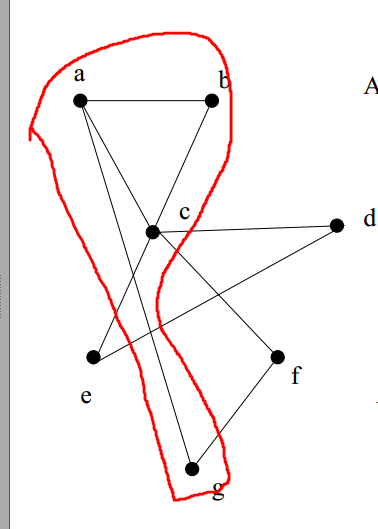
W7:

BFS, explosion形式

一层一层往外扩

以A为中心点

到a距离为1的一圈

**由queue构建**

以a为开头，找他周围一圈，bcg,按照先小后大顺序插入

queue:bcg

check b,skip

queue:cg

check c: def



DFS:

**由stack构建**

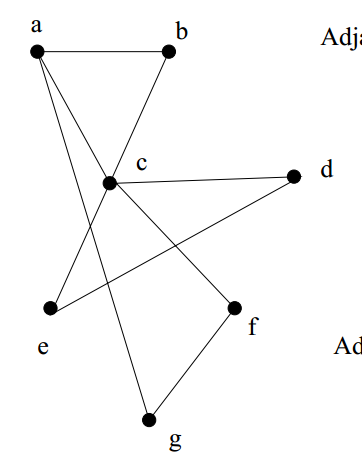
全部点都设置成unexplored

a点开始，放入stack, a设置成explored，随意选择下一个unexplored.并放入stack,设置成explored...

