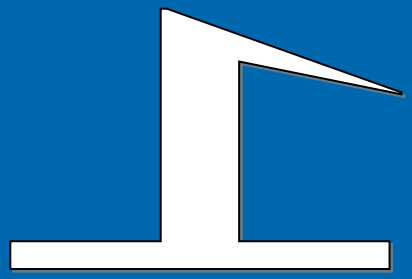
SPFA 算法的优化与应用

广东中山纪念中学 姜碧野



常见问题:

1、在负权图上判断是否存在负环

2、解决有环的动态规划转移方程

这些问题该如何解决?

SPFA

内容概要

在本文中将看到：

第一部分　　简要介绍 SPFA 的基本实现

第二部分　　提出 SPFA 一种新的实现方式

第三部分　　介绍如何灵活使用 SPFA 解题

SPFA 全称 **Shortest Path Faster Algorithm**

基本应用为快速求解单源最短路

相比其他同类算法有什么优点呢?

简洁优美

灵活多变

适用面广

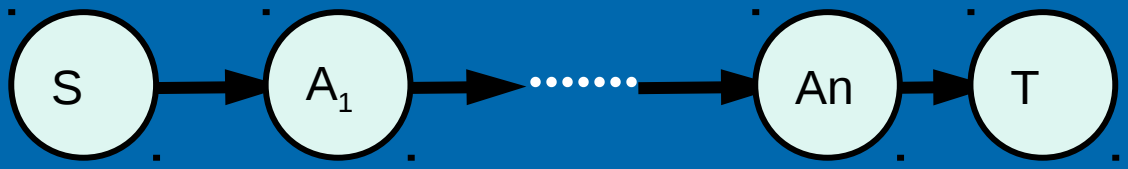
让我们来一起领略!

SPFA 的核心正是松弛操作：

```
Relax(u,v){
    If F(v)>F(u)+W_Cost(u,v) then
        F(v)=F(u)+W_Cost (u,v);
}
```

但松弛操作直接得出的 Bellman-Ford 算法效率低下

For Time=1 to N
For (u,v)∈E
Relax(u,v)

