* **string.cpp**

**int\*** z\_function**(char** s**[],int** n**)**

**{**

**int\*** z **=** **new** **int[**n**];**

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

z**[**0**]** **=** n**;**

**int** L **=** 0**,** R **=** 1**;**

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

**if** **(**R **<=** i **||** z**[**i**-**L**]** **>=** R**-**i**){**

**int** x **=** **(**R **<=** i **?** i **:** R**);**

**while** **(**x **<** n **&&** s**[**x**]** **==** s**[**x**-**i**])** x**++;**

z**[**i**]** **=** x**-**i**;**

**if** **(**i **<** x**)** **{**L **=** i**;** R **=** x**;}**

**}**

**else**

z**[**i**]** **=** z**[**i**-**L**];**

**return** z**;**

**}**

*/\*banana\*/*

**void** IBWT**(){**

vector**<int>** index**[**256**];**

**for(int** i**=**0**;** i**<**N**;** i**++)** index**[**t**[**i**]].**push\_back**(**i**);**

**for(int** i**=**0**,** n**=**0**;** i**<**256**;** i**++)**

**for(int** j**=**0**;** j**<**index**[**i**].**size**();** j**++)**

next**[**n**++]** **=** index**[**i**][**j**];**

**}**

*/\*kmp\*/*

**for(int** i**=**0**,** j**=-**1**;** i**<**t**.**size**();** i**++){**

**while(**j**>=**0 **&&** p**[**j**+**1**]!=**t**[**i**])** j **=** f**[**j**];**

**if(**p**[**j**+**1**]==**t**[**i**])** j**++;**

**if(**j**==**p**.**size**()-**1**){**

**int** ans **=** i **-** p**.**size**();**

j **=** f**[**j**];**

**}**

**}**

* **KMP**

vector**<int>** lps**;** *// longest prefix suffix, 0-based*

**int** match**(const** string **&**text**,** **const** string **&**pat**)** **{**

*/\* Init is included \*/*

lps**.**resize**(**pat**.**size**());**

*/\* DP \*/*

lps**[**0**]** **=** 0**;**

**for** **(int** i**=**1**;** i**<**pat**.**size**();** i**++)** **{**

**int** len **=** lps**[**i**-**1**];**

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

**if** **(**pat**[**i**]** **==** pat**[**len**])** **{**

lps**[**i**]** **=** len **+** 1**;**

**break;**

**}**

**if** **(**len **<=** 0**)** **{**

lps**[**i**]** **=** 0**;**

**break;**

**}**

len **=** lps**[**len **-** 1**];**

**}**

**}**

*/\* Match \*/*

**int** i **=** 0**,** j **=** 0**;**

**while** **(**i **<** text**.**size**()** **&&** j **<** pat**.**size**())** **{**

**if** **(**text**[**i**]** **==** pat**[**j**])** i**++,** j**++;**

**else** **if** **(**j **==** 0**)** i**++;**

**else** j **=** lps**[**j **-** 1**];**

**}**

**if** **(**j **>=** pat**.**size**())** **return** i **-** j**;**

**return** **-**1**;**

**}**

* **Min string rotation**

**int** min\_string\_rotation**(const** string **&**s**){**

**int** n**=**s**.**size**(),**i**=**0**,**j**=**1**,**k**=**0**;**

**while(**i**<**n**&&**j**<**n**&&**k**<**n**){**

**int** t**=**s**[(**i**+**k**)%**n**]-**s**[(**j**+**k**)%**n**];**

**++**k**;**

**if(**t**){**

**if(**t**>**0**)**i**+=**k**;**

**else** j**+=**k**;**

**if(**i**==**j**)++**j**;**

k**=**0**;**

**}**

**}**

**return** min**(**i**,**j**);** *// Starting position*

**}**

* **sa.cpp**

*/\* rank: inverse sa \*/*

*/\* MAXL: Maximum length of string, lcp[i]: LCP(sa[i], sa[i-1]) \*/*

string text**;**

**int** sa**[**MAXL**],** isa**[**MAXL**],** lcp**[**MAXL**],** cnt**[**MAXL**+**ALPHA**];**

**void** build**(const** vector**<int>** **&**\_text**)** **{**

text **=** \_text **+** '\0'**;** *// Must add this, must >= 0*

**int** sz **=** text**.**size**(),** lim **=** ALPHA**;** *// Takes ALPHA time, note when #TC is large*

**for** **(int** i **=** 0**;** i **<** lim**;** i**++)** cnt**[**i**]** **=** 0**;**

**for** **(int** i **=** 0**;** i **<** sz**;** i**++)** cnt**[** isa**[**i**]** **=** text**[**i**]** **]++;**

**for** **(int** i **=** 1**;** i **<** lim**;** i**++)** cnt**[**i**]** **+=** cnt**[**i **-** 1**];**

**for** **(int** i **=** sz **-** 1**;** i **>=** 0**;** i**--)** sa**[** **--**cnt**[**text**[**i**]]** **]** **=** i**;**

lim **=** max**(**sz**,** ALPHA**);**

**int** **\***rk **=** isa**,** **\***nsa **=** lcp**,** **\***nrk **=** lcp**;**

**for** **(int** len **=** 1**;** len **<** sz**;** len **<<=** 1**)** **{**

**int** num **=** 0**;**

**for** **(int** i **=** sz **-** len**;** i **<** sz**;** i**++)** nsa**[**num**++]** **=** i**;**

**for** **(int** i **=** 0**;** i **<** sz**;** i**++)** **if** **(**sa**[**i**]** **>=** len**)** nsa**[**num**++]** **=** sa**[**i**]** **-** len**;**

**for** **(int** i **=** 0**;** i **<** lim**;** i**++)** cnt**[**i**]** **=** 0**;**

**for** **(int** i **=** 0**;** i **<** sz**;** i**++)** cnt**[** rk**[**i**]** **]++;**

**for** **(int** i **=** 1**;** i **<** lim**;** i**++)** cnt**[**i**]** **+=** cnt**[**i **-** 1**];**

**for** **(int** i **=** sz**-**1**;** i **>=** 0**;** i**--)** sa**[** **--**cnt**[**rk**[**nsa**[**i**]]]** **]** **=** nsa**[**i**];**

num **=** 0**;**

nrk**[**sa**[**0**]]** **=** num**++;**

**for** **(int** i **=** 1**;** i **<** sz**;** i**++)** **{**

**bool** cond **=** rk**[**sa**[**i**]]** **==** rk**[**sa**[**i**-**1**]]** **&&** sa**[**i**]** **+** len **<** sz**;**

cond **=** cond **&&** sa**[**i**-**1**]** **+** len **<** sz **&&** rk**[**sa**[**i**]+**len**]** **==** rk**[**sa**[**i**-**1**]+**len**];**

**if** **(**cond**)** nrk**[**sa**[**i**]]** **=** num **-** 1**;**

**else** nrk**[**sa**[**i**]]** **=** num**++;**

**}**

**if** **(**num **>=** sz**)** **break;**

lim **=** num**;**

swap**(**rk**,** nrk**);**

nsa **=** nrk**;**

**}**

**for** **(int** i**=**0**;** i**<**sz**;** i**++)** isa**[**sa**[**i**]]** **=** i**;**

*/\* LCP \*/*

**int** len **=** 0**;**

lcp**[**0**]** **=** 0**;** *// Undefined*

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

**if** **(**isa**[**i**]** **==** 0**)** **continue;**

len **=** max**(**0**,** len**-**1**);**

**int** j **=** sa**[**isa**[**i**]-**1**];**

**while** **(**text**[**i**+**len**]** **==** text**[**j**+**len**])** len**++;**

lcp**[**isa**[**i**]]** **=** len**;**

**}**

**}**

* **AC.cpp**

**int** N**;** *// link: suffix link, next: DFA link*

**int** pa**[**MAXN**],** dep**[**MAXN**],** link**[**MAXN**],** next**[**MAXN**][**ALPHA**];**

**int** trie**[**MAXN**][**ALPHA**],** tag**[**MAXN**],** nocc**[**MAXN**];**

*// next and link always exist, others exist iff values != -1.*

*// nocc: next occurrence, first node with tag != -1 along suffix link*

**int** new\_node**(int** p**)** **{**

pa**[**N**]** **=** p**;**

dep**[**N**]** **=** N **==** 0 **?** 0 **:** dep**[**p**]** **+** 1**;**

link**[**N**]** **=** tag**[**N**]** **=** nocc**[**N**]** **=** **-**1**;**

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

next**[**N**][**i**]** **=** 0**;**

trie**[**N**][**i**]** **=** **-**1**;**

**}**

**return** N**++;**

**}**

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

*// Init -> add all strings -> build*

N **=** 0**;**

new\_node**(**0**);**

**}**

**void** add\_string**(int** id**,** **const** string **&**s**)** **{**

**int** v **=** 0**;**

**for** **(int** i **=** 0**;** i **<** s**.**size**();** i**++)** **{**

**int** to **=** s**[**i**]** **-** 'A'**;**

**if** **(**trie**[**v**][**to**]** **==** **-**1**)** **{**

trie**[**v**][**to**]** **=** next**[**v**][**to**]** **=** new\_node**(**v**);**

**}**

v **=** trie**[**v**][**to**];**

**}**

tag**[**v**]** **=** id**;**

**}**

**void** build**()** **{**

queue**<int>** Q**;**

link**[**0**]** **=** 0**;**

Q**.**push**(**0**);**

**while** **(!**Q**.**empty**())** **{**

**int** v **=** Q**.**front**();**

Q**.**pop**();**

**for** **(int** to **=** 0**;** to **<** ALPHA**;** to**++)** **{**

**if** **(**trie**[**v**][**to**]** **!=** **-**1**)** **{**

**int** u **=** trie**[**v**][**to**];**

link**[**u**]** **=** v **==** 0 **?** 0 **:** next**[**link**[**v**]][**to**];**

nocc**[**u**]** **=** tag**[**link**[**u**]]** **!=** **-**1 **?** link**[**u**]** **:** nocc**[**link**[**u**]];**

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

**if** **(**trie**[**u**][**j**]** **==** **-**1**)** **{**

next**[**u**][**j**]** **=** next**[**link**[**u**]][**j**];**

**}**

**}**

Q**.**push**(**u**);**

**}**

**}**

**}**

**}**

* **Hash.cpp**

#define MAXN 1000000

#define prime\_mod 1073676287

**typedef** **long** **long** T**;**

**char** s**[**MAXN**+**5**];**

T h**[**MAXN**+**5**];**

T h\_base**[**MAXN**+**5**];**

**inline** **void** hash\_init**(int** len**,**T prime**=**0xdefaced**){**

h\_base**[**0**]=**1**;**

**for(int** i**=**1**;**i**<=**len**;++**i**){**

h**[**i**]=(**h**[**i**-**1**]\***prime**+**s**[**i**-**1**])%**prime\_mod**;**

h\_base**[**i**]=(**h\_base**[**i**-**1**]\***prime**)%**prime\_mod**;**

**}**

**}**

**inline** T get\_hash**(int** l**,int** r**){**

**return** **(**h**[**r**+**1**]-(**h**[**l**]\***h\_base**[**r**-**l**+**1**])%**prime\_mod**+**prime\_mod**)%**prime\_mod**;**

