# 快快编程模拟试卷 6

## 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. 已知一维数组定义 a：array[1...10000] of int。每个元素占 4 个字节地址。已知 a[1]的开始地址为内存中第 10000 个字节处，请问 a[2020]的开始地址是第几个字节：（　　）。
A. 18072　　B. 18076　　C. 12019　　D. 12020

答案：B
解释：数据元素本身连续存储，每个元素所占的存储单元大小固定相同，元素的下标是其逻辑地址，而元素存储的物理地址是元素实际的内存地址。可以通过存储区起始地址，加上逻辑地址与存储单元大小（c）的乘积计算得到。即 $L_0+(n-1)*c$ 得到该元素物理地址。10000+(2020-1)*4=10000+2019*4=18076。

2. 在 ASCII 码表中，根据码值由小到大的排列顺序是（　　）。
A. 空格字符、数字符、大写英文字母、小写英文字母
B. 数字符、空格字符、大写英文字母、小写英文字母
C. 空格字符、数字符、小写英文字母、大写英文字母
D. 数字符、大写英文字母、小写英文字母、空格字符

答案：A
解析：由 ASCII 码值表可知，其大小顺序由小到大依次是：空格字符、数字符、大写英文字母、小写英文字母。

3. 由四个完全没有区别的点构成的简单无向连通图的个数是（　）。
A. 小于 5 个　　B. 5 个　　C. 6 个　　D. 大于 6 个

答案：C
解释：根据边数来分类判断：小于 3 条边，不构成连通，排除掉。3 条边：d（度数，也即各点所连接的边数）= [1,2,2,1] 和 [1,3,3,1] 两种（注意因为点是完全没有区别的，所以排列顺序不重要）。4 条边：d=[2,2,2,2]和[1,3,2,2]两种。5 条边：d=[2,2,3,3]一种。6 条边：d=[3,3,3,3]一种。

4. 小明在递归函数内部定义了一个长度为 2020 的 long long 数组，但编译运行时却频频报错，请问最有可能的原因是（　）。
A. 没有缩进
B. 程序效率太低导致超时
C. 数组命名中包含了大写字母
D. 递归层次过多导致栈空间不足

答案：D
解释：编译运行时频频报错可能是递归层次过多。递归调用过程定义了一个长度为 2020 的 long long 数组，也就是每一递归都会新建数组，会占用大量空间导致栈空间不足。

5. 不同类型的存储器组成了多层次结构的存储器体系，按存取速度从快到慢的排列是（　　）。
A. 快存/辅存/主存
B. 外存/主存/辅存
C. 快存/主存/辅存
D. 主存/辅存/外存
答案：C
解析：快存相当于 cache，是为了解决 CPU 和主存之间的速度匹配问题。主存也即内存，辅存既是外存，访问速度由快到慢依次是快存、主存、辅存。

6. 有两个三口之家一起出行去旅游，他们被安排坐在两排座位上，其中前一排有 3 个座位，后面一排有 4 个座位。如果同一个家庭的成员只能被安排在同一排座位并必须相邻而坐，那么共有(　　)种不同的安排方法。
A. 36　　B. 72　　C. 144　　D. 288
答案：C
解析：捆绑法。因为是两个不同的家庭，所以哪个家庭坐在三人一排的位置，哪个家庭坐在四人一排的位置，共有 A(2,2)=2 种排列方式，对于坐到三人一排的家庭，其家庭内部还有 A(3,3)=6 种坐法；对于坐到四人一排的家庭，由于每一个人要相邻而坐，所以将 3 个人捆绑看成一个整体，将四个椅子中的相邻三个捆绑在一起，于是共有 A(2,2)=2 种坐法，三人内部共有 A(3,3)=6 种坐法，因此共有 2×6×2×6=144 种坐法。因此选择 C。