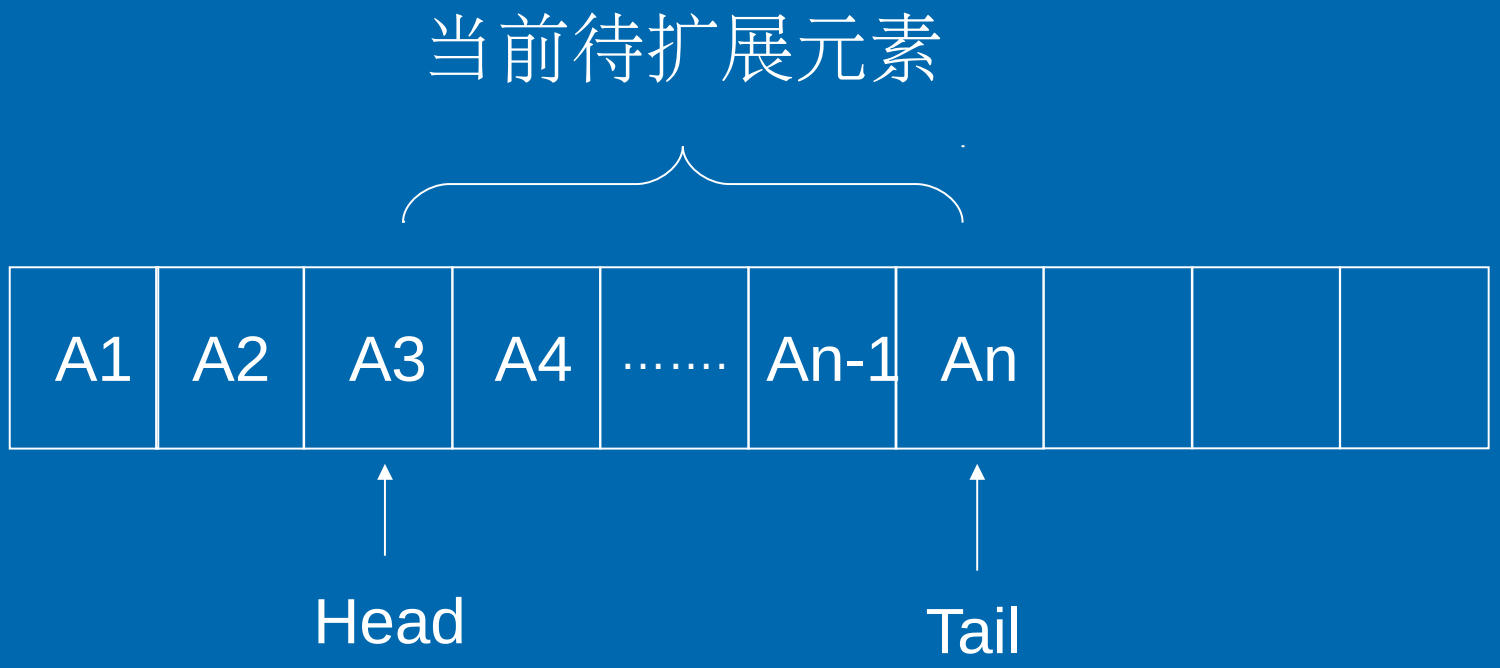


上图数据中，总运算量高达 N^2

而边 (S, A1) 虽然被调用 N 次。

但实际有用的只有一次

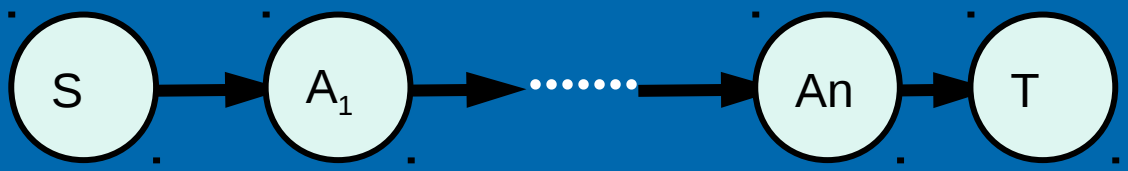
SPFA 则使用队列进行了优化！



思想：

只保存被更新但未扩展的节点。

减少了大量无用的计算，效率大大提高！

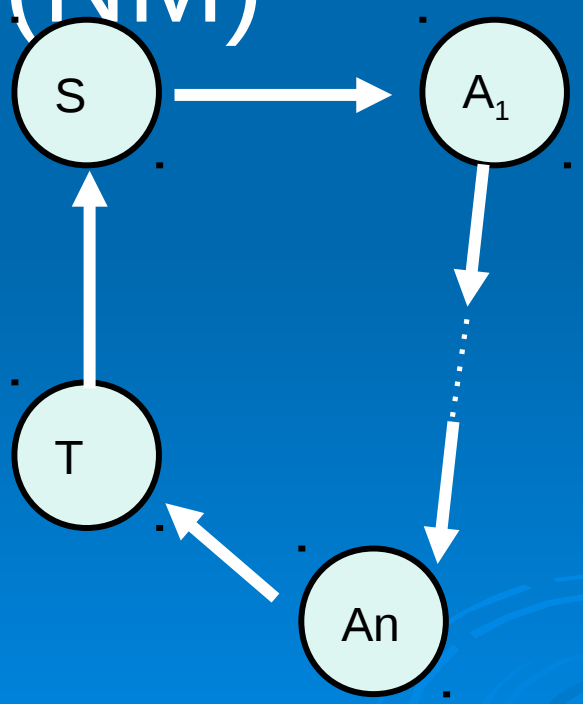


在上图的例子中，每个节点只进队一次，只需 N 次运算。

相比 Bellman-Ford 优势明显。

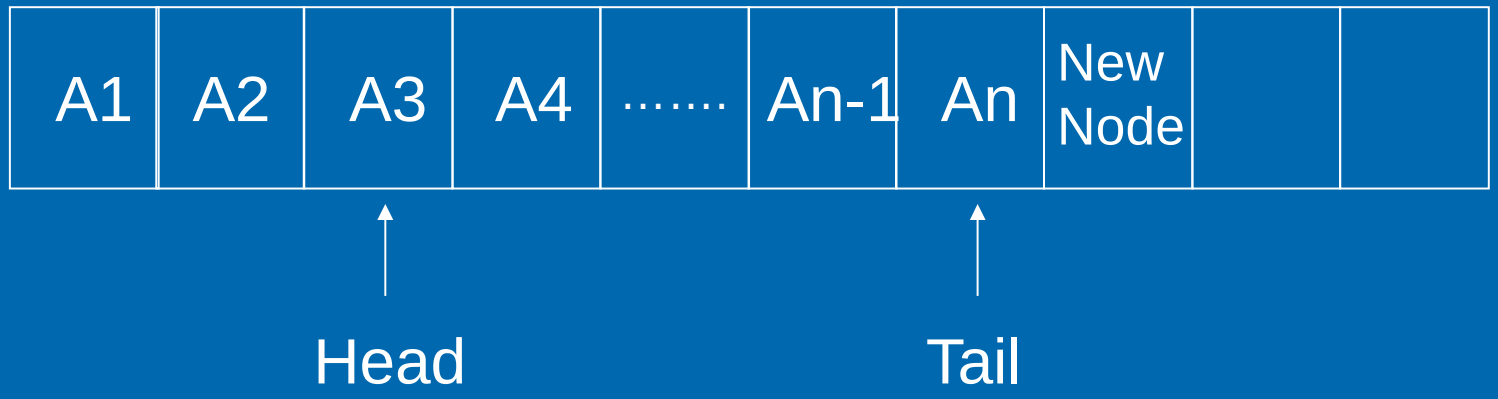
在 1000000 个点， 2000000 条边的随机数据中 SPFA 甚至比使用堆优化的 Dijkstra 还要快。

但有负环时依然退化为 O(NM)



