云南大学数学与统计学院
上机实践报告

课程名称： 数据结构与算法实验	**年级：** 2015 级	**上机实践成绩：**
指导教师： 陆正福	**姓名：** 刘鹏	
上机实践名称： 图结构实验	**学号：** 20151910042	**上机实践日期：** 2017-04-10
上机实践编号： No.14	**组号：**	**上机实践时间：** 上午三、四节

一、实验目的

1. 熟悉与图有关的数据结构与算法
2. 熟悉主讲教材 Chapter 14 的代码片段

二、实验内容

1. 与图有关的数据结构设计与算法设计；
2. 调试主讲教材 Chapter 14 的 Python 程序；

三、实验平台

Windows 10 Enterprise 中文版；
Python 3.6.0；
Wing IDE Professional 6.0.2-1 集成开发环境。

四、实验记录与实验结果分析

```
# 14.2.5 Python Implementation

class Graph:
    """Representation of a simple graph using an adjacency map."""

    #------------------------- nested Vertex class -------------------------
    class Vertex:
        """Lightweight vertex structure for a graph."""
        __slots__ = '_element'

        def __init__(self,x):
            """Do not call constructor directly. Use Graph's insert_vertex(x)."""
            self._element = x

        def element(self):
            """Return element associated with this vertex."""
            return self._element

        def __hash__(self):        #  will allow vertex to be a map/set key
            return hash(id(self))

    #------------------------- nested Edge class -------------------------
    class Edge:
        """Lightweight edge structure for a graph."""
        __slot__ = '_origin','_destination','_element'

        def __init__(self,u,v,x):
            """Do not call constructor directly. Use Graph's insert_edge(u,v,x)."""
            self._origin = u
            self._destination = v
            self._element = x

        def endpoints(self):
            """Return (u,v) tuple for vertices u and v."""
```

```python
            return (self._origin,self._destination)

        def opposite(self,v):
            """Return the vertex that is opposite v on this edge."""
            return self._destination

        def element(self):
            """Return element associated with this edge."""
            return self._element

        def __hash__(self):         # will allow edge to be a map/set key
            return hash((self._origin,self._destination))

    #------------------------- Graph class -------------------------
    def __init__(self,directed=False):
        """Create an empty graph (undirected, by default).

        Graph is directed if optional parameter is set to True.
        """
        self._outgoing = {}
        # only create second map for directed graph; use alias for undirected
        self._incoming = {} if directed else self._outgoing

    def is_directed(self):
        """Return True if this is a directed graph; False if undirected.

        Property is based on the original declaration of the graph, not its contents.
        """
        return self._incoming is not self._outgoing    # directed if maps are distinct

    def vertex_count(self):
        """Return the number of vertices in the graph."""
        return len(self._outgoing)

    def vertices(self):
        """Return an iteration of all vertices of the graph."""
        return self._outgoing.keys()

    def edge_count(self):
        """Return the number of edges in the graph."""
        total = sum(len(self._outgoing[v]) for v in self._outgoing)
        # for undirected graphs, make sure not to double-count edges
        return total if self.is_directed() else total // 2

    def edges(self):
        """Return a set of all edges of the graph."""
        result = set()         # avoid double-reporting edges of undirected graph
        for secondary_map in self._outgoing.values():
            result.update(secondary_map.values())         # add edges to resulting set
        return result

    def get_edge(self,u,v):
        """Return the edge from u to v, or None if not adjacent."""
        return self._outgoing[u].get(v)

    def degree(self,v,outgoing=True):
        """Return number of (outgoing) edges incident to vertex v in the graph.

        If graph is directed, optional parameter used to count incoming edges.
        """
        adj = self._outgoing if outgoing else self._incoming
        return len(adj[v])

    def incident_edges(self,v,outgoing=True):
        """Return all (outgoing) edges incedent to vertex v in the graph.

        If graph is directed, optional parameter used to request incoming edges.
        """
        adj = self._outgoing if outgoing else self._incoming
        for edge in adj[j].values():
            yield edge

    def insert_vertex(self,x=None):
        """Insert and return a new Vertex with element x."""
        v = self.Vertex(x)
        self._outgoing[v] = {}
        if self.is_directed():
            self._incoming[v] = {}     # need distinct map for incoming edges
        return v

    def insert_edge(self):
        """Insert and return a new Edge from u to v with auxiliary element x."""
        e = self.Edge(u,v,x)
        self._outgoing[u][v] = e
        self._incoming[u][v] = e

#------------------------------ my main function ------------------------------

