1. 图的基本概念及图的存储
   1. 有向图，无向图
      1. 图 : Graph， G(V,E)，由顶点集合V(G)和顶点间的二元关系集合E(G)（即边的集合或弧的集合）组成的数据结构
      2. 顶点：vertex， V(G)中的元素
      3. 阶数：图中的顶点数，n
      4. 边：edge，E(G)中的元素
      5. 边数：size，图中的边数，m
      6. 无向边：(a,b）
      7. 有向边：<a,b>
      8. 有向图：所有的边都有方向性
      9. 无向图：所有的边都没有方向性
      10. 弧：Arc，有向图中的边的另一种名称
      11. 有向图的基图：忽略有向图中所有边的方向，得到的无向图就是有向图的基图
   2. 完全/稀疏/稠密图
      1. 完全图：无向图中，所有顶点间都有一条边
      2. 有向完全图：有向图中，所有顶点间都有两条从一点到另一点的弧
      3. 稀疏图：Sparse Graph，一般认为， 即为稀疏图
      4. 稠密图：Dense Graph，边或弧的数目较多的图，接近完全图或有向完全图
      5. 平凡图：Trivial Graph，只有一个顶点的图
      6. 非平凡图：多于一个顶点
      7. 零图：null graph，E(G)为空
   3. 顶点
      1. 邻接顶点：无向图G，u,v是E(G)中的元素， 便称u为v的邻接顶点
      2. 无向边依附于(attach to)两个顶点，或称边与两个顶点相关联(incident)
      3. 邻接边：有一个共同顶点但不同的边
      4. 有向图的弧的起点邻接到(adjacent to)终点，终点邻接自(adjacent from)起点
   4. 顶点度数及度序列
      1. 顶点度数：deg(u)，与其相关联的边的数目
      2. 出度、入度：outdegreee，indegree，分别代表有向图中以顶点u为起点的有向边的数目od(u)和以顶点u为终点的弧的数目id(u)。有向图的顶点的度数：deg(u)=id(u)+od(u)
      3. 定理1：

有向图和无向图中，所有顶点度数总和，等于边数的两倍

* + 1. 奇点，偶点：odd vertex，even vertex，对应于顶点度数
    2. 孤立顶点：isolated vertex，度数为0
    3. 叶/端顶点：leaf/end vertex，度数为1
    4. 最小度：minimum degree，所有顶点的最小度数
    5. 最大度：maximum degree，…最大度数
    6. 度序列：degree sequence，将所有度数排成的一个序列
    7. 定理2：

Havel-Hakimi定理：根据已知序列，判断是否属于某图（可图）

非负整数组成的非增序列可图，当且仅当序列是可图的。

* + 1. 非增序列：递减 >> 为最大度
    2. 需将重排为非增序列
    3. 若出现负数，即为不可图的，若最终所有项变为零，则s可图
    4. 由一个序列构造出的图并不唯一
  1. 二部图、完全二部图
     1. 二部图：bipartite graph，无向图G(V,E)，顶点集合V包含两个没有公共元素的子集X,Y，且E 内所有边都连接一个X的元素和一个Y的元素
     2. 完全二部图：complete bipartite graph，X和Y的任意两个顶点之间都有边相连
     3. 定理3：

一个无向图G是二部图当且仅当G中没有奇数长度的回路

* 1. 同构与同态
     1. 图同构(graph isomorphism)：

Graph and are isomorphic if

1. There is a bijection f from V1 to V2
2. There is a bijection g from E1 to E2 that maps each edge (v,u) to (f(v),f(u))

\*自同构(automorphism)

* + 1. 图同态(graph homomorphism)

A graph homomorphism f from a graph to a graph , written , is a mapping from the vertex set of G to the vertex set of G’ such that implies

* + 1. 同态不必须是双射
  1. 子图与生成树
     1. 设有图G(V,E)和G(V’,E’)满足：，则称G‘为G的子图(subgraph)
     2. 生成树：spanning tree，一个无向连通图的生成树是他的包含所有顶点的极小联通子图。极小指边的数目极小
     3. 如果顶点数为n，则生成树的边数应为n-1。生成树可不唯一
     4. 顶点诱导子图：vertex-induced subgraph，G’为G的子图，若G’中任意顶点u,v，只要(u,v)是G中的边，则一定是G’中的边。则G’是由顶点集合V’诱导的G的子图(subgraph of G induced by V’)
     5. 边诱导子图：edge-induced subgraph，G’为G的子图。若G’以E’作为边集，以至少与E’中一条边关联的顶点构成顶点集V’，则G’是以V’诱导的子图
     6. 路径：path，
     7. 路径长度：路径中边的数目
     8. 简单路径：simple path，路径不重复经过顶点
     9. 回路：第一个顶点与最后一个顶点重合
     10. 简单回路/圈：simple circuit/cycle，除第一个与最后一个，路径不重复经过顶点。分为奇圈和偶圈
  2. 连通性
     1. 点连通：无向图中两点u,v之间有路径
     2. 连通图：connected graph，无向图中任意两点之间有路径
     3. 非连通图
     4. 连通分量：connected component，极大连通子图，极大指顶点个数极大
     5. 强连通图：strong connected graph，有向图中任意两点之间有路径
     6. 强连通分量：strong connected component
     7. 连通分量最少可只包含一个顶点
  3. 权值，有向网/无向网
     1. 权值：weight，与边相关的数
     2. 加权图/网：weighted graph/net，所有边都有权值
     3. 无向网，有向网

