Mục lục

[1. FOR BIT 2](#_Toc467511403)

[2. MONOTONE CHAIN + TERNARY SEARCH 2](#_Toc467511404)

[3. EDMONDS KARP 2](#_Toc467511405)

[4. DINIC 3](#_Toc467511406)

[5. PST 4](#_Toc467511407)

[6. IMPLICIT ST 5](#_Toc467511408)

[7. 2-SAT 5](#_Toc467511409)

[8. MO 6](#_Toc467511410)

[9. LCA 6](#_Toc467511411)

[10. HLD 6](#_Toc467511412)

[11. SLIDING WINDOW 7](#_Toc467511413)

[12. MATRIX POWER 7](#_Toc467511414)

[13. TRIE 7](#_Toc467511415)

[14. FFT 8](#_Toc467511416)

[15. EXTENDED EUCLID 9](#_Toc467511417)

[16. FAT 9](#_Toc467511418)

[17. GEOMETRY 9](#_Toc467511419)

[18. EDGE ARRAY 10](#_Toc467511420)

[19. HASH 10](#_Toc467511421)

[20. KMP 10](#_Toc467511422)

[21. Ukkonen O(n) 10](#_Toc467511423)

[22. SuffixArray and LCP O(n.logn) 11](#_Toc467511424)

[23. Manacher 12](#_Toc467511425)

[24. Aho-Corasick 12](#_Toc467511426)

[25. Palindromic Tree 13](#_Toc467511427)

[26. Longest Common Prefix 13](#_Toc467511428)

[27. Lyndon 14](#_Toc467511429)

[28. K-th order statistic 14](#_Toc467511430)

[29. Z-Function 14](#_Toc467511431)

[30. Tandem Repeats 15](#_Toc467511432)

1. FOR BIT

**for** **(**int split **=** mask **&** **(**mask**-**1**);** split**;** split **=** **(**split**-**1**)** **&** mask**)**

1. MONOTONE CHAIN + TERNARY SEARCH

struct point **{** ll x**,** y**;** **};**

struct polygon **{** ll vertexNumber**;** point vertex**[**MAXPOLY**];}** silk**;**

point frosh**[**MAXPOLY**];**

point upper**[**MAXPOLY**],** lower**[**MAXPOLY**];**

ll calDistance**(**point first**,** point second**)** **{**

**return** **(**first**.**x **-** second**.**x**)\*(**first**.**x **-** second**.**x**)** **+** **(**first**.**y **-** second**.**y**)\*(**first**.**y **-** second**.**y**);** **}**

ll crossProduct**(**point first**,** point second**,** point third**)** **{**

**return** **(**second**.**x **-** first**.**x**)** **\*** **(**third**.**y **-** first**.**y**)** **–**

**(**second**.**y **-** first**.**y**)** **\*** **(**third**.**x **-** first**.**x**);** **}**

bool leftLower**(**point first**,** point second**)** **{**

**if** **(**first**.**x **==** second**.**x**)** **return** first**.**y **<** second**.**y**;**

**return** first**.**x **<** second**.**x**;**

**}**

void convexHull**(**ll vertexNumber**)** **{**

**if** **(**vertexNumber **<=** 3**)** **{**

**for** **(**ll i **=** 0**;** i **<** vertexNumber**;** i**++)**

silk**.**vertex**[**i**]** **=** frosh**[**i**];**

silk**.**vertexNumber **=** vertexNumber**;**

**return;**

**}**

sort**(**frosh**,** frosh **+** vertexNumber**,** leftLower**);**

ll top **=** 2**;**

upper**[**0**]** **=** frosh**[**0**];**

upper**[**1**]** **=** frosh**[**1**];**

**for** **(**ll i **=** 2**;** i **<** vertexNumber**;** i**++)** **{**

upper**[**top**]** **=** frosh**[**i**];**

**while** **(**top **>=** 2 **&&** crossProduct**(**upper**[**top **-** 2**],** upper**[**top **-** 1**],** upper**[**top**])** **>=** 0**)** **{**

upper**[**top **-** 1**]** **=** upper**[**top**];**

top**--;**

**}**

top**++;**

**}**

silk**.**vertexNumber **=** 0**;**

**for** **(**ll i **=** 0**;** i **<** top**;** i**++)**

silk**.**vertex**[**silk**.**vertexNumber**++]** **=** upper**[**i**];**

lower**[**0**]** **=** frosh**[**vertexNumber **-** 1**];**

lower**[**1**]** **=** frosh**[**vertexNumber **-** 2**];**

top **=** 2**;**

**for** **(**ll i **=** vertexNumber **-** 3**;** i **>=** 0**;** i**--)** **{**

lower**[**top**]** **=** frosh**[**i**];**

**while** **(**top **>=** 2 **&&** crossProduct**(**lower**[**top **-** 2**],**

lower**[**top **-** 1**],** lower**[**top**])** **>=** 0**)** **{**

lower**[**top **-** 1**]** **=** lower**[**top**];**

top**--;**

**}**

top**++;**

**}**

**for** **(**ll i **=** 1**;** i **<** top **-** 1**;** i**++)**

silk**.**vertex**[**silk**.**vertexNumber**++]** **=** lower**[**i**];**

**}**

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

ll froshNumber**;**

FOR**(**i**,**0**,**froshNumber**-**1**)** cin **>>** frosh**[**i**].**x **>>** frosh**[**i**].**y**;**

convexHull**(**froshNumber**);**

FOR**(**i**,**silk**.**vertexNumber**,**2**\***silk**.**vertexNumber**-**1**)**

silk**.**vertex**[**i**]** **=** silk**.**vertex**[**i**-**silk**.**vertexNumber**];**

ll res **=** 0**;**

FOR**(**i**,**0**,**silk**.**vertexNumber**-**1**)** **{**

ll LL **=** i**+**1**;**

ll RR **=** i**+**silk**.**vertexNumber**-**1**;**

ll ML**=(**LL**+**LL**+**RR**)/**3**;**

ll MR**=(**LL**+**RR**+**RR**)/**3**;**

**while** **((**LL**!=**ML**)** **&&** **(**ML**!=**MR**)** **&&** **(**MR**!=**RR**))** **{**

**if** **(**calDistance**(**silk**.**vertex**[**i**],** silk**.**vertex**[**ML**])** **<** calDistance**(**silk**.**vertex**[**i**],** silk**.**vertex**[**MR**]))** LL**=**ML**;**

**else** RR**=**MR**;**

ML**=(**LL**+**LL**+**RR**)/**3**;**

MR**=(**LL**+**RR**+**RR**)/**3**;**

**}**

FOR**(**j**,**LL**,**RR**)** res **=** max**(**res**,**

calDistance**(**silk**.**vertex**[**i**],** silk**.**vertex**[**j**]));}}**

1. EDMONDS KARP

int t**,** n**,** m**,** adj**[**410**][**410**],** trace**[**410**],** d**[**410**];**

vector**<**int**>** a**[**410**];**

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

int main**()**

**{**

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

FOR**(**i**,**0**,**n**+**m**+**1**)** **{**

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

FOR**(**j**,**0**,**n**+**m**+**1**)** adj**[**i**][**j**]** **=** 0**;**

**}**

FOR**(**i**,**1**,**n**)** **{** a**[**0**].**pb**(**i**);** a**[**i**].**pb**(**0**);** adj**[**0**][**i**]++;** **}**

