segment

我们被要求走完所有线段，而且只能从上往下走，也就是说前往第 $i+1$ 行时，第 i 行的线段必须已经走完。

而一个线段恰好被走完的时候，我们要么处于线段的左端点，要么处于线段的右端点。

因此设 $f(i, 0/1)$ 表达已经走完第 i 行的线段，现在处于左端点/右端点。

现在考虑怎么转移。

需要注意到一件事情，如果走完线段时在左端点，那么必然是从右端点走到左端点的，不论刚抵达这一行时在什么位置。

也就是说，走完线段时在左端点，一定是从到达第 i 行的位置先走到右端点处，再走到左端点处，反之类似。

```
inline int calc0(int x, int l, int r){// 走完线段时在 l 位置
    return dis(x, r) + dis(r, l);
}
inline int calc1(int x, int l, int r){// 走完线段时在 r 位置
    return dis(x, l) + dis(l, r);
}
```

转移时，考虑走完上一行的线段后是在左端点还是右端点，然后直接走到这一行，再考虑走完这一行的线段后是在左端点还是右端点。

```
f[i][0] = min(f[i - 1][0] + calc0(l[i - 1], l[i], r[i]), f[i - 1][1] + calc0(r[i - 1], l[i], r[i])) + 1;
f[i][1] = min(f[i - 1][0] + calc1(l[i - 1], l[i], r[i]), f[i - 1][1] + calc1(r[i - 1], l[i], r[i])) + 1;
```

最终答案为

```
min(f[n][0] + dis(l[n], n), f[n][1] + dis(r[n], n))
```

时间复杂度 $O(n)$。

cards

设 $f(i)$ 表示已经放了前 i 张牌的最大得分，$s(i)$ 表示前 i 张牌的分数之和。

如果放了第 i 张牌之后什么都不做，那么直接继承 $f(i-1)$。

否则考虑找到之前某一张同色的 j，将这一段取出来，那么就是 $f(j-1)+s(i)-s(j-1)$。

为什么不需要考虑 $[j, i]$ 之间的牌是否在之前的操作中已经被取出来的呢。

假设之前某一次取出来的牌是 $[x, y]$。

- $x<j<y<i$：两个操作不能共存，所以当前转移不可能拿到取 $[x, y]$ 的得分。
- $x<y<j<i$：$[x, y]$ 在 $f(j-1)$ 内已经考虑。
- $j<x<y<i$：那么这一段牌在取出 $[j, i]$ 时也正常计入了，不用特别考虑 $[x, y]$。

```
for(int i = 1; i <= n; i++){
    f[i] = f[i - 1];
    for(int j = 1; j < i; j++){
        if(c[j] == c[i])
            f[i] = max(f[i], f[j - 1] + s[i] - s[j - 1]);
    }
}
```

但是现在 $O(n^2)$ 的时间复杂度显然无法通过此题。

观察转移式 $f(j-1)+s(i)-s(j-1)$，它可以分成两部分，一部分只和 i 有关：$s(i)$，一部分只和 j 有关：$f(j-1)-s(j-1)$。

和 i 相关的部分处理起来很简单，而和 j 相关的部分，我们进行一个前缀和优化，对于每种颜色，存下之前最优的 j 即可。

```
for(int i = 1; i <= n; i++){
    f[i] = max(f[i - 1], g[c[i]] + s[i]);
    g[c[i]] = max(g[c[i]], f[i - 1] - s[i - 1]);
}
```

