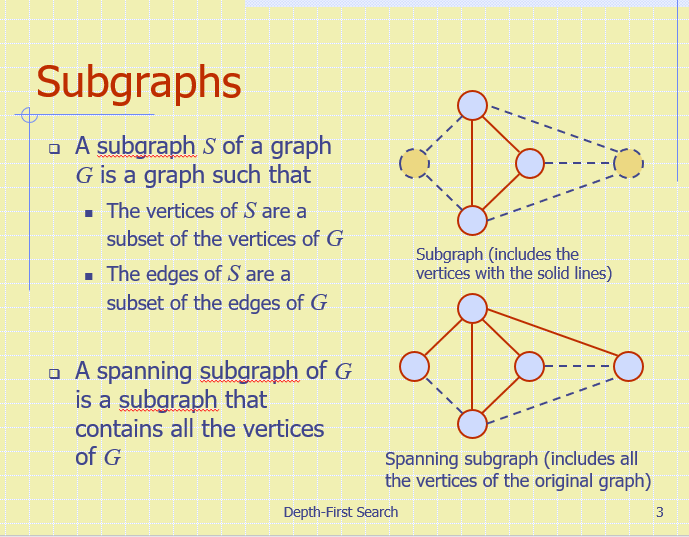
前几张的复习，

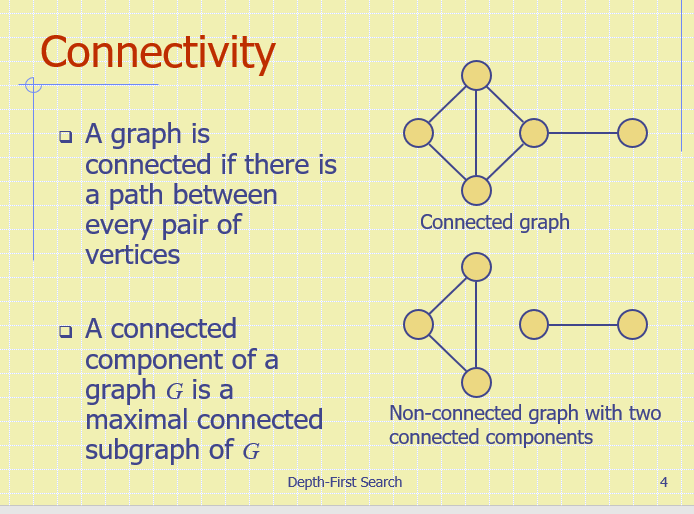
subgraph是vertice与edge的子集

spanning subgraph包含所有vertice但不一定包含所有edge



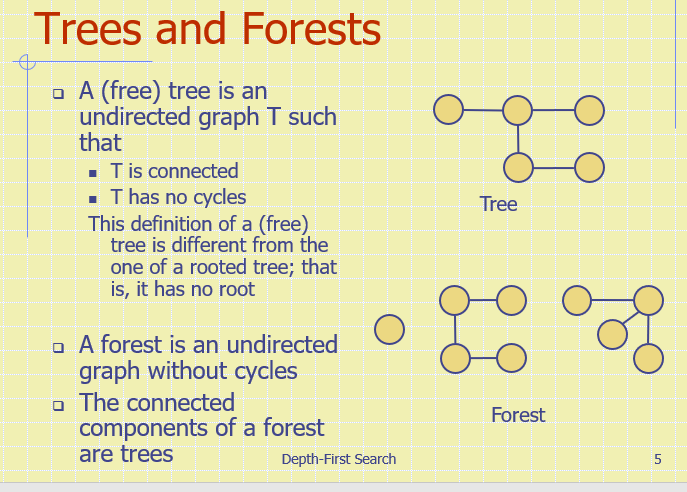
connected:每一部分之间都有path

connected component， 最大的connected subgraph



free tree:undirected graph/connected/nocycle/no root(free)

