图论入门

TA

CST, Tsinghua University

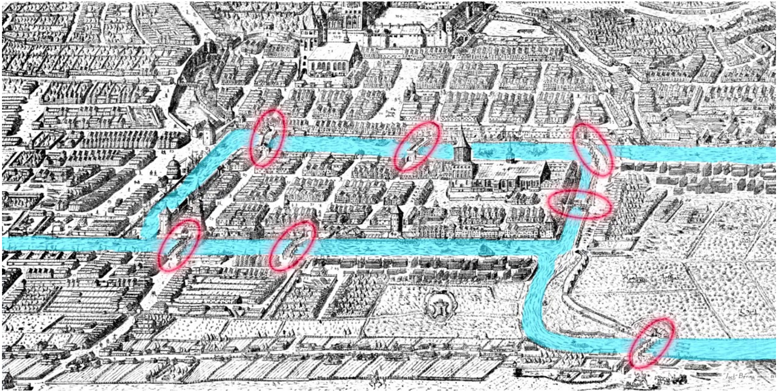
Oct. 2018

Contents

Contents

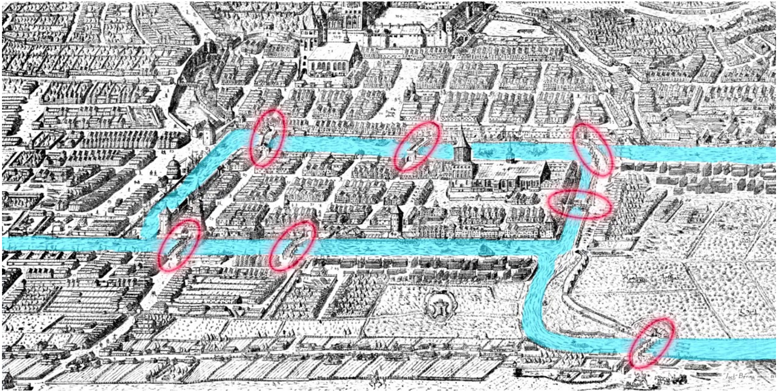
七桥问题

Seven Bridges of Königsberg.



七桥问题

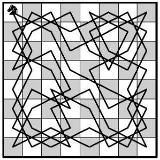
Seven Bridges of Königsberg.



It's resolved to be negative by Leonhard Euler in 1736.

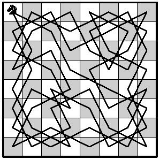
国际象棋中的马

Knight's tour



国际象棋中的马

Knight's tour



Schwenk proved that for any $m \times n$ board with $m \leq n$, a closed knight's tour is always possible unless one or more of these three conditions are met:

- m and n are both odd
- $m = 1, 2,$ or 4
- $m = 3$ and $n = 4, 6,$ or 8.

点与边

节（结，顶）点 (node)：图的基本构成元素，一般以小写字母表示 (u, v, ...).

标号：大部分图论算法是建立在有标号节点之上的.

点与边

节（结，顶）点 (node)：图的基本构成元素，一般以小写字母表示 (u, v, ...).
标号：大部分图论算法是建立在有标号节点之上的.

边 (edge)：节点对 (u, v). 可分为有向边（有序节点对）和无向边（无序节点对），对于有向边，u 称作始点，v 称作终点.
重边：相同的边
自环：(u, u)

图

图 (graph): $G=(V,E)$. 其中 V 是点集, E 是可重边集.

各种图:

- 无向图, 有向图, 混合图.
- 简单图 (无重边和自环)
- 完全图 (K_n), 竞赛图

图

图 (graph): $G = (V, E)$. 其中 V 是点集, E 是可重边集.

各种图:

- 无向图, 有向图, 混合图.
- 简单图 (无重边和自环)
- 完全图 (K_n), 竞赛图

权

- 边权: $w : E \to \mathbb{R}/\mathbb{N}$
- 点权: $w : V \to \mathbb{R}/\mathbb{N}$

图

图 (graph): $G = (V, E)$. 其中 V 是点集, E 是可重边集.

各种图:

- 无向图, 有向图, 混合图.
- 简单图 (无重边和自环)
- 完全图 (K_n), 竞赛图

权

- 边权: $w : E \to \mathbb{R}/\mathbb{N}$
- 点权: $w : V \to \mathbb{R}/\mathbb{N}$

Q: 无向简单图最多有多少条边? 有向简单图最多有多少条边?

边与点的相遇: 度

度: $d : V \to \mathbb{N}$, 与某节点相邻的边数.
正 (出) 度: 以某节点为始点的边数.
负 (入) 度: 以某节点为终点的边数.

