2022初赛讲解

信智学院　陈琰宏

1 单项选择题

1. 以下哪种功能没有涉及C++语言的面向对象特性支持：（ A ）。

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++中构造一个class或struct

D. C++中构造来源于同一基类的多个派生类

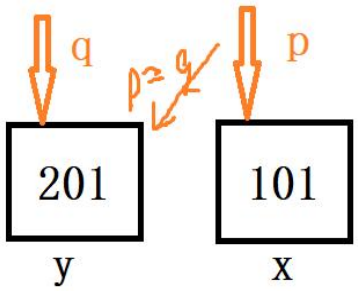
2. 有6个元素，按照6、5、4、3、2、1的顺序进入栈S，请问下列哪个出栈序列是非法的 （ C ）。

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

3. 运行以下代码片段的行为是（ D ）。

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



A. 将x的值赋为201　　B. 将y的值赋为101

B. 将q指向x的地址　　D. 将p指向y的地址

4. 链表和数组的区别包括（ C ）。

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

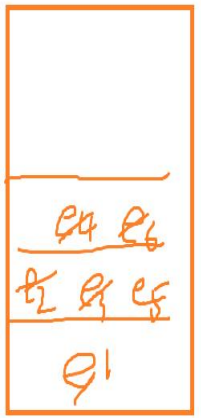
C. 数组大小固定，链表大小可动态调整

D. 以上均正确

1 单项选择题

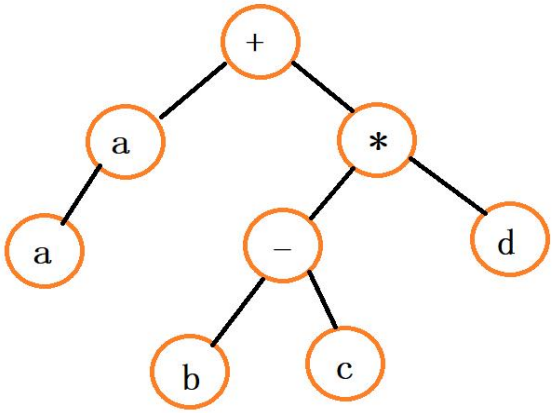
5. 对假设栈S和队列Q的初始状态为空。存在e1~e6六个互不相同的数据，每个数据按照进栈S、出栈S、进队列Q、出队列Q的顺序操作，不同数据间的操作可能会交错。已知栈S中依次有数据e1、e2、e3、e4、e5和e6进栈，队列Q依次有数据e2、e4、e3、e6、e5和e1出队列。则栈S的容量至少是（ B ）个数据。

A. 2　　B. 3　　C. 4　　D. 6



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

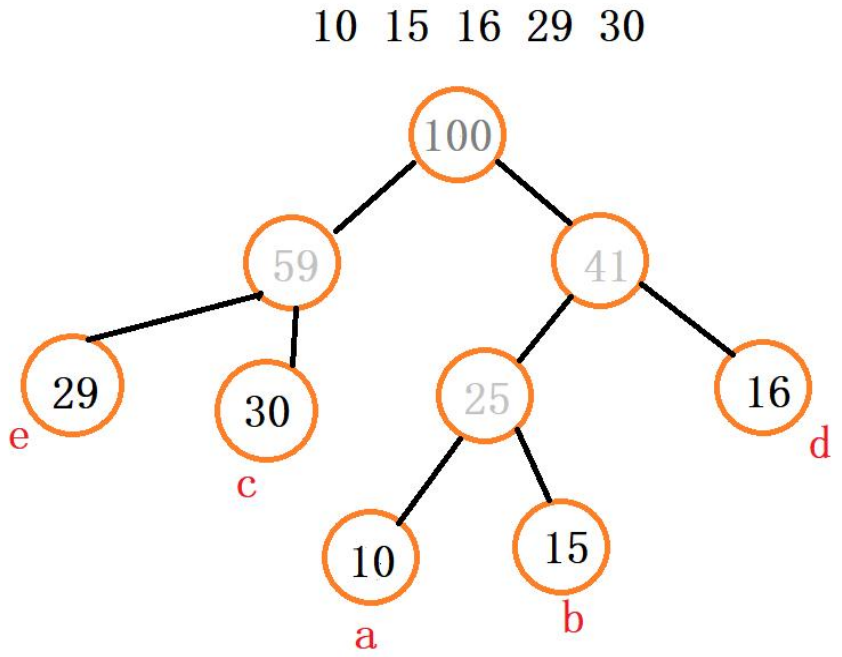
A. *+a-bcd　　B. +a*-bcd　　C. abc-d*+　　D . abc-+d



