7.24 图论基础

Quanzhou No.7 Middle School

July 24, 2017

目录

友情提示

- 从昨天讲课的情况来看，今天的讲座大部分内容将针对 0 基础选手。
- 至少具有（2013 年左右的）NOIP 提高组一等奖水平的同学们可以先看点趣味题打发时间。

趣味题

1. 选择题

Bellman-Ford, Dijkstra, Floyd 是一个（贪心/动态规划）算法。

2. 瓶颈生成树

- 瓶颈生成树是满足树上最大边最小的生成树。
  - 你可以用最小生成树凑数，但我们想要线性算法。

3. 最短简单路径

- 给定一张图，求 S 到 T 每个点最多经过一次的最短路。边权有正负。
  - 试着说明这个问题非常难（无法在多项式时间解决）。

基本概念

- 一张图包括 N 个节点和 M 条边，每条边连接恰好两个节点。

图的变种

边上带权/点上带权
带方向的边
Hyper-Edge

- 本节涉及的问题：最短路，最小生成树。

图的存储

- 只适用于需要把图建出来的场合。
- A_{ij} 表示 i 到 j 的边数，S_i, T_i 表示第 i 条边的始终点
- 现在（5 年前）比较流行的是数组模拟链表方式。
- 数组 $head[x]$ 表示节点 x 的第一条出边在边数组中的位置
- $v, next$ 各有 M 个元素，表示 M 条边
  - $v[i]$ 表示第 i 条边的终点（为什么不用 $u[i]$ 表示第 i 条边的起点？）
  - $next[i]$ 表示第 i 条边在链表中的下一个元素
- 双向边的场合：拆成两条单向边

Code

- 代码只有展示用途
- 个人认为更自然的写法是假设 $head[i] = -1$
-

```
int head[maxN], v[maxM + 1], next[maxM + 1];
// Assume initially head[i] is 0
int total; // # of current edges
void add(int x, int y) {
  v[++total] = y;
  next[total] = head[x];
  head[x] = total;
}
```

DFS

- 第一个问题：怎么拿到一个点的所有出边
  - 从 $head[i]$ 开始，不断走 $next$，经过的所有 v 就是出边目标
- 图的深度优先遍历
  - 从任意未访问节点开始，每次找一条当前节点出发，目标节点未访问的边，访问目标节点。
  - 这是一个递归过程。
- 务必牢记这个算法，我们在下半段会大量运用这玩意。

Code

- 代码只有展示用途
- 你当然可以写不递归的 dfs，但是一般没有这个必要
-

```
bool visited[maxN];
void dfs(int x) {
  visited[x] = true;
  // Do something here
  int current = head[x];
  while (current != 0) {
    int y = v[current];
    if (!visited[y]) dfs(y);
    current = next[current];
  }
  // Also do something here
}
```

Code

- 一个缩短长度的 trick：利用 C++ 的 for 循环

```
int current = head[x];
while (current != 0) {
  // Do something;
  current = next[current];
}

for (int c = head[x], c != 0, c = next[c]) // do
    sth
```

- 需要遍历每个节点的场合

```
void dfs_all() {
  for (int i = 0; i < N; i++)
    if (!visited[i]) dfs(i);
}
```

BFS

- 用 DFS 求网格最近路径？ Are you crazy?
- 另外一些时候，我们希望从特定节点开始，优先遍历靠近这个节点的其他节点
- 为什么要用队列？

Code

- 代码只有展示用途
- 你当然可以写不递归的 dfs，但是一般没有这个必要
- (事实上这个代码在 C++ 一定编译不过，因为它的 *pop*() 是 void)

```
bool visited[maxN];
Queue<int> q;
void bfs(int root) {
  q.push(root), visited[root] = true;
```

Real Code

思考题：为什么在 $push(y)$ 的时候就要改 y 的标记

```
while (!q.empty()) {
  int x = q.pop();
  // Do something here
  int current = head[x];
  while (current != 0) {
    int y = v[current];
    if (!visited[y]) q.push(y), visited[y] =
  true;
    current = next[current];
  }
 }
}
```

