

TABLE OF CONTENTS

Prelude.....	6
The Art of Debugging	9
LookUp Data	10
Learn Forest, Learn	11
Number Theory	18
1. The Art of Problem Solving	18
2. Notes	18
3. Combinatorics.....	26
4. nCr modulo Prime Power	32
5. Sum of nCi where i is upto k.....	33
6. Burnside Lemma.....	36
7. Number of Solutions of a Equation	36
8. Expected value.....	37
9. Prime Factorization Large	38
10. Prime Counting Function	44
11. Power Tower	46
12. Mobius Function	47
13. Modular Inverse.....	47
14. Factoradic Number	47
15. Linear Sieve	49

16. Discrete Logarithm	49
17. Chinese Remainder Theorem	51
18. Miller Robin Primality Test	52
19. Linear Diophantine Equation	53
20. Linear Diophantine Less Than or Equal	55
21. Sum of the K-th Powers.....	55
22. Rational Approximation.....	56
23. Modular Square Root	57
24. Sum of Arithmetic Progression Modular and Divided	58
25. Carmichael Lambda	58
Data Structure	61
26. Policy Based Data Structure	61
27. Monotonous Queue	61
28. Binary Indexed Tree.....	62
BIT Standard	62
BIT with range update and range query	63
BIT 2D	63
BIT 2D with range update and range query.....	64
29. Binary Search Tree	66
30. Segment Tree.....	67
Persistent Segment Tree.....	67
Dynamic Segment Tree	68

Segment Tree 2D	69
Quad Tree	71
31. Disjoint Set Union	74
Persistent DSU	74
Partially Persistent Dsu.....	76
Dynamic Connectivity Problem.....	77
Augmented DSU	80
DSU Bipartite	81
A DSU Problem	81
32. MO's Algorithm	84
MO's Algorithm Standard	84
MO's Algorithm GilbertOrder	85
Mo's on Tree.....	86
Mo's with Update	88
MO's Online	90
MO's with DSU.....	93
33. Sparse Table	96
34. Merge Sort Tree	97
35. SQRT Decomposition	97
36. SQRT Tree	98
SQRT Tree With Update	98
SQRT Tree Without Update	102
37. DSU on tree	104
38. Centroid Decomposition	104
Notes.....	104
Problem Variation 1	105
Problem Variation 2	106
Problem Variation 3	108
39. Heavy Light Decomposition	110
HLD Standard.....	110
HLD with Subtrees and Path Query.....	112
40. Treap	114
41. Wavelet Tree.....	121
42. K-D tree	123
43. Link-Cut Tree.....	125
44. Static to Dynamic Trick	129
45. Queue using two Stacks	130
46. Rope	130
47. Interval Set	130
48. Divide and Conquer for insert and query problems	132
49. Venice Technique	133
50. Cartesian Tree.....	135
Dynamic Programming	136
51. Digit DP	136

Count of Numbers	136	Xor Pairs less than k	155
Sum of Numbers.....	137	Persistent Trie.....	156
52. Convex Hull Trick.....	138	63. String Matching.....	160
Convex Hull Trick Standard	138	Knuth-Morris-Pratt.....	160
Dynamic Convex Hull Trick.....	140	Bitset	161
Persistent Convex Hull Trick.....	141	Z-Algorithm.....	161
Convex Hull Trick 2D.....	143	Aho Corasick	162
Dot Product Optimization	144	Aho Corasick Dynamic.....	164
53. Divide and Conquer Optimization.....	145	A KMP Application.....	167
54. Knuth Optimization.....	145	Aho Corasick All Pair Occurrence Relation.....	168
55. In Out DP	147	64. String Hashing	171
56. Substring DP	147	Notes.....	171
57. Bounded Knapsack.....	148	Hashing	171
58. Knapsack Branch and Bound.....	149	Hashing 2D.....	173
59. Sum of Subsets DP	150	65. String Suffix Structures	174
Standard DP	150	Suffix Array O(n)	174
Notes.....	151	Suffix Array O(nlogn)	177
60. Maximum AND subset with Given Length With Update .	152	Suffix Automaton	179
String Theory	154	Suffix Tree	182
61. Notes	154	66. Palindromes	184
62. Trie.....	154	Palindromic Tree	184
Max/Min.....	154	Manacher's Algorithm	185

Number of Palindromes in range	186
Minimum Palindrome Factorization.....	189
67. Lyndon Factorization And Minimum Rotation	192
68. Expression Parsing	193
Graph Theory.....	196
69. The Art of Problem Solving.....	196
70. Notes	196
71. Strongly Connected Components	198
72. Articulation Points	199
73. Articulation Bridges	200
Articulation Bridges Standard	200
Articulation Bridges Online	201
Articulation Bridge Tree	203
74. Biconnected Components	204
75. Block Cut Tree	206
76. Path Intersection.....	208
77. Spanning Tree	208
Notes.....	208
Prim's Minimum Spanning Tree	209
Directed Minimum Spanning Tree.....	210
Steiner tree	213
Kirchhoff's Theorem.....	215
Manhattan MST.....	216
Minimum Diameter MST	217
78. Down Trick On Tree	221
79. Lowest Common Ancestor	222
80. Shortest Paths.....	223
0-1 BFS	223
Dijkstra's Algorithm	224
Bellman-Ford Algorithm.....	225
Johnson's Algorithm.....	226
Shortest Path Faster Algorithm	227
Floyd-Warshall Algorithm	228
Shortest Path with Fixed Length.....	229
81. Finding Negative Cycle	230
82. Dominator Tree.....	231
83. 2-SAT	232
84. Maximum Clique(Brokerbosch)	234
85. Number of Different Cliques	235
86. Maximum Independent Set(Anticlique)	236
87. Euler Path	238
88. Path Intersection	239
89. Virtual Tree	240
90. Max Flow	242

Notes.....	242	Polynomial.....	283
Dinic's Algorithm	244	104. Polynomial Structure	283
Min Cost Max Flow	245	105. Polynomial Root.....	287
Bipartite Matching Unweighted	248	106. Polynomial Interpolation Standard.....	288
Bipartite matching Weighted.....	249	107. Lagrange Interpolation	289
Matching Unweighted.....	252	108. MultiPoint Evaluation	291
Matching Weighted.....	254	109. Polynomial with more features.....	292
Maximum Closure	256	Geometry	300
Gomory-Hu Tree.....	259	110. Geometry 2D.....	300
91. POSET.....	262	111. Convex Hull	315
92. Betweenness Centrality.....	264	112. Pick's Theorem.....	316
93. ST-numbering.....	265	113. Closest Pair of Points	316
94. Stable Marriage Problem.....	267	Matrix Related Algorithm	318
95. Hall Theorem.....	268	114. The Art of Problem Solving	318
96. Chromatic Number	271	115. Matrix Stucture	318
97. Transitive Closure	272	116. Gaussian Elimination	319
98. Transitive Reduction	272	Standard.....	319
99. DAG Reachability Dynamic	274	Modular	321
100. Minimum Mean Weight Cycle.....	275	Modulo 2	322
101. Prufer Code	277	XOR Basis	322
102. Tree isomorphism	279	Determinant	323
103. 3-CYCLE and 4-CYCLE	282	Determinant Modular	324

Gauss-Jordan Elimination.....	325
Thomas Algorithm	327
117. Mat Expo	328
Notes.....	328
Mat Expo Standard.....	328
118. Linear Recurrence.....	329
Slow.....	329
Fast.....	329
119. FFT and NTT.....	334
Standard FFT and NTT	334
Notes.....	338
120. Online FFT.....	343
121. Fast Walsh Hadamard Transformation.....	344
122. Freivalds Algortihm	346
Game Theory.....	347
123. Hackenbush.....	347
Miscelleneous.....	349
124. Permutation and Inversion.....	349
125. Minimum K Length Inverses To Make all 0.....	349
126. Divide and Conquer on Queries	350
127. Longest Common Subsequence.....	352
128. Fraction Structure	352
129. Berlekamp-Messey Algorithm.....	353
130. Ternary Search.....	354
131. Histogram.....	355
132. Linear Programming	356
133. Permutation Hash.....	360
134. GP Hash Table	361
135. All Pair Max and 2 nd Max Generator.....	363
136. Huffman Coding.....	363
137. All Nearest Smaller Values	367
138. Fraction Binary Search.....	367
139. Longest Zigzag Subsequence	368
140. Bit Hacks.....	369
141. Fibonacci Faster	370
142. System Of Difference Constraints.....	370
143. Slope	372
144. Sequence Merge.....	372

PRECODE

```
//#pragma comment(linker, "/stack:200000000")
```

```

//#pragma GCC optimize("Ofast")
//#pragma                                     GCC
target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
//#pragma GCC optimize("unroll-loops")

#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

#define ll long long
#define ull unsigned long long
#define ld long double
#define pii pair<int,int>
#define pll pair<ll,ll>
#define vi vector<int>
#define vll vector<ll>
#define vc vector<char>
#define vs vector<string>
#define vpll vector<pll>
#define vpii vector<pii>
#define umap unordered_map
#define uset unordered_set
#define PQ priority_queue

#define printa(a,L,R) for(int i=L;i<R;i++) cout<<a[i]<<(i==R-1?'\\n':' ')
#define printv(a) printa(a,0,a.size())

```

```

#define print2d(a,r,c) for(int i=0;i<r;i++) for(int j=0;j<c;j++)
cout<<a[i][j]<<(j==c-1?'\\n':' ')
#define pb push_back
#define eb emplace_back
#define mt make_tuple
#define fbo find_by_order
#define ook order_of_key
#define MP make_pair
#define UB upper_bound
#define LB lower_bound
#define SQ(x) ((x)*(x))
#define issq(x) (((ll)(sqrt((x))))*((ll)(sqrt((x)))))==(x))
#define F first
#define S second
#define mem(a,x) memset(a,x,sizeof(a))
#define inf 1e18
#define E 2.71828182845904523536
#define gamma 0.5772156649
#define nl "\\n"
#define lg(r,n) (int)(log2(n)/log2(r))
#define pf printf
#define sf scanf
#define sf1(a)      scanf("%d",&a)
#define sf2(a,b)    scanf("%d %d",&a,&b)
#define sf3(a,b,c) scanf("%d %d %d",&a,&b,&c)
#define pf1(a)      printf("%d\\n",a);
#define pf2(a,b)    printf("%d %d\\n",a,b)
#define pf3(a,b,c) printf("%d %d %d\\n",a,b,c)
#define sf1ll(a)   scanf("%lld",&a)

```

```

#define sf2ll(a,b)      scanf("%lld %lld",&a,&b)
#define sf3ll(a,b,c)    scanf("%lld %lld %lld",&a,&b,&c)
#define pf1ll(a)         printf("%lld\n",a);
#define pf2ll(a,b)       printf("%lld %lld\n",a,b)
#define pf3ll(a,b,c)    printf("%lld %lld %lld\n",a,b,c)
#define _ccase printf("Case %lld: ",++cs)
#define _case cout<<"Case "<<++cs<<": "
#define by(x) [](const auto& a, const auto& b) { return a.x < b.x; }

#define asche cerr<<"Ekhane asche\n";
#define rev(v) reverse(v.begin(),v.end())
#define srt(v) sort(v.begin(),v.end())
#define grtsrt(v) sort(v.begin(),v.end(),greater<ll>())
#define all(v) v.begin(),v.end()
#define mnv(v) *min_element(v.begin(),v.end())
#define mxv(v) *max_element(v.begin(),v.end())
#define toint(a) atoi(a.c_str())
#define BeatMeScanf ios_base::sync_with_stdio(false)
#define valid(tx,ty) (tx>=0&&tx<n&&ty>=0&&ty<m)
#define one(x) __builtin_popcount(x)
#define Unique(v) v.erase(unique(all(v)),v.end())
#define stree l=(n<<1),r=l+1,mid=b+(e-b)/2
#define fout(x) fixed<<setprecision(x)
string tostr(int n) {stringstream rr;rr<<n;return rr.str();}
inline void yes(){cout<<"YES\n";exit(0);}
inline void no(){cout<<"NO\n";exit(0);}
template <typename T> using o_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
//ll dx[]={1,0,-1,0,1,-1,-1,1};

```

```

//ll dy[]={0,1,0,-1,1,1,-1,-1};
//random_device rd;
//mt19937 random(rd());
int sc()
{
    register int c = getchar();
    register int x = 0;
    int neg = 0;
    for(;((c<48 || c>57) && c != '-');c = getchar());
    if(c=='-') {neg=1;c=getchar();}
    for(;c>47 && c<58;c = getchar()) {x = (x<<1) + (x<<3) + c - 48;}
    if(neg) x=-x;
    return x;
}
inline void out(int n)
{
    int N = n<0?-n:n, rev, cnt = 0;
    rev = N;
    if (N == 0) { putchar('0'); putchar('\n'); return ;}
    while ((rev % 10) == 0) { cnt++; rev /= 10;}
    if(n<0) putchar('-');
    rev = 0;
    while (N != 0) { rev = (rev<<3) + (rev<<1) + N % 10; N /= 10;}
    while (rev != 0) { putchar(rev % 10 + '0'); rev /= 10;}
    while (cnt--) putchar('0');
    putchar('\n');
    return;
}

```

```
#define debug(args...) { string _s = #args; replace(_s.begin(), _s.end(),  
' ', ' '); stringstream _ss(_s); istream_iterator<string> _it(_ss); deb(_it,  
args); }  
void deb(istream_iterator<string> it) {}  
template<typename T, typename... Args>  
void deb(istream_iterator<string> it, T a, Args... args) {  
    cerr << *it << " = " << a << endl;  
    deb(++it, args...);  
}  
  
const int mod=1e9+7;  
const int N=3e5+9;  
const ld eps=1e-9;  
const ld PI=acos(-1.0);  
//ll gc(ll a,ll b){while(b){ll x=a%b;a=b;b=x;}return a;}  
//ll lc(ll a,ll b){return a/gc(a,b)*b;}  
//ll qpow(ll n,ll k) {ll ans=1;assert(k>=0);n%=mod;while(k>0){if(k&1)  
ans=(ans*n)%mod;n=(n*n)%mod;k>>=1;}return ans%mod;}  
  
int main()  
{  
    BeatMeScanf;  
    ll i,j,k,n,m;  
  
    return 0;  
}
```

THE ART OF DEBUGGING

- ✓ Long Longs?
- ✓ Check if m,n aren't misused
- ✓ Printed enough new line or extra new line?
- ✓ Make sure output format is right(includeing YES/NO vs Yes/No or newline vs spaces)
- ✓ Run with n=1
- ✓ Have you cleared the vectors ?
- ✓ Make sure two ints aren't multiplied to get a long long
- ✓ Output enough digits after decimal point
- ✓ The exact constraints
- ✓ Check overflow(ll vs ints)
- ✓ Check all array bounds
- ✓ When using multiple dfs recursions check if inside one dfs another dfs is not called
- ✓ Case number print?
- ✓ Are you using the correct mod value?
- ✓ I spent a lot of my time debugging my solution without any success, after the contest I discovered that the obstacles in the input is 'x' (small one) while I was thinking it was 'X' (capital), I lost a bronze medal because of it :(
-kingofnumbers (a Codeforces id)
- ✓ Set or multiset?
- ✓ Different Variables with same name ?
- ✓ Inside 2d loop are you using i++ instead of j++?
- ✓ Are you using ceil function? Then remove it!
- ✓ Is inf large enough?
- ✓ For multiple queries are you returning 0 inside the queries?

- ✓ For max and min have you initialized the values by a good enough value?
- ✓ Using the local variable of the same name when global variable was required to be used.
- ✓ declared a counter of type char instead of int ,resulted in passing of pretests and failing of system test. :)
- ✓ I subtracted 1 in a for loop from v.size(). Guess what happened when the input vector empty?
- ✓ for (int i = n - 1; i--; i >= 0)
instead of:
for (int i = n - 1; i >= 0; i--)

It passed pretests and failed systests

- ✓ in 2d grid for valid(x,y) check if $x \geq 0 \& \& x < n$ hobe or $x \geq 1 \& \& x \leq n$ hobe. Again x,y duitai ki n,m er sathe compare hobe naki n,n er sathe hobe
- ✓ Are you using memset correctly?
- ✓ Use bool operators using brackets.Beware!!!
- ✓ Have you deleted debug(x) lines?
- ✓ 1 is not a prime number
- ✓ It may be scanf("%d", x). where &x is missing.
- ✓ Is Y vowel or consonant?
- ✓ Instead of printing NO printed NO.(with a zero).
- ✓ are you erasing values from a set or ant stl while parallelly traversing the elements of the stl?Please don't. This is not nice!

LOOKUP DATA

Primes Under N

Power of 10	Number of Primes
1	4
2	25
3	168
4	1,229
5	9,592
6	78,498
7	664,579
8	5,761,455
9	50,847,534
10	455,052,511
11	4,118,054,813
12	37,607,912,018
13	346,065,536,839
14	3,204,941,750,802
15	29,844,570,422,669
16	279,238,341,033,925
17	2,623,557,157,654,233
18	24,739,954,287,740,860

Here prim=largest prime under 10^k

$< 10^k$	prime	# of prime	$< 10^k$	prime
1	7	4	10	9999999967
2	97	25	11	9999999977
3	997	168	12	999999999989
4	9973	1229	13	999999999971
5	99991	9592	14	9999999999973
6	999983	78498	15	99999999999989
7	9999991	664579	16	999999999999937
8	99999989	5761455	17	999999999999997
9	99999937	50847534	18	9999999999999989

LEARN FOREST, LEARN

- $\min(a + b, c) = a + \min(b, c - a)$
- $|a - b| + |b - c| + |c - a| = 2(\max\{a, b, c\} - \min\{a, b, c\})$
- A number is Fibonacci if and only if one or both of $(5*n^2 + 4)$ or $(5*n^2 - 4)$ is a perfect square
- **Find intersection of k paths:**
Idea is very simple, select any node as root and run a dfs. For each node, keep a bitset B of size N which contains how many nodes are there in the path from the root to this node. Since N is only 10000, this is feasible.
For each path a and b, let l be the LCA of a and b. You can then find the path between a and b in $O(N/32)$, using $(B[a] | B[b]) ^ B[\text{parent}[l]]$. To find the intersection between K paths, just keep doing a bitwise AND operation on the bitset between two paths. It's basically the brute force solution, but 32 times

faster. If you look carefully at the constraints, you can see why it should pass comfortably :)

- A polite number is a number which can be represented as sum of two or more consecutive numbers.
So numbers not power of two are polite numbers.
 $n\text{-th polite number} = x + (\text{floor})(\log_2(x + \text{double}))\log_2(x))$
where $x = n + 1$
- $a + b = a \oplus b + 2(a \& b)$
- $a * b \leq c$ and $a \leq \frac{c}{b}$ are same but $a * b < c$ and $a < \frac{c}{b}$ ain't same
- There are only 3 known pairs (n, m) such that $n! + 1 = m^2$. They are $(4, 5), (5, 11), (7, 71)$
- for $i > j, \gcd(i, j) = \gcd(i - j, j) \leq (i - j)$
- for $i > j$ $i \text{ xor } j \geq (i - j)$
- Number of subsets having xor k in set $S = \{0, 1, \dots, 2^n - 1\}$ is $2^{2^n - n}$
- Sum of all subset xors of a given array (array size= n) = OR of all elements * 2^{n-1}
- $\sum_{x=1}^n [d|x^k] = \left\lfloor \frac{n}{\prod_{i=0}^k p_i^{\lfloor e_i \rfloor}} \right\rfloor$, where $d = \prod_{i=0}^k p_i^{e_i}$
- $n \% 2^i = n \& (2^i - 1)$
- $\sum_{i=1}^{n-1} i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}$
- where B_k = the Bernoulli Number
- $a \oplus b = (a|b) \& (\sim a|\sim b)$

- $\gcd(a_L, a_{L+1}, \dots, a_R) = \gcd(a_L, a_{L+1} - a_L, \dots, a_R - a_{R-1})$

If both $f(x)$ and $g(x)$ are multiplicative, then $h(x)=f(x)g(x)$ is also multiplicative.

- "1,1,2,1,2,3,1,2,3,4,1,2,3,4,5...." Is Called "Doubly Fractal Sequence"

n-th element is:

```
x = floor( ( sqrt( 8 * n + 1 ) - 1 ) / 2 );
total = n - (x * (x + 1)) / 2;
if( total == 0 && x > 0 ) cout << x << endl;
else if( total == x ) cout << x << endl;
else if( x > total ) cout << total << endl;
```

- If you convert each point $(X; Y)$ to $(X'; Y')$ so that $X' = X-Y$, $Y' = X+Y$, manhattan distance function becomes $\max(|X_1 - X_2|, |Y_1 - Y_2|)$ instead of original.

$$1.2 + 2.2^2 + 3.2^3 + \dots + n.2^n = (n-1).2^{n+1} + 2$$

- $$\sum_{k=1}^n k^3 = \underbrace{1}_{1^3} + \underbrace{3 + 5}_{2^3} + \underbrace{7 + 9 + 11}_{3^3} + \dots + \underbrace{(n(n-1)+1) + \dots + (n(n+1)-1)}_{n^3}$$

$$\sum_{k=1}^n k^3 = 1 + 3 + 5 + \dots + (n(n+1)-1)$$

The function has a minimum value at $x = a$ if $f'(a) = 0$ and $f''(a) = a$ positive number.

The function has a maximum value at $x = a$ if $f'(a) = 0$ and $f''(a) = a$ negative number.

- if we want to know the solution of something after performing k steps and we know the solution after 2^x steps then perform 2^x steps for every set bit of k . Voila! you got the result. Another

approach is to know the solution after every \sqrt{k} steps and performing a \sqrt{k} solution.

e.g.:

You are given a cyclic array A having N numbers. In an XOR round, each element of the array A is replaced by the bitwise XOR (Exclusive OR) of itself, the previous element, and the next element in the array. All operations take place simultaneously. Can you calculate A after K such XOR rounds?

In case of $k = 2^x$, $x \geq 0$ it is clear to see that answer for every index i will be

$$a_i = a_i \oplus a_{i-2^x} \oplus a_{i+2^x}$$

now use the approach pointed above.

- For finding if multiple of n numbers form a square, just xor all the prime factors of all numbers. If xor is 0 it is a square.
- Given a, b and rectangle with vertices $A(0,0), B(a,0), C(a,b), D(0,b)$. Ball starts from point A with angle 45° . Find the number of lines until ball reaches one of point A, B, C, D and point that ball reaches. $a, b \leq 100000$. Let $g = \gcd(a, b)$, $a' = a/g$ and $b' = b/g$. Count of the reflections is $a'+b'-2$ and there are 3 cases for determining the ending point:

If a' is even and b' is odd, then it is B.

If a' and b' are both odd, then it is C.

If a' is odd and b' is even, then it is D.

So you can bruteforce all of the paths until it stops in $O(n)$

- $1^{\wedge} 2^{\wedge} 3^{\wedge} \dots ^{\wedge} (4k-1) = 0$
- Find the remainder of n by modulating it with 4.
If $\text{rem} = 0$, then xor will be same as n .

- If rem = 1, then xor will be 1.
 If rem = 2, then xor will be n+1.
 If rem = 3 ,then xor will be 0.
- Given a sequence of N integer numbers. At each step it is allowed to increase the value of any number by 1 or to decrease it by 1. The goal of the game is to make the sequence **non-decreasing** with the smallest number of steps.
 $O(n^2)$ -> there exists a non-decreasing sequence, which can be obtained from the given sequence using minimal number of moves and in which all elements are equal to some element from the initial sequence (i.e. which consists only from the numbers from the initial sequence).
- $O(n \log n)$** ->
- ```

int a[N];
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin>>n;
 for(j=1;j<=n;j++) cin>>a[i];
 PQ<int>q;
 ll ans=0;
 q.push(a[1]);
 for(i=2;i<=n;i++){
 q.push(a[i]);
 if(q.top()>a[i]) ans+=q.top()-a[i],q.pop(),q.push(a[i]);
 }
 cout<<ans<<nl;
 return 0;

```

- }
- For strictly increasing sequence remove i from every  $a[i]$  and solve the upper mentioned problem!
- Zeckendorf's theorem is a theorem about the representation of integers as sums of Fibonacci numbers.  
 Zeckendorf's theorem states that every positive integer can be represented uniquely as the sum of one or more distinct Fibonacci numbers in such a way that the sum does not include any two consecutive Fibonacci numbers. More precisely, if  $N$  is any positive integer, there exist positive integers  $c_i \geq 2$ , with  $c_{i+1} > c_i + 1$ , such that
- $$N = \sum_{i=0}^k F_{c_i}$$
- Each number has exactly one representation that uses each Fibonacci number at most once and never uses two consecutive Fibonacci numbers
- in 2-SAT if you are asked to solve  $(x_1 \text{ xor } x_2) \text{ and } (x_2 \text{ xor } x_3) \dots$  then replace  $(x_1 \text{ xor } x_2)$  with  $(x_1 \text{ or } x_2) \text{ and } (\text{not } x_1 \text{ or not } x_2)$  and then do 2-sat
  - Mouular Inverse of a (modulo m) such that  $m=p^k$  is
- $$\frac{1}{a} \% p^k = a^{p^{k-1}(p-1)-1} \% p^k$$
- Number of components in a forest of trees is number of nodes-number of edges
  - we are given n numbers  $a_1, a_2, \dots, a_n$  and our task is to find a value  $x$  that minimizes the sum  $|a_1-x| + |a_2-x| + \dots + |a_n-x|$   
 optimal  $x=\text{median of the array.}$

if n is even  $x=[\text{left median}, \text{right median}]$  i.e. every number in this range

for minimizing

$$(a_1-x)^2 + (a_2-x)^2 + \dots + (a_n-x)^2$$

$$\text{optimal } x = (a_1 + a_2 + \dots + a_n)/n$$

- For minimizing a function  $f(x)$  find the minimum value M possible of  $f(x)$  for any possible value of x. Solve the equation  $f(x)=M$ .

Optimal x=average of roots of the equation

|              | set syntax      | bit syntax      |
|--------------|-----------------|-----------------|
| intersection | $a \cap b$      | $a \& b$        |
| union        | $a \cup b$      | $a \mid b$      |
| complement   | $\bar{a}$       | $\sim a$        |
| difference   | $a \setminus b$ | $a \& (\sim b)$ |

- Number of Unique Common Substrings:

Let  $A$  — set of different strings of a (first given string).

Let  $B$  — set of different strings of b (second given string).

Then  $|A \cap B| = |A| + |B| - |A \cup B|$ . You know how to calculate  $|A|$  and  $|B|$ , and you want to find  $|A \cap B|$ . Think a bit, how to obtain  $|A \cup B|$ .  
solution:

Let  $c = a + \# + b$ .

Then  $|C| = |A \cup B| + (|a| + 1) \times (|b| + 1)$ . Assuming, that character  $\#$  is neutral element, which can't appear in strings.

$(|a| + 1) \times (|b| + 1)$  — the number of substrings in  $c$ , that contain our neutral character  $\#$ .

- number of distinct strings  $S$ , which are substrings of  $A$ , but not substrings of  $B = |A| - |A \cap B|$

- Divisors in  $O(\text{cbrt } n)$

```

N = input()
primes = array containing primes till 106
ans = 1
for all p in primes :
 if p * p * p > N:
 break
 count = 1
 while N divisible by p:
 N = N/p
 count = count + 1
 ans = ans * count
if N is prime:
 ans = ans * 2
else if N is square of a prime:
 ans = ans * 3
else if N ≠ 1:
 ans = ans * 4

```

## 1 GCD on subsegments

Assume you have set of numbers in which you add elements one by one and on each step calculate  $gcd$  of all numbers from set. Then we will have no more than  $\log(a[i])$  different values of  $gcd$ . Thus you can keep compressed info about all  $gcd$  on Subsegments of  $a[i]$ :

```
int a[n];
map<int, int> sub_gcd[n];
/*
Key is gcd,
Value is the largest length such that gcd(a[i - len], ..., a[i]) equals to key.
*/
sub_gcd[0][a[0]] = 0;
for(int i = 1; i < n; i++)
{
 sub_gcd[i][a[i]] = 0;
 for(auto it: sub_gcd[i - 1])
 {
 int new_gcd = __gcd(it.first, a[i]);
 sub_gcd[i][new_gcd] = max(sub_gcd[i][new_gcd], it.second + 1);
 }
}
```

## 2 From Static Set to Expandable via $O(\log(n))$

Assume you have some static set and you can calculate some function  $f$  of the whole set such that  $f(x_1, \dots, x_n) = g(f(x_1, \dots, x_{k-1}), f(x_k, \dots, x_n))$ , where  $g$  is some function which can be calculated fast. For example,  $f()$  as the number of elements less than  $k$  and  $g(a+b) = a+b$ . Or  $f(S)$  as the number of occurrences of strings from  $S$  into  $T$  and  $g$  is a sum again.

With additional  $\log(n)$  factor you can also insert elements into your set. For this let's keep  $\log(n)$  disjoint sets such that their union is the whole set. Let the size of  $k^{th}$  be either 0 or  $2^k$  depending on binary presentation of the whole set size. Now when inserting element you should add it to  $0^{th}$  set and rebuild every set keeping said constraint. Thus  $k^{th}$  set will tell  $F(2^k)$  operations each  $2^k$  steps where  $F(n)$  is the cost of building set over  $n$  elements from scratch which is usually something about  $n$ .

## 4 Cycles in Graph as Linear Space

Assume every set of cycles in graph to be vector in  $E$ -dimensional space over  $Z_2$  having one if corresponding edge is taken into set or zero otherwise. One can consider combination of such sets of cycles as sum of vectors in such space. Then you can see that basis of such space will be included in the set of cycles which you can get by adding to the tree of depth first search exactly one edge. You can consider combination of cycles as the one whole cycle which goes through 1-edges odd number of times and even number of times through 0-edges. Thus you can represent any cycle as combination of simple cycles and any path as combination as one simple path and set of simple cycles. It could be useful if we consider paths in such a way that going through some edge twice annihilates its contribution into some final value. Example: find path from vertex  $u$  to  $v$  with minimum xor-sum.

## 5 Matrix Exponentiation Optimization

Assume we have  $n \times n$  matrix  $A$  and we have to compute  $b = A^m x$  several times for different  $m$ . Naive solution would consume  $O(qn^3 \log(n))$  time. But we can precalculate binary powers of  $A$  and use  $O(\log(n))$  multiplications of matrix and vector instead of matrix and matrix. Then the solution will be  $O((n^3 + qn^2) \log(n))$ .

## 6 Euler Tour Technique

You have a tree and there are lots of queries of kind add number on subtree of some vertex or calculate sum on the path between some vertices.

Let's consider two euler tours: in first we write the vertex when we enter it, in second we write it when we exit from it. We can see that difference between prefixes including subtree of  $v$  from first and second tours will exactly form vertices from  $v$  to the root. Thus problem is reduced to adding number on segment and calculating sum on prefixes.

## 7 From Expandable Set to Dynamic via $O(\log(n))$

Assume for some set we can make non-amortized insert and calculate some queries. Then with additional  $O(\log(n))$  factor we can handle erase queries. Let's for each element  $x$  find the moment when it's erased from set. Thus for each element we will wind segment of time  $[a, b]$  such that element is present in the set during this whole segment. Now we can come up with recursive procedure which handles  $[l, r]$  time segment considering that all elements such that  $[l, r] \subset [a, b]$  are already included into the set. Now, keeping this invariant we recursively go into  $[l, m]$  and  $[m, r]$  subsegments. Finally when we come into segment of length 1 we can handle the query having static set.

## 11 Cayley's Formula

Cayley's formula states that there are  $n^{n-2}$  labeled trees that contain  $n$  nodes. The nodes are labeled  $1, 2, \dots, n$  and two trees are different if either their structure or labeling is different.

## 12 Verifying Matrix Multiplication

We are given three matrices  $A$ ,  $B$  and  $C$ . The task is to verify if  $AB = C$  holds. Of course we can do  $O(n^3)$  multiplication but it's possible to do better than that.

We can solve the problem using a Monte Carlo algorithm whose time complexity is only  $O(n^2)$ . The idea is simple: we choose a random vector  $X$  of  $n$  elements, and calculate the matrices  $ABX$  and  $CX$ . If  $ABX = CX$ , we report that  $AB = C$ .

It can be possible that the algorithm might give wrong answer. To ensure more correctness, we can consider multiple random vectors  $X$  and do the above operation.

## 13 Graph Coloring

Given a graph that contains  $n$  nodes and  $m$  edges, our task is to find a way to color the nodes of the graph using two colors so that for at least  $m/2$  edges, the endpoints have different colors.

The problem can be solved using a Las Vegas algorithm that generates random colorings until a valid coloring has been found. In a random coloring, the color of each node is independently chosen so that the probability of both colors is  $1/2$ . In a random coloring, the probability that the endpoints of a single edge have different colors is  $1/2$ . Hence, the expected number of edges whose endpoints have different colors is  $m/2$ . Since it is expected that a random coloring is valid, we will quickly find a valid coloring in practice.

## 14 Batch Processing

We have a grid with  $n$  cells where initially all of the cells are white except one. We perform  $n - 1$  operations, each of which first calculates the minimum distance from a given white cell to a black cell, and then paints the white cell black.

The solution is to dividing the operations into  $\sqrt{n}$  batches, each of which consists of  $\sqrt{n}$  operations.

Initially, we calculate the distance of each white cell from the black cell using BFS and initialize a list of size  $\sqrt{n}$  to store latest black colored cells. When we are given the operations, we first find the distance of the white cell from the black cells stored in the list. The answer for the white cell will be the distance found from this checking and also the distance found in BFS. We color this white cell black and insert it into the list.

When the size of the list is more than  $\sqrt{n}$ , we run BFS again from the black cells and calculate minimum distance for each white cell by multi-source BFS and empty the list.

We run BFS at most  $O(\sqrt{n})$  times. And when we check linearly the distance between the black cells stored in the list and the given white cell, we can do this at most  $O(\sqrt{n})$  times because its size is not more than  $\sqrt{n}$ . So the total complexity is  $O(n\sqrt{n})$ .

## 15 Knapsack with Sqrt Decomposition

Suppose that we are given a list of integer weights whose sum is  $n$ . Our task is to find out all sums that can be formed using a subset of the weights.

By using the fact that there are at most  $\sqrt{n}$  distinct weights, we can process the weights in groups that consists of similar weights. We can process each group in  $O(\sqrt{n})$  time which yields an  $O(n\sqrt{n})$  time algorithm.

The idea is to use an array that records the sums of weights that can be formed using the groups processed so far. The array contains  $n$  elements: element  $k$  is 1 if the sum  $k$  can be formed and 0 otherwise. To process a group of weights, we scan the array from left to right and record the new sums of weights that can be formed using this group and the previous groups.

```
// dp[i] = can we make sum i
for (int i = 1; i <= n; i++)
 dp[i] = -1;
// v contains sqrt(n) different values
for (int z = 0; z < v.size(); z++)
{
 int len = v[z].first; // value of a weight
 int cnt = v[z].second; // occurrence count of a weight
 for (int x = 0; x + v[z].first <= n; x++)
 {
 int y = x + len; // let's build value y
 if (dp[x] != -1 && dp[y] == -1)
 {
 if (pr[x] != len)
 dp[y] = 1, pr[y] = len;
 else
 if (dp[x] < cnt)
 dp[y] = dp[x] + 1, pr[y] = len;
 }
 }
}
}

●
```

## 16 String Construction

Given a string  $s$  of length  $n$  and a set of strings  $D$  whose total length is  $m$ , consider the problem of counting the number of ways  $s$  can be formed as a concatenation of strings in  $D$ . For example, if  $s = ABAB$  and  $D = A, B, AB$ , there are 4 ways.

Let's assume  $count(k)$  denotes the number of ways to construct prefix  $s[0\dots k]$ .

We can solve the problem by using string hashing and the fact that there are at most  $O(\sqrt{m})$  distinct string lengths in  $D$ . First, we construct a set  $H$  that contains all hash values of the strings in  $D$ . Then, when calculating a value of  $count(k)$ , we go through all values of  $p$  such that there is a string of length  $p$  in  $D$ , calculate the hash value of  $s[k-p+1\dots k]$  and check if it belongs to  $H$ . Since there are at most  $O(\sqrt{m})$  distinct string lengths, this results in an algorithm whose running time is  $O(n\sqrt{m})$ .

## 17 Zeckendorf 's Theorem

Every positive integer can be written uniquely as a sum of distinct non-neighboring Fibonacci numbers. There may be many ways of writing a number as a sum of Fibonacci numbers, but there is only one way of writing it as a sum of non-neighboring Fibonacci numbers. For any given positive integer, a representation that satisfies the conditions of Zeckendorf's theorem can be found by using a greedy algorithm, choosing the largest possible Fibonacci number at each stage.

## 18 Chicken McNugget Theorem

The Chicken McNugget Theorem (or Postage Stamp Problem or Frobenius Coin Problem) states that for any two relatively prime positive integers  $m, n$ , the greatest integer that cannot be written in the form  $am + bn$  for nonnegative integers  $a, b$  is  $mn - m - n$ .

A consequence of the theorem is that there are exactly  $\frac{(m-1)(n-1)}{2}$  positive integers which cannot be expressed in the form  $am + bn$ . The proof is based on the fact that in each pair of the form  $(k, (m-1)(n-1) - k + 1)$ , exactly one element is expressible.

## 19 Number of Spanning Trees in a Bipartite Graph

Number of spanning trees in complete bipartite graph  $G(X, Y)$  is:  $X^{Y-1} \times Y^{X-1}$  such that  $X$  is the number of nodes in the first set and  $Y$  is the number of nodes in the second set.

## 20 Bertrand's Ballot Theorem

In combinatorics, Bertrand's ballot problem is the question: "In an election where candidate A receives  $p$  votes and candidate B receives  $q$  votes with  $p > q$ , what is the probability that A will be strictly ahead of B throughout the count?" The answer is  $\frac{p-q}{p+q}$

## ● Linear Diophantine Less Than or Equal:

```
/// number of integer points the triangle
/// ax + by <=c && x, y > 0; where a, b, c > 0
int64_t count_triangle(int64_t a, int64_t b, int64_t c){
 if(b>a) swap(a, b);
 int64_t m = c/a;
 if(a==b) return m*(m-1)/2;
 int64_t k= (a-1)/b, h = (c-a*m)/b;
```

```

 return m*(m-1)/2*k + m*h + count_triangle(b, a-b*k, c-
b*(k*m+h));
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;

 cout<<count_triangle(2323424,342423,100000000000000)<<
nl;
 return 0;
}

```

- Partitioning an array to into 2 arrays A and B such that xor of all A>xor of All B:

Let sum be the XOR of all point weights, A is the first hand score, and B is the back hand score.