直到遇到一个点，他周围没有unexplored，那么把这个点pop出stack,（回退到前一个点）

如果有unexplored,继续我们的算法，如果没有，再pop

直到遇到有Unexplored点





abcde都很正常，e这时cd都是explored,pop e，stack的顶为d,还是不行,pop d，stack点为c，f,g OK

Topological sort

他只针对DAG directed acyclic graph有效，

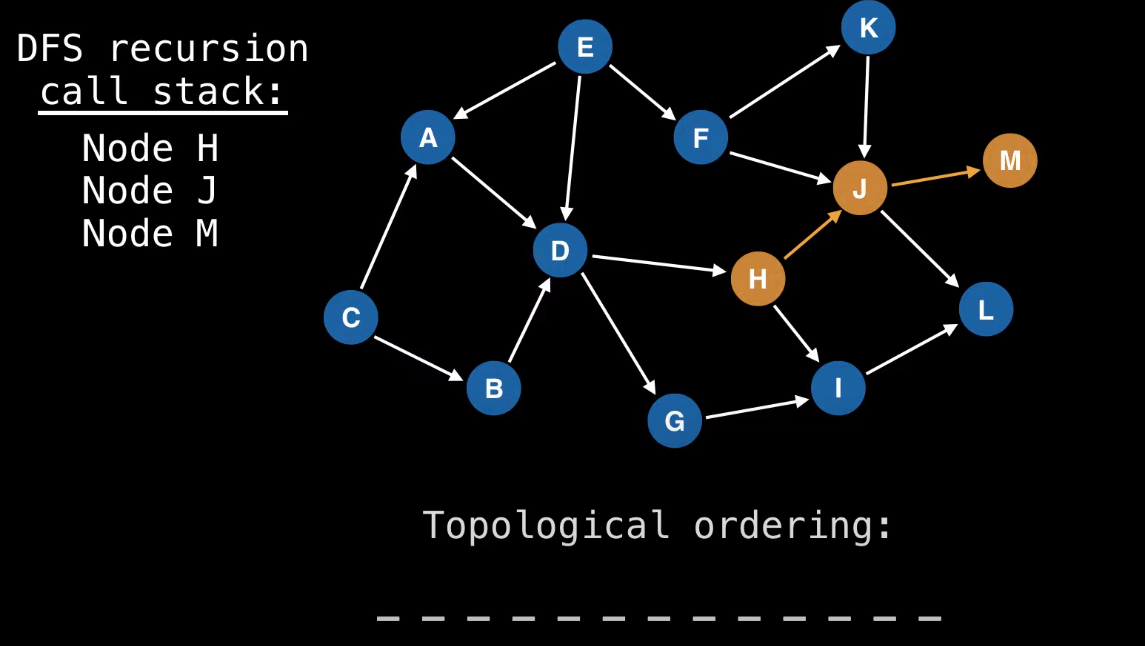
他把所有点排序出一组线性的vertices



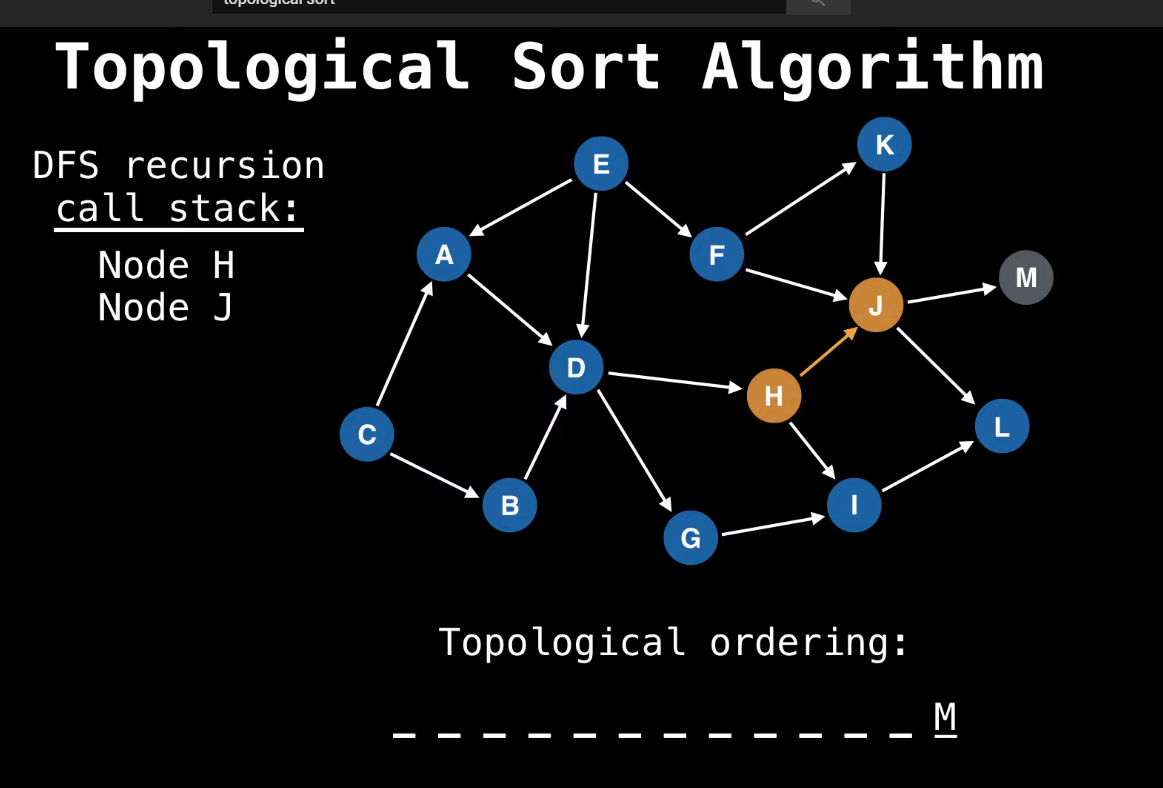
这样如果存在ui uj间存在edge，那么必然是i小于j

咋进行拓扑排序·

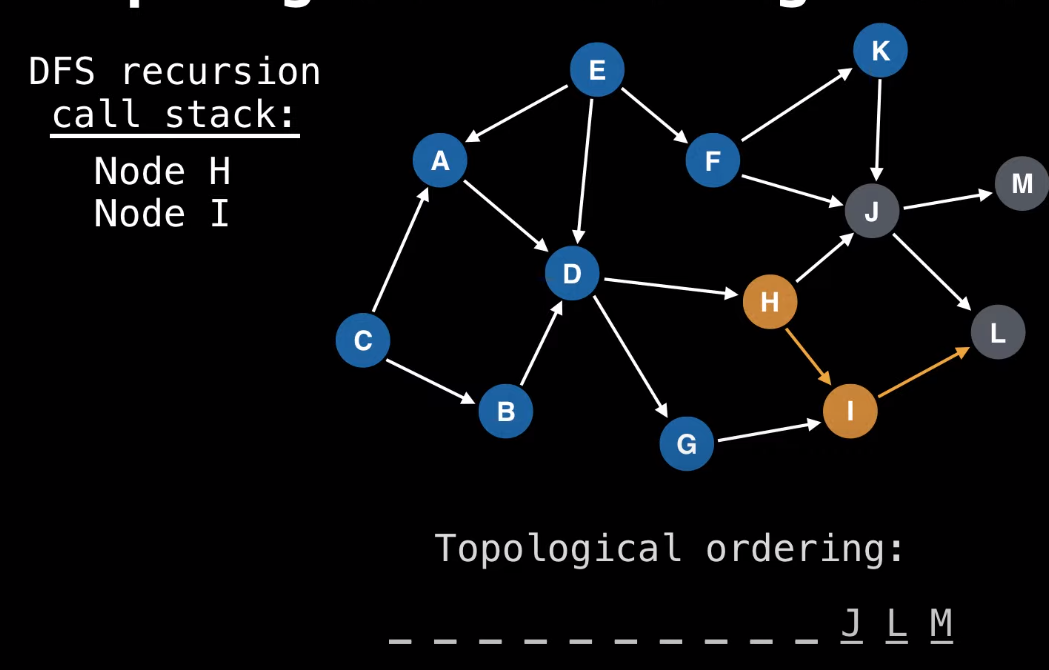
任选一个点例如H，进行DFS,HJM到底

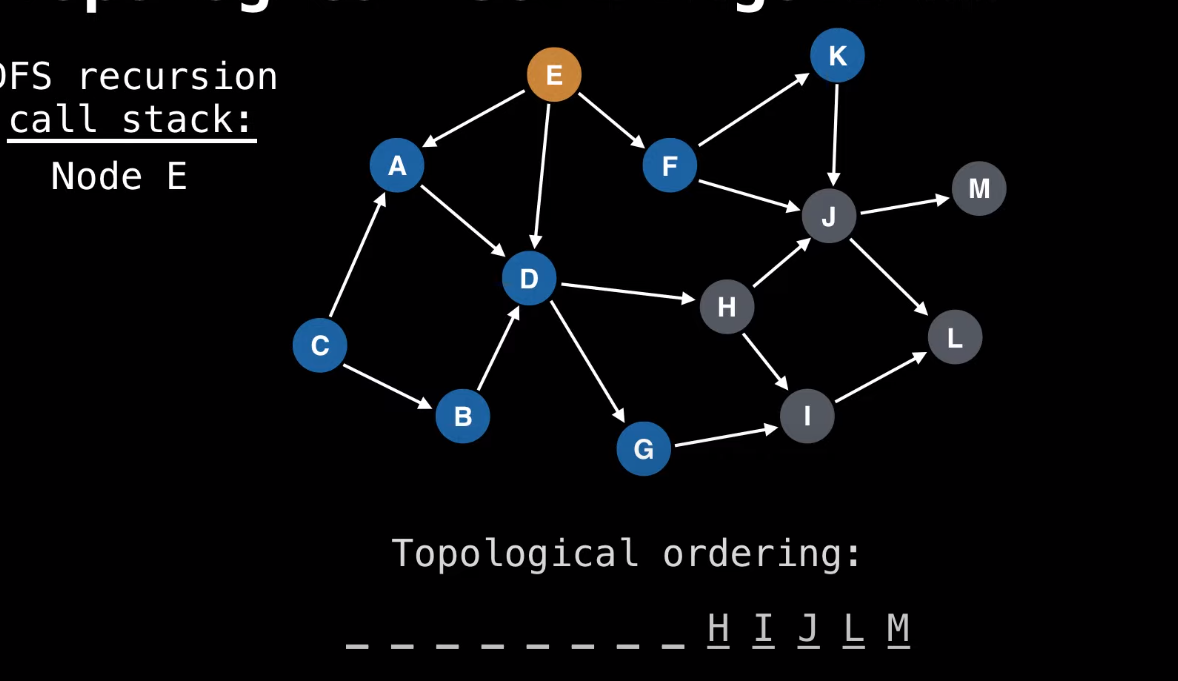


**M back track，设置成灰色，stack pop出来，放进topological ordering末尾**



然回J进L，L到底，排L排J，回H进I，排I，回H排H



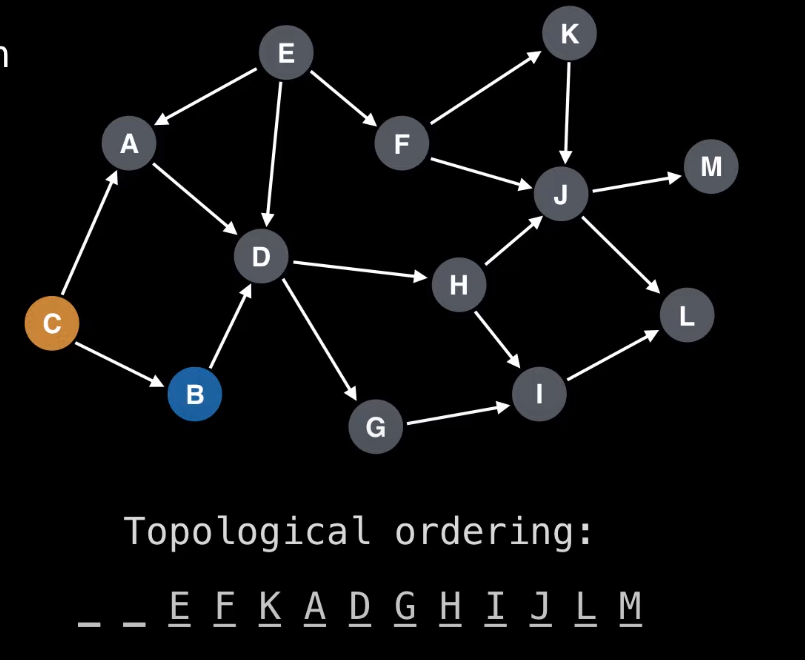


搜完一组再搜一组，E开始

EADG

排G排D排A回E

FK，排K排F 排E



最后CB排B排C

**任选一点，进行DFS**

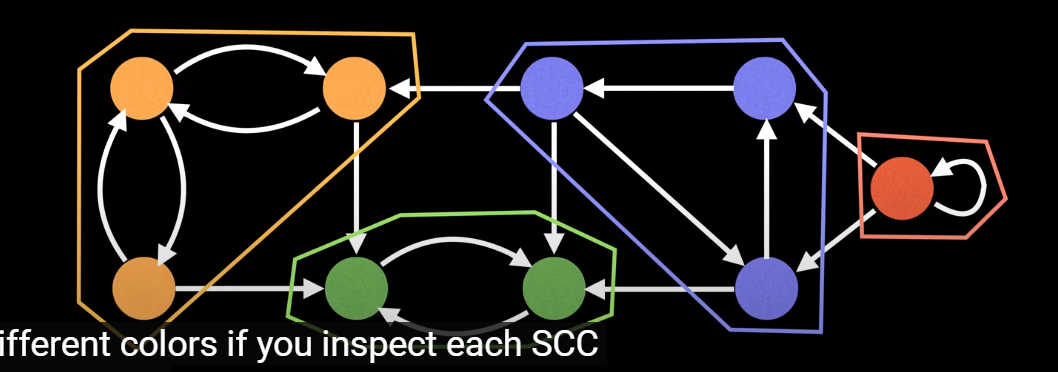
**当一个点到底的时候，pop，pop到topological ordering的末尾**

