# 一、图的表示法（只列出可能用到的）

邻接表：
使用频度：最常用
空间复杂度：O(V+E)
适用范围：点数任意的稀疏图
优点：能快速遍历与某个点邻接的所有点，且可以方便地存储边上的任何信息
缺点：不能快速判断两点间是否有边
加边 O(1)，删边 O(m)，其中 m 为某点发出的边的条数。

邻接阵：
使用频度：较常用
空间复杂度：O(V^2)
适用范围：点数不太多的稠密图
优点：能快速判断两点间是否有边
缺点：遍历与某点邻接的所有点很慢
<简单图> 若边上有权，设为相应的数据类型存储边权即可，无边的设为∞；否则可以设为 bool 型表示两点间是否有边。
<多重图> 若边上有权则需要设为结构体型，结构体中有个链表存储所有连接这两点的边的相关信息，否则只需设为 int 型记录该边出现了多少次。
对多重图边带权的情况：加边 O(m)，删边 O(m)，其中 m 为某两点间边的条数
其他：加边 O(1)，删边 O(1)

前向星：
使用频度：不常用。
空间复杂度：O(E)
适用范围：静态图
优点：能快速遍历与某个点邻接的所有点，且能快速定位一条边
缺点：不能动态地加边、删边，因其开销很大（O(E)）
建立边表：O(ElogE)，一次性将所有边(u, v)读入后，按主关键字 u 和次关键字 v 排序(采用快速排序)，用一个数组 start[u]记录点 u 的邻居列表的开始位置，则点 u 的邻居列表的范围是 start[u]~start[u+1]-1。

池子法：
**推荐使用此法。**
空间复杂度：O(E)
性质与邻接表一样，但使用静态空间（开好池子后便再没有动态申请内存的过程），效率要比单纯的链表或 vector 实现高很多。

具体实现：

```
const int N = 128;       // N 为顶点数
const int M = 10000;   // M 为边数，无向图时要开到实际边数的两倍以上

struct List {
```

```
    int v;          // 元素可根据需要任意添加
    List *next;
} pool[M], *c[N], *pp;

inline void add_edge(int u, int v, List *c[])          // 形参可根据需要任意添加
{
    pp->v = v;
    pp->next = c[u];
    c[u] = pp ++;
}
```

<初始化要做的工作>
(1) 对每一个 i, c[i] = NULL;
(2) pp = pool;

<加边>
add_edge(i, j, c);

若全局邻接表只有一个 c，add_edge 中可不设形参 List *g[]。

根据算法需要灵活选用。

一些习题：
O(V)时间找到一个图中的汇 t，即 indegree(t)=V-1, outdegree(t)=0，假设图采用邻接阵存储。
解法：对任意一条边(u, v)，若该边存在则 u 不是汇，否则 v 不是汇。如此便可在 O(1)时间内排除掉一个点。
那么对于最后剩下的一个点，只需再扫描一次其他点即可。

关联阵 B，则 BB'的每个元素表示什么含义？
bij = -k, i != j 表示顶点 i, j 之间边的总条数
bii = k 表示顶点 i 的度（或入度、出度之和）

# 二、BFS

时间复杂度：邻接表 O(V+E)，邻接阵 O(V^2)
BFS 树不唯一，取决于邻接表中节点的顺序，但 d 值不会变。
任意时刻，设队列中的节点为 v1, v2, …, vr，则 d[vr] <= d[v1] + 1，且 d[vi] <= d[vi+1], i = 1, 2, …, r – 1
在进行黑白染色时，可运用 BFS 而不用 DFS。
边的分类：
无向图：没有后向边和前向边
树边(u, v)：d[v] = d[u] + 1
交叉边(u, v)：d[v] = d[u] 或 d[v] = d[u] + 1
有向图：没有前向边

树边(u, v)：d[v] = d[u] + 1
交叉边(u, v)：d[v] <= d[u] + 1
后向边(u, v)：0 <= d[v] <= d[u]

一些习题：
举一个例子，在有向图 G=(V, E)中，给定源点 s 及前趋子图 G'=(V, E')，该前趋子图不可能由在 G 上运行 BFS 而获得，无论邻接表中的每个点如何排列。
见《算法导论》P539

求树的直径。两遍 BFS。

# 三、DFS

时间复杂度：邻接表 O(V+E)，邻接阵 O(V^2)
**括号定理**：对任意 u, v∈V，[d[u], f[u]] 和 [d[v], f[v]] 要么互不相交，要么一个被另一个完全包含，且被包含的为包含的点的后裔。
**白色路径定理**：深度优先森林中，v 是 u 的后裔当且仅当于 d[u]时刻发现 u 时，可以从 u 开始沿一条只由白色节点组成的路径到达 v
在深度优先森林中，v 是 u 的后裔当且仅当 u 为灰色时 v 被发现。
四种边：树边、反向边、前向边、交叉边
区分：DFS 过程中遇到边(u, v)时，根据 v 当前的颜色区分该边的类型：
白色：树边
灰色：反向边
黑色：d[u] < d[v]: 前向边（d[u] < d[v] < f[v] < f[u]，说明 v 是 u 的后裔）
d[u] > d[v]: 交叉边（d[v] < f[v] < d[u] < f[u]，说明 u, v 没有祖先后裔的关系）
三种颜色（白、灰、黑）之间边的可能性：
有向图：

| | 白 | 灰 | 黑 |
|---|---|---|---|
| 白 | 无 | 无 | 无 |
| 灰 | 树边 | 后向边 | 树/前向/交叉边 |
| 黑 | 无 | 后向边 | 树/前向/交叉边 |