If sum=0 , then A=B , so all possible xor A=xor B  
Otherwise consider the highest 1 of the sum binary, where there must be an odd number of ones . If you take any one of the points that are 1 at first, then B is 0.

- Manhattan Distance

manhattan distance between two k dimensional point A and B is

```

int ans=0;
for(i=0;i<(1<<k);i++) ans=max(ans,dA[i]+dB[rev[i]])
where rev[i]=inverse mask of i, for 101,rev[i]=010

```

ans d[i]=sum of dimensions,+ if bit 1 and – if 0  
dA for A and dB for B

Another approach:

```

int ans=0;
for(i=0;i<(1<<k);i++) ans=max(ans,dA[i]-dB[i])

```

- maximum pair AND in any pair (i,j) such that  $l \leq i < j \leq r$  is:

```

if(l==r) ans=0;
if(l+1==r) ans=l&r;
else ans=max(r&(r-1),(r-1)&(r-2))

```

- maximum pair XOR in any pair (i,j) such that  $l \leq i < j \leq r$  is:

```

int p=l^r;
int k=max set bit in p;
ans=(1<<(k+1))-1

```

- Minimum pair prime xor in range L to R

The brute force method i.e finding xor for every possible pairs of prime numbers b/w L and R and output minimum of them doesn't works here. So we need to think differently.

We should aware of the fact that if  $a \leq b \leq c$  then at least one of  $a \text{ xor } b$  and  $b \text{ xor } c$  is less than  $a \text{ xor } c$ . Here xor means bitwise xor. So we need to only find the xor of consecutive prime numbers in the range L and R.

- Minimum search in O(n) using comparator

```

// Linear search of min algorithm:
auto p = *std::min_element(pos.begin(), pos.end(),
[&](const int p1, const int p2) {
 // Binary search by equal subsequences length:
 //vugichugi
 int low = 0, high = n+1;
}

```

```

while (high - low > 1) {
 int mid = (low + high) / 2;
 if (hash(p1, mid, mxPow) == hash(p2, mid, mxPow)) {
 low = mid;
 } else {
 high = mid;
 }
}
return low < n && a[p1+low] < a[p2+low];
});

```

## NUMBER THEORY

### 1. The Art of Problem Solving

- Try thinking with prime factorization
- Think if the function is multiplicative and then try forming solution for  $p^k$  and merge for all prime.

### 2. Notes

- **Wilsons Theorem**

In number theory, Wilson's theorem states that a natural number  $n > 1$  is a prime number if and only if the product of all the positive integers less than  $n$  is one less than a multiple of  $n$ . That is

$$(n - 1)! \equiv (n - 1) \equiv -1 \pmod{n}$$

exactly when  $n$  is a prime number.

- **The Chicken McNugget Theorem:** It states that for any two relatively prime positive integers  $m, n$ , the greatest integer that cannot be written in the form  $am + bn$  for nonnegative integers  $a, b$  is  $mn - m - n$ .

A consequence of the theorem is that there are

$$\frac{(m - 1)(n - 1)}{2}$$

exactly  $\frac{(m - 1)(n - 1)}{2}$  positive integers which cannot be expressed in the form  $am + bn$ . The proof is based on the fact that in each pair of the form  $(k, (m - 1)(n - 1) - k + 1)$ , exactly one element is expressible.

- no three positive integers  $a, b$ , and  $c$  can satisfy the equation  $a^n + b^n = c^n$  for any integer value of  $n$  greater than two.
- **the four color theorem**, or the four color map theorem, states that, given any separation of a plane into contiguous regions, producing a figure called a map, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color.
- **Euler's conjecture** is a disproved conjecture in mathematics related to Fermat's last theorem. It states that for all integers  $n$  and  $k$  greater than 1, if the sum of  $n$   $k$ th powers of positive integers is itself a  $k$ th power, then  $n$  is greater than or equal to  $k$ :

$$a_1^k + a_2^k + \dots + a_n^k = b^k \Rightarrow n \geq k$$

Although the conjecture holds for the case  $k = 3$  (which follows from Fermat's last theorem for the third powers), it was disproved for  $k = 4$  and  $k = 5$ . It is unknown whether the conjecture fails or holds for any value  $k \geq 6$ .

- the second Hardy–Littlewood conjecture concerns the number of primes in intervals. The conjecture states that  $\pi(x + y) \leq \pi(x) + \pi(y)$  for  $x, y \geq 2$ , where  $\pi(x)$  denotes the prime-counting function, giving the number of prime numbers up to and including  $x$ . This means that the number of primes from  $x + 1$  to  $x + y$  is always less than or equal to the number of primes from 1 to  $y$ .
- sum of two squares theorem:** An integer greater than one can be written as a sum of two squares if and only if its prime decomposition contains no prime congruent to 3 (mod 4) raised to an odd power. So  $2 \cdot 5 \cdot 7^2$  is expressible where  $2 \cdot 5 \cdot 7^3$  is not.
- for odd prime  $p$ , this is true:  

$$p = x^2 + 2y^2 \Leftrightarrow p \equiv 1 \text{ or } p \equiv 3 \pmod{8}$$

$$p = x^2 + 3y^2 \Leftrightarrow p \equiv 1 \pmod{3}.$$

$$a_1 = 1$$

$$a_2 = 5$$

$$a_n = a_{n-1} + n^2(a_{n-2} + 1)$$
- Prove  $a_n = (n+1)! - 1$
- $6|(7^n - 1)$  for all  $n$
- Finite Or Infinite Fraction**

You are given several queries. Each query consists of three integers  $p, q$  and  $b$ . You need to answer whether the result of  $p/q$  in notation with base  $b$  is a finite fraction.

A fraction in notation with base  $b$  is finite if it contains finite number of numerals after the decimal point. It is also possible that a fraction has zero numerals after the decimal point.

**Solution:**

First, if  $p$  and  $q$  are not coprime, divide them on  $\gcd(p, q)$ . Fraction is finite if and only if there is integer  $k$  such that  $q \mid p \cdot b^k$ . Since  $p$  and  $q$  are being coprime now,  $q \mid b^k \Rightarrow$  all prime factors of  $q$  are prime factors of  $b$ . Now we can do iterations  $q = q \div \gcd(b, q)$ , while  $\gcd(q, b) \neq 1$ . If  $q \neq 1$  after iterations, there are prime factors of  $q$  which are not prime factors of  $b \Rightarrow$  fraction is Infinite, else fraction is Finite. But this solution works in  $O(n \log^2 10^{18})$ . Lets add  $b = \gcd(b, q)$  in iterations and name iterations when  $\gcd(b, q)$  changes iterations of the first type and when it doesn't change — iterations of the second type. Iterations of second type works summary in  $O(\log 10^{18})$ . Number of iterations of the first type is  $O(\log 10^{18})$  too but on each iteration  $b$  decreases twice. Note that number of iterations in Euclid's algorithm is equal to number of these decreases. So iterations of first type works in  $O(\log 10^{18})$  summary. Total time complexity is  $O(n \log 10^{18})$

- Euler's Totient function
    - The function is multiplicative.

This means that if  $\gcd(m, n) = 1$ , then  $\phi(mn) = \phi(m) \phi(n)$ .

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$
  - If  $p$  is prime and  $k \geq 1$ , then
$$\varphi(p^k) = p^{k-1}(p-1) = p^k \left(1 - \frac{1}{p}\right)$$
  - $J_k(n)$ , the Jordan totient function, is the number of  $k$ -tuples of positive integers all less than or equal to  $n$  that form a coprime  $(k+1)$ -tuple together with  $n$ . It is a generalization of Euler's totient,  $\phi(n) = J_1(n)$ .
- $$J_k(n) = n^k \prod_{p|n} \left(1 - \frac{1}{p^k}\right)$$
- $$\sum_{d|n} J_k(d) = n^k$$

- $\sum_{d|n} \varphi(d) = n$
- $\varphi(n) = \sum_{d|n} \mu(d) \cdot \frac{n}{d} = n \sum_{d|n} \frac{\mu(d)}{d}$
- $\varphi(n) = \sum_{d|n} d \cdot \mu\left(\frac{n}{d}\right)$ 
  - $a | b \implies \varphi(a) | \varphi(b)$
  - $n | \varphi(a^n - 1)$  for  $a, n > 1$
- $\varphi(mn) = \varphi(m)\varphi(n) \cdot \frac{d}{\varphi(d)}$  where  $d = \gcd(m, n)$

Note the special cases

- $\varphi(2m) = \begin{cases} 2\varphi(m) & \text{if } m \text{ is even} \\ \varphi(m) & \text{if } m \text{ is odd} \end{cases}$
- $\varphi(n^m) = n^{m-1}\varphi(n)$
- $\varphi(\text{lcm}(m, n)) \cdot \varphi(\gcd(m, n)) = \varphi(m) \cdot \varphi(n)$

Compare this to the formula

- $\text{lcm}(m, n) \cdot \gcd(m, n) = m \cdot n$

(See least common multiple.)

- $\varphi(n)$  is even for  $n \geq 3$ . Moreover, if  $n$  has  $r$  distinct odd prime factors,  $2^r | \varphi(n)$

- $\sum_{d|n} \frac{\mu^2(d)}{\varphi(d)} = \frac{n}{\varphi(n)}$
- $\sum_{\substack{1 \leq k \leq n \\ (k,n)=1}} k = \frac{1}{2}n\varphi(n)$  for  $n > 1$

$$\frac{\varphi(n)}{n} = \frac{\varphi(\text{rad}(n))}{\text{rad}(n)} \quad \text{where,}$$

$$\text{rad}(n) = \prod_{\substack{p|n \\ p \text{ prime}}} p$$

- $\left\lfloor \frac{n}{\varphi(n)} \right\rfloor$  is periodic. 1,2,1,2,1,3,1,2,1,2,1,3...
- $\sum_{\substack{1 \leq k \leq n \\ \gcd(k,n)=1}} \gcd(k-1, n) = \varphi(n)d(n)$  where  $d(n)$  is number of divisors.
- same equation for  $\gcd(ak-1, n)$  where  $a$  and  $n$  are coprime.
- for every  $n$  there is at least one other integer  $m \neq n$  such that  $\phi(m) = \phi(n)$ .

### Divisor function

$$\sigma_x(n) = \sum_{d|n} d^x,$$

- It is **multiplicative** i.e. if  $\gcd(a, b) = 1 \implies \sigma_x(ab) = \sigma_x(a)\sigma_x(b)$ .
- $\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$ .
- $\prod_{d|n} d = \begin{cases} n^{\frac{\sigma_0}{2}}, & n \text{ is not a perfect square} \\ \sqrt{n} \cdot n^{\frac{\sigma_0-1}{2}}, & n \text{ is a perfect square} \end{cases}$
- using a set of primes  $P = \{p_1, p_2, \dots, p_n\}$ , we can form a new prime  $p_1.p_2 \cdots p_n + 1$
- a number has odd number of divisors if it is a perfect square.
- the count of numbers such that

1. sum of divisors of each of those is odd  $\leq n$  is  $\sqrt{n} + \sqrt{\frac{n}{2}}$
  2. each has odd number of odd divisors is  $\sqrt{n} + \sqrt{\frac{n}{2}}$
- **Goldbach's conjecture:** Each even integer  $n > 2$  can be represented as a sum  $n = a + b$  so that both  $a$  and  $b$  are primes.
  - **Twin prime conjecture:** There is an infinite number of pairs of the form  $\{p, p+2\}$ , where both  $p$  and  $p+2$  are primes.
  - **Legendre's conjecture:** There is always a prime between numbers  $n^2$  and  $(n+1)^2$ , where  $n$  is any positive integer.

### Fibonacci Number

- $F_0=0, F_1=1$  and  $F_n=F_{n-1}+F_{n-2}$

$$F_n = \sum_{k=0}^{\left\lfloor \frac{n-1}{2} \right\rfloor} \binom{n-k-1}{k}$$

$$F_n = \frac{1}{\sqrt{5}} \left( \frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left( \frac{1-\sqrt{5}}{2} \right)^n$$

$$\sum_{i=1}^n F_i = F_{n+2} - 1$$

$$\sum_{i=0}^{n-1} F_{2i+1} = F_{2n}$$

$$\sum_{i=1}^n F_{2i} = F_{2n+1} - 1.$$

$$\sum_{i=1}^n F_i^2 = F_n F_{n+1}$$

Cassini's identity states that

$$F_n^2 - F_{n+1} F_{n-1} = (-1)^{n-1}$$

Catalan's identity is a generalization:

- $F_n^2 - F_{n+r} F_{n-r} = (-1)^{n-r} F_r^2$
- $F_m F_{n+1} - F_{m+1} F_n = (-1)^n F_{m-n}$
- $F_{2n} = F_{n+1}^2 - F_{n-1}^2 = F_n (F_{n+1} + F_{n-1})$
- $F_m F_n + F_{m-1} F_{n-1} = F_{m+n-1}$
- $F_m F_{n+1} + F_{m-1} F_n = F_{m+n}$ .
- Every third number of the sequence is even and more generally, every  $k$ th number of the sequence is a multiple of  $F_k$ .
- $\gcd(F_m, F_n) = F_{\gcd(m,n)}$
- Any three consecutive Fibonacci numbers are pairwise coprime, which means that, for every  $n$ ,  $\gcd(F_n, F_{n+1}) = \gcd(F_n, F_{n+2}) = \gcd(F_{n+1}, F_{n+2}) = 1$ .
- If  $p$  is a prime,
$$\begin{cases} p = 5 & \Rightarrow p \mid F_p, \\ p \equiv \pm 1 \pmod{5} & \Rightarrow p \mid F_{p-1} \\ p \equiv \pm 2 \pmod{5} & \Rightarrow p \mid F_{p+1} \end{cases}$$
- The only nontrivial square Fibonacci number is 144. Attila Pethő proved in 2001 that there is only a finite number of perfect power Fibonacci numbers. In 2006, Y. Bugeaud, M. Mignotte, and S. Siksek proved that 8 and 144 are the only such non-trivial perfect powers.
- If the members of the Fibonacci sequence are taken mod  $n$ , the resulting sequence is periodic with period at most  $6n$ .

## Sum of floors

```

///sum K=1...n floor((k*p)/q)
///works faster for n,p,q<=1e18
long long floor_sum(long long n, long long p, long long q){
 long long t = __gcd(p, q);
 p/=t, q/=t;
 long long s=0, z=1;
 while(q>0 && n>0){
 t = p/q;
 s+= n*(n+1)/2*z*t;
 p-= q*t;
 t = n/q;
 s+= z*p*t*(n+1) - z*t*(p*q*t+p+q-1)/2;
 n-= q*t;
 t = n*p/q;
 s+= z*t*n;
 n = t;
 swap(p, q);
 z*=-1;
 }
 return s;
}
///sum k=1...n floor(n/k)
///complexity O(sqrt(n))
|| get(|| n)
{
 || sum=0;
 for (|| i = 1, last; i <= n; i = last + 1) {
 last = n / (n / i);
 }
}

```

```

 debug(i,last,n/i);
 sum+=(n/i)*(last-i+1);
 //n / x yields the same value for i <= x <= last.
}
return sum;
}

```

## Mobius Function and Inversion

### Notes

- For any positive integer  $n$ , define  $\mu(n)$  as the sum of the primitive  $n$ th roots of unity. It has values in  $\{-1, 0, 1\}$  depending on the factorization of  $n$  into prime factors:
  - $\mu(n) = 1$  if  $n$  is a square-free positive integer with an even number of prime factors.
  - $\mu(n) = -1$  if  $n$  is a square-free positive integer with an odd number of prime factors.
  - $\mu(n) = 0$  if  $n$  has a squared prime factor.

Here, a root of unity, occasionally called a de Moivre number, is any complex number that gives 1 when raised to some positive integer power  $n$ .

An  $n$ th root of unity, where  $n$  is a positive integer (i.e.  $n = 1, 2, 3, \dots$ ), is a number  $z$ (maybe complex) satisfying the equation  $z^n = 1$ .

An  $n$ th root of unity is said to be primitive if it is not a  $k$ th root of unity for some smaller  $k$ , that is if

$$z^n = 1 \quad \text{and} \quad z^k \neq 1 \text{ for } k = 1, 2, 3, \dots, n-1.$$

- It is a multiplicative function.

- $\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{if } n = 1, \\ 0 & \text{if } n > 1. \end{cases}$
- $\sum_{i=1}^n [\gcd(i, n) = k] = \varphi\left(\frac{n}{k}\right)$
- $\sum_{k=1}^n \gcd(k, n) = \sum_{d|n} d \cdot \varphi\left(\frac{n}{d}\right)$
- $\sum_{k=1}^n \frac{1}{\gcd(k, n)} = \sum_{d|n} \frac{1}{d} \cdot \varphi\left(\frac{n}{d}\right) = \frac{1}{n} \sum_{d|n} d \cdot \varphi(d)$
- $\sum_{k=1}^n \frac{k}{\gcd(k, n)} = \frac{n}{2} \cdot \sum_{d|n} \frac{1}{d} \cdot \varphi\left(\frac{n}{d}\right) = \frac{n}{2} \cdot \frac{1}{n} \cdot \sum_{d|n} d \cdot \varphi(d)$
- $\sum_{k=1}^n \frac{n}{\gcd(k, n)} = 2 * \sum_{k=1}^n \frac{k}{\gcd(k, n)} - 1, \text{ for } n > 1$
- 
- Given several integers, with integer  $x$  appears  $c_x$  times, and some fixed integer  $m$ . It is asked that how many integers that are co-prime to  $m$ . so,

$$\sum_{i=1}^n c_i [\gcd(i, m) = 1] = \sum_{d|m} \mu(d) \sum_{i=1}^{\lfloor n/d \rfloor} c_{id}$$

The classic version states that if  $g$  and  $f$  are arithmetic functions satisfying

$$g(n) = \sum_{d|n} f(d) \quad \text{for every integer } n \geq 1$$

then

- $f(n) = \sum_{d|n} \mu(d) g\left(\frac{n}{d}\right) \quad \text{for every integer } n \geq 1$
- $\sum_{d|n} \mu(d) = [n = 1]$
- $\sum_{i=1}^n \sum_{j=1}^n [\gcd(i, j) = 1] = \sum_{d=1}^n \mu(d) \left\lfloor \frac{n}{d} \right\rfloor^2$
- $\sum_{i=1}^n \sum_{j=1}^n \gcd(i, j) = \sum_{d=1}^n \varphi(d) \left\lfloor \frac{n}{d} \right\rfloor^2$
- if  $F(n) = \prod_{d|n} f(d)$ , then  $f(n) = \prod_{d|n} F\left(\frac{n}{d}\right)^{\mu(d)}$

### mobius function

```
int mob[N];
void mobius()
{
 for(int i=1;i<N;i++) mob[i]=3;
 mob[1]=1;
 for(int i=2;i<N;i++){
 if(mob[i]==3){
 mob[i]=-1;
 for(int j=2*i;j<N;j+=i) mob[j]=(mob[j]==3?-1:mob[j]*(-1));
 if(i<=(N-1)/i) for(int j=i*i;j<N;j+=i*i) mob[j]=0;
 }
 }
}
```

### GCD and LCM

- $\gcd(a, 0) = a$
- $\gcd(a, b) = \gcd(b, a \bmod b)$
- Every common divisor of  $a$  and  $b$  is a divisor of  $\gcd(a, b)$ .
- If  $a$  divides the product  $b \cdot c$ , and  $\gcd(a, b) = d$ , then  $a/d$  divides  $c$ .
- If  $m$  is any integer, then  $\gcd(a + m \cdot b, b) = \gcd(a, b)$
- The gcd is a multiplicative function in the following sense: if  $a_1$  and  $a_2$  are relatively prime, then  $\gcd(a_1 \cdot a_2, b) = \gcd(a_1, b) \cdot \gcd(a_2, b)$ .
- $\gcd(a, b) \cdot \text{lcm}(a, b) = |a \cdot b|$
- $\gcd(a, \text{lcm}(b, c)) = \text{lcm}(\gcd(a, b), \gcd(a, c))$
- $\text{lcm}(a, \gcd(b, c)) = \gcd(\text{lcm}(a, b), \text{lcm}(a, c))$ .

- For non-negative integers  $a$  and  $b$ , where  $a$  and  $b$  are not both zero,

$$\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1.$$

$$\gcd(a, b) = \sum_{k|a \text{ and } k|b} \varphi(k)$$

- $\gcd(\text{lcm}(a, b), \text{lcm}(b, c), \text{lcm}(a, c)) = \text{lcm}(\gcd(a, b), \gcd(b, c), \gcd(a, c))$
- $\gcd(a_L, a_{L+1}, \dots, a_R) = \gcd(a_L, a_{L+1} - a_L, \dots, a_R - a_{R-1})$ .
- Bezout's identity — Let  $a$  and  $b$  be integers with greatest common divisor  $d$ . Then, there exist integers  $x$  and  $y$  such that  $ax + by = d$ . More generally, the integers of the form  $ax + by$  are exactly the multiples of  $d$  and  $d$  is the smallest of the possible numbers.

Bézout's identity can be extended to more than two integers: if

$$\gcd(a_1, a_2, \dots, a_n) = d$$

then there are integers  $x_1, x_2, \dots, x_n$  such that

$$d = a_1x_1 + a_2x_2 + \cdots + a_nx_n$$

has the following properties:

- $d$  is the smallest positive integer of this form
- every number of this form is a multiple of  $d$
- the number of lattice points on segment  $(x_1, y_1)$  to  $(x_2, y_2)$  is  $\gcd(\text{abs}(x_1-x_2), \text{abs}(y_1-y_2))+1$

## Pythagorean Triples

- A Pythagorean triple consists of three positive integers  $a$ ,  $b$ , and  $c$ , such that  $a^2 + b^2 = c^2$ . Such a triple is commonly written  $(a, b, c)$

- Euclid's formula is a fundamental formula for generating Pythagorean triples given an arbitrary pair of integers  $m$  and  $n$  with  $m > n > 0$ . The formula states that the integers

$$a = m^2 - n^2, \quad b = 2mn, \quad c = m^2 + n^2$$

form a Pythagorean triple. The triple generated by Euclid's formula is primitive if and only if  $m$  and  $n$  are coprime and not both odd. When both  $m$  and  $n$  are odd, then  $a$ ,  $b$ , and  $c$  will be even, and the triple will not be primitive; however, dividing  $a$ ,  $b$ , and  $c$  by 2 will yield a primitive triple when  $m$  and  $n$  are coprime and both odd.

- The following will generate all Pythagorean triples uniquely:  

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2)$$
where  $m$ ,  $n$ , and  $k$  are positive integers with  $m > n$ , and with  $m$  and  $n$  coprime and not both odd.

**Theorem.** The number of Pythagorean triples  $\{a, b, n\}$  with  $\max\{a, b, n\} = n$  is given by

$$\frac{1}{2} \left( \prod_{p^\alpha \parallel n} (2\alpha + 1) - 1 \right),$$

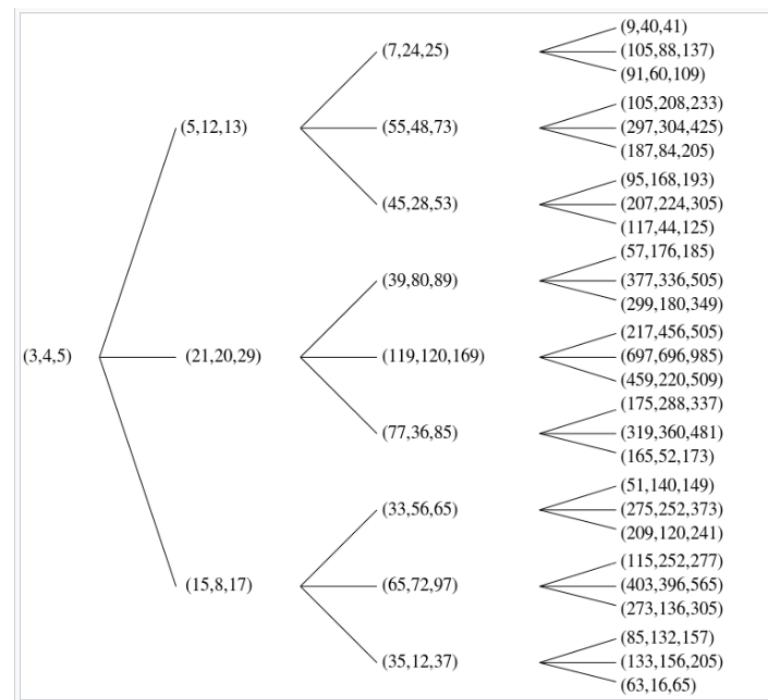
where the product is over all prime divisors  $p$  of the form  $4k + 1$ .

The notation  $p^\alpha \parallel n$  stands for  $p^\alpha \mid n$  and  $p^{\alpha+1} \nmid n$ .

**Example.** For  $n = 2 \cdot 3^2 \cdot 5^3 \cdot 7^4 \cdot 11^5 \cdot 13^6$ , the number of Pythagorean triples with hypotenuse  $n$  is  $\frac{1}{2}(7 \cdot 13 - 1) = 45$ .

To obtain a formula for the number of Pythagorean triples with hypotenuse less than a specific positive integer  $N$ , we may add the numbers corresponding to each  $n < N$  given by the Theorem. There is no simple way to compute this as a function of  $N$ .

- A primitive Pythagorean triple is a Pythagorean triple  $(a, b, c)$  such that  $\text{GCD}(a, b, c) = 1$
- The largest number that always divides  $abc$  is 60 (i.e.  $3 \cdot 4 \cdot 5$ ).
- a tree of primitive Pythagorean triples is a data tree in which each node branches to three subsequent nodes with the infinite set of all nodes giving all (and only) primitive Pythagorean triples without duplication.



- when any of the three matrices

$$A = \begin{bmatrix} 1 & -2 & 2 \\ 2 & -1 & 2 \\ 2 & -2 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 3 \end{bmatrix} \quad C = \begin{bmatrix} -1 & 2 & 2 \\ -2 & 1 & 2 \\ -2 & 2 & 3 \end{bmatrix}$$

is multiplied on the right by a column vector whose components form a Pythagorean triple, then the result is another column vector whose components are a different Pythagorean triple. If the initial triple is primitive, then so is the one that results. Thus each primitive Pythagorean triple has three "children". All primitive Pythagorean triples are descended in this way from the triple  $(3, 4, 5)$ , and no

primitive triple appears more than once. The result may be graphically represented as an infinite ternary tree with (3, 4, 5) at the root node.

### Sum of Squares Function

- The function is defined as

$$r_k(n) = |\{(a_1, a_2, \dots, a_k) \in \mathbf{Z}^k : n = a_1^2 + a_2^2 + \dots + a_k^2\}|$$

- The number of ways to write a natural number as sum of two squares is given by  $r_2(n)$ . It is given explicitly by

$$r_2(n) = 4(d_1(n) - d_3(n))$$

where  $d_1(n)$  is the number of divisors of  $n$  which are congruent with 1 modulo 4 and  $d_3(n)$  is the number of divisors of  $n$  which are congruent with 3 modulo 4.

The prime factorization  $n = 2^g p_1^{f_1} p_2^{f_2} \cdots q_1^{h_1} q_2^{h_2} \cdots$ , where  $p_i$  are the prime factors of the form  $p_i \equiv 1 \pmod{4}$ , and  $q_i$  are the prime factors of the form  $q_i \equiv 3 \pmod{4}$  gives another formula

$r_2(n) = 4(f_1 + 1)(f_2 + 1) \cdots$ , if all exponents  $h_1, h_2, \dots$  are even. If one or more  $h_i$  are odd, then  $r_2(n) = 0$ .

- The number of ways to represent  $n$  as the sum of four squares was due to Carl Gustav Jakob Jacobi and it is eight times the sum of all its divisors which are not divisible by 4, i.e.

$$r_4(n) = 8 \sum_{d|n, 4 \nmid d} d$$

$$r_8(n) = 16 \sum_{d|n} (-1)^{n+d} d^3$$

### Gauss Circle Theorem

- The Gauss circle problem is the problem of determining how many integer lattice points there are in a circle centered at the origin and with radius  $r$ .
- Since the equation of this circle is given in Cartesian coordinates by  $x^2 + y^2 = r^2$ , the question is equivalently asking

how many pairs of integers  $m$  and  $n$  there are such that  $m^2 + n^2 \leq r^2$

- If the answer for a given  $r$  is denoted by  $N(r)$  then

$$N(r) = 1 + 4 \sum_{i=0}^{\infty} \left( \left\lfloor \frac{r^2}{4i+1} \right\rfloor - \left\lfloor \frac{r^2}{4i+3} \right\rfloor \right)$$

- A much simpler sum appears if the sum of squares function  $r_2(n)$  is defined as the number of ways of writing the number  $n$  as the sum of two squares. Then

$$N(r) = \sum_{n=0}^{r^2} r_2(n).$$

## 3. Combinatorics

### Notes

- $\sum_{0 \leq k \leq n} \binom{n-k}{k} = \text{Fib}_{n+1}$
- $\binom{n}{k} = \binom{n}{n-k}$
- $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$
- $k \binom{n}{k} = n \binom{n-1}{k-1}$
- $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$
- $\sum_{i=0}^n \binom{n}{i} = 2^n$
- $\sum_{i \geq 0} \binom{n}{2i} = 2^{n-1}$
- $\sum_{i \geq 0} \binom{n}{2i+1} = 2^{n-1}$
- $\sum_{i=0}^k (-1)^i \binom{n}{i} = (-1)^k \binom{n-1}{k}$
- $\sum_{i=0}^k \binom{n+i}{i} = \binom{n+k+1}{k}$
- $\sum_{i=0}^k \binom{n+i}{n} = \binom{n+k+1}{k}$
- $\sum_{i=0}^k \binom{i}{n} = \binom{k+1}{n+1}$
- $1 \cdot \binom{n}{1} + 2 \cdot \binom{n}{2} + 3 \cdot \binom{n}{3} + \cdots + n \cdot \binom{n}{n} = n \cdot 2^{n-1}$

- $1^2 \cdot \binom{n}{1} + 2^2 \cdot \binom{n}{2} + 3^2 \cdot \binom{n}{3} + \dots + n^2 \cdot \binom{n}{n} = (n+n^2) \cdot 2^{n-2}$
- Vandermonde's Identity:  $\sum_{k=0}^r \binom{m}{k} \binom{n}{r-k} = \binom{m+n}{r}$
- Hockey-Stick Identity:  $n, r \in \mathbb{N}, n > r, \sum_{i=r}^n \binom{i}{r} = \binom{n+1}{r+1}$   
 $\sum_{i=0}^k \binom{k}{i}^2 = \binom{2k}{k}$
- $\sum_{k=0}^n \binom{n}{k} \binom{n}{n-k} = \binom{2n}{n}$
- $\sum_{k=q}^n \binom{n}{k} \binom{k}{q} = 2^{n-q} \binom{n}{q}$
- $\sum_{i=0}^n 3^i \binom{n}{i} = 4^n$
- $\sum_{i=0}^n \binom{2n}{i} = 2^{2n-1} + \frac{1}{2} \binom{2n}{n}$
- $\sum_{i=1}^n \binom{n}{i} \binom{n-1}{i-1} = \binom{2n-1}{n-1}$
- $\sum_{i=0}^n \binom{2n}{i}^2 = \frac{1}{2} \left\{ \binom{4n}{2n} + \binom{2n}{n}^2 \right\}$
- An integer  $n \geq 2$  is prime if and only if all the intermediate binomial coefficients  $\binom{n}{1}, \binom{n}{2}, \dots, \binom{n}{n-1}$  are divisible by  $n$ .
- $\binom{n+k}{k}$  divides  $\frac{\text{lcm}(n, n+1, \dots, n+k)}{n}$
- Kummer's theorem states that for given integers  $n \geq m \geq 0$  and a prime number  $p$ , the largest power of  $p$  dividing  $\binom{n}{m}$  is equal to the number of carries when  $m$  is added to  $n - m$  in base  $p$ .
- Number of different binary sequences of length  $n$  such that no two 0's are adjacent =  $\text{Fib}_{n+1}$

- Combination with repetition: Let's say we choose  $k$  elements from an  $n$ -element set, the order doesn't matter and each element can be chosen more than once. In that case, the number of different combinations is:  $\binom{n+k-1}{k}$
- Number of ways to divide  $n$  different persons in  $n/k$  equal groups i.e. each having size  $k$  is  $\binom{n-1}{k-1}$
- The number non-negative solution of the equation  $x_1+x_2+x_3+\dots+x_k=n$  is  $\binom{n+k-1}{n}$
- Number of binary sequence of length  $n$  and with  $k$  '1' is  $\binom{n}{k}$
- The number of ordered pairs  $(a, b)$  of binary sequences of length  $n$ , such that the distance between them is  $k$ , can be calculated as follows:  $\binom{n}{k} \cdot 2^n$ .  
The distance between  $a$  and  $b$  is the number of components that differs in  $a$  and  $b$  — for example, the distance between  $(0, 0, 1, 0)$  and  $(1, 0, 1, 1)$  is 2.

Note that the "first" binomial coefficient for fixed  $n$  is  $nC0$ . Let  $f(n) = nC0 + nC1 + \dots + nCr-1$ . Using the "Pascal's triangle" identity,  $nCr = (n-1)Cr-1 + (n-1)Cr$  we have

$$\begin{aligned} nC0 + nC1 + nC2 + \dots + nCr-1 \\ &= (n-1)Cr-1 + (n-1)C0 + (n-1)C1 + (n-1)C2 + \dots + (n-1)Cr-2 + (n-1)Cr-1 \\ &= 2[(n-1)C0 + (n-1)C1 + (n-1)C2 + \dots + (n-1)C(r-2)] + (n-1)Cr-1 \\ &= 2[nC0 + \dots + (n-1)Cr-1] - (n-1)Cr-1, \end{aligned}$$

i.e.,  $f(n) = 2f(n-1) - (n-1)Cr-1$  so each of the sums can be computed from the previous by doubling the previous and subtracting  $(n-1)Cr-1$ .

For example, if  $r=3$ , then

$$\begin{aligned} f(0) &= 1, \\ f(1) &= 1 + 1 = 2 = 2f(0) - 0C2, \\ f(2) &= 1 + 2 + 1 = 4 = 2f(1) - 1C2, \\ f(3) &= 1 + 3 + 3 = 7 = 2f(2) - 2C2, \\ f(4) &= 1 + 4 + 6 = 11 = 2f(3) - 3C2, \\ f(5) &= 1 + 5 + 10 = 16 = 2f(4) - 4C2, \end{aligned}$$

- for finding lots of  $f(n,k)=nC0+nC1+..nCr$  queries use MO's algo along with previous formula for transition

## Code

```
int f[N],inv[N],finv[N];
void pre()
{
 f[0]=1;
 for(int i=1;i<N;i++) f[i]=1LL*i*f[i-1]%mod;
 inv[1] = 1;
 for (int i = 2; i < N; i++) {
 inv[i] = (-(1LL*mod/i) * inv[mod%i]) % mod;
 inv[i] = (inv[i] + mod)%mod;
 }
 finv[0]=1;
 for(int i=1;i<N;i++) finv[i]=1LL*inv[i]*finv[i-1]%mod;
}
```

```
int ncr(int n,int r)
{
 if(n<r || n<0 || r<0) return 0;
 return 1LL*f[n]*finv[n-r]%mod*finv[r]%mod;
}
//int C[N][N];
//void pre()
//{
// for(int i=0;i<N;i++) C[i][0]=1;
// for(int i=1;i<N;i++){
// for(int j=1;j<N;j++) C[i][j]=(C[i-1][j-1]+C[i-1][j])%mod;
// }
//}
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 pre();
 cout<<ncr(5,3)<<nl;
 return 0;
}
```

## Catalan numbers

- $\checkmark C_n = \frac{1}{n+1} \binom{2n}{n}$
- $\checkmark C_0 = 1, C_1 = 1$  and  $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
- $\checkmark 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786$

- ✓ Number of correct bracket sequence consisting of n opening and n closing brackets.
- ✓ The number of ways to completely parenthesize  $n+1$  factors.
- ✓ The number of triangulations of a convex polygon with  $n+2$  sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- ✓ The number of ways to connect the  $2n$  points on a circle to form  $n$  disjoint i.e. non-intersecting chords.
- ✓ The number of monotonic lattice paths from point  $(0,0)$  to point  $(n,n)$  in a square lattice of size  $n \times n$ , which do not pass above the main diagonal (i.e. connecting  $(0,0)$  to  $(n,n)$ ).
- ✓ Number of permutations of length  $n$  that can be stack sorted (i.e. it can be shown that the rearrangement is stack sorted if and only if there is no such index  $i < j < k$ , such that  $a_k < a_i < a_j$  ).
- ✓ The number of non-crossing partitions of a set of  $n$  elements.
- ✓ The number of rooted full binary trees with  $n+1$  leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
- ✓ The number of Dyck words of length  $2n$ . A Dyck word is a string consisting of  $n$  X's and  $n$  Y's such that no initial segment of the string has more Y's than X's. For example, the following are the Dyck words of length 6: XXXYYY XYXXXX XYXYXY XXYYXY XXYXXX.

- ✓ The number of different ways a convex polygon with  $n+2$  sides can be cut into triangles by connecting vertices with straight lines (a form of Polygon triangulation)
- ✓ Number of permutations of  $\{1, \dots, n\}$  that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing subsequence. For  $n = 3$ , these permutations are 132, 213, 231, 312 and 321. For  $n = 4$ , they are 1432, 2143, 2413, 2431, 3142, 3214, 3241, 3412, 3421, 4132, 4213, 4231, 4312 and 4321
- ✓ Number of ways to tile a staircase shape of height  $n$  with  $n$  rectangles.

### Narayana numbers

- ✓  $N(n,k) = \frac{1}{n} \binom{n}{k} \binom{n}{k-1}$
- ✓ The number of expressions containing  $n$  pairs of parentheses, which are correctly matched and which contain  $k$  distinct nestings. For instance,  $N(4, 2) = 6$  as with four pairs of parentheses six sequences can be created which each contain two times the sub-pattern '()' :

$((()))$   $(())()$   $((())())$   $(((())))$   $(((())))$   $((((())}))$

- ✓ The number of paths from  $(0, 0)$  to  $(2n, 0)$ , with steps only northeast and southeast, not straying below the x-axis, with  $k$  peaks. And sum of all number of peaks is Catalan number.

