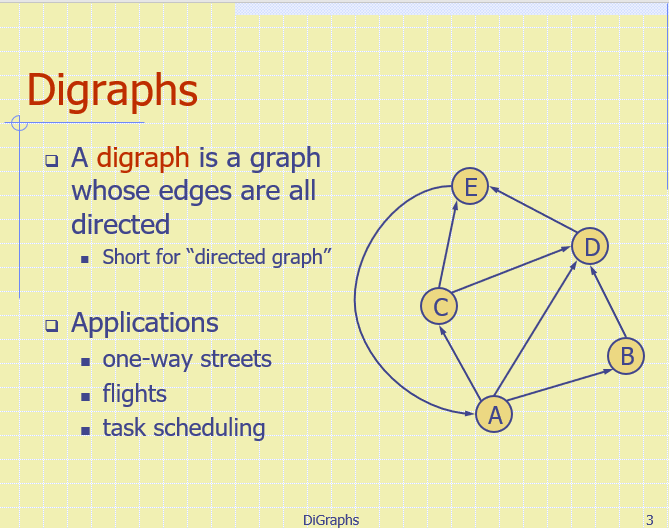
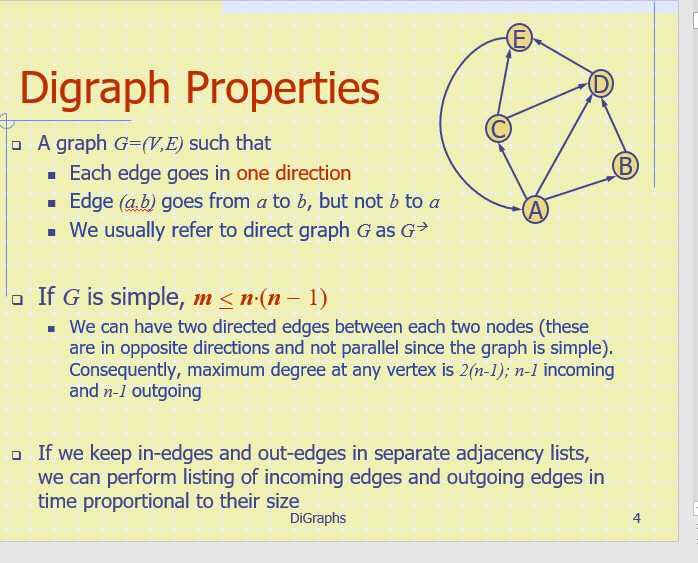
dIgraph:每个edge有方向





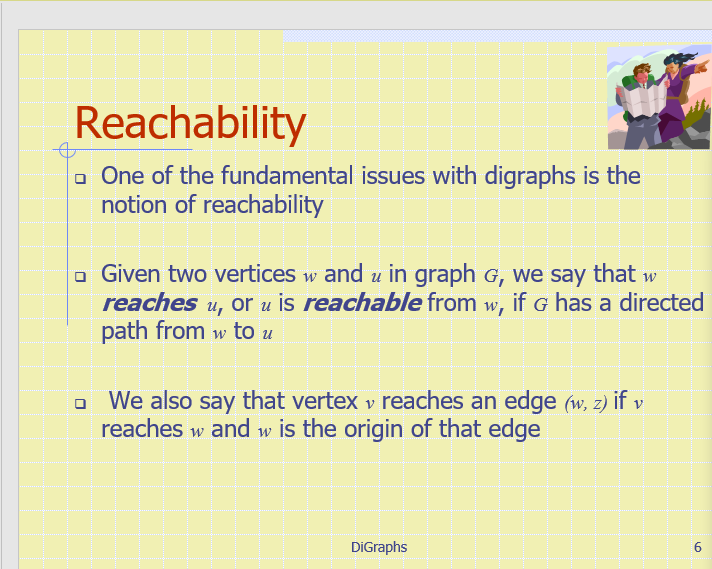
我们通常将graph G 写成G->来表示她是digraph

如果G是simple的，那么m<=n n-1

两个点之间我们能有两个directed edge，一正一反， 那么作为结果，一个vertex的最大degree是2 （n-1），N-1输入，N-1输出

然后一共n个点2n(n-1),最后除以2因为重复计算

如果我们把in-edge 与out edge存储在分别独立的adjacency lists中，那么我们可以按时间比例列出incoming edge与outgoing edge,这个比例与size成正比



如果有两个点wu,并且存在一个directed path从w到u，那么我们可以说w reaches u或者U is reachable from w

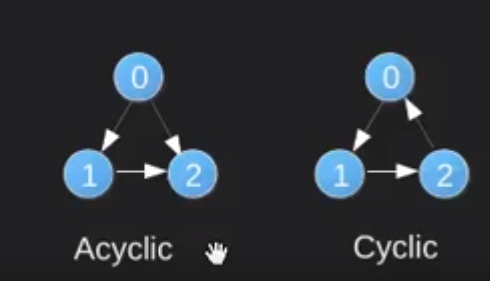
我们也说v reaches一个edge(w,z)如果v reaches w而w是edge源头

