1.

Ωn<=Θn<=On

3n^6-15n^8不存在，会成为负数

Fn=1/N^4可以，只要不是负数，O（1）

2.Recursion.

Linear Recursion：一次method最多只recursion一次

基本线性循环基本就是O（n）

Tail recursion:把线性recursion的recursive call放到最后一步，这类recursion可以转化成没有recursive的method

**public** **static** **int** tailFactorial(**int** n)

{

**return** *tfact*(n, 1); // no recursion here

}

**public** **static** **int** tfact(**int** n, **int** res)

{

**if**(n == 0)

**return** res;

**return** *tfact*(n-1, res\*n); // this is where recursion occurs

}

他把循环单独写了个tfact，

Binary Recursion:一次method叫出两个recursion call

fibonacci不好，因为会重复计算

还是要用linear法

创造一个大小为二的array，初始为00

左边记大数右边记小数

然后A[0]=A[0]+A[1]

A[1]=原始A0

3.stack:pop弹出，push输入

我们可以用array来建立stack，最右边的是top

t是已有的最大index，初始是-1，因为已有的是0

用的空间是O(N),poppush这种操作是O（1）

限制是size固定

我们也可以用Linked list来代替array来构建stack

push就是tail加个新node，pop移除这个node

4.queue这个ADT存出任意OBJECT

enqueue后面加入，dequeue前面减掉

Q最好有一个f一个r来表示首尾

f是开头的index,r是结尾的index+1,空Index

初始都是0

当插入一个以后，r+1，f不变

所以Q最多存放n-1个元素因为 r要指着空

我们使用Mod这样就可以循环

Doublyended q,可以开头插入删除，也可以末尾插入删除

5.Linked List

创建新Node就是在开头让他指向原来Head，更新head，让head指向新Node

head remove.让head指向下一个node

Insert at tail

创建一个新node，加入一个element，让新node指向null，old tail指向新node，让tail指向新Node

remove tail没必要，效率低

我们可以用singly linked list来构建一个stack， top就是first node

需要的space是On

singly linked list，front是第一个node，rear是最后一个node

doubly linked list以及他的相关操作

插入啥的，看见就懂了

6.ARRAYLIST

我们用的arraylist一般都是基于array的

n来跟踪当前size，与最大size不同

insertion remove都要ON

当n到了最大size，我们进行add/double

add的complexity是O(N^2)

