C++

day5 SH-B260105 集训-讲解

T1 星际探险

模拟算法

输入每个地点的初始高度a[i]

对于每个问询

对地点[l,r]高度进行修改

从0号点出发，计算i号点与i-1号点海拔差d

如果d大于0　减少d*t的电量

如果d小于0　累加(-d)*t的电量

时间复杂度?

暴力

```
3  typedef long long ll;
4  const int N=200009;
5  ll n,q,s,t,ans,a[N],sum;
6  int main(){
7      cin>>n>>q>>s>>t;
8      for(int i=0;i<=n;i++) cin>>a[i];
9      for(int u=1;u<=q;u++){
10         int l,r,x;
11         cin>>l>>r>>x;
12         for(int j=l;j<=r;j++) a[j]+=x;
13         sum=0;
14         for(int i=1;i<=n;i++){
15             int d=a[i]-a[i-1];
16             if(d<0) sum-=t*d;
17             else sum-=s*d;
18         }
19         cout<<sum<<endl;
20     }
21     return 0;
22 }
```

时间复杂度?	O(nq)

如何想到加速算法

区间更新

只标记两端

重大发现：区间内的相对海拔是不变的因此电量也不会变。只有区间的边界可能受到影响

差分

```
6  ll change(ll x) {
7      if (x>=0) return -x*s;
8      else return -x*t;
9  }
```

change(x)表示高度差为h时对应电量变化量

```
13     ll sum=0;
14     for(int i=1;i<=n;i++){
15         d[i]=a[i]-a[i-1];
16         sum+=change(d[i]);
17     }
18     for(int u=1;u<=q;u++){
19         int l,r,x;
20         cin>>l>>r>>x;
21         sum-=change(d[l]);
22         d[l]+=x;
23         sum+=change(d[l]);
24         if(r<n) {
25             sum-=change(d[r+1]);
26             d[r+1]-=x;
27             sum+=change(d[r+1]);
28         }
29         cout<<sum<<endl;
30     }
```

对左端进行电量修正

对右端进行电量修正

T2 删除子数组

题意

给定数组，求删除最短的子数组，使得剩余元素的和可以被x整除

数据规模:

20%数据， n<=100

50%数据， n<=1000

100%数据，n<=100000

数组中每个元素<10^9、x<10^9

暴力解法

1.如果数组元素和 mod x=0,返回0

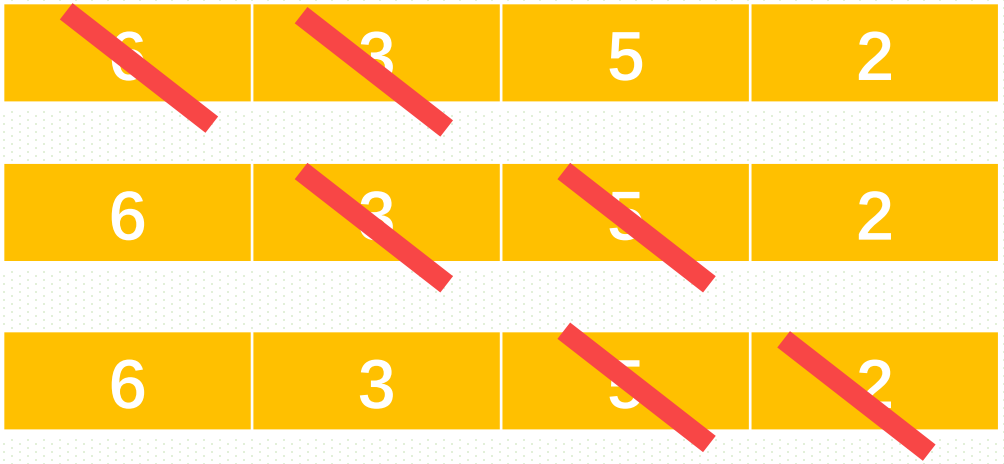
2.从小到大枚举删除子数组的长度

3.从左往右枚举该长度下的所　有子数组,

如果删除后剩余元素和可以整除x,返回该长度。

余数

枚举长度为2的子数组



答案为2

1.为了防止int溢出，在计算过程中，每次加法的结果都需要和x取余数。

2.计算删除子数组和时，可以使用前缀和做差计算连续和。

时间复杂度为多少?

