PHASE1

I turned the moon into something I call a Death Star.

输入wwwwwwwww测试，比较两个字符串是否相等

```
0x565564e2 <+21>:    push   %eax
0x565564e3 <+22>:    push   0x1c(%esp)
0x565564e7 <+26>:    call   0x56556a1a <strings_not_equal>
```

```
(gdb)
0x565564e3 in phase_1 ()
(gdb) i r eax
eax            0x56558144          1448444228
(gdb) x/s 0x56558144
0x56558144:     "I turned the moon into something I call a Death Star."
(gdb) i r esp
esp            0xffffcf24          0xffffcf24
(gdb) x/xw 0xffffcf24+0X1c
0xffffcf40:     0x5655b3a0
(gdb) x/s 0x5655b3a0
0x5655b3a0 <input_strings>:     "WWWWWWWWW"
```

PHASE2

0 1 1 2 3 5

```
0x0000154a <+75>:    add    $0x4,%esi
# 将 esi 指针向前移动 4 字节（指向下一个数字）
0x0000154d <+78>:    cmp    %edi,%esi
# 比较 esi 和 edi
0x0000154f <+80>:    je     0x1562 <phase_2+99>
# 如果相等，跳到 1562，结束循环
0x00001551 <+82>:    mov    0x4(%esi),%eax
# 将下一个数字（esi + 4）加载到 eax 中
0x00001554 <+85>:    add    (%esi),%eax
# 将当前数字（esi）加到 eax 中
0x00001556 <+87>:    cmp    %eax,0x8(%esi)
# 比较 eax 和下一个数字（esi + 8）
0x00001559 <+90>:    je     0x154a <phase_2+75>
# 如果相等，跳回 154a，处理下一组数字
0x0000155b <+92>:    call   0x1b32 <explode_bomb>
# 如果不相等，调用 explode_bomb，程序失败
```

PHASE3

```
1 151      edx:0x565565d5
2 478      edx:0x565565f2
3 53       edx:0x565565f9
4 683      edx:0x56556600
5 231      edx:0x56556607
6 266      edx:0x5655660e
7 129      edx:0x56556615
```

PHASE4

phase_4:

输入两个数x,y

x地址为：%esp+0x8

y地址为：%esp+0x4

```
0x565566ac <+48>:   call   0x56556140 <__isoc99_sscanf@plt>
0x565566b1 <+53>:   add    $0x10,%esp
0x565566b4 <+56>:   cmp    $0x2,%eax
0x565566b7 <+59>:   jne    0x565566c5 <phase_4+73>
```

phase_4:

2<=y<=4

```
0x565566b9 <+61>:   mov    0x4(%esp),%eax
0x565566bd <+65>:   sub    $0x2,%eax
0x565566c0 <+68>:   cmp    $0x2,%eax
0x565566c3 <+71>:   jbe    0x565566ca <phase_4+78>
```

phase_4:

需要计算func4(9,y)

```
0x565566ca <+78>:   sub    $0x8,%esp
0x565566cd <+81>:   push   0xc(%esp)
0x565566d1 <+85>:   push   $0x9
0x565566d3 <+87>:   call   0x5655663b <func4>
```

func4:

%ebx为第一个参数，%edi始终为y

```
0x5655663b <+0>:    push   %edi
0x5655663c <+1>:    push   %esi
0x5655663d <+2>:    push   %ebx
0x5655663e <+3>:    mov    0x10(%esp),%ebx
0x56556642 <+7>:    mov    0x14(%esp),%edi
```

func4:

