专题训练

快速幂

（快速幂）请完善下面的程序，该程序使用分治法求x^p mod m的值。
输入格式:三个不超过10000 的正整数x，p，m。
输出格式:x^p mod m 的值。
提示：若 p 为偶数，$x^p=(x^2)^{p/2}$；若p 为奇数，$x^p=x*(x^2)^{(p-1)/2}$。

手算样例

$x^p \bmod m$

输入样例：
2 4 10000

输出多少？

输出样例：
16

写出计算步骤

$2^4 = (2^2)^2$

$4^2 = (4^2)^1$

$16^1 = 16*(16^2)^0$

$2^4 = 16$

提示：若 p 为偶数，$x^p=(x^2)^{p/2}$；若p 为奇数，$x^p=x*(x^2)^{(p-1)/2}$。

手算样例

$x^p \bmod m$

输入样例：
2 7 10000

输出多少？

输出样例：
128

写出计算步骤

$2^7 = 2*(2^2)^3$

$4^3 = 4*(4^2)^1$

$16^1 = 16*(16^2)^0$

$2^7 = 2*4*16 = 128$

提示：若 p 为偶数，$x^p=(x^2)^{p/2}$；若p 为奇数，$x^p=x*(x^2)^{(p-1)/2}$。

5分钟

完善程序

```
#include <iostream>
using namespace std;
int x, p, m, i, result;
int main() {
    cin >> x >> p >> m;
    result = ____(1)____ ;
    while ( ____(2)____ ) {
        if (p % 2 == 1)
            result = ____(3)____ ;
        p /= 2;
        x = ____(4)____ ;
    }
    cout << ____(5)____ << 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	

完善程序

```
#include <iostream>
using namespace std;
int x, p, m, i, result;
int main() {
    cin >> x >> p >> m;
    result = ____(1)____ ;
    while ( ____(2)____ ) {
        if (p % 2 == 1)
            result = ____(3)____ ;
        p /= 2;
        x = ____(4)____ ;
    }
    cout << ____(5)____ << endl;
    return 0;
}
```

解释变量的作用

x	底数
p	指数
m	对m取模
result	指数运算对m取模的结果
i	干扰变量，无用

完善程序

```
#include <iostream>
using namespace std;
int x, p, m, i, result;
int main() {
    cin >> x >> p >> m;
    result = ____(1)____ ;
    while ( ____(2)____ ) {
        if (p % 2 == 1)
            result = ____(3)____ ;
        p /= 2;
        x = ____(4)____ ;
    }
    cout << ____(5)____ << endl;
    return 0;
}
```

关键语句

按照分治思想计算快速幂

结束条件p值多少

当p>0时，算法会继续进行

完善程序

```
1  #include <iostream>
2  using namespace std;
3  int x, p, m, i, result;
4  int main() {
5      cin >> x >> p >> m;
6      result = ____(1)____ ;
7      while ( ____(2)____ ) {
8          if (p % 2 == 1)
9              result = ____(3)____ ;
10         p /= 2;
11         x = ____(4)____ ;
12     }
13     cout << ____(5)____ << endl;
14     return 0;
15 }
```

关键语句

p是奇数情况的处理

result=x*$(x^2)^{p/2}$

result=x*$(x^2)^{p/2}$
与偶数情况处理方式相同

完善程序

```
#include <iostream>
using namespace std;
int x, p, m, i, result;
int main() {
    cin >> x >> p >> m;
    result = ____(1)____ ;
    while ( ____(2)____ ) {
        if (p % 2 == 1)
            result = ____(3)____ ;
        p /= 2;
        x = ____(4)____ ;
    }
    cout << ____(5)____ << endl;
    return 0;
}
```

关键语句

p是偶数情况的处理

result=$(x^2)^{p/2}$

$x=x^2$

棋盘覆盖

（棋盘覆盖问题）在一个 $2^k \times 2^k$ 个方格组成的棋盘中恰有一个方格与其他方格不同（图中标记为 -1 的方格），称之为特殊方格。现用 L型（占 3个小格）纸片覆盖棋盘上除特殊方格的所有部分，各纸 片不得重叠，于是，用到的纸片数恰好是（4^k -1）/3。在下表给出的一个覆盖方案中，k=2，相同的 3 个数字构成一个纸片。下面给出的程序是用分治法设计的，将棋盘一分为四，依次处理左上角、右上角、左下角、右下角， 递归进行。请将程序补充完整。

2	2	3	3
2	-1	1	3
4	1	1	5
4	4	5	5

手算样例

输入