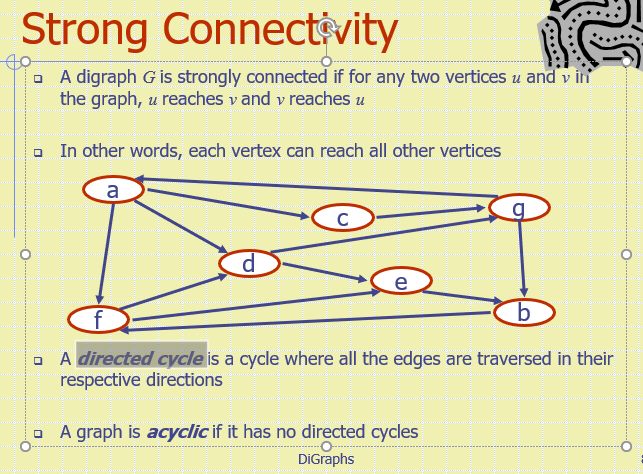
STRONG CONNECTIVITY

如果任意一个点都能reach到其他点，那么这个digraph is strongly connected

direct cycle:就是沿着directed方向，可以回到原点

acyclic:没有directed cycles



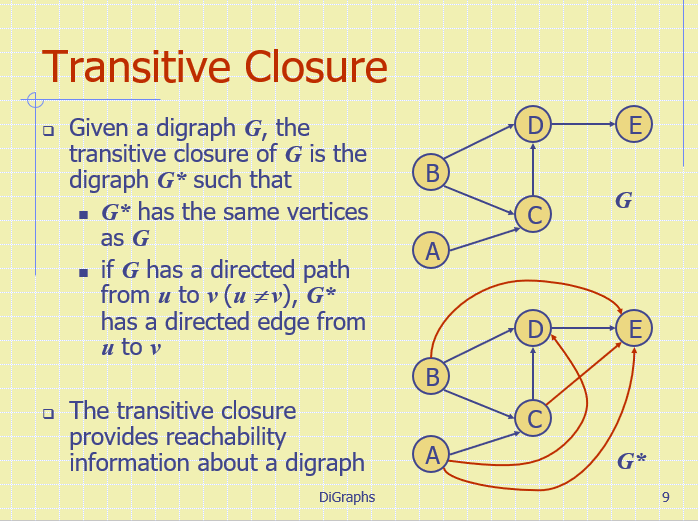


transitive closure

给你一个digraph G,gransitive closure of G 就是另一个DIGRAPH G\*

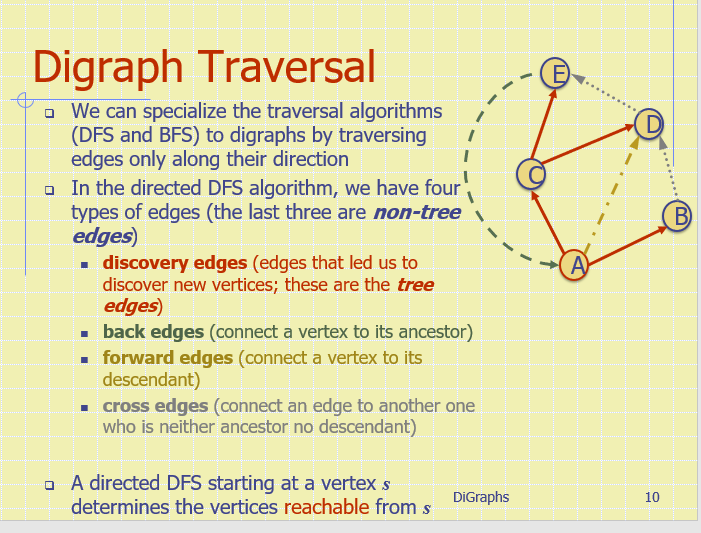
G\*拥有同样的vertice

如果G在uv之间有一条Path，那么G\*就给你直接装一edge



digraph traversal

traversal算法DFS/BFS在digraph中只能沿着他们的direction



directed DFS有四种edge

discovery edge(让我们发现新edge，他们是tree edge)

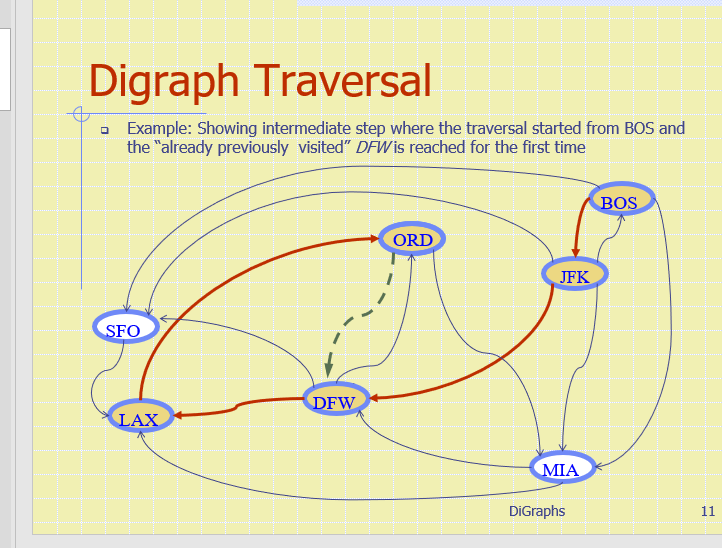
back edge)（把vertx与他的祖宗相连）

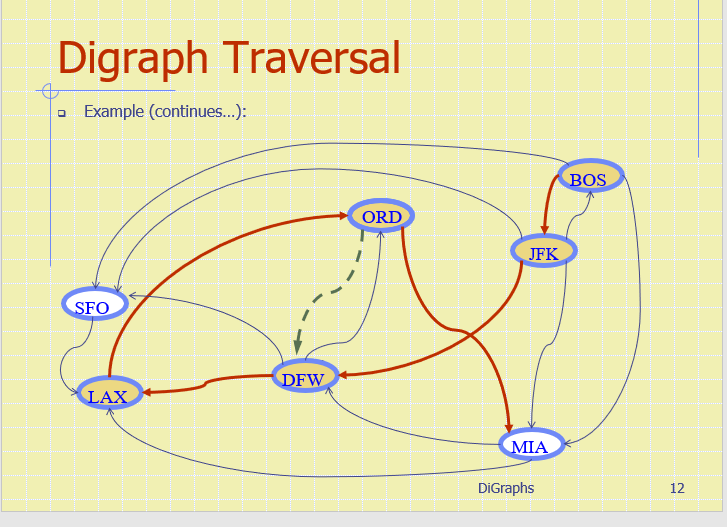
forward edge(把vertex与他的子孙相连)

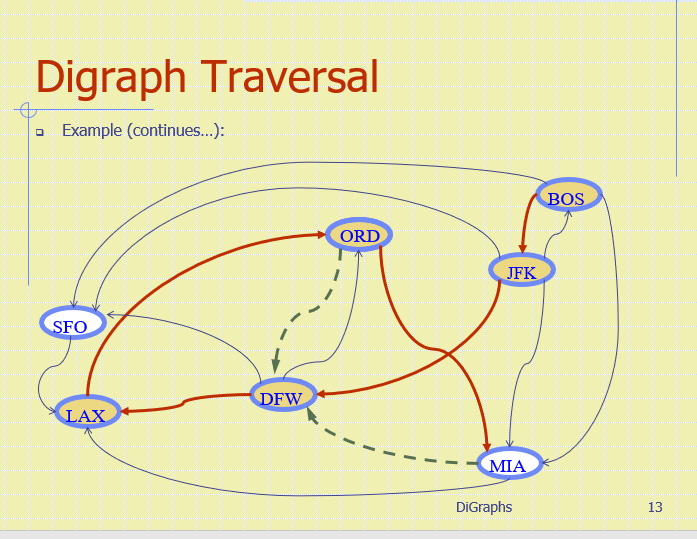
cross edge(练出一条线当既不是祖宗又不是子孙)

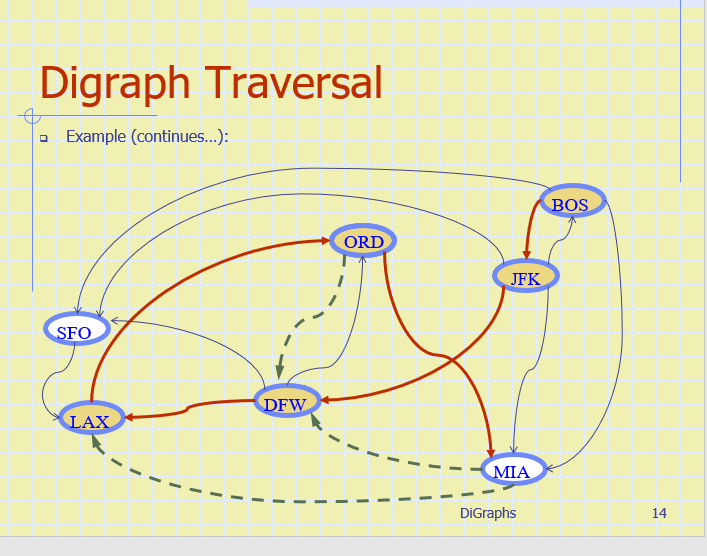
directed DFS假如从点s开始，会找到所有reachable from s的点