边与点的相遇：度

度：$d : V \to \mathbb{N}$，与某节点相邻的边数.
正（出）度：以某节点为始点的边数.
负（入）度：以某节点为终点的边数.

度的性质:

- (简单图)$\sum_{v \in V} d_v = 2|E|$
- 度为奇数的节点必为偶数个.
- $\sum_{v \in V} d_v^+ = \sum_{v \in V} d_v^-$
- (竞赛图)$\sum_{v \in V}(d^+(v_i))^2 = \sum_{v \in V}(d^-(v_i))^2$

图的存储

名称	初始复杂度	构造复杂度
邻接矩阵	$O(V^2)$	$O(E)$
前向星	$O(V)$	$O(V+E\log E)$ / $O(V+E)$
邻接表	$O(V)$	$O(E)$
十字链表	$O(V)$	$O(E)$

图的存储

名称	初始复杂度	构造复杂度
邻接矩阵	$O(V^2)$	$O(E)$
前向星	$O(V)$	$O(V + E\log E)$ / $O(V + E)$
邻接表	$O(V)$	$O(E)$
十字链表	$O(V)$	$O(E)$

查询指定边信息?

- 邻接矩阵: $O(1)$
- 前向星: $O(E)/O(\log E)$
- 邻接表/十字链表: $O(E)/O(\log E)$

Contents

基本概念: 路径

有向路径: $P = (e_1, ..., e_n)$, 其中 e_i 的终点是 e_{i+1} 的始点.
初等: 路径经过的边两两不同.
简单: 路径经过的点两两不同 (除路径的始点与终点外).
环: 路径的始点与终点相同.
无向图上的路径: 存在一种每一条边分配方向的方案使得其成为一条有向图的路径.

基本概念: 自由树

自由树: 连通的无环无向图.

森林: 树的集合.

基本概念: 自由树

自由树: 连通的无环无向图.
森林: 树的集合.

对于一棵自由树而言, 以下 6 条等价:

1. G 是一棵自由树.
2. G 中任何两顶点由唯一简单路径相连.
3. G 是连通的, 但是从图中移除任意一条边得到的图均不连通.
4. G 是连通的, 且 $|E| = |V| - 1$
5. G 是无环的, 且 $|E| = |V| - 1$
6. G 是无环的, 但若向 E 中添加任意一条边, 均会造成该图包含恰有一个环.

基本概念: 有根树与二叉树

有根树: 自由树 + 根.
节点之间的关系: 父亲与儿子, (真) 祖先与 (真) 后代, 叶节点.
子树: u 的所有后代的导出子图.
定义在点集上的函数: 度, 深度, 大小.

$$\sum depth(i) + |V| = \sum size(i)$$

基本概念: 有根树与二叉树

有根树: 自由树 + 根.
节点之间的关系: 父亲与儿子, (真) 祖先与 (真) 后代, 叶节点.
子树: u 的所有后代的导出子图.
定义在点集上的函数: 度, 深度, 大小.

$$\sum depth(i) + |V| = \sum size(i)$$

二叉树: 左子树, 右子树.

基本概念: 有根树与二叉树

有根树: 自由树 + 根.
节点之间的关系: 父亲与儿子, (真) 祖先与 (真) 后代, 叶节点.
子树: u 的所有后代的导出子图.
定义在点集上的函数: 度, 深度, 大小.

$$\sum depth(i) + |V| = \sum size(i)$$

二叉树: 左子树, 右子树.

一些性质

- 任何非空二叉树中, 度为 2 的节点数比叶节点数少 1.
- n 个节点的二叉树高度至少为 $\lfloor \log n \rfloor$
- Kraft 不等式: 将二叉树 T 中每个深度为 d 的叶节点赋予权值 $w(x) = 2^{-d}$, 则 $\sum w(x) \leq 1$.

树的存储

可以保存的信息:

- 边信息
- 父亲
- 儿子

大部分题会将树作为一个无向图来读入，也有的题会告诉你某个点的父亲是谁，保存哪些信息取决于你的需求.

BFS

BFS(breadth first search): 求图中两点间最短路径?(路径长度被定义为其拥有的边数)
时间复杂度: $O(V+E)$
BFS 生成树

DFS

DFS(depth first search): DFS 生成树.
边的分类:

- 树边
- 后向边
- 前向边
- 横向边

DFS

DFS(depth first search): DFS 生成树.
边的分类:

- 树边
- 后向边
- 前向边
- 横向边

DFS 序: DFS 时的时间戳, 具体值无关紧要, 重要的是相对大小. 入栈序 (v_l), 出栈序 (v_r).

