Notes for ACMICPC World Finals 2013

ACMICPC World Finals 2013 **参考资料**

Chinese Edition **中文版**

*Shanghai Jiao Tong University* : **Mithril**



***Coach* 教 练**

Yong YU 俞 勇

***Team member* 队 员**

Jingbo SHANG 商静波

Bin JIN 金斌

Xiaoxu GUO 郭晓旭

目录

[多边形与圆面积交 2](#_Toc414368840)

[二维几何 2](#_Toc414368841)

[nlogn半平面交 5](#_Toc414368842)

[三维几何操作合并 6](#_Toc414368843)

[三维旋转操作 7](#_Toc414368844)

[三维凸包随机增量 8](#_Toc414368845)

[三维凸包随机增量 9](#_Toc414368846)

[直线和凸包交点（返回最近和最远点） 10](#_Toc414368847)

[圆的面积模板(n^2logn) 11](#_Toc414368848)

[三角形的心 13](#_Toc414368849)

[最小覆盖球 14](#_Toc414368850)

[经纬度求球面最短距离 15](#_Toc414368851)

[长方体表面两点最短距离 16](#_Toc414368852)

[最大团 16](#_Toc414368853)

[极大团计数 17](#_Toc414368854)

[KM 18](#_Toc414368855)

[无向图最小割 19](#_Toc414368856)

[带花树 20](#_Toc414368857)

[动态最小生成树 21](#_Toc414368858)

[Hopcroft 24](#_Toc414368859)

[素数判定 25](#_Toc414368860)

[启发式分解 25](#_Toc414368861)

[二次剩余 27](#_Toc414368862)

[Pell方程 28](#_Toc414368863)

[蔡勒公式 29](#_Toc414368864)

[Romberg 29](#_Toc414368865)

[线性规划 29](#_Toc414368866)

[FFT && NTT 30](#_Toc414368867)

[回文串manacher 31](#_Toc414368868)

[后缀数组(nlogn) 32](#_Toc414368869)

[dc3 32](#_Toc414368870)

[后缀自动机 33](#_Toc414368871)

[字符串最小表示 33](#_Toc414368872)

[DancingLinks 34](#_Toc414368873)

[弦图的完美消除序列 35](#_Toc414368874)

[图同构Hash 35](#_Toc414368875)

[魔方旋转群 35](#_Toc414368876)

[直线下有多少个格点 36](#_Toc414368877)

[综合 36](#_Toc414368878)

[基本形 公式 37](#_Toc414368879)

[树的计数 38](#_Toc414368880)

[代数 39](#_Toc414368881)

[三角公式 39](#_Toc414368882)

[积分表 39](#_Toc414368883)

[Java IO 40](#_Toc414368884)

[Vimrc 41](#_Toc414368885)

## 多边形与圆面积交

point ORI**;**

**double** r**;**

**int** n**;**

point info**[**maxn**];**

//用有向面积，划分成一个三角形和圆的面积的交

**double** area2**(**point pa**,** point pb**)** **{**

**if** **(**pa**.**len**()** **<** pb**.**len**())** swap**(**pa**,** pb**);**

**if** **(**pb**.**len**()** **<** eps**)** **return** **0;**

**double** a**,** b**,** c**,** B**,** C**,** sinB**,** cosB**,** sinC**,** cosC**,** S**,** h**,** theta**;**

a **=** pb**.**len**();**

b **=** pa**.**len**();**

c **=** **(**pb **-** pa**).**len**();**

cosB **=** dot**(**pb**,** pb **-** pa**)** **/** a **/** c**;**

B **=** acos**(**cosB**);**

cosC **=** dot**(**pa**,** pb**)** **/** a **/** b**;**

C **=** acos**(**cosC**);**

**if** **(**a **>** r**)** **{**

S **=** **(**C**/2)\***r**\***r**;**

h **=** a**\***b**\***sin**(**C**)/**c**;**

**if** **(**h **<** r **&&** B **<** PI**/2)** S **-=** **(**acos**(**h**/**r**)\***r**\***r **-** h**\***sqrt**(**r**\***r**-**h**\***h**));**

**}** **else** **if** **(**b **>** r**)** **{**

theta **=** PI **-** B **-** asin**(**sin**(**B**)/**r**\***a**);**

S **=** **.5\***a**\***r**\***sin**(**theta**)** **+** **(**C**-**theta**)/2\***r**\***r**;**

**}** **else** **{**

S **=** **.5\***sin**(**C**)\***a**\***b**;**

**}**

//printf("res = %.4f\n", S);

**return** S**;**

**}**

**double** area**()** **{**

**double** S **=** **0;**

**for** **(int** i **=** **0;** i **<** n**;** **++**i**)** **{**

S **+=** area2**(**info**[**i**],** info**[**i **+** **1])** **\*** Sign**(**cross**(**info**[**i**],** info**[**i **+** **1]));**

**}**

**return** fabs**(**S**);**

**}**

## 二维几何

**struct** Point **{**

Point Rotate**(double** a**)** **const** **{**

**return** Point**(**x **\*** cos**(**a**)** **-** y **\*** sin**(**a**),** x **\*** sin**(**a**)** **+** y **\*** cos**(**a**));**

**}**

**};**

**bool** Line\_Intersect**(const** Point **&**a**,** **const** Point **&**b**,** **const** Point **&**c**,** **const** Point **&**d**,** Point **&**e**)** **{**

**double** s1 **=** Det**(**c **-** a**,** d **-** a**);**

**double** s2 **=** Det**(**d **-** b**,** c **-** b**);**

**if** **(!**Sign**(**s1 **+** s2**))** **return** **0;**

e **=** **(**b **-** a**)** **\*** **(**s1 **/** **(**s1 **+** s2**))** **+** a**;**

**return** **1;**

**}**

**int** Side**(const** Point **&**a**,** **const** Point **&**b**,** **const** Point **&**c**)** **{**

**return** Sign**(**Det**(**c **-** a**,** b **-** a**));**

**}**

**bool** In\_The\_Seg**(const** Point **&**a**,** **const** Point **&**b**,** **const** Point **&**c**)** **{**

**if** **(**Sign**(**Dist**(**a**,** b**,** c**)))** **return** **0;**// Not needed when you make sure it does technically.

**return** Sign**(**Dot**(**a **-** c**,** b **-** c**))** **<=** **0;**

**}**

**bool** Seg\_Intersect**(const** Point **&**a**,** **const** Point **&**b**,** **const** Point **&**c**,** **const** Point **&**d**,** Point **&**e**)** **{**

**double** s1 **=** Det**(**c **-** a**,** d **-** a**);**

**double** s2 **=** Det**(**d **-** b**,** c **-** b**);**

**if** **(!**Sign**(**s1 **+** s2**))** **return** **0;**

e **=** **(**b **-** a**)** **\*** **(**s1 **/** **(**s1 **+** s2**))** **+** a**;**

**return** In\_The\_Seg**(**a**,** b**,** e**)** **&&** In\_The\_Seg**(**c**,** d**,** e**);**

**}**

**struct** Circle **{**

Point o**;**

**double** r**;**// Squared

**bool** Inside**(**Point a**)** **{**

**return** Sqr**(**a**.**x **-** o**.**x**)** **+** Sqr**(**a**.**y **-** o**.**y**)** **<=** r**;**

**}**

**void** Calc**(**Point a**,** Point b**)** **{**

o**.**x **=** **(**a**.**x **+** b**.**x**)** **/** **2;**

o**.**y **=** **(**a**.**y **+** b**.**y**)** **/** **2;**

r **=** Sqr**(**a**.**x **-** o**.**x**)** **+** Sqr**(**a**.**y **-** o**.**y**);**

**}**

**void** Calc**(**Point a**,** Point b**,** Point c**)** **{**// Not certain if a, b and c lie in the same line, which needs prejudging.

**double** a1 **=** **2** **\*** **(**a**.**x **-** b**.**x**);**

**double** b1 **=** **2** **\*** **(**a**.**y **-** b**.**y**);**

**double** c1 **=** Sqr**(**a**.**x**)** **-** Sqr**(**b**.**x**)** **+** Sqr**(**a**.**y**)** **-** Sqr**(**b**.**y**);**

**double** a2 **=** **2** **\*** **(**a**.**x **-** c**.**x**);**

**double** b2 **=** **2** **\*** **(**a**.**y **-** c**.**y**);**

**double** c2 **=** Sqr**(**a**.**x**)** **-** Sqr**(**c**.**x**)** **+** Sqr**(**a**.**y**)** **-** Sqr**(**c**.**y**);**

o**.**x **=** **(**c1 **\*** b2 **-** c2 **\*** b1**)** **/** **(**a1 **\*** b2 **-** a2 **\*** b1**);**

o**.**y **=** **(**c1 **\*** a2 **-** c2 **\*** a1**)** **/** **(**a2 **\*** b1 **-** a1 **\*** b2**);**

r **=** Sqr**(**a**.**x **-** o**.**x**)** **+** Sqr**(**a**.**y **-** o**.**y**);**

**}**

**bool** Intersect\_With\_Line**(**Point fr**,** Point to**,** Point **&**A**,** Point **&**B**)** **const** **{**

**if** **(**Sign**(**Det**(**o **-** fr**,** to **-** fr**))** **>** **0)** swap**(**fr**,** to**);**

**double** R **=** Sqrt**(**r**);**

**double** h **=** Dist**(**fr**,** to**,** o**);**

**if** **(**Sign**(**h **-** R**)** **>** **0)** **return** **0;**

Point mm **=** **(**to **-** fr**).**Unit**().**Rotate**(-**pi **/** **2)** **\*** h **+** o**;**

**double** l **=** Sqrt**(**Sqr**(**R**)** **-** Sqr**(**h**));**

Point vv **=** **(**to **-** fr**).**Unit**()** **\*** l**;**

A **=** mm **-** vv**;**

B **=** mm **+** vv**;**

**return** **1;**

**}**

**bool** Contain**(const** Circle **&**a**)** **const** **{**// Not tested

**return** Sign**(**Sqrt**(**a**.**r**)** **+** **(**o **-** a**.**o**).**Length**()** **-** Sqrt**(**r**))** **<** **0;**

**}**

**bool** Disjunct**(const** Circle **&**a**)** **const** **{**// Not tested

**return** Sign**(**Sqrt**(**a**.**r**)** **+** Sqrt**(**r**)** **-** **(**o **-** a**.**o**).**Length**())** **<** **0;**

**}**

**};**

**bool** Intersect**(**Circle a**,** Circle b**,** Point **&**A**,** Point **&**B**)** **{**// Not tested, and must take care if a and b are the same one

**if** **(**a**.**Contain**(**b**)** **||** b**.**Contain**(**a**)** **||** a**.**Disjunct**(**b**))** **return** **0;**

**double** s1 **=** **(**a**.**o **-** b**.**o**).**Length**();**

**double** s2 **=** **(**a**.**r **-** b**.**r**)** **/** s1**;**

**double** aa **=** **(**s1 **+** s2**)** **/** **2;**

**double** bb **=** **(**s1 **-** s2**)** **/** **2;**

Point mm **=** **(**b**.**o **-** a**.**o**)** **\*** **(**aa **/** **(**aa **+** bb**))** **+** a**.**o**;**

**double** h **=** Sqrt**(**a**.**r **-** Sqr**(**aa**));**

Point vv **=** **(**b**.**o **-** a**.**o**).**Unit**().**Rotate**(**pi **/** **2)** **\*** h**;**

A **=** mm **+** vv**;**

B **=** mm **-** vv**;**

**return** **1;**

**}**

**struct** Polygon **{**

Point list**[**maxn**];**

**int** n**;**

Polygon**()** **{**

**}**

Polygon**(const** Polygon **&**a**)** **{**

n **=** a**.**n**;**

**int** i**;**

**for** **(**i **=** **0;** i **<** n**;** i**++)**

list**[**i**]** **=** a**.**list**[**i**];**

**}**

Polygon **&operator** **=** **(const** Polygon **&**a**)** **{**

**if** **(this** **==** **&**a**)** **return** **\*this;**

n **=** a**.**n**;**

**int** i**;**

**for** **(**i **=** **0;** i **<** n**;** i**++)**

list**[**i**]** **=** a**.**list**[**i**];**

**return** **\*this;**

**}**

Polygon Cut**(const** Point **&**a**,** **const** Point **&**b**)** **{**

**static** Polygon res**;**

res**.**n **=** **0;**

**int** i**,** s1**,** s2**;**

Point curr**;**

**for** **(**i **=** **0;** i **<** n**;** i**++)** **{**

s1 **=** Sign**(**Det**(**list**[**i**]** **-** a**,** b **-** a**));**

s2 **=** Sign**(**Det**(**list**[(**i **+** **1)** **%** n**]** **-** a**,** b **-** a**));**

**if** **(**s1 **<=** **0)** res**.**list**[**res**.**n**++]** **=** list**[**i**];**

**if** **(**s1 **\*** s2 **<** **0)** **{**

Line\_Intersect**(**a**,** b**,** list**[**i**],** list**[(**i **+** **1)** **%** n**],** curr**);**

res**.**list**[**res**.**n**++]** **=** curr**;**

**}**

**}**

**return** res**;**

**}**

Polygon Strict\_Cut**(const** Point **&**fr**,** **const** Point **&**to**)** **const** **{**

**static** Polygon res**;**

res**.**n **=** **0;**

**int** i**,** s1**,** s2**;**

Point a**,** b**;**

**for** **(**i **=** **0;** i **<** n**;** i**++)**

**if** **(**Side**(**fr**,** to**,** list**[**i**])** **<** **0)** **break;**

**if** **(**i **==** n**)** **return** res**;**

Point c**;**

**for** **(**i **=** **0;** i **<** n**;** i**++)** **{**

a **=** list**[**i**];**

b **=** list**[(**i **+** **1)** **%** n**];**

s1 **=** Side**(**fr**,** to**,** a**);**

s2 **=** Side**(**fr**,** to**,** b**);**

**if** **(**s1 **<=** **0)** res**.**list**[**res**.**n**++]** **=** a**;**

**if** **(**s1 **\*** s2 **<** **0)** **{**

Line\_Intersect**(**fr**,** to**,** a**,** b**,** c**);**

res**.**list**[**res**.**n**++]** **=** c**;**

**}**

**}**

**return** res**;**

**}**

**bool** Contain**(const** Point **&**curr**)** **const** **{**

**int** i**,** res **=** **0;**

Point A**,** B**;**

**for** **(**i **=** **0;** i **<** n**;** i**++)** **{**

A **=** list**[**i**];**

B **=** list**[(**i **+** **1)** **%** n**];**

**if** **(**In\_The\_Seg**(**A**,** B**,** curr**))** **return** **1;**

**if** **(**Sign**(**A**.**y **-** B**.**y**)** **<=** **0)** swap**(**A**,** B**);**

**if** **(**Sign**(**curr**.**y **-** A**.**y**)** **>** **0)** **continue;**

**if** **(**Sign**(**curr**.**y **-** B**.**y**)** **<=** **0)** **continue;**

res **+=** Sign**(**Det**(**B **-** curr**,** A **-** curr**))** **>** **0;**

**}**

**return** res **&** **1;**

**}**

**};**

## nlogn半平面交

#define cross(p1,p2,p3) ((p2.x-p1.x)\*(p3.y-p1.y)-(p3.x-p1.x)\*(p2.y-p1.y))

Point isSS**(**Point p1**,** Point p2**,** Point q1**,** Point q2**)** **{**

**double** a1 **=** cross**(**q1**,**q2**,**p1**),** a2 **=** **-**cross**(**q1**,**q2**,**p2**);**

**return** **(**p1 **\*** a2 **+** p2 **\*** a1**)** **/** **(**a1 **+** a2**);**

**}**

**struct** Border **{**

**void** setAlpha**()** **{**

alpha **=** atan2**(**p2**.**y **-** p1**.**y**,** p2**.**x **-** p1**.**x**);**

**}**

**};**

**int** n**;**

**const** **int** MAX\_N\_BORDER **=** **20000** **+** **10;**

Border border**[**MAX\_N\_BORDER**];**

**bool** **operator<(const** Border**&**a**,** **const** Border**&**b**)** **{**

**int** c **=** sign**(**a**.**alpha **-** b**.**alpha**);**

**if** **(**c **!=** **0)**

**return** c **==** **1;**

**return** crossOp**(**b**.**p1**,**b**.**p2**,**a**.**p1**)** **>=** **0;**

**}**

**bool** **operator==(const** Border**&**a**,** **const** Border**&**b**)** **{**

**return** sign**(**a**.**alpha **-** b**.**alpha**)** **==** **0;**

**}**

**void** add**(double** x**,** **double** y**,** **double** nx**,** **double** ny**)** **{**

border**[**n**].**p1 **=** Point**(**x**,** y**);**

border**[**n**].**p2 **=** Point**(**nx**,** ny**);**

border**[**n**].**setAlpha**();**

n**++;**

**}**

Point isBorder**(const** Border**&**a**,** **const** Border**&**b**)** **{**

**return** isSS**(**a**.**p1**,** a**.**p2**,** b**.**p1**,** b**.**p2**);**

**}**

Border que**[**MAX\_N\_BORDER**];**

**int** qh**,** qt**;**

**bool** check**(const** Border**&**a**,** **const** Border**&**b**,** **const** Border**&**me**)** **{**

Point is **=** isBorder**(**a**,** b**);**

**return** crossOp**(**me**.**p1**,**me**.**p2**,**is**)** **>** **0;**

**}**

**void** convexIntersection**()** **{**

qh **=** qt **=** **0;**

sort**(**border**,** border **+** n**);**

n **=** unique**(**border**,** border **+** n**)** **-** border**;**

**for** **(int** i **=** **0;** i **<** n**;** **++**i**)** **{**

Border cur **=** border**[**i**];**

**while** **(**qh **+** **1** **<** qt **&&** **!**check**(**que**[**qt **-** **2],** que**[**qt **-** **1],** cur**))**

**--**qt**;**

**while** **(**qh **+** **1** **<** qt **&&** **!**check**(**que**[**qh**],** que**[**qh **+** **1],** cur**))**

**++**qh**;**

que**[**qt**++]** **=** cur**;**

**}**

**while** **(**qh **+** **1** **<** qt **&&** **!**check**(**que**[**qt **-** **2],** que**[**qt **-** **1],** que**[**qh**]))**

