2024 CSP-J1 试题解析

CSP-J 2024 入门组初赛第一轮初赛试题及答案解析

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 32 位 `int` 类型的存储范围是（）

A `-2147483647 ~ +2147483647`
B `-2147483647 ~ +2147483648`
C `-2147483648 ~ +2147483647`
D `-2147483648 ~ +2147483648`

> C，int32有符号表示范围是$-2^{31} \sim 2^{31}-1$

2 计算 $(14_8 - 1010_2) * D_{16} - 1101_2$ 的结果，并选择答案的十进制值（）
A 13
B 14
C 15
D 16

> A，转为 `10` 进制计算：
> $(12-10)*13-13=2*13-13=13$

3 某公司有 10 名员工，分为 3 个部门：A 部门有 4 名员工，B 部门有 3 名员工、C 部门有 3 名员工。现需要从这 10 名员工中选出 4 名组成一个工作组，且每个部门至少要有 1 人。 问有多少种选择方式？（）
A 120
B 126
C 132
D 238

> B，分类讨论+组合。
> 要计算从 10 名员工（A 部门 4 人，B 部门 3 人，C 部门 3 人）中选出 4 人组成工作组，且每个部门至少有 1 人的选择方式数。
> 由于每个部门至少要有 1 人，可能的部门人员分配情况为：

- (2, 1, 1)：即一个部门出 2 人，另两个部门各出 1 人。
- (1, 2, 1)
- (1, 1, 2)
  注意：由于 B 和 C 部门人数相同（各 3 人），但 A 部门有 4 人，因此需要分别计算每种情况。
  具体分配有三种类型：

1. A 部门出 2 人，B 出 1 人，C 出 1 人。
2. A 部门出 1 人，B 出 2 人，C 出 1 人。
3. A 部门出 1 人，B 出 1 人，C 出 2 人。

计算每种情况的选择方式数：
情况 1：A 出 2 人，B 出 1 人，C 出 1 人

- 从 A 部门 4 人中选 2 人：$\binom{4}{2}$
- 从 B 部门 3 人中选 1 人：$\binom{3}{1}$
- 从 C 部门 3 人中选 1 人：$\binom{3}{1}$
- 方式数：$\binom{4}{2} \times \binom{3}{1} \times \binom{3}{1} = 6 \times 3 \times 3 = 54$
  情况 2：A 出 1 人，B 出 2 人，C 出 1 人
- 从 A 部门 4 人中选 1 人：$\binom{4}{1}$
- 从 B 部门 3 人中选 2 人：$\binom{3}{2}$
- 从 C 部门 3 人中选 1 人：$\binom{3}{1}$
- 方式数：$\binom{4}{1} \times \binom{3}{2} \times \binom{3}{1} = 4 \times 3 \times 3 = 36$
  情况 3：A 出 1 人，B 出 1 人，C 出 2 人
- 从 A 部门 4 人中选 1 人：$\binom{4}{1}$
- 从 B 部门 3 人中选 1 人：$\binom{3}{1}$
- 从 C 部门 3 人中选 2 人：$\binom{3}{2}$
- 方式数：$\binom{4}{1} \times \binom{3}{1} \times \binom{3}{2} = 4 \times 3 \times 3 = 36$

将三种情况相加：
$54 + 36 + 36 = 126$
因此，总的选择方式数为 $\boxed{126}$。

4 以下哪个序列对应数组 0 至 8 的 4 位二进制格雷码（Gray code）（）
A `0000，0001，0011，0010，0110，0111，0101，1000`
B `0000，0001，0011，0010，0110，0111，0100，0101`

C `0000，0001，0011，0010，0100，0101，0111，0110`
D `0000，0001，0011，0010，0110，0111，0101，0100`

> D，参照格雷码构造方法：从 `0 1` 开始，先镜像得到 `0 1 1 0`；再补位，原来的码补0，镜像的码补1，得到 `00 01 11 10`。循环进行得到更多位。

