**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<br>第二次运3只羊<br>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 <= 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

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 <= 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**

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const [m, n, x] = line.split(" ").map(Number);

  console.log(getResult(m, n, x));
});

function getResult(sheep, wolf, boat) {
  const ans = [];
  dfs(sheep, wolf, boat, 0, 0, 0, ans);

  if (ans.length) {
    return Math.min.apply(null, ans);
  } else {
    return 0;
  }
}

function dfs(sheep, wolf, boat, oppo_sheep, oppo_wolf, count, ans) {
  if (sheep === 0 && wolf === 0) {
    ans.push(count);
    return;
  }

  if (sheep + wolf <= boat) {
    ans.push(count + 1);
    return;
  }

  // i 代表船上羊数量，最多Math.min(boat, sheep)
  for (let i = 0; i <= Math.min(boat, sheep); i++) {
    // j 代表船上狼数量，最多Math.min(boat, wolf)
    for (let j = 0; j <= Math.min(boat, wolf); j++) {
      // 空运
      if (i + j === 0) continue;
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const [m, n, x] = line.split(" ").map(Number);

  console.log(getResult(m, n, x));
});

function getResult(sheep, wolf, boat) {
  const ans = [];
  dfs(sheep, wolf, boat, 0, 0, 0, ans);

  if (ans.length) {
    return Math.min.apply(null, ans);
  } else {
    return 0;
  }
}

function dfs(sheep, wolf, boat, oppo_sheep, oppo_wolf, count, ans) {
  if (sheep === 0 && wolf === 0) {
    ans.push(count);
    return;
  }

  if (sheep + wolf <= boat) {
    ans.push(count + 1);
    return;
  }

  // i 代表船上羊数量，最多Math.min(boat, sheep)
  for (let i = 0; i <= Math.min(boat, sheep); i++) {
    // j 代表船上狼数量，最多Math.min(boat, wolf)
    for (let j = 0; j <= Math.min(boat, wolf); j++) {
      // 空运
      if (i + j === 0) continue;

      // 超载
      if (i + j > boat) break;

      // 本岸羊 <= 本岸狼，说明狼运少了
      if (sheep - i <= wolf - j && sheep - i !== 0) continue;

      // 对岸羊 <= 对岸狼，说明狼运多了
      if (oppo_sheep + i <= oppo_wolf + j && oppo_sheep + i !== 0) break;

      // 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
      if (oppo_sheep + i === 0 && oppo_wolf + j >= boat) break;

      dfs(
        sheep - i,
        wolf - j,
        boat,
        oppo_sheep + i,
        oppo_wolf + j,
        count + 1,
        ans
      );
    }
  }
}
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int m = sc.nextInt();
    int n = sc.nextInt();
    int x = sc.nextInt();

    System.out.println(getResult(m, n, x));
  }

  /**
   * @param sheep 本岸羊数量
   * @param wolf 本岸狼数量
   * @param boat 船负载
   * @return 最少运送次数
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int m = sc.nextInt();
    int n = sc.nextInt();
    int x = sc.nextInt();

    System.out.println(getResult(m, n, x));
  }

  /**
   * @param sheep 本岸羊数量
   * @param wolf 本岸狼数量
   * @param boat 船负载
   * @return 最少运送次数
   */
  public static int getResult(int sheep, int wolf, int boat) {
    ArrayList<Integer> ans = new ArrayList<>();
    dfs(sheep, wolf, boat, 0, 0, 0, ans);

    if (ans.size() > 0) {
      return Collections.min(ans);
    } else {
      return 0;
    }
  }

  public static void dfs(
      int sheep,
      int wolf,
      int boat,
      int oppo_sheep,
      int oppo_wolf,
      int count,
      ArrayList<Integer> ans) {
    if (sheep == 0 && wolf == 0) {
      ans.add(count);
      return;
    }

    if (sheep + wolf <= boat) {
      ans.add(count + 1);
      return;
    }

    // i 代表船上羊数量，最多Math.min(boat, sheep)
    for (int i = 0; i <= Math.min(boat, sheep); i++) {
      // j 代表船上狼数量，最多Math.min(boat, wolf)
      for (int j = 0; j <= Math.min(boat, wolf); j++) {
        // 空运
        if (i + j == 0) continue;
        // 超载
        if (i + j > boat) break;

        // 本岸羊 <= 本岸狼，说明狼运少了
        if (sheep - i <= wolf - j && sheep - i != 0) continue;

        // 对岸羊 <= 对岸狼，说明狼运多了
        if (oppo_sheep + i <= oppo_wolf + j && oppo_sheep + i != 0) break;

        // 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
        if (oppo_sheep + i == 0 && oppo_wolf + j >= boat) break;

        dfs(sheep - i, wolf - j, boat, oppo_sheep + i, oppo_wolf + j, count + 1, ans);
      }
    }
  }
}
```

Python算法源码

```python
import math

# 输入获取
m, n, x = map(int, input().split())


# 算法入口
def getResult(sheep, wolf, boat):
    ans = []
    dfs(sheep, wolf, boat, 0, 0, 0, ans)

    if len(ans) > 0:
        return min(ans)
    else:
```

Python算法源码

```python
import math

# 输入获取
m, n, x = map(int, input().split())


# 算法入口
def getResult(sheep, wolf, boat):
    ans = []
    dfs(sheep, wolf, boat, 0, 0, 0, ans)

    if len(ans) > 0:
        return min(ans)
    else:
        return 0


def dfs(sheep, wolf, boat, oppo_sheep, oppo_wolf, count, ans):
    if sheep == 0 and wolf == 0:
        ans.append(count)
        return

    if sheep + wolf <= boat:
        ans.append(count + 1)
        return

    # i 代表船上羊数量，最多Math.min(boat, sheep)
    for i in range(min(boat, sheep) + 1):
        # j 代表船上狼数量，最多Math.min(boat, wolf)
        for j in range(min(boat, wolf) + 1):
            # 空运
            if i + j == 0:
                continue

            # 超载
            if i + j > boat:
                break

            # 本岸羊 <= 本岸狼，说明狼运少了
            if sheep - i <= wolf - j and sheep - i != 0:
                continue

            # 对岸羊 <= 对岸狼，说明狼运多了
            if oppo_sheep + i <= oppo_wolf + j and oppo_sheep + i != 0:
                break

            # 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
            if oppo_sheep + i == 0 and oppo_wolf + j >= boat:
                break

            dfs(sheep - i, wolf - j, boat, oppo_sheep + i, oppo_wolf + j, count + 1, ans)


# 算法调用
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。