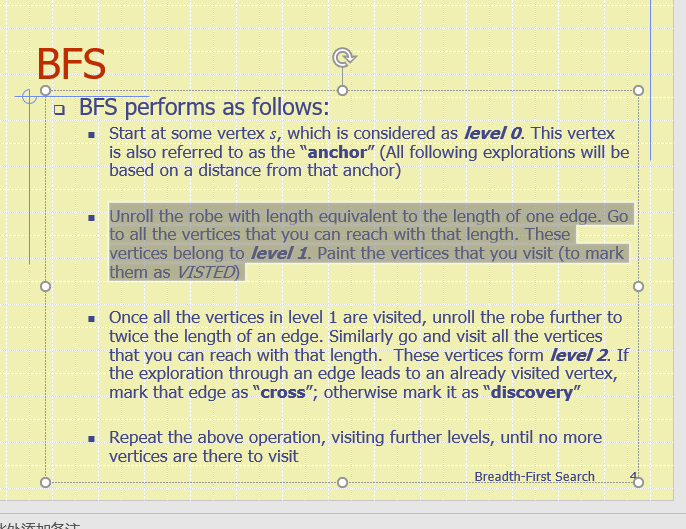
BFS:

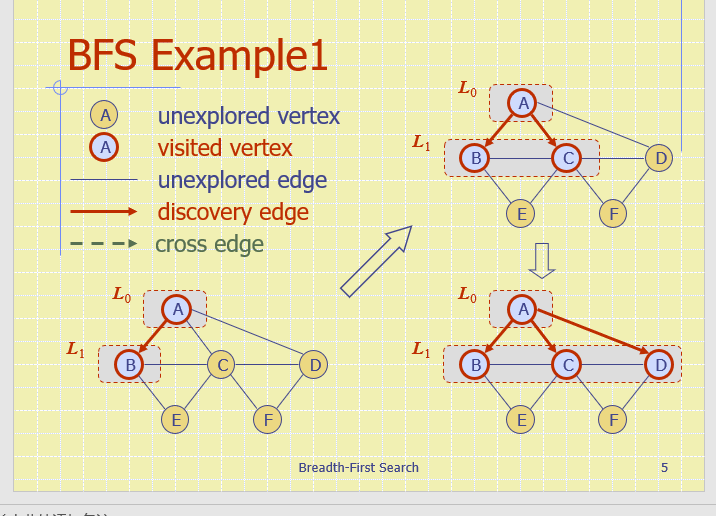
从vertex s开始，被认为是level 0,这个vertex也叫做anchor锚

unroll the robe(意思就是youtube里的exploration,往外展开一格)，去往所有能达到的点，这些点属于level1.把这些vertice设置成visited

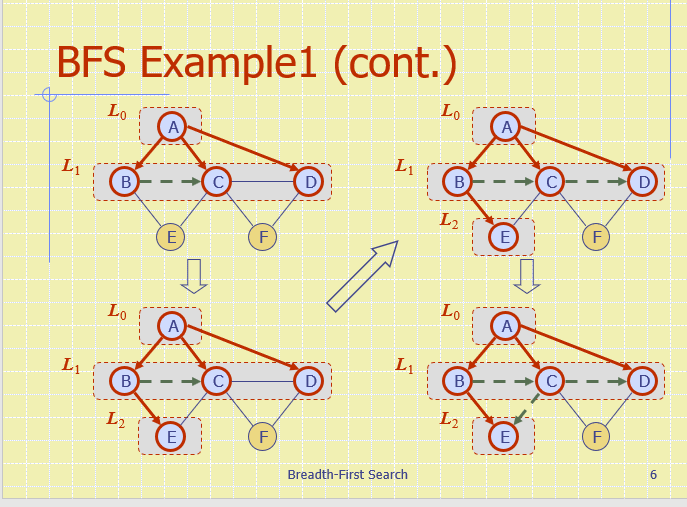
当所有level1的vertice被visited过后，以anchor为原点往外unroll the robe两个edge（实际上确实也是这样的，但是在编代码时，是从level i往外延一格）,这些vertice形成了level2,在往外延伸的时候，如果edge指向一条visited vertex，这个edge设为cross，如果这条edge指向一个新vertex，那么那么就记作discovery

