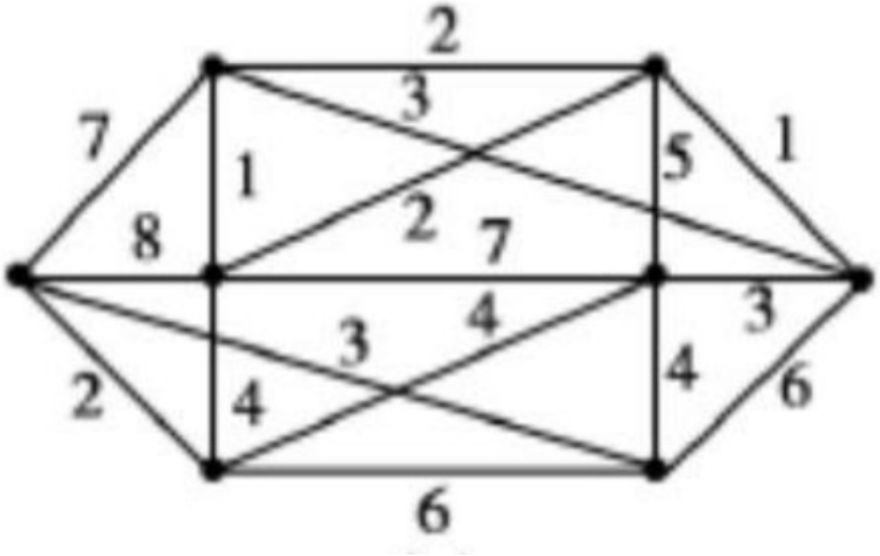
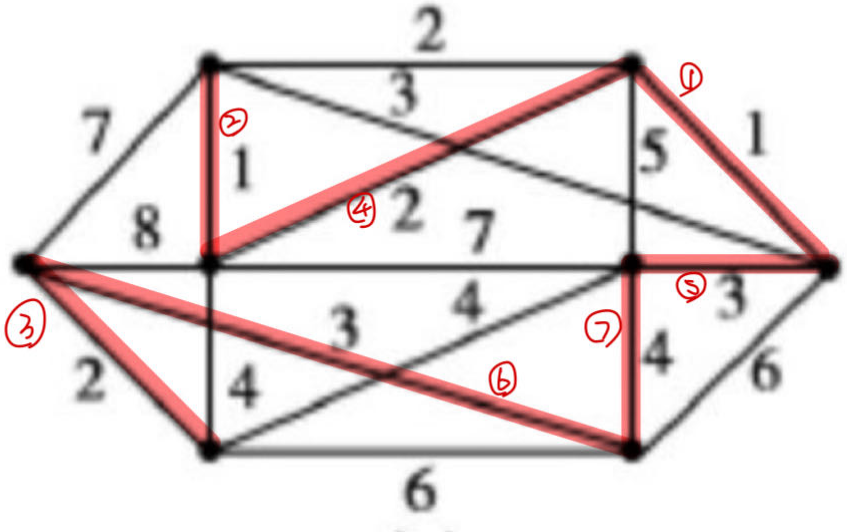
运筹学-图论作业

