## 概要

求A->C最短路，基本思想都是寻找中介点B，若A->B+B->C<A->C，则更新A->C。

当图中有负环时，不存在最短路。

DAG中任意一点出发最短路：dfs+记忆化搜索，边权若相等：bfs。

单源点到其他所有点最短路：Dijkstra（边权不可为负，可以有环），Bellman-Ford和SPFA（边权可负，能检测负环）。

任意节点间最短路：Floyd。

## Dijkstra

Dijkstra算法用来求单源点最短路问题（需保证图中**无负权边**），即可求解从一个点出发到其他所有点的最短路，若图中有负权边则无法求解，因为在选择新的已求解点时，是在通过当前所有已求解点为中介所能达到的最短路的点，但是，如果有负边，该点的最短路径可能还能更小，通过以未求解点为中介，如图所示，假设起点为A，下一个被加入的点为B，最短路长为-1，而实际上应该先加C再加B，最短路长为-3：



而若边权都是正的，再以未求解点为中介不可能更小。

含有动态规划的思想，若A->B最短路为A->x1->x2……->xn->B，显然A->xi最短路必然为A->x1->x2……->xi。Dijkstra算法就是按最短路边数递增的顺序求得源点到各个节点的最短路。

先考虑暴力的方法，假设现在已经求出了源点（设为点A）到若干个点的最短路，这些点看做已标记的，存于集合ans中，剩余未求解的点存于集合tem中，则对于每一个未求解的点，都将每个已求解的点作为中介点，记录下所能达到的最短路径，这时最短路径最短的那个未求解的点（设为点B），其实已经可以算作求解了，它的最短路径就是之前求得的最短路径。因为要达到最短路径，总是需要选取一个点作为中介（源点也存于ans中，也可看作中介点），而该点是以当前所求解的点作为中介点时所能达到的最短路径最短的点，所以该点最终的最短路径，倒数第二个点（中介点）必然应该在当前已求解的点中选取，即选对了。因为若是在未求解的点种选取，任取点X，首先X->B是有距离且大于0的，并且点X以ans中的点作为中介，所能达到的最短路径（设为minX）长于点B的（设为minB），后期minX还是只有可能长与minB，因为新的中介点最短路径是在旧的中介点最短路径上加的，也就是minX+XB永远大于minB，所以B点最短路径就是minB，没有可能再短。

每次都选每一个未求解的点与每一个已求解的点进行最短路更新，既麻烦又耗时，其实在每当确定一个节点的最短路时，更新与之连通的未求解的点的最短路（此最短路只是当前最短路，还未确定）即可。每次将与刚刚求解好的点连通的未求解好的点的最短路都更新后，最短路最短的未求解的点就成为下一个已求解的点，最短这一选取可采用优先队列实现。

注，松弛操作时可以使用path数组记录当前最短路中当前节点的上一节点（不能去记录下一节点，可能是垃圾数据）。

## Bellman-Ford和SPFA

思想一样，SPFA是Bellman-Ford的优化版。

源点与所有其他点的最短路径，构成一棵最短路树，这棵树是原图的子图。

源点为根，每个点都有一个深度，我们每次循环枚举所有的边，尝试进行松弛操作，即对于边A->B，若dis[A]!=INF且dis[A]+边长<dis[B]，就更新B的最短距离，那么我们在第一次循环时，一定就能求出最短路树中，深度为1的节点的最短路，第二次循环时，一定能在深度为1的节点的基础上，进而求出深度为2的节点的最短路，同理第k次循环必能求出深度为k的节点的最短路，这就是一个构造最短路树的过程。

怎么优化呢？Bellman-Ford每次都会遍历所有的边，并发每条边都可能被用上，如果说，有一个点的最短路，在这次循环里，并没有变得更短，那么下一次，从之出发的边，实际上是用不上的，因为这个点最短路没有变短，那么在上一次循环里，它的边能用的就已经用过了，下一次再用也不可能用上了。所以，只有此次循环，距离被更新了的点，从之出发的边，才可能会被用上。拿一个队列取维护这个点集，就是SPFA。每次从队头取出一个点，尝试用从之出发的边进行松弛操作，成功松弛的点，若当前不在队列则再入队，这样，若有一个点入队大于N（顶点数）次，代表原图有环，因为若存在最短路，边数最多为N-1，一个点最多被更新N-1次，即入队N-1次，但因为节点为1时，起初就入队1次，为了统一就设为入队超过N次代表有负环。

注意有负环不代表所有点都会受到负环影响，即从源点到某些点的路径可能都不会经过负环。

## Floyd

易写，效率低，O(N^3)。

尝试以每一个点为中介点，松弛每一对点。

## 机场快线

图中边分成了两类A和B，B类边最多只能走一次，输出起点F到终点T的最短路、所走B类边的起点、最短路距离。

B类边时有限的，在只保留A类边的情况下，求起点与终点分别到所有点的最短路，假设所走B类边起点为x，终点为y，最短距离就是Fx+xy+yT。枚举所有B类边（会走），起点到终点（不走），取最小即可。

和我有仇的一道题，卡了13发才过。UVA在有时明明是PE却显示WA，一直以为是算法错了。题目说每组样例间用空格隔开，调了一上午从WA变成PE，想到是不是最后一组样例不要输出间隔，就让最后一组样例后面什么都不输出，还是PE，最后猜到虽然不用输出间隔的换行，但该行本身后面仍应有换行，然后总是过了，坑爹啊……

## 删边最短路

节点编号为1到N，其中某一条边不能走了，但保证仍有1到N的道路，问最坏情况下的1到N的最短路长度。

先求出1到N的最短路，同时求得最短路径，则只有删了最短路径上的边时才对最短路有影响，枚举每一条有效边即可，求删边后的最大值即为最坏情况下的最优解。

注，邻接矩阵法很容易定位到每条边，但无法表示多重边；邻接表法因为是按读入顺序加的，难以定位到要的边，此处开一个标记数组，那二维就是边的两个端点，标记某条边是否有效，删边时置为无效，松弛操作时只有有效时才进行即可。

## 最短路树应用

可能有重边的无向图，计算每个每个节点到其他节点的最短路长度（若无路径，长度记为L）和，再加起来，记为S。现在可以任意删除一条边，求新的S所能达到的最大值。

对于任意一个节点到其他节点的最短路长度之和，在用Dijkstra算法求得最短路后，将dis数组元素求和即可。而只有删除了最短路树上的边，对该点出发的最短路才可能有影响，所以枚举时只要考虑最短路树上的边即可。