**--**qt**;**

**while** **(**qh **+** **1** **<** qt **&&** **!**check**(**que**[**qh**],** que**[**qh **+** **1],** que**[**qt **-** **1]))**

**++**qh**;**

**}**

**void** calcArea**()** **{**

**static** Point ps**[**MAX\_N\_BORDER**];**

**int** cnt **=** **0;**

**if** **(**qt **-** qh **<=** **2)** **{**

puts**(**"0.0"**);**

**return;**

**}**

**for** **(int** i **=** qh**;** i **<** qt**;** **++**i**)** **{**

**int** next **=** i **+** **1** **==** qt **?** qh **:** i **+** **1;**

ps**[**cnt**++]** **=** isBorder**(**que**[**i**],** que**[**next**]);**

**}**

**double** area **=** **0;**

**for** **(int** i **=** **0;** i **<** cnt**;** **++**i**)** **{**

area **+=** ps**[**i**].**det**(**ps**[(**i **+** **1)** **%** cnt**]);**

**}**

area **/=** **2;**

area **=** fabsl**(**area**);**

cout**.**setf**(**ios**::**fixed**);**

cout**.**precision**(1);**

cout **<<** area **<<** endl**;**

**}**

**void** halfPlaneIntersection**()**

**{**

cin **>>** n**;**

**for** **(int** i **=** **0;** i **<** n**;** **++**i**)** **{**

border**[**i**].**read**();**

**}**

add**(0,** **0,** LARGE**,** **0);**

add**(**LARGE**,** **0,** LARGE**,** LARGE**);**

add**(**LARGE**,** LARGE**,** **0,** LARGE**);**

add**(0,** LARGE**,** **0,** **0);**

convexIntersection**();**

calcArea**();**

**}**

## 三维几何操作合并

**const** **double** pi **=** acos**(-1.0);** **double** a**[4][4];**

**int** dcmp**(const** **double** **&**a**,** **const** **double** **&**b **=** **0,** **const** **double** **&** zero **=** **1e-6){**

**if** **(**a **-** b **<** **-**zero**)** **return** **-1;** **return** a **-** b **>** zero**;}**

**void** multi**(const** **double** a**[4][4],const** **double** b**[4][4],double** c**[4][4]){**

**for(int** i**=0;**i**<4;**i**++)**

**for(int** j**=0;**j**<4;**j**++){**

c**[**i**][**j**]=**a**[**i**][0]\***b**[0][**j**];**

**for(int** k**=1;**k**<4;**k**++)**

c**[**i**][**j**]+=**a**[**i**][**k**]\***b**[**k**][**j**];**

**}}**

**void** multi**(double** a**[4][4],const** **double** b**[4][4]){**

**static** **double** c**[4][4];**

multi**(**a**,**b**,**c**);**

memcpy**(**a**,**c**,sizeof(**a**[0][0])\*16);**

**}**

**void** Macro**(){**

**double** b**[4][4]={1,** **0,** **0,** **0,** **0,** **1,** **0,** **0,** **0,** **0,** **1,** **0,** **0,** **0,** **0,** **1};**

memcpy**(**a**,**b**,sizeof(**a**[0][0])\*16);**

**}**

**void** Translation**(const** Point\_3 **&**s**){**

**double** p**[4][4]={1,** **0,** **0,** **0,** **0,** **1,** **0,** **0,** **0,** **0,** **1,** **0,** s**.**x**,** s**.**y**,** s**.**z**,** **1};**

multi**(**a**,**p**);**

**}**

**void** Scaling**(const** Point\_3 **&**s**){**

**double** p**[4][4]={**s**.**x**,** **0,** **0,** **0,** **0,** s**.**y**,** **0,** **0,** **0,** **0,** s**.**z**,** **0,** **0,** **0,** **0,** **1};**

multi**(**a**,**p**);**

**}**

**void** Rotate**(const** Point\_3 **&**s**,** **double** r**)** **{**

**double** l**=**s**.**Length**();** **double** x**=**s**.**x**/**l**,**y**=**s**.**y**/**l**,**z**=**s**.**z**/**l**;**

**double** SinA**=**sin**(**r**),**CosA**=**cos**(**r**);**

**double** p**[4][4]={**CosA **+** **(1** **-** CosA**)** **\*** x **\*** x**,** **(1** **-** CosA**)** **\*** x **\*** y **-** SinA **\*** z**,** **(1** **-** CosA**)** **\*** x **\*** z **+** SinA **\*** y**,** **0,(1** **-** CosA**)** **\*** y **\*** x **+** SinA **\*** z**,**

CosA **+** **(1** **-** CosA**)** **\*** y **\*** y**,** **(1** **-** CosA**)** **\*** y **\*** z **-** SinA **\*** x**,** **0,**

**(1** **-** CosA**)** **\*** z **\*** x **-** SinA **\*** y**,** **(1** **-** CosA**)** **\*** z **\*** y **+** SinA **\*** x**,** CosA **+** **(1** **-** CosA**)** **\*** z **\*** z**,** **0,** **0,** **0,** **0,** **1};**

multi**(**a**,**p**);**

**}**

Point\_3 opt**(const** Point\_3**&**s**){**

**double** x**,**y**,**z**;**

**return** Point\_3**(** s**.**x **\*** a**[0][0]** **+** s**.**y **\*** a**[1][0]** **+** s**.**z **\*** a**[2][0]** **+** a**[3][0],**

s**.**x **\*** a**[0][1]** **+** s**.**y **\*** a**[1][1]** **+** s**.**z **\*** a**[2][1]** **+** a**[3][1],**

s**.**x **\*** a**[0][2]** **+** s**.**y **\*** a**[1][2]** **+** s**.**z **\*** a**[2][2]** **+** a**[3][2]);**

**}**

**int** main**(){**

Macro**();**

**int** n**;for** **(**scanf**(**"%d"**,** **&**n**);** n**;** n**--)** **{**

**char** c**;** Point\_3 p**;**

scanf**(**"\n%c%lf%lf%lf"**,** **&**c**,** **&**p**.**x**,** **&**p**.**y**,** **&**p**.**z**);**

**if** **(**c **==** 'T'**)** Translation**(**p**);** **if** **(**c **==** 'S'**)** Scaling**(**p**);**

**if** **(**c **==** 'R'**)** **{** **double** r**;**scanf**(**"%lf\n"**,** **&**r**);**

Rotate**(**p**,** r**);** //===========绕OP逆时针旋转r角度

**}}**

**for** **(**scanf**(**"%d"**,** **&**n**);** n**;** n**--)** **{**

Point\_3 p**,** p2**;** scanf**(**"%lf%lf%lf"**,** **&**p**.**x**,** **&**p**.**y**,** **&**p**.**z**);**

p2 **=** opt**(**p**);** printf**(**“**%**f **%**f **%**f\n”**,**p2**.**x**,**p2**.**y**,**p2**.**z**);**

**}}**

## 三维旋转操作

//a点绕Ob向量，逆时针旋转弧度angle, sin(angle),cos(angle)先求出来，减少精度问题。

point e1**,**e2**,**e3**;** point Rotate**(** point a**,** point b**,** **double** angle **){**

b**.**std**();**//单位化，注意b不能为（0，0，0）

e3**=**b**;** **double** lens**=**a**\***e3**;**//dot(a,e3)

e1**=**a **-** e3**\***lens**;** **if** **(**e1**.**len**()>(1e-8))** e1**.**std**();** **else** **return** a**;**

e2**=**e1**/**e3**;** //det(e1,e3)

**double** x1**,**y1**,**x**,**y**;** y1**=**a**\***e1**;** x1**=**a**\***e2**;**

x**=**x1**\***cos**(**angle**)** **-** y1**\***sin**(**angle**);** y**=**x1**\***sin**(**angle**)** **+** y1**\***cos**(**angle**);**

**return** e3**\***lens **+** e1**\***y **+** e2**\***x**;** **}**

## 三维凸包随机增量

#define SIZE(X) (int(X.size()))

#define PI 3.14159265358979323846264338327950288

**struct** Point **{**

Point cross**(const** Point **&**p**)** **const** **{**

**return** Point**(**y **\*** p**.**z **-** z **\*** p**.**y**,** z **\*** p**.**x **-** x **\*** p**.**z**,** x **\*** p**.**y **-** y **\*** p**.**x**);**

**}**

**};**

**int** mark**[1005][1005];**

Point info**[1005];**

**int** n**,** cnt**;**

**double** mix**(const** Point **&**a**,** **const** Point **&**b**,** **const** Point **&**c**)** **{**

**return** a**.**dot**(**b**.**cross**(**c**));**

**}**

**double** area**(int** a**,** **int** b**,** **int** c**)** **{**

**return** **((**info**[**b**]** **-** info**[**a**]).**cross**(**info**[**c**]** **-** info**[**a**])).**length**();**

**}**

**double** volume**(int** a**,** **int** b**,** **int** c**,** **int** d**)** **{**

**return** mix**(**info**[**b**]** **-** info**[**a**],** info**[**c**]** **-** info**[**a**],** info**[**d**]** **-** info**[**a**]);**

**}**

**struct** Face **{**

**int** a**,** b**,** c**;**

Face**()** **{}**

Face**(int** a**,** **int** b**,** **int** c**):** a**(**a**),** b**(**b**),** c**(**c**)** **{}**

**int** **&operator** **[](int** k**)** **{**

**if** **(**k **==** **0)** **return** a**;**

**if** **(**k **==** **1)** **return** b**;**

**return** c**;**

**}**

**};**

vector **<**Face**>** face**;**

**inline** **void** insert**(int** a**,** **int** b**,** **int** c**)** **{**

face**.**push\_back**(**Face**(**a**,** b**,** c**));**

**}**

**void** add**(int** v**)** **{**

vector **<**Face**>** tmp**;**

**int** a**,** b**,** c**;**

cnt**++;**

**for** **(int** i **=** **0;** i **<** SIZE**(**face**);** i**++)** **{**

a **=** face**[**i**][0];**

b **=** face**[**i**][1];**

c **=** face**[**i**][2];**

**if** **(**Sign**(**volume**(**v**,** a**,** b**,** c**))** **<** **0)**

mark**[**a**][**b**]** **=** mark**[**b**][**a**]** **=** mark**[**b**][**c**]** **=** mark**[**c**][**b**]** **=** mark**[**c**][**a**]** **=** mark**[**a**][**c**]** **=** cnt**;**

**else**

tmp**.**push\_back**(**face**[**i**]);**

**}**

face **=** tmp**;**

**for** **(int** i **=** **0;** i **<** SIZE**(**tmp**);** i**++)** **{**

a **=** face**[**i**][0];**

b **=** face**[**i**][1];**

c **=** face**[**i**][2];**

**if** **(**mark**[**a**][**b**]** **==** cnt**)** insert**(**b**,** a**,** v**);**

**if** **(**mark**[**b**][**c**]** **==** cnt**)** insert**(**c**,** b**,** v**);**

**if** **(**mark**[**c**][**a**]** **==** cnt**)** insert**(**a**,** c**,** v**);**

**}**

**}**

**int** Find**()** **{**

**for** **(int** i **=** **2;** i **<** n**;** i**++)** **{**

Point ndir **=** **(**info**[0]** **-** info**[**i**]).**cross**(**info**[1]** **-** info**[**i**]);**

**if** **(**ndir **==** Point**())** **continue;**

swap**(**info**[**i**],** info**[2]);**

**for** **(int** j **=** i **+** **1;** j **<** n**;** j**++)**

**if** **(**Sign**(**volume**(0,** **1,** **2,** j**))** **!=** **0)** **{**

swap**(**info**[**j**],** info**[3]);**

insert**(0,** **1,** **2);**

insert**(0,** **2,** **1);**

**return** **1;**

**}**

**}**

**return** **0;**

**}**

**int** main**()** **{**

**for** **(;** scanf**(**"%d"**,** **&**n**)** **==** **1;** **)** **{**

**for** **(int** i **=** **0;** i **<** n**;** i**++)**

info**[**i**].**Input**();**

sort**(**info**,** info **+** n**);**

n **=** unique**(**info**,** info **+** n**)** **-** info**;**

face**.**clear**();**

random\_shuffle**(**info**,** info **+** n**);**

**if** **(**Find**())** **{**

memset**(**mark**,** **0,** **sizeof(**mark**));**

cnt **=** **0;**

**for** **(int** i **=** **3;** i **<** n**;** i**++)** add**(**i**);**

vector**<**Point**>** Ndir**;**

**for** **(int** i **=** **0;** i **<** SIZE**(**face**);** **++**i**)** **{**

Point p **=** **(**info**[**face**[**i**][0]]** **-** info**[**face**[**i**][1]]).**cross**(**info**[**face**[**i**][2]]** **-** info**[**face**[**i**][1]]);**

p **=** p **/** p**.**length**();**

Ndir**.**push\_back**(**p**);**

**}**

sort**(**Ndir**.**begin**(),** Ndir**.**end**());**

**int** ans **=** unique**(**Ndir**.**begin**(),** Ndir**.**end**())** **-** Ndir**.**begin**();**

printf**(**"%d\n"**,** ans**);**

**}** **else** **{**

printf**(**"1\n"**);**

**}**

**}**

**}**

## 三维凸包随机增量

**double** calcDist**(const** Point **&**p**,** **int** a**,** **int** b**,** **int** c**)** **{**

**return** fabs**(**mix**(**info**[**a**]** **-** p**,** info**[**b**]** **-** p**,** info**[**c**]** **-** p**)** **/** area**(**a**,** b**,** c**));**

**}**

//compute the minimal distance of center of any faces

**double** findDist**()** **{**

//compute center of mass

**double** totalWeight **=** **0;**

Point center**(.0,** **.0,** **.0);**

Point first **=** info**[**face**[0][0]];**

**for** **(int** i **=** **0;** i **<** SIZE**(**face**);** **++**i**)** **{**

Point p **=** **(**info**[**face**[**i**][0]]** **+** info**[**face**[**i**][1]]** **+** info**[**face**[**i**][2]]** **+** first**)** **\*** **.25;**

**double** weight **=** mix**(**info**[**face**[**i**][0]]** **-** first**,** info**[**face**[**i**][1]]** **-** first**,** info**[**face**[**i**][2]]** **-** first**);**

totalWeight **+=** weight**;**

center **=** center **+** p **\*** weight**;**

**}**

center **=** center **/** totalWeight**;**

//compute distance

**double** res **=** **1e100;**

**for** **(int** i **=** **0;** i **<** SIZE**(**face**);** **++**i**)** **{**

res **=** min**(**res**,** calcDist**(**center**,** face**[**i**][0],** face**[**i**][1],** face**[**i**][2]));**

**}**

**return** res**;**

**}**

## 直线和凸包交点（返回最近和最远点）

**double** calc**(**point a**,** point b**){**

**double** k**=**atan2**(**b**.**y**-**a**.**y **,** b**.**x**-**a**.**x**);** **if** **(**k**<**0**)** k**+=**2**\***pi**;return** k**;**

**}**//= the convex must compare y, then x£¬a[0] is the lower-right point

//======= three is no 3 points in line. a[] is convex 0~n-1

**void** prepare**(**point a**[]** **,double** w**[],int** **&**n**)** **{**

**int** i**;** rep**(**i**,**n**)** a**[**i**+**n**]=**a**[**i**];** a**[**2**\***n**]=**a**[**0**];**

rep**(**i**,**n**)** **{** w**[**i**]=**calc**(**a**[**i**],**a**[**i**+**1**]);**w**[**i**+**n**]=**w**[**i**];}**

**}**

**int** find**(double** k**,int** n **,** **double** w**[]){**

**if** **(**k**<=**w**[**0**]** **||** k**>**w**[**n**-**1**])** **return** 0**;** **int** l**,**r**,**mid**;** l**=**0**;** r**=**n**-**1**;**