返回值为%eax

f(0,y)=0 第一个参数为0，返回0

f(1,y)=y 第一个参数为1，返回y(%edi的值)

```
0x56556646 <+11>:   mov    $0x0,%eax
0x5655664b <+16>:   test   %ebx,%ebx
0x5655664d <+18>:   jle    0x56556678 <func4+61>
0x5655664f <+20>:   mov    %edi,%eax
0x56556651 <+22>:   cmp    $0x1,%ebx
0x56556654 <+25>:   je     0x56556678 <func4+61>
```

func4:

f(n,y)=f(n-1,y)+f(n-2,y)+y

```
0x56556656 <+27>:   sub    $0x8,%esp
0x56556659 <+30>:   push   %edi
0x5655665a <+31>:   lea    -0x1(%ebx),%eax
0x5655665d <+34>:   push   %eax
0x5655665e <+35>:   call   0x5655663b <func4>
0x56556663 <+40>:   add    $0x8,%esp
0x56556666 <+43>:   lea    (%eax,%edi,1),%esi
0x56556669 <+46>:   push   %edi
0x5655666a <+47>:   sub    $0x2,%ebx
0x5655666d <+50>:   push   %ebx
0x5655666e <+51>:   call   0x5655663b <func4>
0x56556673 <+56>:   add    $0x10,%esp
0x56556676 <+59>:   add    %esi,%eax
```

phase_4:

最后判断x是否等于f(9,y)

```
0x565566db <+95>:    cmp    %eax,0x8(%esp)
0x565566df <+99>:    jne    0x565566f3 <phase_4+119>
```

func4:

每次调用func4会执行

```
0x5655663b <+0>:     push   %edi
0x5655663c <+1>:     push   %esi
0x5655663d <+2>:     push   %ebx
...
0x56556678 <+61>:    pop    %ebx
0x56556679 <+62>:    pop    %esi
0x5655667a <+63>:    pop    %edi
0x5655667b <+64>:    ret
```

所以递归调用不会改变%edi，%esi，%ebx的值

f(9,2)=176,f(9,3)=264,f(9,4)=352

PHASE5

输入字符串长度为6，%esi为字符串首址

```
0x56556714 <+21>:    push   %esi
0x56556715 <+22>:    call   0x565569fc <string_length>
0x5655671a <+27>:    add    $0x10,%esp
0x5655671d <+30>:    cmp    $0x6,%eax
0x56556720 <+33>:    jne    0x5655674b <phase_5+76>
```

每个字符的ASCII码取低四位再乘4作为以%edi为首址的int型数组的偏移量，并将该偏移地址中的值加到%ecx中

```
0x56556722 <+35>:    mov    %esi,%eax
0x56556724 <+37>:    add    $0x6,%esi
0x56556727 <+40>:    mov    $0x0,%ecx
0x5655672c <+45>:    lea    -0x2d84(%ebx),%edi
0x56556732 <+51>:    movzbl (%eax),%edx
0x56556735 <+54>:    and    $0xf,%edx
```

```
0x56556738 <+57>:    add    (%edi,%edx,4),%ecx
0x5655673b <+60>:    add    $0x1,%eax
0x5655673e <+63>:    cmp    %esi,%eax
0x56556740 <+65>:    jne    0x56556732 <phase_5+51>
```

最终%ecx的值等于53(0x35)

```
0x56556742 <+67>:    cmp    $0x35,%ecx
0x56556745 <+70>:    jne    0x56556752 <phase_5+83>
```

查看以%edi为首址的int型数组信息

```
(gdb) i r edi
edi            0x565581e0          1448444384
(gdb) x/16dw 0x565581e0
0x565581e0 <array.0   >:    2   10  6   1
0x565581f0 <array.0+16>:    12  16  9   3
0x56558200 <array.0+32>:    4   7   14  5
0x56558210 <array.0+48>:    11  8   15  13
```

