

题目描述

羊、狼、农夫都在岸边，当羊的数量小于狼的数量时，狼会攻击羊，农夫则会损失羊。农夫有一艘容量固定的船，能够承载固定数量的动物。

要求求出不损失羊情况下将全部羊和狼运到对岸需要的最小次数。

只计算农夫去对岸的次数，回程时农夫不会运送羊和狼。

备注：农夫在或农夫离开后羊的数量大于狼的数量时狼不会攻击羊。

输入描述

第一行输入为M，N，X， 分别代表羊的数量，狼的数量，小船的容量。

输出描述

输出不损失羊情况下将全部羊和狼运到对岸需要的最小次数（若无法满足条件则输出0）。

用例

输入	5 3 3
输出	3
说明	第一次运2只狼 第二次运3只羊 第三次运2只羊和1只狼
输入	5 4 1
输出	0
说明	如果找不到不损失羊的运送方案，输出0

题目解析

本题求不损失羊的前提下，将羊和狼全部运到对岸的最小次数。

首先，要搞清楚，如何保证不损失羊？

农夫在或农夫离开后羊的数量大于狼的数量时狼不会攻击羊。

这里有个文字断句陷阱，到底是这样断句

- 农夫在时，狼不会攻击羊
- 农夫离开后羊的数量大于狼的数量时，狼不会攻击羊

还是这样断句

- 农夫在，羊的数量大于狼的数量时，狼不会攻击羊
- 农夫离开后，羊的数量大于狼的数量时，狼不会攻击羊

经过一位网友的真实机考反馈，上面画删除线的断句理解是错误的。

那么”农夫在时，狼不会攻击羊“，这句话到底会有什么影响呢？

只计算农夫去对岸的次数，回程时农夫不会运送羊和狼。

通过上面这句话，我们可以理解，农夫回程不会带动物。因此可以认定：

- 农夫如果在本岸带走的动物后，如果本岸羊 <= 狼了，在农夫离开后，羊就会被攻击
- 农夫如果在对岸离开后，对岸的羊 <= 狼，羊就会被攻击

题目解析

本题求不损失羊的前提下，将羊和狼全部运到对岸的最小次数。

首先，要搞清楚，如何保证不损失羊？

农夫在或农夫离开后羊的数量大于狼的数量时狼不会攻击羊。

这里有个文字断句陷阱，到底是这样断句

- 农夫在时，狼不会攻击羊
- 农夫离开后羊的数量大于狼的数量时，狼不会攻击羊

还是这样断句

- 农夫在，羊的数量大于狼的数量时，狼不会攻击羊
- 农夫离开后，羊的数量大于狼的数量时，狼不会攻击羊

经过一位网友的真实机考反馈，上面画删除线的断句理解是错误的。

那么“农夫在时，狼不会攻击羊”，这句话到底会有什么影响呢？

只计算农夫去对岸的次数，回程时农夫不会运送羊和狼。

通过上面这句话，我们可以理解，农夫回程不会带动物。因此可以认定：

- 农夫如果在本岸带走的动物后，如果本岸羊 \leq 狼了，在农夫离开后，羊就会被攻击
- 农夫如果在对岸离开后，对岸的羊 \leq 狼，羊就会被攻击

因此，“农夫在时，狼不会攻击羊”，这句话只会影响：船上，羊和狼的关系，即农夫在船上时，如果羊数量 \leq 狼数量，此时因为农夫在，因此狼不会攻击羊。

本题没有什么好的解题思路，只能通过暴力枚举[👉]所有羊、狼数量情况，只需要满足下面三个条件：

- 农夫离开后，本岸羊 $>$ 本岸狼
- 农夫离开后，对岸羊 $>$ 对岸狼
- 船上由于农夫始终在，因此羊、狼什么数量都可以，只要不超过船载量

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  rl.on("line", (line) => {
10    const [m, n, x] = line.split(" ").map(Number);
11
12    console.log(getResult(m, n, x));
13  });
14
15  function getResult(sheep, wolf, boat) {
16    const ans = [];
17    dfs(sheep, wolf, boat, 0, 0, 0, ans);
18
19    if (ans.length) {
20      return Math.min.apply(null, ans);
21    } else {
22      return 0;
23    }
24  }
25
26  function dfs(sheep, wolf, boat, oppo_sheep, oppo_wolf, count, ans) {
27    if (sheep === 0 && wolf === 0) {
28      ans.push(count);
29      return;
30    }
31
32    if (sheep + wolf <= boat) {
33      ans.push(count + 1);
34      return;
35    }
36
37    // i 代表船上羊数量，最多Math.min(boat, sheep)
38    for (let i = 0; i <= Math.min(boat, sheep); i++) {
39      // j 代表船上狼数量，最多Math.min(boat, wolf)