5 记 `1KB` 是 1024 字节（Byte），`1MB` 是 1024KB，那么 `1MB` 是多少二进制位（bit）（）
A `1000000`
B `1048576`
C `8000000`
D `8388608`

> D，注意进制是1024

6 以下哪个不是 C++中的基本数据类型（）
A `int`
B `float`
C `struct`
D `char`

> C
> `struct` 可以组合基本数据类型，形成复合数据类型不是基本数据类型。复合数据类型还有共用体 `union` 和类 `class`。

7 以下哪个不是 C++中的循环语句（）
A `for`
B `while`
C `do-while`
D `repeat-until`

> D，`repeat-until` 是Lua语言的循环语句

8 在C/C++中，`(char)('a'+13)` 与下面的哪一个值相等（）
A `'m'`
B `'n'`
C `'z'`
D `'3'`

> B，`a`开始数第14个字母是`n`

9 假设有序表中有 1000 个元素，则用二分法查找元素 `x` 最多需要比较（）次
A 25
B 10
C 7
D 1

> B
> 二分查找最多比较次数是对有序表中元素个数对数向上取整
> $\lceil \log_2 1000 \rceil = 10$

10 下面哪一个不是操作系统名字（）
A Notepad
B Linux
C Windows
D macOS

> A
> linux操作系统主要用于服务器
> windows操作系统主要用于个人电脑(PC)
> macOS操作系统主要用于苹果品牌的个人电脑(PC)
> Notepad是电脑文本处理软件，不是操作系统

11 在无向图中，所有顶点的度数之和等于（）
A 图的边数
B 图的边数的两倍
C 图的定点数
D 图的定点数的两倍

> B，一条边链接2个顶点，对应度数为2，n条边即为2n，因此所有顶点的度数之和等于图的边数的两倍；
> 相关知识：在无向图中，所有顶点的度数之和等于图的边数

12 已知二叉树的前序遍历为`[A,B,D,E,C,F,G]`，中序遍历为`[D,B,E,A,F,C,G]`，求二叉树的后序遍历 的结果是（）
A `[D,E,B,F,G,C,A]`
B `[D,E,B,F,G,A,C]`
C `[D,B,E,F,G,C,A]`

D `[D,E,B,F,G,A,C]`
A

13 给定一个空栈，支持入栈和出栈操作。若入栈操作的元素依次是 `1 2 3 4 5 6`,其中 `1` 最先 入栈， `6` 最后入栈，下面哪种出栈顺序是不可能的（）
A `6 5 4 3 2 1`
B `1 6 5 4 3 2`
C `2 4 6 5 3 1`
D `1 3 5 2 4 6`

D
入栈顺序出栈逆序，先后有别。

14 有 5 个男生和 3 个女生站成一排，规定 3 个女生必须相邻，问有多少种不同的排列方式（）
A 4320 种
B 5040 种
C 3600 种
D 2880 种

> A，相邻问题，使用捆绑法
> 把3各女生捆绑在一起和5个男生站成一排， `A(6,6)=6*5*4*3*2*1=720`
> 3个女生内部交换，为不同的顺序，共 `A(3,3)=3*2*1=6` 种
> 根据乘法原理
> $720 * 6 = 4320$种

15 编译器的主要作用是什么（）
A 直接执行源代码
B 将源代码转换为机器代码
C 进行代码调试
D 管理程序运行时的内存

> B，编译器的主要作用是将高级语言代码编译称机器代码，让计算机执行。

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

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

bool isPrime(int n) {
    if (n <= 1) {
        return false;
    }
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}

int countPrimes(int n) {
    int count = 0;
    for (int i = 2; i <= n; i++) {
        if (isPrime(i)) {
            count++;
        }
    }
    return count;
}

int sumPrimes(int n) {
    int sum = 0;
    for (int i = 2; i <= n; i++) {
        if (isPrime(i)) {
            sum += i;
        }
    }
    return sum;
}

int main() {
    int x;
    cin >> x;
    cout << countPrimes(x) << " " << sumPrimes(x) << endl;
```