可以发现

```
10+6+1+12+16+8=53
```

因此偏移地址分别为

```
0x4 0x8 0x12 0x16 0x20 0x34
```

对应的字符低四位为

```
0x1 0x2 0x3 0x4 0x5 0xd
```

因此对应字符ASCII码可以为

```
0x31 0x32 0x33 0x34 0x35 0x3d
```

对应字符串为

```
12345=
```

PHASE6

需要读入6个整数

输入3 5 2 4 6 1

6个数放在一个数组中，数组首地址存放在%edi和%ebp，0xc(%esp)为当前索引值0

```
0x56556789 <+48>:   call   0x56556b67 <read_six_numbers>
0x5655678e <+53>:   mov    %edi,0x28(%esp)
0x56556792 <+57>:   add    $0x10,%esp
0x56556795 <+60>:   mov    %edi,0x10(%esp)
0x56556799 <+64>:   movl   $0x0,0xc(%esp)
0x565567a1 <+72>:   mov    %edi,%ebp
0x565567a3 <+74>:   jmp    0x565567c8 <phase_6+111>
```

```
(gdb) i r ebp
ebp            0xffffcedc          0xffffcedc
(gdb) x/dw 0xffffcedc
0xffffcedc:     3
(gdb) x/6dw 0xffffcedc
0xffffcedc:     3       5       2       4
0xffffceec:     6       1
```

0x10(%esp)为数组首地址，0x14(%esp)为第一个数，第一个数<=6

```
0x565567c8 <+111>:  mov    0x10(%esp),%eax
0x565567cc <+115>:  mov    %eax,%edi
0x565567ce <+117>:  mov    (%eax),%eax
0x565567d0 <+119>:  mov    %eax,0x14(%esp)
0x565567d4 <+123>:  sub    $0x1,%eax
0x565567d7 <+126>:  cmp    $0x5,%eax
0x565567da <+129>:  ja     0x565567a5 <phase_6+76>
...
0x565567a5 <+76>:   call   0x56556b32 <explode_bomb>
```

索引加1(为1)，用%esi(当前为1)保存索引，%esi<=5则跳转

```
0x565567dc <+131>:  addl   $0x1,0xc(%esp)
0x565567e1 <+136>:  mov    0xc(%esp),%esi
0x565567e5 <+140>:  cmp    $0x5,%esi
0x565567e8 <+143>:  jle    0x565567b4 <phase_6+91>
```

a[1]!=a[0],否则爆炸

```
0x565567b4 <+91>:    mov    0x0(%ebp,%esi,4),%eax
0x565567b8 <+95>:    cmp    %eax,(%edi)
0x565567ba <+97>:    jne    0x565567ac <phase_6+83>
0x565567bc <+99>:    call   0x56556b32 <explode_bomb>
```

%esi(1)加1，%esi=6则跳转

a[0]以后的所有数都不等于a[0],否则爆炸

```
0x565567ac <+83>:    add    $0x1,%esi
0x565567af <+86>:    cmp    $0x6,%esi
0x565567b2 <+89>:    je     0x565567c3 <phase_6+106>
0x565567b4 <+91>:    mov    0x0(%ebp,%esi,4),%eax
0x565567b8 <+95>:    cmp    %eax,(%edi)
0x565567ba <+97>:    jne    0x565567ac <phase_6+83>
0x565567bc <+99>:    call   0x56556b32 <explode_bomb>
```

跳转至106行时，%esi=6，0x10(%esp)存放首地址

执行完106行，0x10(%esp)存放第二个数地址，%edi也更新

0x14(%esp)存放第二个数，第二个数<=6

```
0x565567c3 <+106>:   addl   $0x4,0x10(%esp)
0x565567c8 <+111>:   mov    0x10(%esp),%eax
0x565567cc <+115>:   mov    %eax,%edi
0x565567ce <+117>:   mov    (%eax),%eax
0x565567d0 <+119>:   mov    %eax,0x14(%esp)
0x565567d4 <+123>:   sub    $0x1,%eax
0x565567d7 <+126>:   cmp    $0x5,%eax
0x565567da <+129>:   ja     0x565567a5 <phase_6+76>
...
0x565567a5 <+76>:    call   0x56556b32 <explode_bomb>
```

