图论及其算法

北京大学 杨昊翔

图论：提纲

- （一）图论基础知识
- （二）图论算法
- （三）应用和例题
- 字体应组织方统一要求设定为28号字以上的黑体、细黑，目的是方便后排同学看见。
- 因此有的原先一页的ppt会在很尴尬的地方被拆分成了两页，请大家原谅。
- 觉得太简单的同学可以先睡一会儿，但请不要D傻逼的讲课人。

（一）图论基础知识

- ①图论介绍
- ②图的基本定义
- ③图的特殊情况
- ④图的存储和遍历

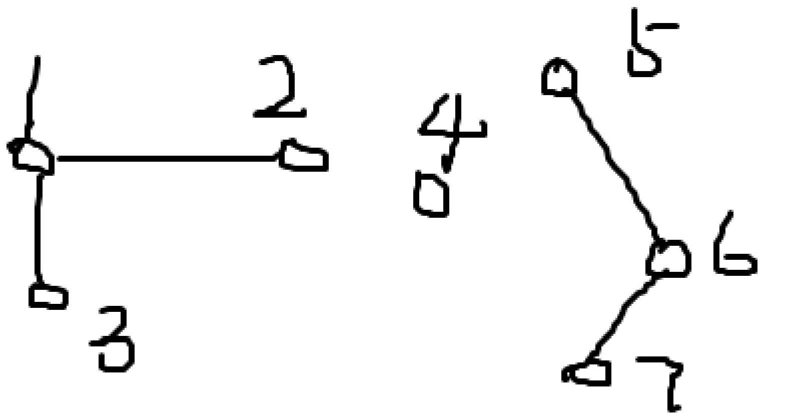
（一）图论基础知识
①图论介绍

- 图论-百度百科：图论是数学的一个分支。
- 图是由若干给定的点及连接两点的线所构成的图形，这种图形通常用来描述某些事物之间的某种特定关系，用点代表事物，用连接两点的线表示相应两个事物间具有这种关系。
- 在信息竞赛中，我们常常把实际问题中存在的类似于点和边的关系看成图，使用图论算法解决问题。

（一）图论基础知识

②图的基本定义

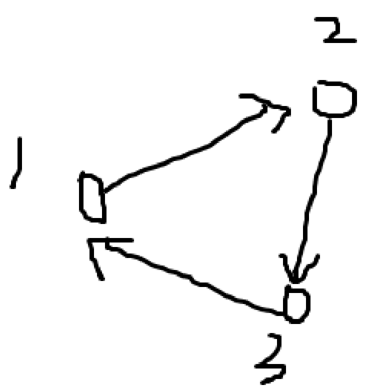
- 首先，让我们来看看图中有哪些结构：
- 点
- 边
- 点权
- 边权
- 点度：和一个点相连的边数
- 联通块

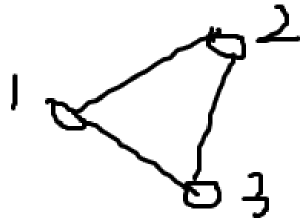


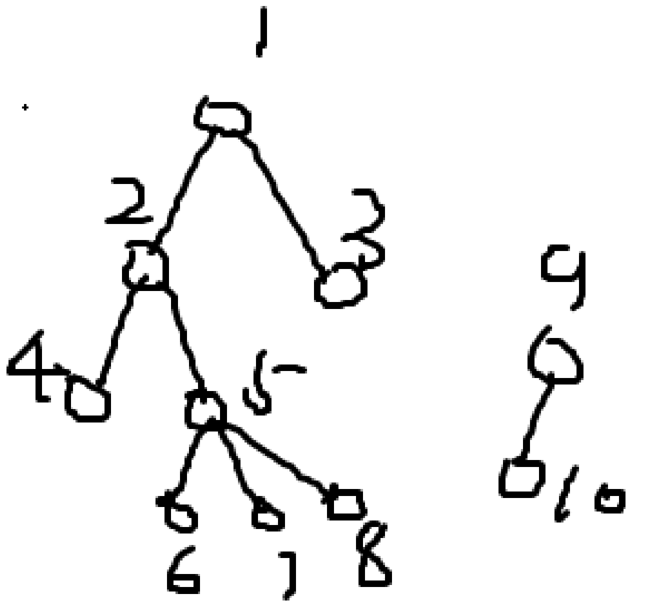
（一）图论基础知识

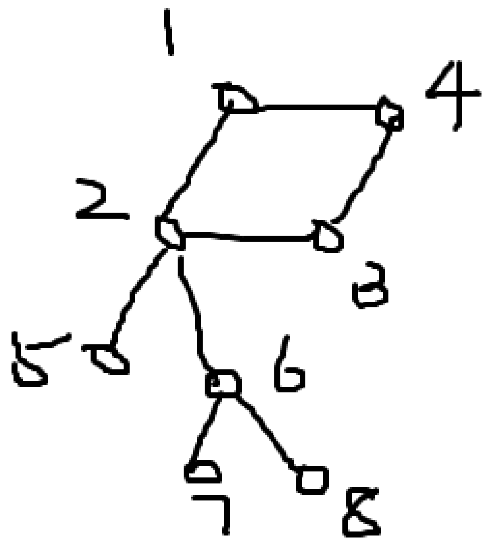
③图的特殊情况

- 接下来，让我们看一看图的形态都有哪些：
- 有向图，无向图
- 树：n 个点 $n-1$ 条边的无向连通图。
- 同时也是“无向无环图”
- 森林：由一些树组成。
- 环
- 重边和自环
- 环套树



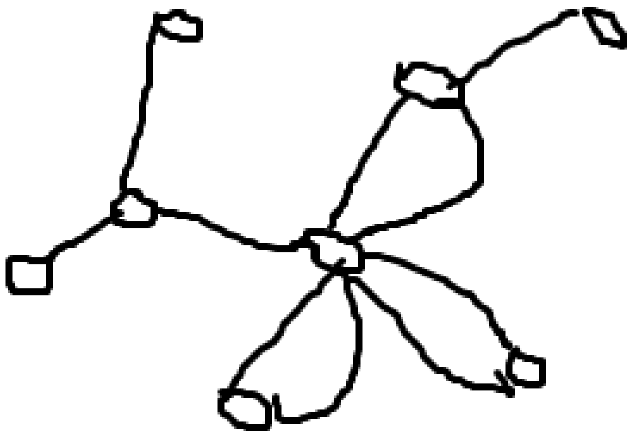


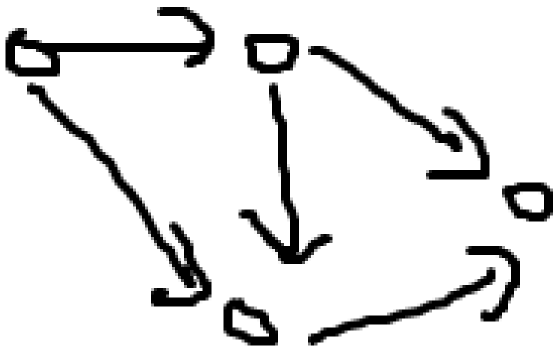


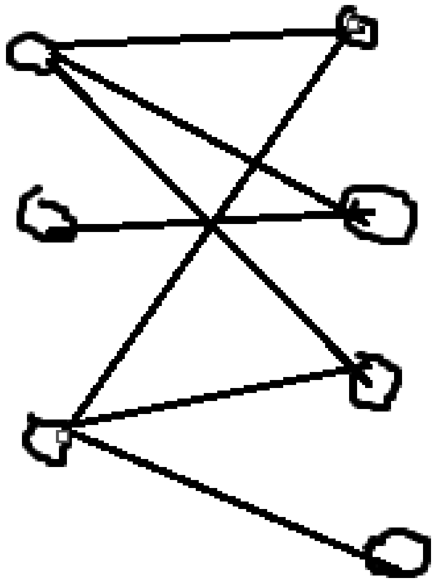


（一）图论基础知识
③图的特殊情况

- 仙人掌：每条边最多在一个环内。
- 仙人球：每个点最多在一个环内。
- DAG：“有向无环图”。
- 二分图：
- 图可以被分成两个部分。
- 每个部分内部没有连边；
- 只有两个部分之间有连边。
- 在网络流中经常用到。

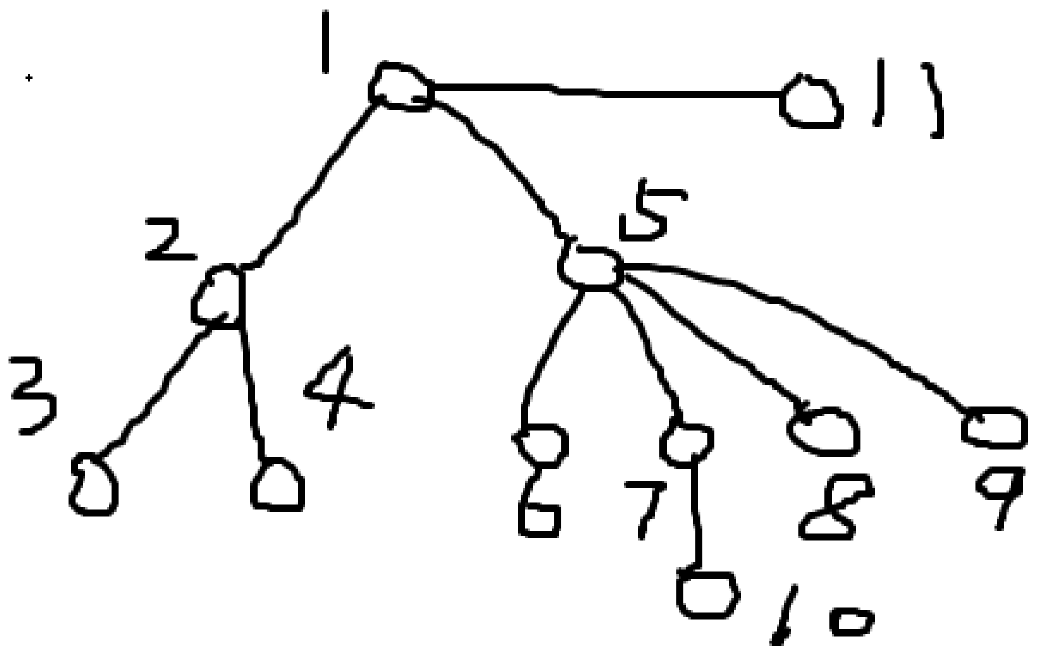






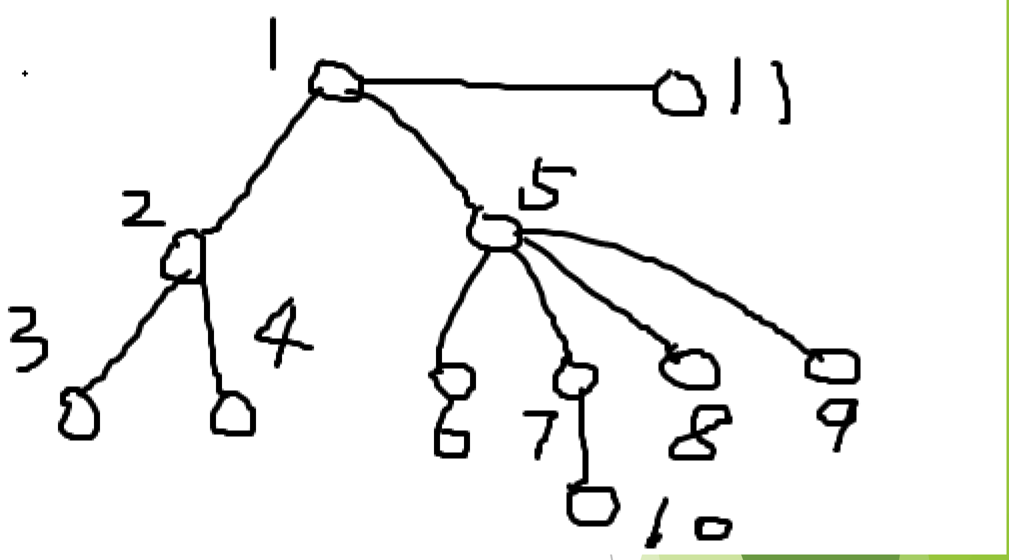
（一）图论基础知识
③图的特殊情况-树

- 我们要特别讲解一种特殊图：树
- 树是一种特殊的无向连通图
- 满足图中不存在环
- 需要掌握的概念：
  - 根节点，叶子节点
  - 父亲，儿子
  - 祖先，子树
  - 以某个点为根时，树的高度（深度）