一些细节：g 的初值应该为 $-\infty$；颜色 c_i 可能需要离散化。

dp 部分的时间复杂度为 $O(n)$。

deskmate

注意，下文中，x,y 能坐同桌都不再强调两人需要为朋友关系，请读者在自行实现过程中记得判断。

因为只有队列中相邻的两人才可以成为同桌，所以 x,y 想要成为同桌，必须 $[x+1,y-1]$ 内的同学已经按照某种方案出列成为同桌了。

所以子问题是一个比较明显的区间的形式，不妨设 $f(l,r)$ 表示 $[l,r]$ 内的同学内部匹配为同桌的方案数。

怎么转移呢？不妨考虑 l 的同桌是谁。

若 l 的同桌是 r，则规约到子问题 $f(l+1,r-1)$。

若 l 的同桌是中间的某个同学 k，则分成了 $f(l+1,k-1),f(k+1,r)$ 两个子问题，注意分出的区间长度必须为偶数。

但在这种情况时，直接将 $f(l+1,k-1),f(k+1,r)$ 相乘时不够的。因为离开队列的顺序不同算是不同的方案。

这是两个独立的区间，我们完全可以先在左边选一对同桌，然后又在右边选一对同桌。

不过两个区间内部的先后顺序是已经确定的，因此只需要考虑两个序列组合的方案数。

记 $A=(k-l+1)/2,B=(r-k)/2$，两个序列组合的方案数即为组合数 $\binom{A+B}{B}$。

$$f(l,r)=f(l+1,r-1)+\sum_k f(l+1,k-1)*f(k+1,r)*\binom{A+B}{B}$$

时间复杂度 $O(n^3)$。

puzzle

不难发现答案具有单调性，当拼图越大的时候，能到达的位置一定更多，所以考虑二分答案。

那么在 check 的过程中，就可以认为拼图的大小是已知且固定的了。

显然在走一条简单路径的情况下，离开一个拼图之后，就不会再回来。

也就是说，把一个拼图看作若干个从 $(0,0)$ 移动到 (x_i,y_i) 的向量集合，一条简单路径只会从中选择一个向量。

而对于向量 (x_i,y_i) 和向量 (x_j,y_j)，不论先走哪一个，最终都是到达 (x_i+x_j,y_i+y_j)，因此拼图的摆放顺序没有不影响答案。

先考虑暴力 dp 的做法。

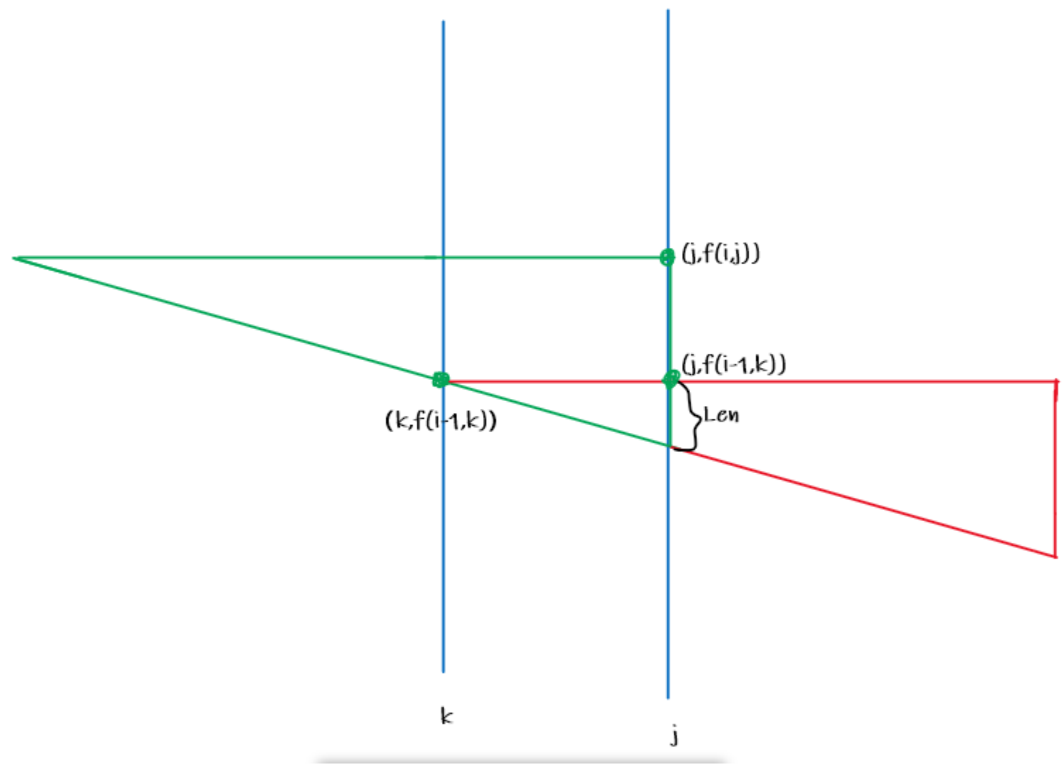
设 $f(i,x,y)$ 为前 i 个三角形能不能到达 (x,y)。

时间复杂度为 $O(nm^4\log V)$。

其中 V 为二分的上界，在题目中为 $2m\times\max\{a_i,b_i\}$，即摆放一个三角形就可以从 $(1,1)$ 到 (m,m) 的情况。

考虑优化，将一维状态存入 dp 值中，设 $f(i,j)$ 为前 i 块拼图能够到达的 (j,k) 中 k 的最大值。

因为如果摆放完一些拼图能够到达 (i,j)，那么将这些拼图换种方式摆放，比如直接往上移动，一定可以到达所有 $k\le j$ 的 (i,k)。



考虑怎么转移，对于确定的 $(j,f(i,j))$，我们的最优决策一定是把三角形的之间顶点放在这个位置。

上图中，三角形由红色部分移至绿色部分，之前连通两点间始终连通，但到达的范围扩大。

扩大 mid 后，三角形的水平直角边长为 $\dfrac{mid}{a_i}$，垂直直角边长为 $\dfrac{mid}{b_i}$。

则最远只能从第 $j-\dfrac{mid}{a_i}$ 列转移过来，使用相似三角形计算新增可到达距离 d。

由 $\frac{len}{j-k}=\frac{\frac{mid}{b_i}}{\frac{mid}{a_i}}=\frac{a_i}{b_i}$ 得 $len=\frac{a_i(j-k)}{b_i}$

而 $d+len=\frac{mid}{b_i}$，故 $d=\frac{mid-a_i(j-k)}{b_i}$

因此转移方程为

$$f(i,j)=\max_{\max(0,j-\lfloor\frac{mid}{a_i}\rfloor)\le k\le j}\{f(i-1,k)+\lfloor\frac{mid-a_i(j-k)}{b_i}\rfloor\}$$

时间复杂度降为 $O(nm^2\log V)$。