40      for (let j = 0; j <= Math.min(boat, wolf); j++) {
41        // 空运
42        if (i + j === 0) continue;
```

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  rl.on("line", (line) => {
10    const [m, n, x] = line.split(" ").map(Number);
11
12    console.log(getResult(m, n, x));
13  });
14
15  function getResult(sheep, wolf, boat) {
16    const ans = [];
17    dfs(sheep, wolf, boat, 0, 0, 0, ans);
18
19    if (ans.length) {
20      return Math.min.apply(null, ans);
21    } else {
22      return 0;
23    }
24  }
25
26  function dfs(sheep, wolf, boat, oppo_sheep, oppo_wolf, count, ans) {
27    if (sheep === 0 && wolf === 0) {
28      ans.push(count);
29      return;
30    }
31
32    if (sheep + wolf <= boat) {
33      ans.push(count + 1);
34      return;
35    }
36
37    // i 代表船上羊数量, 最多Math.min(boat, sheep)
38    for (let i = 0; i <= Math.min(boat, sheep); i++) {
39      // j 代表船上狼数量, 最多Math.min(boat, wolf)
40      for (let j = 0; j <= Math.min(boat, wolf); j++) {
41        // 空运
42        if (i + j === 0) continue;
43
44        // 超载
45        if (i + j > boat) break;
46
47        // 本岸羊 <= 本岸狼, 说明狼还少了
48        if (sheep - i <= wolf - j && sheep - i !== 0) continue;
49
50        // 对岸羊 <= 对岸狼, 说明狼还多了
51        if (oppo_sheep + i <= oppo_wolf + j && oppo_sheep + i !== 0) break;
52
53        // 对岸没羊, 但是对岸狼已经超过船数量, 即下次即便整船都运羊, 也无法保证对岸羊 > 对岸狼
54        if (oppo_sheep + i === 0 && oppo_wolf + j >= boat) break;
55
56        dfs(
57          sheep - i,
58          wolf - j,
59          boat,
60          oppo_sheep + i,
61          oppo_wolf + j,
62          count + 1,
63          ans
64        );
65      }
66    }
67  }
```

Java算法源码

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.Scanner;
4
5  public class Main {
6    public static void main(String[] args) {
7      Scanner sc = new Scanner(System.in);
8
9      int m = sc.nextInt();
10     int n = sc.nextInt();
11     int x = sc.nextInt();
12
13     System.out.println(getResult(m, n, x));
14   }
15
16   /**
17    * @param sheep 本岸羊数量
18    * @param wolf 本岸狼数量
19    * @param boat 船负载
20    * @return 最小运输次数
```

Java算法源码