**while** **(**l**<=**r**)** **{** mid**=(**l**+**r**)/**2**;if** **(**w**[**mid**]>=**k**)** r**=**mid**-**1**;** **else** l**=**mid**+**1**;**

**}return** r**+**1**;**

**}**

**int** dic**(const** point **&**a**,** **const** point **&**b **,** **int** l **,int** r **,** point c**[])** **{**

**int** s**;** **if** **(**area**(**a**,**b**,**c**[**l**])<**0**)** s**=-**1**;** **else** s**=**1**;** **int** mid**;**

**while** **(**l**<=**r**)** **{**

mid**=(**l**+**r**)/**2**;** **if** **(**area**(**a**,**b**,**c**[**mid**])\***s **<=** 0**)** r**=**mid**-**1**;** **else** l**=**mid**+**1**;**

**}return** r**+**1**;**

**}**

point get**(const** point **&**a**,** **const** point **&**b**,** point s1**,** point s2**)** **{**

**double** k1**,**k2**;** point tmp**;** k1**=**area**(**a**,**b**,**s1**);** k2**=**area**(**a**,**b**,**s2**);**

**if** **(**cmp**(**k1**)==**0**)** **return** s1**;** **if** **(**cmp**(**k2**)==**0**)** **return** s2**;**

tmp**=(**s1**\***k2 ¨C s2**\***k1**)** **/** **(**k2**-**k1**);** **return** tmp**;**

**}**

**bool** line\_cross\_convex**(**point a**,** point b **,**point c**[]** **,** **int** n**,** point **&**cp1**,** point **&**cp2 **,** **double** w**[])** **{**

**int** i**,**j**;**

i**=**find**(**calc**(**a**,**b**),**n**,**w**);**

j**=**find**(**calc**(**b**,**a**),**n**,**w**);**

**double** k1**,**k2**;**

k1**=**area**(**a**,**b**,**c**[**i**]);** k2**=**area**(**a**,**b**,**c**[**j**]);**

**if** **(**cmp**(**k1**)\***cmp**(**k2**)>**0**)** **return** **false;** //no cross

**if** **(**cmp**(**k1**)==**0 **||** cmp**(**k2**)==**0**)** **{** //cross a point or a line in the convex

**if** **(**cmp**(**k1**)==**0**)** **{**

**if** **(**cmp**(**area**(**a**,**b**,**c**[**i**+**1**]))==**0**)** **{**cp1**=**c**[**i**];** cp2**=**c**[**i**+**1**];}**

**else** cp1**=**cp2**=**c**[**i**];** **return** **true;**

**}**

**if** **(**cmp**(**k2**)==**0**)** **{**

**if** **(**cmp**(**area**(**a**,**b**,**c**[**j**+**1**]))==**0**)** **{**cp1**=**c**[**j**];**cp2**=**c**[**j**+**1**];}**

**else** cp1**=**cp2**=**c**[**j**];**

**}return** **true;**

**}**

**if** **(**i**>**j**)** swap**(**i**,**j**);** **int** x**,**y**;** x**=**dic**(**a**,**b**,**i**,**j**,**c**);** y**=**dic**(**a**,**b**,**j**,**i**+**n**,**c**);**

cp1**=**get**(**a**,**b**,**c**[**x**-**1**],**c**[**x**]);** cp2**=**get**(**a**,**b**,**c**[**y**-**1**],**c**[**y**]);**

**return** **true;}**

**const** **int** maxn=**200**;**const** **int** oo=**0x7fffffff**;

**int** w[maxn][maxn],x[maxn],y[maxn],px[maxn],py[maxn],sy[maxn],slack[maxn];

int par[maxn];**int** n;**int** pa[**200**][**2**],pb[**200**][**2**],n0,m0,na,nb;**char** s[**200**][**200**];

**void** adjust(**int** v){ sy[v]=py[v]; **if** (px[sy[v]]!=-**2**) adjust(px[sy[v]]);}

**bool** find(**int** v){**for** (int i=**0**;i<n;i++)

**if** (py[i]==-**1**){

**if** (slack[i]>x[v]+y[i]-w[v][i]){

slack[i]=x[v]+y[i]-w[v][i]; par[i]=v;}

**if** (x[v]+y[i]==w[v][i]){

py[i]=v; **if** (sy[i]==-**1**){adjust(i); **return** **1**;}

**if** (px[sy[i]]!=-**1**) **continue**; px[sy[i]]=i;

**if** (find(sy[i])) **return** **1**;

}}**return** **0**;}

**int** km(){**int** i,j,m;

**for** (i=**0**;i<n;i++) sy[i]=-**1**,y[i]=**0**;

**for** (i=**0**;i<n;i++) {x[i]=**0**; **for** (j=**0**;j<n;j++) x[i]=max(x[i],w[i][j]);}

**bool** flag;

**for** (i=**0**;i<n;i++){

**for** (j=**0**;j<n;j++) px[j]=py[j]=-**1**,slack[j]=oo;

px[i]=-**2**; **if** (find(i)) **continue**; flag=**false**;

**for** (;!flag;){

m=oo; **for** (j=**0**;j<n;j++) **if** (py[j]==-**1**) m=min(m,slack[j]);

**for** (j=**0**;j<n;j++){

**if** (px[j]!=-**1**) x[j]-=m;

**if** (py[j]!=-**1**) y[j]+=m;

**else** slack[j]-=m;}

**for** (j=**0**;j<n;j++){

**if** (py[j]==-**1**&&!slack[j]){

py[j]=par[j];

**if** (sy[j]==-**1**){ adjust(j); flag=**true**; **break**;}

px[sy[j]]=j; **if** (find(sy[j])){flag=**true**;**break**;}

}}}}

**int** ans=**0**; **for** (i=**0**;i<n;i++) ans+=w[sy[i]][i];**return** ans;}

## 圆的面积模板(n^2logn)

**struct** Tcir **{**

point o**;**

**double** r**;**

Tcir**()** **{**

**}**

Tcir**(const** point **&**o**,** **double** r**):** o**(**o**),** r**(**r**)** **{**

**}**

**};**

**struct** Tevent **{**

point p**;**

**double** ang**;**

**int** add**;**

Tevent**()** **{**

**}**

Tevent**(const** point **&**\_p**,** **double** \_ang**,** **int** \_add**):** p**(**\_p**),** ang**(**\_ang**),** add**(**\_add**)** **{**

**}**

**bool** **operator** **<(const** Tevent **&**a**)** **const** **{**

**return** ang **<** a**.**ang**;**

**}**

**}** eve**[**maxn **\*** **2];**

**int** E**,** cnt**;**

**double** sqr**(double** x**)** **{** **return** x **\*** x**;** **}**

**void** circleCrossCircle**(const** Tcir **&**a**,** **const** Tcir **&**b**)** **{**

**double** l **=** **(**a**.**o **-** b**.**o**).**len2**();**

**double** s **=** **((**a**.**r **-** b**.**r**)** **\*** **(**a**.**r **+** b**.**r**)** **/** l **+** **1)** **\*** **.5;**

**double** t **=** sqrt**(-(**l **-** sqr**(**a**.**r **-** b**.**r**))** **\*** **(**l **-** sqr**(**a**.**r **+** b**.**r**))** **/** **(**l **\*** l **\*** **4.));**

point dir **=** b**.**o **-** a**.**o**;**

point Ndir **=** point**(-**dir**.**y**,** dir**.**x**);**

point aa **=** a**.**o **+** dir **\*** s **+** Ndir **\*** t**;**

point bb **=** a**.**o **+** dir **\*** s **-** Ndir **\*** t**;**

**double** A **=** atan2**(**aa**.**y **-** a**.**o**.**y**,** aa**.**x **-** a**.**o**.**x**);**

**double** B **=** atan2**(**bb**.**y **-** a**.**o**.**y**,** bb**.**x **-** a**.**o**.**x**);**

eve**[**E**++]** **=** Tevent**(**bb**,** B**,** **1);**

eve**[**E**++]** **=** Tevent**(**aa**,** A**,** **-1);**

**if** **(**B **>** A**)** **{**

cnt**++;**

**}**

**}**

**bool** contain**(int** x1**,** **int** y1**,** **int** r1**,** **int** x2**,** **int** y2**,** **int** r2**)** **{**

**return** r1 **>=** r2 **&&** **(**x1 **-** x2**)** **\*** **(**x1 **-** x2**)** **+** **(**y1 **-** y2**)** **\*** **(**y1 **-** y2**)** **<=** **(**r1 **-** r2**)** **\*** **(**r1 **-** r2**);**

**}**

**bool** disjoint**(int** x1**,** **int** y1**,** **int** r1**,** **int** x2**,** **int** y2**,** **int** r2**)** **{**

**return** **(**x1 **-** x2**)** **\*** **(**x1 **-** x2**)** **+** **(**y1 **-** y2**)** **\*** **(**y1 **-** y2**)** **>=** **(**r1 **+** r2**)** **\*** **(**r1 **+** r2**);**

**}**

**bool** Same**(int** x1**,** **int** y1**,** **int** r1**,** **int** x2**,** **int** y2**,** **int** r2**)** **{**

**return** r1 **==** r2 **&&** x1 **==** x2 **&&** y1 **==** y2**;**

**}**

**bool** g**[**maxn**][**maxn**],** Overlap**[**maxn**][**maxn**];**

**double** Area**[**maxn**];**

**int** cX**[**maxn**],** cY**[**maxn**],** cR**[**maxn**];**

Tcir c**[**maxn**];**

**int** C**;**

**int** main**()** **{**

scanf**(**"%d"**,** **&**C**);**

**for** **(int** i **=** **0;** i **<** C**;** **++**i**)** **{**//去掉重复的圆

scanf**(**"%d%d%d"**,** cX**+**i**,** cY**+**i**,** cR**+**i**);**

**bool** found **=** **false;**

**for** **(int** j **=** **0;** j **<** i**;** **++**j**)** **{**

**if** **(**Same**(**cX**[**i**],** cY**[**i**],** cR**[**i**],** cX**[**j**],** cY**[**j**],** cR**[**j**]))** **{**

found **=** **true;**

**break;**

**}**

**}**

**if** **(**found**)** **{**

i**--;**

C**--;**

**continue;**

**}**

c**[**i**]** **=** Tcir**(**point**(**cX**[**i**],** cY**[**i**]),** cR**[**i**]);**

**}**

**for** **(int** i **=** **0;** i **<=** C**;** **++**i**)** Area**[**i**]** **=** **0;**

**for** **(int** i **=** **0;** i **<** C**;** **++**i**)** **{**

**for** **(int** j **=** **0;** j **<** C**;** **++**j**)** **{**

Overlap**[**i**][**j**]** **=** contain**(**cX**[**i**],** cY**[**i**],** cR**[**i**],** cX**[**j**],** cY**[**j**],** cR**[**j**]);**

**}**

**}**

**for** **(int** i **=** **0;** i **<** C**;** **++**i**)** **{**

**for** **(int** j **=** **0;** j **<** C**;** **++**j**)** **{**

g**[**i**][**j**]** **=** **!(**Overlap**[**i**][**j**]** **||** Overlap**[**j**][**i**]** **||** disjoint**(**cX**[**i**],** cY**[**i**],** cR**[**i**],** cX**[**j**],** cY**[**j**],** cR**[**j**]));**

**}**

**}**

**for** **(int** i **=** **0;** i **<** C**;** **++**i**)** **{**

E **=** **0;**

cnt **=** **1;**

**for** **(int** j **=** **0;** j **<** C**;** **++**j**)** **if** **(**j **!=** i **&&** Overlap**[**j**][**i**])** cnt**++;**

**for** **(int** j **=** **0;** j **<** C**;** **++**j**)** **{**

**if** **(**i **!=** j **&&** g**[**i**][**j**])** **{**

circleCrossCircle**(**c**[**i**],** c**[**j**]);**

**}**

**}**

//cnt表示覆盖次数超过cnt

**if** **(**E **==** **0)** **{**

Area**[**cnt**]** **+=** PI **\*** c**[**i**].**r **\*** c**[**i**].**r**;**

**}** **else** **{**

**double** counts **=** **0;**

sort**(**eve**,** eve **+** E**);**

eve**[**E**]** **=** eve**[0];**

**for** **(int** j **=** **0;** j **<** E**;** **++**j**)** **{**

cnt **+=** eve**[**j**].**add**;**

Area**[**cnt**]** **+=** cross**(**eve**[**j**].**p**,** eve**[**j **+** **1].**p**)** **\*** **.5;**

**double** theta **=** eve**[**j **+** **1].**ang **-** eve**[**j**].**ang**;**

**if** **(**theta **<** **0)** theta **+=** PI **\*** **2.;**

Area**[**cnt**]** **+=** theta **\*** c**[**i**].**r **\*** c**[**i**].**r **\*** **.5** **-** sin**(**theta**)** **\*** c**[**i**].**r **\*** c**[**i**].**r **\*** **.5;**

**}**

**}**

**}**

printf**(**"%.5f\n"**,** Area**[1]);**

**return** **0;**

**}**

## 三角形的心

//传入的参数point a,b,c; 三角形顶点

**double** area**(**point a**,**point b**,**point c**)** //面积

**{**

**return(**fabs**(**det**(**b**-**a**,**c**-**a**))/2);**

**}**

point barycenter**(**point a**,**point b**,**point c**)** //重心

**{**

**return(**point**((**a**.**x**+**b**.**x**+**c**.**x**)/3.0,(**a**.**y**+**b**.**y**+**c**.**y**)/3.0));**

**}**

point orthocenter**(**point a**,**point b**,**point c**)** //垂心

**{**

**double** d**,**dx**,**dy**;**

d**=(**c**.**x**-**b**.**x**)\*(**c**.**y**-**a**.**y**)-(**c**.**x**-**a**.**x**)\*(**c**.**y**-**b**.**y**);**

dx**=(**a**.**y**\*(**c**.**y**-**b**.**y**)+**a**.**x**\*(**c**.**x**-**b**.**x**))\*(**c**.**y**-**a**.**y**)-(**b**.**y**\*(**c**.**y**-**a**.**y**)+**b**.**x**\*(**c**.**x**-**a**.**x**))\*(**c**.**y**-**b**.**y**);**

dy**=(**c**.**x**-**b**.**x**)\*(**b**.**y**\*(**c**.**y**-**a**.**y**)+**b**.**x**\*(**c**.**x**-**a**.**x**))-(**c**.**x**-**a**.**x**)\*(**a**.**y**\*(**c**.**y**-**b**.**y**)+**a**.**x**\*(**c**.**x**-**b**.**x**));**

**return(**point**(**dx**/**d**,**dy**/**d**));**

**}**

point circumcenter**(**point a**,**point b**,**point c**)** //外心

**{**

**double** A**,**B**,**C**;**

A**=**dist**(**b**,**c**),**B**=**dist**(**a**,**c**),**C**=**dist**(**a**,**b**);**

**double** P**,**Q**;**

P**=(**SQR**(**A**)+**SQR**(**B**)+**SQR**(**C**))/2.0;**

Q**=1.0/(1/(**P**-**SQR**(**A**))+1/(**P**-**SQR**(**B**))+1/(**P**-**SQR**(**C**)));**

**double** R**=**sqrt**(**P**-**Q**)/2;** //R为外接圆半径，需要时可用，否则可删去

**double** d1**,**d2**,**d3**;**

d1**=**Q**/(**P**-**SQR**(**A**)),**d2**=**Q**/(**P**-**SQR**(**B**)),**d3**=**Q**/(**P**-**SQR**(**C**));**

**return((1-**d1**)/2.0\***a**+(1-**d2**)/2.0\***b**+(1-**d3**)/2.0\***c**);**

**}**

point incenter**(**point a**,**point b**,**point c**)**

**{**

**double** A**,**B**,**C**;**

A**=**dist**(**b**,**c**),**B**=**dist**(**a**,**c**),**C**=**dist**(**a**,**b**);**

**double** r**=2\***area**(**a**,**b**,**c**)/(**A**+**B**+**C**);** //r为内切圆半径，需要时可用 ，否则可删去

**return(**point**((**A**\***a**.**x**+**B**\***b**.**x**+**C**\***c**.**x**)/(**A**+**B**+**C**),(**A**\***a**.**y**+**B**\***b**.**y**+**C**\***c**.**y**)/(**A**+**B**+**C**)));**