repeat



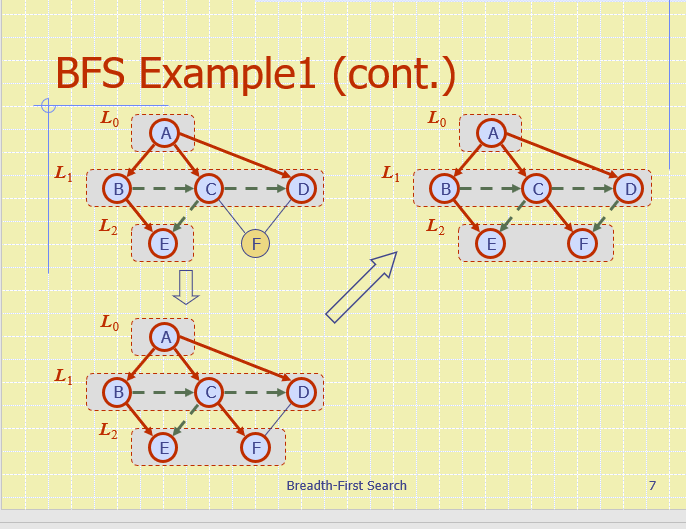


L1

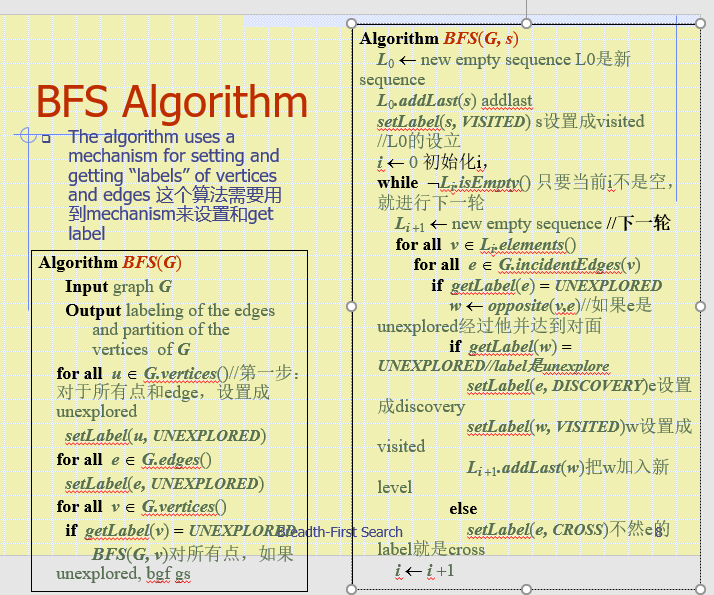


L2，注意L2要标注所有能达到的edge，包括cross，

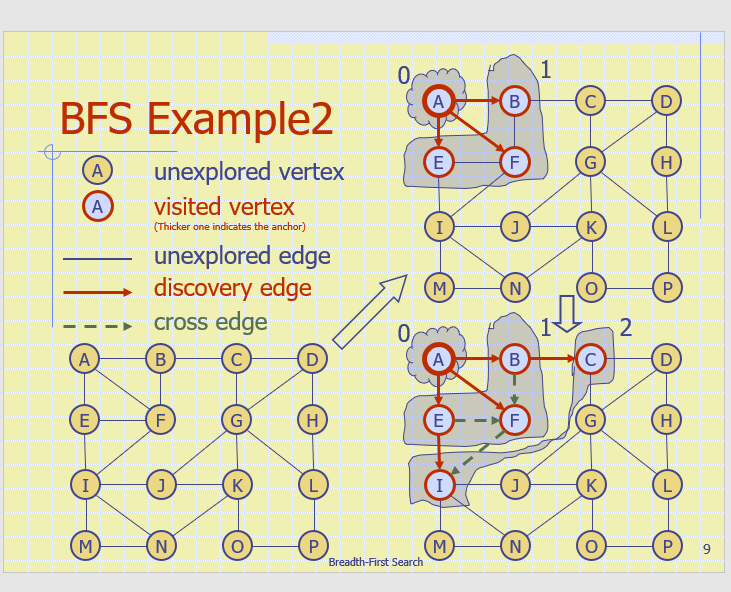
遇到新的点叫做L2



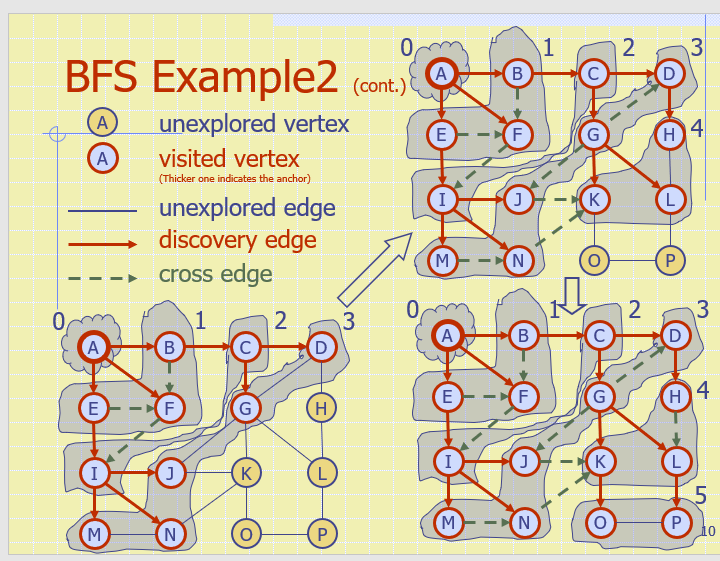
