

题目给的每个案例都要仔细研究!!!

想到一种能够实现的方法, 先写下来再说 (即使复杂度很高)!!! 先提交一遍, 若超时, 则再进行优化!!

注意题目要求的数据类型, 没说是整数的就用double存(A1070)

注意点

全局变量 (int、bool、string、double)、数组初始化为0

BST中序是递增序列!!! (非常有用)

输入整数时可以带上正负号, 会自动将正负号转换为对应整数

结构体不会自动初始化给定的值, 需要另外单独初始化!!!

读入字符用string!!! 不要用char, 否则会读入空格

- stoi 将字符串转整数

浮点数向上舍入为整数: 只需要+0.5在转int

upper_bound和lower_bound用法

- upper_bound和lower_bound都返回迭代器 (数组指针)

```
1 | int a[maxn];
2 | int i = upper_bound(a, a + maxn, x) - a; // 在[a, a + maxn)中寻找第一个>x的位置
3 | int j = lower_bound(a, a + maxn, x) - a; // 在[a, a + maxn)中寻找第一个>=x的位置
```

STL函数

查找最大元素

```
1 | vector<int> a;
2 | int max = *max_element(a.begin(), a.end());
3 | string s;
4 | char max = *max_element(s.begin(), s.end());
```

排序

vector内元素排序:

```
1 | bool cmp(int a, int b) { return a > b; }; // 递增排序
2 | vector<int> v;
3 | sort(v.begin(), v.end(), cmp);
```

二维vector数组排序:

```
1 | vector<vector<int>> v;
2 | sort(v.begin(), v.end());
```

例:

```
原数组：
10 4 10
10 3 3 6 2
10 3 3 6 2
10 5 2 7
排序后的输出
10 3 3 6 2
10 3 3 6 2
10 4 10
10 5 2 7
```

排好序后记录排名 (重要!!!!!!)

- 相同分数排名相同。

例如排名1, 2, 3, 3, 5不应算作1, 2, 3, 3, 4

```
1 struct Node {
2     int score; // 分数（按此标准排名）
3     int rank; // 排名（排好后记录）
4 };
5 vector<Node> students;
6 int main() {
7     sort(students.begin(), students.end()); // 排序（具体怎么排看题目要求）
8     // 记录排名
9     for(int i = 0; i < students.size(); i++) {
10         if(i >= 1 && students[i].score == students[i-1].score) { // 分数与前一个相同，排名也应相同
11             students[i].rank = students[i-1].rank;
12         }else{
13             students[i].rank = i+1;
14         }
15     }
16     return 0;
17 }
```

以字符串为索引时，可以用map将字符串映射为int下标

复杂的排序思路

1. 查询某一个排序元素在某一区间内的排序结果 (输出年龄在某一区间的财富值排名A1055)

可以先排名，输出时再判断年龄是否符合要求！！

复杂度++做法：每次都找到年龄在给定区间的人进行排序

进一步降低复杂度：根据题目要求的输出量，可以直接把某一年龄的最多可能输出个数放到新数组里，减少最后遍历个数，输出时直接根据新数组。（能够显著降低复杂度！！）

错误原因

格式错误

- 字符串多输出了空格
- 只有一个测试点出现莫名其妙的格式错误：可能需要多输出一行空行 (A1101)

段错误

- 递归没写终止条件
- for循环中 `i++` 写成 `i--`
- 嵌套循环变量冲突
- 下标越界
- 邻接矩阵只初始化了一部分

答案错误

- 实在找不到原因可能因为题目数据超过 `int` 范围, 试试用 `long long` .(A1058)

字符串相关

读入字符串

读入一行字符串，**包含空格**，以回车结束：

```
1 string str;
2 getline(cin, str);
```

读入字符串，**以空格结束**：`cin>>str;`

或：

```
1 char s[50];
2 scanf("%s",s);
```

若 `getline(cin, str)` 前有一行其他输入，必须要先用 `getchar()` **接收上一行的回车**，否则 `getline()` 会读入回车

