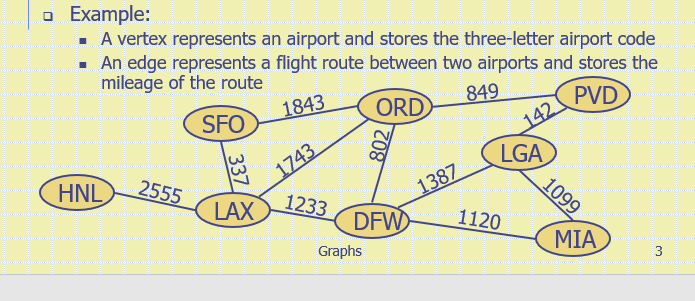
graph就是一组Objects与他们之间的连接组成的collection，这些objects叫做vetices顶点，vertice之间的连接叫做edges

因此一个graph可以被看作是一个pair (V,E)

V是vertice的集合

E是pairs of vertice的集合(一对vertice代表一个edge)，叫做edges

vertice与edge本身都只是positions,并且需要存放element/value



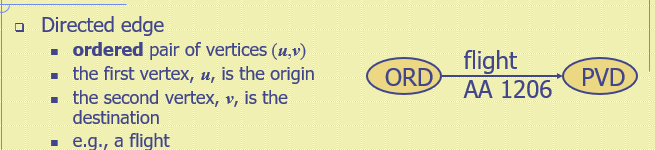
比如，vertex代表一个机场，存储的数据是三字母机场码，edge代表飞行路径，存储数据是里程

edge分类

directed edge

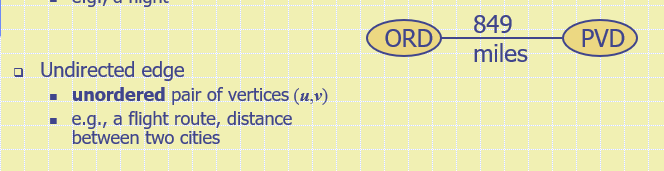
一组有顺序的vertices（u,v）

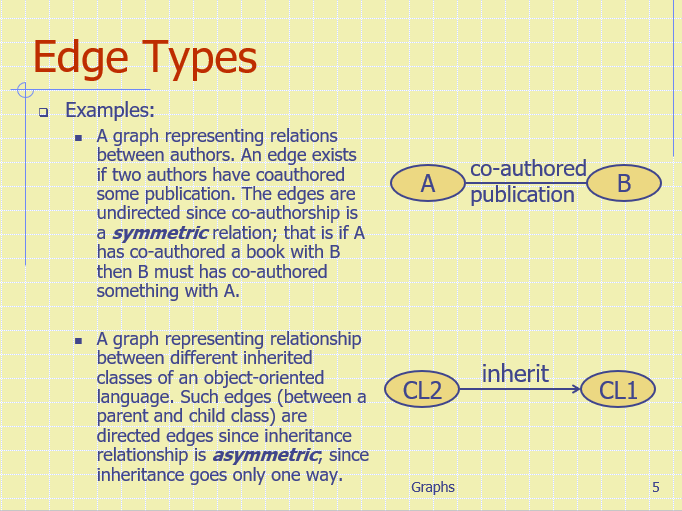
u是起点，v是终点



undirected edge

一组无顺序的vertice（u,v）





例子,A与B和出一本书，那么他们的关系就是symmetric relation，对称关系，所以用Undirected

CL1 inherit了CL2，那么就是不对称的·，因为inherit是单向的

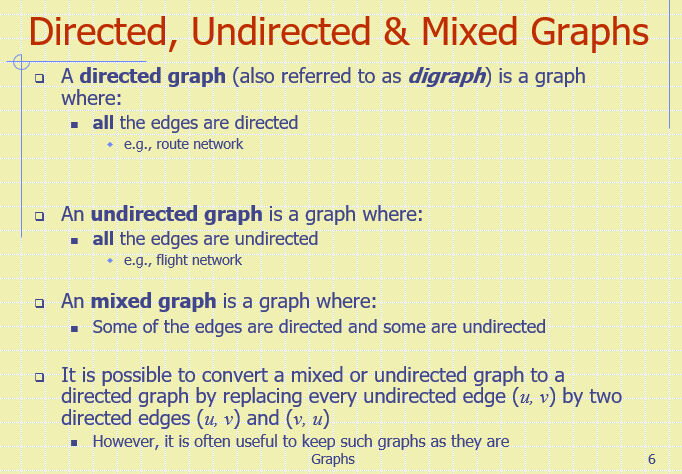
GRAPH分类

directed graph(digraph):所有edge都是directed

undirected graph:所有edge都是undirected

mixed graph:两者兼备

mixed graph 与undirected graph可以转换成directed graph，只要把每一个undirected edge(u,v)替代成两个directed edge(u,v)与(v,u)