7. 有 A，B 和 C 三根柱子，开始时 n 个大小互异的圆盘从小到大叠放在 A 柱上，现要将所有圆盘从 A 移到 C，在移动过程中始终保持小盘在大盘之上，求移动盘子次数的最小值。这类问题叫汉诺塔（Hanoi）问题。求问：对于 n=2020 个圆盘的 Hanoi 塔问题，总的最少移动次数为（　　）。
A. $2^{2019}$　　B. $3^{2020}$　　C. $2^{2020}-1$　　C. $2^{2020}$
答案：C
解析：首先证明 Hanoi 问题的递推公式。首先被移动到 C 盘的必定是最大盘子，否则必定违反“在移动过程中始终保持小盘在大盘之上”的规定。既然要将最大盘移动到 C，此时最大盘之上必定没有任何盘子，也就是即它独自在一根柱子上，要做到这点最优做法是先把较小的 n-1 个盘子从 A 移动到 B，剩下最大盘独自在 A。将 n-1 个盘由 A 移动到 B 花费的最少次数为 H(n-1)。此时再将最大盘由 A 移到 C，移动次数为 1 次。接着把剩下的 n-1 个盘从 B 移动到 C，花费最少次数当然也是 H(n-1)。于是得到总移动次数 2H(k-1)+1。证得 H(k)=2H(k-1)+1。然后通过递推公式可推导通项公式：由 H(k)=2H(k-1)+1 得 H(k)+1=2(H(k-1)+1)，于是{H(k)+1}转换成首项为 H(1)+1=2、公比为 2 的等比数列，求得 $H(k)+1 = 2^k$，所以 $H(k) = 2^k-1$。

8. 中缀表达式(A+B)*(C*(D+E)+F)的后缀表达式是(　　)。
A. ABC*DE+F+*　　B. A+BCDE+*F+*　　C. AB+C+*DEF+*　　D. AB+CDE+*F+*
答案：D
解析：方法 1：先按照运算符的优先级对中缀表达式加括号，变成：((A+B)*((C*(D+E))+F))。

然后，将运算符移到括号后面，变成：((AB)+((C(DE)+)*F)+)*，去掉括号，得到 AB+CDE+*F+*。
方法 2：可以首先将表达式转为二叉树，然后后序遍历获得后缀表达式。

9. 快速排序（Quicksort），又称分区交换排序（partition-exchange sort），简称快排，是一种排序算法，最早由东尼·霍尔提出。在平均状况下，排序 n 个项目需要 O(nlogn)次比较。在最坏状况下则需要 $O(n^2)$次比较，但这种状况并不太常见。现在，对给定的整数序列（541,132,984,746,518,181,946,314,205,827）进行从小到大排序时，我们采用快速排序（以中间元素 518 为基准数），请问第一趟扫描的结果是（　）。
A. (181,132,314,205,541,518,946,827,746,984)
B. (541,132,827,746,518,181,946,314,205,984)
C. (205,132,314,181,518,746,946,984,541,827)
D. (541,132,984,746,827,181,946,314,205,518)
答案：C
解析：根据快速排序的规则，左侧小区中所有数要保证不大于 518，右侧大区中所有数要保证不小于 518。

10. 10000 以内，与 10000 互质的正整数有（　）个。
A. 2000　　B. 4000　　C. 6000　　D. 8000
答案：B
解析：10000 由大量的 2 和 5 相乘得到。因此，所有具有因子 2 的数字或者具有因子 5 的数字，都不与 10000 互质。1-10000 之间，具有因子 2 的数字有 5000 个，具有因子 5 的数字有 2000 个，同时具有因子 2 和 5 的数字有 1000 个，因此被 2 或 5 整除的数字共有 5000+2000-1000=6000 个。故 10000 以内与 10000 互质的正整数有 4000 个。

11. 根节点深度为 0，一颗深度为 h 的满 k（k>1）叉树，也就是说，这棵树除最后一层即叶子层无任何子节点外，每一层上的所有结点都挂满了 k 个子结点。请问这棵树共有（　）个结点。提示：等比数列的求和公式是：$(a_0-a_n*q)/(1-q)$，其中 q 是等比数列的公比，$a_0$ 是首项，$a_n$ 是数列的末项。
A. $(k^{h+1}-1)/(k-1)$
B. $k^{h-1}$
C. $k^h$
D. $k^{h-1}/(k-1)$
答案：A
解析：在本题中，首项是 $k_0$，末项是 $k_h$，公比是 k。应计算的数列是：$k^0+k^1+k^2+k^3+...+k^h=(k^0-k^h*k)/(1-k)$，整理可以得到 A。本题也可以代入特殊值验算。