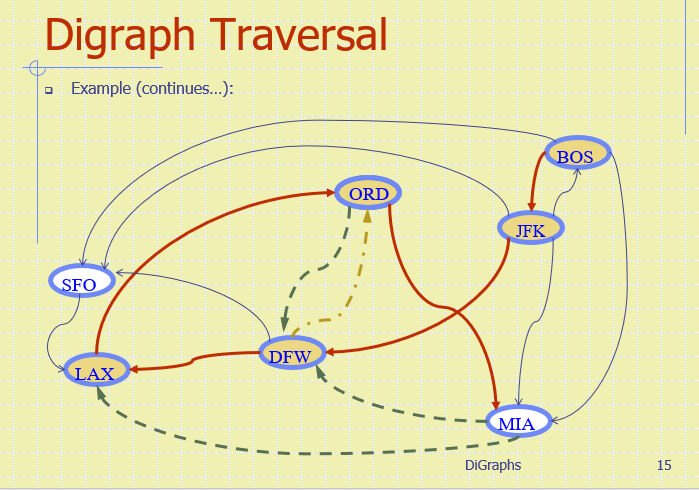
例子，展示从BOS开始的 dfs，我们现在已经到了ORD



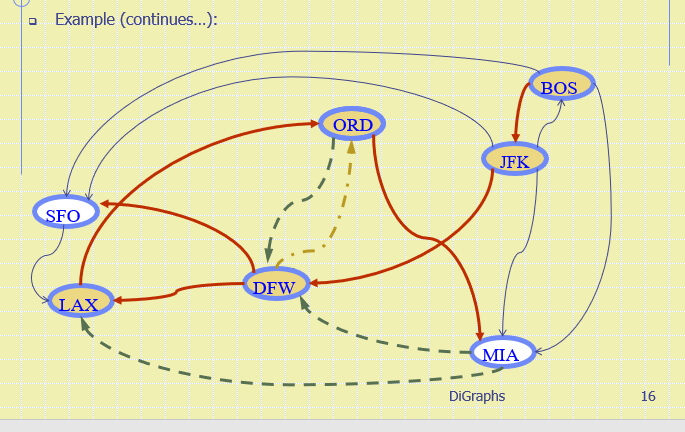


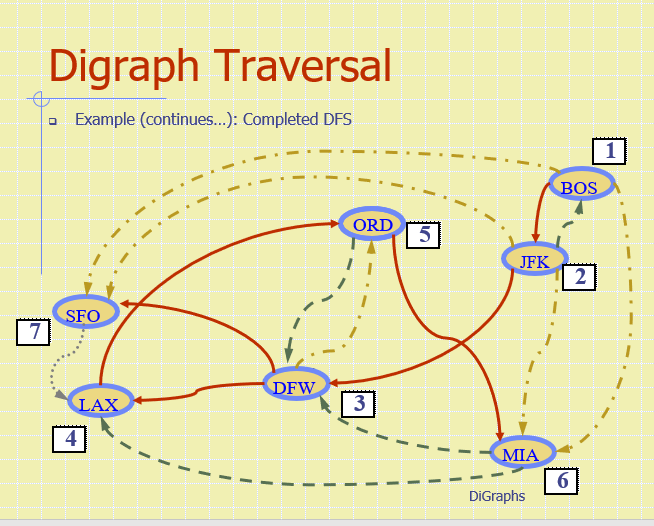






黄线代表他和他的子孙





从vertex s开始，DFS将visit所有reachable vertices

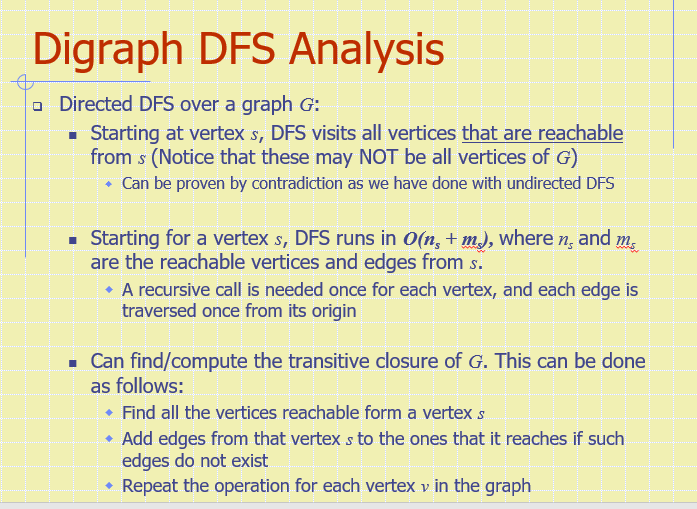
从vertex s开始，DFS将用时O（ns+ms）,ns ms都是s的reachable

可以找到/得到 G的transitive closure（所有reachable点都链接·上edge）

找到所有reachable vertices

在s与这些点之间加上edge如果原本没有edge

对这个graph每个点做以上操作



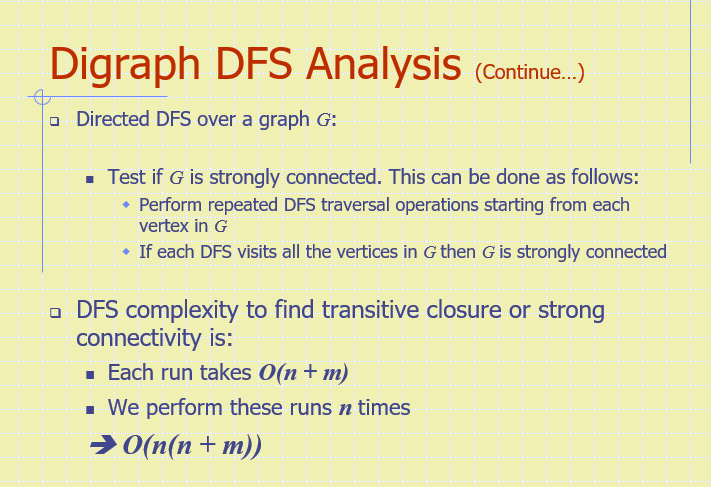
检查graph G是不是strongly connected.