无向图：

| | 白 | 灰 | 黑 |
|---|---|---|---|
| 白 | 无 | 无 | 无 |
| 灰 | 树边 | 后向边 | 树边 |
| 黑 | 无 | 后向边 | 树边 |

一个结点为白色表示它还没有被探寻到，故没有发出的边。

对无向图作 DFS 时，只会出现树边和反向边
当前点的所有祖先一定都是灰色。
DFS 后形成的顶点序列恰好构成一条路径，该路径经过每条边恰好两次（每个方向各一次）
非递归实现：模拟一个系统栈，另设一个数组记录每个点当前访问到了其邻接表中的哪个顶点。

**性质很多而且很美妙，需要勤思考！**

一些习题：

对如下假设举一个反例：在有向图 G 中若从 u 到 v 有一条路径，且在某次 DFS 中 d[u] < d[v]，则在深度优先森林中 v 是 u 的后裔。

见《算法导论》P548

HOJ 2064　Journey to Tibet

# 四、拓扑排序

重要引理：一个有向图无环当且仅当 DFS 不产生反向边。

三种方法：

**反复寻找入度为 0 的顶点。**

实现方法：用队列维护，每次寻找入度为 0 的顶点，处理之并删除从该点发出的所有边。如产生新的入度为 0 的顶点则进队。

判环：队列为空时，处理过的顶点数小于 V

时间复杂度：邻接表 O(V+E)，邻接阵 O(V^2)

使用频度：最常用

优点：由于一般题目拓扑排序都要配合 DP，采用该方法可以方便地进行 DP。

**DFS 并按每个点的完成时间 f[u]逆序排序。**

实现方法：对图 G 运行 DFS，每当一个顶点变黑时就将其插入到链表首部（因为当一个结点变黑时，它的传递闭包均已变黑并插入到链表内，故该点可以安全地加入到链表首部而不破坏拓扑序）。

判环：访问到边(u, v)时，顶点 v 是灰色

时间复杂度：邻接表 O(V+E)，邻接阵 O(V^2)

使用频度：不常用

优点：对一些标号类的题比较适用。

特点：顶点序列呈现块状，即任意一棵子树中所有的结点在拓扑序列中会聚在一起。

**DFS + 回溯**

实现方法：在每层递归中，找到当前所有入度为 0 的顶点，将其分别作为拓扑序列中当前位置的结点并递归下去。

判环：访问到边(u, v)时，顶点 v 是灰色

时间复杂度：不确定

使用频度：较常用

优点：可以输出所有拓扑序

一些习题：

HOJ 1114　Frame Stacking

HOJ 1170　Following Orders

HOJ 1336　Sorting It All Out

HOJ 1352　Mystery
HOJ 1943　平面上的点
HOJ 1992　Dependency Problems
HOJ 2078　Window Pains
HDU 2437　Jerboas
POJ 3249　Test for Job

# 五、强连通分量（有向图）

两种方法：
一、 一遍 DFS（Tarjan 算法）
二、 两遍 DFS
　　第一遍 DFS 为每个顶点标完成时间 f[u]，第二遍在 G’中按 f[u]的降序再对每个点 DFS，这样每个深度优先树中的点就构成了一个 SCC

求有向图的 SCC 还有一种 Gabow 算法，不过不如 Tarjan 实现简单，一般不用。

缩点后重构 SCC 图时，要根据需要判断是否有必要只保留两个 SCC 之间的一条边而去掉重复的。
如何构造一个有向图 G’，使得 G’有着和 G 相同的 SCC 和 SCC 图，且 E’尽可能地小？
只需对 G 中的每个 SCC（假设含 n 个点），在 G’中用 n 条边将其顺次串起来，其他 SCC 间的边不变。可以证明 E’ <= E（因为 G 中每个 SCC 至少含有 n 条边）。

如何确定一个有向图 G=(V, E)是否是半连通的？（半连通的定义是：对任意 u, v，u 可达 v 或者 v 可达 u）
1. 对 G 判弱连通，若不弱连通一定不是半连通的
2. 对每个 SCC 缩点，得到无环图 G’=(V’, E’)
3. 统计 G’中入度为 0 的点的个数，记为 cnt
4. 若 cnt > 1 则 G 一定不是半连通的
5. 以每次寻找入度为 0 的点的方式对 G’进行拓扑排序，若算法结束前某一次迭代之后队列中顶点个数不为 1 则 G 不是半连通的。
6. 拓扑排序正确退出后，G 一定是半连通的。

