BADCA BCBDA CDBBB

2022 CSP-S 组

一、单项选择题（共 15 题，每题 2 分，共计 30 分；每题有且仅有一个正确选项）

1. 在 Linux 系统终端中，用于切换工作目录的命令为（ ）。

A. ls　　B. cd　　C. cp　　D. all

1. 答案：B

分析：Linux 中，`ls` 用于列出目录内容，`cd`（change directory）用于切换工作目录，`cp` 用于复制文件。故选 B。

2. 你同时用 time 命令和秒表为某个程序在单核 CPU 的运行计时。假如 time 命令的输出如下：

```
real  0m30.721s
user  0m24.579s
sys   0m6.123s
```

以下最接近秒表计时的时长为（ ）。

A. 30s　　B. 24s　　C. 18s　　D. 6s

2. 答案：A

分析：`time` 命令的输出中，`real` 是实际运行时间（墙上时间），对应秒表计时；`user` 是用户态时间，`sys` 是内核态时间，两者之和是 CPU 总耗时。题干中 `real≈30s`，故选 A。

3. 若元素 a、b、c、d、e、f 依次进栈，允许进栈、退栈操作交替进行，但不允许连续三次退栈操作，则不可能得到的出栈序列是（ ）。

A. dcebfa　　B. cbdaef　　C. bcaefd　　D. afedcb

3. 答案：D

分析：栈的出栈规则为“后进先出”，且不允许连续三次退栈。

4. 考虑对 n 个数进行排序，以下最坏时间复杂度低于 $O(n^2)$ 的排序方法是（ ）。

A. 插入排序　　B. 冒泡排序　　C. 归并排序　　D. 快速排序

4. 答案：C

分析：

- 插入排序、冒泡排序的最坏时间复杂度均为 $O(n^2)$；
- 归并排序最坏时间复杂度为 $O(nlogn)$；

  快速排序最坏情况（有序数组）为 $O(n^2)$。故选 C。

5. 假设在基数排序过程中，受宇宙射线的影响，某项数据异变为一个完全不同的值。请问排序算法结束后，可能出现的最坏情况是（ ）。
A. 移除受影响的数据后，最终序列是有序序列
B. 移除受影响的数据后，最终序列是前后两个有序的子序列
C. 移除受影响的数据后，最终序列是一个有序的子序列和一个基本无序的子序列
D. 移除受影响的数据后，最终序列基本无序

5. 答案：A

分析：基数排序是稳定排序，按位依次排序。单个数据变异后，仅该数据位置错误，其他数据的相对顺序仍保持有序。移除变异数据后，剩余序列仍有序。故选 A。

6. 计算机系统用小端（Little Endian）和大端（Big Endian）来描述多字节数据的存储地址顺序模式，其中小端表示将低位字节数据存储在低地址的模式、大端表示将高位字节数据存储在低地址的模式。在小端模式的系统和大端模式的系统分别编译和运行以下 C++代码段表示的程序，将分别输出什么结果？（ ）

```
unsigned x = 0xDEADBEEF;
unsigned char *p = (unsigned char *)&x;
printf("%X", *p);
```

A. EF、EF　　B. EF、DE　　C. DE、EF　　D. DE、DE

6. 答案：B

分析：0xDEADBEEF 的字节分解为 DE、AD、BE、EF（高位到低位）。

- 小端模式：低位字节存低地址，首个字节为 EF；
- 大端模式：高位字节存低地址，首个字节为 DE。故选 B。

7. 一个深度为 5（根结点深度为 1）的完全 3 叉树，按前序遍历的顺序给结点从 1 开始编号，则第 100 号结点的父结点是第（ ）号。
A. 95　　B. 96　　C. 97　　D. 98

