4.2-4.5

4.2 Strassen算法

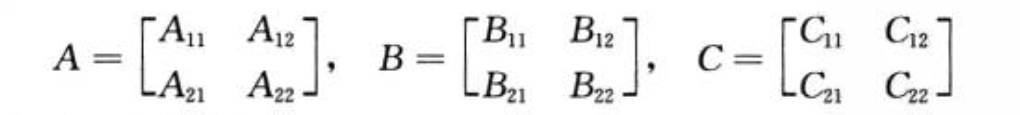
两个矩阵相乘，常规需要n\*n个数要求，每一个都是a1b1+a2b2+a3b3.... n个数，因此是n^3

Strassen方法

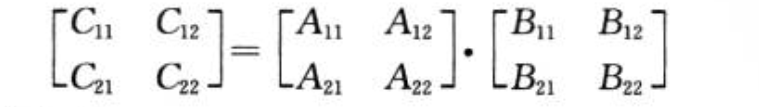
Divide and conquer法

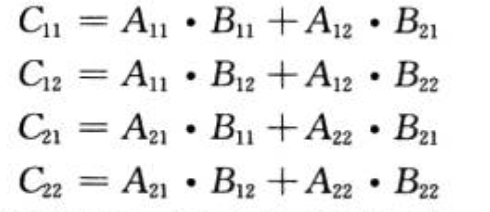
把大的问题分成小的

matrix A\*B=C

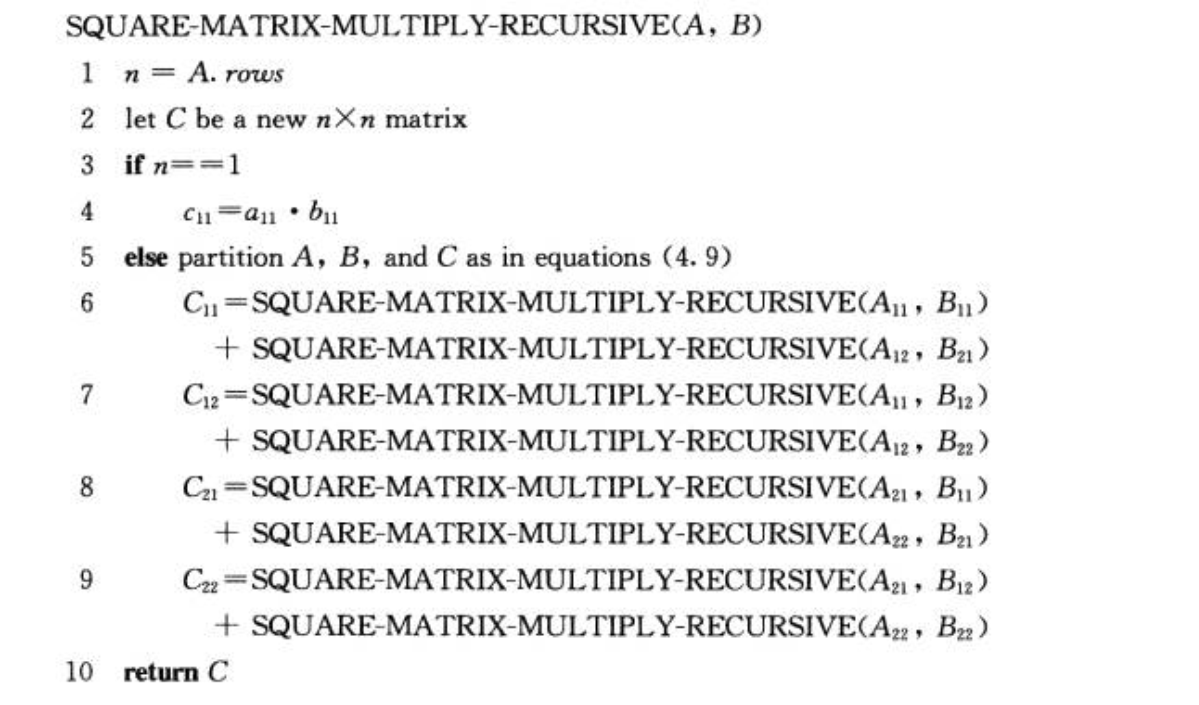


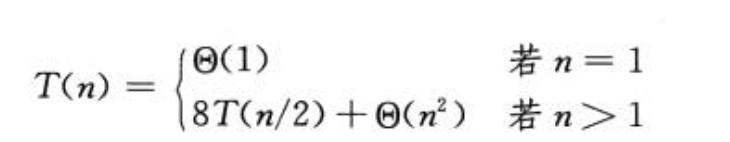
加上辅助线，把这些矩阵四等切割





因此我们可以设计一个recursion的conquer and divide



就变成了这么一个玩意儿，变快了

w1前置知识：

Big-O代表着最坏情况，Big Omega代表着最好情况

注意O可以无穷大，fN如果小于N，必然就小于NLOGN。。。

OMEGA可以无穷小，他如果大于N，必然就大于LOOGN。

都是TRUE，但是不一定是good好的。不是最优解

只有O和OMEGA交界，得到的big-theta才是最优解

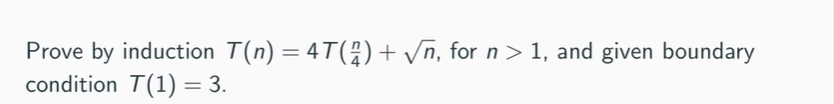
算法只是理论上的，并不代表exact运行时间，所以我们用渐进符号asymptotic notation表示run-time

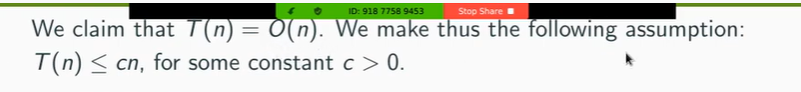
g(n)指 这个asymptotic notation理论所用时间，fn代表实际时间//一般来说fn写具体公式，g(n)就是简化表示法

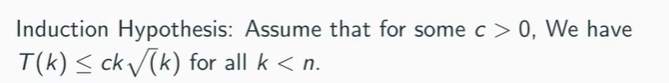
 

小o与大O唯一区别就是不能等于

Substitution Step



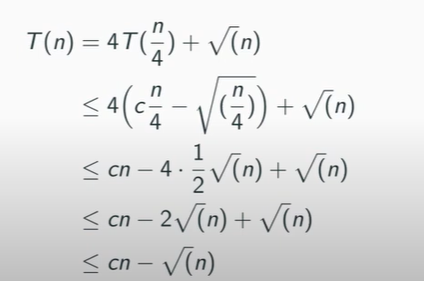


IH: 

**assume that for some c>0,We have for all k<n Omega 猜大于等于，一般不说默认为O（n）**

IS:

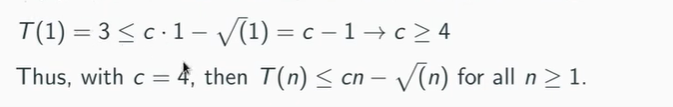
把假设代入右边的T(n)



cause it is omega/big-O, we need XXXX>0

然后证明右边的项大于0时c的取值

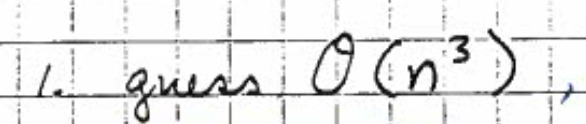
最后要代入



注意，减去一格低阶项是一个很常见的想法，如果如果我们得到了一个O（n^2）这样一个差的不远的东西·，我们第一个就该猜这个

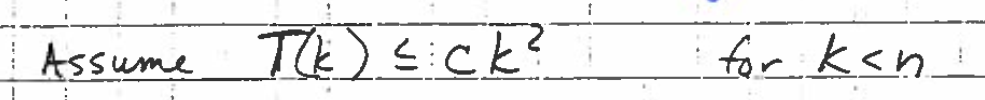


1.Guess



2.

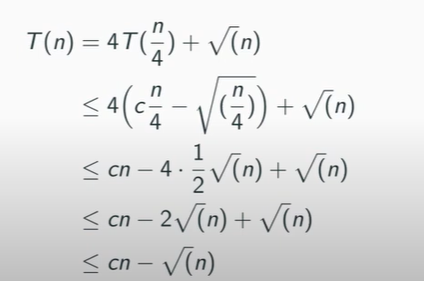
IH: **assume for all k<n**



**Omega 猜大于等于，一般不说默认为O（n）,猜小于等于**

IS:

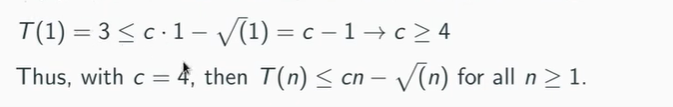
把假设代入右边的T(n)



cause it is omega/big-O, we need XXXX>0

然后证明右边的项大于0时c的取值

最后要代入



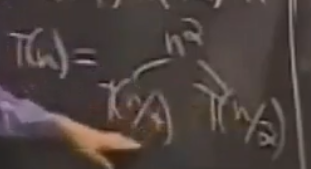
注意，减去一格低阶项是一个很常见的想法，如果如果我们得到了一个O（n^2）这样一个差的不远的东西·，我们第一个就该猜这个



