



東北農業大學  
Northeast Agricultural University

## 《数据结构与算法》课程设计

( 2022 至 2023 学年度 第 1 学期)

课 程 名 称: 数据结构与算法

课 程 编 号: 19600325j

学 生 姓 名: 谢克渊

班 级: 计科 2101

任 课 教 师: 赵语

提 交 日 期: 2021 年 11 月 29 日

成 绩: \_\_\_\_\_

东北农业大学

## 一、 二叉树的遍历

```
1. #include <bits/stdc++.h>
2. using namespace std;
3. typedef long long ll;
4. struct NODE // 树上节点
5. {
6.     int val; // 本节点编号
7.     NODE *s[2];
8. };
9. NODE *rt; // 根节点
10. #define ls(x) x->s[0]
11. #define rs(x) x->s[1]
12. map<int, NODE *> mp; // 由节点编号映射至节点地址
13. NODE *create(void)
14. {
15.     return (NODE *)new NODE;
16. }
17. int n; // n 个节点的二叉树
18.
19. void dfs1(NODE *now) // 先序遍历的递归实现
20. {
21.     printf("%d ", now->val);
22.     if (now->s[0] != NULL)
23.         dfs1(now->s[0]);
24.     if (now->s[1] != NULL)
25.         dfs1(now->s[1]);
26.     return;
27. }
28. void dfs2(NODE *now) // 中序遍历的递归实现
29. {
30.     if (now->s[0] != NULL)
31.         dfs2(now->s[0]);
32.     printf("%d ", now->val);
33.     if (now->s[1] != NULL)
34.         dfs2(now->s[1]);
35.     return;
36. }
37. void dfs3(NODE *now) // 后序遍历的递归实现
38. {
39.     if (now->s[0] != NULL)
40.         dfs3(now->s[0]);
41.     if (now->s[1] != NULL)
42.         dfs3(now->s[1]);
43.     printf("%d ", now->val);
```

```
44.     return;
45. }
46. void query1(void) // 先序遍历的非递归实现
47. {
48.     stack<NODE *> stk;
49.     while (!stk.empty())
50.         stk.pop();
51.     stk.push(rt);
52.     while (!stk.empty())
53.     {
54.         auto now = stk.top();
55.         stk.pop();
56.         if (now == NULL)
57.             continue;
58.         printf("%d ", now->val);
59.         if (now->s[1] != NULL)
60.             stk.push(now->s[1]);
61.         if (now->s[0] != NULL)
62.             stk.push(now->s[0]);
63.     }
64.     return;
65. }
66. void query2(void) // 中序遍历的非递归实现
67. {
68.     stack<NODE *> stk;
69.     while (!stk.empty())
70.         stk.pop();
71.     // puts("AAAA");
72.     auto now = rt;
73.     while (now != NULL || !stk.empty())
74.     {
75.         while (now != NULL)
76.         {
77.             stk.push(now);
78.             now = now->s[0];
79.         }
80.         now = stk.top();
81.         stk.pop();
82.         printf("%d ", now->val);
83.         now = now->s[1];
84.     }
85.     return;
86. }
87. void query3(void) // 后序遍历的非递归实现
88. {
```

```

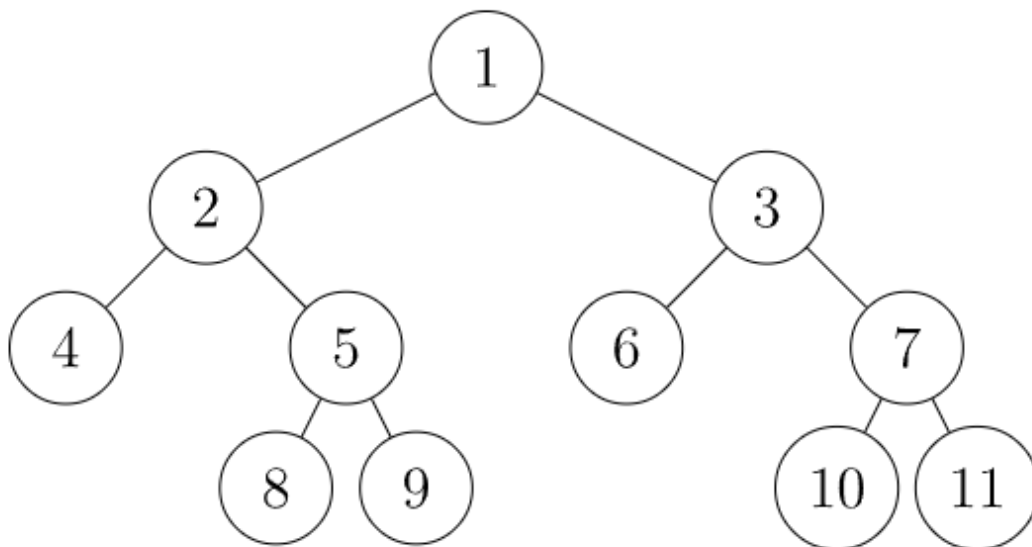
89.     vector<int> res;
90.     stack<NODE *> stk;
91.     while (!stk.empty())
92.         stk.pop();
93.     stk.push(rt);
94.     while (!stk.empty())
95.     {
96.         auto now = stk.top();
97.         stk.pop();
98.         if (now == NULL)
99.             continue;
100.        res.push_back(now->val);
101.        if (now->s[0] != NULL)
102.            stk.push(now->s[0]);
103.        if (now->s[1] != NULL)
104.            stk.push(now->s[1]);
105.    }
106.    reverse(res.begin(), res.end());
107.    for (auto now : res)
108.    {
109.        printf("%d ", now);
110.    }
111.    return;
112. }
113. signed main()
114. {
115.     printf("请输入二叉树的节点个数:");
116.     // scanf("%d",n);
117.     cin >> n;
118.     for (int i = 1; i <= n; ++i)
119.     {
120.         auto now = create();
121.         now->val = i;
122.         now->s[1] = now->s[2] = NULL;
123.         mp[i] = now;
124.     }
125.     mp[0] = NULL;
126.     rt = mp[1];
127.     printf("接下来请根据指引依次输入%d 个节点的信息(如果不存在左儿子或右
        儿子就输入 0):\n", n);
128.     for (int i = 1, l, r; i <= n; ++i)
129.     {
130.         printf("%d 号节点的左儿子和右儿子分别是:  ", i);
131.         cin >> l >> r;
132.         auto now = mp[i];