string类型转字符数组：`str.c_str()`

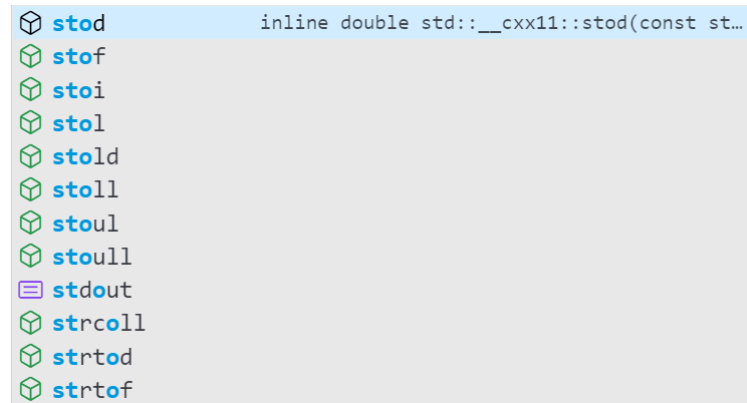
`getchar()` 有返回值，不能空用；

字符串转换

字符串转换成数字

```
1 string str;
2 getline(cin, str);
3 int num = 0; //存放转换后的数字
4 for(int i = 0; i < str.length(); i++){
5     num = num*10 + (str[i]-'0');
6 }
```

string转换函数



`stod` inline double std::__cxx11::stod(const st...

`stof`

`stoi`

`stol`

`stold`

`stoll`

`stoul`

`stoull`

`stdout`

`strcoll`

`strtod`

`strtof`

数字等其他类型转字符串

```
1 string str = to_string(100);
```

字符串转字符数组

```
1 str.c_str();
```

字符数组转各种数据类型（重要！！！！！！）

- `sscanf`和`sprintf`
- `sscanf`从左往右（**字符数组转其他**）；`sprintf`从右往左（**其他转字符数组**）。

例：读入字符串，检查是否是实数。若是实数，检查小数点后是否不超过两位

```
1  int main() {
2      char a[50];
3      double tmp;
4      scanf("%s", a);
5      sscanf(a, "%lf", &tmp); // 将a以double格式存入tmp，若格式不匹配则不修改tmp
6      sprintf(b, "%.2f", tmp); // 将tmp以2位小数格式存入字符串b
7      bool flag = true; // flag = true表明是小数点后小于两位的实数
8      for(int i = 0; i < strlen(a); i++){
9          if(a[i] != b[i]) flag = false; // 如果a和b不严格相等，说明a小数点后位数超过2个
10     }
11     cout << flag << endl;
12     return 0;
13 }
```

提取一行中以一个空格分隔的若干字符串：

使用 `cin` 读入单个字符串，然后用 `getchar()` 接收这个字符串后面的字符。如果是空格，继续读入；如果是换行符，结束读入；

```
1  string str;
2  vector<string> s; //string数组
3  char c; //判断回车的字符
4  while(cin>>str){
5      s.push_back(str);
6      c = getchar();
7      if(c == '\n') break;
8  }
```

读入一行数字，求各位的和 ($n < 1e100$)

```
1  ll sum = 0;
2  while(1) { //一直循环，直到换行符结束
3      char c = getchar();
4      if(c == '\n') break; // 读入的是换行符，结束循环
5      sum += (c - '0');
6  }
7  // cout << sum;
```

字符串或整数输出时一定要注意开头是否要补0！！

C++string 转 C语言字符数组：

- string类中的 `c_str()`：生成一个 `const char*` 指针，指向一空字符终止的数组。
- 由于返回const类型，因此调用此函数后字符串内容不能修改。要想得到非const，可采用如下 `strcpy` 方法：

```
1  int main() {
2
3      string s = "1234";
4      char* c = new char[20];
5      strcpy(c, s.c_str());
6      cout << c << endl;
7      c[2] = 'a'; // c指向的字符数组可修改
8      cout << c << endl;
9
10     return 0;
11 }
12
```

一个字符转字符串（字符串添加一个字符）

```
1 char c = '1';
2 string s = " "; // 该字符串为一个空格
3 s[0] = c; // 将空格改成想要添加的字符
4 ans.insert(0, s); // 在下标0处添加字符串
```

字符串去除前导0

```
1 while(str[0] == '0' && str.size() != 0){
2     str.erase(ans.begin());
3 }
```

反转字符串

```
1 string rev(string s) {
2     reverse(s.begin(), s.end());
3     return s;
4 }
```

- 判断回文串

```
1 bool isPalin(string s) {
2     if(s == rev(s)) return true;
3     return false;
4 }
```

链表

解题步骤

1. 定义静态链表

```
1 struct Node {
2     int address; // 结点地址
3     int data; // 结点数据域
4     int next; // 指针域
5     xxx // 结点的某个性质（视题目而定）
6 }node[maxn];
```