暴力解法

1、枚举删除不同长度的子数组区间
（长度依次从小到大）

2、当删除的子数组区间为[i,j]时，
剩余元素的和为：
[1,i-1]的和 + [j+1,n]的和

预计算

[1,i-1]为前缀和

[j+1,n]为后缀和

3、**第一次**找到符合条件的子数组
即可以结束。，

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=100009;
ll n,x,a[N],s[N],g[N];
int main()
{
    freopen("subarray.in","r",stdin);
    freopen("subarray.out","w",stdout);
    cin>>n>>x;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        s[i]=s[i-1]+a[i];
    }
    if(s[n]%x==0) {cout<<0<<endl; return 0;}
    g[n]=a[n];
    for(int i=n-1;i>=0;i--) g[i]=g[i+1]+a[i];
    for(int k=1;k<=n-1;k++){
        for(int i=1;i<=n-k+1;i++){
            if((s[i-1]+g[i+k])%x==0) {
                cout<<k<<endl; return 0;
            }
        }
    }
    cout<<-1<<endl;
    return 0;
}
```

优化思路

问题描述

删除一个子数组后剩余元素之和能被 x 整除，

等价于**删除的子数组之和模 x 等于 整个数组和模 x**。

设整个数组和模 x为 **total_mod**

total_mod=0
不需要删除任何元素

目标

找到一个连续子数组，其元素之和模 x等于 **total_mod** ，

且该子数组不能是整个数组。 目标是使该**子数组长度最短**。

思考如何找到符合要求的子区间？　　如何确定最小长度？

若该区间满足要求

i-j 为符合要求的子数组长度

则 (s[i]-s[j]) % X = total_mod

(s[i] % X) - (s[j] % X) = total_mod

s[j] % X = (s[i] % X) - total_mod

s[j] % X = (s[i] - total_mod) % X

可以检查每一个前缀和减去total_mod对X取模的结果 在之前有没有出现过?

如果出现过，则该区间为满足要求的区间。

计算过程

1、依次计算前缀和 s[i]

2、计算 (s[i] – total_mod) % X = cur_mod

3、检查之前有没有出现过和cur_mod的结果

4、记录当前cur_mod的结果和出现位置i

供后面的(s[i] - total_mod) % X 配对使用。

思考：如何记录已有的 cur_mod 和 位置？

使用map<int,int> m 存储 (hash表)

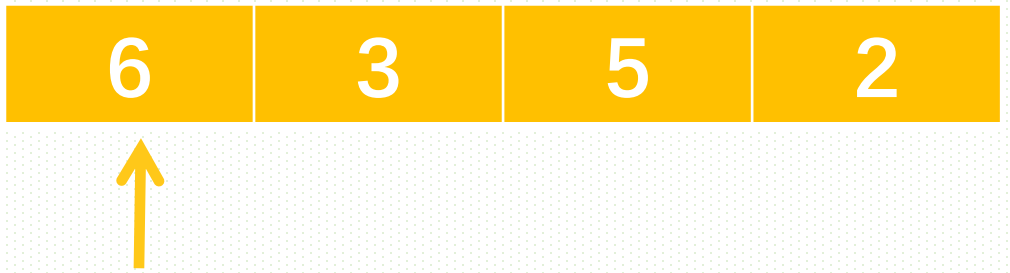
注意：减法取模，可能会出现负数

(s[i] - total_mod + X) % X

优化解法：前缀和+Hash

total_mod=7

当x=9时



cur_mod = 6

```
map<int,int> m ;
```

映射

也是hash表

target_mod =

cur_mod - total_mod + x

=8

查表，无

```
m[0] = 0
m[6] = 1
```

total_mod=7

当x=9时

cur_mod = 0

target_mod =

cur_mod - total_mod + x

=2

查表，无

```
m[0] = 2
m[6] = 1
```

total_mod=7

当x=9时

cur_mod=5

target_mod =

cur_mod - total_mod + x

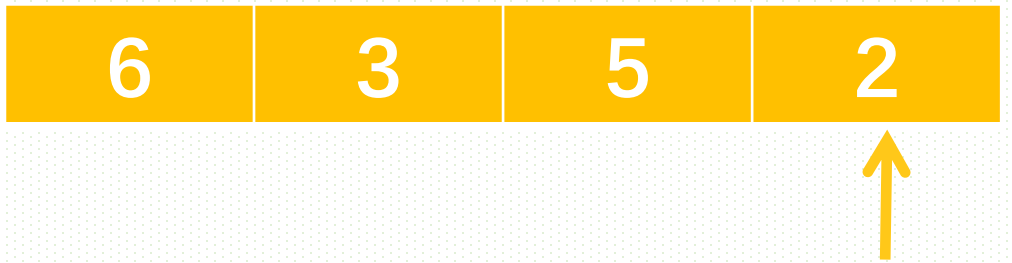
=7

查表，无

```
m[0] = 2
m[6] = 1
m[5] = 3
```

total_mod=7

当x=9时



cur_mod = 7

target_mod =

cur_mod - total_mod

=0

查表