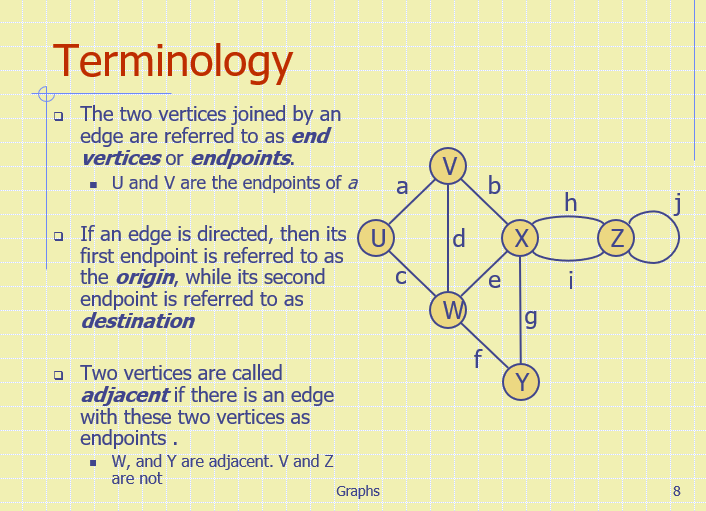


Terminology

edge所连的两个vertice又叫做这个edge的endvertices/endpoints端点

如果edge有方向，第一个endpoint叫做origin,第二个叫做destinition，

如果两个vertices为同一edge的端点，那么这两个vertice叫做adjacent，相临



Degree：一个vertice 为v，那么deg(v)=等于这个v点所连的edge数

in-degree：indeg(v)。指向V的edge数量

out-degree:outdeg(v).由v开始向外指的vertex

graph更倾向于collection而不是set

区别在于collection可以有重复

比如说两个directed edge有着同样的起点和终点，或者有着同样endpoint的undirected edge

这类edge被叫做parallel edge 或者multiple edge

h与i就是parallel edge

self loop: 指一个edge将一个vertex与他自己·相连，j就是一个self loop,注意！self loop指的是edge

Path：

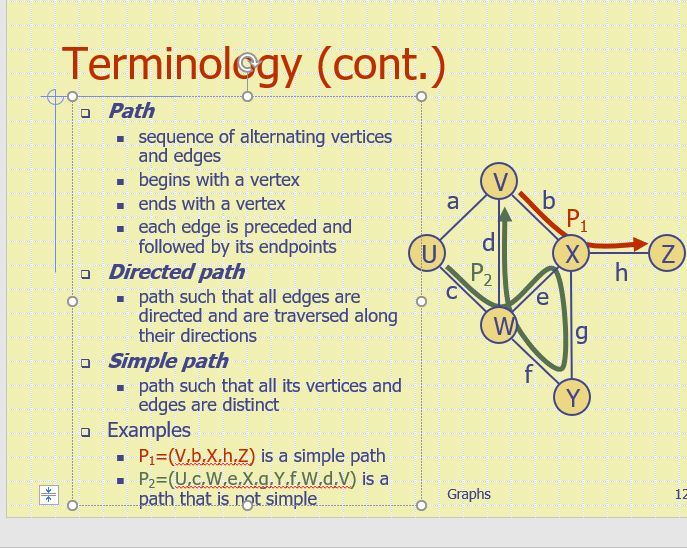
vertice-edge-vertice-edge-vertice-edge-vertice这样结构的的sequence

由一个vertex开始，由一个vertex结束

每条edge的前面后面都有它的endpoint

directed path：每一条edge都是directed并且我们沿着这个direction走

simple path:所有的vertice与edge都是不同的（没有重复vertice或edge）

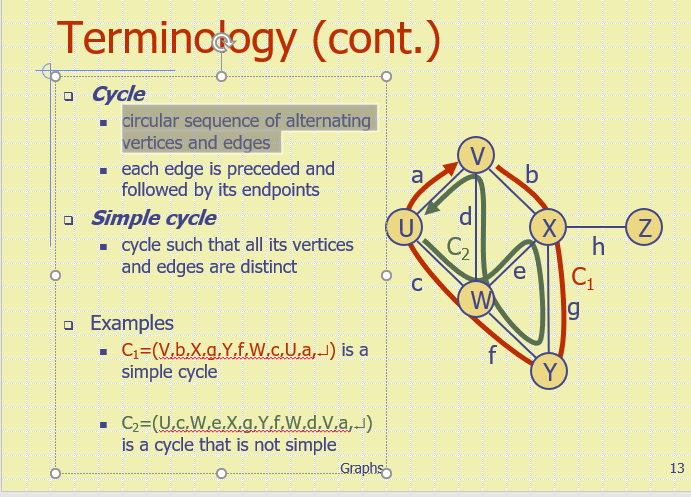


cycle:

vertice与edge交替出现的圆形sequence

每一个edge前后都是endpoint

simple cycle:每一个vertice与edge都没有重复的

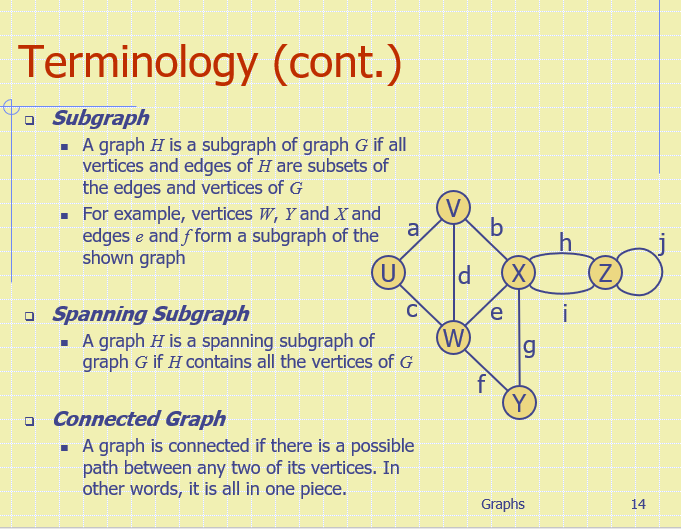


Subgraph

一个graph H是subgraph of graph G当他所有的vertices和eges都是G的vertices与vertices的子集

spanning subgraph:H是G的spanning subgraph当他游了G的所有vertices

Connected graph:一个graph被叫做connected当他的任意两个点之间必存在path，换句话来说，这个graph是一整块



Connected components

如果graph G本身是个disconnected graph(由多块组成)，那么他的最大的connected subgraphs 就叫做connected components

Forest:一个没有cycle的graph，注意forest可能有许多component

Tree

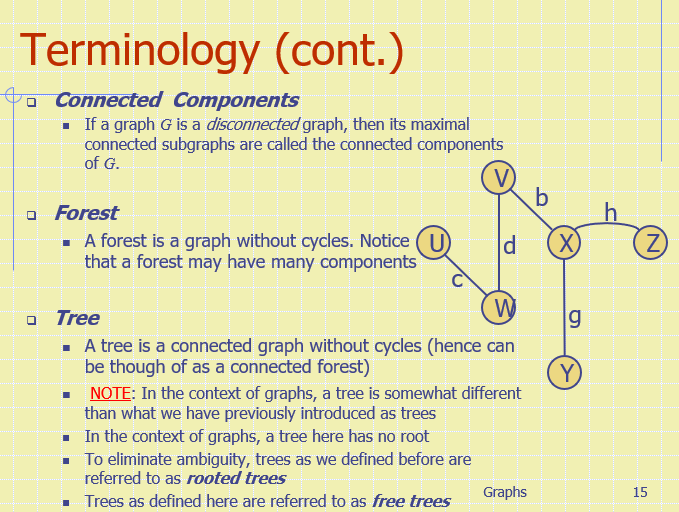
一个tree是一个connected graph没有cycle(因此可以看作是一个connected forest)