## Stirling numbers of the first kind

- ✓ The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).
- ✓  $S(n,k)$  counts the number of permutations of  $n$  elements with  $k$  disjoint cycles.
- ✓  $S(n,k) = (n - 1) * S(n - 1, k) + S(n - 1, k - 1)$ ,  
where  $S(0,0) = 1, S(n,0) = S(0,n) = 0$
- ✓  $\sum_{k=0}^n S(n,k) = n!$

Code:

```
/*add ntt template*/
vi vec[N];
int stirling(int n,int k)
{
 if(n==0&&k==0) return 1;
 if(n<=0| |k<=0) return 0;
 int mx=1;
 while(mx<n) mx<<=1;
 for(int i=0;i<n;i++){
 vec[i].eb(i);
 vec[i].eb(1);
 }
 for(int i=n;i<mx;i++) vec[i].eb(1);
 for(int j=mx;j>1;j>>=1){
 int d=j>>1;
 for(int i=0;i<d;i++) vec[i]=multiply(vec[i],vec[i+d]);
 }
 if(k>=vec[0].size()) return 0;
}
```

```
return vec[0][k];
}
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,a,b;
 cin>>n>>k;
 cout<<stirling(n,k)<<nl;
 return 0;
}
```

## Stirling numbers of the second kind

- ✓ Stirling number of the second kind is the number of ways to partition a set of  $n$  objects into  $k$  non-empty subsets.
- ✓  $S(n,k) = k * S(n - 1, k) + S(n - 1, k - 1)$ ,  
where  $S(0,0) = 1, S(n,0) = S(0,n) = 0$
- ✓  $S(n,2)=2^{n-1}-1$
- ✓  $S(n,k)*k! =$  number of ways to color  $n$  nodes using colors from 1 to  $k$  such that each color is used at least once.

Code:

```
/*
the Stirling number of the second kind
the number of ways of partitioning a set of n elements into k
nonempty sets
*/
int f[N],inv[N],finv[N];
void pre()
{
 f[0]=1;
```

```

for(int i=1;i<N;i++) f[i]=1LL*i*f[i-1]%mod;
inv[1] = 1;
for (int i = 2; i < N; i++) {
 inv[i] = (-(1LL*mod/i) * inv[mod%i]) % mod;
 inv[i] = (inv[i] + mod)%mod;
}
finv[0]=1;
for(int i=1;i<N;i++) finv[i]=1LL*inv[i]*finv[i-1]%mod;
}
int ncr(int n,int r)
{
 if(n<r || n<0 || r<0) return 0;
 return 1LL*f[n]*finv[n-r]%mod*finv[r]%mod;
}

int S[1010][1010];
///O(nk);
int stirling_brute(int n,int k)
{
 if(n==0&&k==0) return 1;
 if(n==0||k==0) return 0;
 int &ret=S[n][k];
 if(ret!=-1) return ret;
 return ret=(1LL*k*stirling_brute(n-1,k)%mod+stirling_brute(n-1,k-1))%mod;
}

///O(klogn)
int stirling_number_of_the_second_kind(int n, int k) {
 assert (0 <= n and 0 <= k);
}

```

```

if(n==0&&k==0) return 1;
if(n==0 || k==0) return 0;
int ans = 0;
for(int i=1;i<=k;i++){
 int parity = ((k - i) % 2 == 0 ? +1 : -1);
 parity=(parity+mod)%mod;
 ans += 1LL*ncr(k, i) * qpow(i,n)%mod * parity%mod;
 ans%=mod;
}
return 1LL*ans * finv[k]%mod;
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 mem(S,-1);
 pre();
 for(i=0;i<100;i++) for(j=0;j<100;j++){

assert(stirling_brute(i,j)==stirling_number_of_the_second_kind(i,j));
 }
 cout<<stirling_number_of_the_second_kind(100000,12345)<<nl;
 return 0;
}

```

### Bell number

- ✓ Counts the number of partitions of a set.
- ✓  $B_{n+1} = \sum_{k=0}^n \binom{n}{k} * B_k$
- ✓  $B_n = \sum_{k=0}^n S(n, k)$ , where  $S(n, k)$  is stirling number of second kind.

- ✓ The number of multiplicative partitions of a **squarefree** number with  $i$  prime factors is the  $i$ -th Bell number,  $B_i$ .
- ✓ If a deck of  $n$  cards is shuffled by repeatedly removing the top card and reinserting it anywhere in the deck (including its original position at the top of the deck), with exactly  $n$  repetitions of this operation, then there are  $n^n$  different shuffles that can be performed. Of these, the number that return the deck to its original sorted order is exactly  $B_n$ . Thus, the probability that the deck is in its original order after shuffling it in this way is  $B_n/n^n$ .

### Lucas Theorem

- ✓ If  $p$  is prime then  $\binom{p^a}{k} \equiv 0 \pmod{p}$
- ✓ For non-negative integers  $m$  and  $n$  and a prime  $p$ , the following congruence relation holds:

$$\binom{m}{n} \equiv \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p},$$

where,

$$m = m_k p^k + m_{k-1} p^{k-1} + \dots + m_1 p + m_0,$$

and

$$n = n_k p^k + n_{k-1} p^{k-1} + \dots + n_1 p + n_0$$

are the base  $p$  expansions of  $m$  and  $n$  respectively. This uses the convention that  $\binom{m}{n} = 0$ , when  $m < n$ .

### Derangement

- ✓ A derangement is a permutation of the elements of a set, such that no element appears in its original position.
- ✓  $d(n) = (n - 1) * (d(n - 1) + d(n - 2))$ ,

where  $d(0) = 1, d(1) = 0$

$$\checkmark d(n) = \left\lfloor \frac{n!}{e} \right\rfloor, n \geq 1$$

### 4. nCr modulo Prime Power

```
//Complexity-(prime^k)*logn
|| F[N];|| N is maximum value of prime^k
```

```
void pre(|| mod,|| p){ // mod is p^a, p is prime
 F[0] = 1;
 for(int i=1;i<=mod;i++)
 {
 if(i%p!= 0) F[i]=(F[i-1]*i)%mod; // we keep in F factorial with the
 terms which are coprime with p
 else F[i]=F[i-1];
 }
}
```

```
|| fact2(|| n,|| mod){
 || cycle = n/mod;
 || n2=n%mod;
 return (qpow(F[mod],cycle,mod)*F[n2])%mod;
}
```

```
|| fact(|| N,|| p,|| mod){ // returns highest power of p that divides N
 and the coprime with p part of N! %mod
 || n = N;
 || ord = 0;
 while(n > 0){
```

```

n /= p;
ord += n;
}

|| ans = 1;
n = N;
while(n > 0){
 ans =(ans*fact2(n,mod))%mod;
 n/=p;
}

return MP(ord,ans);
}

|| modInverse(|| a, || m,|| p) /// modular inverse of a mod m, where
m=p^x type
{
 || pr=m/p;
 pr*=(p-1);
 pr--;
 return qpow(a,pr,m);
}
///a^k
|| get(|| a,|| k)
{
 if(a==1) return 1;
 || ans=1;
 while(k) ans*=a,--k;
 return ans;
}

}
///C(n,r) modulo p^k
|| nCrPower(|| n,|| r,|| p,|| k)
{
 || mod=get(p,k),temp;
 pre(mod,p);
 || x=fact(n,p,mod),y=fact(r,p,mod),z=fact(n-r,p,mod);
 || mul=x.second*modInverse(y.second,mod,p);
 mul%=mod;
 mul*=modInverse(z.second,mod,p);
 mul%=mod;
 || mul2=x.first-y.first-z.first;
 mul*=qpow(p,mul2,mod);
 mul%=mod;
 return mul;
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;

cout<<nCrPower(898989891010201LL,898989891010199LL,103,2)<
<nl;
 return 0;
}

5. Sum of nCi where i is upto k
//returns sum of nCi,0<=i<=k in sqrt(n).log^2(n) (maybe)

```

```
#define maxn 405//maximum value of sqrt(n)
inline int ksm(int x,int k){int s=1;for(;k>>=1,x=1ll*x*x%mod)if(k&1)s=1ll*s*x%mod;return s;}
inline int del(int x){return x<0?x+mod:x;}
inline int add(int x){return x<mod?x:x-mod;}
/** use ntt structure here*/
typedef vector<int> vec;
int n,v,rev,fac[maxn],inv[maxn],mul[maxn];
int gd[maxn],g2d[maxn],fd[maxn],f2d[maxn],dlt[maxn];
int A[maxn],B[maxn],C[maxn];
int pre_[maxn],*pre=pre_+1,suf[maxn];
inline void calc(int *h,int *out,int d,int k)
{
 int off=k-d-1;mul[0]=1;
 for(int i=1;i<=2*d+3;i++)
 {
 int s=(i+off)%mod;s=del(s);
 mul[i]=1ll*mul[i-1]*s%mod;
 }
 int len=1;for(;len<3*d+5;len<<=1);;
 vi a,b;
 for(int i=0;i<=d;i++)
 {
 A[i]=1ll*h[i]*inv[i]%mod*inv[d-i]%mod;
 if((d-i)&1)A[i]=mod-A[i];
 a.eb(A[i]);
 }
 for(int i=0;i<=2*d;i++)B[i]=ksm((i-d+k)%mod,mod-2),B[i]=del(B[i]),b.eb(B[i]);
}
```

```
vi c=ntt::multiply(a,b,mod);
while(sz(c)<len) c.eb(0);
for(int i=0;i<len;i++) C[i]=c[i];
for(int i=0;i<=d;i++)
{
 out[i]=1ll*C[i+d]*mul[i+k-off]%mod*ksm(mul[i+k-d-off-1],mod-2)%mod;
 out[i]=del(out[i]);
}
void solve(int x,int *G,int *F)
{
 if(x==0){G[0]=1;F[0]=0;return;}
 if(x&1)
 {
 solve(x-1,G,F);
 for(int i=0;i<x;i++)gd[i]=G[i];
 calc(gd,dlt,x-1,-1ll*(n+1)*rev%mod);
 for(int i=0;i<x;i++)G[i]=1ll*G[i]*(i*v+x)%mod;
 int p=1;
 for(int i=1;i<=x;i++)p=1ll*p*(v*x+i)%mod;G[x]=p;
 for(int i=0;i<x;i++)
 {
 if((x-1)&1)F[i]=-dlt[i],F[i]=del(F[i]);
 else F[i]+=dlt[i],F[i]=add(F[i]);
 F[i]=1ll*F[i]*(i*v+x)%mod;
 }
 suf[x+1]=1;for(int i=x;i>=0;i--)
 }suf[i]=1ll*suf[i+1]*(v*x+i)%mod;
}
```

```

 pre[-1]=1;for(int i=0;i<=x;i++)pre[i]=1ll*pre[i-1]*(n-i-
v*x)%mod,pre[i]=del(pre[i]);
 p=0;for(int i=0;i<x;i++)p=(p+1ll*suf[i+1]*pre[i-
1])%mod;F[x]=p;
 return;
 }
 int d=x>>1;solve(d,G,F);
 for(int i=0;i<=d;i++)gd[i]=G[i];
 calc(gd,gd+d+1,d,d+1);
 calc(gd,g2d,2*d,1ll*d*rev%mod);
 calc(gd,dlt,2*d,-1ll*(n+1)*rev%mod);
 for(int i=0;i<=d;i++)fd[i]=F[i];
 calc(fd,fd+d+1,d,d+1);
 calc(fd,f2d,2*d,1ll*d*rev%mod);
 for(int i=0;i<=x;i++)
 {
 int s=1ll*dlt[i]*f2d[i]%mod;if(d&1)s=mod-s;
 F[i]=(1ll*g2d[i]*fd[i]+s)%mod;
 }
 for(int i=0;i<=x;i++)G[i]=1ll*gd[i]*g2d[i]%mod;
}
int m,F[maxn],G[maxn];
inline int get(int x,int y)
{
 n=x,m=y;
 v=sqrt(n+0.5);rev=ksm(v,mod-2);
 solve(v,G,F);
 int fac=1;for(int i=0;i<v;i++)fac=1ll*fac*G[i]%mod;
 for(int i=v*v+1;i<=n;i++)fac=1ll*fac*i%mod;
}

```

```

for(int i=0;i<=v;i++)gd[i]=G[i];
calc(gd,dlt,v,-1ll*(n+1)*rev%mod);
pre[-1]=1;for(int i=0;i<=v;i++)pre[i]=1ll*pre[i-1]*dlt[i]%mod;
suf[v]=1;for(int i=v*v+1;i<=n;i++)suf[v]=1ll*suf[v]*i%mod;
for(int i=v-1;i>=0;i--)suf[i]=1ll*suf[i+1]*G[i]%mod;
int ans=0,cr=0;
for(int i=0;i<v&&(i+1)*v-1<=m;i++)
{
 int t=1ll*suf[i+1]*pre[i-
1]%mod*F[i]%mod;if((i*v)&1)t=mod-t;
 ans+=t;ans=add(ans);cr=i+1;
}
ans=1ll*ans*ksm(fac,mod-2)%mod;
int C=fac,Mg=1;
for(int i=0;i<cr;i++)Mg=1ll*Mg*G[i]%mod;
int s=n-cr*v,cp=0;
for(int i=0;i<v&&(i+1)*v<=s;i++)Mg=1ll*Mg*G[i]%mod,cp=i+1;
for(int i=cp*v+1;i<=s;i++)Mg=1ll*Mg*i%mod;
C=1ll*C*ksm(Mg,mod-2)%mod;
for(int i=cr*v;i<=m;i++)
{
 if(i!=cr*v)C=1ll*C*ksm(i,mod-2)%mod*(n-i+1)%mod;
 ans+=C;ans=add(ans);
}
for(int i=0;i<=v;i++)F[i]=G[i]=0;
return ans;
}

int32_t main()

```

```

{
 BeatMeScanf;
 int i,j,k,m,t;
 fac[0]=1;for(int i=1;i<maxn;i++)fac[i]=1ll*fac[i-1]*i%mod;
 inv[maxn-1]=ksm(fac[maxn-1],mod-2);
 for(int i=maxn-2;i>=0;i--)inv[i]=1ll*inv[i+1]*(i+1)%mod;
 cin>>t;
 while(t--){
 cin>>n>>m;
 cout<<get(n,m)<<nl;
 }
 return 0;
}

```

## 6. Burnside Lemma

The task is to count the number of different necklaces from  $n$  beads, each of which can be painted in one of the  $k$  colors. When comparing two necklaces, they can be rotated, but not reversed (i.e. a cyclic shift is permitted).

Solution:

$$ans = \frac{1}{n} \sum_{i=1}^n k^{\gcd(i,n)} = \frac{1}{n} \sum_{d|n} \phi\left(\frac{n}{d}\right) k^d$$

## 7. Number of Solutions of a Equation

- Number of solutions of

$$x_1 + x_2 + x_3 + \cdots + x_k = n, x_i \geq 0$$

is:  $\binom{n+k-1}{k-1}$

- Number of solutions of  
 $x_1 + x_2 + x_3 + \cdots + x_k = n, x_i \geq a_i$   
is:  $\binom{n-\sum_{i=1}^k a_i+k-1}{k-1}$
- Number of solutions of  
 $x_1 + x_2 + x_3 + \cdots + x_k = n, x_i \leq a_i$   
is:  
Problem : Codeforces 451E  
|| f[25];  
|| ncr(|| n,|| k)  
{  
 if(k>n) return 0;  
 || ans=1;  
 k=min(n-k,k);  
 for(|| i=n-k+1;i<=n;i++) ans=(ans\*(i%mod))%mod;  
 ans=ans\*qpow(f[k],mod-2)%mod;  
 return ans;  
}  
|| a[25];  
int main()  
{  
 fast;  
 || i,j,k,n,m,s;  
 cin>>n>>s;// n elements // sum is s  
 for(i=0;i<n;i++) cin>>a[i];  
 f[0]=1;  
 for(i=1;i<25;i++) f[i]=i\*f[i-1]%mod;  
 || ans=0;  
 for(i=0;i<(1<<n);i++){  
 || sum=s,cnt=0;

```

for(j=0;j<n;j++){
 if((i>>j)&1){
 sum-=a[j]+1;
 cnt++;
 }
 ll res=ncr(sum+n-1,n-1);
 if(cnt%2==1) res*=-1;
 ans=(ans+res)%mod;
 ans=(ans+mod)%mod;
}
cout<<ans<<nl;
return 0;
}

```

- Number of solutions of

$$x_1 + x_2 + x_3 + \dots + x_k = n, \\ 1 \leq x_i \leq mx, \text{ all } x_i \text{ are distinct}$$

is:

Problem : Codeforces 403D

```

ll f[50];
int dp[N][N][50];
//number of solutions of equation c1+c2+c3+...c(len)=sum
//such that c1<c2<c3<...<c(len) and 1<=c(i)<=mx
int yo(int sum,int mx,int len)
{
 if(len==0) return 1;
 if(sum<=0) return 0;
 if(mx<=0) return 0;
 int &ret=dp[sum][mx][len];
 if(ret!=-1) return ret;
 ll ans=0;

```

```

 if(sum>=mx) ans+=yo(sum-mx,mx-1,len-1);
 ans+=yo(sum-1,mx,len);
 ans+=yo(sum,mx-1,len);
 ans-=yo(sum-1,mx-1,len);
 if(ans<0) ans+=mod;
 ans%=mod;
 return ret=ans;
}
int main()
{
 fast;
 int i,j,k,n,m,t;
 f[0]=1;
 for(i=1;i<50;i++) f[i]=1LL*i*f[i-1]%mod;
 mem(dp,-1);
 cin>>n>>k;//mx=k
 ll ans=yo(n,n,k);
 ans=(ans*f[k])%mod;
 cout<<ans<<nl;
 return 0;
}

```

## 8. Expected value

Mathematically, for a discrete variable X with probability function P(X), the expected value E[X] is given by  $\sum x_i P(x_i)$  the summation runs over all the distinct values  $x_i$  that the variable can take.

The rule of "linearity of expectation" says that  $E[x_1+x_2] = E[x_1] + E[x_2]$ .

- the expected number of coin flips for getting N consecutive heads is  $(2^{N+1} - 2)$ .

- A fair coin flip experiment is carried out N times. The expected number of heads is  $N/2$ .
- The expected number of coin flips to ensure that there are atleast N heads in  $2N$ .
- Bernoulli trial is a random experiment with exactly two possible outcomes, "success" and "failure".
- If the probability of success of a bernoulli trial is  $p$  then the expected number of trials to get a success is  $1/p$ .
- If probability of success in a bernoulli trial is  $p$ , then the expected number of trials to guarantee  $N$  successes is  $N/p$ .
- A random variable corresponding to a binomial is denoted by  $B(n,k)$  and is said to have a *binomial distribution*. If the probability of success is  $p$  and failure is  $q$  then the probability of exactly  $k$  successes in the experiment  $B(n,k)$  is given by:

$$B(n, k) = \binom{n}{k} p^k q^{n-k}$$

## 9. Prime Factorization Large

```
// Integer factorization in O(N^{1/4})
// uses squof from msieve https://github.com/radii/msieve
// with fixes to work for n = p^3
// works up to 10^18
// probably fails on 5003^5 which is ~10^{18.5}
```

```
namespace NT{
 template<typename T>
 struct bigger_type{};
```

```
template<typename T> using bigger_type_t = typename
bigger_type<T>::type;
template<> struct bigger_type<int>{using type = long long;};
template<> struct bigger_type<unsigned int>{using type =
unsigned long long;};
//template<> struct bigger_type<int64_t>{using type = __int128;};
//template<> struct bigger_type<uint64_t>{using type = unsigned
__int128;};

template<typename int_t = unsigned long long>
struct Mod_Int{
 static inline int_t add(int_t const&a, int_t const&b, int_t
const&mod){
 int_t ret = a+b;
 if(ret>=mod) ret-=mod;
 return ret;
 }
 static inline int_t sub(int_t const&a, int_t const&b, int_t
const&mod){
 return add(a, mod-b);
 }
 static inline int_t mul(int_t const&a, int_t const&b, int_t
const&mod){
 uint64_t ret = a * (uint64_t)b - (uint64_t)((long double)a * b /
mod - 1.1) * mod;
 if(-ret < ret){
 ret = mod-1-(-ret-1)%mod;
 } else {
 ret%=mod;
 }
 }
};
```

```

 }

 //ret = min(ret, ret+mod);
 int64_t out = ret;
 /*if(out != a*(__int128) b % mod){
 cerr << (long double)a * b / mod << " "
<< (uint64_t)((long double)a * b / mod - 0.1) << "\n";
 cerr << mod << " " << ret << " " <<
ret+mod << " " << out << " " << (int64_t)(a*(__int128) b % mod) <<
"\n";
 assert(0);
 }*/
 return out;
 //return a*static_cast(b)%mod;
}

static inline int_t pow(int_t const&a, int_t const&b, int_t
const&mod){
 int_t ret = 1;
 int_t base = a;
 for(int i=0;b>>i;++i){
 if((b>>i)&1) ret = mul(ret, base, mod);
 base = mul(base, base, mod);
 }
 return ret;
}
};

template<typename T>

```

```

typename enable_if<is_integral<T>::value, bool>::type is_prime(T
x){
 if(x<T(4)) return x>T(1);
 for(T i=2;i*i<=x;++i){
 if(x%i == 0) return false;
 }
 return true;
}

template<typename T>
typename enable_if<is_integral<T>::value, bool>::type
miller_rabin_single(T const&x, T base){
 if(x<T(4)) return x>T(1);
 if(x%2 == 0) return false;
 base%=x;
 if(base == 0) return true;

 T xm1 = x-1;
 int j = 1;
 T d = xm1/2;
 while(d%2 == 0){ // could use __builtin_ctz
 d/=2;
 ++j;
 }
 T t = Mod_Int<T>::pow(base, d, x);
 if(t==T(1) || t==T(xm1)) return true;
 for(int k=1;k<j;++k){
 t = Mod_Int<T>::mul(t, t, x);
 if(t == xm1) return true;
 }
}
```

```

 if(t<=1) break;
 }
 return false;
}

template<typename T>
typename enable_if<is_integral<T>::value, bool>::type
miller_rabin_multi(T const&){return true;}

template<typename T, typename... S>
typename enable_if<is_integral<T>::value, bool>::type
miller_rabin_multi(T const&x, T const&base, S const&...bases){
 if(!miller_rabin_single(x, base)) return false;
 return miller_rabin_multi(x, bases...);
}

template<typename T>
typename enable_if<is_integral<T>::value, bool>::type
miller_rabin(T const&x){
 if(x < 316349281) return miller_rabin_multi(x, T(11000544),
T(31481107));
 if(x < 4759123141ull) return miller_rabin_multi(x, T(2), T(7),
T(61));
 return miller_rabin_multi(x, T(2), T(325), T(9375), T(28178),
T(450775), T(9780504), T(1795265022));
}

template<typename T>
typename enable_if<is_integral<T>::value, T>::type
isqrt(T const&x){

```

```

 assert(x>=T(0));
 T ret = static_cast<T>(sqrtl(x));
 while(ret>0 && ret*ret>x) --ret;
 while(x-ret*ret>2*ret)
 ++ret;
 return ret;
}

template<typename T>
typename enable_if<is_integral<T>::value, T>::type
icbrt(T const&x){
 assert(x>=T(0));
 T ret = static_cast<T>(cbrt(x));
 while(ret>0 && ret*ret*ret>x) --ret;
 while(x-ret*ret*ret>3*ret*(ret+1))
 ++ret;
 return ret;
}

/*uint64_t isqrt(unsigned __int128 const&x){
 unsigned __int128 ret = sqrtl(x);
 while(ret>0 && ret*ret>x) --ret;
 while(x-ret*ret>2*ret)
 ++ret;
 return ret;
}*/
vector<uint16_t> saved;
// fast prime factorization from
// https://github.com/radii/msieve
uint64_t squfof_iter_better(uint64_t const&x, uint64_t const&k,
uint64_t const&it_max, uint32_t cutoff_div){

```

```

if(__gcd((uint64_t)k, x)!=1) return __gcd((uint64_t)k, x);
//cerr << "try: " << x << " " << k << "\n";
saved.clear();
uint64_t scaledn = k*x;
if(scaledn>>62) return 1;
uint32_t sqrttn = isqrt(scaledn);
uint32_t cutoff = isqrt(2*sqrttn)/cutoff_div;
uint32_t q0 = 1;
uint32_t p1 = sqrttn;
uint32_t q1 = scaledn-p1*p1;

if(q1 == 0){
 uint64_t factor = __gcd(x, (uint64_t)p1);
 return factor==x ? 1:factor;
}

uint32_t multiplier = 2*k;
uint32_t coarse_cutoff = cutoff * multiplier;
//cerr << "at: " << multiplier << "\n";

uint32_t i, j;
uint32_t p0 = 0;
uint32_t sqrtq = 0;

for(i=0;i<it_max;++i){
 uint32_t q, bits, tmp;

 tmp = sqrttn + p1 - q1;
 q = 1;
}

```

```

if (tmp >= q1)
 q += tmp / q1;

p0 = q * q1 - p1;
q0 = q0 + (p1 - p0) * q;

if (q1 < coarse_cutoff) {
 tmp = q1 / __gcd(q1, multiplier);

 if (tmp < cutoff) {
 saved.push_back((uint16_t)tmp);
 }
}

bits = 0;
tmp = q0;
while(!(tmp & 1)) {
 bits++;
 tmp >>= 1;
}

if (!(bits & 1) && ((tmp & 7) == 1)) {

 sqrtq = (uint32_t)isqrt(q0);

 if (sqrtq * sqrtq == q0) {
 for(j=0;j<saved.size();++j){
 if(saved[j] == sqrtq) break;
 }
 if(j == saved.size()) break;
 }
}

```

```

//else cerr << "skip " << i << "\n";;
}

tmp = sqrtn + p0 - q0;
q = 1;
if (tmp >= q0)
 q += tmp / q0;

p1 = q * q0 - p0;
q1 = q1 + (p0 - p1) * q;

if (q0 < coarse_cutoff) {
 tmp = q0 / __gcd(q0, multiplier);

 if (tmp < cutoff) {
 saved.push_back((uint16_t) tmp);
 }
}

if(sqrtq == 1) { return 1;}
if(i == it_max) { return 1;}

q0 = sqrtq;
p1 = p0 + sqrtq * ((sqrtn - p0) / sqrtq);
q1 = (scaledn - (uint64_t)p1 * (uint64_t)p1) / (uint64_t)q0;

for(j=0;j<it_max;++j) {
 uint32_t q, tmp;
}

```

---

```

tmp = sqrtn + p1 - q1;
q = 1;
if (tmp >= q1)
 q += tmp / q1;

p0 = q * q1 - p1;
q0 = q0 + (p1 - p0) * q;

if (p0 == p1) {
 q0 = q1;
 break;
}

tmp = sqrtn + p0 - q0;
q = 1;
if (tmp >= q0)
 q += tmp / q0;

p1 = q * q0 - p0;
q1 = q1 + (p0 - p1) * q;

if (p0 == p1)
 break;
}

if(j==it_max) {cerr << "RNG\n"; return 1;} // random fail

uint64_t factor = __gcd((uint64_t)q0, x);
if(factor == x) factor=1;

```

```

 return factor;
}
uint64_t squof(uint64_t const&x){
//for using only squof don't comment the following lines.
//for factorizing comment these, no problem.
// for(uint64_t i=2;i<=min((int64_t)5000,(int64_t)x/2-1);i++){
// uint64_t p=(uint64_t)((1.0*x)/(i*1.0)+eps);
// if(p*i==x) return i;
// }
 static array<uint32_t, 16> multipliers{1, 3, 5, 7, 11, 3*5, 3*7,
3*11, 5*7, 5*11, 7*11, 3*5*7, 3*5*11, 3*7*11, 5*7*11, 3*5*7*11};

 uint64_t cbrt_x = icbrt(x);
 if(cbrt_x*cbrt_x*cbrt_x == x) return cbrt_x;

//uint32_t iter_lim = isqrt(isqrt(x))+10;
uint32_t iter_lim = 300;
for(uint32_t iter_fact = 1;iter_fact<20000;iter_fact*=4){
 for(uint32_t const&k : multipliers){
 if(numeric_limits<uint64_t>::max()/k<=x) continue; //would
overflow
 uint32_t const it_max = iter_fact*iter_lim;
 uint64_t factor = squof_iter_better(x, k, it_max, 1);
 if(factor==1 || factor==x) continue;
 return factor;
 }
}
cerr << "failed to factor: " << x << "\n";
assert(0);

```

```

 assert(0);
 return 1;
}

template<typename T>
typename enable_if<is_integral<T>::value, vector<T>>::type
factorize_brute(T x){
 vector<T> ret;
 while(x%2 == 0){
 x/=2;
 ret.push_back(2);
 }
 for(uint32_t i=3;i*(T)i <= x;i+=2){
 while(x%i == 0){
 x/=i;
 ret.push_back(i);
 }
 }
 if(x>1) ret.push_back(x);
 return ret;
}

template<typename T>
typename enable_if<is_integral<T>::value, vector<T>>::type
factorize(T x){
//cerr << "factor: " << x << "\n";
 vector<T> ret;
 const uint32_t trial_limit = 5000;
 auto trial = [&](uint32_t const&i){

```

```

 while(x%i == 0){
 x/=i;
 ret.push_back(i);
 }
 };
 trial(2);
 trial(3);
for(uint32_t i=5, j=2;i<trial_limit && i*i <= x;i+=j, j=6-j{
 trial(i);
}
if(x>1){
 static stack<T> s;
 s.push(x);
 while(!s.empty()){
 x = s.top(); s.pop();
 if(!miller_rabin(x)){
 T factor = squfof(x);
 if(factor == 1 || factor == x){assert(0); return ret;}
 //cerr << x << " -> " << factor << "\n";
 s.push(factor);
 s.push(x/factor);
 } else {
 ret.push_back(x);
 }
 }
}
sort(ret.begin(), ret.end());
return ret;
}

```

```

 }

using ll = int64_t;

int main()
{
 fast;
 ll i,j,k,n,m;
 cin>>n;
 auto v=NT::factorize(n);
 for(auto x:v) cout<<x<<' ';
 return 0;
}

```

## 10. Prime Counting Function

```

namespace pcf{
 /// Prime-Counting Function
 /// initialize once by calling init()
 /// Legendre(n) and Lehmer(n) returns the number of primes less
 /// than or equal to n
 /// Lehmer(n) is faster

#define MAXN 1000010 /// initial sieve limit
#define MAX_PRIMES 1000010 /// max size of the prime array for
sieve
#define PHI_N 100000
#define PHI_K 100

unsigned int ar[(MAXN >> 6) + 5] = {0};

```

```

int len = 0; // total number of primes generated by sieve
int primes[MAX_PRIMES];
int counter[MAXN]; // counter[m] --> number of primes <= i
int phi_dp[PHI_N][PHI_K]; // precal of phi(n,k)

bitset<MAXN> isComp;
//bool isComp[MAXN];
void Sieve(int N){
 int i,j,sq = sqrt(N);
 isComp[1] = true;
 for(i=4;i<=N;i+=2) isComp[i] = true;
 for(i=3;i<=sq;i+=2){
 if(!isComp[i]){
 for(j=i*i;j<=N;j+=i+i) isComp[j] = 1;
 }
 }
 for (i = 1; i <= N; i++){
 if (!isComp[i]) primes[len++] = i;
 counter[i] = len;
 }
}

void init(){
 Sieve(MAXN - 1);

 /// precalculation of phi upto size (PHI_N,PHI_K)
 int k , n , res;
 for(n = 0; n < PHI_N; n++) phi_dp[n][0] = n;
 for (k = 1; k < PHI_K; k++){

```

```

 for (n = 0; n < PHI_N; n++){
 phi_dp[n][k] = phi_dp[n][k - 1] - phi_dp[n / primes[k - 1]][k - 1];
 }
 }
}

/// returns number of integers less or equal n which are
/// not divisible by any of the first k primes
/// recurrence --> phi(n , k) = phi(n , k-1) - phi(n / p_k , k-1)
long long phi(long long n, int k){
 if (n < PHI_N && k < PHI_K) return phi_dp[n][k];
 if (k == 1) return ((++n) >> 1);
 if (primes[k - 1] >= n) return 1;
 return phi(n, k - 1) - phi(n / primes[k - 1], k - 1);
}

long long Legendre(long long n){
 if (n < MAXN) return counter[n];

 int lim = sqrt(n) + 1;
 int k = upper_bound(primes, primes + len, lim) - primes;
 return phi(n, k) + (k - 1);
}

///complexity: n^(2/3).(log n)^(1/3)
long long Lehmer(long long n){
 if (n < MAXN) return counter[n];
}

```

```

long long w , res = 0;
int i, j, a, b, c, lim;
b = sqrt(n), c = Lehmer(cbrt(n)), a = Lehmer(sqrt(b)), b =
Lehmer(b);
res = phi(n, a) + (((b + a - 2) * (b - a + 1)) >> 1);

for (i = a; i < b; i++){
 w = n / primes[i];
 lim = Lehmer(sqrt(w)), res -= Lehmer(w);

 if (i <= c){
 for (j = i; j < lim; j++){
 res += j;
 res -= Lehmer(w / primes[j]);
 }
 }
}
return res;
}
}

```

## 11. Power Tower

```

///Given an array for each query (l,r) find
pow(a[l],pow(a[l+1],pow....a[r])%mod
#define MOD(a,b) ((a<b)?a:b+a%b)
ll qpow(ll n,ll k,ll mod){ll ans=1;while(k){if(k&1)
ans=MOD(ans*n,mod);n=MOD(n*n,mod);k>>=1;}return ans;}
ll a[N];

```

```

map<ll, ll>mp;
ll phi(ll n)
{
 if(mp.count(n)) return mp[n];
 ll i, ans=n, store=n;
 for(i=2; i*i <= n; i++){
 if(n%i==0){
 while(n%i==0) n/=i;
 ans=ans/i*(i-1);
 }
 }
 if(n>1) ans=ans/n*(n-1);
 return mp[store]=ans;
}
ll yo(ll l, ll r, ll mod)
{
 if(l==r) return MOD(a[l],mod);
 if(mod==1) return 1;
 else return qpow(a[l], yo(l+1, r, phi(mod)), mod);
}
int main()
{
 BeatMeScanf;
 ll i, j, k, n, m, mod, q, l, r;
 cin >> n >> mod;
 for(i=1; i <= n; i++) cin >> a[i];
 cin >> q;
 while(q--){
 cin >> l >> r;
 }
}

```

```

cout<<yo(l,r,mod)%mod<<n;
}
return 0;
}

```

## 12. Mobius Function

```

int mob[N];
void mobius()
{
 for(int i=1;i<N;i++) mob[i]=3;
 mob[1]=1;
 for(int i=2;i<N;i++){
 if(mob[i]==3){
 mob[i]=-1;
 for(int j=2*i;j<N;j+=i) mob[j]=(mob[j]==3?-1:mob[j]*(-1));
 if(i<N/i) for(int j=i*i;j<N;j+=i*i) mob[j]=0;
 }
 }
}
int main()
{
 fast;
 ll i,j,k,n,m;
 mobius();
 return 0;
}

```

## 13. Modular Inverse

```
ll inv[N];
```

```

int main()
{
 BeatMeScanf;
 ll i,j,k,n,m;
 inv[1] = 1;
 for (i = 2; i < N; i++) {
 inv[i] = (-(1LL*mod/i) * inv[mod%i]) % mod;
 inv[i] = inv[i] + mod;
 }
 for(i=1;i<10;i++) cout<<inv[i]<<' ';
 return 0;
}

```

## 14. Factoradic Number

```

vi decimal_to_factoradic(int n)
{
 vi v;
 int i=1;
 while(n){
 v.eb(n%i);
 n/=i;
 i++;
 }
 rev(v);
 return v;
}
int factoradic_to_decimal(vi &v)
{
 int n=v.size();

```

```

int ans=0;
for(int i=0,mul=n;i<n;i++,mul--) ans=(ans*mul%mod+v[i])%mod;
return ans;
}
vi permutation(int n,vi &v)
{
 o_set<int>se;
 int sz=v.size();
 vi p;
 for(int i=0;i<n-sz;i++) p.eb(i);
 for(int i=n-sz;i<n;i++) se.insert(i);
 for(int i=0;i<sz;i++){
 int nw=*se.fbo(v[i]);
 p.eb(nw);
 se.erase(nw);
 }
 return p;
}
///returns k-th lexicographically smallest permutation of size n
///0-th permutation is the unit permutation i.e. 0,1,2,...n-1
vi kth_perm(int n,int k)
{
 ///need to return something when k>=n!
 vi v=decimal_to_factoradic(k);
 return permutation(n,v);
}
vi factoradic_order(vi &p)
{
 o_set<int>se;

```

```

 int n=p.size();
 for(int i=0;i<n;i++) se.insert(p[i]);
 vi fac;
 for(int i=0;i<n;i++){
 int x=se.ook(p[i]);
 fac.eb(x);
 se.erase(p[i]);
 }
 return fac;
}
///?-th lexicographically smallest permutation of size n
int order(vi &p)
{
 vi fac=factoradic_order(p);
 return factoradic_to_decimal(fac);
}

///Given two permutations of size n, find Perm((ord(P)+ord(Q))mod n!)
///where Perm(k) is k-th lexicographically smallest permutation
///and ord(P) is the number k of the permutation
int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin>>n;
 vi p(n);

```

```

for(i=0;i<n;i++) cin>>p[i];
vi q(n);
for(i=0;i<n;i++) cin>>q[i];
vi ordp=factoradic_order(p);
vi ordq=factoradic_order(q);
vi sum=ordp;
int carry=0;
for(i=n-1;i>=0;i--){
 sum[i]+=ordq[i]+carry;
 carry=sum[i]/(n-i);
 sum[i]%= (n-i);
}
vi perm=permutation(n,sum);
for(i=0;i<n;i++) cout<<perm[i]<<' ';
return 0;
}

```

## 15. Linear Sieve

```

int spf[N];//smallest prime factor
vi prime;
void linear_sieve()
{
 for(int i=2;i<N;i++){
 if(spf[i]==0) spf[i]=i,prime.eb(i);
 int sz=prime.size();
 for(int j=0;j<sz&&i<=(N-1)/prime[j]&&prime[j]<=spf[i];j++)
 spf[i]*prime[j]=prime[j];
 }
}

