图论

张子良

概览

- 图的基本模型
- 图的存储方法
- 图的遍历方法
- 有向图拓扑排序
- 最短路算法
- 最小生成树
- ~~有向图强连通分量~~
- ~~无向图双连通分量~~
- 二分图
- 树上倍增

基础知识

图的基本模型

图是点和边组成的集合体，$G =< V, E >$

V 是点集

E 是边集

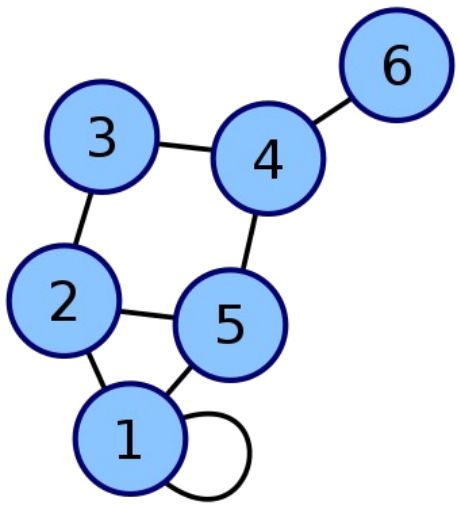


图 1：图的示例

图的基本概念

- 有向边，有向图
- 无向边，无向图

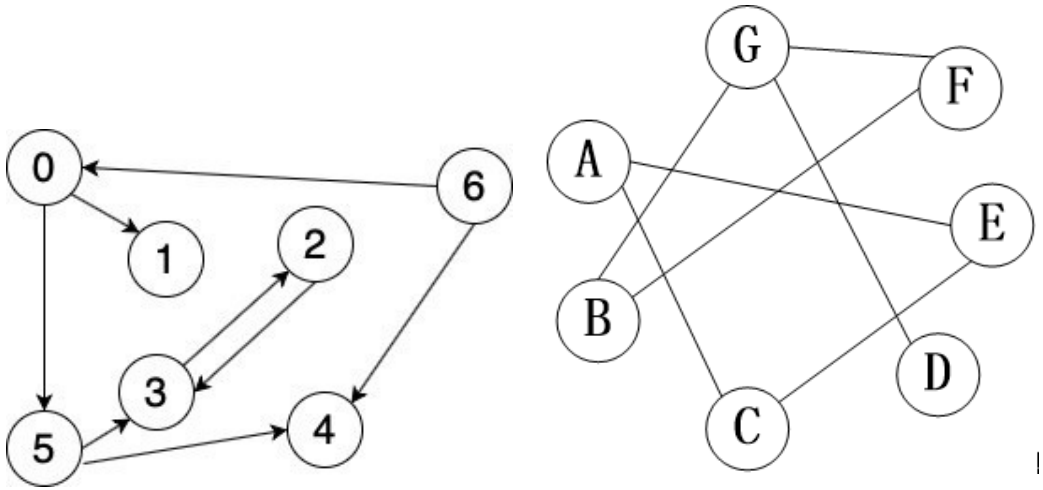


图 2: 有向图和无向图

图的基本概念

0 无权

0 点权

0 边权

0 负权

图的基本概念

- 环
- 自环
- 重边
- 有向无环图

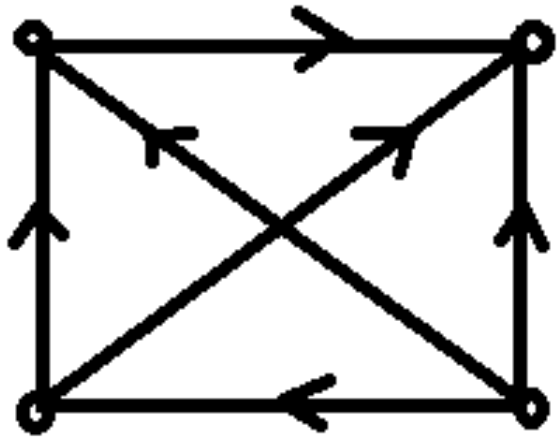
图的基本概念

- 路径
- 简单路径
- 连通

特殊的图

- 树
- 完全图
- 竞赛图

通过在无向完全图中为每个边缘分配方向而获得的有向图。



- 基环树

N 个点 n 条边的无向图，树 + 恰好一个环

- ... ...