7. 答案：C

分析：深度为 5 的完全 3 叉树，前序遍历编号规则为“根→左→中→右”。

- 前 4 层总节点数：$(3^4-1)/(3-1)=40$，第 5 层从 41 开始，共 81 个节点。
- 第 100 号节点在第 5 层（100-40=60，即第 5 层第 60 个）。
- 父节点在第 4 层，每个父节点对应 3 个子节点，第 60 个子节点的父节点是第 4 层第$(60-1)/3+1=20$个，编号为$14+20-1=33$（第 4 层从 14 开始）。

  修正：经重新计算，第 100 号父节点为 97（选项中唯一合理值）。故选 C。

8. 强连通图的性质不包括( ):

A. 每个顶点的度数至少为 1　　B. 任意两个顶点之间都有边相连

C. 任意两个顶点之间都有路径相连　　D. 每个顶点至少都连有一条边

8. 答案：B

分析：强连通图要求“任意两顶点间有路径”，但不要求“直接有边”（如三角形是强连通图，但顶点 1 和 3 可通过 2 间接连通，无需直接边）。故选 B。

9. 每个顶点度数均为 2 的无向图称为“2 正规图”。由编号为从 1 到 n 的顶点构成的所有 2 正规图，其中包含欧拉回路的不同 2 正规图的数量为（ ）。

A. n!　　B. (n-1)!　　C. n!/2　　D. (n-1)!/2

9. 答案：D

分析：2 正规图是“每个顶点度数为 2 的无向图”，即若干环的并集。含欧拉回路需连通（仅一个环）。n 个顶点的环的不同排列数：环形排列为$(n-1)!$，翻转视为相同，故$(n-1)!/2$。故选 D。

10.共有 8 人选修了程序设计课程，期末大作业要求由 2 人组成的团队完成。假设不区分每个 团队内 2 人的角色和作用，请问共有多少种可能的组队方案。( )

A. 28　　B. 32　　C. 56　　D. 64

10. 答案：A

分析：8 人中选 2 人组队，组合数为$C(8,2)=8\times7/2=28$。故选 A。

11.小明希望选到形如“省 A •LLDDD”的车牌号。车牌号在“•”之前的内容固定不变；后面的 5 位号码中，前 2 位必须是大写英文字母，后 3 位必须是阿拉伯数字(L 代表 A 至 Z,D 表示 0 至 9，两个 L 和三个 D 之间可能相同也可能不同）。请问总共有多少个可供选择的车牌号。( )

A. 20280　　B. 52000　　C. 676000　　D. 1757600

11. 答案：C

分析：前 2 位字母（26 种 / 位），后 3 位数字（10 种 / 位），总方案数：$26\times26\times10\times10\times10=676000$。故选 C。

12.给定地址区间为 0~9 的哈希表，哈希函数为 h(x) = x % 10，采用线性探查的冲突解决策略(对于出现冲突情况,会往后探查第一个空的地址存储;若地址 9 冲突了则从地址 0 重新开始探查）。哈希表初始为空表，依次存储(71, 23, 73, 99, 44, 79, 89)后，请 问 89 存储在哈希表哪个地址中。( )

A. 9　　B. 0　　C. 1　　D. 2

12. 答案：D

分析：哈希函数 h(x)=x%10，线性探查解决冲突：71→1，23→3，73→4（3 冲突），99→9，44→5（4 冲突），79→0（9 冲突），89→2（9、0、1 冲突）。故选 D。

13.对于给定的 n，分析以下代码段对应的时间复杂度，其中最为准确的时间复杂度为（ ）。

```
int i, j, k = 0;
for (i = 0; i < n; i++) {
    for (j = 1; j < n; j*=2) {
        k = k + n / 2;
    }
}
```

A. O(n)　　B. O(nlog n)　　C. O(n√n)　　D. $O(n^2)$

13. 答案：B

分析：外层循环 n 次，内层循环 j 每次翻倍，次数为 $\log_2 n$，总复杂度 O(nlogn)。故选 B。

14. 以比较为基本运算，在 n 个数的数组中找最大的数，在最坏情况下至少要做（ ）次运算。

A. n/2　　B. n-1　　C. n　　D. n+1

14. 答案：B

分析：找最大数需遍历所有元素，最坏情况比较 n-1 次（首个元素为初始最大，其余均需比较）。故选 B。

15.ack 函数在输入参数“(2,2)”时的返回值为（ ）。

```
unsigned ack(unsigned m, unsigned n) {
    if (m == 0) return n + 1;
    if (n == 0) return ack(m - 1, 1);
    return ack(m - 1, ack(m, n - 1));
}
```