```
    return 0;
}
```

解析

考点：循环计数，求和；函数；质数判定

`isPrime` 函数判断传入的 `n` 是否是质数。
`countPrimes` 函数统计 `1~n` 中的质数个数。
`sumPrimes` 函数统计 `1~n` 中的质数加和。

16 当输入为 `10` 时，程序的第一个输出为 `4` ，第二个输出为 `17` （）✓
1~10中质数有 $2,3,5,7$ ，有4个。加和为 $2+3+5+7=17$

17 若将 `isPrime(i)` 函数种的条件改为 `i<=n/2` ,输入 `20` 时，`countPrimes(20)` 的 输出将变为 `6` （）
×

由于 $\frac{n}{2} \geq \sqrt{n}$ ，所以 `i` 从2循环到$n/2$和 `i` 从2循环到 $\sqrt{n}$ 的最终效果是一样的。
输入20时， `countPrimes` 函数返回的仍然是1~20中的质数个数，有 $2,3,5,7,11,13,17,19$ ，共8个，应该输出 `8`

18 `sumPrimes` 函数计算的是从 2 到 n 之间的所有素数之和（）
✓

19 当输入为“ `50` ”时， `sumPrimes(50)` 的输出为（）
A `1060`
B `328`
C `381`
D `275`

B
求1~50中的质数加和
$2+3+5+7+11+13+17+19+23+29+31+37+41+43+47=328$ ，选B。

20 如果将 `for(int i=2;i*i<=n;i++)` 改为 `for(int i=2;i<=n;i++)` ,输入 `10` 时，程序的输出()
A 将不能正确计算 `10` 以内素数个数及其和
B 仍然输出 `4` 和 `17`

C 输出 `3` 和 `10`
D 输出结果不变，但运行时间更短

A
`isPrime` 函数的原理是：枚举2到 $\lfloor\sqrt{n}\rfloor$ 的所有整数，如果存在n的因数，则n不是质数。如果不存在，则n是质数。
如果枚举从2到n的所有数字，那么当 `i` 为 `n` 时，一定有 `n%i == 0`，isPrime函数一定会返回 `false`，那么 `isPrime` 函数就失去了判断n是否为质数的功能，因此将不能正确计算10以内的质数个数和加和。

2

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

int compute(vector<int>& cost) {
    int n = cost.size();
    vector<int> dp(n+1, 0);
    dp[1] = cost[0];
    for (int i = 2; i <= n; i++) {
        dp[i] = min(dp[i-1], dp[i-2]) + cost[i-1];
    }
    return min(dp[n], dp[n-1]);
}

int main() {
    int n;
    cin >> n;
    vector<int> cost(n);
    for (int i = 0; i < n; i++) {
        cin >> cost[i];
    }
    cout << compute(cost) << endl;
    return 0;
}
```

解析

考点：动态规划，STL

没有题目背景，难以理解该问题的状态定义。直接当做递推问题来看。
先输入 `cost` 数组，下标从0~n-1。
状态数组是 `dp` 。
初值： `dp[0] = 0` ， `dp[1] = cost[0]`
状态转移方程： `dp[i] = min(dp[i-1], dp[i-2])+cost[i-1];`
最后结果为： `dp[n]` 和 `dp[n-1]` 的较小值。

可见每一步应该是选择cost数组中的较小值，计算。

21 当输入的 `cost` 数组为$10, 15, 20$时，程序的输出为 `15` （）✓
根据状态转移方程计算递推式。

i	cost[i]	dp[i]
0	10	0
1	15	10
2	20	15
3		30

输出 `min(dp[3], dp[2])=15` ，叙述正确。

22 如果将 `dp[i-1]` 改为 `dp[i-3]` ，程序可能会产生编译错误（）✕
将dp[i-1]改为dp[i-3]后，当i为2时，执行到这一句 `dp[i] = min(dp[i-3], dp[i-2])+cost[i-1];`
其中 `dp[i-3]` 的下标为-1，数组下标越界，会产生运行时错误，而不是编译错误。