12. 一个 16 位带符号的整数二进制补码为 1111111111111101，其表示的十进制整数是（　）。
A.-1　　B.-2　　C.-3　　D.-4
答案：C
解析：最高位符号位 1 表示负数。设该数字为 x，其二进制原码为 1[x]的 15 位二进制表示。负数的补码=反码+1,其反码为 111111111111100,反码等于[x]的 15 位二进制位取反，[x]的 15 位二进制表示为 000000000000011，转换成十进制为 3。

13. 计算机术语“中断”是指（　）。
A. 操作系统随意停止一个程序运行
B. 当出现需要时，CPU 暂时停止当前程序的执行，转而执行处理新的情况的过程
C. 因停机而停止一个程序的运行
D. 电脑死机
答案：B
解析：计算机常识题。中断是指：当出现需要时，CPU 暂时停止当前程序的执行，转而执行处理新的情况的过程。

14. 图 G 是一个有序二元组（V，E），其中 V 称为顶点集（Vertices Set），E 称为边集（Edges set）。全部由有方向性的边所构成的图，称为有向图（Directed Graph）。在所有与图中某个点 A 关联的边中，以 A 为起点的边的条数称为出度，而以 A 为终点的边的条数则称为入度。请问：有向图中每个顶点的度等于该顶点的（　）。
A. 入度
B. 出度
C. 入度与出度之和
D. 入度与出度之差
答案：C
解析：涉及图论知识，有向图顶点的度（degree）等于入度（in-degree）与出度（out-degree）之和。

15. 哈夫曼树（Huffman Tree）是在叶子结点和权重确定的情况下，带权路径长度（WPL）最小的二叉树，也被称为最优二叉树。而哈夫曼树所生成的二进制编码，就是哈夫曼编码（Huffman Coding）。这种编码方式由 MIT 的哈夫曼博士发明。它实现了两个目标：一，保证任何字符编码，都不是其他字符编码的前缀；二，通过 WPL 最小保证了信息编码的总长度最小。假设在某次通信中，只用了 5 个字母，a、b、c、d、e，它们出现的频次分别是{2,8,5,4,6}，对其哈夫曼编码的结果是（　）。
A. {a:010;b:11;c:00;d:011;e:10}
B. {a:00;b:11;c:010;d:011;e:10}
C. {a:010;b:11;c:011;d:00;e:10}
D. {a:11;b:010;c:00;d:011;e:10}
答案：A
解析：给 n 个点，每个点都有权值（出现频度），构造一颗哈夫曼树。每次选剩下的两棵根权值最小的数合并成一颗新树，新树的根权值等于两棵合并前树的根权值和，从哈夫曼树根结点开始，对左子树分配码 0，对右子树分配码 1，一直到达叶子结点为止，然后将从树根沿每条路径到达叶子结点的代码排列起来，得到哈夫曼编码，答案是 A。