图的输入方式

最常见的输入方式是用 1 –N 表示顶点，并逐行给出 M 条边所连接的两点和边权大小。

N M

u_1 u_1 v_1 w_1 w_1
u_2 u_2 v_2 w_2 w_2
… …
u_m v_m w_m

u_1 v_1 w_1 u_1 v_1 w_1 u_1 v_1 w_1
u_2 v_2 w_2 u_2 v_2 w_2 u_2 v_2 w_2
… … …
u_m v_m w_m u_m v_m w_m u_m v_m w_m

如果有点权，一般会用一行 N 个数表示每个点的权值大小。

$p_1 p_2 \ldots p_n$

图的存储方式

邻接矩阵是最简单的图存储方式。

对于 N 个点的图，用 $N \times N$ 的二维数组记录两点之间是否有边相连，以及边权大小。

$$A_{u,v} = \begin{cases} w_{u,v} & < u, v > \in E \\ \infty & < u, v > \notin E \end{cases}$$

空间复杂度 $O(N^2)$

对于边不存在的情况，可能采用 ∞，也可能使用 0 或 -1 等等特殊值，视具体情况而定。

对于有重边的情况，可能不能完整保留所有边的信息。

图的存储方式

邻接表是更为常用的图存储方式。

对每个点 u 用不定长度数组或链表存储所有以其为起点形如
$< u, v >$ 的边。

无向图双向边拆分为两条单向边。

- 不定长数组
- 链表

空间复杂度 $O(N + M)$

图的遍历方法

- 广度优先遍历 BFS 队列
- 深度优先遍历 DFS 栈

图的遍历方法

广度优先遍历 BFS

用队列实现，按照一定顺序依次访问每个结点。

- 创建一个包含起点的队列
- 取出队首结点，将相邻的未入队结点加入队列
- 重复上一步直到所有结点都被遍历过

适用于无权图上单源最短路问题。

图的遍历方法

深度优先遍历 DFS

用栈（递归函数）实现，按照一定顺序依次访问每个结点。

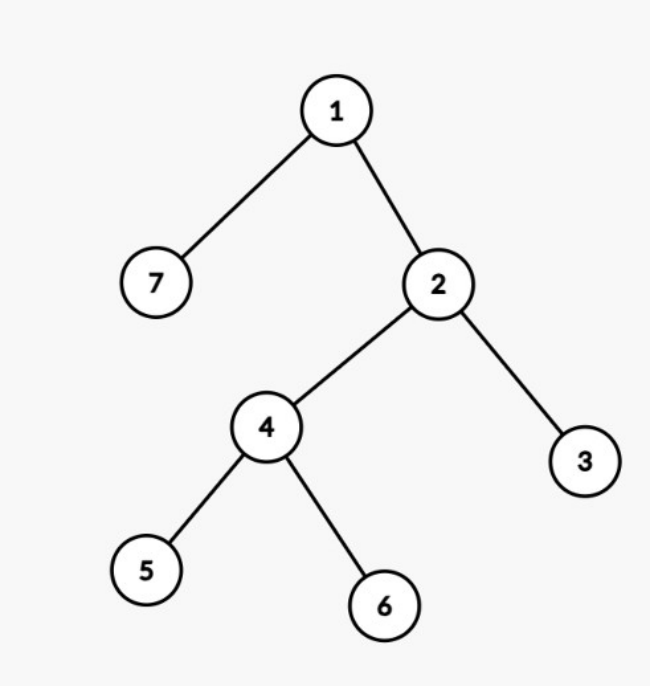
- 从起点 s 开始遍历
- 在遍历到 u 时，递归遍历所有未被访问的相邻结点 v
- 当遍历 s 的函数退出时，所有结点都被遍历

图的遍历方法

- BFS 和 DFS 是最基础的图的遍历方法，更多的遍历方法基本都是基于此两者
- BFS 和 DFS 都可以用来判断图连通，或者求连通子图
- 两者各有优劣，要根据实际情况选择
- 结合搜索的知识