**}**

* **Treap.cpp**

#include <iostream>

#include <algorithm>

#include <cstdio>

**using** **namespace** std**;**

**const** **int** INF **=** 9e9**;**

**struct** Node**{**

**int** val**,** pri**,** size**,** mi**,** tag**;**

**bool** rev**;**

Node **\***l**,** **\***r**;**

Node**(){}**

Node**(int** v**):**val**(**v**),**pri**(**rand**()),**size**(**1**),**rev**(**0**),**mi**(**v**),**tag**(**0**){** l **=** r **=** NULL**;}**

**void** down**();**

**void** up**();**

**}\***root**;**

**int** Size**(**Node **\***o**){** **return** o**?** o**->**size**:**0**;}**

**int** Min**(**Node **\***o**){** **return** o**?** o**->**mi**:**INF**;}**

**int** Val**(**Node **\***o**){** **return** o**?** o**->**val**:-**1**;}**

**void** Node**::**down**(){**

**if(**tag**){**

val **+=** tag**;**

mi **+=** tag**;**

**if(**l**)** l**->**tag **+=** tag**;**

**if(**r**)** r**->**tag **+=** tag**;**

tag **=** 0**;**

**}**

**if(**rev**){**

swap**(**l**,**r**);**

**if(**l**)** l**->**rev **^=** 1**;**

**if(**r**)** r**->**rev **^=** 1**;**

rev **=** 0**;**

**}**

**}**

**void** Node**::**up**(){**

**if(**l**)** l**->**down**();**

**if(**r**)** r**->**down**();**

size **=** 1 **+** Size**(**l**)** **+** Size**(**r**);**

mi **=** std**::**min**(** min**(**Min**(**l**),** Min**(**r**)),**val **);**

**}**

**void** print**(**Node **\***o**){**

**if(**o**){**

print**(**o**->**l**);**

printf**("%d ",** o**->**val**);**

print**(**o**->**r**);**

**}**

**}**

Node**\*** merge**(**Node**\*** a**,**Node **\***b**){**

**if(!**a **||** **!**b**)** **return** a**?** a**:**b**;**

**if(**a**->**pri **<** b**->**pri**){**

a**->**down**();**

a**->**r **=** merge**(**a**->**r**,** b**);**

a**->**up**();**

**return** a**;**

**}else{**

b**->**down**();**

b**->**l **=** merge**(**a**,** b**->**l**);**

b**->**up**();**

**return** b**;**

**}**

**}**

**void** spilt**(**Node **\***o**,** Node **\*&**a**,** Node **\*&**b**,** **int** k**){**

**if(!**o**)** a **=** b **=** NULL**;**

**else{**

o**->**down**();**

**if(**Size**(**o**->**l**)>=**k**){**

b **=** o**;**

spilt**(**o**->**l**,** a**,** b**->**l**,** k**);**

**}**

**else{**

a **=** o**;**

spilt**(**o**->**r**,** a**->**r**,** b**,** k**-**Size**(**o**->**l**)-**1**);**

**}**

o**->**up**();**

**}**

**}**

**void** Insert**(**Node **\*&**o**,** **int** k**,int** v**){**

**if(!**o**)** o **=** **new** Node**(**v**);**

**else{**

Node**\*** tmp **=** **new** Node**(**v**);**

Node **\***a**,** **\***b**;**

spilt**(**o**,** a**,** b**,** k**);**

o **=** merge**(**merge**(**a**,**tmp**),** b**);**

**}**

**}**

**void** Del**(**Node **\*&**o**,** **int** k**){**

**if(!**o**)** **return;**

**else{**

Node **\***a**,** **\***b**,** **\***c**;**

spilt**(**o**,** a**,** b**,** k**);**

spilt**(**a**,** a**,** c**,** k**-**1**);**

o **=** merge**(**a**,** b**);**

**}**

**}**

**int** Min**(**Node **\*&**o**,** **int** x**,int** y**){**

**if(!**o**)** **return** 0**;**

**else{**

Node **\***a**,** **\***b**,** **\***c**;**

spilt**(**o**,** a**,** b**,** y**);**

spilt**(**a**,** a**,** c**,** x**-**1**);**

**if(**c**==**0**)** **return** 0**;**

c**->**up**();**

**int** ans **=** c**->**mi**;**

o **=** merge**(**merge**(**a**,**c**),** b**);**

**return** ans**;**

**}**

**}**

**void** Add**(**Node **\*&**o**,int** x**,int** y **,int** v**){**

**if(!**o**)** **return;**

Node **\***a**,** **\***b**,** **\***c**;**

spilt**(**o**,** a**,** b**,** y**);**

spilt**(**a**,** a**,** c**,** x**-**1**);**

**if(**c**)** c**->**tag **+=** v**;**

o **=** merge**(**merge**(**a**,**c**),** b**);**

**}**

**void** Reverse**(**Node **\*&**o**,int** x**,int** y**){**

**if(!**o**)** **return;**

Node **\***a**,** **\***b**,** **\***c**;**

spilt**(**o**,** a**,** b**,** y**);** *// a b c*

spilt**(**a**,** a**,** c**,** x**-**1**);**

**if(**c**)** c**->**rev **^=** 1**;**

o **=** merge**(**merge**(**a**,**c**),**b**);**

**}**

**void** Rotate**(**Node **\*&**o**,** **int** x**,int** y**,int** t**){**

**if(!**o**)** **return;**

Node **\***a**,** **\***b**,** **\***c**;**

spilt**(**o**,** a**,** b**,** y**);**

spilt**(**a**,** a**,** c**,** x**-**1**);**

Node **\***d**,** **\***e**;**

t **%=** **(**y**-**x**+**1**);**

**if(**t**<**0**)** t **=** y**-**x**+**1**+**t**;**

spilt**(**c**,**d**,**e**,** Size**(**c**)-**t**);**

c **=** merge**(**e**,** d**);**

o **=** merge**(**merge**(**a**,**c**),**b**);**

**}**

*/\**

*ADD x y D: Add D to each number in sub-sequence {Ax ... Ay}. For example, performing "ADD 2 4 1" on {1, 2, 3, 4, 5} results in {1, 3, 4, 5, 5}*

*REVERSE x y: reverse the sub-sequence {Ax ... Ay}. For example, performing "REVERSE 2 4" on {1, 2, 3, 4, 5} results in {1, 4, 3, 2, 5}*

*REVOLVE x y T: rotate sub-sequence {Ax ... Ay} T times. For example, performing "REVOLVE 2 4 2" on {1, 2, 3, 4, 5} results in {1, 3, 4, 2, 5}*

*INSERT x P: insert P after Ax. For example, performing "INSERT 2 4" on {1, 2, 3, 4, 5} results in {1, 2, 4, 3, 4, 5}*

*DELETE x: delete Ax. For example, performing "DELETE 2" on {1, 2, 3, 4, 5} results in {1, 3, 4, 5}*

*MIN x y: query the participant what is the minimum number in sub-sequence {Ax ... Ay}. For example, the correct answer to "MIN 2 4" on {1, 2, 3, 4, 5} is 2*

*\*/*

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

**{**

**int** n**;**

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

root **=** NULL**;**

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

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

root **=** merge**(**root**,** **new** Node**(**a**));**

**}**

**int** m**,** x**,** y**,** c**;**

**char** s**[**20**];**

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

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

scanf**("%s",** s**);**

**if(**s**[**0**]==**'A'**){**

scanf**("%d%d%d",&**x**,&**y**,&**c**);**

Add**(**root**,** x**,** y**,** c**);**

**}**

**else** **if(**s**[**0**]==**'R' **&&** s**[**3**]==**'E'**){**

scanf**("%d%d",&**x**,&**y**);**

Reverse**(**root**,** x**,** y**);**

**}**

**else** **if(**s**[**0**]==**'R'**){**

scanf**("%d%d%d",&**x**,&**y**,&**c**);**

Rotate**(**root**,** x**,** y**,** c**);**

**}**

**else** **if(**s**[**0**]==**'I'**){**

scanf**("%d%d",&**x**,&**y**);**

n**++;**

Insert**(**root**,** x**,** y**);**

**}**

**else** **if(**s**[**0**]==**'D'**){**

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

Del**(**root**,** x**);**

n**--;**

**}**

**else{**

scanf**("%d%d",&**x**,&**y**);**

printf**("%d\n",** Min**(**root**,** x**,** y**));**

**}**

**}**

**}**

**return** 0**;**

**}**

* **LCA**

*// adj[u] : adjacency list of u*

*// par[u][i] : (2^i)-th parent pf u*

**int** LOG **=** 20**;**

**int** time **=** 0**;**

**void** dfs**(int** u**,** **int** p**)** **{**

par**[**u**][**0**]** **=** p**;**

timer\_in**[**u**]** **=** **++**timer**;**

**for** **(int** v **:** adj**[**u**])** **if** **(**v**!=**p**)** dfs**(**v**,** u**);**

time\_out**[**u**]** **=** **++**timer**;**

**}**

**bool** anc**(int** x**,** **int** y**)** **{**

**return** time\_in**[**x**]** **<=** time\_in**[**y**]**

**&&** time\_out**[**y**]** **<=** time\_out**[**x**];**

**}**

**int** lca**(int** x**,** **int** y**)** **{**

**if** **(**anc**(**y**,** x**))** **return** y**;**

**for** **(int** j **=** LOG**;** j **>=** 0**;** j**--)** **{**

**if** **(!**anc**(**par**[**y**][**j**],** x**))** y **=** par**[**y**][**j**];**

**}**

**return** par**[**y**][**0**];**

**}**

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

**int** root **=** 1**;***// set root node*

dfs**(**root**,** root**);**

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

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

pair**[**i**][**j**]** **=** par**[**par**[**i**][**j **-** 1**]][**j **-** 1**];**