## 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  using namespace std;
```

```
3  int main() {
4      int n,n1,n2;
5      int ans=0;
6      int flag=1;
7      int i=2;
8      cin>>n1>>n2;
9      n=n1+n2;
10     while(i*i<=n){
11         if(n%i==0){
12             flag=0;
13         }
14         i++;
15     }
16     ans=n1*n2*flag;
17     cout<<ans<<endl;
18     return 0;
19 }
```

判断题：

1. 程序第 5 行定义整型变量 ans 时，如果去掉初始化为 0 的赋值操作，对本程序输出结果没有任何影响。（　　）　√

2. 输入到 n1 和 n2 的值，不允许同时超过 1e9。（　　）　×

3. 将程序第 7 行定义的整型变量 i 初始化为 1，程序输出结果和修改前一致。（　　）　×

4. 在程序 12 行和 13 行之间插入一行 break 语句，能够在不影响程序输出结果的前提下，减少程序运行的总耗时。（　　）　√

选择题：

5. 对第 10 行 while 循环内条件改写成以下哪种形式，程序输出结果一定不变。（　　）

A. i*i*i<=n　　B. i<=n　　C. i<n　　D. i*i>=n

**答案：C**

6. 若输入以下哪组 n1 和 n2，程序最终输出结果不是 0。（　　）

A. 4 5　　B. 21 12　　C. 100 189　　D. 45 52

**答案：D**

**解析：如果输入的两个数的和是质数，那么结果是输出这两个数的乘积，否则结果是 0。**

2.

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  string s;
4  long long magic(int l, int r){
```

```
5       long long ans = 0;
6       for (int i = l; i <= r; ++i)
7           ans = ans * 4 + s[i]-'a'+1;
8       return ans;
9   }
10  int main(){
11      cin >> s;
12      int len = s.length();
13      int ans = 0;
14      for (int l1 = 0; l1 < len; ++l1)
15          for (int r1 = l1; r1 < len; ++r1){
16              bool bo = true;
17              for (int l2 = 0; l2 < len; ++l2)
18                  for (int r2 = l2; r2 < len; ++r2)
19                      if (magic(l1,r1)==magic(l2,r2)&&(l1!=l2||r1!=r2))
20                          bo = false;
21              if (bo) ans += 1;
22          }
23      cout << ans << endl;
24      return 0;
25  }
```

判断题：

1. 根据程序，如果输入字符串 helloworld，第 12 行整型变量 len 会通过 length 函数的返回值，被初始化为 9。（　　）　**×**

2. 假设输入字符串长度为 n，那么本程序因为主函数中包含一个四重 for 循环，因此总时间复杂度可以表示为 $O(n^4)$。（　　）　**×**

3. 将本程序中 16 行语句放到 13 行和 14 行之间，对输出结果没有影响。（　　）　**×**

4. 程序运行过程中，即使 19 行 if 判断条件成立，bo 被置为 false，也会继续进行 17-18 行的双重 for 循环，直到整个字符串的子段被枚举完毕。（　　）　**√**

**解析：本程序对于任意的一个子段[l1,r1]，只要检测到，任何一个子段[l2,r2]，可以使得 bo 被置为 false（也就是同时符合：①两子段 magic 值相等，②两子段不同），就不会统计到 ans 里。换句话说，ans 统计的是字符串中独一无二的子串。**

选择题：

5. 输入 abacaba，输出结果是（　　）

A. 7　　B. 7!　　C. 3　　D. 16

**答案：D**

6. 如果将第 7 行修改为 ans=ans*1+s[i]-'a'+1, 输入 abacaba, 输出结果与修改前相比(　　)。
A.会变小　　B. 会增大　　C. 程序会报错　　D. 不变
**答案：A**

3.

```
1   #include<iostream>
2   #include<cstring>
3   using namespace std;
4   int main() {
5       string a1, b1;
6       int a[109], b[109], c[109];
7       int lena, lenb, lenc;
8       int i, j, x;
9       cin >> a1 >> b1;
10      memset(a, 0, sizeof(a));
11      memset(b, 0, sizeof(b));
12      memset(c, 0, sizeof(c));
13      lena = a1.length();
14      lenb = b1.length();
15      for(i = 0; i <= lena - 1; i++)
16          a[lena - i] = a1[i] - '0';
17      for(i = 0; i <= lenb - 1; i++)
18          b[lenb - i] = b1[i] - '0';
19      for(i = 1; i <= lena; i++) {
20          x = 0;
21          for(j = 1; j <= lenb; j++) {
22              c[i + j - 1] += a[i] * b[j] + x;
23              x = c[i + j - 1] / 10;
24              c[i + j - 1] = c[i + j - 1] % 10;
25          }
26          c[i + lenb] = x;
27      }
28      lenc = lena + lenb;
29      while(c[lenc] == 0 && lenc > 1)
30          lenc--;
31      for(i = lenc; i >= 1; i--)
32          cout << c[i];
33      cout << endl;
34      return 0;
35  }
```

