初赛模赛

选择题

选择

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

解析：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L0+(n-1)*c得到该元素物理地址。
10000+(2020-1)*4=10000+2019*4=18076。

选择

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。

选择

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

解析：根据边数来分类：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。

选择

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

解析：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。

选择

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。

选择

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。

选择

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：Hanoi问题的递推公式：H(k)=2H(k-1)+1。通过递推公式可推导得到通项公式：由H(k)=2H(k-1)+1得H(k)+1=2(H(k-1)+1)，于是{H(k)+1}是首项为H(1)+1=2、公比为2的等比数列，可以求得$H(k)+1 = 2^k$，所以通项公式是：**$H(k) = 2^k-1$**。

选择

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，可以得到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。

选择

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。

选择

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。

选择

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。

选择

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。

选择

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。

选择

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。

选择

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，每个点对应一个权值（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，新树根的权值等于两棵合并前树的根权值**之和**，加入节点集合，以此类推。完成构建后，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**对应叶子结点**的代码排起来，得到对应点的哈夫曼编码。只有A符合。

哈夫曼编码

哈夫曼编码是一种数据压缩技术，具有如下特点：

- 出现概率高的字符使用较短的编码
- 出现概率低的字符使用较长的编码
- 使编码之后的码字的平均长度最短

在通信中只用到4个字母，a,b,c,d，它们出现的频率分别是{2,7,5,4}，请对其进行哈夫曼编码

编码过程：

1. 构建哈夫曼树
2. 进行哈夫曼编码

编码后

a:110

b:0

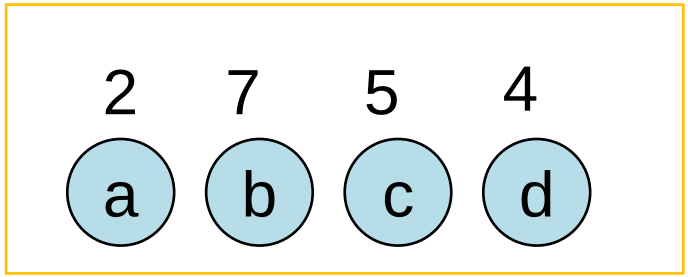
c:10

d:111

哈夫曼树

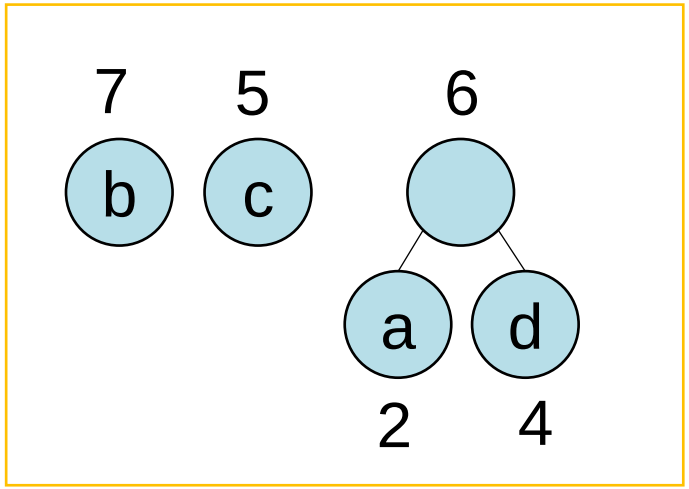
构造哈夫曼树：给n个点，每个点都有权值（出现频度），构造一棵哈夫曼树。每次选剩下的两棵根权值最小的树合并成一棵新树，新树的根权值等于两棵合并前树的根权值和。

