{% note info %} **摘要** Title: 1750. 救生员 Tag: 扫描线、差分、区间合并 Memory Limit: 64 MB Time Limit: 1000 ms {% endnote %}

Powered by:**NEFU AB-IN**

Link

@TOC

1750. 救生员

- 题意

> 农夫约翰为他的牛开设了一个游泳池，他认为这将帮助它们放松并产出更多的奶。 为了确保安全，他雇佣了 N 头奶牛作为救生员，每头奶牛的工作班次都是一段连续的时间。 为了简单起见，游泳池每天的开放时间从时刻 0 到时刻 1000。 每个奶牛的工作班次都可以用两个整数来描述，它们分别表示该奶牛工作班次的开始时刻和结束时刻。 例如，从时刻 t=4 开始工作并在时刻 t=7 结束工作的救生员，它的工作时间为三个时间单位（请注意，时间“段”两端的端点>是时间轴上的“点”） 不幸的是，由于资金紧张问题，约翰不得不解雇一头奶牛。 请问通过合理裁员，剩余救生员的工作班次仍然可以覆盖的最大时间有多长？ 一个时间间隔内如果存在至少一名救生员当值，那么这个时间间隔就认为是被覆盖的。

- 思路

  核心思想都是，枚举删除每个区间，看剩下区间组成的长度最大值

  - **扫描线** $O(nlogn)$ 学习地址 扫描线板子问题，即**区间最大覆盖**问题，**用线段树进行优化** 注意
    - 将时间段转化为时刻 $[l, r) -> [l, r - 1]$
    - 核心思想就是由区间加法的线段树改写来的
    - 如果空间够的话，线段树最好开8倍
    - `tr[p].len = (xs[tr[p].r + 1] - xs[tr[p].l])` 注意我们线段树中每个**叶节点**(控制区间[l,l])不是指xs[l]坐标,而是指区间[**xs[l],xs[l+1]**].线段树中其他节点控制的区间[l,r],也是指的x坐标轴的第l个区间到第r个区间的范围,也就是xs[l]到xs[r+1]坐标的范围.
    - `build(1, 0, len(xs) - 1)` 由于上面说了**叶节点表示的是区间**，所以建树可以少一个节点
    - `tr[1].len` **有效覆盖的最大长度**
  - **区间合并** $O(n^2)$ 区间合并板子题，即合并区间后，求总体长度即可
  - **差分** $O(n^2logn)$ 应该还有更优的差分做法，这里就是遍历每个点位统计长度
  - **一维扫描线** 同差分 将每个1，-1放入数组，并进行排序，枚举即可

- 代码

  - **扫描线** 1004ms

```
'''
Author: NEFU AB-IN
```

```
Date: 2022-02-08 08:59:37
FilePath: \ACM\Acwing\1752.py
LastEditTime: 2022-02-08 21:15:21
'''

#扫描线求区间最大覆盖
#学习地址：https://ncc79601.blog.luogu.org/scan-line

ls = lambda p: p << 1
rs = lambda p: p << 1 | 1


class Node(object):
    def __init__(self, l, r) -> None:
        self.l = l
        self.r = r
        self.len = 0  # 区间内被截的长度
        self.tag = 0  # 被完全覆盖的次数


N = int(1010)
tr = [Node(0, 0) for _ in range(N << 3)]


def pushup(p):
    if tr[p].tag:  #被覆盖过，这个区间是满的，覆盖长度就是这个区间的长度
        tr[p].len = (tr[p].r - tr[p].l + 1)  #更新长度
    else:  #否则
        tr[p].len = tr[ls(p)].len + tr[rs(p)].len  #合并儿子信息


def build(p, l, r):
    tr[p] = Node(l, r)
    if l == r:
        return
    mid = l + r >> 1
    build(ls(p), l, mid)
    build(rs(p), mid + 1, r)
    pushup(p)


#l, r 是固定的，二分的永远是tr[p].l和tr[p].r
def update(p, l, r, d):
    if l <= tr[p].l and tr[p].r <= r:
        tr[p].tag += d
        pushup(p)
        return
    #pushdown(p)
    mid = tr[p].l + tr[p].r >> 1
    if l <= mid:
        update(ls(p), l, r, d)
    if mid < r:
        update(rs(p), l, r, d)
    pushup(p)
```

```
lst = []

if __name__ == "__main__":
    n = int(input())
    build(1, 0, N)
    for i in range(n):
        lst.append(list(map(int, input().split())))
        lst[i][1] -= 1  #时间段转化为时刻
        update(1, lst[i][0], lst[i][1], 1)

    res = 0
    for i in range(n):
        update(1, lst[i][0], lst[i][1], -1)
        res = max(res, tr[1].len)
        update(1, lst[i][0], lst[i][1], 1)
    print(res)
```

