20222209257 杨睿妮

Question 1

输入格式:

第一行包括所有测试用例的个数 M，接下来每个用例占三行。

每个用例中，第一行为二叉树层序遍历数组的长度 N，且 $0 \le N \le 10^8$。

第二行为层序遍历的二叉树形式，空结点用 -1 表示，每个结点值在 32 位有符号整型范围内。

第三行为 K，且 $K \ge 0$。

输出格式:

第 K 小的结点值。

解题代码:

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <vector>
#include <cstring>
using namespace std;

const int N = 100000005;
int tree[N];

void inOrder(int &ans, int curIndex, int &cnt, int k, int n) {
    // 进入递归条件判断。只有当：1.当前索引小于总遍历数组长度；2.当前不为空结点；3.还未找到第K小的数字时，才继续进行递归。
    if (curIndex >= n || tree[curIndex] == -1 || cnt >= k) return;
    inOrder(ans, curIndex * 2 + 1, cnt, k, n); // 中序遍历：先递归左子树。
    cnt ++; // 计数，遍历到第cnt小的结点。
    if (cnt == k) ans = tree[curIndex]; // 如果找到第K小结点，则得到最终结果。
    inOrder(ans, curIndex * 2 + 2, cnt, k, n); // 中序遍历：最后递归右子树。
    return;
}

int main() {
    int m;
    cin >> m;
    for (int j = 0; j < m; ++ j) {
        int n;
        cin >> n;
        for (int i = 0; i < n; ++ i) {
            cin >> tree[i]; // 建树，利用下标关系表示树的父子关系。
        }
        int k;
        cin >> k;
        int cnt = 0;
        int ans = -1;
        inOrder(ans, 0, cnt, k, n);
        cout << ans << endl;
    }
    return 0;
}
```

核心思路:

输入数据为二叉树的层次遍历，可以直接用下标索引到每个结点的孩子结点，二叉搜索树的中序遍历可以依次输出整棵树从小到大排序后的结果，所以我们只需对二叉搜索树执行中序遍历，即可找到第 K 小的结点。

该解法需要数组存放二叉树各结点值，所以空间复杂度为 $O(N)$；由于采用中序遍历，时间复杂度也为$O(N)$，其中 N 为遍历的数组长度。

测试用例:

```
4
8
8 5 9 3 6 -1 -1 1
3
8
8 5 9 3 6 -1 -1 1
9
0

1
7
6 5 -1 4 -1 -1 -1
3
```

正确输出：

```
5
-1
-1
6
```

提供 4 个测试用例，分别如下：

1. 普通二叉搜索树，输出第 3 小的值，应当输出 5；
2. 普通二叉搜索树，但 $K > N$，应当输出 -1；
3. 空树，输出 -1；
4. 链状二叉搜索树，即每个结点仅有一个孩子结点，输出第 3 小的值，应当输出 6。

Question 2

输入格式：

第一行包括所有测试用例的个数 M。

每个用例中，第一行为图的长宽 N，且 $N \le 10^3$。

接下来 N 行，每行有 N 个数字，表示每个方格的芝麻数量，取值范围为 $[0, 10^3]$。

输出格式：

最多捡到的芝麻数。

解题代码：

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

const int N = 1001;
int dp[N][N];

int main() {
    int m, n;
    cin >> m;
    for (int i  = 0; i < m; ++ i) {
        cin >> n;
        for (int j = 0; j < n; ++ j) {
            for (int k = 0; k < n; ++ k) {
                cin >> dp[j][k];
            }
        }
        // 考虑边界处的值
        for (int j = 1; j < n; ++ j) {
            dp[0][j] += dp[0][j - 1];
            dp[j][0] += dp[j - 1][0];
        }
        // 每个方格都可以由左边和上边的方格转移过来，取最大值即可
        for (int j = 1; j < n; ++ j) {
            for (int k = 1; k < n; ++ k) {
                dp[j][k] = max(dp[j - 1][k], dp[j][k - 1]) + dp[j][k];
            }
        }
        cout << dp[n - 1][n - 1] << endl;
    }
    return 0;
}
```

核心思路：

从初始格子走到左边界或上边界的格子，能获得的最多芝麻数即为当前格子芝麻数以及前一格子（左边或者右边）能获得的芝麻数之和，从而确定边界条件。对于其他的格子，其能获得的最多芝麻数可以由左边和上边的方格转移过来，取最大值即可。最终可以获得走到右下角格子能获得的最多芝麻数。

该解法需要创建图，空间复杂度为 $O(N^2)$，其中 N 为长宽。时间复杂度为遍历整个图的时间，为 $O(N^2)$。

测试用例：

```
3
3
1 5 1
1 6 1
2 3 1
0
1
5
```

正确输出：

```
16
0
5
```

提供 3个测试用例，分别如下：

1. 题目给定的测试用例，普通图，应当输出 16；
2. 空图，应当输出 0；
3. 只有一个格子的图，应当输出 5。

Question 3

输入格式：

第一行包括所有测试用例的个数 M。

每个用例中，第一行为输入数字个数 N，且 $N \leq 10^8$。

第二行包含 N 个数字，取值范围为 $[-10^9, 10^9]$，保证每行仅有一个独一无二的数字，且其余数字均恰好成对出现。

输出格式：

独一无二的数字。

解题代码：

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main() {
    int m, n, ans;
    cin >> m;
    for (int i = 0; i < m; ++ i) {
        ans = 0;
        cin >> n;
        for (int j = 0; j < n; ++ j) {
            int x;
            cin >> x;
            ans = ans ^ x; // 成对数字异或后等于 0
        }
        cout << ans << endl;
    }
    return 0;
}
```

核心思路：

对于异或运算来说，1. 两个相同的数字异或为 0；2. 异或满足交换律和结合律；3. 任何数字和0异或都等于它自己。

从而我们可以采用对整个数组中的数字进行异或运算，最终结果即为那个独一无二的数字。

该解法不需要额外空间，空间复杂度为 $O(1)$；时间复杂度为对所有数字异或的时间，为 $O(N)$，其中 N 为数组中元素个数。

测试用例：

```
3
5
2 3 1 3 2
5
2 3 6 3 6
1
5
```

正确输出：

```
1
2
5
```

提供 3个测试用例，分别如下：

1. 题目给定的测试用例，常规数组，应当输出 1；
2. 常规数组，应当输出 2；
3. 只包含一个元素的数组，应当输出 5。