**return** 0**;**

**}**

* **樹鍊剖分**

*// In this template value is on the edge, everything is 1-based*

**int** N**;**

vector**<**Edge**>** G**[**MAXN**+**5**];**

*// Preprocess info, setup in dfs1*

**int** heavy**[**MAXN**+**5**],** pa\_w**[**MAXN**+**5**],** sz**[**MAXN**+**5**];**

**int** pa**[**MAXN**+**5**],** dep**[**MAXN**+**5**],** recorder**[**MAXN**+**5**];** *// Which node record edge i.*

*// HLD info, setup in build, 1-based*

*// pos: position of node i in seg tree.*

*// head: For NODE i, points to head of the chain.*

**int** chain\_no**,** border**,** pos**[**MAXN**+**5**],** head**[**MAXN**+**5**];**

**void** dfs1**(int** v**,** **int** p**)** **{**

pa**[**v**]** **=** p**;**

sz**[**v**]** **=** 1**;**

dep**[**v**]** **=** dep**[**p**]** **+** 1**;**

heavy**[**v**]** **=** **-**1**;**

**for** **(const** Edge **&**e **:** G**[**v**])** **{**

**if** **(**e**.**to **==** p**)** **continue;**

dfs1**(**e**.**to**,** v**);**

pa\_w**[**e**.**to**]** **=** e**.**w**;**

recorder**[**e**.**id**]** **=** e**.**to**;**

sz**[**v**]** **+=** sz**[**e**.**to**];**

**if** **(**heavy**[**v**]** **==** **-**1 **||** sz**[**e**.**to**]** **>** sz**[**heavy**[**v**]])** **{**

heavy**[**v**]** **=** e**.**to**;**

**}**

**}**

**}**

**void** build**(int** v**,** **int** chain\_head**)** **{**

pos**[**v**]** **=** **++**border**;**

head**[**v**]** **=** chain\_head**;**

tree**.**update**(**pos**[**v**],** pa\_w**[**v**],** 1**,** N**,** 1**);**

**if** **(**heavy**[**v**]** **!=** **-**1**)** build**(**heavy**[**v**],** chain\_head**);**

**for** **(const** Edge **&**e **:** G**[**v**])** **{**

**if** **(**e**.**to **==** pa**[**v**]** **||** e**.**to **==** heavy**[**v**])** **continue;**

build**(**e**.**to**,** e**.**to**);**

**}**

**}**

**void** init\_HLD**()** **{**

*/\* Only init used data, be careful. \*/*

*/\* Does not init G!!!!! \*/*

border **=** dep**[**1**]** **=** pa\_w**[**1**]** **=** 0**;**

dfs1**(**1**,** 1**);**

build**(**1**,** 1**);**

**}**

**int** query\_up**(int** a**,** **int** b**)** **{**

**int** ans **=** 0**;**

**while** **(**head**[**a**]** **!=** head**[**b**])** **{**

**if** **(**dep**[**head**[**a**]]** **<** dep**[**head**[**b**]])** swap**(**a**,** b**);**

ans **=** max**(**ans**,** tree**.**query**(**pos**[**head**[**a**]],** pos**[**a**],** 1**,** N**,** 1**));**

a **=** pa**[**head**[**a**]];**

**}**

**if** **(**a **==** b**)** **return** ans**;**

**if** **(**dep**[**a**]** **<** dep**[**b**])** swap**(**a**,** b**);**

*// Query range is pos[b] if value is on node.*

ans **=** max**(**ans**,** tree**.**query**(**pos**[**b**]** **+** 1**,** pos**[**a**],** 1**,** N**,** 1**));**

**return** ans**;**

**}**

* **SCC.cpp**

**struct** Kosaraju **{**

*// Vertex i belong to which SCC, call findSCC to build.*

**int** SCCof**[**MAXV**+**5**],**V**,**cnt**;**

**bool** vis**[**MAXV**+**5**];**

vector**<int>** **\***G**,\***Grev**;**

stack**<int>** stk**;**

**void** dfs**(**vector**<int>** **\***Gcur**,** **int** v**)** **{**

**for** **(auto** u **:** Gcur**[**v**])** **{**

**if** **(!**vis**[**u**])** **{**

vis**[**u**]=true;**

dfs**(**Gcur**,**u**);**

**}**

**}**

**if** **(**Gcur**==**G**)** stk**.**push**(**v**);**

**else** SCCof**[**v**]=**cnt**;**

**}**

**int** findSCC**(int** \_V**,** vector**<int>** **\***\_G**,** vector**<int>** **\***\_Grev**)** **{**

*// G: Adjacency list of graph. Grev: Reverse graph of G.*

*// No need for init, return # of SCC, 1-based*

V**=**\_V**;** G**=**\_G**;** Grev**=**\_Grev**;**

**for** **(int** i**=**1**;**i**<=**V**;**i**++)** vis**[**i**]=**0**;**

**for** **(int** i**=**1**;**i**<=**V**;**i**++)** **{**

**if** **(!**vis**[**i**])** **{**

vis**[**i**]=true;**

dfs**(**G**,**i**);**

**}**

**}**

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

**for** **(int** i**=**1**;**i**<=**V**;**i**++)** vis**[**i**]=**0**;**

**while** **(!**stk**.**empty**())** **{**

**int** v**=**stk**.**top**();**

stk**.**pop**();**

**if** **(!**vis**[**v**])** **{**

cnt**++;**

vis**[**v**]=true;**

dfs**(**Grev**,**v**);**

**}**

**}**

**return** cnt**;**

**}**

**void** compress**(**vector**<int>** **\***Gtar**)** **{**

*// Pack SCC into one vertex, store into Gtar*

*// Call findSCC before this, 1-based*

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

**for** **(auto** j **:** G**[**i**])**

**if** **(**SCCof**[**i**]!=**SCCof**[**j**])**

Gtar**[**SCCof**[**i**]].**push\_back**(**SCCof**[**j**]);**

**}**

**};**

* **DFS flow**

**struct** Edge**{**

**int** to**,**cap**,**rev**;**

Edge**(int** a**,int** b**,int** c**)** **{**

to **=** a**;** cap **=** b**;** rev **=** c**;**

**}**

**};**

*// IMPORETANT, MAXV != MAXN*

vector**<**Edge**>** G**[**MAXV**];**

**int** V**,** flow**[**MAXV**];**

**void** init**(int** \_V**){**

V **=** \_V**;**

**for(int** i**=**0**;** i**<=**V**;** i**++)** G**[**i**].**clear**();**

**}**

**void** add\_edge**(int** f**,int** t**,int** c**,** **bool** directed**){**

**int** s1 **=** G**[**f**].**size**(),** s2 **=** G**[**t**].**size**();**

G**[**f**].**push\_back**(**Edge**(**t**,**c**,**s2**));**

G**[**t**].**push\_back**(**Edge**(**f**,**c**\*!**directed**,**s1**));**

**}**

**int** dfs**(int** v**,** **int** t**)** **{**

**if(**v **==** t**)** **return** flow**[**t**];**

**for(**Edge **&**e **:** G**[**v**]){**

**if(**e**.**cap**==**0**||**flow**[**e**.**to**]!=-**1**)** **continue;**

flow**[**e**.**to**]** **=** min**(**flow**[**v**],** e**.**cap**);**

**int** f **=** dfs**(**e**.**to**,** t**);**

**if** **(**f**!=**0**)** **{**

e**.**cap **-=** f**;**

G**[**e**.**to**][**e**.**rev**].**cap **+=** f**;**

**return** f**;**

**}**

**}**

**return** 0**;**

**}**

**int** max\_flow**(int** s**,int** t**){**

**int** ans **=** 0**,** add **=** 0**;**

**do** **{**

fill**(**flow**,**flow**+**V**+**1**,-**1**);**

flow**[**s**]** **=** INF**;**

add **=** dfs**(**s**,** t**);**

ans **+=** add**;**

**}** **while** **(**add **!=** 0**);**

**return** ans**;**

**}**

* **Dinic flow**

**const** **int** maxn **=** 100**+**10**;**

**const** **int** INF **=** 2147483647**;**

**struct** Edge**{**

**int** f**,**to**;**

T c**;**

Edge**(int** \_f**,int** \_to**,**T \_c**):**f**(**\_f**),**to**(**\_to**),**c**(**\_c**){}**

**};**

*// IMPORETANT*

*// maxn is the number of vertices in the graph*

*// Not the N in the problem statement!!*

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

vector**<**Edge**>** es**;**

**int** level**[**maxn**],**st**,** end**,** n**;**

**int** cur**[**maxn**];**

**void** init**(int** \_n**){**

n **=** \_n**;**

es**.**clear**();**

**for(int** i**=**0**;** i**<=**n**;** i**++)** G**[**i**].**clear**();**

**}**

**void** addEdge**(int** f**,int** t**,**T c**,** **bool** directed**=false){**

es**.**push\_back**(**Edge**(**f**,**t**,**c**));** G**[**f**].**push\_back**(**es**.**size**()-**1**);**

es**.**push\_back**(**Edge**(**t**,**f**,**directed**?**0**:**c**));** G**[**t**].**push\_back**(**es**.**size**()-**1**);**