FOR**(**i**,**n**+**1**,**n**+**m**)** **{**a**[**i**].**pb**(**n**+**m**+**1**);**a **[**n**+**m**+**1**].**pb**(**i**);** adj**[**i**][**n**+**m**+**1**]++;** **}**

int flow **=** 0**;**

**while** **(**1**)** **{**

**while** **(!**q**.**empty**())** q**.**pop**();**

FOR**(**i**,**0**,**n**+**m**+**1**)** **{**

d**[**i**]** **=** 1000000000**;**

trace**[**i**]** **=** **-**1**;**

**}**

q**.**push**(**0**);**

d**[**0**]** **=** 0**;**

bool co **=** **false;**

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

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

**if** **(**u **==** n**+**m**+**1**)** **{**

co **=** **true;**

**break;**

**}**

**for** **(**vector**<**int**>::**iterator it **=** a**[**u**].**begin**();** it**!=**a**[**u**].**end**();** it**++)** **{**

int v **=** **\***it**;**

**if** **(**adj**[**u**][**v**]** **>** 0 **&&** d**[**v**]** **==** 1000000000**)** **{**

d**[**v**]** **=** d**[**u**]** **+** 1**;**

q**.**push**(**v**);**

trace**[**v**]** **=** u**;**

**}**

**}**

**}**

**if** **(!**co**)** **break;**

int p **=** n**+**m**+**1**;**

int fl **=** 1000000000**;**

**while** **(**p **!=** 0**)** **{**

fl **=** min**(**fl**,** adj**[**trace**[**p**]][**p**]);**

p **=** trace**[**p**];**

**}**

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

p **=** n**+**m**+**1**;**

**while** **(**p **!=** 0**)** **{**

adj**[**trace**[**p**]][**p**]** **-=** fl**;**

adj**[**p**][**trace**[**p**]]** **+=** fl**;**

p **=** trace**[**p**];**

**}**

flow **+=** fl**;**

**}**

**}**

1. DINIC

bool bfs**(**int S**,** int T**)**

**{**

CLR**(**d**);**

queue**<**int**>** qu**;**

qu**.**push**(**S**);**

d**[**S**]** **=** 1**;**

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

int u **=** qu**.**front**();** qu**.**pop**();**

**if** **(**u **==** T**)** **return** **true;**

**for** **(**vector**<**int**>::**iterator it **=** a**[**u**].**begin**();** it**!=**a**[**u**].**end**();** it**++)** **{**

int v **=** **\***it**;**

**if** **(!**d**[**v**]** **&&** 0 **<** f**[**u**][**v**])** **{**

qu**.**push**(**v**);**

d**[**v**]** **=** d**[**u**]** **+** 1**;**

**}**

**}**

**}**

**return** **false;**

**}**

int visit**(**int u**,** int Min**)** **{**

**if** **(**u **==** T**)** **return** Min**;**

**if** **(**dfs**[**u**]** **!=** t**)** dfs**[**u**]** **=** t**;** **else** **return** 0**;**

**for** **(**vector**<**int**>::**iterator it **=** a**[**u**].**begin**();** it**!=**a**[**u**].**end**();** it**++)** **{**

int v **=** **\***it**;**

**if** **(**0 **<** f**[**u**][**v**])**

**if** **(**d**[**v**]** **==** d**[**u**]** **+** 1 **&&** dfs**[**v**]** **!=** t**)** **{**

int k **=** min**(**Min**,** f**[**u**][**v**]);**

int x **=** visit**(**v**,** k**);**

**if** **(**x**)** **{**

f**[**u**][**v**]** **-=** x**;**

f**[**v**][**u**]** **+=** x**;**

**return** x**;**

**}**

**}**

**}**

**return** 0**;**

**}**

int main**()**

**{**

cin **>>** n **>>** m **>>** S **>>** T**;**

**while** **(**m**--)** **{**

int u**,** v**,** C**;**

cin **>>** u **>>** v **>>** C**;**

a**[**u**].**pb**(**v**);**

a**[**v**].**pb**(**u**);**

c**[**u**][**v**]** **+=** C**;**

**}**

int flow **=** 0**;**

**while** **(**bfs**(**S**,**T**))** **{**

**while** **(**1**)** **{**

t**++;**

int x **=** visit**(**S**,** 1000000000**);**

**if** **(!**x**)** **break;**

flow **+=** x**;**

**}**

**}**

**}**

1. PST

int n**,** m**,** a**[**100010**],** b**[**100010**],** N**,** nNode**,** ver**[**100010**];**

map**<**int**,** int**>** ma**;**

struct node **{**

int lef**,** rig**,** cnt**;**

**}** t**[**2100010**];**

void build**(**int k**,** int l**,** int r**)**

**{**

**if** **(**l **<=** r**)** nNode **=** max**(**k**,** nNode**);**

**if** **(**l **>=** r**)** **return;**

t**[**k**].**lef **=** k**\***2**;**

t**[**k**].**rig **=** k**\***2**+**1**;**

int m **=** **(**l**+**r**)** **>>** 1**;**

build**(**t**[**k**].**lef**,**l**,**m**);**

build**(**t**[**k**].**rig**,**m**+**1**,**r**);**

**}**

int update**(**int oldId**,** int l**,** int r**,** int u**)**

**{**

**if** **(**l **==** r**)** **{**

**++**nNode**;**

t**[**nNode**].**lef **=** t**[**nNode**].**rig **=** 0**;**

t**[**nNode**].**cnt **=** t**[**oldId**].**cnt**+**1**;**

**return** nNode**;**

**}**

int cur **=** **++**nNode**;**

int m **=** **(**l**+**r**)** **>>** 1**;**

**if** **(**u **<=** m**)** **{**

t**[**cur**].**lef **=** update**(**t**[**oldId**].**lef**,**l**,**m**,**u**);**

t**[**cur**].**rig **=** t**[**oldId**].**rig**;**

**}**

**else** **{**

t**[**cur**].**rig **=** update**(**t**[**oldId**].**rig**,**m**+**1**,**r**,**u**);**

t**[**cur**].**lef **=** t**[**oldId**].**lef**;**

**}**

t**[**cur**].**cnt **=** t**[**t**[**cur**].**rig**].**cnt **+** t**[**t**[**cur**].**lef**].**cnt**;**

**return** cur**;**

**}**

int get**(**int oldId**,** int newId**,** int l**,** int r**,** int k**)**

**{**

**if** **(**l **==** r**)** **return** l**;**

int m **=** **(**l**+**r**)** **>>** 1**;**

int h **=** t**[**t**[**newId**].**lef**].**cnt **-** t**[**t**[**oldId**].**lef**].**cnt**;**

**if** **(**k **<=** h**)** **return** get**(**t**[**oldId**].**lef**,**t**[**newId**].**lef**,**l**,**m**,**k**);**

**return** get**(**t**[**oldId**].**rig**,**t**[**newId**].**rig**,**m**+**1**,**r**,**k**-**h**);**

**}**

int main**()**

**{**

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

FOR**(**i**,**1**,**n**)** **{**

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

ma**[**a**[**i**]];**

**}**

**for** **(**map**<**int**,** int**>::**iterator it **=** ma**.**begin**();** it**!=**ma**.**end**();** it**++)** **{**

ma**[**it**->**X**]** **=** **++**N**;**

b**[**N**]** **=** it**->**X**;**

**}**

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

ver**[**0**]** **=** 1**;**

FOR**(**i**,**1**,**n**)** ver**[**i**]** **=** update**(**ver**[**i**-**1**],**1**,**N**,**ma**[**a**[**i**]]);**