证：
首先证明一个有向无环的半连通图中一定存在唯一一个源点 s，s 可达 V-{s}中的任意一点。
先证存在性。
设 s 为 V 中的任意一点。若 s 可达 V-{s}中的任意一点，则证毕；否则假设 s 不可达点集 V’中所有的点，其中 V’是 V-{s}的子集。那么对任意 u∈V’，u 一定可达 V-V’中的任意一点。否则的话，设 u∈V’，v∈V-V’且 u 不可达 v。若 v=s，由于 s 也不可达 u，与半连通性矛盾；若 v≠s，则 v∈V-V’-{s}，必有 v 可达 u，从而 s 可达 v 进而可达 u，又与半连通性矛盾。那么问题就变为证明在点集 V’中一定存在某个点 s’，s’可达 V’-{s’}中的任意一点。由于每次 V’的规模至少减小 1，最终一定会变为只含一个点，而该点即为所求。
再证唯一性。
假设还有一个点 s’可达 V-{s’}中的任意一点，那么就有 s 可达 s’且 s’可达 s，则 s 与 s’形成了 SCC，与该图为

无环图矛盾，从而证明 s 是唯一的。
证毕。

那么在第 4 步中，使 cnt=1 的那个点即为可达其他所有点的源点 s。

下面证这个源点 s 的入度为 0。
如若不然，设引起 s 入度不为 0 的点为 u，则由于 s 可达 u 且 u 可达 s，图 G’中产生了环，与 G’是无环图矛盾，从而 s 的入度必为 0。
证毕。

下面证如果图 G 是无环且半连通的，那么当去掉其源点 s 及其所有出边后，得到的图 G’依然是无环且半连通的。
无环显然。下面证半连通：如若不然，设 u, v∈G’且 u 不可达 v，v 不可达 u。现在将顶点 s 及其所有出边加入到图 G’中，即得到 G。由于 indegree(s)=0，u, v 不可能中间经过 s 进而可达对方，于是在 G 中也有 u 不可达 v 且 v 不可达 u，与 G 是半连通图矛盾，从而 G’是半连通的。
证毕。

下面证明算法的 5、6 步是正确的。
开始：显然算法开始前 s 是唯一一个入度为 0 的顶点，将其入队并不会破坏我们的要求。
迭代：当从队列中弹出点 s 并去掉图 G’中点 s 及其所有出边后得到的图 G’’仍然是半连通的，那么必然存在 s’∈G’’，s’是 G’’中唯一一个源点，且 s’一定是去掉了 s 的某条出边后使得其入度为 0 的，故 s’一定会入队且不会再有其他的点入队，从而保证了队列中顶点个数为 1。
结束：拓扑排序结束后，我们得到了图 G’的一个拓扑序列，该序列恰为一条包含了 G’中所有顶点的有向路 p，显然对任意 u, v∈p，若 u 在 v 前则 u 可达 v，否则 v 可达 u，从而说明图 G’是半连通的，进而说明图 G 是半连通的。
证毕。

下面分析该算法的时间复杂性：
1. 判弱连通可采用 DFS 或并查集，复杂度 O(E)
2, 3 可以同时执行，O(E)
5. 拓扑排序 O(V+E)
因此算法的时间复杂度为：O(V+E)

wywcgs 18:38:26
你的第一个证明不太严格。数学证明中混杂着算法的东西（集合不断缩小），感觉不太好。而且，为什么剩下最后一个点，那个点就必然可达所有的点？
我提供一个证明。
首先，对于有向无环图，必然存在拓扑序。设该图的某一个拓扑序是 v1, v2, v3, ...., vn，下面证明 vi 必然可达 V-{v1}。
证明采用数学归纳法。
奠基：n=1, v1 可达空集中的所有元素，必然没有错误。
归纳：令 V_k = { v1, v2, ...., vk }。假设 n=k 时成立，即 v1 可达 V_k-{v1}中的所有点，下证 n = k+1 时也成立。
对于 V_{k+1}-{v1}中的每一个元素 vx，如果 vx∈V_k，则根据归纳假设，必然
成立。否则必有 vx = v_{k+1}。

拓扑序中，由于 1 < k+1，所以 v_{k+1} -> v1 的路不可能存在。而根据半连通定义，必有 v1 -> v_{k+1}的路存在。
也就是说，对于任何 vx∈V_{k+1}-{v1}，均有 v1 可达 vx。
根据归纳法原理，结论成立。

“下面证如果图 G 是无环且半连通的，那么当去掉其源点 s 及其所有出边后，得到的图 G’依然是无环且半连通的。”
这一段。
其实证明的基本思路我感觉并没什么错误，只是在叙述上看着有一点不太好。比如说如度为 0 和 uv 不经过 s 是什么关系。你直接说成任何点均无到达 s 的路径更好一些。
比如说，我推荐这样，你看看是不是更舒服一些。
首先证明任何点 u，不存在 u -> s 的路径。有向无环且 s 可达所有点，显然。
所以，对于任意点对 u, v != s，不妨设 u->v 路径存在。考察这条路径 u -> v_x1 -> v_x2 -> ... -> v，对任意 v_xi，显然有 u 可达 v_xi，故有 v_xi
!= s。所以当去掉 s 与其相关的边之后，该路径仍然存在。故此图仍然半连通。

上面转个修正意义不是太大，因为本质是一样的，只是我感觉这么叙述会好一些。
不知道你看过的数学证明有多少。我说不上来，但是总感觉你的叙述有点别□

一些习题：
POJ 2762　Going from u to v or from v to u?
POJ 1236　Network of Schools
HOJ 1520　The Bottom of a Graph
HOJ 2741　The Busiest Man
POJ 2186　Popular Cows
POJ 3592　Instantaneous Transference
POJ 3687　Labeling Balls