第一步



每个节点作为一棵树根

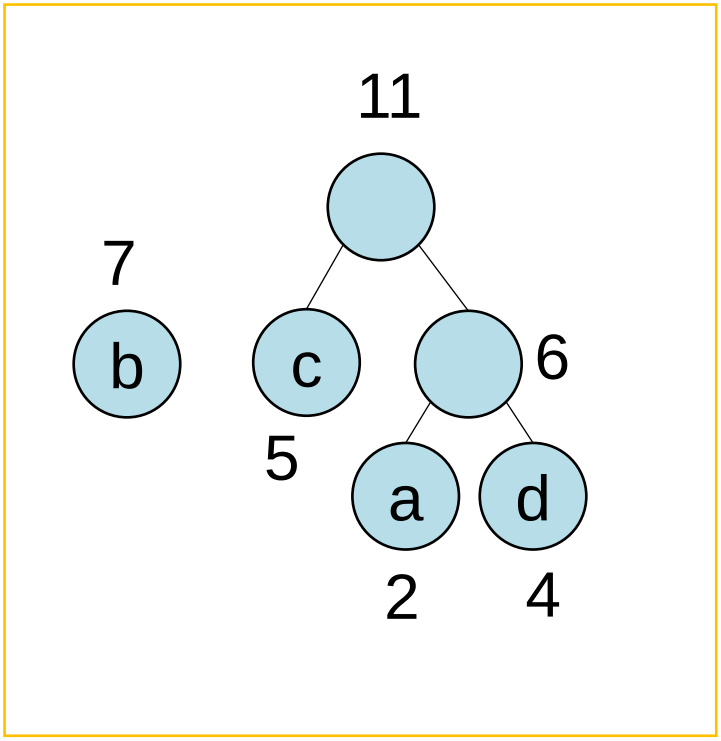
第二步



选根权值最小的两棵树2和4合并，新树的根节点为6

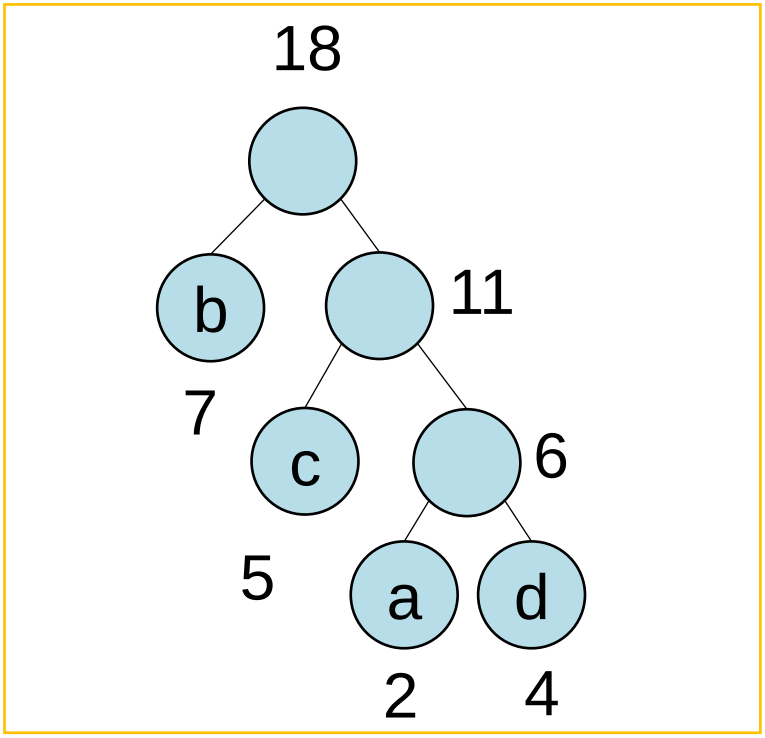
哈夫曼树

第三步



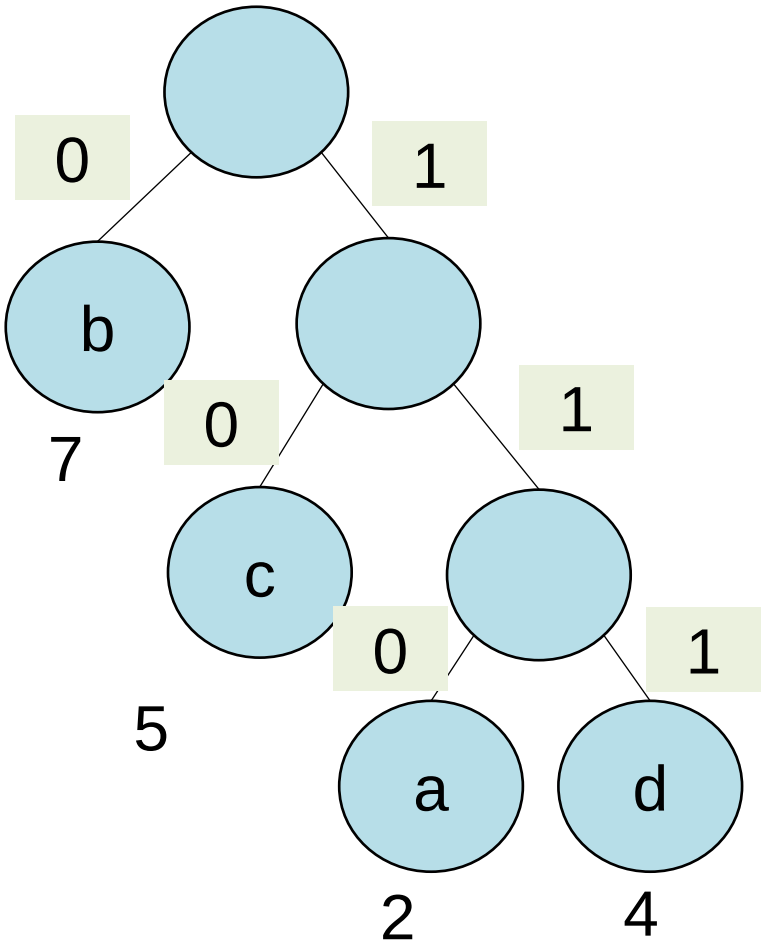
选根权值最小的两棵树5和6合并，新树的根节点为11

第四步



选根权值最小的两棵树7和11合并，新树的根节点为18

哈夫曼编码



从哈夫曼树根节点开始，对左子树分配码“0”，右子树分配码“1”，一直到达叶子节点为止，然后将从树根沿每条路径到达叶子结点的代码排列起来，便得到了哈夫曼编码。

编码后
a:110
b:0
c:10
d:111

阅读程序

2分钟

阅读程序

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int n,n1,n2
5      int ans;
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 }
```

1	识别变量
常见变量名 翻译循环变量 根据变量名的英文推断	

2	找出关键语句
控制结构(for, if) 常见算法的基本操作 函数参数、返回值	

3	理解代码段作用
翻译解释代码段	

阅读程序

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int n,n1,n2
5      int ans;
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 }
```

解释变量的作用

n1,n2	输入两个数字
n	输入两数之和
i	尝试的因子
flag	标记n是否是质数
ans	输出结果

阅读程序

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int n,n1,n2
5      int ans;
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 }
```

关键语句

定义各变量并初始化

输入两个整数n1,n2

计算两者之和n

如果在2到√n之间
存在因子i是n的约数
把标记flag修改0

输出n1*n2*flag

n是**质数**输出n1*n2
否则直接输出0

阅读程序

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int n,n1,n2
5      int ans;
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	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。（　）

阅读程序