索引加1(为2)，用%esi(当前为2)保存索引，%esi<=5则跳转

```
0x565567dc <+131>:   addl   $0x1,0xc(%esp)
0x565567e1 <+136>:   mov    0xc(%esp),%esi
0x565567e5 <+140>:   cmp    $0x5,%esi
0x565567e8 <+143>:   jle    0x565567b4 <phase_6+91>
```

当前%edi为第二个数地址，a[2]!=a[1]，否则爆炸

```
0x565567b4 <+91>:   mov    0x0(%ebp,%esi,4),%eax
0x565567b8 <+95>:   cmp    %eax,(%edi)
0x565567ba <+97>:   jne    0x565567ac <phase_6+83>
0x565567bc <+99>:   call   0x56556b32 <explode_bomb>
```

%esi(2)加1，%esi=6则跳转

a[1]以后的所有数都不等于a[1],否则爆炸

```
0x565567ac <+83>:   add    $0x1,%esi
0x565567af <+86>:   cmp    $0x6,%esi
0x565567b2 <+89>:   je     0x565567c3 <phase_6+106>
0x565567b4 <+91>:   mov    0x0(%ebp,%esi,4),%eax
0x565567b8 <+95>:   cmp    %eax,(%edi)
0x565567ba <+97>:   jne    0x565567ac <phase_6+83>
0x565567bc <+99>:   call   0x56556b32 <explode_bomb>
```

继续循环下去

6个数互不相等且均<=6

循环完后运行至145行

0x18(%esp)和0x1c(%esp)为数组首地址，a[n]=7-a[n]

```
0x565567ea <+145>:  mov    0x1c(%esp),%edx
0x565567ee <+149>:  add    $0x18,%edx
0x565567f1 <+152>:  mov    $0x7,%ecx
0x565567f6 <+157>:  mov    0x18(%esp),%eax
0x565567fa <+161>:  mov    %ecx,%esi
0x565567fc <+163>:  sub    (%eax),%esi
0x565567fe <+165>:  mov    %esi,(%eax)
0x56556800 <+167>:  add    $0x4,%eax
0x56556803 <+170>:  cmp    %eax,%edx
0x56556805 <+172>:  jne    0x565567fa <phase_6+161>
```

%esp+0x2c即为数组首地址

```
0x56556807 <+174>:  mov    $0x0,%esi
0x5655680c <+179>:  mov    %esi,%edi
0x5655680e <+181>:  mov    0x2c(%esp,%esi,4),%ecx
0x56556812 <+185>:  mov    $0x1,%eax
```

```
   0x56556817 <+190>:   lea    0x168(%ebx),%edx
   0x5655681d <+196>:   cmp    $0x1,%ecx
   0x56556820 <+199>:   jle    0x5655682c <phase_6+211>
```

```
0x56556807 in phase_6 ()
(gdb) i r esp
esp            0xffffceb0          0xffffceb0
(gdb) x/72ubx 0xffffceb0
0xffffceb0:     0x01    0x00    0x00    0x00    0x6e    0x80    0x55    0x56
0xffffceb8:     0x03    0x00    0x00    0x00    0x06    0x00    0x00    0x00
0xffffcec0:     0xf0    0xce    0xff    0xff    0x01    0x00    0x00    0x00
0xffffcec8:     0xdc    0xce    0xff    0xff    0xdc    0xce    0xff    0xff
0xffffced0:     0xf0    0xd0    0xff    0xff    0x0a    0x00    0x00    0x00
0xffffced8:     0xf9    0x64    0xda    0xf7    0x04    0x00    0x00    0x00
0xffffcee0:     0x02    0x00    0x00    0x00    0x05    0x00    0x00    0x00
0xffffcee8:     0x03    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0xffffcef0:     0x06    0x00    0x00    0x00    0x50    0x00    0x00    0x00
(gdb) x/6dw 0xffffcedc
0xffffcedc:     4       2       5       3
0xffffceec:     1       6
```