23 （2 分）程序总是输出 `cost` 数组种的最小的元素（）✕
第1题的结果15就不是 `cost` 数组的最小元素。

24 当输入的 `cost` 数组为$1, 100, 1, 1, 1, 100, 1, 1, 100, 1$时，程序的输出为（）
A `6`
B `7`
C `8`
D `9`

A
n=10，列表递推。

i	cost[i]	dp[i]
0	1	0
1	100	1
2	1	100
3	1	2
4	1	3
5	100	3
6		103
7	1	4
8	100	5
9	1	104
10		6

输出 `min(dp[10], dp[9])=6` ，选A。

25 （4 分）如果输入的 `cost` 数组为$10, 15, 30, 5, 5, 10, 20$，程序的输出为（）
A `25`
B `30`
C `35`
D `40`

B
n=7，列表递推

i	cost[i]	dp[i]
0	10	0
1	15	10
2	30	15
3	5	40
4	5	20
5	10	25
6	20	30
7		45

输出min(dp[7], dp[6])=30，选B。

26 若将代码中的 `min(dp[i-1], dp[i-2]) + cost[i-1]` 修改为 `dp[i-1] + cost[i-2]`，输入 `cost` 数组为 `{5,10,15}` 时，程序的输出为 ()

A 10

B 15

C 20

D 25

A

n=3，列表递推

i	cost[i]	dp[i]
0	5	0
1	10	5
2	15	10
3		20

输出 `min(dp[3], dp[2]) = 10`，选A。

3

```
#include <iostream>
#include <cmath>
using namespace std;

int customFunction(int a, int b) {
    if (b == 0) {
        return a;
    }
    return a + customFunction(a, b-1);
}

int main() {
    int x, y;
    cin >> x >> y;
    int result = customFunction(x, y);
    cout << pow(result, 2) << endl;
```

```
    return 0;
}
```

解析

考点： 递归
递归推算， `customFunction(a,b)` 是一个乘法计算，返回值为$x(y+1)$

27 当输入为 `2 3` 时， `customFunction(2,3)` 的返回值为 `64` （） ×
返回值是$2*(3+1)=8$

28 当 `b` 为负数时， `customFunction(a,b)` 会陷入无限递归（） **✓×都可以**
当 `b` 为负数时，参数 `b` 只会越来越小，递归出口的条件 `if(b==0)` 不会满足，因此递归会一直进行下去。
“无限递归”如果指的是一种表象，看起来在不停地递归调用，程序无法停止，那么该叙述是正确的。
但递归实际上不会无限进行下去，每次递归调用都会在栈区占一定的空间，当栈区没有空间时，会发生栈溢出，产生运行时错误。
如果假想栈区空间非常大（大于32G)， `b` 每次递归调用后都会减1，如果 `b` 是正数，在有限次调用后 `b` 会为 `0` ，进入递归出口。如果 `b` 是负数，由于 `b` 是 `int` 类型变量，不断减 `1` 后可以达到的最小值为-2^{31}。再次减1后，根据补码相加的计算原理，b的值会变为$2^{31}-1$，进行有限次递归调用后 `b` 还是会变为 `0` 。因此当不考虑栈区大小的情况下，递归也不会进行无限次。 因此该题选哪个都算对。

29 当 `b` 的值越大，程序的运行时间越长（）
✓×都可以
如果考虑栈区有限，该递归函数调用的次数是 `y` 次，每次调用的时间复杂度是$O(1)$，总时间复杂度是$O(y)$。实参 `y` 是形参 `b` 的值，该函数的运行时间随着b的增大而增大，叙述正确。
如果认为栈区空间非常大（大于32G)，根据上题的解析，则当 `b` 的值为负数时，比当 `b` 为正数时不断进行递归调用直至 `b` 为0，进行的递归调用的次数更多。该描述不正确。
该题选哪个都对。

30 当输入为 `5 4` 时， `customFunction(5,4)` 的返回值为（）
A `5`
B `25`

C `250`
D `625`

B

31 如果输入 `x = 3` 和 `y = 3`，则程序的最终输出为（）
A `27`
B `81`
C `144`
D `256`

C

32 （4 分）若将 `customFunction` 函数改为 `return a + customFunction(a-1, b-1)`；并输 入 `3 3`，则程序的最终输出为（）
A 9
B 16
C 25
D 36