问题1

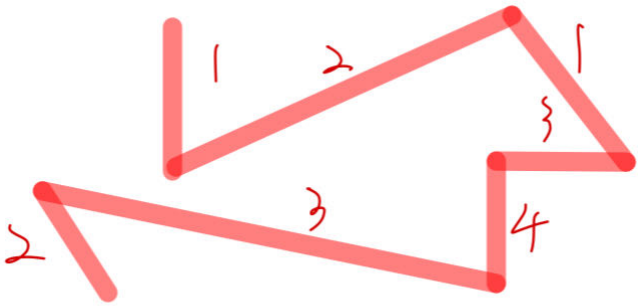
原图：



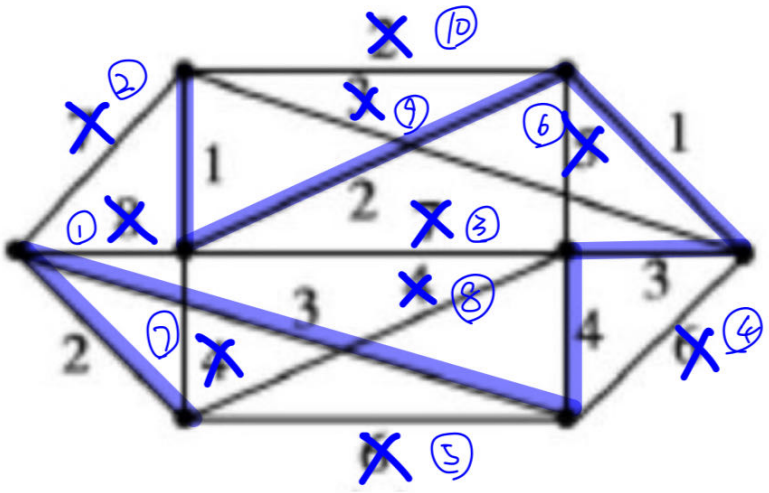
- 避圈法
    - 红线和序号代表每一步选取的边，从权重最小的边开始选取并保证不与前面所选的边构成圈，直到所有顶点间均存在通路为止