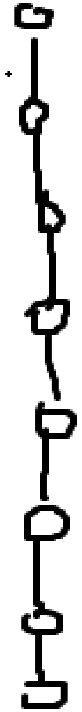


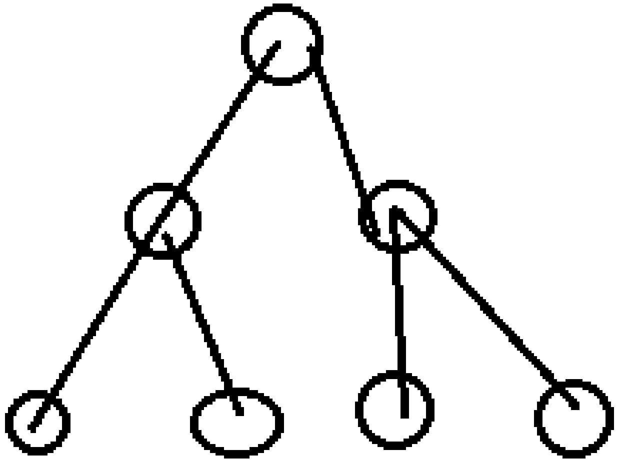
（一）图论基础知识
③图的特殊情况-树

- 树有哪些基本特征?
- ①树上不存在环
- ②点数为 n，边数一定为 $n-1$
- ③任意两点之间的最短路径唯一



（一）图论基础知识
③图的特殊情况-树

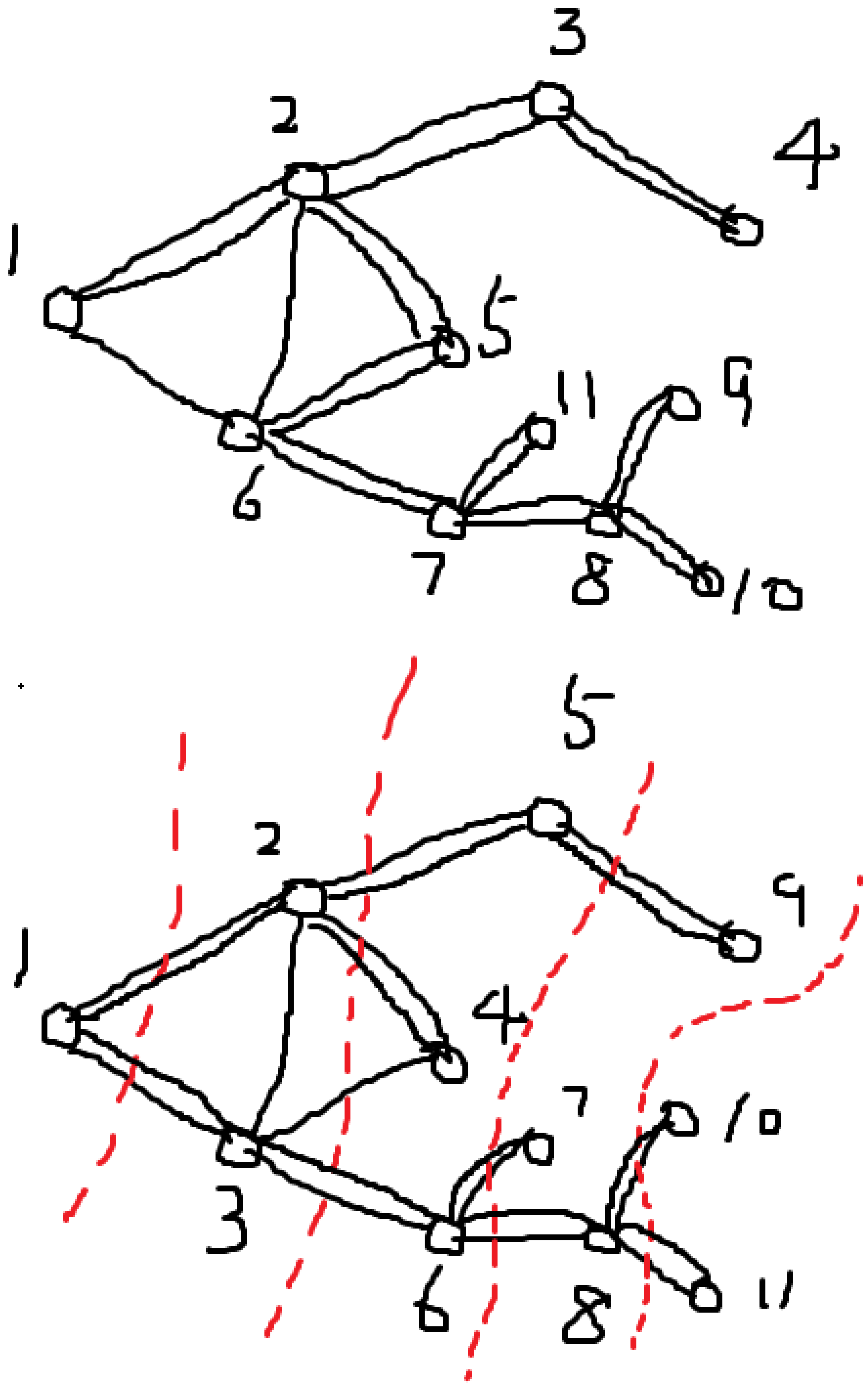




- 在题目中，经常有以下几种树的形态：
- （完全/满）二叉树，链，菊花
- 试从以下角度分析三种树的性质。
- ①挑一个点为根，树的高度最大、最小是多少?
- ②随机选一个点为根，树的高度是什么级别的?
- ③树上每个点的点度（连出去的边数）中最大、最小是多少?
- 如果我们要写部分分，就要会判断给定的是哪种树。

（一）图论基础知识
④图的存储和遍历

- 图的遍历：
- *DFS*，*BFS*
- 一张图的 *DFS* 树，*BFS* 树
- 最短路树
- *DFS* 序和 *BFS* 序
- 图的存储：
- 邻接矩阵，邻接表（边表）
- 前者：$a[i][j]$表示 i 到 j 是否有边

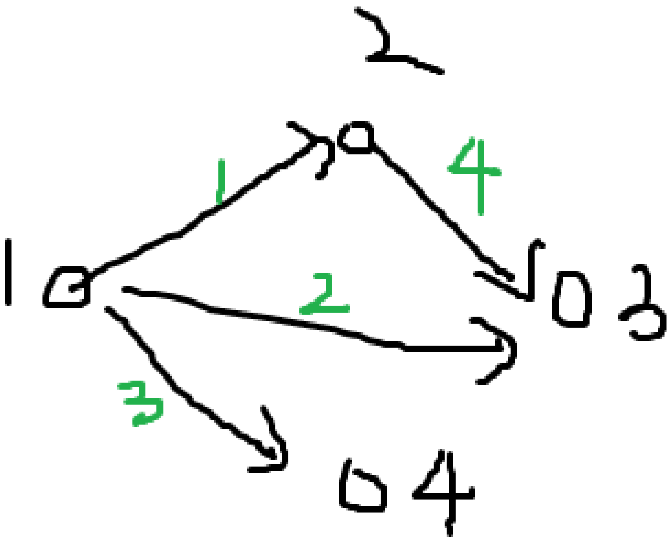


（一）图论基础知识
④图的存储和遍历

- 邻接表
- 以有向图为例，将一个点连出去的边用单向链表挂在这个点下。
- ①对于每个点，维护其连出的第一条边。
- ②对于每条边，维护其下一条边。（没有下一条边的记为 0）
- 如果要遍历一个点的出边，直接遍历链表

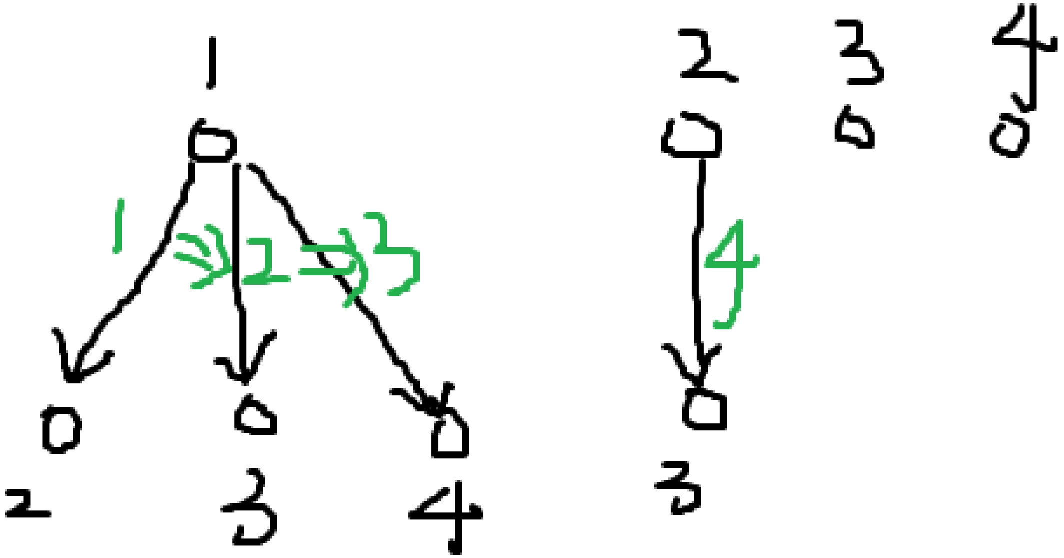
（一）图论基础知识

④图的存储和遍历



黑色的是点的编号

绿色的是边的编号



1号点连出来的第一条边为1号边

1号边的下一条边为2号边

2号边的下一条边为3号边

3号边的下一条边为0

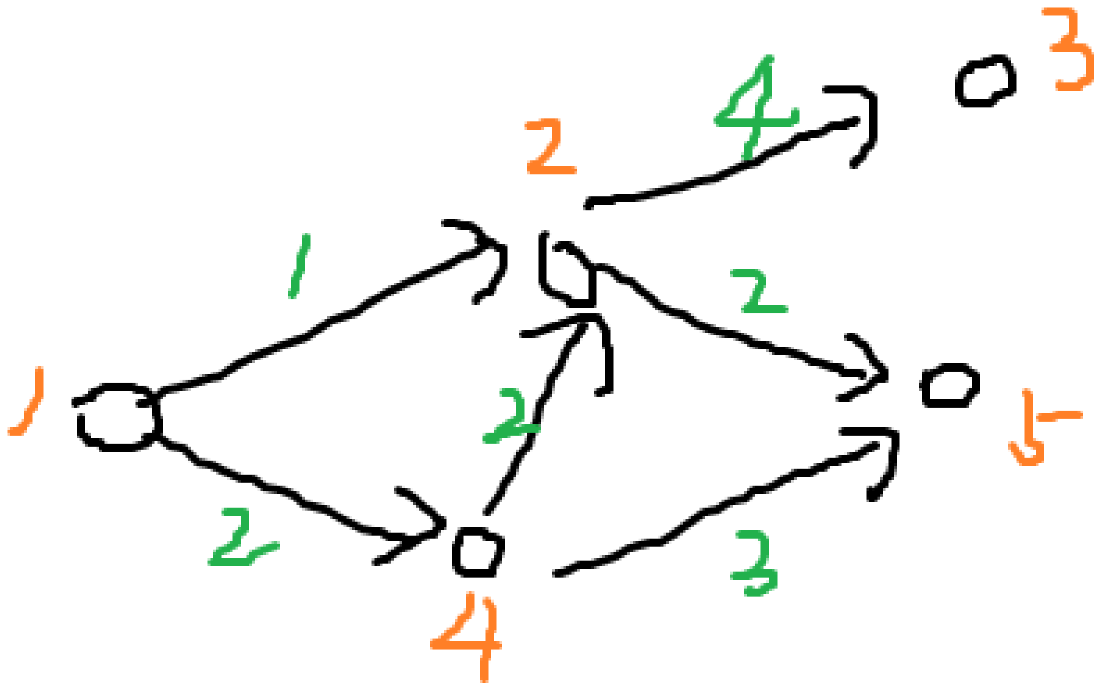
- 如果要往边表里加边，比方说加入 1 –> 5，且这条边的编号为 5。
- 现在 1 号点连出来的第一条边变为 5 号边，5 号边的下一条边为原先 1 号点连出去的第一条边，也就是 1 号边。

（二）图论算法

- ①拓扑排序
- ②最短路
- ③最小生成树
- ④最近公共祖先
- ⑤缩强联通分量

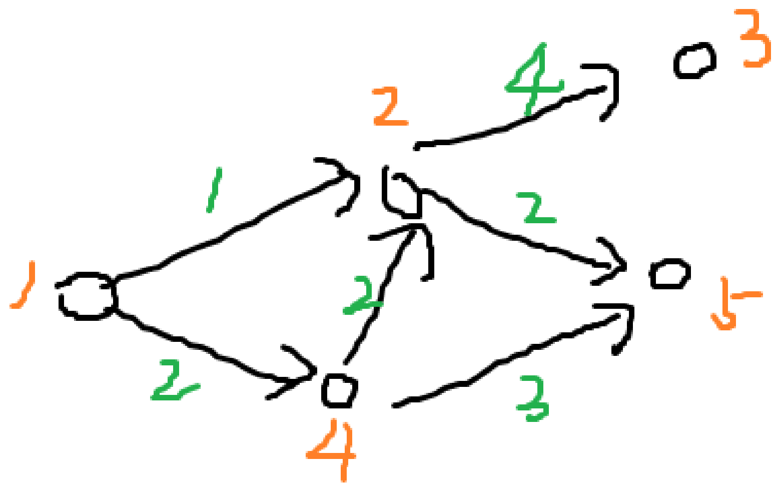
（二）图论算法

①拓扑排序：问题



- 给定一张 DAG（有向无环图）
- 并给定每条边的权值。
- 求这张图的最长链。
- 第一行输入 n, m 表示点数，边数。
- 接下来 m 行，每行输入 u_i, v_i, c_i，表示有一条从 u_i 到 v_i 权值为 c_i 的边。
- ① $n, m \leq 2000$
- ② $n, m \leq 200000$

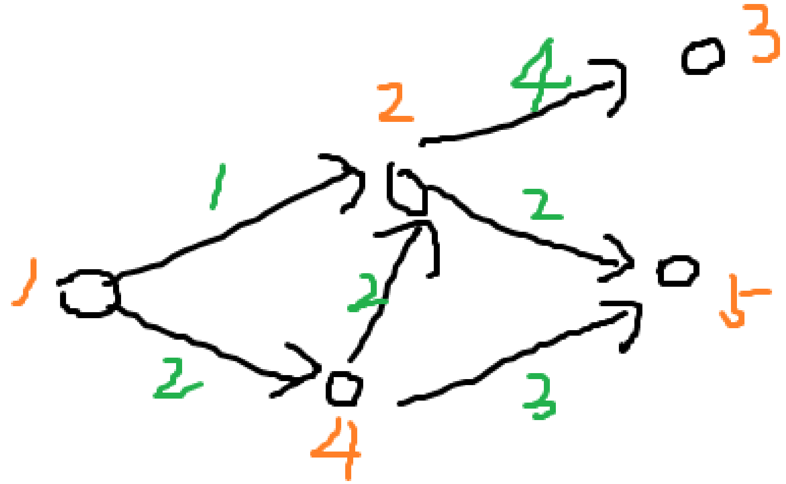
（二）图论算法
①拓扑排序：问题解决



- 我们考虑对每个点求从它开始的最长链多长。
- 对于现在出度为 0 的这些点（3、5），我们知道从它们开始的最长链为 0。
- 知道了点 3、点 5 开始的最长链，那么 2 连出来的所有点的最长链都知道了，因此点 2 开始的最长链也知道了，就是 4。
- 进一步地，点 4 开始的最长链也知道了，点 1 开始的最长链也知道了。

（二）图论算法

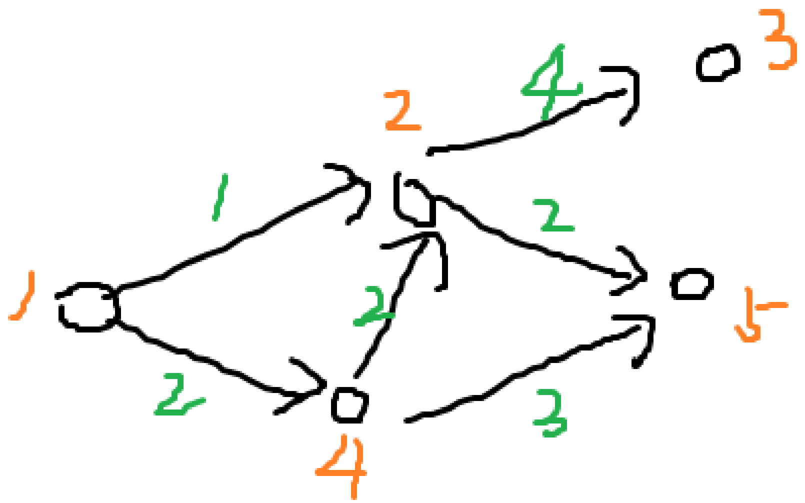
①拓扑排序：问题解决



- 我们预处理每个点的出度。
- 每次找出出度为 0 的点，用它更新那些指向它的点。
- 更新后被指向的点出度 −1。
- 例如现在点 2 的出度为 2，当我们找到点 3 这个出度为 0 的点，我们可以用其开始的最长链长度（0）加上点 2 和点 3 连边的长度（4）得到从点 2 开始的一个可能的最长链长度。
- 当点 3、点 5 都做完，即此时点 2 的出度为 0，那么点 2 的最长链就算好了。

（二）图论算法

①拓扑排序：问题解决



- 实现的时候，我们可以开一个队列。
- 队列存储当前所有出度为 0 的点。
- 每次就从队头取出一个出度为 0 的点，用它去更新那些指向它的点，如果某个指向它的点更新后出度变为 0，就把那个点也加到队列末尾。
- 按照我们的算法，要构建一个反向的图，用链表来存储。
- 这个队列中存储的点的顺序，就是一种拓扑序。
- 我们操作的过程，就是拓扑排序的过程。