**这一组全部pop完毕，再次循环**

Strongly connected components

两个点有一条直接从A到B的path，也有一条直接从B到A的path，，叫做strongly connected

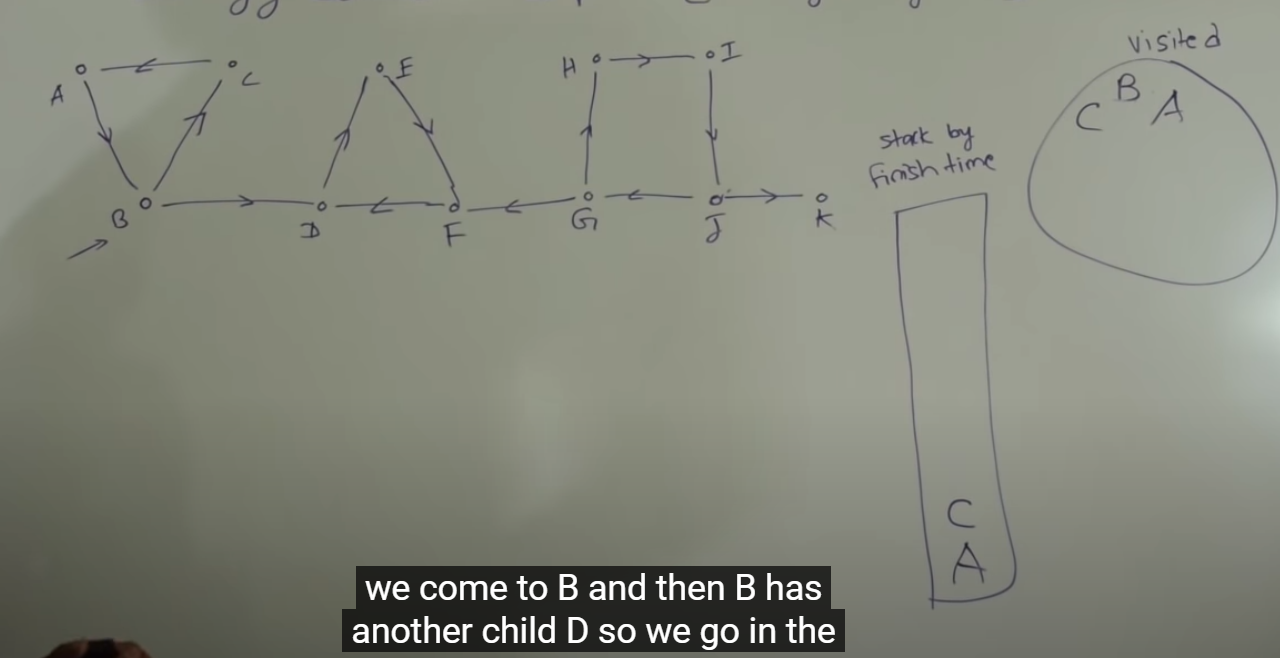
换句话说，任意两点有循环



Strongly connected component就是这样一组 vertices

这组vertices的size不能更大（不然·子component也有可能是strongly connected component）

有一点像拓扑，任选一个点开始DFS



B开头，BCA,这时B已经visited了

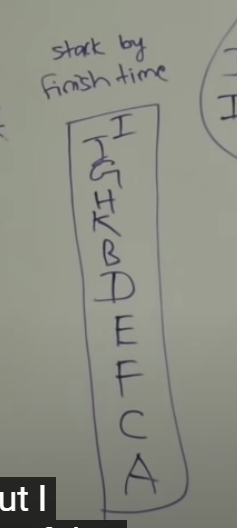
所以A是第一个vertex to finish,所以是第一个被push进去的

B继续DEF

F PUSH,E PUSH D PUSH B PUSH



选I继续,IJK , K PUSH, GH, H PUSH, G PUSH ,J PUSH ,I PUSH

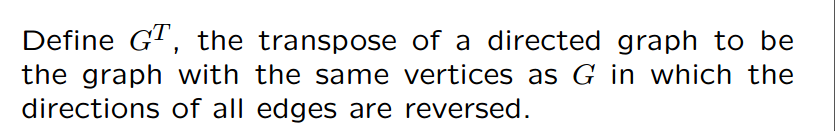


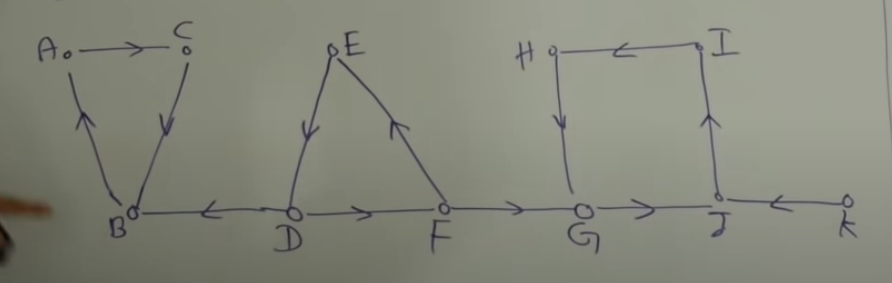
这是我们第一遍loop

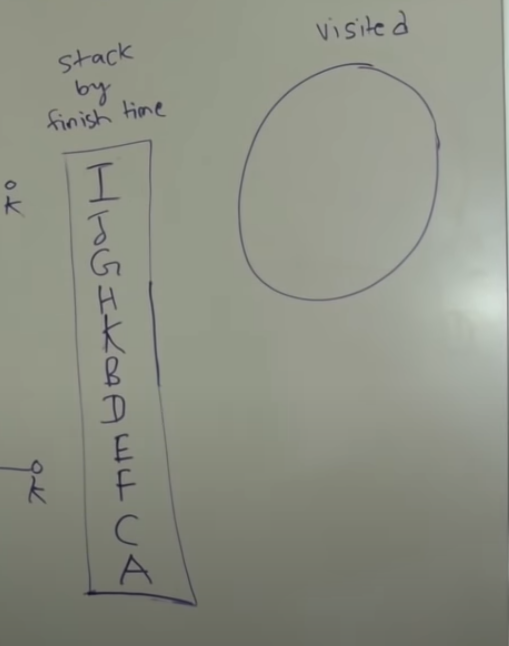
//随意选点，结束的pop进stack里

STEP2

构建GT,他是相同的vertices,方向相反







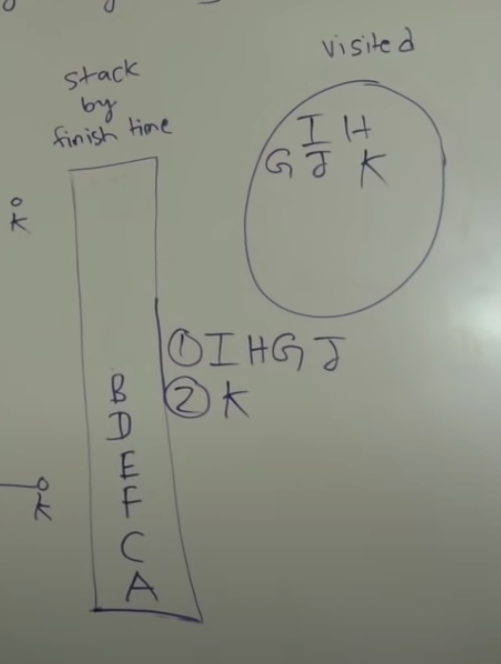
通过decreasing order从上到下处理STACK

还是使用DFS，但不是随机选取点，而是根据栈顶

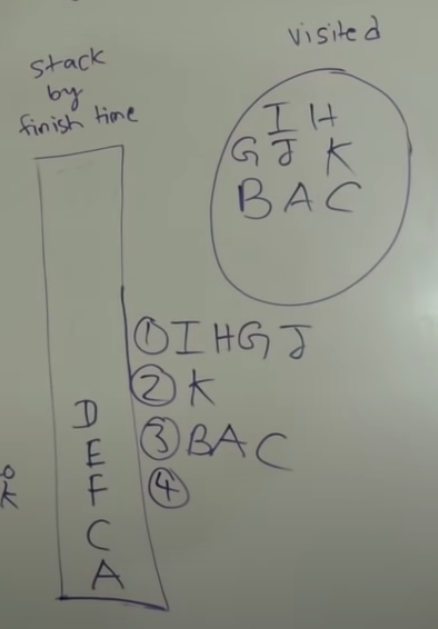
I， IHGJ , STACK POP出IJGH //只要是连续的就行，顺序随意