DFS

DFS(depth first search): DFS 生成树.
边的分类:

- 树边
- 后向边
- 前向边
- 横向边

DFS 序: DFS 时的时间戳, 具体值无关紧要, 重要的是相对大小. 入栈序 (v_l), 出栈序 (v_r).
性质:

- 有向图 G 不存在指向 dfs 序更大的点的横向边, 无向图 G 不存在横向边.
- 括号化定理 ($[v_l, v_r]$): 子树与 DFS 序区间的对应性.

DFS

DFS(depth first search): DFS 生成树.
边的分类:

- 树边
- 后向边
- 前向边
- 横向边

DFS 序: DFS 时的时间戳, 具体值无关紧要, 重要的是相对大小. 入栈序 (v_l), 出栈序 (v_r).
性质:

- 有向图 G 不存在指向 dfs 序更大的点的横向边, 无向图 G 不存在横向边.
- 括号化定理 ($[v_l, v_r]$): 子树与 DFS 序区间的对应性.

Q: 如何求出一个环?
Q: 对一棵树支持单点修改, 求前缀 (某一节点的所有祖先) 和?

基本概念: 连通

连通: 若 (不考虑方向的) 图 G 的任意两点间均存在路径, 则称图 G 连通.
子图: 若 $G' = (V', E')$ 满足 $V' \subset V$, $E' \subset E$, 则称 G' 是 G 的子图.
导出子图: 若 E' 包含了 V' 在 G 中的所有关联的边, 则称 G' 是 G 关于 V' 的导出子图.
(无向图的) 极大连通子图 (连通支): 若 G' 满足不存在 H, 使得 G' 是 H 的子图, H 是 G 的子图, 则称 G' 是 G 的极大连通子图.

基本概念: 连通

连通: 若 (不考虑方向的) 图 G 的任意两点间均存在路径, 则称图 G 连通.
子图: 若 $G' = (V', E')$ 满足 $V' \subset V$, $E' \subset E$, 则称 G' 是 G 的子图.
导出子图: 若 E' 包含了 V' 在 G 中的所有关联的边, 则称 G' 是 G 关于 V' 的导出子图.
(无向图的) 极大连通子图 (连通支): 若 G' 满足不存在 H, 使得 G' 是 H 的子图, H 是 G 的子图, 则称 G' 是 G 的极大连通子图.

Q: 对于无向图而言, G 连通最少需要多少条边? G 不连通最多可以有多少条边?

基本概念: 连通

连通: 若 (不考虑方向的) 图 G 的任意两点间均存在路径, 则称图 G 连通.
子图: 若 $G' = (V', E')$ 满足 $V' \subset V$, $E' \subset E$, 则称 G' 是 G 的子图.
导出子图: 若 E' 包含了 V' 在 G 中的所有关联的边, 则称 G' 是 G 关于 V' 的导出子图.
(无向图的) 极大连通子图 (连通支): 若 G' 满足不存在 H, 使得 G' 是 H 的子图, H 是 G 的子图, 则称 G' 是 G 的极大连通子图.

Q: 对于无向图而言, G 连通最少需要多少条边? G 不连通最多可以有多少条边?

Q: 对于无向图而言, 如何判定两点间是否可达? 如何判定任意两点间是否可达? 如何求一个图的连通支个数?
Q: 对于有向图而言, 如何判定两点间是否可达? 如何判定任意两点间是否可达?

拓扑排序

在 DAG(Directed Acyclic Graph) 中求一个排列 P, 使得对于每个点 v 而言, 若 (u, v) 存在, 则 u 在 P 中在 v 之前.

拓扑排序

在 DAG(Directed Acyclic Graph) 中求一个排列 P, 使得对于每个点 v 而言, 若 (u, v) 存在, 则 u 在 P 中在 v 之前.

Kahn 算法: 不断寻找入度为 0 的点, 删掉这个点, 修改其他点的入度.
基于 DFS 的算法: 按 v_r 逆序.

拓扑排序

在 DAG(Directed Acyclic Graph) 中求一个排列 P, 使得对于每个点 v 而言, 若 (u, v) 存在, 则 u 在 P 中在 v 之前.

Kahn 算法: 不断寻找入度为 0 的点, 删掉这个点, 修改其他点的入度.
基于 DFS 的算法: 按 v_r 逆序.