判断题:
1. 当使用 memset 函数对一个整型数组进行整体地初始化时，通常只能赋值为 0 或-1，因

为 memset 函数是按字节（byte）赋值，对每个字节赋值与同样的值。（　　）　√

2. 可以把 20 行整型变量 x 初始化为 0 的操作，放在 19 行 for 循环的前面，对整个程序输出结果不会有影响。（　　）　×

选择题：

3. 输入 1234　45678，那么 a[1]和 b[4]的值分别是（　　）。

A. 1 4　　B. 4 4　　C. 4 8　　D. 4 5

**答案：D**

**解析：a[1]就是输入第一个数字的最低位（个位），值为 4；b[4]就是输入第一个数字从右往左数的第 4 位（千位），值为 5。**

4. 输入 1234　45678，程序首次执行到第 26 行时，等价于下面的表达式（　　）。

A. c[7]=1　　**B. c[6]=1**　　C. c[7]=2　　D. c[6]=2

**答案：B**

**解析：第一次执行到 26 行时，i=1，lenb=5，所以是给 a[6]赋值；尝试手算计算可知，4*45678=182712，最高位的进位值是 1，所以等价于是 c[6]=1。**

5. 本程序所实现功能可以概括为（　　）

A. 大整数加法　　B. 大整数乘法　　C. 大整数乘方　　D. 大整数开方

**答案：B**

**解析：本题实现了两个大整数的乘法运算。**

6.（4 分）输入 202020192018　202220212020，**将 29 行 while 改成 if**，括号内判断条件不变，程序输出结果（　　）。

A. 不变　　B. 变大　　C. 变小　　D. 可能导致溢出

**答案：A**

**解析：已知 lenc=lena+lenb。同时注意到：两数相乘，数组 c 的第 lena+lenb-1 位，是由第一个数字的最高位（第 lena 位）上的值，乘以第一个数字的最高位（第 lenb 位）上的值所得到的，所以一定是非零的值。所以只可能第一次 lenc== 0，lenc--后，不可能再为 0，因此 for 和 if 效果一致。**

## 三、完善程序（单选题，每小题 3 分，共计 30 分）

1. （素数环）从 1 到 10 这 10 个数摆成一个环，要求相邻的两个数的和是一个素数。从 1 开始，每个空位有 10 种可能，需要填进去的数合法，即与前面的数不相同，同时与左侧相邻的数之和是一个素数。另外在第 10 个数时还要判断和第 1 个数之和是否是素数。输出每一种可能的排列。

```
1  #include<iostream>
2  #include<cstdio>
3  #include<cstdlib>
4  using namespace std;
5
```

```
bool b[11] = {0};
int total = 0;
int a[11] = {0};

bool prime(int x, int y) {
    int k = 2;
    int i = ____(1)____;
    while(____(2)____)
        k++;
    if(k * k > i)
        return 1;
    else
        return 0;
}

int out() {
    total++;
    cout << " <" << total << "> ";
    for(int j = 1; j <= 10; j++) {
        cout << a[j] << " ";
    }
    cout << endl;
}

void search(int t) {
    int i;
    for(i = 1; i <= 10; i++) {
        if(!b[i] && prime(a[t - 1], i)) {
            a[t] = i;
            b[i] = 1;
            if(t == 10) {
                if(prime(a[____(3)____], a[1]))
                    out();
            }
            else
                ____(4)____;
            ____(5)____;
        }
    }
}

int main() {
    search(1);
    cout << total << endl;
```

```
    return 0;
}
```

1. （1）处应填（　　）。
A. x*y　　B. x+y　　C. x%y　　D. x<<y

**答案：B**
**解析：本函数 prime 要计算两参数 x 和 y 相加之和是否为质数。**

2. （2）处应填（　　）。
A. k*k<=i&&i%k!=0　B. k*k<=i&&i%k==0　　C. i%k==0　　D. k*k<=i

**答案：A**
**解析：在可能的因子范围内 k∈[2, floor(sqrt(i))]，检查是否有一个 k 是 i 的因子。因此满足 k*k<=i&&i%k!=0 时，继续往前检查。**

3. （3）处应填（　　）。
A. 1　　B. 0　　C. 9　　D. 10

**答案：D**
**解析：在第 10 个数时，需判断其与第一个数之和是否是素数。**

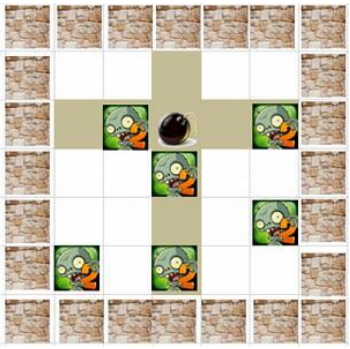
4. （4）处应填（　　）。
A. search(t)　　B. search(t+1)　　C. search(i)　　D. search(i+1)