初始时，%edx为链表首地址，1个结点占12个字节

```
   0x56556822 <+201>:   mov    0x8(%edx),%edx
   0x56556825 <+204>:   add    $0x1,%eax
   0x56556828 <+207>:   cmp    %ecx,%eax
   0x5655682a <+209>:   jne    0x56556822 <phase_6+201>
```

```
[key|value]
[1|457] -> [2|863] -> [3|513] -> [4|516] -> [5|928] -> [6|751] -> NULL
```

```
(gdb)
0x5655681d in phase_6 ()
(gdb) i r edx
edx            0x5655b0cc          1448456396
(gdb) x/24xw 0x5655b0cc
0x5655b0cc <node1>:     0x000001c9      0x00000001      0x5655b0d8      0x0000035f
0x5655b0dc <node2+4>:   0x00000002      0x5655b0e4      0x00000201      0x00000003
0x5655b0ec <node3+8>:   0x5655b0f0      0x00000204      0x00000004      0x5655b0fc
0x5655b0fc <node5>:     0x000003a0      0x00000005      0x5655b068      0x00000000
0x5655b10c:     0x00000000      0x00000000      0x00000000      0x00000000
0x5655b11c:     0x00000000      0x56558399      0x565583b3      0x565583cd
(gdb) x/3xw 0x5655b068
0x5655b068 <node6>:     0x000002ef      0x00000006      0x00000000
```

%esp+0x44处依次存放链表第a[n]个结点的地址(n:0~5)

```
0x5655682c <+211>:  mov    %edx,0x44(%esp,%edi,4)
0x56556830 <+215>:  add    $0x1,%esi
0x56556833 <+218>:  cmp    $0x6,%esi
0x56556836 <+221>:  jne    0x5655680c <phase_6+179>
```

```
(gdb) i r esp
esp            0xffffceb0          0xffffceb0
(gdb) x/24ubx 0xffffceb0+0x44
0xffffcef4:     0xf0    0xb0    0x55    0x56    0xd8    0xb0    0x55    0x56
0xffffcefc:     0xfc    0xb0    0x55    0x56    0xe4    0xb0    0x55    0x56
0xffffcf04:     0xcc    0xb0    0x55    0x56    0x68    0xb0    0x55    0x56
```

重置链表(改变指针的连接)，其key值顺序与a[n]一致

```
0x56556838 <+223>:  mov    0x44(%esp),%esi
0x5655683c <+227>:  mov    0x48(%esp),%eax
0x56556840 <+231>:  mov    %eax,0x8(%esi)
0x56556843 <+234>:  mov    0x4c(%esp),%edx
0x56556847 <+238>:  mov    %edx,0x8(%eax)
0x5655684a <+241>:  mov    0x50(%esp),%eax
0x5655684e <+245>:  mov    %eax,0x8(%edx)
0x56556851 <+248>:  mov    0x54(%esp),%edx
0x56556855 <+252>:  mov    %edx,0x8(%eax)
0x56556858 <+255>:  mov    0x58(%esp),%eax
0x5655685c <+259>:  mov    %eax,0x8(%edx)
0x5655685f <+262>:  movl   $0x0,0x8(%eax)
0x56556866 <+269>:  mov    $0x5,%edi
0x5655686b <+274>:  jmp    0x56556875 <phase_6+284>
```

%esi为链表重置后的首地址

重置后的链表应降序排序

```
0x5655686d <+276>:  mov    0x8(%esi),%esi
0x56556870 <+279>:  sub    $0x1,%edi
0x56556873 <+282>:  je     0x56556885 <phase_6+300>
0x56556875 <+284>:  mov    0x8(%esi),%eax
0x56556878 <+287>:  mov    (%eax),%eax
0x5655687a <+289>:  cmp    %eax,(%esi)
0x5655687c <+291>:  jge    0x5655686d <phase_6+276>
```