长期以来基于队列的 SPFA 并未取得突破

传统的队列实现：



缺点： NewNode 需要之前的元素全部出队后才能扩展

中断了迭代的连续性

猜想：

能否把 **NewNode** 放在 **Head** 后面进行下一次扩展？

猜想 ⟶ 程序

图论中的基本算法 ： 深度优先搜索

基本数据结构：栈

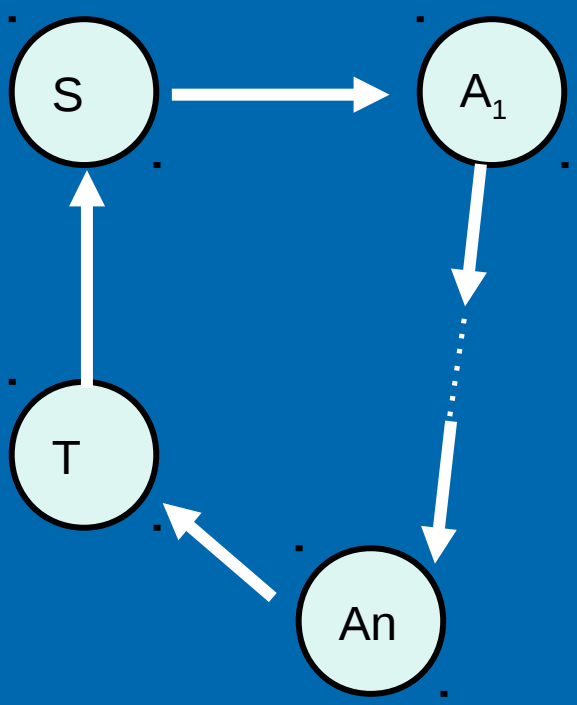
后进先出！

判断存在负环的条件：重新经过某个在当前搜索栈中的节点

```
SPFA_Dfs(Node) {
    For (Node,v) ∈E
        If dis[v] >dis[Node]+w(Node,v) then
          dis[v]=dis[Node]+w(Node,v);
          SPFA_Dfs (v);
        }
}
```