```
#include <iostream>
using namespace std;
int main() {
    int n,n1,n2
    int ans;
    int flag=1;
    int i=2;
    cin>>n1>>n2;
    n=n1+n2;
    while(i*i<=n){
        if(n%i==0){
            flag=0;
        }
        i++;
    }
    ans=n1*n2*flag;
    cout<<ans<<endl;
    return 0;
}
```

判断	2. 输入到n1和n2的值，不允许同时超过1e9。(　　)

阅读程序

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int n,n1,n2
5      int ans;
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	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。(　　)

阅读程序

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int n,n1,n2
5      int ans;
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	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。（　）

阅读程序

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int n,n1,n2
5      int ans;
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 }
```

选择

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

阅读程序

```cpp
#include <iostream>
using namespace std;
int main() {
    int n,n1,n2
    int ans;
    int flag=1;
    int i=2;
    cin>>n1>>n2;
    n=n1+n2;
    while(i*i<=n){
        if(n%i==0){
            flag=0;
        }
        i++;
    }
    ans=n1*n2*flag;
    cout<<ans<<endl;
    return 0;
}
```

选择 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

阅读程序

3分钟

阅读程序

```
#include <bits/stdc++.h>
using namespace std;
string s;
long long magic(int l, int r){
    long long ans = 0;
    for (int i=l;i<=r;++i) ans=ans*4+s[i]-'a'+1;
    return ans;
}
int main(){
    cin >> s;
    int len = s.length();
    int ans = 0;
    for (int l1 = 0; l1 < len; ++l1)
        for (int r1 = l1; r1 < len; ++r1){
            bool bo = true;
            for (int l2 = 0; l2 < len; ++l2)
                for (int r2 = l2; r2 < len; ++r2)
                    if(magic(l1,r1)==magic(l2,r2)&&(l1!=l2||r1!=r2))
                        bo = false;
            if (bo) ans += 1;
        }
    cout << ans << endl;
    return 0;
}
```

1	识别变量
常见变量名 翻译循环变量 根据变量名的英文推断	

2	找出关键语句
控制结构(for, if) 常见算法的基本操作 函数参数、返回值	

3	理解代码段作用
翻译解释代码段	

阅读程序

```
#include <bits/stdc++.h>
using namespace std;
string s;
long long magic(int l, int r){
    long long ans = 0;
    for (int i=l;i<=r;++i) ans=ans*4+s[i]-'a'+1;
    return ans;
}
int main(){
    cin >> s;
    int len = s.length();
    int ans = 0;
    for (int l1 = 0; l1 < len; ++l1)
        for (int r1 = l1; r1 < len; ++r1){
            bool bo = true;
            for (int l2 = 0; l2 < len; ++l2)
                for (int r2 = l2; r2 < len; ++r2)
                    if(magic(l1,r1)==magic(l2,r2)&&(l1!=l2||r1!=r2))
                        bo = false;
            if (bo) ans += 1;
        }
    cout << ans << endl;
    return 0;
}
```

解释变量作用

s	输入字符串
len	字符串长度
l1, r1	所枚举子串的始末下标
l2, r2	所枚举另一子串始末下标
bo	对某子串s[l1,r1] 当找到另一子串s[l2,r2] 符合程序18行条件时 即将bool置为false
d	对s中每个子段s[l1,r1] 只要经过16-19行后bo被置为false，ans就统计上

magic函数是**字符串哈希**：将一个字符子串转化成一个整数，保证只要字符串不同，所得整数ans就不同

关键语句

阅读程序

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

将一段字串转成对应数字
可看成是“四进制”转“十进制”

思考：若要保证**不同串转成不同整数**，输入字母范围有什么限制？

分别枚举起始下标和结尾下标
暴力遍历s中每一个子串s[l1,r1]

对s[l1,r1]，逐个检查s[l2,r2]，只要有一个s[l2,r2]符合18行，即两子段magic值相等且两子段不同，bo就置为false：说明s[l1,r1]可在s的别处找到相同串

如果bo经过16-19行还是true
ans就统计上
因此最后输出的ans：统计s中不重复出现的、独一无二的子串个数

阅读程序

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

判断	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。(　　)

阅读程序

```
#include <bits/stdc++.h>
using namespace std;
string s;
long long magic(int l, int r){
    long long ans = 0;
    for (int i=l;i<=r;++i) ans=ans*4+s[i]-'a'+1;
    return ans;
}
int main(){
    cin >> s;
    int len = s.length();
    int ans = 0;
    for (int l1 = 0; l1 < len; ++l1)
        for (int r1 = l1; r1 < len; ++r1){
            bool bo = true;
            for (int l2 = 0; l2 < len; ++l2)
                for (int r2 = l2; r2 < len; ++r2)
                    if(magic(l1,r1)==magic(l2,r2)&&(l1!=l2||r1!=r2))
                        bo = false;
            if (bo) ans += 1;
        }
    cout << ans << endl;
    return 0;
}
```

判断	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$O(n^4)$。(　　)

阅读程序

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

判断	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。(　　)

阅读程序

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

判断

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。(　　)