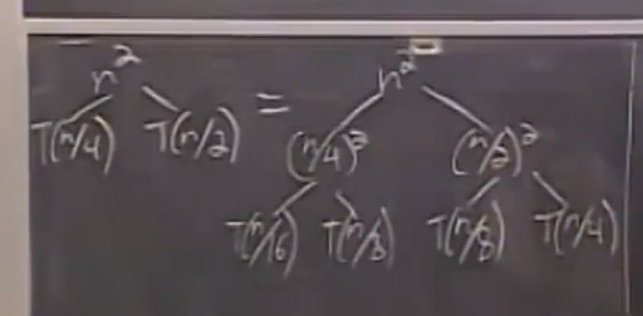
recursion-tree

例如T(n)=T(n/4)+T(n/2)+n^2

第一步，把非recursion项提取出来，作为root，把recursion项作为leaf,构成初始树

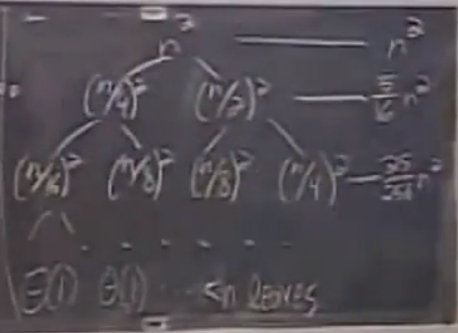
 //左边是n./4，画质不行

然后递归展开叶节点

，我们可以无限递归，最终底部变成常数叶节点，

但是我们是不知道常数叶节点多少的，因为不同的左右子树递归速度不一样

但我们没必要完全展开树



每层都横着加起来

而这一列我们加起来的数加起来，实际上等于整个树的值，而整个树的和，就等于T（n）//它是由T（n）拓展出来的

1+5/16+25/256+.....5^k/16^k

甚至这里也没必要仔细算

因为他小于1+1/2+1/4+1/8+....=2

所以O(n^2)就是一个合理的upper bound

你可以把每一个节点，看成是这一次recursion所需要的代价，

这个方法主要目的是得到一个好的guess，然后我们可以用substition 法来猜测

书里的例子是T=3T(n/4)+cn^2

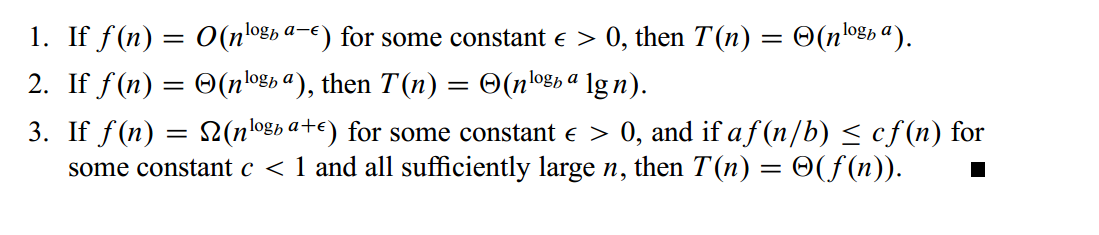


他把3T(n/4)拆分成三个T（n/4）

Master theorem

只有这个形式才能用master method

a要大于等于1，b要大于1,f（n）是渐进正函数

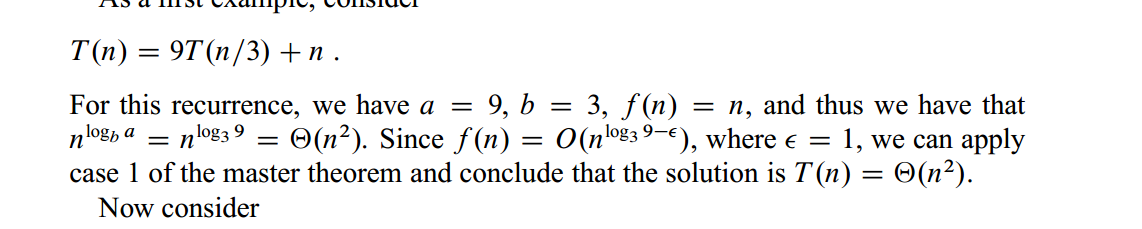


可能这个EPSILON比较令人费解，但要记住，我们比较的核心是后面那个f(n)和n^logba

第一种情况就是fn比nlogba小

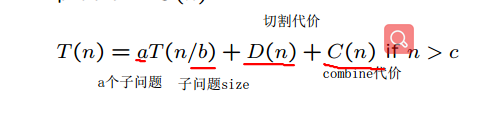
第二种情况就是同一类型，例如n^2与2n^2

第三种情况就是fn大于nlogba

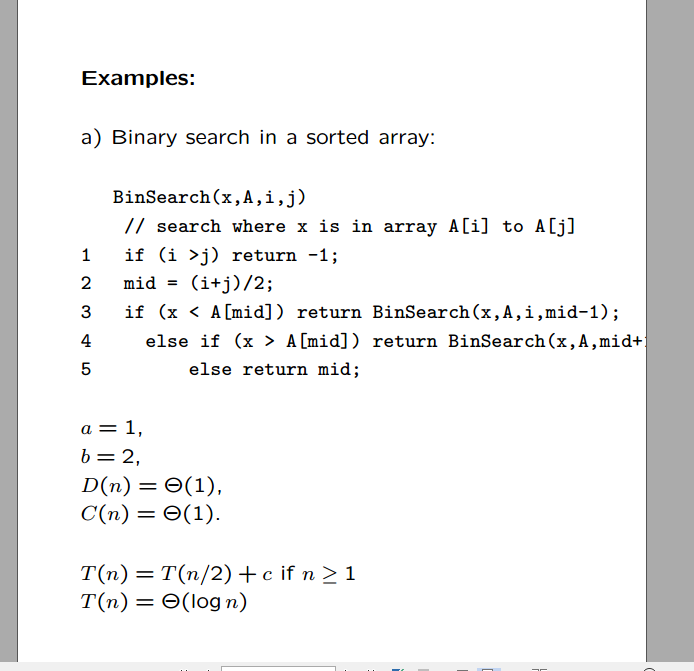


W2

Divide and conquer



将父类问题切割成子问题，再用master theorem



前提这是一个sorted array

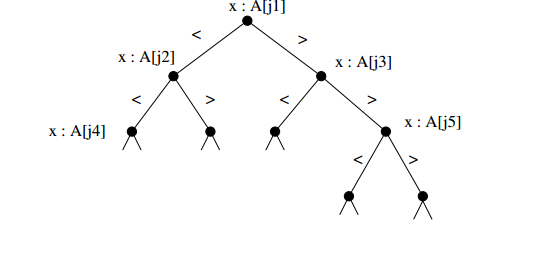
binary search，从i到j之间search x所在，

每次对半分，但只分成了一个子问题，另外一半我们不care

T（n）=T(n/2)+c

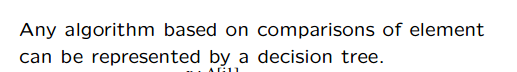
n^logba=1=f(n),第二种情况

Decision tree



每一个node代表array的一个值，如果小于，就往左边，如果大于就往右边，如果等于，那么就是你了

任意比较算法都可以用decision tree表示出来，例如刚刚那个二分法，，每一个node都代表一个中点



comparison算法Depth至少是logn optimal comparison-based search algorithm

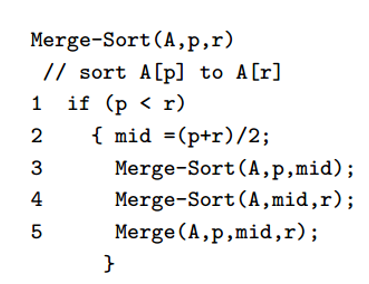


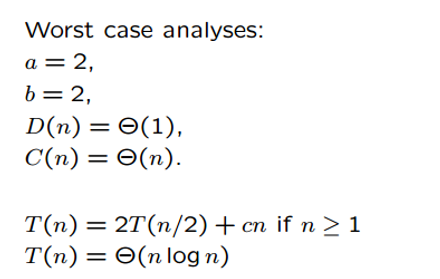
Merge-Sort

关键点：需要额外写一个Merge,Merge需要O（n）的存储空间

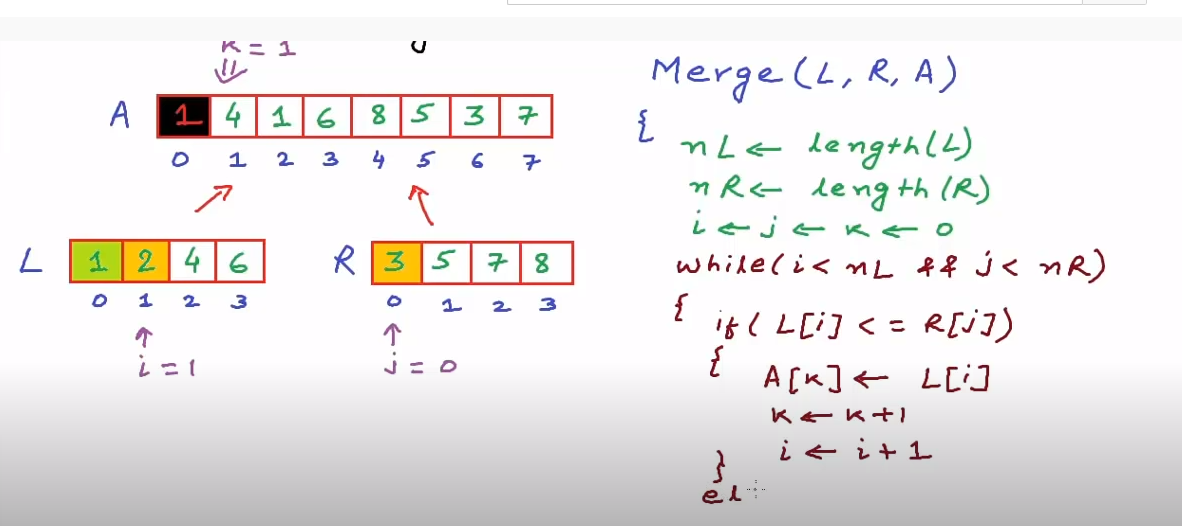
疯狂对半切，一直到底部只剩一个，然后再用merge排上来，因为Merge要求参数必须是sorted的，所以我们要对半切，因为只剩一个的时候算作sorted