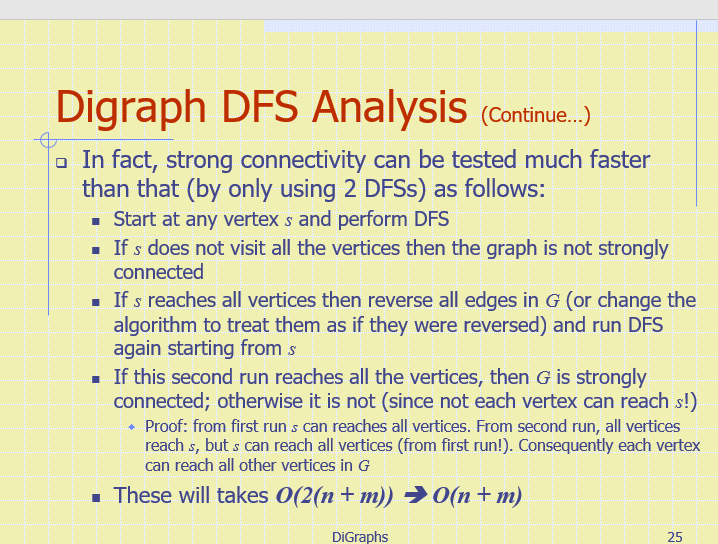
1.对每个verted都作为起点，使用DFS traversal

2.如果每个DFS都visiti到了所有点，那么G就是strongly connected

每次test O(n+M)

总共run n次，O(n(n+m))





事实上，测试strong connectivity比这更快

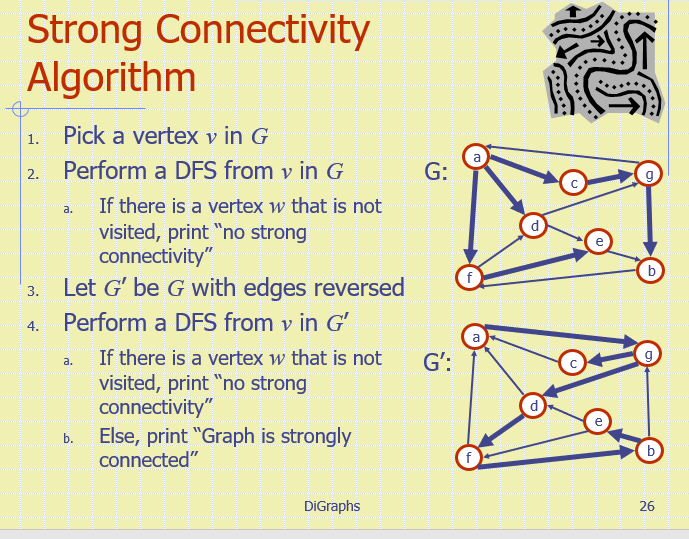
任意一个起点为s，DFS

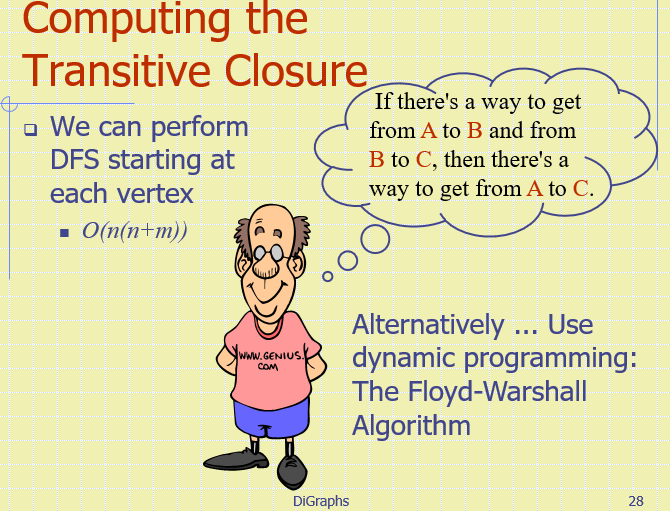
如果没有visit所有点，不是strongly converted

如果s visit了所有点，那么就改变所有edge的方向，（或者改变算法，让他反着识别edge）。再以S为起点，DFS

如果还是reach 了所有点，那么G就是strongly connected

用时为O（n+m）

以点A为例  




计算transitive closure

我们可以用DFS来检查每个点n(n+m)

但我们也可以用floyd-warshall algorithm

假设G是有n个点与m个edge组成的GRAPH

为了得到transitive closure

初始化G0=G

给各个顶点随意编号成V1 TO VN

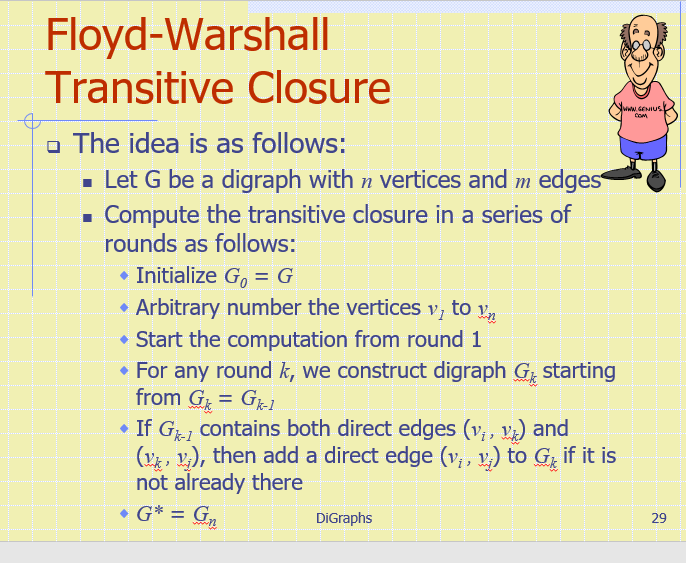
从round 1开始

对于任意round k，我们构建一个digraph Gk 让Gk=Gk-1

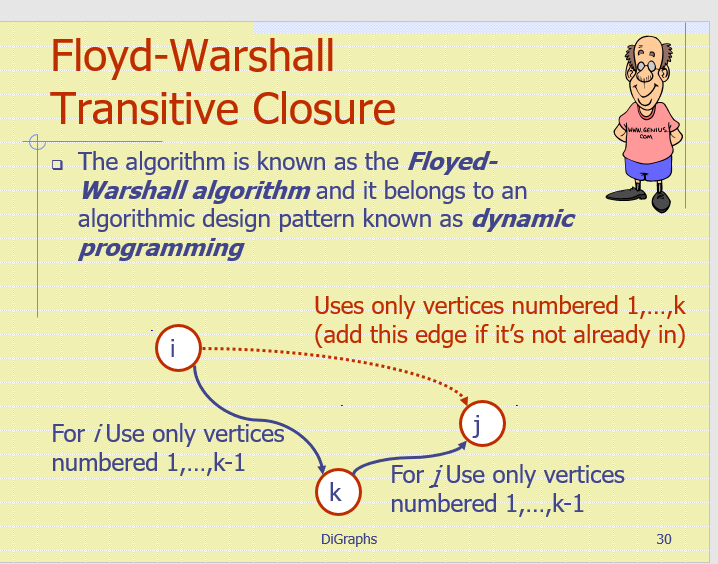
如果Gk-1包含一个vi,vk和vk,vj，那么我们就在Gk加入一个vk,vj的路径