，加入到visited中

2. 栈顶现在是K，K指向J，**visited，舍去， pop**

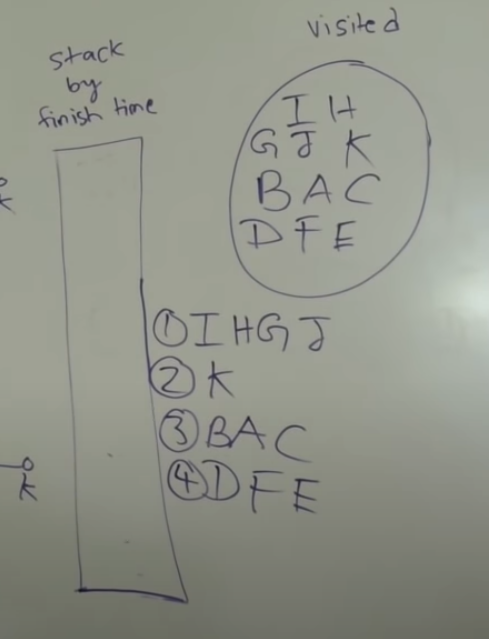


3.栈顶是B.BAC, 只能pop 出一个B， 但是BAC都要加入visited，加入visited的组合要记录下来



4.栈顶是D,DEF

最后剩CA，已经visited，直接舍去



IHGJ,BAC,DFE就是我们的三个SCC

时间复杂度等于（n+m）就是处理点与edge的时间

Space Complexity is O(v)

W8

定义：

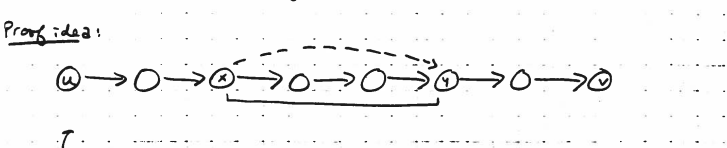
**Shortest-path weight**

**d[v]： 点v点到所求点s最短距离**

**Optimal substructure of shortest path:**

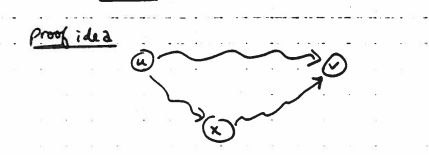
假设有一个最短路径，从v0到vk

那么对于他的子路径vi to vj，必然是vi到vj这两个点之间的最短路径（如果vi到vj之间有多条路径的话）



**Triangle inequality**





三角形定理，uv最短距离必然小于等于ux+xv

注意这里是有等于的，也就是ux,xv加起来同时就是最短的情况

如果是大于，那么uv就会被后面那条路径替代

**Well Defined**

允许negative weight edge,但是不允许negative的cycle，不然可以无限循环无限减小路径长度

**Relaxation**

首先需要给除了初始点以外的点v.d(s到v的最短距离)设置成正无穷

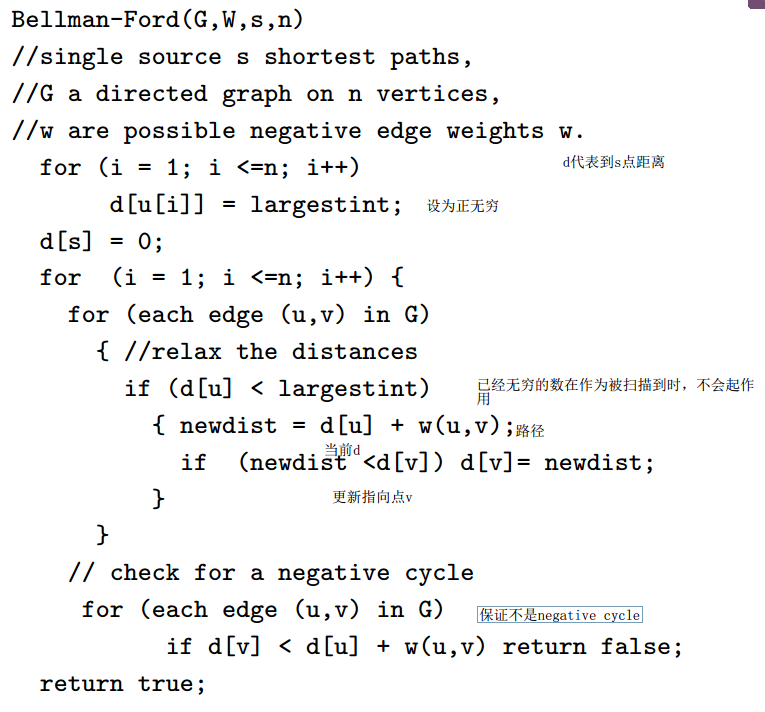
relax点v, 如果点u加一个edge(u,v)到达的点v，新path length(u.d+w(u,v))小于到达的点v当前记录的path length,更新