1 单项选择题

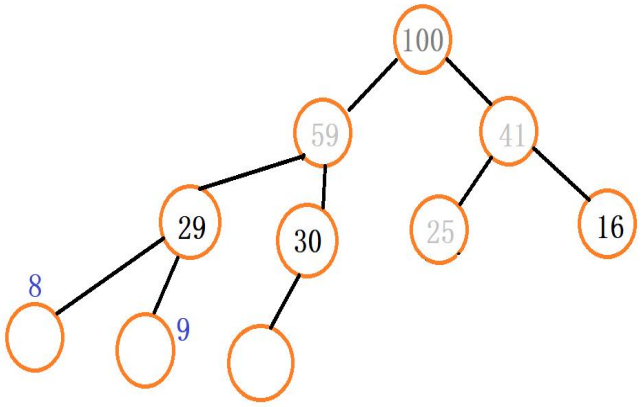
7. 假设字母表 {a, b, c, d, e} 在字符串出现的频率分别为 10%, 15%, 30%, 16%, 29%。若使用哈夫曼编码方式对字母进行不定长的二进制编码，字母d的编码长度为（ B ）位。

A. 1　　B. 2　　C. 2或3　　D. 3



1 单项选择题

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（ C ）。



A. 8、18　　B. 10、18

C. 8、19　　D. 10、19

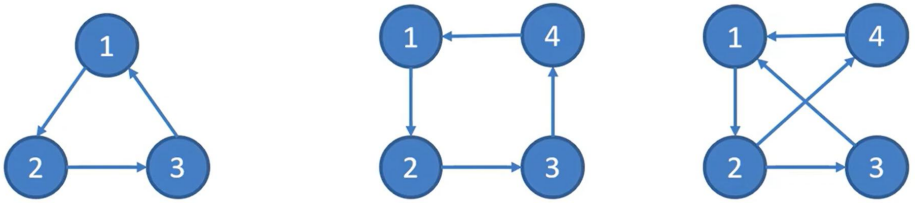
9. 考虑由N个顶点构成的有向连通图，采用邻接矩阵的数据结构表示时，该矩阵中至少存在（ B ）个非零元素。

A. N-1　　B. N　　C. N+1　　D. N^2

N个顶点的连通图，至少存在N-1条边，即存在N-1非零权值。

图中任意两点是连通的，称作连通图。

每个节点最少一个入边，一个出边，形成环。



1 单项选择题

10. 以下对数据结构的表述不恰当的一项为：（ D ）。

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。 线性表

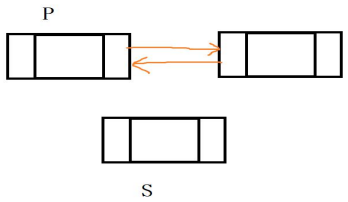
11. 以下哪组操作能完成在双向循环链表结点p之后插入结点s的效果（其中，next域为结点的直接后继，prev域为结点的直接前驱）：（ D ）。

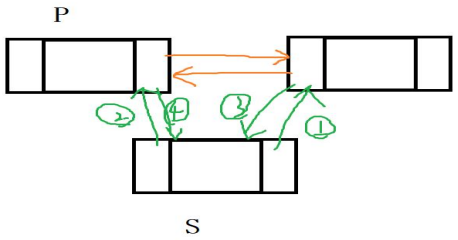
A. p->next->prev=s; s->prev=p; p->next=s; s->next=p->next;

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;





1)S->next=p->next;
2)s->pre->p
3)p->next->pre=s
4) p->next=s

1 单项选择题

12. 以下排序算法的常见实现中，哪个选项的说法是错误的：（B ）。

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

13. 八进制数32.1对应的十进制数是（ C）。

A. 24.125　　B. 24.250　　C. 26.125　　D. 26.250

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（ B ）个内容互不相同的子串。

A. 12　　B. 13　　C. 14　　D. 15

a b c ab bc ca abc bca cab abca bcab abcab 空

15. 以下对递归方法的描述中，正确的是：（ B ）

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

2 阅读程序写结果1[6330]

```
 5  int main()
 6  {
 7      unsigned short x, y;
 8      cin >> x >> y;
 9      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  }
```

本程序考查位运算的知识点。unsigned short表示无符号短整型，数据范围为 0 ~ 65535，占 2个字节。| 为按位或运算，& 为按位与运算，<< 为左移运算。需要注意的是 << 运算优先级高于 |。0x33 表示 十六进制的33，即十进制的51; 0x55 表示 十六机制的55，即十进制的85。

	输入的 x、y 均是不超过 15 的自然数								
	x	0	0	0	0	a	b	c	d
	x<<2	0	0	a	b	c	d	0	0
按位或：|	结果:	0	0	a	b	a|c	b|d	c	d
	0x33	0	0	1	1	0	0	1	1
按位与：&	结果	0	0	a	b	0	0	c	d
	将结果赋值给x								
	x	0	0	a	b	0	0	c	d
	x<<1	0	a	b	0	0	c	d	0
按位或：|	结果:	0	a	a|b	b	0	c	c|d	d
	0x55	0	1	0	1	0	1	0	1
按位与：&	结果	0	a	0	b	0	c	0	d
	将结果赋值给x								
	y	0	0	0	0	e	f	g	h
	程序11、12行运行完后								
	y	0	e	0	f	0	g	0	h