（二）图论算法
②最短路：问题描述

- 讲了拓扑排序之后，我们再来看图论中另一个经典问题：最短路。
- 设图的点数为 n，边数为 m。假设边权非负。
- 问题1：给定一张有向图，求所有点对之间的最短路。
- 要求时间复杂度 $O(n^3)$ 以内。
- 问题2：给定一张有向图，求一个点S到其他所有点的最短路。
- 要求时间复杂度 $O(mlogm)$ 以内。

（二）图论算法
②最短路：*Floyd*

- 问题1：给定一张有向图，求所有点对之间的最短路。
- 用动态规划的方法。
- $F[k][i][j]$：表示除了 i 和 j 外只经过前 k 个结点，从 i 到 j 的最短路。
- 初始值 $F[0][i][j]$ 就是读入的 i 和 j 的连边长度（不存在即为正无穷）。
- 每次加入一个点。当加入了一个顶点 k 之后，最短路如果有变化的话一定是以 k 为中间顶点。

（二）图论算法
②最短路：$Floyd$

- $F[k][i][j]$：表示除了 i 和 j 外只经过前 k 个结点，从 i 到 j 的最短路。
- $F[k][i][j]=\min=F[k-1][i][k]+F[k-1][k][j]$
- k 只从 $k-1$ 转移过来，可以用滚动数组优化。
- For(k,1,n) For(i,1,n) For(j,1,n)
  - F[i][j]=min{F[i][j],F[i][k]+F[k][j]}
- 这个算法称为 $Floyd$ 算法。

（二）图论算法
②最短路：$Floyd$

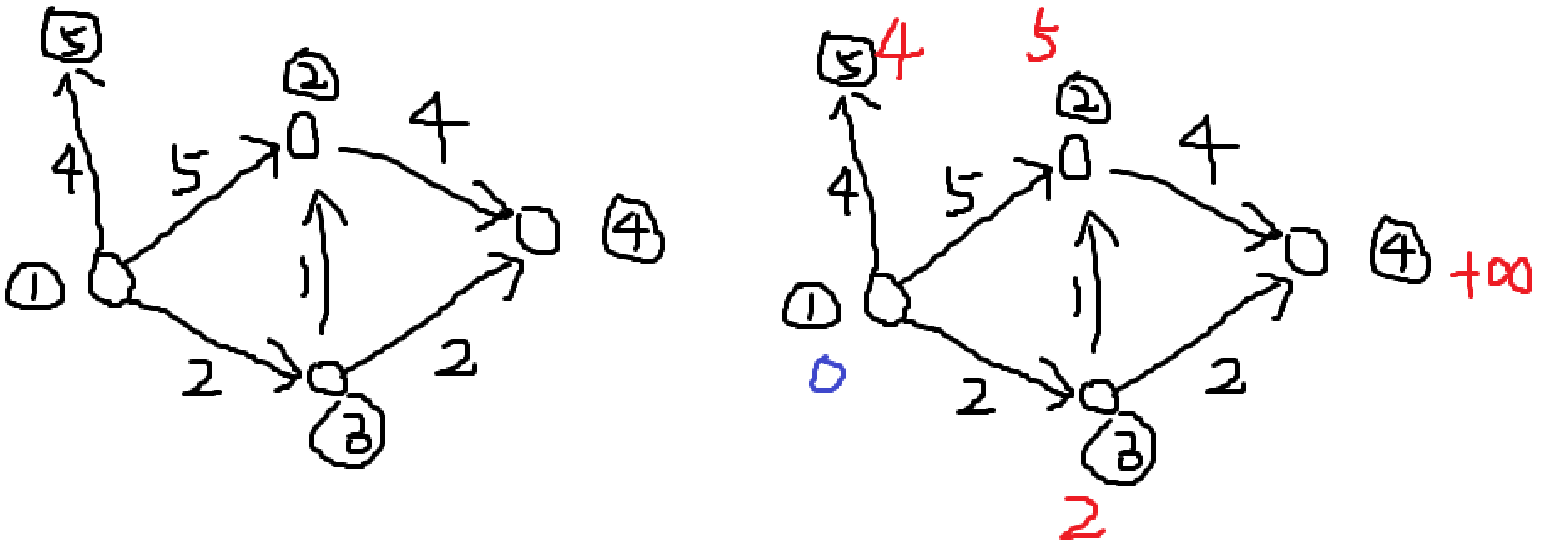
- DP 的边界是除了输入的边，其他的都设为 inf 。
- 时间复杂度为 $O(n^3)$。
- k,i,j 三个不能随意调换顺序，因为动态规划的三维含义不同。
- 如果存在负权边，$Floyd$ 算法还是正确的。

（二）图论算法
②最短路：*Dijkstra*

- 问题2：求一个点S到其他所有点的最短路。
- 用贪心的方法。
- 从起点开始往外拓展。
- 假设当前已经知道了到一些点的最短路。
- 可以得到目前情况下，与这些点相邻的点的最短路。
- 选择其中最短路最小的点走过去。
- 重复上述操作，每次会多得到一个点的最短路，这样一直贪心地走直到走到终点。

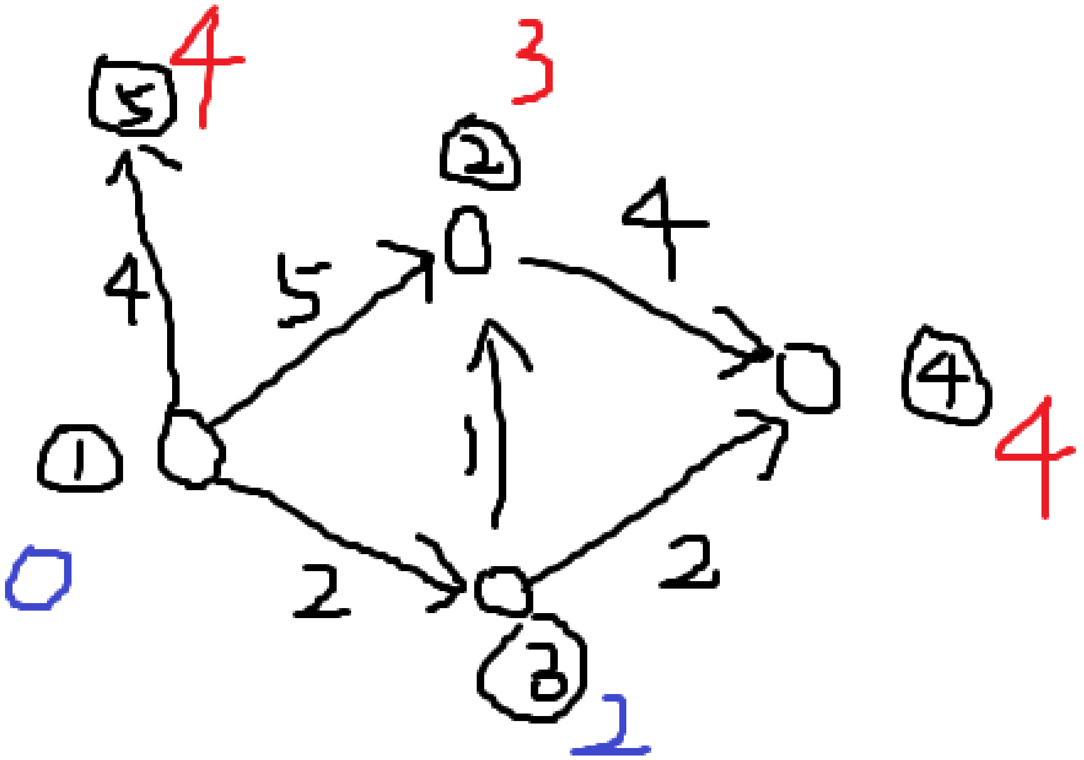
（二）图论算法

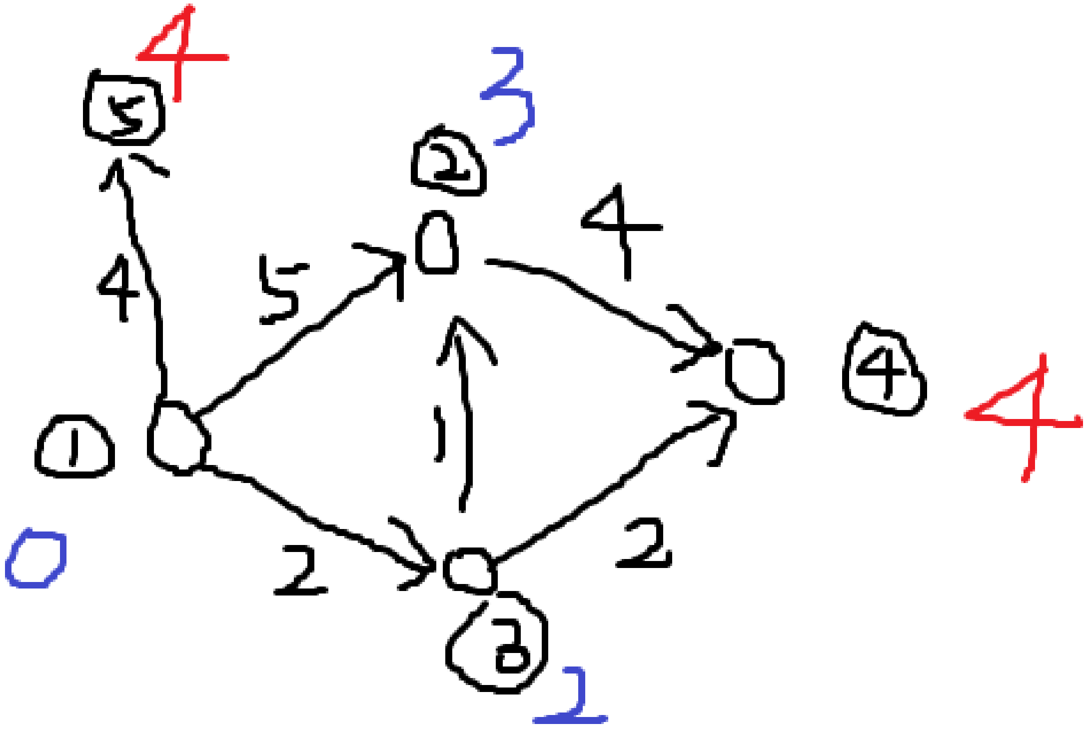
②最短路：*Dijkstra*



（二）图论算法

②最短路：*Dijkstra*





（二）图论算法
②最短路：$Dijkstra$

- 朴素方法：每次枚举一遍所有还未确定最短路的点，从其中找出最短路最小的点。
- 确定这个点的最短路。
- 用这个点的最短路加上一条边的长度去更新与它相邻的点的最短路。
- 时间复杂度是 $O(n^2+m)$。

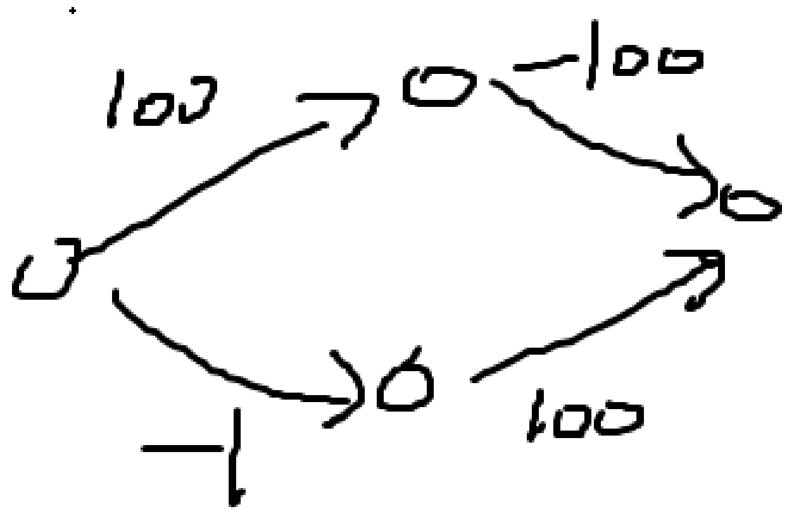
（二）图论算法
②最短路：*Dijkstra*

- 优化方法:
- 如果把还未确定最短路的{点，最短路}看成集合，朴素方法每次是进行以下操作：取出集合中最短路最小的点，并将其从集合中删去；最短路有更新，向集合中插入新元素。
- 一开始正无穷的可以不加进集合。
- 用堆维护一个结构体Data，包含点的编号和最短路长度（堆的关键字是最短路长度），每次取出堆中最小的元素，用这个元素去更新其相邻的点的最短路。
- 如果某点最短路长度有变，直接把新的结构体加进去。

（二）图论算法

②最短路：*Dijkstra*

- 注意到一个点只会被用来更新别的点一次，所以每次更新遍历一遍其出版，所以每条边只会被用于更新一次。总共有 m 条边。
- 所以时间复杂度（由于使用了堆）是 $O(mlogm)$。
- 空间复杂度是 $O(m)$ 的。
- 如果存在负权边，*Dijkstra* 算法还正确吗？



（二）图论算法
②最短路：$SPFA$

- 问题2还有一个解法：$SPFA$。
- 如果一张图边权全为 1，我们可以用 BFS。而 $SPFA$ 的思想类似 BFS。
- 我们维护一个队列，表示当前接下来要更新的点。
- 对于队列的头元素，枚举它的出边，更新其连向的点的最短路。
- 如果该点最短路变小，则把它加到队列末尾。
- 如果已经在队列中，就可以不用加。
- 由于边权不全为 1，一个点可能会重复入队。

（二）图论算法
②最短路：$SPFA$

- 我们可以把 $SPFA$ 就简单理解为一个点可能重复入队的 BFS。
- $SPFA$ 是一个复杂度不靠谱的算法，但是有时非常实用的小技巧。
- 最坏复杂度 $O(nm)$。（完全图等可以卡满）
- 稀疏图的期望复杂度 $O(km)$，k 是一个小常数。

（二）图论算法
②最短路：差分约束

$a - b \le k_1$①

$b - c \le k_2$②

$a - c \le k_3$③

- 问题描述：有 a, b, c 三个数，k_1, k_2, k_3 为常数，它们满足右上角这样的关系：
- 求 $a - c$ 的最大值?
- $a - c$ 受哪些条件约束呢?
- 由③，得 $a - c \le k_3$
- 由①+②，得 $a - c \le k_1 + k_2$
- 那么 $\max(a - c) = \min(k_1 + k_2, k_3)$

$$a - b \le k_1 \text{①}$$

$$b - c \le k_2 \text{②}$$

$$a - c \le k_3 \text{③}$$

（二）图论算法
②最短路：差分约束

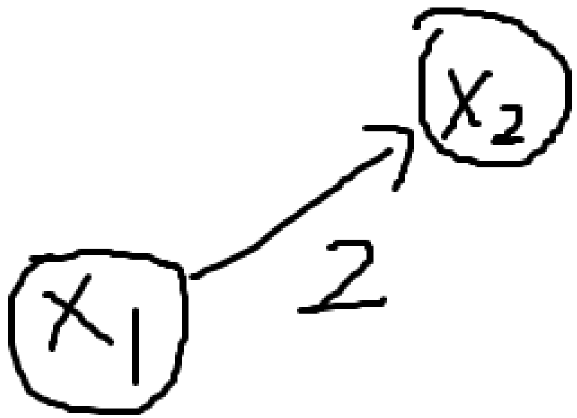
- $\max(a-c) = \min(k_1 + k_2, k_3)$
- 我们把 a 和 b 连边，权值为 k_1；b 和 c 连边，权值为 k_2；a 和 c 连边，权值为 k_3（单向边）。
- 源为 a，汇为 c，跑一遍最短路即可