forest：undirected graph/nocycle/connected components are trees

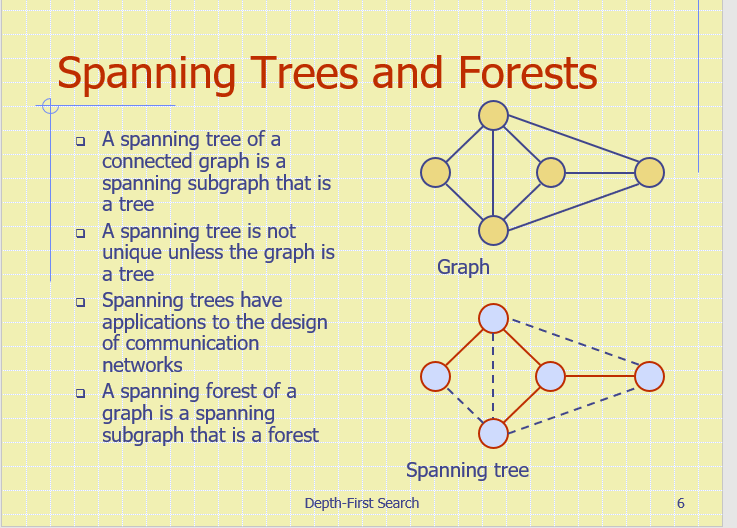


Spanning tree:

即是spanning（包含所有点，）又是tree（没有cycle并connected ,no direction）

Spanning forest

即是spanning又是forest

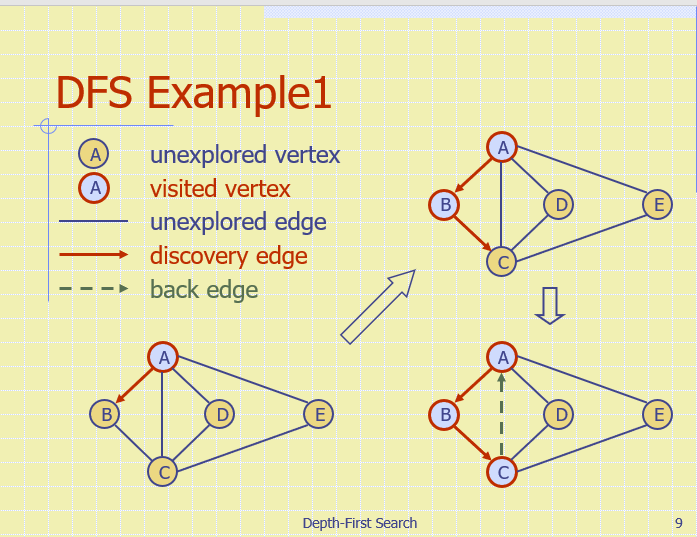


graph traversal

traverse the graph：visit他所有的端点

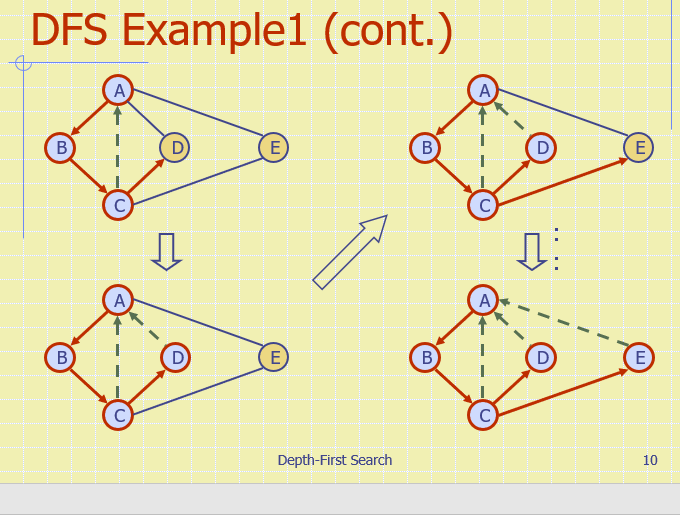
DFS:depth-first search，traversing a graph的一种常见手段

