次小生成树

首先使用 Prim 算法计算最小生成树，该部分思路与课本思路相同，

mst 记录了最小生成树的权值。

显然，最小生成树与次小生成树之间应当只相差一条边，即删除最小生成树中的一条边，并为之添加一条新边。

选取能让上述两边权值差最小的边对即可。

为了求解次小生成树，还需要记录以下变量：

$inTree[i][j]$ 表示边 (i,j) 是否在最小生成树中，方便后续计算次小生成树时选边，

$longest[i][j]$ 表示树上结点对 (i,j) 之间简单路径中最长的边的长度。

最终次小生成树的权值为
$mst + \min(weight[i][j] - longest[i][j]), \forall edge(i,j), inTree[i][j] == false$ 。

Prim 算法时间复杂度为 $O(|E| + |V| \log |V|)$ ，统计 $longest$ 数组的过程时间复杂度为 $O(|V|^2)$ ，

总时间复杂度为 $O(|E| + |V|^2)$ 。

可达性查询

考虑到查询次数较多，如果需要支持 $O(1)$ 查询，则需要先将所有点对的可达性信息离线处理好。

而顶点数量过多，强连通分量数量较少，因此需要先缩点，再求各个强连通分量之间的可达性。

使用 Tarjan 缩点，从所有未访问的结点开始做 DFS ，并为每个结点设置一个时间戳 $dfn[u]$ ，用 $low[u]$ 记录结点能到达的时间戳最小的顶点，用一个栈记录 DFS 的顺序。

每次 DFS 后使用子结点更新父节点的 low 值，若当前结点的 low 值与 dfn 值相等，那么该结点以及栈中在该结点之后的所有结点都是可以互相达到的（考虑假设存在一个结点与其他节点不是互相可达的，那么该节点后续的 DFS 一定会更早结束，并将该节点出栈）。

出栈后将其记为一个新的强连通分量。

求出所有强连通分量之后，使用每个结点的边，用动态规划的方法更新可达性矩阵 $acs[i][j]$ 。

Tarjan 对每个边和每个结点进行了一次遍历，对每个结点进行了一次出栈和更新，时间复杂度为 $O(|V| + |E|)$ 。

动态规划求可达性矩阵时间复杂度为 $O(|E| \times SCC\_Count)$ 。

总时间复杂度为 $O(|V| + |E| \times SCC\_Count)$ 。

顶点距离

Floyd-Warshall 版本

使用三重循环，最外层枚举所有顶点，依次使用所有结点松弛所有结点对之间的距离。

两点之间距离存放在矩阵 $dis[i][j]$ 中。

在所有松弛操作结束后，再进行一次额外的松弛，判断是否存在其他结点对可被松弛，如果存在，那么图中含有负环，否则没有负环。

Johnson 版本

首先使用 Bellman Ford 算法计算额外结点（编号为 0）的单源最短路，并判断是否含有负环。其思想为使用所有边，对所有结点对之间的距离松弛结点个数次。在所有松弛操作结束后，判断是否存在仍可松弛的结点对，若存在，那么图中含有负环，否则没有负环。

在 Bellman Ford 算法后，用额外结点 0 的单源最短路更新原图中所有的边，使原图变为无负权图，其后在新图中对每个结点执行 Dijkstra ，求每个结点的单源最短路，最终再用额外结点 0 的单源最短路对新图中的单源最短路更新，得到原图中的单源最短路。

Bellman Ford 算法时间复杂度为 $O(|V||E|)$ ，堆优化的 Dijkstra 算法时间复杂度为 $O((|V|+|E|)\log|E|)$ ，运行 $|V|$ 次。

总时间复杂度为 $O(|V|^2|E|+|V||E|\log|E|)$ 。

最大流

最短可增广路径

由于此处的最短路径将路径的长度均视为单位长度，因此使用 BFS 即可找到一条合法的最短可增广路径。时间复杂度 $O(|V||E|)$ 。

最宽可增广路径

每次沿着已经遍历过的结点向后寻找新结点时，优先选择边最宽的路径进行搜索，这样一来得到的可增广路径的宽度一定是最宽的，此处涉及排序，使用一个堆解决。时间复杂度 $O(|E|\log|f^*|)$ 。

更新残量网络

通过上述两种方式找可增广路径时，均使用 pre 记录每个结点的前驱。

找到可增广路径后，求该路径的最大可增广流量，并且将路径上的正向边减去该流量，反向边增加该流量。

如果无法找到可增广路径，则已经求得了该网络的最大流。