可用4位2进制表示，x=abcd，y是efgh

	y	0	e	0	f	0	g	0	h
	y<<1	e	0	f	0	g	0	h	0
	x	0	a	0	b	0	c	0	d
按位或：|	结果	e	a	f	b	g	c	h	d
	将结果赋值给z								
	z	e	a	f	b	g	c	h	d

```
x = (x | x << 2) & 0x33;
x = (x | x << 1) & 0x55;
y = (y | y << 2) & 0x33;
y = (y | y << 1) & 0x55;
unsigned short z = x | y << 1;
```

16. 删去第7行与第13行的unsigned，程序行为不变。（T ）

short为16位，删除unsigned，相当于少了一位最高位。0x55 = 01010101B，少一位不影响运算结果。

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

输入为不超过15的自然数，改为char以后，当输入为两位数时，x，y分别读入的是第一个数的十位和个位，改变了程序的行为和结果。

18. 程序总是输出一个整数“0”。（ F ）

代入"2 2"，输出结果为"12"。

19. 当输入为“2 2”时，输出为“10”。（F ）

计算得知，结果是12。转化成二进制: x=0010,a=b=d=0,c=1,y=0010, e=f=h=0, g=1 运行结果z=eafbgchd，带入为二进制00001100，是十进制的12。

20. 当输入为“2 2”时，输出为“59”。（F ）

21. 当输入为“13 8”时，输出为（ B ）。

A. “0”　B. “209”　C. “197”　D. “226”

x=13					a	b	c	d
x	0	0	0	0	1	1	0	1
y=8					e	f	g	h
y	0	0	0	0	1	0	0	0
z	e	a	f	b	g	c	h	d
	1	1	0	1	0	0	0	1
十进制	1+16+64+128=209							

2 阅读程序写结果2[6605]

```
 6
 7  const int MAXN = 105;
 8  const int MAXK = 105;
 9
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  }
```

```cpp
int main()
{
    int n, m;
    cin >> n >> m;
    cout << f(n, m) << endl << g(n, m) << endl;
    return 0;
}
```

```cpp
int g(int n, int m)
{
    for (int i = 1; i <= n; i++)
    h[i][1] = i;
    for (int j = 1; j <= m; j++)
    h[0][j] = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 2; j <= m; j++) {
            h[i][j] = numeric_limits<int>::max();
            for (int k = 1; k <= i; k++)
            h[i][j] = min(
                h[i][j],
                max(h[i - k][j], h[k - 1][j - 1]) + 1);
        }
    }
    return h[n][m];
}
```

递归和递推互为逆运算。本题f(n,m)是递归，g(n,m)是递推。

两个程序功能相同。

```
#include <algorithm>
#include <iostream>
#include <limits>  //c语言库<limits.h>
using namespace std;
const int MAXN = 105;
const int MAXK = 105;
int h[MAXN][MAXK];
int f(int n, int m) {    //递归
  if (m == 1) return n;   //递归返回点
  if (n == 0) return 0;    //递归返回点
  int ret = numeric_limits<int>::max();
   //numeric_limits<int>::max () 返回编译器允许的 int 型数最大值。2^31-1
  for (int i = 1; i <= n; i++)
     ret = min(ret, max(f(n - i, m), f(i - 1, m - 1)) + 1);
  return ret;
}
```

n: 行	f[ ][ ]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	m:列
	[0]	0	0	0	0	0	0	0	
	[1]	1	1	1	1	1	1	1	
	[2]	2	2	2	2	2	2	2	
	[3]	3	2	2	2	2	2	2	
	[4]	4	3	3	3	3	3	3	
	[5]	5	3	3	3	3	3	3	

if (m==1) return n;

n: 行	f[ ][ ]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	m:列
	[0]	0							
	[1]	1							
	[2]	2							
	[3]	3							
	[4]	4							
	[5]	5							

if (n==0) return 0;

n: 行	[1]	[2]	[3]	[4]	[5]	[6]	[7]	m:列
[0]	0	0	0	0	0	0	0	
[1]	1							
[2]	2							
[3]	3							
[4]	4							
[5]	5							

	f[ ][ ]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	m:列
n：行	[0]	0	0	0	0	0	0	0	
	[1]	1	1	1	1	1	1	1	
	[2]	2							
	[3]	3							
	[4]	4							
	[5]	5							