```
m[0] = 2
m[5] = 3
m[6] = 1
```

```
m[0] = 1
```

最短子数组长度为

4-2 = 2

```
 9      cin >> N >> x;
10      int total_mod = 0;
11      for(int i=1; i<=N; i++){
12          cin >> A[i];
13          total_mod = (                    ) % x;  计算总数组和模x的结果
14      }
15      if(total_mod == 0){
16          cout <<   << endl;
17          return 0;
18      }
```

```
19      int ans = N;
20      int cur_mod = 0;   记录当前前缀和模x的结果
21      m[0] = ;   初始化：前缀和模x为0出现在索引0（虚拟位置）
22      for(int i=1; i<=N; i++){
23          cur_mod = (                ) % x;   计算当前前缀和模x的结果
24          int target_mod=(                )%x;   计算需要寻找之前某个前缀模值target_mod
25  查hash表  if(                ) 如果target_mod出现过
26              if ((i-m[target_mod])<ans)
27                  ans = i-m[target_mod];
28          m[cur_mod] = ;   记录当前target_mod出现的位置
29      }
30      if(          )
31          cout << ans <<endl;
32      else
33          cout << -1;
34      return 0;
35  }
```

T3 轮流值班

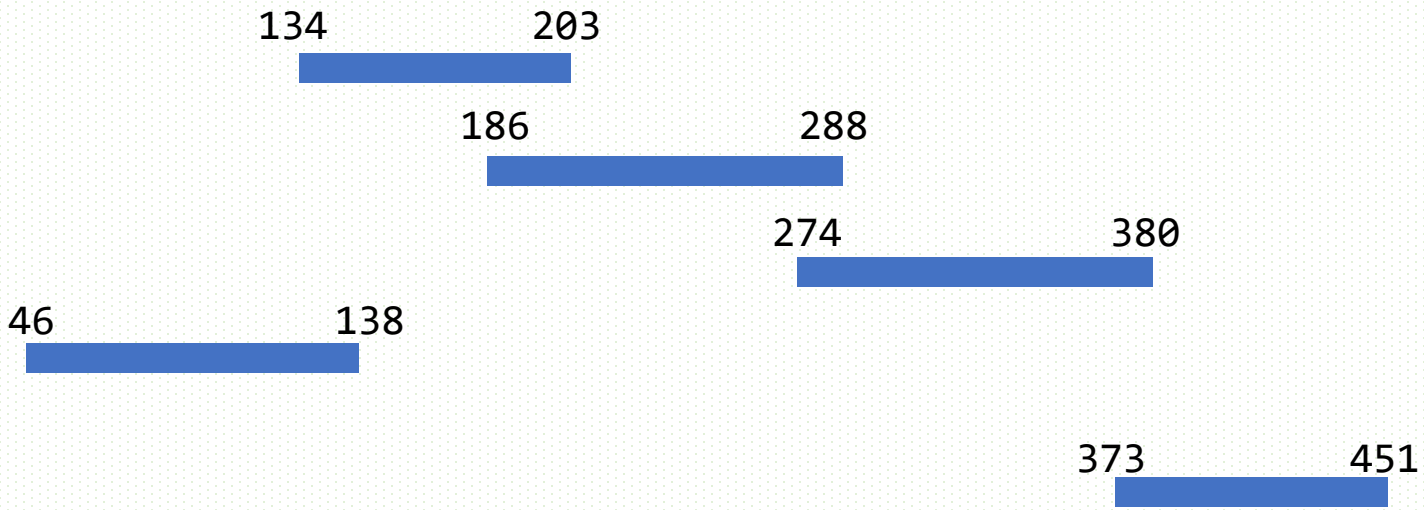
题意
突出核心要点

一维坐标轴上有N条线段

选择其中N-1条

求最多能覆盖的长度

输入
5
134 203
186 288
274 380
46 138
373 451



输出
357

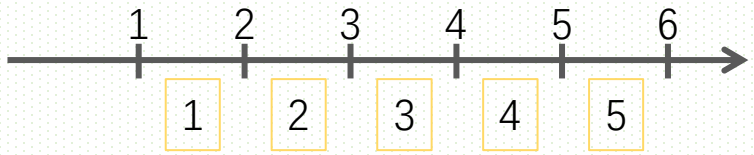
(138-46)+(451-186)=357

请同学分析数据范围

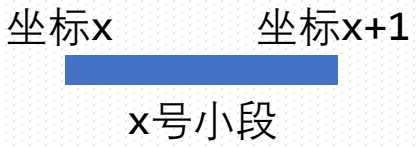
【数据规模与约定】 测试数据共10组
1号数据：n=3
2号数据：n=5
3号数据：n<=10, 0<=a_i<=b_i<=1000
4号数据：n<=100, 0<=a_i<=b_i<=1000
所有数据：2<=n<=100000,
0<=a_i<=b_i<=1000000000

易错点

编号的两种含义
坐标刻度和单位小段



坐标点对应右侧短边



识别无效区间

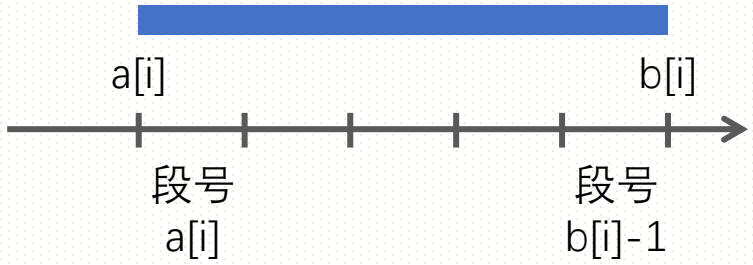
若某个区间A被别的区间B包含
则区间A可以删除
而不影响覆盖长度

请描述“识别无效区间”的算法步骤