```
4
1 1
```

输出

```
-1  2  3  3
 2  2  1  3
 4  1  1  5
 4  4  5  5
```

输入

```
8
4 5
```

输出

```
 3  3  4  4  8  8  9  9
 3  2  2  4  8  7  7  9
 5  2  6  6 10 10  7 11
 5  5  6  1 -1 10 11 11
13 13 14  1  1 18 19 19
13 12 14 14 18 18 17 19
15 12 12 16 20 17 17 21
15 15 16 16 20 20 21 21
```

样例演示

k=1

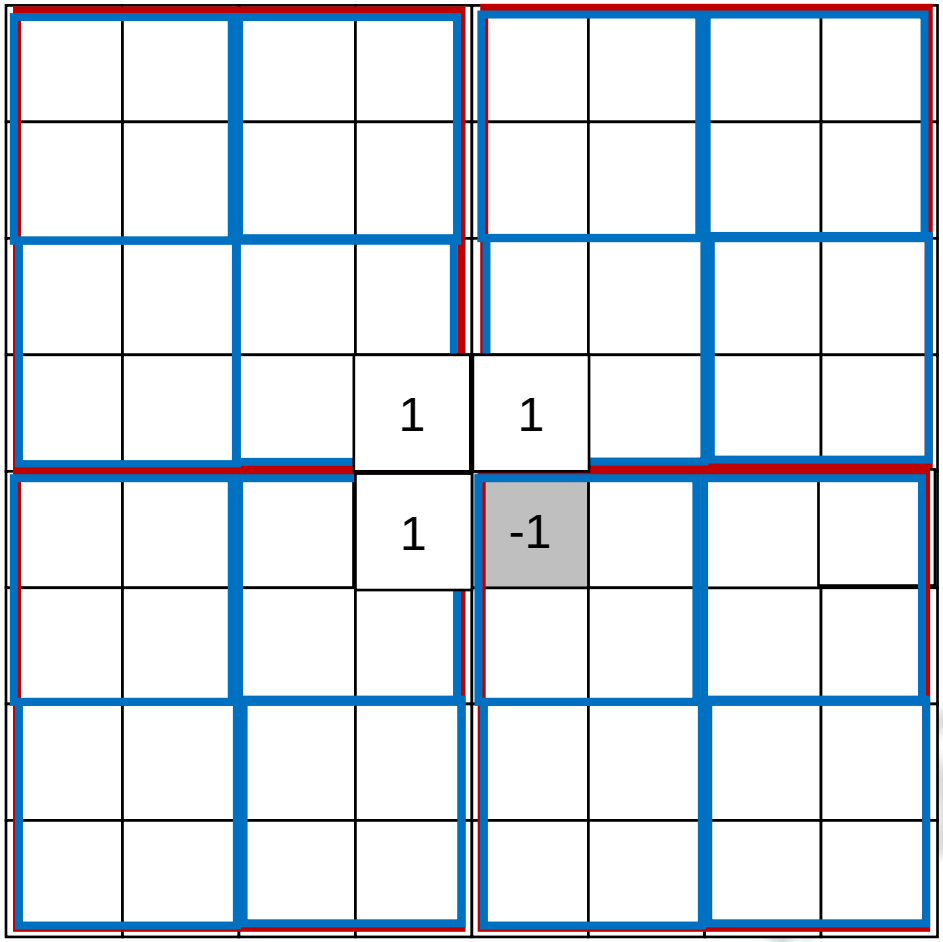
1	1
1	-1

k=2

-1	2	3	3
2	2	1	3
4	1	1	5
4	4	5	5

样例演示

k=3



完善程序

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  int board[65][65],tile; // tile 为纸片编号
5  void chessboard(int tr,int tc,int dr,int dc,int size) // dr,dc 依次为特殊方格的行、列号
41 void prt1(int b[][65],int n)
42 {
43     int i,j;
44     for(i=1;i<=n;i++)
45     {
46         for(j=1;j<=n;j++)
47         cout<<setw(3)<<b[i][j];
48         cout<<endl;
49     }
50 }
51 int main(){
52     int size,dr,dc;
53 //  cout<<"input size(4/8/16/64):"<<endl;
54     cin>>size;
55 //  cout<<"input the position of special block(x,y):"<<endl;
56     cin>>dr>>dc;
57     board[dr][dc]=-1;
58     tile++;
59     chessboard(1,1,dr,dc,size);
60     prt1(board,size);
61     return 0;
62 }
```

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	

完善程序

```
5  void chessboard(int tr,int tc,int dr,int dc,int size) // dr,dc 依次为特殊方格的行、列号
6  {
7      int t,s;
8      if (size==1) ____1____;
9      t=tile++;
10     s=size/2;
11     if(____2____)
12         chessboard(tr,tc,dr,dc,s);
13     else
14     {
15         board[tr+s-1][tc+s-1]=t;
16         ____3____;
17     }
18     if(dr<tr+s && dc>=tc+s)
19         chessboard(tr,tc+s,dr,dc,s);
20     else
21     {
22         board[tr+s-1][tc+s]=t;
23         ____4____;
24     }
25     if(dr>=tr+s && dc<tc+s)
26         chessboard(tr+s,tc,dr,dc,s);
27     else
28     {
29         board[tr+s][tc+s-1]=t;
30         ____5____;
31     }
32     if(dr>=tr+s && dc>=tc+s)
33         chessboard(tr+s,tc+s,dr,dc,s);
34     else
35     {
36         board[tr+s][tc+s]=t;
37         ____6____;
38     }
39 }
```

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	

完善程序

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  int board[65][65],tile; // tile 为纸片编号
5  void chessboard(int tr,int tc,int dr,int dc,int size) // dr,dc 依次为特殊方格的行、列号
41 void prt1(int b[][65],int n)
42 {
43     int i,j;
44     for(i=1;i<=n;i++)
45     {
46         for(j=1;j<=n;j++)
47         cout<<setw(3)<<b[i][j];
48         cout<<endl;
49     }
50 }
51 int main(){
52     int size,dr,dc;
53 //  cout<<"input size(4/8/16/64):"<<endl;
54     cin>>size;
55 //  cout<<"input the position of special block(x,y):"<<endl;
56     cin>>dr>>dc;
57     board[dr][dc]=-1;
58     tile++;
59     chessboard(1,1,dr,dc,size);
60     prt1(board,size);
61     return 0;
62 }
```

解释变量的作用

board	保存棋盘覆盖方案的纸片编号
chessboard()	填写棋盘覆盖方案
prt1()	print，打印方案
size	棋盘的边长
dr	特殊格的行号
dc	特殊格的列号

完善程序