**}**

## 最小覆盖球

**int** npoint**,** nouter**;**

Tpoint pt**[200000],** outer**[4],**res**;**

**double** radius**,**tmp**;**

**inline** **double** dist**(**Tpoint p1**,** Tpoint p2**)** **{**

**double** dx**=**p1**.**x**-**p2**.**x**,** dy**=**p1**.**y**-**p2**.**y**,** dz**=**p1**.**z**-**p2**.**z**;**

**return** **(** dx**\***dx **+** dy**\***dy **+** dz**\***dz **);**

**}**

**void** ball**()** **{**

Tpoint q**[3];** **double** m**[3][3],** sol**[3],** L**[3],** det**;**

**int** i**,**j**;**

res**.**x **=** res**.**y **=** res**.**z **=** radius **=** **0;**

**switch** **(** nouter **)** **{**

**case** **1:** res**=**outer**[0];** **break;**

**case** **2:**

res**.**x**=(**outer**[0].**x**+**outer**[1].**x**)/2;**

res**.**y**=(**outer**[0].**y**+**outer**[1].**y**)/2;**

res**.**z**=(**outer**[0].**z**+**outer**[1].**z**)/2;**

radius**=**dist**(**res**,** outer**[0]);**

**break;**

**case** **3:**

**for** **(**i**=0;** i**<2;** **++**i **)** **{**

q**[**i**].**x**=**outer**[**i**+1].**x**-**outer**[0].**x**;**

q**[**i**].**y**=**outer**[**i**+1].**y**-**outer**[0].**y**;**

q**[**i**].**z**=**outer**[**i**+1].**z**-**outer**[0].**z**;**

**}**

**for** **(**i**=0;** i**<2;** **++**i**)** **for(**j**=0;** j**<2;** **++**j**)**

m**[**i**][**j**]=**dot**(**q**[**i**],** q**[**j**])\*2;**

**for** **(**i**=0;** i**<2;** **++**i **)** sol**[**i**]=**dot**(**q**[**i**],** q**[**i**]);**

**if** **(**fabs**(**det**=**m**[0][0]\***m**[1][1]-**m**[0][1]\***m**[1][0])<**eps**)**

**return;**

L**[0]=(**sol**[0]\***m**[1][1]-**sol**[1]\***m**[0][1])/**det**;**

L**[1]=(**sol**[1]\***m**[0][0]-**sol**[0]\***m**[1][0])/**det**;**

res**.**x**=**outer**[0].**x**+**q**[0].**x**\***L**[0]+**q**[1].**x**\***L**[1];**

res**.**y**=**outer**[0].**y**+**q**[0].**y**\***L**[0]+**q**[1].**y**\***L**[1];**

res**.**z**=**outer**[0].**z**+**q**[0].**z**\***L**[0]+**q**[1].**z**\***L**[1];**

radius**=**dist**(**res**,** outer**[0]);**

**break;**

**case** **4:**

**for** **(**i**=0;** i**<3;** **++**i**)** **{**

q**[**i**].**x**=**outer**[**i**+1].**x**-**outer**[0].**x**;**

q**[**i**].**y**=**outer**[**i**+1].**y**-**outer**[0].**y**;**

q**[**i**].**z**=**outer**[**i**+1].**z**-**outer**[0].**z**;**

sol**[**i**]=**dot**(**q**[**i**],** q**[**i**]);**

**}**

**for** **(**i**=0;**i**<3;++**i**)**

**for(**j**=0;**j**<3;++**j**)** m**[**i**][**j**]=**dot**(**q**[**i**],**q**[**j**])\*2;**

det**=** m**[0][0]\***m**[1][1]\***m**[2][2]**

**+** m**[0][1]\***m**[1][2]\***m**[2][0]**

**+** m**[0][2]\***m**[2][1]\***m**[1][0]**

**-** m**[0][2]\***m**[1][1]\***m**[2][0]**

**-** m**[0][1]\***m**[1][0]\***m**[2][2]**

**-** m**[0][0]\***m**[1][2]\***m**[2][1];**

**if** **(** fabs**(**det**)<**eps **)** **return;**

**for** **(**j**=0;** j**<3;** **++**j**)** **{**

**for** **(**i**=0;** i**<3;** **++**i**)** m**[**i**][**j**]=**sol**[**i**];**

L**[**j**]=(** m**[0][0]\***m**[1][1]\***m**[2][2]**

**+** m**[0][1]\***m**[1][2]\***m**[2][0]**

**+** m**[0][2]\***m**[2][1]\***m**[1][0]**

**-** m**[0][2]\***m**[1][1]\***m**[2][0]**

**-** m**[0][1]\***m**[1][0]\***m**[2][2]**

**-** m**[0][0]\***m**[1][2]\***m**[2][1]**

**)** **/** det**;**

**for** **(**i**=0;** i**<3;** **++**i**)**

m**[**i**][**j**]=**dot**(**q**[**i**],** q**[**j**])\*2;**

**}**

res**=**outer**[0];**

**for** **(**i**=0;** i**<3;** **++**i **)** **{**

res**.**x **+=** q**[**i**].**x **\*** L**[**i**];**

res**.**y **+=** q**[**i**].**y **\*** L**[**i**];**

res**.**z **+=** q**[**i**].**z **\*** L**[**i**];**

**}**

radius**=**dist**(**res**,** outer**[0]);**

**}**

**}**

**void** minball**(int** n**)** **{**

ball**();**

//printf("(%.3f,%.3f,%.3f) %.3f\n", res.x,res.y,res.z,radius);

**if** **(** nouter**<4** **)**

**for** **(int** i**=0;** i**<**n**;** **++**i**)**

**if** **(**dist**(**res**,** pt**[**i**])-**radius**>**eps**)** **{**

outer**[**nouter**]=**pt**[**i**];**

**++**nouter**;**

minball**(**i**);**

**--**nouter**;**

**if** **(**i**>0)** **{**

Tpoint Tt **=** pt**[**i**];**

memmove**(&**pt**[1],** **&**pt**[0],** **sizeof(**Tpoint**)\***i**);**

pt**[0]=**Tt**;**

**}**

**}**

**}**

**int** main**(){**

scanf**(**"%d"**,&**npoint**);**

**for** **(int** i**=0;**i**<**npoint**;**i**++)**

scanf**(**"%lf%lf%lf"**,&**pt**[**i**].**x**,&**pt**[**i**].**y**,&**pt**[**i**].**z**);**

random\_shuffle**(**pt**,**pt**+**npoint**);**

radius**=-1;**

**for** **(int** i**=0;**i**<**npoint**;**i**++){**

**if** **(**dist**(**res**,**pt**[**i**])-**radius**>**eps**){**

nouter**=1;**

outer**[0]=**pt**[**i**];**

minball**(**i**);**

**}**

**}**

printf**(**"%.3f\n"**,**sqrt**(**radius**));**

**}**

## 经纬度求球面最短距离

//lati 为纬度 longi为经度 R为半径

**double** Dist**(double** lati1**,double** longi1**,double** lati2**,double** longi2**,double** R**)**

**{**

**double** pi**=**acos**(-1.0);**

lati1**\*=**pi**/180,**longi1**\*=**pi**/180,**lati2**\*=**pi**/180,**longi2**\*=**pi**/180;**

**double** x1**=**cos**(**lati1**)\***sin**(**longi1**),**y1**=**cos**(**lati1**)\***cos**(**longi1**),**z1**=**sin**(**lati1**);**

**double** x2**=**cos**(**lati2**)\***sin**(**longi2**),**y2**=**cos**(**lati2**)\***cos**(**longi2**),**z2**=**sin**(**lati2**);**

**double** theta**=**acos**(**x1**\***x2**+**y1**\***y2**+**z1**\***z2**);**

**return(**R**\***theta**);**

**}**

## 长方体表面两点最短距离

**int** r**;**

**void** turn**(int** i**,** **int** j**,** **int** x**,** **int** y**,** **int** z**,int** x0**,** **int** y0**,** **int** L**,** **int** W**,** **int** H**)** **{**

**if** **(**z**==0)** **{**

**int** R **=** x**\***x**+**y**\***y**;**

**if** **(**R**<**r**)** r**=**R**;**

**}**

**else{**

**if(**i**>=0** **&&** i**<** **2)**

turn**(**i**+1,** j**,** x0**+**L**+**z**,** y**,** x0**+**L**-**x**,** x0**+**L**,** y0**,** H**,** W**,** L**);**

**if(**j**>=0** **&&** j**<** **2)**

turn**(**i**,** j**+1,** x**,** y0**+**W**+**z**,** y0**+**W**-**y**,** x0**,** y0**+**W**,** L**,** H**,** W**);**

**if(**i**<=0** **&&** i**>-2)**

turn**(**i**-1,** j**,** x0**-**z**,** y**,** x**-**x0**,** x0**-**H**,** y0**,** H**,** W**,** L**);**

**if(**j**<=0** **&&** j**>-2)**

turn**(**i**,** j**-1,** x**,** y0**-**z**,** y**-**y0**,** x0**,** y0**-**H**,** L**,** H**,** W**);**

**}**

**}**

**int** main**(){**

**int** L**,** H**,** W**,** x1**,** y1**,** z1**,** x2**,** y2**,** z2**;**

cin **>>** L **>>** W **>>** H **>>** x1 **>>** y1 **>>** z1 **>>** x2 **>>** y2 **>>** z2**;**

**if** **(**z1**!=0** **&&** z1**!=**H**)**

**if** **(**y1**==0** **||** y1**==**W**)**

swap**(**y1**,**z1**),** std**::**swap**(**y2**,**z2**),** std**::**swap**(**W**,**H**);**

**else**

swap**(**x1**,**z1**),** std**::**swap**(**x2**,**z2**),** std**::**swap**(**L**,**H**);**

**if** **(**z1**==**H**)** z1**=0,** z2**=**H**-**z2**;**

r**=0x3fffffff;** turn**(0,0,**x2**-**x1**,**y2**-**y1**,**z2**,-**x1**,-**y1**,**L**,**W**,**H**);**

cout**<<**r**<<**endl**;**

**return** **0;**

**}**

## 最大团

Int g**[][]**为图的邻接矩阵。

MC**(**V**)**表示点集V的最大团

令Si**={**vi**,** vi**+1,** **...,** vn**},** mc**[**i**]**表示MC**(**Si**)**

倒着算mc**[**i**]**，那么显然MC**(**V**)=**mc**[1]**

此外有mc**[**i**]=**mc**[**i**+1]** **or** mc**[**i**]=**mc**[**i**+1]+1**

**void** init**(){**

**int** i**,** j**;**

**for** **(**i**=1;** i**<=**n**;** **++**i**)** **for** **(**j**=1;** j**<=**n**;** **++**j**)** scanf**(**"%d"**,** **&**g**[**i**][**j**]);**

**}**

**void** dfs**(int** size**){**

**int** i**,** j**,** k**;**

**if** **(**len**[**size**]==0)** **{**

**if** **(**size**>**ans**)** **{**

ans**=**size**;** found**=true;**

**}**

**return;**

**}**

**for** **(**k**=0;** k**<**len**[**size**]** **&&** **!**found**;** **++**k**)** **{**

**if** **(**size**+**len**[**size**]-**k**<=**ans**)** **break;**

i**=**list**[**size**][**k**];**

**if** **(**size**+**mc**[**i**]<=**ans**)** **break;**

**for** **(**j**=**k**+1,** len**[**size**+1]=0;** j**<**len**[**size**];** **++**j**)**

**if** **(**g**[**i**][**list**[**size**][**j**]])** list**[**size**+1][**len**[**size**+1]++]=**list**[**size**][**j**];**

dfs**(**size**+1);**

**}**

**}**

**void** work**(){**

**int** i**,** j**;**

mc**[**n**]=**ans**=1;**

**for** **(**i**=**n**-1;** i**;** **--**i**)** **{**

found**=false;**

len**[1]=0;**

**for** **(**j**=**i**+1;** j**<=**n**;** **++**j**)** **if** **(**g**[**i**][**j**])** list**[1][**len**[1]++]=**j**;**

dfs**(1);**

mc**[**i**]=**ans**;**

**}**

**}**

## 极大团计数

Bool g**[][]** 为图的邻接矩阵，图点的标号由1至n。

**void** dfs**(int** size**){**

**int** i**,** j**,** k**,** t**,** cnt**,** best **=** **0;**

**bool** bb**;**

**if** **(**ne**[**size**]==**ce**[**size**]){**

**if** **(**ce**[**size**]==0)** **++**ans**;**

**return;**

**}**

**for** **(**t**=0,** i**=1;** i**<=**ne**[**size**];** **++**i**)** **{**

**for** **(**cnt**=0,** j**=**ne**[**size**]+1;** j**<=**ce**[**size**];** **++**j**)**

**if** **(!**g**[**list**[**size**][**i**]][**list**[**size**][**j**]])** **++**cnt**;**

**if** **(**t**==0** **||** cnt**<**best**)** t**=**i**,** best**=**cnt**;**

**}**

**if** **(**t **&&** best**<=0)** **return;**

**for** **(**k**=**ne**[**size**]+1;** k**<=**ce**[**size**];** **++**k**)** **{**

**if** **(**t**>0){**

**for** **(**i**=**k**;** i**<=**ce**[**size**];** **++**i**)** **if** **(!**g**[**list**[**size**][**t**]][**list**[**size**][**i**]])** **break;**

swap**(**list**[**size**][**k**],** list**[**size**][**i**]);**

**}**

i**=**list**[**size**][**k**];**

ne**[**size**+1]=**ce**[**size**+1]=0;**

**for** **(**j**=1;** j**<**k**;** **++**j**)if** **(**g**[**i**][**list**[**size**][**j**]])** list**[**size**+1][++**ne**[**size**+1]]=**list**[**size**][**j**];**

**for** **(**ce**[**size**+1]=**ne**[**size**+1],** j**=**k**+1;** j**<=**ce**[**size**];** **++**j**)**

**if** **(**g**[**i**][**list**[**size**][**j**]])** list**[**size**+1][++**ce**[**size**+1]]=**list**[**size**][**j**];**

dfs**(**size**+1);**

**++**ne**[**size**];**

**--**best**;**

**for** **(**j**=**k**+1,** cnt**=0;** j**<=**ce**[**size**];** **++**j**)** **if** **(!**g**[**i**][**list**[**size**][**j**]])** **++**cnt**;**

**if** **(**t**==0** **||** cnt**<**best**)** t**=**k**,** best**=**cnt**;**

**if** **(**t **&&** best**<=0)** **break;**

**}**

**}**

**void** work**(){**

**int** i**;**

ne**[0]=0;** ce**[0]=0;**

**for** **(**i**=1;** i**<=**n**;** **++**i**)** list**[0][++**ce**[0]]=**i**;**

ans**=0;**

dfs**(0);**

**}**

## KM

**int** n**,**b**[**MAXN**],**dx**[**MAXN**],**dy**[**MAXN**],**slack**[**MAXN**],**a**[**MAXN**][**MAXN**];**

**bool** f**[**MAXN**],**g**[**MAXN**];**

**bool** hungary**(int** x**)**

**{**

**if** **(!**x**)**

**return(true);**

f**[**x**]=true;**

**for** **(int** i**=1;**i**<=**n**;**i**++)**

**{**

**if** **(**g**[**i**])**

**continue;**

**int** t**=**dx**[**x**]+**dy**[**i**]-**a**[**x**][**i**];**

**if** **(!**t**)**

**{**

g**[**i**]=true;**

**if** **(**hungary**(**b**[**i**]))**

**{**

b**[**i**]=**x**;**

**return(true);**

**}**

**}**

**else** **if** **(**t**<**slack**[**i**])**

slack**[**i**]=**t**;**

**}**

**return(false);**

**}**

**int** main**()**

**{**

memset**(**dx**,0,sizeof(**dx**));**

memset**(**dy**,0,sizeof(**dy**));**

scanf**(**"%d"**,&**n**);**

**for** **(int** i**=1;**i**<=**n**;**i**++)**

**for** **(int** j**=1;**j**<=**n**;**j**++)**

**{**

scanf**(**"%d"**,&**a**[**i**][**j**]);**

**if** **(**a**[**i**][**j**]>**dx**[**i**])**

dx**[**i**]=**a**[**i**][**j**];**

**}**

**for** **(int** i**=1;**i**<=**n**;**i**++)**

**{**

memset**(**slack**,63,sizeof(**slack**));**

memset**(**f**,0,sizeof(**f**));**

memset**(**g**,0,sizeof(**g**));**

**while** **(!**hungary**(**i**))**

**{**

**int** d**=**inf**;**

**for** **(int** i**=1;**i**<=**n**;**i**++)**

**if** **(!**g**[**i**]** **&&** slack**[**i**]<**d**)**

d**=**slack**[**i**];**

**for** **(int** i**=1;**i**<=**n**;**i**++)**

**{**

**if** **(**f**[**i**])**

dx**[**i**]-=**d**;**

**if** **(**g**[**i**])**

dy**[**i**]+=**d**;**

**}**

memset**(**f**,0,sizeof(**f**));**

memset**(**g**,0,sizeof(**g**));**

**}**

**}**

**}**

## 无向图最小割

**int** cost**[**maxn**][**maxn**],** seq**[**maxn**],** len**[**maxn**],** n**,** m**,** pop**,** ans**;**

**bool** used**[**maxn**];**

**void** Init**()** **{**

**int** i**,** j**,** a**,** b**,** c**;**

**for** **(**i **=** **0;** i **<** n**;** i**++)**

**for** **(**j **=** **0;** j **<** n**;** j**++)**

cost**[**i**][**j**]** **=** **0;**

**for** **(**i **=** **0;** i **<** m**;** i**++)** **{**

scanf**(**"%d %d %d"**,** **&**a**,** **&**b**,** **&**c**);**

cost**[**a**][**b**]** **+=** c**;**

cost**[**b**][**a**]** **+=** c**;**

**}**

pop **=** n**;**

**for** **(**i **=** **0;** i **<** n**;** i**++)**

seq**[**i**]** **=** i**;**

**}**

**void** Work**()** **{**

ans **=** inf**;**

**int** i**,** j**,** k**,** l**,** mm**,** sum**,** pk**;**

**while** **(**pop **>** **1)** **{**

**for** **(**i **=** **1;** i **<** pop**;** i**++)**

used**[**seq**[**i**]]** **=** **0;**

used**[**seq**[0]]** **=** **1;**

**for** **(**i **=** **1;** i **<** pop**;** i**++)**

len**[**seq**[**i**]]** **=** cost**[**seq**[0]][**seq**[**i**]];**

pk **=** **0;**

mm **=** **-**inf**;**

k **=** **-1;**

**for** **(**i **=** **1;** i **<** pop**;** i**++)**

**if** **(**len**[**seq**[**i**]]** **>** mm**)** **{**

mm **=** len**[**seq**[**i**]];**

k **=** i**;**

**}**

**for** **(**i **=** **1;** i **<** pop**;** i**++)** **{**

used**[**seq**[**l **=** k**]]** **=** **1;**

**if** **(**i **==** pop **-** **2)** pk **=** k**;**

**if** **(**i **==** pop **-** **1)** **break;**

mm **=** **-**inf**;**

**for** **(**j **=** **1;** j **<** pop**;** j**++)**

**if** **(!**used**[**seq**[**j**]])** **{**

**if** **((**len**[**seq**[**j**]]** **+=** cost**[**seq**[**l**]][**seq**[**j**]])** **>** mm**)** **{**

mm **=** len**[**seq**[**j**]];**

k **=** j**;**

**};**

**}**

**}**

sum **=** **0;**

**for** **(**i **=** **0;** i **<** pop**;** i**++)**

**if** **(**i **!=** k**)** sum **+=** cost**[**seq**[**k**]][**seq**[**i**]];**

ans **=** min**(**ans**,** sum**);**

**for** **(**i **=** **0;** i **<** pop**;** i**++)**

cost**[**seq**[**k**]][**seq**[**i**]]** **=** cost**[**seq**[**i**]][**seq**[**k**]]** **+=** cost**[**seq**[**pk**]][**seq**[**i**]];**