**}**

**bool** BFS**(int** s**,int** t**){**

queue**<int>** Q**;**

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

level**[**s**]** **=** 1**;**

Q**.**push**(**s**);**

**while(!**Q**.**empty**()){**

**int** x **=** Q**.**front**();** Q**.**pop**();**

**for(int** i**=**0**;** i**<**G**[**x**].**size**();** i**++){**

Edge e **=** es**[** G**[**x**][**i**]** **];**

**if(**e**.**c**==**0 **||** level**[**e**.**to**])** **continue;**

level**[**e**.**to**]** **=** level**[**x**]** **+** 1**;**

Q**.**push**(**e**.**to**);**

**}**

**}**

**return** level**[**t**]!=**0**;**

**}**

T DFS**(int** s**,**T cur\_flow**){** *// can't exceed c*

**if(**s**==**end**)** **return** cur\_flow**;**

T ans **=** 0**,** temp**,** total **=** 0**;**

**for(int&** i**=**cur**[**s**];** i**<**G**[**s**].**size**();** i**++){**

Edge **&**e **=** es**[** G**[**s**][**i**]** **];**

**if(**e**.**c**==**0 **||** level**[**e**.**to**]!=**level**[**s**]+**1**)** **continue;**

temp **=** DFS**(**e**.**to**,** min**(**e**.**c**,** cur\_flow**));**

**if(**temp**!=**0**){**

e**.**c **-=** temp**;**

es**[**G**[**s**][**i**]^**1**].**c **+=** temp**;**

cur\_flow **-=** temp**;**

total **+=** temp**;**

**if(**cur\_flow**==**0**)** **break;**

**}**

**}**

**return** total**;**

**}**

T maxFlow**(int** s**,int** t**){**

*/\* If you want to incrementally doing maxFlow,*

*you need to add the result manually.*

*This function returns difference in that case.\*/*

T ans **=** 0**;**

st **=** s**,** end **=** t**;**

**while(**BFS**(**s**,**t**)){**

**while(true){**

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

T temp **=** DFS**(**s**,**INF**);**

**if(**temp**==**0**)** **break;**

ans **+=** temp**;**

**}**

**}**

**return** ans**;**

**}**

* **最小費用流**

*// 0-based*

#define fi first

#define se second

**struct** Edge **{**

**int** to**,**cap**;**

**int** cost**,**rev**;**

**};**

**static** **const** **int** MAXV **=** 605**;**

**int** V**,**E**;**

vector**<**Edge**>** G**[**MAXV**];**

**void** init**(int** \_V**)** **{**

V**=**\_V**;**

**for** **(int** i**=**0**;**i**<=**V**;**i**++)** G**[**i**].**clear**();**

**}**

**void** add\_edge**(int** fr**,** **int** to**,** **int** cap**,** **int** cost**)** **{**

**int** a **=** G**[**fr**].**size**(),** b **=** G**[**to**].**size**();**

G**[**fr**].**push\_back**({**to**,**cap**,**cost**,**b**});**

G**[**to**].**push\_back**({**fr**,**0**,-**cost**,**a**});**

**}**

**bool** SPFA**(int** s**,** **int** t**,** **int** **&**ans\_flow**,** **int** **&**ans\_cost**)** **{**

queue**<int>** que**;**

PII pre**[**MAXV**];**

**int** flow**[**MAXV**],** dist**[**MAXV**];**

**bool** inque**[**MAXV**];**

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

dist**[**i**]=**INF**;**

inque**[**i**]=false;**

**}**

dist**[**s**]=**0**;**

flow**[**s**]=**INF**;**

inque**[**s**]=true;**

que**.**push**(**s**);**

**while** **(!**que**.**empty**())** **{**

**int** v**=**que**.**front**();** que**.**pop**();**

inque**[**v**]=false;**

**for** **(int** i**=**0**;**i**<**G**[**v**].**size**();**i**++)** **{**

**const** Edge **&**e **=** G**[**v**][**i**];**

**if** **(**e**.**cap**>**0 **&&** dist**[**v**]+**e**.**cost**<**dist**[**e**.**to**])** **{**

flow**[**e**.**to**]=**min**(**flow**[**v**],**e**.**cap**);**

dist**[**e**.**to**]=**dist**[**v**]+**e**.**cost**;**

pre**[**e**.**to**]={**v**,**i**};**

**if** **(!**inque**[**e**.**to**])** que**.**push**(**e**.**to**),**inque**[**e**.**to**]=true;**

**}**

**}**

**}**

**if** **(**dist**[**t**]==**INF**)** **return** **false;**

*//if (dist[t]>=0) return false;*

*// Add above line -> min cost > max flow (priority)*

*// Without -> max flow > min cost*

**int** v**=**t**,**f**=**flow**[**t**];**

ans\_flow**+=**flow**[**t**];**

ans\_cost**+=(**dist**[**t**]\***flow**[**t**]);**

**while** **(**v**!=**s**)** **{**

Edge **&**e **=** G**[**pre**[**v**].**fi**][**pre**[**v**].**se**];**

e**.**cap**-=**f**;**

G**[**v**][**e**.**rev**].**cap**+=**f**;**

v**=**pre**[**v**].**fi**;**

**}**

**return** **true;**

**}**

pair**<int,int>** min\_cost\_flow**(int** s**,** **int** t**)** **{**

**int** ans\_flow**=**0**,** ans\_cost**=**0**;**

**while** **(**SPFA**(**s**,**t**,**ans\_flow**,**ans\_cost**));**

**return** make\_pair**(**ans\_flow**,**ans\_cost**);**

**}**

* **Blossom matching**

**struct** Blossom **{**

#define MAXN 505 *// Max solvable problem, DON'T CHANGE*

*// 1-based, IMPORTANT*

vector**<int>** g**[**MAXN**];**

**int** parent**[**MAXN**],** match**[**MAXN**],** belong**[**MAXN**],** state**[**MAXN**];**

**int** n**;**

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

**static** **int** cases **=** 0**,** used**[**MAXN**]** **=** **{};**

**for** **(++**cases**;** **;** swap**(**u**,** v**))** **{**

**if** **(**u **==** 0**)**

**continue;**

**if** **(**used**[**u**]** **==** cases**)**

**return** u**;**

used**[**u**]** **=** cases**;**

u **=** belong**[**parent**[**match**[**u**]]];**

**}**

**}**

**void** flower**(int** u**,** **int** v**,** **int** l**,** queue**<int>** **&**q**)** **{**

**while** **(**belong**[**u**]** **!=** l**)** **{**

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

**if** **(**state**[**v**]** **==** 1**)**

q**.**push**(**v**),** state**[**v**]** **=** 0**;**

belong**[**u**]** **=** belong**[**v**]** **=** l**,** u **=** parent**[**v**];**

**}**

**}**

**bool** bfs**(int** u**)** **{**

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

belong**[**i**]** **=** i**;**

memset**(**state**,** **-**1**,** **sizeof(**state**[**0**])\*(**n**+**1**));**

queue**<int>** q**;**

q**.**push**(**u**),** state**[**u**]** **=** 0**;**

**while** **(!**q**.**empty**())** **{**

u **=** q**.**front**(),** q**.**pop**();**

**for** **(int** i **=** 0**;** i **<** g**[**u**].**size**();** i**++)** **{**

**int** v **=** g**[**u**][**i**];**

**if** **(**state**[**v**]** **==** **-**1**)** **{**

parent**[**v**]** **=** u**,** state**[**v**]** **=** 1**;**

**if** **(**match**[**v**]** **==** 0**)** **{**

**for** **(int** prev**;** u**;** v **=** prev**,** u **=** parent**[**v**])** **{**

prev **=** match**[**u**];**

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

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

**}**

**return** 1**;**

**}**

q**.**push**(**match**[**v**]),** state**[**match**[**v**]]** **=** 0**;**

**}** **else** **if** **(**state**[**v**]** **==** 0 **&&** belong**[**v**]** **!=** belong**[**u**])** **{**

**int** l **=** lca**(**u**,** v**);**

flower**(**v**,** u**,** l**,** q**);**

flower**(**u**,** v**,** l**,** q**);**

**}**

**}**

**}**

**return** 0**;**

**}**

**int** blossom**()** **{**

memset**(**parent**,** 0**,** **sizeof(**parent**[**0**])\*(**n**+**1**));**

memset**(**match**,** 0**,** **sizeof(**match**[**0**])\*(**n**+**1**));**

**int** ret **=** 0**;**

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

**if** **(**match**[**i**]** **==** 0 **&&** bfs**(**i**))**

ret**++;**

**}**

**return** ret**;**

**}**

**void** addEdge**(int** x**,** **int** y**)** **{**

g**[**x**].**push\_back**(**y**),** g**[**y**].**push\_back**(**x**);**

**}**

**void** init**(int** \_n**)** **{**

n **=** \_n**;**

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

g**[**i**].**clear**();**

**}**

**}** algo**;**

* **穩定婚姻**

**const** **int** maxn **=** 1100**;**

**int** manWant**[**maxn**][**maxn**],** nextW**[**maxn**];**

**int** women**[**maxn**][**maxn**],** order**[**maxn**][**maxn**];**

**int** wife**[**maxn**],** husband**[**maxn**];**

queue**<int>** singleDog**;**

**void** engage**(int** m**,** **int** w**){**

**if(**husband**[**w**]!=**0**){**

wife**[** husband**[**w**]** **]** **=** 0**;**

singleDog**.**push**(** husband**[**w**]** **);**

husband**[**w**]** **=** 0**;**

**}**

husband**[**w**]** **=** m**;**

wife**[**m**]** **=** w**;**

*// cout << m << " --> " << w << endl;*

**}**

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

**{**

**int** Time**,** n**,** cas **=** 0**;**

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

**while(**Time**--** **&&** scanf**("%d",&**n**)==**1**){**

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

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

nextW**[**i**]** **=** 1**;**

wife**[**i**]** **=** 0**;**

singleDog**.**push**(**i**);**

**}**

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

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

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

order**[**i**][** women**[**i**][**j**]** **]** **=** j**;**

**}**

husband**[**i**]** **=** 0**;**

**}**

**while(!**singleDog**.**empty**()){**

**int** x **=** singleDog**.**front**();** singleDog**.**pop**();**

*// cout << x << endl;*

**int** to **=** manWant**[**x**][**nextW**[**x**]++];**

**if(**husband**[**to**]==**0**)** engage**(**x**,** to**);**

**else** **if(**order**[**to**][**husband**[**to**]]** **>** order**[**to**][**x**])** engage**(**x**,** to**);**

**else** singleDog**.**push**(**x**);**

**}**

**if(**cas**++)** printf**("\n");**

**for(int** i**=**1**;** i**<=**n**;** i**++)** printf**("%d\n",** wife**[**i**]);**

**}**

**return** 0**;**

**}**

* **General graph min perfect mat**

**struct** Graph **{**

*// Minimum General Weighted Matching (Perfect Match) 0-base*

**static** **const** **int** MXN **=** 105**;**

**int** n**,** edge**[**MXN**][**MXN**];**

**int** match**[**MXN**],**dis**[**MXN**],**onstk**[**MXN**];**

vector**<int>** stk**;**

**void** init**(int** \_n**)** **{**

n **=** \_n**;**

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

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

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

**}**

**void** add\_edge**(int** u**,** **int** v**,** **int** w**)** **{**

edge**[**u**][**v**]** **=** edge**[**v**][**u**]** **=** w**;**

**}**

**bool** SPFA**(int** u**){**

**if** **(**onstk**[**u**])** **return** **true;**

stk**.**push\_back**(**u**);**

onstk**[**u**]** **=** 1**;**

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

**if** **(**u **!=** v **&&** match**[**u**]** **!=** v **&&** **!**onstk**[**v**]){**

**int** m **=** match**[**v**];**

**if** **(**dis**[**m**]** **>** dis**[**u**]** **-** edge**[**v**][**m**]** **+** edge**[**u**][**v**]){**

dis**[**m**]** **=** dis**[**u**]** **-** edge**[**v**][**m**]** **+** edge**[**u**][**v**];**

onstk**[**v**]** **=** 1**;**

stk**.**push\_back**(**v**);**

**if** **(**SPFA**(**m**))** **return** **true;**

stk**.**pop\_back**();**

onstk**[**v**]** **=** 0**;**

**}**

**}**

**}**

onstk**[**u**]** **=** 0**;**

stk**.**pop\_back**();**

**return** **false;**

**}**

**int** solve**()** **{**

*// find a match*

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

match**[**i**]** **=** i**+**1**,** match**[**i**+**1**]** **=** i**;**

**}**

**for(;;){**

**int** found **=** 0**;**

**for** **(int** i**=**0**;** i**<**n**;** i**++)** dis**[**i**]** **=** onstk**[**i**]** **=** 0**;**

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

stk**.**clear**();**

**if** **(!**onstk**[**i**]** **&&** SPFA**(**i**)){**

found **=** 1**;**

**while** **(**stk**.**size**()>=**2**){**

**int** u **=** stk**.**back**();** stk**.**pop\_back**();**

**int** v **=** stk**.**back**();** stk**.**pop\_back**();**

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

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

**}**

**}**

**}**

**if** **(!**found**)** **break;**

**}**

**int** ret **=** 0**;**

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

ret **+=** edge**[**i**][**match**[**i**]];**

ret **/=** 2**;**

**return** ret**;**

**}**

**}**graph**;**

* **KM**

*// Maximum Bipartite Weighted Matching (Perfect Match)*

**static** **const** **int** MXN **=** 650**;**

**static** **const** **int** INF **=** 2147483647**;** *// LL*

**int** n**,**match**[**MXN**],**vx**[**MXN**],**vy**[**MXN**];**

**int** edge**[**MXN**][**MXN**],**lx**[**MXN**],**ly**[**MXN**],**slack**[**MXN**];**

*// ^^^^ LL*

**void** init**(int** \_n**){**

n **=** \_n**;**

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

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

**}**

**void** addEdge**(int** x**,** **int** y**,** **int** w**)** *// LL*

**{** edge**[**x**][**y**]** **=** w**;** **}**

**bool** DFS**(int** x**){**

vx**[**x**]** **=** 1**;**

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

**if** **(**vy**[**y**])** **continue;**

**if** **(**lx**[**x**]+**ly**[**y**]** **>** edge**[**x**][**y**]){**

slack**[**y**]=**min**(**slack**[**y**],** lx**[**x**]+**ly**[**y**]-**edge**[**x**][**y**]);**