DFS就像在走迷宫，每当你经过一个点 mark一次，当你遇到一个unmarked点，mark it，然后进入他的next node之一，如果你遇到了之前遇过的点，那么回退到一开始的点，

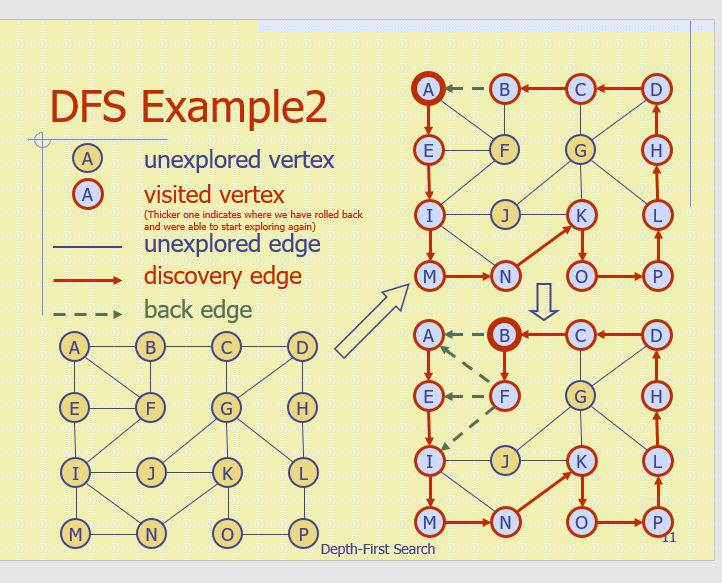


从任意一个点开始

实际路径叫做discovery edge，还没开始的路径叫做unexplored edge，如果路径两头都已经被visited了，那么这条路径叫做back edge

