数据结构 A
作业 12 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：查找
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

下列关键字序列不可能是二叉排序树的查找路径的为（　　）
A.95，22，91，24，94，71
B.92，20，91，34，88，35
C.21，89，77，29，36，38
D.12，25，71，68，33，34

答案：A

解析：对于二叉排列树的任一子树，它的左子树上的节点都大于根，右子树上的节点都小于根。因此可以从前到后根据节点数字逐步缩小范围。

以 A 选项为例，**从 95 开始**，22 小于 95，所以查找范围确定在 $(-\infty, 22]$ 或 $[22, 95]$（分别可能是 22 的左右子树）。

此后遇到 91，所以把范围定在 $[22, 91]$ **或** $[91, 95]$。

接下来遇到 24，发现 24 落在 $[22, 91]$，所以范围确定在 $[22, 24]$ 或 $[24, 91]$。

接下来遇到 94，发现 94 大于 91，无论如何也不会落在 $[22, 24]$ 或 $[24, 91]$，所以 A 选项不可能是二叉排列树的查找路径。

分析 B 选项是同样的，**先确定在** $(-\infty, 20]$ **或** $[20, 92]$，**接着遇到 91**，分裂为 $[20, 91]$ 或 $[91, 92]$，**再遇到 34**，分裂为 $[20, 34]$ 或 $[34, 91]$，**接着 88**，分裂为 $[34, 88]$ 或 $[88, 91]$，**再遇到 35**，落在 $[34, 88]$。查找到最后一个元素的正确位置，所以 B 正确。

分析 C 选项，**先确定在** $[21, 89]$ **或** $[89, \infty)$，**接着遇到 77**，分裂为 $[21, 77]$ 或 $[77, 89]$，**再遇到 29**，分裂为 $[21, 29]$ 或 $[29, 77]$，**接着 36**，分裂为 $[29, 36]$ 或 $[36, 77]$，**再遇到 38**，落在 $[36, 77]$。C 正确。

D 选项同理，**先确定在** $[12, 25]$ **或** $[25, \infty)$，**接着遇到 71**，分裂为 $[25, 71]$ 或 $[71, \infty)$，**再遇到 68**，分裂为 $[25, 68]$ 或 $[68, 71]$，**接着 33**，分裂为 $[25, 33]$ 或 $[33, 68]$，**再遇到 34**，

落在 $[33, 68]$。C 正确。

所以要看当前查找元素是否能在目前可能的范围内找到，如果找不到，那么这个序列就不可能是二叉排列树的查找路径。

习题 2 单选题

下列选项中，错误的是（　　）。

A. 红黑树中，任何一个结点的左右子树高度之差不超过 2 倍

B. AVL 树中，任何一个结点的左右子树高度之差不超过 1

C. 红黑树的查找效率要优于 AVL 树

D. 红黑树和 AVL 树的查找、插入和删除操作的最坏时间复杂度相同

答案：C

解析：排除法，并比较概念。

A 项：红黑树保证了从任一节点到其每个叶子的所有路径包含相同数目的黑色节点，而两个黑色节点之间最多有一个红色节点，因此最长路径不超过最短路径的两倍。

B 项：此选项描述了 AVL 树的一个核心属性，即任何节点的左右子树高度之差不超过 1。这是 AVL 树定义的基础，用于确保树的平衡性，从而优化查找效率。这个描述是正确的。

C 项：AVL 树通常在查找操作上比红黑树更有效。红黑树在插入和删除操作中会进行更少的调整，可能在这些操作上比 AVL 树稍优，但如果要考虑查找效率，因为 AVL 树和红黑树的平衡性不同，并不能做出严格的判断。

D 项：尽管 AVL 树和红黑树在平衡策略上有所不同，但它们都提供 $\log(n)$ 的时间复杂度保证，因为两者都是自平衡的二叉搜索树，可以让树的深度维持在对数级别。

习题 3 单选题

从 19 个元素中查找其中某个元素，如果最多进行 5 次元素之间的比较，则采用的查找方法只可能是（　　）。

A. 折半查找

B. 分块查找

C. 顺序查找

D. 都不可能

答案：A

解析：$n = 19$，折半查找的元素最多比较次数 $= \lceil \log_2(n+1) \rceil = 5$，顺序查找和分块查找所需元素比较次数会更多。