# 六、双连通分量（无向图）

dfn[u]表示顶点 u 的深度优先编号，即发现时间 d[u]。
low[u]表示 u 的所有孩子中从自身或某个后代发出的边所能指到的离根最近的点的 dfn
num[u]表示点 u 属于 BCC 图中的哪一个 BCC（对 V-BCC 而言，由于某个点可能属于多个 V-BCC，故不采用 num 数组，而是用多个 vector<int>记录每个 V-BCC 中包含了哪些点）
stack 栈用来在 DFS 时维护每个 BCC 中的点
割点、点双连通分量（V-BCC）：
若对某条边(u, v)，low[v] >= dfn[u]，则点 u 是割点，stack 中从栈顶到点 u 构成一个 V-BCC，可一边弹栈一边将每个点加入到当前的 vector<int>中。注意要把 u 也加入其中，但不要把 u 弹出，因其可能还属于其他 V-BCC。
桥、边双连通分量（E-BCC）：
若对某条边(u, v)，low[v] > dfn[u]，则边(u, v)是桥，stack 中从栈顶到点 v 构成一个 E-BCC，可一边弹栈一边给每个点标 num 值。注意算法结束后根结点所在的 E-BCC 仍然留在栈中而没有被标号，故必须再将其退栈一一标号。

一些习题：

HOJ 1007　SPF

HOJ 1098　NetWork

HOJ 1789　Electricity

HOJ 2360　Redundant Paths

POJ 2914　Minimum Cut

POJ 2942　Knights of the Round Table

POJ 3177　Redundant Paths

POJ 3352　Road Construction

POJ 3694　Network

# 欧拉（回）路的判定及求解（非平凡图）

无向图：

存在欧拉路：连通且至多有两个奇度顶点（奇度顶点的个数必为偶数）

存在欧拉回路：连通且不存在奇度顶点

求解欧拉（回）路：代码详见模板

**算法思想：**圈套圈，即先递归找到一个主圈（可以证明求欧拉回路时，找到的度为 0 的点必为进入主圈时的第一个点，否则会推出存在奇度顶点而产生矛盾；求欧拉路时，找到的第一个度为 0 的点必为另一个奇度点，之后找到的度为 0 的点同样是进入某个圈的第一个点），然后在回溯过程中一边把主圈上的点入栈，一边把剩下的圈收缩到主圈上的某一点。这个过程是递归的，即被收缩的圈也可能成为主圈而去吸收其他的圈。

**起点的选择：**如果是求欧拉回路，起点任意；否则要选奇度顶点（对有向图而言，选择出度比入度大一的点）作为起点。

有向图：

存在欧拉路：弱连通且只有一个点入度比出度大一，一个点入度比出度小一，其他点入度均等于出度

存在欧拉回路：弱连通且所有点的入度等于出度

求解欧拉（回）路：同无向图

混合图：

存在欧拉路：（暂时搁置）

存在欧拉回路：弱连通且对无向边集存在一组定向，使得所有点的入度等于出度

求解欧拉（回）路：代码详见模板

**算法思想：**

解混合图欧拉回路问题用的是网络流。

把该图的无向边随便定向，计算每个点的入度和出度。如果有某个点出入度之差为奇数，那么肯定不存在欧拉回路。因为欧拉回路要求每点入度 = 出度，也就是总度数为偶数，存在奇数度点必不能有欧拉回路。

好了，现在每个点入度和出度之差均为偶数。那么将这个偶数除以 2，得 x。也就是说，对于每一个点，只要将 x 条边改变方向（入>出就是变入，出>入就是变出），就能保证出 = 入。如果每个点都是出 = 入，

那么很明显，该图就存在欧拉回路。

现在的问题就变成了：我该改变哪些边，可以让每个点出 = 入？构造网络流模型。首先，有向边是不能改变方向的，要之无用，删。一开始不是把无向边定向了吗？定的是什么向，就把网络构建成什么样，边长容量上限 1。另新建 s 和 t。对于入 > 出的点 u，连接边(u, t)、容量为 x，对于出 > 入的点 v，连接边(s, v)，容量为 x（注意对不同的点 x 不同）。之后，察看是否有满流的分配。有就是能有欧拉回路，没有就是没有。欧拉回路是哪个？察看流值分配，将所有流量非 0（上限是 1，流值不是 0 就是 1）的边反向，就能得到每点入度 = 出度的欧拉图。

由于是满流，所以每个入 > 出的点，都有 x 条边进来，将这些进来的边反向，OK，入 = 出了。对于出 > 入的点亦然。那么，没和 s、t 连接的点怎么办？和 s 连接的条件是出 > 入 和 t 连接的条件是入 > 出，那么这个既没和 s 也没和 t 连接的点，自然早在开始就已经满足入 = 出了。那么在网络流过程中，这些点属于“中间点”。我们知道中间点流量不允许有累积的，这样，进去多少就出来多少，反向之后，自然仍保持平衡。