Q: 在有边权的 DAG 中求从 u 出发的最短/最长路径.
Q: 计算 DAG 中从每一点出发可以到达的点的数量/可以到达每一点的点的数量.

n 个火车站，每个车站有一个等级．现有 m 辆火车．已知其停靠的车站信息，若一辆火车停靠了等级为 x 的车站，则在火车的始发站和终点站之间所有等级不小于 x 的车站都必须停靠．请问火车站至少被分为了多少个等级？

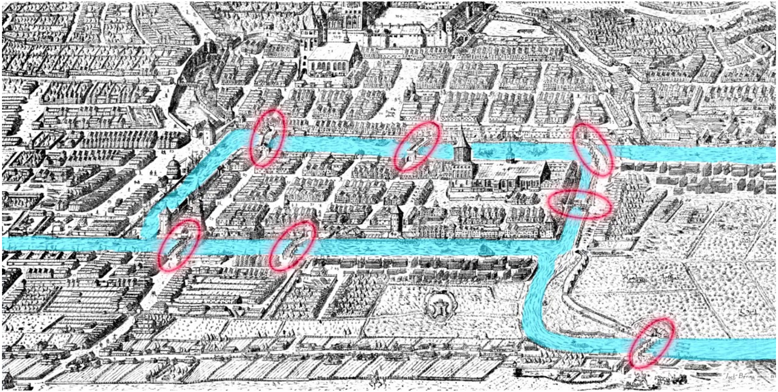
$n, m \leq 1000$

NOIP2013

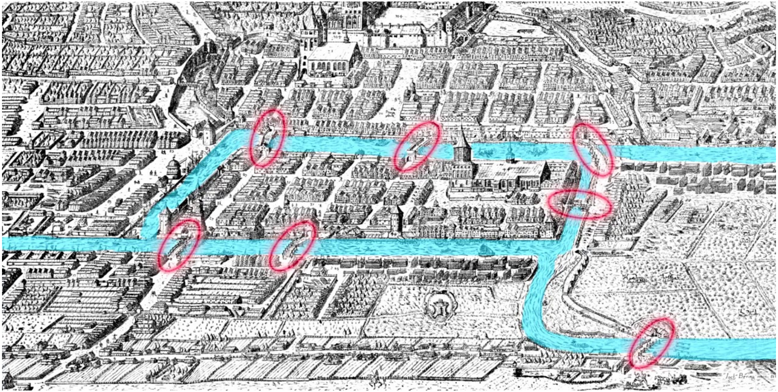
n 个火车站，每个车站有一个等级．现有 m 辆火车．已知其停靠的车站信息，若一辆火车停靠了等级为 x 的车站，则在火车的始发站和终点站之间所有等级不小于 x 的车站都必须停靠．请问火车站至少被分为了多少个等级？
$n, m \leq 1000$

在火车站之间建边，答案是最长路．$O(mn^2)$
为每辆车建两个虚点：表示其经过的车站和没经过的车站．$O(mn)$
火车站的等级不小于火车的等级，在火车之间建边．$O(m^2)$

再谈七桥问题



再谈七桥问题



欧拉路径：经过所有边的初等路径.
欧拉回路：经过所有边的初等环.

欧拉路径 (环)

连通无向图:

- 存在欧拉回路 等价于度为奇数的点有 0 个.
- 存在欧拉路径 等价于度为奇数的点有 0 或 2 个.

连通有向图 (边忽略方向后得到的无向图连通):

- 存在欧拉回路 等价于所有点出入度相等.
- 存在欧拉路径 等价于所有点出入度相等 或恰有一个点入度比出度多 1 和恰有 1 个点出度比入度多 1.

如何求出一条欧拉路径?

欧拉路径 (环)

连通无向图:

- 存在欧拉回路 等价于度为奇数的点有 0 个.
- 存在欧拉路径 等价于度为奇数的点有 0 或 2 个.

连通有向图 (边忽略方向后得到的无向图连通):

- 存在欧拉回路 等价于所有点出入度相等.
- 存在欧拉路径 等价于所有点出入度相等 或恰有一个点入度比出度多 1 和恰有 1 个点出度比入度多 1.

如何求出一条欧拉路径?

Prop. 若 G 中有 k 个度为奇数的点, G 可以划分为 $\frac{k}{2}$ 条初等道路.

Contents

最短路

最短路：有向边权图或无向非负权图中两点间的所有路径中最短的一条.(若不连通，定义为 ∞)

Review：无边权？DAG？

变体

- 单源（目的地）最短路径，单节点对最短路径
- 所有节点对最短路径

最短路

最短路：有向边权图或无向非负权图中两点间的所有路径中最短的一条.(若不连通，定义为 ∞)
Review：无边权？DAG？
变体

- 单源（目的地）最短路径，单节点对最短路径
- 所有节点对最短路径

负权边：负权图的最短路径中可能有无穷多条边.
环路：最短路径可能包含回路嘛？
松弛操作：$d(x,v) = \min(d(x,v), d(x,u) + w(u,v))$

最短路的性质

三角不等式: $d(x, v) \leq d(x, u) + w(u, v)$
最短路径图: $\{(u, v) \in E | w(u, v) = d(v) - d(u)\}$
最短路径树: 存在性由最短路径图可知.