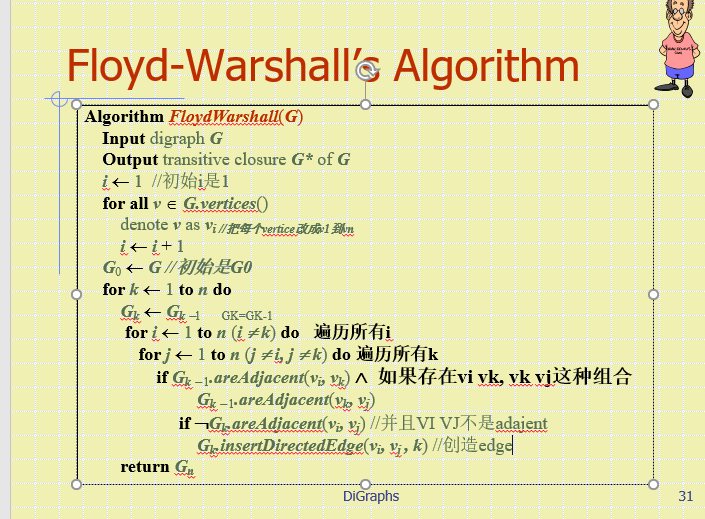
i与J需要遍历所有n

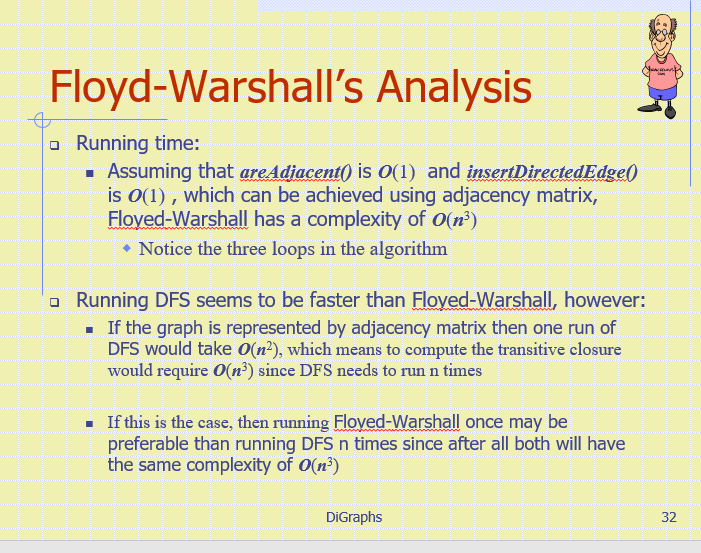
最终G\*=Gn



这叫做floyd-warshall algorithm，是动态program的一种







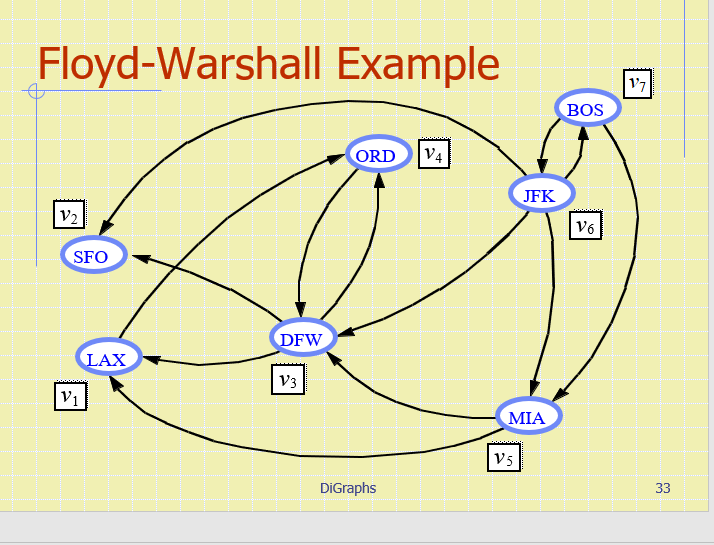
floyed-warshall的耗时是On^3

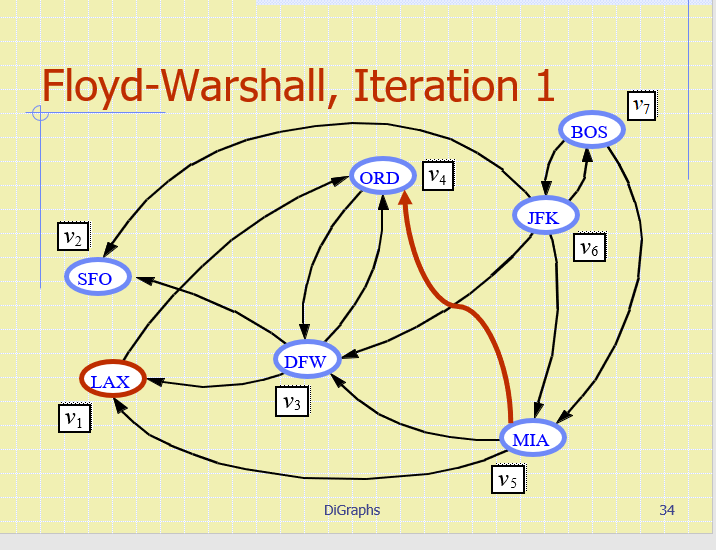
DFS看起来比floyed更快，然而：

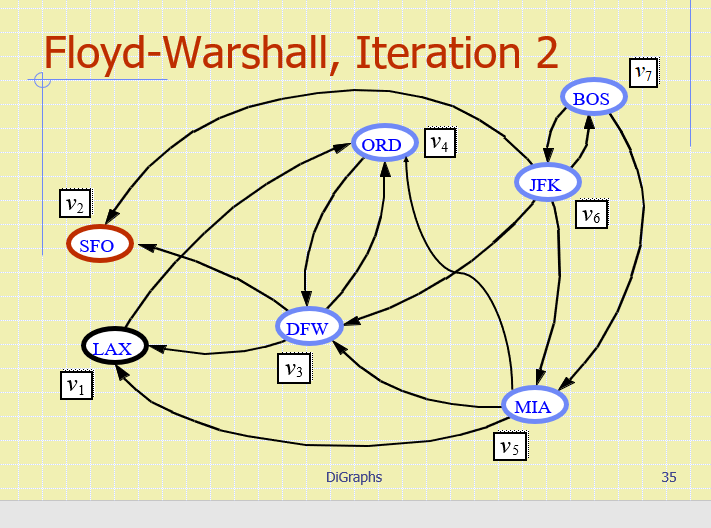
如果GRAPH以adjacency matrix表示，运行一次DFS需要On^2