参考教材 9.2.2，折半查找的比较树高度是 $\lceil \log_2(n+1) \rceil = 5$，从根节点出发，刚好在每一个节点处比较一次，因此最多比较次数是 5。

而对于分块查找，一般查找的时间复杂度是 $O(\sqrt{n})$，因为会将整个序列分成约 $\sqrt{n}$ 块，需要先用 $O(\sqrt{n})$ 次左右找到所在块，然后还要具体找到位置，所以最多比较次数会超过 5。

补充说明：对于 $n = 19$，分块查找当把数据分为 4 块或 5 块时，可以把最多查找次数控制在 9。最坏情况下需要先花 $\sqrt{n}$ 次找到所在块，然后再花 $\sqrt{n}$ 找到具体位置。

顺序查找最坏需要用 n 次，因此也不可能。

习题 4 单选题

采用分块查找，如果线性表一共有 625 个元素，查找每个元素的概率相同，假设采用顺序查找来确定结点所在的块，每块分为（　　）个结点最佳

A.9

B.25

C.6

D.625

答案：B

解析：结论是块数取 $\sqrt{n}$ 附近的值，下为分析过程。

采用分块查找的时候，我们的目的是最小化整体查找的平均时间。在这种情况下，我们假设线性表被分成若干个块，并且采用顺序查找来确定元素所在的块，然后在该块内使用顺序查找来找到具体的元素。

如果线性表共有 n 个元素，分成 b 个块，每个块有 k 个元素，其中 $k = n/b$。平均查找时间 T 可以表示为**确定块的时间加上在块内查找的时间**：

$$T = \frac{b}{2} + \frac{k}{2}$$

我们希望最小化这个时间。将 k 代为 n/b 得：

$$T = \frac{b}{2} + \frac{n/b}{2}$$

为了找到最优的 b（即每个块的大小 k），我们对 T 关于 b 进行求导，并令导数等于 0 找到最小值。

$$\frac{\mathrm{d}}{\mathrm{d}b}\left(\frac{b}{2} + \frac{n/b}{2}\right) = 0$$

计算后得：

$$\frac{1}{2} - \frac{n}{2b^2} = 0$$
$$b^2 = n$$
$$b = \sqrt{n}$$

在本题中 $n = 625$，所以 $b = \sqrt{625} = 25$。这表示最佳的情况是每块包含 25 个元素。因此选 B。

另一种理解方式是，查找块的位置要查询 a 次，在块内确定位置要查询 b 次，总查询次数为 $a + b$。而 ab 是恒定情况下，要最小化 $a + b$，显然是基本不等式取最值时最优，也就是 $a = b = \sqrt{ab} = \sqrt{n}$

习题 5 单选题

数据元素有序元素个数较多，且元素固定不变的情况下，应该采用（ ）法

A. 折半查找
B. 分块查找
C. 二叉排序树查找
D. 顺序查找

答案：A

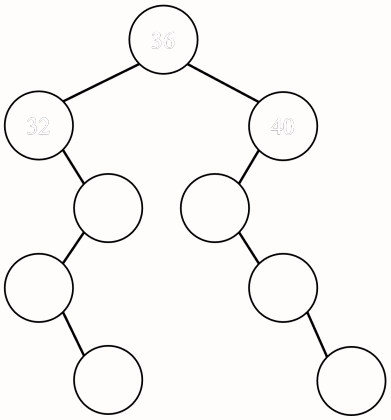
解析：在处理有序且固定不变的大量数据元素时，折半查找（也称为二分查找）是非常高效的查找方法，其时间复杂度为 $O(\log n)$。这是因为折半查找可以在每一步将查找范围缩小到一半，从而大幅度减少比较的次数。