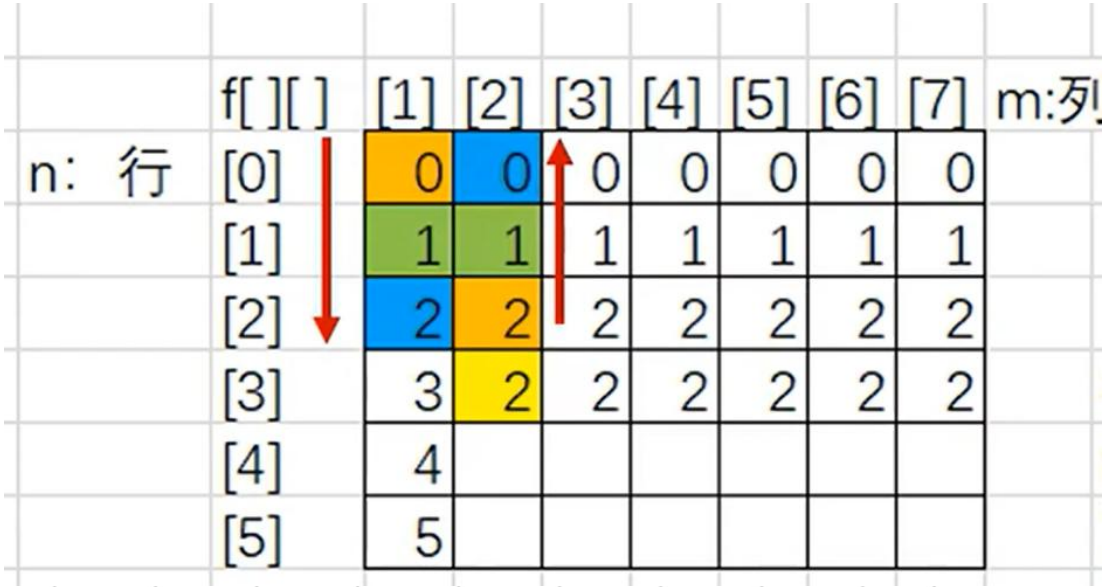
```
for (int i = 1; i <= n; i++)
    ret = min(ret, max(f(n - i, m), f(i - 1, m - 1)) + 1);
    // ret=min(ret,max()+1)
```

n=1, m=2, for循环循环1次， i=1

ret=min(ret， max(f[0][2],f[0][1])+1)=1

n=1, m=3, for循环循环1次， i=1

ret=min(ret， max(f[0][3],f[0][2])+1)=1



f[ ][ ]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	m:列
n：行 [0]	0	0	0	0	0	0	0	
[1]	1	1	1	1	1	1	1	
[2]	2	2	2	2	2	2	2	
[3]	3	2	2	2	2	2	2	
[4]	4							
[5]	5							

```
for (int i = 1; i <= n; i++)
        ret = min(ret, max(f(n - i, m), f(i - 1, m - 1)) + 1);
```

n=3, m=2, for循环循环3次

橙 i=1 ret=min(ret, max(f[2][2],f[0][1])+1)=3

绿 i=2 ret=min(ret, max(f[1][2],f[1][1])+1)=2

蓝 i=3 ret=min(ret, max(f[0][2],f[2][1])+1)=2

	h[ ][ ]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	j:列
i: 行	[0]	0	0	0	0	0	0	0	
	[1]	1	1	1	1	1	1	1	
	[2]	2							
	[3]	3							
	[4]	4							
	[5]	5							

```
int g(int n, int m) {
  for (int i = 1; i <= n; i++)
     h[i][1] = i;  //黄格
  for (int j = 1; j <= m; j++)
     h[0][j] = 0;  //橙格
  for (int i = 1; i <= n; i++) {   //第1行
     for (int j = 2; j <= m; j++) {  //第2列到第m列
        h[i][j] = numeric_limits<int>::max();  //初始化极大值
        for (int k = 1; k <= i; k++)  //循环1遍， k=1
           h[i][j] = min( h[i][j],max(h[i - k][j], h[k - 1][j - 1]) + 1);
     } //h[1][j]=min(h[1][j],max(h[0][j],h[0][j-1])+1)
  } // h[1][2]=min(h[1][2],max(h[0][2],h[0][1])+1)
```

i: 行	h[ ][ ]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	j:列
	[0]	0	0	0	0	0	0	0	
	[1]	1	1	1	1	1	1	1	
	[2]	2	2	2	2	2	2	2	
	[3]	3							
	[4]	4							
	[5]	5							