```

五、实验体会

Translation

Chapter 14 Graph Algorithm

＊第十四章 图论算法

A ***graph*** is a way of representing relationships that exist between pairs of objects. That is, a graph is a set of objects, called vertices, together with a collection of pairwise connections between them, called edges. Graphs have applications in modeling many domains, including mapping, transportation, computer networks, and electrical engineering. By the way, this notion of a “graph” should not be confused with bar charts and function plots, as these kinds of “graphs” are unrelated to the topic of this chapter.

＊图是表示对象之间存在的关系的一种方式。也就是说，图是一组对象，即顶点与边（顶点之间的成对连接）的集合。图在许多建模领域中有应用，包括地图，运输，计算机网络和电气工程。顺便提一句，这个“图”的概念不应该与条形图和函数图像混淆，因为这些“图”与本章的主题无关。

Viewed abstractly, a ***graph*** *G* is simply a set *V* of ***vertices*** and a collection *E* of pairs of ***edges***. Thus, a graph is a way of representing connections or relationships between pairs of objects from some set *V*. Incidentally, some books use different terminology for graphs and refer to what we call vertices as nodes and what we call edges as arcs. We use the terms “vertices” and “edges.”

＊抽象地看，图 *G* 只是顶点集合 *V* 与顶点之间边的集合 *E* 的并集。因此，图是表示来自某些集合 *V* 的对象之间的连接性或关系的方式。顺便提一下，一些书籍对于图使用不同的术语，并且将所谓的顶点称为节点，我们将所谓的边缘称为辐。我们使用的是“顶点”和“边”。

Edges in a graph are either directed or undirected. An edge (u,v) is said to be directed from u to v if the pair(u,v) if ordered, with u preceding v. An edge (u,v) is said to be undirected if the pair (u,v) is not ordered. Undirected edges are sometimes denoted with set notation, as {u,v}, but for simplicity we use the pair notation (u,v) , noting that in the undirected case (u,v) is the same as (v,u). Graphs are typically visualized by drawing the vertices as ovals or rectangles and the edges as segments or curves connecting pairs of ovals and rectangles. The following are some examples of directed and undirected graphs.

＊图中的边可能是有向的，也有可能是无向的。如果(u,v)是有序的，即 u 在 v 前面，那么(u,v)被认为是从 u 到 v 的有向边。如果(u,v)是无序的，那么(u,v)被认为是无向边。有时用{u,v}表示无向边，但为了简单起见，我们使用符号串(u,v)，注意在无向的情况下(u,v)与(v,u)相同。通常通过将顶点绘制为椭圆形或矩形，并将边缘作为连接一对椭圆或者矩形的线段或曲线来显示图形。以下是有向和无向图的一些示例。

If all the edges in a graph are undirected, then we say the graph is an undirected graph. Likewise, a directed graph, also called digraph, is a graph whose edges are all directed. A graph that has both directed and undirected edges is often called a mixed graph. Note that an undirected or mixed graph can be converted into a directed graph by replacing every undirected edge (u,v) by the pair of directed edges (u,v) and (v,u). It is often useful, however, to keep undirected and mixed graphs represented as they are, for such graph graphs have several applications, as the following example.

