某OJ某水题：饥饿的牛

题目

描述 Description

牛在饲料槽前排好了队，饲料槽依次用 `1` 到 `N` 编号。
每天晚上，一头幸运的牛根据约翰的规则，吃其中一些槽里的饲料。
约翰提供 `B` 个区间的清单。一个区间是一对整数 `s-e` ，表示一些连续的饲料槽。
比如 `1-3, 7-8, 3-4` 等等。牛可以任意选择区间，但是牛选择的区间不能有重叠。
当然，牛希望自己能够吃得越多越好。给出一些区间，帮助这只牛找一些区间，使它能吃到最多的东西。
在上面的例子中， `1-3` 和 `3-4` 是**重叠**的；聪明的牛选择 `{1-3, 7-8}` ，这样可以吃到5个槽里的东西。

输入描述 Input Description

第一行，整数 `B`
第 `2` 到 `B + 1` 行，每行两个整数，表示一个区间，**较小的端点在前面**

输出描述 Output Description

仅一个整数，表示最多能吃到多少个槽里的食物。

样例输入 Sample Input

```
3
1 3
7 8
3 4
```

样例输出 Sample Output

```
3
```

数据范围及提示 Data Size & Hint

```
1 <= N <= 2000
1 <= B <= 1000
1 <= s <= e <=N
```

题目本质

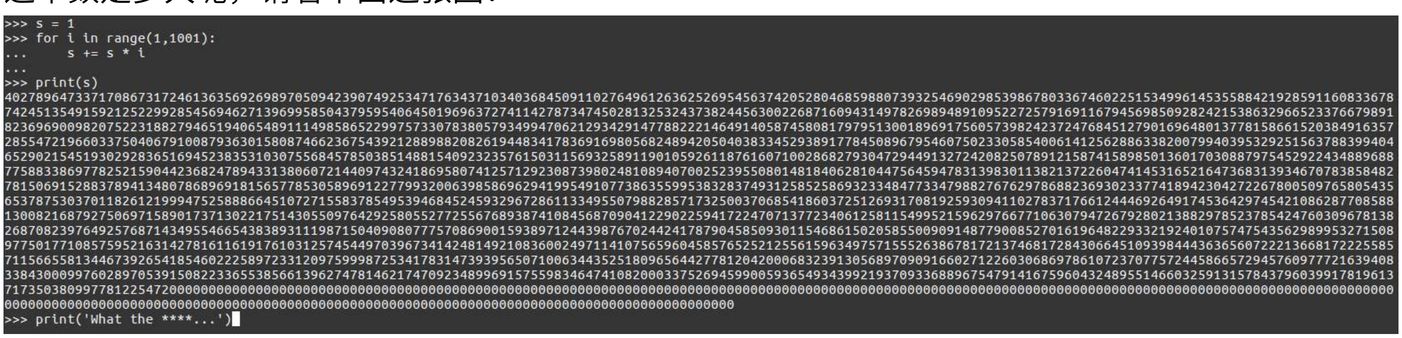
题目中说了一大堆，实际上是在讲一个这样的问题：

> *已知 `B` 个区间，每个区间是一个闭区间 `[i, j]` ，其中 `1 <= i <= j <= N`*
> *现在求一个最大的区间集合，使得集合内任意两个区间的交集为空集*
> *输出区间长度总和*

分析

暴力?

当然不可能啦，因为我们需要考虑 `1000!` 种情况。
这个数是多大呢，请看下面这张图：



所以不要考虑了…

贪心?