2. 程序的开始，对静态链表初始化。

下面假设xxx表示结点是否在链表上，初始化为0

```
1 for(int i = 0; i < maxn; i++) {
2     node[i].xxx = 0;
3 }
```

3. 根据题目所给的首结点地址，遍历整个链表。同时对xxx标记和统计有效结点个数

```
1 int p = begin, count = 0;
2 while(p != -1) { // -1代表链表结束
3     xxx = 1;
4     count++; // 记录链表有效结点个数
5     p = node[p].next;
6 }
```

4. 排序，将有效结点移至静态链表前面。同时根据题目要求进行二级排序。

```

1  bool cmp(Node a, Node b) {
2      if(a.xxx == -1 || b.xxx == -1) { // 只要有一个无效结点，就把它放到数组后面
3          return a.xxx > b.xxx; // 无效结点的xxx更小
4      } else {
5          // 第二级排序
6      }
7  }

```

要注意特判题目数据全为无效结点的情况!!!

图论

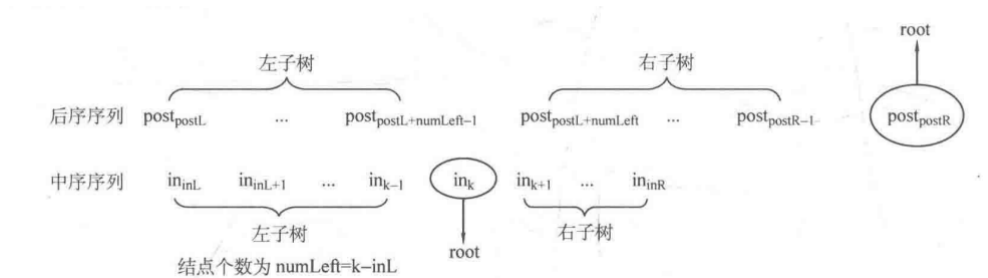
- Dijkstra外层循环n(顶点数)次不能用 while(n--), 只能用 for(int i = 0; i < n; i++) !!! (调试五小时的教训)

原因: 内层循环需要用到顶点数n

- 邻接矩阵一定要记得初始化!!! (调试2h)
- 看见level用BFS
- BFS 判断 if(!inq[v]) 括号内记得 inq[v] = true, 否则程序会死机无输出!!!
- 题目给的图有可能不连通, 要先用DFS判断连通性!!!!!! (A1126)

树

后序+中序建二叉树



```

1  /* 中序+后序建二叉树 */
2  Node* create(int postL, int postR, int inL, int inR) {
3      if(postL > postR) return NULL; // 递归边界
4      Node* root = new Node;
5      root->data = post[postR];
6      int k = inL;
7      for(; k <= inR; k++) { // 找树根下标k
8          if(in[k] == post[postR]) break;
9      }
10     // 此时in[k]为树根结点值, 中序中k左边为左子树[inL, k-1], 右边为右子树[k+1, inR]
11     int numLeft = k - inL; // numLeft用于判断后序中的左右子树区间
12     // 后序中左子树[postL, postL+numLeft-1], 右子树[postL+numLeft, postR-1]
13     root->lchild = create(postL, postL + numLeft - 1, inL, k - 1);
14     root->rchild = create(postL + numLeft, postR - 1, k + 1, inR);
15     return root;
16 }

```

- 要输出结点权值递减的树根到叶子的序列, 可以在读入结点时就对孩子结点递减排序。 (A1053 Path of Equal Weight)

统计最深层的叶子个数

- 由于不需要考虑结点的点权，因此直接用 `vector<int> children[maxn]` 来存储
- 设置叶子个数 `num` 和 `maxLevel`，均初始化为0

以DFS为例：

- 递归边界：当前结点id的子结点个数为0，表示到达叶结点。此时判断层数是否大于最大深度 `maxLevel`。若大于，更新 `maxLevel` 并重置 `num` 为1；若不大于，判断是否等于 `maxLevel`：若等于，`num++`。判断结束返回。
- 递归式：对当前结点id的所有子结点进行递归

```
1  int num = 0;           // 最深叶子个数
2  int maxLevel = 0;      // 最大深度
3  vector<int> children[maxn];
4
5  void DFS(int id, int level) {
6      if(children[id].size() == 0) {
7          if(level > maxLevel) { // 此结点层数更大，更新最大层数，并值num为1
8              maxLevel = level;
9              num = 1;
10         }
11         else if(level == maxLevel) num++; // 层数与最大层数一致，num++
12         return;
13     }
14     for(int i = 0; i < children[id].size(); i++) { // 对所有子结点递归
15         DFS(children[id][i], level + 1);
16     }
17 }
```