BFS算法



注意是从当前level往外延一格而不是从Level0



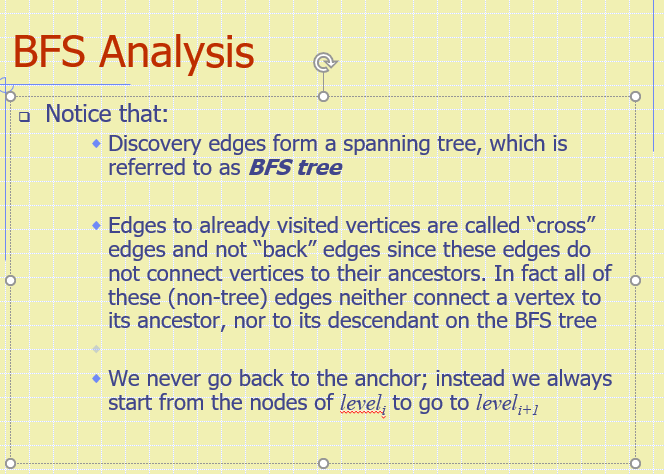
注意了每一次都要设置路径。不仅是点



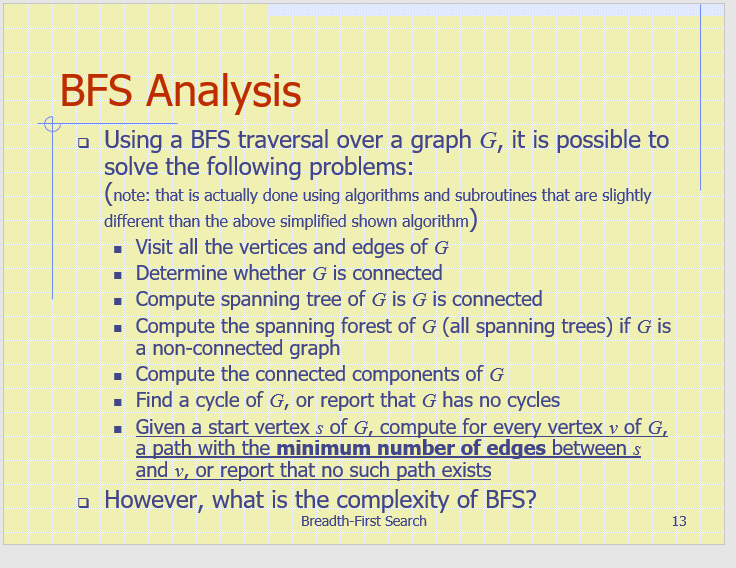
最后discovery edge(实心edge)会形成一个spanning tree，叫做BFS tree