```
78  int main(){
79      freopen("duty.in","r",stdin);
80      freopen("duty.out","w",stdout);
81      input();
82      if(n<=100&&rightmost<=1000)
83          solveBF();
84      else
85          solve();
86      return 0;
87  }
```

差分标记两端

tag[p]表示p号小段比p-1号小段多覆盖了几层




```
14  for(int i=1;i<=n;++i){
15      tag[a[i]]++;
16      tag[b[i]]--;
17  }
```

```
11  const int R=1009;
12  int tag[R];
13  void solveBF(){
14      for(int i=1;i<=n;++i){
15          tag[a[i]]++;
16          tag[b[i]]--;
17      }
18      int ans=0;
19      for(int i=1;i<=n;++i){
20          tag[a[i]]--;
21          tag[b[i]]++;
22          int now=0,cnt=0;
23          for(int j=0;j<=rightmost;++j){
24              now+=tag[j];
25              if(now>=1) cnt++;
26          }
27          ans=max(ans,cnt);
28
29
30      }
31      cout<<ans<<endl;
32  }
```

删除差分标记

补回差分标记

50分

复杂度分析

时间复杂度O(n*R)

空间复杂度O(n+R)

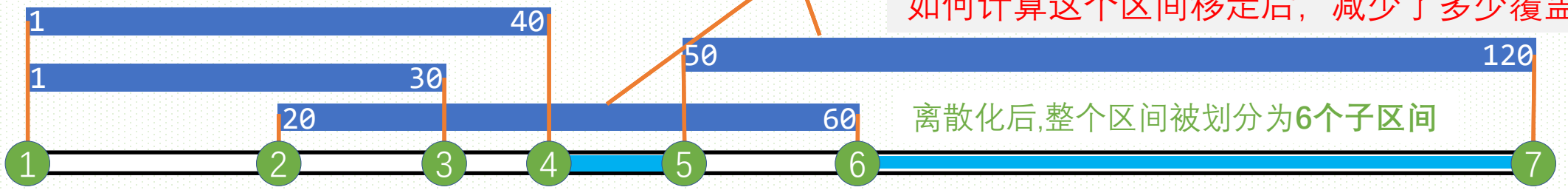
思考如何优化?

空间优化

离散化/坐标压缩

只有这两个区间**单独**移除后影响覆盖范围

如何计算这个区间移走后，减少了多少覆盖长度?

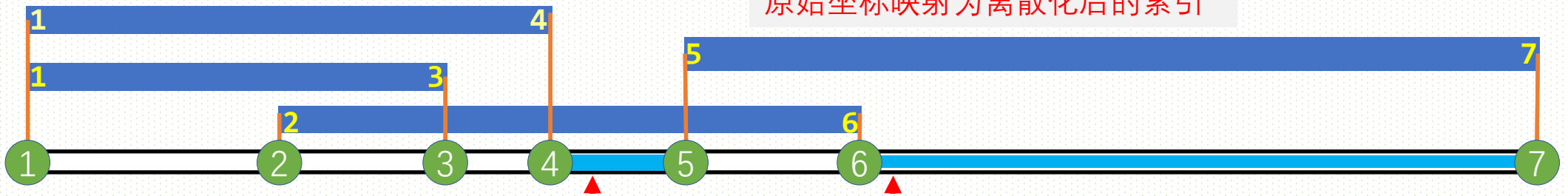


离散化后,整个区间被划分为**6个子区间**

坐标压缩
离散化

问题转化为：离散化后，找到只被一个区间覆盖的“子区间”

原始坐标映射为离散化后的索引



w[1]=19 w[2]=10 w[3]=10 w[4]=10 w[5]=10 w[6]=10

计算离散化后每段的实际宽度

差分 d[1]=2 d[2]=1 d[3]=-1 d[4]=-1 d[5]=1 d[6]=-1 d[7]=-1

计算每个子区间的覆盖次数(差分)

前缀和 cv[1]=2 cv[2]=3 cv[3]=2 cv[4]=1 cv[5]=2 cv[6]=1

只计算所有受影响的子区间宽度的前缀和

枚举每一个原始区间被移走后，计算对应的子区间受影响的宽度。

空间优化

离散化/坐标压缩