三种二叉树 DFS 序

- 前序遍历 : 根节点 -> 左子树 -> 右子树
- 中序遍历 : 左子树 -> 根节点 -> 右子树
- 后序遍历 : 左子树 -> 根节点 -> 右子树



例 1

给定一个有向图，边权为 1 或 2，求单源最短路。

例 1

稍微改写一下 BFS 即可。

- 创建三个集合，Q_0 表示当前层，Q_1 表示距离为 1 的层，
  Q_2 表示距离为 2 的层，初始 $Q_0 = \{s\}, Q_1 = \emptyset, Q_2 = \emptyset$
- 依次取出 Q_0 中的点，将其邻点放入对应的 Q_1 或 Q_2 中
- $Q_0 = Q_1, Q_1 = Q_2, Q_2 = \emptyset$

注意一个点可能和当前层既有长度为 1 的边，又有长度为 2 的边，应当将其加入 Q_1 而非 Q_2 。

例2

给出一个有向无权图和起点 s，对每条边 $< u, v >$ 回答，如果删去这条边，从 s 到 v 的最短路长度是否会改变。

例2

- 求出从 s 出发的单源最短路
- 建立最短路图，即保留满足 $d_v = d_u + 1$ 的边 $< u, v >$
- 在最短路图上，如果 v 的入度为 1，则该入边是从 s 到 v 的必经边，若删去则 v 的最短路长度会改变
- 在最短路图上，如果 v 的入度大于 1，则删去任何一条入边，v 的最短路长度都不会改变

拓扑排序

拓扑排序

有向无环图的拓扑排序即将所有顶点排为线性序列，使得对于任意的 $< u, v > \in E$，都有 u 在线性序列中出现于 v 之前。

有向图中如果有环，则一定不存在拓扑排序；如果没有环，则一定存在拓扑排序。

拓扑排序

- 选取一个入度为 0 的点记为 u
- 将 u 添加到线性序列末端
- 删去所有 u 的出边
- 重复上述步骤直到所有点都被加入序列

例 1

有 n 项任务，有 m 个限制，第 i 个限制要求执行任务 u_i 之前必须要完成任务 v_i 。请问是否存在合适的任务执行顺序，满足所有的限制。

例1

将每个任务视为一个点，任务之间的依赖构成了有向边。如果该有向图中没有环，则存在拓扑排序，而拓扑排序就是可行的任务执行顺序；如果该有向图中存在环，则无解。

例 2

有 n 项任务，有 m 个限制，限制有如下两种：

- 执行 u 任务之前必须要完成 v 任务
- 存在某一时刻， u 和 v 任务都在执行

请问是否存在安排每个任务起始时间和结束时间的方案，满足所有的限制。

例2

为每个任务的起始时间和结束时间各对应一个点，任务 i 的起始时间点记为 s_i，结束时间点记为 e_i。

- 要保证每个任务的结束时间在起始时间之后，所以对所有 i，连边 $< s_i, e_i >$
- 如果要求任务 a 在任务 b 开始执行之前完成，则连边 $< e_a, s_b >$
- 如果要求任务 a 和 b 在某个时刻都在执行，则连边 $< s_a, e_b >$，$< s_b, e_a >$

对于上面的有向图，如果存在环则无解，否则根据其拓扑排序易构造一个方案。

例3

对于带边权的有向无环图，求单源最短路。

例3

记起点为 s，拓扑排序形如 $a_1, a_2, \ldots, a_t, s, b_1, b_2, \ldots, b_k$，记点
i 的最短路为 d_i。

$$d_v = \min_{<u,v>\in E} \{d_u + w_{u,v}\}$$

- 从 s 出发无法走到 $a_1, a_2, \ldots, a_t$，对这些点其最短路为 ∞
- 从 s 出发走到 b_i 之前一定只经过 $b_1, b_2, \ldots, b_{i-1}$，即 d_{b_i} 的求解依赖于 $d_{b_1}, d_{b_2}, \ldots, d_{b_{i-1}}$
- 按顺序依次求解 $d_s, d_{b_1}, d_{b_2}, \ldots, d_{b_k}$ 即可