DOUBLE的总工时间是ON

Node list

就是创建另一个List叫做position list，每一个PositioN指向一个

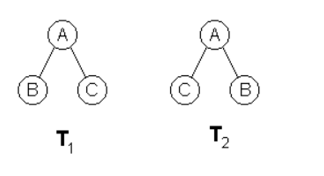
7.tree

depth of a node：他拥有ancestor的数量

height of tree,任意node最大depth，通常比视觉上小1

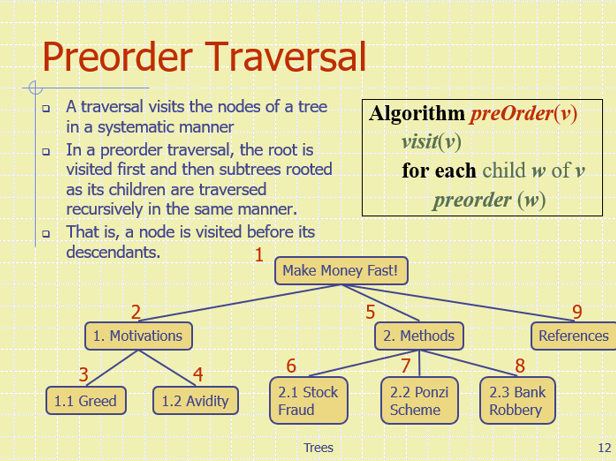
descendant：子孙

ordered tree，顺序重要

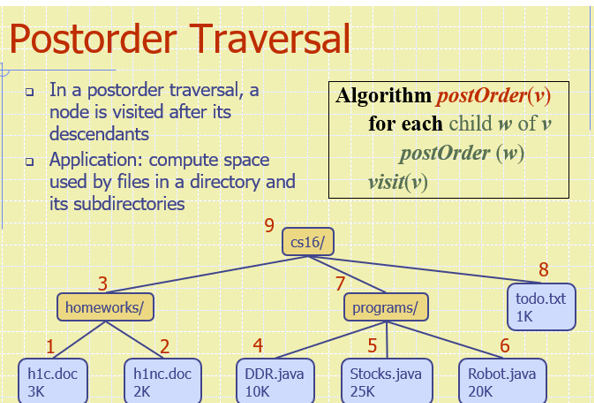
ordered 不相等，unorded相等

children(p)将return所有children

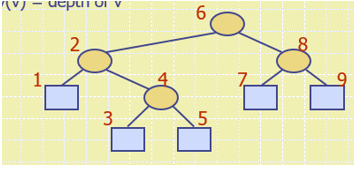
Preorder



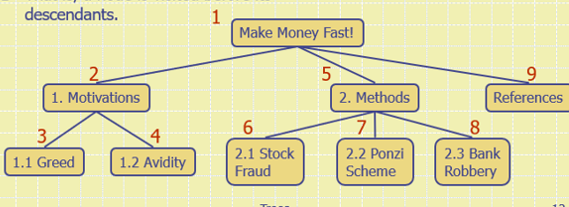
post order



inorder

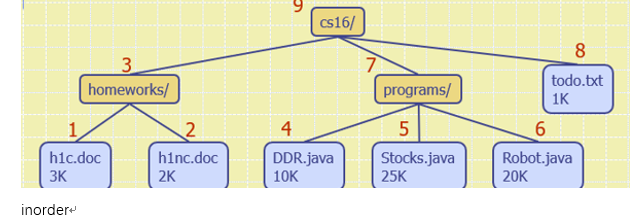


总结，preorder就是root开始，从左往右一个subtree一个subtree的排

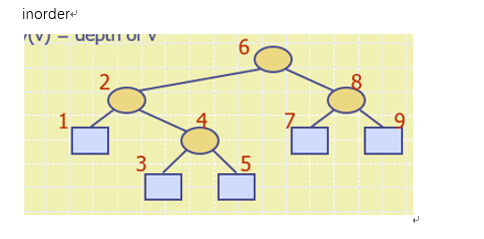


post order就是从下往上

一个subtree一个subtree的排，最后root



Inorder



先左再root，再右

binarytree的Linked structure，

存储的element，

parent node，左node，右node

binary tree也可以是array based

root rank=0,左node 2i+1，右Node 2i+2

PQ是有entry组成的colection

remove会remove最小Key

PQ可以接受两个entry有同一个Key

PQ由两种Object构成，entry们与一个comparator

创建 comparator需要

public class xxx implements Comparator

PQ SORT:把PQ作为辅助工具，把sequence的每一个都移到PQ里，然后不停removeMin,插回空sequence

基于sequence的PQ

如果PQ是由unsorted list implement的

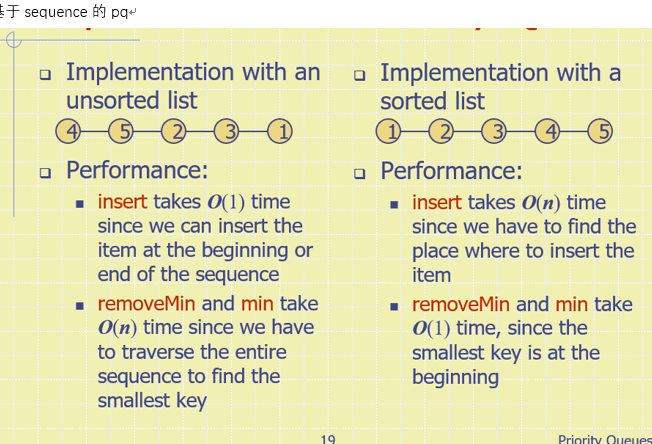
insert O(1)爱插哪插哪

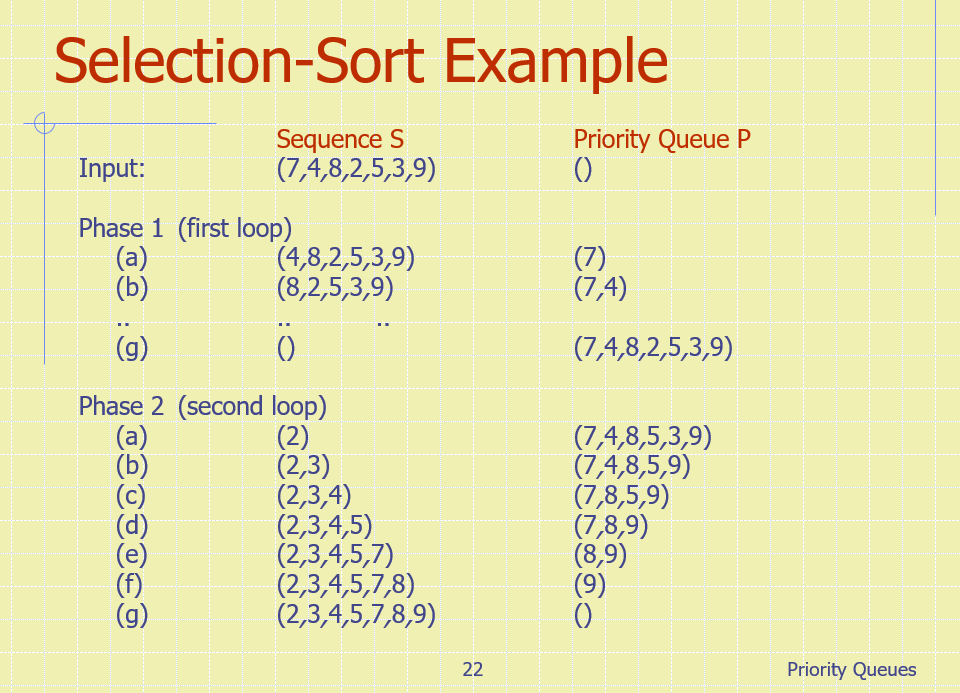
REMOVE MIN ON

如果sorted

insert On

remove O1



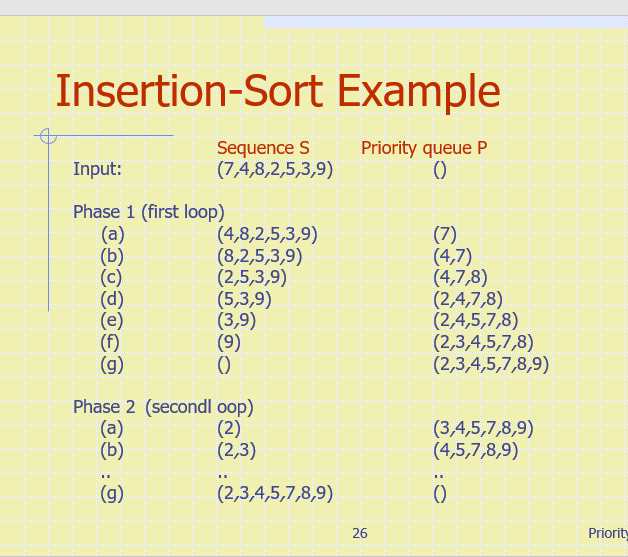


selection sort，无脑塞入PQ，然后每次remove min

selection sort：找到最小的插入index0,第二小的插入index1...