```
5  void chessboard(int tr,int tc,int dr,int dc,int size) // dr,dc 依次为特殊方格的行、列号
6  {
7      int t,s;
8      if (size==1) ____1____;
9      t=tile++;
10     s=size/2;
11     if(____2____)
12         chessboard(tr,tc,dr,dc,s);
13     else
14     {
15         board[tr+s-1][tc+s-1]=t;
16         ____3____;
17     }
18     if(dr<tr+s && dc>=tc+s)
19         chessboard(tr,tc+s,dr,dc,s);
20     else
21     {
22         board[tr+s-1][tc+s]=t;
23         ____4____;
24     }
25     if(dr>=tr+s && dc<tc+s)
26         chessboard(tr+s,tc,dr,dc,s);
27     else
28     {
29         board[tr+s][tc+s-1]=t;
30         ____5____;
31     }
32     if(dr>=tr+s && dc>=tc+s)
33         chessboard(tr+s,tc+s,dr,dc,s);
34     else
35     {
36         board[tr+s][tc+s]=t;
37         ____6____;
38     }
39 }
```

tr	当前处理格子的左上角行号
tc	当前处理格子的左上角列号
dr	特殊格的行号
dc	特殊格的列号
size	当前处理的棋盘边长
t	当前可用的纸片编号
s	下一次递归处理棋盘的边长

关键语句

完善程序

```
5  void chessboard(int tr,int tc,int dr,int dc,int size) // dr,dc 依次为特殊方格的行、列号
6  {
7      int t,s;
8      if (size==1) ____1____;
9      t=tile++;
10     s=size/2;
11     if(____2____)
12         chessboard(tr,tc,dr,dc,s);
13     else
14     {
15         board[tr+s-1][tc+s-1]=t;
16         ____3____;
17     }
18     if(dr<tr+s && dc>=tc+s)
19         chessboard(tr,tc+s,dr,dc,s);
20     else
21     {
22         board[tr+s-1][tc+s]=t;
23         ____4____;
24     }
25     if(dr>=tr+s && dc<tc+s)
26         chessboard(tr+s,tc,dr,dc,s);
27     else
28     {
29         board[tr+s][tc+s-1]=t;
30         ____5____;
31     }
32     if(dr>=tr+s && dc>=tc+s)
33         chessboard(tr+s,tc+s,dr,dc,s);
34     else
35     {
36         board[tr+s][tc+s]=t;
37         ____6____;
38     }
39 }
```

棋盘边长为1，无需继续递归返回

递归填写左上1/4部分棋盘

递归填写右上1/4部分棋盘

递归填写左下1/4部分棋盘

递归填写右下1/4部分棋盘

完善程序

```
5   void chessboard(int tr,int tc,int dr,int dc,int size) // dr,d
6   {
7       int t,s;
8       if (size==1) ____1____;
9       t=tile++;
10      s=size/2;
11      if(____2____)
12          chessboard(tr,tc,dr,dc,s);
13      else
14      {
15          board[tr+s-1][tc+s-1]=t;
16          ____3____;
17      }
```

左上1/4棋盘开始坐标

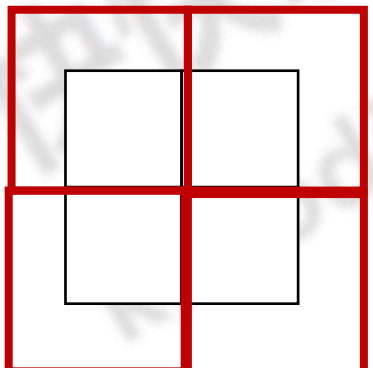
左上1/4棋盘特殊格坐标

关键语句

递归填写左上1/4部分棋盘

特殊格在左上方，继续递归填写左上方

特殊格不在左上方，那么将左上1/4棋盘的右下角作为特殊格，继续递归填写左上方棋盘



完善程序

关键语句

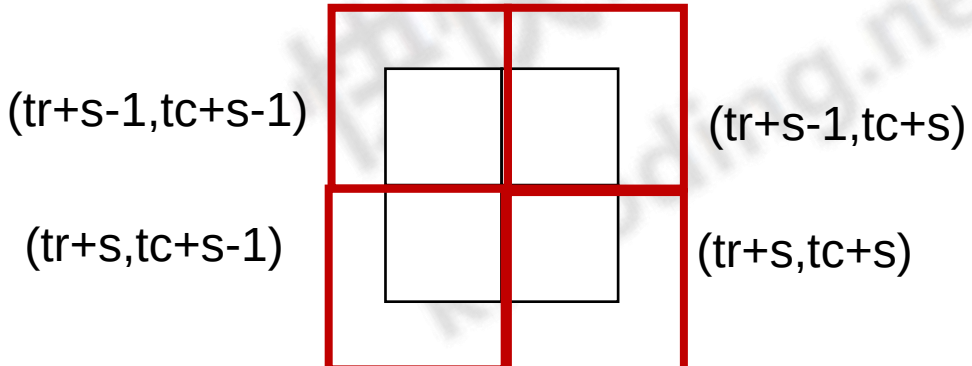
递归填写右上1/4部分棋盘