**答案：B**
**解析：递归搜索的基本操作，当第 t 个位置选定后，在它基础上，递归搜索第 t+1 个位置的值。因此参数应该是**

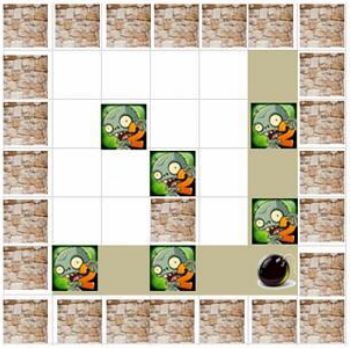
5. （5）处应填（　　）。
A. b[t]=0　　B. b[i]=0　　C. a[i]=t　　D. a[t]=0

**答案：B**
**解析：回溯法。**

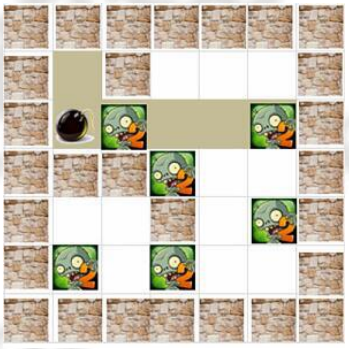
2. （大战僵尸）妈妈得知小新成功地解决了难题，奖励他去看电影《植物大战僵尸》，小新看着看着，就替女主角担心起来了，因为她要对付那么多的僵尸怪物，小新恨不得扔颗炸弹消除可恶的僵尸们，他脑袋里开始构思出这样的场景。



在一个 N 行 M 列单元格构成的地图中，去放置一个炸弹，这种炸弹威力巨大，以放置点为中心进行行列延伸炸到同行同列的僵尸，但不能穿墙。上图中可以把炸弹放置在**第 3 行第 4 列**，最多可以炸到 4 个僵尸，如果对地图稍加改动（如下图），在第 5 行第 4 列处加入一个墙体，又如何呢？答案其实还是最多炸到 4 个僵尸，只不过此时此刻，最佳炸弹放置点需要发生了变化，应该放到第 6 行第 6 列的位置。



当然炸弹要靠勇敢的小新去放，他只能在地图中朝上下左右四个方向行进（不能斜对角移动），**他不能穿墙，也不能穿越僵尸**，要保证他安全。如下图，告诉你小新起始位置是第 2 行第 2 列，那么他最佳放置炸弹位置应该是第 3 行第 2 列：最多炸到 2 个僵尸。



现在请聪明的你也一起加入到消除僵尸的队伍来。

**输入文件 boom.in**

第一行四个用空格隔开的正整数表示 N,M,X,Y，分别表示 N 行 M 列的地图，小新起始位置第 X 行，第 Y 列。

接下来 N 行 M 列用来描述地图上每个单元格，‘G’表示僵尸，‘#’表示墙体，只有‘.’表示的单元格才是小新能够正常行走，能够放置炸弹的单元格。（数据保证四面都是墙体，也就是第 1 行、第 N 行、第 1 列、第 M 列肯定都是墙体）。

**输出文件 boom.out**

输出一个整数，最多能炸掉的僵尸数量。

**输入样例**

13 13 4 2
#############
###..GG#GGG.#
###.#G#G#G#G#
#.....#..G#
#G#.###.#G#G#
#GG.GGG.#.GG#
#G#.#G#.#.#.#
#G…G…..#
#G#.#G###.#G#
#…G#GGG.GG#
#G#.#G#G#.#G#
#GG.GGG#G.GG#
#############

