2022 CSP-J 组第一轮初赛真题答案及全面解析

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.以下哪种功能没有涉及 C++语言的面向对象特性支持:( )。

A.C++中调用 printf 函数

B.C++中调用用户定义的类成员函数

C.C++中构造 1 个 class 或 struct

D.C++中构造来源于同一基类的多个派生类

正确答案:A

解析:面向对象考察的内容与类相关，题中唯一没有出现类的选项是 A 选项。printf 函数在 c 语言中就存在。

2.有 6 个元素，按照 6、5、4、3、2、1 的顺序进入栈 S，请问下列哪个出栈序列是非法的()。

A.5 4 3 6 1 2

B.4 5 3 1 2 6

C.3 4 6 5 2 1

D.2 3 4 1 5 6

正确答案：C

解析：栈的特征：后进先出。

A 选项：65 进栈，5 出栈，4 进栈，4 出栈，3 进栈，3 出栈，6 出栈，21 进栈，1 出栈，2 出栈。

B 选项：654 进栈，4 出栈，5 出栈，3 进栈，3 出栈，21 进栈，1 出栈，2 出栈，6 出栈。

C 选项：6543 进栈，3 出栈，4 出栈，5 没有出栈，6 就无法出栈，这里非法。

D 选项：65432 进栈，234 出栈，1 进栈，1 出栈，5 出栈，6 出栈。

3. 运行以下代码片段的行为是( )。

```
int x = 101;
int y = 201;
int *p = &x;
int *q = &y;
p = q;
```

A. 将 x 的值赋为 201

B. B.将 y 的值赋为 101

C.将 q 指向 x 的地址

D.将 p 指向 y 的地址

正确答案：D

解析：第四条语句的意思是将 p 指向 x 的地址，第五条语句的意思是将 q 指向 y 的地址。第六条语句将 q 赋值给 p，于是 p 就指向原来 q 指向的 y 的地址了。

4.链表和数组的区别包括()。

A.数组不能排序，链表可以

B.链表比数组能存储更多的信息

C.数组大小固定，链表大小可动态调整

D.以上均正确

正确答案：C

解析：数组在定义时需要请求空间，所以数组的大小为固定值。链表使用指针完成连接，可动态调整。

5.对假设栈 S 和队列 Q 的初始状态为空。存在 e1~e6 六个互不相同的数据，每个数据按照进栈 S、出栈 S、进队列 Q、出队列 Q 的顺序操作，不同数据间的操作可能会交错。已知栈 S 中依次有数据 e1、e2、e3、e4、e5 和 e6 进栈，队列 Q 依次有数据 e2、e4、e3、e6、e5 和 e1 出队列。则栈 S 的容量至少是( )个数据。

A. 2　　B. 3　　C .4　　D.6

正确答案：B

【解析】栈：后进先出。队列：先进先出。

本道题目只需判断出栈的先后顺序即可。

12 进栈，2 出栈，容量为 2。

34 进栈，4 出栈，3 出栈，容量为 3。

56 进栈，6 出栈，5 出栈，1 出栈，容量为 3。

综上，栈 S 容量至少为 3。

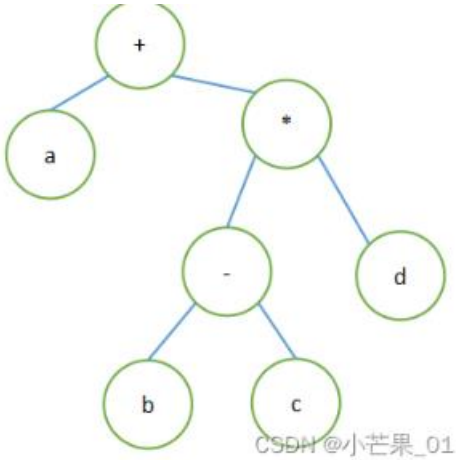
6.对表达式 a+(b-c)*d 的前缀表达式为( )，其中+、-、*是运算符。

A. *+a-bcd　　B. +a*-bcd

C. abc-d*+　　D. abc-+d

正确答案：B

【解析】前缀表达式，先遍历根节点，再遍历左节点，最后遍历右节点。如下图所示：该表达式的前缀表达式为：+a*-bcd。



7.假设字母表 {a，b，c，d，e}在字符串出现的频率分别为10%，15%，38%，16%29%。若使用哈夫曼编码方式对字母进行不定长的二进制编码，字母 d 的编码长度为( )位。