```
18
19
20
21
22
23
24
```

```
if(dr<tr+s && dc>=tc+s)
    chessboard(tr,tc+s,dr,dc,s);
else
{
    board[tr+s-1][tc+s]=t;
    ____4____;
}
```

特殊格在右上方，继续递归填写右上方

特殊格不在右上方，那么将右上1/4棋盘的左下角作为特殊格，继续递归填写右上方棋盘



完善程序

关键语句

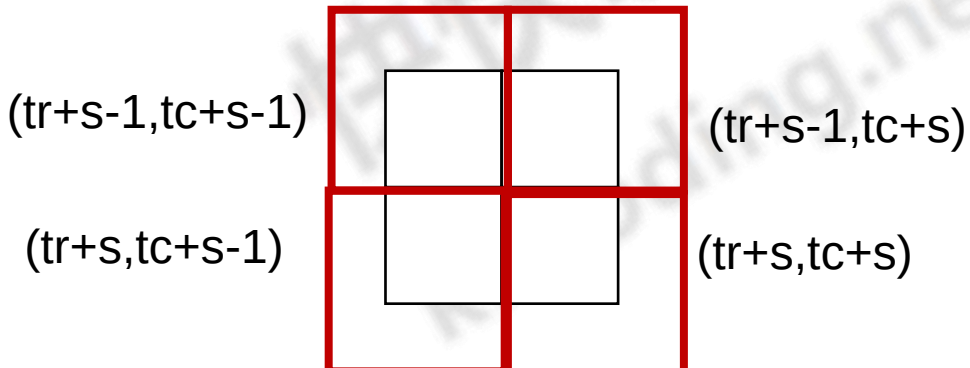
递归填写左下1/4部分棋盘

```
25
26
27
28
29
30
31
```

```
if(dr>=tr+s && dc<tc+s)
    chessboard(tr+s,tc,dr,dc,s);
else
{
    board[tr+s][tc+s-1]=t;
    ____5____;
}
```

特殊格在左下方，继续递归填写左下方

特殊格不在左下方，那么将左下1/4棋盘的右上角作为特殊格，继续递归填写左下方棋盘



完善程序

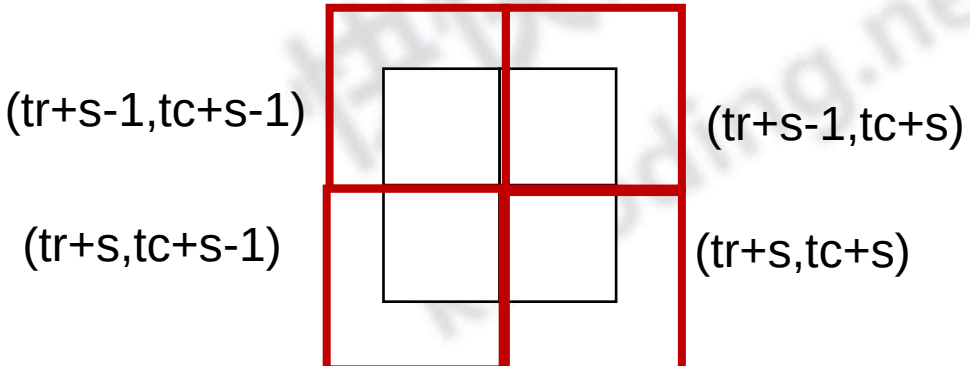
关键语句

递归填写右下1/4部分棋盘

```
32  if(dr>=tr+s && dc>=tc+s)
33      chessboard(tr+s,tc+s,dr,dc,s);
34  else
35  {
36      board[tr+s][tc+s]=t;
37      ____6____;
38  }
```

特殊格在右下方，继续递归填写右下方

特殊格不在右下方，那么将右下1/4棋盘的左上角作为特殊格，继续递归填写右下方棋盘



国王放置

在n*m的棋盘上放置k个国王，要求k个国王互相不攻击，有多少种不同的放置方法。假设国王放置在第(x,y)格，国王的攻击的区域是:(x-1,y-1), (x-1,y),(x-1,y+1),(x,y-1),(x,y+1),(x+1,y-1),(x+1,y),(x+1,y+1)。

题目利用回溯法求解。棋盘行标号为0~n-1，列标号为0~m-1。
输入格式:
读入三个数n,m,k，
输出格式:
输出总放置方法数。

完善程序

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4  int n,m,k,ans;
5  int hash[5][5];
```

```
43 int main()
44 {
45     cin >> n >> m >> k;
46     ans=0;
47     memset(hash,0,sizeof(hash));
48     ____(5)____;
49     cout << ans << endl;
50     return 0;
51 }
```

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	

完善程序

```
6   void work(int x,int y,int tot)
7   {
8       int i,j;
9       if (tot==k) {
10          ans++;
11          return;
12      }
13      do {
14          while (hash[x][y]) {
15              y++;
16              if (y==m) {
17                  x++;
18                  y=____(1)____;
19              }
20              if (x==n)
21                  return;
22          }
23          for (i=x-1; i<=x+1; i++)
24              if (i>=0&&i<n)
25                  for (j=y-1; j<=y+1; j++)
26                      if (j>=0&&j<m)
27                          ____(2)____;
28          ____(3)____;
```

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	

完善程序

```
28          ____(3)____;
29          for (i=x-1; i<=x+1; i++)
30              if (i>=0&&i<n)
31                  for (j=y-1; j<=y+1; j++)
32                      if (j>=0&&j<m)
33                          ____(4)____;
34          y++;
35          if (y==m) {
36              x++;
37              y=0;
38          }
39          if (x==n)
40              return;
41      } while (1);
42  }
```

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	

完善程序

解释变量的作用

```
#include <iostream>
#include <cstring>
using namespace std;
int n,m,k,ans;
int hash[5][5];
```

hash[i][j]	格子（i,j）受到周围几个国王的攻击
n	格子的行数
m	格子的列数

```
int main()
{
    cin >> n >> m >> k;
    ans=0;
    memset(hash,0,sizeof(hash));
    ____(5)____;
    cout << ans << endl;
    return 0;
}
```

