并查集

Disjoint Set Union / Union-Find Set

Idea：相关联的点处于同一集合之中，无关的点处于不同集合之中。用形似单向链表的方式维护集合，使之成为一种树形结构。

Optimization：

- 路径压缩：在查询时顺便使该元素指向该集合的代表元素。
- 按秩合并：合并时将小的集合合并到大的集合中去。

Complexity：

- Worst-Case $O(n)$ 【朴素】
- Worst-Case $O(\lg n)$，Average-Case $O(\alpha(n))$ 【路径压缩】
- $O(\lg n)$ 【按秩合并】
- $O(\alpha(n))$ 【路径压缩+按秩合并】

Code（仅路径压缩）：

```
int fa[N];
int findfa(int x){ return x == fa[x] ? x : fa[x] = findfa(fa[x]); }
inline void unionn(int x, int y){ fa[findfa(y)] = findfa(x); }
```

Code（仅按秩合并，记得初始化 `sz[i]=1`）：

```
int fa[N], sz[N];
int findfa(int x){ return x == fa[x] ? x : findfa(fa[x]); }
inline void unionn(int x, int y){
    x = findfa(x), y = findfa(y);
    if(x == y)  return;
    if(sz[x] < sz[y])   swap(x, y);
    fa[y] = x, sz[x] += sz[y];
}
```

Code（路径压缩+按秩合并，记得初始化 `sz[i]=1`）：

```
int fa[N], sz[N];
int findfa(int x){ return x == fa[x] ? x : fa[x] = findfa(fa[x]); }
inline void unionn(int x, int y){
    x = findfa(x), y = findfa(y);
    if(x == y)  return;
    if(sz[x] < sz[y])   swap(x, y);
    fa[y] = x, sz[x] += sz[y];
}
```