insertion sort:从前到后排序，Index 0小于index1就交换

insertion sort，塞入PQ的时候会找顺序，然后直接remove最前面的



都是On^2

9,heap

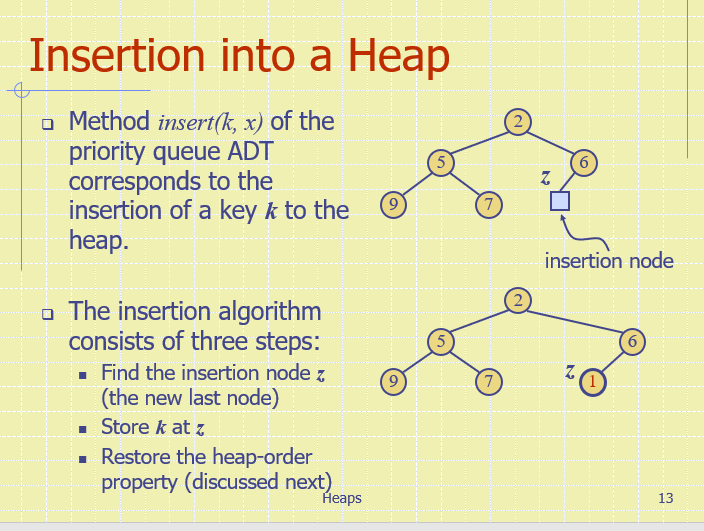
heap是用来代替LIST实现PQ的

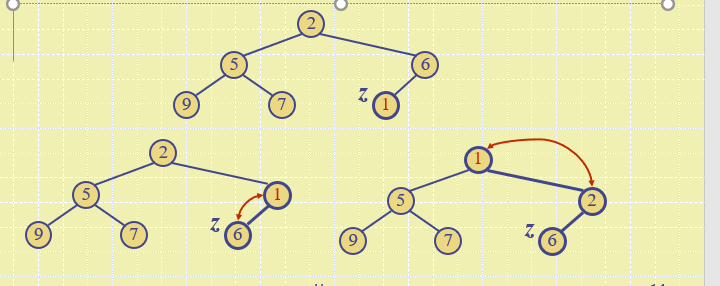
从上到下小的上大的下

是complete binary tree，无限左下

存储n个key height就是Ologn

insertion,创造一个新last node，然后往上遍历

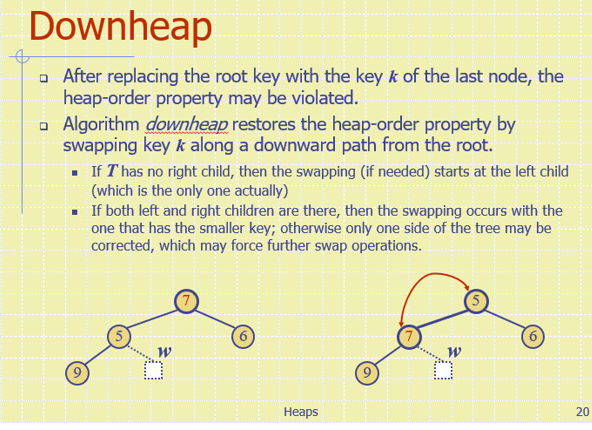




这个叫做UPHEAP

remove

减去root 然后用last node替代它，与key较小的换



叫做down heap

因为每一次交换实际上用时O1，那么高是Logn，downheap与upheap都是logn

heap sort

PQsort就是把所有的sequence元素倒入PQ里，然后remove min,complexity是n^2

heap sort就是把所有的sequence插入到HEAP里，，每一次是logn，n个元素就是nlogn，然后再remove出来，那就是nlogn

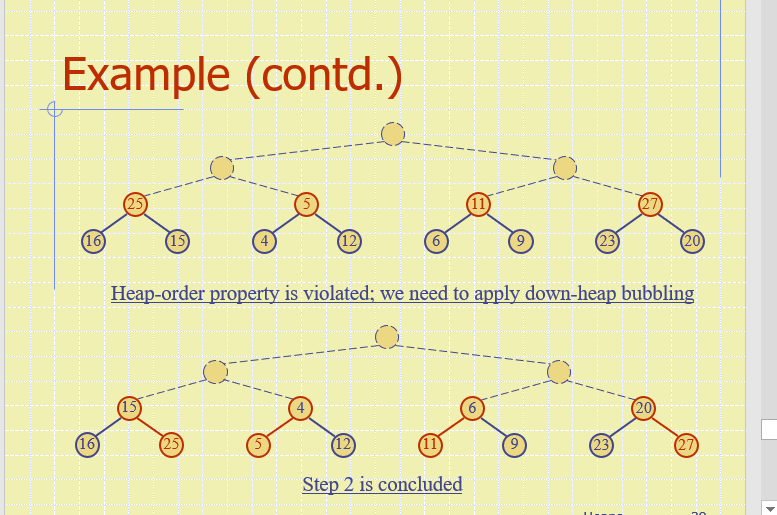
我们可以用array实现heap，root在0,2i+1,2i+2

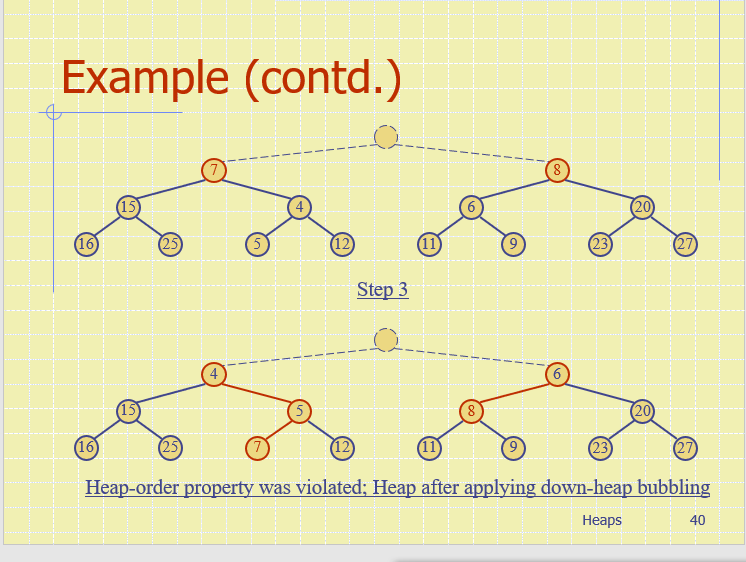
merge两个Heap，需要两个Heap和一个

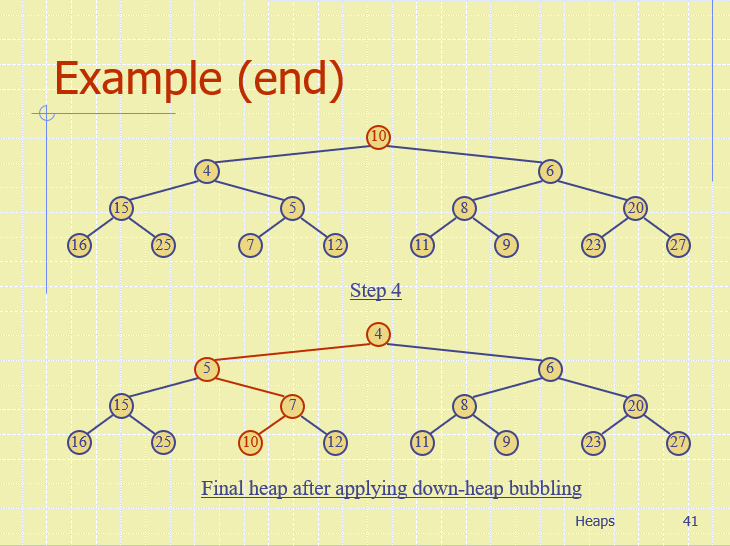


直接合在downheap

用merge法建立一个heap而不是插入法







直接按顺序merge每一轮都需要downheap，最终还是nlogn

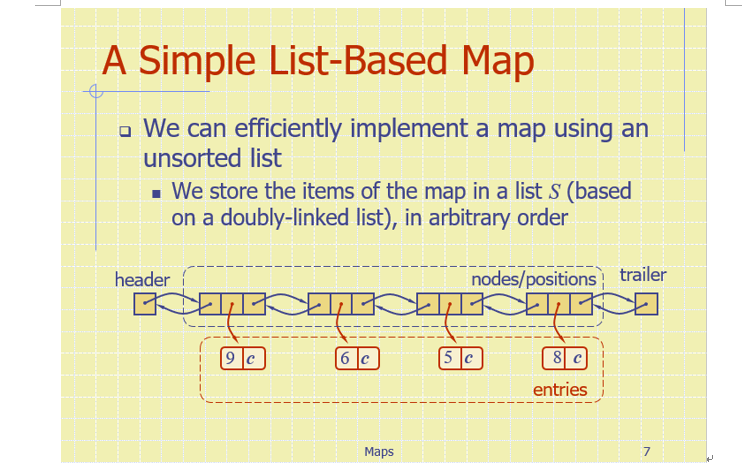
10.map

可以搜索的key-value组成的collection

一个Key只能对应一个entry

当你put （2,C）又put(2,E)就会用2E替代2C并return C

我们可以用双链表来构建Map



put 与get/remove都需要ON,PUT需要我们确信没有重复Key，get remove需要遍历直到找到这个Key

11.hashtable

hash function：由两部分构成 hash code：把keys 转成integer

2。compression function，把integer挤压到[0,N-1]

同样的原始key对应同样的hash code

hash function里的good：所有entry被均匀分在不同index中

控制collision:seperate chaining，cell指向一个Linkedlist（实际是map）

load factor n/N,

我们最好让n/N<1既创造比n还多的index

优秀的hash table get put remove的操作都是O[n/N]

与seperate chaining相对的，我们还有open addressing来控制seperate chaining

他需要load factor最多就是1，不然不够存

Open addressing分为两种，

1.linear probing

找到对应Index，不够就一直往后

删除entry需要大量的shifting，不如直接把他改成AVAILABLE

2.double hashing

他有一个次级hash function dk

当我们主要hash function 遇到collission

就开始激活次级function,在原有基础上加上次级function得到的值，不够再加直到找到

table的size必须是质数

还有一种与Linear相对的 open addressing

quadric probing,每次加一个平方数而不是一个一个找

open addressing并不一定比seperate chaning更快，但是节省了space

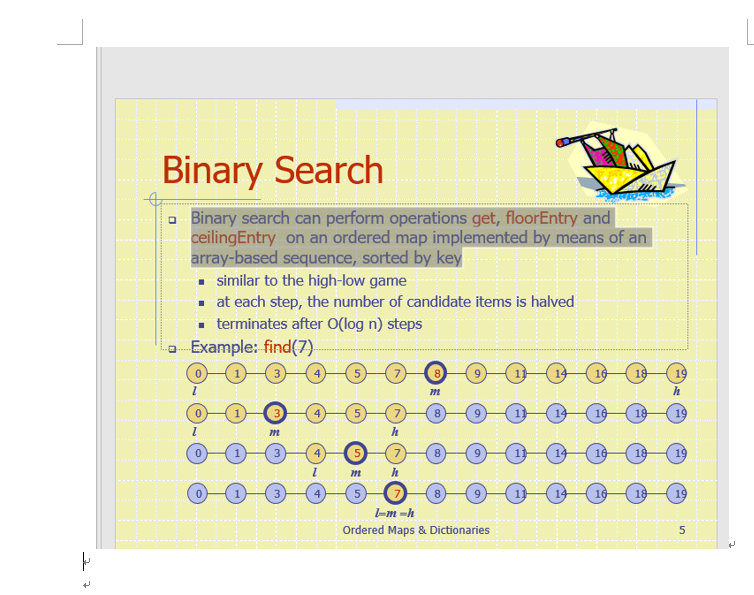
hash 理论最好的runing time是O1，就是没有collission的情况，最差是On，就是所有的都累积到一个bucket

如果load factor过于大，open addressing需要小于0.5,seperate chaining需要小于0.9

那么我们就翻倍N

12.

ordered map，我们通过一个comparator来按顺序插入entry 到map中，这样search速度就加快了，每次对半分，LOGN就可以找到



ordered map是一个基于array的用key sort过得map

put与remove还是On因为我们要remove

适合用ordered map的map

1:size很小，put remove就算On也没啥影响

2.map的绝大部分操作都是get,floorentry与cellingentry，这样就不用Put和remove

Dictionary：

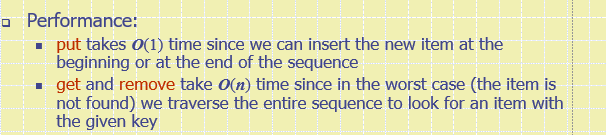
允许key重复

基于unsorted List的dictionary

List-based dictionary

log file\audit trail, 一个由unsorted sequence建立的dictionary

我们把dictionary的item存储在sequence里（doublelinked list 或者array）



put就插在开头或者结尾，O1

因为是unsorted，所以是On，（没找到）

log file可以使用的情况

1.size小

2.最长够用的操作时insertion，而不是get或者remove

基于hash table的dictionary

我们可以创造一个hash-table dictionary

如果我们用separate chaining来处理 collision，那么每个操作都可以被分配到不同的list-based dictionary，这些dictionary存储在hash table中

这样dictionary method效率能到达O（1）

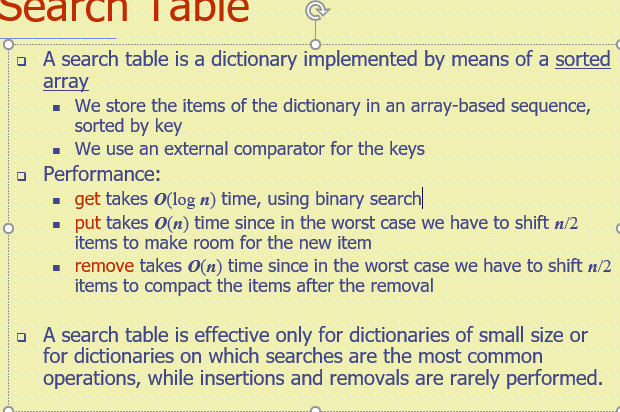
search table：用sorted array构建成的dictionary