```

## 16. Discrete Logarithm

```

struct DiscreteLogarithm
{
 int powmod(int a, int b, int m)
 {
 int res = 1;
 while (b > 0) {
 if (b & 1) {
 res = (1LL * res * a) % m;
 }
 a = (1LL * a * a) % m;
 b >>= 1;
 }
 return res;
 }

 //returns the primitive root modulo p
 //g is a primitive root modulo p if and only if for any integer a such
 //that
 //gcd(a,p)=1, there exists an integer k such that: g^k = a(mod p).
 //this code assumes p is prime
 int PrimitiveRoot(int p)
 {
 vector<int> fact;
 int phi = p - 1, n = phi;//Beware!!! if p is not prime calculate the
 value of 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;
}

/// baby step - giant step
///find any integer x such that a^x = b (mod m)
///where a and m are co-prime
int DiscreteLog(int a, int b, int m)
{
 int n = (int) sqrt (m + .0) + 1;
 int an = 1;
 for (int i = 0; i < n; ++i) an = (1LL*an * a) % m;
 umap<int, int> vals;
 for (int p = 1, cur = an; p <= n; ++p) {
 if (!vals.count(cur)) vals[cur] = p;
 cur = (1LL*cur * an) % m;
 }
 for (int q = 0, cur = b; q <= n; ++q) {
 if (vals.count(cur)) {

```

```

 int ans = vals[cur] * n - q;
 if(powmod(a,ans,m)==b%m) return ans;//check the
question if ans need to be less than m?
 }
 cur = (1LL*cur * a) % m;
 }
 return -1;
}

///returns any or all numbers x such that x^k = a (mod n)
int DiscreteRoot(int k, int a, int n) {
 if (a == 0) return 1;
 int g = PrimitiveRoot(n);
 int phi=n-1;//Beware!!! if n is not a prime calculate the value of
phi
 //run baby step-giant step
 int sq = (int) sqrt (n + .0) + 1;
 vector< pair<int,int> > dec (sq);
 for (int i=1; i<=sq; ++i) dec[i-1] = make_pair (powmod (g, 1LL*i *
sq %phi * k % phi, n), i);
 sort (dec.begin(), dec.end());
 int any_ans = -1;
 for (int i=0; i<sq; ++i) {
 int my = powmod (g, 1LL* i * k % phi, n) * 1LL * a % n;
 auto it =lower_bound (dec.begin(), dec.end(), make_pair (my,
0));
 if (it != dec.end() && it->first == my) {
 any_ans = it->second * sq - i;
 break;
 }
 }
 return any_ans;
}

```

```

 }
}

if(any_ans== -1) return -1;///no solution

///for any answer
int delta = (n-1) / __gcd (k, n-1);
for (int cur=any_ans%delta; cur<n-1; cur+=delta) return
(powmod (g, cur, n));

///for all possible answers
//int delta = (n-1) / __gcd(k, n-1);
//vector<int> ans;
//for (int cur = any_ans % delta; cur < n-1; cur += delta)
ans.push_back(powmod(g, cur, n));
//sort(ans.begin(), ans.end());
//return ans;
}
}d;

int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cout<<d.PrimitiveRoot(786433);
 return 0;
}

```

## 17. Chinese Remainder Theorem

```
class garner {
```

```

private:
vector<ll> prime,pprep,mult,rem;
public:
ll fpow(ll p, ll q, ll mod) {
 ll ret = 1;
 for (; q; q >>= 1) {
 if (q & 1) {
 ret = ret * p % mod;
 }
 p = p * p % mod;
 }
 return ret;
}
void prep(vector<ll> prm, vector<ll> rm) {
 prime.clear();
 for (auto v : prm) { prime.push_back(v); }
 ll prv = 1;
 for (int i = 0; i < prime.size(); i++) {
 mult.push_back(prv);
 pprep.push_back(fpow(prv, prime[i]-2,
prime[i]));
 prv = prv*prime[i];
 }
 for (auto v : rm) { rem.push_back(v); }
}
ll calc(ll bigmode) {
 ll ans = 0,prv=0;
 for (int i = 0; i < prime.size(); i++) {
 ll xx=((rem[i]+prv)*pprep[i])%prime[i];
 ans = ans + xx*mult[i];
 ans = ans % bigmode;
 }
 return ans;
}

```

```

 ans = (ans + xx*mult[i]);
 while (ans > bigmode) { ans = ans - bigmode; };
 while (ans < 0) { ans = ans + bigmode; };
 prv = prv - (xx*mult[i]);
 }
 return ans;
}
};

ll n;
ll get(ll mod)
{
 ll ans=1;
 vll v(mod+10,1);
 for(ll i=1;i<=mod;i++) v[i]=v[i-1]*i%mod;
 for(ll i=1;i<=min(mod,n);i++) ans=ans*v[i]%mod;
 return ans;
}
int main()
{
 BeatMeScanf;
 ll i,j,k,m1=186583,m2=587117;
 cin>>n;
 vll prime,rem;
 prime.eb(m1);rem.eb(get(m1));
 prime.eb(m2);rem.eb(get(m2));
 garner x;
 x.prep(prime,rem);
 cout<<x.calc(1LL*587117*186583)<<nl;
 return 0;
}

```

```
}
```

## 18. Miller Robin Primality Test

```

ll mulmod(ll a,ll b, ll mod)
{
 ll ans=0;
 a=a%mod;
 while(b){
 if(b&1) ans=(ans+a)%mod;
 a=(a*2)%mod;
 b>>=1;
 }
 return ans;
}

ll qpow(ll n,ll k,ll mod)
{
 ll ans=1;
 n=n%mod;
 while(k){
 if(k&1) ans=mulmod(ans,n,mod);
 n=mulmod(n,n,mod);
 k>>=1;
 }
 return ans;
}

random_device rd;
mt19937 rnd(rd());
bool test(ll n,ll d)

```

```

{
 ll i,j,k,a,x;
 a=2LL+rnd()%n-4;
 x=qpow(a,d,n);
 if(x==1 || x==n-1) return 1;
 while(d!=n-1){
 x=mulmod(x,x,n);
 d*=2;
 if(x==1) return 0;
 if(x==n-1) return 1;
 }
 return 0;
}
bool millerrobin(ll n)
{
 if(n<=1 || n==4) return 0;
 if(n<=3) return 1;
 ll i,d=n-1;
 while(d%2==0) d/=2;
 for(i=1;i<=20;i++) if(test(n,d)==0) return 0;
 return 1;
}
int main()
{
 fast;
 ll i,j,k,n,m;
 cin>>n;
 cout<<millerrobin(n);
 return 0;
}

```

{

## 19. Linear Diophantine Equation

A Linear Diophantine Equation (in two variables) is an equation of the general form:

$$ax + by = c$$

where  $a, b, c$  are given integers, and  $x, y$  are unknown integers.

In this article, we consider several classical problems on these equations:

- finding one solution
- finding all solutions
- finding the number of solutions and the solutions themselves in a given interval
- finding a solution with minimum value of  $x + y$

### The degenerate case

A degenerate case that need to be taken care of is when  $a = b = 0$ . It is easy to see that we either have no solutions or infinitely many solutions, depending on whether  $c = 0$  or not. In the rest of this article, we will ignore this case.

### The degenerate case

A degenerate case that need to be taken care of is when  $a = b = 0$ . It is easy to see that we either have no solutions or infinitely many solutions, depending on whether  $c = 0$  or not. In the rest of this article, we will ignore this case.

### Finding a solution

To find one solution of the Diophantine equation with 2 unknowns, you can use the extended Euclidean algorithm. First, assume that  $a$  and  $b$  are non-negative. When we apply extended Euclidean algorithm for  $a$  and  $b$ , we can find their greatest common divisor  $g$  and 2 numbers  $x_g$  and  $y_g$  such that:

$$ax_g + by_g = g$$

If  $c$  is divisible by  $g = \gcd(a, b)$ , then the given Diophantine equation has a solution, otherwise it does not have any solution. The proof is straight-forward: a linear combination of two numbers is divisible by their common divisor.

Now supposed that  $c$  is divisible by  $g$ , then we have:

$$a \cdot x_g \cdot \frac{c}{g} + b \cdot y_g \cdot \frac{c}{g} = c$$

Therefore one of the solutions of the Diophantine equation is:

$$x_0 = x_g \cdot \frac{c}{g},$$

$$y_0 = y_g \cdot \frac{c}{g}.$$

The above idea still works when  $a$  or  $b$  or both of them are negative. We only need to change the sign of  $x_0$  and  $y_0$  when necessary.

Finally, we can implement this idea as follows (note that this code does not consider the case  $a = b = 0$ ):

```

int gcd(int a, int b, int &x, int &y) {
 if (a == 0) {
 x = 0; y = 1;
 return b;
 }
 int x1, y1;
 int d = gcd(b%a, a, x1, y1);
 x = y1 - (b / a) * x1;
 y = x1;
 return d;
}

bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
 g = gcd(abs(a), abs(b), x0, y0);
 if (c % g) {
 return false;
 }

 x0 *= c / g;
 y0 *= c / g;
 if (a < 0) x0 = -x0;
 if (b < 0) y0 = -y0;
 return true;
}

```

## Getting all solutions

From one solution  $(x_0, y_0)$ , we can obtain all the solutions of the given equation.

Let  $g = \gcd(a, b)$  and let  $x_0, y_0$  be integers which satisfy the following:

$$a \cdot x_0 + b \cdot y_0 = c$$

Now, we should see that adding  $b/g$  to  $x_0$ , and, at the same time subtracting  $a/g$  from  $y_0$  will not break the equality:

$$a \cdot \left(x_0 + \frac{b}{g}\right) + b \cdot \left(y_0 - \frac{a}{g}\right) = a \cdot x_0 + b \cdot y_0 + a \cdot \frac{b}{g} - b \cdot \frac{a}{g} = c$$

Obviously, this process can be repeated again, so all the numbers of the form:

$$x = x_0 + k \cdot \frac{b}{g}$$

$$y = y_0 - k \cdot \frac{a}{g}$$

are solutions of the given Diophantine equation.

Moreover, this is the set of all possible solutions of the given Diophantine equation.

## Finding the number of solutions and the solutions in a given interval

From previous section, it should be clear that if we don't impose any restrictions on the solutions, there would be infinite number of them. So in this section, we add some restrictions on the interval of  $x$  and  $y$ , and we will try to count and enumerate all the solutions.

Let there be two intervals:  $[min_x; max_x]$  and  $[min_y; max_y]$  and let's say we only want to find the solutions in these two intervals.

Note that if  $a$  or  $b$  is 0, then the problem only has one solution. We don't consider this case here.

First, we can find a solution which have minimum value of  $x$ , such that  $x \geq min_x$ . To do this, we first find any solution of the Diophantine equation. Then, we shift this solution to get  $x \geq min_x$  (using what we know about the set of all solutions in previous section). This can be done in  $O(1)$ . Denote this minimum value of  $x$  by  $l_{x1}$ .

Similarly, we can find the maximum value of  $x$  which satisfy  $x \leq max_x$ . Denote this maximum value of  $x$  by  $r_{x1}$ .

Similarly, we can find the minimum value of  $y$  ( $y \geq min_y$ ) and maximum values of  $y$  ( $y \leq max_y$ ). Denote the corresponding values of  $y$  by  $l_{x2}$  and  $r_{x2}$ .

The final solution is all solutions with  $x$  in intersection of  $[l_{x1}, r_{x1}]$  and  $[l_{x2}, r_{x2}]$ . Let denote this intersection by  $[l_x, r_x]$ .

Following is the code implementing this idea. Notice that we divide  $a$  and  $b$  at the beginning by  $g$ . Since the equation  $ax + by = c$  is equivalent to the equation  $\frac{a}{g}x + \frac{b}{g}y = \frac{c}{g}$ , we can use this one instead and have  $\gcd(\frac{a}{g}, \frac{b}{g}) = 1$ , which simplifies the formulas.

```

void shift_solution(int &x, int &y, int a, int b, int cnt) {
 x += cnt * b;
 y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
 int x, y, g;
 if (!find_any_solution(a, b, c, x, y, g))
 return 0;
 a /= g;
 b /= g;

 int sign_a = a > 0 ? +1 : -1;
 int sign_b = b > 0 ? +1 : -1;

 shift_solution(x, y, a, b, (minx - x) / b);
 if (x < minx)
 shift_solution(x, y, a, b, sign_b);
 if (x > maxx)
 return 0;
 int lx1 = x;

 shift_solution(x, y, a, b, (maxx - x) / b);
 if (x > maxx)
 shift_solution(x, y, a, b, -sign_b);
 int rx1 = x;
}

```

```

shift_solution(x, y, a, b, -(miny - y) / a);
if (y < miny)
 shift_solution(x, y, a, b, -sign_a);
if (y > maxy)
 return 0;
int lx2 = x;

shift_solution(x, y, a, b, -(maxy - y) / a);
if (y > maxy)
 shift_solution(x, y, a, b, sign_a);
int rx2 = x;

if (lx2 > rx2)
 swap(lx2, rx2);
int lx = max(lx1, lx2);
int rx = min(rx1, rx2);

if (lx > rx)
 return 0;
return (rx - lx) / abs(b) + 1;
}

```

Once we have  $l_x$  and  $r_x$ , it is also simple to enumerate through all the solutions. Just need to iterate through  $x = l_x + k \cdot \frac{b}{g}$  for all  $k \geq 0$  until  $x = r_x$ , and find the corresponding  $y$  values using the equation  $ax + by = c$ .

### Find the solution with minimum value of $x + y$

Here,  $x$  and  $y$  also need to be given some restriction, otherwise, the answer may become negative infinity.

The idea is similar to previous section: We find any solution of the Diophantine equation, and then shift the solution to satisfy some conditions.

Finally, use the knowledge of the set of all solutions to find the minimum:

$$x' = x + k \cdot \frac{b}{g},$$

$$y' = y - k \cdot \frac{a}{g}.$$

Note that  $x + y$  change as follows:

$$x' + y' = x + y + k \cdot \left( \frac{b}{g} - \frac{a}{g} \right) = x + y + k \cdot \frac{b-a}{g}$$

If  $a < b$ , we need to select smallest possible value of  $k$ . If  $a > b$ , we need to select the largest possible value of  $k$ . If  $a = b$ , all solution will have the same sum  $x + y$ .

## 20. Linear Diophantine Less Than or Equal

```

/// number of integer points the triangle
/// ax + by <=c && x, y > 0; where a, b, c > 0
int64_t count_triangle(int64_t a, int64_t b, int64_t c){

```

```

if(b>a) swap(a, b);
int64_t m = c/a;
if(a==b) return m*(m-1)/2;
int64_t k= (a-1)/b, h = (c-a*m)/b;
return m*(m-1)/2*k + m*h + count_triangle(b, a-b*k, c-
b*(k*m+h));
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;

cout<<count_triangle(2323424,342423,10000000000000)<<nl;
 return 0;
}

```

## 21. Sum of the K-th Powers

/\*\*

Find  $1^k + 2^k + \dots + n^k \% \text{mod}$

Complexity:  $k^2$

$$x^k = \sum_{i=1}^k \text{Stirling2}(k, i) * i! * \text{ncr}(x, i)$$

$$\sum_{x=0}^n x^k$$

$$= \sum_{i=0}^k \text{Stirling2}(k, i) * i! * \sum_{x=0}^n$$

$\text{ncr}(x, i)$

$$= \sum_{i=0}^k \text{Stirling2}(k, i) * i! * \text{ncr}(n+1, i+1)$$

```

 = sum (i = 0 to k) Stirling2(k, i) * i! * (n + 1)! / (i + 1)! /
(n - i)!

 = sum (i = 0 to k) Stirling2(k, i) * (n - i + 1) * (n - i + 2) *
... (n + 1) / (i + 1)
*/
#define MAX 2505
int S[MAX][MAX], inv[MAX];

///generate stirling numbers of the 2nd kind
void stirling(){
 int i, j;
 for (i = 0; i < MAX; i++) inv[i] = qpow(i, mod-2);

 S[0][0] = 1;
 for (i = 1; i < MAX; i++){
 S[i][0] = 0;
 for (j = 1; j <= i; j++){
 S[i][j] = (((long long)S[i - 1][j] * j) + S[i - 1][j - 1]) % mod;
 }
 }
}

///this part is O(k)
int faulhaber(long long n, int k){
 n %= mod;
 if (!k) return n;
 long long res = 0, p = 1;
 for (int j = 0; j <= k; j++){

```

```

 p = (p * (n + 1 - j)) % mod;
 res = (res + (((S[k][j] * p) % mod) * inv[j + 1])) % mod;
 }
 return (res % mod);
}

int main()
{
 BeatMeScanf;
 ll i,j,k,n,m;
 stirling();
 cout<<faulhaber(1001212, 1000)<<n;
 return 0;
}

```

## 22. Rational Approximation

```
/**
Given n and a real number x >= 0, finds the closest rational
approximation p/q with p, q <= n.
It will obey that |p/q - x| is minimum for p, q <= n
Time: O(log n)
**/
```

```
typedef double d; // for n ~ 1e7; long double for n ~ 1e9
pair<ll, ll> approximate(d x, ll n) {
 ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
 for (;;) {
 ll lim = min(P ? (n-LP) / P : inf, Q ? (n-LQ) / Q : inf),
 a = (ll)floor(y), b = min(a, lim),
```

```

NP = b*P + LP, NQ = b*Q + LQ;
if (a > b) {
 // If b > a/2, we have a semi-convergent that
gives us a
 // better approximation; if b = a/2, we *may*
have one.
 // Return {P, Q} here for a more canonical
approximation.
 return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P /
(d)Q)) ?
 make_pair(NP, NQ) : make_pair(P, Q);
}
if (abs(y = 1/(y - (d)a)) > 3*n) {
 return {NP, NQ};
}
LP = P; P = NP;
LQ = Q; Q = NQ;
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,m;
 double q=3232.456;
 ll n=102323232230;
 ll ans=approximate(q,n);
 cout<<ans.F<<' '<<ans.S;
 return 0;
}

```

```
}
```

## 23. Modular Square Root

```

//find sqrt(a)%p, i.e. find any x such that x^2=a (mod p)
//p is prime
//complexity: O(log^2 p) worst case, O(log p) average
//Tonelli-Shanks algorithm
ll sqrt(ll a, ll p) {
 a %= p; if (a < 0) a += p;
 if (a == 0) return 0;
 assert(qpow(a, (p-1)/2, p) == 1); //if a solution exists or not
 if (p % 4 == 3) return qpow(a, (p+1)/4, p);
 // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
 ll s = p - 1, n = 2;
 int r = 0, m;
 while (s % 2 == 0)
 ++r, s /= 2;
 // find a non-square mod p
 while (qpow(n, (p - 1) / 2, p) != p - 1) ++n;
 ll x = qpow(a, (s + 1) / 2, p);
 ll b = qpow(a, s, p), g = qpow(n, s, p);
 for (;;) r = m {
 ll t = b;
 for (m = 0; m < r && t != 1; ++m)
 t = t * t % p;
 if (m == 0) return x;
 ll gs = qpow(g, 1LL << (r - m - 1), p);
 g = gs * gs % p;
 x = x * gs % p;
 }
}
```

```

 b = b * g % p;
}
}

```

```

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cout<<sqrt(3,10007);
 return 0;
}

```

## 24. Sum of Arithmetic Progression Modular and Divided

```

/**
Sums of mod'ed arithmetic progressions.
modsum(to, c, k, m) = sum i=0,1,2...(to-1) {(k*i+c)% m}.
divsum(to, c, k, m) = sum i=0,1,2...(to-1) {(k*i+c)/ m}.
Time: log(m), with a large constant.
*/

```

```
ull sumsq(ull to) { return to / 2 * ((to-1) + 1); }
```

```

ull divsum(ull to, ull c, ull k, ull m) {
 ull res = k / m * sumsq(to) + c / m * to;
 k %= m; c %= m;
 if (!k) return res;
 ull to2 = (to * k + c) / m;
}

```

```

 return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}

ll modsum(ull to, ll c, ll k, ll m) {
 c = ((c % m) + m) % m;
 k = ((k % m) + m) % m;
 return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}

int32_t main()
{
 BeatMeScanf;
 int i,j;
 ll to=10,c=23,k=12,m=7;
 ll sum=0;
 for(i=0;i<to;i++) sum+=(k*i+c)%m;
 cout<<sum<<' '<<modsum(to,c,k,m)<<nl;
 sum=0;
 for(i=0;i<to;i++) sum+=(k*i+c)/m;
 cout<<sum<<' '<<divsum(to,c,k,m)<<nl;
 return 0;
}

```

## 25. Carmichael Lambda

```

/**
Carmichael Lambda (Universal Totient Function)
lambda(n) is a smallest number that satisfies
a^lambda(n) = 1 (mod n) for all a coprime with n.

```

Complexity:

```
carmichael_lambda(n): O(sqrt(n)) by trial division.
carmichael_lambda(lo,hi): O((hi-lo) loglog(hi)) by prime sieve.
**/
/**
```

By the fundamental theorem of arithmetic any  $n > 1$  can be written in a unique way as

$$n = p_1^{r_1} p_2^{r_2} \dots p_k^{r_k}$$

where  $p_1 < p_2 < \dots < p_k$  are primes and  $r_1, \dots, r_k$  are positive integers. It is then the case that  $\lambda(n)$  is the least common multiple of the  $\lambda$  of each of its prime power factors:

$$\lambda(n) = \text{lcm}(\lambda(p_1^{r_1}), \lambda(p_2^{r_2}), \dots, \lambda(p_k^{r_k})).$$

Carmichael's theorem explains how to compute  $\lambda$  of a prime power: for a power of an odd prime and for 2 and 4,  $\lambda(n)$  is equal to the Euler totient  $\phi(n)$ ; for powers of 2 greater than 4 it is equal to half of the Euler totient.

$$\lambda(p^k) = \begin{cases} \phi(p^k), p \% 2 = 1 \text{ or } p^k = 2 \text{ or } p^k = 4 \\ \frac{\phi(p^k)}{2}, p = 2 \text{ and } p^k \geq 4 \end{cases}$$

$\lambda(n)$  divides  $\phi(n)$

$$a|b \Rightarrow \lambda(a)|\lambda(b)$$

$$\lambda(\text{lcm}(a, b)) = \text{lcm}(\lambda(a), \lambda(b))$$

If  $n$  has maximum prime exponent  $r_{\max}$  under prime factorization, then for all  $a$  (including those not co-prime to  $n$ ) and all  $r \geq r_{\max}$

$$a^r \equiv a^{r+\lambda(n)} \pmod{n}$$

In particular, for squarefree  $n$  ( $r_{\max} = 1$ ), for all  $a$

$$a \equiv a^{\lambda(n)+1} \pmod{n}$$

\*\*/

```
|| gcd(|| a, || b) {
 for (; a; swap(b %= a, a));
```

```
 return b;
}
|| lcm(|| a, || b) {
 return a * (b / gcd(a, b));
}

|| carmichael_lambda(|| n) {
 || lambda = 1;
 if (n % 8 == 0) n /= 2;
 for (|| d = 2; d*d <= n; ++d) {
 if (n % d == 0) {
 n /= d;
 || y = d - 1;
 while (n % d == 0) {
 n /= d;
 y *= d;
 }
 lambda = lcm(lambda, y);
 }
 }
 if (n > 1) lambda = lcm(lambda, n-1);
 return lambda;
}
```

```
vector<||> primes(|| lo, || hi) { /// primes in [lo, hi)
const || M = 1 << 14, SQR = 1 << 16;
vector<bool> composite(M), small_composite(SQR);

vector<pair<||,||>> sieve;
```

```

for (ll i = 3; i < SQR; i+=2) {
 if (!small_composite[i]) {
 ll k = i*i + 2*i*max(0.0, ceil((lo - i*i)/(2.0*i)));
 sieve.push_back({2*i, k});
 for (ll j = i*i; j < SQR; j += 2*i)
 small_composite[j] = 1;
 }
}
vector<ll> ps;
if (lo <= 2) { ps.push_back(2); lo = 3; }
for (ll k = lo | 1, low = lo; low < hi; low += M) {
 ll high = min(low + M, hi);
 fill(all(composite), 0);
 for (auto &z: sieve)
 for (; z.S < high; z.S += z.F)
 composite[z.S - low] = 1;
 for (; k < high; k+=2)
 if (!composite[k - low]) ps.push_back(k);
}
return ps;
}
vector<ll> primes(ll n) { /// primes in [0,n)
 return primes(0,n);
}
vector<ll> carmichael_lambda(ll lo, ll hi) { /// lambda(n) for all n in [lo, hi)
 vector<ll> ps = primes(sqrt(hi)+1);
 vector<ll> res(hi-lo), lambda(hi-lo, 1);
 iota(all(res), lo);
}

```

```

for (ll k = ((lo+7)/8)*8; k < hi; k += 8) res[k-lo] /= 2;
for (ll p: ps) {
 for (ll k = ((lo+(p-1))/p)*p; k < hi; k += p) {
 if (res[k-lo] < p) continue;
 ll t = p - 1;
 res[k-lo] /= p;
 while (res[k-lo] > 1 && res[k-lo] % p == 0) {
 t *= p;
 res[k-lo] /= p;
 }
 lambda[k-lo] = lcm(lambda[k-lo], t);
 }
}
for (ll k = lo; k < hi; ++k) {
 if (res[k-lo] > 1)
 lambda[k-lo] = lcm(lambda[k-lo], res[k-lo]-1);
}
return lambda; // lambda[k-lo] = lambda(k)
}

int main() {
 int n = 100;
 auto lmbd = carmichael_lambda(10000000000, 10000000010);
 printv(lmbd);
 return 0;
}

```

# DATA STRUCTURE

## 26. Policy Based Data Structure

### Ordered Set and Ordered Map

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

template <typename T> using o_set = tree<T, null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;

template <typename T,typename R> using o_map = tree<T, R,
less<T>, rb_tree_tag, tree_order_statistics_node_update>;

int main()
{
 fast;
 ll i,j,k,n,m;
 o_set<ll>se;
 se.insert(1);
 se.insert(2);
 cout<<*se.find_by_order(0)<<nl;//k th element
 cout<<se.order_of_key(2)<<nl;//number of elements less than k
 o_map<ll,ll>mp;
 mp.insert({1,10});
 mp.insert({2,20});
```

```
cout<<mp.find_by_order(0)->second<<nl;//k th element
cout<<mp.order_of_key(2)<<nl;//number of first elements less
than k
return 0;
}
```

## 27. Monotonous Queue

```
///Complexity: O(n)
///Given an array and an integer k,
///find the maximum for each and every contiguous subarray of size
k.
///monotonous queue for minimum is similar
struct monotonous_queue_max
{
 //pair.first - the actual value,
 //pair.second- how many elements were deleted between it and
the one before it.
 deque<pair<int, int>> q;
 void push(int val)
 {
 int cnt = 0;
 while(!q.empty() && q.back().F < val)
 {
 cnt += q.back().S + 1;
 q.pop_back();
 }
 q.eb(val, cnt);
 };
 int top()
 {
```

```

 if(q.empty()) return -1e9;
 return q.front().F;
}
void pop ()
{
 if(q.empty()) return;
 if (q.front().S > 0)
 {
 q.front().S--;
 return;
 }
 q.pop_front();
}
};

monotonous_queue_max q;
int a[N];
int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin>>n;
 for(i=1;i<=n;i++) cin>>a[i];
 cin>>k;
 for(i=1;i<k;i++) q.push(a[i]);
 int ans=0;
 for(i=k;i<=n;i++){
 q.push(a[i]);
 cout<<q.top()<<' ';
 q.pop();
 }
}

```

```

 return 0;
}
```

## 28. Binary Indexed Tree

### BIT Standard

```

template <class T>
struct BIT
{
 ///1-indexed
 int sz; //max size of array+1
 vector<T> t;

 void init(int n)
 {
 sz = n;
 t.assign(sz,0);
 }

 T query(int idx)
 {
 T ans = 0;
 for(; idx >= 1; idx -= (idx & -idx)) ans += t[idx];
 return ans;
 }

 void upd(int idx, T val)
 {
 if(idx <= 0) return;
 for(; idx <sz; idx += (idx & -idx)) t[idx] += val;
 }
}
```

```

T query(int l, int r) { return query(r) - query(l - 1); }
};

int main()
{
 BeatMeScanf;
 ll i,j,k,n,m;
 BIT<ll>t;
 t.init(N);
 t.upd(7,5);
 t.upd(3,10);
 cout<<t.query(1,10)<<nl;
 return 0;
}

```

### BIT with range update and range query

```

struct BIT {
 ll dataMul[M],dataAdd[M];
 void init()
 {
 mem(dataMul,0);
 mem(dataAdd,0);
 }
 void internalUpdate(int at, long long mul, long long add) {
 while (at < M) {
 dataMul[at] += mul;
 dataAdd[at] += add;
 at |= (at + 1);
 }
 }
}

```

```

void update(int left, int right, long long by) {
 internalUpdate(left, by, -by * (left - 1));
 internalUpdate(right, -by, by * right);
}

long long query(int at) {
 long long mul = 0;
 long long add = 0;
 int start = at;
 while (at >= 0) {
 mul += dataMul[at];
 add += dataAdd[at];
 at = (at & (at + 1)) - 1;
 }
 return (mul * start + add);
}

long long query(int l,int r)
{
 return query(r)-query(l-1);
}
}t;

```

### BIT 2D

```

template<typename T>
struct BIT
{
 ///1-indexed
 int szr,szc;///max size of array+1
 vector<vector<T>>t;
 void init(int n,int m)
 {

```

```

szr=n,szc=m;
t.assign(szr,vector<T>(szc,0));
}
void upd(int r,int c,T val) //add val to a[i][j]
{
 for(int i=r;i<szr;i+=i&-i) for(int j=c;j<szc;j+=j&-j) t[i][j]+=val;
}
T query(int r,int c)
{
 T sum=0;
 for(int i=r;i>0;i-=i&-i) for(int j=c;j>0;j-=j&-j) sum+=t[i][j];
 return sum;
}
T query(int x1,int y1,int x2,int y2) //returns sum of the
corresponding rectangle
{
 return query(x2,y2)-query(x2,y1-1)-query(x1-1,y2)+query(x1-
1,y1-1);
}
//in case of range update single query
//for range update use upd(x1,y1,val),upd(x1,y2+1,-
val),upd(x2+1,y1,-val),upd(x2+1,y2+1,val)
int main()
{
 fast;
 ll i,j,k,n,m,q,x1,y1,x2,y2,typ;
 cin>>n>>m;
 BIT<ll>t;
 t.init(N,N);
}

```

```

for(i=1;i<=n;i++) for(j=1;j<=m;j++) cin>>k,t.upd(i,j,k);
cin>>q;
while(q--){
 cin>>typ;
 if(typ==1){
 cin>>i>>j>>k;
 //add k to a[i][j]
 t.upd(i,j,k);
 }
 else{
 cin>>x1>>y1>>x2>>y2;
 //make sure that (x1,y1) is top-left and (x2,y2) is bottom-
right
 cout<<t.query(x1,y1,x2,y2)<<nl;
 }
}
return 0;
}

BIT 2D with range update and range query
//works for range xor update and range xor sum too
|| multree[N][N][2],addtree[N][N][2];
|| yo(|| x)
{
 //for range sum
 return x;
 //for range xor
 //return (x%2);
}
|| query2(|| tree[N][N][2],|| x,|| y)

```

```

{
 ll mul=0,add=0;
 for(ll i=y;i>0;i-=i&-i){
 mul+=tree[x][i][0];
 add+=tree[x][i][1];
 //mul^=tree[x][i][0];
 //add^=tree[x][i][1];
 }
 return (mul*yo(y))+add;
 //return (mul*yo(y))^add;
}
ll query1(ll x,ll y)
{
 ll mul=0,add=0;
 for(ll i=x;i>0;i-=i&-i){
 mul+=query2(multree,i,y);
 add+=query2(addtree,i,y);
 //mul^=query2(multree,i,y);
 //add^=query2(addtree,i,y)
 }
 return (mul*yo(x))+add;
 //return (mul*yo(x))^add;
}
ll query(ll x1,ll y1,ll x2,ll y2)
{
 return (query1(x2,y2)-query1(x1-1,y2)-query1(x2,y1-1)+query1(x1-1,y1-1));
 //return (query1(x2,y2)^query1(x1-1,y2)^query1(x2,y1-1)^query1(x1-1,y1-1));
}

```

```

void upd2(ll tree[N][N][2],ll x,ll y,ll mul,ll add)
{
 for(ll i=x;i<N;i+=i&-i){
 for(ll j=y;j<N;j+=j&-j){
 tree[i][j][0]+=mul;
 tree[i][j][1]+=add;
 //tree[i][j][0]^=mul;
 //tree[i][j][1]^=add;
 }
 }
}
void upd1(ll x,ll y1,ll y2,ll mul,ll add)
{
 //for range sum
 upd2(multree,x,y1,mul,-mul*yo(y1-1));
 upd2(multree,x,y2,-mul,mul*yo(y2));
 upd2(addtree,x,y1,add,-add*yo(y1-1));
 upd2(addtree,x,y2,-add,add*yo(y2));
 //for range xor
 //upd2(multree,x,y1,mul,mul*yo(y1-1));
 //upd2(multree,x,y2,mul,mul*yo(y2));
 //upd2(addtree,x,y1,add,add*yo(y1-1));
 //upd2(addtree,x,y2,add,add*yo(y2));
}
void upd(ll x1,ll y1,ll x2,ll y2,ll val)
{
 //for range sum
 upd1(x1,y1,y2,val,-val*yo(x1-1));
 upd1(x2,y1,y2,-val,val*yo(x2));
 //for range xor
}

```

```

//upd1(x1,y1,y2,val,val*yo(x1-1));
//upd1(x2,y1,y2,val,val*yo(x2));
}
int main()
{
 fast;
 ll i,j,k,n,m,tt,x1,y1,x2,y2,q,ans;
 cin>>n>>m;
 for(i=1;i<=n;i++){
 for(j=1;j<=m;j++){
 cin>>k;
 upd(i,j,i,j,k);
 }
 }
 cin>>q;
 while(q--){
 cin>>tt;
 if(tt==1){
 cin>>x1>>y1>>x2>>y2>>val;
 /// add val from top-left(x1,y1) to bottom-right (x2,y2);
 upd(x1,y1,x2,y2,val);
 }
 else{
 cin>>x1>>y1>>x2>>y2;
 /// output sum from top-left(x1,y1) to bottom-right (x2,y2);
 cout<<query(x1,y1,x2,y2)<<nl;
 }
 }
 return 0;
}

```

## 29. Binary Search Tree

```

//the code returns a BST which will create if we add the values one
by one
///here nodes are indicated by values and every node must be
distinct
set<ll>se;
map<ll,ll>le,ri;///le contains the left child of the node,ri contains
right child of the node
int main()
{
 fast;
 ll i,j,k,n,m,ans;
 cin>>n;
 cin>>k;//root of the tree
 se.insert(k);
 for(i=1;i<=n;i++){
 cin>>k;
 auto it=se.UB(k);
 if(it!=se.end()&&le.find(*it)==le.end()) le[*it]=k;
 else --it,ri[*it]=k;
 se.insert(k);
 }
 for(i=1;i<=n;i++) cout<<le[i]<<' '<<ri[i]<<nl;
 return 0;
}

```

## 30. Segment Tree

### Persistent Segment Tree

```

struct node
{
 int l,r,val;
 node(){l=r=val=0;}
 node(int _l,int _r,int _val){
 l=_l,r=_r,val=_val;
 }
}t[20*N];//size will be nlogn
int root[N],a[N],cnt;
void build(int cur,int b,int e)
{
 if(b==e){
 t[cur]=node(0,0,0);
 return ;
 }
 int left,right,mid=(b+e)/2;
 t[cur].l=left=++cnt;
 t[cur].r=right=++cnt;
 build(left,b,mid);
 build(right,mid+1,e);
 t[cur].val=t[left].val+t[right].val;
}
void upd(int pre,int cur,int b,int e,int i,int v)
{
 if(i<b || i>e) return;
 if(b==e){
 t[cur].val+=v;
 }
}

```

```

 return;
}
int left,right,mid=(b+e)/2;
if(i<=mid){
 t[cur].r=right=t[pre].r;
 t[cur].l=left=++cnt;
 upd(t[pre].l,t[cur].l,b,mid,i,v);
}
else{
 t[cur].l=left=t[pre].l;
 t[cur].r=right=++cnt;
 upd(t[pre].r,t[cur].r,mid+1,e,i,v);
}
t[cur].val=t[left].val+t[right].val;
}

int query(int pre,int cur,int b,int e,int k)
{
 if(b==e) return b;
 int cnt=t[t[cur].l].val-t[pre].l].val;
 int mid=(b+e)/2;
 if(cnt>=k) return query(t[pre].l,t[cur].l,b,mid,k);
 else return query(t[pre].r,t[cur].r,mid+1,e,k-cnt);
}
///1 2 2 3 , 3rd number is 3
///the code returns k-th unique number in a range l to r if the range
were sorted
int flag[N];
int main()
{
 fast;
}
```

```

int i,j,k,n,m,q,t,x,l,r,c=0;
map<int,int>mp;
cin>>n>>q;
for(i=1;i<=n;i++) cin>>a[i],mp[a[i]];
for(auto x:mp) mp[x.F]=++c,flag[c]=x.F;
root[0]=++cnt;
build(root[0],1,n);
for(i=1;i<=n;i++){
 root[i]=++cnt;
 upd(root[i-1],root[i],1,n,mp[a[i]],1);
}
while(q--){
 cin>>l>>r>>k;
 cout<<flag[query(root[l-1],root[r],1,n,k)]<<nl;
}

```

## Dynamic Segment Tree

```

///Complexity: $O(q \log n)$
///Given an zero array of size 1e9
///0 x y v,add v to segment [x,y]
///1 x y,output the sum of segment [x,y]
struct node
{
 node *l,*r;
 ll lazy;
 ll sum;
 node()
 {
 l=NULL;
 r=NULL;
 lazy=sum=0;
 }
}

```