稳定nlogn

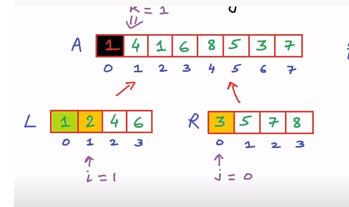




merge



Merge需要三个参数，一个Left,一个Right,一个Array



Left Right各有一个pointer,当前pointer比较大小，小的那个右移一格并且填充到A当前pointer,就一直这样移



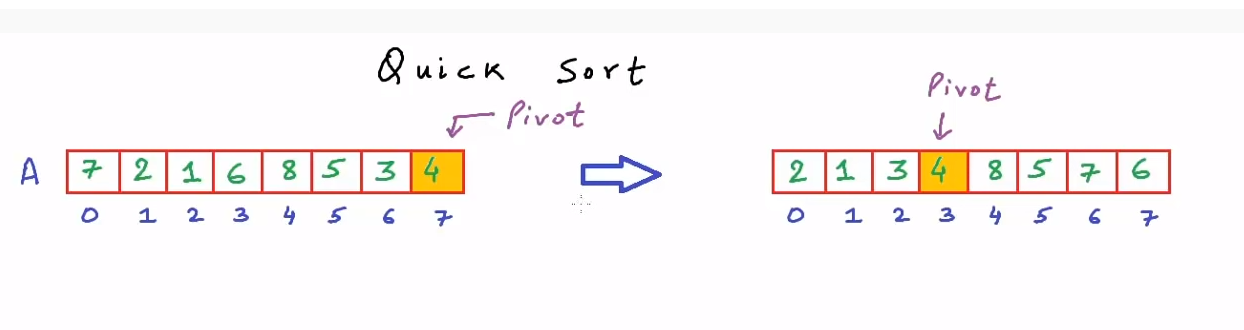
缺点在于merge 需要额外空间

Quick Sort，

Quick sort worst case 是On^2，但是我们可以使用randomized quick sort, 几乎可以完全避免，

优势在于in-place，不需要额外的空间

randomized quick sort有很高几率做到nlogn



pivot,非random里，选最右或最左作为Pivot

partition: 选好Pivot以后，小的放左边，大的放右边 ,A就是输入array，r是末尾，也是我们的Pivot

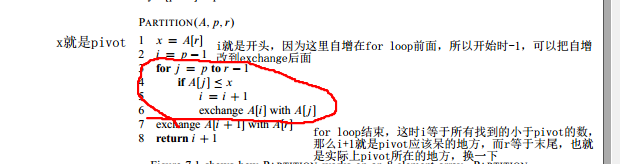
i是小于的当前index，我们要和他换，换完以后加一

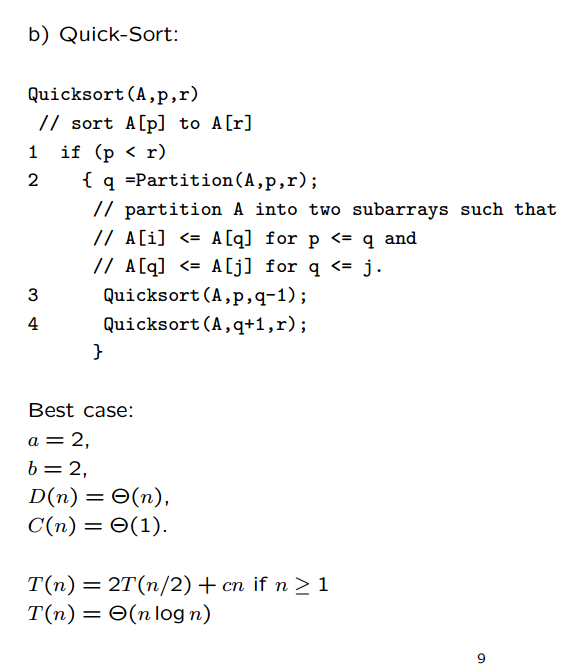
j是一直都在加1，如果大于那其实是不用换的，因为i与j之间就是大于，最后换pivot

**因为j跑得比i快，i指的是所有小于r的，而j是自增的，那么r-i 自然就是大于r的，这也是这个算法精妙之处**

**最后把r到中间**

P是开头





worst case

每次都选取最大值或最小值，那么他就没有把size切割成1/2，也没有切成两份

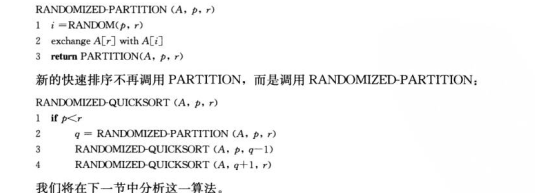
//it is already sorted or reverse sorted// 因为我们partition是取的结尾

//但是如果说左边切了9个，右边只剩一个，我们还认为他是nlogn

randomized quick sort

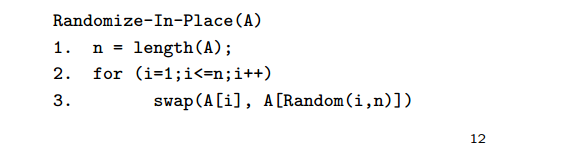
那么，我们可以通过随机选取一个Pivot，或者打乱array，来尽量达到nlogn