核心思想：

每次从刚刚被更新的节点开始递归进行下一次迭代！



相比队列，深度优先有着先天优势：

在环上走一圈，回到已遍历过的点即有负环。

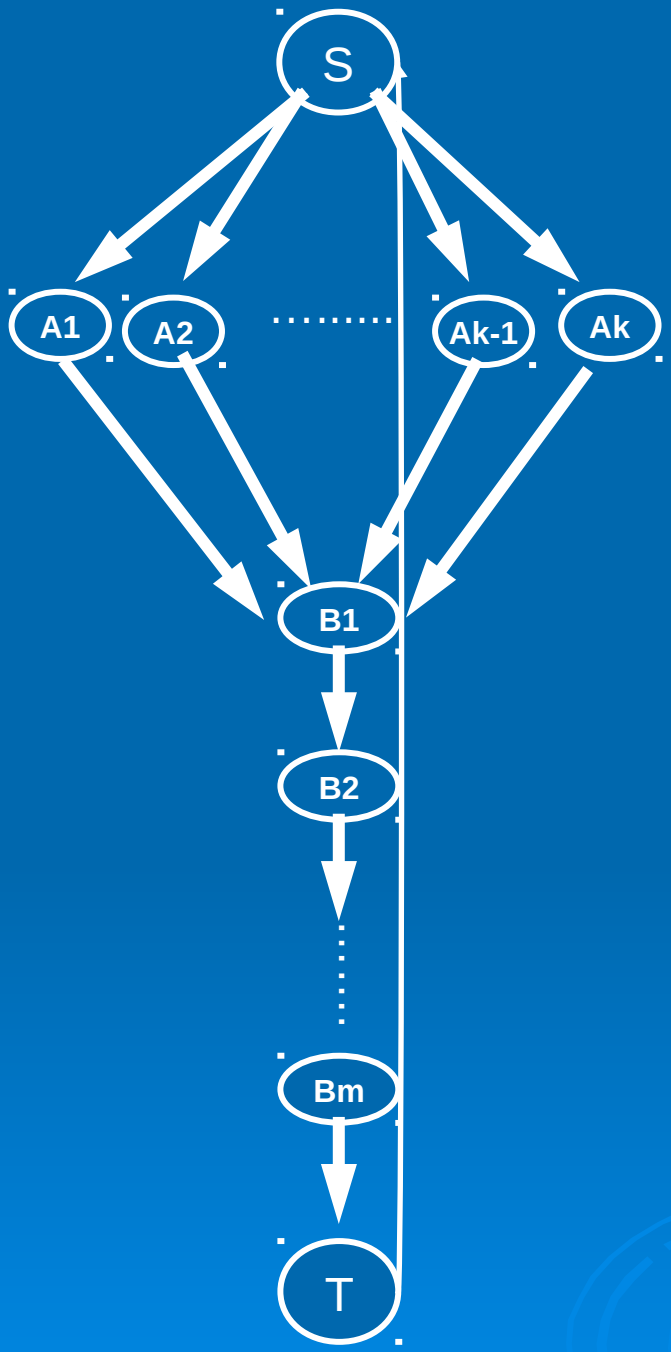
绝大多数情况下时间为 O(M) 级别。

实战 :WordRings
(ACM-ICPC Centrual European2005)

676 个点， 100000 条边，查找负环。
DFS 只需 219ms ！

一个简洁的数据结构和算法
在一定程度上解决了大问题。

T SPFA



还有不足吗?