阅读程序

```cpp
#include <bits/stdc++.h>
using namespace std;
string s;
long long magic(int l, int r){
    long long ans = 0;
    for (int i=l;i<=r;++i) ans=ans*4+s[i]-'a'+1;
    return ans;
}
int main(){
    cin >> s;
    int len = s.length();
    int ans = 0;
    for (int l1 = 0; l1 < len; ++l1)
        for (int r1 = l1; r1 < len; ++r1){
            bool bo = true;
            for (int l2 = 0; l2 < len; ++l2)
                for (int r2 = l2; r2 < len; ++r2)
                    if(magic(l1,r1)==magic(l2,r2)&&(l1!=l2||r1!=r2))
                        bo = false;
            if (bo) ans += 1;
        }
    cout << ans << endl;
    return 0;
}
```

选择 5. 输入abacaba，输出结果是（ ）

A. 7 B. 7! C. 3 D. 16

阅读程序

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

选择

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

阅读程序

3分钟

阅读程序

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

1	识别变量
常见变量名 翻译循环变量 根据变量名的英文推断	

2	找出关键语句
控制结构(for, if) 常见算法的基本操作 函数参数、返回值	

3	理解代码段作用
翻译解释代码段	

解释变量作用

阅读程序

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

a1, a[ ]	字串输入大数a1 存入数组
b1, b[ ]	字串输入大数b1 存入数组
c[ ]	所得结果同样保存数组
lena, lenb, lenc	大整数的位数
x	进位值：保证数组每个元素存的是逐位上的值

超大整数a,b通过数组做乘法运算

关键语句

阅读程序

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

定义各变量、数组，并将两个大整数输入到字符串a1,b1中

将数组a,b,c每个元素都初始化为0

将字符串a1,a2中逐位数字存到数组a和b中

请回答逐位存储的顺序是怎样的？

对a1从低到高的每一位i
我们去枚举a2从低到高的每一位j
r让这两位数字相乘

由乘法原理，第i位乘第j位结果在第i+j-1位。同时通过x保存进位值

算出乘积结果数组c的位数lenc
从**高位到低位**依次输出即得答案

阅读程序

```
int main() {
    string a1, b1;
    int a[109], b[109], c[109], lena, lenb, lenc;
    int i, j, x;
    cin >> a1 >> b1;
    memset(a, 0, sizeof(a));
    memset(b, 0, sizeof(b));
    memset(c, 0, sizeof(c));
    lena = a1.length(); lenb = b1.len
    for(i = 0; i <= lena - 1; i++)
        a[lena - i] = a1[i] - '0';
    for(i = 0; i <= lenb - 1; i++)
        b[lenb - i] = b1[i] - '0';
    for(i = 1; i <= lena; i++) {
        x = 0;
        for(j = 1; j <= lenb; j++) {
            c[i + j - 1] += a[i] * b[j] + x;
            x = c[i + j - 1] / 10;
            c[i + j - 1] = c[i + j - 1] % 10;
        }
        c[i + lenb] = x;
    }
    lenc = lena + lenb;
    while(c[lenc] == 0 && lenc > 1) lenc--;
    for(i = lenc; i >= 1; i--) cout << c[i];
    cout << endl;
    return 0;
```

判断

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。(　　)

阅读程序

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

判断

2. 可以把18行整型变量x初始化为0的操作，放在17行for循环的前面，对整个程序输出结果不会有影响。(　　)

阅读程序

```
int main() {
    string a1, b1;
    int a[109], b[109], c[109], lena, lenb, lenc;
    int i, j, x;
    cin >> a1 >> b1;
    memset(a, 0, sizeof(a));
    memset(b, 0, sizeof(b));
    memset(c, 0, sizeof(c));
    lena = a1.length(); lenb = b1.length();
    for(i = 0; i <= lena - 1; i++)
        a[lena - i] = a1[i] - '0';
    for(i = 0; i <= lenb - 1; i++)
        b[lenb - i] = b1[i] - '0';
    for(i = 1; i <= lena; i++) {
        x = 0;
        for(j = 1; j <= lenb; j++) {
            c[i + j - 1] += a[i] * b[j] + x;
            x = c[i + j - 1] / 10;
            c[i + j - 1] = c[i + j - 1] % 10;
        }
        c[i + lenb] = x;
    }
    lenc = lena + lenb;
    while(c[lenc] == 0 && lenc > 1) lenc--;
    for(i = lenc; i >= 1; i--) cout << c[i];
    cout << endl;
    return 0;
```

选择
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

阅读程序

```
int main() {
    string a1, b1;
    int a[109], b[109], c[109], lena, lenb, lenc;
    int i, j, x;
    cin >> a1 >> b1;
    memset(a, 0, sizeof(a));
    memset(b, 0, sizeof(b));
    memset(c, 0, sizeof(c));
    lena = a1.length(); lenb = b1.length();
    for(i = 0; i <= lena - 1; i++)
        a[lena - i] = a1[i] - '0';
    for(i = 0; i <= lenb - 1; i++)
        b[lenb - i] = b1[i] - '0';
    for(i = 1; i <= lena; i++) {
        x = 0;
        for(j = 1; j <= lenb; j++) {
            c[i + j - 1] += a[i] * b[j] + x;
            x = c[i + j - 1] / 10;
            c[i + j - 1] = c[i + j - 1] % 10;
        }
        c[i + lenb] = x;
    }
    lenc = lena + lenb;
    while(c[lenc] == 0 && lenc > 1) lenc--;
    for(i = lenc; i >= 1; i--) cout << c[i];
    cout << endl;
    return 0;
```

