Disjoint Set

Disjoint Set（不相交集合） 是一种数据结构，主要用于处理**集合的合并**和**查询某个元素属于哪个集合**的问题。它通常用于**解决连通性问题**，例如判断两个元素是否属于同一个集合，或者合并两个集合。

📌 Disjoint Set (不相交集合) 的核心概念

概念	解释
集合 (Set)	一组互不相交的元素，每个元素只能属于一个集合
合并 (Union)	将两个不同的集合合并成一个集合
查找 (Find)	查询某个元素属于哪个集合（通常返回集合的代表元/根）
路径压缩 (Path Compression)	优化 `find` 操作，使查询速度更快
按秩合并 (Union by Rank)	优化 `union` 操作，减少树的高度，加快查询速度

The Disjoint Sets data structure has two operations:

● connect(x, y): Connects x and y.
● isConnected(x, y): Returns true if x and y are connected. Connections can be transitive, i.e. they don't need to be direct.

```
JAVA
ds = DisjointSets(7)
ds.connect(0, 1)
ds.connect(1, 2)
ds.connect(0, 4)
ds.connect(3, 5)
ds.isConnected(2, 4): true
ds.isConnected(3, 0): false
ds.connect(4, 2)
ds.connect(4, 6)
ds.connect(3, 6)



public interface
DisjointSets {



/*Connects two items P and
Q. */
void connect(int p, int q);
/*Checks to see if two
items are connected. */
boolean isConnected(int p,
int q);
}
```

Quick Find: Pick Data Structures to Support Tracking of Sets

bad one:

Map<Integer, Integer> -- first number represents set and second represents item
● Slow because you have to iterate to find which set something belongs to.
Assume elements are numbered from 0 to N-1.

Idea #1: List of sets of integers, e.g. [{0}, {1}, {2}, {3}, {4}, {5}, {6}]

● In Java: List<Set>.
● Very intuitive idea.
● Requires iterating through all the sets to find anything.
Complicated and slow!
○ Worst case: If nothing is connected, then isConnected(5, 6) requires
iterating through N-1 sets to find 5, then N sets to find 6.
Overall runtime
of Θ(N).

Idea #2: list of integers where ith entry gives set number (a.k.a. "id") of item i.

● connect(p, q): Change entries that equal id[p] to id[q]
实现 Quick-Find（快速查找） 数据结构的一部分，它是用于实现 Disjoint Sets（不相交集合） 的一种方法。
宝贝，这段代码是实现 Quick-Find（快速查找） 数据结构的一部分，它是用于实现 Disjoint Sets（不相交集合） 的一种方法。妈妈带你详细分析下这个代码的逻辑和性能好不好？✨
Quick-Find 的核心逻辑
目标：
管理多个集合，支持快速查找两个元素是否属于同一个集合（isConnected）和合并两个集合（connect）。

构造函数：QuickFindDS(int N)

- 初始化数组 `id`，每个元素的集合编号等于其索引值：

```
id = new int[N];
for (int i = 0; i < N; i++) {
    id[i] = i;
}
```

方法 1：isConnected(int p, int q)

- 检查两个元素是否属于同一个集合：

```
return id[p] == id[q];
```

- **时间复杂度：**
- 仅需两次数组访问，复杂度是 $\Theta(1)$。

方法 2：connect(int p, int q)

- 合并包含 p 和 q 的两个集合：
  1. 获取 p 和 q 的集合编号：

```
int pid = id[p];
int qid = id[q];
```

2. 遍历整个数组，把属于集合 pid 的所有元素的编号改成 qid：

```
for (int i = 0; i < id.length; i++) {
    if (id[i] == pid) {
        id[i] = qid;
    }
}
```

🛠️ Disjoint Set 的常见实现

1️⃣ 基本数据结构

通常使用 **数组** 或 **哈希表** 来存储集合的关系：
- **每个集合都有一个代表（root）**
- **每个元素存储它的父节点**

```
PYTHON
# 初始化每个元素的父节点为自己
parent = {i: i for i in range(10)}
```

2️⃣ Find 操作（查询集合的代表元）

Find 操作用于找到元素属于哪个集合，通常用**递归**实现：

```
PYTHON
def find(x):
    if parent[x] != x:
        parent[x] = find(parent[x]) # 递归查找并路径压缩
    return parent[x]
```

路径压缩的作用：让所有查找的节点直接指向根，提高查询效率。

3️⃣ Union 操作（合并两个集合）

如果 `find(x) != find(y)`，说明 `x` 和 `y` 在不同的集合，需要合并：

```
PYTHON
def union(x, y):
    rootX = find(x)
    rootY = find(y)
    if rootX != rootY:
        parent[rootY] = rootX # 把 y 的根指向 x 的根
```

为了优化，我们通常使用 **按秩合并 (Union by Rank)**，让树保持尽可能矮，提高效率。

💡 复杂度分析

操作	普通实现	路径压缩优化后
Find	$O(n)$	$O(\alpha(n))$（近乎常数时间）
Union	$O(n)$	$O(\alpha(n))$

其中 $\alpha(n)$ **是阿克曼函数的反函数**，在实际应用中几乎是常数级别，所以 Disjoint Set 非常高效！