完善程序

解释变量的作用

```
6   void work(int x,int y,int tot)
7   {
8       int i,j;
9       if (tot==k) {
10          ans++;
11          return;
12      }
13      do {
14          while (hash[x][y]) {
15              y++;
16              if (y==m) {
17                  x++;
18                  y=____(1)____;
19              }
20              if (x==n)
21                  return;
22          }
23          for (i=x-1; i<=x+1; i++)
24              if (i>=0&&i<n)
25                  for (j=y-1; j<=y+1; j++)
26                      if (j>=0&&j<m)
27                          ____(2)____;
28          ____(3)____;
```

x	当前处理的格子的x坐标
y	当前处理的格子的y坐标
tot	放置的国王数量
m	棋盘的列数
n	棋盘的行数

完善程序

解释变量的作用

```
28          ____(3)____;
29          for (i=x-1; i<=x+1; i++)
30              if (i>=0&&i<n)
31                  for (j=y-1; j<=y+1; j++)
32                      if (j>=0&&j<m)
33                          ____(4)____;
34          y++;
35          if (y==m) {
36              x++;
37              y=0;
38          }
39          if (x==n)
40              return;
41      } while (1);
42  }
```

x	当前处理的格子的x坐标
y	当前处理的格子的y坐标
m	棋盘的列数
n	棋盘的行数

完善程序

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4  int n,m,k,ans;
5  int hash[5][5];
```

关键语句

```
43 int main()
44 {
45     cin >> n >> m >> k;
46     ans=0;
47     memset(hash,0,sizeof(hash));
48     ____(5)____;
49     cout << ans << endl;
50     return 0;
51 }
```

hash数组初始化清零

函数调用，填写初始参数

完善程序

```
 6  void work(int x,int y,int tot)
 7  {
 8      int i,j;
 9      if (tot==k) {
10          ans++;
11          return;
12      }
13      do {
14          while (hash[x][y]) {
15              y++;
16              if (y==m) {
17                  x++;
18                  y=____(1)____;
19              }
20              if (x==n)
21                  return;
22          }
23          for (i=x-1; i<=x+1; i++)
24              if (i>=0&&i<n)
25                  for (j=y-1; j<=y+1; j++)
26                      if (j>=0&&j<m)
27                          ____(2)____;
28          ____(3)____;
```

关键语句

放置国王数量满足k，找到一个方案，方案数+1，返回

(x,y)上受到其他国王攻击，不能放置，枚举下一格。枚举规则是选右边一个，如果到m列则换行

超出棋盘行数，无解返回

(i,j)为(x,y)的8个邻格，标记被(x,y)的国王攻击

完善程序

关键语句

```
28          ____(3)____;
29          for (i=x-1; i<=x+1; i++)
30              if (i>=0&&i<n)
31                  for (j=y-1; j<=y+1; j++)
32                      if (j>=0&&j<m)
33                          ____(4)____;
34          y++;
35          if (y==m) {
36              x++;
37              y=0;
38          }
39          if (x==n)
40              return;
41      } while (1);
42  }
```

回溯，收回在(x,y)放置国王，同时撤销国王对周围8个邻格的攻击

枚举下一格

超出棋盘行数，无解返回

完善程序

（过河问题） 在一个月黑风高的夜晚,有一群人在河的右岸,想通过唯一的一根独木桥走到河的左岸.在伸手不见五指的黑夜里,过桥时必须借照灯光来照明,不幸的是,他们只有一盏灯.另外,独木桥上最多能承受两个人同时经过,否则将会坍塌.每个人单独过独木桥都需要一定的时间,不同的人要的时间可能不同.两个人一起过独木桥时,由于只有一盏灯,所以需要的时间是较慢的那个人单独过桥所花费的时间.现在输入N(2<=N<1000)和这N个人单独过桥需要的时间,请计算总共最少需要多少时间,他们才能全部到达河左岸.

例如,有3个人甲、乙、丙,他们单独过桥的时间分别为1、2、4,则总共最少需要的时间为7.具体方法是:甲、乙一起过桥到河的左岸,甲单独回到河的右岸将灯带回,然后甲、丙在一起过桥到河的左岸,总时间为2+1+4=7.

手算样例

输入样例：