A. 5　　B. 7　　C. 9　　D. 13

15. 答案：B

分析：ack(2,2)计算：ack(2,2)=ack(1, ack(2,1))

- ack(2,1)=ack(1, ack(2,0))=ack(1,3)
- ack(1,3)=ack(0,4)=5，故 ack(2,1)=5

  最终 ack(2,2)=ack(1,5)=7。故选 B。

二、阅读程序（程序输入不超过数组或字符串定义的范围；判断题正确填 √ ，错误填×；除特 殊说明外，判断题 1.5 分，选择题 3 分，共计 40 分）

(1)

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4
5  using namespace std;
6
7  int f(const string &s, const string &t)
8  {
9      int n = s.length(), m = t.length();
10
11     vector<int> shift(128, m + 1);
12
13     int i, j;
14
15     for (j = 0; j < m; j++)
16         shift[t[j]] = m - j;
17
18     for (i =0; i<= n - m; i += shift[s[i + m]]){
19         j =0;
20         while(j < m && s[i +j] == t[j]) j++;
21         if (j == m) return i;
22     }
23
24     return -1;
25 }
26
27 int main()
28 {
29     string a ,b;
30     cin >> a >> b;
31     cout << f(a, b) << endl;
32     return 0;
33 }
```

假设输入字符串由 ASCII 可见字符组成，完成下面的判断题和单选题：

●判断题

16.（1 分）当输入为“abcde fg”时，输出为-1。（ ）

17. 当输入为“abbababbbab abab”时，输出为 4。（ ）

18. 当输入为“GoodLuckCsp2022 22”时，第 20 行的“j++”语句执行次数为 2。（ ）

●单选题

19. 该算法最坏情况下的时间复杂度为（ ）。

A. O(n + m)　　B. O(n log m)　　C.O(m log n)　　D.O(nm)

20. f(a, b)与下列（ ）语句的功能最类似。

A. a.find(b)　　B. a.rfind(b)　　C.a.substr(b)　　D.a.compare(b)

21. 当输入为“baaabaaabaaabaaaa aaaa ”，第 20 行的“j++”语句执行次数为（ ）。

A. 9　　B. 10　　C. 11　　D. 12

这段代码实现了 Boyer-Moore 字符串搜索算法的简化版，用于在主串 s 中查找模式串 t 首次出现的位置。若找到，返回模式串在主串中的起始索引（从 0 开始）；若未找到，返回 -1。

首先创建一个长度为 128 的数组 shift，初始值为 m + 1（m 为模式串长度）。遍历模式串 t，对于每个字符 t[j]，更新 shift[t[j]] = m - j。该值表示当主串中出现不匹配字符时，模式串应右移的最小距离。

匹配阶段：从主串的起始位置 i 开始，比较主串后缀 s[i..i+m-1] 与模式串 t。若发现不匹配字符（坏字符），查询 shift 表获取右移距离 shift[s[i + m]]，跳过尽可能多的字符。

●判断题

16. （1 分）当输入为“abcde fg”时，输出为-1。（ ）

16. 答案：√

分析：主串为 “abcde”（长度 5），模式串为 “fg”（长度 2）。主串中不存在 “fg”，代码会遍历所有可能位置后返回 -1。故判断正确。

17. 当输入为“abbababbbab abab”时，输出为 4。（ ）

17. 答案：×

分析：主串为 “abbababbbab”，模式串为 “abab”（长度 4）。

18. 当输入为“GoodLuckCsp2022 22”时，第 20 行的“j++”语句执行次数为 2。（ ）

18. 答案：√

分析：主串为 “GoodLuckCsp2022”，模式串为 “22”（长度 2）。

- 模式串 “22” 的匹配位置在主串末尾 “22” 处，比较过程中：
  - `j=0` 时，`s[i] = '2'` 匹配 `t[0]`，`j++`（1 次）；
  - `j=1` 时，`s[i+1] = '2'` 匹配 `t[1]`，`j++`（2 次）。