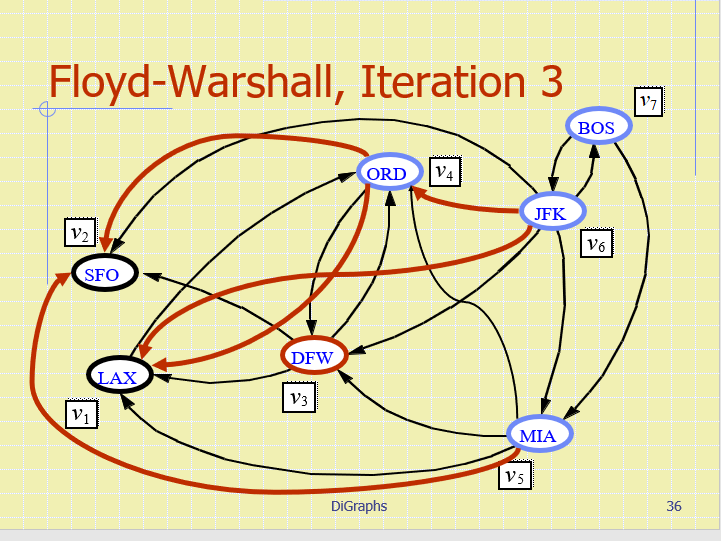
n个点需要On^3

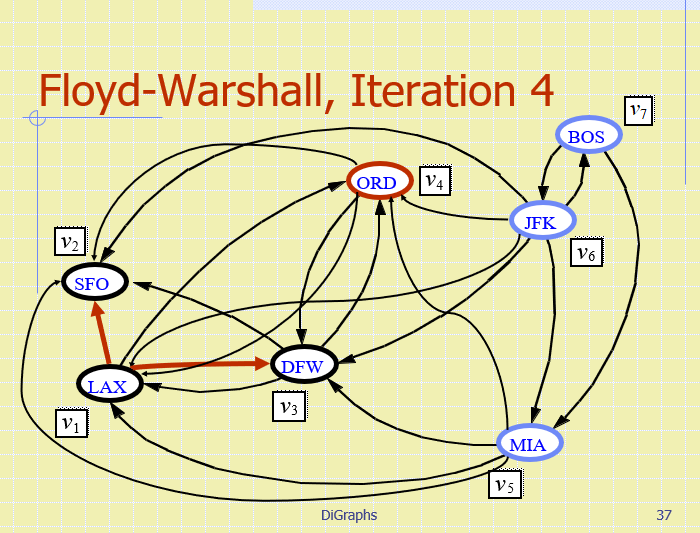
这种情况，floyed warshall更偏爱

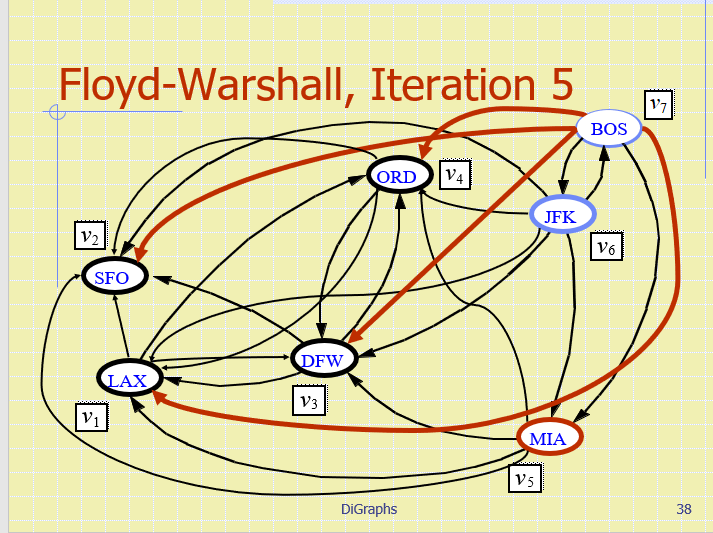


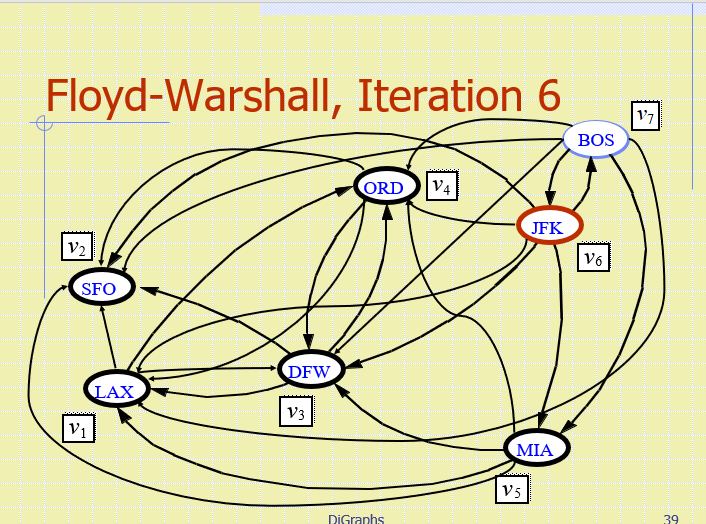


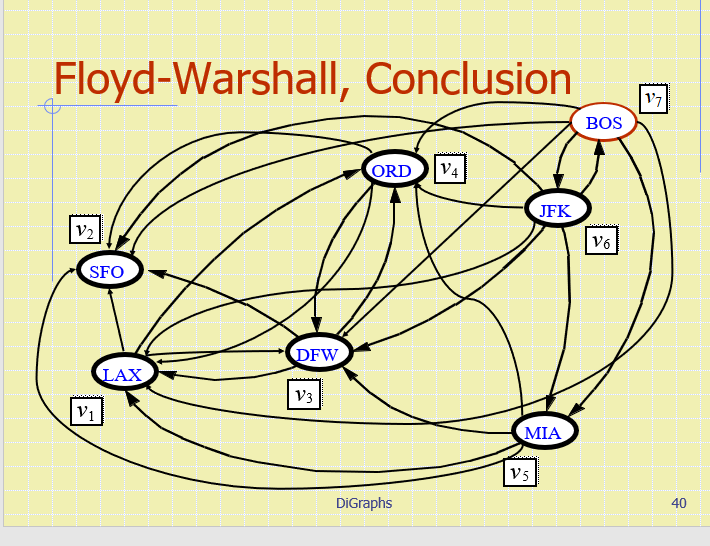








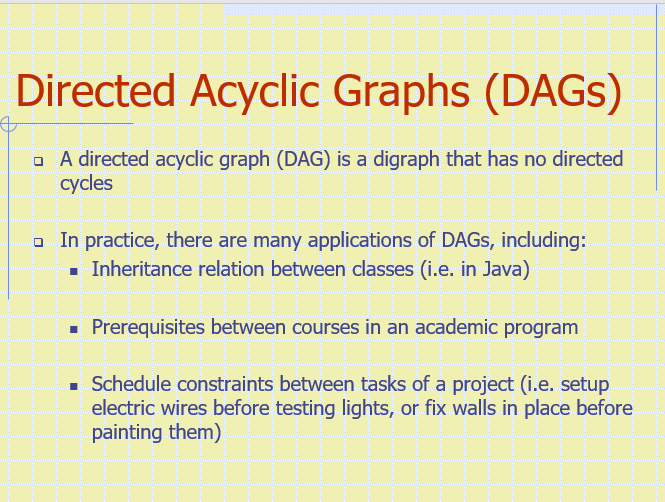




换句话来说，对每个v1,vn我们找有没有以他为中间站的两个path，另外两个端点没有连线就连上

Directed Acyclic Graph (DAG)