我们把item存在对应Key的位置

get Ologn，用binary search

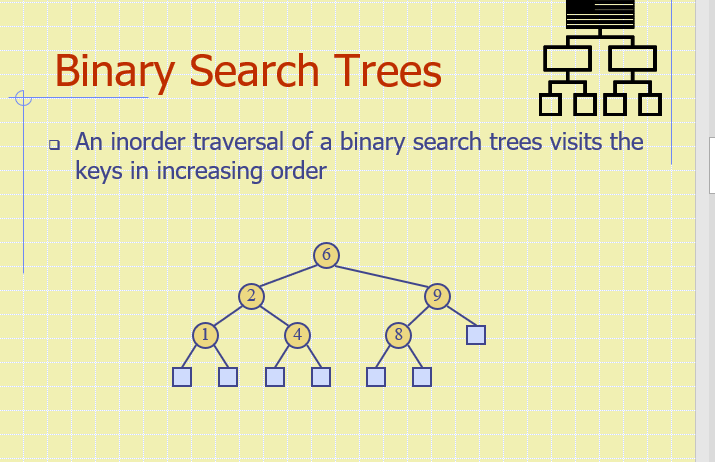
put On

remove On ,



13.binary search tree，左边的永远小于等于root永远小于等于右边的,

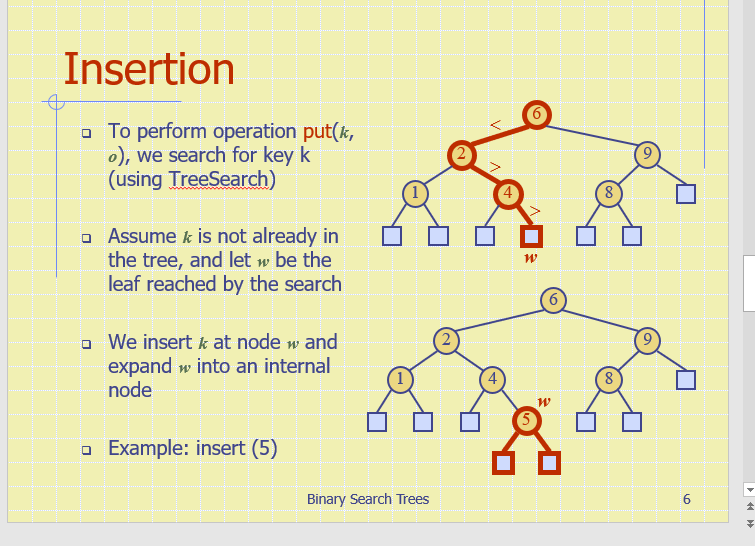
external node不存储项目

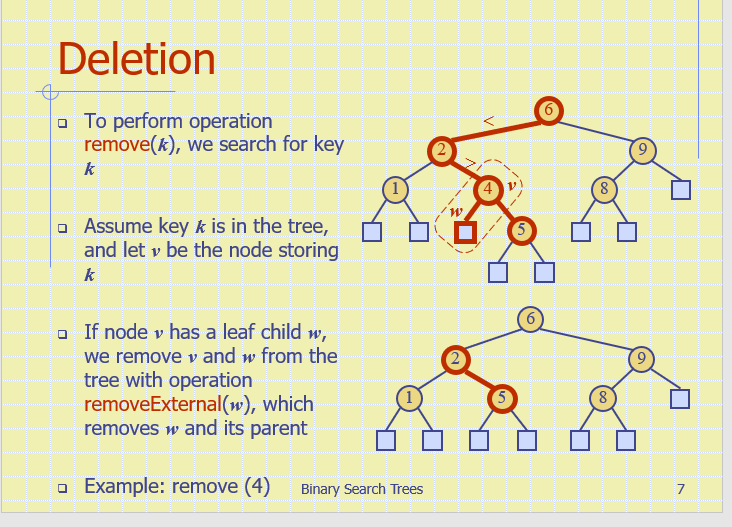


inorder traversal会升序visit keys

search就看大小就完事儿了，小于的左边，大于的右边直到到了external会return null

insertion，先search，假设不存在那么我们就插入他所在的点





remove还是先搜寻，假设v是我们要的点，那我们就remove他和他的左边空node

把5代替他

但如果他的两个都是internal

我们找到inorder顺序下紧跟着他的node，然后把5替换成3，删掉对应的external node

h最好是Ologn最差是On



space 是O(N),其他的都是O（h）,h是Onworst case,ologn best case

14.AVL TREE

AVL TREE是binary search tree的一种，但是加了一条规定，左children右children高度差最多为1

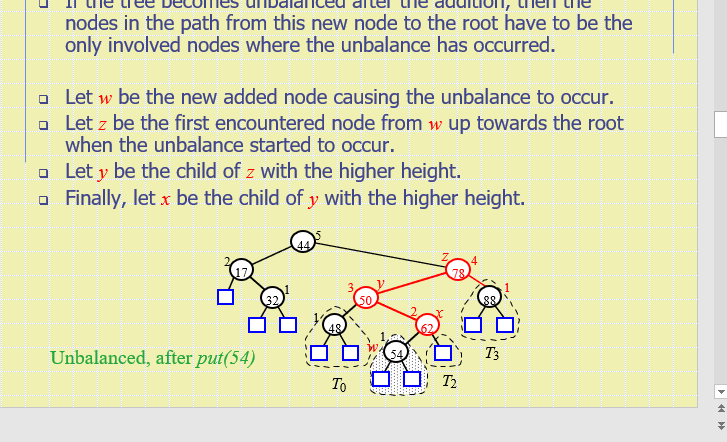
一个存储了n个key的avl tree的height是Ologn

insertion会导致高度变化，所以要重新构建‘’