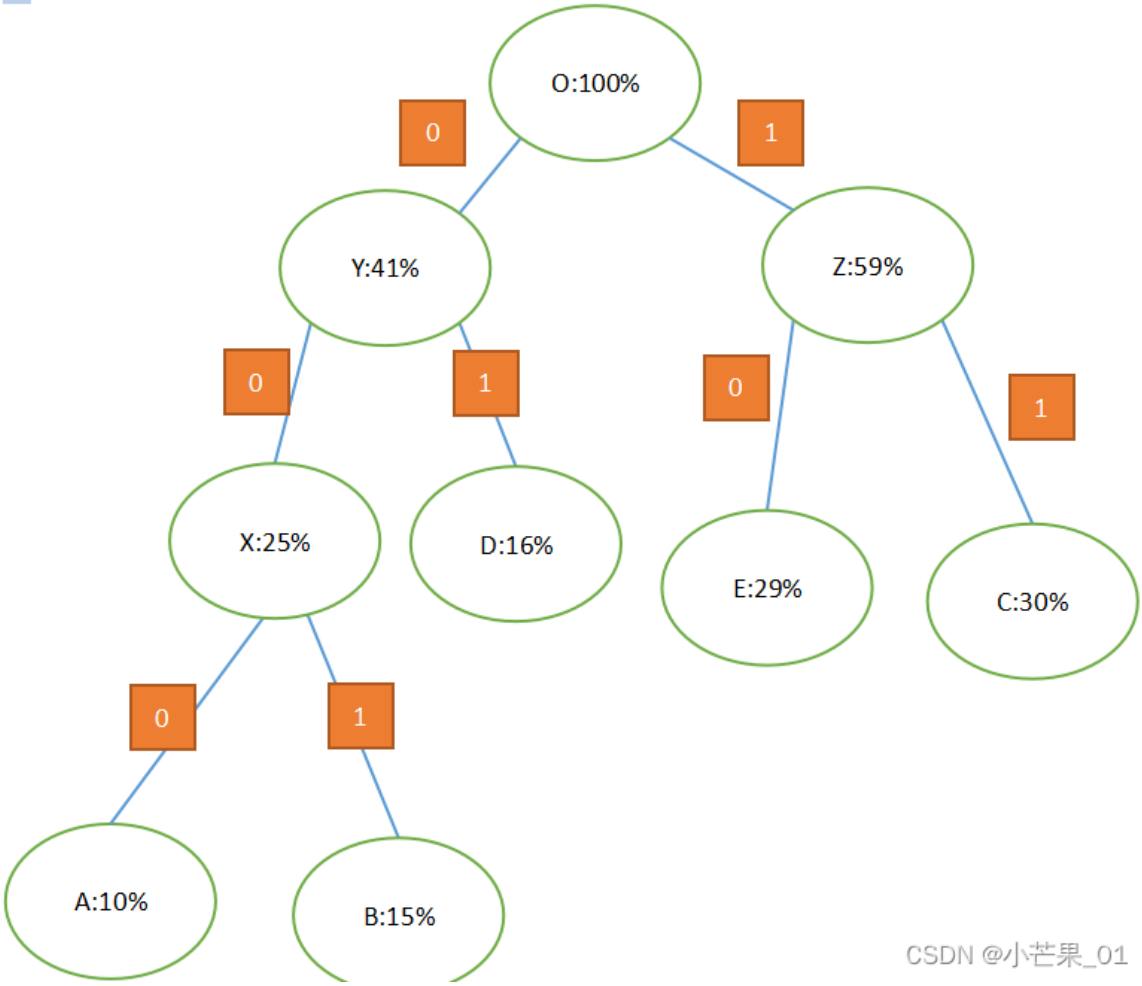
A. 1　　B. 2　　C.2 或 3　　D.3

正确答案:B

哈夫曼编码(Huffman Coding)，又称霍夫曼编码，是一种编码方式，哈夫曼编码是可变字长编码(VLC)的一种。Huffman 于1952 年提出一种编码方法，该方法完全依据字符出现概率来

构造异字头的平均长度最短的码字，有时称之为最佳编码，一般就叫做 Huffman 编码（有时也称为霍夫曼编码）。

依据哈夫曼编码原理，可构造如下二叉树：



字母	编码
a	000
b	001
c	11
d	01
e	10

所以 d 的编码长度为 2。

8.一棵有 n 个结点的完全二叉树用数组进行存储与表示，已知根结点存储在数组的第 1 个位置。若存储在数组第 9 个位置的结点存在兄弟结点和两个子结点，则它的兄弟结点和右子结点的位置分别是()。

A. 8、18　　B. 10、18

C. 8、19　　D. 10、19

正确答案：C

【解析】：完全二叉树的排列是按照从上至下，从左至右的顺序进行排列。

那么 9 号节点在第三层的第二个节点的位置，是个右节点，它的兄弟节点（左节点）编号为 8，它的孩子节点：右节点编号为 2*i*+1，即 19；左节点编号为 2i，即 18。

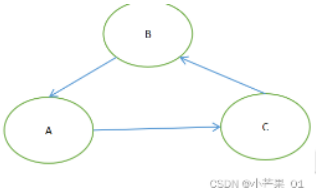
9.考虑由 N 个顶点构成的有向连通图，采用邻接矩阵的数据结构表示时，该矩阵中至少存在()个非零元素。

A. N-1　B. N　C. N+1　D. N^2

正确答案:B

解析:n 个顶点的有向联通图至少存在 n 个有向边。

如下图所示，3 个顶点的有向连通图（最少边）G 和它的邻接矩阵 A



$$A=\begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 0 & 0\end{bmatrix}$$

10.以下对数据结构的表述不恰当的一项为:( )。

A.图的深度优先遍历算法常使用的数据结构为栈。

B.栈的访问原则为后进先出，队列的访问原则是先进先出。

C.队列常常被用于广度优先搜索算法。

D.栈与队列存在本质不同，无法用栈实现队列。

正确答案：D

【解析】栈的特征为：先进后出，或者说后进先出，那么使用两个栈即可实现先进先出，即队列。

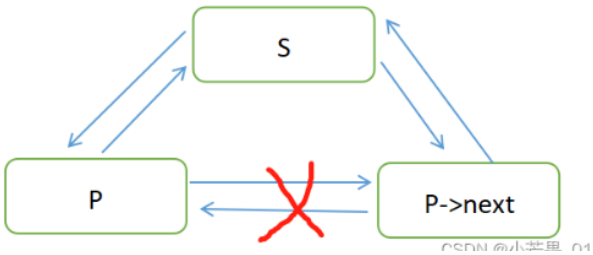
11.以下哪组操作能完成在双向循环链表结点 p 之后插入结点 s 的效果(其中，next 域为结点的直接后继，prev 域为结点的直接前驱):()。