```

```

133.         auto L = mp[l], R = mp[r];
134.         now->s[0] = L, now->s[1] = R;
135.     }
136.
137.     printf("先序遍历的递归实现结果: \n");
138.     dfs1(rt);
139.     printf("\n 先序遍历的非递归实现结果: \n");
140.     query1();
141.     printf("\n 中序遍历的递归实现结果: \n");
142.     dfs2(rt);
143.     printf("\n 中序遍历的非递归实现结果: \n");
144.     query2();
145.     printf("\n 后序遍历的递归实现结果: \n");
146.     dfs3(rt);
147.     printf("\n 后序遍历的非递归实现结果: \n");
148.     query3();
149.     return 0;
150. }

```



```

请输入二叉树的节点个数:11
接下来请根据指引依次输入11个节点的信息(如果不存在左儿子或右儿子就输入0):
1号节点的左儿子和右儿子分别是: 2 3
2号节点的左儿子和右儿子分别是: 4 5
3号节点的左儿子和右儿子分别是: 6 7
4号节点的左儿子和右儿子分别是: 0 0
5号节点的左儿子和右儿子分别是: 8 9
6号节点的左儿子和右儿子分别是: 0 0
7号节点的左儿子和右儿子分别是: 10 11
8号节点的左儿子和右儿子分别是: 0 0
9号节点的左儿子和右儿子分别是: 0 0
10号节点的左儿子和右儿子分别是: 0 0
11号节点的左儿子和右儿子分别是: 0 0
先序遍历的递归实现结果:
1 2 4 5 8 9 3 6 7 10 11
先序遍历的非递归实现结果:
1 2 4 5 8 9 3 6 7 10 11
中序遍历的递归实现结果:
4 2 8 5 9 1 6 3 10 7 11
中序遍历的非递归实现结果:
4 2 8 5 9 1 6 3 10 7 11
后序遍历的递归实现结果:
4 8 9 5 2 6 10 11 7 3 1
后序遍历的非递归实现结果:
4 8 9 5 2 6 10 11 7 3 1

