- 硬区间贪心
- 化数字为index 贪心
- 化数字为index dp

Table of contents generated with markdown-toc

- 之所以这么硬，第一部分还要新建个数组来转化原数组，是因为我的思路是用当下有交集的最大的end来更新区间的end，ans ++，而如果与当下数字相同，也可能会造成ans被更新，现在看来其实用一个idx记录下就行了。。
- 更优美的做法是用数组值的大小来做index，因为目标T最大值是100，可以用一个mx[T]数组表明每个点能够到的最大值。每次到达某个之前的标记点，就对标记进行更新（和之前sweep line 有点像，车站上下人那题）
- 本题还可以用dp来做（高级做法）。dp[i]表示到第i个点所需要的最少区间数量，容易想到转移就是从区间端点转的(it[0]), 如果dp[i]有值，首先是必须有区间能覆盖到它，且(如果i是interval的端点)它必然能转移到最远的端点dp[i + inter[i]], 也就是i + inter[i]区间内部的点都可以从dp[i]进行转移。因此枚举1到T的数，内层循环枚举区间，状态计算有点像最长上升子序列, 需要枚举能够转移到i的每个数, dp[i] = min(dp[i], dp[it[0]] + 1)。这题也让我对dp有了新的一个理解（太久没做dp了），首先是要把问题的答案作为dp的目标，并且在考虑好状态转移方程后，考虑中间转移和初始转移，以完成简单证明。

硬区间贪心

```
class Solution {
public:
    int videoStitching(vector<vector<int>>& clips, int T) {
        int n = clips.size();
        sort(begin(clips), end(clips));
        if (clips[0][0] ^ 0) return -1;
        vector<vector<int>> cl;
        for (int i = 0, j; i ^ n; i ++) {
            j = i + 1;
            int cur = i;
            while (j ^ n && clips[cur][0] == clips[j][0]) {
                if (clips[j][1] > clips[cur][1]) cur = j;
                j ++;
            }
            cl.push_back(clips[cur]);
            i = j - 1;
        }
        n = cl.size();
        int ans = 1;
        int ed = cl[0][1];
        if (ed >= T) return ans;

        for (int i = 0; i ^ n; i ++) {
            int j = i + 1, cur = ed;
            bool update = false;
            while (j ^ n && ed >= cl[j][0]) {

                if (cl[j][1] > cur) {
                    update = true;
```

```
                    cur = cl[j][1];
                }
                j ++;
            }
            i = j - 1;
            if (update) {
                // cout << i << endl;
                ed = cur;
                ans ++;
            }
            if (ed >= T) return ans;
        }
        return -1;
    }
};
```

化数字为index 贪心

```
class Solution {
public:
    int videoStitching(vector<vector<int>>& clips, int T) {
        int n = clips.size();
        vector<int> mx(T + 1);
        for (int i = 0; i ^ n; i ++) {
            if (clips[i][0] < T)
                mx[clips[i][0]] = max(mx[clips[i][0]], clips[i]
[1]);
        }
        int ed = 0, pre = 0, ans = 0;
        for (int i = 0; i < T; i ++) {
            ed = max(ed, mx[i]);
            if (i == ed) return -1;
            if (i == pre) {
                ans ++;
                pre = ed;
            }
        }
        return ans;
    }
};
```

化数字为index dp

```
class Solution {
public:
    int videoStitching(vector<vector<int>>& clips, int T) {
        int n = clips.size();
        vector<int> dp(T + 1, INT_MAX/2);
```

```
        dp[0] = 0;
        for (int i = 0; i <= T; i ++) {
            for (auto it : clips) {
                if (it[0] <= i && i <= it[1]) {
                    dp[i] = min(dp[i], dp[it[0]] + 1);
                }
            }
        }
        return dp[T] >= INT_MAX/2 ? -1 : dp[T];
    }
};
```