```

 }
};

void propagate(node* t,int b,int e)
{
 if(t->lazy==0) return;
 t->sum+=1LL*(e-b+1)*t->lazy;
 if(!t->l) t->l=new node();
 if(!t->r) t->r=new node();
 t->l->lazy+=t->lazy;
 t->r->lazy+=t->lazy;
 t->lazy=0;
}

void upd(node* t,int b,int e,int i,int j,ll v)
{
 propagate(t,b,e);
 if(!t || b>j || e<i) return;
 if(b>=i & & e<=j){
 t->lazy+=v;
 propagate(t,b,e);
 return;
 }
 int mid=(b+e)/2;
 if(!t->l) t->l=new node();
 if(!t->r) t->r=new node();
 upd(t->l,b,mid,i,j,v);
 upd(t->r,mid+1,e,i,j,v);
 t->sum=t->l->sum+t->r->sum;
}

ll query(node* t,int b,int e,int i,int j)
{

```

```

propagate(t,b,e);
if(!t| b>j| e<i) return 0;
if(b>=i&&e<=j) return t->sum;
int mid=(b+e)/2;
if(!t->l) t->l=new node();
if(!t->r) t->r=new node();
return query(t->l,b,mid,i,j)+query(t->r,mid+1,e,i,j);
}
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,ty,l,r,q,tt;
 ll v;
 node* root;
 cin>>tt;
 while(tt--){
 root=new node();
 cin>>n;///max index of array,can be upto 1e9 or more
 cin>>q;
 while(q--){
 cin>>ty>>l>>r;
 if(ty==0){
 cin>>v;
 upd(root,1,n,l,r,v);
 }
 else cout<<query(root,1,n,l,r)<<nl;
 }
 }
 return 0;
}

```

## Segment Tree 2D

```

int n,m;
struct segtree
{
 int a[N*4];
 segtree()
 {
 for(int i=0;i<N*4;i++) a[i]=0;
 }
 void reset()
 {
 for(int i=0;i<N*4;i++) a[i]=0;
 }
 // update i-th column by val
 void upd(int node,int b,int e,int i,int val,vi &v)
 {
 v.pb(node);
 if(b==e)
 {
 a[node]=val;
 return;
 }
 int stree;
 if(i<=mid) upd(l,b,mid,i,val,v);
 else upd(r,mid+1,e,i,val,v);
 a[node]=a[l]+a[r];
 }
 //sum from column i to j
 int query(int node,int b,int e,int i,int j)

```

```

{
 if(j<b || i>e) return 0;
 if(b>=i&&e<=j) return a[node];
 int stree;
 return query(l,b,mid,i,j)+query(r,mid+1,e,i,j);
}
};

struct segtree2d
{
 segtree a[N*4];
 void reset()
 {
 for(int i=0;i<N*4;i++) a[i].reset();
 }
 vi v;
 //set a[i][j]=val
 void upd(int node,int b,int e,int i,int j,int val)
 {
 if(b==e)
 {
 v.clear();
 a[node].upd(1,1,m,j,val,v);
 return;
 }
 int stree;
 if(i>=b&&i<=mid) upd(l,b,mid,i,j,val);
 else upd(r,mid+1,e,i,j,val);
 for(auto x:v) a[node].a[x]=a[l].a[x]+a[r].a[x];
 }
 //return sum from top-left (i,y1) to bottom-right (j,y2)
}

```

```

int query(int node,int b,int e,int i,int y1,int j,int y2)
{
 if(j<b || i>e) return 0;
 if(b>=i&&e<=j) return a[node].query(1,1,m,y1,y2);
 int stree;
 return query(l,b,mid,i,y1,j,y2)+query(r,mid+1,e,i,y1,j,y2);
}
};

segtree2d t;
int main()
{
 // fast;
 int i,j,k,x,x1,y1,x2,y2,typ,q;
 // cin>>n>>m;
 // for(i=1;i<=n;i++){
 // for(j=1;j<=m;j++){
 // cin>>k;
 // t.upd(1,1,n,i,j,k);
 // }
 // }
 int tt;
 tt=sc();
 while(tt--){
 n=sc();
 m=n;
 while(1){
 char s[5];
 sf("%s",&s);
 if(s[2]=='D') break;//end
 else if(s[2]=='T'){//set a[i][j] as k

```

```

 i=sc(),j=sc(),x=sc();
 i++,j++;
 t.upd(1,1,n,i,j,x);
}
else{
 x1=sc(),y1=sc(),x2=sc(),y2=sc();
 x1++,y1++,x2++,y2++;
 // make sure (x1,y1) is top-left and (x2,y2) is bottom-right
 pf("%d\n",t.query(1,1,n,x1,y1,x2,y2));//sum of the
rectangle
}
t.reset();
}
return 0;
}

```

## Quad Tree

```

#define Max 501
#define INF (1 << 30)
int P[Max][Max]; // container for 2D grid

/* 2D Segment Tree node */
struct Point {
 int x, y, mx;
 Point() {}
 Point(int x, int y, int mx) : x(x), y(y), mx(mx) {}

 bool operator< (const Point& other) const {

```

```

 return mx < other.mx;
 }
};

struct Segtree2d {

 // I didn't calculate the exact size needed in terms of 2D container
 // size.
 // If anyone, please edit the answer.
 // It's just a safe size to store nodes for MAX * MAX 2D grids which
 // won't cause stack overflow :)
 Point T[500000]; // TODO: calculate the accurate space needed

 int n, m;

 // initialize and construct segment tree
 void init(int n, int m) {
 this->n = n;
 this->m = m;
 build(1, 1, 1, n, m);
 }

 // build a 2D segment tree from data [(a1, b1), (a2, b2)]
 // Time: O(n logn)
 Point build(int node, int a1, int b1, int a2, int b2) {
 // out of range
 if (a1 > a2 or b1 > b2)
 return def();

 // if it is only a single index, assign value to node

```

```

if (a1 == a2 and b1 == b2)
 return T[node] = Point(a1, b1, P[a1][b1]);

// split the tree into four segments
T[node] = def();
T[node] = maxNode(T[node], build(4 * node - 2, a1, b1, (a1 + a2) / 2, (b1 + b2) / 2));
T[node] = maxNode(T[node], build(4 * node - 1, (a1 + a2) / 2 + 1, b1, a2, (b1 + b2) / 2));
T[node] = maxNode(T[node], build(4 * node + 0, a1, (b1 + b2) / 2 + 1, (a1 + a2) / 2, b2));
T[node] = maxNode(T[node], build(4 * node + 1, (a1 + a2) / 2 + 1, a2, b2, x1, y1, x2, y2));

// helper function for query(int, int, int, int);
Point query(int node, int a1, int b1, int a2, int b2, int x1, int y1, int x2, int y2) {
 // if we out of range, return dummy
 if (x1 > a2 or y1 > b2 or x2 < a1 or y2 < b1 or a1 > a2 or b1 > b2)
 return def();

 // if it is within range, return the node
 if (x1 <= a1 and y1 <= b1 and a2 <= x2 and b2 <= y2)
 return T[node];

 // split into four segments
 Point mx = def();

```

```

 mx = maxNode(mx, query(4 * node - 2, a1, b1, (a1 + a2) / 2, (b1 + b2) / 2, x1, y1, x2, y2));
 mx = maxNode(mx, query(4 * node - 1, (a1 + a2) / 2 + 1, b1, a2, (b1 + b2) / 2, x1, y1, x2, y2));
 mx = maxNode(mx, query(4 * node + 0, a1, (b1 + b2) / 2 + 1, (a1 + a2) / 2, b2, x1, y1, x2, y2));
 mx = maxNode(mx, query(4 * node + 1, (a1 + a2) / 2 + 1, a2, b2, x1, y1, x2, y2));

 // return the maximum value
 return mx;
 }

 // query from range [(x1, y1), (x2, y2)]
 // Time: O(logn)
 Point query(int x1, int y1, int x2, int y2) {
 return query(1, 1, 1, n, m, x1, y1, x2, y2);
 }

 // helper function for update(int, int, int);
 Point update(int node, int a1, int b1, int a2, int b2, int x, int y, int value) {
 if (a1 > a2 or b1 > b2)
 return def();

 if (x > a2 or y > b2 or x < a1 or y < b1)
 return T[node];

 if (x == a1 and y == b1 and x == a2 and y == b2)
 return T[node] = Point(x, y, value);
 }
}
```

```

Point mx = def();
mx = maxNode(mx, update(4 * node - 2, a1, b1, (a1 + a2) / 2,
(b1 + b2) / 2, x, y, value));
mx = maxNode(mx, update(4 * node - 1, (a1 + a2) / 2 + 1, b1,
a2, (b1 + b2) / 2, x, y, value));
mx = maxNode(mx, update(4 * node + 0, a1, (b1 + b2) / 2 + 1,
(a1 + a2) / 2, b2, x, y, value));
mx = maxNode(mx, update(4 * node + 1, (a1 + a2) / 2 + 1, (b1 +
b2) / 2 + 1, a2, b2, x, y, value));
return T[node] = mx;
}

// update the value of (x, y) index to 'value'
// Time: O(logn)
Point update(int x, int y, int value) {
 return update(1, 1, 1, n, m, x, y, value);
}

// utility functions; these functions are virtual because they will be
overridden in child class
virtual Point maxNode(Point a, Point b) {
 return max(a, b);
}

// dummy node
virtual Point def() {
 return Point(0, 0, -INF);
}
};

```

```

/* 2D Segment Tree for range minimum query; a override of
Segtree2d class */
struct Segtree2dMin : Segtree2d {
 // overload maxNode() function to return minimum value
 Point maxNode(Point a, Point b) {
 return min(a, b);
 }

 Point def() {
 return Point(0, 0, INF);
 }
};

// initialize class objects
Segtree2d Tmax;
Segtree2dMin Tmin;

/* Drier program */
int main(void) {
 int n, m;
 // input
 scanf("%d %d", &n, &m);
 for(int i = 1; i <= n; i++)
 for(int j = 1; j <= m; j++)
 scanf("%d", &P[i][j]);

 // initialize
 Tmax.init(n, m);
}

```

```

Tmin.init(n, m);

// query
int x1, y1, x2, y2;
scanf("%d %d %d %d", &x1, &y1, &x2, &y2);

cout<<Tmax.query(x1, y1, x2, y2).mx<<endl;
cout<<Tmin.query(x1, y1, x2, y2).mx<<endl;

// update
int x, y, v;
scanf("%d %d %d", &x, &y, &v);
Tmax.update(x, y, v);
Tmin.update(x, y, v);

return 0;
}

```

## 31. Disjoint Set Union

### Persistent DSU

```

///Standard DSU with 'last added edges can be removed' capability
///Here is u,v connected? Query er answer deowa ache
///for connected component query see Dynamic Connectivity
Problem
struct persistent_dsu
{
 struct state
 {
 int u, v, rnku, rnkv;
 state() {u = -1; v = -1; rnkv = -1; rnku = -1;}
 }

```

```

state(int _u, int _rnku, int _v, int _rnkv)
{
 u = _u;
 rnku = _rnku;
 v = _v;
 rnkv = _rnkv;
}

stack<state> st;
int par[N], depth[N];
persistent_dsu() {memset(par, -1, sizeof(par));
memset(depth, 0, sizeof(depth));}

int root(int x)
{
 if(x == par[x]) return x;
 return root(par[x]);
}

void init(int n)
{
 for(int i = 0; i <= n; i++)
 {
 par[i] = i;
 depth[i] = 1;
 }
}

bool connected(int x, int y)

```

```

{
 return root(x) == root(y);
}

void unite(int x, int y)
{
 int rx = root(x), ry = root(y);
 st.push(state(rx, depth[rx], ry, depth[ry]));

 if(depth[rx] < depth[ry])
 par[rx] = ry;
 else if(depth[ry] < depth[rx])
 par[ry] = rx;
 else
 {
 par[rx] = ry;
 depth[rx]++;
 }
}
//how many last added edges you want to erase
void backtrack(int c)
{
 while(!st.empty() && c)
 {
 par[st.top().u] = st.top().u;
 par[st.top().v] = st.top().v;
 depth[st.top().u] = st.top().rnku;
 depth[st.top().v] = st.top().rnkv;
 st.pop();
 c--;
 }
}

};

persistent_dsu d;
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v;
 cin>>n>>m;
 d.init(n);
 while(m--){
 int ty;
 cin>>ty;
 if(ty==1){
 //add an edge
 cin>>u>>v;
 d.unite(u,v);
 }
 else if(ty==2){
 //remove last added k edges
 cin>>k;
 d.backtrack(k);
 }
 else{
 //if u and v is connected
 cin>>u>>v;
 if(d.connected(u,v)) cout<<"YES\n";
 else cout<<"NO\n";
 }
 }
}

```

```

}

Partially Persistent Dsu

//0 indexed
struct PartiallyPersistentDsu {
 vector<vector<pair<int, int>>> par; // (par index, modified time)
 int now = 0; // time = 0 is the initial state
 PartiallyPersistentDsu(int n) : par(n, {{-1,0}}) {}

 //merging components of u and v
 bool unite(int u, int v) {
 ++now;
 u = root(u, now); v = root(v, now);
 if (u == v) return false;
 if (par[u].back().F > par[v].back().F) swap(u, v);
 par[u].push_back({par[u].back().F+par[v].back().F, now});
 par[v].push_back({u, now});
 return true;
 }

 //if u and v is in the same component at time t
 bool same(int u, int v, int t) { return root(u, t) == root(v, t); }

 //root of u at time t
 int root(int u, int t) {
 if (par[u].back().F >= 0 && par[u].back().S <= t)
 return root(par[u].back().F, t);
 return u;
 }
}

//size of the component of u at time t
int size(int u, int t) {
 u = root(u, t);
 int lo = 0, hi = par[u].size();
 while (lo + 1 < hi) {
 int mid = (lo + hi) / 2;
 if (par[u][mid].S <= t) lo = mid;
 else hi = mid;
 }
 return -par[u][lo].F;
};

///given m pairs unite them one by one and then for each query (u,v)
///find the first time when u and v appears in the same component
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v,q;
 cin>>n;
 PartiallyPersistentDsu uf(n);
 cin>>m;
 for(i=0;i<m;i++) cin>>u>>v,&u,&v,uf.unite(u,v);
 cin>>q;
 while(q--){
 cin>>u>>v;
 --u,--v;
 int l=0,r=m,ans=-1;

```

```

while(l<=r){
 int mid=(l+r)/2;
 if(uf.same(u,v,mid)) ans=mid,r=mid-1;
 else l=mid+1;
}
cout<<ans<<nl;
}
return 0;
}

```

### Dynamic Connectivity Problem

*Complexity: O(m log m), where m is number of edges in the graph*

```

/// +,Add an edge to the graph
/// -, Delete an edge from the graph
/// ?, Count the number of connected components in the graph
struct persistent_dsu
{

```

```

 struct state
 {
 int u, v, rnku, rnkv;
 state() {u = -1; v = -1; rnku = -1; rnkv = -1;}
 state(int _u, int _rnku, int _v, int _rnkv)
 {
 u = _u;
 rnku = _rnku;
 v = _v;
 rnkv = _rnkv;
 }
 }
}
```

```

};

stack<state> st;
int par[N], depth[N];
int comp;
persistent_dsu() {comp=0;memset(par, -1, sizeof(par));
memset(depth, 0, sizeof(depth));}

int root(int x)
{
 if(x == par[x]) return x;
 return root(par[x]);
}

void init(int n)
{
 comp=n;
 for(int i = 0; i <= n; i++)
 {
 par[i] = i;
 depth[i] = 1;
 }
}

bool connected(int x, int y)
{
 return root(x) == root(y);
}
```

```

void unite(int x, int y)
{
 int rx = root(x), ry = root(y);
 if(rx==ry){
 st.push(state());
 return;
 }
 if(depth[rx] < depth[ry])
 par[rx] = ry;
 else if(depth[ry] < depth[rx])
 par[ry] = rx;
 else
 {
 par[rx] = ry;
 depth[rx]++;
 }
 comp--;
 st.push(state(rx, depth[rx], ry, depth[ry]));
}
//how many last added edges you want to erase
void backtrack(int c)
{
 while(!st.empty() && c)
 {
 if(st.top().u==-1){
 st.pop();
 c--;
 continue;
 }
 }
}

```

```

par[st.top().u] = st.top().u;
par[st.top().v] = st.top().v;
depth[st.top().u] = st.top().rnku;
depth[st.top().v] = st.top().rnkv;
st.pop();
c--;
comp++;

}

};

persistent_dsu d;
vpii alive[4*N];
void upd(int n,int b,int e,int i,int j,pii &p)
{
 if(b>j || e<i) return;
 if(b>=i & & e<=j){
 alive[n].eb(p);///this edge was alive in this time range
 return;
 }
 int stree;
 upd(l,b,mid,i,j,p);
 upd(r,mid+1,e,i,j,p);
}

int ans[N];
void query(int n,int b,int e)
{
 if(b>e) return;
 int prevsz=d.st.size();
 ///add edges which were alive in this range
}
```

```

for(auto p:alive[n]) d.unite(p.F,p.S);
if(b==e){
 ans[b]=d.comp;
 d.backtrack(d.st.size()-prevsz);
 return;
}
int stree;
query(l,b,mid);
query(r,mid+1,e);
d.backtrack(d.st.size()-prevsz);
}
struct HASH{
 size_t operator()(const pair<int,int>&x) const{
 return hash<long long>()(((long long)x.first)^(((long long)x.second)<<32));
 }
};
set<pii>se;
bool isquery[N];
map<pii,int>st;
int main()
{
 BeatMeScanf;
 //freopen("connect.in", "r", stdin);
 //freopen("connect.out", "w", stdout);
 int i,j,k,n,m,u,v;
 cin>>n>>m;
 d.init(n);
 for(i=1;i<=m;i++){

```

```

 string ty;
 cin>>ty;
 if(ty=="?"){
 isquery[i]=1;
 }
 else if(ty=="+"){
 cin>>u>>v;
 if(u>v) swap(u,v);
 pii p={u,v};
 se.insert(p);
 st[p]=i;
 }
 else{
 cin>>u>>v;
 if(u>v) swap(u,v);
 pii p={u,v};
 se.erase(p);
 upd(1,1,m,st[p],i-1,p);///in this time range this edge was in the
DSU
 }
 }
 for(auto p:se) upd(1,1,m,st[p],m,p);///update rest of the edges
 se.clear();
 query(1,1,m);
 for(i=1;i<=m;i++) if(isquery[i]) cout<<ans[i]<<nl;
 return 0;
}

```

## Augmented DSU

```

///Application:- used for maintaining a system of equations of the
form (y-x = d) along
///with their consistencial queries dynamically using disjoint set
union and find data structure.
int flaw; //counting numbers of inconsistent assertions
int val[N]; //val[i]=a[i]-a[root[i]] where root[i]=root of the
corresponding dsu of i
int par[N]; //adding a[i]-a[j]=d means setting j=par[i] and updating
val[i]

void init(int n)
{
 flaw=0;
 for(int i=1;i<=n;++i)
 {
 par[i]=i;
 val[i]=0;
 }
}
int find_(int x)
{
 if(par[x]==x) return x;
 int rx=find_(par[x]); // rx is the root of x
 val[x]=val[par[x]]+val[x]; //update all val along the
path,i.e.,val calculated wrt root
 par[x]=rx;
 return rx;
}

```

```

void merge_(int a,int b,int d)
{
 int ra=find_(a);
 int rb=find_(b);
 if(ra==rb && val[a]-val[b]!=d) flaw++;
 else if(ra!=rb)
 {
 if(rand()%2){
 val[ra]=d+val[b]-val[a];
 par[ra]=rb;
 }
 else{
 val[rb]=d+val[a]-val[b];
 par[rb]=ra;
 }
 }
}

int main()
{
 int i,j,k,n,m;
 cin>>n; //no. of variables
 cin>>m; //no. of equations
 init(n);
 for(int i=1;i<=m;++i) //consider 1-based indexing of
variables
 {

```

```

int a,b,d; //asserting a-b=d;
cin>>a>>b>>d;
merge_(a,b,d);
}
cout<<"No. of inconsistencies= "<<flaw;
///queries of type y-x=? can be given through val[y]-val[x]
///(only when then are in same component
///i.e., can be extracted from the information so far)
return 0;
}

```

### DSU Bipartite

```

pair<int, int> e[N];
int col[N], par[N];
vector<int> li[N];

int root(int u) { if(u == par[u]) return u; return (par[u] = root(par[u]));
}

///add an edge and check if the graph is still bipartite or not
void unite(int u, int v)
{
 if(root(u) == root(v))
 {
 if(col[u] == col[v])
 {
 cout << "NO" << endl;
 exit(0);
 }
 }
}

```

```

return;
}
if(li[root(u)].size() > li[root(v)].size()) swap(u, v);
if(col[u] == col[v]) for(int ver: li[root(u)]) col[ver] ^= 1;
for(int ver: li[root(u)]) li[root(v)].push_back(ver);
par[root(u)] = root(v);
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin >> n >> m;
 for(int i = 0; i < m; i++) cin >> e[i].first >> e[i].second;
 for(int i = 1; i <= n; i++) col[i] = 1, li[i].push_back(i), par[i] = i;
 for(int i = 0; i < m; i++) unite(e[i].first, e[i].second);
 cout << "YES" << endl;
 return 0;
}

```

### A DSU Problem

/\*\* Problem: You are given a graph with edge-weights. Each node has some color.  
Now you are also given some queries of the form (starting\_node, weight). The query means you can start from the starting\_node and you can visit only those edges which have weight  $\leq$  weight. You need to print the color which will occur

the maximum time in your journey. If two color occurs the same, output the lower indexed one.

You need to solve it online

Solution: idea is to use DSU small to large merging with binary-lifting.

\*\*/

```
const int LG = 17;
int n, m, a[N], parent[N], up[N][LG], weight[N][LG];
/// best (count,color_number) pair on a component after adding an
edge
pii best[N];
/// (weight,color) pair which means you can get max-occurrence of
'color' with 'weight'
vpii ans[N];
/// (color,cnt) pair for each component, stores all of them
set<pii> color[N];
vector<pair<int,pair<int,int>>> edges;
```

void clear()

```
{
 /// Initialize weights and 2^j parent of each node.
 /// Initially, each node is 2^j -th parent of itself
 for(int i=1;i<=n;i++)
 {
 for(int j=0;j<LG;j++) weight[i][j] = 1e9, up[i][j]=i;
 }
 edges.clear();
 for(int i=1;i<=n;i++) color[i].clear(), ans[i].clear();
}
```

```
int find(int r)
{
 if(parent[r]==r) return r;
 return parent[r]=find(parent[r]);
}

void merge(int u, int v)
{
 /// merge u into v
 for(auto it: color[u])
 {
 auto curr = color[v].lower_bound({it.first,-1});
 int cnt = it.second;

 if(curr!=end(color[v]) && curr->first==it.first)
 {
 cnt+=curr->second;
 color[v].erase(curr);
 }

 color[v].insert({it.first,cnt});
 /// best (cnt,color) pair for component v, -it.first for
ensuring
 /// that we get the smallest index while max-ing
 best[v] = max(best[v],{cnt,-it.first});
 }
}
```

```

void solve()
{
 for(int i=1;i<=n;i++) parent[i] = i;
 for(auto it: edges)
 {
 int w=it.F,u=it.S,F,v=it.S.S;
 u = find(u);
 v = find(v);

 if(color[u].size()>color[v].size())
 swap(u,v);
 if(u!=v)
 {
 merge(u,v);
 parent[u] = v;
 /// after merging (u,v), we store best answer
 /// for the component v in ans[v]
 ans[v].pb({w,-best[v].second});
 up[u][0] = v;
 weight[u][0] = w;
 }
 /// note that if u==v, that edge and its weight won't
matter as
 /// we have already added the smaller edges and the
nodes are already
 /// connected
 }
 for(int i=1; i<LG; i++)
 {

```

```

 for(int j=1; j<=n; j++)
 {
 /// 2^ith component of j in dsu
 up[j][i] = up[up[j][i-1]][i-1];
 /// weight that we need to consider
 weight[j][i] = weight[up[j][i-1]][i-1];
 }
 }

int main()
{
 int tc, cs = 1;

 scanf("%d", &tc);
 while(tc--)
 {
 scanf("%d%d", &n, &m);
 clear();
 for(int i=1;i<=n;i++)
 {
 scanf("%d", &a[i]);
 /// initializing each node as a single component
 color[i].insert({a[i],1});
 best[i] = {1,-a[i]};
 ans[i].pb({0,a[i]});
 }
 int u, v, w;

```

```

for(int i=1;i<=m;i++)
{
 scanf("%d%d%d", &u, &v, &w);
 edges.pb({w,{u,v}});
}
sort(begin(edges),end(edges));
solve();

int last = 0, q;
scanf("%d", &q);
printf("Case #%d:\n", cs++);
while(q--)
{
 scanf("%d%d", &u, &w);
 // the problem used this xor-ing to make the solution
 online
 u^=last, w^=last;
 // u will be the component we can visit
 for(int i=LG-1; i>=0; i--) if(weight[u][i]<=w) u=up[u][i];
 int idx = lower_bound(all(ans[u]), pii{w,2e9}) -
begin(ans[u]) - 1;
 last = ans[u][idx].second;
 printf("%d\n", last);
}
clear();
}
return 0;
}

```

## 32. MO's Algorithm

### MO's Algorithm Standard

Complexity:  $O((n + q)\sqrt{n})$

```

struct sj
{
 int l,r,idx;
}q[mxn];
int block,a[mxn],curl,curr,cnt[mxn*5];
ll ans,sol[mxn];
bool cmp(sj a,sj b)
{
 if(a.l/block==b.l/block) return a.r<b.r;
 else return a.l/block<b.l/block;
}
inline void add(int x)//careful x is the number not the index
{
// When adding a number, we first nullify it's effect on current
// answer, then update cnt array, then account for it's effect again.
 ans-=1LL*cant[x]*cant[x]*x;
 cant[x]++;
 ans+=1LL*cant[x]*cant[x]*x;
}
inline void remov(int x)
{
// Removing is much like adding.
 ans-=1LL*cant[x]*cant[x]*x;
 cant[x]--;
 ans+=1LL*cant[x]*cant[x]*x;
}

```

```

int main()
{
 fast;
 int i,j,k,n,m,t;
 cin>>n>>t;
 for(i=1;i<=n;i++) cin>>a[i];
 for(i=0;i<t;i++){
 cin>>q[i].l>>q[i].r;
 q[i].idx=i;
 }
 block=(int)sqrt(n);
 sort(q,q+t,cmp);
 for(i=0;i<t;i++){
 while(curl<q[i].l) remov(a[curl++]);
 while(curl>q[i].l) add(a[--curl]);
 while(curr<q[i].r) add(a[++curr]);
 while(curr>q[i].r) remov(a[curr--]);
 sol[q[i].idx]=ans;
 }
 for(i=0;i<t;i++) cout<<sol[i]<<nl;
 return 0;
}

```

### MO's Algorithm GilbertOrder

Complexity:  $O(n\sqrt{q})$

```

//always better than standard algorithm
//far better for small amount of queries
struct sj
{

```

```

 int l,r,idx;
 int64_t ord;
}q[N];
int a[N],curl,curr,cnt[N*5];
ll ans,sol[N];
inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
 if (pow == 0) {
 return 0;
 }
 int hpow = 1 << (pow-1);
 int seg = (x < hpow) ? (
 (y < hpow) ? 0 : 3
) : (
 (y < hpow) ? 1 : 2
);
 seg = (seg + rotate) & 3;
 const int rotateDelta[4] = {3, 0, 0, 1};
 int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
 int nrot = (rotate + rotateDelta[seg]) & 3;
 int64_t subSquareSize = int64_t(1) << (2*pow - 2);
 int64_t ans = seg * subSquareSize;
 int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
 ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add -
1);
 return ans;
}
bool cmp(sj a,sj b)
{
 return a.ord<b.ord;
}

```

```

inline void add(int x)//careful x is the number not the index
{
// When adding a number, we first nullify it's effect on current
// answer, then update cnt array, then account for it's effect again.
 ans-=1LL*cnt[x]*cnt[x]*x;
 cnt[x]++;
 ans+=1LL*cnt[x]*cnt[x]*x;
}
inline void remov(int x)
{
// Removing is much like adding.
 ans-=1LL*cnt[x]*cnt[x]*x;
 cnt[x]--;
 ans+=1LL*cnt[x]*cnt[x]*x;
}
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,t;
 cin>>n>>t;
 for(i=1;i<=n;i++) cin>>a[i];
 for(i=0;i<t;i++){
 cin>>q[i].l>>q[i].r;
 q[i].idx=i;
 }
 for(i=0;i<t;i++) q[i].ord=gilbertOrder(q[i].l,q[i].r,21,0);
 sort(q,q+t,cmp);
 for(i=0;i<t;i++){
 while(curl<q[i].l) remov(a[curl++]);
 while(curl>q[i].l) add(a[--curl]);
 }
}

```

```

while(curr<q[i].r) add(a[++curr]);
while(curr>q[i].r) remov(a[curr--]);
sol[q[i].idx]=ans;
}
for(i=0;i<t;i++) cout<<sol[i]<<nl;
return 0;
}

```

## Mo's on Tree

```

///unique elements in path u to v
int ans,curl,curr,timee,que,block,val[mxn],node[mxn*2],n;
int
st[mxn],en[mxn],level[mxn],par[mxn][20],cnt[mxn],id[mxn],sol[mxm
];
bool vis[mxn];
vi g[mxn];
void remov(int u);
struct sj
{
 int l,r,lc,idx;
}q[mxm];
void dfs(int u,int prev)
{
 par[u][0]=prev;
 level[u]=level[prev]+1;
 node[timee]=u;
 st[u]=timee++;
 for(auto v:g[u]){
 if(v==prev) continue;
 dfs(v,u);
 }
}

```

```

 }
 node[timee]=u;
 en[u]=timee++;
}
int lca(int u,int v)
{
 if(level[u]<level[v]) swap(u,v);
 for(int k=19;k>=0;k--) if(level[par[u][k]]>=level[v]) u=par[u][k];
 if(u==v) return u;
 for(int k=19;k>=0;k--) if(par[u][k]!=par[v][k])
 u=par[u][k],v=par[v][k];
 return par[u][0];
}
bool cmp(sj x,sj y)
{
 int l=(x.l-1)/block;
 int r=(y.l-1)/block;
 if(l==r) return x.r<y.r;
 else return l<r;
}
void add(int u)
{
 int x=id[u];
 if(cnt[x]+==0) ans++;
 return;
}
void remov(int u)
{
 int x=id[u];
 if(--cnt[x]==0) ans--;
}

```

```

 return;
}
void add_list(int u)
{
 if(vis[u]==0) add(u);
 else remov(u);
 vis[u]^=1;
}
void remov_list(int u)
{
 if(vis[u]==0) add(u);
 else remov(u);
 vis[u]^=1;
}
void mos_algo()
{
 int i,j,k,u,v,l,r;
 sort(q,q+que,cmp);
 curl=q[0].l,curr=q[0].l-1;
 ans=0;
 for(i=0;i<que;i++){
 l=q[i].l,r=q[i].r;
 while(curl<l) remov_list(node[curl++]);
 while(curl>l) add_list(node[--curl]);
 while(curr<r) add_list(node[++curr]);
 while(curr>r) remov_list(node[curr-]);
 u=node[curl],v=node[curr];
 if(q[i].lc!=u&&q[i].lc!=v) add_list(q[i].lc);
 sol[q[i].idx]=ans;
 if(q[i].lc!=u&&q[i].lc!=v) remov_list(q[i].lc);
 }
}

```

```

 }
}

void compress()
{
 map<int,int>mp;
 for(int i=1;i<=n;i++){
 if(mp.find(val[i])==mp.end()) mp[val[i]]=mp.size();
 id[i]=mp[val[i]];
 }
}
int main()
{
 fast;
 int i,j,k,m,u,v;
 while(cin>>n>>que){
 timee=1;
 ans=0;
 for(i=1;i<=n;i++) cin>>val[i];
 compress();
 for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
 dfs(1,0);
 for(k=1;k<20;k++) for(i=1;i<=n;i++) par[i][k]=par[par[i][k-1]][k-1];
 timee--;
 block=(int)(sqrt(timee)+eps);
 for(i=0;i<que;i++){
 cin>>u>>v;
 if(level[u]>level[v]) swap(u,v);
 q[i].lc=lca(u,v);
 if(q[i].lc==u) q[i].l=st[u],q[i].r=st[v];
 else q[i].l=en[u],q[i].r=st[v];
 }
 }
}

```

```

 q[i].idx=i;
}
mos_algo();
for(i=0;i<que;i++) cout<<sol[i]<<nl;
for(i=0;i<=n;i++){
 level[i]=0;
 vis[i]=0,cnt[i]=0,g[i].clear();
 for(k=0;k<20;k++) par[i][k]=0;
}
return 0;
}

```

### Mo's with Update

**Complexity:**  $O(n^{\frac{5}{3}} + q \cdot n^{\frac{2}{3}})$

///sum of distinct numbers in range with update

```

int block;
struct query
{
 int l,r;//query range
 int updcnt;//number of update happened before this query
 int idx;//query index
 bool operator<(const query& x) const
 {
 if(l/block==x.l/block){
 if(r/block==x.r/block) return updcnt<x.updcnt;
 else return r/block<x.r/block;
 }
 else return l/block<x.l/block;
 }
}

```

```

 }
}q[N];
struct Upd
{
 int idx;//update index
 int prv;//array had value prv before this update
 int nxt;//array will have value nxt after this update
}upd[N];
int curl,curr,a[N],cnt[5*N],val[5*N];
ll ans;
umap<int,int>mp;
/// When adding a number, we first nullify it's effect on current
/// answer, then update corresponding array, then account for it's
effect again.
inline void add(int x)
{
 cnt[x]++;
 if(cnt[x]==1) ans+=val[x];
}
///removing is quite same as adding
inline void remov(int x)
{
 cnt[x]--;
 if(cnt[x]==0) ans-=val[x];
}
inline void do_upd(int upd_idx)
{
 int i=upd[upd_idx].idx;
 int now=upd[upd_idx].nxt;

```

```

 ///if the update is within our current range then re-correct our
ans
 if(i>=curl&&i<=curr) remov(a[i]);
 a[i]=now;
 if(i>=curl&&i<=curr) add(a[i]);
}
inline void undo_upd(int upd_idx)
{
 int i=upd[upd_idx].idx;
 int now=upd[upd_idx].prv;
 ///if the update is within our current range then re-correct our
ans
 if(i>=curl&&i<=curr) remov(a[i]);
 a[i]=now;
 if(i>=curl&&i<=curr) add(a[i]);
}
int aux[N];
ll res[N];
int main()
{
 //BeatMeScanf;
 int i,j,k,n,m,U=0,Q=0,que,l,r,t,z=0;
 char ch;
 sf1(n);
 for(i=1;i<=n;i++){
 sf1(a[i]);
 if(!mp.count(a[i])) mp[a[i]]+=z,val[z]=a[i];//compressing
 a[i]=mp[a[i]];
 aux[i]=a[i];
 }
}
```

```

}
sf1(que);
for(i=0;i<que;i++){
 sf("%c",&ch);
 sf2(l,r);
 if(ch=='U'){
 ++U;
 upd[U].idx=l;
 if(!mp.count(r)) mp[r]=++z,val[z]=r;//compressing
 upd[U].nxt=mp[r];
 upd[U].prv=aux[l];
 aux[l]=mp[r];
 }
 else{
 q[Q].l=l;
 q[Q].r=r;
 q[Q].updcnt=U;
 q[Q].idx=i;
 Q++;
 }
}
block=(int)(cbrt(n*1.0)+eps);
block*=block;
sort(q,q+Q);
int cur=0;//tracks the number of updates already happened
mem(res,-1);
for(i=0;i<Q;i++){
 l=q[i].l,r=q[i].r,t=q[i].updcnt;
 while(cur<t){
 cur++;
}

```

```

 do_upd(cur);
 }
 while(cur>t){
 undo_upd(cur);
 cur--;
 }
 while(curl<l) remov(a[curl++]);
 while(curl>l) add(a[-curl]);
 while(curr<r) add(a[++curr]);
 while(curr>r) remov(a[curr--]);
 res[q[i].idx]=ans;
}
for(i=0;i<que;i++) if(res[i]!=-1) pf1ll(res[i]);
return 0;
}

```

### MO's Online

```

#include <bits/stdc++.h>
#define endl '\n'

using namespace std;
const int MAXN = (int)1e5 + 42;
const int B = 2172;//MAXN/B~50

///upd a[i]=val
///query return distinct elements in range
///Complexity: n/B*n/B*(q+B)+q*B
int n, q;
int a[MAXN];

```

```

void read()
{
 cin >> n >> q;
 for(int i = 0; i < n; i++)
 cin >> a[i];
}

struct my_set
{
 int answer;
 unordered_map<int, int> cnt;
 my_set() { answer = 0; cnt.clear(); }

 void insert(int val)
 {
 int memo = ++cnt[val];
 if(memo == 1) answer++;
 }

 void erase(int val)
 {
 int memo = --cnt[val];
 if(memo == 0) answer--;
 }

 void clear()
 {
 answer = 0;
 cnt.clear();
 }
}

class Node
{
public:
 int query() { return answer; }

private:
 int st_block[MAXN / B + 42], en_block[MAXN / B + 42], cnt_blocks = 0;
 my_set st[(MAXN / B + 1) * (MAXN / B + 1) + 42], nw_st;

 int query(int l, int r)
 {
 int Lblock = l / B, Rblock = r / B;
 if(r != en_block[Rblock]) Rblock--;
 if(l != st_block[Lblock]) Lblock++;

 if(Rblock < Lblock)
 {
 nw_st.clear();
 for(int i = l; i <= r; i++)
 nw_st.insert(a[i]);
 return nw_st.query();
 }

 int mid = Lblock * cnt_blocks + Rblock;
 for(int i = l; i < st_block[Lblock]; i++) st[mid].insert(a[i]);
 for(int i = en_block[Rblock] + 1; i <= r; i++) st[mid].insert(a[i]);

 int answer = st[mid].query();
 }
}

```

```

 for(int i = l; i < st_block[Lblock]; i++) st[mid].erase(a[i]);
 for(int i = en_block[Rblock] + 1; i <= r; i++) st[mid].erase(a[i]);

 return answer;
 }

void update(int mid, int pos, int val)
{
 st[mid].erase(a[pos]);
 st[mid].insert(val);
}

void solve()
{
 for(int i = 0; i < n; i++)
 {
 if(i % B == 0) st_block[i / B] = i, cnt_blocks++;
 if(i % B == B - 1 || i == n - 1)
 en_block[i / B] = i;
 }

 for(int i = 0; i < cnt_blocks; i++)
 for(int j = i; j < cnt_blocks; j++)
 {
 int mid = i * cnt_blocks + j;
 st[mid] = my_set();
 for(int p = st_block[i]; p <= en_block[j]; p++)
 st[mid].insert(a[p]);
 }
}

```