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

vy**[**y**]** **=** 1**;**

**if** **(**match**[**y**]** **==** **-**1 **||** DFS**(**match**[**y**]))**

**{** match**[**y**]** **=** x**;** **return** **true;** **}**

**}**

**}**

**return** **false;**

**}**

**int** solve**(){**

fill**(**match**,**match**+**n**,-**1**);**

fill**(**lx**,**lx**+**n**,-**INF**);** fill**(**ly**,**ly**+**n**,**0**);**

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

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

lx**[**i**]** **=** max**(**lx**[**i**],** edge**[**i**][**j**]);**

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

fill**(**slack**,**slack**+**n**,**INF**);**

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

fill**(**vx**,**vx**+**n**,**0**);** fill**(**vy**,**vy**+**n**,**0**);**

**if** **(** DFS**(**i**)** **)** **break;**

**int** d **=** INF**;** *// long long*

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

**if** **(!**vy**[**j**])** d **=** min**(**d**,** slack**[**j**]);**

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

**if** **(**vx**[**j**])** lx**[**j**]** **-=** d**;**

**if** **(**vy**[**j**])** ly**[**j**]** **+=** d**;**

**else** slack**[**j**]** **-=** d**;**

**}**

**}**

**}**

**int** res**=**0**;**

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

res **+=** edge**[**match**[**i**]][**i**];**

**return** res**;**

**}**

* **FFT**

**const** **double** PI **=** acos**(-**1**);**

**using** cd **=** complex**<double>;**

*// Do FFT. invert=true to do iFFT.*

*// n MUST be power of 2.*

**void** fft**(**cd a**[],** **int** n**,** **bool** invert**)** **{**

**for** **(int** i **=** 1**,** j **=** 0**;** i **<** n**;** i**++)** **{**

**int** bit **=** n **>>** 1**;**

**for** **(;** j **&** bit**;** bit **>>=** 1**)**

j **^=** bit**;**

j **^=** bit**;**

**if** **(**i **<** j**)**

swap**(**a**[**i**],** a**[**j**]);**

**}**

**for** **(int** len **=** 2**;** len **<=** n**;** len **<<=** 1**)** **{**

**double** ang **=** 2 **\*** PI **/** len **\*** **(**invert **?** **-**1 **:** 1**);**

cd wlen**(**cos**(**ang**),** sin**(**ang**));**

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

cd w**(**1**);**

**for** **(int** j **=** 0**;** j **<** len **/** 2**;** j**++)** **{**

cd u **=** a**[**i**+**j**],** v **=** a**[**i**+**j**+**len**/**2**]** **\*** w**;**

a**[**i**+**j**]** **=** u **+** v**;**

a**[**i**+**j**+**len**/**2**]** **=** u **-** v**;**

w **\*=** wlen**;**

**}**

**}**

**}**

**if** **(**invert**)** **{**

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

a**[**i**]** **/=** n**;**

**}**

**}**

* **NTT**

**const** LL mod **=** 998244353**;**

**const** LL p\_root **=** 3**;**

**const** LL root\_pw **=** 1LL **<<** 23**;**

*// Do NTT under mod. invert=true to do iNTT.*

*// mod MUST be a prime, if mod=c\*2^k+1, then*

*// p\_root is any primitive root of mod*

*// root\_pw=2^k, and n(size) MUST <= 2^k*

*// n MUST be power of 2.*

*// mod=2013265921, root\_pw=1LL<<27, p\_root=31*

**void** ntt**(**LL a**[],** **int** n**,** **bool** invert**)** **{**

LL root **=** qpow**(**p\_root**,** **(**mod**-**1**)/**root\_pw**,** mod**);**

LL root\_1 **=** mod\_inv**(**root**,** mod**);**

**for** **(int** i **=** 1**,** j **=** 0**;** i **<** n**;** i**++)** **{**

LL bit **=** n **>>** 1**;**

**for** **(;** j **&** bit**;** bit **>>=** 1**)**

j **^=** bit**;**

j **^=** bit**;**

**if** **(**i **<** j**)**

swap**(**a**[**i**],** a**[**j**]);**

**}**

**for** **(int** len **=** 2**;** len **<=** n**;** len **<<=** 1**)** **{**

LL wlen **=** invert **?** root\_1 **:** root**;**

**for** **(int** i **=** len**;** i **<** root\_pw**;** i **<<=** 1**)**

wlen **=** wlen **\*** wlen **%** mod**;**

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

LL w **=** 1**;**

**for** **(int** j **=** 0**;** j **<** len **/** 2**;** j**++)** **{**

LL u **=** a**[**i**+**j**],** v **=** a**[**i**+**j**+**len**/**2**]** **\*** w **%** mod**;**

a**[**i**+**j**]** **=** u **+** v **<** mod **?** u **+** v **:** u **+** v **-** mod**;**

a**[**i**+**j**+**len**/**2**]** **=** u **-** v **>=** 0 **?** u **-** v **:** u **-** v **+** mod**;**

w **=** w **\*** wlen **%** mod**;**

**}**

**}**

**}**

**if** **(**invert**)** **{**

LL n\_1 **=** mod\_inv**(**n**,** mod**);**

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

a**[**i**]** **=** a**[**i**]** **\*** n\_1 **%** mod**;**

**}**

**}**

**}**

* **Primitive root**

**int** generator **(int** p**)** **{**

*// Return primitive root of p*

*// Exist is p is 2, 4 or odd prime power or 2 \* odd prime power*

*// O(ANS \* log^2(p))*

vector**<int>** fact**;**

**int** phi **=** p**-**1**,** n **=** phi**;**

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

**if** **(**n **%** i **==** 0**)** **{**

fact**.**push\_back **(**i**);**

**while** **(**n **%** i **==** 0**)**

n **/=** i**;**

**}**

**}**

**if** **(**n **>** 1**)** fact**.**push\_back**(**n**);**

**for** **(int** res**=**2**;** res**<=**p**;** **++**res**)** **{**

**bool** ok **=** **true;**

**for** **(**size\_t i**=**0**;** i**<**fact**.**size**()** **&&** ok**;** **++**i**)**

ok **&=** powmod **(**res**,** phi **/** fact**[**i**],** p**)** **!=** 1**;**

**if** **(**ok**)** **return** res**;**

**}**

**return** **-**1**;**

**}**

* **EXT\_GCD**

**typedef** pair **<** LL**,** LL**>** ii**;**

ii exd\_gcd**(** LL a**,** LL b**)** **{**

**if** **(**a **%** b **==** 0**)** **return** ii**(**0**,** 1**);**

ii T **=** exd\_gcd**(**b**,** a **%** b**);**

**return** ii**(** T**.**second**,** T**.**first **-** a **/** b **\*** T**.**second**);**

**}**

LL mod\_inv**(**LL x**)** **{**

*// P is mod number, gcd(x,P) must be 1*

**return** **(**exd\_gcd**(**x**,**P**).**first**%**P**+**P**)%**P**;**

**}**

* **LUCAS**

**struct** Lucas **{**

*// P is mod number, must be prime*

LL fac**[**MAXP**+**5**],**P**;**

**void** init**(**LL \_P**)** **{**

P**=**\_P**;**

LL i**;**

fac**[**0**]** **=**1**;**

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

fac**[**i**]** **=** fac**[**i**-**1**]\***i **%** P**;**

**}**

**}**

LL qpow**(**LL a**,** LL p**)** **{**

**if** **(**p**<=**0**)** **return** 1**;**

LL tmp**=**qpow**(**a**,**p**/**2**);**

**if** **(**p**&**1**)** **return** tmp**\***tmp**%**P**\***a**%**P**;**

**return** tmp**\***tmp**%**P**;**

**}**

LL C**(**LL n**,** LL k**)** **{**

**if(**k **>** n**)** **return** 0**;**

**return** fac**[**n**]\***qpow**(**fac**[**k**]\***fac**[**n**-**k**]%**P**,** P**-**2**)** **%** P**;**

**}**

LL lucas**(**LL n**,** LL k **)** **{**

**if(**k **==**0**)** **return** 1**;**

**else** **return** **(**C**(**n**%**P**,** k**%**P**)\***lucas**(**n**/**P**,** k**/**P**))%**P**;**

**}**

**};**

* **Miller Rabin**

LL mod\_mul**(**LL a**,** LL b**,** LL mod**)** **{**

*// return (\_\_int128)a\*b%mod;*

*/\* In case \_\_int128 doesn't work(32\* multi to avoid ovf) \*/*

LL x**=**0**,**y**=**a**%**mod**;**

**while(**b **>** 0**){**

**if** **(**b**&**1**)** x **=** **(**x**+**y**)%**mod**;**

y **=** **(**y**\***2**)%**mod**;**

b **>>=** 1**;**

**}**

**return** x**%**mod**;**

**}**

LL qpow**(**LL a**,** LL p**,** LL mod**)** **{**

**if** **(**p**<=**0**)** **return** 1**;**

LL temp **=** qpow**(**a**,**p**/**2**,**mod**);**

temp **=** mod\_mul**(**temp**,**temp**,**mod**);**

**if** **(**p**&**1**)** **return** mod\_mul**(**temp**,**a**,**mod**);**

**return** temp**;**

**}**

**bool** MRtest**(**LL a**,** LL d**,** LL n**)** **{**

LL x **=** qpow**(**a**,**d**,**n**);**

**if** **(**x**==**1 **||** x**==**n**-**1**)** **return** **true;**

**while** **(**d **!=** n**-**1**)** **{**

x **=** mod\_mul**(**x**,**x**,**n**);**

d **\*=** 2**;**

**if** **(**x**==**n**-**1**)** **return** **true;**

**if** **(**x**==**1**)** **return** **false;**

**}**

**return** **false;**

**}**

**bool** is\_prime**(**LL n**)** **{**

**if** **(**n**==**2**)** **return** **true;**

**if** **(**n**<**2 **||** n**%**2**==**0**)** **return** **false;**

LL table**[**7**]** **=** **{**2**,** 325**,** 9375**,** 28178**,** 450775**,** 9780504**,** 1795265022**},** d**=**n**-**1**;**

