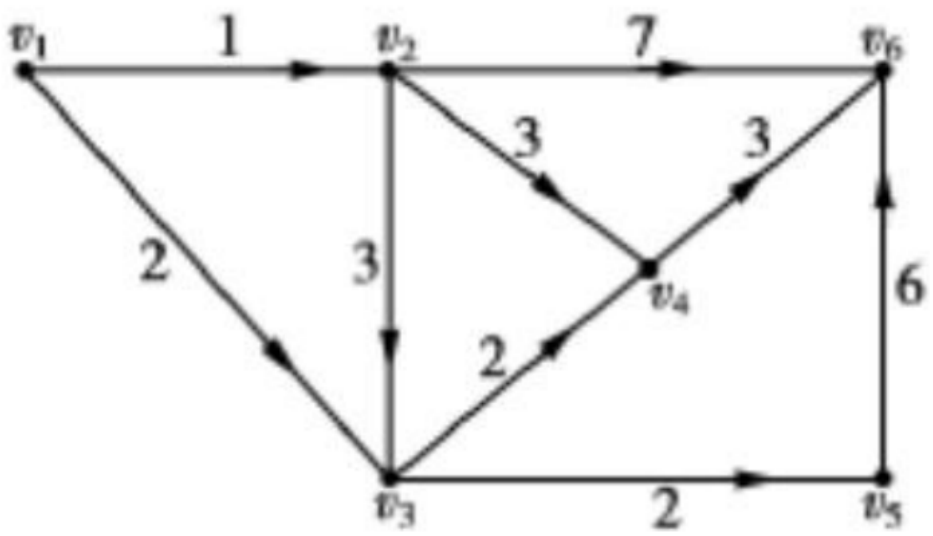
4. 总结

- 使用两种方法得到的最小生成树均如图，不过实际上还有另一种方案，即用最上面的“2”替换下面斜边的“2”，得到的效果是一样的
- 最小生成树边的总权值为16

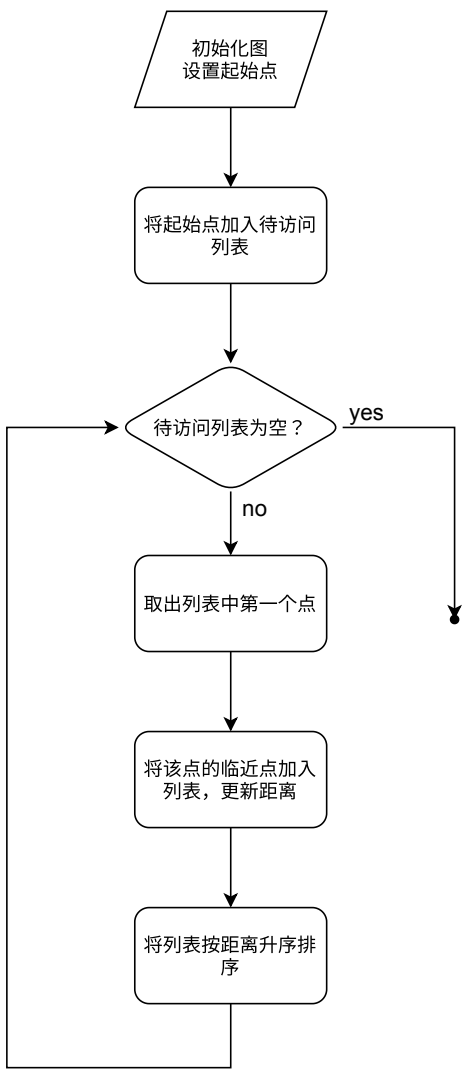


Prob 2

原图



算法思路



输出结果

- 输出为从起始点v_1到各个点的距离
- 手工对照验证，发现没问题

```
[0, 1, 2, 4, 4]
```

v1	v2	v3	v4	v5	v6
0	1	2	4	4	7

附录

Prob2 Code

```
import numpy as np
from queue import PriorityQueue


class Graph:
    def __init__(self, n):
        self.v = n
        self.edges = [[-1 for i in range(n)] for j in range(n)]
        self.visited = []

    ## 有向图
    def add_edge(self, u, v, weight):
        self.edges[u][v] = weight


def dijkstra(graph, s):
        distance = [np.inf for i in range(graph.v)]
        distance[s] = 0          # 起始点离自己距离为0

        to_visit = PriorityQueue()
        to_visit.put((0,s))      # 访问起始点

        while not to_visit.empty():
            (dist, current_v) = to_visit.get()            # 取出最小的边
            graph.visited.append(current_v)               # 确认ok的点

            for neighbor in range(graph.v):
                if graph.edges[current_v][neighbor] != -1 and neighbor not in
graph.visited:
                    d = graph.edges[current_v][neighbor]
                    old_dis = distance[neighbor]
                    new_dis = distance[current_v] + d
                    if new_dis < old_dis:
```

```
                        to_visit.put((new_dis, neighbor))
                        distance[neighbor] = new_dis
        return distance


if __name__ == '__main__':
    ## Init
    start_v = 1
    n = 6
    G = Graph(n+1)
    G.add_edge(1,2,1)
    G.add_edge(1,3,2)
    G.add_edge(2,3,3)
    G.add_edge(2,4,3)
    G.add_edge(2,6,7)
    G.add_edge(3,4,2)
    G.add_edge(3,5,2)
    G.add_edge(4,6,3)
    G.add_edge(5,6,6)

    ## Dijkstra
    distance = dijkstra(G, start_v)[1:]
    print(distance)
```