分块查找的效率比折半查找稍低，但较为方便统计其他信息，适用于数据一定程度上动态变化的情况。

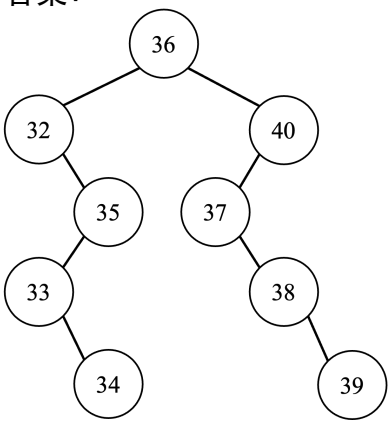
二叉排序树查找的效率相对折半查找不够有保证，顺序查找则效率更低。

习题 6 简答题

一棵二叉排序树的结构如图所示，其中各结点的关键字依次为 32 ~ 40，请标出各结点的关键字。

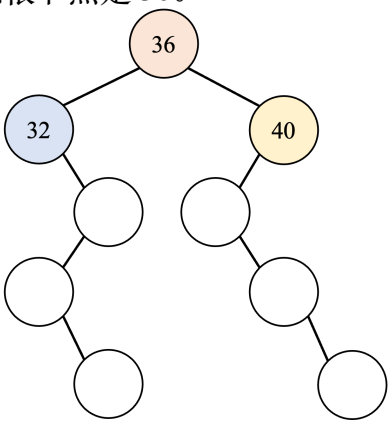


答案：



解析：本题有两种主要的思路可以参考。第一种是递归地看当前节点左子树有多少节点、右子树有多少节点。因为元素都在 $32 \sim 40$ 的范围，所以每次可以确定根节点的值是多少。

也就是在下图中，根节点左边是 4 个节点，说明 $32 \sim 40$ 里有 4 个数字比它小，所以根节点是 36。



另一种方式是先找到最小值。在二叉排序树中，最小值一定是顺着根节点的左儿子一直走下去，直到左儿子为空。所以 32 应该填在图上蓝色位置。

接着可以找到次小值，如果最小值有右儿子，那么最小值就在右子树里面，同样顺着根节点往左走即可；如果没有右儿子，则需要进行回溯。这种方式较为复杂，如果是填空题的话需要多做验证。

同理最大值 40 也可以用这种方式确定出来。

习题 7 求中位数

【问题描述】

给定 n 个整数 x_1，x_2，…，x_n，计算每对整数的差值 $|x_i - x_j|$ $(1 \le i < j \le n)$，可以得到 $\mathrm{C}(n,2)$ 个差值，现在你的任务是尽快找到这些差值的中位数。注意在此问题中，如果差值的个数 m 为偶数，则中位数定义为第 $m/2$ 个最小数，例如 $m = 6$ 时需要求第 3 个最小数。

【输入形式】

在每个测试用例中，第一行给出 n，然后给出 n 个整数，分别代表 x_1，x_2，…，x_n（$x_i \le 1000000000$，$3 \le n \le 100000$）。

【输出形式】

在单独的行中输出中位数。

【样例输入】

```
4
1 3 2 4
```

【样例输出】

```
1
```

【样例说明】

测试数据的文件名为 `in.txt`

【评分标准】

该题目有 10 个测试用例，每通过一个测试得 10 分。

答案：

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;
const int MAXN=1000005;
int x[MAXN];
int n,m;
bool Judge(int mid)      //如果bigcnt<=m/2就满足条件，返回true
{
    int bigcnt=0;      //x+n作为尾端指针减去mid所在的位置，得到的是大于mid的个数
    for(int i=0;i<n;i++)
        bigcnt+= x+n-upper_bound(x+i+1,x+n,x[i]+mid);
```