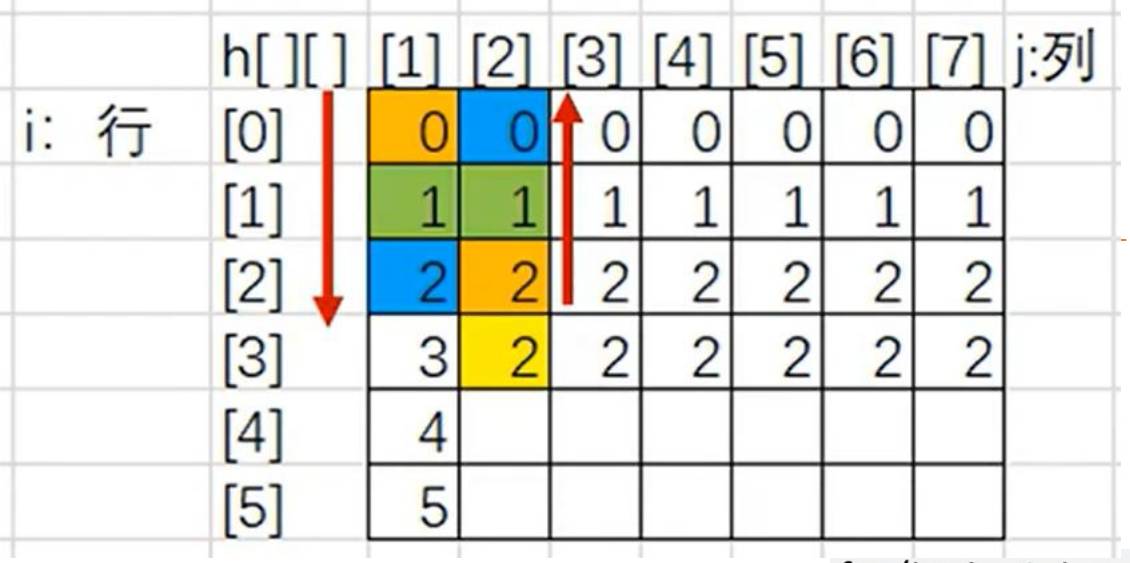
```
for (int i = 1; i <= n; i++) {  //第2行
   for (int j = 2; j <= m; j++) {  //第2列到第m列
      h[i][j] = numeric_limits<int>::max();  //初始化极大值
       for (int k = 1; k <= i; k++)
          h[i][j] = min( h[i][j],max(h[i - k][j], h[k - 1][j - 1]) + 1);
   }  //max( h[][当前列]，h[][当前列-1列])
}
```

i=2, j=2,黄格

k循环执行两次

k=1：h[2][2] = min(h[2][2],max(h[1][2], h[0][1]) + 1);
h[2][2]=2 //橙格

k=2：h[2][2] = min(h[2][2],max(h[0][2], h[1][1]) + 1);
h[2][2]=2 //绿格




```
for (int i = 1; i <= n; i++) {  //第3行
   for (int j = 2; j <= m; j++) {  //第2列到第m列
      h[i][j] = numeric_limits<int>::max();  //初始化极大值
       for (int k = 1; k <= i; k++)
          h[i][j] = min( h[i][j],max(h[i - k][j], h[k - 1][j - 1]) + 1);
   }
}
```

i=3, j=2, 黄格

k循环执行3次

k=1： h[3][2] = min(h[3][2],max(h[2][2], h[0][1]) + 1);
h[3][2]=3 //橙格

k=2： h[3][2] = min(h[3][2],max(h[1][2], h[1][1]) + 1);
h[3][2]=2 //绿格

k=3： h[3][2] = min(h[3][2],max(h[0][2], h[2][1]) + 1);
h[3][2]=2 //蓝格

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。（ F ） 448次

	g[ ][ ]	[1]	[2]	[3]	m:列
n:行	[0]	0	0	0	
	[1]	0	1	1	
	[2]	0			
	[3]	0			
	[4]	0			
	[5]	0			
	[6]	0			
	[7]	0			

```
for (int i = 1; i <= n; i++)
    ret = min(ret, max(f(n - i, m), f(i - 1, m - 1)) + 1);
```

设**g[i][j]**是计算**i**行**j**列的**f[i][j]min**函数的执行次数。

n=1, m=2, 黄格

i循环执行1次

i=1：ret=**min**(ret，max(**f[0][2]**,**f[0][1]**)+1)

g[1][2]=1+g[0][2]+g[0][1]=1

同理：g[1][3]=1。

	g[ ][ ]	[1]	[2]	[3]	m:列
n:行	[0]	0	0	0	
	[1]	0	1	1	
	[2]	0	3	4	
	[3]	0			
	[4]	0			
	[5]	0			
	[6]	0			
	[7]	0			

n=2, m=2, 黄格

i循环执行2次

i=1：ret=**min**(ret，max(**f[1][2]**,**f[0][1]**)+1)

i=2：ret=**min**(ret，max(**f[0][2]**,**f[1][1]**)+1)

g[2][3]=2+g[1][2]+g[0][2]+g[0][2]+g[1][2]=3

同理：g[2][4]=4

n:行	g[ ][ ]	[1]	[2]	[3]	m:列
	[0]	0	0	0	
	[1]	0	1	1	
	[2]	0	3	4	
	[3]	0	7	12	
	[4]	0			
	[5]	0			
	[6]	0			
	[7]	0			

n:行	g[ ][ ]	[1]	[2]	[3]	m:列
	[0]	0	0	0	
	[1]	0	1	1	
	[2]	0	3	4	
	[3]	0	7	12	
	[4]	0	15	32	
	[5]	0			
	[6]	0			
	[7]	0			

```
for (int i = 1; i <= n; i++)
    ret = min(ret, max(f(n - i, m), f(i - 1, m - 1)) + 1);
```