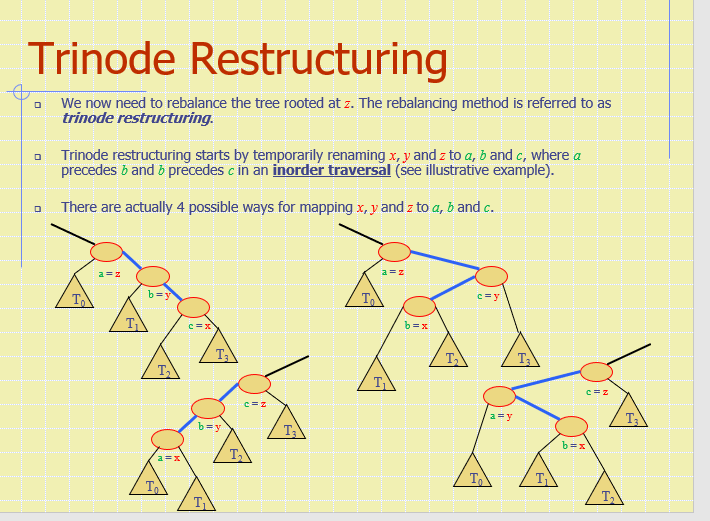
第一步找xyz，

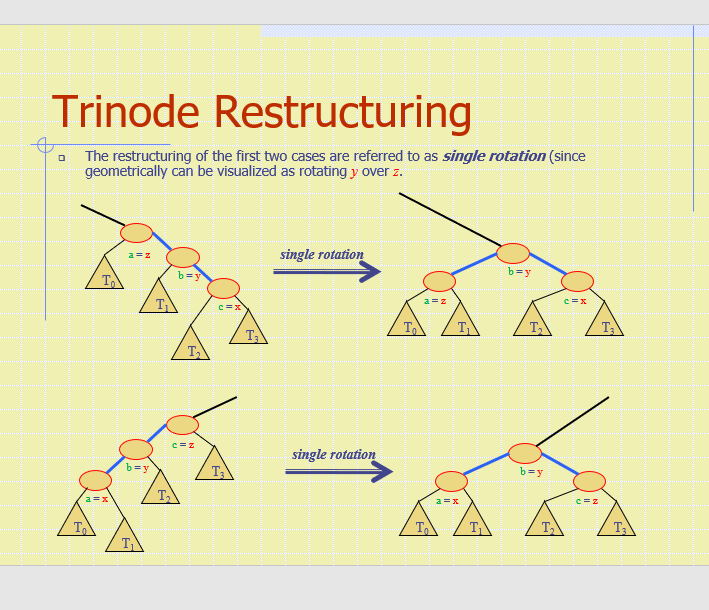
就往上三层

X是拥有W的，Y是X的爹，Z是Y的爹



2。找到对应abc

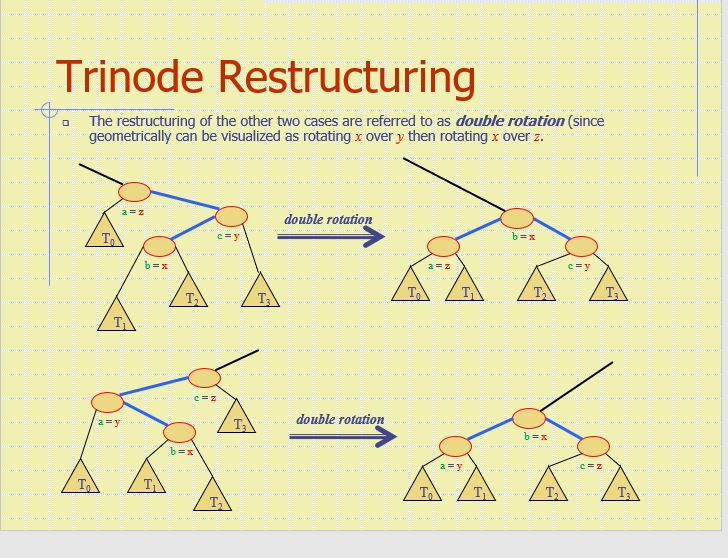




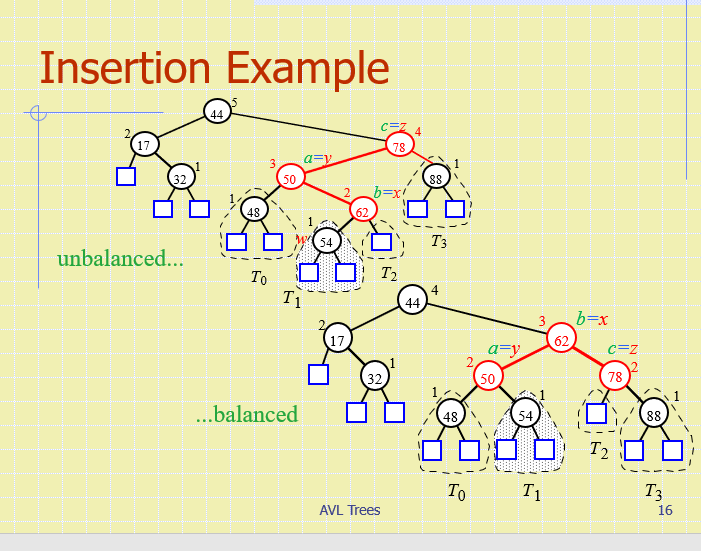
一条竖线，中间Y点提留起来，a左b右

拐弯的，

最下面的提起来



a左c右



这个就属于拐弯的，62提起来，注意分子树，xyz一共会产生4个子树，从左往右自己分配

removal一样需要restructure

get用时Ologn

put与remove同样，因为他们都是建立在get上的

restructure只要O1

15.merge sort

divide分开,recur解决子问题,conquer合并

base case是0或1

当A不等于空且B不等于空时

如果A的firstelement小于B的firstelement

那么就把A的第一个remove并且加到S的最后

不然就把B的第一个remove并且加到S的最后

如果A空了

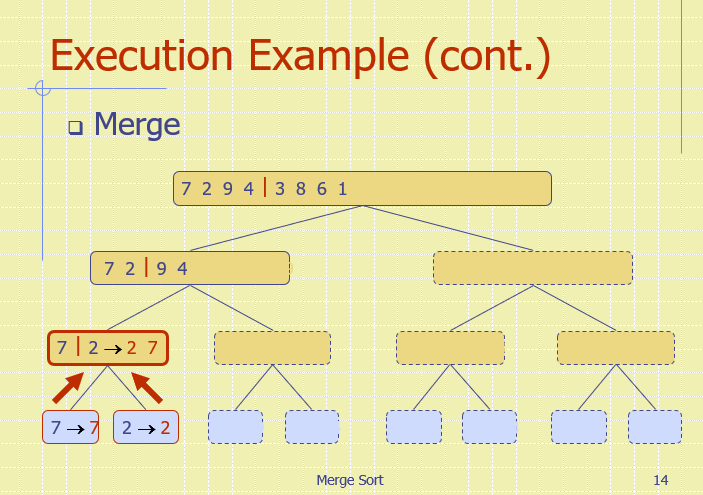
把A的第一个REMOVE并加到S的最后

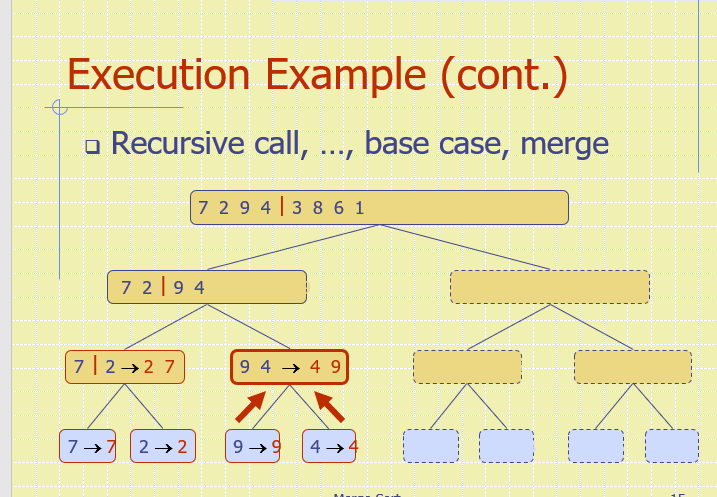
如果B空了

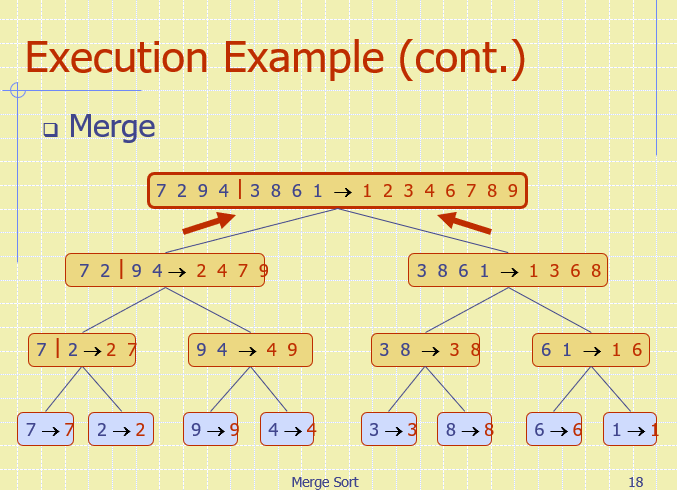
把B的第一个REMOVE并加到S的最后

注意A与B本身是SORTED的









把两个sort array合并起来的用时是N,高度是Ologn，总消耗是NLOGN

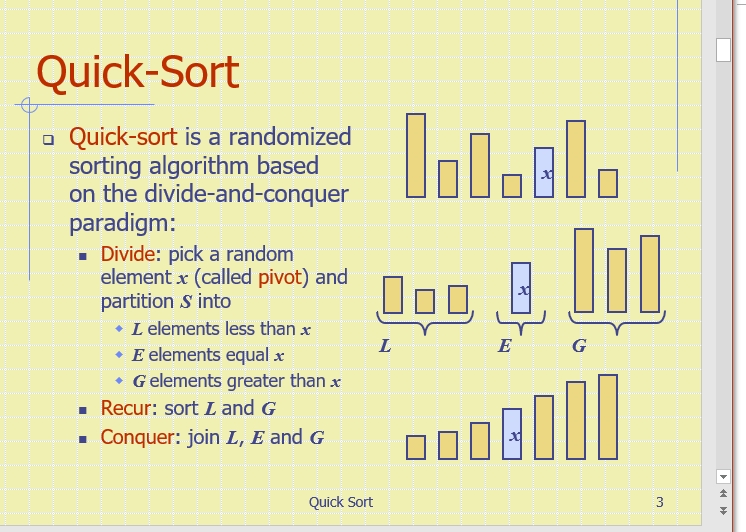
heap sort不需要辅助PQ

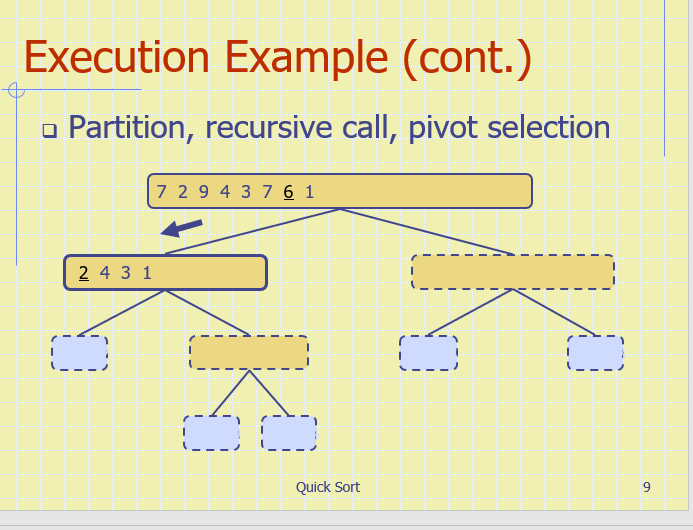
16.quickSort

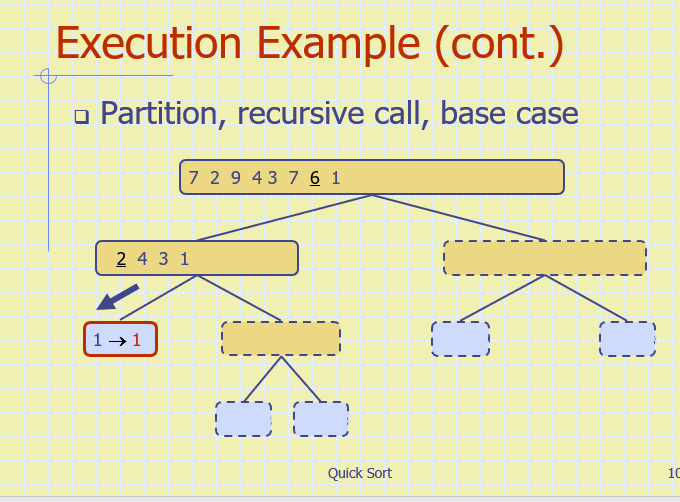
divide：选取一个pivot， 小于X的放在L，等于的放在E，大于的放在G

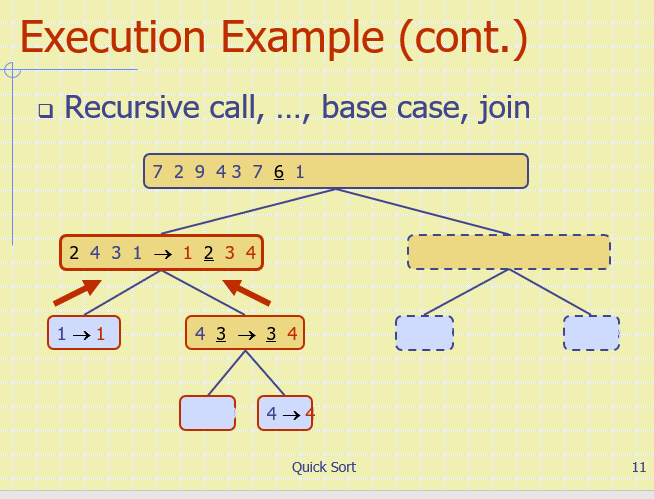
recur 对LG排序

conquer合并

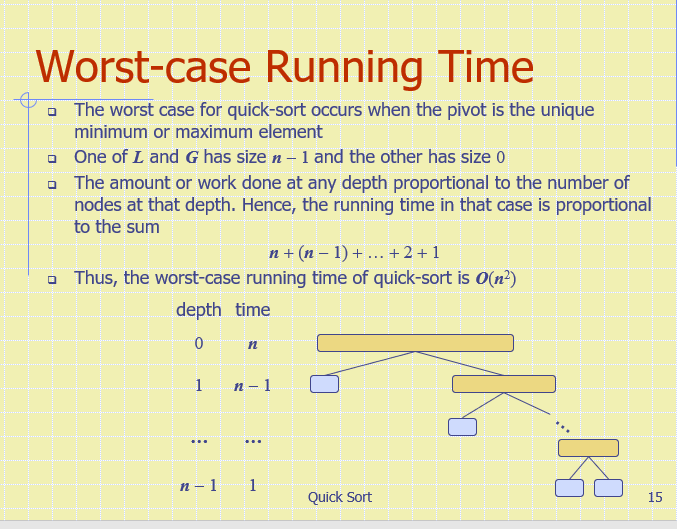








看原笔记即可



最差的情况：

pivot始终是最小的/最大的element

一个L或G的size是n-1,另一个是0

每一个需要做的时间就是他的深度

n+n-1+...1，复杂程度就是n^2了

理论running time，

假设上一个node的size为s

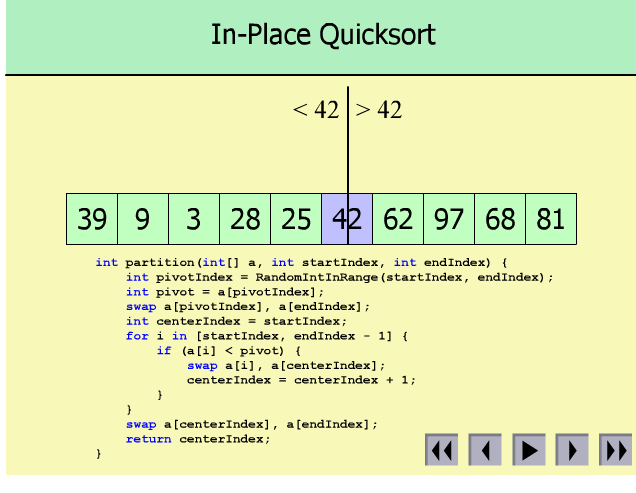
good call:L和G的size都小于3S/4

bad call:一个 L或G的size大于3S/4

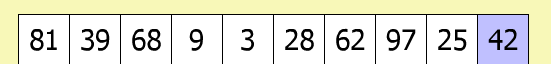
一个call good的几率是1/2

那么总共的running time 就是NLOGN

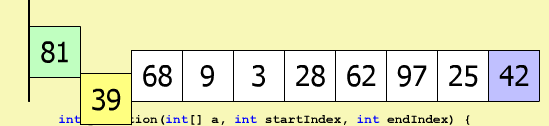
inplace



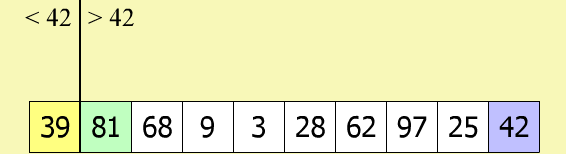