```
38 void solve(){
39     for(int i=1;i<=n;++i){
40         xs[i]=a[i];
41         xs[i+n]=b[i];
42     }
43     sort(xs+1,xs+1+n*2);
44     int nUnq=unique(xs+1,xs+1+n*2)-(xs+1);
45     for(int i=1;i<=n;++i){
46         a[i]=lower_bound(xs+1,xs+1+nUnq,a[i])-xs;
47         b[i]=lower_bound(xs+1,xs+1+nUnq,b[i])-xs;
48     }
49     for(int p=1;p<=nUnq-1;++p)
50         width[p]=                ;
```

排序

unique()
*去除容器或数组中**相邻的重复**元素。*
它不会真正删除元素，而是将重复的元素移动到容器的末尾，
并返回去重后的尾地址

原始坐标映射为离散化后的索引

计算离散化后每段的实际宽度

从差分标记复原覆盖情况

d[p]表示p号小段比p-1号小段多覆盖了几层

cv[p]表示p号段被覆盖的次数

```
51      for(int i=1;i<=n;++i){
52          d[a[i]]++;
53          d[b[i]]--;
54      }
55      for(int p=1;p<=nUnq-1;++p)
56          cv[p]=
```

差分

差分前缀和

计算每个子区间的覆盖次数

计算总覆盖长度len

```
57      int len=0;
58      for(int p=1;p<=nUnq-1;++p)
59          if(cv[p]>0) len+=
```

观察每次变化的影响

删除[a,b]区间时
该区间内原本恰好覆盖1次
的小段会变成覆盖0次

需要快速计算[a,b]区间
原本恰好覆盖1次的小段有几段

预计算前缀区间[1,p]
原本恰好覆盖1次的小段有几段

前缀区间内
恰好覆盖1次的长度

```
60      for(int p=1;p<=nUnq-1;++p){
61          if(          )   只被覆盖一次的子区间
62              s[p]=                累加该段的长度
63          else
64              s[p]=s[p-1];  多次覆盖的不纳入计算
65      }                     因为移除一个区间后，不影响该子区间被覆盖
```

a[i] 和 b[i] 为离散化后的索引

枚举删除区间编号i

```
66      int ans=0;
67      for(int i=1;i<=n;i++){
68          int l=a[i];          离散化后的区间左端点
69          int r=          ;    离散化后的区间右端点
70          int rmv=          ;  删除该区间后，减少的贡献值
71          ans=max(ans,          );  打擂台，保留最大的覆盖长度
72      }
73      cout<<ans<<endl;
```

复杂度分析

O(nlogn+n)

查错方法

打印关键变量

```
//  cout<<"len="<<len<<endl;
//  for(int p=1;p<=nUnq;++p)cout<<"xs["<<p<<"]="<<xs[p]<<endl;
//  for(int p=1;p<=nUnq-1;++p)cout<<"width["<<p<<"]="<<width[p]<<endl;
//  for(int p=1;p<=nUnq-1;++p)cout<<"cv["<<p<<"]="<<cv[p]<<endl;
//  for(int p=1;p<=nUnq-1;++p)cout<<"s["<<p<<"]="<<s[p]<<endl;
```

T4 堵车

题目大意

已知无向正权图中包含n个节点和m条边。某条边的边权翻倍后，最差情况下，从1号节点到n号节点的最短路**增量**最大是多少？

部分分策略讨论

	测试点编号	特殊性质
特殊情况	1~2	m=n-1，Ai=i，Bi=i+1，符合链条形态
复杂度很差的解	3~4	n⩽10
复杂度较差的解	5~6	n⩽100
正解	7~10	无

数据分流

```
75  int main(){
76      freopen("congestion.in","r",stdin);
77      freopen("congestion.out","w",stdout);
78      input();
79      if(isChain())
80          solveChain();
81      else
82          solve();
83      return 0;
84  }
```