最坏情况下需要 KM 次运算

+∞ 优化：随机调整边的顺序

则期望 k+mLogk

最短路问题其实只是 **SPFA** 迭代思想在图论

中的一个特例，在其他各类动态规划，迭代法解

方程，不等式等问题中往往也能发挥奇效！

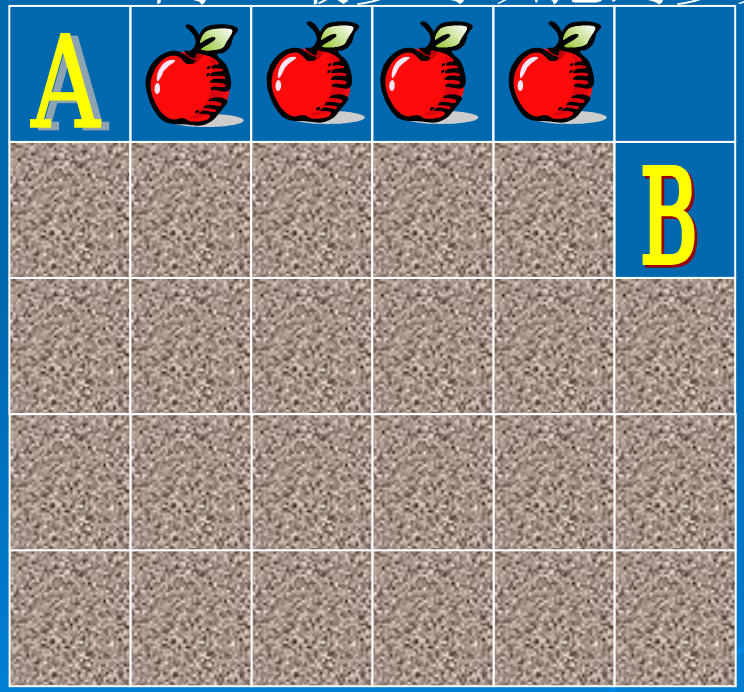
让我们结合一道题目来进行探讨

苹果争夺战

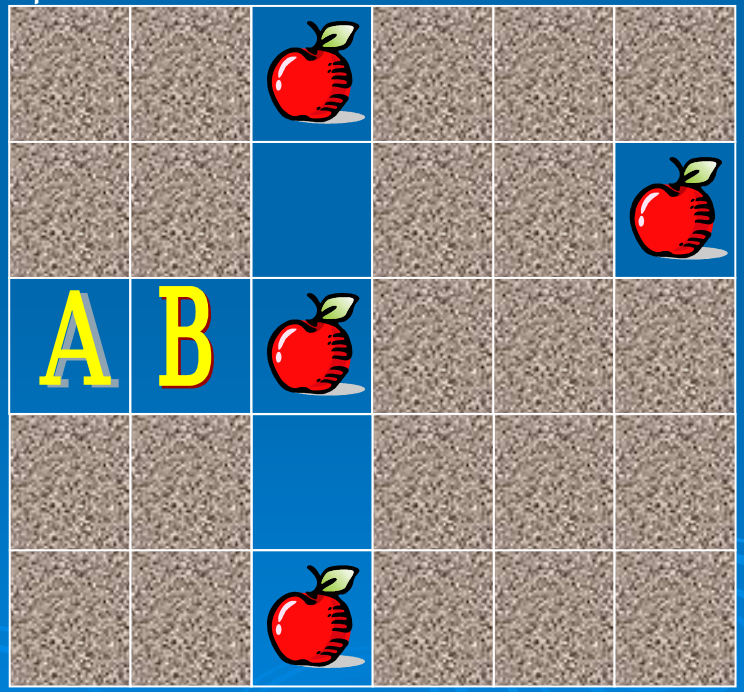
两个人 A,B 在一个 5*6 的矩阵里抢夺苹果。矩阵包含空地，4 棵苹果树和障碍物，每个苹果树上有 3 个苹果。A 先行动，然后两人轮流操作，每一回合每人可以向四周移动一格或停留在一棵苹果树下，如果苹果树非空可以摘下一个苹果。