所以链表key值顺序：5 2 6 4 3 1

a[n]：5 2 6 4 3 1

7-a[n]：2 5 1 3 4 6

SECRET_PHASE

每次通过一关后调用phase_defused()函数

检查是否通过六关,未通过则退出该函数

```
0x56556ceb <+27>:   cmpl   $0x6,0x42c(%ebx)
0x56556cf2 <+34>:   je     0x56556d0a <phase_defused+58>
```

需要在第四关添加特定字符串

```
0x56556d1c <+76>:   lea    -0x2bdb(%ebx),%eax
0x56556d22 <+82>:   push   %eax
0x56556d23 <+83>:   lea    0x52c(%ebx),%eax
0x56556d29 <+89>:   push   %eax
0x56556d2a <+90>:   call   0x56556140 <__isoc99_sscanf@plt>
```

```
0x56556d22 in phase_defused ()
(gdb) i r eax
eax            0x56558389          1448444809
(gdb) x/s 0x56558389
0x56558389:     "%d %d %s"
(gdb) ni
0x56556d23 in phase_defused ()
(gdb)
0x56556d29 in phase_defused ()
(gdb) i r eax
eax            0x5655b490          1448457360
(gdb) x/s 0x5655b490
0x5655b490 <input_strings+240>: "352 4"
```

%eax检查第四关是否输入了字符串，所以先随便输入123abc进行测试

```
0x56556d32 <+98>:   cmp    $0x3,%eax
0x56556d35 <+101>:  je     0x56556d4b <phase_defused+123>
```

调用strings_not_equal前依次将比较的两个字符串地址压栈

可知需要在第四关添加"DrEvil"

```
0x56556d4b <+123>:  sub    $0x8,%esp
0x56556d4e <+126>:  lea    -0x2bd2(%ebx),%eax
0x56556d54 <+132>:  push   %eax
0x56556d55 <+133>:  lea    0x18(%esp),%eax
0x56556d59 <+137>:  push   %eax
0x56556d5a <+138>:  call   0x56556a1a <strings_not_equal>
```

```
0x56556d54 in phase_defused ()
(gdb) i r eax
eax            0x56558392          1448444818
(gdb) x/s 0x56558392
0x56558392:     "DrEvil"
(gdb) ni
0x56556d55 in phase_defused ()
(gdb)
0x56556d59 in phase_defused ()
(gdb) i r eax
eax            0xffffcecc          -12596
(gdb) x/s 0xffffcecc
0xffffcecc:     "123abc"
(gdb) ni
0x56556d5a in phase_defused ()
(gdb) si
0x56556a1a in strings_not_equal ()
```

添加"DrEvil"后

```
0x56556d5a <+138>:  call   0x56556a1a <strings_not_equal>
0x56556d5f <+143>:  add    $0x10,%esp
0x56556d62 <+146>:  test   %eax,%eax
0x56556d64 <+148>:  jne    0x56556d37 <phase_defused+103>
0x56556d66 <+150>:  sub    $0xc,%esp
0x56556d69 <+153>:  lea    -0x2d0c(%ebx),%eax
0x56556d6f <+159>:  push   %eax
0x56556d70 <+160>:  call   0x565560f0 <puts@plt>
```

```
(gdb)
0x56556d6f in phase_defused ()
(gdb) i r eax
eax            0x56558258          1448444504
(gdb) x/s 0x56558258
0x56558258:     "Curses, you've found the secret phase!"
(gdb) ni
0x56556d70 in phase_defused ()
(gdb) ni
Curses, you've found the secret phase!
0x56556d75 in phase_defused ()
```

接下来进入secret_phase

```
0x56556d75 <+165>:  lea    -0x2ce4(%ebx),%eax
0x56556d7b <+171>:  mov    %eax,(%esp)
0x56556d7e <+174>:  call   0x565560f0 <puts@plt>
0x56556d83 <+179>:  call   0x565568f0 <secret_phase>
```

```
(gdb)
0x56556d7e in phase_defused ()
(gdb) i r eax
eax            0x56558280          1448444544
(gdb) x/s 0x56558280
0x56558280:     "But finding it and solving it are quite different..."
(gdb) ni
But finding it and solving it are quite different...
0x56556d83 in phase_defused ()
(gdb) si
0x565568f0 in secret_phase ()
```