**Bellman-Ford algorithm**

算从一个点到其余点之间最短距离的算法

好处在于，允许edge是-weight,因为它能够检测negative cycle

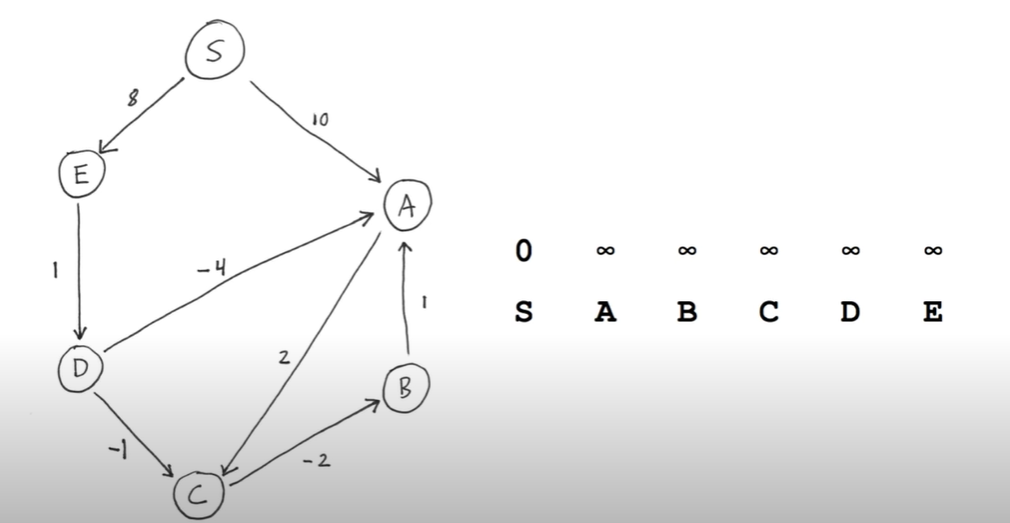
坏处在于用是比较久



别忘了最后检测这一步，要对所有的edge进行排查，如果dv<du+w(u,v),return false,

u是起点，v是终点

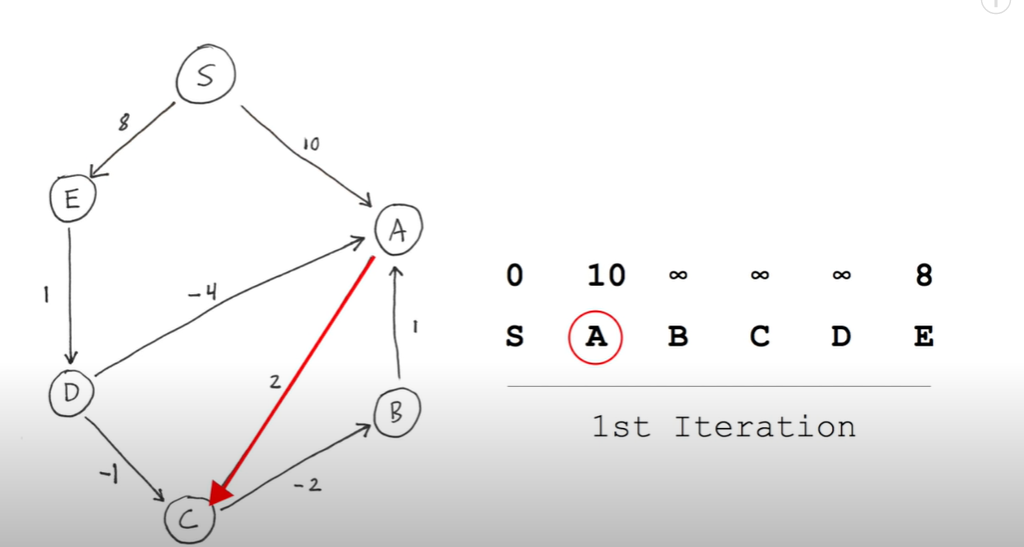
进行n-1次循环



一开始除了起始点1，每个点设为无穷

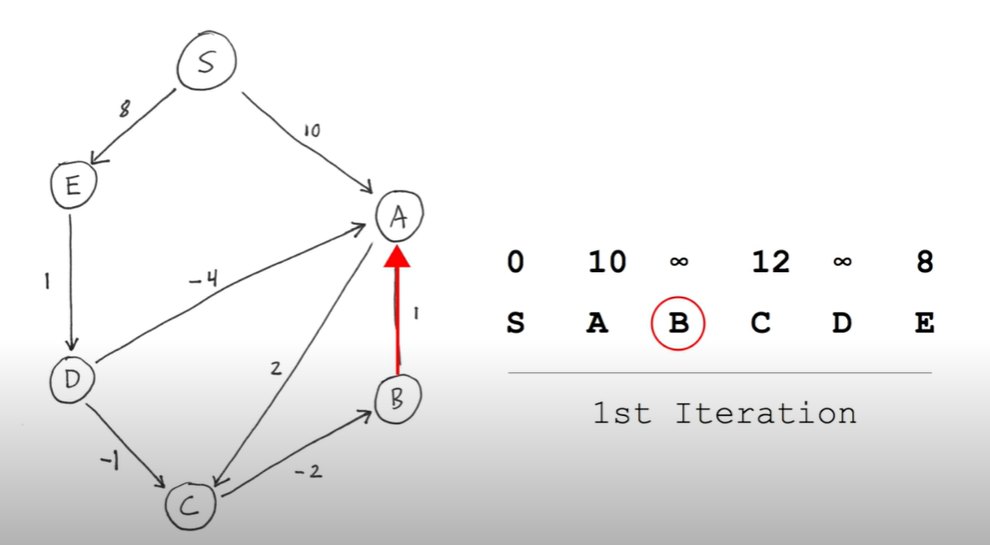
从左到右一个一个循环，更新这个点所指向的点的数据

S:指向EA,



EA被更新，A，指向C

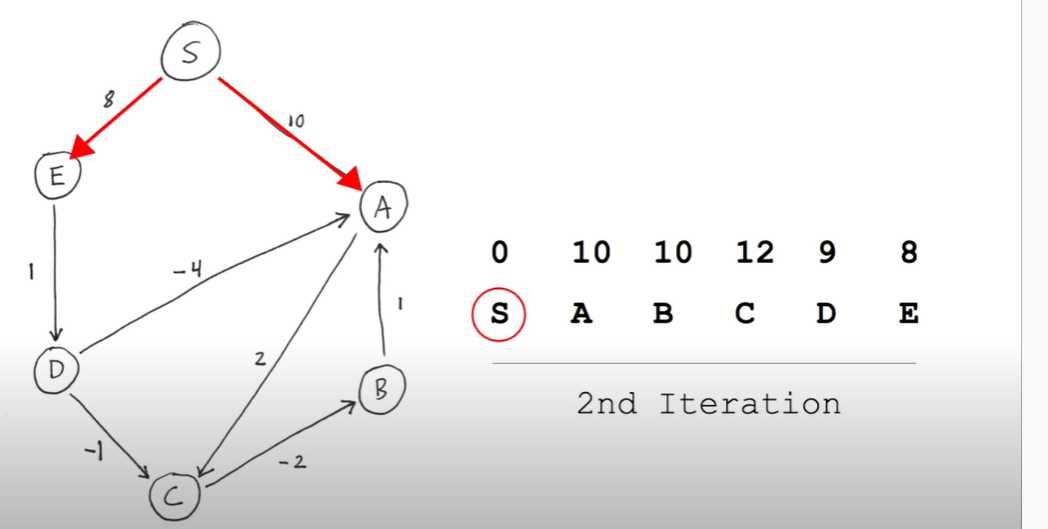
C被更新



B当前是无穷，不用管

C指向B,B更新成10

D不用管，E指向D，D改成9



重新进行第二次循环

S:EA不用改，

A:C不用改

B:A不用改

C:B不用改

D:C改成8，A改成5,

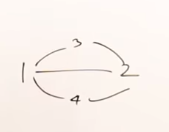
E:不用改

或者第n次pass仍然变小了，那也有negative cycle

**Floyd-Warshall:**

找到所有点之间最短路径

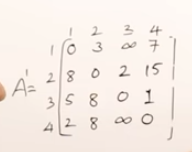
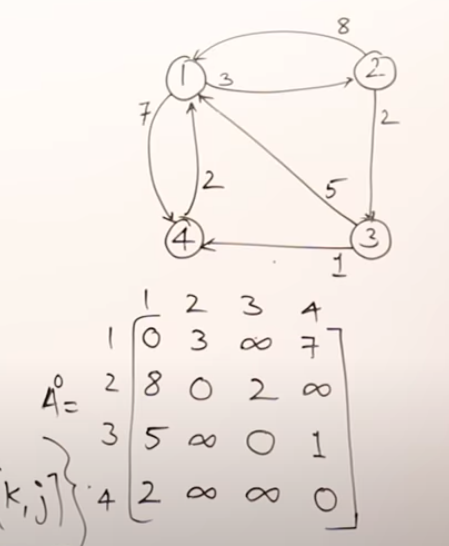
两个点之间最短距离无非两种情况

首先我们想知道1到2的最短距离，有可能

情况1. 直接1到2，压缩在1的遍历中

情况2. 3或4为中点，到了1到2，压缩在3或4的遍历中

第一步构建matrix



点到自己距离是0 //斜对角全是0

横轴是起始点，竖轴是终点，如果起始点到终点之间一开始没有直观的edge，取正无穷

然后开始遍历所有vertices

STEP2:

遍历vertices1, 斜着的·0，不变，1的竖轴横轴不变，对应情况，其他的点代表情况2，代表别人通过它进行的改变，例如A(2,4)