- 同理对于 n 个变量，以及 m 个限制条件，形如
- $x_i + c_{ij} \ge x_j$
- 可以把每个限制条件看成一条边，跑Dijk、SPFA等

（二）图论算法
②最短路：差分约束

- Q1：问是否存在解：等价于判断是否存在负环
- 怎么判断一张图正环（或者负环）？
- 如果源点 S 能走到一个负环，这个负环的最短路可以无限小。
- 另外，若一个点入队次数大于节点数，则存在负环。
- 还有更快的 DFS 判断负环的方法。
- 见 2009 年论文《$SPFA$ 的优化与应用》

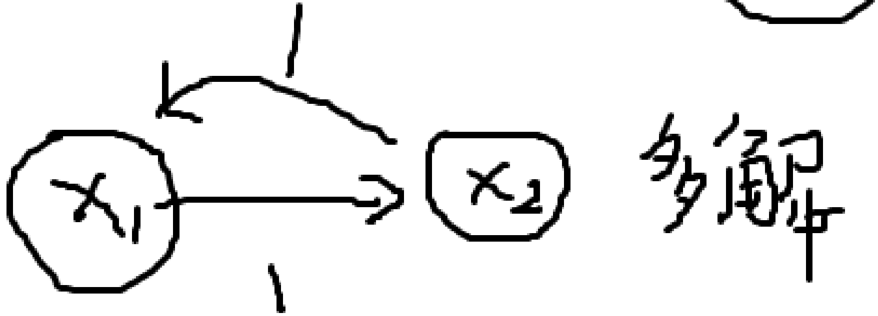
（二）图论算法
②最短路：差分约束



- Q2：问两个变量差的最大值
- 假设有方程 $x_1 + 2 \geq x_2$，那么由最短路算法可以得到 x_1 到 x_2 的最短路是2。
- 这个就是 $x_2 - x_1$ 的最大值，可以看做 x_1 等于 0 时 x_2 的最大值。
- 也就是说 x_2 还可以更小而不能更大。
- 注意我们用最短路算法求出的这个值是最大值而不是最小值。

（二）图论算法
②最短路：差分约束

- Q3：判断解是否唯一
- 可以看右图这个例子。



- 注意到如果 x_1 到 x_3 的最短路等于 x_3 到 x_1 的最短路的相反数，那么解就只有一个。
- 我们先对原图求一遍最短路。
- 然后将原图取反，边权取反，求一遍最长路。
- 一个对应的是能取到的最小值，一个是最大值。
- 如果相同则解唯一。

（二）图论算法
②最短路：$01BFS$

- 问题描述：对于给定的一张有向图，边权只有 0 和 1，求某个点到所有点的最短路。
- 我们把 bfs 的队列换成双端队列，也就是可以在头部加元素的队列。当我们取出队首，用队首更新别的点的时候，如果连出去的边是 0，那么我们希望优先增广这条边连出去的那个点。
- 如果边权是 0，连出去那个点加入在队列头部。
- 否则是边权是 1，仍然加在队列尾部。
- 时间复杂度 $O(m)$。

（二）图论算法
②最短路：总边权不超过W的BFS

- 问题描述：对于给定的一张有向图，保证边权和不超过 W，边权为正，求某个点到所有点的最短路。
- 使用 $Dijkstra$ 算法。
- 性质：如果用 x 更新周围的点 t 的最短路，那么源到 t 的最短路长度一定大于到 x 的最短路长度。
- 用 0..W 的桶+链表代替堆。
- 从小到大枚举值来取出当前最小值。根据性质，加入的元素一定只会加到当前枚举的这个值的后面。
- 时间复杂度 $O(m+W)$。

（二）图论算法
③最小生成树：问题

- 说完了最短路，我们再来讲一下图论中另一个经典问题：最小生成树。
- 给定一张 n 个点 m 条边的无向联通图，每条边会给定一个边权。
- 要求你选出最少的条边，在保持图联通的情况下，最小化选出的边的边权和。
- 第一种数据：$n \le 5000, m \le n^2$
- 第二种数据：$n, m \le 500000$

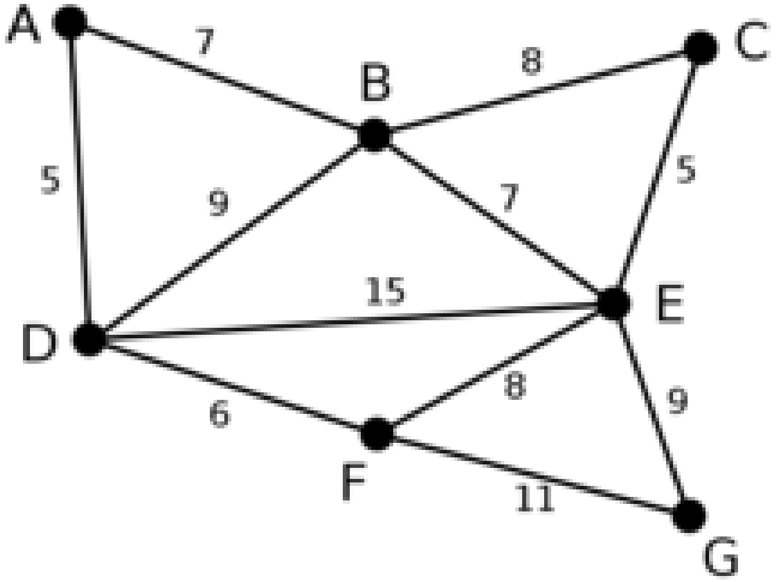
（二）图论算法
③最小生成树：朴素算法

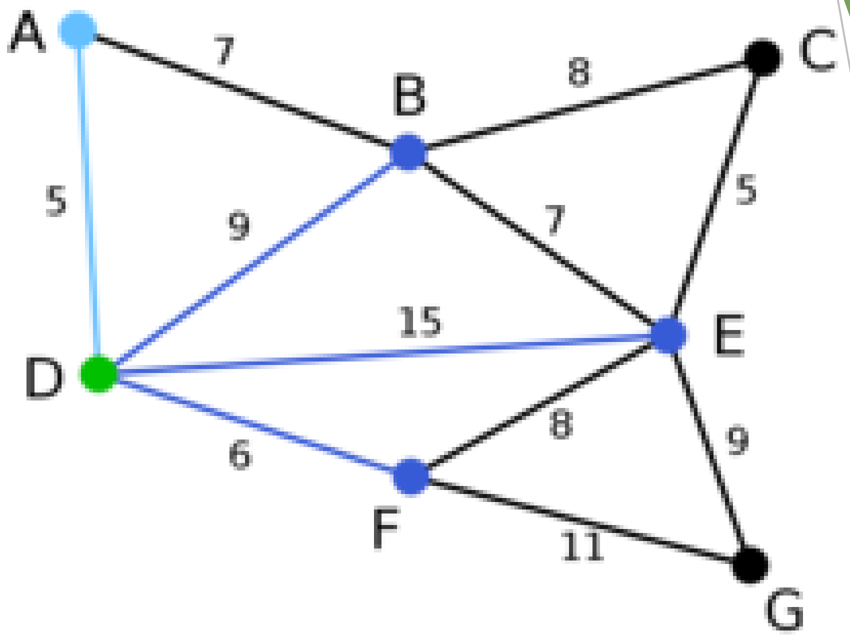
- 方法一：
- 直接枚举每条边是否在树上，$O(n)$ 判断是否连通。
- 时间复杂度 $O(2^m * n)$。
- 方法二：
- 用 Dfs 枚举 m 条边中选的是哪 $n-1$ 条边，这样枚举的复杂度是组合数。
- 时间复杂度 $O(C(m,n) * n)$。

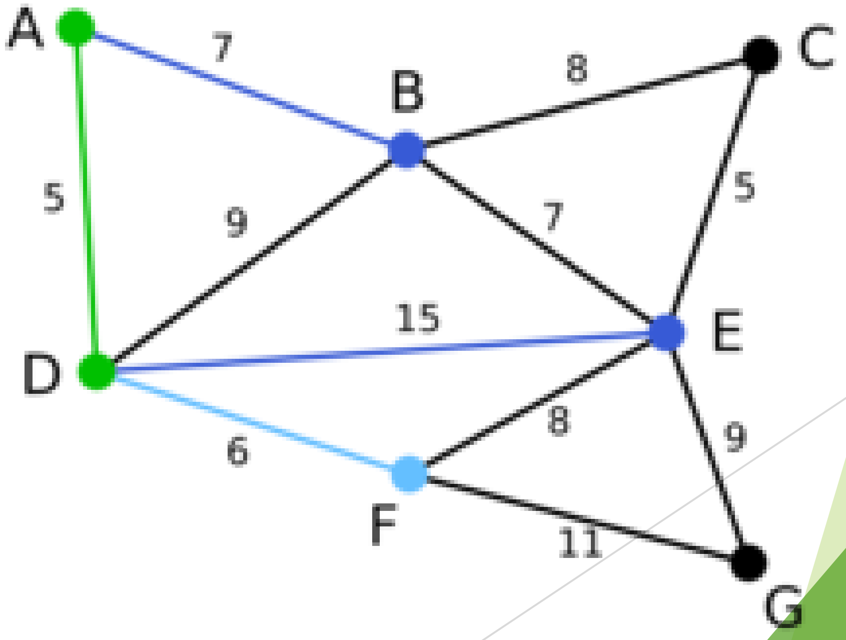
（二）图论算法
③最小生成树：*Prim*

- 这个题除了朴素算法以外，还可以用贪心的方法。
- 从任意一个点开始，像Dijkstra一样每次从相邻一层的点中拓展。
- 只不过不是拓展“到源点最近的点”，而是“从所有相邻边中选最小的一条”。
- 或者说是选择有最小连边的那个点。

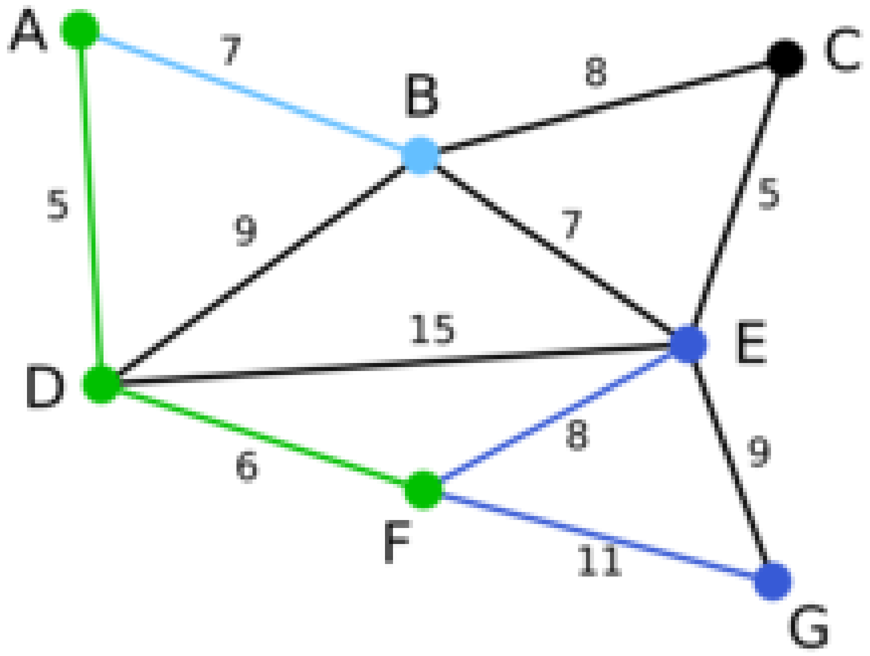
（二）图论算法
③最小生成树：*Prim*

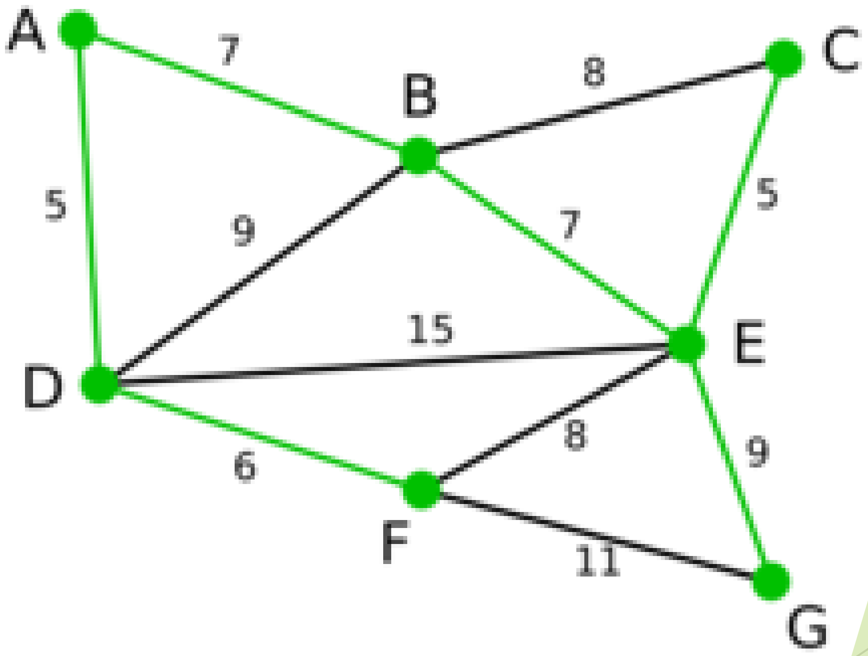






（二）图论算法
③最小生成树：*Prim*





（二）图论算法
③最小生成树：*Prim*

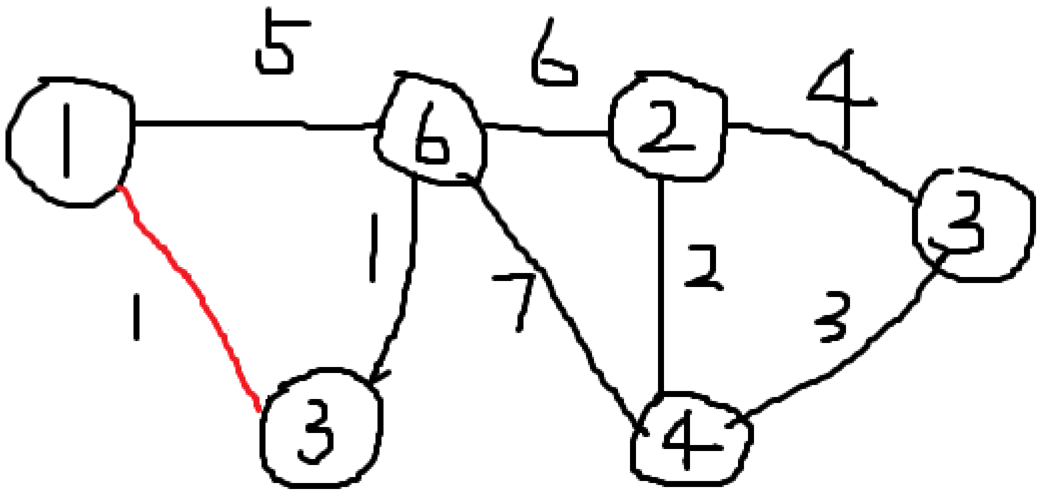
- 记录 $f[i]$ 表示当前由已经选出来的最小生成树到 i 点的最小边是多少。（如果不能到就设为正无穷）
- 那么每次就是找出还没有加进最小生成树的点中，最小边最小的点（枚举至多$O(n)$）。
- 现在我们把这个点加进最小生成树。这个点可能会连出去一些边（至多n条），这些边连出去的点的f的值要更新。
- 每次是操作最多是 $O(n)$ 的，我们一共会加 $O(n)$ 次点，因此时间复杂度是 $O(n^2)$。内存复杂度是 $O(n)$。

