1. 图论
2. 图的定义与术语
3. 结构定义：图是由顶点集V {v}和弧集E {<u,v>}构成的数据结构
4. 有向图与无向图
   1. 若每条边规定了方向<u,v>则称其为有向图
   2. 若不规定方向<u,v>=<v,u>则称其为无向图（一条边相当于两条弧）
5. 网：弧或边上带权的图称为有向网与无向网
6. 子图：点集和弧集都是原图点集和弧集的子集，但是需要符合图的定义，如增加一条不包括两头节点的弧则不是子图
7. 完全图：
   1. 含有条边的图称为完全无向图
   2. 含有条弧的图称为完全有向图
8. 度：
   1. 出度ID：指向顶点a的弧的个数
   2. 入度OD：由顶点a出发的弧的个数
   3. 顶点的度D：度=入度+出度
9. 路径与回路
   1. 简单路径：序列中顶点不重复出现的路径
   2. 简单回路：第一个顶点与最后一个相同的简单路径
10. 连通图与连通分量
    1. 无向图中任意两个顶点之间都有路径相通，则称为连通图
    2. 若无向图是非连通图，则图中的极大连通子图称为此图的连通分量
    3. 有向图中任意两个顶点之间都有有向路径相同，则称为强连通图
    4. 一个连通图有n个顶点与e条边，其中n-1条边和n个顶点构成了一个极小连通子图，称此子图为此连通图的生成树
11. 图的存储与表示
12. 图的邻接矩阵存储表示（二维数组）
    1. 当节点i与节点j之间存在有向弧时A ij=1否则=0

//对于带权值的网，当i与j之间存在有向弧时A ij=w权值

当i与j之间没有有向弧时A ij=large（large是一个比所有权值都大的数）

* 1. 无向图的邻接矩阵是对称矩阵

//故而可以存储为三角矩阵来减少内存占用

* 1. 有向图的邻接矩阵不是对称矩阵
  2. 有向图第i行1的个数是顶点i的出度，第j列1的个数是顶点j的入度

无向图第i行1的个数（与第i列相同）可得顶点i的度

1. 图的邻接表存储表示（链表）

先用线性表存储各个顶点，然后每个顶点用链表指向下一个顶点，无向图中尽量将相互连接的排在先后紧跟的位置（有向图中不存在这样的问题），从而体现图的拓扑结构。但是在有向图的邻接表中只便于找某顶点出发的弧而不易找指向某点的弧。

//逆邻接表：每个顶点连接的是指向该顶点的弧

1. 图的遍历
2. 算法理论
   1. 只遍历顶点的线性表的话，无法遍历所有的边，无法体现拓扑结构
   2. 深度优先搜索遍历：从某个顶点v0出发，递归访问未访问过的邻接点，直至所有和v0有路径相通的点都被访问，产生深度优先生成树
   3. 广度优先搜索遍历：通过访问距离来人为使图产生层次
3. 算法实现：通过vis[w]为每一个顶点w标记“是否被访问”
4. 图的最小生成树

Above all：构造最小生成树的准则

①按照生成树的定义，n 个顶点的连通网络的生成树有 n 个顶点、n-1 条边；

②必须使用且仅使用该网络中的n-1条边来联结网络中的 n 个顶点；

③不能使用产生回路的边；

④各边上的权值的总和达到最小。

1. prim算法（每次操作添加点）
   1. 选取图中任意一个顶点v作为根节点，之后添加新的节点w
   2. 在添加的顶点w和已经在树上的顶点v之间必须要存在一条边，并且该边的权值在所有连通v与w的边中最小

//添加顶点时不能使边出现回路，若需要产生回路则回溯

//已添加顶点和未添加顶点视作两个集合

* 1. 直到所有的顶点都被连入了此树

Attention:在生成树的构造过程中，图中 n 个顶点分属两个集合：已落在生成树上的顶点集 U 和尚未落在生成树上的顶点集V-U ，则应在所有连通U中顶点和V-U中顶点的边中选取权值最小的边。

1. Kruskal算法（每次操作添加边）
   1. 使生成树上每一条边的权值尽可能小
   2. 先构造一个只含有n个顶点的子图
   3. 然后从权值最小的边开始，若它的添加不使子图中产生回路，则加上这条边
   4. 如此重复，直至加上 n-1 条边为止，即所有顶点都被连入了此树
2. 有向无环图（DAG）及其应用
3. 拓扑排序
   1. 从有向图中选取一个没有前驱的顶点并输出（有多个选择时最好按字典序）

//事实上是可以任选输出，所以结果可能不唯一

* 1. 从有向图中删去此顶点以及所有以它为尾的弧
  2. 重复以上步骤直到图空或者只剩下回路（没有无前驱的顶点）

——>explain：

这样的图用顶点表示活动，用有向边表示优先关系，称为AOV

没有前驱的节点：入度为0的节点

删去以它为尾的弧：所有它发出的弧的终点入度-1

1. 判断回路的方法
   1. 将入度为0的顶点存入栈或队列中，如果输出顶点小于网中所有节点个数，则说明存在环（空间与算法较为复杂）
   2. 利用记录节点入度的count数组（因为每次入度为0的节点被删除，此空间不再存储其他数据），设置一个指针指向每次被删去的入度为0的节点，构建链式栈，每次有节点出栈时便将指针指向上一个出栈的位置
2. 关键路径
3. AOE网：用边表示活动的网，是一种边带有权值的有向无环图，用顶点代表事件，边表示活动，边的权值表示活动需要的时间
4. 关键路径：从源点到汇点的最长路径长度，表示最短完成时间
5. ve(arly)与vl(ate)分别表示某事件的最早发生时间和最晚发生时间，由拓扑排序的过程确定各个时间，正排确定ve，倒排确定vl

Attention：不是任意一个关键活动的加速都能减少工期，因为别的节点或边可能成为新的关键路径

1. 最短路径
2. 问题描述：从网中的某个节点到另一个节点的路径不止一条，则寻找一条路径是的此路径上各边的权值之和最小
3. 单源点最短路径问题（Dijkstra）

思想：首先求出长度最短的一条最短路径，再参照它求出长度次短的一条最短路径，依次类推，直到从顶点v到其它各顶点的最短路径全部求出为止。

* 1. 初始化dist：
  2. 选择vj，若经过最短路径绕路至目标点能减少路径长度，则更新最短路径
  3. 重复此操作直到得出源点到所有点的最短路径

有边存在就取边的权值，无边存在则取∞（很大的值）

Explain：

（1）最短路径：在这条路径上，必定只含一条弧，并且这条弧的权值最小。

（2）次短路径：它只可能有两种情况，或者是直接从源点到该点(只含一条弧)； 或者是从源点经过顶点v1，再到达该顶点(由两条弧组成)。

1. 所有节点之间的最短路径（Floyd）

思想： 从 vi 到 vj 的所有可能存在的路径中，选出一条长度最短的路径。

* 1. 若<vi,vj>存在，则构造路径{vi,vj}
  2. 在路径上增加v1，若<vi,v1>,<v1,vj>存在，则比较路径{vi,v1,vj}和{vi,vj}长度，取长度短者为从vi到vj中间只经过序号不大于1的顶点的最短路径，即路径中所含顶点序号不大于1；
  3. 依次在路径上增加v3,…,vn。经过n次比较后，最后求得的必是从vi到vj的最短路径。

Explain：原理也是绕路