设g[i][j]是计算i行j列的f[i][j]min函数的执行次数。

n=3, m=2, 黄格

i循环执行3次

i=1：ret=**min**(ret，max(**f[2][2]**,**f[0][1]**)+1)

i=2：ret=**min**(ret，max(**f[1][2]**,**f[1][1]**)+1)

i=3：ret=**min**(ret，max(**f[0][2]**,**f[2][1]**)+1)

g[3][2]=3+上面6个橙色g[ ][ ]=3+4

n=4, m=2, 黄格

i循环执行4次

g[4][2]=4+上面8个橙色g[ ][ ]=15

发现n行第2列的$g[i][2]=2^i-1$

	g[ ][ ]	[1]	[2]	[3]	m:列
n:行	[0]	0	0	0	
	[1]	0	1	1	
	[2]	0	3	4	
	[3]	0	7	12	g[3][3]=3+上面6个g[][]=3+9=12
	[4]	0	15	32	g[4][3]=4+上面8个g[][]=4+28=32
	[5]	0	31	80	g[5][3]=5+上面10个g[][]=5+75=80
	[6]	0	63	192	g[6][3]=6+上面12个g[][]=6+186=192
	[7]	0	127	448	g[7][3]=7+上面14个g[][]=7+441=448

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。（F ）
23. 输出的两行整数总是相同的。（ T ）
24. 当m为1时，输出的第一行总为n。（ T ）

单选题
25. 算法g(n,m)最为准确的时间复杂度分析结果为（C ）。

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

26. 当输入为“20 2”时，输出的第一行为（ C ）。
“4” B. “5” C. “6” D. “20”

27.（4分）当输入为“100 100”时，输出的第一行为（ B ）。
“6” B. “7” C. “8” D. “9”

23. 输出的两行整数总是相同的。（T） 两个函数功能相同
24. 当m为1时，输出的第一行总为n。（T）

单选题
25. 算法g(n,m)最为准确的时间复杂度分析结果为（ ）。
n3/2　　B.　　C. 2　　D. 2)

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

26. 当输入为“20 2”时，输出的第一行为（ ）。
“4”　　B. “5”　　C. “6”　　D. “20”

27.（4分）当输入为“100 100”时，输出的第一行为（B）。
“6”　　B. “7”　　C. “8”　　D. “9”

f[ ][ ]	[1]	[2]
[0]	0	0
[1]	1	1
[2]	2	2
[3]	3	2
[4]	4	3
[5]	5	3
[6]	6	3

橙色max+1=3

绿色max+1=4

蓝色max+1=4

min=3

f[ ][ ]	[1]	[2]
[0]	0	0
[1]	1	1
[2]	2	2
[3]	3	2
[4]	4	3
[5]	5	3
[6]	6	3
[7]	7	4

橙色max+1=4

绿色max+1=4

蓝色max+1=4

min=4

f[ ][ ]	[1]	[2]
[0]	0	0
[1]	1	1
[2]	2	2
[3]	3	2
[4]	4	3
[5]	5	3
[6]	6	3
[7]	7	4
[8]	8	4

橙色max+1=4

绿色max+1=5

蓝色max+1=4

min=4

扔鸡蛋问题

有一幢楼房高 n层。某人准备了 m 个鸡蛋供试验。他想知道鸡蛋从几层扔下不会摔碎，并确定出最高安全楼层。试验过程中，鸡蛋没有摔碎则可以继续使用，摔碎了则需要换一个鸡蛋继续试验。为保证试验成功，要设计一个程序，以最小化最坏情况的试验次数 f(n,m)。 作为一个数学抽象，本问题采用一些理想化假设：所有鸡蛋抗摔能力相同，不计重复坠地的累积损伤，且忽略试验结果的偶然性。试验成功的标准是在 m个鸡蛋用完之前，精确确定最高安全楼层是哪一层（答案是x，则[1,x]层不碎，[x+1,n]会碎）。允许有鸡蛋剩余。