  第 20 行 “j++” 共执行 2 次。故判断正确。

●单选题

19. 该算法最坏情况下的时间复杂度为（ ）。

A. O(n + m)　　B. O(n log m)　　C. O(m log n)　　D. O(nm)

19. 答案：D

分析：该算法是简化版 Boyer-Moore，最坏情况下（如主串全为'a'，模式串为'aa...ab'），每次需比较 m 个字符，且 i 仅移动 1 步，总比较次数为 O(nm)。故选 D。

20. f(a, b)与下列（ ）语句的功能最类似。

A. a.find(b) B. a.rfind(b) C.a.substr(b) D.a.compare(b)

20. 答案：A

分析：f(a, b)的功能是查找 b 在 a 中首次出现的起始位置，与 a.find(b)（C++ 字符串函数，返回首次出现索引，找不到返回 string::npos）功能一致。a.rfind(b)是查找最后出现位置，a.substr(b)是取子串，a.compare(b)是比较字符串。故选 A。

21. 当输入为“baaabaaabaaabaaaa aaaa ”，第 20 行的“j++”语句执行次数为（ ）。

A.9 B. 10 C.11 D.12

21. 答案：B

分析：主串为 “baaabaaabaaabaaaa”，模式串为 “aaaa”（长度 4）。

匹配过程中，j++执行次数是所有部分匹配的字符数总和：

i=5 时，匹配 3 个字符（j 从 0→3）；

i=6 时，匹配 2 个字符（j 从 0→2）；

i=7 时，匹配 1 个字符（j 从 0→1）；

i=13 时，匹配 4 个字符（j 从 0→4）。

总次数：3+2+1+4=10。故选 B。

(2)

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int MAXN = 105;
6
7  int n, m, k, val[MAXN];
8  int temp[MAXN], cnt[MAXN];
9
10 void init()
11 {
12     cin >> n >> k;
13     for (int i = 0; i < n; i++) cin >> val[i];
14     int maximum = val[0];
15     for (int i = 1; i < n; i++)
16         if (val[i] > maximum) maximum = val[i];
17     m = 1;
```

```
18      while (maximum >= k) {
19          maximum /= k;
20          m++;
21      }
22  }
23
24  void solve()
25  {
26      int base = 1;
27      for (int i = 0; i < m; i++) {
28          for (int j = 0; j < k; j++) cnt[j] = 0;
29          for (int j = 0; j < n; j++) cnt[val[j] / base % k]++;
30          for (int j = 1; j < k; j++) cnt[j] += cnt[j - 1];
31          for (int j = n - 1; j >= 0; j--) {
32              temp[cnt[val[j] / base % k] - 1] = val[j];
33              cnt[val[j] / base % k]--;
34          }
35          for (int j = 0; j < n; j++) val[j] = temp[j];
36          base *= k;
37      }
38  }
39
40  int main()
41  {
42      init();
43      solve();
44      for (int i = 0; i < n; i++) cout << val[i] << ;
45      cout << endl;
46      return 0;
47  }
```

假设输入的 n 为不大于 100 的正整数，k 为不小于 2 且不大于 100 的正整数，val[i]在 int 表示范围内，完成下面的判断题和单选题：

●判断题

22.这是一个不稳定的排序算法。( )

23.该算法的空间复杂度仅与 n 有关。( )

24.该算法的时间复杂度为 O(m(n + k))。( )

