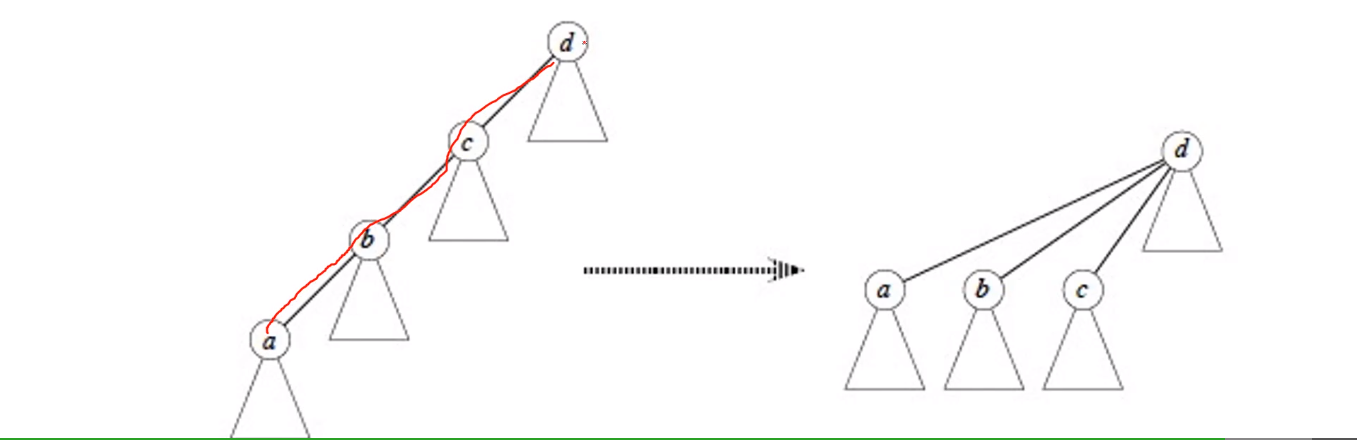
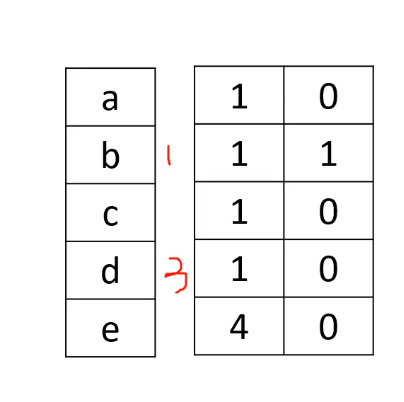
**并查集**

1. 可以用来查看某个事情是否是联通的，主要有两个操作：一个并一个查
2. 用树的双亲表示方法
   1. 那么我们需要通过每个元素的根来确定两个被差元素是否是相连通的
   2. 这样就非常方便我们进行根的查找了
3. 那么我们设计的时候需要有parent（父亲节点），rank（高度）
4. 并的操作：
   1. 基于rank的优化：就是把高度小的树并到高度高的树上
   2. 路径压缩：就是整个森林高度为2，也就是只有一个根其他全是孩子



也就是这样的来操作

1. 查操作
   1. 我们怎么进行查找操作呢
   2. 通过查找元素的parent节点，如果是本身吗，则已经是根了，如果不是则继续进行寻找，直到找到的是本身
2. 代码
   1. 例如要对这样的进行操作
      1. 我们可以看到，有5个
      2. 且初始值得parent都为自身，高度都为0
      3. 下面我们用rank来进行优化的创建并查集



// 基于rank优化的并查集.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

#include <iostream>

#include <vector>

using namespace std;

class Union\_Find\_Set {

struct Node {

int parent{ -1 }; //父亲初始化为-1

int rank{ 0 }; //高度初始化为0

};

vector<Node> forest;

public:

//构造函数

Union\_Find\_Set(const int n) {

forest.resize(n);

for (int i{ 0 }; i < n; i++) {

forest[i].parent = i;

}

}

//查找操作，看元素的根是谁

int Find(int x) {

while (x != forest[x].parent) {

x = forest[x].parent;

}

return x;

}

//合并操作做

void Merger(int a, int b) {

int x = Find(a);

int y = Find(b);

//那我们现在是基于rank的优化，所以我们需要比较两个元素的rank的大小

if (forest[x].rank >= forest[y].rank) {

forest[y].parent = x;

}

else {

forest[x].parent = y;

}

}

};

int main()

{

Union\_Find\_Set disjoin\_set(4);

disjoin\_set.Merger(0, 1);

disjoin\_set.Merger(1, 2);

disjoin\_set.Merger(2, 3);

cout << disjoin\_set.Find(0) << endl;

cout << disjoin\_set.Find(1) << endl;

cout << disjoin\_set.Find(2) << endl;

cout << disjoin\_set.Find(3) << endl;

}