选择
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（ ）。
B. c[6]=1

阅读程序

```
int main() {
    string a1, b1;
    int a[109], b[109], c[109], lena, lenb, lenc;
    int i, j, x;
    cin >> a1 >> b1;
    memset(a, 0, sizeof(a));
    memset(b, 0, sizeof(b));
    memset(c, 0, sizeof(c));
    lena = a1.length(); lenb = b1.length();
    for(i = 0; i <= lena - 1; i++)
        a[lena - i] = a1[i] - '0';
    for(i = 0; i <= lenb - 1; i++)
        b[lenb - i] = b1[i] - '0';
    for(i = 1; i <= lena; i++) {
        x = 0;
        for(j = 1; j <= lenb; j++) {
            c[i + j - 1] += a[i] * b[j] + x;
            x = c[i + j - 1] / 10;
            c[i + j - 1] = c[i + j - 1] % 10;
        }
        c[i + lenb] = x;
    }
    lenc = lena + lenb;
    while(c[lenc] == 0 && lenc > 1) lenc--;
    for(i = lenc; i >= 1; i--) cout << c[i];
    cout << endl;
    return 0;
```

选择

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

阅读程序

```
int main() {
    string a1, b1;
    int a[109], b[109], c[109], lena, lenb, lenc;
    int i, j, x;
    cin >> a1 >> b1;
    memset(a, 0, sizeof(a));
    memset(b, 0, sizeof(b));
    memset(c, 0, sizeof(c));
    lena = a1.length(); lenb = b1.length();
    for(i = 0; i <= lena - 1; i++)
        a[lena - i] = a1[i] - '0';
    for(i = 0; i <= lenb - 1; i++)
        b[lenb - i] = b1[i] - '0';
    for(i = 1; i <= lena; i++) {
        x = 0;
        for(j = 1; j <= lenb; j++) {
            c[i + j - 1] += a[i] * b[j] + x;
            x = c[i + j - 1] / 10;
            c[i + j - 1] = c[i + j - 1] % 10;
        }
        c[i + lenb] = x;
    }
    lenc = lena + lenb;
    while(c[lenc] == 0 && lenc > 1) lenc--;
    for(i = lenc; i >= 1; i--) cout << c[i];
    cout << endl;
    return 0;
```

选择
6.（4分）输入202020192018202220212020，将27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

完善程序

完善程序

（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。

3分钟

完善程序

```
5   bool b[11] = {0};
6   int total = 0;
7   int a[11] = {0};
8   bool prime(int x, int y) {
9       int k = 2;
10      int i = ____(1)____;
11      while(k * k <= i && ____(2)____) k++;
12      if(k * k > i) return 1;
13      else return 0;
14  }
15  int out() {
16      total++; cout << " <" << total << "> ";
17      for(int j = 1; j <= 10; j++) cout << a[j] << " ";
18      cout << endl;
19  }
20  void search(int t) {
21      for(int i = 1; i <= 10; i++) {
22          if(!b[i] && prime(a[t - 1], i)) {
23              a[t] = i; b[i] = 1;
24              if(t == 10)
25                  if(prime(a[____(3)____], a[1])) out();
26              else ____(4)____;
27              ____(5)____;
28          }
29      }
30  }
31  int main() {
32      search(1); cout << total << endl;
```

1	识别变量
常见变量名 翻译循环变量 根据变量名的英文推断	

2	找出关键语句
控制结构(for, if) 常见算法的基本操作 函数参数、返回值	

3	理解代码段作用
翻译解释代码段	

解释变量作用

完善程序

```
5   bool b[11] = {0};
6   int total = 0;
7   int a[11] = {0};
8   bool prime(int x, int y) {
9       int k = 2;
10      int i = ____(1)____;
11      while(k * k <= i && ____(2)____) k++;
12      if(k * k > i) return 1;
13      else return 0;
14  }
15  int out() {
16      total++; cout << " <" << total << "> ";
17      for(int j = 1; j <= 10; j++) cout << a[j] << " ";
18      cout << endl;
19  }
20  void search(int t) {
21      for(int i = 1; i <= 10; i++) {
22          if(!b[i] && prime(a[t - 1], i)) {
23              a[t] = i; b[i] = 1;
24              if(t == 10)
25                  if(prime(a[____(3)____], a[1])) out();
26              else ____(4)____;
27              ____(5)____;
28          }
29      }
30  }
31  int main() {
32      search(1); cout << total << endl;
```

a[t]	第t序位上所安排数字
b[i]	标记数字i是否被选中
k	因子
i	待检验数，按题意是x+y
total	计数，用于输出格式
i	依次枚举每个数字i

程序使用递归进行搜索与回溯

完善程序

代码段作用

初始化

prime函数检验x+y是否是质数

out函数输出某一种合法排列a

search函数逐层递归搜索排列

```
5   bool b[11] = {0};
6   int total = 0;
7   int a[11] = {0};
8   bool prime(int x, int y) {
9       int k = 2;
10      int i = ____(1)____;
11      while(k * k <= i && ____(2)____) k++;
12      if(k * k > i) return 1;
13      else return 0;
14  }
15  int out() {
16      total++; cout << " <" << total << "> ";
17      for(int j = 1; j <= 10; j++) cout << a[j] << " ";
18      cout << endl;
19  }
20  void search(int t) {
21      for(int i = 1; i <= 10; i++) {
22          if(!b[i] && prime(a[t - 1], i)) {
23              a[t] = i; b[i] = 1;
24              if(t == 10)
25                  if(prime(a[____(3)____], a[1])) out();
26              else ____(4)____;
27              ____(5)____;
28          }
29      }
30  }
31  int main() {
32      search(1); cout << total << endl;
```