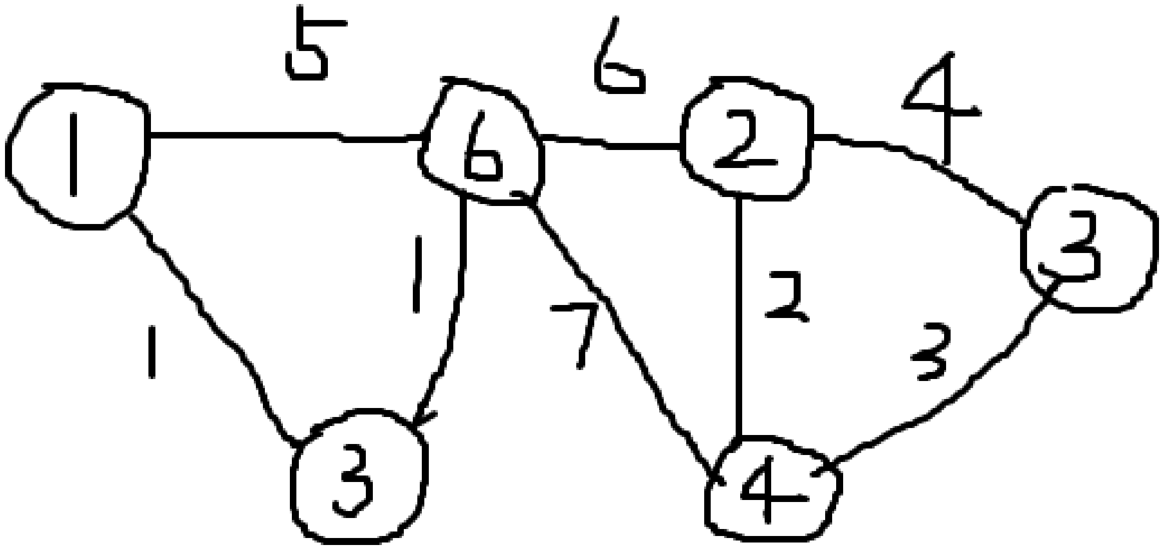
（二）图论算法
③最小生成树：$Kruskal$

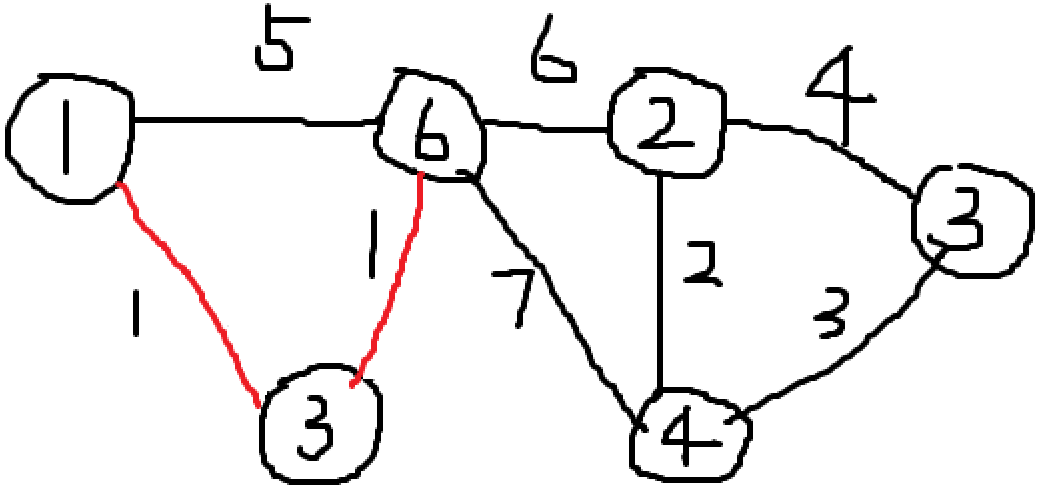
- 我们还有一个算法： $Kruskal$ 算法
- 同样是用到贪心，但是是不同的贪心。
- 算法过程是：把边从小到大排序，按顺序加到图上。
- 如果加入一条边 (u,v,c) 的时候 u 和 v 已经连通了，就不加这条边。
- 否则 u,v 在两个连通块内，此时就加这条边。

（二）图论算法

③最小生成树：*Kruskal*

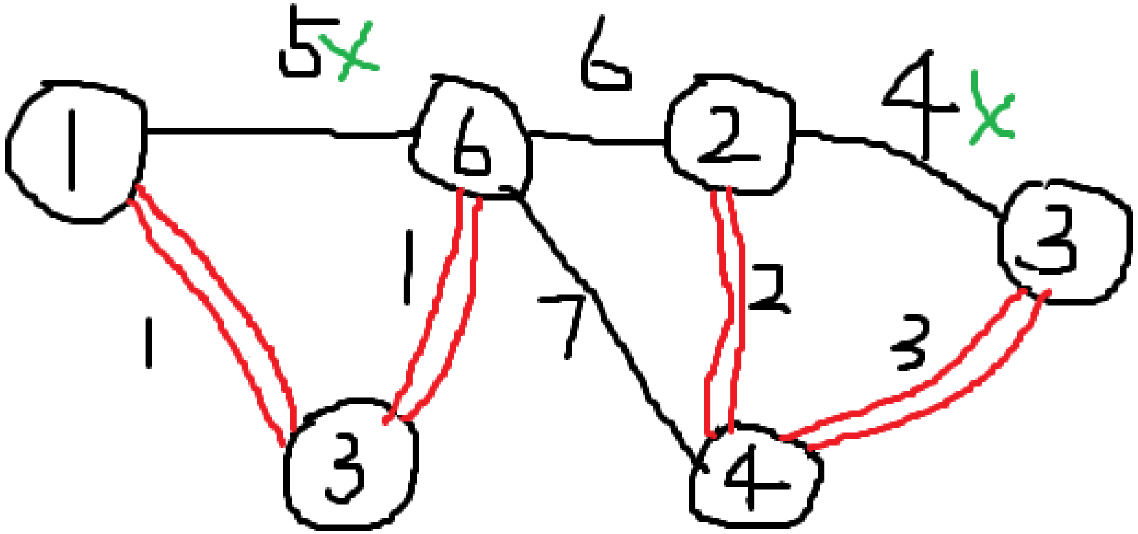


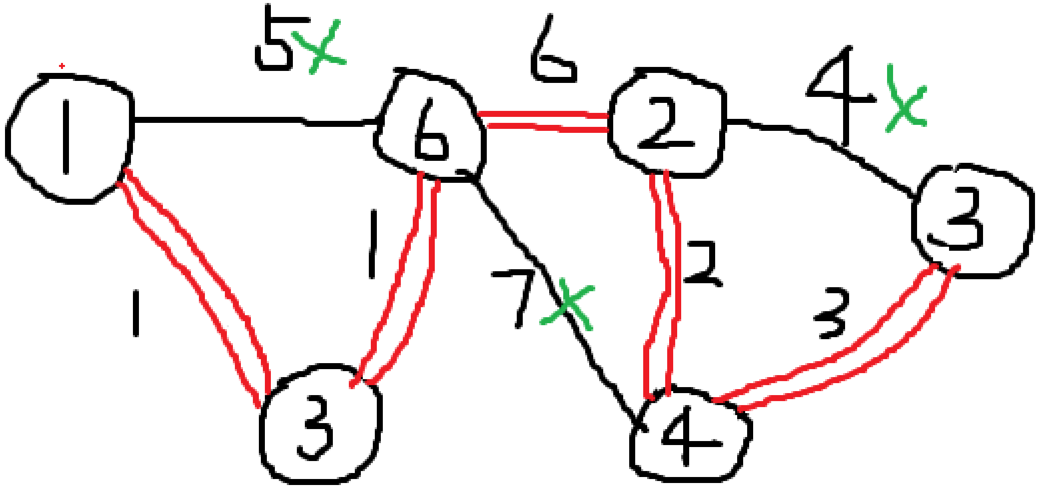


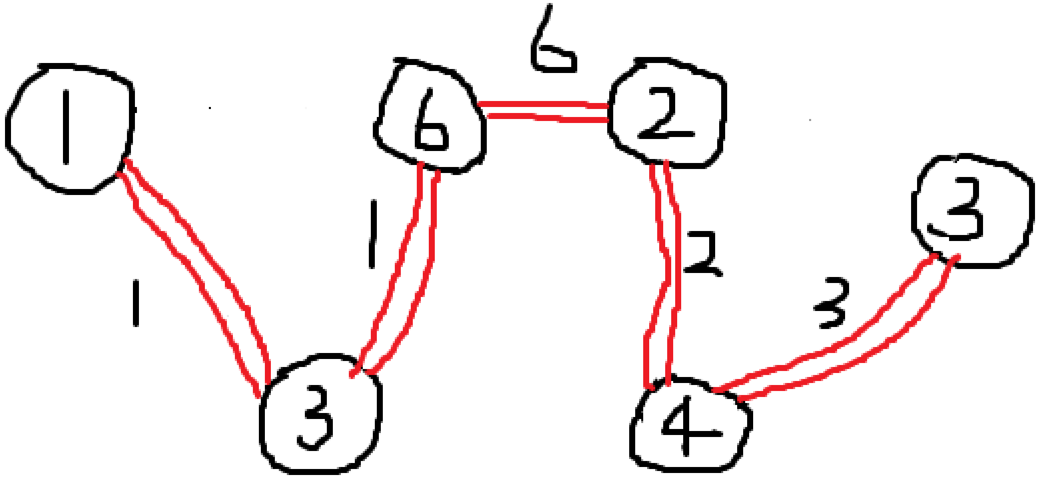


（二）图论算法

③最小生成树：*Kruskal*







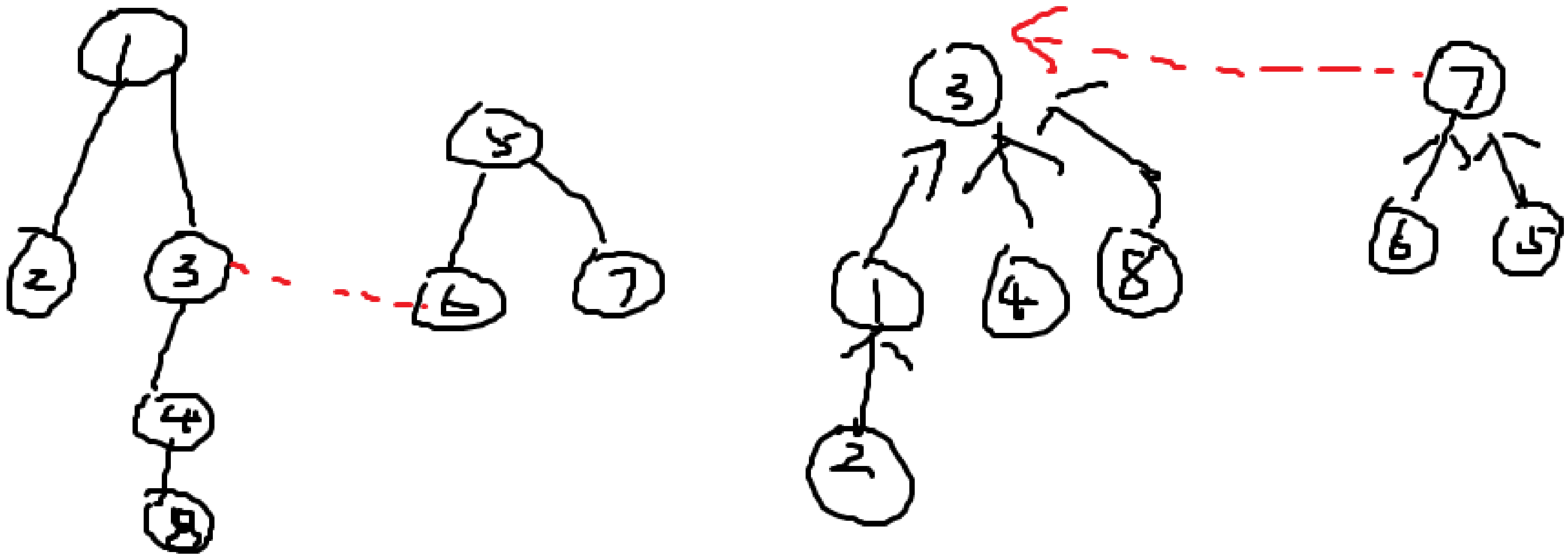
（二）图论算法
③最小生成树：$Kruskal$

- 怎么支持连接两个点，同时动态判断两个点是否连通?
- 数据结构：并查集，时间复杂度（单次）不超过$O(logn)$。
- 并查集的代码特别简洁:
- int Fa[];
- int Find(int x) { return Fa[x] == x ? x : Fa[x] = Find(Fa[x]); }

（二）图论算法
③最小生成树：*Kruskal*

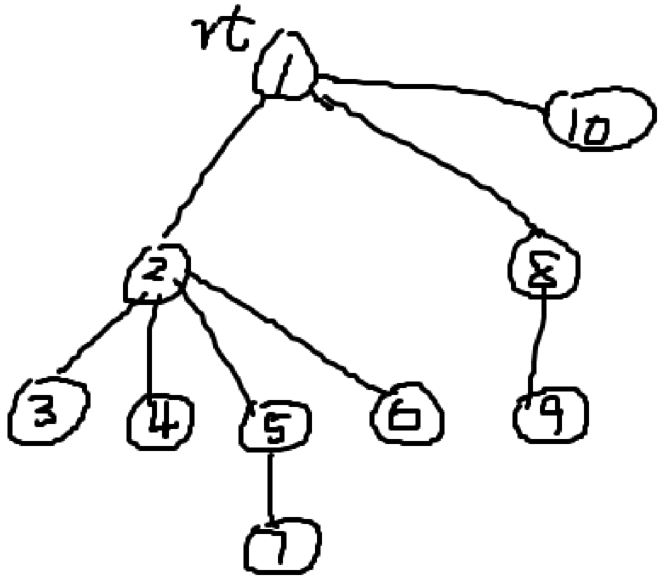
- 把在一个连通块内的点用有向树连起来。
- ①查询两个点是否在一个连通块内：比较树根是否相等。
- ②连接原图的一条无向边：连接两棵有向树的树根。
- 注意到这样树高可能会很高。我们可以在访问到一个点的时候，强行把它缩到根上去变成菊花。
- 一开始每个点单独一棵树，Fa 指向自己。