最短路问题

- 给定一张图 G 和两个点 S、T，求 S 到 T 的最短路（所经过的边权之和最小）。

最短路问题的变种

- 有向图/无向图
- 单位边权/边权非负/无限制
- 从 S 出发到所有节点最短路/任意节点对最短路

Bellman-Ford / SPFA

- 单源最短路，有向图，边权任意
- 出现负权回路：（由问题定义）最短路为 $-\infty$
- 为啥不定义简单最短路？
  - 这个问题太难了
- dp[i,j]：从起点走至多 i 步到节点 j 的最短距离
  - 初始状态，转移方程，终止条件

Bellman-Ford / SPFA

- dp[i,j]：从起点走至多 i 步到节点 j 的最短距离
  - 初始值：dp[0,s]=0, dp[0,i]=$+\infty$
  - 转移：dp[i,k]$\leftarrow$dp[i-1,j]+w[j,k]
  - 不用存 dp[i-2] 之前的东西
  - 实际使用的公式（即松弛）：dp'[k]<-dp'[j]+w[j,k]
  - 判断负权回路：dp[n+1][i]<dp[n][i]

Bellman-Ford 算法的复杂度

- 无负权环，最短路最多 n-1 条边
- 每轮更新 O(m)，总时间 O(n*m)

- SPFA 算法
  - 如果 dp[i][k]=dp[i-1][k]，下一轮不考虑 k 的出边
  - 最差情况退化到 Bellman-Ford，但没人卡

Code / SPFA

- 现在没人写 bellman-ford 了
- again，代码仅供展示用途，不要试图编译或者内嵌这段代码到你的程序里
-

```
int n, dist[maxN];
bool in_queue[maxN];
void spfa(int s) {
  q.push(s), in_queue[s] = true, dist[s] = 0;

}
```

Code / SPFA

```
  while (!q.empty()) {
    int x = q.pop();
    in_queue[x] = false;
    for (int c = head[x]; c != 0; c = next[c]) {
      int y = u[c], w = weight[c];
      if ((dist[y] > dist[x] + w) && !in_queue[y])
        q.push(y), in_queue[y] = true, dist[y] =
   dist[x] + w
    }
  }
}
```

Dijkstra

- 单源最短路，有向图，非负边权
- 没有负权回路是不够的
- 贪心算法：维持一个已知最短距离的点集，每轮确定一个最近节点
  - 可以用拟阵（matroid）理论证明 Dijkstra 的正确性
- 堆/平衡二叉树维护，O(mlogn)

扩展：A* 算法

- 见前面暴力出奇迹那节课的内容
- Dijkstra 相当于估计到目标距离永远为 0，由于边权非负，这个估计永远合理（Admissible Heruistic）

Implementation

- Dijkstra 的一个主要部件是平衡二叉结构，我们就假设你有这个东西了
  - $update(s)$ 通知数据结构节点 x 的最短路有变化，其他就是优先队列
  - 假设数据结构开始存放了所有顶点，且所有 $dist$ 都是正无穷
- 这里现在只能给出伪代码，所以你显然不能复制到程序里
- 具体的模板请上网自学 STL Priority_Queue

Code

思考：*visited* 数组去哪里了，以及这段代码什么时候会炸

```
void dijkstra(int s) {
  dist[s] = 0, q.update(s);
  for (int z = 1; z < n; z++) {
    int x = q.pop();
    for (int c = head[x]; c != 0; c = next[c])
      if (dist[v[c]] > dist[x] + weight[c])
        dist[v[c]] = dist[x] + weight[c],
    update(v[c]);
  }
}
```

Floyd-Warshall

- 所有点对最短路，其他同 Bellman-Ford（而且好写）
- 动态规划
  - dp[i,j,k] 表示从 i 到 j，只使用前 k 个节点中转，得到的最短距离
  - 初始状态，转移方程，终止条件

Floyd-Warshall

- dp[i,j,k] 表示从 i 到 j，只使用前 k 个节点中转，得到的最短距离
  - 初始：dp[i,j,0]=w[i,j]
  - 转移：dp[i,j,k]←dp[i,k,k-1]+dp[k,j,k-1]
  - 压缩空间：dp[*,*,k-2] 及之前数据无用
  - 时间复杂度：O(n^3)

Code

Floyd 算法人人爱。如果你真的想复制这段代码，我不做任何保证。

```
void floyd(int n) {
  for (int k = 0; k < n; k++)
  for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
    d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}
```

算法总结

- 最短路的三个算法:
  - (比较慢) 的 Bellman-Ford 和 SPFA
  - 边权非负才能用的 Dijkstra
  - 求所有点对最短路用的 Floyd
- 在单位边权时，可以直接用 BFS，效率更高