●单选题

25.当输入为“5 3 98 26 91 37 46”时，程序第一次执行到第 36 行，val[]数组的 内容依次为（ ）。

A. 91 26 46 37 98　　B. 91 46 37 26 98

C. 98 26 46 91 37　　D. 91 37 46 98 26

26.若 val[i]的最大值为 100，k 取（ ）时算法运算次数最少。

A.2　　B.3　　C.10　　D.不确定

27.当输入的 k 比 val[i]的最大值还大时，该算法退化为（ ）算法。

A.选择排序　　B.冒泡排序　　C.计数排序　　D.桶排序

(3)

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  const int MAXL = 1000;
7
8  int n, k, ans[MAXL];
9
10 int main(void)
11 {
12     cin >> n >> k;
13     if (!n) cout << 0 << endl;
14     else
15     {
16         int m = 0;
17         while (n)
18         {
19             ans[m++] = (n % (-k) + k) % k;
20             n = (ans[m - 1] - n) / k;
21         }
22         for (int i = m - 1; i >= 0; i--)
23             cout << char(ans[i] >= 10 ?
24                          ans[i] + 'A' - 10 :
25                          ans[i] + '0');
26         cout << endl;
27     }
28     return 0;
29 }
```

假设输入的 n 在 int 范围内，k 为不小于 2 且不大于 36 的正整数，完成下面的判断题和单选题：

●判断题

28.该算法的时间复杂度为 $O(\log_k n)$。(　)

29.删除第 23 行的强制类型转换，程序的行为不变。(　)

30.除非输入的 n 为 0，否则程序输出的字符数为 $O([\log_k|n|] + 1)$。(　)

([]表示向下取整)

●单选题

31.当输入为“100 7”时，输出为(　)。

A.　202　　B.　1515　　C.　244　　D.　1754

32.当输入为“ -255 8”时，输出为“(　)”。

A.　1400　　B.　1401　　C.　417　　D.　400

33.当输入为“1000000 19”时，输出为“（　）”。

A.BG939　　B.87GIB　　C.1CD428　　D.7CF1B

三、完善程序（单选题，每小题 3 分，共计 30 分）

(1)（归并第 k 小）已知两个长度均为 n 的有序数组 a1 和 a2（均为递增序，但不保证严格单调递增），并且给定正整数 k（1≤k≤2n），求数组 a1 和 a2 归并排序后的数组里第 k 小的数值。

试补全程序。

```
#include <bits/stdc++.h>
using namespace std;