```

 }

 for(int p = 0; p < q; p++)
 {
 int type;
 cin >> type;

 if(type == 1)
 {
 int l, r;
 cin >> l >> r;
 cout << query(l, r) << endl;
 }
 else
 {
 int pos, val;
 cin >> pos >> val;

 for(int i = 0; i < cnt_blocks; i++)
 for(int j = i; j < cnt_blocks; j++)
 if(st_block[i] <= pos && pos <=
en_block[j])
 update(i * cnt_blocks +
j, pos, val);

 a[pos] = val;
 }
 }
}

```

```

int main()
{
 ios_base::sync_with_stdio(false);

 read();
 solve();
 return 0;
}

```

## MO's with DSU

```

///Complexity: $O((m + q)\sqrt{m} \cdot \text{some small constant})$
///This code runs for $1 \leq n, m, q \leq 2e5$ in 2 second
///Given n vertices and m edges perform q queries of type (l,r)
///output the number of connected components if we added edges i
such that $l \leq i \leq r$
const int N = 2e5+9;
struct query
{
 int l, r, idx;
 query() {l = 0; r = 0; idx = 0;}
 query(int _l, int _r, int _idx)
 {
 l = _l;
 r = _r;
 idx = _idx;
 }
};

struct persistent_dsu

```

```

{
 struct state
 {
 int u, ru, v, rv;
 state() {u = 0; ru = 0; v = 0; rv = 0;}
 state(int _u, int _ru, int _v, int _rv)
 {
 u = _u;
 ru = _ru;
 v = _v;
 rv = _rv;
 }
 };
 int cnt;
 int depth[N], par[N];
 stack<state> st;

 persistent_dsu()
 {
 cnt = 0;
 memset(depth, 0, sizeof(depth));
 memset(par, 0, sizeof(par));
 while(!st.empty()) st.pop();
 }

 void init(int _sz)
 {
 cnt = _sz;
 for(int i = 0; i <= _sz; i++)

```

```

 par[i] = i, depth[i] = 1;
 }

int root(int x)
{
 if(x == par[x]) return x;
 return root(par[x]);
}

bool connected(int x, int y)
{
 return root(x) == root(y);
}

void unite(int x, int y)
{
 int rx = root(x), ry = root(y);
 if(rx == ry) return;

 if(depth[rx] < depth[ry])
 par[rx] = ry;
 else if(depth[ry] < depth[rx])
 par[ry] = rx;
 else par[rx] = ry, depth[ry]++;
}

cnt--;
st.push(state(rx, depth[rx], ry, depth[ry]));
}

```

```

void snapshot() { st.push(state(-1, -1, -1, -1)); }

void rollback()
{
 while(!st.empty())
 {
 if(st.top().u == -1)
 return;

 ++cnt;
 par[st.top().u] = st.top().u;
 par[st.top().v] = st.top().v;
 depth[st.top().u] = st.top().ru;
 depth[st.top().v] = st.top().rv;
 st.pop();
 }
};

struct edge
{
 int u, v;
 edge() {u = 0; v = 0;}
 edge(int _u, int _v)
 {
 u = _u;
 v = _v;
 }
};

```

```

int n, ed, m;
edge a[N];
query q[N];

void read()
{
 cin >> n >> ed >> m;

 for(int i = 1; i <= ed; i++)
 {
 int u, v;
 cin >> u >> v;
 a[i] = edge(u, v);
 }
}

```

```

int rt, cnt_q;
persistent_dsu d;

```

```

bool cmp(query fir, query sec)
{
 if(fir.l / rt != sec.l / rt) return fir.l / rt < sec.l / rt;
 return fir.r < sec.r;
}

```

```

int answer[N];
void add(int idx) { d.unite(a[idx].u, a[idx].v); }

void solve()
{

```

```

d.init(n);
d.snapshot();
rt = sqrt(ed);
cnt_q = 0;

int fm = m;
for(int i = 0; i < m; i++)
{
 int l, r;
 cin >> l >> r;

 if(r - l + 1 <= rt)
 {
 for(int k = l; k <= r; k++) add(k);
 answer[i] = d.cnt;
 d.rollback();
 continue;
 }

 q[cnt_q++] = query(l, r, i);
}

m = cnt_q;
sort(q, q + m, cmp);
int last, border, last_block = -1, block;

for(int i = 0; i < m; i++)
{
 block = q[i].l / rt;
 if(last_block != block)

```

```

{
 d.init(n);
 border = rt * (block + 1);
 last = border;
}

last_block = block;
for(int k = last + 1; k <= q[i].r; k++) add(k);
d.snapshot();

for(int k = q[i].l; k <= border; k++) add(k);
answer[q[i].idx] = d.cnt;
d.rollback();

last = q[i].r;

}

for(int i = 0; i < fm; i++)
 cout << answer[i] << endl;
}

int main()
{
 ios_base::sync_with_stdio(false);
 int t;
 cin >> t;
 while(t--){
 read();
 solve();
 }
}

```

```

}
return 0;
}
```

### 33. Sparse Table

```

///Standard RMQ problem
int t[N][21],lg2[N];
int main()
{
 fast;
 int i,j,k,n,m,l,r,q;
 lg2[1]=0;
 for(i=2;i<N;i++) lg2[i]=lg2[i/2]+1;
 cin>>n;
 for(i=1;i<=n;i++) cin>>k,t[i][0]=k;
 for(j=1;j<=20;j++) for(i=1;i+(1<<(j-1))<=n;i++) t[i][j]=min(t[i][j-1],t[i+(1<<(j-1))][j-1]);
 cin>>q;
 while(q--){
 cin>>l>>r;
 l++,r++;
 //int ans=INT_MAX;
 //for(i=20;i>=0;i--) if(l+(1<<i)-1<=r)
 ans=min(ans,t[l][i]),l+=(1<<i);
 k=lg2[r-l+1];
 int ans=min(t[l][k],t[r-(1<<k)+1][k]);
 cout<<ans<<nl;
 }
 return 0;
}
```

```
}
```

### 34. Merge Sort Tree

```
//number of elements greater than k in a range
vi t[4*N];
int a[N];
void build(int n,int b,int e)
{
 if(b==e){
 t[n].eb(a[b]);
 return;
 }
 int stree;
 build(l,b,mid);
 build(r,mid+1,e);
 merge(all(t[l]),all(t[r]),back_inserter(t[n]));
}
int query(int n,int b,int e,int i,int j,int k)
{
 if(b>j || e<i) return 0;
 if(b>=i && e<=j){
 return (int)t[n].size()-(UB(all(t[n]),k)-t[n].begin());
 }
 int stree;
 return query(l,b,mid,i,j,k)+query(r,mid+1,e,i,j,k);
}
int main()
{
 BeatMeScarf;
}
```

```
int i,j,k,n,m,q;
cin>>n;
for(i=1;i<=n;i++) cin>>a[i];
build(1,1,n);
for(i=1;i<=4*n;i++){
 cout<<i<<" ";
 for(auto x:t[i]) cout<<x<<' ';
 cout<<nl;
}
cin>>q;
int ans=0;
while(q--){
 cin>>i>>j>>k;
 i^=ans;
 j^=ans;
 k^=ans;
//online query
 ans=query(1,1,n,i,j,k);
 cout<<ans<<nl;
}
return 0;
}
```

### 35. SQRT Decomposition

```
// number of elements greater than k in range with update
int n,idx,block=400,a[mxn],t[200][10010];
void upd(int k,int i,int v)
{
 while(i<10010) t[k][i]+=v,i+=i&-i;
}
```

```

int query(int k,int i)
{
 int ans=0;
 while(i>0) ans+=t[k][i],i-=i&-i;
 return ans;
}
void build()
{
 for(int i=1;i<=n;i++){
 if(i%block==0) idx++;
 upd(idx,a[i],1);
 }
}
void upd(int i,int v)
{
 int ind=i/block;
 upd(ind,a[i],-1);
 upd(ind,v,1);
 a[i]=v;
}
int query(int l,int r,int k)
{
 int ans=0;
 while(l<=r&&l%block!=0) ans+=(a[l]>k),l++;
 while(l+block<=r) ans+=query(l/block,10010)-query(l/block,k),l+=block;
 while(l<=r) ans+=(a[l]>k),l++;
 return ans;
}
int main()

```

```

{
 fast;
 int i,j,k,m,q,l,r,typ,v;
 cin>>n;
 for(i=1;i<=n;i++) cin>>a[i];
 build();
 cin>>q;
 while(q--){
 cin>>typ;
 if(typ==0){
 cin>>i>>v;
 upd(i,v);
 }
 else{
 cin>>l>>r>>k;
 cout<<query(l,r,k)<<nl;
 }
 }
 return 0;
}

```

## 36. SQRT Tree

### SQRT Tree With Update

///Given an array a that contains n elements and the  
 ///operation op that satisfies associative property:  
 ///(x op y) op z=x op (y op z) is true for any x, y, z.

///The following implementation of Sqrt Tree can perform the  
 following operations:

```

///build in O(nlogn),
///answer queries in O(1) and update an element in O(sqrt(n)).

#define SqrtTreeItem int//change for the type you want

SqrtTreeItem op(const SqrtTreeItem &a, const SqrtTreeItem &b)
{
 return a+b;//just change this operation for different problems,no
change is required inside the code
}

inline int log2Up(int n) {
 int res = 0;
 while ((1 << res) < n) {
 res++;
 }
 return res;
}
///0-indexed
struct SqrtTree {
 int n, llg, indexSz;
 vector<SqrtTreeItem> v;
 vector<int> clz, layers, onLayer;
 vector<vector<SqrtTreeItem>> pref, suf, between;

 inline void buildBlock(int layer, int l, int r) {
 pref[layer][l] = v[l];
 for (int i = l+1; i < r; i++) {
 pref[layer][i] = op(pref[layer][i-1], v[i]);
 }
 }
}

```

```

suf[layer][r-1] = v[r-1];
for (int i = r-2; i >= l; i--) {
 suf[layer][i] = op(v[i], suf[layer][i+1]);
}
}

inline void buildBetween(int layer, int lBound, int rBound, int
betweenOffs) {
 int bSzLog = (layers[layer]+1) >> 1;
 int bCntLog = layers[layer] >> 1;
 int bSz = 1 << bSzLog;
 int bCnt = (rBound - lBound + bSz - 1) >> bSzLog;
 for (int i = 0; i < bCnt; i++) {
 SqrtTreeItem ans;
 for (int j = i; j < bCnt; j++) {
 SqrtTreeItem add = suf[layer][lBound + (j << bSzLog)];
 ans = (i == j) ? add : op(ans, add);
 between[layer-1][betweenOffs + lBound + (i << bCntLog) +
j] = ans;
 }
 }
}

inline void buildBetweenZero() {
 int bSzLog = (llg+1) >> 1;
 for (int i = 0; i < indexSz; i++) {
 v[n+i] = suf[0][i << bSzLog];
 }
 build(1, n, n + indexSz, (1 << llg) - n);
}

```

```

inline void updateBetweenZero(int bid) {
 int bSzLog = (lrg+1) >> 1;
 v[n+bid] = suf[0][bid << bSzLog];
 update(1, n, n + indexSz, (1 << lrg) - n, n+bid);
}

void build(int layer, int lBound, int rBound, int betweenOffs) {
 if (layer >= (int)layers.size()) {
 return;
 }
 int bSz = 1 << ((layers[layer]+1) >> 1);
 for (int l = lBound; l < rBound; l += bSz) {
 int r = min(l + bSz, rBound);
 buildBlock(layer, l, r);
 build(layer+1, l, r, betweenOffs);
 }
 if (layer == 0) {
 buildBetweenZero();
 } else {
 buildBetween(layer, lBound, rBound, betweenOffs);
 }
}

void update(int layer, int lBound, int rBound, int betweenOffs, int
x) {
 if (layer >= (int)layers.size()) {
 return;
 }
 int bSzLog = (layers[layer]+1) >> 1;
}

```

```

int bSz = 1 << bSzLog;
int blockIdx = (x - lBound) >> bSzLog;
int l = lBound + (blockIdx << bSzLog);
int r = min(l + bSz, rBound);
buildBlock(layer, l, r);
if (layer == 0) {
 updateBetweenZero(blockIdx);
} else {
 buildBetween(layer, lBound, rBound, betweenOffs);
}
update(layer+1, l, r, betweenOffs, x);

inline SqrtTreeItem query(int l, int r, int betweenOffs, int base) {
 if (l == r) {
 return v[l];
 }
 if (l + 1 == r) {
 return op(v[l], v[r]);
 }
 int layer = onLayer[clz[(l - base) ^ (r - base)]];
 int bSzLog = (layers[layer]+1) >> 1;
 int bCntLog = layers[layer] >> 1;
 int lBound = (((l - base) >> layers[layer]) << layers[layer]) + base;
 int lBlock = ((l - lBound) >> bSzLog) + 1;
 int rBlock = ((r - lBound) >> bSzLog) - 1;
 SqrtTreeItem ans = suf[layer][l];
 if (lBlock <= rBlock) {
 SqrtTreeItem add = (layer == 0) ? (
 query(n + lBlock, n + rBlock, (1 << lrg) - n, n)

```

```

) : (
 between[layer-1][betweenOffs + lBound + (lBlock <<
bCntLog) + rBlock]
);
ans = op(ans, add);
}
ans = op(ans, pref[layer][r]);
return ans;
}

inline SqrtTreeItem query(int l, int r) {
 return query(l, r, 0, 0);
}

inline void update(int x, const SqrtTreeItem &item) {
 v[x] = item;
 update(0, 0, n, 0, x);
}

SqrtTree(const vector<SqrtTreeItem>& a)
: n((int)a.size()), llg(log2Up(n)), v(a), clz(1 << llg), onLayer(llg+1) {
clz[0] = 0;
for (int i = 1; i < (int)clz.size(); i++) {
 clz[i] = clz[i >> 1] + 1;
}
int tllg = llg;
while (tllg > 1) {
 onLayer[tllg] = (int)layers.size();
 layers.push_back(tllg);
 tllg = (tllg+1) >> 1;
}
}

} // namespace
}
for (int i = llg-1; i >= 0; i--) {
 onLayer[i] = max(onLayer[i], onLayer[i+1]);
}
int betweenLayers = max(0, (int)layers.size() - 1);
int bSzLog = (llg+1) >> 1;
int bSz = 1 << bSzLog;
indexSz = (n + bSz - 1) >> bSzLog;
v.resize(n + indexSz);
pref.assign(layers.size(), vector<SqrtTreeItem>(n + indexSz));
suf.assign(layers.size(), vector<SqrtTreeItem>(n + indexSz));
between.assign(betweenLayers, vector<SqrtTreeItem>((1 << llg
+ bSz)));
 build(0, 0, n, 0);
}
};

int main()
{
 BeatMeScanf;
 int i,j,k,n,m,q,l,r;
 cin>>n;
 vi v;
 for(i=0;i<n;i++) cin>>k,v.eb(k);
 SqrtTree t=SqrtTree(v);
 cin>>q;
 while(q--){
 cin>>l>>r;
 --l,--r;
 cout<<t.query(l,r)<<nl;
 }
}

```

```
}
```

## SQRT Tree Without Update

```
///Use Same strategy as before
///One can use previous code for without update but in onsite
contest this is faster to code
int op(int a, int b)
{
}

inline int log2Up(int n)
{
 int res = 0;
 while ((1 << res) < n)
 {
 res++;
 }
 return res;
}

struct SqrtTree
{
 int n, llg;
 vector<int> v;
 vector<int> clz;
 vector<int> layers;
 vector<int> onLayer;
 vector<vector<int> > pref;
 vector<vector<int> > suf;
```

```
vector< vector<int> > between;

void build(int layer, int lBound, int rBound)
{
 if (layer >= (int)layers.size())
 {
 return;
 }
 int bSzLog = (layers[layer]+1) >> 1;
 int bCntLog = layers[layer] >> 1;
 int bSz = 1 << bSzLog;
 int bCnt = 0;
 for (int l = lBound; l < rBound; l += bSz)
 {
 bCnt++;
 int r = min(l + bSz, rBound);
 pref[layer][l] = v[l];
 for (int i = l+1; i < r; i++)
 {
 pref[layer][i] = op(pref[layer][i-1], v[i]);
 }
 suf[layer][r-1] = v[r-1];
 for (int i = r-2; i >= l; i--)
 {
 suf[layer][i] = op(v[i], suf[layer][i+1]);
 }
 build(layer+1, l, r);
 }
 for (int i = 0; i < bCnt; i++)
 {
```

```

int ans = 0;
for (int j = i; j < bCnt; j++)
{
 int add = suf[layer][lBound + (j << bSzLog)];
 ans = (i == j) ? add : op(ans, add);
 between[layer][lBound + (i << bCntLog) + j] = ans;
}
}

inline int query(int l, int r)
{
 if (l == r)
 {
 return v[l];
 }
 if (l + 1 == r)
 {
 return op(v[l], v[r]);
 }
 int layer = onLayer[clz[l ^ r]];
 int bSzLog = (layers[layer]+1) >> 1;
 int bCntLog = layers[layer] >> 1;
 int lBound = (l >> layers[layer]) << layers[layer];
 int lBlock = ((l - lBound) >> bSzLog) + 1;
 int rBlock = ((r - lBound) >> bSzLog) - 1;
 int ans = suf[layer][l];
 if (lBlock <= rBlock)
 {
 ans = op(ans, between[layer][lBound + (lBlock << bCntLog) +
rBlock]);
 }
 }

 }

}

ans = op(ans, pref[layer][r]);
return ans;
}

SqrtTree(const vector<int>& v
: n((int)v.size()), llg(log2Up(n)), v(v), clz(1 << llg), onLayer(llg+1)
{
 clz[0] = 0;
 for (int i = 1; i < (int)clz.size(); i++)
 {
 clz[i] = clz[i >> 1] + 1;
 }
 int tllg = llg;
 while (tllg > 1)
 {
 onLayer[tllg] = (int)layers.size();
 layers.push_back(tllg);
 tllg = (tllg+1) >> 1;
 }
 for (int i = llg-1; i >= 0; i--)
 {
 onLayer[i] = max(onLayer[i], onLayer[i+1]);
 }
 pref.assign(layers.size(), vector<int>(n));
 suf.assign(layers.size(), vector<int>(n));
 between.assign(layers.size(), vector<int>(1 << llg));
 build(0, 0, n);
}
};

}

```

### 37. DSU on tree

```
// how many node have color u in subtree of u
vll g[N];
ll ans[N], col[N], sz[N], cnt[N];
bool big[N];
void dfs(ll u, ll pre)
{
 sz[u] = 1;
 for(auto v:g[u]){
 if(v==pre) continue;
 dfs(v,u);
 sz[u] += sz[v];
 }
}
void add(ll u, ll pre, ll x)
{
 cnt[col[u]] += x;
 for(auto v:g[u]){
 if(v==pre || big[v]==1) continue;
 add(v,u,x);
 }
}
void dsu(ll u, ll pre, bool keep)
{
 ll bigchild=-1, mx=-1;
 for(auto v:g[u]){
 if(v==pre) continue;
 if(sz[v]>mx) mx=sz[v], bigchild=v;
 }
}
```

```
}
for(auto v:g[u]){
 if(v==pre || v==bigchild) continue;
 dsu(v,u,0);
}
if(bigchild!=-1) dsu(bigchild,u,1), big[bigchild]=1;
add(u,pre,1);
ans[u]=cnt[u];
if(bigchild!=-1) big[bigchild]=0;
if(keep==0) add(u,pre,-1);
}
int main()
{
 fast;
 ll i,j,k,n,m,u,v;
 cin>>n;
 for(i=1;i<=n;i++) cin>>col[i];
 for(i=1;i<n;i++) cin>>u>>v, g[u].pb(v), g[v].pb(u);
 dfs(1,0);
 dsu(1,0,1);
 for(i=1;i<=n;i++) cout<<ans[i]<<nl;
 return 0;
}
```

### 38. Centroid Decomposition

#### Notes

We pick the centroid as the root  $r$  and find the number of paths passing through  $r$ . Then, the other paths won't pass through  $r$ , so we can remove  $r$  and split the tree into more subtrees, and recursively

solve for each subtree as well. This is the basic solution relating to all pair nodes type problems. Sample code is in problem variation 2.

When we cannot specifically get the answer for paths passing through r but can answer for all pair of paths of subtree r then we can answer those question in the way of problem variation 3.

And problems like closest to some node can be solved in the way of problem variation 1.

### Problem Variation 1

```
//root node is red, find closest red node from a node, all nodes can
be blue or red after every update
```

```
vll g[N];
ll cenpar[N],sz[N],subtree_sz,dep[N],par[N][20],ans[N];
bool done[N];
//preprocessing part
void dfs(ll u,ll pre)
{
 dep[u]=dep[pre]+1;
 par[u][0]=pre;
 for(auto v:g[u]){
 if(v==pre) continue;
 dfs(v,u);
 }
}
ll lca(ll u,ll v)
{
 if(dep[u]<dep[v]) swap(u,v);
 for(ll k=19;k>=0;k--) if(dep[par[u][k]]>=dep[v]) u=par[u][k];
 if(u==v) return u;
 for(ll k=19;k>=0;k--) if(par[u][k]!=par[v][k]) u=par[u][k],v=par[v][k];
```

```
return par[u][0];
}
ll dist(ll u,ll v)
{
 return dep[u]+dep[v]-2*dep[lca(u,v)];
}
//Decomposition part
void set_subtree_size(ll u,ll pre)
{
 subtree_sz++;
 sz[u]=1;
 for(auto v:g[u]){
 if(v==pre || done[v]) continue;
 set_subtree_size(v,u);
 sz[u]+=sz[v];
 }
}
ll get_centroid(ll u,ll pre)
{
 for(auto v:g[u]){
 if(v==pre || done[v]) continue;
 else if(sz[v]>subtree_sz/2) return get_centroid(v,u);
 }
 return u;
}
void decompose(ll u,ll pre)
{
 subtree_sz=0;
 set_subtree_size(u,pre);
 ll centroid=get_centroid(u,pre);
```

```

cenpar[centroid]=pre;
done[centroid]=1;
for(auto v:g[centroid]){
 if(v==pre || done[v]) continue;
 decompose(v,centroid);
}
//query part
void upd(II x)
{
 II u=x;
 while(x){
 ans[x]=min(ans[x],dist(u,x));
 x=cenpar[x];
 }
}
II query(II x)
{
 II ret=1e9,u=x;
 while(x){
 ret=min(ret,ans[x]+dist(u,x));
 x=cenpar[x];
 }
 return ret;
}
int main()
{
 fast;
 II i,j,k,n,m,q,x,u,v,typ;
 cin>>n>>q;
}

```

```

for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
dfs(1,0);
for(k=1;k<20;k++) for(i=1;i<=n;i++) par[i][k]=par[par[i][k-1]][k-1];
decompose(1,0);
//make sure to set ans as INF
for(i=0;i<=n;i++) ans[i]=1e9;
upd(1);
while(q--){
 cin>>typ>>x;
 if(typ==1) upd(x);
 else cout<<query(x)<<nl;
}
return 0;
}

```

### Problem Variation 2

```

//Given a tree and values of the nodes find the sum of all pair xor
sum of nodes in the tree
vi g[N];
int cenpar[N],sz[N],subtree_sz;
bool done[N];

void set_subtree_size(int u,int pre)
{
 subtree_sz++;
 sz[u]=1;
 for(auto v:g[u]){
 if(v==pre || done[v]) continue;
 set_subtree_size(v,u);
 sz[u]+=sz[v];
 }
}

```

```

}

int get_centroid(int u,int pre)
{
 for(auto v:g[u]){
 if(v==pre || done[v]) continue;
 else if(sz[v]>subtree_sz/2) return get_centroid(v,u);
 }
 return u;
}
vi vec;
int a[N];
void dfs(int u,int pre,int x)
{
 vec.eb(a[u]^x);
 for(auto v:g[u]){
 if(v==pre || done[v]) continue;
 dfs(v,u,a[u]^x);
 }
}
int one[30];
ll solve(int u,int pre)
{
 mem(one,0);
 for(int i=0;i<25;i++) if((a[u]>>i)&1) one[i]++;
 int tot=1;
 ll ans=0;
 for(auto v:g[u]){
 if(v==pre || done[v]) continue;
 vec.clear();
 dfs(v,u,0);
 }
}

```

```

for(auto x:vec){
 for(int i=0;i<25;i++){
 if((x>>i)&1) ans+=1LL*(tot-one[i])*(1<<i);
 else ans+=1LL*one[i]*(1<<i);
 }
}
for(auto x:vec){
 x^=a[u];
 for(int i=0;i<25;i++){
 if((x>>i)&1) one[i]++;
 }
 tot++;
}
//add answer for u to u path
return ans+a[u];
}
ll decompose(int u,int pre)
{
 subtree_sz=0;
 set_subtree_size(u,pre);
 int centroid=get_centroid(u,pre);
 cenpar[centroid]=pre;
 done[centroid]=1;
 ll ans=solve(centroid,pre);
 for(auto v:g[centroid]){
 if(v==pre || done[v]) continue;
 ans+=decompose(v,centroid);
 }
 return ans;
}

```

```

}

int main()
{
 fast;
 int i,j,k,n,m,q,x,u,v,typ;
 cin>>n;
 for(i=1;i<=n;i++) cin>>a[i];
 for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
 cout<<decompose(1,0)<<n;
 return 0;
}

```

### Problem Variation 3

```

//number of paths havinf length<=mxlen and weight<=mxw
vp ii g[N];
int cenpar[N],sz[N],subtree_sz;
bool done[N];

void set_subtree_size(int u,int pre)
{
 subtree_sz++;
 sz[u]=1;
 for(auto x:g[u]){
 int v=x.F;
 if(v==pre || done[v]) continue;
 set_subtree_size(v,u);
 sz[u]+=sz[v];
 }
}
int get_centroid(int u,int pre)

```

```

{
 for(auto x:g[u]){
 int v=x.F;
 if(v==pre || done[v]) continue;
 else if(sz[v]>subtree_sz/2) return get_centroid(v,u);
 }
 return u;
}
vpii vec;
void dfs(int u,int pre,int len,int w)
{
 vec.eb(w,len);
 for(auto x:g[u]){
 int v=x.F,we=x.S;
 if(v==pre || done[v]) continue;
 dfs(v,u,len+1,w+we);
 }
}
template <class T>
struct BIT
{
 ///1-indexed
 int sz;
 vector<T> t;

 void init(int n) ///max size of array
 {
 sz = n;
 t.assign(sz,0);
 }
}
```

```

T query(int idx)
{
 T ans = 0;
 for(; idx >= 1; idx -= (idx & -idx)) ans += t[idx];
 return ans;
}

void upd(int idx, T val)
{
 if(idx <= 0) return;
 for(; idx <sz; idx += (idx & -idx)) t[idx] += val;
}

T query(int l, int r) { return query(r) - query(l - 1); }

};

BIT<int>t;
int mxlen,mxw;
ll solve(int u,int pre,int len,int w)
{
 vec.clear();
 dfs(u,pre,len,w);
 ll ans=0;
 srt(vec);
 for(auto x:vec) t.upd(x.S+1,1);
 //how many pairs of sum <=(mxw,mxlen)
 int l=0,r=vec.size()-1;
 while(l<=r){
 t.upd(vec[l].S+1,-1);
 while(l<r&&vec[l].F+vec[r].F>mxw) t.upd(vec[r].S+1,-1),r--;
 }
}

```

```

ans+=t.query(mxlen-vec[l].S+1);
l++;
}
return ans;
}
ll ans;
void decompose(int u,int pre)
{
 subtree_sz=0;
 set_subtree_size(u,pre);
 int centroid=get_centroid(u,pre);
 cenpar[centroid]=pre;
 done[centroid]=1;
 for(auto x:g[centroid]){
 int v=x.F;
 if(v==pre || done[v]) continue;
 decompose(v,centroid);
 }
 //add answer for all pair of paths from this subtree
 ans+=solve(centroid,pre,0,0);
 for(auto x:g[centroid]){
 int v=x.F;
 if(v==pre || done[v]) continue;
 //remove answer for all pair of paths from centroid's child's
 subtree
 ans-=solve(v,centroid,1,x.S);
 }
 done[centroid]=0;
}
int main()

```

```
{
fast;
int i,j,k,n,m,q,x,u,v,w,typ;
t.init(N);
cin>>n>>mxlen>>mxw;
for(i=2;i<=n;i++) cin>>u>>w,g[u].eb(i,w),g[i].eb(u,w);
decompose(1,0);
cout<<ans<<nl;
return 0;
}
```

## 39. Heavy Light Decomposition

### HLD Standard

```
//query on change a node value and sum of the path u to v
const int LG=16;
int a[N],total_chain,T,node[N],pos[N];
int par[N][LG+1],son[N],chain_head[N],sz[N],dep[N],chain_no[N];
vi g[N];
struct segtree
{
 int t[4*N];
 void build(int n,int b,int e)
 {
 if(b==e){
 t[n]=a[node[b]];
 return;
 }
 }
}
```

```
int stree;
build(l,b,mid);
build(r,mid+1,e);
t[n]=t[l]+t[r];
}
void upd(int n,int b,int e,int i,int x)
{
 if(b>i || e<i) return;
 if(b==e&&b==i){
 t[n]=x;
 return;
 }
 int stree;
 upd(l,b,mid,i,x);
 upd(r,mid+1,e,i,x);
 t[n]=t[l]+t[r];
}
int query(int n,int b,int e,int i,int j)
{
 if(b>j || e<i) return 0;
 if(b>=i&&e<=j) return t[n];
 int stree;
 int a=query(l,b,mid,i,j);
 int p=query(r,mid+1,e,i,j);
 return a+p;
}
}t;

void dfs(int u,int pre)
```

```

{
 dep[u]=dep[pre]+1;
 int mx=-1;
 sz[u]=1;
 par[u][0]=pre;
 for(int i=1;i<=LG;i++) par[u][i]=par[par[u][i-1]][i-1];
 for(auto v:g[u]){
 if(v==pre) continue;
 dfs(v,u);
 sz[u]+=sz[v];
 if(sz[v]>mx) mx=sz[v],son[u]=v;
 }
}
int lca(int u,int v)
{
 if(dep[u]<dep[v]) swap(u,v);
 for(int k=LG;k>=0;k--) if(dep[par[u][k]]>=dep[v]) u=par[u][k];
 if(u==v) return u;
 for(int k=LG;k>=0;k--) if(par[u][k]!=par[v][k])
 u=par[u][k],v=par[v][k];
 return par[u][0];
}
int kth(int u,int k)
{
 assert(k>=0);
 for(int i=0;i<=LG;i++) if(k&(1<<i)) u=par[u][i];
 return u;
}
int dist(int u,int v)

```

```

{
 int lc=lca(u,v);
 return dep[u]+dep[v]-2*dep[lc];
}
//kth node from u to v,0th node is u
int go(int u,int v,int k)
{
 int lc=lca(u,v);
 int d=dep[u]+dep[v]-2*dep[lc];
 assert(k<=d);
 if(dep[lc]+k<=dep[u]) return kth(u,k);
 k-=dep[u]-dep[lc];
 return kth(v,dep[v]-dep[lc]-k);
}
void hld(int u,int pre)
{
 if(chain_head[total_chain]==-1) chain_head[total_chain]=u;
 pos[u]=++T;
 node[T]=u;
 chain_no[u]=total_chain;
 if(son[u]==-1) return;
 hld(son[u],u);
 for(auto v:g[u]){
 if(v==pre || v==son[u]) continue;
 total_chain++;
 hld(v,u);
 }
}
//v is an ancestor of u

```

```

int query_up(int u,int v)
{
 int ans=0;
 int chain1=chain_no[u];
 int chain2=chain_no[v];
 int chd_u=chain_head[chain1];
 int chd_v=chain_head[chain2];
 while(chd_u!=chd_v){
 ans+=t.query(1,1,T,pos[chd_u],pos[u]);///queries should be
from low depth node to high depth node
 u=par[chd_u][0];
 chain1=chain_no[u];
 chd_u=chain_head[chain1];
 }
 ans+=t.query(1,1,T,pos[v],pos[u]);
 return ans;
}
int query(int u,int v)
{
 int lc=lca(u,v);
 int ans=query_up(u,lc);
 if(v!=lc) ans+=query_up(v,kth(v,dep[v]-dep[lc]-1));
 return ans;
}
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v,q,tt,w;
 cin>>n;
}

```

```

for(i=1;i<=n;i++) cin>>a[i];
for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
mem(son,-1);
mem(chain_head,-1);
dfs(1,0);
hld(1,0);
t.build(1,1,T);
cin>>q;
while(q--){
 cin>>tt;
 if(tt==1){
 cin>>u>>w;
 a[u]=w;
 t.upd(1,1,T,pos[u],w);
 }
 else{
 cin>>u>>v;
 cout<<query(u,v)<<nl;
 }
}
return 0;
}

```

### HLD with Subtrees and Path Query

```

///add t value: Add value to all nodes in subtree rooted at t
///max a b: Report maximum value on the path from a to b
ll a[mxn],t[mxn*4],lazy[mxn*4],ind,node[mxn],st[mxn],en[mxn];
ll par[mxn],son[mxn],sz[mxn],dep[mxn],head[mxn];
vll g[mxn];
void build(ll n,ll b,ll e)

```

```

{
 if(b==e){
 t[n]=a[node[b]];
 return;
 }
 ll mid=(b+e)/2,l=2*n,r=2*n+1;
 build(l,b,mid);
 build(r,mid+1,e);
 t[n]=max(t[l],t[r]);
}
void propagate(ll n)
{
 if(lazy[n]==0) return;
 t[2*n]+=lazy[n];
 t[2*n+1]+=lazy[n];
 lazy[2*n]+=lazy[n];
 lazy[2*n+1]+=lazy[n];
 lazy[n]=0;
}
void upd(ll n,ll b,ll e,ll i,ll j,ll val)
{
 if(b>j || i>e) return;
 if(b>=i && e<=j){
 t[n]+=val;
 lazy[n]+=val;
 return;
 }
 propagate(n);
 ll mid=(b+e)/2,l=2*n,r=2*n+1;
 upd(l,b,mid,i,j,val);
 upd(r,mid+1,e,j,val);
 t[n]=max(t[l],t[r]);
}
ll query(ll n,ll b,ll e,ll i,ll j)
{
 if(b>j || i>e) return -1e18;
 if(b>=i & e<=j) return t[n];
 propagate(n);
 ll mid=(b+e)/2,l=2*n,r=2*n+1;
 return max(query(l,b,mid,i,j),query(r,mid+1,e,i,j));
}
void dfs(ll u,ll pre)
{
 par[u]=pre;
 dep[u]=dep[pre]+1;
 sz[u]=1;
 for(auto v:g[u]){
 if(v==pre) continue;
 dfs(v,u);
 sz[u]+=sz[v];
 if(sz[v]>sz[son[u]]) son[u]=v;
 }
}
void hld(ll u,ll pre)
{
 st[u]=++ind;
 node[st[u]]=u;
 if(son[par[u]]==u) head[u]=head[par[u]];
 else head[u]=u;
 if(son[u]) hld(son[u],u);
}

```

```

for(auto v:g[u]){
 if(v==pre || v==son[u]) continue;
 hld(v,u);
}
en[u]=ind;
}
|| solve(|| u,|| v)
{
 || ans=-1e18;
 while(head[u]!=head[v]){
 if(dep[head[u]]<dep[head[v]]){
 swap(u,v);
 }
 ans=max(ans,query(1,1,ind,st[head[u]],st[u]));///path query
 u=par[head[u]];
 }
 if(dep[u]>dep[v]) swap(u,v);
 ans=max(ans,query(1,1,ind,st[u],st[v]));
 return ans;
}
int main()
{
 fast;
 || i,j,k,n,m,u,v,q,w;
 string tt;
 cin>>n;
 for(i=1;<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
 dfs(1,0);
 hld(1,0);
 // build(1,1,ind);
}

```

```

cin>>q;
while(q--){
 cin>>tt;
 if(tt[0]=='a'){
 cin>>u>>w;
 upd(1,1,ind,st[u],en[u],w);///subtree query
 }
 else{
 cin>>u>>v;
 cout<<solve(u,v)<<nl;
 }
}
return 0;
}

```

## 40. Treap

```

random_device rd;
mt19937 random(rd());///random generator
///If some compiler throws compilation error then
///use natural rand() function instead of mt19937

struct treap
{
 ///This is an implicit treap which investigates here on an array
 struct node
 {
 int val, sz, priority, lazy, sum, mx, mn, repl;
 bool repl_flag, rev;
 node *l, *r, *par;
 }
}

```

```

node() { lazy = 0; rev = 0; val = 0; sum=0;sz =
0;mx=0;mn=0;repl=0;repl_flag=0; priority = 0; l = NULL; r = NULL; par
= NULL; }

node(int _val)
{
 val = _val;
 sum = _val;
 mx=_val;
 mn=_val;
 repl=0;
 repl_flag=0;
 rev = 0;
 lazy = 0;
 sz = 1;
 priority = random();
 l = NULL;
 r = NULL;
 par = NULL;
}
};

typedef node* pnode;
pnode root;
map<int, pnode> position;///positions of all the values
///clearing the treap
void clear()
{
 root = NULL;
 position.clear();
}

```

```

treap() { clear(); }

int size(pnode t) { return t ? t->sz : 0; }
void update_size(pnode &t) { if(t) t->sz = size(t->l) + size(t->r)
+ 1; }

void update_parent(pnode &t)
{
 if(!t) return;
 if(t->l) t->l->par = t;
 if(t->r) t->r->par = t;
}
//add operation
void lazy_sum_upd(pnode &t){
 if(!t or !t->lazy) return;
 t->sum+=t->lazy*size(t);
 t->val += t->lazy;
 t->mx += t->lazy;
 t->mn+=t->lazy;
 if(t->l) t->l->lazy += t->lazy;
 if(t->r) t->r->lazy += t->lazy;
 t->lazy = 0;
}

///replace update
void lazy_repl_upd(pnode &t){
 if(!t or !t->repl_flag) return;
 t->val = t->mx =t->mn= t->repl;
 t->sum = t->val*size(t);
}

```