（二）图论算法
③最小生成树：*Kruskal*



- 连接两个点 u 和 v，先找到其在新有根树上的树根。
- 然后直接把 rtu 和 rtv 连起来。

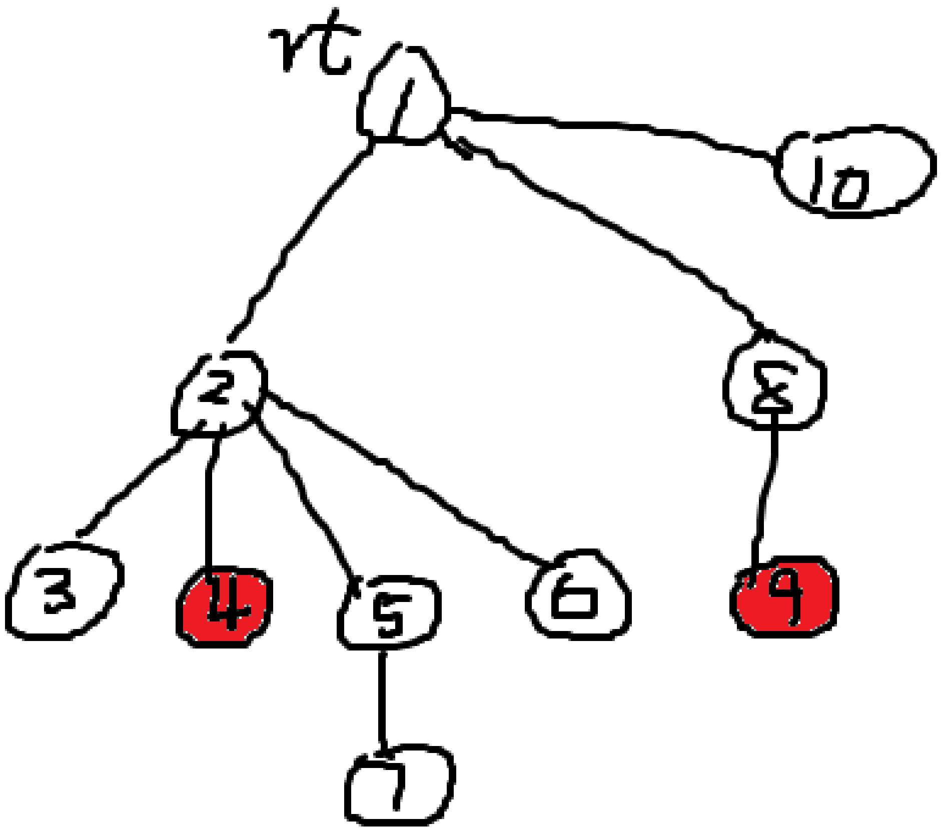
（二）图论算法
④最近公共祖先：问题



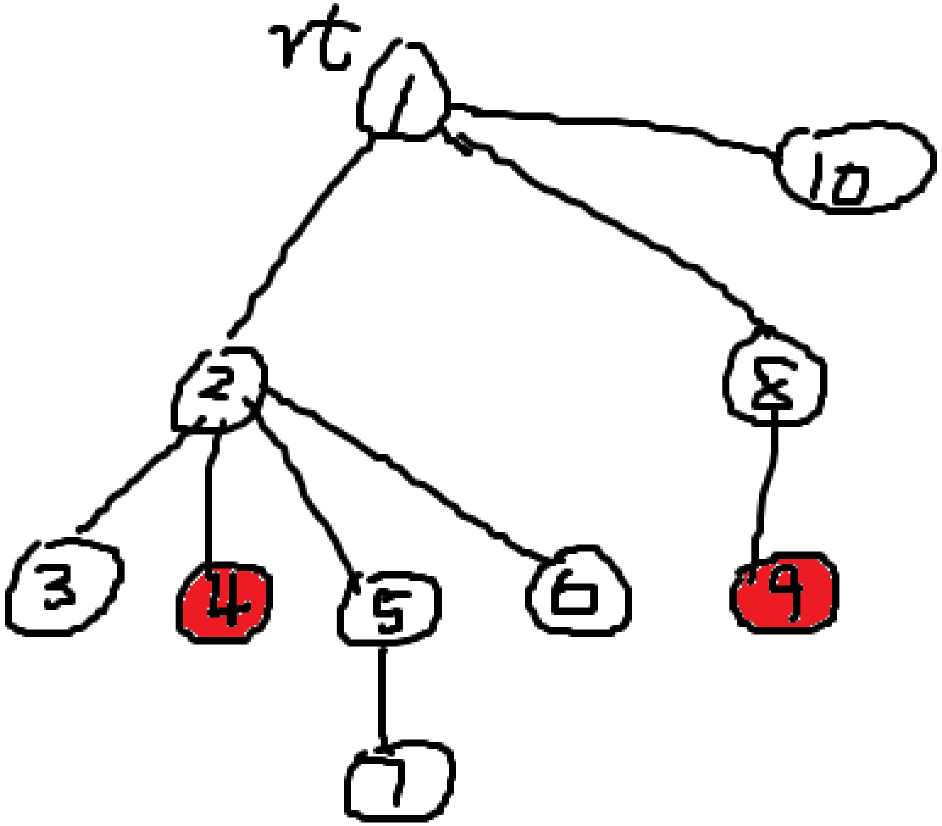
- 请看右上角的树。
- 最近公共祖先问题：LCA（Lowest Common Ancestor）
- 给定一棵有根树，有多组询问，每次询问两个点的最近公共祖先。
- 最近公共祖先就是在两个点的所有祖先中选出深度最大的那个。
- 输入正整数 n 表示树的点数，树的根为 1 号点。
- 然后输入树。
- 整数 q 表示询问组数。每组询问输入 u, v。
- 对于每组询问，你要输出一行一个正整数表示答案。

（二）图论算法
④最近公共祖先：朴素算法

- 方法一：
- 从 u 点往祖先跳。
- 给每个祖先（包括自己）打标记。
- 然后从 v 点往祖先跳。
- 遇到第一个有标记的点就是 LCA。



（二）图论算法
④最近公共祖先：朴素算法



- 方法二：
- u, v 同时跳。
- 哪个深跳哪个。
- 直到 u 和 v 重合。
- 两种方法都需要预处理，即：在所有询问之前，要从 1 号点（也可以是别的点）开始遍历一遍树，找出每个点的父亲。
- 方法二还要预处理出每个点的深度。

（二）图论算法
④最近公共祖先：倍增算法

- 什么样的图会让这种方法变得很慢?
- 从根连出来两条链，这样会使“跳”跳得特别慢。
- 怎么加速这个过程?
- 我们需要仔细分析这个“跳”究竟是怎么跳的。比方说方法二，我们会先把比较深的点跳到和比较浅的点同样高，然后两个点分别往上跳一格（可以看做同时往上跳），直到跳到相同的点为止。

（二）图论算法
④最近公共祖先：倍增算法

- 即：
- ①把比较深的点跳到和比较浅的点同样高
- ②找出一个最小距离，使两个点同时往上跳这个距离之后相等。
- 要解决第一步，必须要知道每个点往上跳一定高度会到哪。
- 如果记录 $Fa[i][j]$ 表示 i 往上跳 j 步到哪，这个数组得 $O(n^2)$，肯定是不行的。

（二）图论算法
④最近公共祖先：倍增算法

- 我们可以记录 $Fa[i][j]$，表示从 i 号点往上跳 2^j 步到达哪个点。
- 初始情况：$Fa[i][0]$ 就是 i 点树上的父亲。
- 即只记录一部分 j。
- 如果要从 i 号点往上跳 k 步，就把 k 在二进制下分解成几个 2 的次幂，利用 Fa 就可以一次多跳几步。

（二）图论算法
④最近公共祖先：倍增算法

- 例如从 3 号点往上跳 5 步:
- 先把 5(101) 看成 4 步 + 1 步。
- 先从 3 号点往上跳 4 步，4 步是 2 的 2 次方，答案是 $Fa[3][2]$。
- 然后再往上跳 1 步，1 步是 2 的 0 次方，是 $Fa\left[Fa[3][2]\right][0]$。
- 因此从 3 号点往上跳 5 步，跳到的点是 $Fa\left[Fa[3][2]\right][0]$。
- Fa 数组可以在预处理的时候顺便完成。
- $Fa[i][j] = Fa\left[Fa[i][j-1]\right][j-1]$

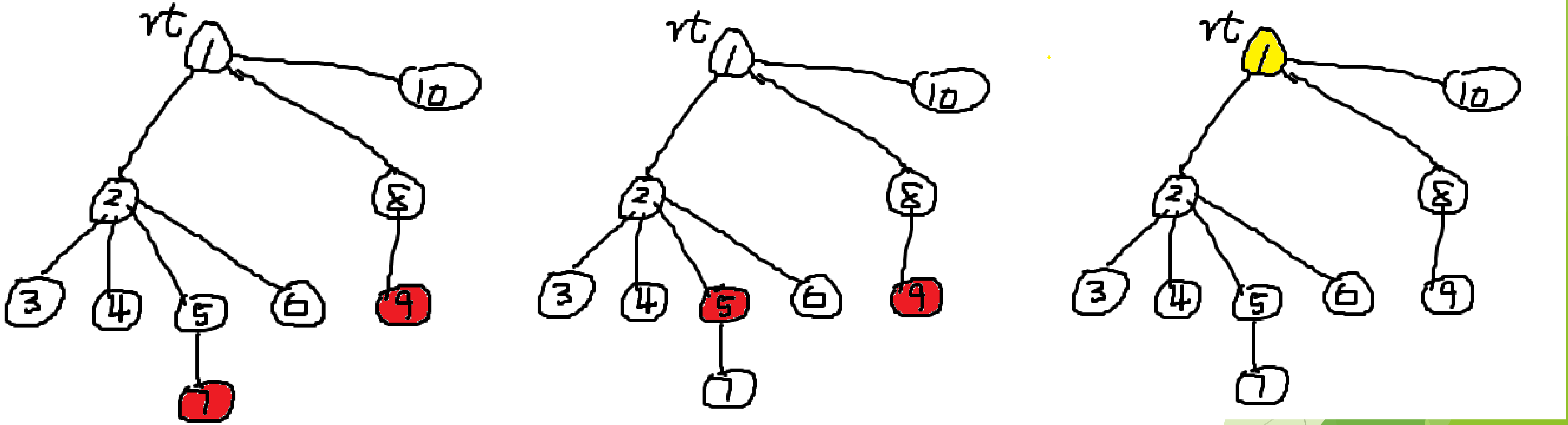
（二）图论算法
④最近公共祖先：倍增算法

- $Fa[i][j]$ 表示从 i 号点往上跳 2^j 步到达哪个点
- ①把比较深的点跳到和比较浅的点同样高
- 通过预处理每个点的深度就知道要跳的步数 k。
- 如果要从 i 号点往上跳 k 步，就把 k 在二进制下分解。
- 再利用 Fa 逐一跳上去。
- 可以先从大到小或者从小到大枚举 j。
- 然后用左移/右移和与运算判断 k 的第 j 位是否为 1。

（二）图论算法
④最近公共祖先：倍增算法

- $Fa[i][j]$ 表示从 i 号点往上跳 2^j 步到达哪个点
- ②找出一个最小距离，使两个点同时往上跳这个距离之后相等。
- 同样利用 $Fa[i][j]$ 数组。
- 从大到小枚举 j，如果 $Fa[u][j]=Fa[v][j]$，则跳得太多了。
- 否则我们可以跳一步，即令 $u=Fa[u][j], v=Fa[v][j]$。
- 注意这样最后可能会跳到 LCA 的两个儿子，还要再跳一步。

（二）图论算法
④最近公共祖先：倍增算法

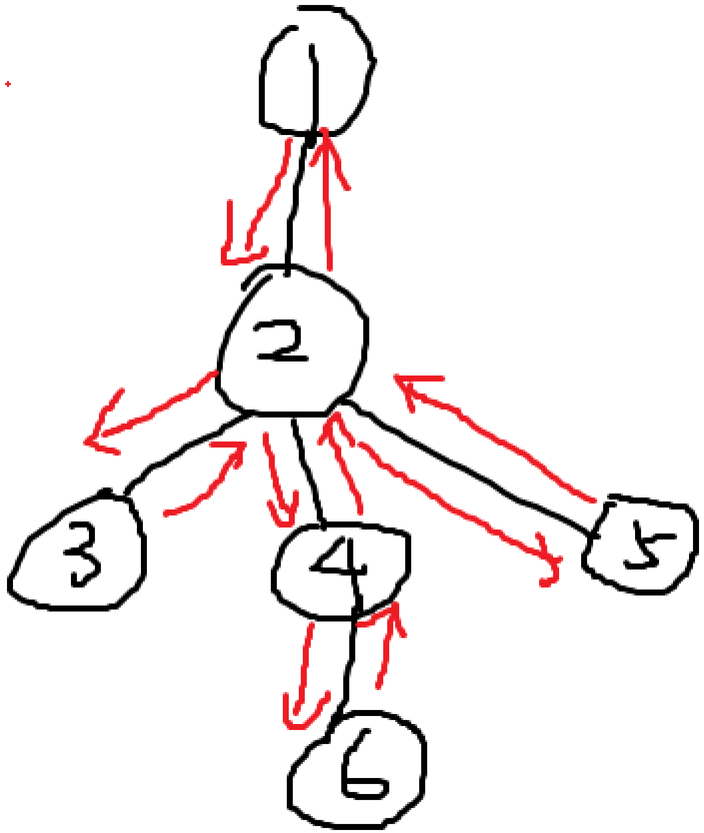


- 时间：预处理 $O(nlogn)$，询问 $O(qlogn)$。
- 空间 $O(nlogn)$。

（二）图论算法
④最近公共祖先

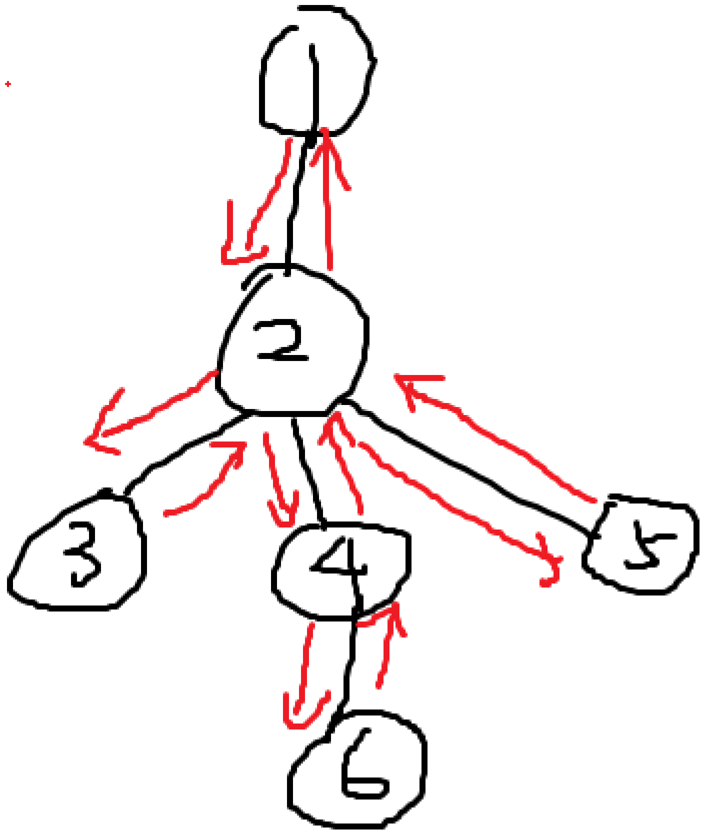
- 之前的方法比较容易想到，但是无论是时间还是空间都不优秀。
- 预处理 $O(nlogn)$，询问 $O(qlogn)$，空间 $O(nlogn)$。
- 上述算法常数比较满。
- 利用树的性质的方法： RMQ。
- 预处理 $O(nlogn)$，询问 $O(q)$，空间 $O(nlogn)$。
- 还有一种方法：树链剖分。
- 预处理 $O(n)$，询问 $O(qlogn)$，空间 $O(n)$。