可能运气特别特别差，每次都选到最大值，那还是n^2,但即使是这样，问题也是出在随机，我们不必再依赖输入的array

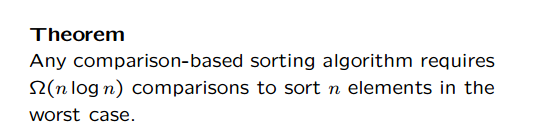


我们先随机p到r之间一个数，把他换到末尾，再进行原来的partition

或者在quick sort前，打乱array



random可以在i与n间选一个数



任何comparison-based sorting需要最少nlogn

Median and Order stastics

ith order statics:一个集合里n个元素里第i小的element

median:一个集合里的中位数

当n时奇数，median是(n+1)/2

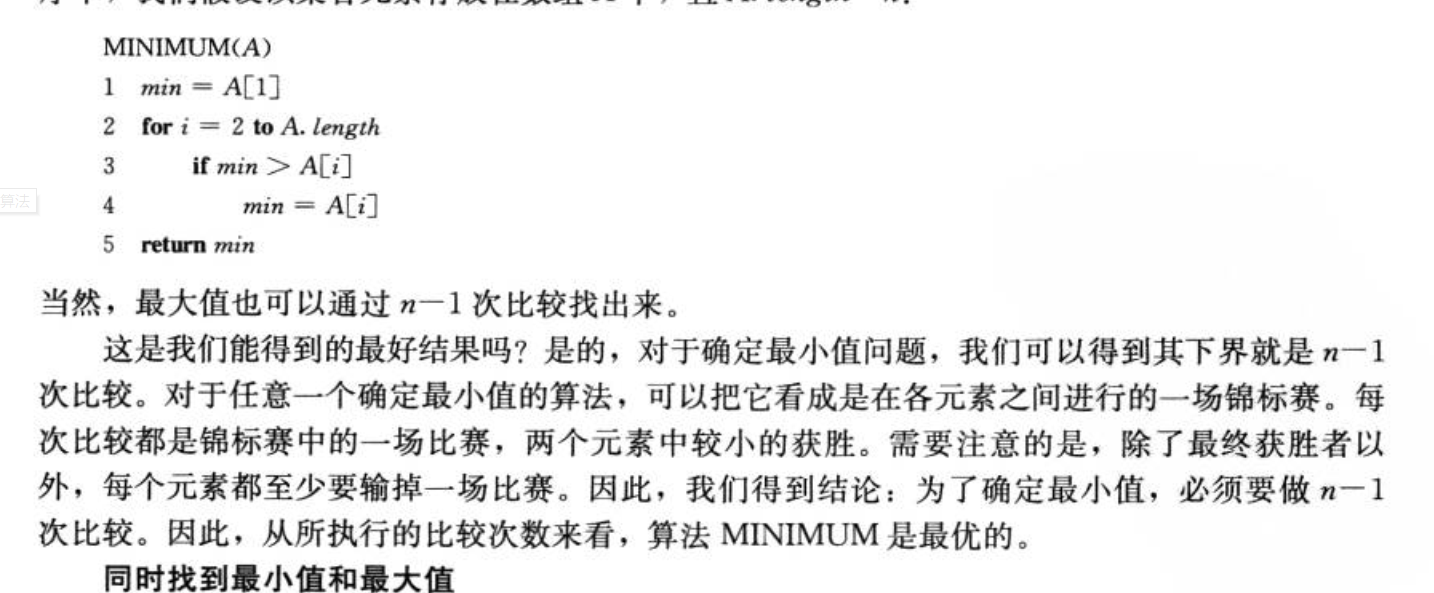
当n是偶数，lower median是n/2

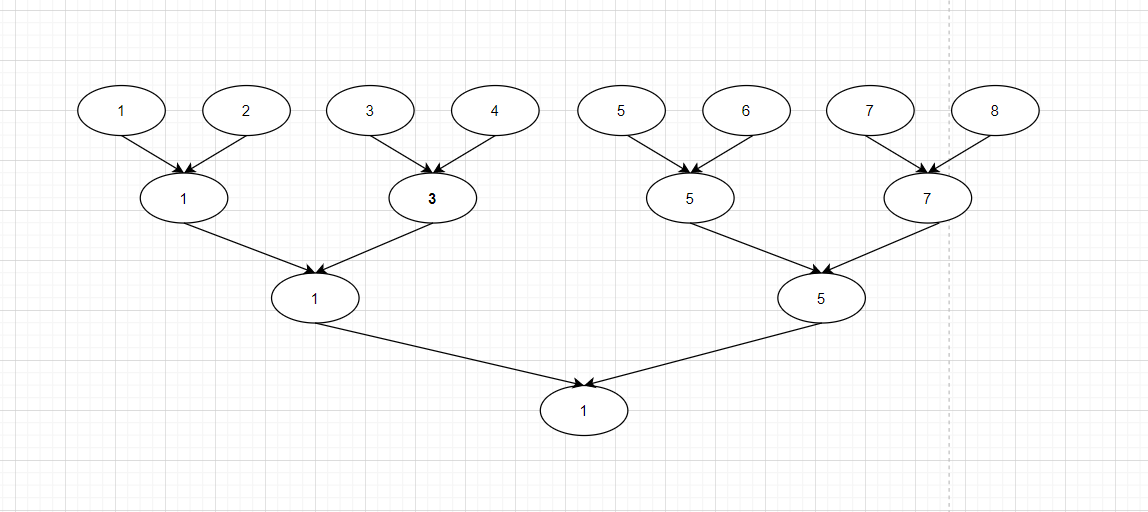
upper median是n/2+1

Selection Problem，给你一个有个n个不同元素的array，找到ith order statistics

第一种思路，直接sorting整个array,那么ith order自然就是index i，这种方法需要Onlogn

但是关于这个问题，实际上我们能够达到On，通过divide and conquer



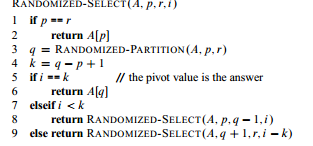


1+2+4+8+…n/2 //geometric

=(n/2\*2-1)/1=n-1

N-1场锦标赛

RANDAMIZED-SELECT,用来找第i小的元素



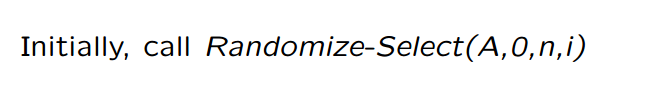
p就是开头，r是结尾，i是我们所需要找的第i小的数

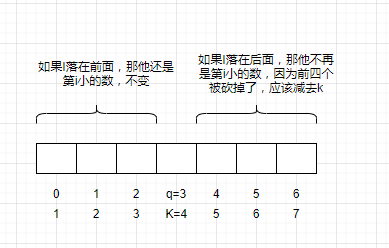
3：randomized-partition是前一章的那个，能够随机选一个pivot，小的在左边大的在右边，q是pivot最终index

4:k=q-p+1,就是index这个Pivot实际上是第几小的数，例如array有六个数，他是第三小的数，那么他index就是2， 2-0+1=3,也就是他是第三小的数//而且这样可以是相对的位置，q是index，而p是subarray的开头

8：如果在左侧，舍去包括q在内的所有array

9：如果在右侧：舍去包括q在内的所有array，而且，我们要找的不再是i，因为舍去了这么多数，他不再是第i小的数，i-k





Randomized select

最差情况是O（n^2）,每次随机选取都是最小或最大，

然后T（n）=T(n-1)+O(n) //pivot所以会减一

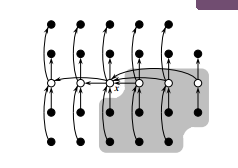
期待run-time是

最坏情况下还是线性的算法

SELECT

第一步，把array所有元素排成5\*n/5网络

每一列称作一个group，最后一列可以不满五个，



第二步：找到每个group（列）的中位数，把这个goup重新排列，中间的就是median,需要的用时就是 O(n), 因为array5的比较时间是固定的，是O（1）,乘上n/5就好



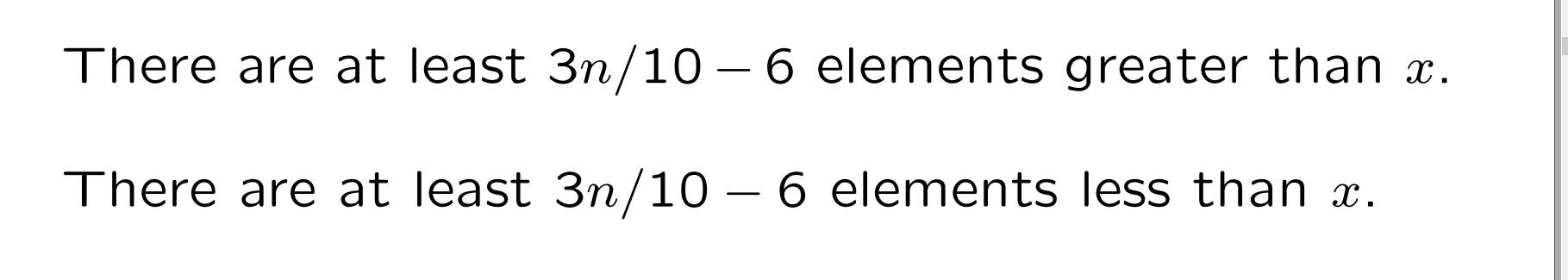
第三步：用SELECT recursively找这群median的median X，需要T（n/5）的时间 ,

//因为SELECT可以找指定第i小的数，自然能找中位数

第四步，把这个最终中位数x作为pivot，开始PARTITION(PARTITION完以后小于的在左边大于的在右边) //以上四步就是每次选pivot的标准

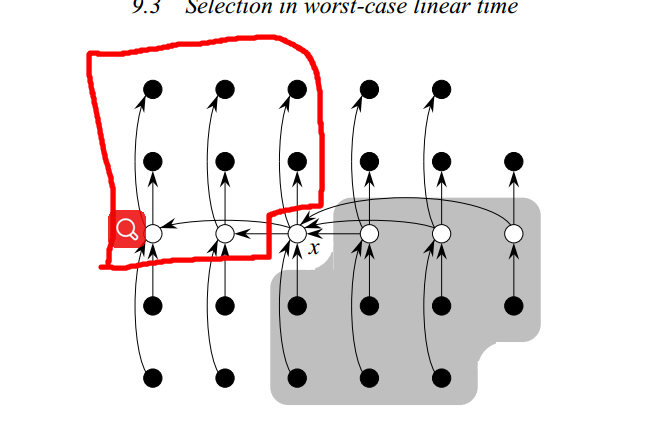
第五步：然后无限循环，如果i=k，return x，不然如果i<k，找i，i》k，找右边的i-k

这样的算法好处在于，

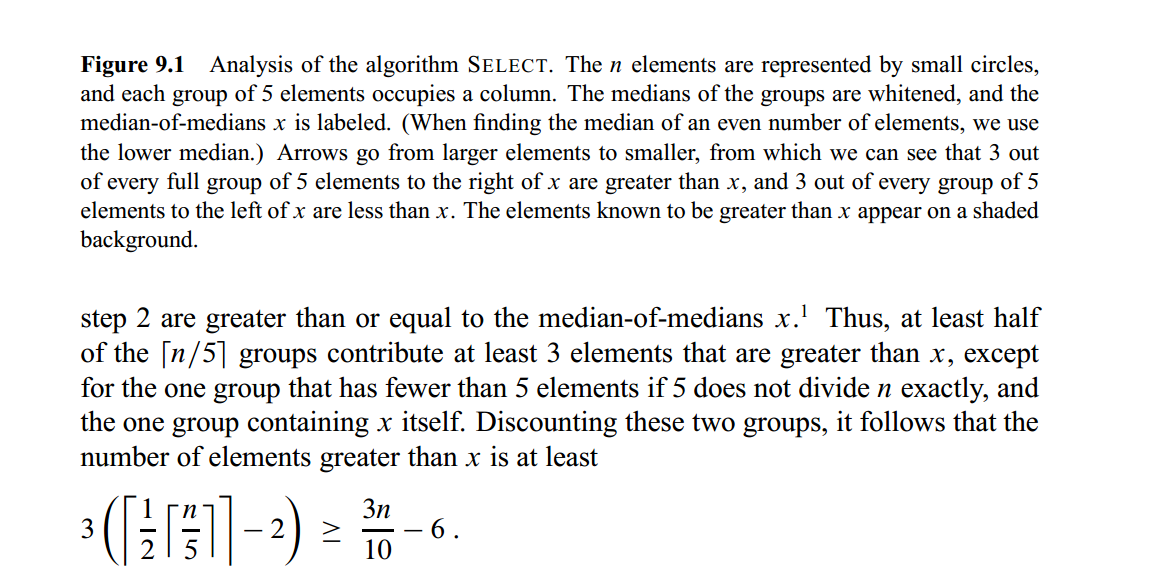


可以尽可能的取到中间的位置

为啥



这是一列的3/5



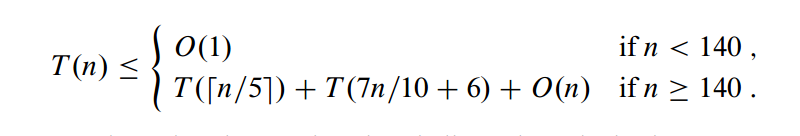
n/5代表有几个group， 一半的group大于x，减二是因为有一列包含了x,有一列不完整。