Dijkstra

算法流程: 维护已经找到最短路的点集 S, 不断地寻找距离 S 最近的点将其加入 S.
仅适用于边权非负的图.
时间复杂度: $O((V+E)\log(V+E))$ / $O((V+E)\log V)$ / $O(V\log V+E)$

Bellman-Ford


```
for(int i=1;i<=n;++i)
    for(int j=m;j;--j)
        dis[v[j]]=min(dis[v[j]],
            dis[u[j]]+w[j]);
```

内层第 i 次循环可以保证求得 BFS 最短路径树前 i 层的最短路径.

时间复杂度: $O(VE)$

Bellman-Ford

```
for(int i=1;i<=n;++i)
    for(int j=m;j;--j)
        dis[v[j]]=min(dis[v[j]],
            dis[u[j]]+w[j]);
```

内层第 i 次循环可以保证求得 BFS 最短路径树前 i 层的最短路径.
时间复杂度: $O(VE)$

SPFA: deprecated
Hack: 网格图, 次短路条数很多的图.

Bellman-Ford


```
for(int i=1;i<=n;++i)
    for(int j=m;j;--j)
        dis[v[j]]=min(dis[v[j]],
            dis[u[j]]+w[j]);
```

内层第 i 次循环可以保证求得 BFS 最短路径树前 i 层的最短路径.
时间复杂度: $O(VE)$

SPFA: deprecated
Hack: 网格图, 次短路条数很多的图.
Q: 如何找到负权回路?(在 $O(VE)$ 的时间内)
Q: 求出距离每个点最近的点的距离?(在 $O(VE)$ 的时间内)

Floyd

```
for(int k=n;k;--k)
    for(int i=n;i;--i)
        for(int j=n;j;--j)
            dis[i][j]=min(dis[i][j],
                dis[i][k]+dis[k][j]);
```

可以看作 DP.
同时求出每对点间的最短路.
时间复杂度: $O(V^3)$

Floyd


```
for(int k=n;k;--k)
    for(int i=n;i;--i)
        for(int j=n;j;--j)
            dis[i][j]=min(dis[i][j],
                dis[i][k]+dis[k][j]);
```

可以看作 DP.
同时求出每对点间的最短路.
时间复杂度: $O(V^3)$

Q: 求有向图的最小环? 求无向图的最小环?

Johnson

重新赋予权重: $\hat{w}(u, v) = w(u, v) + d(0, u) - d(0, v)$
一遍 Bellman-Ford+$|V|$ 遍 Dijkstra.
时间复杂度: $O(V^2 \log V + VE)$

例题: Vijos1754

n 个城市间以 m 条有向道路连接, 小 T 从 1 号城市出发, 将要去往 n 号城市.
小 T 观察到一款商品 Z 在不同的城市的价格可能不尽相同, 小 T 想要在旅行中的某一个城市购买一件商品 Z, 在另一个城市卖出. 因为旅途劳顿, 这种买卖小 T 只打算做一次.
请问小 T 能够获得的最大收益是多少?
$n, m \leq 10^5$

例题：Vijos1754

n 个城市间以 m 条有向道路连接，小 T 从 1 号城市出发，将要去往 n 号城市.
小 T 观察到一款商品 Z 在不同的城市的价格可能不尽相同，小 T 想要在旅行中的某一个城市购买一件商品 Z，在另一个城市卖出. 因为旅途劳顿，这种买卖小 T 只打算做一次.
请问小 T 能够获得的最大收益是多少？
$n, m \leq 10^5$

将求最短路改为求最小/最大值.

例题：CodeVS1183

一个无向图中，对于每条道路 i 有长度 l_i 和时间 t_i，请问从 1 号点到 n 号点速度最大是多少？

$n, m \leq 10^3$

例题：CodeVS1183

一个无向图中，对于每条道路 i 有长度 l_i 和时间 t_i，请问从 1 号点到 n 号点速度最大是多少？
$n, m \leq 10^3$

分数规划：二分答案 v.

$$\exists \frac{\sum l_i}{\sum t_i} \geq v$$
$$\Leftrightarrow \exists \sum (l_i - t_i v) \geq 0$$

问题转化为求最长路.
时间复杂度：$O(nm \log A)$(A 为所需精度要求)