（二）图论算法
④最近公共祖先：RMQ



- 两个点在树上的 LCA 有什么性质?
- 利用欧拉序:
- 每次走过一条边，就把到达的点加进欧拉序。
- 一种可能的欧拉序是 1,2,3,2,5,2,4,6,4,2,1
- 长度恰好为 $2n-1$。原因是一条边会进一次出一次，一共 $n-1$ 条边，加上一开始的 1。
- 再标出对应节点的深度
- 1,2,3,2,3,2,3,4,3,2,1

（二）图论算法
④最近公共祖先：RMQ



- 1,2,3,2,5,2,4,6,4,2,1
- 1,2,3,2,3,2,3,4,3,2,1
- 求两个点 u,v 的 LCA:
- 例子：3 和 6 的 LCA 是 2。
- 先找出这两个点在欧拉序上对应的位置。
- 可能有很多次出现，可以任意找一次，不妨找第一次出现的位置，位置不妨设为 x,y。
- 那么只要找到 $[x,y]$ 区间内深度最小的点就可以了

（二）图论算法
④最近公共祖先：RMQ

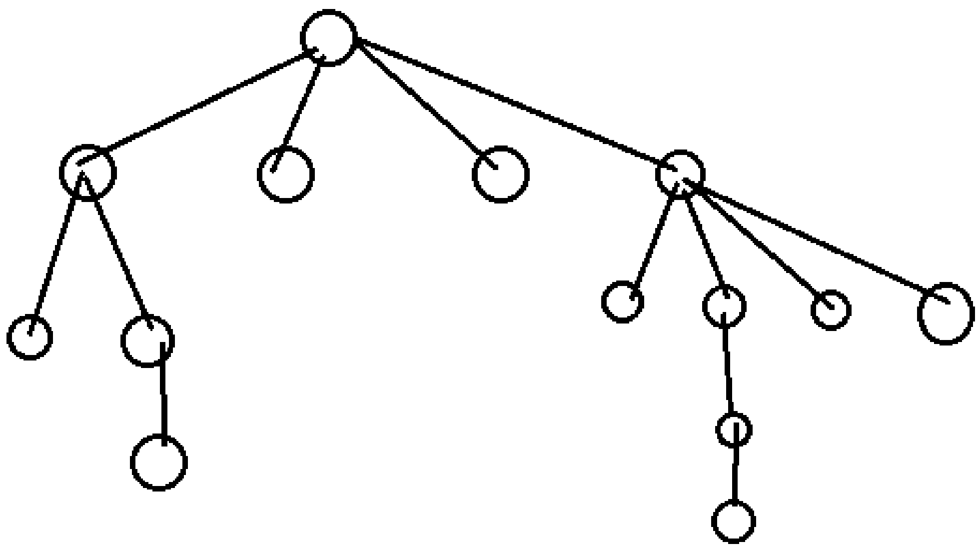
- ①DFS 一遍求出欧拉序、每个点深度。
- ②对于一个固定的序列欧拉序，多次询问区间内最小的数及其位置。
- 用 ST 表/RMQ：
- 和之前倍增的思路一样，我们记 $Min[i][j]$ 表示从 i 开始，长度为 2^j 区间内最小的数是多少；为了求位置，再记个 $MinPos[i][j]$。
- 就是预处理了每个位置开始长度为2的次幂的所有区间内的最小值。

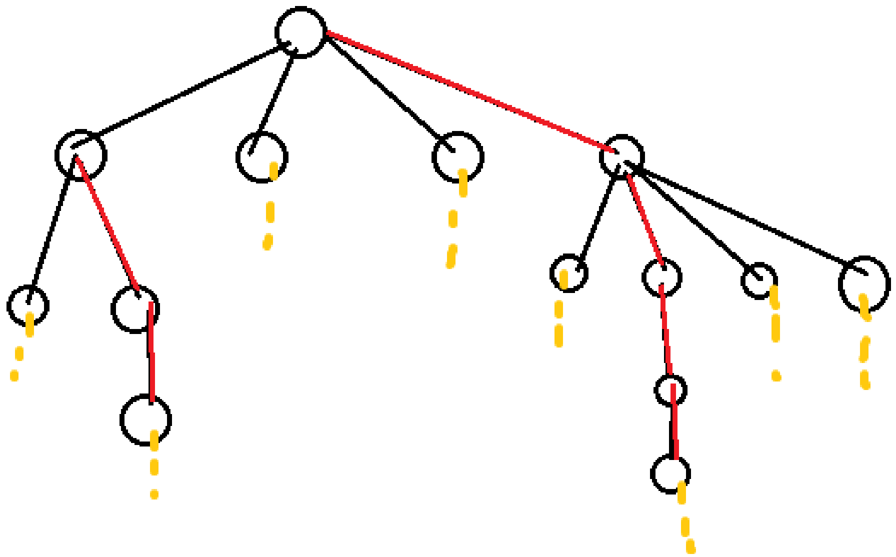
（二）图论算法
④最近公共祖先：*RMQ*

- 接下来来要计算 $[x,y]$ 内的最小值。
- 可以利用预处理的信息取两个长度均为 2 的次幂的区间使得其能覆盖 $[x,y]$。
- 其实就是取 $t=\lfloor\log y-x+1\rfloor, l=2^t$。
- $\min[x,y]=\min\{\min[x,x+l-1],\min[y-l+1,y]\}$。
- 由这两个区间中的最小值和最小值位置来得到答案。
- 注意！单组询问的时间复杂度是 $O(1)$ 的。
- $Min[i][j]=min\{Min[i][j-1],Min[i+2^{j-1}][j-1]\}$

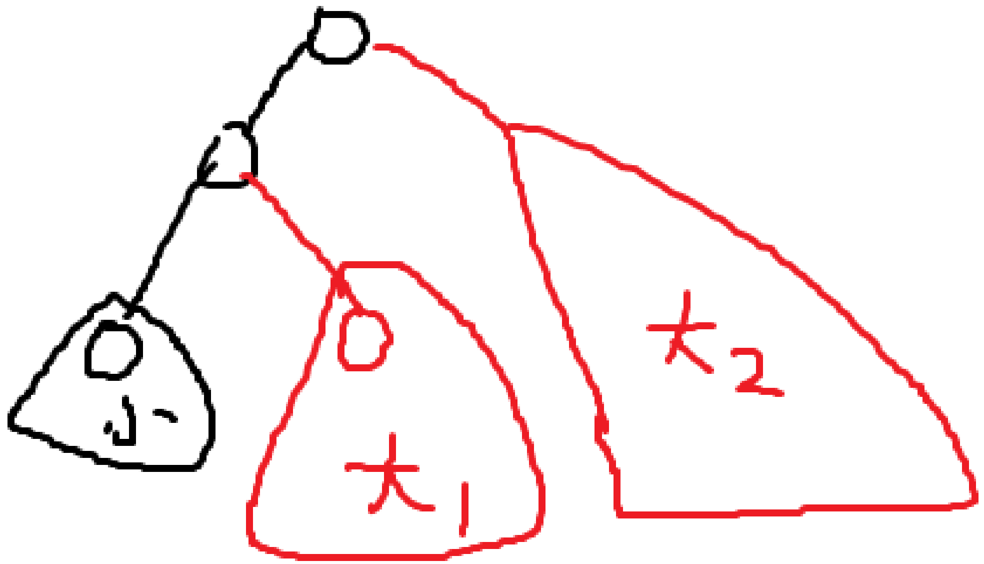
（二）图论算法
④最近公共祖先：树链剖分

- 我们还有一种时间复杂度、常数均非常优秀的做法。
- 树链剖分：选出每个点最“重”的儿子，就是子树大小最大的那个儿子，并将这条边标为“重边”，重边连成“重链”。





（二）图论算法
④最近公共祖先：树链剖分



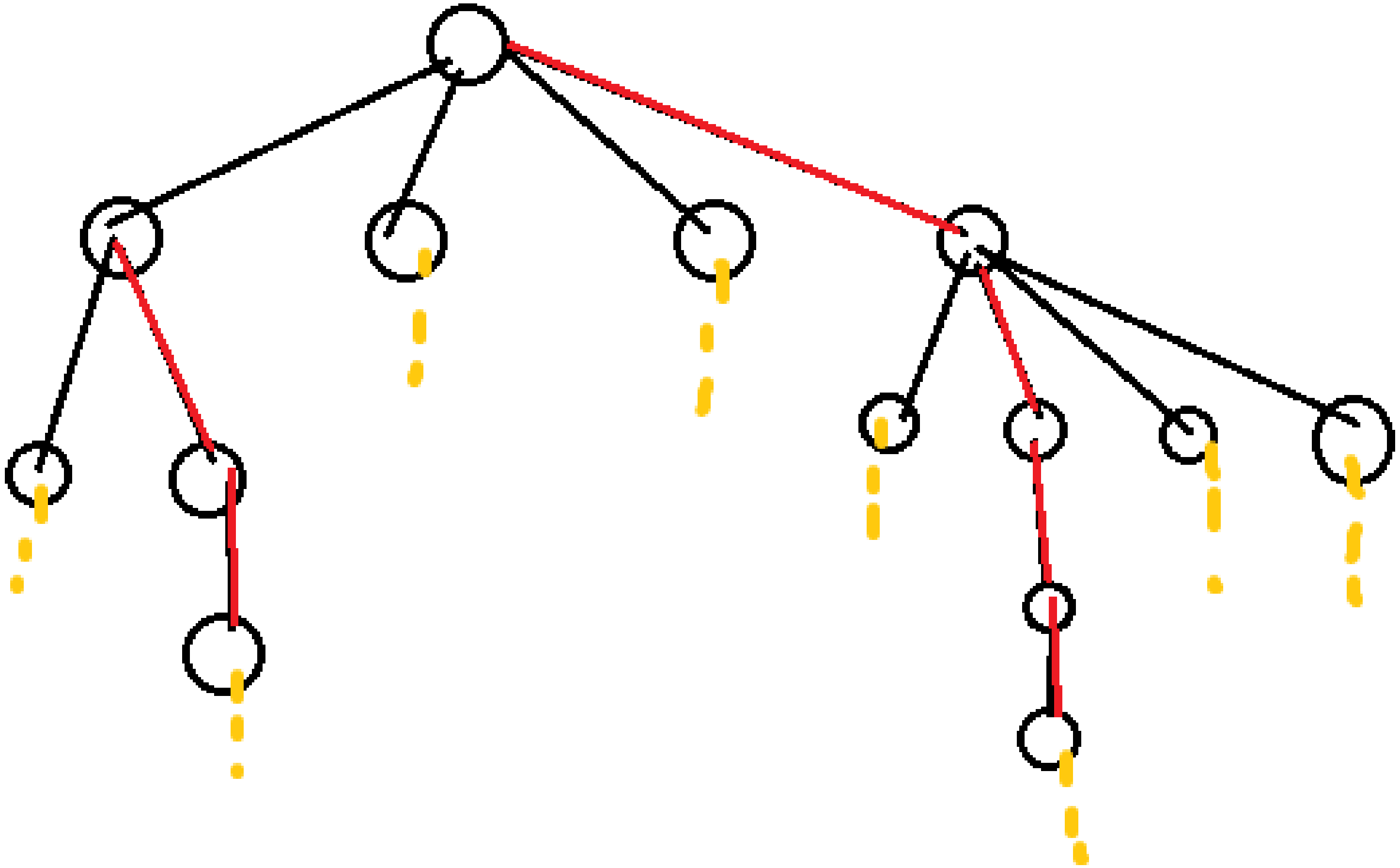
- 可以看右侧的图。
- 每个点往根跳，所经过的轻边数（连续的重链数）不会超过 $O(logn)$ 条。
- 证明：每经过一条轻边，树的点数至少翻倍。
- 预处理：
- 第一次 DFS 求出每个点的父亲、子树大小、深度、重儿子。
- 第二次 DFS 求出每个点的重链链顶，以方便之后用。
- 我们设 x 的重链链顶为 $Top[x]$。

（二）图论算法
④最近公共祖先：树链剖分

- 求 $LCA(u,v)$：注意一个点只有一个重儿子，所以 u 和 v 往祖先的两条路径，至少一条是从 LCA 出来的轻边。
- 每次看看 $Top[u]$ 和 $Top[v]$ 哪个深度更大，如果是 u，就把 u 跳到 $Fa\left[Top[u]\right]$。
- 注意不是比较 u 和 v 哪个深度更大，而是比较其重链链顶。
- 直到两个点在同一条重链上，LCA 就是此时深度比较小的点。
- 该算法优秀之处在于：只要一次能跳一条重链，复杂度就是等于重链条数 $O(logn)$。