```
2
4 5
```

输出多少？

```
5
```

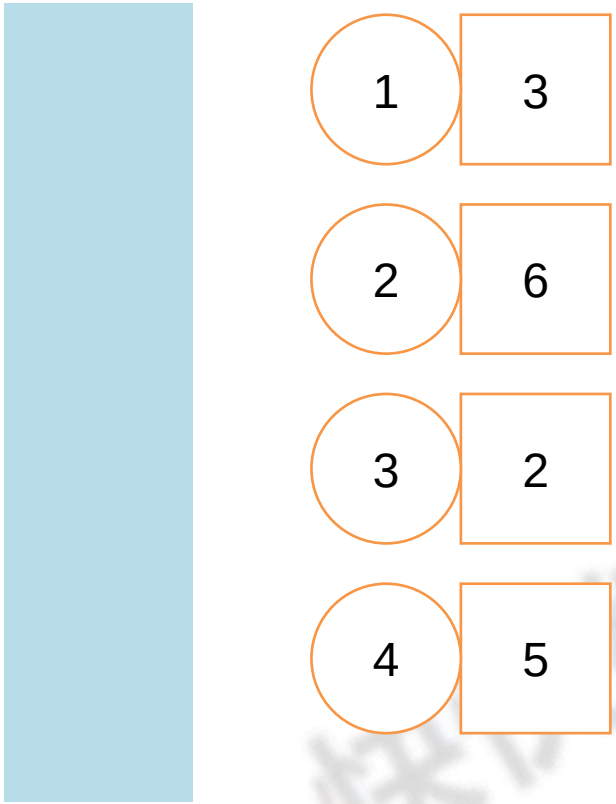
手算样例

输入样例：
4 3 6 2 5

输出多少？
17

样例演示

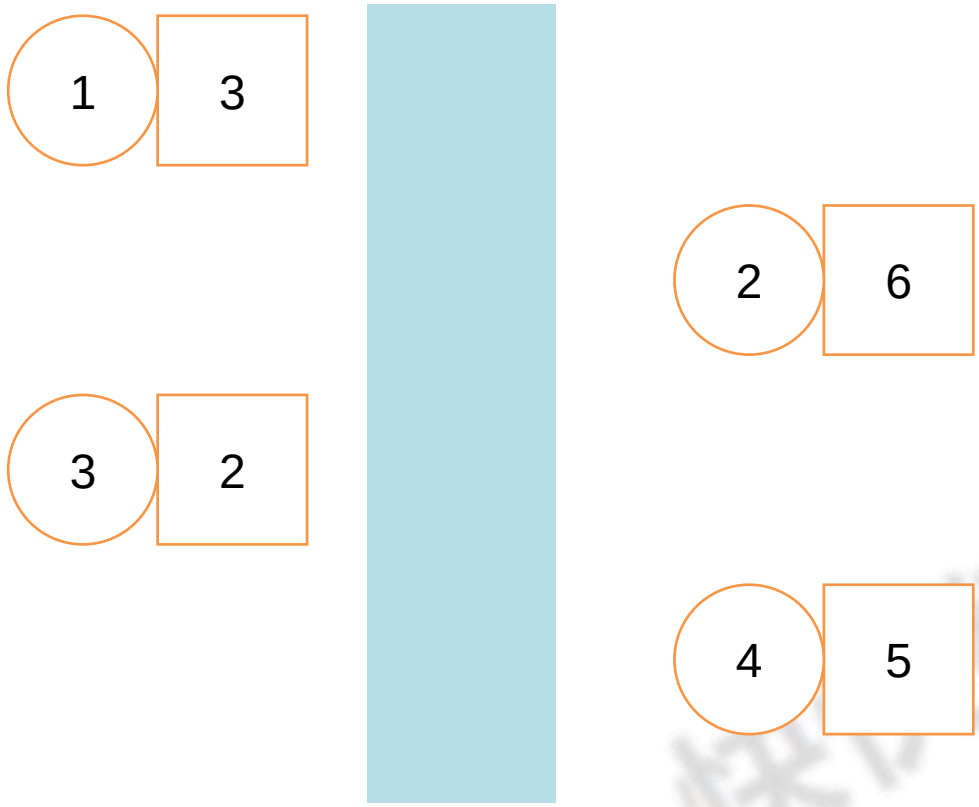
输入样例：
4
3 6 2 5



时长 0

样例演示

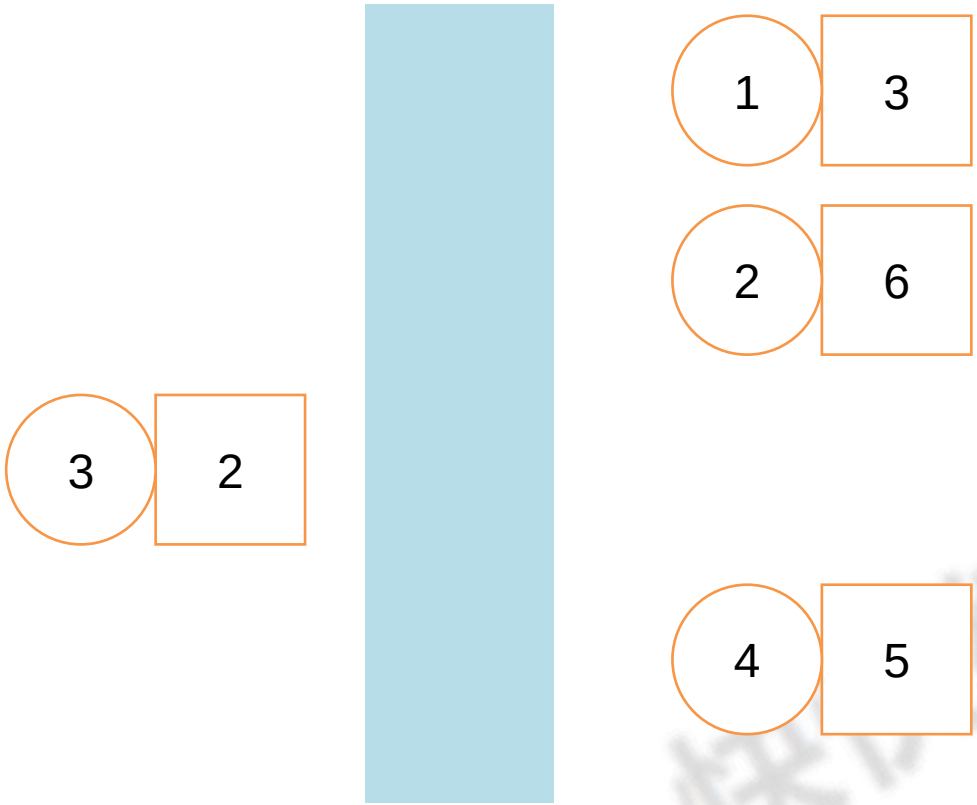
输入样例：
4
3 6 2 5



时长 3

样例演示

