**二分图**

设G=(V,E)是一个无向图，如果顶点V可分为两个不相交的子集(A,B)，并且图中每条边(i,j)所关联的两个顶点i, j分别属于这两个不同的顶点集(i in A, j in B)，则称G为二分图，即属于同一个顶点集的任意两个点没有相邻的边。

染色法判断二分图：

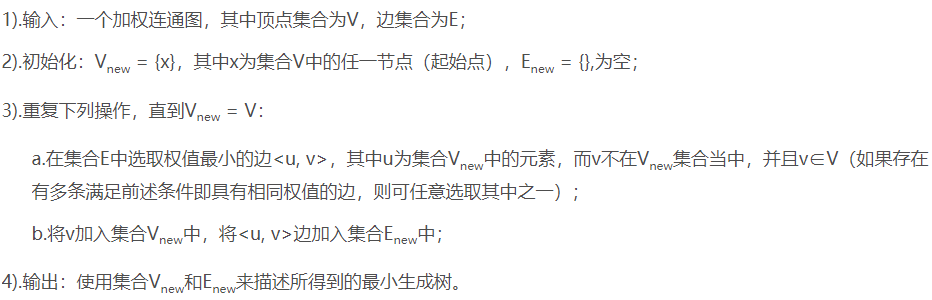
算法思路：

1. 初始选定所有点未染色
2. 随意取出一个未染色的顶点u，把它染成一种颜色
3. 取出与它连接的节点v，如果v未染色则将v染成与u不同的颜色，如果v已经染色，则判断u和v颜色相同则染色失败，不是二分图，结束
4. 遍历所有节点重复步骤3

**最小生成树**

**Prim算法：**

算法核心：在带权连通图中V是所有顶点的集合，U表示已经在最小生成树中的节点，从图中任意一点v开始，此时有U={v}，重复以下操作：找出一点在U，另一点不在U中的边中权值最小的边，将这条边(u,w)加入到已找到的边的集合，并且将点w加入到集合U中。直到U=V时，就得到了最小生成树。



伪代码：

q = PriorityQueue() #优先队列用于找出最小权值的边

visit = set(所有顶点) #标记待访问的节点集

res = 0 #记录最终的权值

q.put((0, 0)) #(weight, point\_id)标记权值和点标号，prim算法从任意节点出发，此处选择0

while visit: #如果存在待访问的节点

dis, pos = q.get()

if pos not in visit:

continue #已经访问的点跳过

visit.remove(pos) #访问过的点移除待访问节点集合

res += dis

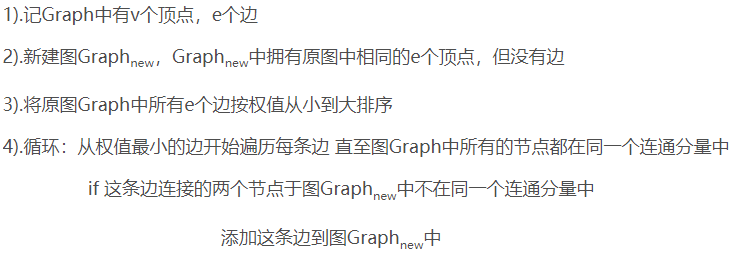
for i in visit: #将当前点到所有未访问点的距离都求一遍加入到队列

q.put((weight, i))

return res

**Kruskal算法**

算法核心：在带权连通图中V是所有顶点的集合，E是所有边的集合，将E按照权值大小排序，从权值最小的边开始遍历，如果该边连接的两个点在已生成的树中不在同一个联通分量，即加入该边不会产生环，则将该边加入到生成树中，并记录该权值。重复上述操作直到得到生成树。在判断是否属于同一个联通分量时使用并查集进行判断。



伪代码：

def find(x, y) #找到x和y的父节点，即联通分量中的代表节点

def union(x, y) #合并x和y节点到同一个连通分量中

q = PriorityQueue()

q.put(所有边的权值和from, to点) #这样得到所有边的优先队列，保证每次取出的是最小权值的边

cnt\_edge = 0 #记录已有的边的个数

res = 0 #记录MST的权值和

while cnt\_edge < n: #n个点的树具有n-1个边

weight, from, to = q.pop()

if find(from) != find(to): #如果from和to不在同一个连通分量上

res += weight

union(from, to)

cnt\_edge += 1

return res