输入1 3进行测试

```
0x56556908 <+24>:   push   $0xa
0x5655690a <+26>:   push   $0x0
0x5655690c <+28>:   push   %eax
0x5655690d <+29>:   call   0x565561b0 <strtol@plt>
```

```
(gdb)
0x5655690c in secret_phase ()
(gdb) i r eax
eax            0x5655b580          1448457600
(gdb) x/s 0x5655b580
0x5655b580 <input_strings+480>: "1 3"
```

第一个数<=1001

```
0x56556912 <+34>:    mov    %eax,%esi
0x56556914 <+36>:    lea    -0x1(%eax),%eax
0x56556917 <+39>:    add    $0x10,%esp
0x5655691a <+42>:    cmp    $0x3e8,%eax
0x5655691f <+47>:    ja     0x56556953 <secret_phase+99>
...
0x56556953 <+99>:    call   0x56556b32 <explode_bomb>
```

传入fun7函数的是第一个数和二叉树的根结点的地址

```
0x56556924 <+52>:    push   %esi
0x56556925 <+53>:    lea    0x114(%ebx),%eax
0x5655692b <+59>:    push   %eax
0x5655692c <+60>:    call   0x5655689f <fun7>
```

```
(gdb)
0x56556925 in secret_phase ()
(gdb) i r esi
esi            0x1                 1
(gdb) ni
0x5655692b in secret_phase ()
(gdb)
0x5655692c in secret_phase ()
(gdb) i r eax
eax            0x5655b078          1448456312
(gdb) x/32ubx 0x5655b078
0x5655b078 <n1>:        0x24    0x00    0x00    0x00    0x84    0xb0    0x55    0x56
0x5655b080 <n1+8>:      0x90    0xb0    0x55    0x56    0x08    0x00    0x00    0x00
0x5655b088 <n21+4>:     0xb4    0xb0    0x55    0x56    0x9c    0xb0    0x55    0x56
0x5655b090 <n22>:       0x32    0x00    0x00    0x00    0xa8    0xb0    0x55    0x56
```

进入fun7函数，%edx为根结点地址，%ecx为第一个数

```
0x565568a3 <+4>:    mov    0x10(%esp),%edx
0x565568a7 <+8>:    mov    0x14(%esp),%ecx
```

```
(gdb)
0x565568ab in fun7 ()
(gdb) i r edx
edx            0x5655b078          1448456312
(gdb) i r ecx
ecx            0x1                 1
```

第一个数小于结点值则进入左子树查找，大于则进入右子树查找

查找到或进入右子树，%eax赋为0

查找完左子树，%eax=2*%eax

查找完右子树，%eax=%eax*2+1

```
// %edx为0表示没找到，返回-1
0x565568ab <+12>:   test   %edx,%edx
0x565568ad <+14>:   je     0x565568e9 <fun7+74>
0x565568af <+16>:   mov    (%edx),%ebx
0x565568b1 <+18>:   cmp    %ecx,%ebx
0x565568b3 <+20>:   jg     0x565568c1 <fun7+34>
0x565568b5 <+22>:   mov    $0x0,%eax
0x565568ba <+27>:   jne    0x565568d4 <fun7+53>
...
0x565568c1 <+34>:   sub    $0x8,%esp
0x565568c4 <+37>:   push   %ecx
0x565568c5 <+38>:   push   0x4(%edx) //左子树
0x565568c8 <+41>:   call   0x5655689f <fun7>
0x565568cd <+46>:   add    $0x10,%esp
0x565568d0 <+49>:   add    %eax,%eax
0x565568d2 <+51>:   jmp    0x565568bc <fun7+29>
...
0x565568d4 <+53>:   sub    $0x8,%esp
0x565568d7 <+56>:   push   %ecx
0x565568d8 <+57>:   push   0x8(%edx) //右子树
0x565568db <+60>:   call   0x5655689f <fun7>
0x565568e0 <+65>:   add    $0x10,%esp
0x565568e3 <+68>:   lea    0x1(%eax,%eax,1),%eax
0x565568e7 <+72>:   jmp    0x565568bc <fun7+29>
```

```
...
0x565568bc <+29>:    add    $0x8,%esp
0x565568bf <+32>:    pop    %ebx
0x565568c0 <+33>:    ret
```