代表A(2,1)+A(1,4)=8+7=15

遍历完所有点就完事儿了

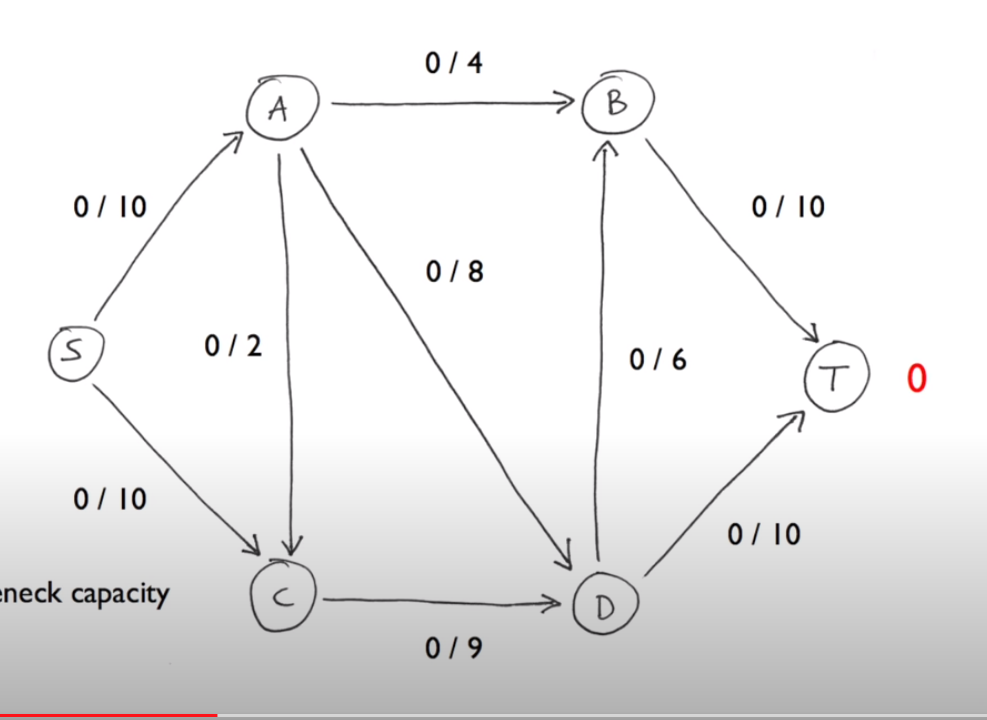
**Max Flow问题**

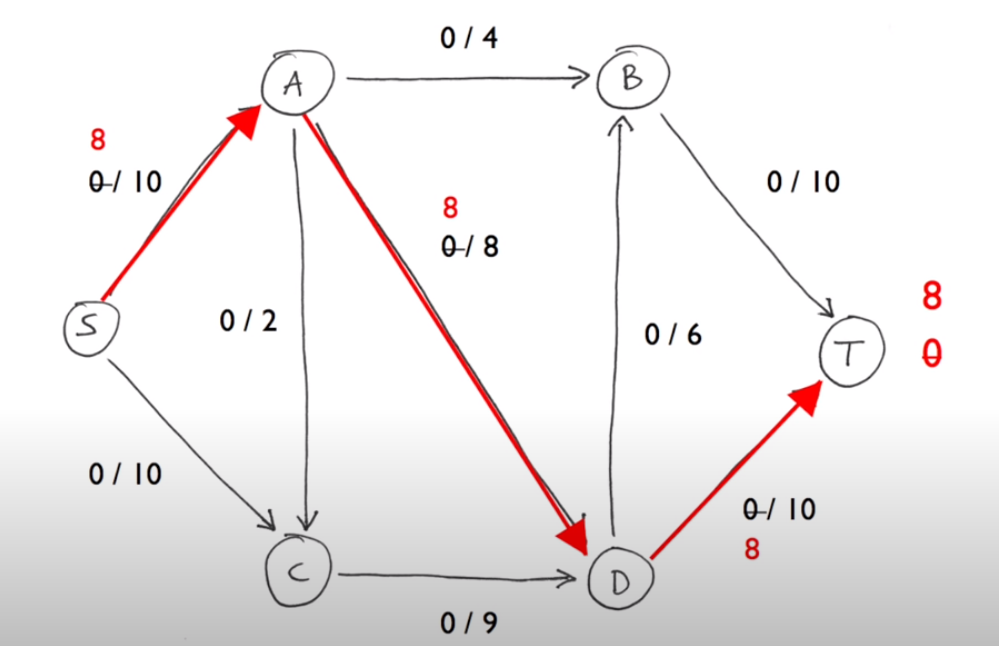
augmenting path: 从s到t，如果一条path仍然存在unused capacity，那么就是augmenting path

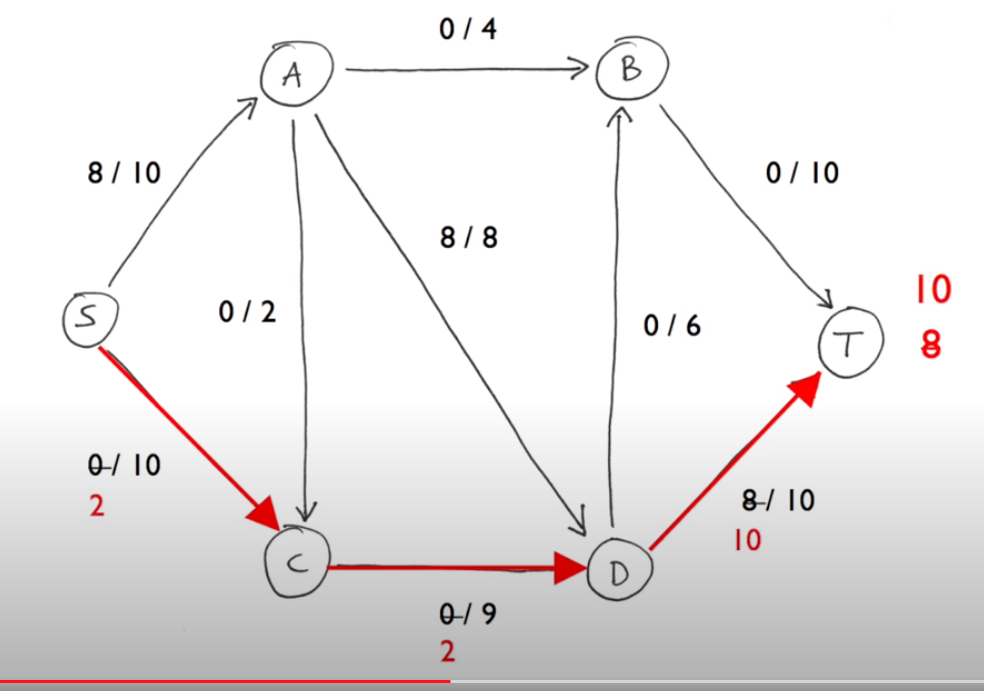
当没有augmenting path时，达到max flow

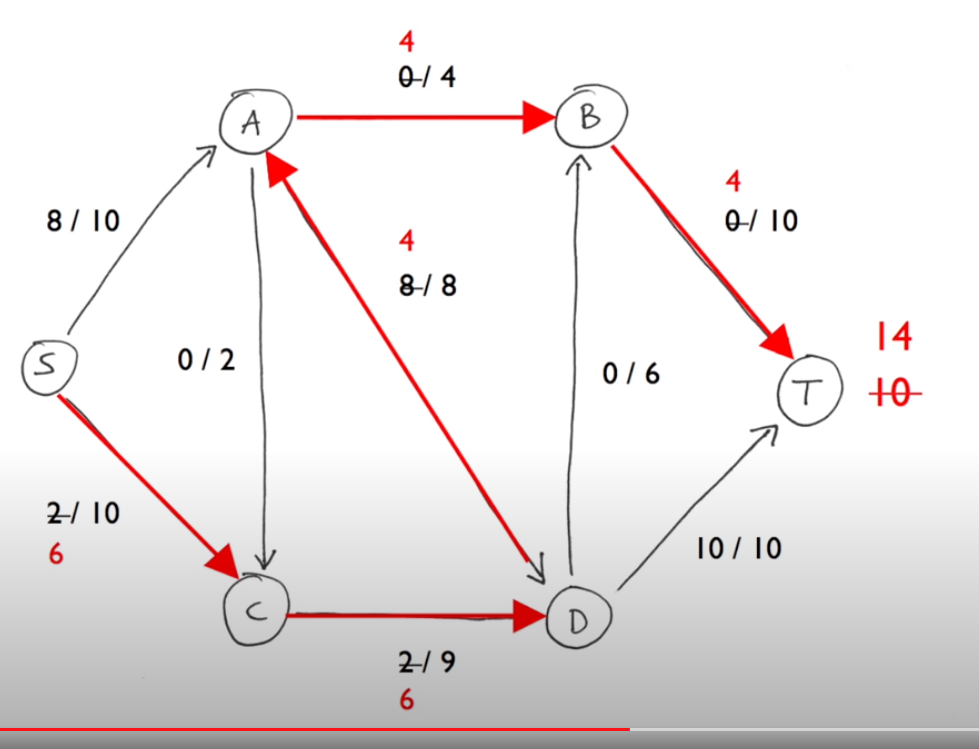
Ford-Fulkerson

就是不停的找augmenting path，找到以后减去这个path所能所能和承受的capacity,直到再也找不到augmenting path//具体顺序没啥要求