注意：在graph的内容中所指的tree，与我们之前的不同

graph的时候，tree是没有root的

之前的tree 叫做rooted trees

这里的tree 叫做free trees



Spanning tree

一个spanning tree T of a graph G就是这个T不仅是G的spanning subgraph(包含全部端点)，而且还是个free tree

1.包含全部端点

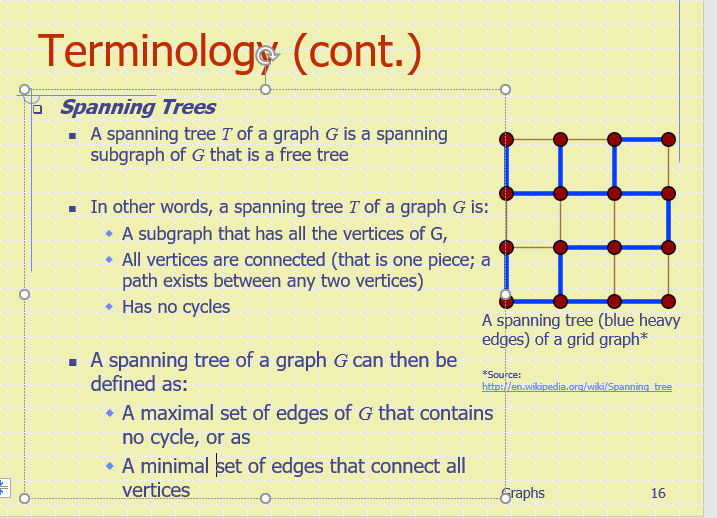
2.任意两个点之间有path，是One Piece

3.没有循环

Spanning tree of a graph G也可以描述成：

1. ，不包含cycle所能达到的最大set of edges

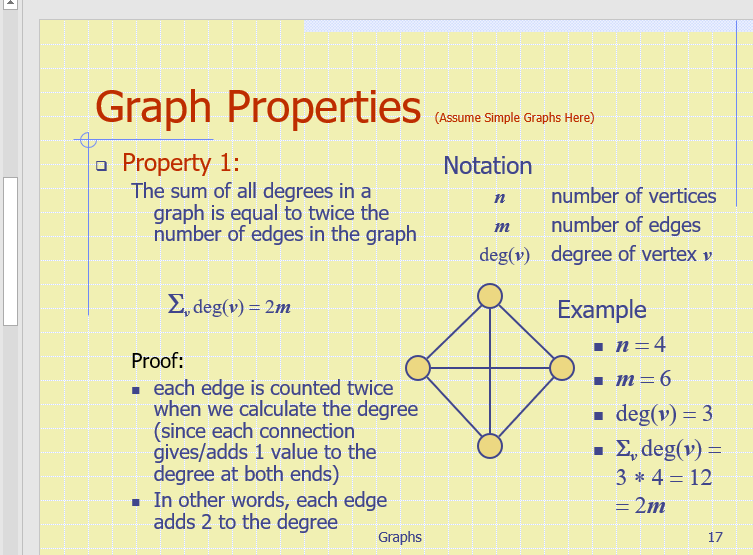
2.包含所有点所用的最小set of edges

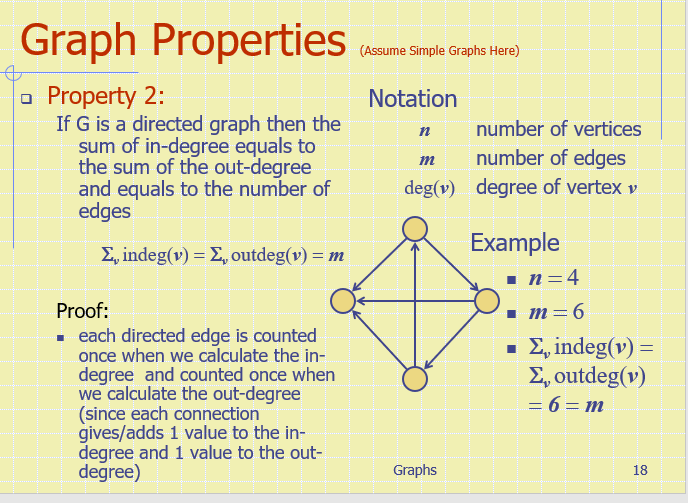


Graoh性质

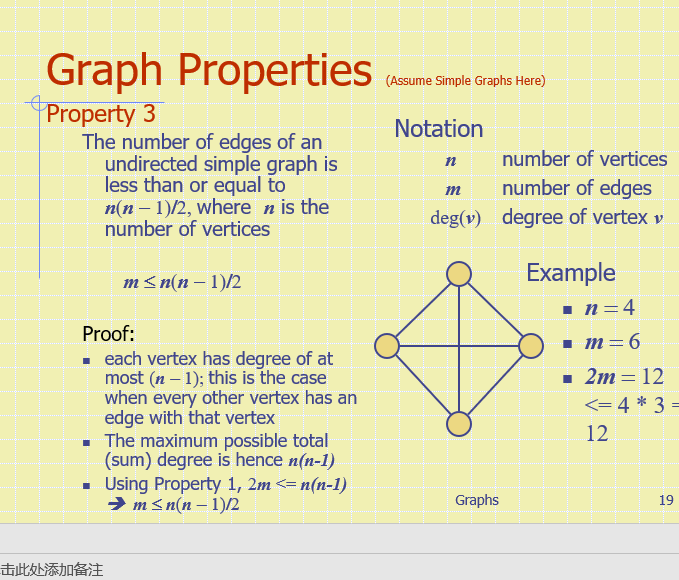
所有的点的degree加起来最终等于两倍的edge

因为每个edge被他的前后两个点各统计一次，最后除以2就完事儿了





如果G是一个directed grath，那么他的indegree等于他的outdegree等于他的edge



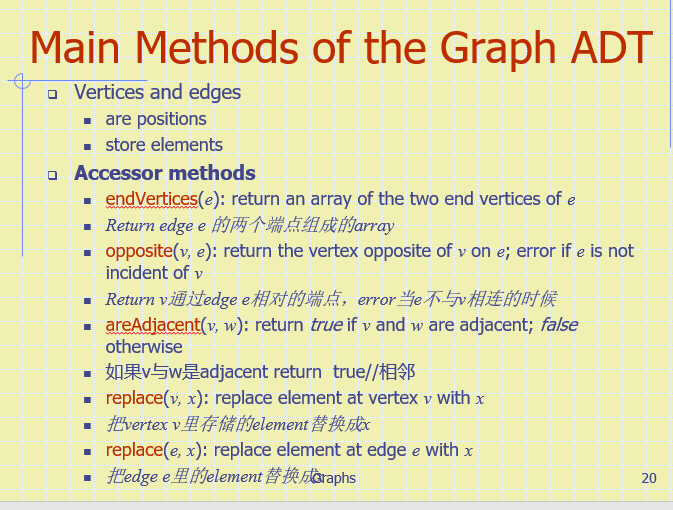
性质三：undirected simple graph 小于等于 n（n-1）/2,simple代表着没有重复edge