```
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Scanner;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8
9         int m = sc.nextInt();
10        int n = sc.nextInt();
11        int x = sc.nextInt();
12
13        System.out.println(getResult(m, n, x));
14    }
15
16    /**
17     * @param sheep 本岸羊数量
18     * @param wolf 本岸狼数量
19     * @param boat 船负载
20     * @return 最少运送次数
21     */
22    public static int getResult(int sheep, int wolf, int boat) {
23        ArrayList<Integer> ans = new ArrayList<>();
24        dfs(sheep, wolf, boat, 0, 0, 0, ans);
25
26        if (ans.size() > 0) {
27            return Collections.min(ans);
28        } else {
29            return 0;
30        }
31    }
32
33    public static void dfs(
34        int sheep,
35        int wolf,
36        int boat,
37        int oppo_sheep,
38        int oppo_wolf,
39        int count,
40        ArrayList<Integer> ans) {
41        if (sheep == 0 && wolf == 0) {
42            ans.add(count);
43            return;
44        }
45
46        if (sheep + wolf <= boat) {
47            ans.add(count + 1);
48            return;
49        }
50
51        // i 代表船上羊数量, 最多Math.min(boat, sheep)
52        for (int i = 0; i <= Math.min(boat, sheep); i++) {
53            // j 代表船上狼数量, 最多Math.min(boat, wolf)
54            for (int j = 0; j <= Math.min(boat, wolf); j++) {
55                // 空运
56                if (i + j == 0) continue;
57                // 超载
58                if (i + j > boat) break;
59
60                // 本岸羊 <= 本岸狼, 说明狼运少了
61                if (sheep - i <= wolf - j && sheep - i != 0) continue;
62
63                // 对岸羊 <= 对岸狼, 说明狼运多了
64                if (oppo_sheep + i <= oppo_wolf + j && oppo_sheep + i != 0) break;
65
66                // 对岸没羊, 但是对岸狼已经超过船载量, 即下次即便整船都运羊, 也无法保证对岸羊 > 对岸狼
67                if (oppo_sheep + i == 0 && oppo_wolf + j >= boat) break;
68
69                dfs(sheep - i, wolf - j, boat, oppo_sheep + i, oppo_wolf + j, count + 1, ans);
70            }
71        }
72    }
73 }
```

Python算法源码

```
1 import math
2
3 # 输入获取
4 m, n, x = map(int, input().split())
5
6
7 # 算法入口
8 def getResult(sheep, wolf, boat):
9     ans = []
10    dfs(sheep, wolf, boat, 0, 0, 0, ans)
11
12    if len(ans) > 0:
13        return min(ans)
14    else:
```

Python算法源码

```
1 import math
2
3 # 输入获取
4 m, n, x = map(int, input().split())
5
6
7 # 算法入口
8 def getResult(sheep, wolf, boat):
9     ans = []
10    dfs(sheep, wolf, boat, 0, 0, 0, ans)
11
12    if len(ans) > 0:
13        return min(ans)
14    else:
15        return 0
16
17
18 def dfs(sheep, wolf, boat, oppo_sheep, oppo_wolf, count, ans):
19     if sheep == 0 and wolf == 0:
20         ans.append(count)
21         return
22
23     if sheep + wolf <= boat:
24         ans.append(count + 1)
25         return
26
27     # i 代表船上羊数量, 最多Math.min(boat, sheep)
28     for i in range(min(boat, sheep) + 1):
29         # j 代表船上狼数量, 最多Math.min(boat, wolf)
30         for j in range(min(boat, wolf) + 1):
31             # 空运
32             if i + j == 0:
33                 continue
34
35             # 超载
36             if i + j > boat:
37                 break
38
39             # 本岸羊 <= 本岸狼, 说明狼运少了
40             if sheep - i <= wolf - j and sheep - i != 0:
41                 continue
42
43             # 对岸羊 <= 对岸狼, 说明狼运多了
44             if oppo_sheep + i <= oppo_wolf + j and oppo_sheep + i != 0:
45                 break
46
47             # 对岸没羊, 但是对岸狼已经超过船载量, 即下次即使整船都运羊, 也无法保证对岸羊 > 对岸狼
48             if oppo_sheep + i == 0 and oppo_wolf + j >= boat:
49                 break
50
51             dfs(sheep - i, wolf - j, boat, oppo_sheep + i, oppo_wolf + j, count + 1, ans)
52
53
54 # 算法调用
55 print(getResult(m, n, x))
```

原型题

本题非常类似于《算法乐趣》一书中的：妖怪和和尚过河问题，关于此问题算法，JS可以参考下面文章

[Java乘船_妖怪和和尚过河问题\(javascript实现\)_王元祺的博客-CSDN博客](#)

也可以观看如下视频科普：

[S1E5 合作过河 River Crossing Riddle_哔哩哔哩_bilibili](#)

但是，上面妖怪过河问题是基于暴力枚举法+状态搜索树实现的，我试了一下5 3 3的用例，发现时间复杂度贼高。

因此可能不适合本题，在这里将这个思路告知大家，看看大家有没有什么思路，欢迎大家将见解在评论中发出来。