首先，选取任意一个数，把它放到最后

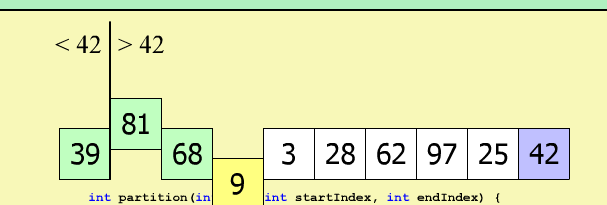


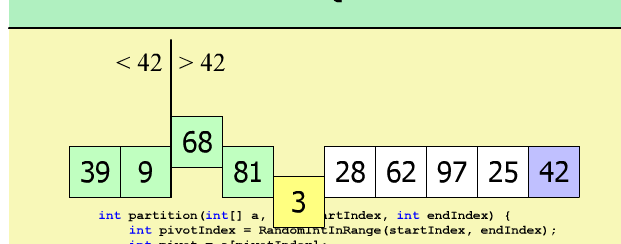
从一个大于Pivot的数开始数，如果遇到了一个小于Pivot的数，swap



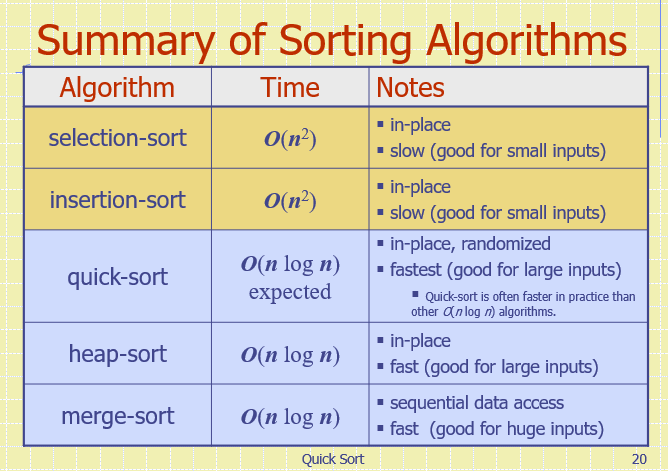
当你成功换一次，指示线 前进一格







最后停止的地方就是42要呆的地方



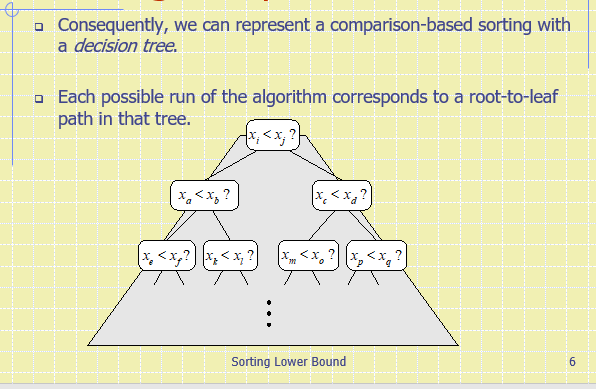
quick sort expected是nlogn最快，实际上最坏是n^2

17 lower bound

基于比较的sorting法不可能超越ONLOGN

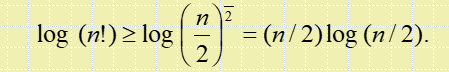
因为假如我们有一个sequence(x1,x2,.xn)

我们终将形成这样一个比较树



从root到external node（leaf），每一条路径代表着一种排序可能，如果有N个树比较，那么leaves的数量是n!, 高度至少是Log n!,.

因此这类 comnparison-based sortring 算法至少使用log n! time



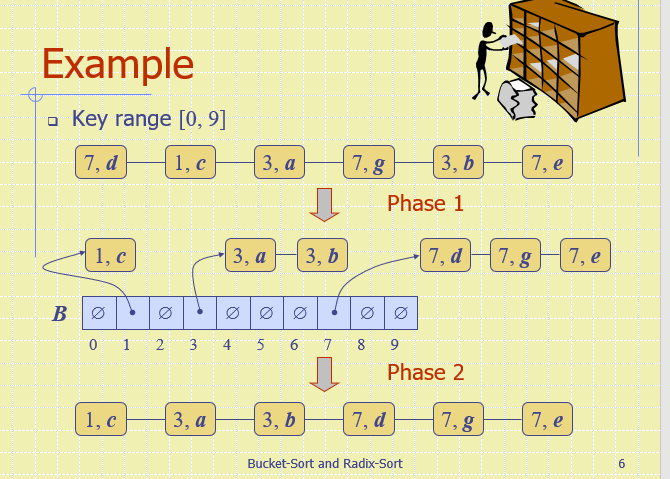
二log n!>=nlogn

所以任何 comparison-based sorting algorithm至少要用 nlogn ，不可能比这还好

18.bucket sort

S是一组n entry的Key,范围在[0,N-1之间]

我们只要把它插入对应的bucket B【k】就行



第一步，拆，第二步装，使用辅助hashtable B

分析：第一步需要O（n） time

第二步需要O（n+N） time

因此，总共时间是O（n+N）

然而N又是O（n），那么sort in O(n) time

必须要注意的是，为了让算法有效，N不能显著大于n

我们队key的范围是有限制的【11，20】就会产生一个[0,9]的hashtable

11放到0就可以了

我们接受的key只有整数和string，string长度就是bucket sort的key

Stable Sorting

一个排序算法被定义为稳定"stable"当：在它形成sorted sequence的过程中，如果两个entry有着同样的key，那么在新sorted sequence中，他们的前后顺序是与unsorted sequencer相同的

我们之前对bucket-sort的非正式描述其实不能保证stability稳定性

1.我们没有限制entry是怎样从unsorted sequence提取出来的

2.我们没有限制entry是怎样从辅助的bucket array中提取回去的

通过以下方法，我们可以让他stable:

1.强制执行在unsorted sequence中removal item的顺序：从前到后

2.每一次insert:，先找到自己所属的bucketB[k]，然后插入B【k】这个list的tail

3.强制执行removal回去，从前到后，从head到tail

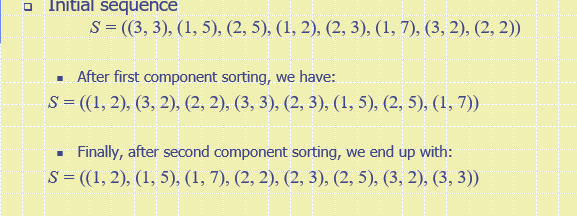
d-tuple (x,y,z,kd)有d个key的叫做d tuple

x优先，y次之...