seq**[**pk**]** **=** seq**[--**pop**];**

**}**

printf**(**"%d\n"**,** ans**);**

**}**

## 带花树

vector**<int>** link**[**maxn**];**

**int** n**,** match**[**maxn**],** Queue**[**maxn**],** head**,** tail**;**

**int** pred**[**maxn**],** base**[**maxn**],** start**,** finish**,** newbase**;**

**bool** InQueue**[**maxn**],** InBlossom**[**maxn**];**

**void** push**(int** u**)** **{**

Queue**[**tail**++]** **=** u**;** InQueue**[**u**]** **=** **true;**

**}**

**int** pop**()** **{**

**return** Queue**[**head**++];**

**}**

**int** FindCommonAncestor**(int** u**,** **int** v**)** **{**

**bool** InPath**[**maxn**];**

**for** **(int** i **=** **0;** i **<** n**;** i**++)**

InPath**[**i**]** **=** **0;**

**while(true)** **{**

u **=** base**[**u**];**

InPath**[**u**]** **=** **true;**

**if(**u **==** start**)** **break;**

u **=** pred**[**match**[**u**]];**

**}**

**while(true)** **{**

v **=** base**[**v**];**

**if(**InPath**[**v**])** **break;**

v **=** pred**[**match**[**v**]];**

**}**

**return** v**;**

**}**

**void** ResetTrace**(int** u**)** **{**

**int** v**;**

**while(**base**[**u**]** **!=** newbase**)** **{**

v **=** match**[**u**];**

InBlossom**[**base**[**u**]]** **=** InBlossom**[**base**[**v**]]** **=** **true;**

u **=** pred**[**v**];**

**if(**base**[**u**]** **!=** newbase**)** pred**[**u**]** **=** v**;**

**}**

**}**

**void** BlossomContract**(int** u**,** **int** v**)** **{**

newbase **=** FindCommonAncestor**(**u**,** v**);**

**for** **(int** i **=** **0;** i **<** n**;** i**++)**

InBlossom**[**i**]** **=** **0;**

ResetTrace**(**u**);** ResetTrace**(**v**);**

**if(**base**[**u**]** **!=** newbase**)** pred**[**u**]** **=** v**;**

**if(**base**[**v**]** **!=** newbase**)** pred**[**v**]** **=** u**;**

**for(int** i **=** **0;** i **<** n**;** **++**i**)**

**if(**InBlossom**[**base**[**i**]])** **{**

base**[**i**]** **=** newbase**;**

**if(!**InQueue**[**i**])** push**(**i**);**

**}**

**}**

**bool** FindAugmentingPath**(int** u**)** **{**

**bool** found **=** **false;**

**for(int** i **=** **0;** i **<** n**;** **++**i**)** pred**[**i**]** **=** **-1,** base**[**i**]** **=** i**;**

**for** **(int** i **=** **0;** i **<** n**;** i**++)**

InQueue**[**i**]** **=** **0;**

start **=** u**;** finish **=** **-1;**

head **=** tail **=** **0;**

push**(**start**);**

**while(**head **<** tail**)** **{**

**int** u **=** pop**();**

**for(int** i **=** link**[**u**].**size**()** **-** **1;** i **>=** **0;** i**--)** **{**

**int** v **=** link**[**u**][**i**];**

**if(**base**[**u**]** **!=** base**[**v**]** **&&** match**[**u**]** **!=** v**)**

**if(**v **==** start **||** **(**match**[**v**]** **>=** **0** **&&** pred**[**match**[**v**]]** **>=** **0))**

BlossomContract**(**u**,** v**);**

**else** **if(**pred**[**v**]** **==** **-1)** **{**

pred**[**v**]** **=** u**;**

**if(**match**[**v**]** **>=** **0)**

push**(**match**[**v**]);**

**else** **{**

finish **=** v**;**

**return** **true;**

**}**

**}**

**}**

**}**

**return** found**;**

**}**

**void** AugmentPath**()** **{**

**int** u**,** v**,** w**;**

u **=** finish**;**

**while(**u **>=** **0)** **{**

v **=** pred**[**u**];**

w **=** match**[**v**];**

match**[**v**]** **=** u**;**

match**[**u**]** **=** v**;**

u **=** w**;**

**}**

**}**

**void** FindMaxMatching**()** **{**

**for(int** i **=** **0;** i **<** n**;** **++**i**)** match**[**i**]** **=** **-1;**

**for(int** i **=** **0;** i **<** n**;** **++**i**)**

**if(**match**[**i**]** **==** **-1)**

**if(**FindAugmentingPath**(**i**))**

AugmentPath**();**

**}**

## 动态最小生成树

/\* 动态最小生成树 Q(logQ)^2

(qx[i], qy[i])表示将编号为qx[i]的边的权值改为qy[i]

删除一条边相当于将其权值改为\infinity

加入一条边相当于将其权值从\infinity变成某个值 \*/

**const** **int** qsize **=** maxm **+** **3\***maxq**;**

**int** x**[**qsize**],**y**[**qsize**],**z**[**qsize**];**

**int** qx**[**maxq**],**qy**[**maxq**],** n**,**m**,**Q**;**

**void** init**()**

**{**

scanf**(**"%d%d"**,&**n**,&**m**);**

**for(int** i**=0;**i**<**m**;**i**++)**

scanf**(**"%d%d%d"**,**x**+**i**,**y**+**i**,**z**+**i**);**

scanf**(**"%d"**,&**Q**);**

**for(int** i**=0;**i**<**Q**;**i**++)**

**{**

scanf**(**"%d%d"**,**qx**+**i**,**qy**+**i**);**

qx**[**i**]--;**

**}**

**}**

**int** a**[**maxn**];**

**int** **\***tz**;**

**int** find**(** **int** x **)**

**{**

**int** root **=** x**;**

**while(** a**[**root**]** **)** root **=** a**[**root**];**

**int** next**;**

**while(** next **=** a**[**x**]** **)**

**{**

a**[**x**]** **=** root**;**

x **=** next**;**

**}**

**return** root**;**

**}**

**inline** **bool** cmp**(** **const** **int** **&**a**,const** **int** **&**b **)**

**{**

**return** tz**[**a**]** **<** tz**[**b**];**

**}**

**int** kx**[**maxn**],**ky**[**maxn**],**kt**;**

**int** vd**[**maxn**],**id**[**maxm**];**

**int** app**[**maxm**];**

**bool** extra**[**maxm**];**

**long** **long** printState**(** **int** **\***qx**,int** **\***qy**,int** Q**,int** n**,int** **\***x**,int** **\***y**,int** **\***z**,int** m**,long** **long** ans **)**

**{**

printf**(**"%d %d\n"**,**n**,**m**);**

**for(int** i**=0;**i**<**m**;**i**++)** printf**(**"%d %d %d\n"**,**x**[**i**],**y**[**i**],**z**[**i**]);**

printf**(**"Q = %d\n"**,**Q**);**

**for(int** i**=0;**i**<**Q**;**i**++)** printf**(**"%d %d\n"**,**qx**[**i**],**qy**[**i**]);**

**return** ans**;**

**}**

**void** solve**(** **int** **\***qx**,int** **\***qy**,int** Q**,int** n**,int** **\***x**,int** **\***y**,int** **\***z**,int** m**,long** **long** ans **)**

**{**

**if(**Q**==1)**

**{**

**for(int** i**=1;**i**<=**n**;**i**++)** a**[**i**]** **=** **0;**

z**[** qx**[0]** **]** **=** qy**[0];**

**for(int** i**=0;**i**<**m**;**i**++)** id**[**i**]** **=** i**;**tz **=** z**;**

sort**(**id**,**id**+**m**,**cmp**);**

**int** ri**,**rj**;**

**for(int** i**=0;**i**<**m**;**i**++)**

**{**

ri **=** find**(** x**[**id**[**i**]]** **);**

rj **=** find**(** y**[**id**[**i**]]** **);**

**if(**ri**!=**rj**)**

**{**

ans**+=**z**[**id**[**i**]];**

a**[**ri**]** **=** rj**;**

**}**

**}**

printf**(**"%I64d\n"**,**ans**);**

**return;**

**}**

**int** ri**,**rj**;**

//contract

kt **=** **0;**

**for(int** i**=1;**i**<=**n**;**i**++)** a**[**i**]** **=** **0;**

**for(int** i**=0;**i**<**Q**;**i**++)**

**{**

ri **=** find**(** x**[**qx**[**i**]]** **);**

rj **=** find**(** y**[**qx**[**i**]]** **);**

**if(**ri**!=**rj**)** a**[**ri**]** **=** rj**;**

**}**

**int** tm **=** **0;**

**for(int** i**=0;**i**<**m**;**i**++)** extra**[**i**]** **=** **true;**

**for(int** i**=0;**i**<**Q**;**i**++)** extra**[** qx**[**i**]** **]** **=** **false;**

**for(int** i**=0;**i**<**m**;**i**++)** **if(**extra**[**i**])**

id**[**tm**++]** **=** i**;**

tz **=** z**;**

sort**(** id**,**id**+**tm**,**cmp **);**

**for(int** i**=0;**i**<**tm**;**i**++)**

**{**

ri **=** find**(** x**[**id**[**i**]]** **);**

rj **=** find**(** y**[**id**[**i**]]** **);**

**if(**ri**!=**rj**)**

**{**

a**[**ri**]** **=** rj**;**

ans **+=** z**[**id**[**i**]];**

kx**[**kt**]** **=** x**[**id**[**i**]];**

ky**[**kt**]** **=** y**[**id**[**i**]];**

kt**++;**

**}**

**}**

**for(int** i**=1;**i**<=**n**;**i**++)** a**[**i**]** **=** **0;**

**for(int** i**=0;**i**<**kt**;**i**++)**

a**[** find**(** kx**[**i**]** **)** **]** **=** find**(** ky**[**i**]** **);**

**int** n2 **=** **0;**

**for(int** i**=1;**i**<=**n**;**i**++)** **if(**a**[**i**]==0)**

vd**[**i**]** **=** **++**n2**;**

**for(int** i**=1;**i**<=**n**;**i**++)** **if(**a**[**i**])**

vd**[**i**]** **=** vd**[**find**(**i**)];**

**int** **\***Nx **=** x **+** m**;**

**int** **\***Ny **=** y **+** m**;**

**int** **\***Nz **=** z **+** m**;**

**int** m2 **=** **0;**

**for(int** i**=0;**i**<**m**;**i**++)** app**[**i**]** **=** **-1;**

**for(int** i**=0;**i**<**Q**;**i**++)** **if(** app**[**qx**[**i**]]==-1** **)**

**{**

Nx**[**m2**]** **=** vd**[** x**[** qx**[**i**]** **]** **];**

Ny**[**m2**]** **=** vd**[** y**[** qx**[**i**]** **]** **];**

Nz**[**m2**]** **=** z**[** qx**[**i**]** **];**

app**[**qx**[**i**]]** **=** m2**;**

m2**++;**

**}**

**for(int** i**=0;**i**<**Q**;**i**++)**

**{**

z**[** qx**[**i**]** **]** **=** qy**[**i**];**

qx**[**i**]** **=** app**[**qx**[**i**]];**

**}**

**for(int** i**=1;**i**<=**n2**;**i**++)** a**[**i**]** **=** **0;**

**for(int** i**=0;**i**<**tm**;**i**++)**

**{**

ri **=** find**(** vd**[** x**[**id**[**i**]]** **]** **);**

rj **=** find**(** vd**[** y**[**id**[**i**]]** **]** **);**

**if(**ri**!=**rj**)**

**{**

a**[**ri**]** **=** rj**;**

Nx**[**m2**]** **=** vd**[** x**[**id**[**i**]]** **];**

Ny**[**m2**]** **=** vd**[** y**[**id**[**i**]]** **];**

Nz**[**m2**]** **=** z**[**id**[**i**]];**

m2**++;**

**}**

**}**

**int** mid **=** Q**/2;**

solve**(** qx**,**qy**,**mid**,**n2**,**Nx**,**Ny**,**Nz**,**m2**,**ans **);**

solve**(** qx**+**mid**,**qy**+**mid**,**Q**-**mid**,**n2**,**Nx**,**Ny**,**Nz**,**m2**,**ans **);**

**}**

**void** work**()**

**{**

**if(**Q**)** solve**(** qx**,**qy**,**Q**,**n**,**x**,**y**,**z**,**m**,0** **);**

**}**

**int** main**()**

**{**

freopen**(**"input.txt"**,**"r"**,**stdin**);**

init**();**

work**();**

**return** **0;**

**}**

## Hopcroft

**int** from**[1010],** wh**[1010],** g**[1010];**

**int** num**[100010],** nxt**[100010],** tot**;**

**int** n**,** m**,** ans**,** h**,** t**,** q**[1010],** dx**[1010],** dy**[1010];**

**bool** bfs**(){**

**bool** ret**=false;**

h**=0;** t**=0;**

**for** **(** **int** i**=0;** i**<**n**;** i**++** **)**

**if** **(** wh**[**i**]==-1** **){**

t**++;** q**[**t**]=**i**;**

**}**

memset**(** dx**,** **0,** **sizeof(** dx **)** **);**

memset**(** dy**,** **0,** **sizeof(** dy **)** **);**

**while** **(** h**<**t **){**

h**++;**

**for** **(** **int** i**=**g**[**q**[**h**]];** i**!=0;** i**=**nxt**[**i**]** **)**

**if** **(** dy**[**num**[**i**]]==0** **){**

dy**[**num**[**i**]]=**dx**[**q**[**h**]]+1;**

**if** **(** from**[**num**[**i**]]==-1** **)**

ret**=true;**

**else** **{**

dx**[**from**[**num**[**i**]]]=**dx**[**q**[**h**]]+2;**

t**++;** q**[**t**]=**from**[**num**[**i**]];**

**}**

**}**

**}**

**return** ret**;**

**}**

**bool** dfs**(** **int** x **){**

**for** **(** **int** i**=**g**[**x**];** i**!=0;** i**=**nxt**[**i**]** **){**

**if** **(** dy**[**num**[**i**]]==**dx**[**x**]+1** **){**

dy**[**num**[**i**]]=0;**

**if** **(** from**[**num**[**i**]]==-1** **||** dfs**(** from**[**num**[**i**]]** **)** **){**

wh**[**x**]=**num**[**i**];** from**[**num**[**i**]]=**x**;** **return** **true;**

**}**

**}**

**}**

**return** **false;**

**}**

**void** hopcroft**(){**

memset**(** from**,** **-1,** **sizeof(** from **)** **);**

memset**(** wh**,** **-1,** **sizeof(** wh **)** **);**

**while** **(** bfs**()** **){**

**for** **(** **int** i**=0;** i**<**n**;** i**++** **)**

**if** **(** wh**[**i**]==-1** **&&** dfs**(**i**)** **)** ans**++;**

**}**

**}**

**void** insert**(** **int** x**,** **int** y **){**

tot**++;** num**[**tot**]=**y**;** nxt**[**tot**]=**g**[**x**];** g**[**x**]=**tot**;**