**while** **(**m**--)** **{**

int u**,** v**,** k**;**

scanf**(**"%d %d %d"**,** **&**u**,** **&**v**,** **&**k**);**

printf**(**"%d\n"**,** b**[**get**(**ver**[**u**-**1**],** ver**[**v**],** 1**,** N**,** k**)]);**

**}**

**}**

1. IMPLICIT ST

struct seg **{**

ll b**,** l**,** r**;**

bool lz**;**

seg **\***L**,** **\***R**;**

seg**(){**b **=** l **=** r **=** lz **=** 0**;** L **=** R **=** **NULL;}**

seg**(**ll x**,** ll y**){**l **=** x**,** r **=** y**,** b **=** 0**,** lz **=** 0**;** L **=** R **=** **NULL;}**

inline void le**(){**

**if** **(**l **==** r **||** L **!=** **NULL)** **return;**

ll mid **=** **(**l**+**r**)/**2**;**

L **=** **new** seg**(**l**,** mid**);**

**}**

inline void ri**(){**

**if** **(**l **==** r **||** R **!=** **NULL)** **return;**

ll mid **=** **(**l**+**r**)/**2**;**

R **=** **new** seg**(**mid**+**1**,** r**);**

**}**

inline void color**(**bool h**)** **{**

**if** **(!**h**)** **return;**

lz **=** **!**lz**;**

b **=** r **-** l **+** 1 **-** b**;**

**}**

inline void shift**()** **{**

le**();**ri**();**

L **->** color**(**lz**);**

R **->** color**(**lz**);**

lz **=** **false;**

**}**

inline void upd**(**ll x**,** ll y**){**

**if** **(**y **<** l **||** r **<** x **||** x **>** y**)** **return;**

**if** **(**x **<=** l **&&** r **<=** y**)** **{**

color**(true);**

**return;**

**}**

shift**();**

L **->** upd**(**x**,** y**);**

R **->** upd**(**x**,** y**);**

b **=** L **->** b **+** R **->** b**;**

**}**

inline ll cnt**(**ll x**,** ll y**){**

**if** **(**y **<** l **||** r **<** x **||** x **>** y**)** **return** 0**;**

**if** **(**x **<=** l **&&** r **<=** y**)** **return** b**;**

shift**();**

**return** L **->** cnt**(**x**,** y**)** **+** R **->** cnt**(**x**,** y**);**

**}**

**};**

1. 2-SAT

void orClause**(**int u**,** int v**){** a**[**u**^**1**].**pb**(**v**);** a**[**v**^**1**].**pb**(**u**);** **}**

void xorClause**(**int u**,** int v**)** **{** orClause**(**u**,**v**);** orClause**(**u**^**1**,**v**^**1**);** **}**

void andClause**(**int u**,** int v**)** **{** a**[**u**].**pb**(**v**);** a**[**v**].**pb**(**u**);** **}**

void nXorClause**(**int u**,** int v**)** **{** orClause**(**u**^**1**,**v**);** orClause**(**u**,**v**^**1**);** **}**

void nAndClause**(**int u**,** int v**)** **{** a**[**u**].**pb**(**v**^**1**);** a**[**v**].**pb**(**u**^**1**);** **}**

void mustTrue**(**int u**)** **{** a**[**u**^**1**].**pb**(**u**);** **}**

void scc**(**int u**)** **{**

low**[**u**]** **=** num**[**u**]** **=** **++**dfsNum**;**

s**.**push**(**u**);**

visited**[**u**]** **=** **true;**

**for** **(**vector**<**int**>::**iterator it **=** a**[**u**].**begin**();** it**!=**a**[**u**].**end**();** it**++)** **{**

int v **=** **\***it**;**

**if** **(**num**[**v**]** **==** **-**1**)** scc**(**v**);**

**if** **(**visited**[**v**])** low**[**u**]** **=** min**(**low**[**u**],** low**[**v**]);**

**}**

**if** **(**low**[**u**]** **==** num**[**u**])** **{**

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

scc\_cnt**++;**

**do** **{**

k **=** s**.**top**();**

s**.**pop**();**

w**[**k**]** **=** scc\_cnt**;**

visited**[**k**]** **=** **false;**

**}** **while** **(**k **!=** u**);**

**}**

**}**

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

visited**[**u**]** **=** **true;**

**for** **(**vector**<**int**>::**iterator it **=** a**[**u**].**begin**();** it**!=**a**[**u**].**end**();** it**++)** **{**

int v **=** **\***it**;**

**if** **(!**visited**[**v**])** dfs**(**v**);**

**}**

ts**.**pb**(**u**);**

**}**

FOR**(**i**,**0**,**N**-**1**)** num**[**i**]** **=** **-**1**;**

memset**(**visited**,** **false,** **sizeof(**visited**));**

dfsNum **=** **-**1**;**

FOR**(**i**,**0**,**N**-**1**)** **if** **(**num**[**i**]** **==** **-**1**)** scc**(**i**);**

bool co **=** **true;**

FOR**(**i**,**0**,**N**-**1**)** **if** **(**w**[**i**]** **==** w**[**i**^**1**])** co **=** **false;**

FOR**(**i**,**0**,**N**-**1**)** **if** **(!**visited**[**i**])** dfs**(**i**);**

reverse**(**ts**.**begin**(),** ts**.**end**());**

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

**for** **(**vector**<**int**>::**iterator it **=** ts**.**begin**();** it**!=**ts**.**end**();** it**++)** **{**

int v **=** **\***it**;**

**if** **(**v**%**2 **==** 0**)** **{**

int k **=** v**^**1**;**

**if** **(**p**[**k**])** kq**[++**res**]** **=** v**/**2**+**1**;**

p**[**v**]** **=** **true;**

**}**

**else** p**[**v**]** **=** **true;**

**}** //If ¬xi is after xi in topological sort, xi should be FALSE. It should be TRUE otherwise.

1. MO

void add**(**ll u**)** **{** res **+=** **(**1 **+** 2**\***cnt**[**a**[**u**]]++)\***a**[**u**];** **}**

void del**(**ll u**)** **{** res **-=(-**1 **+** 2**\***cnt**[**a**[**u**]]--)\***a**[**u**];** **}**

ll k **=** long**(**sqrt**(**n**));**

part **=** n **/** k **+** **(**n **%** k **>** 0**);**

FOR**(**i**,**1**,**q**)** **{**

scanf**(**"%I64d %I64d"**,** **&**b**[**i**].**l**,** **&**b**[**i**].**r**);**

b**[**i**].**cs **=** i**;**

b**[**i**].**f **=** b**[**i**].**l **/** part **-** **(**b**[**i**].**l**%**part**==** 0**);**

**}**

ll currL **=** 1**;**

ll currR **=** 1**;**

res **=** a**[**1**];**

cnt**[**a**[**1**]]++;**

FOR**(**i**,**1**,**q**)** **{**

**while** **(**currR **>** b**[**i**].**r**)** del**(**currR**--);**

**while** **(**currR **<** b**[**i**].**r**)** add**(++**currR**);**

**while** **(**currL **<** b**[**i**].**l**)** del**(**currL**++);**

**while** **(**currL **>** b**[**i**].**l**)** add**(--**currL**);**

p**[**b**[**i**].**cs**]** **=** res**;**

**}**

1. LCA

**if** **(**d**[**v**]** **<** d**[**u**])** FORE**(**i**,**17**,**0**)** **if** **(**d**[**parent**[**u**][**i**]]** **>=** d**[**v**])** u **=** parent**[**u**][**i**];**

**if** **(**d**[**u**]** **<** d**[**v**])** FORE**(**i**,**17**,**0**)** **if** **(**d**[**parent**[**v**][**i**]]** **>=** d**[**u**])** v **=** parent**[**v**][**i**];**