A.p->next->prev=s;s->prev=p;p->next=s;s->next=p->next;B.p->next->prev=s;p->next=s;s->prev=p;s->next=p->next;C.s->prev=p;s->next=p->next;p->next=s;p->next->prev=s;

D.s->next=p->next;p->next->prev=s;s->prev=p;p->next=s

正确答案：D

【解析】：题目描述为在 p 后插入一个节点 s。如图所示：



插入一个 S 节点，从后往前插入，先让 s 的后继指向 p 的后继，再让 p 的后继的前驱设为 s。然后将 s 的前驱设为 p，p 的后继重新设为 s。

A 选项中第三句：p 的后继已经修改，s 的后继就不能再设为原本 p 的后继了，出现矛盾。

B 选项第二句也出现了与 A 相同的问题。

D 选项中第三句，p 的后继已指向 s，那么 p 的后继的前驱便自动指向 p，现在又重新赋值为 s，自相矛盾。

12.以下排序算法的常见实现中，哪个选项的说法是错误的:()。

A.冒泡排序算法是稳定的

B.简单选择排序是稳定的

C.简单插入排序是稳定的

D.归并排序算法是稳定的

正确答案：B

解析:排序算法稳定性是指在多个具有相同关键词的记录在待排序的序列中重新排序后,被排序的记录相对位置保持不变 。堆排序、快速排序、希尔排序、直接选择排序是不稳定的排序算法，而冒泡排序、直接插入排序、折半插入排序、

归并排序是稳定的排序算法。

13.八进制数 32.1 对应的十进制数是( )。

A. 24.125　　B. 24.250

C.26.125　　D.26.250

答案：C

解析：3*8+2*1+1/8=26.125 ，故选 C。

14.一个字符串中任意个连续的字符组成的子序列称为该字符串的子串，则字符串 abcab 有( )个内容互不相同的子串。

A. 12　B. 13　C. 14　D. 15

正确答案：B

【解析】

(1)空串

(2)a,b,c

(3)ab,bc,ca

(4)abc,bca,cab

(5)abca,bcab

(6)abcab

共 1+3+3+3+2+1=13 个互不相同的子串。

15.以下对递归方法的描述中，正确的是:()

A.递归是允许使用多组参数调用函数的编程技术

B.递归是通过调用自身来求解问题的编程技术

C.递归是面向对象和数据而不是功能和逻辑的编程语言模型

D.递归是将用某种高级语言转换为机器代码的编程技术

正确答案：B

【解析】递归用一句话来表示就是自己调用自己。它是一种函数用法。

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 v，错误填 x;除特 殊说明外，判断题 1.5 分，选择题 3 分，共计 40 分)

(1)

```
01 #include <iostream>
02
03 using namespace std;
04
05 int main()
06 {
07     unsigned short x, y;
08     cin >> x >> y;
09     x = (x | x << 2) & 0x33;
10     x = (x | x << 1) & 0x55;
11     y = (y | y << 2) & 0x33;
12     y = (y | y << 1) & 0x55;
13     unsigned short z = x | y << 1;
14     cout << z << endl;
15     return 0;
16 }
```

判断题

16.删去第 7 行与第 13 行的 unsigned，程序行为不变。()

17.将第 7 行与第 13 行的 short 均改为 char，程序行为不变( )

18.程序总是输出一个整数“8”()

19.当输入为“2 2”时，输出为“10”()

20.当输入为“2 2”时，输出为“59”()

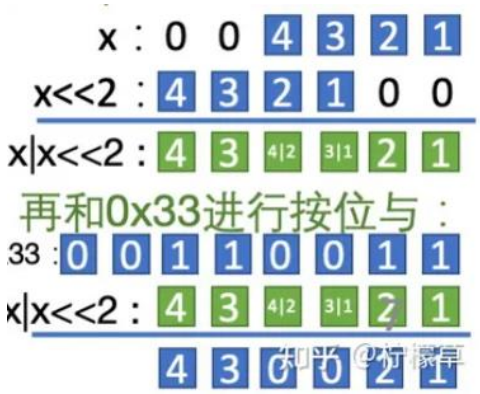
2.单选题

21.当输入为“13 8”时，输出为()

A.0 B.“209 C.“197” D.“226

题目解析:本题主要考察位运算。观察题目数据也不大，暴算也不是不可能。但是我们不妨来研究一下这些位运算乱象下的真相。这样做起题目会更轻松些。

观察到类似 x=(xx<<2)&0x33 这种描述，0x33 就是和二进制的 00110011 进行按位与，这就是在取某些位。而前面是把 x 和 x<<2 进行或，x 和自己的错位进行运算，根据去年 base64 一题的经验，我们可以想到，画图来理解会更直观，



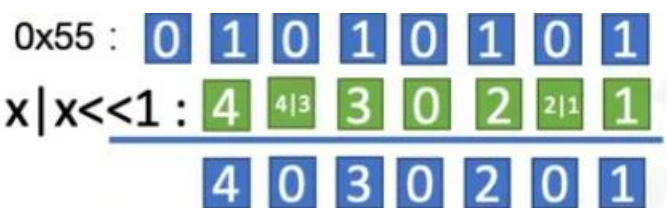
观察到题目说 x 和 y 都不 那么 x|x<<2 的效果如下:

超过 15，因此其实 x 和 y 的二进制顶多是 4 位(15 的二进制是 1111)。因此我们可以用 4321 分别代表 x 的二进制从高到低的每一位

第 9 句结束后，x 的二进制就从原来的 4321，变成了 430021!