＊如果图中的所有边都是无向的，那么我们说图是一个无向图。同样地，有方向的图也称为有向图，其边都是有向的。有向和无向边都存在的图通常称为混合图。请注意，通过用一对有向边(u,v)和(v,u)替换每个无向边(u,v)，可以将无向或混合图形转换为有向图。然而，保持未定向和混合的图表通常是有用的，因为这样的图形图具有若干应用，如以下示例。

The two vertices joined by an edge are called the ***end vertices*** (or ***endpoints***) of the edge. If an edge is directed, its first endpoint is its ***origin*** and the other is the ***destination*** of the edge. Two vertices *u* and *v* are said to be ***adjacent*** if there is an edge whose end vertices are *u* and *v*. An edge is said to be ***incident*** to a vertex if the vertex is one of the edge’s endpoints. The ***outgoing edges*** of a vertex are the directed edges whose origin is that vertex. The ***incoming edges*** of a vertex are the directed edges whose destination is that vertex. The ***degree*** of a vertex *v*, denoted deg(*v*), is the number of incident edges of *v*. The ***in-degree*** and ***out-degree*** of a vertex *v* are the number of the incoming and outgoing edges of *v*, and are denoted indeg(*v*) and outdeg(*v*), respectively.

＊由边连接的两个顶点被称为边的端点（或端点）。如果边是有向的，则其第一个端点是其起点，而另一个是边的终点。如果存在一个端点为 u 和 v 的边，则两个顶点 u 和 v 被认为是相邻的。如果顶点是边的端点之一，则称边入射到顶点。顶点的出边是起始于该顶点的有向边。顶点的入边是终点为该顶点的有向边。顶点 v 的度数，表示为 deg(v)，指的是 v 的相邻边的数量。顶点 v 的 in-degree 和 out-degree 分别指的是 v 的入边和出边的数量，并且被分别表示为 indeg(v)和 outdeg(v)。

The definition of a graph refers to the group of edges as a ***collection***, not a ***set***, thus allowing two undirected edges to have the same end vertices, and for two directed edges to have the same origin and the same destination. Such edges are called ***parallel edges*** or ***multiple edges***. A flight network can contain parallel edges (Example 14.5), such that multiple edges between the same pair of vertices could indicate different flights operating on the same route at different times of the day. Another special type of edge is one that connects a vertex to itself. Namely, we say that an edge (undirected or directed) is a ***self-loop*** if its two endpoints coincide. A self-loop may occur in a graph associated with a city map (Example 14.3), where it would correspond to a “circle” (a curving street that returns to its starting point).

＊在图的定义中，我们将边组称为 collection，而不是 set，因为它允许两个无向边具有相同的顶点，同样也允许两个有向边具有相同的起点和终点。这些边被称为平行边或多元边。飞机航线图允许包含平行边缘（例 14.5），而这表示有不同的航班在一天中的不同时间飞行于这两点之间。另一种特殊类型的边是将顶点连接到其自身的边。如果边的两个端点重合，那么我们称这个边为是回路。在与城市地图相关联的图表（例 14.3）中可能会出现回路，其中它将对应于“环形街道”（返回到其起点的弯曲的街道）。

With few exceptions, graphs do not have parallel edges or self-loops. Such graphs are said to be ***simple***. Thus, we can usually say that the edges of a simple graph are a ***set*** of vertex pairs (and not just a collection). Throughout this chapter, we assume that a graph is simple unless otherwise specified.

＊除了少数例外，图中没有有平行边或圈。这样的图形被称为简单图。因此，我们通常可以说简单图的边是一组顶点的 set（而不仅仅是一个 collection）。在本章中我们所给出的图都是简单的，除非另有说明。

A ***path*** is a sequence of alternating vertices and edges that starts at a vertex and ends at a vertex such that each edge is incident to its predecessor and successor vertex. A ***cycle*** is a path that starts and ends at the same vertex, and that includes at least one edge. We say that a path is ***simple*** if each vertex in the path is distinct, and we say that a cycle is ***simple*** if each vertex in the cycle is distinct, except for the first and last one. A ***directed path*** is a path such that all edges are directed and are traversed along their direction. A ***directed cycle*** is similarly defined. For example, in Figure 14.2, (BOS, NW 35, JFK, AA 1387, DFW) is a directed simple path, and (LAX, UA 120, ORD, UA 877, DFW, AA 49, LAX) is a directed simple cycle. Note that a directed graph may have a cycle consisting of two edges with opposite direction between the same pair of vertices, for example (ORD, UA 877, DFW, DL 335, ORD) in Figure 14.2. A directed graph is ***acyclic*** if it has no directed cycles. For example, if we were to remove the edge UA 877 from the graph in Figure 14.2, the remaining graph is acyclic. If a graph is simple, we may omit the edges when describing path P or cycle C, as these are well defined, in which case P is a list of adjacent vertices and C is a cycle of adjacent vertices.