之前有一道活动安排的题目，和这道题很相似。

在活动安排中，按照结束时间的升序来考虑便可得到最多能安排的活动数。
但在此处是要求求出最大区间长度总和，在用贪心时处理**重合的区间**时会非常麻烦。
选择区间的后效性比较大，贪心难以胜任这个任务。
因此不考虑贪心。

动态规划！

初步分析

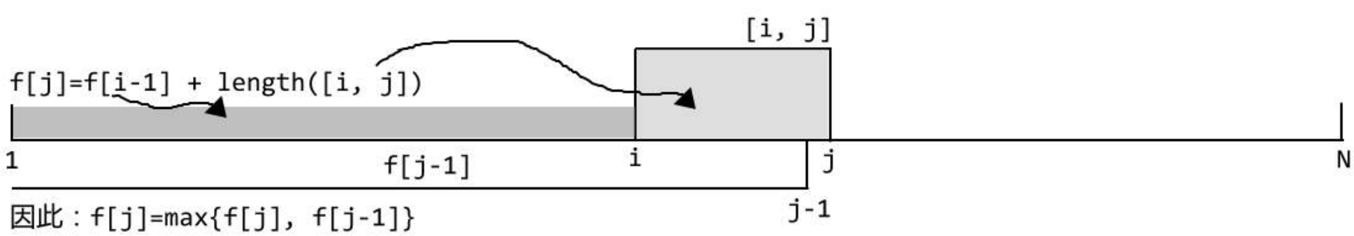
和贪心相似，在考虑动态规划时也是考虑区间结尾的。
在考虑之前，对于选不选取一个区间，有如下的性质：

> *若区间 `A` 为 `[i, j]` ，当 `N < j` 时，这个区间不能被选入*

因此，只有当 `N == j` 时，我们才考虑是否选取该区间。
此外，**可能有多个区间**的结尾是一样的。对于这种情况，只要选出一个使区间长度总和最大的即可。

状态转移

先放图：



其中： `f[j]` 表示在 `[1, j]` 内区间长度总和能达到的最大值。
显而易见，对于某一个区间只有选或不选两种状态。

1. 选择它

我们先考虑必须要选择一个区间时，如何求得最大值：

1. 首先是区间自身的长度 `length = j - i + 1` ，它是会是最大值的一部分。
2. 因为不能有区间与它重合，所以从 `f[i]` 到 `f[j - 1]` 的所有的值都是不可用的，如果使用将可能有区间与它重叠。
   所以，我们只需知道 `f[i - 1]` 的值，即在这个区间之前能达到的最大值。
   相加得到的就是最大值。

2. 不选它

这个情况很简单，只要使 `N < j` 就不会选到它。
又因为要使值尽可能最大，所以应当取 `f[j - 1]` 。

3. 综合

最后，我们得到状态转移方程即为：

```
f[j] = max{f[j - 1], f[i - 1] + length}
```

代码实现

有了状态转移方程，代码就很好写了。
只是需要注意之前在**初步分析**时的讨论到的结尾重合的情况，然后这道题就AC啦～

伪代码：

```
f = []

# 初始化f数组，长度为N + 1，默认值均为0
for i in range(0, N + 1):
    f.append(0)

for j in range(1, N + 1):
    # 找到符合条件的区间
    for interval in intervals:
        s = begin_of(interval)                  # 区间起点
        e = end_of(interval)                    # 区间结尾
        length = length_of(interval)            # 区间长度

        if e == j:                              # 结尾为j
            f[e] = max(f[e], f[s - 1] + length) # 取最大值

    f[j] = max(f[j - 1], f[j])                  # 比较两种状态，选取最大值

print(f[N])
```

提问

1. 下面的状态转移方程是否正确？请说明理由。

   ```
   f[j] = max{f[j - 1], f[j - length] + length}
   ```

2. 上面的状态转移方程使你想到哪一类动规？
3. 之前的伪代码中，每次计算 `f[j]` 时，都会将所有区间遍历一遍，找到结尾为 `j` 的区间时再做操作。这样会导致效率十分低下，其时间复杂度为 `O(NB)` 。请改进该代码使其时间复杂度变为 `O(N + B)` 。
4. 上面的伪代码只算出了区间长度总和的最大值，即最优解的值，并没有算出最优解。请改进该代码使得在计算过程中能算出最优解。