接下来同理进行 x=(xx<<1)&0x55 的运算。

再和 0x55(二进制 0101 0101)进行按位与

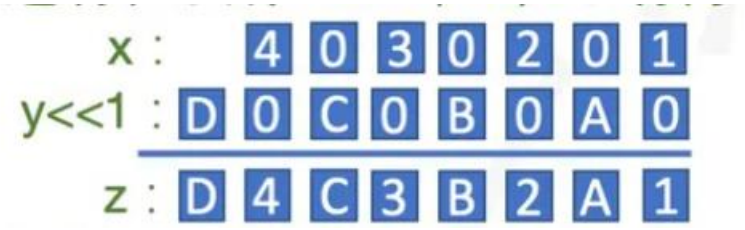


可以看到，第 10 句结束后，x 的二进制就从最开始的 4321，

变成了 4030201!

y 的变化和 x 完全相同。

如果把原始 x 的二进制表示成 4321，原始 y 的二进制表示成 DCBA，那么 z 将为



发现位运算乱象下果然藏着很得到这个结论，”有意思的真相。

1.判断题解析：

16.删去第 7 行与第 13 行的 unsigned，程序行为不变。()

正确答案:正确

解析:unsigned short 是 16 位，删掉 unsigned 以后，只要能保证在非符号位的后 15 位以内进行操作都没问题。通过上述分析可以看到 z 的位数最多是 8 位，没有超过 15 位。因此程序行为不变。

17.将第 7 行与第 13 行的 short 均改为 char，程序行为不变()

正确答案：错误

解析:看似没有问题。但是 cout<<z;会因为 z 是 char 类型而输出字符，不再输出整数。

18.程序总是输出一个整数“8”()

正确答案：错误

解析:通过上述分析可以看到如果 x 是 4321,y 是 DCBA，那么 z 是 D4C3B2A1，并不是 0。

19.当输入为“2 2”时，输出为“10”()

正确答案：错误

解析:通过上述分析可以看到如果 x 是 4321y 是 DCBA，那么 z 是 D4C3B2A1。现在 x 的二进制是 0010，y 也是 0010，那么答案将是 1100，即 12.

20.当输入为“2 2”时，输出为“59”()

正确答案：错误

解析:通过上述分析可以看到如果 x 是 4321y 是 DCBA，那么 z 是 D4C3B2A1。现在 x 的二进制是 0010，y 也是 0010，那么答案将是 1100，即 12.

2. 选择题解析

21.当输入为“13 8”时，输出为()

A.0　B.“209　C.“197”　D.“226

正确答案：B

解析:通过上述分析可以看到如果 x 是 4321y 是 DCBA，那么 z 是 D4C3B2A1.现在 x 的二进制是 1101，y 也是 1000,，那么答案将是 11010001，即 209

(2)

```
01 #include <algorithm>
02 #include <iostream>
03 #include <limits>
04
05 using namespace std;
06
07 const int MAXN = 105;
08 const int MAXK = 105;
09
10 int h[MAXN][MAXK];
11

12 int f(int n, int m)
13 {
14     if (m == 1) return n;
15     if (n == 0) return 0;
16
17     int ret = numeric_limits<int>::max();
18     for (int i = 1; i <= n; i++)
19         ret = min(ret, max(f(n - i, m), f(i - 1, m - 1)) + 1);
20     return ret;
21 }
```

```
22
23 int g(int n, int m)
24 {
25     for (int i = 1; i <= n; i++)
26         h[i][1] = i;
27     for (int j = 1; j <= m; j++)
28         h[0][j] = 0;
29
30     for (int i = 1; i <= n; i++) {
31         for (int j = 2; j <= m; j++) {
32             h[i][j] = numeric_limits<int>::max();
33             for (int k = 1; k <= i; k++)
34             h[i][j] = min(
35                 h[i][j],
36                 max(h[i - k][j], h[k - 1][j - 1]) + 1);
37         }
38     }
39
40     return h[n][m];
41 }
42
43 int main()
44 {
45     int n, m;
46     cin >> n >> m;
47     cout << f(n, m) << endl << g(n, m) << endl;
48     return 0;
49 }
```

假设输入的 n、m 均是不超过 100 的正整数，完成下面的判断题和单选题:

判断题

22.当输入为“7 3”时，第 19 行用来取最小值的 min 函数执行了 449 次。( )

23.输出的两行整数总是相同的。()

24.当 m 为 1 时，输出的第一行总为 n。()

选择题

25.算法 g(n,m)最为准确的时间复杂度分析结果为( )。

A. $O(n^{3/2}m)$ B.O(nm) C. O(nm) D.$O(nm^2)$

26. 当输入为“20 2”时，输出的第一行为( )

A.4　　B.5　　C.6　　D.20

27.(4 分)当输入为“108 100”时，输出的第一行为( )

A.6　B.7　C“8”　D.9

题目分析:首先分析一下 f 函数的意义。可以看到是信奥经典题型:有返回值的搜索。参数有两个，这个情况我们经常采用数形结合的方式，即把 n 成二维数组的行，m 当成列，这样会达到记忆化的效果。