```

## 二、二叉排序树

- (1) 实现二叉树的检索，若成功，记录检索位置，若不成功，记录父节点位置信息
- (2) 调用检索算法实现插入，并输出插入后的二叉排序树
- (3) 实现删除算法，删除后仍然满足二叉排序树定义，并输出删除后结果

```

1. #include <bits/stdc++.h>
2. using namespace std;
3. const int inf = 1e9 + 10;
4. namespace SPALY //二叉搜索树
5. {
6.     struct NODE
7.     {
8.         int val; // 值
9.         int cnt; // 值为val的元素个数
10.        int size; // 这个节点的子树（包括自身）的节点数量
11.        NODE *s[2], *fa;
12.    };
13.    NODE *Null;
14. #define ls(x) x->s[0]
15. #define rs(x) x->s[1]
16. #define cnt(x) x->cnt
17.    NODE *root; // 根节点
18.    stack<NODE *> stk; // 垃圾箱 删除节点后并不释放该节点内存而是将该节点对应的内存放入垃圾箱重复利用 此举可以节省时间
19.    void pushup(NODE *x) // 更新该节点的子树

```

```

20.     {
21.         if (x == Null)
22.             return;
23.         x->size = ls(x)->size + rs(x)->size + x->cnt;
24.     }
25.     inline void connect(NODE *x, NODE *y, int lr) // 连接两个节点
26.     {
27.         x->fa = y, y->s[lr] = x;
28.     }
29.     inline NODE *create(int val, NODE *fa = Null)
30.     {
31.         NODE *now;
32.         if (stk.empty())
33.             now = (NODE *)new NODE;
34.         else
35.         {
36.             now = stk.top();
37.             stk.pop();
38.         }
39.         now->val = val;
40.         now->s[0] = now->s[1] = Null;
41.         now->size = now->cnt = 1;
42.         // now->fa = fa;
43.         now->fa = fa;
44.         if (fa != Null)
45.         {
46.             connect(now, fa, val > fa->val);
47.             pushup(fa);
48.         }
49.         return now;
50.     }
51.     inline int ide(NODE *x) // 查询 x 是自己父亲的左儿子还是右儿子
52.     {
53.         return (x->fa->s[1]) == x;
54.     }
55.
56.     inline void rotate(NODE *x) // 旋转操作，用来保持二叉搜索树接近平衡
    树
57.     {
58.         auto y = x->fa, z = y->fa;
59.         auto ys = ide(x), zs = ide(y);
60.         // y 是 x 的父亲 z 是 x 的祖父 旋转后整棵树依旧满足二插查找树的性质
61.         connect(x->s[ys ^ 1], y, ys);
62.         connect(y, x, ys ^ 1);
63.         connect(x, z, zs);

```

```

64.     pushup(y);
65.     pushup(z);
66.     return;
67. }
68. void splay(NODE *x, NODE *k) // splay 操作 用于将 x 节点旋转到 k 节点
    下面 k 成为 x 的父亲
69. {
70.     auto y = x->fa;
71.     while (y = x->fa, y != k)
72.     {
73.         auto z = y->fa;
74.         if (z != k && z != Null)
75.         {
76.             if (ide(x) != ide(y))
77.                 rotate(x);
78.             else
79.                 rotate(y);
80.         }
81.         rotate(x);
82.     }
83.     if (k == Null)
84.         root = x;
85.     return;
86. }
87. NODE *findv(int val) // 查找
88. {
89.     auto x = root, y = x;
90.     while (x != Null)
91.     {
92.         y = x;
93.         if (x->val == val)
94.             return x;
95.         else if (val > x->val)
96.             x = rs(x);
97.         else
98.             x = ls(x);
99.     }
100.    return y;
101. }
102. int behind_(NODE *x, int val)
103. {
104.     if (x == Null)
105.         return -inf;
106.     if (x->val >= val)
107.         return behind_(ls(x), val);