每个vertex最多有n-1个edge，（和其余所有点相连）总共有n个vertex，那么总degree=n(n-1)

再用性质1，实际edge等于总degree/2

GRAPH ADT的主要method

vertice 与edge都是positions ,并且存储element



三种最常见的表示graph的方法

1.edge list structure

2.adjacency list structure

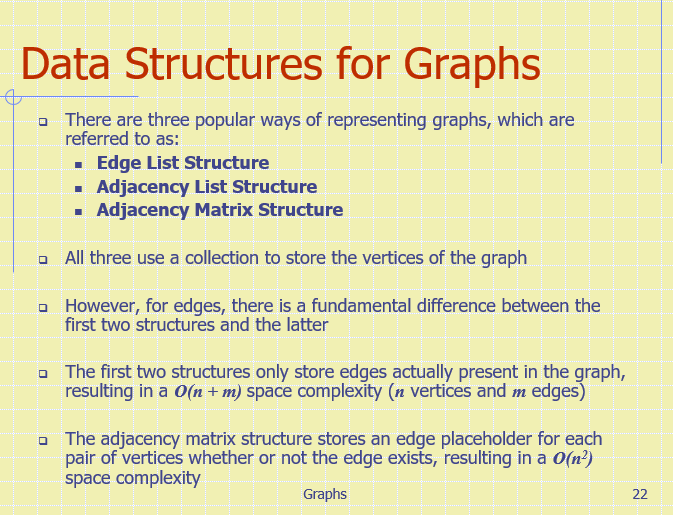
3。adjacency matrix structure

这三种方法都是存储vertice的collection

然而对于存储edge,前两种与后一种有着本质上的不同

前面两种只存graph里真实出现的edge，最后会占用O（n+m） space complexity(n个vertice与m个edge)

adjacency matrix structure 存储了一个edge placehoder这个placeholder存储了每一种可能的edge，无论存在与否，于是最终spacecomplexity是On^2



Edge List Structure

在Edge List Structure中

每一个vertice与edge都被记录成objects

vertice objects被存储在 collection/sequence V里，edges被存储在另一个collectionE里

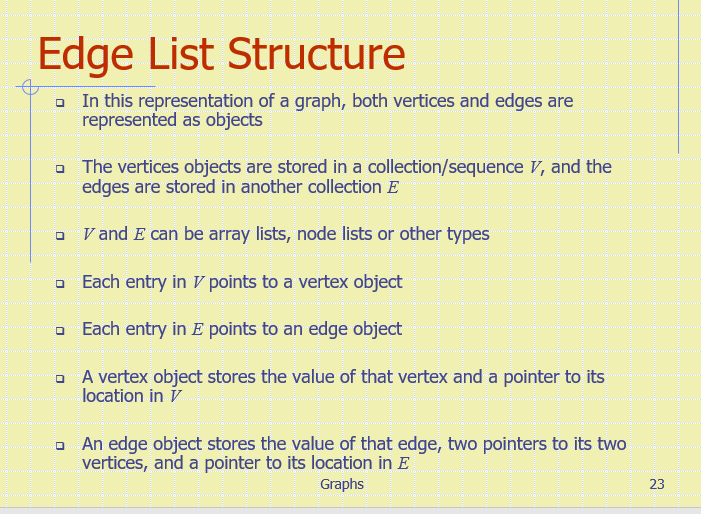
VE可以是array lists,node list.....