接下来用人话来分析:

```
12 int f(int n, int m) →求第n行第m列的值
13 {
14     if (m == 1) return n; →第m=1列的答案为此时的行号n
15     if (n == 0) return 0; →第n=0列的答案一律为0
16
17     int ret = numeric_limits<int>::max(); →没学过这个函数的盲猜ret赋值为int最大值
18     for (int i = 1; i <= n; i++)
19         ret = min(ret, max(f(n - i, m), f(i - 1, m - 1)) + 1);
20     return ret;
21 }
22
```

→第n-i行m列　→第i-1行m-1列

两者取较大值后+1

最后取所有i里面的最小值，即为f(n,m)的答案

举个实际的例子，比如 n=4，m=5

i=1 时，f(4-1,5)和 f(1-1,4) 即 f(3,5)和 f(0,4)

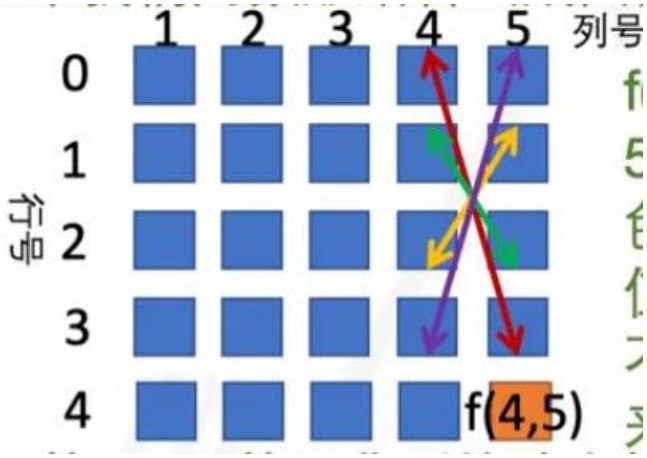
)i=2 时，f(4-2,5)和 f(2-1,4)即 f(2,5) 和 f(1,4)

i=3 时，f(4-3,5)和 f(3-1,4) 即 f(1,5) 和 f(2,4)

i=4 时，f(4-4,5)和 f(4-1,4) 即 f(0,5)和 f(3,4)

如果我们用数形结合画成图，将是以下的现象

f(4,5)的值，将会由第 5-1 列，第 5 列相同颜色连线的格子的较大值+1 后，不同颜色取最小值来决定



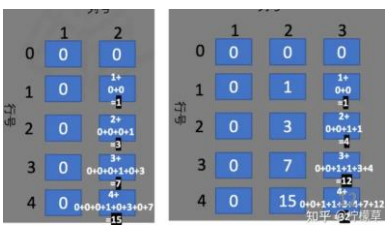
g 函数:g 函数是典型的动态规划的写法。根据经验，动态规划和有返回值的搜索经常可以划等号因此很有可能和 f 函数做同样的事情。同样把 i 当成行，j 当成列，对每句话说人话。经过了 f 函数的数形结合分析，可以发现 g 函数和 f 函数是一样的效果。

22.当输入为“7 3”时，第 19 行用来取最小值的 min 函数执行了 449 次。( )

正确答案：错误

解析：根据刚才的分析，f(3,2)的值由 A<->F，C<->D，E<->B，这三对的最小值决定。而 f(3,2)里 min 的执行次数，将是本函数 min 的次数 3(因为有三对斜线)+ 调用的所有子函数 ABCDEF 里 min 的执行次数之和。

以下我们可以手推一下 min 的执行次数(注意以下方块里的值并不是 f(n,m)，而是 f(n,m)里 min 的执行次数)



顺着规律爆算下去(其实计算过程中很多都可以重复利用)，n=7 时，答案为 448.或者可以总结出一些递推公式。比如 m=2，规律就是 2^n-1。m=3 也可以试着自己找到递推公式。

23.输出的两行整数总是相同的。()

正确答案:正确

解析：根据上述分析，确实答案一样。

24.当 m 为 1 时，输出的第一行总为 n。()

正确答案：正确

解析:送分题。根据第 14 行，m=1 时 return n。

25.算法 g(n,m)最为准确的时间复杂度分析结果为( )。

A. $O(n^{3/2}m)$ B.O(nm) C. O(nm) D.$O(nm^2)$

正确答案：C

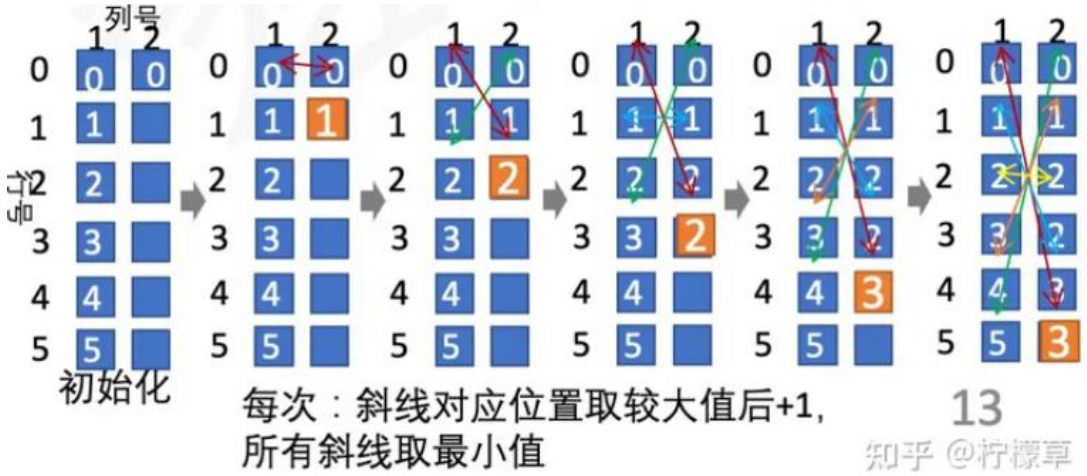
解析：观察 g 函数的构造，发现是由三层嵌套循环构成，循环次数分别为 n，m，n，因此时间复杂度是 $O(n^2m)$