**输出样例**

10

**数据规模**

30%的数据，保证 N,M<14，并且小新一定能够抵达最佳炸弹放置点
40%的数据，保证 N,M<14
70%的数据，保证 N,M<101
100%的数据，保证 N,M<2001
100%的数据，保证 X<N
100%的数据，保证 Y<M

**小新苦心冥想，写出了如下代码，但有些地方还不太确定。请帮帮他，把代码补全。**

```
#include<bits/stdc++.h>
using namespace std;
const int SIZE=10009;
struct node{
    int x,y;
};
char a[SIZE][SIZE];
bool vst[SIZE][SIZE];
int ans=0;
int dx[4]={-1,1,0,0};
int dy[4]={0,0,-1,1};
node _next;
queue<node> q;
//上：该点之上能炸到几个僵尸
int _up(int x,int y){
    if(a[x][y]=='#') //障碍则停
```

```
        return 0;
    if(a[x][y]=='G') //炸到僵尸
        return ____(1)____;  //
    return _up(x-1,y);
}
//下：该点之下能炸到几个僵尸
int _down(int x,int y){
    if(a[x][y]=='#')
        return 0;
    if(a[x][y]=='G')
        return 1+_down(x+1,y);
    return _down(x+1,y);
}
//左
int _left(int x,int y){
    if(a[x][y]=='#')
        return 0;
    if(a[x][y]=='G')
        return 1+_left(x,y-1);
    return _left(x,y-1);
}
//右
int _right(int x,int y){
    if(a[x][y]=='#')
        return 0;
    if(a[x][y]=='G')
        return 1+_right(x,y+1);
    return _right(x,y+1);
}
//统计炸到僵尸数目
int count(int x,int y){
    return ____(2)____;
}
//广搜求最大值
void bfs(){
    while(!q.empty()){ //队列 q 为空则停止
        node now=q.front(); //队首元素
        ans=max(ans, count(now.x, now.y)); //打擂台
        for(____(3)____){ //int i=0;i<4;i++
            //四个方向扩展
            int r=now.x+dx[i];
            int c=now.y+dy[i];
            if(____(4)____){
                _next.x=r;
```

```
61                  _next.y=c;
62                  vst[r][c]=true; //标记已访问
63                  q.push(_next);
64              }
65          }
66          q.pop();
67      }
68  }
69
70  int main(){
71      freopen("boom.in","r",stdin);
72      freopen("boom.out","w",stdout);
73      int n,m,x,y;
74      cin>>n>>m>>x>>y;
75      //读入地图
76      for(int i=1;i<=n;i++)
77          for(int j=1;j<=m;j++)
78              cin>>a[i][j];
79      memset(vst,0,sizeof(vst)); //初始化标记用布尔数组
80      _next.x=x;
81      _next.y=y;
82      vst[x][y]=true;
83      q.push(_next);
84      bfs();
85      cout<<____(5)____<<endl;
86      return 0;
87  }
```

1. （1）处应填（　　）。

A. up(x-1,y)　　B. up(x+1,y)　　C. 1+_up(x-1,y)　　D. 1+_up(x,y+1)

**答案：C**

2. （2）处应填（　　）。

A. _up(x,y)+_down(x,y)+_left(x,y)+_right(x,y)

B. _up(x+1,y)+_down(x-1,y)+_left(x,y+1)+_right(x,y-1)

C. _up(x,y-1)+_down(x,y+1)+_left(x-1,y)+_right(x+1,y)

D. _up(x-1,y)+_down(x+1,y)+_left(x,y-1)+_right(x,y+1)

**答案：D**

3. （3）处应填（　　）。

A. int i=0;i<4;i++

B. int i=1;i<=4;i++

C. int i=x;i<=y;i++

D. !q.empty()

**答案：A**

4. （4）处应填（　　）。
A. !vst[r][c]&&a[r][c]=='G'
B. vst[r][c]&&a[r][c]=='#'
C. !vst[r][c]&&a[r][c]=='.'
D. vst[r][c]&&(a[r][c]=='#'|| a[r][c]=='G')
**答案：C**

5. （5）处应填（　　）。
A. vst[x][y]
B. a[x][y]
C. q.front()
D. ans
**答案：D**