1. 图的遍历与活动网络问题
   1. DFS遍历
      1. 有回退，递归算法
      2. 描述：

由某一顶点v出发，访问其邻接顶点w1，再访问w1的还未被访问的邻接顶点w2，…，直至到达不再有可被访问的邻接顶点的顶点u；然后返回，看是否有其他未被访问的邻接顶点

* + 1. 时间复杂度：

其中，b生成树的为分支系数（如二叉树b=2），m为生成树的深度，n为图的顶点个数

* + 1. DFS无法求得最短路径，且若某一分支有无穷长度，则DFS将不收敛。这种情况可应用有界DFS
  1. BFS遍历
     1. 没有回退，非递归
     2. 对一个无向连通图，访问某一顶点v后，访问v所有未被访问的顶点w1 w2 ...wn，然后再按顺序访问w1 w2 ...wn的所有未被访问的顶点；直至图中所有顶点都被访问过
     3. 时间复杂度：

其中V为阶数，E为边数

最差情况下，BFS需查找所有顶点的所有路径因此为O(|V|+|E|)

* + 1. 完全性：在解存在的情况下，BFS一定可以找到
    2. 若解不存在，且图无限大，BFS将不收敛
    3. 不加权或所有边权值相等的情况下，BFS可以回传最短路径（最短≠最优），若为加正权图，应用dijkstra算法
    4. BFS应用：
* 查找连接组件

运行BFS所经过的所有节点构成一个连接组件

* 查找连接组件中的所有节点
* 查找非加权图中的最短路径
* 测试是否为二分图

从某一节点开始搜索，并交替的给顶点标签。如起点标为0，而顶点的所有未被标记的邻居标为1，之后BFS，对标为0的顶点将所有未被标记的邻居标为1，对标为1的顶点将所有未被标记的邻居标为0，以此类推。若任一顶点与跟其由相同标签的节点关联，则不是二分图。否则至所有顶点都被访问也没有这种情况，则为二分图

* 1. AOV活动网络
     1. 活动网络：描述工程中各子工程的安排问题，分为AOV和AOE网络
     2. 活动：activity，子工程
     3. AOV网络：Activity On Vertices，用顶点表示活动，用有向边表示不同活动的必要先后顺序，成为顶点表示活动的网络
     4. AOV网络中的弧没有加权
     5. AOV网络只表示不同活动间的先后次序
     6. 活动关系：若存在弧<u,v>则活动u必须在活动v之前完成
     7. 直接前驱：immediate predecessor，存在<u,v>，u是v的直接前驱
     8. 直接后继：immediate successor，存在<u,v>，v是u的直接后继
     9. 后继：successor，存在有向路径，v是u的后继
     10. 传递性：transivitity，b是a的后继，c是b 的后继，则c是a的后继
     11. 反自反性：irreflexivity，顶点不可是自身的前驱或后继

由此，AOV中不可以由有向回路/有向环，即AOV必须是一个有向无环图(Directed Acylic Graph,DAG)

* + 1. 拓扑排序：topological sort，将各个顶点排列成一个线性有序的序列
    2. 如果AOV无法完成拓扑排序，则该AOV必定含有有向环
    3. 若拓扑排序存在，则可不唯一
    4. 实现拓扑排序：

1. 从AOV网络中选择一个id=0（即没有直接前驱）的顶点并输出该顶点
2. 删除该顶点及从该顶点出发的弧
3. 当所有顶点均被访问，则实现了该AOV的拓扑排序
   * 1. 拓扑排序与BFS：

* 两种算法很相似，但拓扑排序将某一顶点入栈的条件是该顶点入度为0；而BFS则需要该顶点邻接自栈内某一顶点
* 对栈或队列的使用，在BFS中，算法需要首先记录某一深度下所有顶点的所有未被访问的邻接顶点，结束当前深度的访问后，在从内存中了解下一深度所包含的顶点，因此BFS需要使用队列来记录待访问的顶点；拓扑排序则在同时存在多个0入度的顶点时，则首先选择不影响算法正确性的顶点，因此拓扑排序即可使用队列，也可使用栈
  1. AOE活动网络
     1. AOE活动网络：在有向无环图中用有向边表示一个工程中的各项活动，由有向边的权值表示活动的持续时间，顶点表示事件(event)，叫做用边表示活动的网络(Activity On Edges)
     2. AOE网络包含所有工程的工时及不同工程间的先后次序
     3. 开始点/源点：source，整个工程只有一个开始点，id=0
     4. 结束点/汇点：sink，只有一个结束点，od=0
     5. 不具有先后次序关系的活动可以并行进行
     6. 完成工程需要所有路径上的活动都完成才可达到