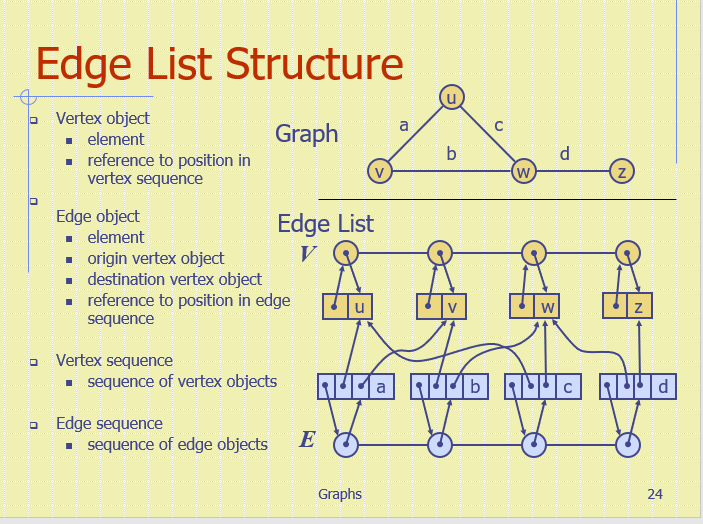
每一个V中的entry都指向一个vertex object

每一个E中的entry都指向一个edge object

一个vertex object存储了 vertex的值与他在V中位置的Pointer

一个edge object存储了 edge的值与他在E中位置的Pointer





Vertex object:

1.本身所存element 2.他们在vertex sequence中的position

Edge object

1.本身所存element 2.指向起点vertex object, 3.指向终点vertex object， 4.他们在edge sequence 中的Position