一些简单题

- 给定有向图 G，边权非负整数，起终点 S 和 T。要求 O(mlogn)。
- 0. 输出 S 到 T 的一条最短路径。
- 1. 输出 S 到 T 的最短路径里边数最少有几条。
- 2. 对每条边，输出这条边有没有可能在 S 到 T 的最短路径上。
- 3. 输出 S 到 T 的字典序最小的最短路径。
- 4. 输出 S 到 T 有多少条不同的最短路径。

Solution

- 0. Dijkstra 更新时记录更新来源。
- 1. trick：给每条边加 1e-7 的边权（假设 N<=10^6）。
- 2. 把图反向，从 T 跑一次单源最短路。对于边 i,j，如果 DS[i]+w[i,j]+DT[j] 恰为 S-T 最短路，则边可以在最短路上。

最短路图

- 一个图 G 经过上述操作留下的边组成一个有向无环图 (DAG)。
- 在这个图上，任意一条 S 到 T 的路径都是最短路，且任意 S 到 T 的最短路都在这个图上。
- 以下两问需要用到这个性质。

- 3. 在最短路图上从 S 开始贪心地走标号最小的边。
- 4. 在最短路图上以拓扑序 dp。

Problem ?

- 差分约束系统很重要。。必须讲
- 有 n 个变量和 m 个约束，每个约束为 $x_i - x_j \leq d_{ij}$ 的形式
- 求一组可行解
- 提示：构造一个无解情形

Solution

三角不等式

- 若 d_{ij} 表示某个图 G 上 i 到 j 的最短路长度
- 则 $\forall i, j, k, d_{i,j} \leq d_{i,k} + d_{k,j}$
- Corollary： $\forall i, j, k, d_{i,j} \leq d_{i,k} + w_{k,j}$

- 如果 $\{x_i\}$ 是一组解，那么 $\{x_i + c\}$ 是一组解
- 假定 $x_1 = 0$，考虑从 1 出发的最短路 $A_i = d_{1,i}$，则有 $A_j \leq A_k + w_{k,j}$
- $A_i - A_j \leq w_{ji} \leftrightarrow x_i - x_j \leq d_{ij}$
- 令 $w_{ij} = d_{ji}$ 跑最短路即可

Solution(cont'd)

- 显然这题还没完
- $A_i = -\infty \to$ 有负权环，无解
- 如果跑出来的 $A_i = +\infty$ 怎么办?
- 新建虚拟点 0，强制 $A_0 \le A_i + 0$ 且 $A_0 = 0$
  - 也就是 0 向所有点连边，费用为 0

Problem 5

- 一个数 N 为伪二进制数，当且仅当它的十进制表示只包含 0 和 1。
- 例如 1, 0, 110, 110110, 110110110110110。
- 输入 P(<=10^5)，输出整除 P 需要的最小长度。
- 例如输入 55，则输出 3（110 是 55 的倍数）。

Solution

- 建立 P 个点表示当前数模 p 余 0，1,⋯⋯p-1 的状态。
- 从 i 到 10*i mod p，(10*i+1) mod p 连边，表示在当前数后面加上一个 0 或者 1。
- 找一个 0->0 节点的最短路（环），输出长度。

Problem 6

- 游戏地图是一张有向图 G，每条边有花费时间 w，有的边上有一只怪物。
- 某无名勇者要从节点 S 到节点 T 去交任务。
- 他的装备快坏了，所以只能经过最多 K 条有怪物的边。
- 求满足上述条件下的最短旅行时间。
- N<=10000, K<=5。

Solution

拆点/拆边

- 在图论题目中，拆点/拆边是常见套路之一。
- 常见于带某种约束的最短路，拆点表示约束的不同状态。
- 也有按照奇偶性拆点/拆边的例子。

- 一个点拆成 K+1 个，第 i 个点表示勇者到该节点时还能砍 i 次。
- 跑普通最短路即可。
- 改成至少需要经过 K 条特定边呢?