**}**

**int** main**(){**

**while** **(** scanf**(**"%d %d"**,** **&**n**,** **&**m**)==2** **){**

tot**=0;**

memset**(** g**,** **0,** **sizeof(** g **)** **);**

**for** **(** **int** i**=0;** i**<**n**;** i**++** **){**

**int** x**;**

scanf**(**"%d"**,** **&**x**);**

**for** **(** **int** j**=0;** j**<**x**;** j**++** **){**

**int** y**;**

scanf**(**"%d"**,** **&**y**);**

y**--;**

insert**(** i**,** y **);**

**}**

**}**

ans**=0;**

hopcroft**();**

printf**(**"%d\n"**,** ans**);**

**}**

**}**

## 素数判定

**int** isprime**(long** **long** n**)** **{**

**if(**n**<2)** **return** **0;**

**if(**n**<4)** **return** **1;**

**if(**strong\_pseudo\_primetest**(**n**,2)==0)** **return** **0;**

**if(**strong\_pseudo\_primetest**(**n**,3)==0)** **return** **0;**

**if(**n**<1373653LL)** **return** **1;**

**if(**strong\_pseudo\_primetest**(**n**,5)==0)** **return** **0;**

**if(**n**<25326001LL)** **return** **1;**

**if(**strong\_pseudo\_primetest**(**n**,7)==0)** **return** **0;**

**if(**n**==3215031751LL)** **return** **0;**

**if(**n**<25000000000LL)** **return** **1;**

**if(**strong\_pseudo\_primetest**(**n**,11)==0)** **return** **0;**

**if(**n**<2152302898747LL)** **return** **1;**

**if(**strong\_pseudo\_primetest**(**n**,13)==0)** **return** **0;**

**if(**n**<3474749660383LL)** **return** **1;**

**if(**strong\_pseudo\_primetest**(**n**,17)==0)** **return** **0;**

**if(**n**<341550071728321LL)** **return** **1;**

**if(**strong\_pseudo\_primetest**(**n**,19)==0)** **return** **0;**

**if(**strong\_pseudo\_primetest**(**n**,23)==0)** **return** **0;**

**if(**strong\_pseudo\_primetest**(**n**,29)==0)** **return** **0;**

**if(**strong\_pseudo\_primetest**(**n**,31)==0)** **return** **0;**

**if(**strong\_pseudo\_primetest**(**n**,37)==0)** **return** **0;**

**return** **1;**

**}**

## 启发式分解

**int** ansn**;**

LL ans**[1000];**

LL mod\_mul**(**LL x**,**LL y**,**LL n**){**

**long** **long** d **=** **(long** **long)((long** **double)**x**\***y**/**n**);**

d**=**x**\***y**-**n**\***d**;**

**while** **(**d **<** **0)** d **+=** n**;**

**while** **(**d **>=** n**)** d **-=** n**;**

**return** d**;**

**}**

LL myrand**(){**

LL a**=**rand**();**

a**\*=**rand**();**

**return** abs**(**a**);**

**}**

LL mod\_exp**(**LL a**,**LL x**,**LL n**){**

LL ret**=1;**

**while(**x**){**

**if(**x**&1)** ret**=**mod\_mul**(**ret**,**a**,**n**);**

a**=**mod\_mul**(**a**,**a**,**n**);**

x**>>=1;**

**}**

**return** ret**;**

**}**

LL gcd**(**LL x**,**LL y**){**

LL q**;**

**while(1){**

**if(!**y**)** **return** abs**(**x**);**

q**=**x**,**x**=**y**,**y**=**q**%**y**;**

**}**

**}**

**bool** Rabin\_Miller**(**LL n**){**

LL k**=0,**i**,**j**,**m**,**a**;**

**if(**n**<2)** **return** **0;**

**if(**n**==2)** **return** **1;**

**if(!(**n**&1))** **return** **0;**

m**=**n**-1;**

**while(!(**m**&1))** m**>>=1,**k**++;**

**for(**i**=0;**i**<20;**i**++){**

a**=**myrand**()%(**n**-2)+2;**

a**=**mod\_exp**(**a**,**m**,**n**);**

**if(**a**==1)** **continue;**

**for(**j**=0;**j**<**k**;**j**++){**

**if(**a**==**n**-1)**

**break;**

a**=**mod\_mul**(**a**,**a**,**n**);**

**}**

**if(**j**<**k**)** **continue;**

**return** **0;**

**}**

**return** **1;**

**}**

LL func**(**LL x**,**LL n**){**

**return** **(**mod\_mul**(**x**,**x**,**n**)+1)%**n**;**

**}**

LL Pollard**(**LL n**){**

LL i**,**x**,**y**,**p**;**

**if(**Rabin\_Miller**(**n**))** **return** n**;**

**if(!(**n**&1))** **return** **2;**

**for(**i**=1;**i**<20;**i**++){**

x**=**i**;**

y**=**func**(**x**,**n**);**

p**=**gcd**(**y**-**x**,**n**);**

**while(**p**==1){**

x**=**func**(**x**,**n**);**

y**=**func**(**func**(**y**,**n**),**n**);**

p**=**gcd**((**y**-**x**+**n**)%**n**,**n**)%**n**;**

**}**

**if(**p**==0||**p**==**n**)**

**continue;**

**return** p**;**

**}**

**}**

**void** factor**(**LL n**){**

LL x**;**

x**=**Pollard**(**n**);**

**if(**x**==**n**){**

ans**[**ansn**++]=**x**;**

**return;**

**}**

factor**(**x**);**

factor**(**n**/**x**);**

**}**

**void** output**(){**

**int** i**,**j**;**

LL tmp**;**

**for(**i**=0;**i**<**ansn**;**i**++)**

**for(**j**=**i**+1;**j**<**ansn**;**j**++)**

**if(**ans**[**i**]>**ans**[**j**]){**

tmp**=**ans**[**i**];**

ans**[**i**]=**ans**[**j**];**

ans**[**j**]=**tmp**;**

**}**

**for** **(**i **=** **0;** i **<** ansn**;** i **+=** j**)** **{**

**for** **(**j **=** **0;** i **+** j **<** ansn **&&** ans**[**i**]** **==** ans**[**i **+** j**];** j**++);**

**if** **(**i**)** cout **<<** " \*"**;**

cout **<<** " " **<<** ans**[**i**];**

**if** **(**j **>** **1)** cout **<<** "^" **<<** j**;**

**}**

cout **<<** endl**;**

**}**

**int** main**(){**

LL n**;**

srand**((unsigned)**time**(NULL));**

**int** tt**;**

scanf**(**"%d"**,** **&**tt**);**

**while(**tt**--){**

cin **>>** n**;**

**if(**n**==1){**

cout**<<**"1 = 1"**<<**endl**;**

**continue;**

**}**

**if(**n**<0)**

**break;**

ansn**=0;**

factor**(**n**);**

cout **<<** n **<<** " ="**;**

output**();**

**}**

**return** **0;**

**}**

## 二次剩余

**int** power**(int** a**,** **int** b**,** **const** **int** MODE**)** **{**

**if** **(**b **==** **0)** **return** **1;**

**int** t **=** power**(**a**,** b **/** **2,** MODE**);**

t **=** **(**t **\*** t**)** **%** MODE**;**

**if** **(**b **&** **1)** t **=** **(**t **\*** a**)** **%** MODE**;**

**return** t**;**

**}**

**void** calcH**(int** **&**t**,** **int** **&**h**,** **const** **int** p**)** **{**

**int** tmp **=** p **-** **1;**

**for** **(**t **=** **0;** **(**tmp **&** **1)** **==** **0;** tmp **/=** **2)** t**++;**

h **=** tmp**;**

**}**

// solve equation x^2 mod p = a

**bool** solve**(int** a**,** **int** p**,** **int** **&**x**,** **int** **&**y**)** **{**

srand**(19920225);**

**if** **(**p **==** **2)** **{**

x **=** y **=** **1;**

**return** **true;**

**}**

**int** p2 **=** p **/** **2;**

**int** tmp **=** power**(**a**,** p2**,** p**);**

**if** **(**tmp **==** p **-** **1)** **return** **false;**

**if** **((**p **+** **1)** **%** **4** **==** **0)** **{**

x **=** power**(**a**,** **(**p **+** **1)** **/** **4,** p**);**

y **=** p **-** x**;**

**return** **true;**

**}** **else** **{**

**int** t**,** h**,** b**,** pb**;**

calcH**(**t**,** h**,** p**);**

**if** **(**t **>=** **2)** **{**

**do** **{**

b **=** rand**()** **%** **(**p **-** **2)** **+** **2;**

**}**

**while** **(**power**(**b**,** p **/** **2,** p**)** **!=** p **-** **1);**

pb **=** power**(**b**,** h**,** p**);**

**}**

**int** s **=** power**(**a**,** h **/** **2,** p**);**

**for** **(int** step **=** **2;** step **<=** t**;** step**++)** **{**

**int** ss **=** **(((**s **\*** s**)** **%** p**)** **\*** a**)** **%** p**;**

**for** **(int** i **=** **0;** i **<** t **-** step**;** i**++)** ss **=** **(**ss **\*** ss**)** **%** p**;**

**if** **(**ss **+** **1** **==** p**)** s **=** **(**s **\*** pb**)** **%** p**;**

pb **=** **(**pb **\*** pb**)** **%** p**;**

**}**

x **=** **(**s **\*** a**)** **%** p**;**

y **=** p **-** x**;**

**}**

**return** **true;**

**}**

## Pell方程

ULL A**,**B**,**p**[**maxn**],**q**[**maxn**],**a**[**maxn**],**g**[**maxn**],**h**[**maxn**];**

**int** main**()**

**{**

**int** n**;**

**for** **(int** test**=1;**scanf**(**"%d"**,&**n**)** **&&** n**;++**test**)**

**{**

printf**(**"Case %d: "**,**test**);**

**if** **(**fabs**(**sqrt**(**n**)-**floor**(**sqrt**(**n**)+1e-7))<=1e-7)**

**{**

**int** a**=(int)(**floor**(**sqrt**(**n**)+1e-7));**

printf**(**"%d %d\n"**,**a**,1);**

**}else**

**{**

//求x^2-ny^2=1的最小正整数根,n不是完全平方数

p**[1]=1;**p**[0]=0;**

q**[1]=0;**q**[0]=1;**

a**[2]=(int)(**floor**(**sqrt**(**n**)+1e-7));**

g**[1]=0;**h**[1]=1;**

**for** **(int** i**=2;**i**;++**i**)**

**{**

g**[**i**]=-**g**[**i**-1]+**a**[**i**]\***h**[**i**-1];**

h**[**i**]=(**n**-**sqr**(**g**[**i**]))/**h**[**i**-1];**

a**[**i**+1]=(**g**[**i**]+**a**[2])/**h**[**i**];**

p**[**i**]=**a**[**i**]\***p**[**i**-1]+**p**[**i**-2];**

q**[**i**]=**a**[**i**]\***q**[**i**-1]+**q**[**i**-2];**

**if** **(**sqr**((**ULL**)(**p**[**i**]))-**n**\***sqr**((**ULL**)(**q**[**i**]))==1)**

**{**

A**=**p**[**i**];**B**=**q**[**i**];**

**break;**

**}**

**}**

cout **<<** A **<<** ' ' **<<** B **<<**endl**;**

**}**

**}**

**return** **0;**

**}**

## 蔡勒公式

**int** zeller**(int** y**,int** m**,int** d**)** **{**

**if** **(**m**<=2)** y**--,**m**+=12;**

**int** c**=**y**/100;**

y**%=100;**

**int** w**=((**c**>>2)-(**c**<<1)+**y**+(**y**>>2)+(13\*(**m**+1)/5)+**d**-1)%7;**

**if** **(**w**<0)** w**+=7;**

**return(**w**);**

**}**

## Romberg

**template<class** T**>**

**double** romberg**(const** T**&**f**,double** a**,double** b**,double** eps**=1e-8){**

std**::**vector**<double>**t**;**

**double** h**=**b**-**a**,**last**,**curr**;**

**int** k**=1,**i**=1;**

t**.**push\_back**(**h**\*(**f**(**a**)+**f**(**b**))/2);** // 梯形

**do{**

last**=**t**.**back**();**

curr**=0;**

**double** x**=**a**+**h**/2;**

**for(int** j**=0;**j**<**k**;++**j**){**

curr**+=**f**(**x**);**

x**+=**h**;**

**}**

curr**=(**t**[0]+**h**\***curr**)/2;**

**double** k1**=4.0/3.0,**k2**=1.0/3.0;**

**for(int** j**=0;**j**<**i**;**j**++){**

**double** temp**=**k1**\***curr**-**k2**\***t**[**j**];**

t**[**j**]=**curr**;**

curr**=**temp**;**

k2**/=4\***k1**-**k2**;** // 防止溢出

k1**=**k2**+1;**

**}**

t**.**push\_back**(**curr**);**

k**\*=2;**

h**/=2;**

i**++;**

**}while(**std**::**fabs**(**last**-**curr**)>**eps**);**

**return** t**.**back**();**

**}**

## 线性规划

Max cx subject to Ax<=b and x>=0

vector**<double>** simplex**(**vector**<**vector**<double>** **>** A**,** vector**<double>** b**,** vector**<double>** c**)**

**{**

**int** n **=** A**.**size**(),** m **=** A**[0].**size**()** **+** **1,** r **=** n**,** s **=** m **-** **1;**

vector**<**vector**<double>** **>** D**(**n **+** **2,** vector**<double>(**m **+** **1,** **0));**

vector**<int>** ix**(**n **+** m**);**

**for** **(int** i **=** **0;** i **<** n **+** m**;** **++** i**)** ix**[**i**]** **=** i**;**

**for** **(int** i **=** **0;** i **<** n**;** **++** i**)** **{**

**for** **(int** j **=** **0;** j **<** m **-** **1;** **++** j**)** D**[**i**][**j**]** **=** **-**A**[**i**][**j**];**

D**[**i**][**m **-** **1]** **=** **1;**

D**[**i**][**m**]** **=** b**[**i**];**

**if** **(**D**[**r**][**m**]** **>** D**[**i**][**m**])** r **=** i**;**

**}**

**for** **(int** j **=** **0;** j **<** m **-** **1;** **++** j**)** D**[**n**][**j**]** **=** c**[**j**];**

D**[**n **+** **1][**m **-** **1]** **=** **-1;**

**for** **(double** d**;** **;** **)** **{**

**if** **(**r **<** n**)** **{**

**int** t **=** ix**[**s**];** ix**[**s**]** **=** ix**[**r **+** m**];** ix**[**r **+** m**]** **=** t**;**

D**[**r**][**s**]** **=** **1.0** **/** D**[**r**][**s**];**

vector**<int>** speedUp**;**

**for** **(int** j **=** **0;** j **<=** m**;** **++** j**)** **if** **(**j **!=** s**)** **{**

D**[**r**][**j**]** **\*=** **-**D**[**r**][**s**];**

**if(**D**[**r**][**j**])** **{**

speedUp**.**push\_back**(**j**);**

**}**

**}**

**for** **(int** i **=** **0;** i **<=** n **+** **1;** **++** i**)** **if** **(**i **!=** r**)** **{**

**for(int** j **=** **0;** j **<** speedUp**.**size**();** **++** j**)**

D**[**i**][**speedUp**[**j**]]** **+=** D**[**r**][**speedUp**[**j**]]** **\*** D**[**i**][**s**];**

D**[**i**][**s**]** **\*=** D**[**r**][**s**];**

**}**

**}**

r **=** **-1;** s **=** **-1;**

**for** **(int** j **=** **0;** j **<** m**;** **++** j**)** **if** **(**s **<** **0** **||** ix**[**s**]** **>** ix**[**j**])** **{**

**if** **(**D**[**n **+** **1][**j**]** **>** EPS **||** **(**D**[**n **+** **1][**j**]** **>** **-**EPS **&&** D**[**n**][**j**]** **>** EPS**))** s **=** j**;**

**}**

**if** **(**s **<** **0)** **break;**

**for** **(int** i **=** **0;** i **<** n**;** **++** i**)** **if** **(**D**[**i**][**s**]** **<** **-**EPS**)** **{**

**if** **(**r **<** **0** **||** **(**d **=** D**[**r**][**m**]** **/** D**[**r**][**s**]** **-** D**[**i**][**m**]** **/** D**[**i**][**s**])** **<** **-**EPS **||** **(**d **<** EPS **&&** ix**[**r **+** m**]** **>** ix**[**i **+** m**]))**

r **=** i**;**

**}**

**if** **(**r **<** **0)** **return** vector**<double>();** // no bound

**}**

**if** **(**D**[**n **+** **1][**m**]** **<** **-**EPS**)** **return** vector**<double>();** // no solution

vector**<double>** x**(**m **-** **1);**

**for** **(int** i **=** m**;** i **<** n **+** m**;** **++** i**)** **if** **(**ix**[**i**]** **<** m **-** **1)** x**[**ix**[**i**]]** **=** D**[**i **-** m**][**m**];**