所以，就这样，混合图欧拉回路问题，解了。

一些习题：

HOJ 1294　Sightseeing Tour

# 最小生成树

**算法思想：**

维护一个边集 A，A 是某个 MST 的子集，初始化为空。每次寻找一个不妨碍 A 的割，将轻边加入 A。可运用“剪切”+“粘贴”技术证明这样的轻边是安全边。

轻边是安全边，但安全边不一定是轻边。

动态规划的思想——最优子结构——在这里也有所体现，即 MST 的某个连通块在其导出子图中仍然是 MST。（可用反证法证明）

若 L 是某个 MST T 中所有树边权值的列表，那么该图的其他 MST 所有树边的权值列表也是 L，因为其他 MST 一定在 T 的邻树集中。

若已经求出 MST T，此时减小图中某条边(u, v)的权值，再求 MST。

若(u, v)∈T，则 T 仍是 MST；否则求出 T 中从 u 到 v 的路径上的最大权值 max，并与 w(u, v)比较即可。

理解这两种最常用的算法最根本的思想就可以了，其他就是熟练掌握代码的实现

算法一：Prim

时间复杂度：O(V^2) 适用于稠密图

二叉堆 O(ElogV)

Fib 堆 O(E+VlogV)

若所有边权均为 1..W 的正整数或均匀分布在[0, 1)内，则可实现一个基于 bucket 的 PQ，从而将复杂度降为线性。

算法二：Kruskal

时间复杂度：O(ElogE) 适用于稀疏图
若所有边权均为 1..W 的正整数或均匀分布在[0, 1)内，则可采用线性时间排序从而将复杂度降为线性。
不同的排序结果可能导致不同的 MST。若要指明生成某个特定的 MST T，可如下构造排序后的序列：先将 T 中所有边的权值排序，然后将图中其他边插入到该序列中最后一个它能插入的位置。

算法三：Sollin (Boruvka)
时间复杂度：O(ElogV)
不常用

扩展问题：
MST 唯一性判定：
最稳妥的办法，求出次小生成树并与 MST 比较。Prim 和 Kruskal 算法判唯一性存在一些问题，暂时没有想明白，所以不能用。

最小度限制生成树：
若对每个点都有度限制则是 NP 难的，用搜索解决。
若只对单个点 v0 限制其度为 k，则可以采用贪心策略：
1. 删掉 v0，对剩下的每个连通块求一次 MST 并找一条与 v0 相连的权值最小的边加进来（设此时 v0 的度为 k0），这便是 v0 度最小的 MST
2. 若 k < k0 则无解
3. 做 k – k0 次“差额最小增删操作”，即加入一条与 v0 邻接的边 e1，在形成的环中去掉一条不与 v0 邻接的边 e2，使得 w(e2) – w(e1)最大。

前 k 小生成树
定理：设 T1, T2, …, Tk 为图的前 k 小生成树，则生成图集合{T1, T2, …, Tk}的邻树中的边权和最小者可作为第 k+1 小生成树（可能有和相同的情况）。
依据此定理，可以得出一个复杂度不是很好的算法：
1. 先求出最小生成树，记为 T1
2. 求出 T1 的邻树集 S1，取一个最小者作为 T2
3. 求出 T2 的邻树集 S2，在(S1∪S2) \ {T1∪T2} 中取一个最小者作为 T3
4. 求出 T3 的邻树集 S3，在(S1∪S2∪S3) \ {T1∪T2∪T3} 中取一个最小者作为 T4
5. 重复此过程，直到求出 Tk

最小树形图：
朱-刘算法

最优比率生成树：

一些习题：
《算法导论》P574 23.2-8

# 最短路径

单源最短路径的变体：
单终点：把每条边反向后即为单源
单对顶点：直接单源
每对顶点：Floyd 或者 Dijkstra + 堆
诸如此类的变种一般都可以很容易地转化为我们熟知的问题

最短路径满足最优子结构性质
如果不存在从源 s 可达的负权回路，那么最短路径的定义就是有意义的
最短路径一定是无环的（可分负环和非负环两种情况讨论）
最短路径树的应用（曾经考过，HDU 2433 Travel）

几种算法：
DAG 上的最短路径：按拓扑序对每个点的邻接点进行松弛
DAG 上的最长路径：边权取反，或者初始化为负无穷大并且松弛操作取反
需要求两点间边权乘积最小的路径，可先对边权取 lg，求最短路后再乘幂
若从源 s 出发的边有负权，所有其他边权都非负，且不存在负权回路，那么 Dijkstra 仍然正确
若边权均为 0..W 的非负整数，如何修改 Dijkstra 算法使其运行时间为 O((V + E)logW)？Gabow 定标算法（了解即可，不必掌握）
差分约束系统：构造约束图用 Bellman-Ford 求解。考虑运行过 Bellman-Ford 算法的图 G 中的边(u, v)，有 d[v] <= d[u] + w(u, v)，即 d[v] – d[u] <= w(u, v)，依此不等式构图。
Dijkstra 与 Prim 的算法思想非常相近，所以代码的实现上也是惊人的相似
Dijkstra 不仅可以求最短路，还可以求极小化极大等问题
Djikstra 的证明过程不是很令我信服，我只认为掌握它最核心的思想就可以了：每次确定一个不可能再改变其估计值的点，用它再去调整其他未确定的点的估计值。可以非形式化地在脑海中略想一下是否正确即可