**while** **(**d**%**2 **!=** 0**)** d**>>=**1**;** *// n-1 = d \* 2^r, d is odd.*

**for** **(int** i**=**0**;** i**<**7**;** i**++)** **{**

LL a **=** table**[**i**]** **%** n**;**

**if** **(**a**==**0 **||** a**==**1 **||** a**==**n**-**1**)** **continue;**

**if** **(!**MRtest**(**a**,**d**,**n**))** **{**

**return** **false;**

**}**

**}**

**return** **true;**

**}**

* **Computational Geometry**

**const** **double** PI**=**acos**(-**1**);**

**struct** Point **{**

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

**double** cross**(const** Point **&**v**)** **const** **{**

**return** x**\***v**.**y**-**y**\***v**.**x**;**

**}**

**double** dot**(const** Point **&**v**)** **const** **{**

**return** x**\***v**.**x**+**y**\***v**.**y**;**

**}**

Point normal**()** **{** *// Normal vector to the left*

**return** **{-**y**,**x**};**

**}**

**double** angle**(const** Point **&**v**)** **const** **{**

*// Angle from \*this to v in [-pi,pi].*

**double** ang **=** atan2**(**cross**(**v**),**dot**(**v**));**

**return** ang **<** 0 **?** ang **+** PI **\*** 2 **:** ang**;**

**}**

**double** getA**()const{***//angle to x-axis*

T A**=**atan2**(**y**,**x**);***//<0 when exceed PI*

**if(**A**<=-**PI**/**2**)**A**+=**PI**\***2**;**

**return** A**;**

**}**

Point rotate\_about**(double** theta**,** **const** Point **&**p**)** **const** **{**

*// Rotate this point conterclockwise by theta about p*

**double** nx**=**x**-**p**.**x**,**ny**=**y**-**p**.**y**;**

**return** **{**nx**\***cos**(**theta**)-**ny**\***sin**(**theta**)+**p**.**x**,**nx**\***sin**(**theta**)+**ny**\***cos**(**theta**)+**p**.**y**};**

**}**

**};**

**struct** Line **{**

*// IMPORTANT, remember to transform between two-point form*

*// and normal form by yourself, some methods may need them.*

Point p1**,**p2**;**

**double** a**,**b**,**c**;** *// ax+by+c=0*

Line**(){}**

**void** pton**()** **{**

a**=**p1**.**y**-**p2**.**y**;**

b**=**p2**.**x**-**p1**.**x**;**

c**=-**a**\***p1**.**x**-**b**\***p1**.**y**;**

**}**

**int** relation**(const** Point **&**p**)** **{**

*// For line, 0 if point on line*

*// -1 if left, 1 if right*

Point dir**=**p2**-**p1**;**

**double** crs**=**dir**.**cross**(**p**-**p1**);**

**return** crs**==**0**?**0**:**crs**<**0**?-**1**:**1**;**

**}**

Point normal**()** **{** *// normal vector to the left.*

Point dir**=**p2**-**p1**;**

**return** **{-**dir**.**y**,**dir**.**x**};**

**}**

**bool** on\_segment**(const** Point **&**p**)** **{**

*// Point on segment*

**return** relation**(**p**)==**0**&&(**p2**-**p**).**dot**(**p1**-**p**)<=**0**;**

**}**

**bool** parallel**(const** Line **&**l**)** **{**

*// Two line parallel*

**return** **(**p2**-**p1**).**cross**(**l**.**p2**-**l**.**p1**)==**0**;**

**}**

**bool** equal**(const** Line **&**l**)** **{**

*// Two line equal*

**return** relation**(**l**.**p1**)==**0**&&**relation**(**l**.**p2**)==**0**;**

**}**

**bool** cross\_seg**(const** Line **&**seg**)** **{**

*// Line intersect segment*

Point dir**=**p2**-**p1**;**

**return** dir**.**cross**(**seg**.**p1**-**p1**)\***dir**.**cross**(**seg**.**p2**-**p1**)<=**0**;**

**}**

**int** seg\_intersect**(const** Line **&**s**)** **const{**

*// Two segment intersect*

*// 0 -> no, 1 -> one point, -1 -> infinity*

Point dir**=**p2**-**p1**,** dir2**=**s**.**p2**-**s**.**p1**;**

**double** c1**=**dir**.**cross**(**s**.**p2**-**p1**);**

**double** c2**=**dir**.**cross**(**s**.**p1**-**p1**);**

**double** c3**=**dir2**.**cross**(**p2**-**s**.**p1**);**

**double** c4**=**dir2**.**cross**(**p1**-**s**.**p1**);**

**if** **(**c1**==**0**&&**c2**==**0**)** **{**

**if((**s**.**p2**-**p1**).**dot**(**s**.**p1**-**p1**)>**0**&&(**s**.**p2**-**p2**).**dot**(**s**.**p1**-**p2**)>**0**&&**

**(**p1**-**s**.**p1**).**dot**(**p2**-**s**.**p1**)>**0**&&(**p1**-**s**.**p2**).**dot**(**p2**-**s**.**p2**)>**0**)return** 0**;**

**if(**p1**==**s**.**p1**&&(**p2**-**p1**).**dot**(**s**.**p2**-**p1**)<=**0**)return** 1**;**

**if(**p1**==**s**.**p2**&&(**p2**-**p1**).**dot**(**s**.**p1**-**p1**)<=**0**)return** 1**;**

**if(**p2**==**s**.**p1**&&(**p1**-**p2**).**dot**(**s**.**p2**-**p2**)<=**0**)return** 1**;**

**if(**p2**==**s**.**p2**&&(**p1**-**p2**).**dot**(**s**.**p1**-**p2**)<=**0**)return** 1**;**

**return** **-**1**;**

**}else** **if(**c1**\***c2**<=**0**&&**c3**\***c4**<=**0**)return** 1**;**

**return** 0**;**

**}**

Point line\_intersection**(const** Line **&**l**)** **const{**

*// Intersection of lines*

*// pton(); l.pton();*

**double** deno**=**a**\***l**.**b**-**l**.**a**\***b**;**

**if** **(**deno**!=**0**)** **{**

**return** **{** **(**l**.**c**\***b**-**c**\***l**.**b**)/**deno**,** **(**l**.**a**\***c**-**a**\***l**.**c**)/**deno**};**

**}**

*// Reaches here means no intersection. (parallel)*

**return** **{**1234**,**4321**};**

**}**

Point seg\_intersection**(**Line **&**s**)** **const** **{**

Point dir**=**p2**-**p1**,** dir2**=**s**.**p2**-**s**.**p1**;**

*// pton(); l.pton();*

**double** c1**=**dir**.**cross**(**s**.**p2**-**p1**);**

**double** c2**=**dir**.**cross**(**s**.**p1**-**p1**);**

**double** c3**=**dir2**.**cross**(**p2**-**s**.**p1**);**

**double** c4**=**dir2**.**cross**(**p1**-**s**.**p1**);**

**if** **(**c1**==**0**&&**c2**==**0**)** **{**

**if(**p1**==**s**.**p1**&&(**p2**-**p1**).**dot**(**s**.**p2**-**p1**)<=**0**)return** p1**;**

**if(**p1**==**s**.**p2**&&(**p2**-**p1**).**dot**(**s**.**p1**-**p1**)<=**0**)return** p1**;**

**if(**p2**==**s**.**p1**&&(**p1**-**p2**).**dot**(**s**.**p2**-**p2**)<=**0**)return** p2**;**

**if(**p2**==**s**.**p2**&&(**p1**-**p2**).**dot**(**s**.**p1**-**p2**)<=**0**)return** p2**;**

**}else** **if(**c1**\***c2**<=**0**&&**c3**\***c4**<=**0**)return** line\_intersection**(**s**);**

*// Reaches here means either INF or NOT ANY*

*// Use seg\_intersect to check OuO*

**return** **{**1234**,**4321**};**

**}**

**double** dist**(const** Point **&**p**,** **bool** is\_segment**)** **const** **{**

*// Point to Line/segment*

Point dir**=**p2**-**p1**,**v**=**p**-**p1**;**

**if** **(**is\_segment**)** **{**

**if** **(**dir**.**dot**(**v**)<**0**)** **return** v**.**len**();**

**if** **((**p1**-**p2**).**dot**(**p**-**p2**)<**0**)** **return** **(**p**-**p2**).**len**();**

**}**

**double** d**=**abs**(**dir**.**cross**(**v**))/**dir**.**len**();**

**return** d**;**

**}**

**};**

**template<typename** T**>**

**struct** polygon**{**

polygon**(){}**

vector**<**point**<**T**>** **>** p**;***//counterclockwise*

T area**()const{**

T ans**=**0**;**

**for(int** i**=**p**.**size**()-**1**,**j**=**0**;**j**<(int)**p**.**size**();**i**=**j**++)**

ans**+=**p**[**i**].**cross**(**p**[**j**]);**

**return** ans**/**2**;**

**}**

point**<**T**>** center\_of\_mass**()const{**

T cx**=**0**,**cy**=**0**,**w**=**0**;**

**for(int** i**=**p**.**size**()-**1**,**j**=**0**;**j**<(int)**p**.**size**();**i**=**j**++){**

T a**=**p**[**i**].**cross**(**p**[**j**]);**

cx**+=(**p**[**i**].**x**+**p**[**j**].**x**)\***a**;**

cy**+=(**p**[**i**].**y**+**p**[**j**].**y**)\***a**;**

w**+=**a**;**

**}**

**return** point**<**T**>(**cx**/**3**/**w**,**cy**/**3**/**w**);**

**}**

**char** ahas**(const** point**<**T**>&** t**)const{***//return 1 if in simple polygon, -1 if on, 0 if no.*

**bool** c**=**0**;**

**for(int** i**=**0**,**j**=**p**.**size**()-**1**;**i**<**p**.**size**();**j**=**i**++)**

**if(**line**<**T**>(**p**[**i**],**p**[**j**]).**point\_on\_segment**(**t**))return** **-**1**;**

**else** **if((**p**[**i**].**y**>**t**.**y**)!=(**p**[**j**].**y**>**t**.**y**)&&**

t**.**x**<(**p**[**j**].**x**-**p**[**i**].**x**)\*(**t**.**y**-**p**[**i**].**y**)/(**p**[**j**].**y**-**p**[**i**].**y**)+**p**[**i**].**x**)**