**return** x**;** // max answer is D[n][m]

**}**

## FFT && NTT

**const** **int** N **=** **;**

**const** **int** P **=** **786433;**

**const** **int** G **=** **10;**

**int** modPow**(long** **long** a**,** **int** b**,** **int** c**)**

**{**

**int** ret **=** **1;**

**for(** **;** b**;** b **>>=** **1)** **{**

**if** **(**b **&** **1)**

ret **=** **(long** **long)**ret **\*** a **%** c**;**

a **=** **(long** **long)**a **\*** a **%** c**;**

**}**

**return** ret**;**

**}**

**void** dft**(int** **\***x**,** **int** on**,** **int** n**)**

**{**

**int** k**,** id**,** r**,** tmp**,** u**,** t**;**

**for(int** i **=** **1,** j **=** n **>>** **1;** i **<** n **-** **1;** **++** i**)** **{**

**if** **(**i **<** j**)** swap**(**x**[**i**],** x**[**j**]);**

**for(**k **=** n **>>** **1;** j **>=** k**;** j **-=** k**,** k **>>=** **1);**

j **+=** k**;**

**}**

**for(int** h **=** **2;** h **<=** n**;** h **<<=** **1)** **{**

r **=** modPow**(**G**,** **(**P **-** **1)** **/** h**,** P**);**

**if** **(**on **<** **0)** r **=** modPow**(**r**,** P **-** **2,** P**);**

**int** p **=** h **>>** **1;**

**for(int** j **=** **0;** j **<** n**;** j **+=** h**)** **{**

**int** w **=** **1;**

**for(int** k **=** j**;** k **<** j **+** p**;** k **++)** **{**

u **=** x**[**k**];**

id **=** k **+** p**;**

t **=** **(long** **long)**w **\*** x**[**id**]** **%** P**;**

x**[**k**]** **=** **(**u **+** t**)** **%** P**;**

x**[**id**]** **=** **(**u **-** t **+** P**)** **%** P**;**

w **=** **(long** **long)**w **\*** r **%** P**;**

**}**

**}**

**}**

**}**

**int** xa**[**N**],** xb**[**N**];**

**void** dft**(int** **\***a**,** **int** lenA**,** **int** **\***b**,** **int** lenB**,** **int** **\***ans**,** **int** **&**lenAns**)**

**{**

**for(**lenAns **=** **1;** lenAns **<** lenA **+** lenB**;** lenAns **<<=** **1);**

**for(int** i **=** **0;** i **<** lenAns**;** **++** i**)** **{**

xa**[**i**]** **=** xb**[**i**]** **=** **0;**

**}**

**for(int** i **=** **0;** i **<** lenA**;** **++** i**)** **{**

xa**[**i**]** **=** a**[**i**]** **%** P**;**

**}**

**for(int** i **=** **0;** i **<** lenB**;** **++** i**)** **{**

xb**[**i**]** **=** b**[**i**]** **%** P**;**

**}**

dft**(**xa**,** **1,** lenAns**);**

dft**(**xb**,** **1,** lenAns**);**

**for(int** i **=** **0;** i **<** lenAns**;** **++** i**)** **{**

xa**[**i**]** **=** **(long** **long)**xa**[**i**]** **\*** xb**[**i**]** **%** P**;**

**}**

dft**(**xa**,** **-1,** lenAns**);**

**int** tmp **=** modPow**(**lenAns**,** P **-** **2,** P**);**

**for(int** i **=** **0;** i **<** lenAns**;** **++** i**)** **{**

ans**[**i**]** **=** **(long** **long)**xa**[**i**]\*** tmp **%** P**;**

**}**

**}**

## 回文串manacher

**for(int** i **=** **1,** j **=** **0;** i **!=** **(**n **<<** **1)-** **1;** **++** i**)**

**{**

**int** p **=** i **>>** **1,** q **=** i **-** p**,** r **=** **((**j **+** **1)** **>>** **1)** **+** l**[**j**]** **-** **1;**

l**[**i**]** **=** r **<** q**?** **0:** min**(**r **-** q **+** **1,** l**[(**j **<<** **1)** **-** i**]);**

**while(**p **-** l**[**i**]** **!=** **-1** **&&** q **+** l**[**i**]** **!=** n **&&** s**[**p **-** l**[**i**]]** **==** s**[**q **+** l**[**i**]])**

l**[**i**]** **++;**

**if(**q **+** l**[**i**]** **-** **1** **>** r**)** j **=** i**;**

a **+=** l**[**i**];**

**}**

## 后缀数组(nlogn)

**const** **int** MAX\_N **=** **1000000** **+** **10;**

**int** rank**[**MAX\_N**],** height**[**MAX\_N**];**

**int** cmp**(int** **\***x**,** **int** a**,** **int** b**,** **int** d**)** **{**

**return** x**[**a**]** **==** x**[**b**]** **&&** x**[**a **+** d**]** **==** x**[**b **+** d**];**

**}**

**void** doubling**(int** **\***a**,** **int** N**,** **int** M**)** **{**

**static** **int** sRank**[**MAX\_N**],** tmpA**[**MAX\_N**],** tmpB**[**MAX\_N**];**

**int** **\***x **=** tmpA**,** **\***y **=** tmpB**;**

**for(int** i **=** **0;** i **<** M**;** **++** i**)** sRank**[**i**]** **=** **0;**

**for(int** i **=** **0;** i **<** N**;** **++** i**)** **++** sRank**[**x**[**i**]** **=** a**[**i**]];**

**for(int** i **=** **1;** i **<** M**;** **++** i**)** sRank**[**i**]** **+=** sRank**[**i **-** **1];**

**for(int** i **=** N **-** **1;** i **>=** **0;** **--** i**)** sa**[--** sRank**[**x**[**i**]]]** **=** i**;**

**for(int** d **=** **1,** p **=** **0;** p **<** N**;** M **=** p**,** d **<<=** **1)** **{**

p **=** **0;** **for(int** i **=** N **-** d**;** i **<** N**;** **++** i**)** y**[**p **++]** **=** i**;**

**for(int** i **=** **0;** i **<** N**;** **++** i**)**

**if** **(**sa**[**i**]** **>=** d**)** y**[**p **++]** **=** sa**[**i**]** **-** d**;**

**for(int** i **=** **0;** i **<** M**;** **++** i**)** sRank**[**i**]** **=** **0;**

**for(int** i **=** **0;** i **<** N**;** **++** i**)** **++** sRank**[**x**[**i**]];**

**for(int** i **=** **1;** i **<** M**;** **++** i**)** sRank**[**i**]** **+=** sRank**[**i **-** **1];**

**for(int** i **=** N **-** **1;** i **>=** **0;** **--** i**)** sa**[--** sRank**[**x**[**y**[**i**]]]]** **=** y**[**i**];**

swap**(**x**,** y**);** x**[**sa**[0]]** **=** **0;** p **=** **1;**

**for(int** i **=** **1;** i **<** N**;** **++** i**)**

x**[**sa**[**i**]]** **=** cmp**(**y**,** sa**[**i**],** sa**[**i **-** **1],** d**)** **?** p **-** **1** **:** p **++;**

**}**

**}**

**void** calcHeight**()** **{**

**for(int** i **=** **0;** i **<** N**;** **++** i**)** rank**[**sa**[**i**]]** **=** i**;**

**int** cur **=** **0;**

**for(int** i **=** **0;** i **<** N**;** **++** i**)**

**if** **(**rank**[**i**])** **{**

**if** **(**cur**)** cur **--;**

**for(** **;** a**[**i **+** cur**]** **==** a**[**sa**[**rank**[**i**]** **-** **1]** **+** cur**];** **++** cur**);**

height**[**rank**[**i**]]** **=** cur**;**

**}**

**}**

## dc3

//DC3 待排序的字符串放在r 数组中，从r[0]到r[n-1]，长度为n，且最大值小于m。

//约定除r[n-1]外所有的r[i]都大于0, r[n-1]=0。

//函数结束后，结果放在sa 数组中，从sa[0]到sa[n-1]。

#define maxn 10000

#define F(x) ((x)/3+((x)%3==1?0:tb))

#define G(x) ((x)<tb?(x)\*3+1:((x)-tb)\*3+2)

**int** wa**[**maxn**],**wb**[**maxn**],**wv**[**maxn**],**wss**[**maxn**];** //必须这么大

**int** s**[**maxn**\*3],**sa**[**maxn**\*3];**

**int** c0**(int** **\***r**,int** a**,int** b**){return** r**[**a**]==**r**[**b**]&&**r**[**a**+1]==**r**[**b**+1]&&**r**[**a**+2]==**r**[**b**+2];}**

**int** c12**(int** k**,int** **\***r**,int** a**,int** b**){**

**if(**k**==2)** **return** r**[**a**]<**r**[**b**]||**r**[**a**]==**r**[**b**]&&**c12**(1,**r**,**a**+1,**b**+1);**

**else** **return** r**[**a**]<**r**[**b**]||**r**[**a**]==**r**[**b**]&&**wv**[**a**+1]<**wv**[**b**+1];**

**}**

**void** sort**(int** **\***r**,int** **\***a**,int** **\***b**,int** n**,int** m**){**

**int** i**;** **for(**i**=0;**i**<**n**;**i**++)** wv**[**i**]=**r**[**a**[**i**]];**

**for(**i**=0;**i**<**m**;**i**++)** wss**[**i**]=0;** **for(**i**=0;**i**<**n**;**i**++)** wss**[**wv**[**i**]]++;**

**for(**i**=1;**i**<**m**;**i**++)** wss**[**i**]+=**wss**[**i**-1];**

**for(**i**=**n**-1;**i**>=0;**i**--)** b**[--**wss**[**wv**[**i**]]]=**a**[**i**];**

**}**

**void** dc3**(int** **\***r**,int** **\***sa**,int** n**,int** m**){**

**int** i**,**j**,\***rn**=**r**+**n**,\***san**=**sa**+**n**,**ta**=0,**tb**=(**n**+1)/3,**tbc**=0,**p**;**

r**[**n**]=**r**[**n**+1]=0;**

**for(**i**=0;**i**<**n**;**i**++)** **if(**i**%3!=0)** wa**[**tbc**++]=**i**;**

sort**(**r**+2,**wa**,**wb**,**tbc**,**m**);** sort**(**r**+1,**wb**,**wa**,**tbc**,**m**);** sort**(**r**,**wa**,**wb**,**tbc**,**m**);**

**for(**p**=1,**rn**[**F**(**wb**[0])]=0,**i**=1;**i**<**tbc**;**i**++)**

rn**[**F**(**wb**[**i**])]=**c0**(**r**,**wb**[**i**-1],**wb**[**i**])?**p**-1:**p**++;**

**if** **(**p**<**tbc**)** dc3**(**rn**,**san**,**tbc**,**p**);**

**else** **for** **(**i**=0;**i**<**tbc**;**i**++)** san**[**rn**[**i**]]=**i**;**

**for** **(**i**=0;**i**<**tbc**;**i**++)** **if(**san**[**i**]<**tb**)** wb**[**ta**++]=**san**[**i**]\*3;**

**if(**n**%3==1)** wb**[**ta**++]=**n**-1;**

sort**(**r**,**wb**,**wa**,**ta**,**m**);**

**for(**i**=0;**i**<**tbc**;**i**++)** wv**[**wb**[**i**]=**G**(**san**[**i**])]=**i**;**

**for(**i**=0,**j**=0,**p**=0;**i**<**ta **&&** j**<**tbc**;**p**++)**

sa**[**p**]=**c12**(**wb**[**j**]%3,**r**,**wa**[**i**],**wb**[**j**])?**wa**[**i**++]:**wb**[**j**++];**

**for(;**i**<**ta**;**p**++)** sa**[**p**]=**wa**[**i**++];** **for(;**j**<**tbc**;**p**++)** sa**[**p**]=**wb**[**j**++];}**

**int** main**(){**

**int** n**,**m**=0;** scanf**(**"%d"**,&**n**);**

**for** **(int** i**=0;**i**<**n**;**i**++)** scanf**(**"%d"**,&**s**[**i**]),**s**[**i**]++,**m**=**max**(**s**[**i**]+1,**m**);**

printf**(**"%d\n"**,**m**);** s**[**n**++]=0;** dc3**(**s**,**sa**,**n**,**m**);**

**for** **(int** i**=0;**i**<**n**;**i**++)** printf**(**"%d "**,**sa**[**i**]);**printf**(**"\n"**);**

**}**

## 后缀自动机

**struct** State **{**

**int** length**;**

State **\***parent**;**

State **\***go**[**C**];**

State**(int** length**):** length**(**length**),** parent**(NULL)** **{**

memset**(**go**,** **0,** **sizeof(**go**));**

**}**

State**\*** extend**(**State **\***start**,** **int** token**)** **{**

State **\***p **=** **this;**

State **\***np **=** **new** State**(this->**length **+** **1);**

**while** **(**p **!=** **NULL** **&&** p**->**go**[**token**]** **==** **NULL)** **{**

p**->**go**[**token**]** **=** np**;**

p **=** p**->**parent**;**

**}**

**if** **(**p **==** **NULL)** **{**

np**->**parent **=** start**;**

**}** **else** **{**

State **\***q **=** p**->**go**[**token**];**

**if** **(**p**->**length **+** **1** **==** q**->**length**)** **{**

np**->**parent **=** q**;**

**}** **else** **{**

State **\***nq **=** **new** State**(**p**->**length **+** **1);**

memcpy**(**nq**->**go**,** q**->**go**,** **sizeof(**q**->**go**));**

nq**->**parent **=** q**->**parent**;**

np**->**parent **=** q**->**parent **=** nq**;**

**while** **(**p **!=** **NULL** **&&** p**->**go**[**token**]** **==** q**)** **{**

p**->**go**[**token**]** **=** nq**;**

p **=** p**->**parent**;**

**}**

**}**

**}**

**return** np**;**

**}**

**};**

## 字符串最小表示

std**::**string find**(**std**::**string s**)** **{**

**int** i**,** j**,** k**,** l**;**

**int** N **=** s**.**length**();**

s **+=** s**;**

**for** **(**i **=** **0,** j **=** **1;** j **<** N**;** **)** **{**

**for** **(**k **=** **0;** k **<** N **&&** s**[**i **+** k**]** **==** s**[**j **+** k**];** k **++);**

**if** **(**k **>=** N**)** **break;**

**if** **(**s**[**i **+** k**]** **<** s**[**j **+** k**])** **{**

j **+=** k **+** **1;**

**}** **else** **{**

l **=** i **+** k**;**

i **=** j**;**

j **=** max**(**l**,** j**)** **+** **1;**

**}**

**}**

**return** s**.**substr**(**i**,** N**);**

**}**

## DancingLinks

如果矩阵中所有的列均被删除**,** 找到一组合法解**,** 退出．

任意选择一个未被删除的列c**,**

枚举一个未被删除的行r**,** 且Matrix**[**r**][**c**]** **=** **1,** 将**(**r**,** c**)**加入Ans．

枚举所有的列j**,** Matrix**[**r**][**j**]** **=** **1,** 将第j列删除．

枚举所有的行i**,** Matrix**[**i**][**j**]** **=** **1,** 将第i行删除．

Algorithm\_X**(**Dep **+** **1)**

Procedure Algorithm\_X**(**Dep**)**

如果h**^.**right **=** h**(**即所有的列均被删除**),** 找到一组解**,** 退出．

利用h和right指针找到一个c**,** 满足size**[**c**]**最小．

如果size**[**c**]** **=** **0(**当前列无法被覆盖**),** 无解**,** 退出．

Cover**(**c**)**

**for** **(**i **=** c**^.**down**;** i **!=** c**;** i ← i**^.**down**)**

**for** **(**j **=** i**^.**right**;** j **!=** i**;** j ← j**^.**right**)** Cover**(**j**^.**col**)**

将i结点加入Ans**,** Algorithm\_X**(**Dep **+** **1)**

**for** **(**j **=** i**^.**left**;** j **!=** i**;** j ← j**^.**left**)** Recover**(**j**^.**col**)**

Recover**(**c**)**

Soduku问题可以转化一个Exact Cover Problem：16 **\*** **16** **\*** **16行,** **(**i**,** j**,** k**)**表示**(**i**,** j**)**这个格子填上字母k．16 **\*** **16** **\*** **4列分别表示第i行中的字母k,** 第i列中的字母k**,** 第i个子矩阵中的字母k**,** 以及**(**i**,** j**)**这个格子．对于每个集合**(**i**,** j**,** k**),** 它包含了4个元素：Line**(**i**,** k**),** Col**(**j**,** k**),** Sub**(**P**[**i**][**j**],** k**),** Grid**(**i**,** j**),** 其中P**[**i**][**j**]**表示**(**i**,** j**)**这个格子所属的子矩阵．本题转化为一个4096行**,** **1024列,** 且1的个数为16384个的矩阵．下面介绍解决一般的Exact Cover Problem的Algorithm X．