每对顶点间的最短路径——基于动态规划思想
朴素的动态规划：O(V^4)
状态：dp[k][i][j]表示从 i 到 j 至多包含 k 条边的最短路径长
方程：dp[k][i][j] = min{ dp[k – 1][i][j], min{ dp[k – 1][i][l] + w(l, j) } }, l = 1..n
利用矩阵快速取幂可优化到 O(V^3logV)
Floyd-Warshall：O(V^3)
状态：dp[k][i][j]表示从 i 到 j 中间经过的点在集合{1, 2, …, k}中的最短路径长
方程：dp[k][i][j] = min{ dp[k – 1][i][j], dp[k – 1][i][k] + d[k – 1][k][j] }
稀疏图上的 Johnson 算法：O(V^2logV + VE)
运用 Bellman-Ford 算法求出每个顶点的标号 h(u)，对每条边(u, v)重赋权 w’(u, v) = w(u, v) + h(u) – h(v)，对每个点 u 运行 Dijkstra 算法（堆优化）后，对每个点 v，d[u, v] = d’[u, v] – h(u) + h(v)
渐近复杂度要比 Floyd 好，但编程量大大增加，本人认为非迫不得已时一般不用此法

扩展问题：
求网络中的极大化极小或极小化极大问题

求前 K 短路

POJ 2449　Remmarguts' Date

SGU 314　Shortest Paths（TLE）

# 二分图

建模汇总：

【最小覆盖】= 最大匹配数

用最少的点（A 集合或 B 集合的都行）让每条边都至少和其中一个点关联。

【最小路径覆盖】= 顶点数 - 最大匹配数

用最少的不相交简单路径覆盖有向无环图 G 的所有结点（若允许相交，则先求传递闭包）。

把每个点 u 拆成两个，u+在 A 中，u-在 B 中。如果有边<u, v>，则在二分图中引入边<u+, v->。

【最大独立集】= 顶点数 - 最大匹配数

在无向图 G 中选出 M 个点，使这 M 个点两两之间没有边。求 M 的最大值。

如果图 G 满足二分图条件，则建立二分图之后求最大匹配。

【定理】

如果 A 结点和 B 结点一样多，而且所有点的度数都相等，则一定存在一个完美匹配。

《HOJ 1021　Housing Complexes》

【题目大意】

有若干块 M * N 的土地，每块土地上有一些居民。政府要在这些土地上盖 H * W 的房子，每块最多只盖一座。若政府购买某个居民某块土地上的房子，那么保证在这块土地上会把该居民的所有房子全部购买，且不会再买这块土地上其他居民的房子，也不会再买该居民在其他土地上的房子。求政府最多能盖几座房子。

【建模方法】

若某块土地不需要购买居民的房子就可以建政府的房子，那么++ cnt，并不再考虑这块土地。A 集合为土地，B 集合为居民。对每块土地 i，若购买某个居民 j 的土地后便可以盖政府的房子，则连边<i, j>。求最大匹配。

《HOJ 1056　Machine Schedule》

【题目大意】

有两台机器 A、B 和若干件工作。A 有 n 种工作模式 0..n-1，B 有 m 种工作模式 0..m-1，初始时均在 0 模式。每件工作 i 既可以在 A 的 x 模式下完成，也可以在 B 的 y 模式下完成，记为(i, x, y)。机器每转换一次模式就要重新启动。求完成所有工作需要的最少转换次数。

【建模方法】

最小覆盖。A 集合中的点为 A 的所有模式，B 集合中的点为 B 的所有模式。由于初始时 A、B 均在 0 模式，所以所有可在 0 模式下完成的工作都不需要考虑了。对每一组(i, x, y)，若 x, y 均大于 0，则连边<x, y>。求最大匹配。

《HOJ 1086　Don't Get Rooked》

【题目大意】
给出一个 N * N 的棋盘，有些格子是墙，不可穿过。问最多能摆放多少个车，使得它们互不攻击。
【建模方法】
给每个可放置车的格子一个行标号和一个列标号，A 集合为所有的行标号，B 集合为所有的列标号。对每个可放置车的格子，其行标号和列标号对应的点间连边。标号方法（以行为例）：对每一行，把连通可见的格子标成相同的号。求最大匹配。

《HOJ 1335　What's In A Name?》

【题目大意】
有若干个人，每人都有一个真名和一个假名。另有一间屋子，初始时里面没有人。共有三种操作：(1) E name，表示某人进入了屋子，该人的假名是 name。(2) L name，表示某人离开了屋子，该人的假名是 name。(3) M name，表示屋内某个人的真名是 name。给出每个人的真名以及若干操作，要求输出能唯一确定的真名和假名的映射关系。输入保证每个假名都至少出现一次。
【建模方法】
初始时建立完全二分图。每次遇到 M 操作时，把该真名与所有目前不在屋内的假名之间的边去掉。最后枚举去掉每条剩下的边，看是否存在完美匹配，不存在则此边可确定一对真名和假名之间的映射关系。

《HOJ 1624　Sorting Slides》

【题目大意】
有若干张幻灯片，每张上面都标有一个数字。现在它们重叠在一起了，无法分辨哪张幻灯片上是哪个数。要求输出所有可唯一确定的幻灯片与数字的搭配，不存在则输出"none"。
【建模方法】
确定匹配的必需边。A 集合为幻灯片，B 集合为数字，将每张幻灯片与它包含的数字之间连边，求一次最大匹配并保留匹配信息。之后顺次检查每条匹配边是否是必需边，是则输出。检查的方法：去掉该边，看能否再从端点找到增广路，找不到则是必需边。