```
 6  const int N=509;
 7  const int INF=1e9;
 8  int G[N][N];
 9  int n,m;
10  void input(){
11      cin>>n>>m;
12      for(int u=1;u<=n;++u)
13          for(int v=1;v<=n;++v)
14              G[u][v]=INF;
15      for(int i=1;i<=m;++i){
16          int u,v,w;
17          cin>>u>>v>>w;
18          G[u][v]=G[v][u]=w;
19      }
20  }
```

链条形态

找到最大边权

```
21 bool isChain(){
22     if(m!=n-1) return 0;
23     for(int i=1;i<=m;++i)
24         if(                    ) return 0;
25     return 1;
26 }
27 void solveChain(){
28     int mx=0;
29     for(int i=1;i<=m;++i)
30         mx=max(                    );
31     cout<<mx<<endl;
32 }
```

不符合链状形态

不符合链状形态

复杂度较差的解

枚举法

枚举m条边中的每一条边

对该条边的边权翻倍

再求从1号到n号的最短路

最短路采用朴素版Dijkstra，复杂度$O(n^2)$

整体复杂度$O(mn^2)$

$n \le 500$，$m \le 100000$

思考优化方法

若修改的边不在原始最短路上

对最短路没有影响

枚举法+优化

只枚举原始最短路上的每一条边

对该条边的边权翻倍

再求从1号到n号的最短路

最短路采用朴素版Dijkstra，复杂度$O(n^2)$

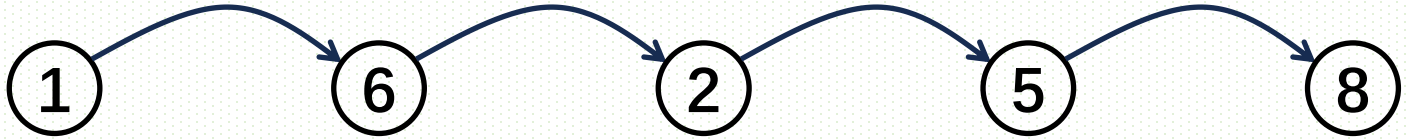
整体复杂度$O(n^3)$

$n \le 500$，$m \le 100000$

如何记录：原始最短路上的每一条边

pre[v]表示从1号到v号的最短路上

v号节点的前驱节点



pre[1]=0　pre[6]=5　pre[2]=6　pre[5]=2　pre[8]=5

思考pre[v]数组的数据怎么维护的?

```
58 void solve(){
59     Dijkstra();
60     int ansOld=d[n];
61     int ansNew=    ;
62     int v=n;
63     while(        ){
64         int u=pre[v];
65         G[u][v]*=2;
66         G[v][u]*=2;
67         Dijkstra();
68         ansNew=max(ansNew,d[n]);
69         
70         
71         
72     }
73     cout<<ansNew-ansOld<<endl;
74 }
```

ansNew的作用? 初始值?

枚举最短路上每一条边，思考循环条件?

阅读后续代码
分析循环条件

每次取v的前驱结点

边权翻倍

边权恢复

Dijkstra朴素版 + pre[]