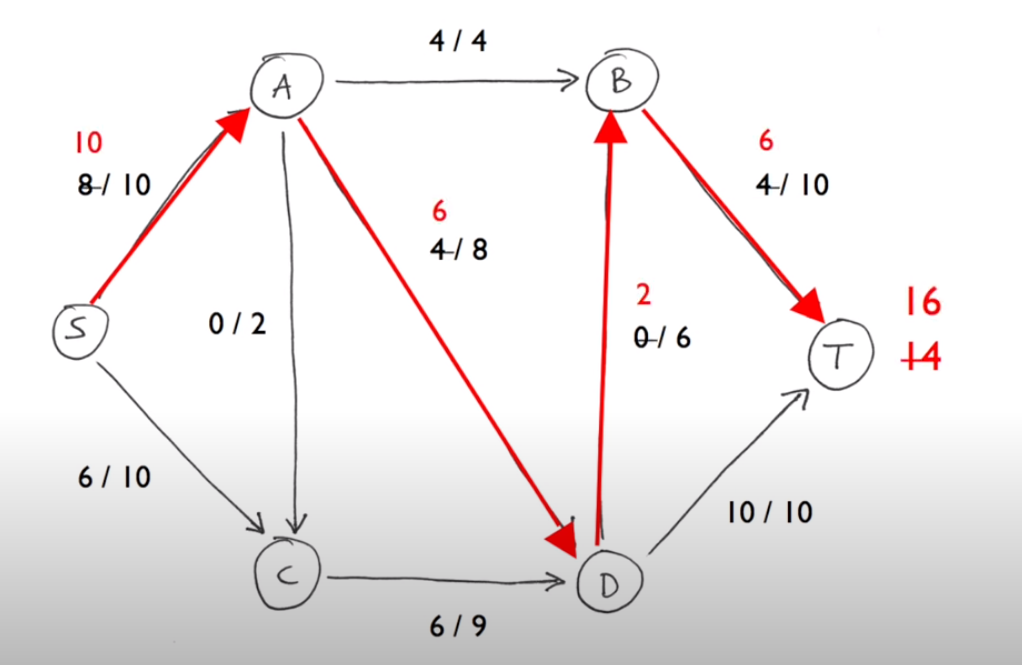


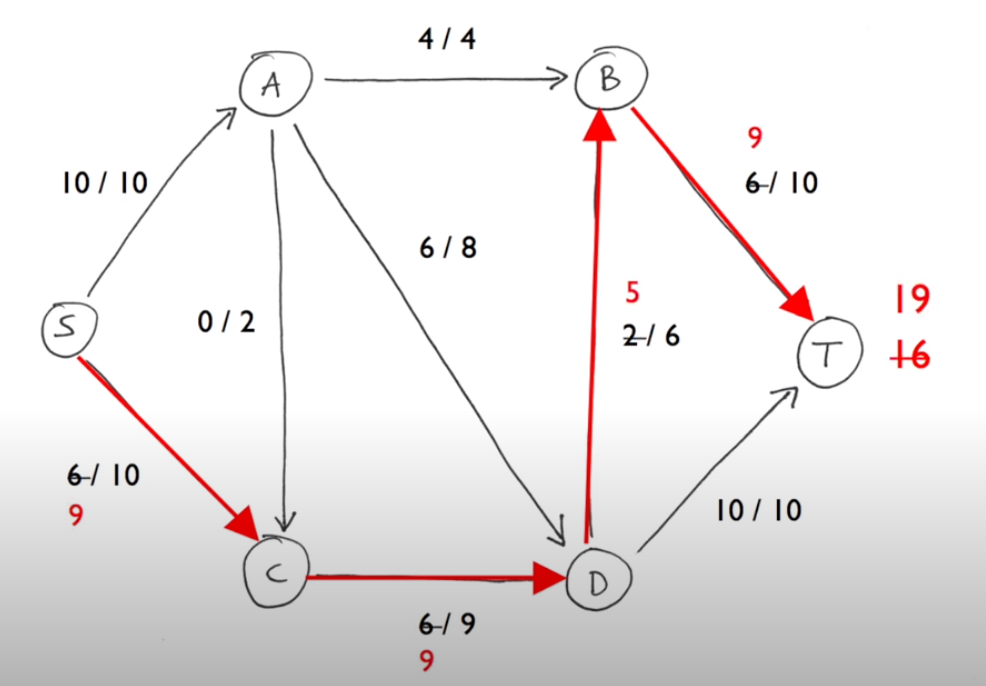






注意这一步，实际上D到A是没有edge的，但是在求max flow时，可以让我们的路逆着走，逆着走的时候减去当前flow就好了，它代表着原来A指向8to D, 0 to B,现在被拆分成了4 to D, 4 to B,也就是一个把指向点拆分实际输出的过程，到D的输出被拆分成了A到B，A到D





到极限了

Residual graph:

在我们分配好一条path以后，把箭头反过来代表undo 余额， 虽然是虚拟概念，但我们可以用他当做真实path看

**Ford-Fulkson,**

利用Residual graph的也可以算作ford-fulkson,主要和Edmond-Karp的区别在于BFS

Cut:用一条直线切割，S在一边，T在另一边，切割线经过的edge的capacity和就叫cut

**Min-Cut:cut**有很多种切法，最小的cut值必然等于实际Max Flow，虽然我们不见得能切出min-cut但我们能渐进估计max flow

**Edmond-Karp**

前面一种瞎找augmenting path的方法是有随机性的，他的表现取决于找augmenting path的顺序

使用Edmonds-karp algorithm，他的表现是polynomial time//稳定的多项式时间

使用BFS找到所用edge最少的path，

然后对这个path进行常规操作

循环

也就是说我们使用的augmenting path的edge是单调递增的

https://www.youtube.com/watch?v=w3Nl2XA0pxA

Maximum Bipartite matching

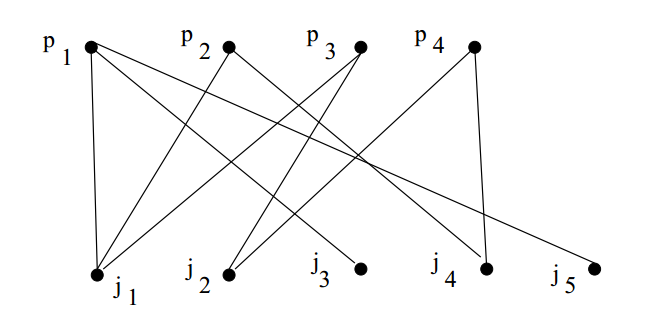
我们有n个人，m个job

每个人可以做指定的工作

现在每人最多做一个工作

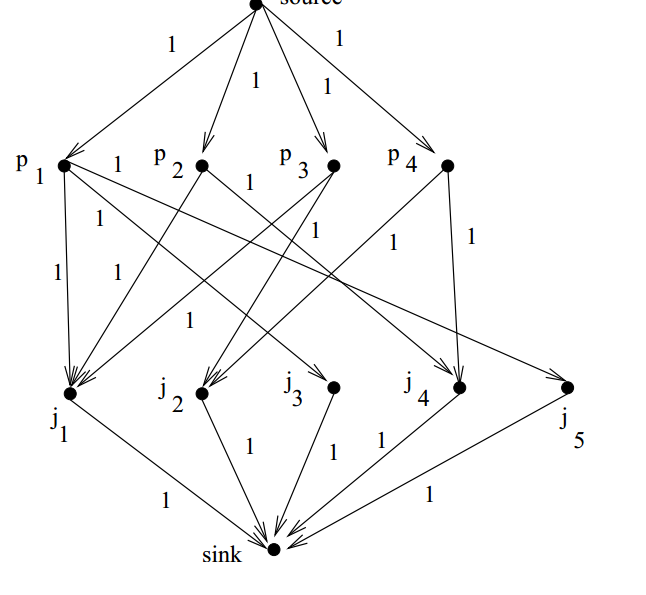
最多能完成多少工作

这个问题能被用bipartite graph表示 //二分图



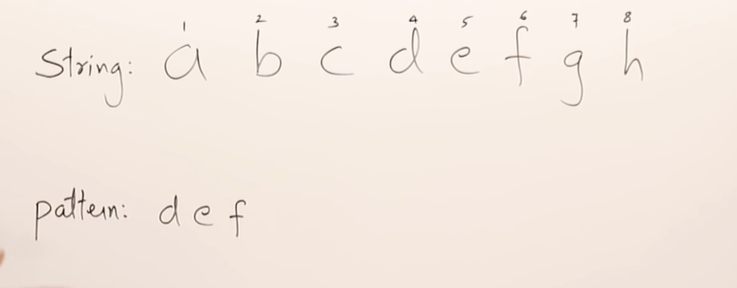
这类问题可以转化成一个maximal flow问题.

引入一个source,一个sink，每个capacity=1



有greedy： prove greedy choice property and optimal substructure property

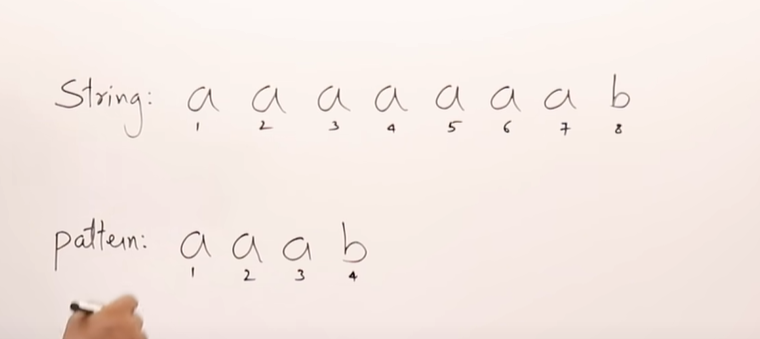
W9 String matching



就是这么一串String，查找有没有连续的pattern (要找的String)

Naïve String matching

一个一个滑



例如aaab,第一个a对上了，第二个a对上了，第三个a对上了，第四个b，没对上，证明第一个a肯定不是我们要match的，这时跳到第二个a，这个方法效率比较低因为第二个a我们已经match过了，但是naïve法还是要Match，

如果是这个例子，每个a都要match pattern的长度



Rabin-Karp Algorithm,

主要思路就是把pattern hash，

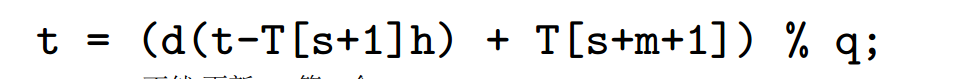
然后再在被checking string上几个为一组，进行hash数的比较，如果hash相等，在比较是否完全匹配

.