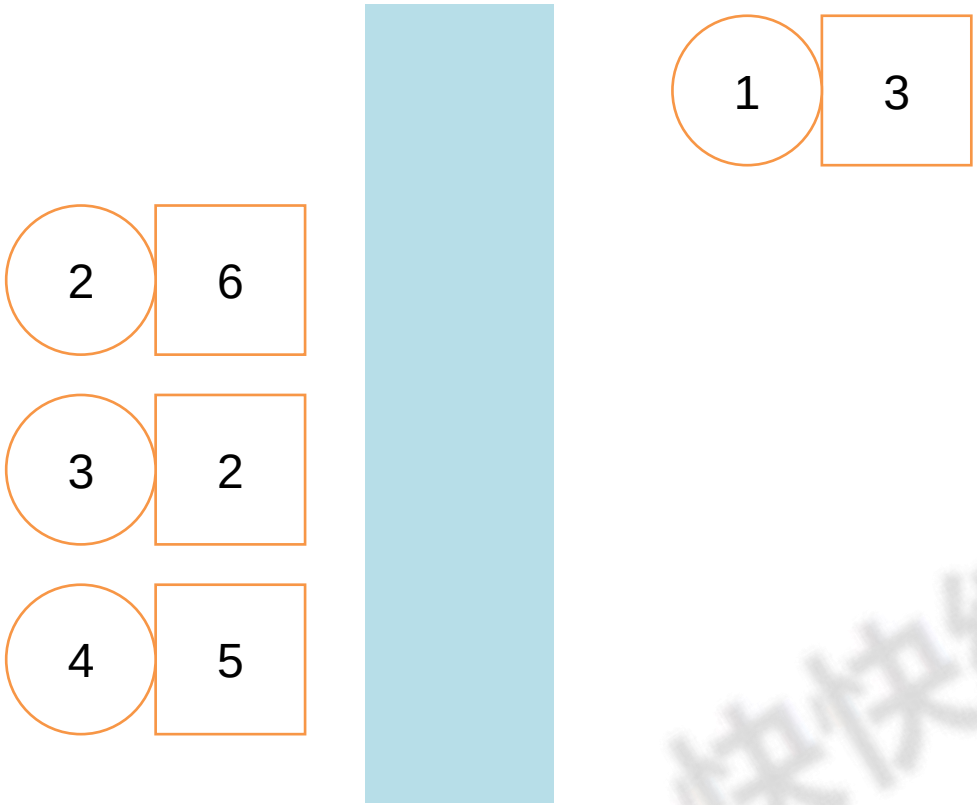
输入样例：
4
3 6 2 5



时长	6

样例演示

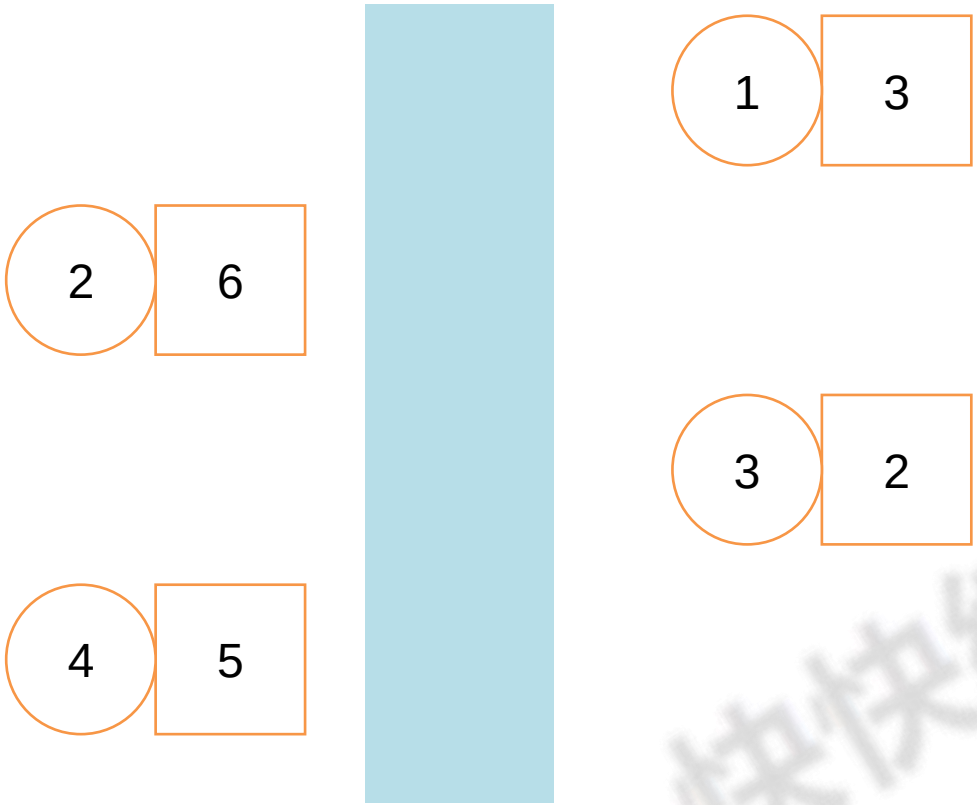
输入样例：
4
3 6 2 5



时长	12

样例演示

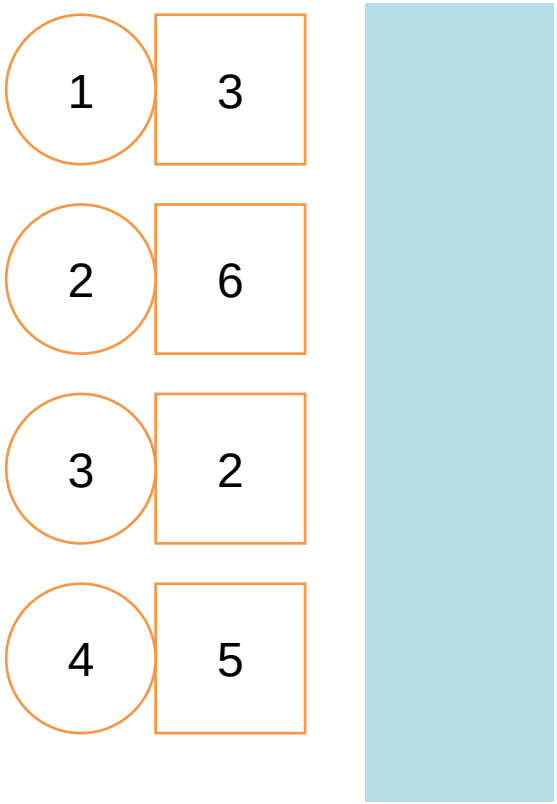
输入样例：
4
3 6 2 5



时长	14

样例演示

输入样例：
4
3 6 2 5



时长	17

1分钟

完善程序