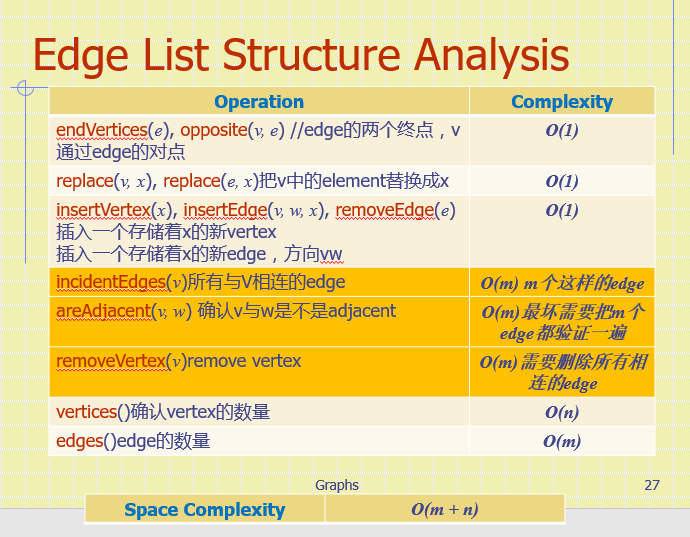
Vertex sequence

由vertex objects组成的sequence

Edge sequence

由edge objects组成的sequence

Edge List Structure 容易Implement,但有很多限制



Adjacency list structure

adjacency list structure包括 edge list structure拥有的所有 结构元素

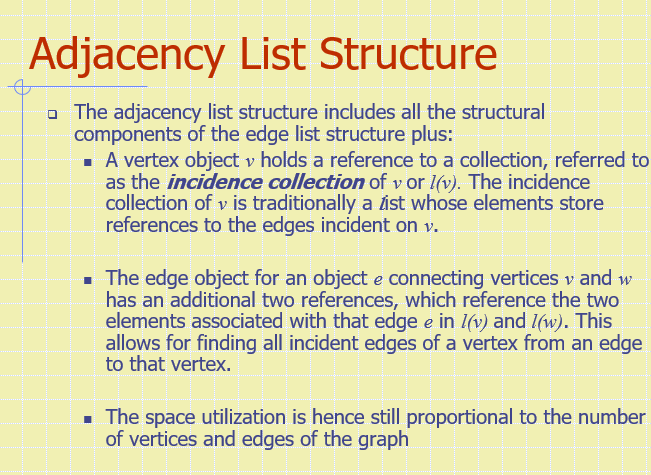
但是还要加上

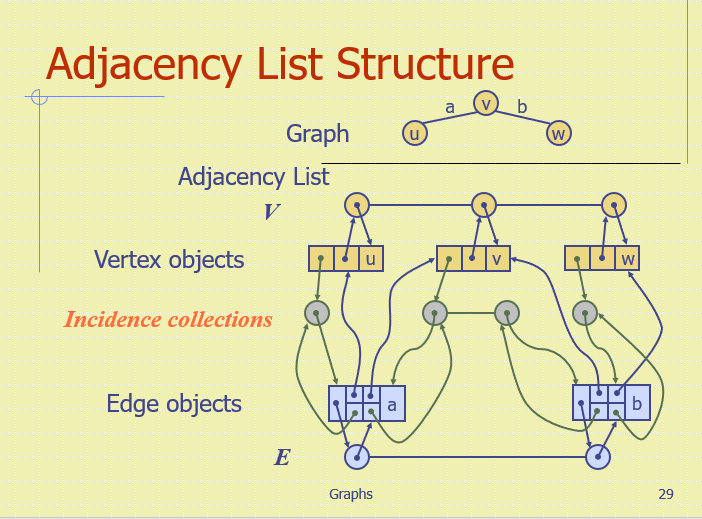
一个vertex object v在原来基础上 需要加上对一个特殊collection的reference，这个collection叫做v的incidence collection，记录着所有与v相连的edges 的reference//注意记录的是reference而不是直接edge

l(v):v的incidence collection

一个edge object e，假设他连接了v和w，那么需要两个额外的reference，指向l(v)与l(w)。这样我们就可以通过vertex的一条edge找到其他的incident edge

因此，空间利用率仍然与vertice与edge的数量成比例





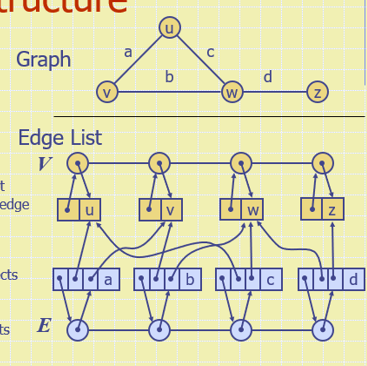
区别在于vertex现在多了一个对incidence collection 的node

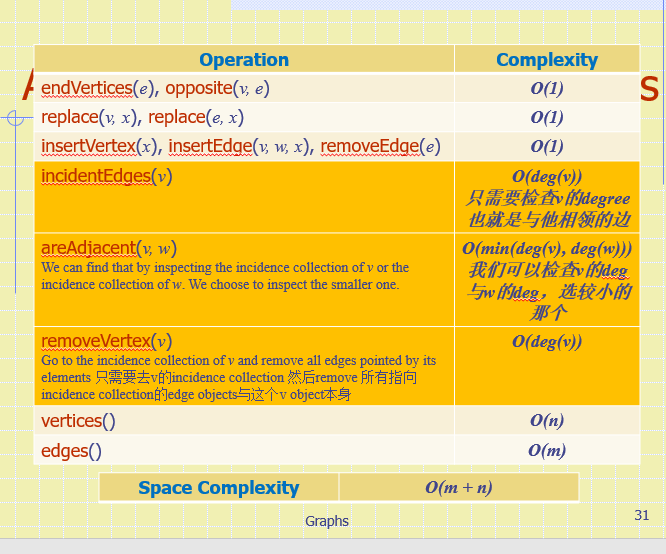
incidence collection存储了所有他incident edge的路径，以u为例，他只有a，那么最左边的incidence collection就指向a，一个小黑点代表一个sequence里的小元素

比如中间的两个黑点，其实还是一个sequence，

edge object存了到他两个端点的incidence collection，

“这样我们就可以通过vertex的一条edge找到其他的incident edge”解释：比如说我们a到v的l(v) reference，进入了这个sequence后，我们可以自然而然的找到第二个黑点，也就是v到b的edge