两人不能移动到矩阵外，障碍物上或是对方的位置，且两人绝顶聪明。

问 A 最多可以抢到多少个苹果。



此时 B 不能再向左移动
而 A 可以逐步摘下 3 棵树的所有苹果



开始时 A 无法移动

之后 B 一直不动，
A 无法得到任何苹果

问题分析：

经典的博弈模型，数据规模比较小，考虑动态规划

F[X,Y,K] 表示轮到 A 行动， A 的位置为 X ， B 的位置为苹果树状态为 K(使用状态压缩的 4 位 4 进制表示）时 A 最多获得多少苹果。

G[X,Y,K] 类似表示轮到 B 行动时， A 最少获得的苹果数。

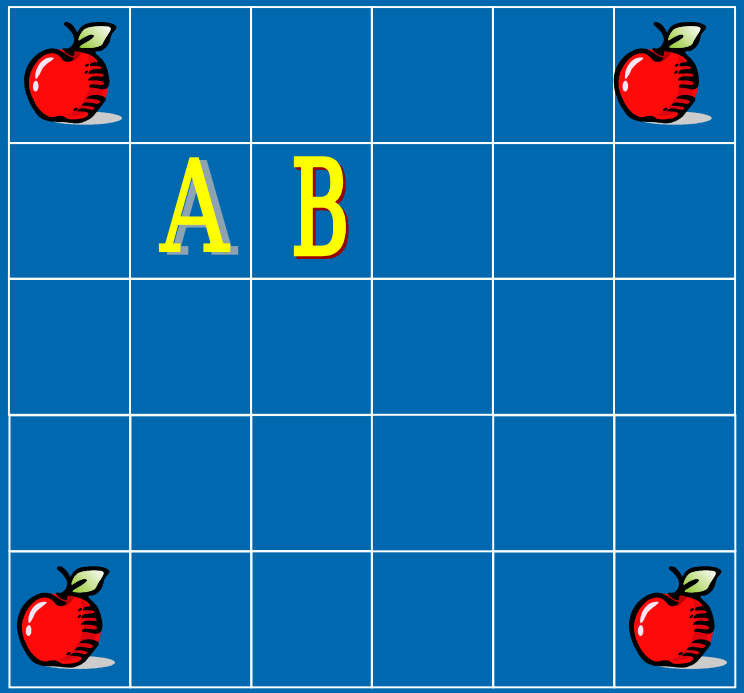
状态数为 30*30*256*2 ≈500000 可以承受

转移方程也简单，直接枚举 5 种行动

F[X,Y,K]=Max(G[X’,Y’,K’]+Apple)

G[X,Y,K]=Min(F[X’,Y’,K’])

但是....



单纯的状态转移会出现环，怎么办呢?

解决存在环的动态规划，常规思路：

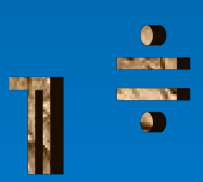
一：利用标号法

通过已经得出最优解的状态递推出其他状态。

如何找出最优解?

值最大的？但 G[ ] 可能被其他较小的 F[ ] 更新，并不是最优解。

值最小的？ F[ ] 同样可能被其他更大的 G[ ] 更新，

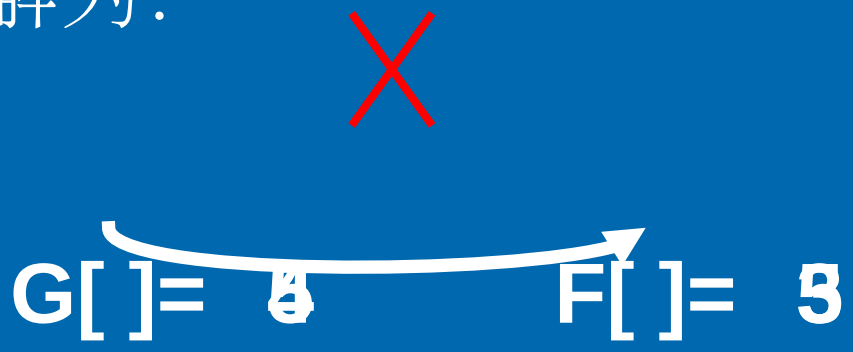


因此标号法并不适用

思路二：参考负权图上求最短路的思想

通过局部的较优值一步步迭代得到最优解

假设当前解为：



之后 G[ ] 得出最优解 4

两种常规解法都失败了，我们需要从新的角度来思考

猜想：

能否越过状态间纷繁复杂的转移关系

直接考虑最终状态呢?

回归原方程：

F[X,Y,K]=Max(G[X’,Y’,K’]+Apple)

G[X,Y,K]=Min(F[X’,Y’,K’])

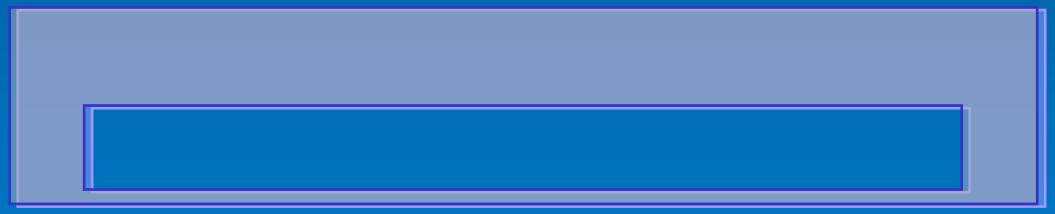
既是转移方程，也是终状态

联想： SPFA 在图论求最短路中的本质：

三角不等式：最短路的终状态，对于所有边 $(u,v)\in E$

有 $distance(s,v)<=distance(s,u)+w(u,v)$

当某边三角不等式不成立时，用松弛操作调整之。



在本题中适用吗?