3代表着group里有三个元素能取到

箭头起始点代表大的，终点代表小的

那么根据箭头的延续，红色区域内的element小于x, 黑色区域内的大于x， 而不在红色区域和黑色区域的白色区域不确定

这就是3n/10-6怎么求到的





这个算法把问题分解成了一个选中位数问题加上更小的recursive问题

T n/5就是取中位数用时， T（7n/10+6）是剩下的部分，O（n）Step124

https://ita.skanev.com/09/03/01.html

Dynamic programming

矩阵连乘

STEP1

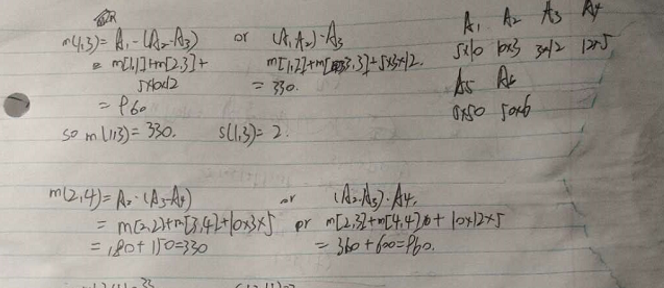
创造两个n\*n表格

mtable 对角线全是0，在上面一层就是很单纯的12互乘，23互乘，34互乘…

s table没对角线，在上面一层是12互乘就是1,接下来是2,3.。。。

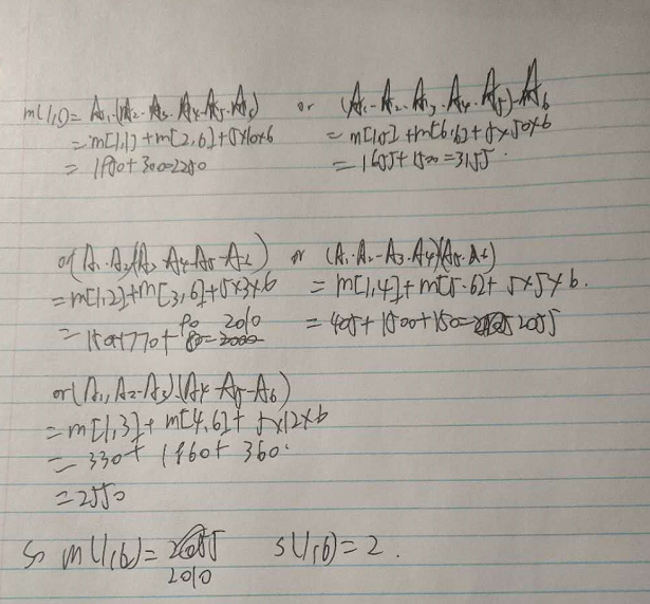
STEP2

一层一层往上面推



注意比较大范围的那种

永远是看做两个多项式相乘，而不要1\*3\*1这种



STEP3

跟着建立完S表以后

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| s | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 |  | 1 | 2 | 2 | 4 | 2 |
| 2 |  |  | 2 | 2 | 2 | 2 |
| 3 |  |  |  | 3 | 4 | 4 |
| 4 |  |  |  |  | 4 | 4 |
| 5 |  |  |  |  |  | 5 |
| 6 |  |  |  |  |  |  |

s(1,6)=2

so (a1\*a2)(a3\*a4\*a5\*a6)

s(1,2)=1 only 2 elements, ignore  
s(3,6)=4

so(a1\*a2)(a3\*a4)(a5\*a6)

so the final answer is (a1\*a2)((a3\*a4)(a5\*a6))

从右上角第一个开始看

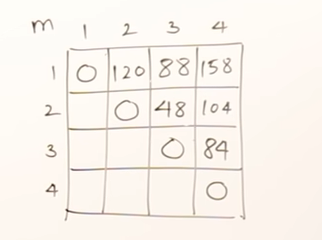
然后做出第一步分割

再分别切割两个多项式

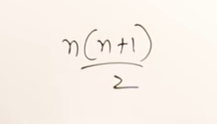
a1a2就切 12，是1不用管，因为就剩两项

a3a6就切36，是4就切a3a4

因为我们要准备一半的表

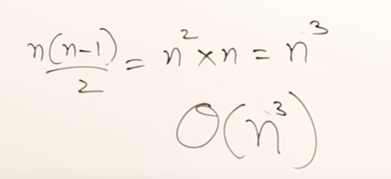


也就是1+2+3+….n的等差数列，

是n^2

但计算每个值我们需要计算可能性，然后寻找最小值

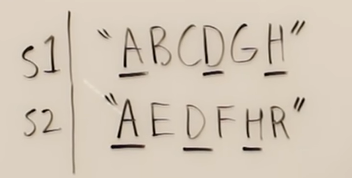
需要n



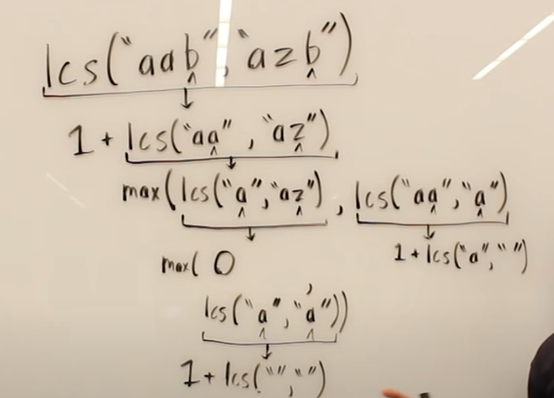
】

2.Longest Subsequence

首先，Longest common subsequence并不追求一定要是continuous的



我们希望算法lcs能够return最长的subsequence



分为三种情况，

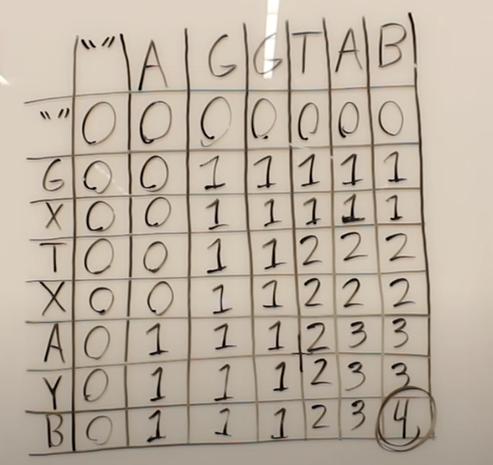
第一种，左右末端相等，这时全部去掉，length+1,

第二种，前面一个数的子选项是无关紧要的，可以去掉

第三种，the last char of last one is unimportant, we can remove it

Then we pick that max of case 2 and case 3

表格法，注意，表格法与上面的内在思路是完全一样的

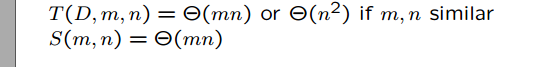


STEP1，第一行与第一列全部填0，因为这是理论1，null string和所有String common都是0

STEP2，从上到下一行一行填，从左到右，就像正常写文章的顺序

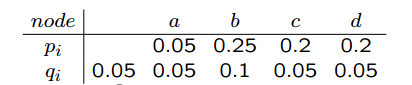
行与列不相同时，往上看一格或者往左看一格，取最大值，//对应定理23， 如果不一样去掉一个，取最大的

行与列相同时，在左上角一格的基础上加一//对应定理1，相同时，加一



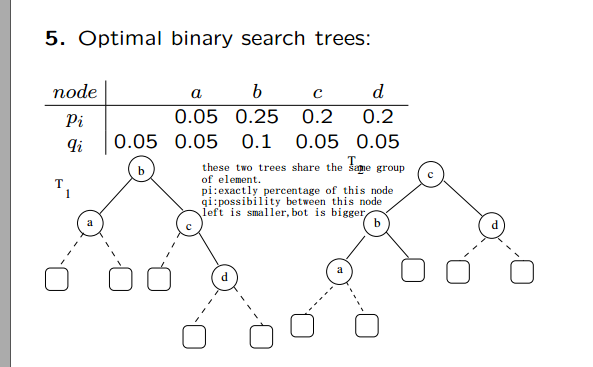
Time complexity与Space complexity， m与n指的是两个String的长度

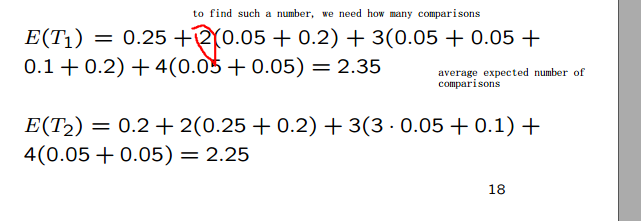
Optimal binary search tree



p是遇到a,b,c，d的几率

q是在这之间的几率，左边是小于，下面是大于

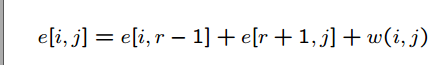




E是期待比较次数，

第二层的数，为了比较他，所以我们要找两次





我们不能用暴力解算出所有可能性，但可以用dynamic programming

e[i,j]:key i 到key j的estimated cost

E[i,j]的cost等于e[i,r-1]+e[r+1,j]+w(i,j)