完善程序

代码段作用

```
5   bool b[11] = {0};
6   int total = 0;
7   int a[11] = {0};
8   bool prime(int x, int y) {
9       int k = 2;
10      int i = ____(1)____;
11      while(k * k <= i && ____(2)____) k++;
12      if(k * k > i) return 1;
13      else return 0;
14  }
15  int out() {
16      total++; cout << " <" << total << "> ";
17      for(int j = 1; j <= 10; j++) cout << a[j] << " ";
18      cout << endl;
19  }
20  void search(int t) {
21      for(int i = 1; i <= 10; i++) {
22          if(!b[i] && prime(a[t - 1], i)) {
23              a[t] = i; b[i] = 1;
24              if(t == 10)
25                  if(prime(a[____(3)____], a[1])) out();
26              else ____(4)____;
27              ____(5)____;
28          }
29      }
30  }
31  int main() {
32      search(1); cout << total << endl;
```

初始化

prime函数检验x+y是否是质数

检查在范围内是否能找到
可以整除x+y的因子k

out函数输出某一种合法排列a

search(t)=第t层递归=第t序位选择

依次枚举每个数字i，只要i没被选且它与前者之和是**质数，则选**

若t==10 即座次已经选满，且这时a[10]+a[1]也是质数，则输出

否则选好后递归继续下一个位序

第27行：回溯

完善程序

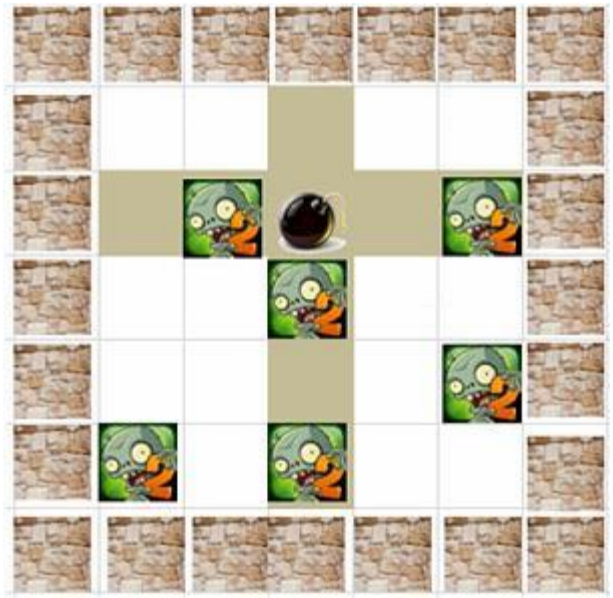
```
5   bool b[11] = {0};
6   int total = 0;
7   int a[11] = {0};
8   bool prime(int x, int y) {
9       int k = 2;
10      int i = x+y ;
11      while( k*k<=i&&i%k!=0 ) k++;
12      if(k * k > i) return 1;
13      else return 0;
14  }
15  int out() {
16      total++; cout << " <" << total << "> ";
17      for(int j = 1; j <= 10; j++) cout << a[j] << " ";
18      cout << endl;
19  }
20  void search(int t) {
21      for(int i = 1; i <= 10; i++) {
22          if(!b[i] && prime(a[t - 1], i)) {
23              a[t] = i; b[i] = 1;
24              if(t == 10)
25                  if(prime(a[ 10 ], a[1])) out();
26              else search(t+1)
27              b[i]=0
28          }
29      }
30  }
31  int main() {
32      search(1); cout << total << endl;
```

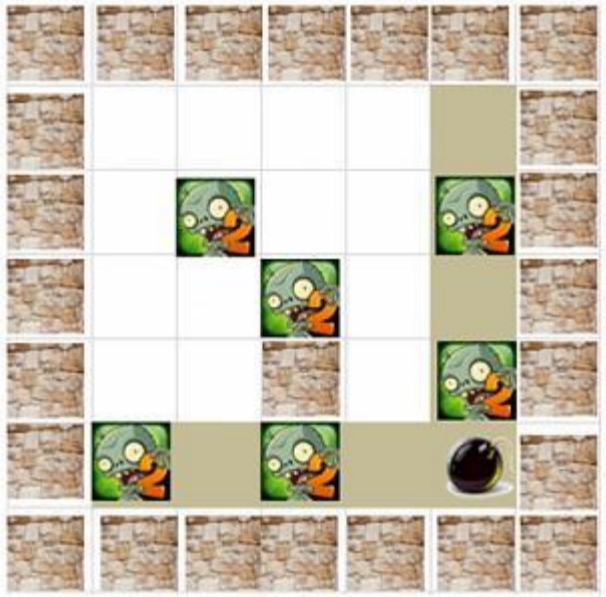
完善程序

完善程序

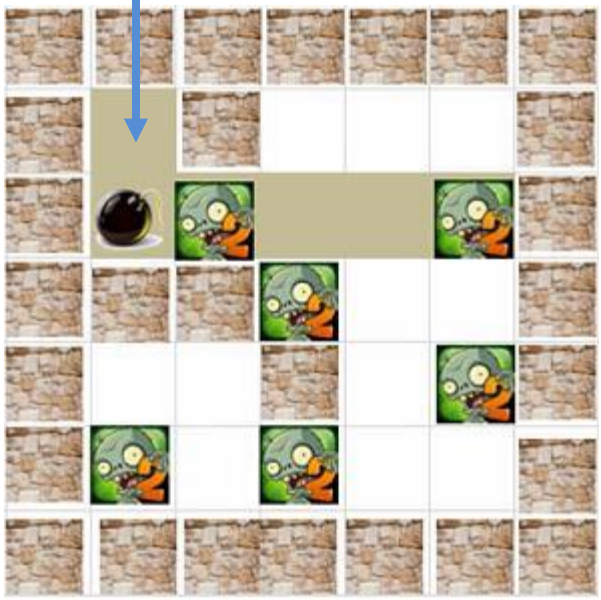
（大战僵尸）
有一个 N 行 M 列单元格构成的地图（四周都是墙体），小新从起始位置（X,Y）开始，可以往四方向不断行走。不能穿墙#，也不能越过僵尸G，只能在空地行走。
现在小新要放置一个炸弹。该种炸弹在不穿墙前提下，可以炸掉同行同列的所有僵尸。
程序输出小新放置炸弹**最多**能炸掉的僵尸数量。