FORE**(**i**,**17**,**0**)** **if** **(**parent**[**u**][**i**]** **!=** parent**[**v**][**i**])** **{** u **=** parent**[**u**][**i**];** v **=** parent**[**v**][**i**];** **}**

**while** **(**u **!=** v**)** **{** u **=** parent**[**u**][**0**];** v **=** parent**[**v**][**0**];** **}**

1. HLD

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

**{**

f**[**u**]** **=** 1**;**

**for(**vector**<**int**>::**iterator it **=** a**[**u**].**begin**();** it**!=**a**[**u**].**end**();** it**++)** **{**

int v **=** **\***it**;**

**if** **(**v **!=** p**)** **{**

parent**[**v**][**0**]** **=** u**;**

d**[**v**]** **=** d**[**u**]+**1**;**

dfs**(**v**,** u**);**

f**[**u**]** **+=** f**[**v**];**

**}**

**}**

**}**

void hld**(**int u**,** int p**)**

**{**

**if** **(**chainHead**[**nChain**]** **==** 0**)** chainHead**[**nChain**]** **=** u**;**

chainInd**[**u**]** **=** nChain**;**

posInBase**[**u**]** **=** **++**nBase**;**

q**[**nBase**]** **=** u**;**

int mVtx **=** **-**1**;**

**for(**vector**<**int**>::**iterator it **=** a**[**u**].**begin**();** it**!=**a**[**u**].**end**();** it**++)** **{**

int v **=** **\***it**;**

**if** **(**v **!=** p**)**

**if** **(**mVtx **==** **-**1 **||** f**[**mVtx**]** **<** f**[**v**])** mVtx **=** v**;**

**}**

**if** **(**mVtx **>** **-**1**)** hld**(**mVtx**,** u**);**

**for(**vector**<**int**>::**iterator it **=** a**[**u**].**begin**();** it**!=**a**[**u**].**end**();** it**++)** **{**

int v **=** **\***it**;**

**if** **(**v **!=** p **&&** v **!=** mVtx**)** **{**

nChain**++;**

hld**(**v**,** u**);**

**}**

**}**

**}**

int gethld**(**int u**,** int g**)**

**{**

int uchain **=** chainInd**[**u**];**

int gchain **=** chainInd**[**g**];**

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

**while** **(**1**)** **{**

**if** **(**uchain **==** gchain**)** **{**

res **^=** get**(**1**,**1**,**n**,**posInBase**[**g**],**posInBase**[**u**]);**

**break;**

**}**

res **^=** get**(**1**,**1**,**n**,**posInBase**[**chainHead**[**uchain**]],**posInBase**[**u**]);**

u **=** parent**[**chainHead**[**uchain**]][**0**];**

uchain **=** chainInd**[**u**];**

**}**

**return** res**;**

**}**

d**[**1**]** **=** 1**;**

dfs**(**1**,**0**);**

nChain **=** 1**;**

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

nBase **=** 0**;**

hld**(**1**,** 0**);**

update**(**1**,**1**,**n**,**posInBase**[**u**],**v**);**

1. SLIDING WINDOW

void SlidingWindow**()** **{**

deque**<**II**>** window**;**

FOR**(**i**,**0**,**n**-**1**)** **{**

**while** **(!**window**.**empty**()** **&&** window**.**back**().**first **>=** s**[**i**])** window**.**pop\_back**();**

window**.**push\_back**(**II**(**s**[**i**],** i**));**

**while** **(**window**.**front**().**second **<=** i **-** m**)** window**.**pop\_front**();**

**if** **(**i **+** 1 **>=** m**)** f**[**i**]** **=** window**.**front**().**first**;**

**}**

**}**

1. MATRIX POWER

matrix **operator\*** **(**matrix a**,** matrix b**)** **{**

FOR**(**i**,**0**,**N**-**1**)** FOR**(**j**,**0**,**N**-**1**)** **{**

c**.**a**[**i**][**j**]** **=** 0**;**

FOR**(**k**,**0**,**N**-**1**)** c**.**a**[**i**][**j**]** **=** **(**c**.**a**[**i**][**j**]** **+** **(**a**.**a**[**i**][**k**]** **\*** b**.**a**[**k**][**j**])** **%** base**)** **%** base**;**

**}**

**return** c**;**

**}**

matrix pow**(**matrix a**,** ll p**)** **{**

**if** **(**p **==** 1**)** **return** a**;**

matrix e **=** pow**(**a**,** p**/**2**);**

**if** **(**p**%**2**)** **return** e**\***e**\***bmt**;**

**return** e**\***e**;**

**}**

1. TRIE

struct data **{**

ll cnt**,** g**[**2**];**

data**()** **{**

cnt **=** g**[**0**]** **=** g**[**1**]** **=** 0**;**

**}**

**}** t**[**100010**\***32**];**

void ins**(**ll k**)** **{**

ll p **=** 0**;**

t**[**p**].**cnt**++;**

FORE**(**i**,**30**,**0**)** **{**

bool e **=** k **&** **(**1LL**<<**i**);**

**if** **(**t**[**p**].**g**[**e**]** **==** 0**)** **{**

t**[**p**].**g**[**e**]** **=** **++**nw**;**

t**[**nw**]** **=** data**();**

**}**

p **=** t**[**p**].**g**[**e**];**

t**[**p**].**cnt**++;**

**}**

**}**

1. FFT

#include <bits/stdc++.h>

#define base 1000000007LL

#define ll long long

#define X first

#define Y second

#define mp make\_pair

#define pb push\_back

#define CLR(a) memset(a,0,sizeof(a))

#define FOR(i,a,b) for(ll i=(a),\_b=(b); i<=\_b; i++)

#define FORE(i,a,b) for(ll i=(a),\_b=(b); i>=\_b; i--)

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

**typedef** pair**<**int**,** int**>** II**;**

**typedef** vector**<**II**>** vi**;**

ll my\_round**(**double x**)** **{**

**if** **(**x **<** 0**)** **return** **-**my\_round**(-**x**);**

**return** **(**ll**)** **(**x **+** 1e-3**);**

**}**

const double PI **=** acos**((**double**)** **-**1.0**);**

const ll MN **=** 300000**;**

**typedef** complex**<**double**>** cplex**;**

ll rev**[**MN**],** a**[**MN**],** b**[**MN**];**

ll d**[**MN**];**

cplex wlen\_pw**[**MN**],** fa**[**MN**],** fb**[**MN**];**

void fft**(**cplex a**[],** ll n**,** bool invert**)** **{**

FOR**(**i**,**0**,**n**-**1**)** **if** **(**i **<** rev**[**i**])** swap **(**a**[**i**],** a**[**rev**[**i**]]);**

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

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

ll len2 **=** len **>>** 1LL**;**

wlen\_pw**[**0**]** **=** cplex**(**1**,** 0**);**

cplex wlen**(**cos**(**alpha**),** sin**(**alpha**));**

**for** **(**ll i **=** 1**;** i **<** len2**;** **++**i**)** wlen\_pw**[**i**]** **=** wlen\_pw**[**i**-**1**]** **\*** wlen**;**

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

cplex t**,** **\***pu **=** a**+**i**,** **\***pv **=** a **+** i **+** len2**,**