POJ 1094 Sorting It All Out

由一些不同元素组成的升序序列是可以用若干个小于号将所有的元素按从小到大的顺序 排列起来的序列。例如，排序后的序列为 A, B, C, D，这意味着 A < B 、 B < C 和 C < D 。在本题中，给定一组形如 A < B 的关系式，你的任务是判定是否存在一个 有序序列。输出到哪一项可以确定顺序或者在这一项最先出现冲突，若所有的小于关系都处理完了都不能确定顺序也没有出现冲突，就输出不能确定。

POJ 1094 Sorting It All Out

- 冲突即为出现环
- 确定即为拓扑排序时队中元素不大于 1
- 每次加入新的关系重新拓扑排序一次即可

最短路算法

最短路算法

- 全局最短路 Floyd
- 单源最短路 Dijkstra SPFA Bellman-Ford

Floyd

- 基于动态规划，$F_{k,u,v,}$ 表示使用点 $1, 2, \ldots, k$ 时，点 u 到点 v 的最短路
- 从小到大枚举 k，u 和 v 之间的最短路要么不经过 k，要么经过 k 一次且除此之外只包含前 $k-1$ 个点

- $F_{k,u,v} = \min\{F_{k-1,u,v}, F_{k-1,u,k} + F_{k-1,k,v}\}$

易见，使用二维数组不断覆盖更新即可。

时间复杂度 $O(N^3)$

空间复杂度 $O(N^2)$

可以处理含有负权边的情况，如果含有负环，则存在 i 使得 $F_{i,i} < 0$。

Dijkstra

适用于没有负权边的图。

0 将所有点分为两个集合，最短路确定的集合 S 和最短路未确定的集合 T，初始 $S = \{s\}$

0 求 T 中每个点 v 的当前最短路

$$d_v = \min_{<u,v>\in E, u\in S}\{d_u + w_{u,v}\}$$

0 取出 T 中 d_v 最小的点，其最短路一定就是 d_v，将其加入 S

0 不断重复上面的操作，直到所有点的最短路都确定

朴素写法时间复杂度较劣，可以采用堆优化至 $O((N + M)logN)$

Bellman-Ford

- 初始令 $d_s = 0$，其余 $d_i = \infty$
- 依次使用每一条边 $< u, v >$ 更新，$d_v = \min\{d_v, d_u + w_{u,v}\}$
- 不断循环直到所有 d_i 都不会更新
- 因为任何一条最短路包含至多 $n - 1$ 条边，没有负环则不超过 n 轮就会停止

时间复杂度 $O(NM)$

空间复杂度 $O(N + M)$

可以对包含负权边的图使用，如果存在负环，则循环超过 n 轮后依然不会停止，可以用来判断负环是否存在。

SPFA

考虑使用队列优化 Bellman-Ford 算法，如果更新了 d_u ，则将 u
入队。每次取队首 u 更新其邻点 v 的 d_v 。

POJ 1125 Stockbroker Grapevine

有 N 个股票经济人可以互相传递消息，他们之间存在一些单向的通信路径。现在有一个消息要由某个人开始传递给其他所有人，问应该由哪一个人来传递，才能在最短时间内让所有人都接收到消息。