一个DAG指的是没有directed cycles 的digraph

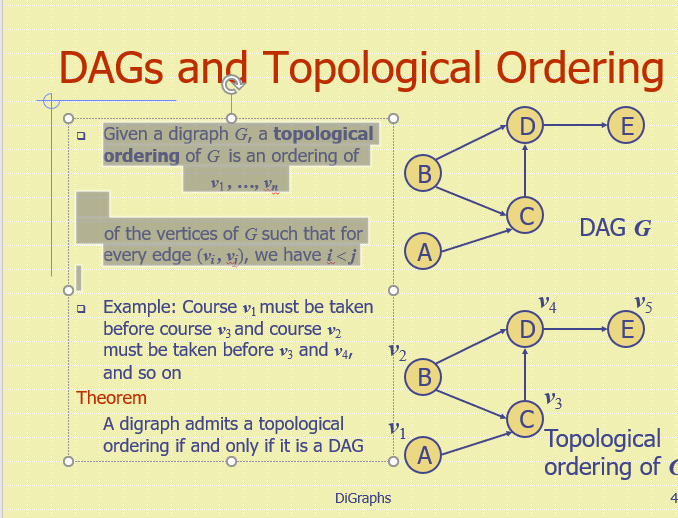


DAG and Topological Ordering DAG与拓扑排序

给你一个digraph G, topological ordering of G指的是v1,v2..vn这样的order

给G的vertice编上码，让每个edge(vi,vj)我们有 i<j

理论：digraph可以接受topological ordering当且仅当他是一个DAG时



Topological Sorting的算法

假设一个DAG G

因为G是acyclic的，那么必然有一个vertex v他的incoming edge=0

如果v被remove了，那么剩下的graph必然仍然是acyclic的，那么这时存在另一个vertex w，他的incoming edge=0

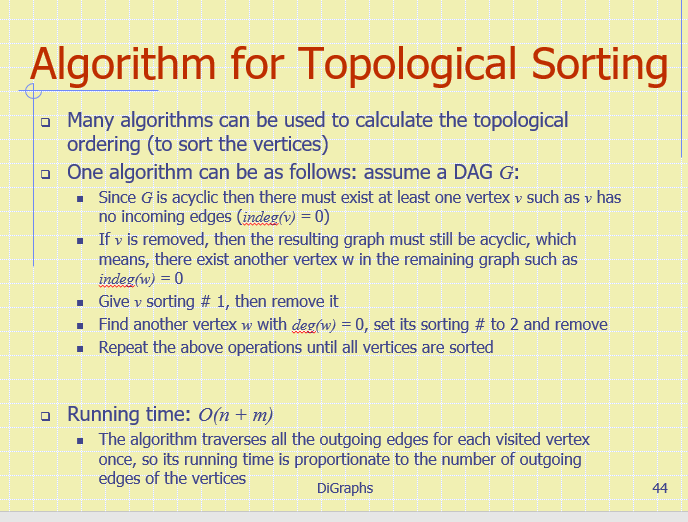
v在sort中排v1，移去