应用：差分约束系统

求解简单线性不等式组:

$$\begin{cases} x_{i-1} - x_{j-1} \leq k_1 \\ \cdots \\ x_{i_m} - x_{j_m} \leq k_m \end{cases}$$

应用: 差分约束系统

求解简单线性不等式组:

$$\begin{cases} x_{i-1} - x_{j-1} \le k_1 \\ \cdots \\ x_{i_m} - x_{j_m} \le k_m \end{cases}$$

若 $x = (x_1, \cdots, x_n)$ 是差分约束系统的一个解, 则 $x + d$ 也是差分约束系统的一个解.
约束图: 新建 0 点, $w(0, i)(1 \le i \le n)$ 为 0; $w(j_l, i_l)$ 为 k_l.

Contents

强连通分量

强连通分量 (SCC, Strongly Connected Component): 有向图的极大点集 $C \subset V$, C 中任意两点可达.

强连通分量 (SCC, Strongly Connected Component): 有向图的极大点集 $C \subset V$, C 中任意两点可达.

Kosaraju 算法: 以 v_r 逆序对 $G^T = (V, E^T)$ 作 DFS, 则每棵 DFS 树会是一个 SCC.
Tarjan 算法: 使用一个栈来维护尚未被分到 SCC 中的节点, 依据一个节点 u 是否能走到其 DFS 树中的祖先 ($low(u)$), 来取出栈中节点分配成一个新的 SCC.
Gabow 算法: 使用两个栈来取代 low, S 表示尚未被分配的节点, P 表示尚未被分配 SCC 的节点.
时间复杂度均为 $O(V + E)$

强连通分量

强连通分量 (SCC, Strongly Connected Component): 有向图的极大点集 $C \subset V$, C 中任意两点可达.

Kosaraju 算法: 以 v_r 逆序对 $G^T = (V, E^T)$ 作 DFS, 则每棵 DFS 树会是一个 SCC.
Tarjan 算法: 使用一个栈来维护尚未被分到 SCC 中的节点, 依据一个节点 u 是否能走到其 DFS 树中的祖先 ($low(u)$), 来取出栈中节点分配成一个新的 SCC.
Gabow 算法: 使用两个栈来取代 low, S 表示尚未被分配的节点, P 表示尚未被分配 SCC 的节点.
时间复杂度均为 $O(V + E)$

将 SCC 缩点, 原图变为 DAG.

边双连通分量

桥 (割边): 删掉之后会导致图不连通的边.
边双连通分量 (BCC, biconnected component) 的等价定义

- 极大连通导出子图, 不含桥.
- 极大连通导出子图, 使得任意两点间均有两条边不相交路径.
- 极大连通导出子图, 使得任意两条边都存在于一个简单回路中.
- 极大连通导出子图, 任意三点 a, b, c, 存在边不相交的路径 $a \rightsquigarrow b$, $b \rightsquigarrow c$.
- 极大连通导出子图, 存在一种给边赋向的方案使得其为 SCC.

边双连通分量

桥 (割边): 删掉之后会导致图不连通的边.
边双连通分量 (BCC, biconnected component) 的等价定义

- 极大连通导出子图, 不含桥.
- 极大连通导出子图, 使得任意两点间均有两条边不相交路径.
- 极大连通导出子图, 使得任意两条边都存在于一个简单回路中.
- 极大连通导出子图, 任意三点 a, b, c, 存在边不相交的路径 $a \rightsquigarrow b$, $b \rightsquigarrow c$.
- 极大连通导出子图, 存在一种给边赋向的方案使得其为 SCC.

判断割边: $low(u) = u_l$

边双连通分量

桥 (割边): 删掉之后会导致图不连通的边.
边双连通分量 (BCC, biconnected component) 的等价定义

- 极大连通导出子图, 不含桥.
- 极大连通导出子图, 使得任意两点间均有两条边不相交路径.
- 极大连通导出子图, 使得任意两条边都存在于一个简单回路中.
- 极大连通导出子图, 任意三点 a, b, c, 存在边不相交的路径 $a \rightsquigarrow b$, $b \rightsquigarrow c$.
- 极大连通导出子图, 存在一种给边赋向的方案使得其为 SCC.

判断割边: $low(u) = u_l$

将 BCC 缩点, 原图变为树.

点双连通分量

割点：删掉之后会导致图不连通的点.
点双连通分量 (块, block) 的等价定义:

- 极大连通导出子图, 删除其中任意一个点不会使其不连通.
- 极大连通导出子图, 任意两个点之间有两条点不相交路径.
- 极大连通导出子图, 任意三点 a, b, c 之间, 两条点不相交路径.