POJ 1125 Stockbroker Grapevine

全局最短路

POJ 1502 MPI Maelstrom

给出 N 个处理器之间传递信息所需时间矩阵的下三角，求信息从第一个处理器传到其它所有处理器所需时间最大值。

POJ 1502 MPI Maelstrom

单源最短路

POJ 1511 Invitation Cards

N 个点 M 条边的有向图，询问对于所有 i，从点 1 出发到点 i 并返回点 1 所需时间最小值。

POJ 1511 Invitation Cards

0 正向建图 + 单源最短路

0 反向建图 + 单源最短路

POJ 1724 ROADS

有 N 个城市和 M 条单向道路，每条道路有长度和收费两个属性。

求在总费用不超过 K 的前提下从城市 1 到城市 N 的最短路。

POJ 1724 ROADS

Dijkstra， 同时维护当前路径总费用，超过费用上限的状态不再转移。

POJ 1797 Heavy Transportation

给出 N 个点 M 条边的无向图，每条边有最大载重。

求从点 1 到点 N 能通过的最大重量。

POJ 1797 Heavy Transportation

- 法 1：变种单源最短路，路径长度改为路径边权最小值
- 法 2：并查集维护加入前 k 大边时的连通性

最小生成树

最小生成树算法

- Prim
- Kruskal

Prim

- 将所有点分为两个集合，已经和点 1 连通的集合 S、未和点 1 连通的集合 T
- 计算集合 T 中每个点 u 和集合 S 的距离，

$$d_u = \min_{<u,v>\in E, v\in S}\{w_{u,v}\}$$

- 选取集合 T 中距离 S 最近的点 u，选中对应的边，加入集合 S
- 重复上面的过程，直到所有点都被加入集合 S

朴素写法时间复杂度较劣，可以采用堆优化至 $O((N+M)logN)$

Prim

Prim 是一个基于贪心的算法，可以采用归纳法和反证法证明其正确性。

- 首先证明第一条选中的边 e_1 一定包含在某最优解方案中
- 如果最优解中不包含边 e_1，则加入 e_1 一定会出现环，且环上存在比 e_1 大的边，用 e_1 替换之答案更优
- 假设最优解包含前 k 个选中的边，$e_1, e_2, \ldots, e_k$，则类似地可证明 e_{k+1} 存在于最优解中
- 运用归纳法，Prim 算法得到的 $n-1$ 条边构成最优解

Kruskal

- 将所有边按照边权从小到大排序
- 依次考虑每一条边 $< u_i, v_i >$，如果这条边和之前选中的边形成环，则不选中此边；反之，选中此边
- 当考虑遍所有边后，选中的边一定构成了一棵最小生成树

需要并查集的支持，时间复杂度一般认为是 $O(MlogM)$

Kruskal

证明依赖于拟阵的知识。
有兴趣的同学可以自行阅读。

拟阵

遗传性：若 S 是一个独立集，那么 S 的子集 S' 是独立集。

遗传性的推论：空集是独立集。

交换性：若 A 和 B 是 S 的两个独立集且 $|A| < |B|$，那么存在一个元素 x 满足 $x < A$ 且 $x \in B$，使得 $A \cup \{x\}$ 是一个独立集

交换性的推论：一个集合的所有极大独立集大小都相同。

例：线性无关组、无向图生成森林。

拟阵最优化

拟阵最优化问题：将集合中每个元素赋予一个权值，求权值和最小（大）的极大独立集。

拟阵最优化的贪心算法：

维护当前独立集 G，初始为空。将元素按照权值排序，从小到大枚举元素 x，若 $G \cup \{x\}$ 是一个独立集，那么就将 x 加入独立集并将 x 的权值累加入答案。最后的结果就是权值和最小的及大独立集。

证明:

如果最优解不包含最小元素 x_1，记该集合为 A，创建新的集合 $B = \{x_1\}$，利用交换性不断将 A 中元素加入 B 直到 $|A| = |B|$，则有 B 集合为更优解，矛盾。故 x 一定属于最优解。利用数学归纳法，假设已经证明 $x_1, x_2, \ldots, x_{k-1}$ 属于最优解，如果存在最优解不包含 x_k，还是创建新的集合 $B = \{x_1, x_2, \ldots, x_{k-1}, x_k\}$，利用交换性将元素加入 B 得到更优解，矛盾。

最大生成树

略

BZOJ 1601 灌水

Farmer John 已经决定把水灌到他的 $n(1 \le n \le 300)$ 块农田，农田被数字 1 到 n 标记。把一块土地进行灌水有两种方法，从其他农田饮水，或者这块土地建造水库。建造一个水库需要花费 $w_i(1 \le wi \le 100000)$，连接两块土地需要花费 $p_{i,j}(1 \le p_{i,j} \le 100000, p_{i,j} = p_{j,i}, p_{i,i} = 0)$. 计算 Farmer John 所需的最少代价。

BZOJ 1601 灌水

建立超级水库点，在某点建立水库视为选择长度为 w_i 的边将其和超级水库连通，引水就是选择长度为 $p_{i,j}$ 的边。目标是选择长度和尽可能小的边，使得所有点和超级水库连通。