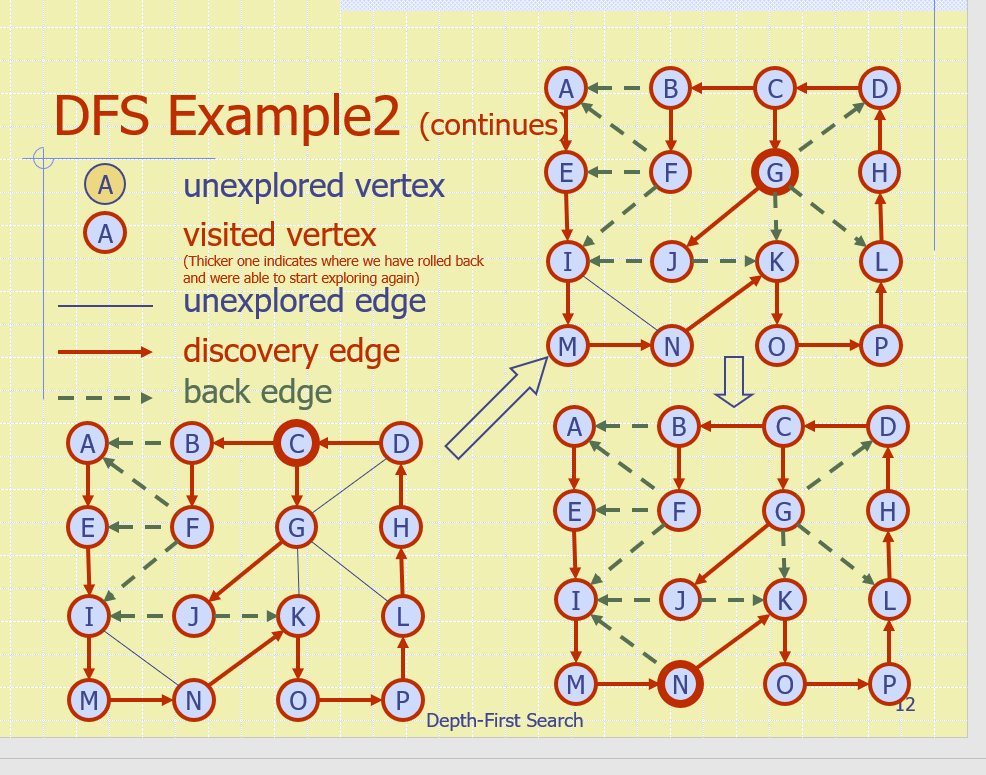


D遇到A是back edge,回退到C，然后再CE

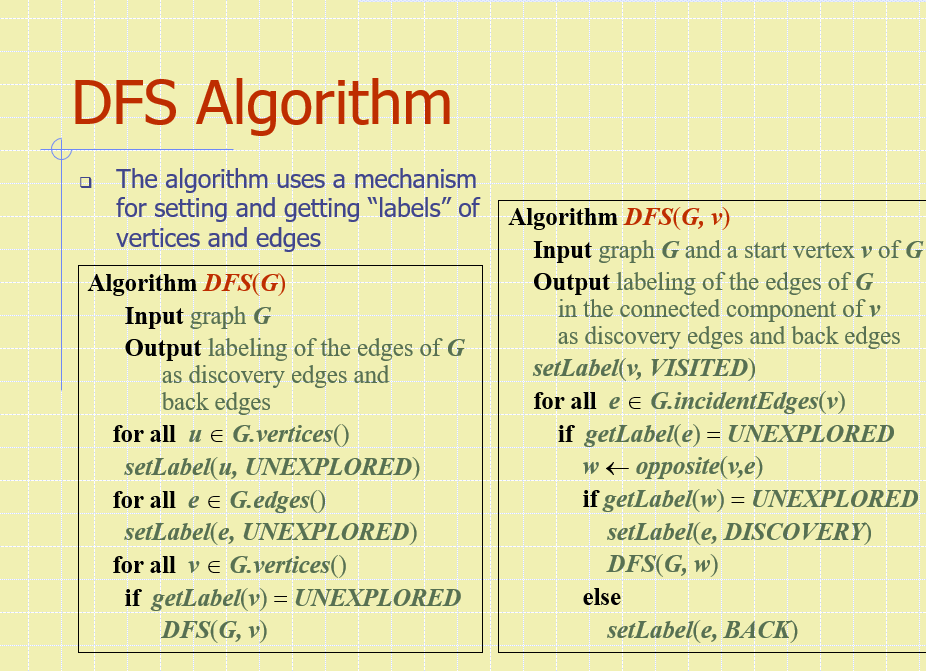


较厚的一层visited它代表此时位置

一路顺风到B，然后继续往前到A，发现已经visited过了，回退到B，尝试别的，发现F



F不管AEI都是已经visited过了，就会回退到B，B也都visite过了回退到C，C开始G,J



DFS算法，首先，每个vertice与edge设置成unexplored

对于每个vertice，如果没有被explored过，DFS (G,v)。不是recursion当前函数，参数不同

DFS(G,v):作用是标记edge是discovery edge或是back edge

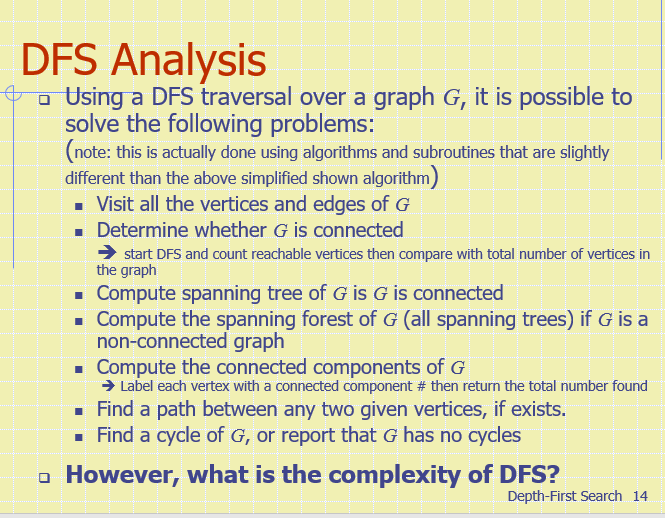
对于每个edge，如果还是unexplored状态，

对输入V通过这个e求他的对面端点w  
如果w是Unexplored，

那么这条路径就是 discovery

循环，以w为端点再DFS

如果w已经explored了，说明这条edge两端都已经是用过的点，那么他的label就是BACK