**\***pu\_end **=** a **+** i **+** len2**,** **\***pw **=** wlen\_pw**;**

**for** **(;** pu **!=** pu\_end**;** **++**pu**,** **++**pv**,** **++**pw**)** **{**

t **=** **\***pv **\*** **\***pw**;**

**\***pv **=** **\***pu **-** t**;**

**\***pu **+=** t**;**

**}**

**}**

**}**

**if** **(**invert**)** FOR**(**i**,**0**,**n**-**1**)** a**[**i**]** **/=** n**;**

**}**

void calcRev**(**ll n**,** ll logn**)** **{**

FOR**(**i**,**0**,**n**-**1**)** **{**

rev**[**i**]** **=** 0**;**

FOR**(**j**,**0**,**logn**-**1**)** **if** **(**i **&** **(**1LL **<<** j**))** rev**[**i**]** **|=** 1LL **<<** **(**logn **-** 1 **-** j**);**

**}}**

void mulpoly**(**ll a**[],** ll b**[],** ll c**[],** ll na**,** ll nb**,** ll **&**n**)** **{**

ll l **=** max**(**na**,** nb**),** logn **=** 0**;**

**for** **(**n **=** 1**;** n **<** l**;** n **<<=** 1LL**)** **++**logn**;**

n **<<=** 1LL**;** **++**logn**;**

calcRev**(**n**,** logn**);**

FOR**(**i**,**0**,**n**-**1**)** fa**[**i**]** **=** fb**[**i**]** **=** cplex**(**0**);**

FOR**(**i**,**0**,**na**-**1**)** fa**[**i**]** **=** cplex**(**a**[**i**]);**

FOR**(**i**,**0**,**nb**-**1**)** fb**[**i**]** **=** cplex**(**b**[**i**]);**

fft**(**fa**,** n**,** **false);**

fft**(**fb**,** n**,** **false);**

FOR**(**i**,**0**,**n**-**1**)** fa**[**i**]** **\*=** fb**[**i**];**

fft**(**fa**,** n**,** **true);**

FOR**(**i**,**0**,**n**-**1**)** c**[**i**]** **=** **(**ll**)(**fa**[**i**].**real**()** **+** 0.5**);**

**}**

int main**()**

**{**

ll n**;**

ios**::**sync\_with\_stdio**(**0**);**

freopen**(**"inp.txt"**,** "r"**,** stdin**);**

//freopen("out.txt", "w", stdout);

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

FOR**(**i**,**1**,**n**)** **{**

ll k**;**

cin **>>** k**;**

a**[**k**+**50000**]++;**

**}**

FOR**(**i**,**1**,**n**)** **{**

ll k**;**

cin **>>** k**;**

b**[**k**+**50000**]++;**

**}**

ll m **=** n**;**

mulpoly**(**a**,**b**,**d**,**100001**,**100001**,**n**);**

ll res **=** 0**;**

FOR**(**i**,**1**,**m**)** **{**

ll k**;**

cin **>>** k**;**

res **+=** d**[**k**+**100000**];**

**}**

cout **<<** res**;**

**return** 0**;**

**}**

1. EXTENDED EUCLID

void extendedEuclid**(**int a**,** int b**)** **{**

**if** **(**b **==** 0**)** **{** x **=** 1**;** y **=** 0**;** d **=** a**;** **return;** **}**

extendedEuclid**(**b**,** a **%** b**);**

int x1 **=** y**;**

int y1 **=** x **-** **(**a **/** b**)** **\*** y**;**

x **=** x1**;**

y **=** y1**;**

**}** // 25x+18y= 839 // store x, y, and d as global variables

1. FAT

Cayley’s Formula**:** There are n**^(**n−2**)** spanning trees of a complete graph with n labeled vertices**.**

Derangement**:** A permutation of the elements of a set such that none of the elements appear in their original position**.** The number of derangements der**(**n**)** can be computed as follow**:** der**(**n**)=(**n−1**)**×**(**der**(**n−1**)+**der**(**n−2**))** where der**(**0**)** **=** 1 **and** der**(**1**)** **=** 0.

Euler’s Formula **for** Planar Graph**:** V−E**+**F**=**2**,**where F is the number of faces of the Planar Graph**.**

Moser’s Circle**:** Determine the number of pieces into which a circle is divided **if** n points on its circumference are joined by chords with no three internally concurrent**.** Solution **:** g**(**n**)=** nC4 **+** nC2 **+** 1

Pick’s Theorem **:** Let i be the number of integer points in the polygon**,** A be the area of the polygon**,** **and** b be the number of integer points on the boundary**,** then A**=** i **+** b**/**2 −1**.**

The number of spanning tree of a complete bipartite graph K**(**n**,** m**)** is m**^(**n−1**)**×n**^(**m−1**)**

Erdos Gallai’s Theorem gives a necessary **and** sufficient condition **for** a finite sequence of natural numbers to be the degree sequence of a simple graph**.** A sequence of non negative integers d1 **>=** d2 **>=** **..** **>=** dn can be the degree sequence of a simple graph on n vertices **if** sum**(**di**)** is even **and** sum**(**I **=** 1 **->** k**)** di **<=** k**\*(**k**-**1**)** **+** sum**(**I **=** k**+**1 **->** n**)** min**(**di**,**k**)** holds **for** 1**<=**k**<=**n

1. GEOMETRY

point rotate**(**point p**,** double theta**)** **{**

double rad **=** DEG\_to\_RAD**(**theta**);**

**return** point**(**p**.**x **\*** cos**(**rad**)** **-** p**.**y **\*** sin**(**rad**),** p**.**x **\*** sin**(**rad**)** **+** p**.**y **\*** cos**(**rad**));**

**}**

bool circle2PtsRad**(**point p1**,** point p2**,** double r**,** point **&**c**)** **{**

double d2 **=** **(**p1**.**x **-** p2**.**x**)** **\*** **(**p1**.**x **-** p2**.**x**)** **+**

**(**p1**.**y **-** p2**.**y**)** **\*** **(**p1**.**y **-** p2**.**y**);**

double det **=** r **\*** r **/** d2 **-** 0.25**;**

**if** **(**det **<** 0.0**)** **return** **false;**

double h **=** sqrt**(**det**);**

c**.**x **=** **(**p1**.**x **+** p2**.**x**)** **\*** 0.5 **+** **(**p1**.**y **-** p2**.**y**)** **\*** h**;**

c**.**y **=** **(**p1**.**y **+** p2**.**y**)** **\*** 0.5 **+** **(**p2**.**x **-** p1**.**x**)** **\*** h**;**

**return** **true;**

**}**

double rInCircle**(**double ab**,** double bc**,** double ca**)** **{return** area**(**ab**,** bc**,** ca**)** **/** **(**0.5 **\*** perimeter**(**ab**,** bc**,** ca**));** **}**

double rInCircle**(**point a**,** point b**,** point c**)** **{return** rInCircle**(**dist**(**a**,** b**),** dist**(**b**,** c**),** dist**(**c**,** a**));** **}**

double rCircumCircle**(**double ab**,** double bc**,** double ca**)** **{return** ab **\*** bc **\*** ca **/** **(**4.0 **\*** area**(**ab**,** bc**,** ca**));** **}**

double rCircumCircle**(**point a**,** point b**,** point c**)** **{return** rCircumCircle**(**dist**(**a**,** b**),** dist**(**b**,** c**),** dist**(**c**,** a**));** **}**

double area**(**const vector**<**point**>** **&**P**)** **{**

double result **=** 0.0**,** x1**,** y1**,** x2**,** y2**;**

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

x1 **=** P**[**i**].**x**;** x2 **=** P**[**i**+**1**].**x**;**

y1 **=** P**[**i**].**y**;** y2 **=** P**[**i**+**1**].**y**;**

result **+=** **(**x1 **\*** y2 **-** x2 **\*** y1**);**

**}**

**return** fabs**(**result**)** **/** 2.0**;**

**}**

double gcDistance**(**double pLat**,** double pLong**,**double qLat**,** double qLong**,** double radius**)** **{**

pLat **\*=** PI **/** 180**;** pLong **\*=** PI **/** 180**;** // convert degree to radian

qLat **\*=** PI **/** 180**;** qLong **\*=** PI **/** 180**;**

**return** radius **\*** acos**(**cos**(**pLat**)\***cos**(**pLong**)\***cos**(**qLat**)\***cos**(**qLong**)** **+** cos**(**pLat**)\***sin**(**pLong**)\***cos**(**qLat**)\***sin**(**qLong**)** **+**

sin**(**pLat**)\***sin**(**qLat**));**

**}**

1. EDGE ARRAY

E **=** 0**;** memset**(**last**,** **-**1**,** **sizeof(**last**));**

void addEdge**(**int x**,** int y**)** **{** adj**[**E**]** **=** y**;** next**[**E**]** **=** last**[**x**];** last**[**x**]** **=** E**++;** **}**