c**=!**c**;**

**return** c**;**

**}**

**char** point\_in\_convex**(const** point**<**T**>&**x**)const{**

**int** l**=**1**,**r**=(int)**p**.**size**()-**2**;**

**while(**l**<=**r**){***//return 1 if in convex polygon, -1 if on, 0 if no.*

**int** mid**=(**l**+**r**)/**2**;**

T a1**=(**p**[**mid**]-**p**[**0**]).**cross**(**x**-**p**[**0**]);**

T a2**=(**p**[**mid**+**1**]-**p**[**0**]).**cross**(**x**-**p**[**0**]);**

**if(**a1**>=**0**&&**a2**<=**0**){**

T res**=(**p**[**mid**+**1**]-**p**[**mid**]).**cross**(**x**-**p**[**mid**]);**

**return** res**>**0**?**1**:(**res**>=**0**?-**1**:**0**);**

**}else** **if(**a1**<**0**)**r**=**mid**-**1**;**

**else** l**=**mid**+**1**;**

**}**

**return** 0**;**

**}**

vector**<**T**>** getA**()const{***//angle of each edge to x-axis*

vector**<**T**>**res**;***//must be increasing*

**for(**size\_t i**=**0**;**i**<**p**.**size**();++**i**)**

res**.**push\_back**((**p**[(**i**+**1**)%**p**.**size**()]-**p**[**i**]).**getA**());**

**return** res**;**

**}**

**bool** line\_intersect**(const** vector**<**T**>&**A**,const** line**<**T**>** **&**l**)const{***//O(logN)*

**int** f1**=**upper\_bound**(**A**.**begin**(),**A**.**end**(),(**l**.**p1**-**l**.**p2**).**getA**())-**A**.**begin**();**

**int** f2**=**upper\_bound**(**A**.**begin**(),**A**.**end**(),(**l**.**p2**-**l**.**p1**).**getA**())-**A**.**begin**();**

**return** l**.**cross\_seg**(**line**<**T**>(**p**[**f1**],**p**[**f2**]));**

**}**

polygon cut**(const** line**<**T**>** **&**l**)const{**

polygon ans**;***//convex polygon cut by a line, left side of the line is remained.*

**for(int** n**=**p**.**size**(),**i**=**n**-**1**,**j**=**0**;**j**<**n**;**i**=**j**++){**

**if(**l**.**ori**(**p**[**i**])>=**0**){**

ans**.**p**.**push\_back**(**p**[**i**]);**

**if(**l**.**ori**(**p**[**j**])<**0**)**

ans**.**p**.**push\_back**(**l**.**line\_intersection**(**line**<**T**>(**p**[**i**],**p**[**j**])));**

**}else** **if(**l**.**ori**(**p**[**j**])>**0**)**

ans**.**p**.**push\_back**(**l**.**line\_intersection**(**line**<**T**>(**p**[**i**],**p**[**j**])));**

**}**

**return** ans**;**

**}**

**static** **bool** graham\_cmp**(const** point**<**T**>&** a**,const** point**<**T**>&** b**){***//CMP for finding hull*

**return** **(**a**.**x**<**b**.**x**)||(**a**.**x**==**b**.**x**&&**a**.**y**<**b**.**y**);**

**}**

**void** graham**(**vector**<**point**<**T**>** **>** **&**s**){***//convex hull*

sort**(**s**.**begin**(),**s**.**end**(),**graham\_cmp**);**

p**.**resize**(**s**.**size**()+**1**);**

**int** m**=**0**;**

**for(**size\_t i**=**0**;**i**<**s**.**size**();++**i**){**

**while(**m**>=**2**&&(**p**[**m**-**1**]-**p**[**m**-**2**]).**cross**(**s**[**i**]-**p**[**m**-**2**])<=**0**)--**m**;**

p**[**m**++]=**s**[**i**];**

**}**

**for(int** i**=**s**.**size**()-**2**,**t**=**m**+**1**;**i**>=**0**;--**i**){**

**while(**m**>=**t**&&(**p**[**m**-**1**]-**p**[**m**-**2**]).**cross**(**s**[**i**]-**p**[**m**-**2**])<=**0**)--**m**;**

p**[**m**++]=**s**[**i**];**

**}**

**if(**s**.**size**()>**1**)--**m**;**

p**.**resize**(**m**);**

**}**

T diameter**(){**

**int** n**=**p**.**size**(),**t**=**1**;**

T ans**=**0**;**p**.**push\_back**(**p**[**0**]);**

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

point**<**T**>** now**=**p**[**i**+**1**]-**p**[**i**];**

**while(**now**.**cross**(**p**[**t**+**1**]-**p**[**i**])>**now**.**cross**(**p**[**t**]-**p**[**i**]))**t**=(**t**+**1**)%**n**;**

ans**=**max**(**ans**,(**p**[**i**]-**p**[**t**]).**abs2**());**

**}**

**return** p**.**pop\_back**(),**ans**;**

**}**

T min\_cover\_rectangle**(){***// find convex hull before call this*

**int** n**=**p**.**size**(),**t**=**1**,**r**=**1**,**l**;**

**if(**n**<**3**)return** 0**;**

T ans**=**1e99**;**p**.**push\_back**(**p**[**0**]);**

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

point**<**T**>** now**=**p**[**i**+**1**]-**p**[**i**];**

**while(**now**.**cross**(**p**[**t**+**1**]-**p**[**i**])>**now**.**cross**(**p**[**t**]-**p**[**i**]))**t**=(**t**+**1**)%**n**;**

**while(**now**.**dot**(**p**[**r**+**1**]-**p**[**i**])>**now**.**dot**(**p**[**r**]-**p**[**i**]))**r**=(**r**+**1**)%**n**;**

**if(!**i**)**l**=**r**;**

**while(**now**.**dot**(**p**[**l**+**1**]-**p**[**i**])<=**now**.**dot**(**p**[**l**]-**p**[**i**]))**l**=(**l**+**1**)%**n**;**

T d**=**now**.**abs2**();**

T tmp**=**now**.**cross**(**p**[**t**]-**p**[**i**])\*(**now**.**dot**(**p**[**r**]-**p**[**i**])-**now**.**dot**(**p**[**l**]-**p**[**i**]))/**d**;**

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

**}**

**return** p**.**pop\_back**(),**ans**;**

**}**

T dis2**(**polygon **&**pl**){***//square of distance of two convex polygon*

vector**<**point**<**T**>** **>** **&**P**=**p**,&**Q**=**pl**.**p**;**

**int** n**=**P**.**size**(),**m**=**Q**.**size**(),**l**=**0**,**r**=**0**;**

**for(int** i**=**0**;**i**<**n**;++**i**)if(**P**[**i**].**y**<**P**[**l**].**y**)**l**=**i**;**

**for(int** i**=**0**;**i**<**m**;++**i**)if(**Q**[**i**].**y**<**Q**[**r**].**y**)**r**=**i**;**

P**.**push\_back**(**P**[**0**]),**Q**.**push\_back**(**Q**[**0**]);**

T ans**=**1e99**;**

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

**while((**P**[**l**]-**P**[**l**+**1**]).**cross**(**Q**[**r**+**1**]-**Q**[**r**])<**0**)**r**=(**r**+**1**)%**m**;**

ans**=**min**(**ans**,**line**<**T**>(**P**[**l**],**P**[**l**+**1**]).**seg\_dis2**(**line**<**T**>(**Q**[**r**],**Q**[**r**+**1**])));**

l**=(**l**+**1**)%**n**;**

**}**

**return** P**.**pop\_back**(),**Q**.**pop\_back**(),**ans**;**

**}**

**static** **char** sign**(const** point**<**T**>&**t**){**

**return** **(**t**.**y**==**0**?**t**.**x**:**t**.**y**)<**0**;**

**}**

**static** **bool** angle\_cmp**(const** line**<**T**>&** A**,const** line**<**T**>&** B**){**

point**<**T**>** a**=**A**.**p2**-**A**.**p1**,**b**=**B**.**p2**-**B**.**p1**;**

**return** sign**(**a**)<**sign**(**b**)||(**sign**(**a**)==**sign**(**b**)&&**a**.**cross**(**b**)>**0**);**

**}**

**int** halfplane\_intersection**(**vector**<**line**<**T**>** **>** **&**s**){**

sort**(**s**.**begin**(),**s**.**end**(),**angle\_cmp**);***//half plane is left side of the line*

**int** L**,**R**,**n**=**s**.**size**();**

vector**<**point**<**T**>** **>** px**(**n**);**

vector**<**line**<**T**>** **>** q**(**n**);**

q**[**L**=**R**=**0**]=**s**[**0**];**

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

**while(**L**<**R**&&**s**[**i**].**ori**(**px**[**R**-**1**])<=**0**)--**R**;**

**while(**L**<**R**&&**s**[**i**].**ori**(**px**[**L**])<=**0**)++**L**;**

q**[++**R**]=**s**[**i**];**

**if(**q**[**R**].**parallel**(**q**[**R**-**1**])){**

**--**R**;**

**if(**q**[**R**].**ori**(**s**[**i**].**p1**)>**0**)**q**[**R**]=**s**[**i**];**

**}**

**if(**L**<**R**)**px**[**R**-**1**]=**q**[**R**-**1**].**line\_intersection**(**q**[**R**]);**

**}**

**while(**L**<**R**&&**q**[**L**].**ori**(**px**[**R**-**1**])<=**0**)--**R**;**

p**.**clear**();**

**if(**R**-**L**<=**1**)return** 0**;**

px**[**R**]=**q**[**R**].**line\_intersection**(**q**[**L**]);**

**for(int** i**=**L**;**i**<=**R**;++**i**)**p**.**push\_back**(**px**[**i**]);**

**return** R**-**L**+**1**;**

**}**

**};**

**template<typename** T**>**

**struct** triangle**{**

point**<**T**>** a**,**b**,**c**;**

triangle**(){}**

triangle**(const** point**<**T**>** **&**a**,const** point**<**T**>** **&**b**,const** point**<**T**>** **&**c**):**a**(**a**),**b**(**b**),**c**(**c**){}**