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

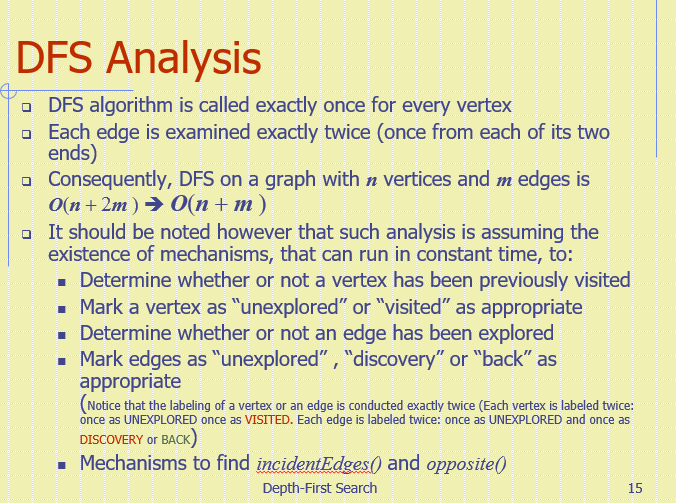
可以决定·G是否CONNECTED(开始一次DFS遍历，然后看得到的vertice和总vertice比较)

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分析

上面这些额外的mechanisms需要额外的space也许会影响DFS的运行时间

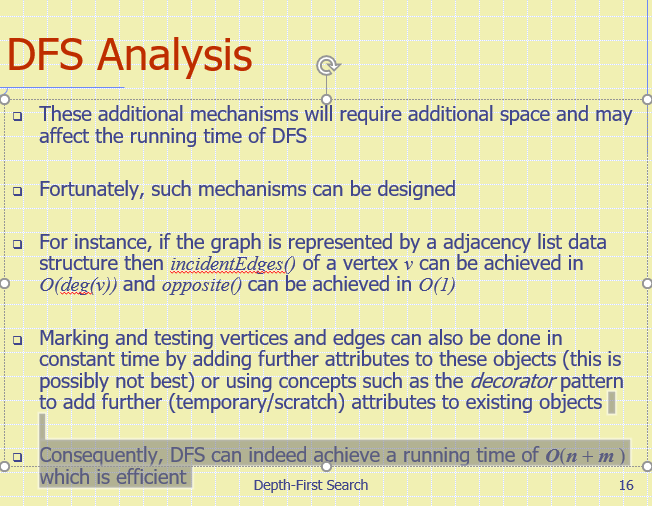
幸运的是，这些mechanisms都是可以通过设计改善的

例如，如果graph是adjacency list data strucuture，那么incidentEdges()的用时是O(DEGv)

opposite是O（1）

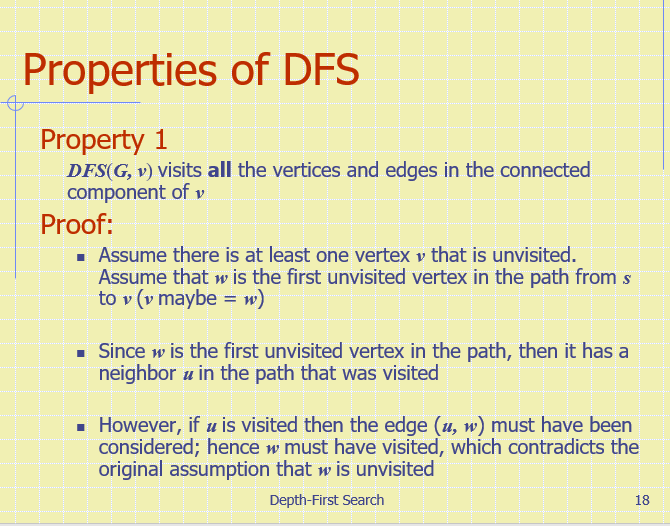
测试与mark vertice与edge也可以在constant time内完成，只有给这些object设立时加上额外参数或者用decorator pattern对已存在的object加上额外参数

