# 【算法学习】左偏树

# 

# **左偏树**

左偏树是一种可以快速合并的可并堆，相对于普通的二叉堆，左偏树的合并可以做到log(p1+p2)log(p\_1 + p\_2)*log*(*p*1​+*p*2​)，是一种十分优秀的数据结构。

# **性质**

左偏树是一种可并堆的实现，是一颗二叉树，它除了有二叉树的左右儿子，还有2个属性，键和距离。下面是左偏树的一些基本性质。

1. 节点的键值小于或等于左右子节点的键值。这是左偏树的堆性质。
2. 节点的左子节点的距离不小于右子节点的距离。
3. 节点的距离等于它的右子节点距离加一。
4. 若左偏树的距离为一定值，则节点数最少的左偏树是完全二叉树。
5. 若一棵左偏树的距离为kk*k*，则这棵左偏树至少有2k+1−12^{k + 1} -12*k*+1−1个节点。
6. 一棵NN*N*个节点的左偏树距离最多为⌊log(N+1)−1⌋\lfloor log(N+1) -1 \rfloor⌊*log*(*N*+1)−1⌋。

# **操作**

以下操作均维护一个小根堆，如果涉及到两个左偏树，那么树根分别为x,yx,y*x*,*y*，否则树根为xx*x*。

数据结构如下：

struct node {

int val, lc, rc, dis, fa;}tree[nmax];

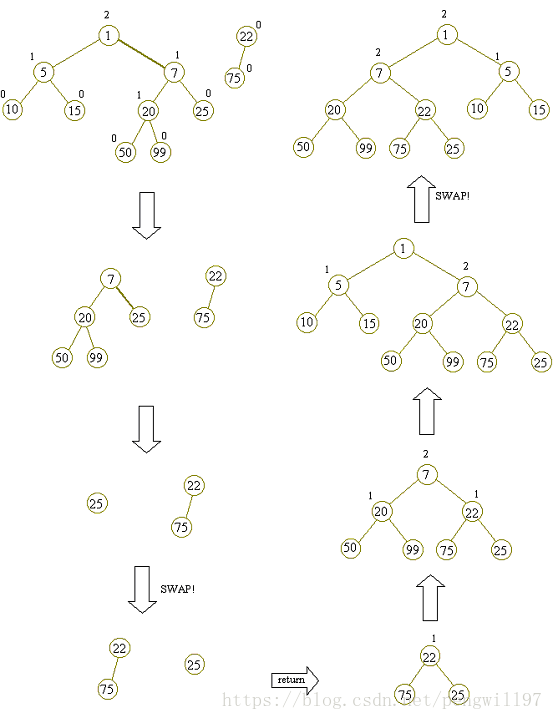
val表示值，lc合rc分别表示左子树合右子树，dis表示节点的距离，fa表示父亲（根据实际情况，有些题目不需要）。

## **合并**

合并是最基本的操作，根据性质1，可以设计出以下算法。

1. 简单情况，如果xx*x*为空树，直接返回yy*y*；如果yy*y*为空树，直接返回xx*x*。
2. 如果不满足1，继续判断。如果xx*x*的根节点val大于yy*y*根节点的val，那么swap(xx*x*, yy*y*)，目的是保证把大的合并到小的上。
3. 将yy*y*的根和xx*x*的右子树判断是否满足2的规则，递归合并，缩小规模，直到一个叶子节点，那么直接修改tree[x].rc=ytree[x].rc = y*tree*[*x*].*rc*=*y*，这样完成合并。
4. 检查合并是否符合规则，在合并过程中可能出现不满足性质2的情况，如果出现了，那么就交换左右子树即可，也就是swap(xx*x*, yy*y*)。
5. 合并完成后，根据性质3，应该将节点的距离变为右子树节点距离加一。
6. 返回合并后树的根。因为是把大的合并到小的上，也就是把yy*y*合并到xx*x*上，返回根xx*x*，完成合并，