手算样例



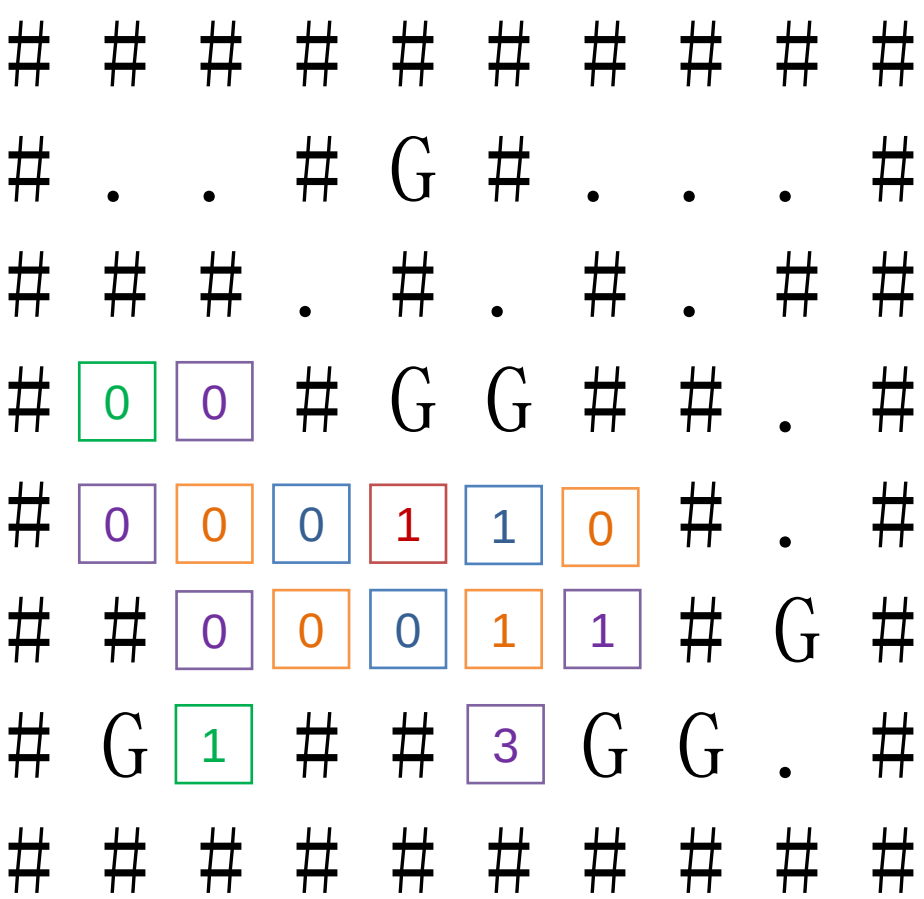


假设起始点是(2,2)
最佳坐标则是(3,2)
此时能炸掉2个僵尸



炸弹不能穿墙
炸掉同行同列所有僵尸

步骤演示



假设从(5,5)开始往四周探索

(5,5)处只能炸掉头上的一个G

(5,5)直接邻居有(4,5) (6,5) 它们无法炸掉G，(5,6)炸掉1个G

接着再向外移动一步
依次寻找(4,5) (6,5) (5,6)的
未被访问过的邻居点
并标记它们各自能炸掉多少G

继续扩散这个BFS找寻过程
直到所有可达点均被访问到

最后可得在(7,6)可炸得最多的3个僵尸

3分钟

完善程序

```
4   struct node{int x,y;};
5   char a[SIZE][SIZE];
6   bool vst[SIZE][SIZE];
7   int ans=0;
8   int dx[4]={-1,1,0,0}, dy[4]={0,0,-1,1};
9   node _next; queue<node> q;
10  int  up(int x,int y){
17  int  down(int x,int y){
24  int  left(int x,int y){
31  int  right(int x,int y){
38  int count(int x,int y){ return ____(2)____; }
39  void bfs(){
40      while(!q.empty()){
41          node now=q.front(); q.pop();
42          ans=max(ans, count(now.x, now.y));
43          for(____(3)____){
44              int r=now.x+dx[i], c=now.y+dy[i];
45              if(____(4)____){
46                  _next.x=r; _next.y=c; vst[r][c]=true;
47                  q.push(_next);
48              }
49          }
50      }
51  }
52  int main(){
53      int n,m,x,y; cin>>n>>m>>x>>y;
54      for(int i=1;i<=n;i++)
55          for(int j=1;j<=m;j++) cin>>a[i][j];
56      memset(vst,0,sizeof(vst));
57      _next.x=x; _next.y=y; vst[x][y]=true;
58      q.push(_next); bfs();
59      cout<<____(5)____<<endl;
```