1. HASH

ll getHashT**(**int i**,**int j**)** **{** **return** **(**hashT**[**j**]-**hashT**[**i**-**1**]\***POW**[**j**-**i**+**1**]+**base**\***base**)%**base**;** **}**

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

string T**,**P**;**

cin **>>** T **>>** P**;**

int m**=**T**.**size**(),**n**=**P**.**size**();**

T**=**" "**+**T**;**P**=**" "**+**P**;**

POW**[**0**]=**1**;**

FOR**(**i**,**1**,**m**)** POW**[**i**]=(**POW**[**i**-**1**]\***26**)** **%** base**;**

FOR**(**i**,**1**,**m**)** hashT**[**i**]=(**hashT**[**i**-**1**]\***26**+**T**[**i**]-**'a'**)** **%** base**;**

ll hashP**=**0**;**

FOR**(**i**,**1**,**n**)** hashP**=(**hashP**\***26**+**P**[**i**]-**'a'**)** **%** base**;**

FOR**(**i**,**1**,**m**-**n**+**1**)** **if(**hashP**==**getHashT**(**i**,**i**+**n**-**1**))** printf**(**"%d "**,**i**);**

**}**

1. KMP

const int maxN **=** 1000100**;**

char s**[**maxN**],** t**[**maxN**];**

int prev**[**maxN**];**

int main**()**

**{**

scanf**(**"%s %s"**,**s**+**1**,**t**+**1**);**

int n **=** strlen**(**s**+**1**),** m **=** strlen**(**t**+**1**);**

**for(**int i**=**0**;**i**<=**m**;**i**++)** prev**[**i**]=**0**;**

prev**[**0**]=-**1**;**

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

**{**

**for(**j**=**prev**[**i**-**1**];** j**>=**0 **and** t**[**j**+**1**]!=**t**[**i**];** j**=**prev**[**j**]);**

prev**[**i**]=**j**+**1**;**

**}**

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

**if(**s**[**i**]==**t**[**j**])**

**{**

**if(**j**==**m**)**

**{**

printf**(**"%d "**,**i**-**m**+**1**);**

j **=** prev**[**j**];**

**}**

**}** **else**

**{**

**for(**j**--;** j**>=**0 **and** t**[**j**+**1**]!=**s**[**i**];** j**=**prev**[**j**]);**

j**++;**

**}**

**return** 0**;**

**}**

1. Ukkonen O(n)

#include <bits/stdc++.h>

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

#define fpos adla

#define link suffixLink

const int inf **=** 1e9**;**

const int maxn **=** 5e5**+**100**;**

char s**[**maxn**];**

int to**[**maxn**][**256**];**

int len**[**maxn**],** fpos**[**maxn**],** link**[**maxn**],** par**[**maxn**];**

int ver**[**maxn**],** prev**[**maxn**],** fin**[**maxn**],** nE**;**

int node**,** pos**;**

int sz **=** 1**,** n **=** 0**;**

void addLink**(**int x**,**int y**)**

**{**

ver**[++**nE**]=**y**;** prev**[**nE**]=**fin**[**x**];** fin**[**x**]=**nE**;** par**[**y**]=**x**;**

**}**

int make\_node**(**int \_pos**,** int \_len**)**

**{**

fpos**[**sz**]** **=** \_pos**;** len **[**sz**]** **=** \_len**;**

**return** sz**++;**

**}**

void go\_edge**()**

**{**

**while(**pos **>** len**[**to**[**node**][(**int**)**s**[**n **-** pos**]]])**

**{**

node **=** to**[**node**][(**int**)**s**[**n **-** pos**]];**

pos **-=** len**[**node**];**

**}**

**}**

void add\_letter**(**int c**)**

**{**

s**[**n**++]** **=** c**;**

pos**++;**

int last **=** 0**;**

**while(**pos **>** 0**)**

**{**

go\_edge**();**

int edge **=** s**[**n **-** pos**];**

int **&**v **=** to**[**node**][**edge**];**

int t **=** s**[**fpos**[**v**]** **+** pos **-** 1**];**

**if(**v **==** 0**)**

**{**

v **=** make\_node**(**n **-** pos**,** inf**);**

addLink**(**node**,**v**);**

link**[**last**]** **=** node**;**

last **=** 0**;**

**}**

**else** **if(**t **==** c**)**

**{**

link**[**last**]** **=** node**;**

**return;**

**}**

**else**

**{**

int u **=** make\_node**(**fpos**[**v**],** pos **-** 1**);**

to**[**u**][**c**]** **=** make\_node**(**n **-** 1**,** inf**);**

to**[**u**][**t**]** **=** v**;**

addLink**(**u**,**to**[**u**][**c**]);**

addLink**(**u**,**to**[**u**][**t**]);**

fpos**[**v**]** **+=** pos **-** 1**;**

len **[**v**]** **-=** pos **-** 1**;**

v **=** u**;**

addLink**(**node**,**v**);**

link**[**last**]** **=** u**;**

last **=** u**;**

**}**

**if(**node **==** 0**)**

pos**--;**

**else**

node **=** link**[**node**];**

**}**

**}**

bool cmp**(**int i**,**int j**)** **{** **return** s**[**fpos**[**i**]]** **<** s**[**fpos**[**j**]];** **}**

void dfs**(**int u**,**int d**)**

**{**

d **+=** len**[**u**];**

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

int st**[**256**],** top**=**0**;**

**for(**int i**=**fin**[**u**];**i**>**0**;**i**=**prev**[**i**])** **if(**par**[**ver**[**i**]]==**u**)** st**[**top**++]=**ver**[**i**],** cnt**++;**

sort**(**st**,**st**+**top**,**cmp**);**

**for(**int i**=**0**;**i**<**top**;**i**++)** dfs**(**st**[**i**],**d**);**

**if(**cnt**==**0 **and** d**>**1**)** printf**(**"%d\n"**,**n**-**d**);**

**}**

char ss**[**maxn**];**

int main**()**

**{**

len**[**0**]** **=** inf**;**

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

strcat**(**ss**,**"$"**);**

int l **=** strlen**(**ss**);**

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

add\_letter**(**ss**[**i**]);**

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

len**[**i**]** **=** min**(**l **-** fpos**[**i**],** len**[**i**]);**

len**[**0**]=**0**;**

dfs**(**0**,**0**);**

**}**

1. SuffixArray and LCP O(n.logn)

#include <bits/stdc++.h>

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

const int maxN **=** 500000**+**10**;**

const int maxC **=** 300**;**

char s**[**maxN**];**

int head**[**maxN**],** key**[**maxN**],** b**[**maxN**],** mark**[**maxN**];**

int a**[**maxN**],** lcp**[**maxN**],** rk**[**maxN**];**

void buildSA**()**

**{**

int n **=** strlen**(**s**);**

s**[**n**++]=**'$'**;**

s**[**n**]=**'\0'**;**

**for(**int i**=**0**;**i**<**n**;**i**++)** a**[**i**]=**i**,** key**[**i**]=**s**[**i**],** mark**[**i**]=**0**;**

int nkey**=**256**;**

**for(**int k**=**0**;**k**<**n**;**k**<<=**1**,**k**+=**k**==**0**)**

**{**

**for(**int c**=**0**;**c**<**nkey**;**c**++)** head**[**c**]=**0**;**

**for(**int i**=**0**;**i**<**n**;**i**++)** b**[**i**]=(**a**[**i**]-**k**+**n**)%**n**,** head**[**key**[**b**[**i**]]]++;**