```
33  int d[N];
34  bool ok[N];
35  int pre[N];
36  void Dijkstra(){
37      fill(d,d+n+2,INF);
38      fill(ok,ok+n+1,0);
39      d[1]=0;
40      for(int k=1;k<=n;++k){
41          int u=n+1;
42          for(int v=1;v<=n;++v)
43              if(!ok[v]&&d[v]<d[u])
44                  u=v;
45          ok[u]=1;
46          for(int v=1;v<=n;++v){
47              int cost=d[u]+G[u][v];
48              if(d[v]<=cost) continue;
49              d[v]=cost;
50              
51          }
52      }
53  }
```

讨论题

以上算法步骤中

循环查看pre[]数组内容时

每次跑Dijkstra后

pre[]数组会被修改

结果是否错误了?

并没有错!

T5 黄浦江

题目大意

南北隧道口依次编号为1-n, 有m条隧道设计方案,

安全约束：可共享出入口，但在江底不能交叉。

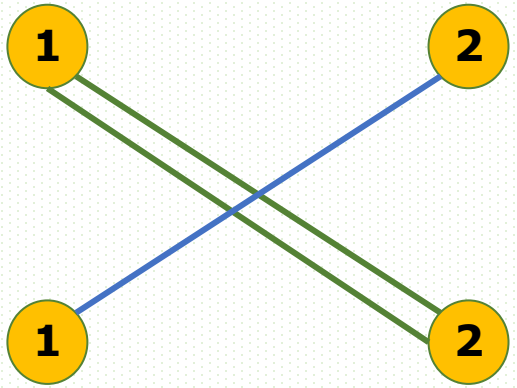
求：至少要否决几条隧道方案，满足安全约束条件。

互补计数

至少删几条=总条数-最多留几条

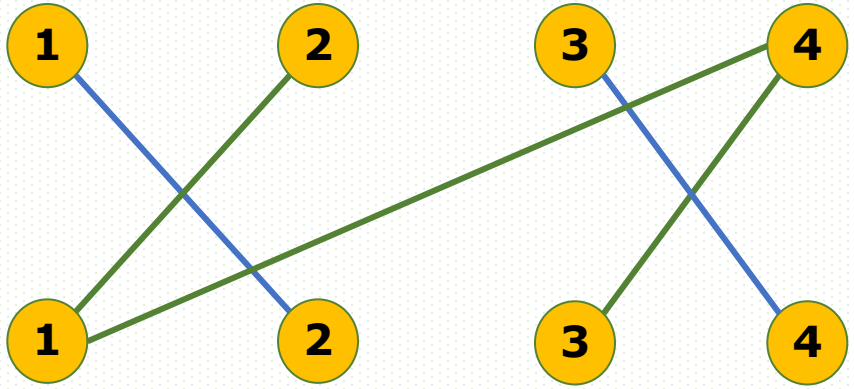
输入样例
2 3
1 2
1 2
2 1

输出样例
1

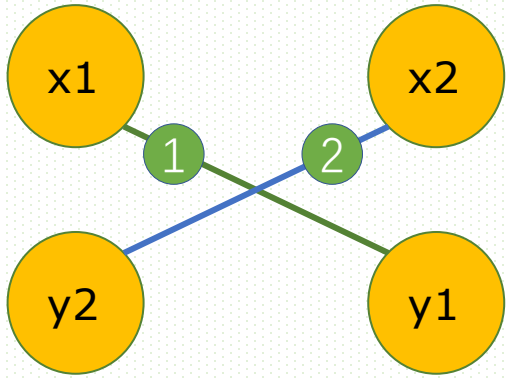


输入样例
4 5
4 3
3 4
4 1
2 1
1 2

输出样例
2



分析冲突的场景