同样：
- **F[X,Y,K]=Max(G[X’,Y’,K’]+Apple)**
- **G[X,Y,K]=Min(F[X’,Y’,K’])**

是问题的终状

一旦方程整体不成立便重赋值！

将松弛操作推广！

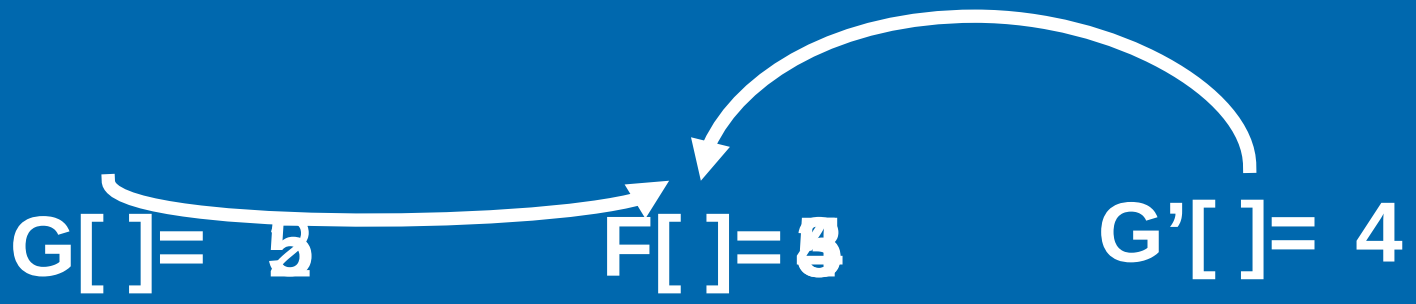
算法：

先对边界状态赋初值为 0 。

使用 SPFA 不断考察每条转移方程是否成立，

不成立则更新。

假设当前解为：



F[ ]=MAX(G[ ],G'[ ])

之后 G[ ] 得出最优解 2

特点：

赋值时考虑的是一个整体，即需要在所有与当前节点关联的状态中取最值，保证了合法性。

G[ ] 被更新时 **F[ ]** 还要重新考虑 **G'[ ]** 。

算法正确吗？让我们继续分析。

性质 1： 该算法结束时求得的解为正确解。

证明：该结论显然，算法结束意味各个方程均成立

性质 2： 该算法一定会结束。

证明：

把状态按照其最终值的大小分层，则可以发现当前 K 层确定时，对于第 K+1 层有：

1.G[ ] 可以从前 K+1 层取得最小值。

2.F[ ] 的最大值只能从前 K+1 层取，否则其最终值不可能为 K+1 。

因此状态会逐层确定并最终停止。

回顾思考过程，我们似乎感到：

最终算法完完全全建立在原方程之上，没有转弯，没有变形，只需“简单机械”地赋值。

而与之类似的传统迭代法却并不可行。

不！　让我们对比几种算法。

更新时需遍历所有相关节点（本题算法）

优化：

利用最短路问题中当前解只会成为次优解，而不会成为非法解的性质。

更新时只需考虑点对间关系（最短路迭代算法）

利用标号法则使用贪心思想再优化

每个节点只扩展一次（标号法）

三者的本质都是统一的，但随着算法的优化适用面逐步缩窄

优化算法是好的，但如果没有对算法有着深刻的认识，忽略了算法的适用条件，思维的定势很容易使我们得出错误算法。

SPFA

总结

在对 Bellman-Ford 算法的合理优化中，诞生了高效的 SPFA 算法。

在查找负环中，抛开了传统的实现方式，我们得出一种崭新的架构使效率大大提高

在动态规划中，摆脱了思维定势的影响，我们才得出正确的解法。

SPFA 并不是一个死板的经典算法，我们只有灵活运用才能发挥其应有的奇效。