1	识别变量
常见变量名 翻译循环变量 根据变量名的英文推断	

2	找出关键语句
控制结构(for, if) 常见算法的基本操作 函数参数、返回值	

3	理解代码段作用
翻译解释代码段	

```
4   struct node{int x,y;};
5   char a[SIZE][SIZE];
6   bool vst[SIZE][SIZE];
7   int ans=0;
8   int dx[4]={-1,1,0,0}, dy[4]={0,0,-1,1};
9   node _next; queue<node> q;
10  int  up(int x,int y){
17  int  down(int x,int y){
24  int  left(int x,int y){
31  int  right(int x,int y){
38  int count(int x,int y){ return ____(2)____; }
39  void bfs(){
40      while(!q.empty()){
41          node now=q.front(); q.pop();
42          ans=max(ans, count(now.x, now.y));
43          for(____(3)____){
44              int r=now.x+dx[i], c=now.y+dy[i];
45              if(____(4)____){
46                  _next.x=r; _next.y=c; vst[r][c]=true;
47                  q.push(_next);
48              }
49          }
50      }
51  }
52  int main(){
53      int n,m,x,y; cin>>n>>m>>x>>y;
54      for(int i=1;i<=n;i++)
55          for(int j=1;j<=m;j++) cin>>a[i][j];
56      memset(vst,0,sizeof(vst));
57      _next.x=x; _next.y=y; vst[x][y]=true;
58      q.push(_next); bfs();
59      cout<<____(5)____<<endl;
```

解释变量作用

node	结构体，维护格点行列
a	二维字符数组表示地图
vst[i][j]	标记(i,j)点是否已被访问
ans	维护最多能炸到僵尸数
dx, dy	四方向移动的方向数组
queue<int> q	BFS搜索队列
n, m	整个棋盘格的行列数
x, y	小新起始点的行列号

```
4   struct node{int x,y;};
5   char a[SIZE][SIZE];
6   bool vst[SIZE][SIZE];
7   int ans=0;
8   int dx[4]={-1,1,0,0}, dy[4]={0,0,-1,1};
9   node _next; queue<node> q;
10  int _up(int x,int y){
11      if(a[x][y]=='#') return 0;
12      if(a[x][y]=='G') return ____(1)____;
13      return _up(x-1,y);
14  }
15  int _down(int x,int y){
16      if(a[x][y]=='#') return 0;
17      if(a[x][y]=='G') return 1+_down(x+1,y);
18      return _down(x+1,y);
19  }
20  int _left(int x,int y){
21      if(a[x][y]=='#') return 0;
22      if(a[x][y]=='G') return 1+_left(x,y-1);
23      return _left(x,y-1);
24  }
25  int _right(int x,int y){
26      if(a[x][y]=='#') return 0;
27      if(a[x][y]=='G') return 1+_right(x,y+1);
28      return _right(x,y+1);
29  }
30  int count(int x,int y){ return ___(2)___; }
```

定义并初始化

该点以及该点往上**递归**搜索
总共能炸到几个僵尸

每次搜索起始点一定是“空地”
故返回(x,y)上方能炸几个僵尸

若是墙，后续无法炸到，返回0
若是僵尸，这个净算1同时，继续递归调用_up**往上方**进行搜索
若是空地，直接继续往上搜索
即直接返回_up(x-1,y)的结果

分别对应剩余三个方向上
按规则能够炸到的僵尸数

count函数帮忙做统计
返回(x,y)四方向**共炸僵尸数**

关键语句

```
15 int _down(int x,int y){
20 int _left(int x,int y){
25 int _right(int x,int y){
30 int count(int x,int y){ return ___(2)___; }
31 void bfs(){
32     while(!q.empty()){
33         node now=q.front();
34         ans=max(ans, count(now.x, now.y));
35         for(____(3)____){
36             int r=now.x+dx[i], c=now.y+dy[i];
37             if(____(4)____){
38                 _next.x=r; _next.y=c; vst[r][c]=true;
39                 q.push(_next);
40             }
41         }
42         q.pop();
43     }
44 }
45 int main(){
46     int n,m,x,y; cin>>n>>m>>x>>y;
47     for(int i=1;i<=n;i++)
48         for(int j=1;j<=m;j++) cin>>a[i][j];
49     memset(vst,0,sizeof(vst));
50     _next.x=x; _next.y=y; vst[x][y]=true;
51     q.push(_next); bfs();
52     cout<<____(5)____<<endl;
53     return 0;
```

只要队列非空，取出队首now
count函数算now点能炸僵尸数，
并与维护最大值的ans打擂台

枚举(x,y)四方向上邻居点(r,c)
只要该点是“空地”且未访问
标记该点并且加入队列q

输入地图行列数和起始点

输入整张棋盘格字符到a

起始点入队，开始BFS搜索

关键语句

```
int _up(int x,int y){
    if(a[x][y]=='#') return 0;
    if(a[x][y]=='G') return
    return _up(x-1,y);
}
int count(int x,int y){ return ___(2)___; }
void bfs(){
    while(!q.empty()){
        node now=q.front();
        ans=max(ans, count(now.x, now.y));
        for(____(3)____){
            int r=now.x+dx[i], c=now.y+dy[i];
            if(____(4)____){
                _next.x=r; _next.y=c; vst[r][c]=true;
                q.push(_next);
            }
        }
        q.pop();
    }
}
int main(){
    int n,m,x,y; cin>>n>>m>>x>>y;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++) cin>>a[i][j];
    memset(vst,0,sizeof(vst));
    _next.x=x; _next.y=y; vst[x][y]=true;
    q.push(_next); bfs();
    cout<<____(5)____<<endl;
```

1+_up(x-1,y)

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

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

!vst[r][c]&&a[r][c]=='.'

ans