D
该问题可以直接模拟递归过程，也可以使用递归变递推的方法
设c[a][b]的值为 `customFunction(a, b)` 的返回值。
初始状态为：当 `b` 为 `0` 时，值为 `a`。 则 c[0][0] = 0 递推关系为 c[a][b] = a+c[a-1][b-1] 可以通过 c[a-1][b-1] 推出 c[a][b] 。那么就可以通过 c[0][0] 推出 c[1][1] 再推出 c[2][2] 再推出 c[3][3]`。

```
c[1][1] = 1+c[0][0] = 1
c[2][2] = 2+c[1][1] = 3
c[3][3] = 3+c[2][2] = 6
```

调用 `customFunction(3,3)` 返回 `6`，注意主函数的最终结果需要再求平方（`pow(result,2)`），结果为 `36`，选D。

三、完善程序

(单选题，每小题3分，共计 3 分)

1 判断平方数

问题：给定一个正整数 `n`，判断这个数 是不是完全平方数，即存在一个正 整数 `x` 使得 `x` 的平方等于 `n`。

试补全程序。

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

bool isSquare(int num){
    int i = ___①___;
    int bound = ___②___;
    for (; i <= bound; ++i) {
        if (___③___) {
            return ___④___;
        }
    }
    return ___⑤___;
}
int main(){
    int n;
    cin >> n;
    if(isSquare(n)){
        cout<< n << " is a Square number" << endl;
    }
    else{
        cout<< n << " is not a Square number" << endl;
    }
    return 0;
}
```

解析

考点：枚举
根据主函数中 `isSquare` 的用法可知， `isSquare` 函数的作用是判断 `num` 是否是完全平方数。
该问题就是要求我们写出判断一个数是否是完全平方数的函数。

33 ①处应填（）
A `1`

B `2`
C `3`
D `4`
如果一个正整数n是完全平方数，那么一定存在正整数i满足$n = i^2$
我们可以枚举所有可能的`i`，`i`最小时应该为`1`（因为当`n`为`1`时，`i`应该为`1`，此时判断出`n`是完全平方数），因此第(1)空填`1`，选A。

34 ②处应填（）
A `(int) floor(sqrt(num)-1)`
B `(int)floor(sqrt(num))`
C `floor(sqrt(num/2))-1`
D `floor(sqrt(num/2))`
`i`最大不能超过$\sqrt{n}$，当n是完全平方数时，$\lfloor\sqrt{n}\rfloor = \sqrt{n}$，其他情况下$\sqrt{n}$是浮点数，应该向下取整，因此`i`的最大值`bound`设为$\lfloor\sqrt{n}\rfloor$。填`(int)floor(sqrt(num))`，第(2)空选B。

> 为什么强制类型转换为`int`？因为`sqrt()`和`floor()`函数的返回值都是`double`，而`bound`的类型为`int`。

35 ③处应填（）
A `num=2*i`
B `num== 2*i`
C `num=i*i`
D `num==i*i`
此处是判断是否为完全平方数，选D。

36 ④处应填（）
A `num=2*i`
B `num== 2*i`
C `true`
D `false`

37 ⑤处应填（）
A `num=2*i`
B `num!= 2*i`
C `true`
D `false`
如果$i^2 = num$，那么`num`就是完全平方数。第(3)空填`num == i*i`，选D。第

(4)空填 `true` ，选C。
如果从1到$\lfloor\sqrt{n}\rfloor$的数字的平方都不为 `num` ，那么 `num` 不是完全平方数，返回 `false` ，第(5)空选D。

> 第（4）空A选项为 `num= 2*i` ，由于 `i>1` ，所以该赋值表达式的值不为0，转为 `bool` 类型后会被认为是 `true` ，因此如果选A在逻辑上也不算错，但从代码书写角度来说，写这一句是无意义且不合理的。但如果该题选A，判题时算是对的了。）

2 Hanoi塔问题

给定三根柱子，分别标记为 A、B 和 C。初始状态下，柱子 A 上有若干个圆盘，这些圆盘从上到下按从小到大的顺序排列。任务是将这些圆盘全部移到柱子 C 上，且必须保持原有顺序不变。在移动过程中，需要遵守以下规则:

1. 只能从一根柱子的顶部取出圆盘，并将其放入另一根柱子的顶部。
2. 每次只能移动一个圆盘。
3. 小圆盘必须始终在大圆盘之上。

试补全程序。

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

void move(char src, char tgt) {
    cout << "从柱子" << src << "挪到柱子上" << tgt << endl;
}
void dfs(int i, char src, char tmp, char tgt) {
    if (i == ___①___) {
        move(___②___);
        return;
    }
    dfs(i - 1, ___③___);
    move(src, tgt);
    dfs(___⑤___, ___④___);
}

int main() {
    int n;
```

```
    cin >> n;
    dfs(n, 'A', 'B', 'C');
}
```