26.当输入为“20 2”时，输出的第一行为( )

A.4 B.5 C.6 D.20

正确答案：C

解析:m=2。因为 m=1 直接有答案，因此需要算的就是第 2 列。虽然 n=20 很大，但是可以把 n 从小到大逐步计算，同时观察规律。



依次计算下去，你会意识到第 2 列会按 1，2，2，….的规律变化下去。即 3、3.3.4.4.4.1 个 1，2 个 2，3 个 3，4 个 4..这样我们会发现因此答案可以选 6。

27.(4 分)当输入为“108 100”时，输出的第一行为( )

A.6　B.7　C“8”　D.9

正确答案：B

解析：当鸡蛋数很多时(m 很大)，次数会很小。观察数值 100 和选项，我们可以想到高楼扔鸡蛋问题的另一个做法:二分。即我们可以二分 100，看能二分几次，就是最多测试多少次。测试层数依次为 mid=50(1-100 范围内)，mid=75(51-100 范围内)，mid=88(76-100 范围内)mid= 94(89-100 范围内)，mid=97(95-100 范围内)，mid=99(98-100 范围内)，mid=100(100-100 范围内)，共 7 次

第三题

(3)

```
01 #include <iostream>
02
03 using namespace std;
04
05 int n, k;
06
07 int solve1()
08 {
09     int l = 0, r = n;
10     while (l <= r) {
11         int mid = (l + r) / 2;
12         if (mid * mid <= n) l = mid + 1;
13         else r = mid - 1;
14     }
15     return l - 1;
16 }
17
18 double solve2(double x)
19 {
20     if (x == 0) return x;
21     for (int i = 0; i < k; i++)
22         x = (x + n / x) / 2;
23     return x;
24 }
```

```
18 double solve2(double x)
19 {
20     if (x == 0) return x;
21     for (int i = 0; i < k; i++)
22         x = (x + n / x) / 2;
23     return x;
24 }
25
26 int main()
27 {
28     cin >> n >> k;
29     double ans = solve2(solve1());
30     cout << ans << ' ' << (ans * ans == n) << endl;
31     return 0;
32 }
```

假设 int 为 32 位有符号整数类型，输入的 n 是不超过 47000 的自然数、k 是不超过 int 表示范围的自然数，

完成下面的判断题和单选题

28.该算法最准确的时间复杂度分析结果为 0(logn+k)。( )

29.当输入为“9801 1”时，输出的第一个数为“99”。 ( )

30.对于任意输入的 n，随着所输入 k 的增大，输出的第二个数会变成“1”。( )

31.该程序有存在缺陷。当输入的 n 过大时，第 12 行的乘法有可能溢出，因此应当将 mid 强制转换为 64 位整数再计算。()

单选题

32.当输入为“2 1”时，输出的第一个数最接近(

A. 1 B. 1.414 C.1.5 D.2

33.当输入为“3 10”时，输出的第一个数最接近( )。

A. 1.7 B. 1.732 C.1.75 D. 2

34.当输入为“256 11”时，输出的第一个数( )。

A. 等于 16 B 接近但小于 16

C.接近但大于 16 D.前三种情况都有可能

题目解析:首先观察程序，发现 solve1 函数是一个单纯的二分查找，solve2 函数也只是一个一层循环。虽然在 solve2 在做什么可能一头雾水，但是我们可以手动模拟一下 solve1。观察 solve1 之和输入的 n 有关，而且出现了 mid*mid<=n 的描述，推测和平方根有关系。为了加以验证，我们可以举个例子，比如 n=9，代入进行验算

l=0，r=9，mid=4，if不成立，r变成4-1=3
l=0，r=3，mid=1，if成立，l变成1+1=2
l=2，r=3，mid=2，if成立，l变成2+1=3
l=3，r=3，mid=3，if成立，l变成3+1=4
l=4，r=3，while结束，return 3。

发现果然收敛定位到了 9 的平方根处，这是正好有平方根的时候，为了严谨，我们可以再举一个 n=8 的例子，看看最终答案是根号 8 上取整还是下取整。

因此，solve1 的意义是找到 √n。(没有整数平方根时，返回 √n)

接下来分析 solve2 的意义

发现 k 仅仅代表循环次数，不参与循环。

因此 x 不断进行自我迭代。为了发现它的意义，

依然代入具体数值找规律。比如 n=9 时，

solve1 求得 x=3

x=(x+n/x)/2=(3+3)/2=3

x=(x+n/x)/2=(3+3)/2=3

那当 n 没有整数平方根时呢?