```
x1 < x2 && y1 > y2
```

或

```
x2 < x1 && y2 < y1
```

交叉

则，不交叉的条件为

```
x1 < x2 && y1 <= y2
```

```
x1 > x2 && y1 >= y2
```

```
x1 == x2 && y1 == y2
```

目标：选择尽可能多的隧道使得它们两两不交叉，等价于寻找一个最大的隧道子集，其中任意两条隧道满足上述不交叉条件。

全序：一个序列中，任意两个元素之间有明确的大小关系。

偏序：一个序列中，只要部分元素之间有明确的大小关系。

问题分析

目标：选择尽可能多的隧道使得它们两两不交叉，

等价于寻找一个最大的隧道子集，其中任意两条隧道满足不交叉条件。

两条隧道是否交叉，取决于两条隧道端点之间的顺序关系 **二维偏序问题**

涉及两个维度上需要比较。 直接处理二维点对之间的交叉关系比较复杂。

降维处理 ⟸

需要考虑**所有**点对之间的两两关系

约束是**全局**的，不能独立处理单个点

暴力搜索所有子集是**指数复杂度**，不可行

核心思想：固定一个维度的顺序，将另一个维度的约束转化为序列问题。

降维处理

一条隧道有南北两个位置（x,y）

所有隧道按照x序列升序排列

二维的交叉约束转化为y序列的单调非递减约束

当x升序排列后，所有的xi的大小关系已确定

则x对应的y值，应该满足

$x_i < x_{i+1}$

$y_i >= y_{i+1}$

longest non-decreasing subsequence

最长不下降子序列

问题转化为：按照 x 从小到大排序，求对应的y值的最长不下降子序列

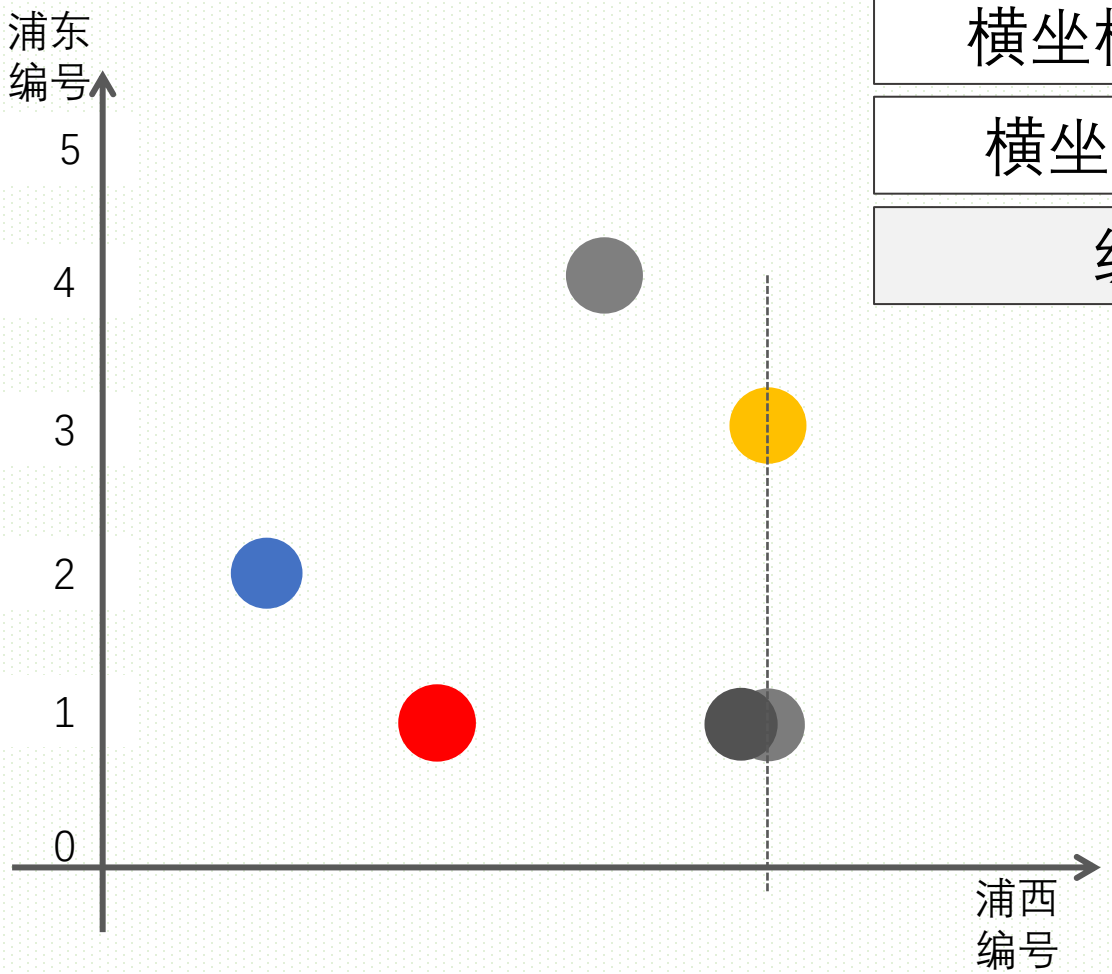
LNDS

x相同时，y从小到大排序

longest non-decreasing subsequence
最长不下降子序列

排序降维+LNDS

以某维度主导排序，达到降维效果



横坐标越小，排序越靠前

横坐标相同时，怎么办?

纵坐标小的靠前

对排序后的点的纵坐标求解LNDS

自定义+排序

```
struct tunnel {
    int x, y;
} t[N];
bool cmp(const tunnel&a, const tunnel&b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
```

优先比较横坐标x越小越靠前　　*若横坐标相同，比较纵坐标y越小越靠前*

```
for (int i = 1; i <= m; i++)
    cin >> t[i].x >> t[i].y;

sort(t + 1, t + 1 + m, cmp);
```

LNDS 解法1 DP

```
int f[N]; //f[i]:以i结尾最长不降子序列长度

for (int i = 2; i <= m; i++) {
    f[i] = 1;
    for (int j = 1; j < i; j++) {
        if (t[i].y >= t[j].y)
            f[i] = max(f[i], f[j] + 1);
    }
}

int LNDS = *max_element(f + 1, f + 1 + m);

cout << m - LNDS << endl;
```

时间复杂度？

$O(m^2)$

m<=200000

LNDS 解法2

Dilworth定理

求y序列的最长不降子序列长度 <=> *求y序列的的下降子序列最小划分数*

不降子序列 <<反链>> *下降子序列*

O(mlogm)

LNDS2

y[]最长不降子序列 = y[]下降子序列最小划分

```
int d[N]; //d[i]:记录第i+1个下降子序列的最后一个元素的值(最小值)

fill(d, d + m, INF); //初始化为较大值(2e9)

for (int i = 1; i <= m; i++)
    *upper_bound(d, d + m, t[i].y) = [          ];
```

思考：如果找不到可以接上的子序列怎么办？

找到第一个大于t[i].y的位置，即找到可以接上的下降子序列，把该子序列最小值设为t[i].y

```
int LNDS = lower_bound(d, d + m, [    ]) - d;

cout << [          ] << endl;
```

SH-B260105 订正题号

T1	3450	星际探险
T2	2209	删除子数组
T3	3113	轮流值班
T4	3466	堵车
T5	899	黄浦江