《HOJ 1708　Taxi Cab Scheme》

【题目大意】
有若干个等待出租车的客户，每辆出租车可以接送多个客户，但要在其出发前 1 分钟赶到才能将其送走。给出每个客户的出发时间及所在位置，求最少需要几辆车便可将所有客户接走。
【建模方法】

最小路径覆盖。O(n^2)枚举任意两个客户可否由一辆车接送，套用最小路径覆盖的框架。

《HOJ 1859　Knights》

【题目大意】
给出一个 N * N 的棋盘，其中 M 个格子已被拿掉而不能放子。求最多能放置几个马，使得它们互不攻击。
【建模方法】
最小覆盖。将棋盘黑白染色，那么马只能从黑格走到白格，或者反过来。故可将黑格放入 A 集合，白格放入 B 集合。不考虑已被拿掉的格子，将能互相攻击的黑、白格子之间连边，求最大匹配。最后结果即为 N * N - M - MaxMatch()。

《HOJ 2077　T-Shirt Gumbo》

【题目大意】
有若干个人和若干件 T-shirt，T-shirt 共有五种型号，每种型号有若干件。每个人有一个他喜欢的型号范围。问是否存在一个分配方案，使得所有人都能穿到喜欢的型号。
【建模方法】
拆点。将每种型号的若干件 T-shirt 拆成一件一件的。

《HOJ 2407　Evacuation》

【题目大意】
给定一个 N * M 的区域，四周是墙，上面嵌有若干个门。内部也有一些墙且不可通过。其余的空地上每个格子初始时都有一个人。人只能从门离开，且门一次只能允许一个人通过，但空地上每个格子可同时容纳多个人。求最少的时间，使得所有人都撤离这个区域。
【建模方法】
A 集合为所有的空地，B 集合为按时间拆点后的门。先 BFS 预处理所有空地到所有门的最短距离，然后由小至大枚举时间 t，不断增加门的数量，若某块空地 i 与某个门 j 满足 d[i][j] <= t，则连边。当 MaxMatch() == 空地数时，当前的 t 即为所求。

《POJ 2724　Purifying Machine》

【题目大意】
给出 2^N 个 01 串中被感染的串，求最少的修复次数使得所有被感染的串复原且不使未被感染的串变坏（修复未被感染的串将导致其变坏）。每次修复既可单独修复某个串，也可同时修复两个只有一位不同的 01 串。
【建模方法】
将所有被感染的串挑出，按含 1 个数的奇偶性分成两个集合，可用一次操作修复的两个串之间连边。结果即为 N + M – MaxMatch()。

《POJ 3163　King of Fighters》

【题目大意】

给出一个 N * M 的获胜概率矩阵，Aij 表示第 i 轮比赛与对方第 j 个角色较量的获胜概率。每一轮对方的角色都不能相同。求最大的获胜概率。

【建模方法】

此题即是求最佳匹配，不过并不是和最大，而是积最大。那么只要先将所有的概率取 lg，KM 算法求出最佳匹配后再取 10 的幂就可以了。

《POJ 3686　The Windy's》

【题目大意】

有 M 个车间和 N 个订单，第 i 个订单在第 j 个车间做花费的时间为 Zij。每个订单只能完整地在某个车间内全部做完。一个车间只有在前一个订单做完之后才能继续做下一个，这之间转换的时间不计。求做完这 N 个订单花费的最小平均时间。

【建模方法】

拆点。若某个车间按顺序执行 k 个任务，且 k 个任务的执行时间分别为 t1, t2…tk，则 k 个任务总的执行时间是 t1*k + t2*(k-1) + t3*(k-2) + … + tk-1*2 + tk。将每个车间拆成 N 个点，第 p 个点表示该车间完成倒数第 p 个任务，连接该点的边权即为-t[i][j] * p。调用 KM 求最小权匹配即可。

《POJ 3692　Kindergarten》

【题目大意】

有 B 个男孩和 G 个女孩，男孩之间两两认识，女孩之间两两认识，部分男孩和女孩也认识。要求找出最多的孩子，使得它们两两认识。

【建模方法】

最小覆盖。A 集合为男孩，B 集合为女孩，在不认识的男、女孩之间连边，求最大匹配。最后结果即为 G + B - MaxMatch()。

《UVa 11383　Golden Tiger Claw》

【题目大意】

给出一个 N * N 的矩阵，每个元素均为 1..100 的正整数。现要求给每行、每列一个标号 row[i], col[j]，使得对矩阵中的每个元素(i, j)，都有 row[i] + col[j] >= w[i][j]。求一个标号方案，使得 sigma(row[i] + col[i]) 最小。

【建模方法】

利用 KM 算法顶标的性质。将每个行标号作为 A 集合中的点，列标号作为 B 集合中的点，w[i][j]为相应边的权值，构建完全二分图。套用 KM 算法，lx[i]和 ly[j]即为所求。

# 网络流

最小割：
求一次最大流，然后在最终的残留网络中以 s 为源 DFS，所有遍历到的结点即组成 S，剩下的点组成 T。

# 2-SAT

k > 2 时，k-SAT 问题是 NPC 的。
2-SAT 问题其实就是考察分析问题和建图的能力。图建好后只需套用模板即可进行如下两项：
（1）判定是否可行
（2）输出一组解