N皇后问题：关键是构建Exact Cover问题的矩阵：N **\*** N行对应了N **\*** N个格子**,** **6N-2列对应了N行,** N列**,** **2N-1条主对角线,** **2N-1条副对角线．第i行共4个1,** 分别对应**(**i**,** j**)**这个格子所处的行**,** 列**,** 主对角线和副对角线．直接对这个矩阵作Algorithm X是错误的**,** 虽然每行**,** 每列都恰好被覆盖一次**,** 但是对角线是最多覆盖一次**,** 它可以不被覆盖**,** 这与Exact Cover问题的定义是不同的．

有两种处理的方法：

**1)** 新增4N**-2行,** 每行只有一个1**,** 分别对应了2N**-1条主对角线和2N-1条副对角线,** 这样就可以保证某个对角线不被覆盖的时候**,** 可以使用新增行来覆盖．

**2)** 每次选择一个size**[]**值最小的列c进行覆盖**,** 而这一步**,** 我们忽略掉所有的对角线列**,** 只考虑c为行和列的情况．

事实证明**,** 第2**)**种方法的效果好很多**,** 因此这个问题可以使用Algorithm X轻松得到解决．

**struct** data

**{**

**int** l**,**r**,**u**,**d**,**x**,**y**;**

**};**

data a**[5101];**

**int** sum**[310];**

**void** del**(int** x**)**

**{**

a**[**a**[**x**].**l**].**r**=**a**[**x**].**r**;**

a**[**a**[**x**].**r**].**l**=**a**[**x**].**l**;**

**for** **(int** i**=**a**[**x**].**d**;**i**!=**x**;**i**=**a**[**i**].**d**)**

**for** **(int** j**=**a**[**i**].**r**;**j**!=**i**;**j**=**a**[**j**].**r**)**

**{**

sum**[**a**[**j**].**y**]--;**

a**[**a**[**j**].**u**].**d**=**a**[**j**].**d**;**

a**[**a**[**j**].**d**].**u**=**a**[**j**].**u**;**

**}**

**}**

**void** renew**(int** x**)**

**{**

a**[**a**[**x**].**l**].**r**=**x**;**

a**[**a**[**x**].**r**].**l**=**x**;**

**for** **(int** i**=**a**[**x**].**u**;**i**!=**x**;**i**=**a**[**i**].**u**)**

**for** **(int** j**=**a**[**i**].**l**;**j**!=**i**;**j**=**a**[**j**].**l**)**

**{**

sum**[**a**[**j**].**y**]++;**

a**[**a**[**j**].**u**].**d**=**j**;**

a**[**a**[**j**].**d**].**u**=**j**;**

**}**

**}**

**bool** search**()**

**{**

**if** **(**a**[0].**r**==0)**

**return(true);**

**int** k**,**min**=20000000;**

**for** **(int** i**=**a**[0].**r**;**i**!=0;**i**=**a**[**i**].**r**)**

**if** **(**sum**[**i**]<**min**)**

min**=**sum**[**k**=**i**];**

del**(**k**);**

**for** **(int** i**=**a**[**k**].**d**;**i**!=**k**;**i**=**a**[**i**].**d**)**

**{**

**for** **(int** j**=**a**[**i**].**r**;**j**!=**i**;**j**=**a**[**j**].**r**)**

del**(**a**[**j**].**y**);**

**if** **(**search**())**

**return(true);**

**for** **(int** j**=**a**[**i**].**l**;**j**!=**i**;**j**=**a**[**j**].**l**)**

renew**(**a**[**j**].**y**);**

**}**

renew**(**k**);**

**return(false);**

**}**

## 弦图的完美消除序列

从n到1的顺序依次给点标号（标号为i的点出现在完美消除序列的第i个）

设lable[i]表示第i个点与多少个已标号的点相邻，每次选择label[i]最大的未标号的点进行标号。

任取一个已标号的与当前新标号的点相邻的点，如果与其他的已标号的且与当前点相邻的点之间没有边，则无解。

弦图里的团数等于色数，色数（从后往前）和最大独立集（从前往后）都可以按完美消除序列的顺序贪心。

## 图同构Hash

枚举点a，迭代K次后求得的就是a点所对应的hash值。

其中K、A、B、C、D、P为hash参数，可自选。

## 魔方旋转群

其中1到54是面,55到66是棱,67到74是顶点

int R[75]={0,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,

1,2,3,4,5,6,7,8,9,43,40,37,44,41,38,45,42,39,48,51,54,47,50,53,46,49,52,

2,3,4,1,6,7,8,5,10,11,12,9,

2,3,4,1,6,7,8,5};

int D[75]={0,46,47,48,49,50,51,52,53,54,16,13,10,17,14,11,18,15,12,45,44,43,42,41,40,39,38,37,

30,33,36,29,32,35,28,31,34,1,2,3,4,5,6,7,8,9,27,26,25,24,23,22,21,20,19,

5,10,1,9,7,11,3,12,8,6,2,4,

5,6,2,1,8,7,3,4};

int F[75]={0,3,6,9,2,5,8,1,4,7,48,51,54,47,50,53,46,49,52,25,22,19,26,23,20,27,24,21,

39,42,45,38,41,44,37,40,43,12,15,18,11,14,17,10,13,16,30,33,36,29,32,35,28,31,34,

10,6,11,2,9,8,12,4,1,5,7,3,

2,6,7,3,1,5,8,4};

## 直线下有多少个格点

求

LL count**(**LL n**,** LL a**,** LL b**,** LL m**)** **{**

**if** **(**b**==**0**)** **return** n **\*** **(**a **/** m**);**

**if** **(**a**>=**m**)** **return** n **\*** **(**a **/** m**)** **+** count**(**n**,** a **%** m**,** b**,** m**);**

**if** **(**b**>=**m**)** **return** **(**n **-** 1**)** **\*** n **/** 2 **\*** **(**b **/** m**)** **+** count**(**n**,** a**,** b **%** m**,** m**);**

**return** count**((**a **+** b **\*** n**)** **/** m**,** **(**a **+** b **\*** n**)** **%** m**,** m**,** b**);**

**}**

## 综合

S=a+ b/2 - 1。

　　(其中a表示多边形内部的点数,b表示多边形边界上的点数,S表示多边形的面积)

定理1：最小覆盖数 = 最大匹配数

定理2：最大独立集S 与 最小覆盖集T 互补。

算法：

1. 做最大匹配，没有匹配的空闲点∈S

2. 如果u∈S那么u的临点必然属于T

3. 如果一对匹配的点中有一个属于T那么另外一个属于S

4. 还不能确定的，把左子图的放入S，右子图放入T

算法结束

p是素数且2^p－1的是素数,n不超过258的全部梅森素数终于确定，是

n=2,3,5,7,13,17,31,61,89,107,127,257

有上下界网络流，求可行流部分，增广的流量不是实际流量。若要求实际流量应该强算一遍源点出去的流量。

求最小下届网络流：

方法一：加t-s的无穷大流，求可行流，然后把边反向后（减去下届网络流），在残留网络中从汇到源做最大流。

方法二：在求可行流的时候，不加从汇到源的无穷大边，得到最大流X， 加上从汇到源无穷大边后，再求最大流得到Y。

那么Y即是答案最小下届网络流。

原因：感觉上是在第一遍已经把内部都消耗光了，第二遍是必须的流量。

路径剖分，取节点数最多的子树伸出来的路径。

序列差分表由它的第0行确定，也就是原序列，但同时也可以由第0条对角线上的元素确定。

换句话说，由差分表的第0条对角线就可以确定原序列。有这样两个公式：

原序列为h\_i，第0条对角线为c\_o,c\_1,…,c\_p,0,0,0,…

则h\_n = c\_0\*C(n,0)+c\_1\*C(n,1)+…+c\_p\*C(n,p)，

Σh\_k(k=0..n) = c\_0\*C(n+1,1)+c\_2\*(n+1,2)+…+c\_p\*C(n+1,p+1)。

记住这两个公式，差分表（的第0条对角线）就变得非常有用了。

平面图一定存在一个度小于等于5的点,且可以四染色

（ 欧拉公式 ） 设 G 是连通的平面图，n , m, r 分别是其顶点数、边数和面数，n-m+r=2

极大平面图 m≤3n-6

平方数的和是平方数的问题。

a=k\*(s^2 - t^2);b=2\*k\*s\*t;c=k(s^2 + t^2);则c^2=a^2+b^2 完全的公式

重加权的方法如下：增加人工结点s，直接到所有点连一条弧，权均为0，然

后以s为起点运行bellman-ford，求出dist(v)。如果有负权圈则退出，否则对于原图中的每个条边(u,v)，设新权w'(u,v)=dist(u)+w(u,v)-dist(v)，则它是非负的

k-连通(k-connected) ：对于任意一对结点都至少存在结点各不相同的k条路。

点连通度(vertex connectivity) ：把图变成非连通图所需删除的最少点数。

这两个定义是互通的，因为我们有：

Whitney定理：一个图是k-连通的当且仅当它的点连通度至少为k。

Gessel–Viennot lemma

给定一个图与n个起点n个终点。则从对应的起点至终点的不相交路径条数为det(A)，这里A={a[i][j]}，a[i][j]表示从第i个起点至第j个终点的路径条数。

Stirling公式 说明: n! \approx \sqrt{2\pi n}\, \left(\frac{n}{e}\right)^{n}.

欧拉常数

0.577215,66490,15328,60606,51209,00824,02431,04215,93359,39923,59880

圆周率

3.141592653589793238462643383279502884197169399375105820974944592307816

连分数表示[3;7,15,1,292,1,1,1,2,1,3,1,14,2,1,1,2…]前几个近似22/7,333/106,355/113

n个球放入m个箱子里，有多少种不同放法（不一定是球和箱子，也可能是其他的元素与其他的放置位置,例如N个人分到M个单位，每班至少一人，里面已经暗中说明球不同，单位不同）

看似很简单的问题其实非常复杂，球是否相同，箱是否相同？是否允许有空盒

不难看出一共8类情况

1) 球同，盒同，无空箱 2) 球同，盒同，允许空箱

3) 球同，盒不同，无空箱 4) 球同，盒不同，允许空箱

5) 球不同，盒相同，无空箱 6) 球不同，盒相同，允许空箱

7) 球不同，盒不同，无空箱 8) 球不同，盒不同，允许空箱

3的公式是把n个球排成一排，（一种方法），它们中间有n-1个空。取m-1个小棍，放到空上，就把它们分成m部分，由于小棍不相邻，所以没有空箱子。它的方法数有

C(N-1,M-1),也就是球减1里面挑M-1个箱子做组合

4的公式在3的基础上升华出来的，为了避免空箱子，先在每一个箱子假装都放一个球，这样就有n+m个球，C（n+m-1,m-1)，多了M个元素而已

关于1,2类情况，直接f[i][j]计数。

先来分析最特殊的8号：N球不同，M箱不同，允许空。每个球都有M种选择，N个球就有M的N次方分法。

S(n,1)=S(n,n)=1,S（n,k)=S(n-1,k-1)+k\*S(n-1,k)

当遇见类型5即：N不同球，M同箱子，无空箱。一共有S(N,M)种分法。

而类型6，N不同球，M同箱，允许空的时候（在类型5的基础上允许空箱）。明显是N个球不变，一个空箱子都没有+有一个空箱子+有两个空箱子+有三个空箱子+…… 都装在一个箱子。说的简单点一共有就是

S(N,1)+S(N,2)+S(N,3)+..........S(N,M)=也就是说第N排开始第1个数字一直加到第M个数字就是总的分法

而类型7同样是在类型5的基础上升华，因为5是箱同的，而7箱不同，所以箱子自身多了P(M,M)=M!倍可能

所以类型7的公式就是M!乘以S(N,M)

Bell number:

Legendre symbol:

Jacobi symbol:

BEST Throrem:

线性规划强对偶定理:

有值时两者相等,一个无解等价于另一个解无界.

## 基本形 公式

**椭圆：**

椭圆，其中离心率焦点参数

椭圆上(x,y)点处的曲率半径为 ,其中分别为(x,y)与两焦点的距离。设点A和点M的坐标分别为(a,0)和(x,y)，则AM的弧长为

椭圆的周长为 ，其中

设椭圆上点M(x,y),N(x,-y),x,y>0,A(a,0),原点O(0,0)。

扇形OAM的面积 弓形MAN的面积

方程，5个点确定一个圆锥曲线。

为(x,y)点关于椭圆中心的极角，r为(x,y)到椭圆中心的距离，椭圆极坐标方程:

**抛物线**

标准方程 曲率半径

弧长：设M(x,y)是抛物线上一点，则]

弓形面积：设M，D是抛物线上两点，且分居一、四象限。作一条平行于MD且与抛物线相切的直线L。若M到L的距离为h。则有

**重心**

半径为r、圆心角为的扇形的重心与圆心的距离为

半径为r、圆心角为的圆弧的重心与圆心的距离为

椭圆上半部分的重心与圆心的距离为

抛物线中弓形MOD的重心满足 , P是直线L与抛物线的切点，Q在MD上且PQ平行x轴。C是重心。

**额外的公式**

***四边形***: D1,D2为对角线,M对角线中点连线,A为对角线夹角

1.a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2 2. S=D1D2sin(A)/2

(以下对圆的内接四边形)

3. ac+bd=D1D2 4.S=sqrt((P-a)(P-b)(P-c)(P-d)),P为半周长

***正n边形:***R为外接圆半径,r为内切圆半径

1. 中心角 A=2PI/n 2. 内角C=(n-2)PI/n

3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)

4. 面积S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

***圆:*** 1. 弧长 l=rA 2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)

3. 弓形高h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2

4.扇形面积S1=rl/2=r^2A/2

5.弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2

***棱柱:*** 1. 体积 V=Ah,A为底面积,h为高

2. 侧面积S=lp,l为棱长,p为直截面周长 3. 全面积 T=S+2A

***棱锥:*** 1.体积 V=Ah/3,A为底面积,h为高 (以下对正棱锥)

2. 侧面积S=lp/2,l为斜高,p为底面周长 3. 全面积 T=S+A

***棱台:***1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2为上下底面积,h为高

(以下为正棱台)

2. 侧面积 S=(p1+p2)l/2,p1.p2为上下底面周长,l为斜高

3. 全面积 T=S+A1+A2

**算法**

## 树的计数

**有根树的计数**

令

于是，n+1个结点的有根树的总数为

附：

**无根树的计数**

当n是奇数时，则有 种不同的无根树。

当n是偶数时，则有这么多种不同的无根树。

## 代数

**Burnside引理**

**三次方程求根公式**

其中 j=0,1,2,

当求解时， 令 再求解y，即转化成的形式

**组合公式**

错排：

## 三角公式

## 积分表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | | | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | | | | | | |
|  | | | | |  | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | |  |
|  | | | | | | | |
|  |  | | | | | | |  |
|  |  | | | | | | |  |
|  | | |  | | | | | |
|  | |  | | | | |  | |
|  | | |  | | | | | |
|  | | |  | | | | | |
|  | | |  | | | | | |

## Java IO

import java.io.\*;

import java.util.\*;

import java.math.\*;

public class Main {

void run() throws Exception {

reader.close();

writer.close();

}

public static void main(String[] args) throws Exception {

(new Main()).run();

}

BufferedReader reader =

new BufferedReader(new InputStreamReader(System.in));

PrintWriter writer = new PrintWriter(System.out);

StringTokenizer tokenizer = null;

String next() throws Exception {

for (; tokenizer == null || !tokenizer.hasMoreTokens(); ) {

tokenizer = new StringTokenizer(reader.readLine());

}

return tokenizer.nextToken();

}

int nextInt() throws Exception {

return Integer.parseInt(next());

}

}

//BigInteger

a.modPow(b,c);//a^b mod c; a.isProbablePrime(**int** certainty);

a.nextProbablePrime(); a.shiftLeft(**int**);

bitCount() bitLength() clearBit(int i)

setBit(int i) flipBit(int i) testBit(int i)

//BigDecimal

**static** **int** ROUND\_CEILING,ROUND\_DOWN,ROUND\_FLOOR,

ROUND\_HALF\_DOWN,ROUND\_HALF\_EVEN,ROUND\_HALF\_UP,ROUND\_UP;

a.stripTrailingZeros();

//Vector

a.add((index),elem); a.remove(index); a.set(index,elem);

//Queue

a.add(elem); a.peek();//front  a.poll();//pop

cout.setf(ios::fixed,ios::floatfield);

cout.precision(**3**); cout<<**double**(u)<<endl;

## Vimrc

set nu ai ci si mouse=a ts=4 sts=4 sw=4

nmap<C-A> ggVG

vmap<C-C> "+y

nmap<F3> : vs %<.in <CR>

nmap<F4> : !gedit % <CR>

nmap<F5> : !./%< <CR>

nmap<F8> : !./%< < %<.in <CR>

nmap<F9> : make %< <CR>

"nmap<F9> :!export CXXFLAGS=-Wall && make %< <CR>

"autocmd BufNewFile \*.cpp 0r ~/temp.cpp

"set hlsearch incseach

"syntax on

"filetype plugin indent on