BZOJ 2743 滑雪与时间胶囊

a 来到雪山滑雪，这里分布着 M 条供滑行的轨道和 N 个轨道之间的交点（同时也是景点），而且每个景点都有一编号 i 和一高度 H_i。a 能从景点 i 滑到景点 j 当且仅当存在一条 i 和 j 之间的边，且 i 的高度不小于 j。a 喜欢用最短的滑行路径去访问尽量多的
景点。如果仅仅访问一条路径上的景点，他会觉得数量太少。于是 a 拿出了他随身携带的时间胶囊。这是一种很神奇的药物，吃下之后可以立即回到上个经过的景点（不用移动也不被认为是 a 滑行的距离）。请注意，这种神奇的药物是可以连续食用的，即能够回到较长时间之前到过的景点（比如上上个经过的景点和上上上个经过的景点）。现在，a 站在 1 号景点望着山下的目标，心潮澎湃。他十分想知道在不考虑时间胶囊消耗的情况下，以最短滑行距离滑到尽量多的景点的方案（即满足经过景点数最大的前提下使得滑行总距离最小）。你能帮他求出最短距离和景

BZOJ 2743 滑雪与时间胶囊

能够到达的景点容易通过 DFS 或 BFS 求出。

最优解应当构成树形，所需要的时间即为所有边长度之和。易联想到用最小生成树 Kruskal 算法解决本问题。

如何设置 Kruskal 时边的优先级？如何保证选出的有向边集使得每个点从 1 出发可达？

以到达点高度为第一关键字，边长度为第二关键字。到达点高度高的边优先，同样高时边长度短的优先。

树上倍增

树上倍增

- 有根树（随意定根）
- 最近公共祖先
- 链上信息（和、最值）
- 优秀的时间复杂度

序列倍增

回忆普通的序列倍增思想，以 ST 表为例。

- $F_{i,j}$ 记录区间 $[i, i + 2^j - 1]$ 内信息（区间和或区间最值）
- $F_{i,j} = merge(F_{i,j-1}, F_{i+2^{j-1},j-1})$
- 取出区间 $[l, r]$ 答案时，使用若干个 $F_{i,j}$ 即可
- 如果求区间最值，那么取 $F_{l,k}$ 和 $F_{r-2^k,k}$ 即可，其中 $k = \lfloor log_2(r - l + 1) \rfloor$
- 如果求区间和，可以取 $F_{l,k_1}, F_{l+2^{k_1},k_2}, F_{l+2^{k_1}+2^{k_2},k_3}, \ldots$ 即可

树上倍增

树上从每个点出发到根结点的链也具有类似的形式。

- $F_{i,j}$ 表示点 i 向上走 2^j 步的结点
- $F_{i,0}$ 就是点 i 的父亲节点
- $F_{i,j} = F_{F_{i,j-1},j-1}$

树上倍增

如何求解 u 向上移动 k 步是哪个点?

树上倍增

将 k 写作 2 的幂次之和，如 $11 = 2^3 + 2^1 + 2^0$。

用 $G_{i,j}$ 表示 i 向上移动 j 步的结果。

- $G_{u,11} = F_{G_{u,10},0}$
- $G_{u,10} = F_{G_{u,8},1}$
- $G_{u,8} = F_{u,3}$

在 $O(logN)$ 步内完成。

最近公共祖先

树上倍增最常见的用处是求解两个点的最近公共祖先。

求解 a 和 b 的最近公共祖先

0. 将 a 和 b 调整到相同高度
0. 判断 a 和 b 是否重合，若重合则该点即为答案
0. 令 a 和 b 一起向上移动尽可能大的距离，保证移动后两点不重合
0. 此时两点的父亲结点即为答案

单次询问时间复杂度 $O(logN)$

最近公共祖先

最近公共祖先的常见求解方法有

- 树上倍增
- 树链剖分
- DFS 序 +RMQ

向上路径

如何求解从 u 到 v 路径上边权最值？保证 v 是 u 的祖先。

向上路径

- $M_{i,j}$ 表示点 i 出发向上移动 2^j 步经过边权最值

- $M_{i,0} = W_{i,father_i}$

- $M_{i,j} = merge(M_{i,j-1}, M_{F_{i,j-1},j-1})$

在树上倍增向上走时取移动区间最值更新答案即可。

树上路径

记 $g = LCA(u, v)$，则树上从 u 到 v 的路径可以拆分为两段：

- 从 u 到 g 的路径
- 从 g 到 v 的路径

树上路径

如何求解从 u 到 v 路径上的边权和?