比如我们可以举单选题 33 题里的 n=3，solve1 求得 x 是=1 的，

x=(x+n/x)/2=(1+3/1)/2=2

x=(x+n/x)/2=(2+3/2)/2=1.75

x=(x+n/x)/2=(1.75 + 3/1.75)/2=1.732

如果对√3 的值熟悉，会知道它大概是 1.73 左右。可想而知，再算下去，答案也只会在 1.73 左右了经过演算，我们可以渐渐感知到，如果 n 没有整数平方根，那么就会由[、n]以及和它对称的 n/Wn]进行平均，更新 x，有一种调和的功效，从而达到越来越接近√n 的效果!

28.该算法最准确的时间复杂度分析结果为 O(logn+k)。( )

正确答案：正确

解析:solve1 二分的时间复杂度是 O(logn)，solve2 是循环 k 次，O(k)。总时间复杂度确实是 O(logn+k)。

29.当输入为“9801 1”时，输出的第一个数为“99”。 ( )

正确答案：正确

解析:根据题意分析，先看看 9801 有没有整数平方根，发现 99*99 正好就等于 9801，因此不管 k 是多少，最后一定输出 99.

30.对于任意输入的 n，随着所输入 k 的增大，输出的第二个数会变成“1”。( )

正确答案：错误

解析:看似合理，但是如果没有整数平方根还是会有一定的浮点误差。有一说和有理数无理数有关

31.该程序有存在缺陷。当输入的 n 过大时，第 12 行的乘法有可能溢出，因此应当将 mid 强制转换为 64 位整数再计算。

正确答案：错误

解析:看似合理，因为 n 在 47000 以内，如果是普通二分，那么最坏可能出现类似 mid=(47000+47000)/2，再平方约为 22 亿,确实超过 int。但是本题二分检查的第一个 mid 为(0+47000)/2=2 万多，它不可能继续向右二分会迅速收敛到左侧更小的数值，因此并不会超过 int。

32.当输入为“2 1”时，输出的第一个数最接近(

A. 1　　B. 1.414　　C.1.5　　D.2

正确答案：C

解析:2 没有整数平方根，因此 x=1，代入 solve2 :x=(x+n/x)/2=(1+2/1)/2=1.5

33.当输入为“3 10”时，输出的第一个数最接近()

A. 1.7　　B. 1.732　　C.1.75　　D.2

正确答案：B

解析:上述解析已经分析过此例子,第 3 次迭代已经接近 1.732

34.当输入为“256 11”时，输出的第一个数( )。

A. 等于 16　　B 接近但小于 16

C.接近但大于 16　　D.前三种情况都有可能

正确答案：A

解析:256 有整数平方根，为 16，因此 k 不管是多少，最终都输出 16。256 是 2 的整幂次方，也不会出现浮点误差。

三、完善程序(单选题，每小题 3 分，共计 30 分)

(枚举因数)从小到大打印正整数 n 的所有正因数

```
01 #include <bits/stdc++.h>
02 using namespace std;
03
04 int main() {
05   int n;
06   cin >> n;

07
08   vector<int> fac;
09   fac.reserve((int)ceil(sqrt(n)));
10
11   int i;
12   for (i = 1; i * i < n; ++i) {
13     if (①) {
14       fac.push_back(i);
15     }
16   }
17
18   for (int k = 0; k < fac.size(); ++k) {
19     cout << ② << " ";
20   }
21   if (③) {
22     cout << ④ << " ";
23   }
24   for (int k = fac.size() - 1; k >= 0; --k) {
25     cout << ⑤ << " ";
26   }
27 }
```

35.①处应填()

A.n%i==0　　B.n%i==1

C. n%(i-1)==0　　D.n%(i-1)==1

36.②处应填( )

A. n /fac[k]　　B.fac[k]

c.fac[k]-1　　D.n/(fac[k]-1)

37.③处应填()

A(i-1)*(i-1)== n　　B.(i-1)*i == n

Ci*i ==n　　D.i*(i-1)== n

38.④处应填 ()

A.n-i　　B. n-i+1

C.i-1　　D.i

39.⑤处应填()

A. n/fac[k]　　B.fac[k]

C.fac[k]-1　　D.n/(fac[k]-1)

程序分析

```
08   vector<int> fac; 定义fac列表
09   fac.reserve((int)ceil(sqrt(n)));
10   把fac列表的空间初始化为⌈√n⌉这么多
11   int i;
12   for (i = 1; i * i < n; ++i) {
13     if (①) {
14       fac.push_back(i);
15     }
16   }  找到√n之前的所有因数
17
18   for (int k = 0; k < fac.size(); ++k) {
19     cout << ② << " ";
20   }  输出√n之前的所有因数
21   if (③) {  看看n有没有整数平方根
22     cout << ④ << " ";
23   }
24   for (int k = fac.size() - 1; k >= 0; --k) {
25     cout << ⑤ << " ";  按顺序
26   }                    输出√n之后的所有因数
```

35.①处应填()

A.n%i==0　　B.n%i==1

C. n%(i-1)==0　　D.n%(i-1)==1

正确答案:A

解析:找到√n 之前的因数。

36.②处应填( )

A. n /fac[k]　　B.fac[k]

c.fac[k]-1　　D.n/(fac[k]-1)

正确答案：B

解析:输出列表里存储的√n 之前的因数。

37.③处应填()

A(i-1)*(i-1)== n　　B.(i-1)*i == n

Ci*i ==n　　D.i*(i-1)== n

正确答案：C

解析:如果 n 有整数平方根，输出一次。第 12 句的 i*i<n 不满足循环才会停止，因此循环停止后的 i 一定>= √n。

38.④处应填 ()