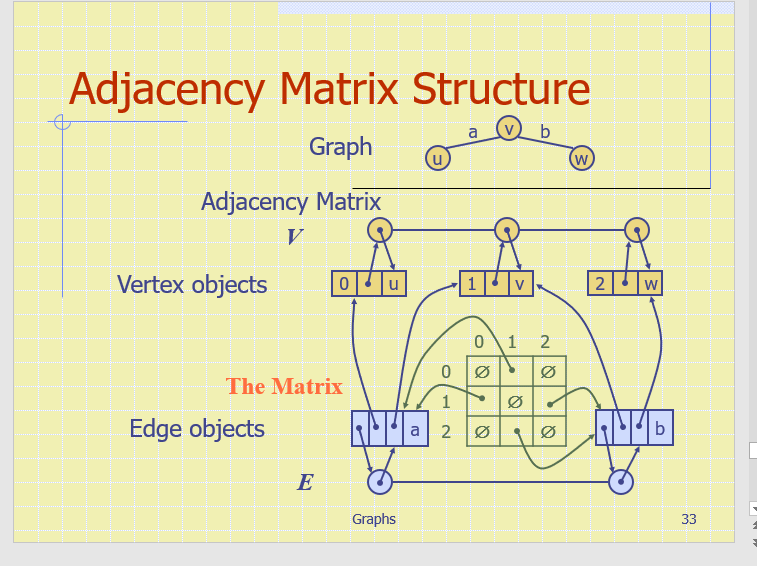


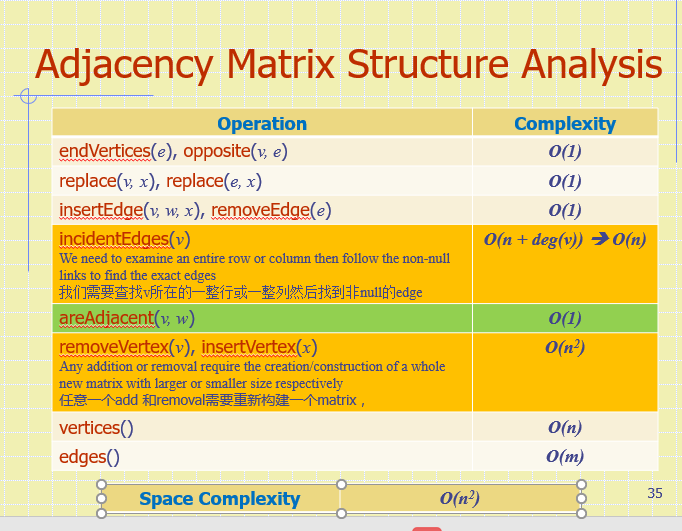
adjacency matrix structure

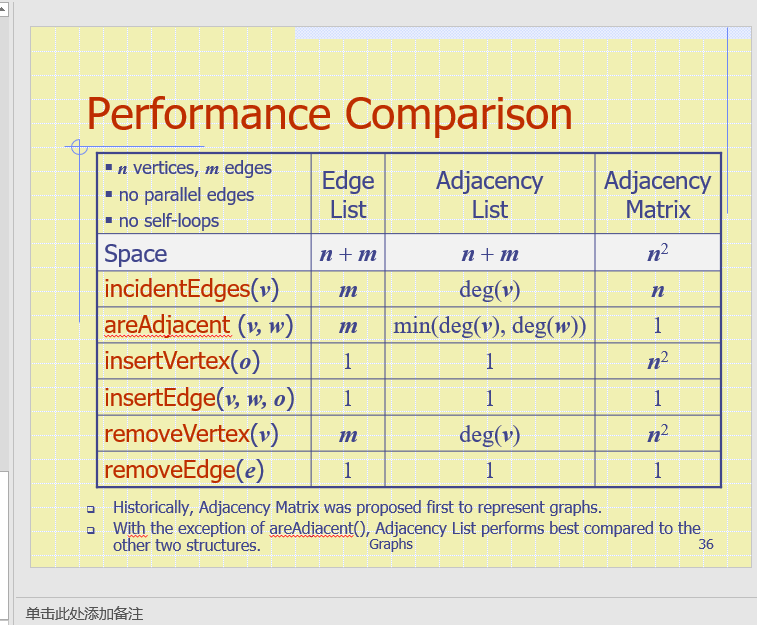
包含了所有edge list structure的结构元素加上//最原始的那种

vertex object需要加上不重复的[0,N-1]之间的一个整数，记作v的index

一个matrix，n\*n的 2-dimensional array，让A【i,j】 存储edge(v,w)的reference，v是vertex v 而这个v存储了index i, w 同理，如果vw不存在，a[i,j]holds null







历史上来看，adjacency matrix最早用来表示graph

除了areAdjacent(matrix最好)，其他的都是adjacency list最好