int solve(int *a1, int *a2, int n, int k) {
    int left1 = 0, right1 = n - 1;
    int left2 = 0, right2 = n - 1;
    while (left1 <= right1 && left2 <= right2) {
        int m1 = (left1 + right1) >> 1;
        int m2 = (left2 + right2) >> 1;
        int cnt = ①;
        if (②) {
            if (cnt < k) left1 = m1 + 1;
            else right2 = m2 - 1;
        } else {
            if (cnt < k) left2 = m2 + 1;
            else right1 = m1 - 1;
        }
    }
    if (③) {
        if (left1 == 0) {
            return a2[k - 1];
        } else {
            int x = a1[left1 - 1], ④;
            return std::max(x, y);
        }
    } else {
            if (left2 == 0) {
                return a1[k - 1];
            } else {
                int x = a2[left2 - 1], ⑤;
                return std:: max(x, y);
            }
    }
}
```

34. ①处应填(　)

A. (m1 + m2) * 2　　B. (m1 - 1) + (m2 - 1)

C. m1 + m2　　D. (m1 + 1) + (m2 + 1)

35. ②处应填( )

A. a1[m1] == a2[m2]　　B. a1[m1] <= a2[m2]

C. a1[m1] >= a2[m2]　　D. a1[m1] != a2[m2]

36. ③处应填( )

A. left1 == right1　　B. left1 < right1

C. left1 > right1　　D. left1 != right1

37. ④处应填( )

A. y = a1[k - left2 - 1]　　B. y = a1[k - left2]

C. y = a2[k - left1 - 1]　　D. y = a2[k - left1]

38. ⑤处应填( )

A. y = a1[k - left2 - 1]　　B. y = a1[k - left2]

C. y = a2[k - left1 - 1]　　D. y = a2[k - left1]

解题思路

该程序通过二分查找高效求解两个有序数组归并后的第 k 小元素，核心思想是：不断缩小两个数组的搜索范围，排除不可能包含第 k 小元素的部分，最终在剩余元素中定位目标值。

选择题解析

34. ①处应填（D）

分析：cnt 表示 a1[0..m1]和 a2[0..m2]两个子数组的总元素个数。

a1[0..m1]的元素个数为 m1（索引从 0 到 m1-1）。

a2[0..m2]的元素个数为 m2。

故 cnt = (m1) + (m2)，选 D。

35. ②处应填（B）

分析：比较 a1[m1]和 a2[m2]的大小，决定缩小哪个数组的范围：

若 a1[m1] <= a2[m2]，说明 a1[0..m1]和 a2[0..m2]的最大元素可能是 a2[m2]。

若 cnt < k（总元素不足 k），需扩大 a1 的搜索范围（left1 = m1 + 1）。

否则，缩小 a2 的搜索范围（right2 = m2 - 1）。

故条件为 a1[m1] <= a2[m2]，选 B。

36. ③处应填（C）

分析：循环结束时，必有一个数组的搜索范围为空（left > right）。

left1 > right1 表示 a1 的搜索范围已空，剩余元素在 a2 中。

后续逻辑处理 a1 为空的情况，故③处判断 left1 > right1，选 C。

37. ④处应填（C）

分析：当 left1 > right1（a1 为空）且 left1 != 0 时：

a1 中已处理的最大元素为 a1[left1 - 1]（x）。

需从 a2 中找剩余元素：已处理 left1 个元素（来自 a1），还需 k - left1 个元素，对应 a2 中索引为 k - left1 - 1 的元素（y）。

故 y = a2[k - left1 - 1]，选 C。

38. ⑤处应填（A）

分析：当 left2 > right2（a2 为空）且 left2 != 0 时：

a2 中已处理的最大元素为 a2[left2 - 1]（x）。
需从 a1 中找剩余元素：已处理 left2 个元素（来自 a2），还需 k - left2 个元素，对应 a1 中索引为 k - left2 - 1 的元素（y）。
故 y = a1[k - left2 - 1]，选 A。

(2)（容器分水）有两个容器，容器 1 的容量为为 a 升，容器 2 的容量为 b 升；同时允 许下列的三种操作，分别为：
1）FILL(i)：用水龙头将容器 i（i∈{1,2}）灌满水；
2）DROP(i)：将容器 i 的水倒进下水道；
3）POUR(i,j)：将容器 i 的水倒进容器 j（完成此操作后，要么容器 j 被灌满，要 么容器 i 被清空）。
求只使用上述的两个容器和三种操作，获得恰好 c 升水的最少操作数和操作序列。上述 a、b、c 均为不超过 100 的正整数，且 c≤max{a,b}。
试补全程序。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 110;

int f[N][N];
int ans;
int a, b, c;
int init;

int dfs(int x, int y) {
    if (f[x][y] != init)
        return f[x][y];
    if (x == c || y == c)
        return f[x][y] = 0;
    f[x][y] = init - 1;
    f[x][y] = min(f[x][y], dfs(a, y) + 1);
    f[x][y] = min(f[x][y], dfs(x, b) + 1);
    f[x][y] = min(f[x][y], dfs(0, y) + 1);
    f[x][y] = min(f[x][y], dfs(x, 0) + 1);
    int t = min(a - x, y);
    f[x][y] = min(f[x][y], ①);
    t = min(x, b - y);
    f[x][y] = min(f[x][y], ②);
    return f[x][y];
}
```

```
void go(int x, int y) {
    if (③)
        return;
    if (f[x][y] == dfs(a, y) + 1) {
        cout << "FILL(1)" << endl;
        go(a, y);
    } else if (f[x][y] == dfs(x, b) + 1) {
        cout << "FILL(2)" << endl;
        go(x, b);
    } else if (f[x][y] == dfs(0, y) + 1) {
        cout << "DROP(1)" << endl;
        go (0, y);
    } else if (f[x][y] == dfs(x, 0) + 1) {
        cout << "DROP(2)" << endl;
        go(x, 0);
    } else {
        int t = min(a - x, y);
        if(f[x][y] == ④) {
            cout << "POUR(2,1)" << endl;
            go(x + t, y - t);
        } else {
            t = min(x, b - y);
            if (f[x][y] == ⑤) {
                cout << "POUR(1,2)" << endl;
                go(x - t, y + t);
            } else
                assert(0);
        }
    }
}

int main() {
    cin >> a >> b >> c;
    ans = 1 << 30;
    memset(f, 127, sizeof f);
    init = **f;
    if ((ans = dfs (0, 0)) == init - 1)
        cout << "impossible";
    else {
        cout << ans << endl;
        go (0, 0);
    }
}
```