**for(**int c**=**1**;**c**<**nkey**;**c**++)** head**[**c**]+=**head**[**c**-**1**];**

**for(**int i**=**n**-**1**;**i**>-**1**;**i**--)** a**[--**head**[**key**[**b**[**i**]]]]=**b**[**i**];**

**for(**int i**=**1**;**i**<**n**;**i**++)** mark**[**i**]|=**key**[(**a**[**i**]+**k**)%**n**]!=**key**[(**a**[**i**-**1**]+**k**)%**n**];**

nkey**=**1**;**

**for(**int i**=**0**;**i**<**n**;**i**++)** nkey**+=**mark**[**i**],** key**[**a**[**i**]]=**nkey**-**1**;**

**if(**nkey**==**n**)** **break;**

**}**

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

lcp**[**0**]=**0**;**

**for(**int i**=**0**;**i**<**n**-**1**;**i**++)**

**{**

int**&** q**=**lcp**[**rk**[**i**]]=**max**(**0**,**lcp**[**rk**[**i**-**1**]]-**1**);**

int j**=**a**[**rk**[**i**]-**1**];**

**for(;**s**[**i**+**q**]==**s**[**j**+**q**];**q**++);**

**}**

**}**

int main**()**

**{**

strcpy**(**s**,**"banana"**);**

buildSA**();**

**return** 0**;**

**}**

1. Manacher

const int maxN **=** int**(**1e5**)+**111**;**

char S**[**2**\***maxN**];**

int F**[**2**\***maxN**];**

int Length\_Of\_Longest\_Palindrome**(**char S**[],** int N**)**

**{**

int i**,** L**,** R **=** L **=** **-**1**,** M**,** d **=** 0**,** res **=** 0**;**

M **=** 2**\***N**+**1**;**

**for(**i **=** M**-**1**;** i **>=** 0**;** i**--)**

**{**

**if** **(**d**)** S**[**i**]** **=** S**[(**i**-**1**)** **>>** 1**];**

**else** S**[**i**]** **=** '#'**;**

d **=** 1**-**d**;**

**}**

**for(**i **=** 0**;** i **<=** M**-**1**;** i**++)**

**{**

**if** **(**i **<=** R**)** d **=** min**(**F**[**L**+**R**-**i**],** R**-**i**);**

**else** d **=** 0**;**

**while** **((**i**-**d**-**1 **>=** 0**)** **&&** **(**i**+**d**+**1 **<** M**)** **&&** **(**S**[**i**-**d**-**1**]** **==** S**[**i**+**d**+**1**]))** d**++;**

F**[**i**]** **=** d**;**

**if** **(**i**+**d **>** R**)** L **=** i**-**d**,** R **=** i**+**d**;**

res **=** max**(**res**,** d**);**

**}**

**return** res**;**

**}**

1. Aho-Corasick

**namespace** trie

**{**

const int N **=** 2003**;**

int a**[**N**][**128**],** Peak**=**0**;**

int Prev**[**N**];**

bool Leaf**[**N**];**

void clear**()**

**{**

Peak**=**0**;**

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

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

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

**}**

void insert**(**char s**[])**

**{**

int u**=**0**;**

**for** **(**int i**=**0**;** char k**=**s**[**i**];** i**++)**

**{**

**if** **(!**a**[**u**][**k**])** a**[**u**][**k**]** **=** **++**Peak**;**

u**=**a**[**u**][**k**];**

**}**

Leaf**[**u**]=true;**

**}**

int next**(**int u**,** char k**)**

**{**

**for** **(**int i**=**u**;** i**!=-**1**;** i**=**Prev**[**i**])**

**if** **(**a**[**i**][**k**])** **return** a**[**i**][**k**];**

**return** 0**;**

**}**

void bfs**()**

**{**

queue**<**int**>** qu**;**

qu**.**push**(**0**);**

Prev**[**0**]=-**1**;**

**while** **(**qu**.**size**())**

**{**

int u**=**qu**.**front**();**

qu**.**pop**();**

**for** **(**int k**=**0**;** k**<**128**;** k**++)**

**if** **(**int v**=**a**[**u**][**k**])**

**{**

Prev**[**v**]** **=** next**(**Prev**[**u**],** k**);**

Leaf**[**v**]** **|=** Leaf**[**Prev**[**v**]];**

qu**.**push**(**v**);**

**}**

**}**

**}**

**};**

1. Palindromic Tree

const int MAXN **=** 105000**;**

struct node

**{**

int next**[**26**];**

int len**;**

int sufflink**;**

int num**;**

**};**

int len**;**

char s**[**MAXN**];**

node tree**[**MAXN**];**

int num**;**

int suff**;**

long long ans**;**

bool addLetter**(**int pos**)**

**{**

int cur **=** suff**,** curlen **=** 0**;**

int let **=** s**[**pos**]** **-** 'a'**;**

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

**{**

curlen **=** tree**[**cur**].**len**;**

**if** **(**pos **-** 1 **-** curlen **>=** 0 **&&** s**[**pos **-** 1 **-** curlen**]** **==** s**[**pos**])**

**break;**

cur **=** tree**[**cur**].**sufflink**;**

**}**

**if** **(**tree**[**cur**].**next**[**let**])**

**{**

suff **=** tree**[**cur**].**next**[**let**];**

**return** **false;**

**}**

num**++;**

suff **=** num**;**

tree**[**num**].**len **=** tree**[**cur**].**len **+** 2**;**

tree**[**cur**].**next**[**let**]** **=** num**;**

**if** **(**tree**[**num**].**len **==** 1**)**

**{**

tree**[**num**].**sufflink **=** 2**;**

tree**[**num**].**num **=** 1**;**

**return** **true;**

**}**

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

**{**

cur **=** tree**[**cur**].**sufflink**;**

curlen **=** tree**[**cur**].**len**;**

**if** **(**pos **-** 1 **-** curlen **>=** 0 **&&** s**[**pos **-** 1 **-** curlen**]** **==** s**[**pos**])**

**{**

tree**[**num**].**sufflink **=** tree**[**cur**].**next**[**let**];**

**break;**

**}**

**}**

tree**[**num**].**num **=** 1 **+** tree**[**tree**[**num**].**sufflink**].**num**;**