```



```

108.
109.         return max(x->val, behind_(rs(x), val));
110.     }
111.     NODE *behind(int val) // 找到 val 的前驱
112.     {
113.         auto now = findv(behind_(root, val));
114.         return splay(now, Null), now;
115.     }
116.
117.     int next_(NODE *x, int val) // 找 val 的后继
118.     {
119.         if (x == Null)
120.             return inf;
121.         if (x->val <= val)
122.             return next_(rs(x), val);
123.         return min(x->val, next_(ls(x), val));
124.     }
125.     NODE *next(int val)
126.     {
127.         auto now = findv(next_(root, val));
128.         return splay(now, Null), now;
129.     }
130.     NODE *insert(int val)
131.     {
132.         auto x = findv(val);
133.         if (x->val != val)
134.         {
135.             create(val, x);
136.         }
137.         else
138.             x->cnt++;
139.         splay(x, Null);
140.         return x;
141.     }
142.     bool delet(int val) // 只删除一个元素
143.     {
144.         auto x = findv(val);
145.         if (x->val != val)
146.             return false; // 删除失败 树中无此元素
147.         if (x->cnt != 1) // 树种有多个该元素 删除一个即可
148.             return x->cnt--, splay(x, Null), true;
149.         else // 树中只有一个该元素
150.         {
151.             auto y = behind(val), z = next(val);
152.             // y 是该点的前驱 z 是该点的后继

```

```

153.             splay(y, Null), splay(z, y); // 将 y 转至树顶使 y 成为根 将
           z 转至 y 底下 使 z 成为 y 的右儿子 此时在 z 的左子树中 仅有 x 一个元素
154.             z->s[0] = Null;             // z 的左儿子(也就是 x)删去
155.             stk.push(x);                // 将 x 扔进垃圾堆
156.             pushup(z);
157.             splay(z, Null);
158.         }
159.         return true;
160.     }
161.     int find_k_v(int k) // 查询第 k 大元素的值
162.     {
163.         auto x = root;
164.         while (k != 0)
165.         {
166.             if (ls(x)->size >= k)
167.                 x = ls(x);
168.             else if (ls(x)->size + x->cnt >= k)
169.                 return x->val;
170.             else
171.                 k -= ls(x)->size + x->cnt, x = rs(x);
172.         }
173.         return x->val;
174.     }
175.     int find_v_k(int val) // 查询值为 val 的元素是第几大元素
176.     {
177.         auto x = findv(val);
178.         splay(x, Null);
179.         return ls(x)->size + 1;
180.     }
181.     void dfs(NODE *x)
182.     {
183.         if (ls(x) != Null)
184.             dfs(ls(x));
185.
186.         for (int i = 1; i <= x->cnt; ++i)
187.         {
188.             printf("%d ", x->val);
189.         }
190.         if (rs(x) != Null)
191.             dfs(rs(x));
192.         return;
193.     }
194.     void debug()
195.     {
196.         printf(" ");

```

```

197.         dfs(root);
198.         puts("");
199.     }
200.     #undef ls
201.     #undef rs
202. };
203. using namespace SPALY;
204.
205. int n;
206. signed main()
207. {
208.     // read(n);
209.     Null = (NODE *)new NODE; //创建一个“空节点”，防止访问到空指针
210.     Null->size = 0, Null->cnt = 0;
211.     Null->val = -1;
212.     Null->fa = Null;
213.     cin >> n;
214.     root = create(-inf);
215.     root->fa = Null;
216.     insert(inf);
217.     // debug();
218.     while (n--)
219.     {
220.         int opt, k;
221.         // read(opt, k);
222.         // puts(" ");
223.         cin >> opt >> k;
224.
225.         if (opt == 1)
226.             insert(k);
227.         else if (opt == 2)
228.             delet(k);
229.         else if (opt == 3)
230.             printf("%d\n", find_v_k(k) - 1);
231.         else if (opt == 4)
232.             printf("%d\n", find_k_v(k + 1));
233.         else if (opt == 5)
234.             printf("%d\n", behind(k)->val);
235.         else if (opt == 6)
236.             printf("%d\n", next(k)->val);
237.         // debug();
238.     }
239.     return 0;
240. }

```

已通过[洛谷 P3369 普通平衡树](#)

洛谷 / 题目列表 / 题目详情

P3369 【模板】普通平衡树

提交答案

加入题单

提交

通过

时间限制

内存限制

239.40k

85.98k

1.00s

128.00MB

题目描述

复制Markdown 展开

您需要写一种数据结构（可参考题目标题），来维护一些数，其中需要提供以下操作：

1. 插入  $x$  数

2. 删除  $x$  数(若有多个相同的数，因只删除一个)