Problem 7

- 某交易所有 N 种货币可以交换，第 i 种到第 j 种汇率为 w[i,j]。没有手续费。
- 某些时候，由于交易所疏忽，你可以拿着某种货币，进行若干次换汇后，获得更多的该种货币，即套汇。
- 给定交易所 N 种货币两两之间的汇率，问是否有套汇问题。
- $N \leq 100$

Solution

- $\prod w_i > 1 \leftrightarrow \sum lgw_i > 0$
- 汇率取对数的相反数，判新图有没有负权环。
- 可以直接用乘法替代对数加法避免算对数的操作。

Problem 8

- 给定图 G，有点权和边权。
- 一条路径总权值定义为边权之和加上中途经过的最大点权。
- 求所有点对间最小权值路径。
- $N \leq 100$

Solution

- 把节点按点权从小到大排序，跑 floyd。
- answer[i,j]=min(dp[i,j,k]+w[k])

最小生成树

- 给定无向图 G，边上带权，输出 G 的一个生成树，树边权值之和最小
- 有向图的对应问题：最小树形图，比较复杂
- 边权正负无所谓
- Prim 和 Kruskal 算法都是贪心思想的应用

相关算法

Prim 算法

- Dijkstra：每次选取当前离 S 最近的点 Prim：每次选取当前离生成树最近的点
- $O(mlogn)$，用神奇的数据结构（斐波那契堆）可以做到 $O(m+nlogn)$

Kruskal 算法

- 把边按权值排序，从小到大贪心地加入生成树
- 可以用并查集优化合并过程
- 瓶颈在排序，$O(mlogn)$

- 个人喜好 Kruskal，比较精简，一般会一种即可。

Kruskal 算法

MST 的环性质

- 对于 MST 外的一条边，它和 MST 上的边组成的环里外边权值一定最大。
- 否则我们可以把外边和一条内边交换，连通性不变。
- 可以用这个证明 Kruskal 的正确性

- 这时候希望边集用什么方法存下来？（肯定不是链表）

Code

再次强调，不要试着编译这段代码，或者套入你的程序中。假设你已经有了一个并查集 (dfu)，且边已经排好序。

```
int n, m, s[maxM], t[maxM], w[maxM], f[maxN];
int kruskal() {
  int ans = 0;
  for (int e = 0; e < m; e++)
    if (dfu(s[e]) != dfu(t[e])) {
      f[s[e]] = t[e];
      ans += w[e];
  }
  return ans;
}
```

Problem 9

- 给定一张无向图 G。对每条边，判断它是否一定不在最小生成树内。

Solution

- 跑一下 Kruskal 算法，在加入每组相同边权的边前用并查集判断每条边是不是连接了两个分量。
- Kruskal 完成后剩下的边都不会在 MST 里。
- 处理起来需要小心以保证复杂度不退化。

Problem 10

- 求一棵树的次小生成树。
- 边权可以相同。

Solution

- 一定是换一条边进来，换一条边出去。可以证明没必要多换。
- 枚举换进来的边，出去的边是生成树上两个端点间权值最大的边。
- 用倍增稀疏表维护是最省事的方式。

Problem 11

- 你面前有 N 个篮子，每个篮子里有一个球，或没有球。
- 你现在可以询问第 i 个到第 j 个篮子总球数的奇偶性，费用为 c[i][j]。
- 求弄清楚每个篮子里有没有球需要的最少费用。

Solution

- 说的好听，实际上直接跑 MST 就过了。

中场休息

概述

- （无向）图的 DFS 树和 DFS 序
- 求割点、割边的 Tarjan 算法
- 少量二分图相关
- 本节重点在理解相关算法上，因而例题会少一些。

趣味题时间

趣味题 4. 交错边

- 给定无向图 G，有 N 个点 M 条边。给定 Q 个询问 (x,y)，问 x 和 y 之间是否存在长度为奇数的简单路径。
  - 简单路径指的是不重复经过两个节点的路径。
  - $N, M, Q \leq 10^5$，在线回答
  - 注意：我们更注重正确性证明而非如何实现。

趣味题 5. 最短路

- 给定有向图 G，每条边边权实际上是 2^w，求最短路
- $N, w \leq 10000$

无向图的 DFS

- 对无向连通图进行 dfs，dfs 过程中走的边形成一颗树，称为 DFS 树。
- 对非连通图会形成一个森林。
- 节点在 DFS 过程中访问有先后，称为 DFS 序。
- 把图 G 的不在 DFS 树上的边称为非树边

非树边都是返祖边

- 给定一颗生成树，树外连接一个节点和其祖先的边称为返祖边。
- 连接两个没有直接祖孙关系的节点则为横叉边。
- DFS 树外不会有横叉边，可以试着脑补一下证明。