最终DFS的时间就是O n+m



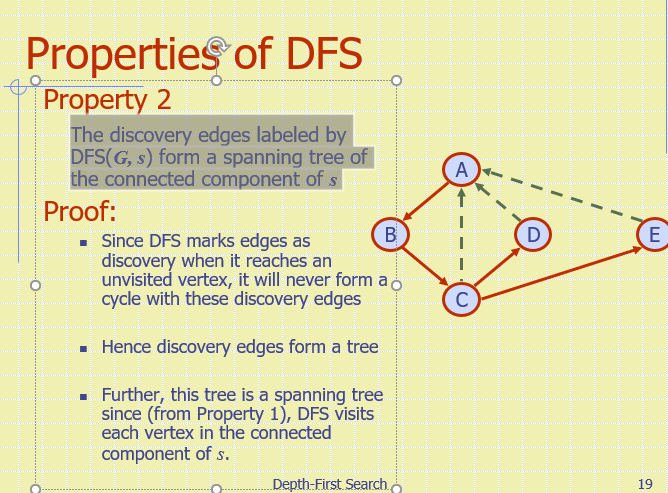
DFS的性质们

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

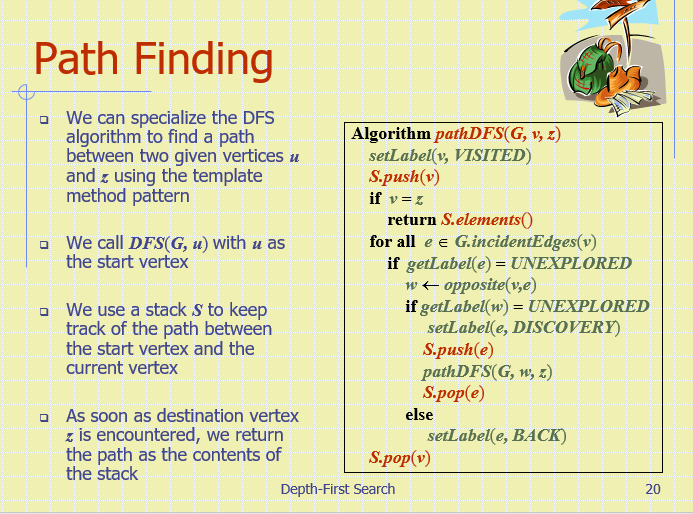


2.DFS(G,s)

由DFS（G,S）发现标记的edge将组成一个spanning tree



Path Finding



我们可以设置一个特殊的DFS算法版本来找到两个vertice uv之前的path

我们用一个stack S 来跟踪start vertex与current vertex之间的path

当我们遇到z的一瞬间，我们就return path，以stack 内容的形式

算法翻译，

把v设置成VISITED

PUSHv

如果v=z,returnS.element

对于v的所有incident edge遍历

如果edge是unexplored，w就是v经过e得到的vertex

如果w是unexplored，那么就让e的label为discovery

push e,

recursion GWZ

POP E

如果w是explored，把e设置成Back

最后pop v

总结：在一直找到z之前，他第一步会把起点vpush到stack上，

情况1.如果v=z了（到终点了），return S.elements（会把整个当前S的element return）

情况2。如果v不等于z，那么就找到一条可以通过的edge，并且把v设置成w，stack会push这条path，最后path是v,e,w,e,.....,遇到终点的一瞬间会return所有element

Cycle Finding

我们也可以用DFS来寻找简单cycle（没有重复点或edge）

我们用S stack来记录start点与当前点的路径

当我们遇到一个backedge的瞬间，我们return这个cycle，这个cycle是顶部到vertex w的部分

算法翻译：