3. 查询  $x$  数的排名(排名定义为比当前数小的数的个数 +1)

4. 查询排名为  $x$  的数

5. 求  $x$  的前驱(前驱定义为小于  $x$ ，且最大的数)

6. 求  $x$  的后继(后继定义为大于  $x$ ，且最小的数)

输入格式

第一行为  $n$ ，表示操作的个数,下面  $n$  行每行有两个数  $opt$  和  $x$ ， $opt$  表示操作的序号( $1 \leq opt \leq 6$ )

输出格式

对于操作 3, 4, 5, 6 每行输出一个数，表示对应答案

输入输出样例

输入 #1

复制

输出 #1

复制

10

1 106465

4 1

1 317721

1 460929

1 644985

1 84185

1 89851

6 81968

1 492737

5 493598

106465

84185

492737

题目提供者

HansBug

难度

提高+/省选-

历史分数

100

提交记录

查看题解

标签

查看算法标签

相关讨论

进入讨论版

查看讨论

推荐题目

查看推荐

洛谷谷出品教材

深入浅出 程序设计竞赛

送官方特典

官方网店

## 二、图

本题使用[洛谷 P3366 【模板】最小生成树](#)来判断代码正确性

P3366 【模板】最小生成树

提交答案

加入题单

提交

通过

时间限制

内存限制

259.95k

92.72k

1.00s

128.00MB

题目描述

复制Markdown 展开

如题，给出一个无向图，求出最小生成树，如果该图不连通，则输出 `orz`。

输入格式

第一行包含两个整数  $N, M$ ，表示该图共有  $N$  个结点和  $M$  条无向边。

接下来  $M$  行每行包含三个整数  $X_i, Y_i, Z_i$ ，表示有一条长度为  $Z_i$  的无向边连接结点  $X_i, Y_i$ 。

输出格式

如果该图连通，则输出一个整数表示最小生成树的各边的长度之和。如果该图不连通则输出 `orz`。

输入输出样例

输入 #1

复制

输出 #1

复制

4 5

1 2 2

1 3 2

1 4 3

2 3 4

3 4 3

7

题目提供者

HansBug

难度

普及-

历史分数

100

提交记录

查看题解

标签

查看算法标签

相关讨论

进入讨论版

查看讨论

推荐题目

查看推荐

最小生成树

(1) Prim 算法

```
1. /* prim 算法实现最小生成树 */
2. #include <bits/stdc++.h>
3. using namespace std;
4. typedef long long ll;
5. typedef unsigned long long ull;
6. typedef double db;
7. typedef pair<int, int> pii;
8. typedef pair<ll, ll> pll;
9. typedef pair<db, db> pdd;
10. #define mpr(x, y) make_pair(x, y)
11.
12. namespace Graph_Chain_forward_star
13. {
14.     const int PN = 5e3 + 10, EN = 5e5 + 10; // 点数 边数
15.     int head[PN], tot;
16.     struct EDGE
17.     {
18.         int to, next, val;
19.     } e[EN];
20.     void add(int u, int v, int w)
21.     {
22.         e[tot] = {v, head[u], w};
23.         head[u] = tot++;
24.     }
25.     void initg(int _n = 0)
26.     {
27.         if (!_n)
28.             memset(head, -1, sizeof head);
29.         else
30.         {
31.             for (int i = 0; i <= _n; ++i)
32.             {
33.                 head[i] = -1;
34.             }
35.         }
36.         tot = 0;
37.     }
38. };
39. using namespace Graph_Chain_forward_star;
40.
41. map<int, bool> vis; // 标记某个点是否被遍历过
42.
43. int ans; // 最小生成树的边权总和
44. int n, m;
```

```

45. priority_queue<pair<int, int>, vector<pii>, greater<pii>> q;
46. // 小根堆 pair(x,y)表示目前未加入最小生成树的点 y 距离最小生成树的最近距离是
    x
47. void prim(int now)
48. {
49.     vis[now] = true;
50.
51.     for (int i = head[now], to, val; ~i; i = e[i].next)
52.     {
53.         to = e[i].to;
54.         val = e[i].val;
55.         if (vis[to])
56.             continue;
57.         q.push(mpr(val, to));
58.     }
59.     while (!q.empty())
60.     {
61.         auto [x, y] = q.top();
62.         q.pop();
63.         if (vis[y])
64.             continue;
65.         ans += x;
66.         prim(y);
67.         break;
68.     }
69. }
70. signed main()
71. {
72.     initg();
73.     cin >> n >> m;
74.     for (int i = 1, u, v, w; i <= m; ++i)
75.     {
76.         cin >> u >> v >> w;
77.         add(u, v, w), add(v, u, w);
78.     }
79.     prim(1);
80.     for (int i = 2; i <= n; ++i)
81.     {
82.         if (!vis[i])
83.         {
84.             puts("orz");
85.             return 0;
86.         }
87.     }
88.     cout << ans << endl;

