MST(yoj-806) Report

2021201709 李俊霖

基本思路

- 直接使用朴素的**最小生成树**算法，生成的树中的红边数x不一定正好为k条红边。
- 有三种情况，$x < k$，$x > k$，$x = k$。
  - 若$x = k$，满足条件，该生成树符合条件；
  - 若$x < k$，生成树中的红边不够，说明红色边权更大，贪心算法采纳了更多蓝边；因此给红边的边权减去值a，红边比重上升；
  - 若$x > k$，生成树中的红边过多，说明红色边权更小，贪心算法采纳了更多红边；因此给红边的边权加上值a，红边比重下降。
- 基本思路有之后，发现这个需要加上/减去的值是可以根据当前的情况（红边边数）动态调整的，因此考虑采用**二分法**的方式来寻找。
- 二分思路：
  - 初始边界为-101到101，mid为二分中值，每次二分红边的权重加上中值；
  - 初始化集合祖先节点为自己，跑一遍$kruskal$；
  - 红边大于等于需要的，说明红边权重要加一些，左端点右移到mid+1，即 `l = mid + 1` ；更新 `ans = sum - need_r * mid;` ；否则， `r = mid - 1` ；
- 并查集和$kruskal$算法代码如下：

```
int find(int x) //并查集，查
{
    if (fa[x] == x)
        return x;
    return fa[x] = find(fa[x]);
}
```

```
int sum;       //累计权重和
int ans;       //最终权重和
int temp_red;  //当前红边计数
int cnt_e = 0; //当前引入的边数之和
```

```
void kruskal() //最小生成树
{
    sort(e + 1, e + 1 + m, cmp);          // e边从第一个开始
    for (int i = 1; cnt_e != n - 1; i++) //直到贪心地加入了n-1条边
    {
        int x = find(e[i].from); //起点所在集合
        int y = find(e[i].to);
        if (x == y) //判断两端点是否在同一集合内
            continue;
        else if (x != y)
        {
            cnt_e++;
            fa[x] = y; //以终点所在的集合的祖先为祖先
            if (e[i].color == 0)
                temp_red++; //红边数量统计
            sum += e[i].value;
        }
    }
}
```

- **注意**:
  - 每次循环二分加上mid之后要把每一个红边的边权加上的a减掉，否则影响下一次循环 `binary_search` 。
  - $sort$规则中，边权从小到大排序，边权相同时红色优先。

时间复杂度分析

- 并查集找祖先节点的操作时间复杂度为$O(1)$。
- $kruskal$分析：对于一个有m条边和n个顶点的图。在for循环中主要操作是合并不同的连通分量，大循环语句增加m条边的复杂度为$O(m)$，每增加一条边的平均复杂度是$log_2 n$，所以$kruskal$算法的平均时间复杂度为$O(mlog_2 n)$。
- 二分查找由于已经限定了范围，因此复杂度为$O(1)$。