前面两项代表两个子树的Estimated cost

而W(i,j)代表要加上新的root kr所需要的cost

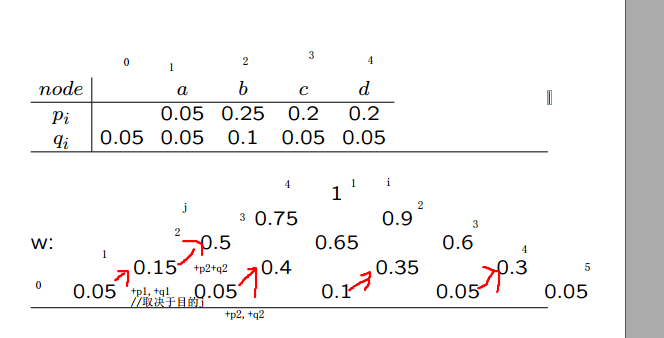
假如加了kr，所有cost都要多比较一次，例如原来是0.5\*3要变成0.5\*4

因此

就是所有可能性都算一遍，可能性分为pk以及qk，也就是正好和之间的 (qk可以看成另一形式的pk)

而w[i,j]又可以计算成

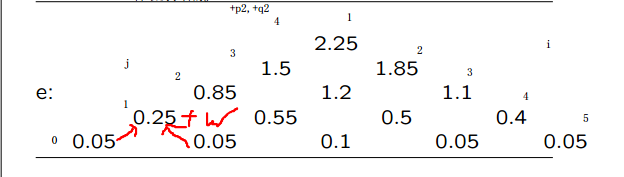




w的计算就是左下角的加上目的所在的j所对应的pj,qj

w计算完毕以后计算

e第二行就是左下角右下角相加加上w



第三行以后就复杂了



i<=r<=j

因此以e[1,2]=0.85为例

可能是e[1,0]+e[2,2] //R=1 = 0.05+0.55+0.5=1.1

可能是e[1,1]+e[3,2] //R=2 =0.25+0.1+0.5=0.85, 选0,8

取了多少R=就在root写多少

**W5 Greedy Algorithm**

贪心算法可能并不是最优解，

分为optimal solution

good approxiamation

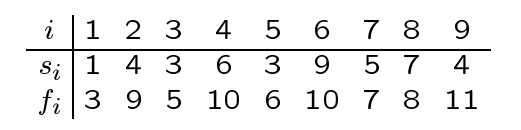
不合适 BST

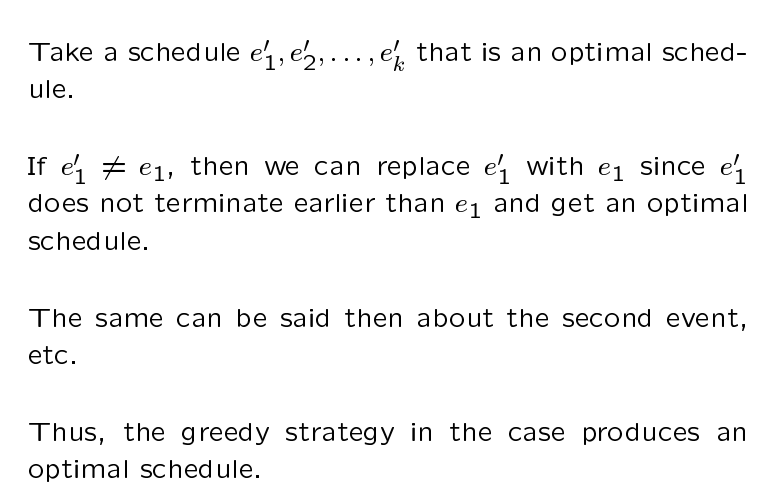
证明最优解3

Activity selection problem

选课问题，每个课有一个start time si与finish time fi

怎么选出最多的课





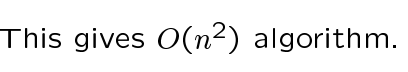
因此结果是最佳的，但这并不代表最快算法

进一步优化

现在算法我们需要每次都搜索整个array找到最先结束的

如果request没有sort，需要

排n次搜n个课程



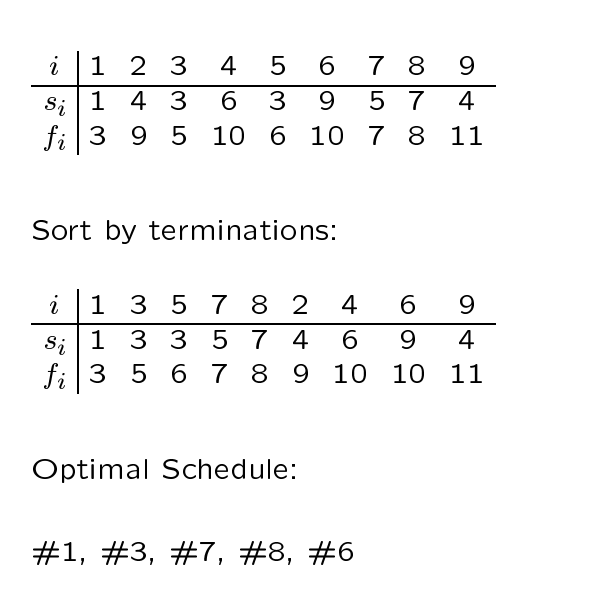
优化版本

优化版本，先用nlogn排序

然后对着sorted使用优化算法，一次for loop 循环就能O（n）搞定



然后**sort结束时间**

根据结束时间选，**越早结束越好 ，这样代表同样的课我们在短时间内结束了，后面上课就更充裕**

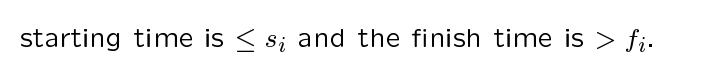
Scheduling变种问题：把所有requerst都用上，最小数量房间，

第一步排序，但是是**按照开始时间sort**

如果没有冲突（第二个的开始时间在第一个结束时间后面），放入空闲房间

如果和所有房间都有冲突，开一个新房间

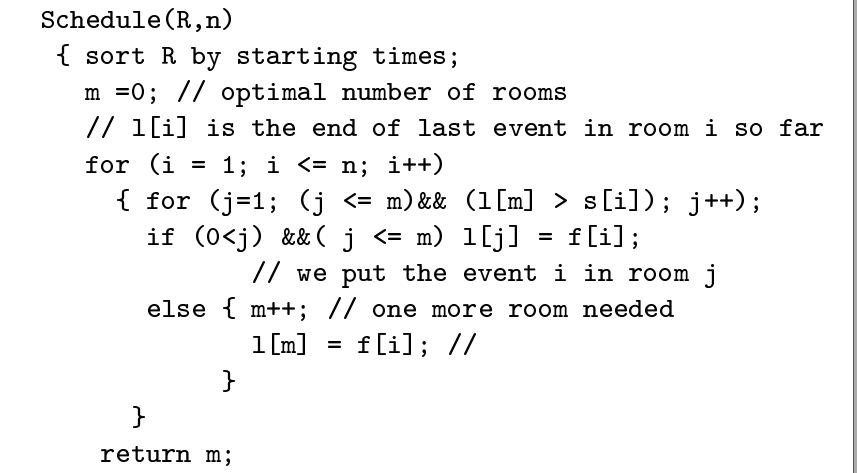
这是最优解吗

假设在同一个时间段，有m个冲突的，现在我们要上新课，那些老课早开始晚结束，

那么无论如何，这第m+1个也是incompatible的

因此我们必须要m+1个房间

因此是最优解



算法有两个for，因此是N^2

我们能进一步优化吗

可以，我们原来是一个一个线性搜索是否有房间满足下个request的

我们可以将request放在最早avaible的房间里

使用一个priority queue来存储每个房间的当前event的结束时间 //这个pq用来implement heapq,它必然会Pop最小的，pop用时logn，因此是nlogn

**核心**

**greedy 算法什么时候work**

**他通过一系列的决定得到一个最终解**

**每一个决定都只能建立在local informaiton上**

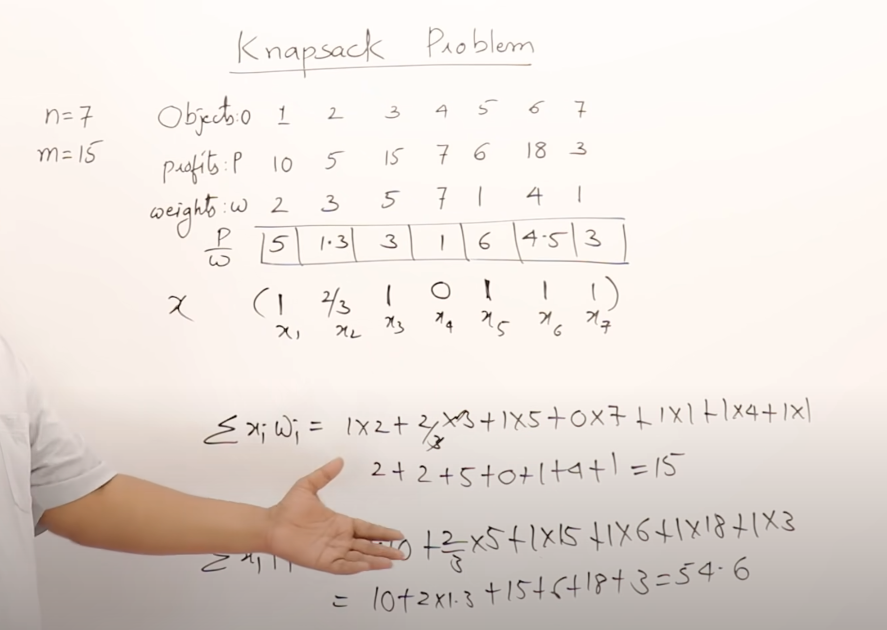
**local decision将创建一个子问题，而子问题和原问题是same type的**