```

```


89.     return 0;
90. }

```

[测试点信息](#)
[源代码](#)

测试点信息

#1 AC 3ms/684.00KB	#2 AC 4ms/680.00KB	#3 AC 6ms/808.00KB	#4 AC 8ms/684.00KB	#5 AC 26ms/876.00KB	#6 AC 19ms/796.00KB	#7 AC 19ms/712.00KB
#8 AC 555ms/7.32MB	#9 AC 478ms/6.85MB	#10 AC 510ms/7.11MB	#11 AC 3ms/692.00KB	#12 AC 4ms/692.00KB	#13 AC 3ms/692.00KB	



所属题目 P3366 【模板】最小生成树  
 评测状态 Accepted  
 评测分数 100  
 提交时间 2023-01-15 12:06:53

## (2) Kruskal 算法

```

1. #include <bits/stdc++.h>
2. using namespace std;
3. typedef long long ll;
4. typedef unsigned long long ull;
5. typedef double db;
6. typedef pair<int, int> pii;
7. typedef pair<ll, ll> pll;
8. typedef pair<db, db> pdd;
9. #define mpr(x, y) make_pair(x, y)
10.
11. int n, m, ans;
12. namespace DSU_ // 并查集 用来维护某个点是否在最小生成树中
13. {
14.     const int DSUN = 1e5 + 10;
15.     struct DSU
16.     {
17.         int f[DSUN], sz[DSUN];
18.         void init(void)
19.         {
20.             iota(f, f + DSUN, 0);
21.             fill(sz, sz + DSUN, 1);
22.         }
23.         int find(int x)
24.         {
25.             return f[x] = (f[x] == x) ? x : find(f[x]);
26.         }
27.         int operator()(int x)
28.         {
29.             return sz[find(x)];
30.         }
31.         int &operator[](int x)
32.         {
33.             return f[find(x)];

```

```

34.     }
35.     bool operator()(int x, int y)
36.     {
37.         x = find(x), y = find(y);
38.         if (x == y)
39.             return false;
40.         if (sz[x] > sz[y])
41.             swap(x, y);
42.         // x 规模小 y 规模大
43.         sz[y] += sz[x];
44.         f[x] = y;
45.         return true;
46.     }
47. };
48. };
49. using namespace DSU_;
50.
51. struct EDGE
52. {
53.     int u, v, val;
54. };
55.
56. bool operator<(EDGE a, EDGE b) { return a.val < b.val; };
57. bool operator>(EDGE a, EDGE b) { return a.val > b.val; };
58. priority_queue<EDGE, vector<EDGE>, greater<EDGE>> q;
59. // 小根堆
60. signed main()
61. {
62.     cin >> n >> m;
63.     for (int i = 1, u, v, w; i <= m; ++i)
64.     {
65.         cin >> u >> v >> w;
66.         q.push({u, v, w});
67.     }
68.     DSU dsu;
69.     dsu.init();
70.     while (!q.empty())
71.     {
72.         auto [u, v, w] = q.top();
73.         q.pop();
74.         if (dsu(u, v) == false)
75.             continue;
76.         else
77.             ans += w;
78.     }

```



```

79.     int k = dsu(1);
80.     if (k != n)
81.     {
82.         puts("orz");
83.         return 0;
84.     }
85.     else
86.         cout << ans << endl;
87.     return 0;
88. }

```

[测试点信息](#)
[源代码](#)