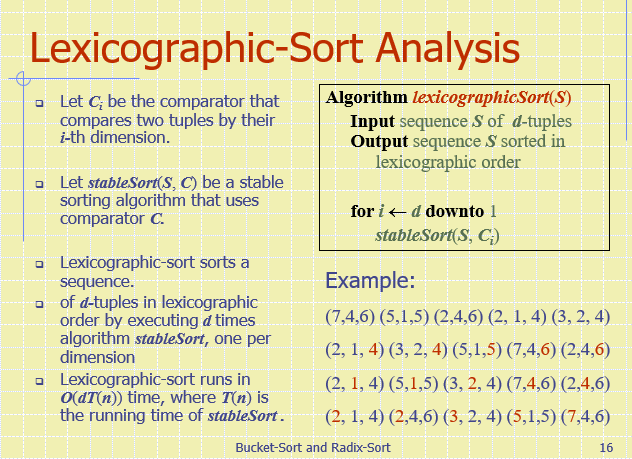
这就叫做lexicographic order

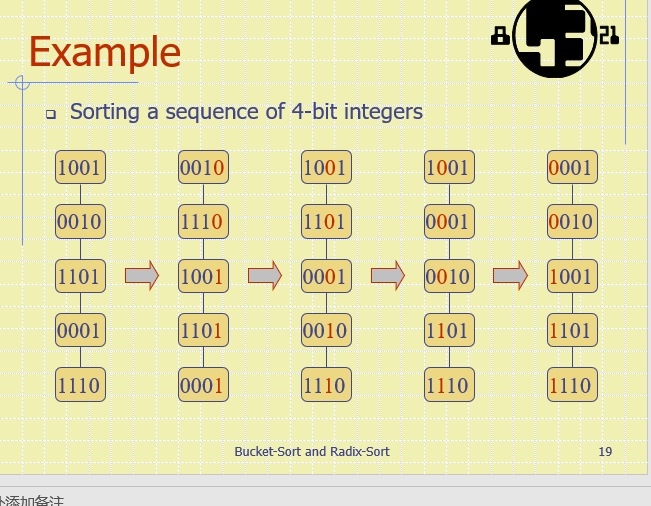
radix sort就是用lexicographic order来比较多key 元素

从小的数字开始sort（z,y,x的顺序）



注意保持stable





先排最后一位，再排倒数第二位，....

19.graph

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)

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

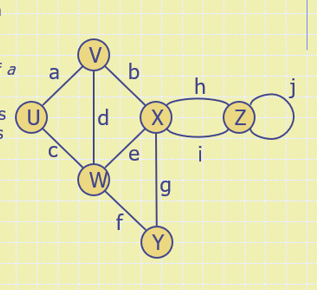
区别在于collection可以有重复

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



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）

spanning subgraph:H是G的spanning subgraph当他有了G的所有vertices

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

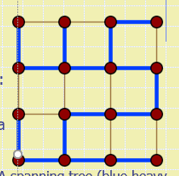
Spanning tree

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



性质：所有的点的degree加起来最终等于两倍的edge

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

undirected simple graph 的edge数量小于等于 n（n-1）/2

三中用来表示graph的方法

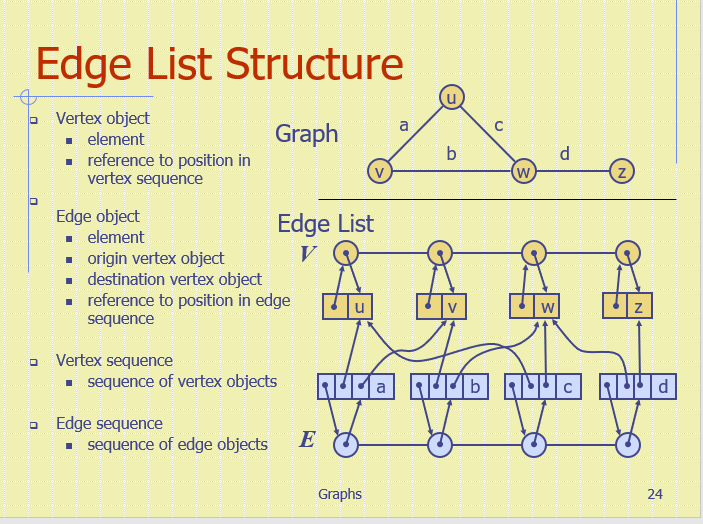
1.edge list structure

2.adjacency list structure

3,adjacency matrix structure

前面两种占用n+m space complexity

第三种n^2



Vertex object:

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

Edge object

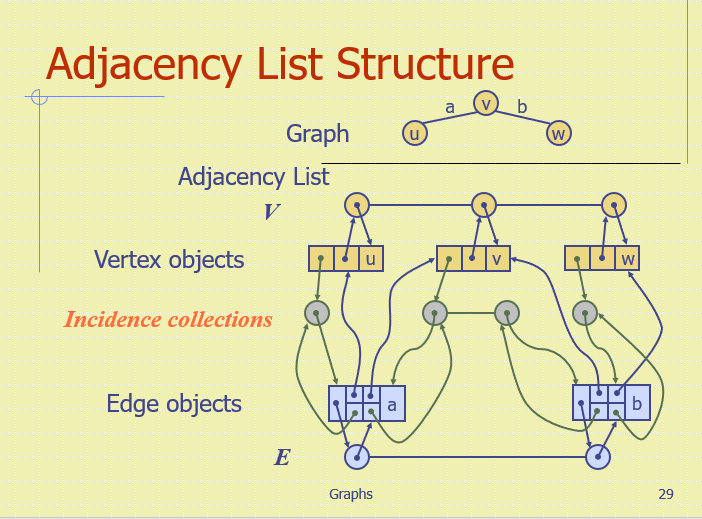
1.本身所存element 2.指向起点vertex object, 3.指向终点vertex object， 4.他们在edge sequence 中的Position

Vertex sequence

由vertex objects组成的sequence

Edge sequence

由edge objects组成的sequence



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

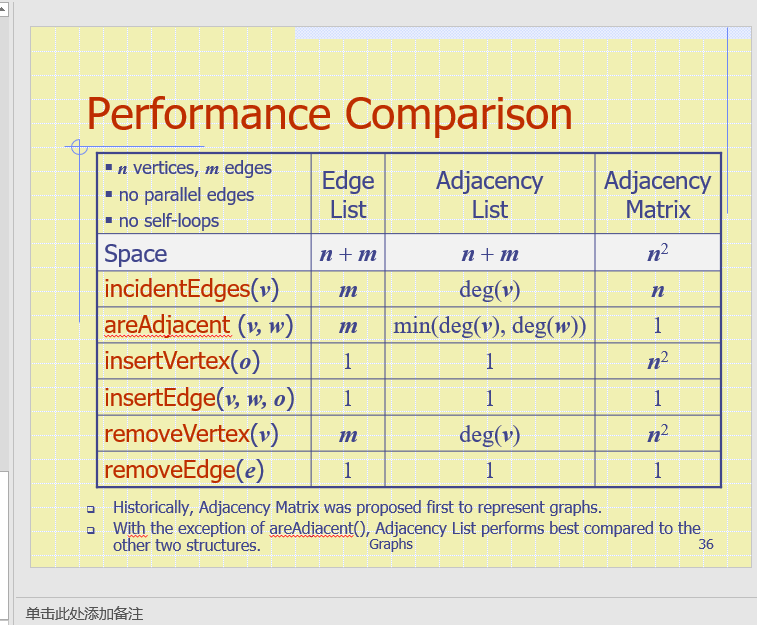
一个edge object e，假设他连接了v和w，那么需要两个额外的reference，指向l(v)与l(w)。这样我们就可以通过vertex的一条edge找到其他的incident edge

因此，空间利用率仍然与vertice与edge的数量成比例

adjacency matrix structure

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

DFS

首先明确几个状态

初始unexplored

VERTEX状态visited

edge状态discovery/back

我们使用DFS可以 visit G的所有vertice与edge（哪怕不是connected）

可以决定·G是否CONNECTED(开始一次DFS遍历，然后看得到的vertice和总vertice比较)

最后的路径将是一个spanning tree(如果G是connected)

最后的路径将是一个spanning forest(如果G是一个 non-connected)

找到两个点之间的路径（如果存在）

找到G之间的cycle(或报告G没cycle)

DFS的算法

他会给每个vertex call一次

每个edge会被call两次（每次都来自于其中一个点）

最终DFS的 complexity是n+m

但是这个分析是建立在 假设存在以下mechanisms，每个都可以在固定时间内完成

1.决定一个点是否visited

2.设置一个点事unexplored or visited

3.决定一个edge是否被explored

4.设置一个unexplored的edge，让它成为discovery或back

记住每个点都会被label两次，一次是Unexplored一次是visited

每个edge也会被·label两次，一次是初始的时候Unexplored，一次是discovery或back

5.还要有mechanism来寻找，incidentEdges()，opposite()