4.Knapsack problem

knapsack problem有两种： 1. fractional knapsack problem ， 2. 0-1 knapsack

这个问题就是给你一个承重固定的包，给你一连串item，每个item有其profit 和重量，求包能装下的item，最大能多少profit

1.fractional knapsack problem, 在这种里面，item是可以分解的，我们可以取部分item，因此我们只要用利益除以重量，求单位质量 利益最高的item，简单地说，就想象成贵金属，啥贵拿啥



p/w 就是单位质量

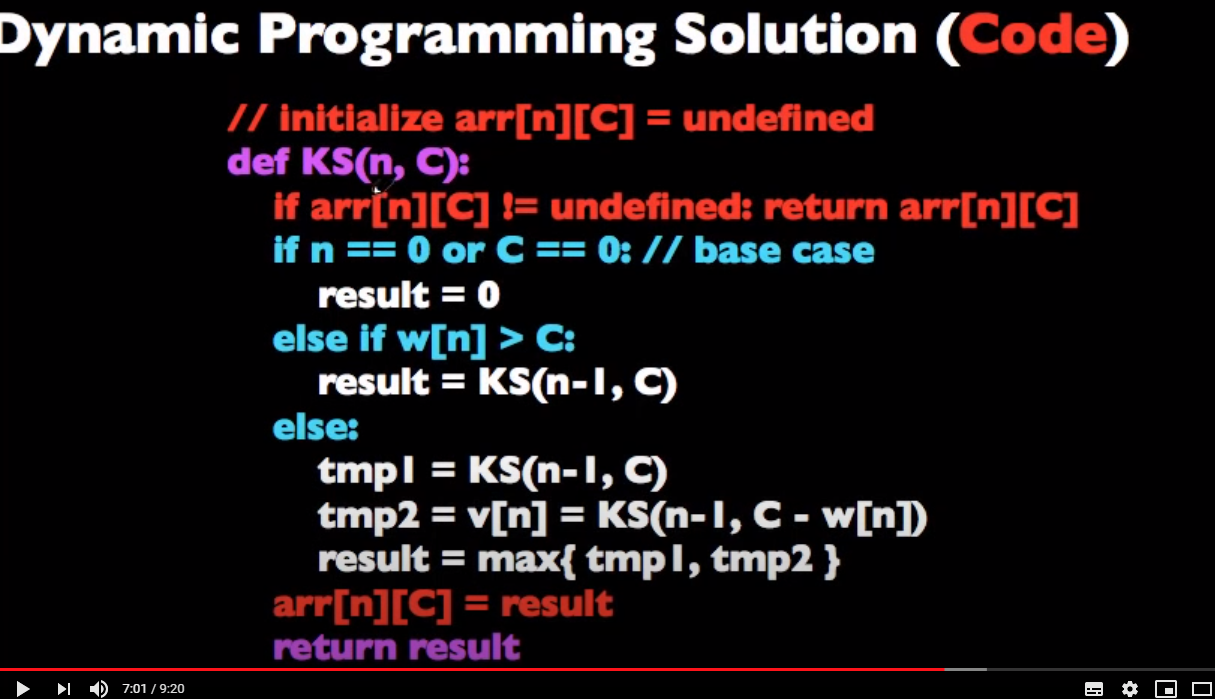
我们列个X表， 自己排序看看啥能拿1，最后质量不够的时候拿部分，质量用完拿0

2. 0-1 knapsack, 在这种里面，item是不可分解的，因此不能用greedy算法，而要用dynamic programming

为啥不能用greedy，

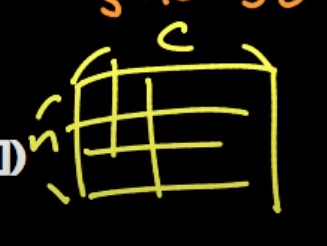
假设我们前面还是拿单价最高的item，如果最后剩下一点空间，不能再放任何东西了

但是我们拿掉一个已经得到的高价item，剩下的空间能放两个中价item，那显然中价item这种拿法利益更高，greedy不是最优解



dynamic版本

就算所有的排列组合全部算出来，也就n\*C种

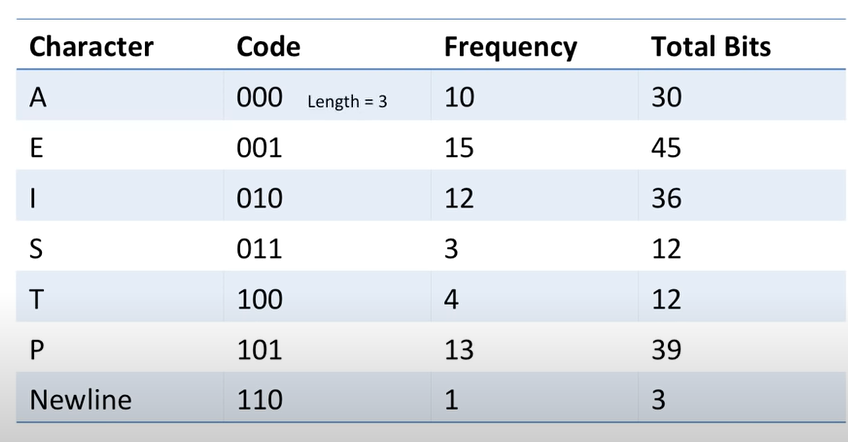


我们建立一个表，把各种都填进去，

Huffman codes

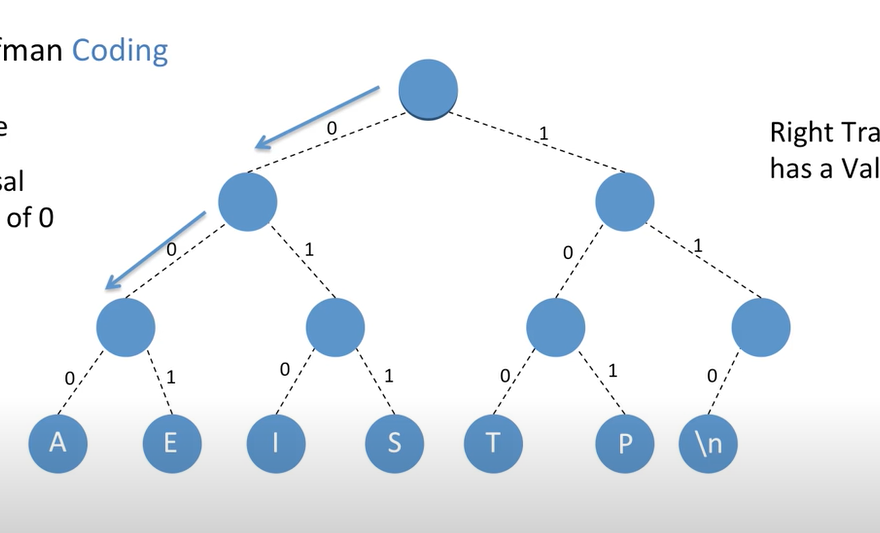
用来在不损失信息的情况下压缩文章大小，

通常来说，我们都用一套fixed-length code来表示char



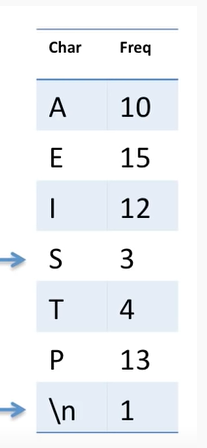
例如这个，我们都用length为3的code

构建成树的话左手边是0，右手是1



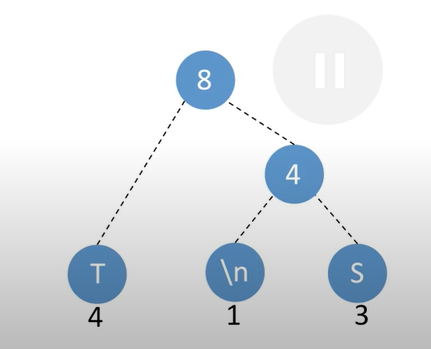
Huffman code的核心思路就是频率比较高的char，用简短的code表示，频率比较低的char，用长的char表示