返回到secret_phase后，%eax的值需为3

二叉树仅有4行

所以该结点为第3行第4个107,或第4行第7个99

```
0x56556934 <+68>:    cmp    $0x3,%eax
0x56556937 <+71>:    jne    0x5655695a <secret_phase+106>
```

```
(gdb) i r edx
edx            0x5655b078          1448456312
(gdb) x/96ubx 0x5655b078
0x5655b078 <n1>:        0x24    0x00    0x00    0x00    0x84    0xb0    0x55    0x56
0x5655b080 <n1+8>:      0x90    0xb0    0x55    0x56    0x08    0x00    0x00    0x00
0x5655b088 <n21+4>:     0xb4    0xb0    0x55    0x56    0x9c    0xb0    0x55    0x56
0x5655b090 <n22>:       0x32    0x00    0x00    0x00    0xa8    0xb0    0x55    0x56
0x5655b098 <n22+8>:     0xc0    0xb0    0x55    0x56    0x16    0x00    0x00    0x00
0x5655b0a0 <n32+4>:     0x44    0xb0    0x55    0x56    0x2c    0xb0    0x55    0x56
0x5655b0a8 <n33>:       0x2d    0x00    0x00    0x00    0x08    0xb0    0x55    0x56
0x5655b0b0 <n33+8>:     0x50    0xb0    0x55    0x56    0x06    0x00    0x00    0x00
0x5655b0b8 <n31+4>:     0x14    0xb0    0x55    0x56    0x38    0xb0    0x55    0x56
0x5655b0c0 <n34>:       0x6b    0x00    0x00    0x00    0x20    0xb0    0x55    0x56
0x5655b0c8 <n34+8>:     0x5c    0xb0    0x55    0x56    0xc9    0x01    0x00    0x00
0x5655b0d0 <node1+4>:   0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

```
(gdb) x/12ubx 0x5655b014
0x5655b014 <n41>:       0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5655b01c <n41+8>:     0x00    0x00    0x00    0x00
(gdb) x/12ubx 0x5655b038
0x5655b038 <n42>:       0x07    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5655b040 <n42+8>:     0x00    0x00    0x00    0x00
(gdb) x/12ubx 0x5655b044
0x5655b044 <n43>:       0x14    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5655b04c <n43+8>:     0x00    0x00    0x00    0x00
(gdb) x/12ubx 0x5655b02c
0x5655b02c <n44>:       0x23    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5655b034 <n44+8>:     0x00    0x00    0x00    0x00
```

```
(gdb) x/12ubx 0x5655b008
0x5655b008 <n45>:       0x28    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5655b010 <n45+8>:     0x00    0x00    0x00    0x00
(gdb) x/12ubx 0x5655b050
0x5655b050 <n46>:       0x2f    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5655b058 <n46+8>:     0x00    0x00    0x00    0x00
(gdb) x/12ubx 0x5655b020
0x5655b020 <n47>:       0x63    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5655b028 <n47+8>:     0x00    0x00    0x00    0x00
(gdb) x/12ubx 0x5655b05c
0x5655b05c <n48>:       0xe9    0x03    0x00    0x00    0x00    0x00    0x00    0x00
0x5655b064 <n48+8>:     0x00    0x00    0x00    0x00
```