可以结合图来理解上述流程：



上述操作的代码如下：

int merge(int x, int y) {

if (x == 0) return y;

if (y == 0) return x;

if (tree[x].val > tree[y].val || (tree[x].val == tree[y].val && x > y) )

swap(x, y);

tree[x].rc = merge(tree[x].rc, y);

tree[tree[x].rc].fa = x;

if (tree[tree[x].rc].dis > tree[tree[x].lc].dis)

swap(tree[x].rc, tree[x].lc);

tree[x].dis = tree[x].rc == 0 ? 0 : tree[tree[x].rc].dis + 1;

return x;}

## 

## **新增节点**

把新增节点当做一颗左偏树，合并到原有树中即可。

上述操作的代码如下：

int add(int val, int x) {

tree[tot].lc = tree[tot].rc = tree[tot].dis = 0;

tree[tot++].val = val;

return merge(tot - 1, x);}

## 

## **删除最小节点**

修改最小节点（树根或堆顶）的信息，即初始化最小节点的左右子树信息，然后将最小节点的左右子树合并即可。

上述操作的代码如下：

int del(int x) {

int l = tree[x].lc, r = tree[x].rc;

tree[x].fa = tree[x].lc = tree[x].rc = tree[x].dis = 0;

tree[x].val = -INF;

tree[l].fa = l, tree[r].fa = r;

return merge(l, r);}

## 

## **左偏树的构建**

初始状态每个节点是一棵树，按照构建二叉堆的方法来构建左偏树。

将所有节点加入FIFO队列，然后依次从队列中取出2个，构建左偏树，再放入队列中。如此往复，直到队列中只有1个元素。

上述操作的代码如下：

int build() {

queue<int> q;

for (int i = 1; i <= n; ++i)

q.push(i);

while (!q.empty()) {

if (q.size() == 1) break;

else {

int x = q.front(); q.pop();

int y = q.front(); q.pop();

q.push(merge(x, y));

}

}

int finally = q.front(); q.pop();

return finally;}

# 

# **举例**

下面以洛谷P3377为例，给出代码。

#include <bits/stdc++.h>

using namespace std;

const int nmax = 1e6 + 7;

const int INF = 0x3f3f3f3f;

struct node {

int val, lc, rc, dis, fa;

}tree[nmax];

int tot = 0;

int n, m;

void init(int x) {

for (int i = 0; i <= x; ++i) {

tree[i].lc = tree[i].rc = tree[i].dis = 0;

tree[i].fa = i;

}}

int merge(int x, int y) {

if (x == 0) return y;

if (y == 0) return x;

if (tree[x].val > tree[y].val || (tree[x].val == tree[y].val && x > y) )

swap(x, y);

tree[x].rc = merge(tree[x].rc, y);

tree[tree[x].rc].fa = x;

if (tree[tree[x].rc].dis > tree[tree[x].lc].dis)

swap(tree[x].rc, tree[x].lc);

tree[x].dis = tree[x].rc == 0 ? 0 : tree[tree[x].rc].dis + 1;

return x;}int findset(int x) {

while (tree[x].fa != x) {

x = tree[x].fa;

}

return x;

}

int add(int val, int x) {

tree[tot].lc = tree[tot].rc = tree[tot].dis = 0;

tree[tot++].val = val;

return merge(tot - 1, x);

}

int del(int x) {

int l = tree[x].lc, r = tree[x].rc;

tree[x].fa = tree[x].lc = tree[x].rc = tree[x].dis = 0;

tree[x].val = -INF;

tree[l].fa = l, tree[r].fa = r;

return merge(l, r);

}

int build() {

queue<int> q;

for (int i = 1; i <= n; ++i)

q.push(i);

while (!q.empty()) {

if (q.size() == 1) break;

else {

int x = q.front(); q.pop();

int y = q.front(); q.pop();

q.push(merge(x, y));

}

}

int finally = q.front(); q.pop();

return finally;

}

int main() {*// freopen("in.txt", "r", stdin);*

scanf("%d %d", &n, &m);

init(n);

for (int i = 1; i <= n; ++i) {

scanf("%d", &tree[i].val);

}

int op, a, b;

for (int i = 1; i <= m; ++i) {

scanf("%d", &op);

if (op == 1) {

scanf("%d %d", &a, &b);

int xx = findset(a), yy = findset(b);

if (tree[a].val == INF || tree[b].val == INF || xx == yy) {

continue;

} else {

merge(xx, yy);

}

} else {

scanf("%d", &a);

if (tree[a].val == -INF) {

printf("-1\n");

} else {

int tmp = findset(a);

printf("%d\n", tree[tmp].val);

del(tmp);

}

}

}

return 0;

}