因此：

完成整个工程所需的时间取决于从源点到汇点最长路径长度

* + 1. 关键路径：critical path，拥有最长路径长度的从源点到汇点的路径
  1. 求解关键路径
     1. 关键活动：critical activity，不按期完成就会影响整个工程完成的活动
     2. 事件标记为E0 ~ E­n-1
     3. 关键路径上的所有活动都是关键活动
     4. 关于关键活动的时间量，事件Ei
* 事件Ei最早可能开始时间：**Ee[i]**
* 从源点E0到Ei的最长路径长度
* 事件不可能在之前的工程未完成的情况下开始，因此需要等待从源点到自身的最长工程路径完成
* 因此求最早可能开始时间不同于求最短路径
* 称为“最早可能开始”是因为不可能比这个时间更早，实际工程中由于延误工期等原因，事件发生时可能比Ee[i]更晚
* 事件Ei最迟允许开始时间：**El[i]**
* 在保证En-1在Ee[n-1]时刻完成的前提下，事件Ei的允许最迟开始时间
* 等于Ee[n-1]减去从Ei到En-1的最长路径长度
* 活动ak的最早可能开始时间：**e[k]**
* e[k]等于弧的起点Ei的最早开始时间Ee[i]
* 活动ak的最迟允许开始时间：**l[k]**
* 设活动ak在有向边<Ei,Ej>上，l[k]是使后继事件Ej在其最迟允许开始时间El[j]时发生，活动的开始时间

dur项是该活动所需的时间，即该有向边的权值

* + 1. 松弛时间：slack time，l[k]-e[k]，活动最迟与最早开始时间的时间余量
    2. 关键活动：critical activity，松弛时间为0的活动
    3. 各项关键时间量的递推公式：
* Ee[k]
* V是所有指向顶点Ei的有向边<Eh,Ei>的集合
* 最早可能开始时间需满足：事件之前的所有工程均已完成
* E[0]=0
* El[i]
* 最迟允许开始时间需满足：事件结束后，再经过最长工程链到达汇点，不会使事件En-1在Ee[n-1]之后发生
* 假设事件Ej之前发生了延误，则最多使Ej延误到什么时候发生，也不会让最后的工程延迟结束。（Ej之前延误达到极大值）
* El[n-1]=Ee[n-1]
* 以上两个递推公式的计算必须分别在拓扑有序和逆拓扑有序的前提下进行
* 逆拓扑有序：首先输出出度为零的顶点，然后以相反的次序输出拓扑排序序列
* 也就是说，在计算Ee(i)时，Ei的所有前驱顶点Ej的Ee[j]均已求出；反之，计算El[i]时，Ei的所有后继顶点Ej的El[j]均已求出的条件下，才可以计算
* e[i]，l[i]

设活动ak对应带权有向边<Ei,Ej>，持续时间dur(<Ei,Ej>)，则有：

1. 树和图的生成树
   1. 树：不存在回路的无向连通图
   2. 树的定理
      1. 用一条边连接树中的任意两个顶点都会产生一个回路
   3. 森林：forest，包含多个树无向图

森林是非连通图

* 1. 生成树：spanning tree，
     1. 无向连通图G的一个子图如果是一棵包含G所有顶点的树，则该子图成为G的生成树。
     2. 生成树是连通图的极小连通子图。极小指：若在树中任意增加一条边，则将出现一个回路；若去掉一个边，则会变成非连通图
     3. 设G为有n个顶点的连通图，则其生成树有n个顶点，n-1条边
     4. 不同遍历方法会有不同的生成树
     5. 不同顶点出发会有不同的生成树
  2. 最小生成树
     1. 权：weight，各边的权值的总和称为生成树的权
     2. 最小生成树/最小代价生成树：minimum spanning tree/minimum-cost spanning tree，从无向连通图生成的使权为最小的生成树
     3. 所有边的权值相同时，所有生成树都是最小生成树
     4. 所有边的权值不同时，最小生成树只有一个。证明：
* 假设有两个最小生成树A和B
* 由于A和B是不同的最小生成树，那么A中总有一个或多个边不在B中，设这些边的权值最低的那一条为ek
* 由于B是一个最小生成树，由定理：

“用一条边连接树中的任意两个顶点都会产生一个回路”

可知必然包含一个环C

* 因为环C中存在一条边em它的权值比ek要大
* 因此用ek替代em，B成为了一个拥有更小权值的生成树。这和B是最小生成树相矛盾，所以不可能存在两个最小生成树
  + 1. 生成准则：
* 只利用原图中的边来构造树
* 必须使用且仅适用n-1条边来连接网络中的n个顶点
* 不能使用产生回路的边
  1. 生成树算法
     1. 克鲁斯卡尔算法 – Kruskal
        1. 步骤：