注意到两个块可以有至多一个交点.

点双连通分量

割点：删掉之后会导致图不连通的点.

点双连通分量 (块, block) 的等价定义:

- 极大连通导出子图，删除其中任意一个点不会使其不连通.
- 极大连通导出子图，任意两个点之间有两条点不相交路径.
- 极大连通导出子图，任意三点 a, b, c 之间，两条点不相交路径.

注意到两个块可以有至多一个交点.

割点的等价条件:

- 根节点是割点当且仅当其 DFS 树至少有两个子树.
- 非根节点是割点当且仅当 $low(u) < u_l$.

点双连通分量

割点：删掉之后会导致图不连通的点.
点双连通分量 (块, block) 的等价定义:

- 极大连通导出子图，删除其中任意一个点不会使其不连通.
- 极大连通导出子图，任意两个点之间有两条点不相交路径.
- 极大连通导出子图，任意三点 a, b, c 之间，两条点不相交路径.

注意到两个块可以有至多一个交点.

割点的等价条件:

- 根节点是割点当且仅当其 DFS 树至少有两个子树.
- 非根节点是割点当且仅当 $low(u) < u_l$.

将 Block 缩成环，原图变为仙人掌.

BZOJ2959

维护一个带点权的树，支持三种操作:

- 加边
- 改变点权
- 求从 u 到 v 经过的所有 BCC 的权值和

$n, m \leq 10^5$

BZOJ2959

维护一个带点权的树，支持三种操作:

- 加边
- 改变点权
- 求从 u 到 v 经过的所有 BCC 的权值和

$n, m \leq 10^5$

使用并查集维护每一个 BCC.
将一个 BCC 的权值集中在其 LCA 处，则一条链经过的 BCC 必满足以下两者之一:

- BCC 的 LCA 在链上
- 是 LCA(u,v) 所属的 BCC

使用树状数组维护权值和.

Contents

基本概念：环与割

切割：$(S, V-S)$，对 V 的一个划分.
横跨：(u, v) 一个端点在 S，另一个端点在 $V-S$，则称 (u, v) 横跨割 $(S, V-S)$.
尊重：边集 A 中不存在横跨割 $(S, V-S)$ 的边，则称 $(S, V-S)$ 尊重 A.
轻量级边：(u, v) 是横跨 $(S, V-S)$ 的所有边中权重最小的.

MST 基本定理

对于任意 $A \subset E$, 存在 G 的最小生成树 T, $A \subset T$, 任意 $e = (u, v) \in A$, 存在尊重 A 的切割 $(S, V - S)$, (u, v) 是 $(S, V - S)$ 的一条轻量级边, 令 $B = A \cup (u, v)$, 则存在最小生成树 T', $B \subset T'$.(这条边被称作安全边)

MST 的求解算法

Prim 算法: 类似于 BFS/Dijkstra, 维护一个连通的 A, 不断寻找 $(A, S - A)$ 中的轻量级边.
时间复杂度: $O((V + E)\log V)$ / $O(V\log V + E)$

MST 的求解算法

Prim 算法: 类似于 BFS/Dijkstra, 维护一个连通的 A, 不断寻找 $(A, S - A)$ 中的轻量级边.
时间复杂度: $O((V + E)\log V)$ / $O(V\log V + E)$

Kruskal 算法: 从小到大枚举每条边 (u, v), 若 u 和 v 在 A 中尚不连通, 就将 (u, v) 加入 A.
时间复杂度: $O(V + E\log E)$

MST 的求解算法

Prim 算法: 类似于 BFS/Dijkstra, 维护一个连通的 A, 不断寻找 $(A, S - A)$ 中的轻量级边.
时间复杂度: $O((V + E)\log V)$ / $O(V\log V + E)$

Kruskal 算法: 从小到大枚举每条边 (u, v), 若 u 和 v 在 A 中尚不连通, 就将 (u, v) 加入 A.
时间复杂度: $O(V + E\log E)$

Borůvka 算法: 找到每个点的最小邻边, 形成了环套树森林, 将每一个环套树缩点.
时间复杂度: $O((V + E)\log V)$

MST 的求解算法

Prim 算法: 类似于 BFS/Dijkstra, 维护一个连通的 A, 不断寻找 $(A, S-A)$ 中的轻量级边.
时间复杂度: $O((V+E)\log V)$ / $O(V\log V+E)$

Kruskal 算法: 从小到大枚举每条边 (u,v), 若 u 和 v 在 A 中尚不连通, 就将 (u,v) 加入 A.
时间复杂度: $O(V+E\log E)$

Borůvka 算法: 找到每个点的最小邻边, 形成了环套树森林, 将每一个环套树缩点.
时间复杂度: $O((V+E)\log V)$

MST 的性质

- 若图 G 的一条边 (u, v) 在某棵最小生成树 T 中，则该条边是某个切割 $(S, V - S)$ 的轻量级边.
- 任意两棵最小生成树的有序边权序列相同.
- 对于非负边权图，MST 等价于权值和最小的边集，使得图连通.
- MST 是瓶颈生成树 (最大边权最小的生成树).