**return** **true;**

**}**

void initTree**()**

**{**

num **=** 2**;**

suff **=** 2**;**

tree**[**1**].**len **=** **-**1**;**

tree**[**1**].**sufflink **=** 1**;**

tree**[**2**].**len **=** 0**;**

tree**[**2**].**sufflink **=** 1**;**

**}**

1. Longest Common Prefix

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

lcp**[**0**]=**0**;**

**for(**int i**=**0**,**q**=**0**;**i**<**n**-**1**;**i**++)**

**{**

int j **=** a**[**r**[**i**]-**1**];**

**for(;**s**[**i**+**q**]==**s**[**j**+**q**];**q**++);**

lcp**[**r**[**i**]]=**q**--;**

q**+=(**q**<**0**);**

**}**

1. Lyndon

string min\_cyclic\_shift **(**string s**)** **{**

s **+=** s**;**

int n **=** **(**int**)** s**.**length**();**

int i**=**0**,** ans**=**0**;**

**while** **(**i **<** n**/**2**)** **{**

ans **=** i**;**

int j**=**i**+**1**,** k**=**i**;**

**for(;**j **<** n **&&** s**[**k**]** **<=** s**[**j**];**j**++)**

**if** **(**s**[**k**]** **<** s**[**j**])** k **=** i**;** **else** **++**k**;**

**while** **(**i **<=** k**)** i **+=** j **-** k**;**

**}**

**return** s**.**substr **(**ans**,** n**/**2**);**

**}**

1. K-th order statistic

template **<**class T**>**

T order\_statistics **(**std**::**vector**<**T**>** a**,** unsigned n**,** unsigned k**)**

**{**

**using** std**::**swap**;**

**for** **(**unsigned l**=**1**,** r**=**n**;** **;** **)**

**{**

**if** **(**r **<=** l**+**1**)**

**{**

**if** **(**r **==** l**+**1 **&&** a**[**r**]** **<** a**[**l**])**

swap **(**a**[**l**],** a**[**r**]);**

**return** a**[**k**];**

**}**

unsigned mid **=** **(**l **+** r**)** **>>** 1**;**

swap **(**a**[**mid**],** a**[**l**+**1**]);**

**if** **(**a**[**l**]** **>** a**[**r**])**

swap **(**a**[**l**],** a**[**r**]);**

**if** **(**a**[**l**+**1**]** **>** a**[**r**])**

swap **(**a**[**l**+**1**],** a**[**r**]);**

**if** **(**a**[**l**]** **>** a**[**l**+**1**])**

swap **(**a**[**l**],** a**[**l**+**1**]);**

unsigned

i **=** l**+**1**,**

j **=** r**;**

const T

cur **=** a**[**l**+**1**];**

**for** **(;;)**

**{**

**while** **(**a**[++**i**]** **<** cur**)** **;**

**while** **(**a**[--**j**]** **>** cur**)** **;**

**if** **(**i **>** j**)** **break;**

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

**}**

a**[**l**+**1**]** **=** a**[**j**];**

a**[**j**]** **=** cur**;**

**if** **(**j **>=** k**)** r **=** j**-**1**;**

**if** **(**j **<=** k**)** l **=** i**;**

**}**

**}**

1. Z-Function

#define N 500005

int n**,** z**[**N**];**

char a**[**N**];**

void make\_z**(**char a**[],** int n**,** int F**[])** **{**

int L**=-**1**,** R**=-**1**;** F**[**0**]=**n**;**

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

**if** **(**i**>**R**)** **{**

L**=**i**;** R**=**i**-**1**;**

**while** **(**R**<**n**-**1 **&&** a**[**R**+**1**]==**a**[**R**-**L**+**1**])** R**++;**

F**[**i**]=**R**-**L**+**1**;**

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

**if** **(**F**[**i**-**L**]<**R**-**i**+**1**)** F**[**i**]=**F**[**i**-**L**];**

**else** **{**

L**=**i**;**

**while** **(**R**<**n**-**1 **&&** a**[**R**+**1**]==**a**[**R**-**L**+**1**])** R**++;**

F**[**i**]=**R**-**L**+**1**;**

**}**

**}**

**}**

**}**

1. Tandem Repeats

vector**<**int**>** z\_function **(**const string **&** s**)** **{**

int n **=** **(**int**)** s**.**length**();**

vector**<**int**>** z **(**n**);**

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

**if** **(**i **<=** r**)**

z**[**i**]** **=** min **(**r**-**i**+**1**,** z**[**i**-**l**]);**

**while** **(**i**+**z**[**i**]** **<** n **&&** s**[**z**[**i**]]** **==** s**[**i**+**z**[**i**]])**

**++**z**[**i**];**

**if** **(**i**+**z**[**i**]-**1 **>** r**)**

l **=** i**,** r **=** i**+**z**[**i**]-**1**;**

**}**

**return** z**;**

**}**

void output\_tandem **(**const string **&** s**,** int shift**,** bool left**,** int cntr**,** int l**,** int l1**,** int l2**)** **{**

int pos**;**

**if** **(**left**)**

pos **=** cntr**-**l1**;**

**else**

pos **=** cntr**-**l1**-**l2**-**l1**+**1**;**

cout **<<** "[" **<<** shift **+** pos **<<** ".." **<<** shift **+** pos**+**2**\***l**-**1 **<<** "] = " **<<** s**.**substr **(**pos**,** 2**\***l**)** **<<** endl**;**

**}**

void output\_tandems **(**const string **&** s**,** int shift**,** bool left**,** int cntr**,** int l**,** int k1**,** int k2**)** **{**

**for** **(**int l1**=**1**;** l1**<=**l**;** **++**l1**)** **{**

**if** **(**left **&&** l1 **==** l**)** **break;**

**if** **(**l1 **<=** k1 **&&** l**-**l1 **<=** k2**)**

output\_tandem **(**s**,** shift**,** left**,** cntr**,** l**,** l1**,** l**-**l1**);**

**}**

**}**

inline int get\_z **(**const vector**<**int**>** **&** z**,** int i**)** **{**

**return** 0**<=**i **&&** i**<(**int**)**z**.**size**()** **?** z**[**i**]** **:** 0**;**

**}**

void find\_tandems **(**string s**,** int shift **=** 0**)** **{**

int n **=** **(**int**)** s**.**length**();**

**if** **(**n **==** 1**)** **return;**

int nu **=** n**/**2**,** nv **=** n**-**nu**;**

string u **=** s**.**substr **(**0**,** nu**),**

v **=** s**.**substr **(**nu**);**

string ru **=** string **(**u**.**rbegin**(),** u**.**rend**()),**

rv **=** string **(**v**.**rbegin**(),** v**.**rend**());**

find\_tandems **(**u**,** shift**);**

find\_tandems **(**v**,** shift **+** nu**);**

vector**<**int**>** z1 **=** z\_function **(**ru**),**

z2 **=** z\_function **(**v **+** '#' **+** u**),**

z3 **=** z\_function **(**ru **+** '#' **+** rv**),**

z4 **=** z\_function **(**v**);**

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

int l**,** k1**,** k2**;**

**if** **(**cntr **<** nu**)** **{**

l **=** nu **-** cntr**;**

k1 **=** get\_z **(**z1**,** nu**-**cntr**);**

k2 **=** get\_z **(**z2**,** nv**+**1**+**cntr**);**

**}**

**else** **{**

l **=** cntr **-** nu **+** 1**;**

k1 **=** get\_z **(**z3**,** nu**+**1 **+** nv**-**1**-(**cntr**-**nu**));**

k2 **=** get\_z **(**z4**,** **(**cntr**-**nu**)+**1**);**

**}**

**if** **(**k1 **+** k2 **>=** l**)**

output\_tandems **(**s**,** shift**,** cntr**<**nu**,** cntr**,** l**,** k1**,** k2**);**

**}**

**}**