思路：

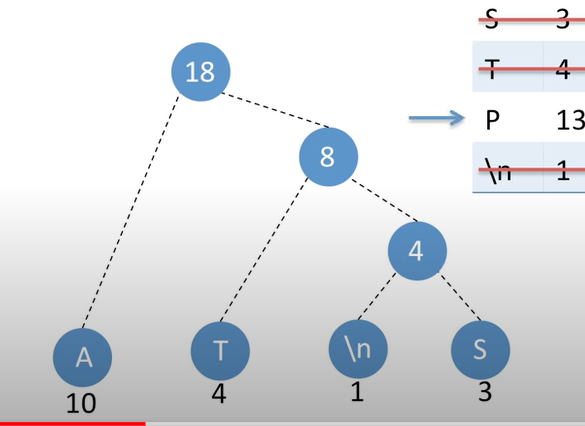
第一步先取两个最低频率的char

以他们为基础构建树，node相加作为父Node

第二步选第三小的，和我们构建的tree链接

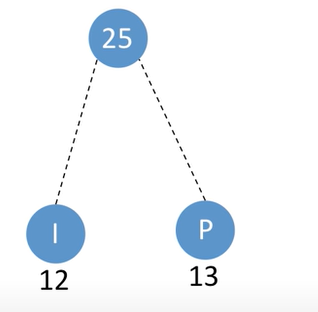
‘

只要先前一步生成的树的root<=接下来的node，就玩命添一个

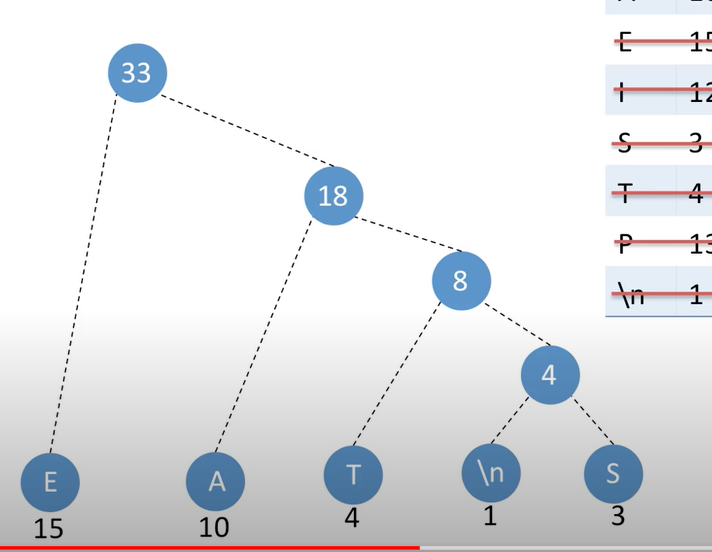


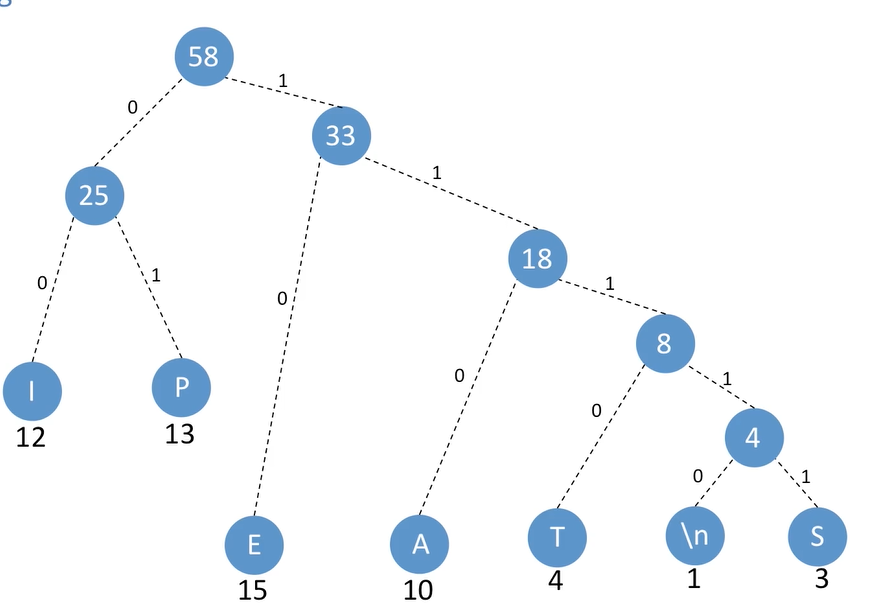
这时是剩下12 13,15，而18比他们大了，不能再添了

挑最小的两个，构建新tree

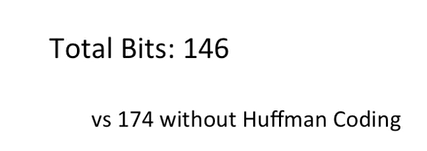
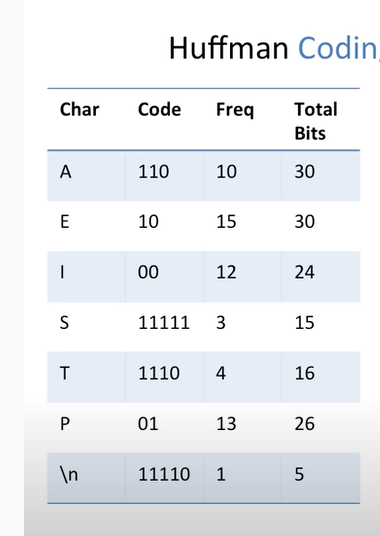


最后只剩下一个的时候，attach到root更小的 subtree //其实attach到那边都无所谓，，因为如果连接的时候，他们都是直接到root的，所用的code-length相同



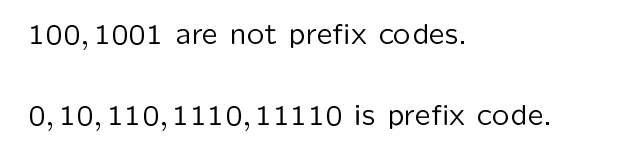


链接构建新tree以及对应code

压缩成功

Prefix-codes：前缀prefix不重复的code

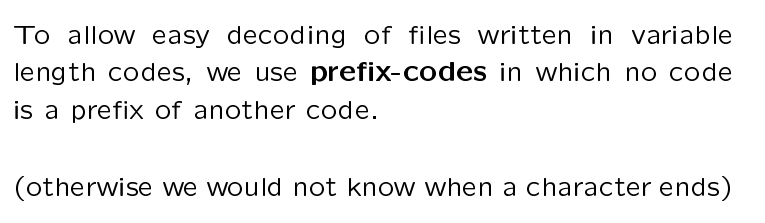
例如



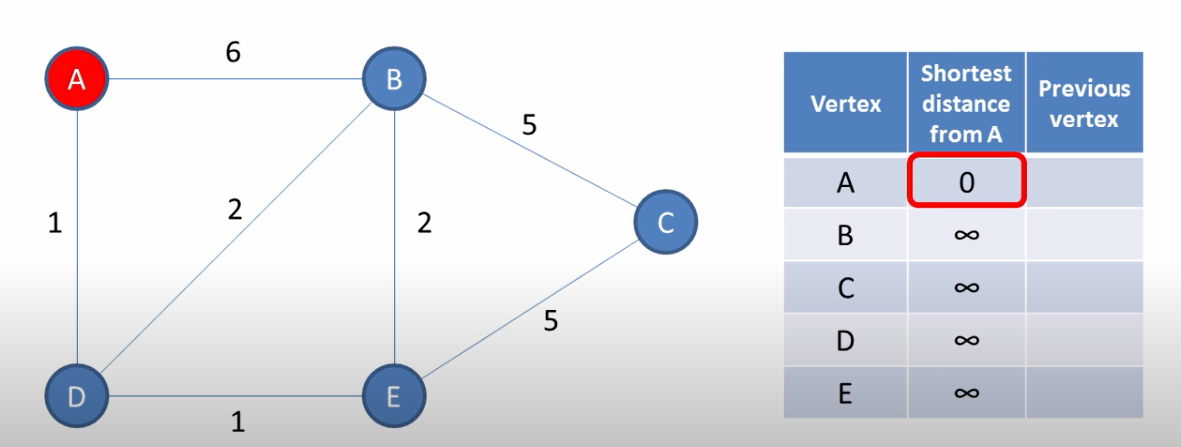
我们必须要使用prefix-codes否则我们不知道什么时候结束这个char

我们确保prefix code的方式就是使用full binary trees，也就是说char是leave, code=path to leave

//full binary tree代表每个node都有两个子node



**Dijkstra Shortest Path**



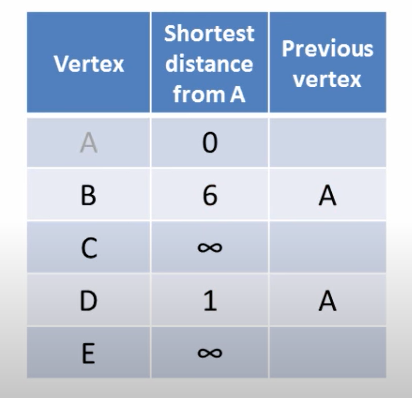
一开始设置成unvisited

除了A以外都是0，

然后找到当前最小的shortest distance from A，设置成visited

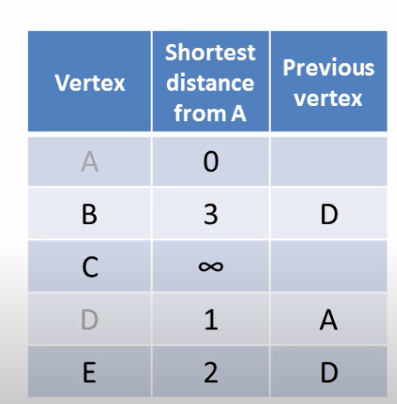
更新这个点周围的unvisited

第一个点自然是A，周围的是BD



第二个点就是距离最短的D，D设置成visited

周围的点是 BE //A已经是visited了

 更新BE

一直重复循环