（二）图论算法

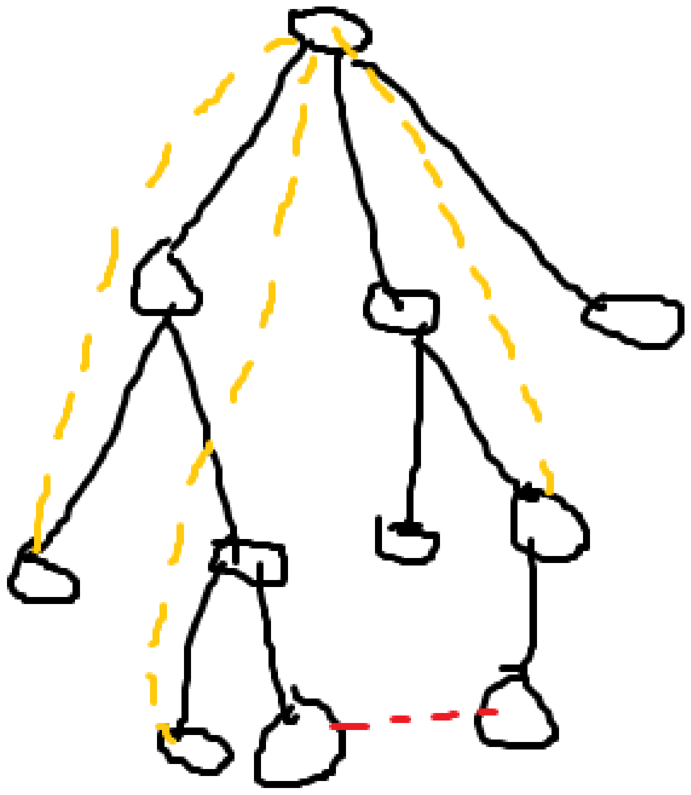
④最近公共祖先：树链剖分



（二）图论算法
⑤缩强联通分量：问题

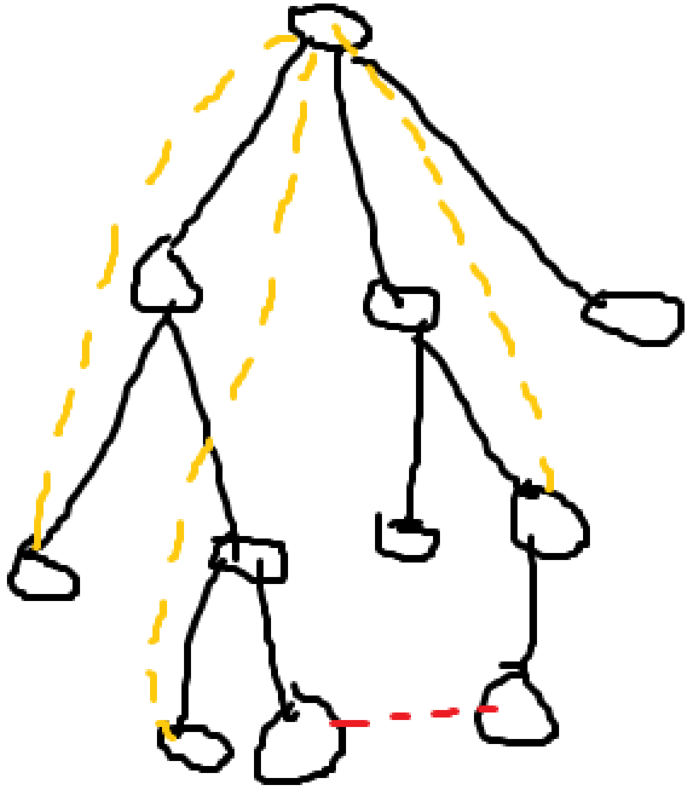
- 无向连通图的割点：删掉这个点后图不连通。
- 无向连通图的割边（桥）：删掉这条边后图不连通。
- 现在给定一张无向连通图，要求出其割点、割边。
- 设点数为 n，边数为 m。
- $n, m \leq 10^5$。

（二）图论算法
⑤缩强联通分量：*Tarjan*



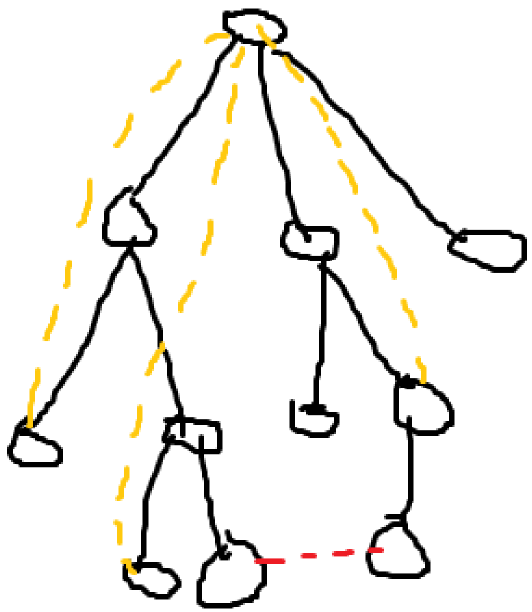
- *Tarjan*：只要写一个 *Dfs* 就可以了。
- *Dfs* 一张无向图，形成一棵生成树。
- 性质：这棵 *dfs* 树上只有返祖边
- 也就是非 *dfs* 树的边连向祖先。
- 右图中标为黄色。
- 没有横叉边。
- 也就是连向另一端的边，标为红色。

（二）图论算法
⑤缩强联通分量：$Tarjan$



- 如何判断一个点是不是割点?
- 根：在 dfs 树上要有至少两个儿子。
- 其余节点：
- 注意到每个节点往祖先有一个连通块。
- 如果一个点有一棵子树无法通过返祖边到达往祖先的连通块，就会导致这个点成为割点。
- 因此一个点成为割点的条件是：其所有儿子中，至少有一个儿子无法通过返祖边到达这个点的上部。

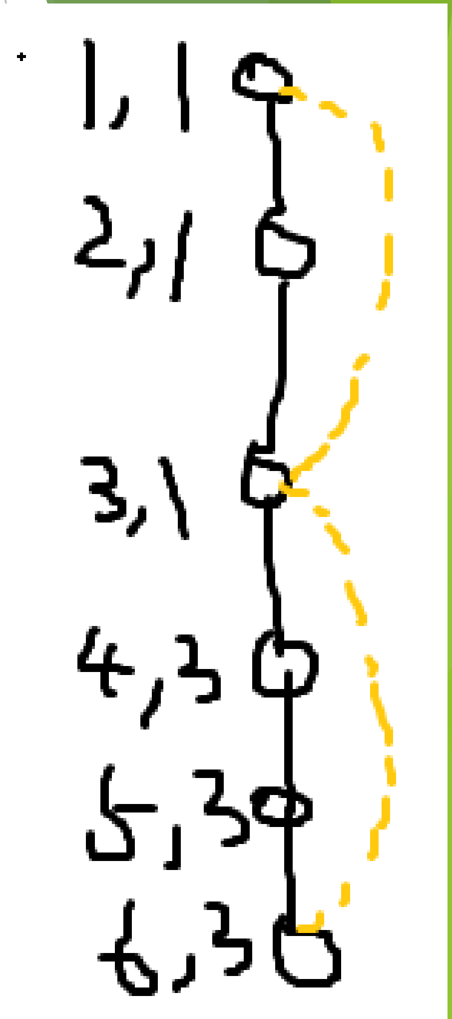
（二）图论算法
⑤缩强联通分量：*Tarjan*



- 对于一个点 x 记录：
- $Dfn[x]$，表示 x 是第几个遍历到的点。
- 我们称这个为 x 的 dfs 序。
- $Low[x]$，表示 x 沿子树内的边或者返祖边，能走到的所有点的 dfs 序中，最小的 dfs 序是多少。
- 通过判断一个点的所有儿子的 Low 来判定割点。
- 这个定义实际上是有点问题的。
- 黄色的边有可能会连续出现几条。我们接下来考虑这种情况下怎么走。

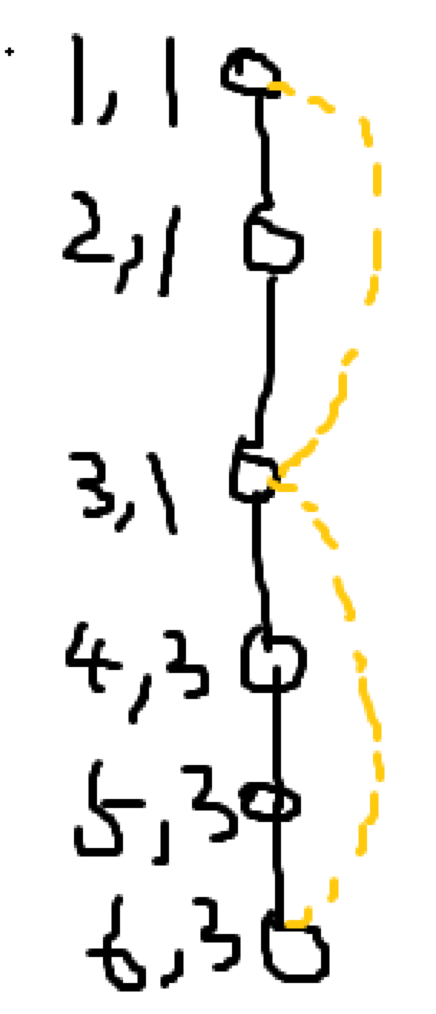
（二）图论算法
⑤缩强联通分量：*Tarjan*

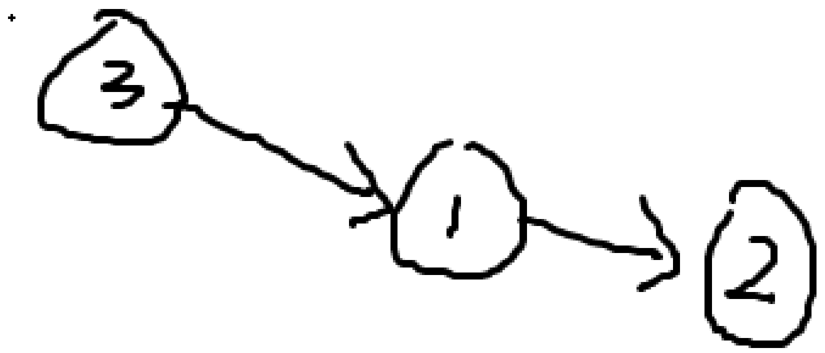
- $Dfn = 3$ 的点，它是个割点。
- 为了能求出 3 这个割点，$Low[4/5/6]$ 应该等于 3 而不是 1。
- 完善我们对 Low 的定义：Low 是不走 Dfs 树上的边，能走到的点中 dfs 序最小的点的编号。
- 这个走的过程中，不允许 4/5/6 三个点沿着黄色的边往上跳两次。
- 求一个点的 Low 时，最多只能往上走一条非树边。



（二）图论算法
⑤缩强联通分量：*Tarjan*

- 处理一个点 x 的时候，枚举出边。
- ①如果出边是来的时候的边，*continue*
- 注意可能有重边，所以不能判“来时的点”。
- ②如果出边连向一个未访问过的点 to，Dfs 这个点将其考虑进当前的生成树内，递归完成这个点后，$Low[x] = \min\{Low[x], Low[to]\}$。
- ③如果出边连向一个已经访问过的点 to，并且 to 在当前 dfs 栈内，不再 Dfs 这个点，$Low[x] = \min\{Low[x], Dfn[to]\}$。



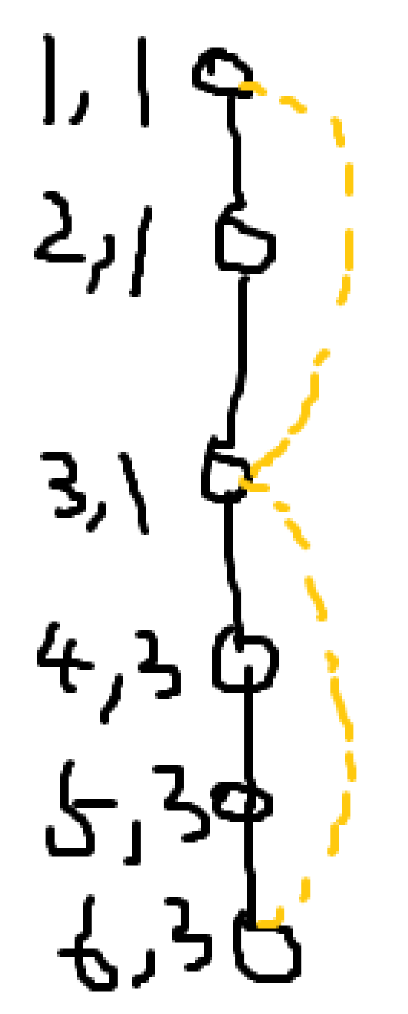


（二）图论算法
⑤缩强联通分量：*Tarjan*

- 如果有多个连通块，我们枚举每个点开始 dfs。
- 这个方法适用于有向图，不过要注意有向图会出现右上图的情况。
- 我们会从每个点开始遍历 dfs，已经遍历过的点不再遍历。
- 不过如果先枚举到 1，再枚举到 3，那么从 3 dfs 的时候，$Low[3]$ 不能用 $dfn[1]$ 来贡献答案。
- 也就是说 $Low[x] = \min\{Low[x], Dfn[to]\}$ 仅适用于 to 在当前的 dfs 栈内的情况。

（二）图论算法
⑤缩强联通分量：$Tarjan$

- 如何判断一条边是不是割边?
- 返祖边：一定不是割边，因为把这条边割了不会影响连通性。
- 对于一条树边，假设其从 x 连向 to。
- 我们看这条边从 to 往下有没有边能跨到x甚至更高的点，如果能则不是割边。
- 注意如果能跨到从 to 往别的地方沿返祖边走能走到 x，那么这条边也不是割边，如右图 $3-4$。
- 也就是说比较 $Low[to]$ 和 $Dfn[x]$。

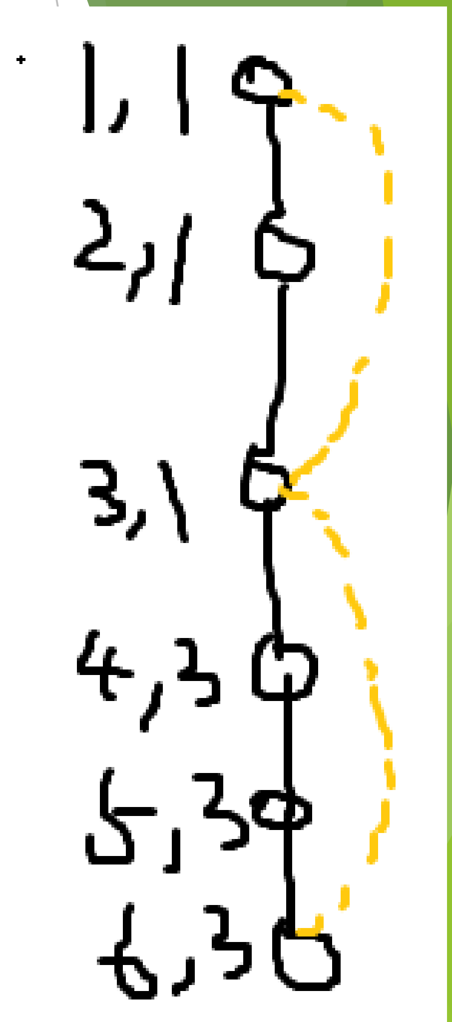


（二）图论算法
⑤缩强联通分量：边双

- 无向图的强联通分量：边双联通分量/点双联通分量。
- 边双联通分量：是原图点集的一个子集，这个点集里任意两个点之间至少存在两条路径相连，这两条路径边不重复。换句话说有至少一个环串起了这个点集。
- 点双联通分量：是原图点集的一个子集，这个点集里任意两个点之间至少存在两条路径相连，这两条路径点不重复。
- 缩强联通分量就是将无向图的强联通分量（点/边）求出来，以求出这张图的割点、割边。

（二）图论算法
⑤缩强联通分量：边双

- 问题：求出每个双联通分量。
- 为了求出每个边双联通分量，我们开了一个栈记录当前正在 Dfs 的点。
- 每次访问到一个新节点，将其加入栈中。
- 如果一个点 x 的 $Dfn[x]=Low[x]$，说明这个点是一个边双联通分量的至高点。
- 弹栈直到把 x 弹出，弹出来的这些点就是 x 所在的边双联通分量的点。



（三）应用和例题

- 刚才的课讲得可能比较仓促，大家或许会担心自己掌握得是否到位。
- 下面列出了一些模板题供大家参考：
- ①拓扑排序：http://acm.hdu.edu.cn/showproblem.php?pid=1285
- ②最短路：http://acm.hdu.edu.cn/showproblem.php?pid=2544
- ③最小生成树：http://acm.hdu.edu.cn/showproblem.php?pid=1863
- ④最近公共祖先：http://acm.hdu.edu.cn/showproblem.php?pid=2586
- ⑤缩强联通分量：http://acm.hdu.edu.cn/showproblem.php?pid=1269
- 大家可以到网络上搜索相关题目的题解。