T是目标：

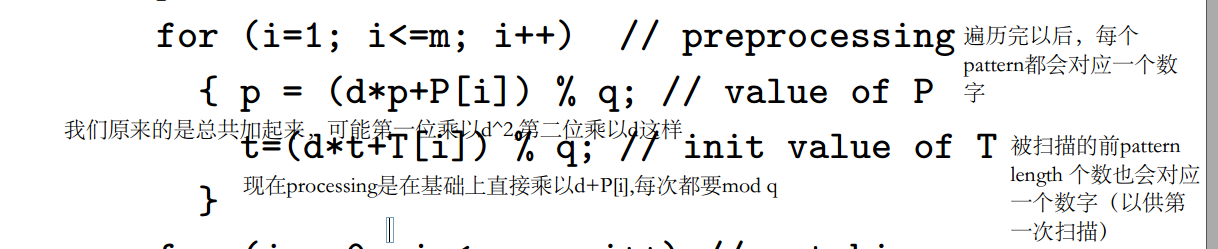
D是number of char in alphabet

q是一个prime number



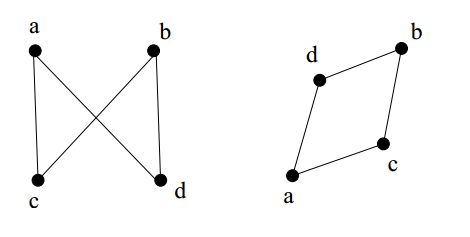
减去第一个以后乘以d，加上新的一项，mod q.

Preprocessing



**Computational Geometry**

在graph里，点的位置只是一个相对关系，关键的是他的临近关系，但是在Geometrical问题中，点的位置也很重要



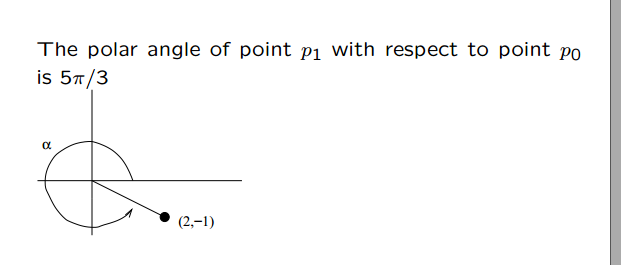
作为graph，她们是同态的isomorphic，作为geometrical object，是不同的

讲了一段polar的



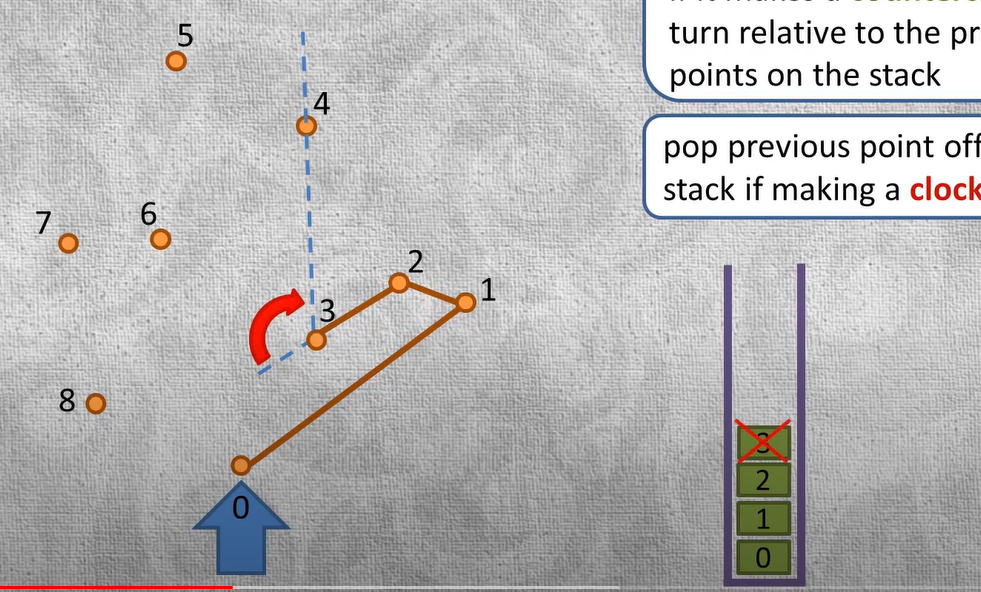
这三个点都是标准point点，

那么换成polar表示法，角就是5pi/3

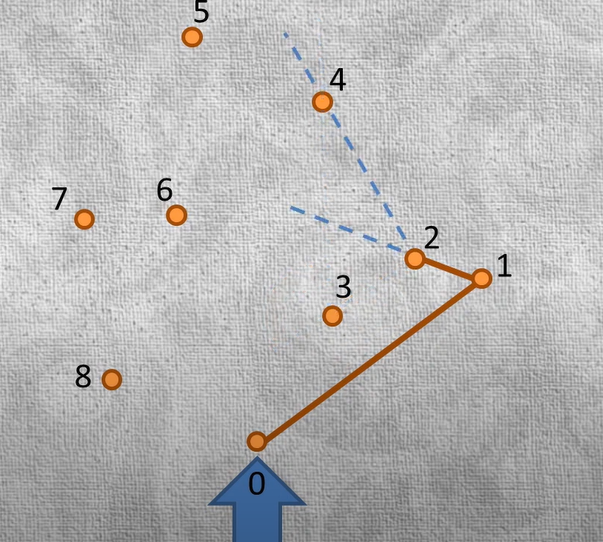


Convex Hull：包含所有点的凸多边形

Graham Scan:用最低的点作为射线，进行扫描，如果遇到的点是逆时针的//形成一个凸角，保留，添加到candidate stack里，如果是顺时针的//形成一个凹角，pop candidate,直到这个凹角变成凸角



扫1，扫2凸，存，扫3凸存，扫到4凹，pop 3



42还是凹，pop 2,留下4，

5凸，留下，6凸留下

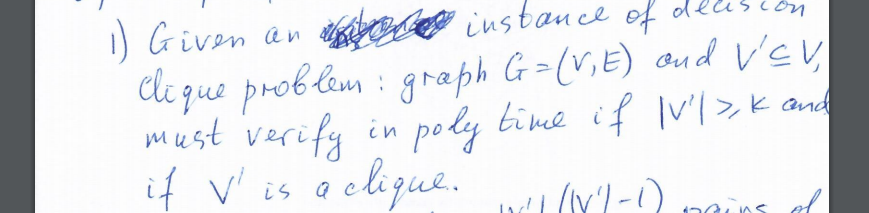
7凹pop6,

8凸

最后回到0凸，结束

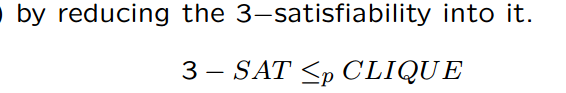
标准NP complete过程

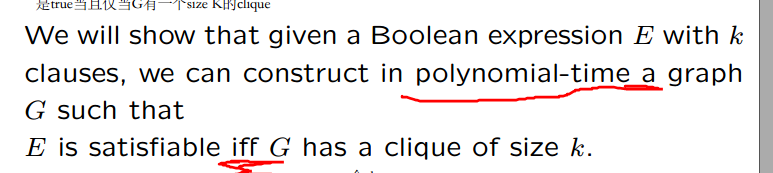
1.



给你一个instance,在polynomial time内能check

2.





we need to show that given a ….. we can construct …. in polynomial-time a graph g such that 前者满足iff后者满足

反着来，如果我们有clique，那么我们能证明E里每个至少一个是对的，E也satisfiable

W10直接看笔记