找到树中结点个数最大的一层，输出个数和层号

- 由于不需要考虑结点的点权，因此直接用 `vector<int> children[maxn]` 来存储
- 用一个map记录每层结点个数

```
1  map<int, int> mp;    (层数, 个数)
```

- 使用DFS，参数为当前结点和层数，每次递归将当前层号的结点个数+1。

```
1  map<int, int> mp;    // (层数, 个数)
2  void DFS(int id, int level) {
3      mp[level]++;
4      for(int i = 0; i < children[id].size(); i++) {
5          DFS(children[id][i], level + 1);
6      }
7  }
8
9  int main() {
10     DFS(1, 1); // 根结点为1，层数为1
11     int maxnum = 0, ansLevel;
12     for(map<int, int>::iterator it = mp.begin(); it != mp.end(); it++) {
13         if(it->second > maxnum) { // 找到结点最多的一层
14             maxnum = it->second; // 记录结点数量
15             ansLevel = it->first; // 记录层号
16         }
17     }
18     cout << maxnum << " " << ansLevel << endl;
19 }
```

判断是否是完全二叉树

1. 柳神

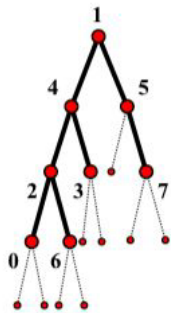
题目大意：给出一个n表示有n个结点，这n个结点为0~n-1，给出这n个结点的左右孩子，求问这棵树是不是完全二叉树

分析：递归出最大的下标值，完全二叉树一定把前面的下标充满：最大的下标值 == 最大的节点数；
不完全二叉树前满一定有位置是空，会往后挤：最大的下标值 > 最大的节点数~

```
1  int n, maxOrder = -1;
2  int last;    // 树的最后一个结点
3  void dfs(int root, int index) {    // 层序遍历,同时与对应完全二叉树序号比较
4      if(index > maxOrder) {
5          maxOrder = index;
6          last = root;
7      }
8      if(node[root].lchild != -1) dfs(node[root].lchild, index*2);
9      if(node[root].rchild != -1) dfs(node[root].rchild, index*2 + 1);
10 }
11
12 dfs(root, 1);
13 if(maxOrder == n) {
14     cout << "YES " << last << endl;
15 }else cout << "NO " << root << endl;
```

2. 晴神

以 Sample Two 的图为例，如果把空结点也标到图中的话，可以形成下面的二叉树，其中虚线和小结点表示实际不存在的空结点。



可以发现，如果按照层次遍历的顺序，并且让空结点也在整个过程中被遍历的话，那么在遍历完 1->4->5->2->3 之后，接下来就会碰到一个空结点，在这个空结点之后才继续遍历完剩余的所有结点，即 7->0->6。可以注意到，在这种遍历过程中，在访问完 N 个非空结点之前就已经碰到了非空结点，因此一定不是完全二叉树，因为对完全二叉树来说，只有当访问完所有 N 个非空结点之后才会访问到非空结点。

由此可以得到完全二叉树的判断方法，即进行层次遍历，并且让空结点也入队，如果在访问完 N 个非空结点之前访问到了空结点，那么说明不是完全二叉树。与此同时可以让一个变量 last 代表二叉树的最后一个结点的编号，不断将其赋值为最后访问到的非空结点即可。

```
1  bool BFS(int root, int& last, int n) {    // n为非空结点个数
2      queue<int> q;
3      q.push(root);
4      while(n) {    // 只要n不为0，即还没有访问完全部非空结点
5          int front = q.front();
6          q.pop();
7          if(front == -1) return false;    // 遍历到空结点，一定不是完全二叉树
8          n--;    // 不是空结点，n--
9          last = front;    // 每次遍历到非空结点就更新last
10         q.push(node[front].lchild);    // 左右结点入队（包括空结点）
11         q.push(node[front].rchild);
12     }
13     return true;    // 连续访问完所有非空结点，即为完全二叉树
14 }
```


BST建树只需要知道前序接着一个个插入就行!!!

英语

单词	翻译
radix	进制
Quadratic Probing	二次探测法
Palindromic Number	回文数
Polynomial	多项式
Parenthesis	圆括号 ()
interval	区间
invert	反转