### 测试点信息

#1 AC 4ms/1.09MB	#2 AC 4ms/1.13MB	#3 AC 6ms/1.25MB	#4 AC 6ms/1.21MB	#5 AC 15ms/1.44MB	#6 AC 12ms/1.34MB	#7 AC 12ms/1.27MB
#8 AC 272ms/3.56MB	#9 AC 224ms/3.46MB	#10 AC 254ms/4.43MB	#11 AC 4ms/1.09MB	#12 AC 4ms/1.08MB	#13 AC 4ms/1.08MB	

所属题目: P3366 【模板】最小生成树

评测状态: Accepted

评测分数: 100

提交时间: 2023-01-15 12:32:02

## 四、排序实现

本题使用[洛谷 P1177 【模板】快速排序](#)来判断代码正确性

### P1177 【模板】快速排序

[提交答案](#)
[加入题单](#)

提交: 551.77k

通过: 169.76k

时间限制: 3.00s

内存限制: 125.00MB

#### 题目描述

利用快速排序算法将读入的  $N$  个数从小到大排序后输出。

快速排序是信息学竞赛的必备算法之一。对于快速排序不是很了解的同学可以自行上网查询相关资料，掌握后独立完成。（C++ 选手请不要试图使用 `STL`，虽然你可以使用 `sort` 一遍过，但是你并没有掌握快速排序算法的精髓。）

#### 输入格式

第 1 行为一个正整数  $N$ ，第 2 行包含  $N$  个空格隔开的正整数  $a_i$ ，为你需要进行排序的数，数据保证了  $a_i$  不超过  $10^9$ 。

#### 输出格式

将给定的  $N$  个数从小到大输出，数之间空格隔开，行未换行且无空格。

#### 输入输出样例

输入 #1

复制

5  
4 2 4 5 1

输出 #1

复制

1 2 4 4 5

题目提供者: 洛谷

难度: 普及-

历史分数: 100

[提交记录](#) [查看题解](#)

标签: [查看算法标签](#)

相关讨论: [进入讨论版](#) [查看讨论](#)

推荐题目: [查看推荐](#)

快速排序：

```

1. #include <bits/stdc++.h>
2. using namespace std;
3. typedef long long ll;
4. typedef unsigned long long ull;
5. typedef double db;

```

```

6. typedef pair<int, int> pii;
7. typedef pair<ll, ll> pll;
8. typedef pair<db, db> pdd;
9. #define mpr(x, y) make_pair(x, y)
10. namespace quick_read // 快速输入快速输出 用来减少程序运行时间
11. {
12.     template <class T>
13.     inline bool read(T &ret)
14.     {
15.         char c;
16.         int sgn;
17.         if (c = getchar(), c == EOF)
18.             return false;
19.         while (c != '-' && (c < '0' || c > '9'))
20.             c = getchar();
21.         ret = (c == '-') ? 0 : (c - '0');
22.         sgn = (c == '-') ? -1 : 1;
23.         while (c = getchar(), (c >= '0' && c <= '9'))
24.             ret = ret * 10 + (c - '0');
25.         ret *= sgn;
26.         return true;
27.     }
28.     template <class T, class... V>
29.     inline bool read(T &a, V &...b)
30.     {
31.         return read(a) && read(b...);
32.     }
33.     template <class T>
34.     inline void qout(T x)
35.     {
36.         if (x < 0)
37.             putchar('-'), x *= -1;
38.         if (x > 9)
39.             qout(x / 10);
40.         putchar(x % 10 + '0');
41.         return;
42.     }
43. };
44. using namespace quick_read;
45.
46. int n, a[1000001];
47. int ran(int l, int r)
48. {
49.     return rand() % (r - l + 1) + l;
50. }

