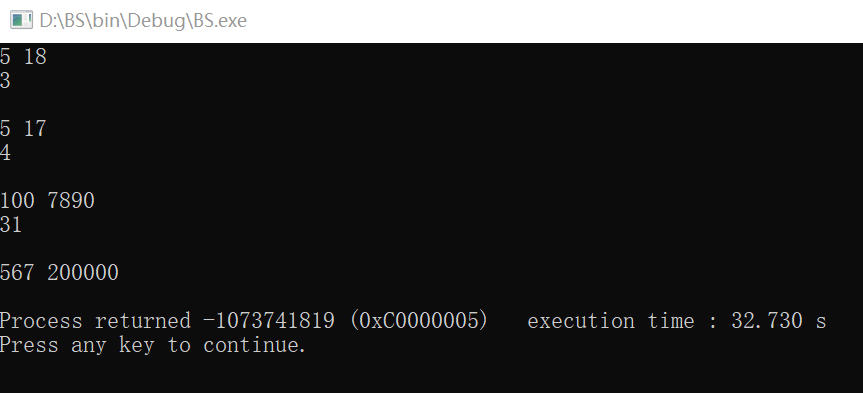
**1190201411 蒋婷婷**

**题目：**  
DESCRIPTION:  
Bob在和同学打篮球的时候，不小心把篮球扔到了很远的地方，他现在要捡回这个篮球。可是他想要在捡球的同时顺带练习技术动作，于是他有两种移动方式: 滑步 和 右跳步。  
假设Bob开始处于他与篮球连线上的一个点N（），而篮球在点K（）的位置。  
滑步：每一次Bob可以从任意点X移至X-1或者X+1的位置。  
右跳步：每一次Bob可以从任意点X跳到2\*X的位置。  
篮球静止，Bob至少用多少次，才能捡到篮球。  
   
INPUT:  
第1行：两个以空格分隔的整数：N和K  
   
OUTPUT:  
第一行为Bob所用最少次数捡到篮球  
   
SAMPLE INPUT:  
5 18  
   
SAMPLE OUTPUT:  
3  
   
NOTE:  
5->10->9->18  
  
提交要求:  
1.代码，运行截图 2.你的算法 3.使用的数据结构 4.时间复杂性，空间复杂性分析

一．代码

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. **typedef** **struct** node
5. {
6. **int** Nowpos;  //记录当前位置
7. **int** step;  //记录达到当前位置的步数
8. } node;
10. **typedef** **struct** queue //定义队列
11. {
12. **int** front;
13. **int** later;
14. node str[200000];
15. } queue;
17. queue bark;
18. **int** mark[200000];  //标记数组用来记录当前位置是否已经过遍历
20. //进入队列
21. **void** enqueue(node one)
22. {
23. bark.str[bark.later]=one;
24. bark.later++;
25. }
26. //弹出队列
27. node dequeue()
28. {
29. **int** two=bark.front;
30. bark.front++;
31. **return** bark.str[two];
32. }
33. //判断队列是否为空
34. **int** isempty()
35. {
36. **if**(bark.front == bark.later)
37. **return** 1;
38. **else**
39. **return** 0;
40. }
41. //计算到达篮球所在处需要经过的最少步骤
42. **int** minstep(**int** n,**int** end)
43. {
44. node one,two;
45. bark.front = bark.later=1;
46. one.Nowpos=n;
47. one.step=0;
48. mark[n]=1;
49. enqueue(one);    //第一个点进入队列
50. **while**(!isempty())
51. {
52. one=dequeue();
53. mark[one.Nowpos]=1;  //弹出队头元素，然后进行广度搜索
54. **if**(one.Nowpos==end)
55. **return** one.step; //满足条件，输出结果
56. **if**(one.Nowpos<end && 1!=mark[one.Nowpos+1]) ////往 X＋1 处移动
57. {
58. two.Nowpos = one.Nowpos+1;
59. two.step = one.step+1;
60. mark[two.Nowpos]=1;
61. enqueue(two);      //满足条件的节点入队
62. }
63. **if**(one.Nowpos>0 && 1!=mark[one.Nowpos-1])  //往 X-1 处移动
64. {
65. two.Nowpos = one.Nowpos-1;
66. two.step = one.step+1;
67. mark[two.Nowpos]=1;
68. enqueue(two);//满足条件的节点入队
69. }
70. **if**(one.Nowpos<end && 1!=mark[one.Nowpos\*2]) //可以向X\*2处移动
71. {
72. two.Nowpos=one.Nowpos\*2;
73. two.step=one.step+1;
74. mark[two.Nowpos]=1;
75. enqueue(two);//满足条件的节点入队
76. }
77. }
78. **return** 0;
79. }
80. **int** main()
81. {
82. **int** n,one;
83. **while**(scanf("%d%d",&n,&one)!=EOF)
84. {
85. memset(mark,0,**sizeof**(mark)); //将标记数组初始化
86. printf("%d\n",minstep(n,one));//输出所需最少步骤
87. }
88. **return** 0;
89. }

**二．程序运行截图：**

****

**三．算法：**

# 1.广度优先搜索算法（Breadth-First Search，BFS）按照移动步数依次遍历**队首**，后出队，并拓展出下一步所有可移动的位置并依次压入**队列**中，此处还使用了STL中的queue。

# 2.队列的顺序实现

**四．使用的数据结构**

**逻辑结构：受限线性表（队列）**

**存储结构：数组**

**五．时间复杂性，空间复杂性分析**

**空间复杂度分析：**

广度优先搜索算法需要两个变量，分别是一个辅助数组和一个辅助队列。对于辅助数组，若有n个节点，则其空间复杂度为O(n)；对于辅助队列需要进行一个入队操作，所以当情况最坏时其最大的空间复杂度也为O(n)，所以综合来看，其空间复杂度即为O(n)。

**时间复杂度分析：**

采用邻接表或者邻接矩阵其时间复杂度会有所不同。此处为邻接矩阵的存储方式。时间复杂度分为两个部分：1.访问每个节点所花费的时间，2.在每个节点处找下一步符合要求的移动位置所需的时间。

对于步骤1. 访问每个节点所花费的时间，n个节点需要O(n)；对于2.在每个节点处找下一步符合要求的移动位置所需的时间，由于是邻接矩阵，对于节点i，需要扫描第i行的每一个元素，循环的检验每一个值，看其是否符合要求，每一个节点都有一个循环的过程，所以为O( n²)。，取数量级较大的，所以为O( n²)。