- **扫描线+离散化** 985ms

```
'''
Author: NEFU AB-IN
Date: 2022-02-08 08:59:37
FilePath: \ACM\Acwing\1752.py
LastEditTime: 2022-02-09 11:29:37
'''

#扫描线求区间最大覆盖
#学习地址: https://ncc79601.blog.luogu.org/scan-line

ls = lambda p: p << 1
rs = lambda p: p << 1 | 1


class Node(object):
    def __init__(self, l, r) -> None:
        self.l = l
        self.r = r
        self.len = 0  # 区间内被截的长度
        self.tag = 0  # 被完全覆盖的次数


N = int(1010)
tr = [Node(0, 0) for _ in range(N << 3)]
lst = []
xs = []
L = [0 for _ in range(N)]
R = [0 for _ in range(N)]


def find(x):
```

```
    l = 0
    r = len(xs) - 1
    while l < r:
        mid = l + r >> 1
        if xs[mid] >= x:
            r = mid
        else:
            l = mid + 1
    return r


def pushup(p):
    if tr[p].tag:  #被覆盖过，这个区间是满的，覆盖长度就是这个区间的长度
        tr[p].len = (xs[tr[p].r + 1] - xs[tr[p].l])  #更新长度
    else:  #否则
        tr[p].len = tr[ls(p)].len + tr[rs(p)].len  #合并儿子信息


def build(p, l, r):
    tr[p] = Node(l, r)
    if l == r:
        return
    mid = l + r >> 1
    build(ls(p), l, mid)
    build(rs(p), mid + 1, r)
    pushup(p)


#l, r 是固定的，二分的永远是tr[p].l和tr[p].r
def update(p, l, r, d):
    if l <= tr[p].l and tr[p].r <= r:
        tr[p].tag += d
        pushup(p)
        return
    #pushdown(p)
    mid = tr[p].l + tr[p].r >> 1
    if l <= mid:
        update(ls(p), l, r, d)
    if mid < r:
        update(rs(p), l, r, d)
    pushup(p)


if __name__ == "__main__":
    n = int(input())
    for i in range(n):
        l, r = map(int, input().split())
        xs.append(l)
        xs.append(r)
        lst.append([l, r])
    xs = list(set(xs))
    xs.sort()
    build(1, 0, len(xs) - 1)
    for i in range(n):
```

```
            L[i] = find(lst[i][0])
            R[i] = find(lst[i][1])
            update(1, L[i], R[i] - 1, 1)
        res = 0
        for i in range(n):
            update(1, L[i], R[i] - 1, -1)
            res = max(res, tr[1].len)
            update(1, L[i], R[i] - 1, 1)
        print(res)
```

- **区间合并** 1035ms

```
'''
Author: NEFU AB-IN
Date: 2022-02-08 16:10:23
FilePath: \ACM\Acwing\1752.1.py
LastEditTime: 2022-02-09 09:22:21
'''

lst = []

if __name__ == "__main__":
    n = int(input())
    for i in range(n):
        lst.append(list(map(int, input().split())))
    lst.sort() #先按左端点进行排序
    res = 0
    for i in range(n):  #枚举删哪个数
        cnt = 0
        st, ed = -1, -1  #设置维护的区间的起始和终止
        for j in range(n):
            if i != j:
                if ed < lst[j][0]:  #当维护的区间终点比枚举的区间起点小时，
说明这个区间已经不会有别的区间和它交集了
                    #那么就记录区间，并设置新的起始和终止
                    if st != -1 and ed != -1:
                        cnt += ed - st
                    st = lst[j][0]
                    ed = lst[j][1]
                else:
                    #否则更新区间终止点的最大值
                    ed = max(ed, lst[j][1])
        #别忘了记录最后一个区间的
        if st != -1 and ed != -1:
            cnt += ed - st
        res = max(res, cnt)
    print(res)
```

- **差分** 1108ms

```python
'''
Author: NEFU AB-IN
Date: 2022-02-09 10:05:28
FilePath: \ACM\Acwing\1752.2.py
LastEditTime: 2022-02-09 10:22:02
'''
from collections import Counter

d = Counter()
a = []

if __name__ == "__main__":
    n = int(input())
    for i in range(n):
        l, r = map(int, input().split())
        d[l] += 1
        d[r] -= 1
        a.append((l, r))
    res = 0
    for i in range(n):
        d[a[i][0]] -= 1
        d[a[i][1]] += 1
        last, cnt, sum = 0, 0, 0
        for x in sorted(d.keys()):
            if cnt >= 1:
                sum += (x - last)
            cnt += d[x]
            last = x
        res = max(sum, res)
        d[a[i][0]] += 1
        d[a[i][1]] -= 1
    print(res)
```