解析

考点：递归；深度优先搜索（DFS）

> `src`：source源头
> `tgt`：target目标
> `tmp`：temporary临时的

结合题目背景来看，`move`函数用来打印从`src`到`tgt`的移动过程；而`dfs`函数是将`i`层的hanoi塔从`src`柱经`tmp`柱移动到`tgt`柱。

我们尝试用例子分析递归过程（把ABC三个柱子称为起点，暂存点，终点）：
1层hanoi塔

```
 1
---      ---      ---
src      tmp      tgt
移动过程：
1: src -> tgt
共移动3次
```

2层hanoi塔

```
 1
 2
---      ---      ---
src      tmp      tgt

移动过程：  1移开，2到tgt，1再移到tgt
1: src -> tmp
2: src -> tgt
1: tmp -> tgt
共移动3次
```

3层

```
 1
 2
 3
---      ---      ---
src      tmp      tgt

移动过程： 先把1、2移开，终点是tmp；再移动3到终点tgt；最后把1、2移动到终点tgt

1: src -> tgt
2: src -> tmp
1: tgt -> tmp

3: src -> tgt


1: tmp -> src
2: tmp -> tgt
1: src -> tgt
```

以此类推，
观察可发现递归过程是每一层：（把ABC三个柱子称为起点，暂存点，终点）上层圆盘给下层圆盘让位，移到暂存点，下层圆盘先移动到终点，上层圆盘然后移动到终点。这样就完成一层递归。

结合代码，归纳递归过程：
对于一般情况，
第一步，先将 `i-1` 层Hanoi塔从 `src` 柱开始，临时经过 `tgt` 柱移动到 `tmp` 柱；
第二步，将第 `i` 层Hanoi塔从 `src` 柱移动到 `tgt` 柱；
第三步，将第 `i-1` 层Hanoi塔从 `tmp` 柱开始，临时经过 `src` 柱移动到 `tgt` 柱

我们只需要在过程中处理起点，暂存点，终点的变化就可以了。

38 ①处应填（）
A `0`
B `1`
C `2`
D `3`
选B，因为 `i=1` 时只有一层Hanoi塔，直接移动就可以。

```
 1
---     ---     ---
src     tmp     tgt

1: src -> tgt
```

39 ②处应填（）

A `src,tmp`

B `src,tgt`

C `tmp,tgt`

D `tgt,tmp`

B

`move` 函数用来打印从 `src` 到 `tgt` 的移动过程，在 `dfs` 函数中，实参就是 `src` 和 `tgt` 。

40 ③处应填（）

A `src,tmp,tgt`

B `src, tgt, tmp`

C `tgt, tmp, src`

D `tgt, src, tmp`

B

41 ④处应填（）

A `src, tmp, tgt`

B `tmp, src, tgt`

C `src, tgt, tmp`

D `tgt, src, tmp`

B

42 ⑤处应填（）

A `0`

B `1`

C `i-1`

D `i`

C