T area**()const{**

T t**=(**b**-**a**).**cross**(**c**-**a**)/**2**;**

**return** t**>**0**?**t**:-**t**;**

**}**

point**<**T**>** barycenter**()const{***//center of mass*

**return** **(**a**+**b**+**c**)/**3**;**

**}**

point**<**T**>** circumcenter**()const{***//outer center*

**static** line**<**T**>** u**,**v**;**

u**.**p1**=(**a**+**b**)/**2**;**

u**.**p2**=**point**<**T**>(**u**.**p1**.**x**-**a**.**y**+**b**.**y**,**u**.**p1**.**y**+**a**.**x**-**b**.**x**);**

v**.**p1**=(**a**+**c**)/**2**;**

v**.**p2**=**point**<**T**>(**v**.**p1**.**x**-**a**.**y**+**c**.**y**,**v**.**p1**.**y**+**a**.**x**-**c**.**x**);**

**return** u**.**line\_intersection**(**v**);**

**}**

point**<**T**>** incenter**()const{***//inner center*

T A**=**sqrt**((**b**-**c**).**abs2**()),**B**=**sqrt**((**a**-**c**).**abs2**()),**C**=**sqrt**((**a**-**b**).**abs2**());**

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

**}**

point**<**T**>** perpencenter**()const{***// perpendicular(?) center*

**return** barycenter**()\***3**-**circumcenter**()\***2**;**

**}**

**};**

* **Smallest circle**

**using** PT**=**point**<**T**>;** **using** CPT**=const** PT**;**

PT circumcenter**(**CPT **&**a**,**CPT **&**b**,**CPT **&**c**){**

PT u**=**b**-**a**,** v**=**c**-**a**;**

T c1**=**u**.**abs2**()/**2**,**c2**=**v**.**abs2**()/**2**;**

T d**=**u**.**cross**(**v**);**

**return** PT**(**a**.**x**+(**v**.**y**\***c1**-**u**.**y**\***c2**)/**d**,**a**.**y**+(**u**.**x**\***c2**-**v**.**x**\***c1**)/**d**);**

**}**

**void** solve**(**PT p**[],int** n**,**PT **&**c**,**T **&**r2**){**

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

c**=**p**[**0**];** r2**=**0**;** *// c,r2 = ??,????*

**for(int** i**=**1**;**i**<**n**;**i**++)if((**p**[**i**]-**c**).**abs2**()>**r2**){**

c**=**p**[**i**];** r2**=**0**;**

**for(int** j**=**0**;**j**<**i**;**j**++)if((**p**[**j**]-**c**).**abs2**()>**r2**){**

c**.**x**=(**p**[**i**].**x**+**p**[**j**].**x**)/**2**;**

c**.**y**=(**p**[**i**].**y**+**p**[**j**].**y**)/**2**;**

r2**=(**p**[**j**]-**c**).**abs2**();**

**for(int** k**=**0**;**k**<**j**;**k**++)if((**p**[**k**]-**c**).**abs2**()>**r2**){**

c**=**circumcenter**(**p**[**i**],**p**[**j**],**p**[**k**]);**

r2**=(**p**[**i**]-**c**).**abs2**();**

**}**

**}**

**}**

**}**

* **Matrix**

**template<typename** T**>**

**struct** Matrix**{**

**using** rt **=** std**::**vector**<**T**>;**

**using** mt **=** std**::**vector**<**rt**>;**

**using** matrix **=** Matrix**<**T**>;**

**int** r**,**c**;**

mt m**;**

Matrix**(int** r**,int** c**):**r**(**r**),**c**(**c**),**m**(**r**,**rt**(**c**)){}**

rt**&** **operator[](int** i**){return** m**[**i**];}**

matrix **operator+(const** matrix **&**a**){**

matrix rev**(**r**,**c**);**

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

**for(int** j**=**0**;**j**<**c**;++**j**)**

rev**[**i**][**j**]=**m**[**i**][**j**]+**a**.**m**[**i**][**j**];**

**return** rev**;**

**}**

matrix **operator-(const** matrix **&**a**){**

matrix rev**(**r**,**c**);**

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

**for(int** j**=**0**;**j**<**c**;++**j**)**

rev**[**i**][**j**]=**m**[**i**][**j**]-**a**.**m**[**i**][**j**];**

**return** rev**;**

**}**

matrix **operator\*(const** matrix **&**a**){**

matrix rev**(**r**,**a**.**c**);**

matrix tmp**(**a**.**c**,**a**.**r**);**

**for(int** i**=**0**;**i**<**a**.**r**;++**i**)**

**for(int** j**=**0**;**j**<**a**.**c**;++**j**)**

tmp**[**j**][**i**]=**a**.**m**[**i**][**j**];**

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

**for(int** j**=**0**;**j**<**a**.**c**;++**j**)**

**for(int** k**=**0**;**k**<**c**;++**k**)**

rev**.**m**[**i**][**j**]+=**m**[**i**][**k**]\***tmp**[**j**][**k**];**

**return** rev**;**

**}**

**bool** inverse**(){**

Matrix t**(**r**,**r**+**c**);**

**for(int** y**=**0**;**y**<**r**;**y**++){**

t**.**m**[**y**][**c**+**y**]** **=** 1**;**

**for(int** x**=**0**;**x**<**c**;++**x**)**

t**.**m**[**y**][**x**]=**m**[**y**][**x**];**

**}**

**if(** **!**t**.**gas**()** **)**

**return** **false;**

**for(int** y**=**0**;**y**<**r**;**y**++)**

**for(int** x**=**0**;**x**<**c**;++**x**)**

m**[**y**][**x**]=**t**.**m**[**y**][**c**+**x**]/**t**.**m**[**y**][**y**];**

**return** **true;**

**}**

T gas**(){**

vector**<**T**>** lazy**(**r**,**1**);**

**bool** sign**=false;**

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

**if(** m**[**i**][**i**]==**0 **){**

**int** j**=**i**+**1**;**

**while(**j**<**r**&&!**m**[**j**][**i**])**j**++;**

**if(**j**==**r**)continue;**

m**[**i**].**swap**(**m**[**j**]);**

sign**=!**sign**;**

**}**

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

**if(**i**==**j**)continue;**

lazy**[**j**]=**lazy**[**j**]\***m**[**i**][**i**];**

T mx**=**m**[**j**][**i**];**

**for(int** k**=**0**;**k**<**c**;++**k**)**

m**[**j**][**k**]=**m**[**j**][**k**]\***m**[**i**][**i**]-**m**[**i**][**k**]\***mx**;**

**}**

**}**

T det**=**sign**?-**1**:**1**;**

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

det **=** det**\***m**[**i**][**i**];**

det **=** det**/**lazy**[**i**];**

**for(auto** **&**j**:**m**[**i**])**j**/=**lazy**[**i**];**

**}**

**return** det**;**

**}**

**};**

* **Gaussian elimination**

**typedef** **double** Matrix**[**maxn**][**maxn**];**

**void** guauss\_elimination**(**Matrix A**,** **int** n**){**

**int** r**;**

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

r **=** i**;**

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

**if(**fabs**(**A**[**j**][**i**])** **>** fabs**(**A**[**r**][**i**]))** r **=** j**;**

**if(**r**!=**i**)** **for(int** j**=**0**;** j**<=**n**;** j**++)** swap**(**A**[**r**][**i**],** A**[**i**][**j**]);**

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

**double** f **=** A**[**k**][**i**]/**A**[**i**][**i**];**

**for(int** j**=**i**;** j**<=**n**;** j**++)** A**[**k**][**j**]** **-=** f**\***A**[**i**][**j**];**

**}**

**}**

**for(int** i**=**n**-**1**;** i**>=**0**;** i**--){**

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

A**[**i**][**n**]** **-=** A**[**j**][**n**]** **\*** A**[**i**][**j**];**

A**[**i**][**n**]** **/=** A**[**i**][**i**];**

**}**

**}**

* **LL multiplication**

**long** **long** mul**(long** **long** a**,** **long** **long** b**)** **{**

**long** **long** ans **=** 0**,** step **=** a **%** MOD**;**

**while** **(**b**)** **{**

**if** **(**b **&** 1L**)** ans **+=** step**;**

**if** **(**ans **>=** MOD**)** ans **%=** MOD**;**

step **<<=** 1L**;**

**if** **(**step **>=** MOD**)** step **%=** MOD**;**

b **>>=** 1L**;**

**}**

**return** ans **%** MOD**;**

**}**

* **Formulas**

若多項式f(x)有有理根P/Q(PQ互質), 則P必為常數項a0之因數, Q必為領導係數an之因數

滿足ceil(n/i)=k之最大i:

INF, if k=1

n/(k-1)-1, else if k-1 整除 n

n/(k-1), else

滿足floor(n/i)=k之最大i: floor(n/k)

尤拉函數: phi(n)=n乘上所有(1-1/p)，對n之所有質因數p

費馬小定理: a \* a^(p-2) = 1 (mod p), a,p互質

次方同餘定理: a^k mod p = (a mod p)^(k mod p-1) p是質數

Modulo inverse: inv[i] ≡ - floor(p / i) × inv[p % i] (mod p)

中國剩餘定理: x=Ai(mod mi), mi互質, Mi=所有m的乘積/mi, Ti=Mi^-1(mod mi), 則x=sigma(Mi\*Ti\*Ai)(mod M)

枚舉擴展歐幾里得之解:

若x0,y0為a\*x+b\*y = k之一組解，則

x=x0+t\*b/gcd(a,b), y=y0+t\*a/gcd(a,b)亦為解，t為整數

Sigma{i : gcd(i,n) = 1 and i in [1, n]} = n\*phi(n)/2 for n > 1

尤拉公式: V-E+F = C+1

最大獨立集: 點的集合，其內點不相鄰

最小點覆蓋: 點的集合，所有邊都被覆蓋

最大匹配: 邊的集合，其內邊不共用點

最小邊覆蓋: 邊的集合，所有點都被覆蓋

最大獨立集+最小點覆蓋=V(數值)

最大匹配+最小邊覆蓋=V(數值)

最大匹配=最大流(二分圖)

最大匹配=最小點覆蓋(二分圖)

最小點覆蓋+最小邊覆蓋=V(數值，二分圖)

對於可以換樹根的樹 以點1為準建LCA:

若r為當前根, x, y為兩點，則新樹中x,y的lca為:

lowest vertex({ LCA(x,y), LCA(r,x), LCA(r,y) }) (深度最深)

一矩陣A所有eigen value之合=對角線合

一矩陣A所有eigen value之積=det(A)

三角形ABC, 對邊長abc:

area=sqrt(s(s-a)(s-b)(s-b)), s=周長/2

a/sinA = b/sinB = c/sinC = 2R, R為外接圓半徑

內接圓半徑=2\*area/(a+b+c)

外接圓半徑=abc/4\*area

枚舉submask: for (int s=m; s; s=(s-1)&m) // Take care of ZERO by yourself

某些質數:

54018521, 370248451, 6643838879, 119218851371, 5600748293801

39916801, 479001599, 87178291199, 8589935681, 433494437, 2971215073

* **Vimrc**

se ai nu ru cul mouse=a

se cin et ts=2 sw=2 sts=2

colo desert

se gfn=Monospace\ 14