DFS的性质们

1.dfs(G,v)将visit v所在的connected component的所有vertice与edges

由DFS（G,S）发现标记的edge将组成一个spanning tree

BFS

anchor LEVEL0

LEVEL1。。。。

vertex unexplored/visited

edge unexplored/discovery/cross

我们不会回头，只会从node level i到level i+1

使用BFS你可以做到以下问题：

visit所有的vertice与edge

决定G是否connected，

得到一条spanning forest 如果G是non-connected

得到一条spanning tree如果G是connected

找到有没有 cycle

给你一个起始vertex s，那么就会得到每个G的vertex v，并且得到sv之间的最小路径，或报告这个路径不存在

也是m+n,

性质1：BFS(G,S)将会遍历所有vertice与edge

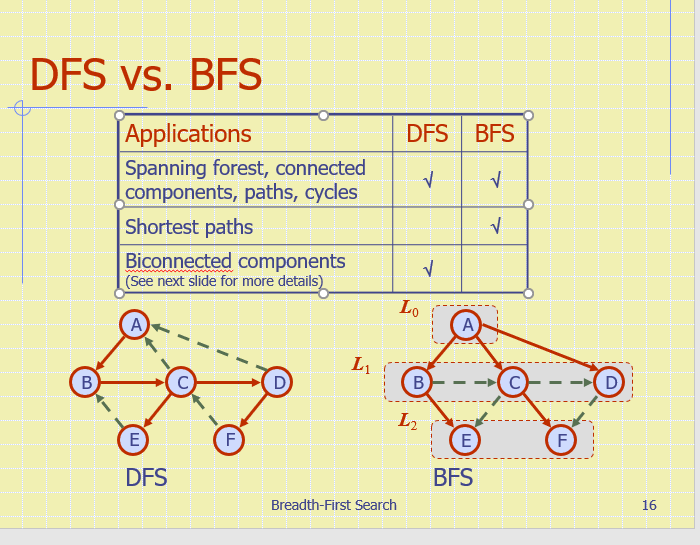
性质2，discovery edge将构成一个spanning tree

xingzhi13, 对于每个vertex v

假设这个v除于level i

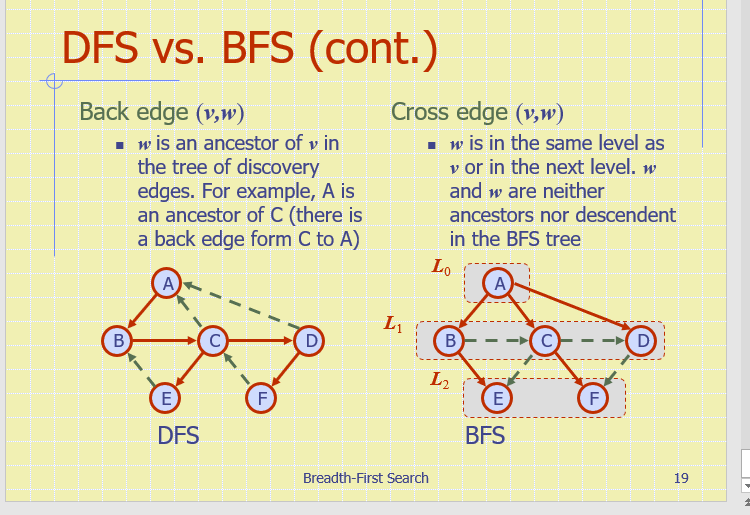
那么Ts路径，从s到v有i 个edge

每一个从s到v的路径都是shortest path(实线)



back edge，A是C的ancestor，那么c到a就是back edge

cross edge:vw在一个level或者 w在下个level.V与w之间完全不存在ancestor或descendent问题



22..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

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

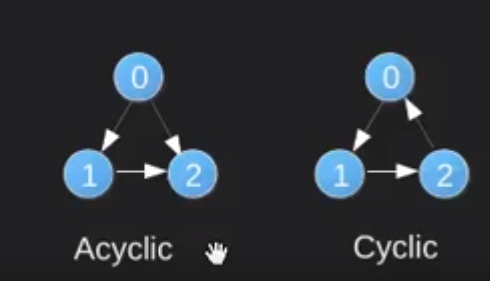
我们也说v reaches一个edge(w,z)如果v reaches w而w是edge源头

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

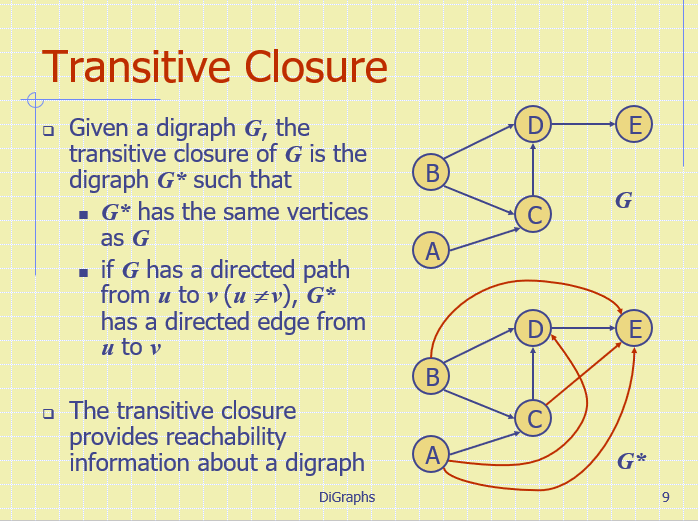


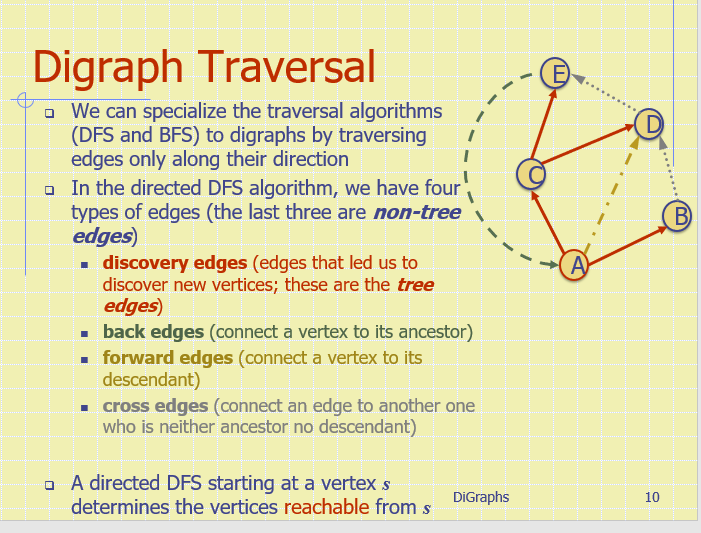
transitive closure

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





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

.

为了得到transitive closure of G,

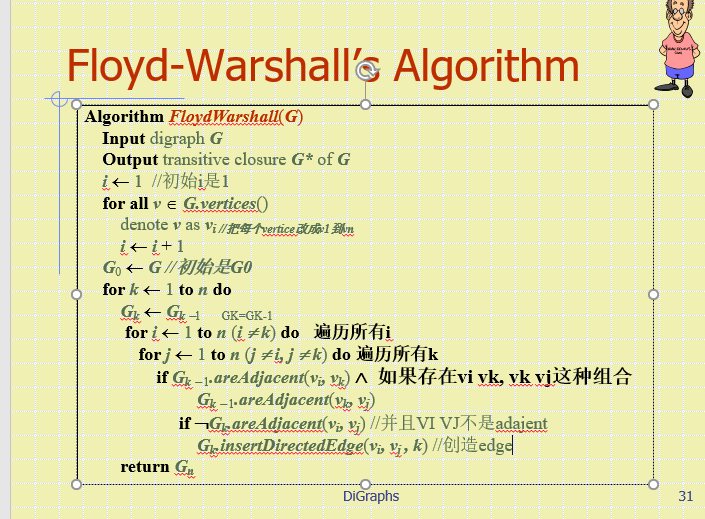
可以找到/得到 G的transitive closure（所有reachable点都链接·上edge）

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

或



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

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

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

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

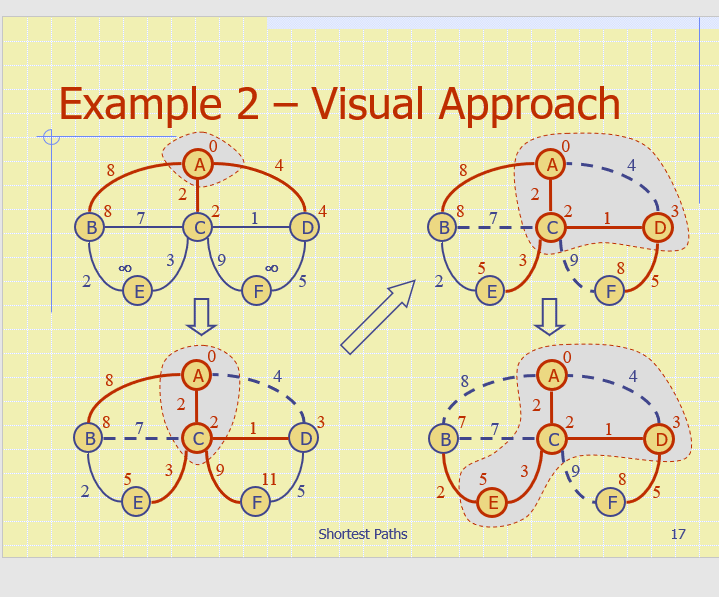
v在sort中排v1，移去

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



初始为A，到自己是0，做一个cloud，包含的三条path到bcd，那么现在BCD的最短path为824，  
朝最小的path扩展，我们发现到D的路径可以由2+1=3,, 2+7>8,所以8保持，4舍去

抄最短路径延伸，3，到F最短路径·为8

D延伸到头了，开始向第二短路径延伸，E，

他这个算法真他妈令人费解

网课版本

A能直接碰到的BCD,这时B是8，C是2，D是4

找最小的C，C能直接触碰到的BEFD，这时B是8不变，C是2，E是5，F是11，D是3

这时找最小的D，D能直接触碰到的F，F更新为8

这时找最小的E，E能直接触碰到的B，B更新为7，

B没啥可更新的，

找到F，

用时

***O(n2 log n)***