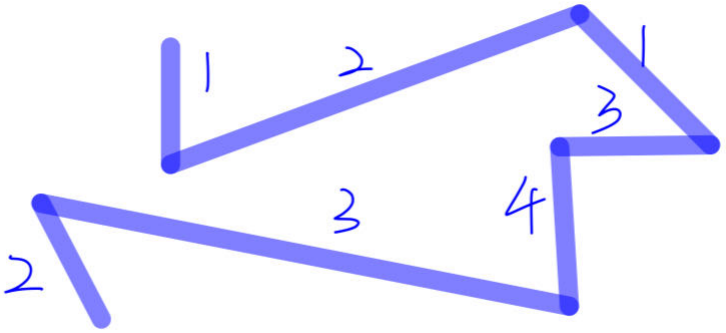
    - 最终所得的最小生成树，其权重之和为16



- 破圈法
    - 蓝叉和序号代表每一步删去的边，每一步都任取一个圈，从圈中删去权重最大的一条边，直到得到一个不含圈的图为止。蓝线所连即为剩下的最小生成树。

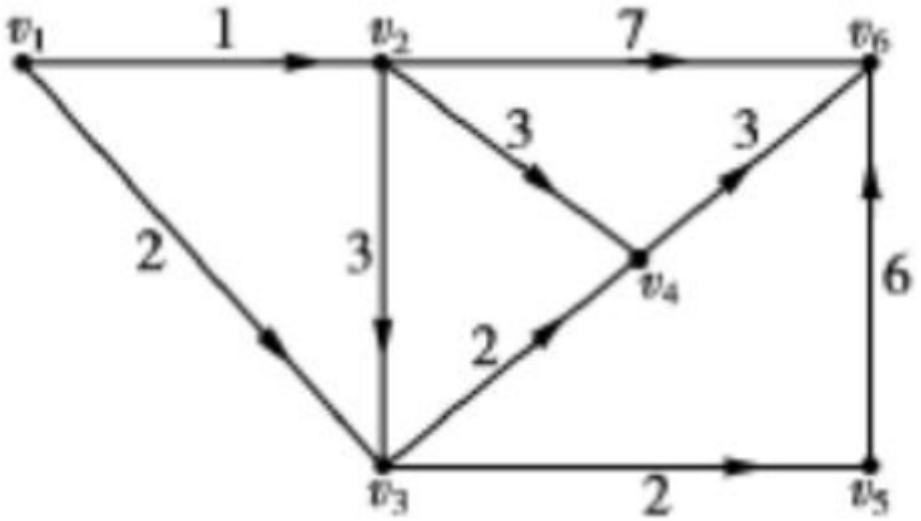


    - 最终所得的最小生成树，其权重之和为16

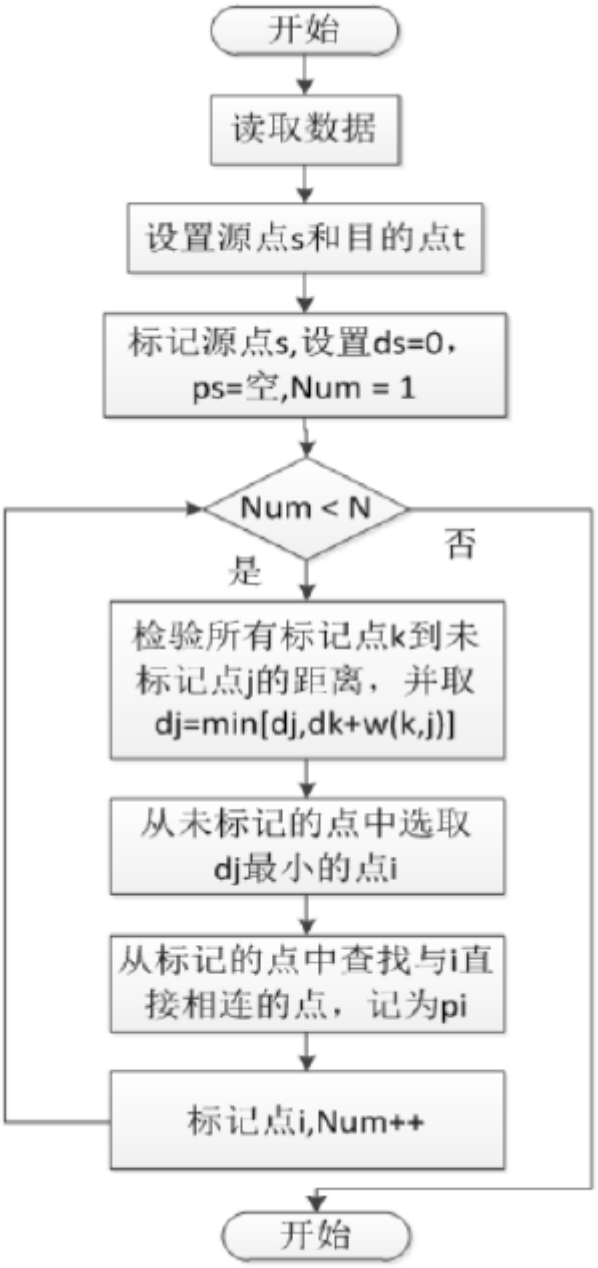


问题2

原图



算法流程图：



代码如下：

```
import heapq

def dijkstra(graph, start_vertex):
    # Initialize distances and priority queue
    distances = {vertex: float('infinity') for vertex in graph}
    distances[start_vertex] = 0
    priority_queue = [(0, start_vertex)]
    predecessors = {vertex: None for vertex in graph}
```

```
9
10      while priority_queue:
11          current_distance, current_vertex =
    heapq.heappop(priority_queue)
12
13          # Nodes can get added to the priority
    queue multiple times.
14          # We only process a vertex the first
    time we remove it from the priority queue.
15          if current_distance >
    distances[current_vertex]:
16              continue
17
18          # Update the distance for each
    neighbor
19          for neighbor, weight in
    graph[current_vertex].items():
20              distance = current_distance +
    weight
21
22              # Only consider this new path if
    it's better
23              if distance < distances[neighbor]:
24                  distances[neighbor] = distance
25                  predecessors[neighbor] =
    current_vertex
26                  heapq.heappush(priority_queue,
    (distance, neighbor))
27
28      return distances, predecessors
29
30  def get_shortest_path(predecessors,
    start_vertex, end_vertex):
31      path = []
```

```
    current_vertex = end_vertex
    while current_vertex is not None:
        path.append(current_vertex)
        current_vertex = predecessors[current_vertex]
    path.reverse()
    return path

# Define the graph based on the image
graph = {
    'v1': {'v2': 1, 'v3': 2},
    'v2': {'v3': 3, 'v4': 3, 'v6': 7},
    'v3': {'v4': 2, 'v5': 4},
    'v4': {'v5': 2, 'v6': 3},
    'v5': {'v6': 6},
    'v6': {}
}

# Get the shortest path from v1 to all vertices
distances, predecessors = dijkstra(graph, 'v1')

# Get the specific path from v1 to v6
shortest_path_to_v6_path = get_shortest_path(predecessors, 'v1', 'v6')
print(shortest_path_to_v6_path)

```

运行结果：

```
(base) ➜ pytorch /home/geek2/anaconda3/bin/python /home/geek2/d2l-zh/pytorch/op_3.py
['v1', 'v2', 'v4', 'v6']
```

即，v1到v6的最短路为：

$v_1 \to v_2 \to v_4 \to v_6$, 权值和为7