＊路径是一系列交替的顶点和边，它们从顶点开始并以顶点结束，其中的每条边都有起点与终点。圈是在同一顶点开始和结束的路径，并且圈至少包含一条边。我们说如果路径中的每个顶点是不同的，那我们就认为这是个简单路径；如果一个圈中除了起点与终点之外的顶点都是不同的，那么我们认为这个圈也是简单的。如果路径中的所有边都是有向边，并且路径方向沿边的方向，那么这个路径就是有向路径。有向圈也可以类似地定义。例如，在图 14.2 中，(BOS,NW 35,JFK,AA 1387,DFW)是一个有向的简单路径，(LAX,UA 120,ORD,UA 877,DFW,AA 49,LAX)是一个有向的简单圈。注意，在有向图中，允许存在由相同的一对顶点之间、具有相反方向的两个有向边所组成的圈，例如(ORD,UA 877,DFW,DL 335,ORD)。如果无有向圈，那有向图是无圈的。例如，如果我们从图 14.2 中删除边 UA 877，那么剩余的图是无圈的。如果图形是简单的，我们可以在描述路径 P 或圈 C 的时候省略边，因为这些都是已经规定好的，而在这种情况下，P 是相邻顶点的线性列表，C 是相邻顶点的圈形列表（即首尾合一）。

Given vertices u and v of a (directed) graph G, we say that u ***reaches*** v, and that v is ***reachable*** from u, if G has a (directed) path from u to v. In an undirected graph, the notion of ***reachability*** is symmetric, that is to say, u reaches v if an only if v reaches u. However, in a directed graph, it is possible that u reaches v but v does not reach u, because a directed path must be traversed according to the respective directions of the edges. A graph is ***connected*** if, for any two vertices, there is a path between them. A directed graph $\vec{G}$ is ***strongly connected*** if for any two vertices u and v of $\vec{G}$, u reaches v and v reaches u. (See Figure 14.3 for some examples.)

＊给定（有向）图 G 的顶点 u 和 v，我们说 u 指向 v，或者说 v 可以从 u 到达，意思是说 G 具有从 u 到 v 的（有向）路径，在无向图中，可达性的概念是对称的，也就是说，如果有 v 达到 u，则有 u 达到 v。然而在有向图中，可能 u 到达 v 但 v 不能到达 u，因为必须根据边的相应方向遍历有向路径。如果任意两个顶点之间有一条路径，则图形是连通的。如果对于 G 的任何两个顶点 u 和 v 可以互相到达，那么就称有向图 G 是强连接通的。（例子见图 14.3）

A ***subgraph*** of a graph G is a graph H whose vertices and edges are subsets of the vertices and edges of G, respectively. A ***spanning subgraph*** of G is a subgraph of G that contains all the vertices of the graph G. If a graph G is not connected, its maximal connected subgraphs are called the ***connected components*** of G. A ***forest*** is a graph without cycles. A ***tree*** is a connected forest, that is, a connected graph without cycles. A ***spanning tree*** of a graph is a spanning subgraph that is a tree. (Note that this definition of a tree is somewhat different from the one given in Chapter 8, as there is not necessarily a designated root.)

＊图 G 的子图 H 的顶点和边分别是 G 的顶点和边的子集。G 的支撑子图包含图 G 的所有顶点。如果图 G 是非连通的，则其最大的连通子图被称为 G 的连通分量。森林是没有圈的图。一棵树是一个连通的森林（只有一棵树），也就是没有圈的连通图。图的生成树是图的支撑子图，当然更是一棵树。（请注意，这里的树的定义与第 8 章给出的定义有些不同，因为这里的树不一定指定了根。）

END

六、参考文献

[1] Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, *Data Structures and Algorithms in Python*, Chapter 4
[2] 实验教材：汪萍，陆正福等编著，数据结构与算法的问题与实验