7.7

(1)

```
      a  b  c  d  e  f  g  h
   a [0, 4, 3, 0, 0, 0, 0, 0]
   b [4, 0, 5, 5, 9, 0, 0, 0]
   c [3, 5, 0, 5, 0, 0, 0, 5]
   d [0, 5, 5, 0, 7, 6, 5, 0]
   e [0, 9, 0, 7, 0, 3, 0, 0]
   f [0, 0, 0, 6, 3, 0, 2, 0]
   g [0, 0, 0, 5, 0, 2, 0, 6]
   h [0, 0, 5, 0, 0, 0, 6, 0]
```

最小生成树, 边长和27

```
a -> c -> d -> g -> f ->e
 \    \->h
  \->b
```

(2)

```
a -> [(c, 3), (b, 4)]
b -> [(a, 4), (d, 5), (c, 5), (e, 9)]
c -> [(a, 3), (h, 5), (d, 5), (b, 5)]
d -> [(g, 5), (c, 5), (b, 5), (f, 6), (e, 7)]
e -> [(f, 3), (d, 7), (b, 9)]
f -> [(g, 2), (e, 3), (d, 6)]
g -> [(f, 2), (d, 5), (h, 6)]
h -> [(c, 5), (g, 6)]
```

最小生成树, 边长和27

```
a -> b -> d -> g -> f ->e
 \-> c ->h
```

7.9

七种拓扑排序

```
1 5 2 6 3 4
1 5 2 3 6 4 #7.5.1 算法求得
1 5 6 2 3 4
5 1 2 6 3 4
5 1 2 3 6 4
5 1 6 2 3 4
5 6 1 2 3 4
```

7.22

python语言实现

```
def havepath(self, start, dest):
    '''判断两个顶点之间是否可达\n
        start 起点
        dest 终点
        返回值 True/False
    '''
    if start == dest:
        return True
    # 用list模拟一个stack
    # 里面存放顶点索引
    have_way = False
    stack = list()
    stack.append(self.vertex[start])
    visited = [False for i in range(self.vertex_num)]
    while True:
        try:
            # 取栈顶元素
            vertex = stack.pop()
        except IndexError:
            # 栈空时抛出 IndexError异常，此时跳出循环
            # print("Empty")
            break
        if visited[vertex.index] is True:
            continue
        # 该顶点vertex == dest 找到,break
        if vertex.index == dest:
            have_way = True
            break
        visited[vertex.index] = True
        # vertex 中的每个未被访问过的邻接顶点入栈
        for edge in self.adjlist[vertex.index]:
            adj_vertex = self.vertex[edge.v2]
            if visited[adj_vertex.index] is False:
                stack.append(adj_vertex)

    return have_way
```

7.24

```
def DFS(self, start=0):
    '''Depth First Search 深度优先搜索\n
    start 搜索起点的索引
    返回值 一个包含搜索结果的列表
    '''
    ret = list()
    # 用list模拟一个stack
    # 里面存放顶点索引
    stack = list()
    stack.append(self.vertex[start])
    visited = [False for i in range(self.vertex_num)]
    while True:
        try:
            # 取栈顶元素
            vertex = stack.pop()
        except IndexError:
            # 栈空时抛出 IndexError异常，此时跳出循环
            # print("Empty")
            break
        if visited[vertex.index] is True:
            continue
        # 该顶点vertex未被访问,则访问
        ret.append(vertex)
        visited[vertex.index] = True
        # vertex 中的每个未被访问过的邻接顶点入栈
        for edge in self.adjlist[vertex.index]:
            adj_vertex = self.vertex[edge.v2]
            if visited[adj_vertex.index] is False:
                stack.append(adj_vertex)

    return ret
```

完整代码在
https://github.com/chenjr15/Data_Structure_Assignments/tree/master/assignment_12_10