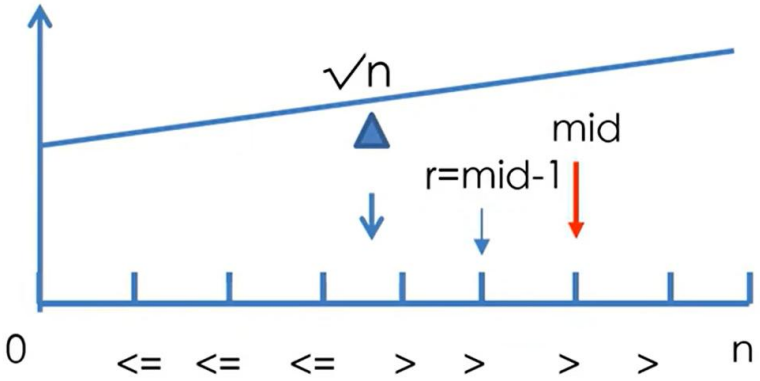
2 阅读程序写结果2[6605]

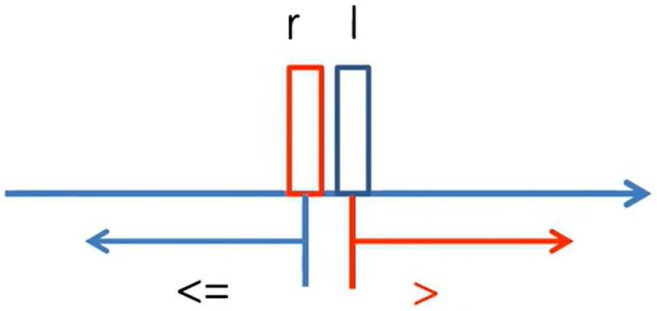
```
5  int n, k;
6
7  int solve1()
8  {
9      int l = 0, r = n;
10     while (l <= r) {
11         int mid = (l + r) / 2;
12         if (mid * mid <= n) l = mid + 1;
13         else r = mid - 1;
14     }
15     return l - 1;
16 }
17
18 double solve2(double x)
19 {
20     if (x == 0) return x;
21     for (int i = 0; i < k; i++)
22         x = (x + n / x) / 2;
23     return x;
24 }
```

本程序考查二分法及牛顿迭代法求算术平方根。solve1函数用二分法求出近似的算术平方根，然后用solve2函数进行牛顿迭代法，求出n的算术平方根。k为迭代的次数。

```
26 int main()
27 {
28     cin >> n >> k;
29     double ans = solve2(solve1());
30     cout << ans << ' ' << (ans * ans == n) << endl;
31     return 0;
32 }
```

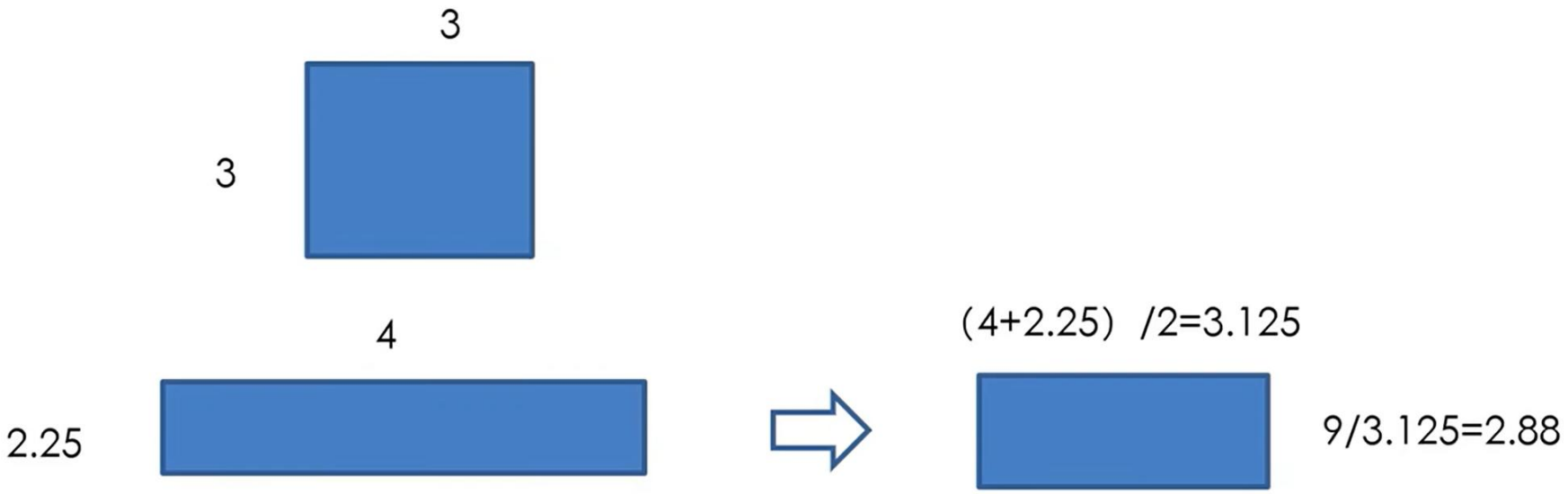
查找<=根号n的最大的整数。区间右端点r，每次改变的条件是：mid*mid>n，此时r=mid-1。所以，r的值是否>根号num不确定，但从r+1开始一定是>根号n