连通性

- 连通性就是字面意思。
  - 如果一个图 G 任意两点间有路径，则 G 连通。
- 割点/割边：如果图 G 删除某一点/边后不再连通，则该点/边为割点/割边。
- 有时候割边也称为桥。
- 不存在割点的连通图称为点双联通图。
- 不存在割边（桥边）的连通图称为边双连通图。
- 一个图 G 的极大的满足以上两种性质的联通子图称为点/边双联通分量。

小测验

1. 判断：对于一棵树，每个节点都是割点。
2. 判断：对于一棵树，每条边都是割边。
3. 列举一个是点双联通，但不是边双联通的图。
4. 列举一个是边双联通，但不是点双联通的图。
5. 判断：一张图每条边最多只属于一个边双联通分量。
6. 判断：一张图每个点最多只属于一个点双联通分量。

Solution

1. 错误，只有度为 2 以上的点是割点。
2. 正确。
3. 2 个节点和 1 条边的图。
4. 5 个节点的沙漏状图。
   - 一般情况下，点双联通条件强于边双联通条件。
5. 正确，割边不属于任何边双联通分量，其他每条边恰好在一个边双联通分量内。
   - 常见的“双联通收缩”就是把一个边双联通分量缩为一个点，之间由割边连接。
6. 错误，每个割点属于多个点双联通分量，其他每个点恰好在一个点双联通分量内。

Tarjan 算法

- 对图 G 建立 DFS 树，用 DFS[i] 表示节点 i 的 DFS 序，LOW[i] 表示 i 或其子树中通过返祖边连接的 DFS 序最低的节点。

割点的判定

- (1)i 是树根并且有多于一个子树
- (2)i 在 dfs 树上有一个孩子 j，且 DFS[i]<=LOW[j]
- 移除割点后，图成为若干联通块，每个联通块和相邻的割点组成点双联通块

割边的判定

- (i,j) 是树边，且 DFS[i]<LOW[j]
- 移除割边后，各个联通块为边双联通块

实现的一些细节

- 一般为了省事都是写成递归形式
- 需要输出双联通分量的场合，不需要单独求割点割边再分割图，可以在 Tarjan 算法过程中一起处理
- 一般使用栈存放当前已经考虑的点/边
- 这里不再讨论具体实现
  - 三四屏幕都放不下了

Problem 12

给定无向图 G,

1. 最少添加多少条边后可以成为边双联通图?
2. 添加一条边后,原图至少有几条桥边?

Solution

双联通分量缩点

- 在这一类连通性问题中，经常会遇到需要将边双联通分量缩点的情形。
- 可以用并查集维护每个点的归属。
- 如果原图连通，缩点后变成一棵树，每个节点是原图的一个双联通分量，每条边对应原图一条割边。

- 将图进行缩点。每个度为 1 的节点都需要至少一条边。
- 第一问答案为 $(x+1)/2$，x 是树上度为 1 的节点个数。
- 第二问先求树的直径（两点间最长路径），用树边数减去直径就是答案。

Problem 13

- 给定无向图 G，问有多少个点不在任意奇数长度的简单环内（即一条边不经过两次）。
- 保证图 G 无自环。
- $n, m \leq 100000$
- 提示：二分图上是没有奇环的

Solution

- 先将图进行缩点。桥边肯定不在任何简单环上。
- 一个点可以在奇环上 $\leftrightarrow$ 所在的双联通分量不是二分图。
  - 证明?

Problem 14

- 给定无向图 G。标记最少数量的点，使得删除任意节点后，每个节点都和至少一个标记点联通。

Solution

- 答案是仅邻接一个割点的点双联通分量个数。
- 整个图双连通则答案为 1。

Problem 15

- 给定无向图 G，问哪些边一定在最小生成树上。

Solution

- 和之前一样，边权不同的部分互相不影响。
- 在每组边权之前做连通分量缩点。
- 子问题：给你一张图，问哪些边一定在这张图的生成树上
- 答案是所有的树边。
- 需要谨慎处理避免炸掉。

前言

- 有一些比较神奇的玩意，难度太高了就没有放在前面
- 看剩余时间，随机选讲

趣味题题解 1

瓶颈生成树

- 二分答案。先用线性中位数算法找到权值居中的边，然后把权值小的一半边连上。
- 如果图已经联通，说明答案在前一半，把图清空对前一半递归查找。如果图没有联通，答案在后一半，缩联通块后对后一半递归查找。
- 复杂度是 $O(E) + O(E/2) + O(E/4) + \cdots = O(E)$