```
13      return bigcnt<=m/2;
14  }
15  int BinSearch()       //二分查找
16  {   int low=0,high=x[n-1]-x[0];
17      while(high-low>1)     //直到low和high相差1时，停止二分
18      { int mid=(low+high)/2;
19          if (Judge(mid))
20              high=mid;
21          else
22          low=mid;
23      }
24      return high;
25  }
26  int main()
27  {
28      freopen("in.txt","r",stdin);
29      scanf("%d",&n);
30      for(int i=0;i<n;i++)
31          scanf("%d",&x[i]);
32      m=n*(n-1)/2;
33      sort(x,x+n);
34      int ans=BinSearch();
35      printf("%d\n",ans);
36      return 0;
37  }
```

解析：为了确定中位数，我们可以使用二分判断的方式。假设我们检查的值为 *mid*，就可以去判断差值超出 *mid* 的数对有多少个。因此先对输入数据进行排序，这样每个数与其他数的差值大小就更方便计算。使用 upper_bound 可以快速找出与当前数差值超出 *mid* 的数边界位置。最后如果超出 *mid* 的数对大于 $m/2$，就将 *mid* 定为下界，否则定为上界。

习题 8 前 m 大的数

【问题描述】

给定一个包含 N（$N \leq 3000$）个正整数的序列，每个数不超过 5000，对它们两两相加得到 $N \times (N-1)/2$ 个和，求出其中前 M 大的数（$M \leq 1000$）并按从大到小的顺序排列。

【输入形式】

输入可能包含多组数据，其中每组数据包括两行，第一行两个数 N 和 M，第二行 N 个数，表示该序列。

【输出形式】

输入可能包含多组数据，其中每组数据包括两行，第一行两个数 N 和 M，第

二行 N 个数，表示该序列。

【样例输入】

```
4 4
1 2 3 4
```

【样例输出】

```
7 6 5 5
```

【样例说明】

测试数据的文件名为 in.txt

【评分标准】

该题目有 5 个测试用例，每通过一个测试得 20 分。

答案：

```cpp
#include<iostream>
#include<unordered_map>
using namespace std;
#define MAXN 3005       //数据数量最大值
#define MAXV 10001      //数据差值可能的最大值
int a[MAXN];
int main()
{
    int n,m;
    freopen("in.txt","r",stdin);
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        unordered_map<int,int>hmap;//将int作为关键字进行映射
        for(int i=0;i<n;i++)
            cin>>a[i];
        for(int i=0;i<n;i++)
            for(int j=i+1;j<n;j++)
                hmap[a[i]+a[j]]++;
        bool first=true;
        int i=MAXV; //从大到小查询是否有这个元素
        while(i>0&&m>0)
        {
            if(hmap[i]!=0)//如果这个元素出现过，就把出现次数-1，输出一下
            { if(first)
                { cout<<i;
                    first=false;
                }
                else cout<<' '<<i;
                hmap[i]--;
                m--;
```

```
            }
            else i--;
        }
        cout<<endl;
    }
}
```

解析：因为序列长度只有 3000，所以可以用 n^2 次将所有数对的和进行相加。又因为数的大小是不超过 5000 的，相加结果不会超过 10000。那么就从 10000 从大到小找计算结果是否出现，出现过多少次就输出多少次，直到输出够 M 次。

当然，这里基本没有考察查找的知识点，主要是对 `map/unordered_map` 数据结构的应用。如果期末考试要求大家不使用 STL，本题可能需要大家自己写哈希表来完成 `map` 的功能。

本题还有一种思路，就是对所有数进行排序，然后利用二分查找来确定有多少个数对的和大于等于某个值 *mid*，这样可以用减少时间复杂度做到 $O(n \log^2 n)$，感兴趣的同学可以找助教学习这种做法。

总结

题目：查找

日期：2024 年 6 月 21 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：本题主要考察对二叉排序树的查找路径的理解，需要掌握二叉排序树的性质和特点。

习题 2：考察对 AVL 树和红黑树的理解，主要是概念，基本不会考代码。所以要区分这些高级数据结构特点和它们之间的区别。

习题 3：考察对不同查找方法的复杂度掌握情况，此外还需要对查找细节有一些了解，才能确定到具体的数字。期末考试的填空题更有可能考的是数字，所以希望大家能够尽量了解各算法的细节。

习题 4：主要是对分块的理解。其实这个题可以盲猜，或者记住结论，分块就是开根号。

习题 5：考察对不同查找方式的理解。

习题 6：考察对二叉排序树的理解，并希望大家掌握一部分查找、建树的技巧。

习题 7：二分查找有相当多的例题和模型，可以多体会。二分的题目最主要的类型是：**使得** *xxx* **满足条件时**，*yyy* **最小**。这时只需要大家完成检验的 `judge` 函数就可以了。

习题 8：主要考察哈希表的建立或者对 `map` 的掌握。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。