- 区间左端点l，每次改变的条件是：mid*mid<=n，此时l=mid+1。所以，l的值是否<=根号n不确定，但从l-1开始向左一定是<=根号n

牛顿迭代法求平方根。有一个正方形，面积是n。有一个长方形，面积也是n，长方形的长是x，宽是n/x。把长方形的（长+宽）/2作为新的长方形的长x，新的宽是n/x，再迭代…最后边长x趋向于根号n。



28. 该算法最准确的时间复杂度分析结果为。（ T）
29. 当输入为“9801 1”时，输出的第一个数为“99”。（ T ）
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”。（ F）
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。（ F ）

单选题

32. 当输入为“2 1”时，输出的第一个数最接近（ C ）。

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（ B）。

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 ）。

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

3 完善程序1 [6607] 枚举因数

```
4  int main() {
5      int n;
6      cin >> n;
7
8      vector<int> fac;
9      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  << " "; //②
20     }
21     if (③  ) { //③
22         cout << /④  << " "; //④
23     }
24     for (int k = fac.size() - 1; k >= 0; --k) {
25         cout <<⑤  << " "; //⑤
26     }
27 }
```

```cpp
int main() {
    int n;
    cin >> n;

    vector<int> fac;
    fac.reserve((int)ceil(sqrt(n)));

    int i;
    for (i = 1; i * i < n; ++i) {
        if (n % i == 0 ) { //①
            fac.push_back(i);
        }
    }

    for (int k = 0; k < fac.size(); ++k) {
        cout << fac[k]  << " "; //②
    }
    if (i * i == n ) { //③
        cout << i << " "; //④
    }
    for (int k = fac.size() - 1; k >= 0; --k) {
        cout << n / fac[k]  << " "; //⑤
    }
}
```

35. ①处应填（A ）

A. n % i == 0　　B. n % i == 1

C. n % (i-1) == 0　　D. n % (i-1) == 1

36. ②处应填（ B ）

A. n / fac[k]　　B. fac[k]

C. fac[k]-1　　D. n / (fac[k]-1)

37. ③处应填（ C ）

A. (i-1) * (i-1) == n　　B. (i-1) * i == n

C. i * 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)

3 完善程序2 [6608]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。

```
42  int main() {
43      char image[ROWS][COLS] = {
44      {'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g'},
45      {'g', 'g', 'g', 'g', 'g', 'g', 'r', 'r'},
46      {'g', 'r', 'r', 'g', 'g', 'r', 'g', 'g'},
47      {'g', 'b', 'b', 'b', 'b', 'r', 'g', 'r'},
48      {'g', 'g', 'g', 'b', 'b', 'r', 'g', 'r'},
49      {'g', 'g', 'g', 'b', 'b', 'b', 'b', 'r'},
50      {'g', 'g', 'g', 'g', 'g', 'b', 'g', 'g'},
51      {'g', 'g', 'g', 'g', 'g', 'b', 'b', 'g'}};
52      Point cur(4, 4);
53      char new_color = 'y';
54
55      flood_fill(image, cur, new_color);
56
57      for (int r = 0; r < ROWS; r++) {
58          for (int c = 0; c < COLS; c++) {
59              cout << image[r][c] << " ";
60          }
61          cout << endl;
62      }
63      return 0;
64  }
```

```cpp
#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 &&
        image[r][c] == prev_color  && image[r][c] != new_color); // ①
 }

```

```
20  void flood_fill(char image[ROWS][COLS], Point cur, int new_color) {
21      queue<Point> queue;
22      queue.push(cur);
23
24      int prev_color = image[cur.r][cur.c];
25      image[cur.r][cur.c] = new_color; //②
26
27      while (!queue.empty()) {
28          Point pt = queue.front();
29          queue.pop();
30
31          Point points[4] = {Point(pt.r+1, pt.c) , Point(pt.r - 1, pt.c), //③
32          Point(pt.r, pt.c + 1), Point(pt.r, pt.c - 1)};
33          for (auto p : points) {
34              if (is_valid(image, p, prev_color, new_color)) {
35                      image[p.r][p.c] = new_color; //④
36                      queue.push(p) ; //⑤
37              }
38          }
39      }
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

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

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)

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

⑤处应填（ ）

A.queue.push(p)　　B.queue.push(pt)

C.queue.push(cur)　　D.queue.push(Point(ROWS,COLS))

今天的课程结束啦……

下课了…
同学们再见！