```

```

51. void qsort(int l, int r) // 应用二分思想
52. {
53.     int mid = a[ran(l, r)]; // 中间数
54.     int i = l, j = r;
55.     do
56.     {
57.         while (a[i] < mid)
58.             i++; // 查找左半部分比中间数大的数
59.         while (a[j] > mid)
60.             j--; // 查找右半部分比中间数小的数
61.         if (i <= j) // 如果有一组不满足排序条件（左小右大）的数
62.         {
63.             swap(a[i], a[j]); // 交换
64.             i++;
65.             j--;
66.         }
67.     } while (i <= j); // 这里注意要有=
68.     if (l < j)
69.         qsort(l, j); // 递归搜索左半部分
70.     if (i < r)
71.         qsort(i, r); // 递归搜索右半部分
72. }
73. void work()
74. {
75.     srand(time(0));
76.     read(n);
77.     for (int i = 1; i <= n; i++)
78.         read(a[i]);
79.     qsort(1, n);
80.     for (int i = 1; i <= n; i++)
81.         qout(a[i]), cout << " ";
82.     return;
83. }
84.
85. signed main()
86. {
87.     int TT = 1;
88.     // read(TT);
89.     while (TT--)
90.         work();
91.     return 0;
92. }

```

测试点信息

源代码

测试点信息

#1	AC	#2	AC	#3	AC	#4	AC	#5	AC
4ms/680.00KB		34ms/828.00KB		19ms/736.00KB		20ms/832.00KB		21ms/820.00KB	

serct

所属题目

P1177 【模板】快速排序

评测状态

Accepted

评测分数

100

提交时间

2023-01-15 12:40:40

归并排序:

```
1. #include <bits/stdc++.h>
2. using namespace std;
3. typedef long long ll;
4. typedef unsigned long long ull;
5. typedef double db;
6. typedef pair<int, int> pii;
7. typedef pair<ll, ll> pll;
8. typedef pair<db, db> pdd;
9. #define mpr(x, y) make_pair(x, y)
10. namespace quick_read
11. {
12.     template <class T>
13.     inline bool read(T &ret)
14.     {
15.         char c;
16.         int sgn;
17.         if (c = getchar(), c == EOF)
18.             return false;
19.         while (c != '-' && (c < '0' || c > '9'))
20.             c = getchar();
21.         ret = (c == '-') ? 0 : (c - '0');
22.         sgn = (c == '-') ? -1 : 1;
23.         while (c = getchar(), (c >= '0' && c <= '9'))
24.             ret = ret * 10 + (c - '0');
25.         ret *= sgn;
26.         return true;
27.     }
28.     template <class T, class... V>
29.     inline bool read(T &a, V &...b)
30.     {
31.         return read(a) && read(b...);
32.     }
33.     template <class T>
34.     inline void qout(T x)
35.     {
36.         if (x < 0)
37.             putchar('-'), x *= -1;
```

```

38.         if (x > 9)
39.             qout(x / 10);
40.             putchar(x % 10 + '0');
41.         return;
42.     }
43. };
44. using namespace quick_read;
45.
46. const int N = 1e5 + 10;
47. int a[N], n, b[N];
48. void mergesort(int l, int r)
49. {
50.     if (l >= r)
51.         return;
52.     int mid = l + r >> 1;
53.     mergesort(l, mid), mergesort(mid + 1, r);
54.
55.     int i = l, j = mid + 1, k = l;
56.     while (i <= mid && j <= r)
57.     {
58.         if (a[i] < a[j])
59.             b[k++] = a[i++];
60.         else
61.             b[k++] = a[j++];
62.     }
63.     while (i <= mid)
64.         b[k++] = a[i++];
65.     while (j <= r)
66.         b[k++] = a[j++];
67.     for (int i = l; i <= r; ++i)
68.     {
69.         a[i] = b[i];
70.     }
71. }
72. void work(void)
73. {
74.     read(n);
75.     for (int i = 1; i <= n; ++i)
76.     {
77.         read(a[i]);
78.     }
79.     mergesort(1, n);
80.     for (int i = 1; i <= n; ++i)
81.     {
82.         qout(a[i]), putchar(' ');

```

```
83.     }
84.     return;
85. }
86. signed main()
87. {
88.     int TT = 1;
89.     // read(TT);
90.     while (TT--)
91.         work();
92.     return 0;
93. }
```

测试点信息

源代码

测试点信息

#1	#2	#3	#4	#5
AC	AC	AC	AC	AC
4ms/684.00KB	29ms/1.14MB	17ms/1.19MB	17ms/1.11MB	18ms/1.20MB



所属题目

P1177 [【模板】快速排序](#)

评测状态

Accepted

评测分数

100

提交时间

2023-01-15 12:54:11