通向已经visited过的vertice叫做cross，而不是back，因为这些edge并没有把这些vertice与他们的ancestor相连，事实上这些cross edge 既不与vetex的ancestor相连，也不会与BFS 树上的后代相连

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，

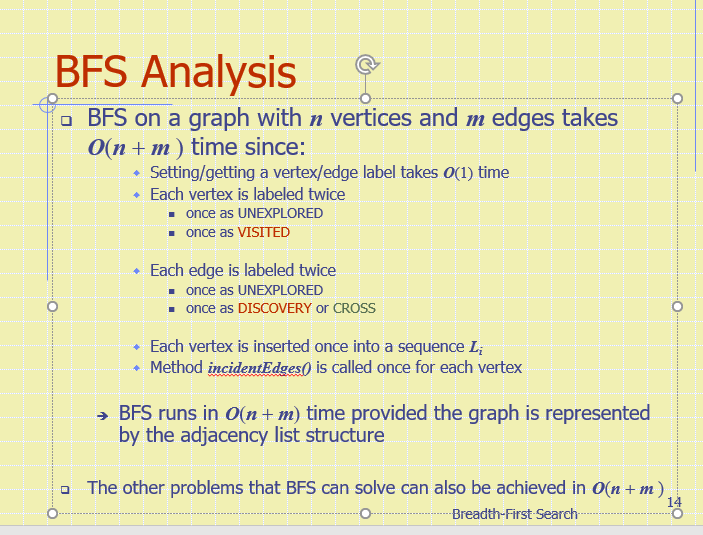
得到一条spanning forest 如果G是non-connected

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,



BFS的性质

Notation:Gs:s（起始点）所在的connected component

性质1：BFS(G,S)将会遍历所有vertice与edge

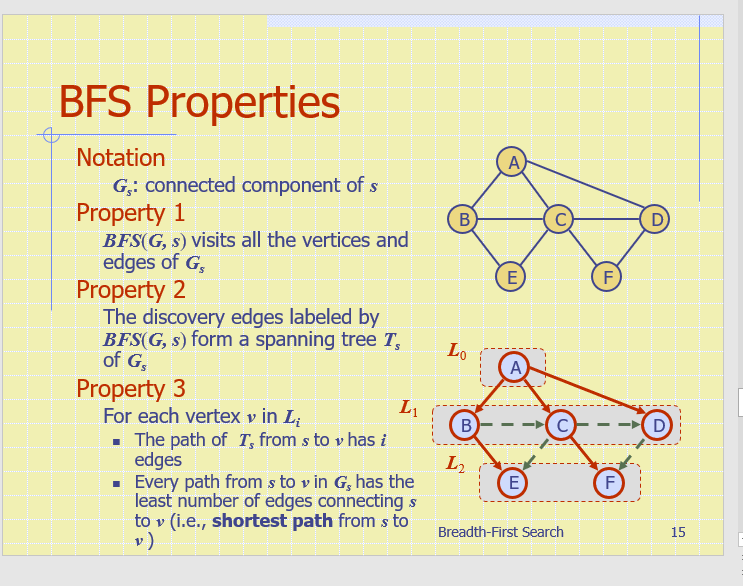
性质2，discovery edge将构成一个spanning tree

xingzhi13, 对于每个vertex v

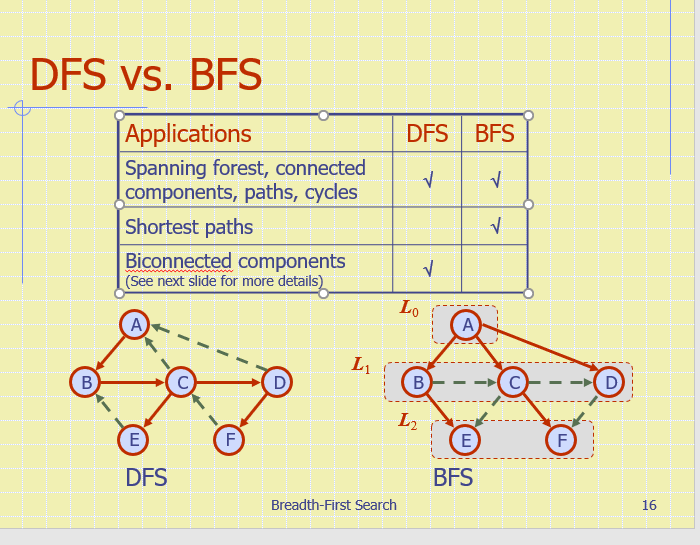
假设这个v除于level i