算法：
1. 构图
2. 缩点
3. 判合法
4. 拓扑排序
5. 自底向上进行选择、删除
6. 输出解

建图的经验和技巧：

首先要建立若干个二元组(a, !a)，每个二元组中的两个元素互为对立，必须且只能选择一个。若题目中已存在满足该条件的若干点对，直接利用图中的点即可（如 Peaceful Commision），否则需要另行添加 n 个点，表示原图中每个点的非。

（1）最基础的 AND，OR，XOR 等运算（设 a 为 1，!a 为 0）

a AND b = 1 => (!a -> a), (!b -> b)
a AND b = 0 => (a -> !b), (b -> !a)
a OR b = 1 => (!a -> b), (!b -> a)
a OR b = 0 => (a -> !a), (b -> !b)
a XOR b = 1 => (a -> !b), (!a -> b), (b -> !a), (!b -> a)
a XOR b = 0 => (a -> b), (b -> a), (!a -> !b), (!b -> !a)

（2）一个简单的原则：

当 a, b 两者发生某种冲突时，只需分别考虑从 a, !a, b, !b 这四个点是否有必要发出边去实现某种控制。比如若 a, b 不能同时出现，那么如果 a 出现，只能!b 出现；如果 b 出现，只能!a 出现。!a 或!b 的出现则不会形成任何约束。

一些习题：

HOJ 1917　Peaceful Commission

HOJ 2632　Blanks in a Table

POJ 2723　Get Luffy Out

POJ 2749　Building roads

POJ 3207　Ikki's Story IV - Panda's Trick

POJ 3648　Wedding

POJ 3678　Katu Puzzle

POJ 3683　Priest John's Busiest Day

# 顶点的度序列 Havel 定理

给定一个非负整数序列 D = ( d[1], d[2], ..., d[n] )，若存在一个无向图 G=(V, E)使得图中各点的度与此序列一一对应，则称此序列**可图化**。进一步，若 G 为简单图，则称此序列**可简单图化**。

可图化的判定比较简单：d[1]+d[2]+...d[n]=0(mod2)。关于 G 的构造，我们可以简单地把奇度点配对（奇度点必为偶数个），剩下的全部搞成自环。

可简单图化的判定，有一个 Havel 定理：把序列排成不增序，即 d[1]>=d[2]>=...>=d[n]，则 D 可简单图化当且仅当 D' = ( d[2]-1, d[3]-1, ..., d[d[1]+1]-1, d[d1+2], d[d1+3], ..., d[n] )可简单图化。这个定理写起来麻烦，实际上就是说，我们把 D 排序以后，找出度最大的点(设度为 d[1])，把它和度次大的 d[1]个点之间连边，然后这个点就可以不管了，一直继续这个过程，直到建出完整的图，或出现负度等明显不合理的情况。

定理的简单证明如下：

<= 若 D'可简单图化，我们只需把原图中的最大度点和 D'中度最大的 d[1]个点连边即可，易得此图必为简单图。

=> 若 D 可简单图化，设得到的简单图为 G。分两种情况考虑：

(1) 若 G 中存在边(V[1], V[2]), (V[1], V[3]), ..., (V[1], V[d[1]+1])，则把这些边除去得简单图 G'，于是 D'可简单图化为 G'

(2) 若存在点 V[i], V[j]使得 i < j, (V[1], V[i])不在 G 中，但(V[1], V[j])在 G 中。这时，因为 d[i]>=d[j]，必存在 k 使得(V[i], V[k])在 G 中但(V[j], V[k])不在 G 中。这时我们可以令 G''=G - {(V[i], V[k]), (V[1], V[j])} + {(V[k], V[j]), (V[1], V[i])}。G''的度序列仍为 D，我们又回到了情况(1)。

一些习题：

POJ 1659 - Frogs' Neighborhood

# 弦图

定义：

无向图中，如果任意边数大于 3 的环，至少存在一条边连接环中不相邻的两

个点，则称此图为弦图（Chordal Graph）。

以下是时间复杂度为 O(n+m)的算法，n 是图的点数，m 是图的边数。

第一步：给节点编号

设已编号的节点集合为 A，未编号的节点集合为 B
开始时 A 为空，B 包含所有节点。

```
for num=n-1 downto 0 do
{
    在 B 中找节点 x，使与 x 相邻的在 A 集合中的节点数最多，将 x 编号为 num，
    并从 B 移入 A
}
```

第二步：检查

```
for num=0 to n-1 do
{
    对编号为 num 的节点 x，设所有编号大于 num 且与 x 相邻的节点集合为 C，
    在集合 C 中找出编号最小的节点 y，如果集合 C 中存在不等于 y 的节点 z，
    且 y 与 z 间没有边，则此图不是弦图，退出。
}
```

检查完了，则此图是弦图。

# 其他有趣的问题

Ramsey 问题
任意 6 个人中存在 3 个人，他们要么互相认识，要么互相不认识。
证明：用 6 个点表示这 6 个人，两个人认识则连一条实边，否则连一条虚边。问题变为证明图中存在一个实线三角形或虚线三角形。考虑点 p，它和其他 5 个点的连线中，至少有 3 条同为实线或同为虚线（5 = 5+0 = 4+1 = 3+2），不妨设同为实线，且这三条边的另一端点分别为 u, v, w，若 uvw 组成一个虚线三角形，问题得证；否则必存在一条实线，不妨设为(u, v)，则 puv 形成了实线三角形。
证毕。