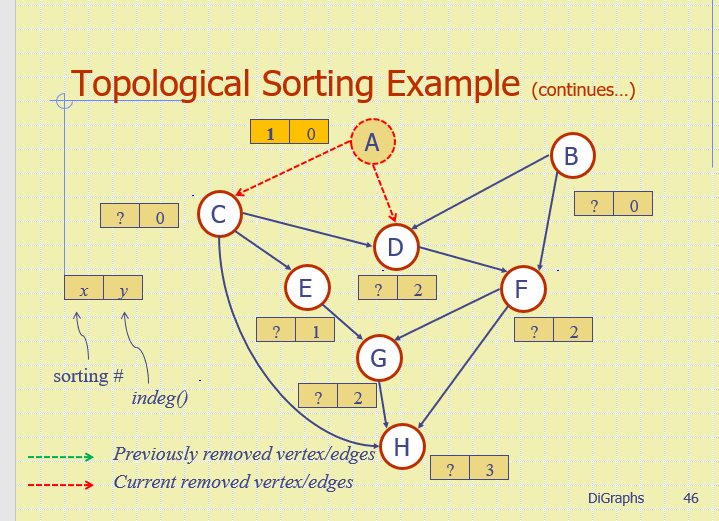
w在sort中排v2，移去

重复直到所有vertice都在sort后有位置

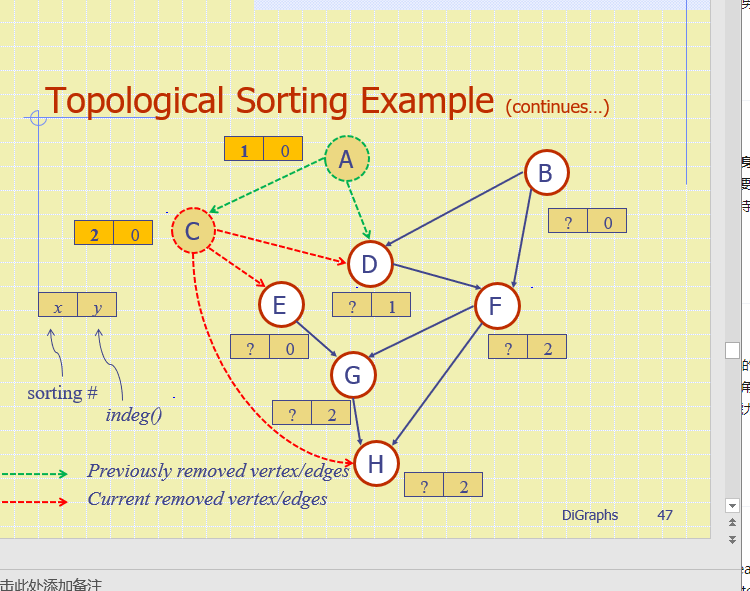
用时O（n+m）

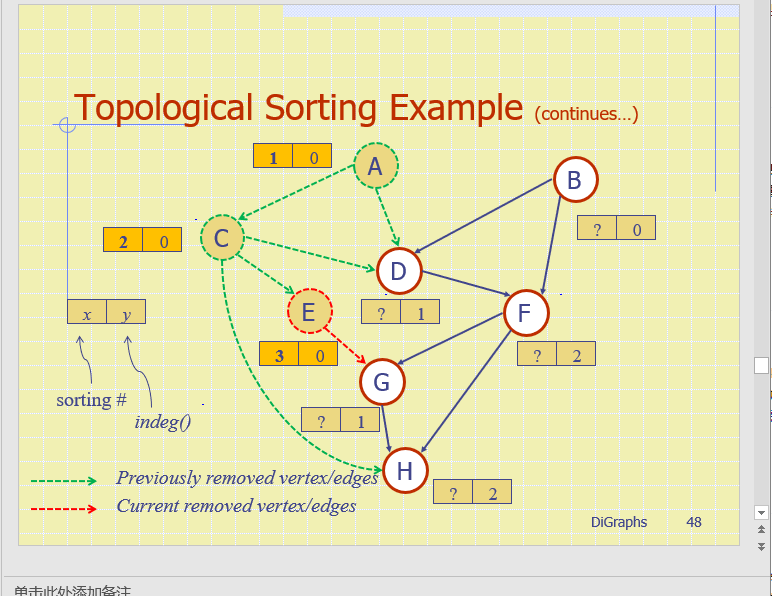
这个算法会遍历所有的vertex与对应的outgoing edge

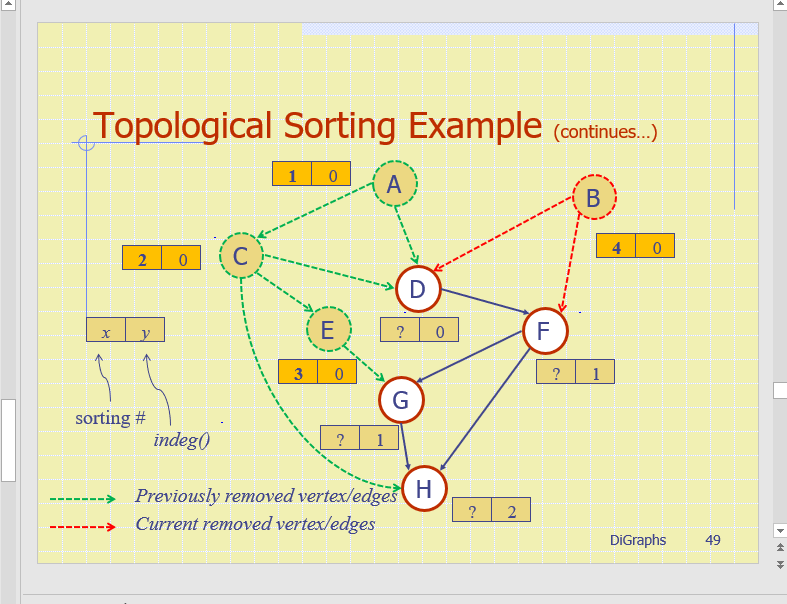


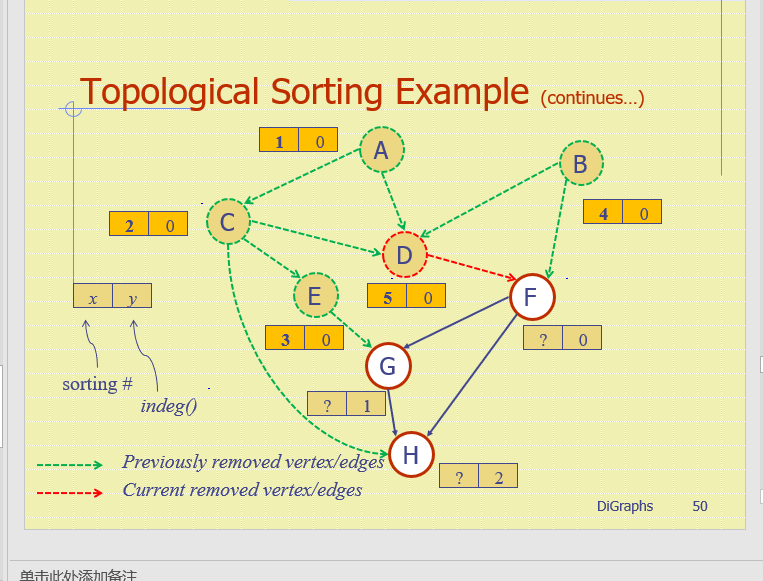


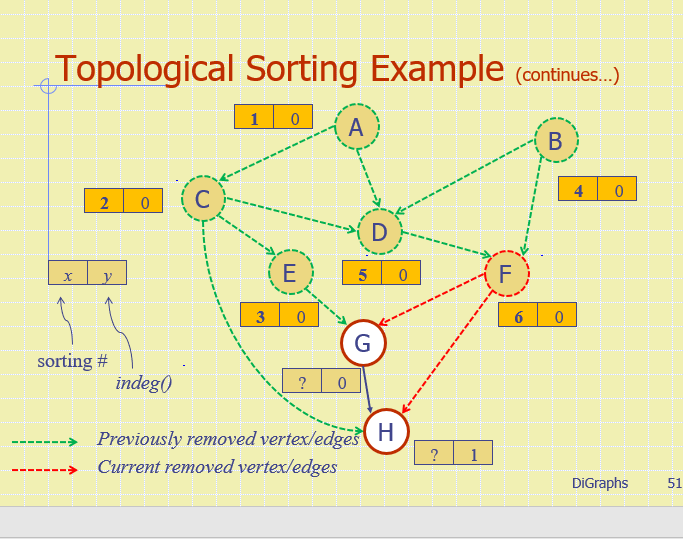
第一个没in coming edge的

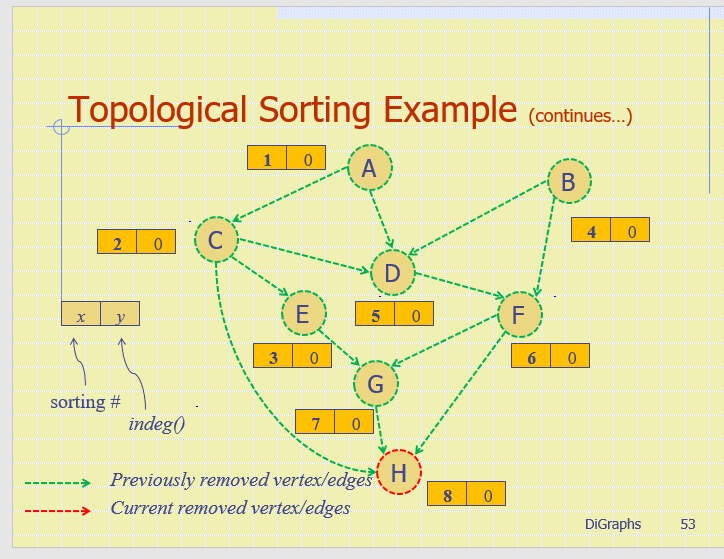












别的topological sorting的算法

找到没有outgoing的edge，remove，设定为n  
第二个，remove，设置为n-1

无限循环

用时还是O（n+m）