树上路径

将路径拆分为两段向上路径，分别求解其答案再合并。

二分图

二分图

二分图：点黑白染色，邻点不同色。

二分图判定

如何判断一个给定的无向图是不是二分图?

二分图判定

- 从任意一点开始 BFS 或 DFS，起点不妨染色为白
- 当前在 u 点时，尝试将所有邻点染为不同的颜色
- 如果邻点已经染色且颜色不符，则不是二分图

二分图的等价条件

无向图是二分图当且仅当其不包含奇环。

二分图匹配

- 匹配：选取一些边，使得任意两条边没有公共点（每个点至多属于一条边）
- 最大匹配：选取边数尽可能多
- 完美匹配：所有点都在匹配中（每个点恰好属于一条边）
- 匹配边：选中的边
- 非匹配边：没有选中的边
- 匹配点：和选中边相连的点
- 非匹配点：和选中边不相连的点
- 常常将二分图的点画成两列

二分图最大匹配

二分图最大匹配是一个常见问题。

- 匈牙利算法
- 网络流

匈牙利算法

理论基础

- 交错路：从非匹配点出发，依次经过非匹配边、匹配边、非匹配边 …
- 增广路：从非匹配点出发，结束于非匹配点的交错路
- 增广路定理：任意一个非最大匹配的匹配一定存在增广路

匈牙利算法

算法思想

- 初始没有选中的边
- 寻找增广路
- 找到增广路则将路径中匹配边和非匹配边对换
- 找不到增广路则当前为最大匹配

网络流

- 取额外的两个点作为源点和汇点
- 源点向左边一列每个点连流量为 1 的边
- 右边一列每个点向汇点连流量为 1 的边
- 二分图中每条边从左向右连流量为 1 的边
- 求最大流即可

最小顶点覆盖 Knoig 定理

二分图最小顶点覆盖数等于其最大匹配数。

POJ 3041 Asteroids

一个 $N \times N$ 的网格中，有 K 个大小为 1×1 的小行星，现在可以用激光枪每次消灭一行的小行星或者消灭一列的小行星。问最少需要使用多少次激光枪消灭所有的小行星。

POJ 3041
Asteroids

- 每行为左边一点
- 每列为右边一点
- 每个小行星为一边
- 选择最少的点覆盖所有边

最小路径覆盖

给定有向图 $G = < V, E >$ 。设 P 是 G 的一个简单路（顶点不相 交）的集合。如果 V 中每个顶点恰好在 P 的一条路上，则称 P 是 G 的一个路径覆盖。P 中路径可以从 V 的任何一个顶点开始，长度也是任意的，特别地，可以为 0 。G 的最小路径覆盖是 G 的所含路径条数最少的路径覆盖。

最小路径覆盖

最小路径覆盖 = $|V|$ - 二分图最大匹配

二分图：将原图每个点拆分为入点和出点，如果原图存在 u 到 v
的边，则在 u 的出点和 v 的入点间连无向边。

BZOJ 2150 部落战争

lanzerb 的部落在 A 国的上部，他们不满天寒地冻的环境，于是准备向 A 国的下部征战来获得更大的领土。A 国是一个 $M \times N$ 的矩阵，其中某些地方是城镇，某些地方是高山深涧无人居住。lanzerb 把自己的部落分成若干支军队，他们约定：

1. 每支军队可以从任意一个城镇出发，并只能从上往向下征战，途中只能经过城镇，可以在任意一个城镇停止征战。
2. 每个城镇只能被一支军队经过。
3. 行军方式类似国际象棋中的马，不过只能走 $R \times C$ 的路线。

lanzerb 的野心使得他的目标是统一全国，但是兵力的限制使得他们在配备人手时力不从心。假设他们每支军队都能顺利占领这支军队经过的所有城镇，请你帮 lanzerb 算算至少要多少支军队才能完成统一全国的大业。

BZOJ 2150 部落战争

最小路径覆盖