```

if(t->l){
 t->l->repl = t->repl;
 t->l->repl_flag = true;
}
if(t->r){
 t->r->repl = t->repl;
 t->r->repl_flag = true;
}
t->repl_flag = false;
t->repl = 0;
}
//reverse update
void lazy_rev_upd(pnode &t){
 if(!t or !t->rev) return;
 t->rev = false;
 swap(t->l, t->r);
 if(t->l) t->l->rev ^= true;
 if(t->r) t->r->rev ^= true;
}
//reset the value of current node assuming it now
//represents a single element of the array
void reset(pnode &t)
{
 if(!t) return;
 t->sum = t->val;
 t->mx=t->val;
 t->mn=t->val;
}

```

```

//combine node l and r to form t by updating corresponding
queries
void combine(pnode &t, pnode l, pnode r)
{
 if(!l) { t = r; return; }
 if(!r) { t = l; return; }
 //Beware!!!Here t can be equal to l or r anytime
 //i.e. t and (l or r) is representing same node
 //so operation is needed to be done carefully
 //e.g. if t and r are same then after t->sum=l->sum+r-
>sum operation,
 //r->sum will be same as t->sum
 //so BE CAREFUL
 t->sum = l->sum + r->sum;
 t->mx=max(l->mx,r->mx);
 t->mn=min(l->mn,r->mn);
}
//perform all operations
void operation(pnode &t)
{
 if(!t) return;
 reset(t);
 lazy_rev_upd(t->l);
 lazy_rev_upd(t->r);
 lazy_repl_upd(t->l);
 lazy_repl_upd(t->r);
 lazy_sum_upd(t->l);
 lazy_sum_upd(t->r);
 combine(t, t->l, t);
}
```

```

combine(t, t, t->r);
}

//split node t in l and r by key k
///so first k+1 elements(0,1,2,...k) of the array from node t
///will be splitted in left node and rest will be in right node
void split(pnode t, pnode &l, pnode &r, int k, int add = 0)
{
 if(t == NULL) { l = NULL; r = NULL; return; }

 lazy_rev_upd(t);
 lazy_repl_upd(t);
 lazy_sum_upd(t);

 int idx = add + size(t->l);
 if(idx <= k)
 split(t->r, t->r, r, k, idx + 1), l = t;
 else
 split(t->l, l, t->l, k, add), r = t;

 update_parent(t);
 update_size(t);
 operation(t);
}

///merge node l with r in t
void merge(pnode &t, pnode l, pnode r)
{
 lazy_rev_upd(l);
 lazy_rev_upd(r);
 lazy_repl_upd(l);
 lazy_repl_upd(r);
 lazy_sum_upd(l);
}

```

```

lazy_sum_upd(r);
if(!l) { t = r; return; }
if(!r) { t = l; return; }

if(l->priority > r->priority)
 merge(l->r, l->r, r), t = l;
else
 merge(r->l, l, r->l), t = r;

update_parent(t);
update_size(t);
operation(t);

}

///insert val in position a[pos]
///so all previous values from pos to last will be right shifted
void insert(int pos, int val)
{
 if(root == NULL)
 {
 pnode to_add = new node(val);
 root = to_add;
 position[val] = root;
 return;
 }

 pnode l, r, mid;
 mid = new node(val);
 position[val] = mid;
}

```

```

 split(root, l, r, pos - 1);
 merge(l, l, mid);
 merge(root, l, r);
 }
 //erase from qL to qR indexes
 //so all previous indexes from qR+1 to last will be left shifted
qR-qL+1 times
void erase(int qL, int qR)
{
 pnode l, r, mid;

 split(root, l, r, qL - 1);
 split(r, mid, r, qR - qL);
 merge(root, l, r);
}
//returns answer for corresponding types of query
int query(int qL, int qR)
{
 pnode l, r, mid;

 split(root, l, r, qL - 1);
 split(r, mid, r, qR - qL);

 int answer = mid->sum;///for sum query
 //int answer=mid->mx;///for max query
 //int answer=mid->mn;///for min query
 merge(r, mid, r);
 merge(root, l, r);
}

```

```

 return answer;
 }
 //add val in all the values from a[qL] to a[qR] positions
void update(int qL, int qR, int val)
{
 pnode l, r, mid;

 split(root, l, r, qL - 1);
 split(r, mid, r, qR - qL);
 lazy_repl_upd(mid);
 mid->lazy += val;

 merge(r, mid, r);
 merge(root, l, r);
}
//reverse all the values from qL to qR
void reverse(int qL, int qR)
{
 pnode l, r, mid;

 split(root, l, r, qL - 1);
 split(r, mid, r, qR - qL);

 mid->rev ^= 1;
 merge(r, mid, r);
 merge(root, l, r);
}
//replace all the values from a[qL] to a[qR] by v
void replace(int qL, int qR, int v)
{
}
```

```

{
 pnode l, r, mid;

 split(root, l, r, qL - 1);
 split(r, mid, r, qR - qL);
 lazy_sum_upd(mid);
 mid->repl_flag=1;
 mid->repl=v;
 merge(r, mid, r);
 merge(root, l, r);
}

//it will cyclic right shift the array k times
//so for k=1, a[qL]=a[qR] and all positions from qL+1 to qR will
//have values from previous a[qL] to a[qR-1]
//if you make left_shift=1 then it will do the opposite
void cyclic_shift(int qL, int qR, int k, bool left_shift=0)
{
 if(qL == qR) return;
 k %= (qR - qL + 1);

 pnode l, r, mid, fh, sh;
 split(root, l, r, qL - 1);
 split(r, mid, r, qR - qL);

 if(left_shift==0) split(mid, fh, sh, (qR - qL + 1) - k - 1);
 else split(mid, fh, sh, k-1);
 merge(mid, sh, fh);

 merge(r, mid, r);
}

```

```

merge(root, l, r);
}

bool exist;
//returns index of node curr
int get_pos(pnode curr, pnode son = nullptr)
{
 if(exist==0) return 0;
 if(curr==NULL){
 exist=0;
 return 0;
 }
 if(!son)
 {
 if(curr == root) return size(curr->l);
 else return size(curr->l) + get_pos(curr->par,
curr);
 }
 if(curr == root)
 {
 if(son == curr->l) return 0;
 else return size(curr->l) + 1;
 }
 if(curr->l == son) return get_pos(curr->par, curr);
 else return get_pos(curr->par, curr) + size(curr->l) + 1;
}
//returns index of the value
//if the value has multiple positions then it will

```

```

//return the last index where it was added last time
//returns -1 if it doesn't exist in the array
int get_pos(int value)
{
 if(position.find(value)==position.end()) return -1;
 exist=1;
 int x=get_pos(position[value]);
 if(exist==0) return -1;
 else return x;
}
//returns value of index pos
int get_val(int pos)
{
 return query(pos,pos);
}
//returns size of the treap
int size()
{
 return size(root);
}
///inorder traversal to get indexes chronologically
void inorder(pnode cur)
{
 if(cur==NULL) return;
 inorder(cur->l);
 cout<<cur->val<<' ';
 inorder(cur->r);
}
///print current array values serially

```

```

void print_array()
{
 // for(int i=0;i<size();i++) cout<<get_val(i)<<' ';
 // cout<<nl;
 inorder(root);
 cout<<nl;
}
bool find(int val)
{
 if(get_pos(val)==-1) return 0;
 else return 1;
}
treap t;
///Beware!!!here treap is 0-indexed

int main()
{
 BeatMeScanf;
 int i,j,k,n,m,l,r,q;
 for(i=0;i<10;i++) t.insert(i,i*10);
 t.cyclic_shift(4,5,1);
 t.update(2,5,1);
 t.replace(2,5,100);
 t.reverse(2,9);
 t.replace(2,5,200);
 cout<<t.query(0,7)<<nl;
 t.cyclic_shift(2,3,2,1);
}

```

```

cout<<t.get_pos(20)<<nl;
t.erase(2,2);
cout<<t.find(30)<<nl;
t.print_array();
return 0;
}

```

## 41. Wavelet Tree

```

const int MAXN = (int)3e5+9;
const int MAXV = (int)1e9+9;//maximum value of any element in
array

//array values can be negative too, use appropriate minimum and
maximum value
struct wavelet_tree
{
 int lo, hi;
 wavelet_tree *l, *r;
 int *b, *c, bsz, csz;/// c holds the prefix sum of elements

 wavelet_tree() { lo = 1; hi = 0; bsz = 0; csz=0, l = NULL; r = NULL;
}

 void init(int *from, int *to, int x, int y)
 {
 lo = x, hi = y;
 if(from >= to) return;
 int mid = (lo + hi) >> 1; auto f = [mid](int x) { return x
<= mid; };

```

```

 b = (int*)malloc((to - from + 2) * sizeof(int)); bsz = 0;
 b[bsz++] = 0;
 c = (int*)malloc((to - from + 2) * sizeof(int)); csz = 0;
 c[csz++] = 0;
 for(auto it = from; it != to; it++){
 b[bsz] = (b[bsz - 1] + f(*it));
 c[csz] = (c[csz - 1] + (*it));
 bsz++;
 csz++;
 }
 if(hi==lo) return;
 auto pivot = stable_partition(from, to, f);
 l = new wavelet_tree();
 l->init(from, pivot, lo, mid);
 r = new wavelet_tree();
 r->init(pivot, to, mid+1, hi);
 }
 ///kth smallest element in [l, r]
 ///for array [1,2,1,3,5] 2nd smallest is 1 and 3rd smallest is 2
 int kth(int l, int r, int k)
 {
 if(l > r) return 0;
 if(lo == hi) return lo;
 int inLeft = b[r] - b[l - 1], lb = b[l - 1], rb = b[r];
 if(k <= inLeft) return this->l->kth(lb + 1, rb, k);
 return this->r->kth(l - lb, r - rb, k - inLeft);
 }
 ///count of numbers in [l, r] Less than or equal to k
 int LTE(int l, int r, int k)

```

```

{
 if(l > r || k < lo) return 0;
 if(hi <= k) return r - l + 1;
 int lb = b[l - 1], rb = b[r];
 return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r -
rb, k);
}
///count of numbers in [l, r] equal to k
int count(int l, int r, int k)
{
 if(l > r || k < lo || k > hi) return 0;
 if(lo == hi) return r - l + 1;
 int lb = b[l - 1], rb = b[r];
 int mid = (lo + hi) >> 1;
 if(k <= mid) return this->l->count(lb + 1, rb, k);
 return this->r->count(l - lb, r - rb, k);
}
///sum of numbers in [l ,r] less than or equal to k
int sum(int l, int r, int k) {
 if(l > r or k < lo) return 0;
 if(hi <= k) return c[r] - c[l-1];
 int lb = b[l-1], rb = b[r];
 return this->l->sum(lb+1, rb, k) + this->r->sum(l-lb, r-
rb, k);
}
~wavelet_tree()
{
 delete l;
 delete r;
}

```

```

}
wavelet_tree t;
int a[N];
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,q,l,r;
 cin>>n;
 for(i=1;i<=n;i++) cin>>a[i];
 t.init(a+1,a+n+1,-MAXV,MAXV);
 cin >> q;
 while(q--){
 int x;
 cin>>x;
 cin >> l >> r >> k;
 if(x == 0){
 ///kth smallest
 cout << t.kth(l, r, k) << endl;
 }
 else if(x == 1){
 ///less than or equal to K
 cout << t.LTE(l, r, k) << endl;
 }
 else if(x == 2){
 ///count occurence of K in [l, r]
 cout << t.count(l, r, k) << endl;
 }
 if(x == 3){

```

```

 //sum of elements less than or equal to K in [l,
r]
 cout << t.sum(l, r, k) << endl;
}
return 0;
}

```

## 42. K-D tree

```

///Complexity: O(log n)
///Average Complexity: O(3d log n), where d is dimension
///Works for random points
///search for nearest point which has minimum euclidean distance
///from this point
const long long INF = 20000000000000000000000000000007;
const int d=2;//dimension

struct point {
 int p[d];
 bool operator !=(const point &a) const {
 bool ok=1;
 for(int i=0;i<d;i++) ok&=(p[i]==a.p[i]);
 return !ok;
 }
};

struct kd_node {
 int axis,value;
 point p;

```

```

 kd_node *left, *right;
};

struct cmp_points {
 int axis;
 cmp_points(){}
 cmp_points(int x): axis(x) {}
 bool operator () (const point &a, const point &b) const {
 return a.p[axis]<b.p[axis];
 }
};

typedef kd_node* node_ptr;

int tests,n;
point arr[N],pts[N];
node_ptr root;
long long ans;

long long squared_distance(point a, point b) {
 long long ans=0;
 for(int i=0;i<d;i++) ans+=(a.p[i]-b.p[i])*1||*(a.p[i]-b.p[i]);
 return ans;
}

void build_tree(node_ptr &node, int from, int to, int axis) {
 if(from>to) {
 node=NULL;
 return;
 }
}
```

```

}

node=new kd_node();

if(from==to) {
 node->p=arr[from];
 node->left=NULL;
 node->right=NULL;
 return;
}

int mid=(from+to)/2;

nth_element(arr+from,arr+mid,arr+to+1,cmp_points(axis));
node->value=arr[mid].p[axis];
node->axis=axis;
build_tree(node->left,from,mid,(axis+1)%d);
build_tree(node->right,mid+1,to,(axis+1)%d);
}

void nearest_neighbor(node_ptr node, point q, long long &ans) {
 if(node==NULL) return;

 if(node->left==NULL && node->right==NULL) {
 if(q!=node->p) ans=min(ans,squared_distance(node->p,q));///Beware!!!need to take care here
 return;
 }
}

```

```

if(q.p[node->axis]<=node->value) {
 nearest_neighbor(node->left,q,ans);
 if(q.p[node->axis]+sqrt(ans)>=node->value)
nearest_neighbor(node->right,q,ans);
}

else {
 nearest_neighbor(node->right,q,ans);
 if(q.p[node->axis]-sqrt(ans)<=node->value)
nearest_neighbor(node->left,q,ans);
}

int main() {
 int i,j,k,m;
 scanf("%d", &tests);
 while(tests--) {
 scanf("%d", &n);
 for(i=1;i<=n;i++) {
 for(j=0;j<d;j++) scanf("%d",&arr[i].p[j]);
 pts[i]=arr[i];
 }
 build_tree(root,1,n,0);

 for(i=1;i<=n;i++) {
 ans=INF;
 nearest_neighbor(root,pts[i],ans);
 printf("%lld\n", ans);
 }
 }
}

```

```

 }
 }

 return 0;
}

43. Link-Cut Tree

random_device rd;
mt19937_64 mt(rd());

struct node
{
 int sz, prior, id, rev;
 node *par, *pp, *l, *r;
 node() { id = 0; sz = 0; rev = 0; prior = 0; par = NULL; l = NULL;
r = NULL; pp=NULL; }
 node(int v) { id = v; sz = 1; rev = 0; prior = mt(); l = NULL; r =
NULL; par=NULL; pp=NULL; }
};

typedef node* pnode;

inline int size(pnode v) { return v ? v->sz : 0; }

void push(pnode &t)
{
 if(!t) return;
 if(t->rev)
 {
 swap(t->l, t->r);

```

```

 if(t->l) t->l->rev ^= 1;
 if(t->r) t->r->rev ^= 1;
 t->rev = 0;
 }
}

void pull(pnode &v)
{
 if(!v) return;

 push(v->l);
 push(v->r);

 v->par = NULL;
 v->sz = size(v->l) + size(v->r) + 1;

 if(v->l) v->l->par = v;
 if(v->r) v->r->par = v;

 if(v->l && v->l->pp) v->pp = v->l->pp, v->l->pp = NULL;
 if(v->r && v->r->pp) v->pp = v->r->pp, v->r->pp = NULL;
}

void merge(pnode &t, pnode l, pnode r)
{
 push(l), push(r);
 if(!l) { t = r; return; }
 if(!r) { t = l; return; }
}

```

```

if(l->prior > r->prior)
 merge(l->r, l->r, r), t = l;
else
 merge(r->l, l, r->l), t = r;

pull(t);
}

void split(pnode t, pnode &l, pnode &r, int k, int add = 0)
{
 push(t);
 if(!t) { l = NULL; r = NULL; return; }

 int idx = add + size(t->l);
 if(idx <= k)
 split(t->r, t->r, r, k, idx + 1), l = t;
 else
 split(t->l, l, t->l, k, add), r = t;

 pull(t);
}

pnode get_root(pnode t) { if(!t) return NULL; while(t->par) t = t->par;
return t; }

pnode remove_right(pnode t)
{
 pnode rt = t;
}

```

```

int pos = size(rt->l);
if(rt->rev) pos = size(rt) - pos - 1;
while(rt->par)
{
 if(rt->par->r == rt) pos += size(rt->par->l) + 1;
 if(rt->par->rev) pos = size(rt->par) - pos - 1;
 rt = rt->par;
}

pnode l, r, pp = rt->pp;
rt->pp = NULL;
split(rt, l, r, pos);

l->pp = pp;
if(r) r->pp = t;

return l;
}

pnode remove_left(pnode t)
{
 pnode rt = t;

 int pos = size(rt->l);
 if(rt->rev) pos = size(rt) - pos - 1;
 while(rt->par)
 {
 if(rt->par->r == rt) pos += size(rt->par->l) + 1;
 if(rt->par->rev) pos = size(rt->par) - pos - 1;
 }
}

```

```

 rt = rt->par;
 }

pnode l, r, pp = rt->pp;
rt->pp = NULL;
split(rt, l, r, pos - 1);

l->pp = pp;
return r;
}

pnode merge_trees(pnode u, pnode t)
{
 u = get_root(u);
 t = get_root(t);
 t->pp = NULL;
 merge(u, u, t);
 return u;
}

struct link_cut_tree
{
 pnode ver[N];

 pnode access(pnode t)
 {
 t = remove_right(t);
 while(t->pp)
 {
 pnode u = t->pp;
 u = remove_right(u);
 t = merge_trees(u, t);
 }
 return t;
 }

 pnode find_root(pnode u)
 {
 u = access(u);
 push(u); while(u->l) u = u->l, push(u);
 access(u);
 return u;
 }

 void make_root(pnode u)
 {
 u = access(u);
 u->rev ^= 1;
 push(u);
 }

 void link(pnode u, pnode w)
 {
 make_root(u);
 access(w);
 merge_trees(w, u);
 }
}

```

```

void cut(pnode p)
{
 access(p);
 remove_left(p);
}

int depth(pnode u)
{
 u = access(u);
 return size(u);
}

pnode lca(pnode u, pnode v)
{
 if(u == v) return u;
 if(depth(u) > depth(v)) swap(u, v);

 access(v);
 access(u);

 return get_root(v)->pp;
}

///creating vertices of the tree
void init(int c)
{
 for(int i = 0; i <= c; i++) ver[i] = new node(i);
}

///returns lca of two vertices
inline int lca(int u, int v)
{
 return lca(ver[u], ver[v])->id;
}

///finds the root of tree which has node u
inline int root(int u)
{
 return find_root(ver[u])->id;
}

///add an edge from vertex v to u, making u a child of v,
///where initially u and v are in different trees.
inline void link(int u, int v) ///add directed edge v to u
{
 link(ver[u], ver[v]);
}

///make u the root of its representative tree
inline void make_root(int u)
{
 make_root(ver[u]);
}

///depth of vertex u in its own tree
inline int depth(int u)
{
 return depth(ver[u]);
}

///remove edge from u to its parent, where u is a non-root vertex.
inline void cut(int u)
{
}

```

```

 cut(ver[u]);
 }
};

link_cut_tree lct;

int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v;
 cin>>n>>m;
 lct.init(n);
 while(m--)
 {
 string type;
 cin >> type;

 if(type == "add") //add an edge
 {
 int u, w;
 cin >> u >> w;
 lct.link(u, w);
 }
 else if(type == "conn") ///if u and v is connected
 {
 int u, v;
 cin >> u >> v;
 }
 }
}

```

```

cout << (lct.root(u) == lct.root(v) ? "YES" :
"NO") << endl;
}

else if(type == "rem") //remove edge
{
 int u, v;
 cin >> u >> v;
 if(lct.depth(u) > lct.depth(v)) swap(u, v);
 lct.cut(v);
}

return 0;
}

```

## 44. Static to Dynamic Trick

Assume you have some static set and you can calculate some function  $f$  of the whole set such that  $f(x_1, \dots, x_n) = g(f(x_1, \dots, x_{k-1}), f(x_k, \dots, x_n))$ , where  $g$  is some function which can be calculated fast. For example,  $f(\cdot)$  as the number of elements less than  $k$  and  $g(a, b) = a + b$ . Or  $f(S)$  as the number of occurrences of strings from  $S$  into  $T$  and  $g$  is a sum again.

With additional  $\log n$  factor you can also insert elements into your set. For this let's keep  $\log n$  disjoint sets such that their union is the whole set. Let the size of  $k^{th}$  be either  $0$  or  $2^k$  depending on binary presentation of the whole set size. Now when inserting element you should add it to  $0^{th}$  set and rebuild every set keeping said constraint. Thus  $k^{th}$  set will take  $F(2^k)$  operations each  $2^k$  steps where  $F(n)$  is

the cost of building set over  $n$  elements from scratch which is usually something about  $n$ .

For Code check the aho corasick dynamic section here. [#link](#)

## 45. Queue using two Stacks

Keep 2 stacks, let's call them `inbox` and `outbox`.

### Enqueue:

- Push the new element onto `inbox`

### Dequeue:

- If `outbox` is empty, refill it by popping each element from `inbox` and pushing it onto `outbox`
- Pop and return the top element from `outbox`

Using this method, each element will be in each stack exactly once - meaning each element will be pushed twice and popped twice, giving amortized constant time operations.

## 46. Rope

```
#include <ext/rope> //header with rope
using namespace __gnu_cxx; //namespace with rope and some
additional stuff
int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 rope<char>v;//0-indexed,just like a string
 string s;
 cin>>s;
 for(auto x:s) v.pb(x);
 cin>>m;
 while(m--){
 cout<<s[m]<<endl;
 }
}
```

```
int ty,l,r;
cin>>ty>>l;
if(ty==1){
 //Cut the rope segment from X to Y and join at the front of
 rope.
 cin>>r;
 rope<char>cur=v.substr(l,r-l+1);
 v.erase(l,r-l+1);
 v.insert(v.mutable_begin(),cur);///use mutable_begin()
instead of begin()
}
else if(ty==2){
 //Cut the rope segment from X to Y and join at the back of
 rope.
 cin>>r;
 rope<char>cur=v.substr(l,r-l+1);
 v.erase(l,r-l+1);
 v.insert(v.mutable_end(),cur);
}
else{
 //Print the Alphabet on l-th position of current rope.
 cout<<v[l]<<nl;
}
return 0;
}
```

## 47. Interval Set

//for Q assign operation it takes Qlogn time in total

```

template<class T>
struct interval_set
{
 map<pair<int, int>, T> value;///{r,l}=val

 void init(int n) { value[{n, 1}] = (T)0; }///initial value
 ///assign a[i]=val for l<=i<=r
 ///returns affected ranges before performing this assign
operation
 vector<pair<pair<int, int>, T> > assign(int l, int r, T val)
 {
 auto bg = value.lower_bound({l, 0})->first;
 if(bg.second != l)
 {
 T val = value[bg];
 value.erase(bg);
 value[{l - 1, bg.second}] = val;
 value[{bg.first, l}] = val;
 }

 auto en = value.lower_bound({r, 0})->first;
 if(en.first != r)
 {
 T val = value=en];
 value.erase(en);
 value[{en.first, r + 1}] = val;
 value[{r, en.second}] = val;
 }
 }
}

```

```

vector<pair<pair<int, int>, T> > ret;
auto itt = value.lower_bound({l, 0});
while(true)
{
 if(itt == value.end() || itt->first.first > r) break;
 ret.push_back({{itt->first.second, itt->first.first}, itt->second});
 ++itt;
}

for(auto it: ret)
 value.erase({it.first.second, it.first.first});

value[{r, l}] = val;
return ret;
};

interval_set<int>se;
///assign a value in range in each query
///in the end print the sum of the array elements
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m,q,l,r,v;
 cin>>n>>q;
 se.init(n);
 while(q--){
 cin>>l>>r>>v;
 se.assign(l,r,v);
 }
}

```

```

}
int ans=0;
for(auto x:se.value){
 ans+=1LL*((x.F.F-x.F.S+1)%mod)*(x.S%mod)%mod;
 ans%=mod;
}
cout<<ans<<n;
return 0;
}

```

## 48. Divide and Conquer for insert and query problems

```

/**
if problems have following characteristics-
1.it can be solved offline
2.if every insert operations were before query operations
each query operations could have been solved in O(L)(i mean faster
like logn)
3.query operation is cumulative i.e. we don't need every insert at
once to solve this query

```

this type of problem can be solved using divide and conquer  
complexity:  $n \log n * O(L)$

```
**/
```

Problem:

You have an empty set. You need to perform following operations

—  
Insert a given number X in the set.

Count how many numbers in the set is less than or equal to a given number X

Solution:

```

op[1...n] = operations
ans[1...n] = array to store answers
solve(l, r) {
 if op[l...r] has no queries: return
 m = (l + r) / 2
 ds = statically built DS using insert operations in op[l...m]
 for each query operation i in op[m+1...r]:
 ans[i] += ds.query(op[i])
 solve(l, m)
 solve(m+1, r)
}

```

```

/**
some dp can be solved using this trick
*/

```

Problem:

Lets have a look at the LIS problem. it has following solution-

```

for(int i = 1; i <= n; i++) {
 for(int j = 1; j < i; j++) if(a[j] < a[i])
 dp[i] = max(dp[i], dp[j] + 1)
}

```

it can modeled like update-query problem

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

```

dp[i] = query(a[i]);
insert({a[i], dp[i]});
}

```

Solution:

```

solve(l, r) {
 m = (l + r) / 2
 solve(l, m)
 make ds with a[l..m] and dp[l..m]
 update dp[m+1..r] using the ds
 solve(m+1, r)
}

```

## 49. Venice Technique

```
/**
```

We want a data structure capable of doing three main update-operations and some sort of query. The three modify operations are: add: Add an element to the set. remove: Remove an element from the set. updateAll: This one normally changes in this case subtract X from ALL the elements. For this technique it is completely required that the update is done to ALL the values in the set equally. And also for this problem in particular we may need one query: getMin: Give me the smallest number in the set.

```
*/
/**Interface of the Data Structure
```

```

struct VeniceSet {
 void add(int);
 void remove(int);
 void updateAll(int);
 int getMin(); // custom for this problem
 int size();
};
*/
/**
```

Imagine you have an empty land and the government can make queries of the following type: \* Make a building with A floors. \* Remove a building with B floors. \* Remove C floors from all the buildings. (A lot of buildings can be vanished) \* Which is the smallest standing building. (Obviously buildings which are already banished don't count)

The operations 1,2 and 4 seems very easy with a set, but the 3 is very cost effective probably O(N) so you might need a lot of workers. But what if instead of removing C floors we just fill the streets with enough water (as in venice) to cover up the first C floors of all the buildings :O. Well that seems like cheating but at least those floor are now vanished :). So in order to do that we apart from the SET we can maintain a global variable which is the water level. so in fact if we

have an element and want to know the number of floors it has we can just do  
 $\text{height} - \text{water\_level}$  and in fact after water level is for example 80, if we want to make a building of 3 floors we must make it of 83 floors so that it can touch the land.

```
*/
struct VeniceSet {
 multiset<int> S;
 int water_level = 0;
 void add(int v) {
 S.insert(v + water_level);
 }
 void remove(int v) {
 S.erase(S.find(v + water_level));
 }
 void updateAll(int v) {
 water_level += v;
 }
 int getMin() {
 return *S.begin() - water_level;
 }
 int size() {
 return S.size();
 }
};

VeniceSet se;
```

```
///add new element V[i]
///decrease T[i] from every element. If element is less than T[i] remove it
///How much will be decreased totally?
int V[N],T[N];
int main()
{
 int n;
 cin>>n;
 for(int i=0;i<n;i++) cin>>V[i]>>T[i];
 for (int i = 0; i < n; ++i) {
 se.add(V[i]);
 se.updateAll(T[i]); // decrease all by T[i]
 int total = T[i] * se.size(); // we subtracted T[i] from all elements
 // in fact some elements were already less than T[i]. So we probbaly are counting
 // more than what we really subtracted. So we look for all those elements
 while (se.size()&&se.getMin() < 0) {
 // get the negative number which we really did not subtracted
 T[i]
 int toLow = se.getMin();
 // remove from total the amount we over counted
 total -= abs(toLow);
```

```

 // remove it from the set since it will never be able to subtract
 from it again
 se.remove(toLow);
}
cout << total << endl;
}
cout << endl;
return 0;
}

```

## 50. Cartesian Tree

/\*\*

A Cartesian tree is a tree data structure created from a set of data that obeys the following structural invariants:

1. The tree obeys the min (or max) heap property – each node is less (or greater) than its children.

2. An inorder traversal of the nodes yields the values in the same order

in which they appear in the initial sequence.

in this tree  $\text{lca}(i,j)=\text{rmq}(i,i+1,\dots,j-1,j)$

$l[i]$ =left child of  $i$ , i.e. nearest value  $<i$  such that  $a[l[i]] < a[i]$  (for min heap)

$r[i]$ =right child of  $i$ , i.e. nearest value  $>i$  such that  $a[r[i]] < a[i]$  (for min heap)

\*\*/

```

int n, tot, st[N], l[N], r[N], vis[N];
ll inv[N], ans;
pii a[N];

int dfs(int u){
 int sz = 1;
 //cout<<u<<": "<<l[u]<<' '<<r[u]<<nl;
 if (l[u]) sz += dfs(l[u]);
 if (r[u]) sz += dfs(r[u]);
 //debug(sz);
 ans = ans * inv[sz] % mod;
 return sz;
}

///O(n)
///returns root of the tree
///has max heap property
int cartesian_tree(){
 tot = 0;
 for (int i = 1; i <= n; i++) l[i] = r[i] = vis[i] = 0;
 for (int i = 1; i <= n; i++) {
 int k = tot;
 while (k > 0 && a[st[k - 1]] < a[i]) k--; //use > for min heap
 if (k) r[st[k - 1]] = i;
 if (k < tot) l[i] = st[k];
 st[k++] = i;
 tot = k;
 }
 for (int i = 1; i <= n; i++) vis[l[i]] = vis[r[i]] = 1;
}

```

```

int rt = 0;
for (int i = 1; i <= n; i++) {
 if (vis[i] == 0) rt = i;
}
return rt;
}

```

```

///given an array find the probability of any array having each element
in [0,1](real numbers)
///and isomorphic to the array
///two array is isomorphic if index set(before sorting) of first array is
equal to index set(before sorting) of
///second array after sorting the numbers
///(1,2,1) and (2,3,2) is isomorphic, (1,2,1) and (3,2,3) is not
int main(){
 inv[1] = 1;
 for (int i = 2; i < N; i++) inv[i] = inv[mod%i] * (mod - mod / i) % mod;
 int t;
 scanf("%d", &t);
 while (t--) {
 scanf("%d", &n);
 for (int i = 1; i <= n; i++) {
 int x;
 scanf("%d", &x);
 a[i] = make_pair(x, -i); //for making the array elements
distinct, for min heap use (+i)
 }
 ans = 1;
 }
}

```

```

int rt = cartesian_tree();
dfs(rt);
printf("%I64d\n", ans);
}
return 0;
}

```

## DYNAMIC PROGRAMMING

### 51. Digit DP

#### Count of Numbers

```

///count of numbers x such that l<=x<=r ans distinct digit in x=max
digit in x
vll digit;
ll sz, dp[20][1111][2], cnt[1111], mx[1111];
ll yo(ll idx, ll mask, bool badha)
{
 if(idx == -1){
 if(cnt[mask] == mx[mask]) return 1;
 else return 0;
 }
 ll &ret = dp[idx][mask][badha];
 if(ret != -1 && badha != 1) return ret;
 ll ans = 0;
 ll mxhere = badha ? digit[idx] : 9;
 for(ll i = 0; i <= mxhere; i++){
 bool next_badha = (i == digit[idx] ? badha : 0);
 if(next_badha)
 ans += yo(idx + 1, mask | (1 << i), next_badha);
 }
 ret = ans;
}

```

```

ans+=yo(idx-
1,(mask==0&&i==0)?mask:mask|(1LL<<i),next_badha);
}
if(badha==0) ret=ans;
return ans;
}
|| get(|| n)
{
if(n<0) return 0;
digit.clear();
while(n) digit.pb(n%10),n/=10;
sz=digit.size();
return yo(sz-1,0,1);
}
int main()
{
fast;
|| i,j,k,n,m,l,r,t;
mem(dp,-1);
cnt[0]=1;
for(i=0;i<1111;i++){
 for(j=0;j<=10;j++) if(i&(1<<j)) cnt[i]++,mx[i]=j;
}
cin>>t;
while(t--){
 cin>>l>>r;
 cout<<get(r)-get(l-1)<<nl;
}
return 0;
}

```

```
}
```

## Sum of Numbers

```

///sum of numbers x such that l<=x<=r and distinct digit in x<=k
vll digit;
bool vis[20][1111][2];
|| sz;
pll dp[20][1111][2];
|| cnt[1111],pw[30],k;
pll yo(|| idx,|| mask,bool badha)
{
if(idx==-1){
 if(cnt[mask]<=k) return MP(0,1);
 else return MP(0,0);
}
pll &ret=dp[idx][mask][badha];
bool &x=vis[idx][mask][badha];
if(x==1&&badha!=1) return ret;
pll ans={0,0};
|| mxhere=badha?digit[idx]:9;
for(|| i=0;i<=mxhere;i++){
 bool next_badha=(i==digit[idx]?badha:0);
 pll
 p=yo(idx-
1,(mask==0&&i==0)?mask:mask|(1LL<<i),next_badha);
 if(p.S==0) continue;
 ans.F+=(p.F+pw[idx]*i%mod*p.S%mod)%mod;
 ans.F%=mod;
 ans.S+=p.S;
 ans.S%=mod;
}
}
```

```

 }
 if(badha==0) x=1,ret=ans;
 return ans;
}
ll get(ll n)
{
 if(n<0) return 0;
 digit.clear();
 while(n) digit.pb(n%10),n/=10;
 sz=digit.size();
 return yo(sz-1,0,1).F%mod;
}
int main()
{
 fast;
 ll i,j,n,m,l,r,t;
 pw[0]=1;
 for(i=1;i<30;i++) pw[i]=10LL*pw[i-1]%mod;
 cnt[0]=1;
 for(i=0;i<1111;i++){
 for(j=0;j<=10;j++) if(i&(1<<j)) cnt[i]++;
 }
 cin>>t;
 while(t--){
 cin>>l>>r>>k;
 cout<<((get(r)-get(l-1))%mod+mod)%mod<<nl;
 }
 return 0;
}

```

## 52. Convex Hull Trick

### Convex Hull Trick Standard

```

ll dp[120000];
struct cline {
 ll M, C;
 cline() {}
 cline(ll m, ll c): M(m), C(c) {}
};
int last=0,pointer=0;

//pointer=0,last=0 should be made initially
cline line[N]; //y=mx+c we need only m(slope) and c(constant)

//Returns true if either line l1 or line l3 is always better than line l2
bool bad(const cline & l1,const cline & l2,const cline & l3) {
 /*
 intersection(l1,l2) has x-coordinate (c1-c2)/(m2-m1)
 intersection(l1,l3) has x-coordinate (c1-c3)/(m3-m1)
 set the former greater than the latter, and cross-multiply to
 eliminate division
 */
 //if the query x values is non-decreasing (reverse(> sign) for vice
 //verse)
 return (double)(l3.C-l1.C)*(double)(l1.M-l2.M)<=(double)(l2.C-
 l1.C)*(double)(l1.M-l3.M);
}

//Adding should be done serially

```

```

//If we want minimum y coordinate(value) then maximum valued m
should be inserted first
//If we want maximum y coordinate(value) then minimum valued m
should be inserted first
void add(cline l) {
 //First, let's add it to the end
 line[last++]=l;
 //If the penultimate is now made irrelevant between the
 antepenultimate
 //and the ultimate, remove it. Repeat as many times as necessary
 //in short convex hull main convex hull tecnique is applied here
 while(last>=3&&bad(line[last-3],line[last-2],line[last-1])) {
 line[last-2]=line[last-1];
 last--;
 }
}

//Returns the minimum y-coordinate of any intersection between a
given vertical
//line(x) and the lower/upper envelope(pointer)
//This can only be applied if the query of vertical line(x) is already
sorted
//works better if number of query is huge
long long query(long long x) {
 //If we removed what was the best line for the previous query, then
 the
 //newly inserted line is now the best for that query
 if (pointer>=last)
 pointer=last-1;
}

```

```

//Any better line must be to the right, since query values are
//non-decreasing
// Min Value wanted... (reverse(> sign) for max value)
while (pointer<last-1) &&
line[pointer+1].M*x+line[pointer+1].C<=line[pointer].M*x+line[pointer].C
pointer++;
return line[pointer].M*x+line[pointer].C;
}

//for any kind of query(sorted or not) it can be used
//it works because of the hill property
//works better if number of query is few
long long bs(int st,int end,long long x,int last) {
 int mid=(st+end)/2;
 // Min Value wanted... (reverse(> sign) for max value)
 if(mid+1<last) &&
line[mid+1].M*x+line[mid+1].C<line[mid].M*x+line[mid].C return
bs(mid+1,end,x,last);
 // Min Value wanted... (reverse(> sign) for max value)
 if(mid-1>=0) &&
line[mid-1].M*x+line[mid-1].C<line[mid].M*x+line[mid].C return
bs(st,mid-1,x,last);
 return line[mid].M*x+line[mid].C;
}
int b[120000],ara[1200000];
int main() {
 int i,j,k,l,m,n;
 scanf("%d",&n);
 for(i=0;i<n;i++){

```

```

scanf("%d",&ara[i]);
}
for(i=0;i<n;i++)cin>>b[i];
cline gr;
ll ans=0;
gr.M=b[0];
gr.C=0;
add(gr);
for(int i=1;i<n;i++){
 ans=query(ara[i]);
 gr.M=b[i];
 gr.C=ans;
 add(gr);
}
cout<<ans<<endl;
return 0;
}

```

### Dynamic Convex Hull Trick