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)



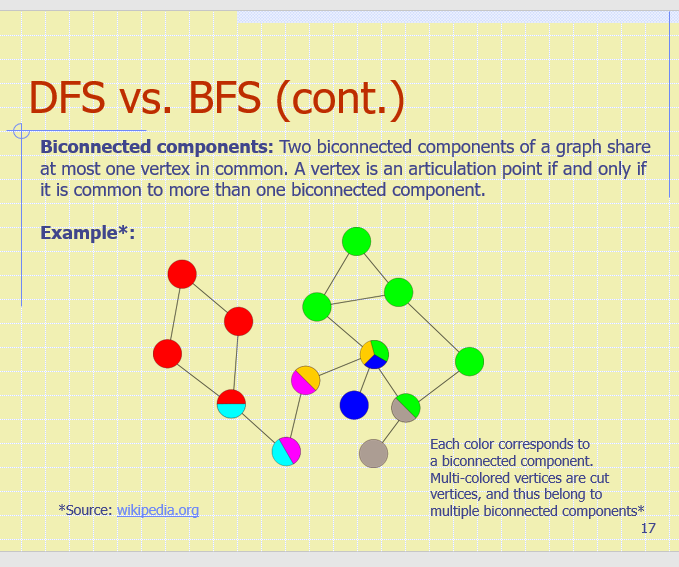
dfs与BFS区别



BFS能找到最短路径，DFS可以biconnected component

biconnected component:两个biconnected component最多有一个vertex重合，

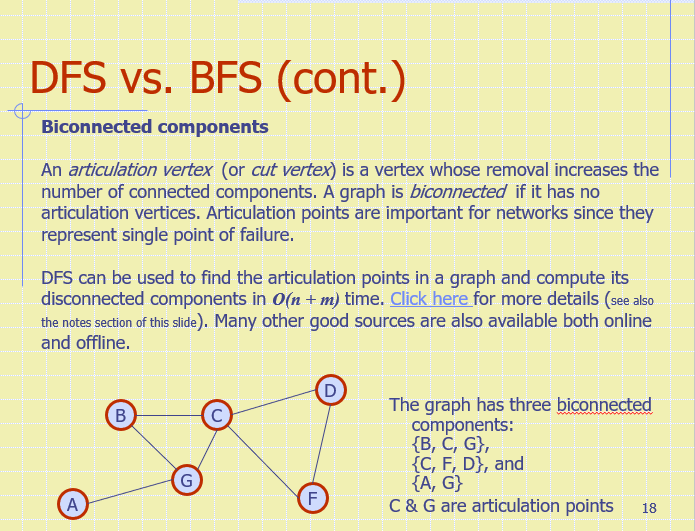
一个vertex被称作aticulation point当且仅当他属于多个biconnected component的时候



每个颜色代表一个biconnected component, articulation point有多个颜色

一个articulation vertex是一个vertex——如果你移去了他，就会增加connected component的数量，一个graph是biconnected如果他没有articulation verticle。

biconnected component的特性是两个点之间至少至少有两条路径，删去任意一个顶点并不影响其他顶点是连通的



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