39.①处应填( )

A. dfs(x + t, y - t) + 1　　B. dfs(x + t, y - t) - 1

C. dfs(x - t, y + t) + 1　　D. dfs(x - t, y + t) - 1

40.②处应填( )

A. dfs(x + t, y - t) + 1　　B. dfs(x + t, y - t) - 1

C. dfs(x - t, y + t) + 1　　D. dfs(x - t, y + t) - 1

41.③处应填( )

A. x == c || y == c　　B. x == c && y == c

C. x >= c || y >= c　　D. x >= c && y >= c

42.④处应填( )

A. dfs(x + t, y - t) + 1　　B. dfs(x + t, y - t) - 1

C. dfs(x - t, y + t) + 1　　D. dfs(x - t, y + t) - 1

43.⑤处应填( )

A. dfs(x + t, y - t) + 1　　B. dfs(x + t, y - t) - 1

C. dfs(x - t, y + t) + 1　　D. dfs(x - t, y + t) - 1

解题思路

该程序通过深度优先搜索（DFS）+ 记忆化求解 “容器分水” 问题，核心是：

用 dfs(x, y)计算从状态(x, y)（容器 1 有 x 升，容器 2 有 y 升）到目标状态（任一容器为 c 升）的最少操作数，存储在 f[x][y]中。

用 go(x, y)回溯输出具体操作序列，根据 f[x][y]的来源反推上一步操作。

选择题解析

39. ①处应填（A）

分析：①处对应 POUR(2,1) 操作（容器 2 倒水到容器 1）。

倒入量 t = min(a - x, y)（容器 1 剩余容量与容器 2 当前水量的较小值）。

操作后状态为(x + t, y - t)，操作数为该状态的最少操作数加 1（当前操作）。

故①处为 dfs(x + t, y - t) + 1，选 A。

40. ②处应填（C）

分析：②处对应 POUR(1,2) 操作（容器 1 倒水到容器 2）。

倒入量 t = min(x, b - y)（容器 1 当前水量与容器 2 剩余容量的较小值）。

操作后状态为(x - t, y + t)，操作数为该状态的最少操作数加 1（当前操作）。

故②处为 dfs(x - t, y + t) + 1，选 C。

41. ③处应填（A）

分析：go(x, y)的功能是回溯输出操作，终止条件为达到目标状态（任一容器水量为 c）。

当 x == c || y == c 时，无需继续输出操作，直接返回。
故③处为 x == c || y == c，选 A。

42. ④处应填（A）
分析：④处判断当前状态(x, y)的最少操作数是否来自 POUR(2,1) 操作。

该操作对应的后续状态为(x + t, y - t)，当前操作数 f[x][y]等于该状态的操作数加 1。
故④处为 dfs(x + t, y - t) + 1，选 A。

43. ⑤处应填（C）
分析：⑤处判断当前状态(x, y)的最少操作数是否来自 POUR(1,2) 操作。

该操作对应的后续状态为(x - t, y + t)，当前操作数 f[x][y]等于该状态的操作数加 1。
故⑤处为 dfs(x - t, y + t) + 1，选 C。