A.n-i　　B. n-i+1

C.i-1　　D.i

正确答案：D

解析:循环停止后的 i 一定>=√n。如果有整平方根，那么 i 此时一定就是√n 。

39.⑤处应填()

A. n/fac[k]　　B.fac[k]

C.fac[k]-1　　D.n/(fac[k]-1)

正确答案：A

解析:和 fac[k]对应的因数是 n/fac[k]。

(洪水填充)现有用字符标记像素颜色的 8x8 图像。颜色填充的操作描述如下:给定起始像素的位置和待填充的颜色，将起始像素和所有可达的像素(可达的定义:经过一次或多次的向上、下、左、右四个方向移动所能到达且终点和路径上所有像素的颜色都与起始像素颜色相同)，替换为给定的颜色。

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

const int ROWS = 8;
const int COLS = 8;

struct Point {
  int r, c;
  Point(int r, int c) : r(r), c(c) {}
};

bool is_valid(char image[ROWS][COLS], Point pt,
              int prev_color, int new_color) {
  int r = pt.r;
  int c = pt.c;
  return (0 <= r && r < ROWS && 0 <= c && c < COLS &&
          ① && image[r][c] != new_color);
}

void flood_fill(char image[ROWS][COLS], Point cur, int new_color) {
  queue<Point> queue;
  queue.push(cur);

  int prev_color = image[cur.r][cur.c];
  ②;

  while (!queue.empty()) {
    Point pt = queue.front();
    queue.pop();

    Point points[4] = {③, Point(pt.r - 1, pt.c),
                       Point(pt.r, pt.c + 1), Point(pt.r, pt.c - 1)};
    for (auto p : points) {
      if (is_valid(image, p, prev_color, new_color)) {
        ④;
        ⑤;
      }
    }
  }
}
```

```
int main() {
  char image[ROWS][COLS] = {{'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g'},
                            {'g', 'g', 'g', 'g', 'g', 'g', 'r', 'r'},
                            {'g', 'r', 'r', 'g', 'g', 'r', 'g', 'g'},
                            {'g', 'b', 'b', 'b', 'b', 'r', 'g', 'r'},
                            {'g', 'g', 'g', 'b', 'b', 'r', 'g', 'r'},
                            {'g', 'g', 'g', 'b', 'b', 'b', 'b', 'r'},
                            {'g', 'g', 'g', 'g', 'g', 'b', 'g', 'g'},
                           {'g', 'g', 'g', 'g', 'g', 'b', 'b', 'g'}};

  Point cur(4, 4);
  char new_color = 'y';

  flood_fill(image, cur, new_color);

  for (int r = 0; r < ROWS; r++) {
    for (int c = 0; c < COLS; c++) {
      cout << image[r][c] << " ";
    }
    cout << endl;
  }

// 输出:
// g g g g g g g g
// g g g g g g r r
// g r r g g r g g
// g y y y y r g r
// g g g y y r g r
// g g g y y y y r
// g g g g g y g g
// g g g g g y y g

  return 0;
}
```

40.①处应填()

A.image[r][c]== prev_color

B.image[r][c]!= prev_color

C.image[r][c]== new_color

D.image[r][c]!= new_color

41.②处应填()

A.image[cur.r+1][cur.c]=new_color

B.image[cur.r][cur.c]=new_color

C.image[cur.r][cur.c+1]=new_color

D.image[cur.r][cur.c]= prev_color

42.③处应填( )

A. Point(pt.r, pt.c)　　B. Point(pt.r,pt.c+1)

C. Point(pt.r+1,pt.c)　　D. Point(pt.r+1,pt.c+1)

43.④处应填()

A.prev_color = image[p.r][p.c]

B.new_color= image[p.r][p.c]

C.image[p.r][p.c]= prev_color

D.image[p.r][p.c]= new color

44.⑤处应填()

A.queue.push(p)　　B.queue.push(pt)

C.queue.push(cur)　　D.queue.push(Point(Rows,COLs))

程序解析：

题意分析:本题是经典问题洪水填充问题，根据 43 行的输入和 63 行的输出，进一步验证了本题的目的。程序是由广搜实现。

40.①处应填()

A.image[r][c]== prev_color

B.image[r][c]!= prev_color

C.image[r][c]== new_color

D.image[r][c]!= new_color

正确答案：A

解析:不越界，颜色相等，不重复走，这三个条件中还没有体现颜色相等，因此选 A。

41.②处应填()

A.image[cur.r+1][cur.c]=new_color

B.image[cur.r][cur.c]=new_color

C.image[cur.r][cur.c+1]=new_color

D.image[cur.r][cur.c]= prev_color

正确答案：B

解析:④是为每个邻居涂色，而起点本身需要在这里单独涂色。

42.③处应填( )

A. Point(pt.r, pt.c)　　B. Point(pt.r,pt.c+1)

C. Point(pt.r+1,pt.c)　　D. Point(pt.r+1,pt.c+1)

正确答案：B

解析:这里是在找上下左右四个方向的邻居缺少了 C 选项。

43.④处应填()

A.prev_color = image[p.r][p.c]

B.new_color= image[p.r][p.c]

C.image[p.r][p.c]= prev_color

D.image[p.r][p.c]= new color

正确答案：D

解析:对邻居 p 进行涂色。

44.⑤处应填()

A.queue.push(p)　　B.queue.push(pt)

C.queue.push(cur)　　D.queue.push(Point(Rows,COLs))

正确答案：A

解析:将邻居 p 入队,让邻居等待搜索