```
#include<iostream>
#include<cstring>
using namespace std;
const int SIZE=100;
const int INFINITY = 10000;
const bool LEFT=true;
const bool RIGHT =false;
const bool LEFT_TO_RIGHT=true;
const bool RIGHT_TO_LEFT=false;
int n,hour[SIZE];
bool pos[SIZE];
int max(int a,int b){
    if(a>b)
        return a;
    else
        return b;
}
```

1	准备工作
记录关键常量及变量 写出已知完整函数功能	

可以根据变量名称推算变量作用

5分钟

完善程序

```
int go(bool stage){
    int i,j,num,tmp,ans;
    if(stage==RIGHT_TO_LEFT){
        num=0;
        ans=0;
        for(i=1;i<=n;i++)
            if(pos[i]==RIGHT){
                num++;
                if(hour[i]>ans)
                    ans=hour[i];
            }
        if(____(1)____)
            return ans;
        ans=INFINITY;
        for(i=1;i<=n-1;i++)
            if(pos[i]==RIGHT)
                for(j=i+1;j<=n;j++)
                    if(pos[j]==RIGHT){
                        pos[i]=LEFT;
                        pos[j]=LEFT;
                        tmp=max(hour[i],hour[j])+____(2)____;
                        if(tmp<ans)
                            ans=tmp;
                        pos[i]=RIGHT;
                        pos[j]=RIGHT;
                    }
        return ans;
    }
    if(stage==LEFT_TO_RIGHT){
        ans=INFINITY;
        for(i=1;i<=n;i++)
            if(____(3)____){
                pos[i]=RIGHT;
                tmp=____(4)____;
                if(tmp<ans)
                    ans=tmp;
                ____(5)____;
            }
        return ans;
    }
    return 0;
}
```

1	识别变量
2	找出关键语句
3	理解代码段作用

完善程序

关键语句

计算河右岸所有人完成过河总时长

```
18 int go(bool stage){
19     int i,j,num,tmp,ans;
20     if(stage==RIGHT_TO_LEFT){
21         num=0;
22         ans=0;
23         for(i=1;i<=n;i++)
24             if(pos[i]==RIGHT){
25                 num++;
26                 if(hour[i]>ans)
27                     ans=hour[i];
28             }
29         if(____(1)____)
30             return ans;
31         ans=INFINITY;
32         for(i=1;i<=n-1;i++)
33             if(pos[i]==RIGHT)
34                 for(j=i+1;j<=n;j++)
35                     if(pos[j]==RIGHT){
36                         pos[i]=LEFT;
37                         pos[j]=LEFT;
38                         tmp=max(hour[i],hour[j])+____(2)____;
39                         if(tmp<ans)
40                             ans=tmp;
41                         pos[i]=RIGHT;
42                         pos[j]=RIGHT;
43                     }
44         return ans;
45     }
```

num统计还在右岸的人数

ans记录这些人的最大值

递归边界，人数<=2时，两人直接过河，问题求解完毕

完善程序

关键语句

计算河右岸所有人完成过河总时长

```
int go(bool stage){
    int i,j,num,tmp,ans;
    if(stage==RIGHT_TO_LEFT){
        num=0;
        ans=0;
        for(i=1;i<=n;i++)
            if(pos[i]==RIGHT){
                num++;
                if(hour[i]>ans)
                    ans=hour[i];
            }
        if(____(1)____)
            return ans;
        ans=INFINITY;
        for(i=1;i<=n-1;i++)
            if(pos[i]==RIGHT)
                for(j=i+1;j<=n;j++)
                    if(pos[j]==RIGHT){
                        pos[i]=LEFT;
                        pos[j]=LEFT;
                        tmp=max(hour[i],hour[j])+____(2)____;
                        if(tmp<ans)
                            ans=tmp;
                        pos[i]=RIGHT;
                        pos[j]=RIGHT;
                    }
        return ans;
    }
```

双重循环枚举还在河右岸的i和j

i号和j号过河

总时长=此次过河时长+派一个人从左侧回来直到剩下人全部过河的时长

回溯状态

完善程序

计算从左往右返回一个人到所有人完成过河总时长

关键语句

```
46  if(stage==LEFT_TO_RIGHT){
47      ans=INFINITY;
48      for(i=1;i<=n;i++)
49          if(____(3)____){
50              pos[i]=RIGHT;
51              tmp=____(4)____;
52              if(tmp<ans)
53                  ans=tmp;
54              ____(5)____;
55          }
56      return ans;
57  }
58  return 0;
59 }
```

循环枚举在左侧返回的i

总时长=i号返回的时长+河右岸剩余所有人(包括i)过河时长

回溯状态

作业：完善程序

1998 快速幂

1994 棋盘覆盖

1989 国王放置

1995 过河问题