最短简单路径

- 如果一张无向图所有边权都是-1，那么长度为 $-n$ 的简单环就是一个哈密尔顿环。
- 换言之，如果这个问题可以多项式时间解决，那么哈密尔顿环就可以多项式时间解决。

趣味题题解 2：交错路径

- 对图随便建个生成树。如果 x 到 y 在树上路径长就是奇数，直接 yes。
- 定理：x 到 y 的答案为 YES，当且仅当 x 到 y 的路径上有一条边在图 G 的一个奇环上。
  - 必要性：找到一条奇数长度路径，和树上路径取对称差，至少得到一个奇环。
  - 充分性：找到一个有覆盖的奇环 C，使得 C 和路径有覆盖，然后取对称差。
  - 严格证明需要考虑出现对称差后出现（1）孤立环，(2) 非简单环的情形。
  - (1）中有奇环直接完成证明。(2）中需要找到一条非简单环上被对称差抹去的边后处理。

趣味题题解 2： 交错路径

- 给定图 G，节点 S 和 T。一个 G 的不包括 S 和 T 的节点子集 A 是隔离集，当且仅当移除 A 后，S 和 T 不再连通。

Menger 定理

- 在图 G 上有两个顶点 S 和 T，假设 S 和 T 无直接连边，最小隔离集大小等于 S 和 T 之间两两顶点不相交的最大路径数。
- 证明重点：S 和 T 往最小隔离集每个节点有互相不相交路径。
- 对于点双联通分量，需要移除节点个数至少为 2。

趣味题题解 2：交错路径

- 定理：对于每个点，它在一个奇环上当且仅当它所在的某个点双联通分量无法进行 2 染色。
  - 也就是说，如果一个点双联通分量有一个奇环，分量内每个点都在一个奇环上。
  - 取分量上一个点 X，和奇环两个相邻点 A、B（直接和两个奇环上点相邻则马上可证。）。
  - 根据 Menger 定理，X 到 A 有两条不相交路径。如果这两条路径和奇环有任意 A 以外的交点，则取 C 和 D 分别为两条路径第一次和奇环相交点。无论 C-D 的方向，此时 X-C-D 为一个简单环，证明完毕。

趣味题题解 2：交错路径

- 否则，X 到 A 有一条不经过奇环的路径，X 到 B 也有一条不经过奇环的路径。
- 如果这两条路径不相交，无论 A-B 的方向，X-A-B 是一个简单环。
- 否则，这两条路径的最后交叉点为 C。
  - 上面的都是我瞎说的，其实这题我也不会做。(flag：我觉得我快证完了，然后想了好几天。)

df

2-最短路

- 给定一张图 G，求 S 到 T 的一条路径，使得 $\sum 2^{w_i}$ 最小。
- 就是每条边输入边权是 w，实际上就是 2^w。
- $n, m, w \sim 100000$

Solution

- 边权太大了显然没法直接搞
  - 支持把一个数加上 2 的某个次方，以及比较两个数?
  - 持久化数据结构!
- 每一步大概是 $O(logw)$ 的

矩阵树定理

- 给定一张无向图 G，求它的生成树个数。
- $n \leq 100$。
- 这 TM 能做?

矩阵树定理

Kirchhoff Matrix-Tree Theorem

- 令 w_{ij} 表示 i 和 j 之间边的数目，d_i 表示 i 的度数。
- 构建矩阵 M，主对角线上 $M_{ii} = d_i$，其他地方 $M_{ij} = -w_{ij}$。
- M 的任意一个代数主子式（去除 M 的一行一列得到的矩阵）的行列式是 G 的生成树个数。

- 矩阵乘法利用高斯消元可以做到 $O(n^3)$
- 这 TM 能证?

矩阵树定理

矩阵树定理的证明

- 注意到构造的矩阵 M 其实是图 G 关联矩阵的外积 $(M = WW^T)$
- 关联矩阵 W 的一个 (n-1) 阶余子式非 0$\leftrightarrow$ 余子式对应列是一棵树
- 利用柯西-比内公式得到结论

Still not over yet??

- 接着讲昨天没讲完的东西
- 反演理论
- 高斯消元
- 群论基础