Contents

LCA 问题

LCA(Lowest Common Ancestor, 最近公共祖先): LCA(u,v) 定义为 u 和 v 的所有公共祖先中深度最大的那一个.

LCA 问题

LCA(Lowest Common Ancestor, 最近公共祖先): LCA(u,v) 定义为 u 和 v 的所有公共祖先中深度最大的那一个.

倍增: fa[u][j] 表示从 u 向上跳 2^j 步可以到达的点.
时间复杂度: $O(n \log n) - O(\log n)$

LCA 问题

LCA(Lowest Common Ancestor, 最近公共祖先): LCA(u,v) 定义为 u 和 v 的所有公共祖先中深度最大的那一个.

倍增: fa[u][j] 表示从 u 向上跳 2^j 步可以到达的点.
时间复杂度: $O(n \log n) - O(\log n)$

树链剖分: 重儿子是所有儿子中 size 最大的那个.
时间复杂度: $O(n) - O(\log n)$

LCA 问题

LCA(Lowest Common Ancestor, 最近公共祖先): LCA(u,v) 定义为 u 和 v 的所有公共祖先中深度最大的那一个.

倍增: fa[u][j] 表示从 u 向上跳 2^j 步可以到达的点.
时间复杂度: $O(n\log n) - O(\log n)$

树链剖分: 重儿子是所有儿子中 size 最大的那个.
时间复杂度: $O(n) - O(\log n)$

DFS 序 +ST: 利用 DFS 序的性质将问题转化为求区间最小值, 再使用 Sparse Table 求解.
时间复杂度: $O(n\log n) - O(1)$

LCA 问题

LCA(Lowest Common Ancestor, 最近公共祖先): LCA(u,v) 定义为 u 和 v 的所有公共祖先中深度最大的那一个.

倍增: fa[u][j] 表示从 u 向上跳 2^j 步可以到达的点.
时间复杂度: $O(n \log n) - O(\log n)$

树链剖分: 重儿子是所有儿子中 size 最大的那个.
时间复杂度: $O(n) - O(\log n)$

DFS 序 +ST: 利用 DFS 序的性质将问题转化为求区间最小值, 再使用 Sparse Table 求解.
时间复杂度: $O(n \log n) - O(1)$

Tarjan: 离线查询, 使用并查集维护当前栈中节点已经遍历过的子树.
时间复杂度: $O(n + m)$

与 RMQ 问题的联系

RMQ(Range Minimum/Maximum Query): 查询区间最值.

与 RMQ 问题的联系

RMQ(Range Minimum/Maximum Query): 查询区间最值.

静态: ST(Sparse Table)($O(n\log n) - O(1)$).
动态: 线段树 ($O(n) - O(\log n)$).

与 RMQ 问题的联系

RMQ(Range Minimum/Maximum Query): 查询区间最值.

静态: ST(Sparse Table)($O(n\log n) - O(1)$).
动态: 线段树 ($O(n) - O(\log n)$).

从 LCA 到 RMQ: 利用 DFS 序.(特殊的 ± 1 RMQ)
从 RMQ 到 LCA: 利用笛卡尔树.

与 RMQ 问题的联系

RMQ(Range Minimum/Maximum Query): 查询区间最值.

静态: ST(Sparse Table)($O(n \log n) - O(1)$).
动态: 线段树 ($O(n) - O(\log n)$).

从 LCA 到 RMQ: 利用 DFS 序.(特殊的 ± 1 RMQ)
从 RMQ 到 LCA: 利用笛卡尔树.

在线查询的线性做法: 以 $\frac{\log n}{2}$ 为大小分块, 再使用 Sparse Table.
时间复杂度: $O(n) - O(1)$

Contents

参考资料

[1] 戴一奇, 胡冠章, 陈卫. 图论与代数结构. 清华大学出版社.
[2] 刘汝佳. 算法竞赛入门经典. 清华大学出版社.
[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms(Third Edition). MIT Press.
[4] https://en.wikipedia.org/
[5] Lloyd E., Wilson R. Graph Theory. Oxford University Press.
[6] https://www.zhihu.com/question/292283275
[7] https://oi-wiki.org/