```

/// Keeps upper hull for maximums.
/// add lines with -m and -b and return -ans to
/// make this code working for minimums.
const ll is_query = -(1LL<<62);
struct line {
 ll m, b;
 mutable function<const line*()> succ;
 bool operator<(const line& rhs) const {
 if (rhs.b != is_query) return m < rhs.m;
 const line* s = succ();

```

```

 if (!s) return 0;
 ll x = rhs.m;
 return b - s->b < (s->m - m) * x;
 }
};

/// Dynamic Convex Hull Trick
struct CHT : public multiset<line> { /// will maintain upper hull for
maximum
 bool bad(iterator y) {
 auto z = next(y);
 if (y == begin()) {
 if (z == end()) return 0;
 return y->m == z->m && y->b <= z->b;
 }
 auto x = prev(y);
 if (z == end()) return y->m == x->m && y->b <= x->b;
 return 1.0*(x->b - y->b)*(z->m - y->m) >= 1.0*(y->b - z->b)*(y->m
- x->m);
 }
 void add(ll m, ll b) {
 auto y = insert({ m, b });
 y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
 if (bad(y)) { erase(y); return; }
 while (next(y) != end() && bad(next(y))) erase(next(y));
 while (y != begin() && bad(prev(y))) erase(prev(y));
 }
 ll query(ll x) {
 auto l = *lower_bound((line) { x, is_query });
 return l.m * x + l.b;
 }
};

```

```

 }
};

ll a[N], b[N];
CHT* x;
int main()
{
 fast;
 ll i, j, k, n, m, ans = 0;
 cin >> n;
 for(i=0; i<n; i++) cin >> a[i];
 for(i=0; i<n; i++) cin >> b[i];
 x = new CHT();
 x->add(-b[0], 0);
 for(i=1; i<n; i++){
 ans = -x->query(a[i]);
 x->add(-b[i], -ans);
 }
 cout << ans << endl;
 return 0;
}

```

## Persistent Convex Hull Trick

```

/// You can only remove last added line with this code
const ll nsz = 5e4 + 9; // maximum number of lines
ll msz; // make it 0 for restarting the CHT
ll outside = nsz - 1;
ll M[nsz], B[nsz]; // y = M*X + B formatted lines, must be sorted in
// advanced by M // clear M, B for test cases, make qptr = 0

```

```

bool bad(int l1, int l2, int l3, bool lowerPart = 1) // returns true if l1-l3
line is better than l2
{
/*
intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
*/
// cout << (B[l3]-B[l1])*(M[l1]-M[l2]) << " " << (B[l2]-B[l1])*(M[l1]-
M[l3]) << endl;
if (lowerPart == 1)
 return 1.00 * (B[l3]-B[l1])*(M[l1]-M[l2]) <= 1.00 * (B[l2]-
B[l1])*(M[l1]-M[l3]);
else return 1.00 * (B[l3]-B[l1])*(M[l1]-M[l2]) >= 1.00 * (B[l2]-
B[l1])*(M[l1]-M[l3]);
}
struct data // information to undo change in CHT
{
 ll m, b, pos;
 data(ll _m = 0, ll _b = 0, ll _pos = 0)
 {
 m = _m, b = _b, pos = _pos;
 }
};
data add(ll m, ll b, bool lowerPart = 1)
{
// lowerPart is called upper hull. For m decreasing, this creates lower
part, but if m increasing, it does reverse
// lower part is needed for finding minimum, upper part for maximum
M[outside] = m, B[outside] = b;

```

```

while (msz >= 2 && bad(msz-2, msz-1, outside, lowerPart))
{
 msz--;
}
data temp(M[msz], B[msz], msz);
M[msz] = m;
B[msz] = b;
msz++;
return temp;
}
|| query(|| x, bool findMin = 1) //online query
{
 int lo = 0, hi = msz - 1;
 || ans = LLONG_MAX;
 if (findMin)
 ans = -LLONG_MAX;
 while(lo <= hi)
 {
 int diff = (hi-lo)/3;
 int mid1 = lo + diff;
 int mid2 = hi - diff;
 || y1 = M[mid1]*x + B[mid1], y2 = M[mid2]*x + B[mid2];
 if(y1 <= y2)
 {
 ans = y1;
 if (findMin)
 hi = mid2 - 1;
 else lo = mid1 + 1;
 }
 }
}

else
{
 ans = y2;
 if (findMin)
 lo = mid1 + 1;
 else hi = mid2 - 1;
}
}
return ans;
}
|| sum[N],val[N],a,b,ans;
|| g[N];
void dfs(|| u,|| pre=0)
{
 sum[u]=val[u];
 for(auto v:g[u]){
 if(v==pre) continue;
 dfs(v,u);
 sum[u]+=sum[v];
 }
}
void yo(|| u,|| pre=0)
{
 bool leaf=1;
 for(auto v:g[u]){
 if(v==pre) continue;
 leaf=0;
 || res=query(sum[v])+a*sum[v]*sum[v]+b;
 ans=min(ans,res+a*sum[v]*sum[v]+b);
 }
}

```

```

|| prvsz=msz;
data undo=add(-2*a*sum[v],a*sum[v]*sum[v]+res,0);
yo(v,u);
msz=prvsz;
M[undo.pos]=undo.m;
B[undo.pos]=undo.b;
}
//if(leaf) ans=min(ans,query(0)+b);
}
int main()
{
 fast;
 || i,j,k,n,m,t,u,v;
 cin>>t;
 while(t--){
 cin>>n>>a>>b;
 for(i=1;i<=n;i++) cin>>val[i];
 for(i=1;i<n;i++){
 cin>>u>>v;
 g[u].eb(v);
 g[v].eb(u);
 }
 dfs(1);
 ans=a*sum[1]*sum[1]+b;
 msz=0;
 add(-2*a*sum[1],a*sum[1]*sum[1]);
 yo(1);
 cout<<ans<<nl;
 mem(sum,0);
 }
}

```

```

for(i=1;i<=n;i++) g[i].clear();
}
return 0;
}

Convex Hull Trick 2D
CHT *t;
|| dp[110][N];///dp[i][j]=minimum cost for dividing [1...j] in i parts
|| a[N],d[N],h[N],ti[N],s[N];
|| cost(int i,int j)
{
 return a[j]*(j-i+1)-(s[j]-s[i-1]);
}
int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin>>n>>m>>k;
 for(i=2;i<=n;i++) cin>>d[i],d[i]+=d[i-1];
 for(i=1;i<=m;i++) cin>>h[i]>>ti[i];
 for(i=1;i<=m;i++) a[i]=ti[i]-d[h[i]],s[i]=s[i-1]+a[i];
 sort(a+1,a+m+1);
 ///dp[k][j]=min(dp[k-1][i]+a[j]*(j-i)-(s[j]-s[i]))
 ///Beware!!!!!
 /// dp[k][j]=min(dp[k-1][i]+cost(i+1,j)) not cost(i,j)
 for(i=1;i<=m;i++) dp[1][i]=cost(1,i);
 ///solution for exactly k partition or is it?
 for(int kk=2;kk<=k;kk++){
 t=new CHT();

```

```

for(int i=1;i<=m;i++){
 t->add(i,-dp[kk-1][i]-s[i]);///add line as dp[kk-1][i] is a constant
now
 dp[kk][i]=-t->query(a[i])+i*a[i]-s[i];
}
cout<<dp[k][m]<<nl;
return 0;
}

```

### Dot Product Optimization

///We are given n 2D vectors. The problem asks us to calculate the  
 ///maximum dot product of every vector with any other vector  
 ///including itself.

///idea is  $\max(ax + by) = y \cdot \max(a \cdot \frac{x}{y} + b)$  which is classical CHT

with some case to take care of

/\*\*

consider the lines  $F_i(x)=a + bx$  for each point  $(a, b)$ .

Now, if we want to find the minimum dot product of a vector  $(c, d)$   
 with some other, we need to minimize  $ac + bd = c(a + b * (d/c))$ .

So, depending on the sign we should either minimize the function  $F_j$   
 $(d / c)$  for some  $j$

or maximize it. we should also take care at cases that contain 0s

\*\*/

/\*insert dynamic CHT code\*/

double a[N],b[N];

CHT \*x1,\*x2,\*y,\*y2;

int main()

{

```

fast;
int i,j,k,n,m;
cin>>n;
x1=new CHT();
x2=new CHT();
y=new CHT();
y2=new CHT();
for(i=0;i<n;i++) cin>>a[i]>>b[i],x1->add(-b[i],-a[i]),x2->add(-a[i],-b[i]),
y->add(b[i],a[i]),y2->add(a[i],b[i]);
for(i=0;i<n;i++){
 int z1=0,z2=0;
 if(fabs(a[i]-0.0)<eps) z1=1;
 if(fabs(b[i]-0.0)<eps) z2=1;
 double ans;
 if(z1&&z2) ans=0.0;
 else{
 if(z1) swap(a[i],b[i]);
 double p=b[i]/a[i];
 ans=fmax(a[i]*-(z1?x2:x1)->query(p),a[i]*(z1?y2:y)->query(p));
 }
 cout<<fout(3)<<ans<<nl;
}
return 0;
}

```

### 53. Divide and Conquer Optimization

```

///Divide 1,2,3...n people in k consecutive parts so that sum of cost of
each individual part is minimum
int a[N][N],c[N][N],dp[810][N];//dp[i][j]=minimum cost for dividing
[1..j] in i parts
int cost(int i,int j)
{
 if(i>j) return 0;
 return c[j][j]-c[i-1][j]-c[j][i-1]+c[i-1][i-1];
}
void yo(int i,int l,int r,int optl,int optr)
{
 if(l>r) return;
 int mid=(l+r)/2;
 dp[i][mid]=2e9;//for maximum cost change it to 0
 int opt=-1;
 for(int k=optl;k<=min(mid,optr);k++){
 int c=dp[i-1][k]+cost(k+1,mid);
 if(c<dp[i][mid]){//for maximum cost just change < to > only and
rest of the algo should not be changed
 dp[i][mid]=c;
 opt=k;
 }
 }
 //for opt[1..j]<=opt[1..j+1] i.e. cost(1,j)<=cost(1,j+1)
 yo(i,l,mid-1,optl,opt);
 yo(i,mid+1,r,opt,optr);
 //for opt[1..j]>=opt[1..j+1] i.e. cost(1,j)>=cost(1,j+1)
}

```

```

///yo(i,l,mid-1,opt,optr);
///yo(i,mid+1,r,optl,opt);

}

int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 n=sc();
 k=sc();
 for(i=1;i<=n;i++) for(j=1;j<=n;j++) a[i][j]=sc();
 for(i=1;i<=n;i++){
 for(j=1;j<=n;j++){
 c[i][j]=a[i][j]+c[i-1][j]+c[i][j-1]-c[i-1][j-1];
 }
 }
 for(i=1;i<=n;i++) dp[1][i]=cost(1,i);
 for(i=2;i<=k;i++) yo(i,1,n,1,n);
 cout<<dp[k][n]/2<<nl;
 return 0;
}

```

### 54. Knuth Optimization

```

/*
Knuths optimization works for optimization over sub arrays
for which optimal middle point depends monotonously on the end
points.
Let mid[l,r] be the first middle point for (l,r) sub array which gives
optimal result.

```

It can be proven that  $\text{mid}[l, r-1] \leq \text{mid}[l, r] \leq \text{mid}[l+1, r]$   
- this means monotonicity of mid by l and r.  
Applying this optimization reduces time complexity from  $O(k^3)$  to  $O(k^2)$   
because with fixed s (sub array length) we have  $m_{\text{right}}(l) = \text{mid}[l+1][r] = m_{\text{left}}(l+1)$ .  
That's why nested l and m loops require not more than  $2k$  iterations overall.

```
*/
```

```
int n,k;
int a[N],mid[N][N];
ll res[N][N];
ll solve()
{
 for (int s = 0; s<=k; s++){ //s - length of the subarray
 for (int l = 0; l+s<=k; l++) //l - left point
 {
 int r = l + s; //r - right point
 if (s < 2)
 {
 res[l][r] = 0; //base case- nothing to break
 mid[l][r] = l; //mid is equal to left border
 continue;
 }
 int mleft = mid[l][r-1];
 int mright = mid[l+1][r];
 res[l][r] = 2e18;
```

```
for (int m = mleft; m<=mright; m++) //iterating for m in
the bounds only
{
 ll tmp = res[l][m] + res[m][r] + (a[r]-a[l]);
 if (res[l][r] > tmp) //relax current solution
 {
 res[l][r] = tmp;
 mid[l][r] = m;
 }
}
}
ll ans = res[0][k];
return ans;
}
int main()
{
 BeatMeScanf;
 int i,j,m;
 while(cin>>n>>k){
 for(i=1;i<=k;i++) cin>>a[i];
 a[0]=0;
 a[k+1]=n;
 k++;
 cout<<solve()<<nl;
 }
 return 0;
}
```

## 55. In Out DP

```

///number of subtrees going through i-th node
vi g[N];
int dp1[N],dp2[N];
void dfs(int u,int pre)
{
 g[u].erase(remove(all(g[u]),pre),g[u].end());///Don't forget this
 dp1[u]=1;
 for(auto v:g[u]){
 if(v==pre) continue;
 dfs(v,u);
 dp1[u]=1LL*dp1[u]*(dp1[v]+1)%mod;
 }
}
void yo(int u,int pr)
{
 vi pre(sz(g[u]),1),suf(sz(g[u]),1);
 for(int i=0;i<sz(g[u]);i++){
 int v=g[u][i];
 pre[i]=(dp1[v]+1)%mod;
 if(i) pre[i]=1LL*pre[i]*pre[i-1]%mod;
 }
 for(int i=sz(g[u])-1;i>=0;i--){
 int v=g[u][i];
 suf[i]=(dp1[v]+1)%mod;
 if(i+1<sz(g[u])) suf[i]=1LL*suf[i]*suf[i+1]%mod;
 }
 for(int i=0;i<sz(g[u]);i++){

```

```

 int v=g[u][i];
 dp2[v]=1LL*(dp2[u]+1)%mod*(i-1<0?1:pre[i-1])%mod*(i+1>=sz(g[u])?1:suf[i+1])%mod;
 yo(v,u);
 }
}
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v;
 cin>>n>>mod;
 for(i=1;i<n;i++) cin>>u>>v,g[u].eb(v),g[v].eb(u);
 dfs(1,0);
 yo(1,0);
 for(i=1;i<=n;i++) cout<<1LL*dp1[i]*(dp2[i]+1)%mod<<nl;
 return 0;
}

```

## 56. Substring DP

```

///calculate the number of regular bracket sequences of length 2n
///containing the given bracket sequence s as a substring
vector<int> build_lps(string p)
{
 int sz = p.size();
 vector<int> lps;
 lps.assign(sz + 1, 0);
 int j = 0;
 lps[0] = 0;
 for(int i = 1; i < sz; i++)

```

```

{
 while(j >= 0 && p[i] != p[j])
 {
 if(j >= 1) j = lps[j - 1];
 else j = -1;
 }
 j++;
 lps[i] = j;
}

return lps;
}
vi lps;
int n,m,dp[N][N][N];
string s;

int add(int j,char ch)
{
 if(j==m) return j;
 while(j >= 0 && s[j] != ch)
 if(j >= 1) j = lps[j - 1];
 else j = -1;
 j++;
 return j;
}

int yo(int i,int j,int sum)
{
 if(i==2*n) return (j==m&&sum==0);
}

```

```

int &ret=dp[i][j][sum];
if(ret!=-1) return ret;
int ans=(yo(i+1,add(j,'('),sum+1)+(sum?yo(i+1,add(j,')'),sum-1):0)%mod;
return ret=ans;
}

int main()
{
 BeatMeScanf;
 int i,j,k;
 cin>>n>>s;
 m=s.size();
 lps=build_lps(s);
 mem(dp,-1);
 cout<<yo(0,0,0)<<nl;
 return 0;
}

```

## 57. Bounded Knapsack

```

///ps-profits
///ws-weights
///ms-maximum limit of each element
///W-maximum weight
///O(n*W)
int boundedKnapsack(vector<int> ps,vector<int> ws,vector<int> ms,int W) {
 int n = ps.size();
 vector<vector<int>> dp(n+1, vector<int>(W+1));

```

```

for (int i = 0; i < n; ++i) {
 for (int s = 0; s < ws[i]; ++s) {
 int alpha = 0;
 queue<int> que;
 deque<int> peek;
 for (int w = s; w <= W; w += ws[i]) {
 alpha += ps[i];
 int a = dp[i][w]-alpha;
 que.push(a);
 while (!peek.empty() && peek.back() < a) peek.pop_back();
 peek.push_back(a);
 while (que.size() > ms[i]+1) {
 if (que.front() == peek.front()) peek.pop_front();
 que.pop();
 }
 dp[i+1][w] = peek.front()+alpha;
 }
 }
 int ans = 0;
 for (int w = 0; w <= W; ++w)
 ans = max(ans, dp[n][w]);
 return ans;
}
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 n = 10;
}

```

```

int W = 100;
vector<int> ps(n), ws(n), ms(n);
for (int i = 0; i < n; ++i) {
 ps[i] = rand() % n + 1;
 ws[i] = rand() % n + 1;
 ms[i] = rand() % n + 1;
}
cout << boundedKnapsack(ps, ws, ms, W) << endl;
return 0;
}

```

## 58. Knapsack Branch and Bound

```

ll knapsack_problem_branch_and_bound(int n, ll max_w, vector<ll>
const & a_v, vector<ll> const & a_w) {
 vector<ll> v(n), w(n);
 vector<int> xs(n);
 iota(all(xs), 0);
 sort(all(xs), [&](int i, int j) {
 return a_v[i] *(double) a_w[j] > a_v[j] *(double) a_w[i];
 });
 for(int i=0;i<n;i++) {
 v[i] = a_v[xs[i]];
 w[i] = a_w[xs[i]];
 }
 ll ans = 0;
 function<void (int, ll, ll)> go = [&](int i, ll cur_v, ll cur_w) {
 if (max_w < cur_w) return; // not executable
 if (i == n) {

```

```

 chmax(ans, cur_v);
 return; // terminate
}
// lr_v = cur_v; // linear relaxation
// lr_w = cur_w;
int j = i;
for (; j < n and lr_w + w[j] <= max_w; ++j) { // greedy
 lr_w += w[j];
 lr_v += v[j];
}
if (lr_w == max_w or j == n) {
 chmax(ans, lr_v);
 return; // accept greedy
}
double lr_ans = lr_v + v[j] * ((max_w - lr_w) / (double) w[j]);
if (lr_ans <= ans) return; // bound
go(i + 1, cur_v + v[i], cur_w + w[i]);
go(i + 1, cur_v, cur_w);
};
go(0, 0, 0);
return ans;
}

```

## 59. Sum of Subsets DP

### Standard DP

```
/*
we must initialize dp array with value based on problem.
```

Only one inner loop will be activated based on problem  
`*/`  
`/**`  
actual dp state we write optimize version .  
dp[mask][i] contains all subsets of mask which differ from it only in  
the first i bits  
if(mask & (1<<i))  
 dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];  
else  
 dp[mask][i] = dp[mask][i-1];  
`*/`  
const int BIT = 20 ;  
int dp[(1<<BIT)+7] ;  
void rec()  
{  
 for(int i=0 ; i<BIT ; i++)  
 {  
 for(int mask=0 ; mask<(1<<BIT) ; mask++)  
 {  
 /\*\*
 this loop is used if we want mask&x == x
 dp[mask]=sum of all sub masks of mask
 \*/
 if(mask&(1<<i)) dp[mask] += dp[mask^(1<<i)] ;
 }
 for(int mask=(1<<BIT)-1 ; mask>=0 ; mask--)  
 {
 /\*\*
 this loop is used if we want mask&x == mask
 \*/
 }
 }
}

```

dp[mask]= sum of all super masks of mask
*/
if((mask&(1<<i)) == 0) dp[mask] += dp[mask^(1<<i)];
}
}
}

```

## Notes

- You have been given an integer array a of size n. You must report the number of ordered pairs (i, j) such that  $a[i] \& a[j]$  is 0.  

```

int ans=0;
dp[i]=sum of submasks of i
for(i=0;i<n;i++) ans+=dp[((1<<BIT)-1)^a[i]]

```
- You are given an array a of n integers. How many subsets of these integers have 0 when applied bitwise and on the subset?///for getting and=k check the next problem,it is similar to that,only use supermasks  

```

/// apply inclusion-exclusion
ll ans=0;
dp[i]=sum of super masks of i
twos[i]=2^i
for(int i=0; i<N; i++)
{
 int bits=__builtin_popcount(i);
 ans+=((bits&1) ? (-1) : 1)*twos[dp[i]];
 ans%=mod;
 if(ans<0) ans+=mod;
}

```

- You are given an array a of n integers and a value k. How many subsets of these integers have a value of k when applied bitwise or on the subset?  

```

ll ans=0;
dp[i]=sum of sub masks of i
twos[i]=2^i
///O(number of submasks of k)
for(int i=k; i>=0; i--)
{
 if((i|k)!=k) continue;///check if i sub mask of k
 int curr=__builtin_popcount(k^i);
 ans+=(curr%2==0 ? 1 : -1)*(twos[dp[i]]-1);
 ans%=mod;
 if(ans<0) ans+=mod;
}

```
- Minimum cost submasks to get the mask (11111...11)  

```

const int BIT=16;
int dp[(1<<BIT)+9];
///dp[i]= minimum cost of masks you need to take to get a
super mask of 'mask
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 for(i=0;i<(1<<BIT);i++) dp[i]=1e9;
 cin>>n;
 for(i=0;i<n;i++) cin>>k>>m,dp[k]=min(dp[k],m);///costs of
masks
 ///update mask from the supermasks of them
}

```

```

for(i=0;i<BIT;i++) for(j=(1<<BIT)-1;j>=0;j--) if(!((j>>i)&1))
dp[j]=min(dp[j],dp[j^(1<<i)]);

for(i=0;i<(1<<BIT);i++){
 for(int sub=i;;sub=(sub-1)&i){
 dp[i]=min(dp[i],dp[sub]+dp[i^sub]);
 if(sub==0) break;
 }
}
cout<<dp[(1<<BIT)-1]<<n;
return 0;
}

```

- maximum and pair in an array?  
try to set the max bits one first. let the current processed ans be ret. we are at i-th bit. check if there are at least two masks with  $mask \& (ret | (1 << i)) == (ret | (1 << i))$ . if there are, then update  $ret = ret | (1 << i)$  else  $ret = ret$  and process the next smaller bit.

## 60. Maximum AND subset with Given Length With Update

```
/**
```

You are given an array of n numbers and q queries. In each query, you are asked to add a number, delete a number or answer value of  $f(x)$  where  $f(x)$  denotes the value of the maximum bitwise-and

of all the subsequences of length x of the array. The idea is to use Yate's dp(for maximum and) along with SQRT Decomposition on queries.

```

*/
const int bsz=512;
int cnt[N], a[N], n, m, k, dp[N], idx=0;///k is max BIT
pii bucket[bsz+5];
void calc()
{
 for(int i=0;i<(1<<k);i++) dp[i]=cnt[i];
 // dp[mask] = total numbers x such that x&mask=mask i.e.
 supermasks of mask
 for(int i=0; i<k; i++)
 {
 for(int mask=(1<<k)-1;mask>=0;mask--)
 {
 if(!(mask&(1<<i))) dp[mask]+=dp[mask^(1<<i)];
 }
 }
}
void add(int x)
{
 cnt[x]++;
 bucket[idx]={x,1};
 idx++;
 if(idx==bsz) /// bucket full, recalc dp
 {
 calc();
 idx=0;
 }
}

```

```

 }
}

void rem(int x)
{
 cnt[x]--;
 bucket[idx] = {x, -1};
 idx++;
 if(idx==bsz) /// bucket full, recalc dp
 {
 calc();
 idx=0;
 }
}

/// how many values t are there such that t&curr==curr?
int canget(int curr)
{
 int ret=dp[curr];
 for(int i=0;i<idx;i++)
 {
 if((bucket[i].first & curr)==curr) ret+=bucket[i].second;
 }
 return ret;
}

int getans(int x)
{
 int ret=0;
 /// try to ensure the higher bits first
 for(int i=k-1; i>=0; i--)
 {
 int curr=(ret^(1<<i));
 /// There are total greater equal x numbers t such
 /// that t&curr==curr;
 if(canget(curr)>=x) /// We can achieve curr
 ret|=1<<i;
 }
 return ret;
}

void handleQueries()
{
 int t, x;
 for(int i=0;i<m;i++)
 {
 scanf("%d%d", &t, &x);
 if(t==1) add(x);
 else if(t==2) rem(x);
 else printf("%d\n", getans(x));
 }
}

int main()
{
 scanf("%d%d%d", &n, &m, &k);
 for(int i=0;i<n;i++)
 {
 scanf("%d", &a[i]);
 cnt[a[i]]++;
 }
 calc();
 handleQueries();
}

```

```
return 0;
}
```

## STRING THEORY

### 61. Notes

- What is the largest n such that a given string can be written as a concatenation of exactly n substring? For example, answer for ababab is 3 as it can be written as concatenation of 3 ab.  
`k=s.size()  
if(k%(k-lps[k-1])==0) cout<<k/(k-lps[k-1])<<nl;  
else cout<<1<<nl;`

- solve the previous problem for each substring of the string and return the maximum value.

```
/*suffix array code*/
int solve(int n)
{
 int ret=0;
 for(int len=1; len<=n; len++)
 {
 for(int j=0; j+len<n; j+=len)
 {
 int curr=lcp_(j,j+len);
 int k=j-(len-curr%len);
 curr=curr/len+1;
 if(k>=0 && lcp_(k,k+len)>=len)
 {

```

```
 curr++;
 }
 ret=max(ret,curr);
 }
 }
 return ret;
}
```

### 62. Trie

#### Max/Min

```
struct node
{
 node* next[2];
 node()
 {
 next[0]=next[1]=NULL;
 }
}*root;
int sum[mxn];
void insert_num(int x)
{
 node* cur=root;
 int b;
 for(int i=31;i>=0;i--){
 b=(x>>i)&1;
 if(cur->next[b]==NULL) cur->next[b]=new node();
 cur=cur->next[b];
 }
}
```

```

}

int get_max(int x)
{
 node* cur=root;
 int k,ans=0;
 for(int i=31;i>=0;i--){
 k=(x>>i)&1;
 if(cur->next[!k]) cur=cur->next[!k],ans<=>1,ans++;
 else cur=cur->next[k],ans<=>1;
 }
 return ans;
}

int get_min(int x)
{
 node* cur=root;
 int k,ans=0;
 for(int i=31;i>=0;i--){
 k=(x>>i)&1;
 if(cur->next[k]) cur=cur->next[k],ans<=>1;
 else cur=cur->next[!k],ans<=>1,ans++;
 }
 return ans;
}

void del(node* cur)
{
 for(int i=0;i<2;i++) if(cur->next[i]) del(cur->next[i]);
 delete(cur);
}

int main()

```

```

{
 //fast;
 int i,j,k,n,m,t,cs=0;
 sf(t);
 while(t--){
 root= new node();
 int mx=-inf,mn=inf;
 sf(n);
 sum[0]=0;
 for(i=1;i<=n;i++) sf(k),sum[i]=sum[i-1]^k;
 insert_num(0);
 for(i=1;i<=n;i++){
 mx=max(mx,get_max(sum[i]));
 mn=min(mn,get_min(sum[i]));
 insert_num(sum[i]);
 }
 _ccase;
 pf("%d %d\n",mx,mn);
 for(i=1;i<=n;i++) sum[i]=0;
 del(root);
 }
 return 0;
}

```

### Xor Pairs less than k

```

struct node
{
 node* next[2];
 int sz;//size of subtree

```

```

node()
{
 next[0]=next[1]=NULL;
 sz=0;
}
}*root;
void insert_num(int x)
{
 node* cur=root;
 cur->sz++;
 for(int i=31;i>=0;i--){
 int b=(x>>i)&1;
 if(cur->next[b]==NULL) cur->next[b]=new node();
 cur=cur->next[b];
 cur->sz++;
 }
}
///number of pairs having xor with x less than k
ll get_count(int x,int k)
{
 node* cur=root;
 ll ans=0;
 for(int i=31;i>=0;i--){
 if(cur==NULL) break;
 int b1=(x>>i)&1;
 int b2=(k>>i)&1;
 if(b2==1){
 if(cur->next[b1]) ans+=1LL*cur->next[b1]->sz;
 cur=cur->next[!b1];
 }
 }
}

```

```

 }
 else cur=cur->next[b1];
}
return ans;
}
int a[N];

///given an array find number of subarrays having xor less than k
int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin>>n>>k;
 for(i=1;i<=n;i++) cin>>a[i],a[i]^=a[i-1];
 root=new node();
 ll ans=0;
 insert_num(0);///for subarray xor it must be done,but for pair xor
 will it be added?
 for(i=1;i<=n;i++){
 ans+=get_count(a[i],k);
 insert_num(a[i]);
 }
 cout<<ans<<nl;
 return 0;
}

```

### Persistent Trie

//Type 0: Add the integer number x at the end of the array.  
//Type 1: On the interval L..R find a number y, to maximize (x xor y).

```
///Type 2: Delete last k numbers in the array
```

```
///Type 3: On the interval L..R, count the number of integers less than
or equal to x.
```

```
///Type 4: On the interval L..R, find the kth smallest integer (kth order
statistic).
```

```
struct trie
```

```
{
```

```
 trie* nxt[2];
```

```
 int cnt = 0;
```

```
};
```

```
int n, q, r, key, curRoot=0;
```

```
trie* root[N];
```

```
trie* head;
```

```
///the next node if we traverse b from current trie
```

```
trie* get(trie* node, int b)
```

```
{
```

```
 if(!node || !node->nxt[b])
```

```
 return NULL;
```

```
 return node->nxt[b];
```

```
}
```

```
///number of elements in the trie
```

```
int getCnt(trie* node)
```

```
{
```

```
 if(!node)
```

```
 return 0;
```

```
 return node->cnt;
```

```
}
```

```
void addNew(trie* old, trie* cur, int val, int i=20)
```

```
{
```

```
 cur->cnt = getCnt(old) + 1;
```

```
 if(i < 0)
```

```
 return;
```

```
 int b = val>>i & 1;
```

```
 cur->nxt[b] = new trie();
```

```
 addNew(get(old, b), cur->nxt[b], val, i-1);
```

```
 cur->nxt[!b] = get(old, !b);
```

```
}
```

```
void add(int par, int u, int val)
```

```
{
```

```
 trie* cur = new trie();
```

```
 addNew(root[par], cur, val, 20);
```

```
 root[u] = cur;
```

```
}
```

```
///minimum xor with x in the whole (cur) trie
```

```
int minXor(trie* cur, int x, int i)
```

```
{
```

```
 if(i < 0)
```

```
 return 0;
```

```
 int b = x>>i & 1;
```

```
 if(cur->nxt[b] && cur->nxt[b]->cnt>0)
```

```
 return minXor(cur->nxt[b], x, i-1);
```

```
 else
```

```

 return (1LL<<i) + minXor(cur->nxt[!b], x, i-1);
 }

///maximum xor with x in the whole (cur) trie
int maxXor(trie* cur, int x, int i)
{
 if(i < 0)
 return 0;
 int b = x>>i & 1;
 if(cur->nxt[!b] && cur->nxt[!b]->cnt>0)
 return (1LL<<i) + maxXor(cur->nxt[!b], x, i-1);
 else
 return maxXor(cur->nxt[b], x, i-1);
}

///maximum xor with x in a[l...r]
int getMaxXor(trie* L, trie* R, int x, int i=20)
{
 if(i < 0)
 return 0;
 int b = x>>i & 1;
 int have = getCnt(get(R, !b)) - getCnt(get(L, !b));
 if(have > 0)
 return (1LL<<i) | getMaxXor(get(L, !b), get(R, !b), x, i-1);
 else
 return getMaxXor(get(L, b), get(R, b), x, i-1);
}
///the array element for which xor with x is maximum in a[l...r]

```

```

int getMax(trie* L, trie* R, int x, int i=20)
{
 if(i < 0)
 return 0;
 int b = x>>i & 1;
 int have = getCnt(get(R, !b)) - getCnt(get(L, !b));
 if(have > 0)
 return ((!b)<<i) | getMax(get(L, !b), get(R, !b), x, i-1);
 else
 return b<<i | getMax(get(L, b), get(R, b), x, i-1);
}
///number of elements less than or equal x in a[l..r]
int getLeq(trie* L, trie* R, int x, int i=20)
{
 if(i < 0)
 return getCnt(R) - getCnt(L);
 int b = x>>i & 1;
 int ans = 0;
 if(b)
 ans += getCnt(get(R, 0)) - getCnt(get(L, 0));
 return ans + getLeq(get(L, b), get(R, b), x, i-1);
}
///kth smallest element(not unique) in a[l...r]
int getKth(trie* L, trie* R, int k, int i)
{
 if(i < 0)
 return 0;
 int have = getCnt(get(R, 0)) - getCnt(get(L, 0));
 if(have >= k)

```

```

 return getKth(get(L, 0), get(R, 0), k, i-1);
 else
 return (1LL<<i) + getKth(get(L, 1), get(R, 1), k-have, i-
1);
}

void Q0(int x)
{
 add(curRoot, curRoot+1, x);
 curRoot++;
}

void Q1(int l, int r, int x)
{
 cout<<getMax(root[l-1], root[r], x, 20)<<endl;
}

void Q2(int k)
{
 curRoot-=k;
}

void Q3(int l, int r, int x)
{
 cout<<getLeq(root[l-1], root[r], x, 20)<<endl;
}

void Q4(int l, int r, int k)
{

```

```

 cout<<getKth(root[l-1], root[r], k, 20)<<endl;
}

int32_t main()
{
 BeatMeScanf;
 root[0] = new trie();
 int q;
 cin>>q;
 while(q--)
 {
 int type, l, r, x;
 cin>>type;
 if(type==0)
 cin>>x, Q0(x);
 else if(type==1)
 cin>>l>>r>>x, Q1(l, r, x);
 else if(type==2)
 cin>>x, Q2(x);
 else if(type==3)
 cin>>l>>r>>x, Q3(l, r, x);
 else
 cin>>l>>r>>x, Q4(l, r, x);
 }
 return 0;
}
```

## 63. String Matching

### Knuth-Morris-Pratt

```

///returns the longest proper prefix array of pattern p
///where lps[i]=longest proper prefix which is also suffix of p[0...i]
vector<int> build_lps(string p)
{
 int sz = p.size();
 vector<int> lps;
 lps.assign(sz + 1, 0);
 int j = 0;
 lps[0] = 0;
 for(int i = 1; i < sz; i++)
 {
 while(j >= 0 && p[i] != p[j])
 {
 if(j >= 1) j = lps[j - 1];
 else j = -1;
 }
 j++;
 lps[i] = j;
 }

 return lps;
}

vector<int>ans;
///returns matches in vector ans in 0-indexed
void kmp(vector<int> lps, string s, string p)
{

```

```

 int psz = p.size(), sz = s.size();
 int j = 0;
 for(int i = 0; i < sz; i++)
 {
 while(j >= 0 && p[j] != s[i])
 if(j >= 1) j = lps[j - 1];
 else j = -1;

 j++;
 if(j == psz)
 {
 j = lps[j - 1];
 //pattern found in string s at position i-psz+1
 ans.eb(i-psz+1);
 }
 //after each loop we have j=longest common suffix of
 s[0..i] which is also prefix of p
 }

 int main()
 {
 int i,j,k,n,m,t;
 cin>>t;
 while(t--){
 string s,p;
 cin>>s>>p;
 vector<int>lps = build_lps(p);
 kmp(lps, s, p);

```

```

if(ans.empty()) cout<<"Not Found\n";
else{
 cout<<ans.size()<<nl;
 for(auto x:ans) cout<<x<<' ';
 cout<<nl;
}
ans.clear();
cout<<nl;
}
return 0;
}

```

## Bitset

```

///Complexity: $\frac{n^2}{64}$
vi v;
bitset<N>bs[26],oc;
int main()
{
 fast;
 int i,j,k,n,q,l,r;
 string s,p;
 cin>>s;
 for(i=0;s[i];i++) bs[s[i]-'a'][i]=1;
 cin>>q;
 while(q--){
 cin>>p;
 oc.set();
 for(i=0;p[i];i++) oc&=(bs[p[i]-'a']>>i);
 cout<<oc.count()<<nl;///number of occurrences
 }
}

```

```

int ans=N,sz=p.size();
int pos=oc._Find_first();
v.pb(pos);
pos=oc._Find_next(pos);
while(pos<N){
 v.pb(pos);
 pos=oc._Find_next(pos);
}
for(auto x:v) cout<<x<<' ';///position of occurences
cout<<nl;
v.clear();
cin>>l>>r;//number of occurrences from l to r, where l and r is 1-indexed
if(sz>r-l+1) cout<<0<<nl;
else cout<<(oc>>(l-1)).count()-(oc>>(r-sz+1)).count()<<nl;
}
return 0;
}

```

## Z-Algorithm

```

///An element Z[i] of Z array stores length of the longest substring
///starting from str[i] which is also a prefix of str[0..n-1].
///The first entry of Z array is meaning less as complete string is
always prefix of itself.
///Here Z[0]=0.
vector<int> z_function(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;
}
///for pattern searching use P$T version (P=pattern,T=text).
///for text indexes if (z[i]==pattern length) then pattern is found from
///that index
int main()
{
 fast;
 ll i,j,k,n,m;
 string s;
 cin>>s;
 vi ans=z_function(s);
 for(auto x:ans) cout<<x<<' ';
 return 0;
}

```

## Aho Corasick

Time Complexity: $O(n)$

Space Complexity for index of all occurrences: $O(m\sqrt{m})$ , where  $m = \text{sum of all patterns}$

///beware! if k distinct patterns are given having sum of length m then size of ending array and oc array will

```

///be at most $m.\sqrt{m}$,But for similar patterns one must act with
them differently
struct aho_corasick
{
 bool is_end[N];
 int link[N]; ///A suffix link for a vertex p is a edge that
points to
 ///the longest proper suffix of
 ///the string corresponding to the vertex p.
 int psz; ///tracks node numbers of the trie
 map<char, int> to[N]; //tracks the next node
 vi ending[N]; ///ending[i] stores the indexes of patterns
which ends
 ///at node i(from the trie)
 vi oc[N]; ///oc[i] stores ending index of all occurrences
of pattern[i]
 ///so real oc[i][j]=oc[i][j]-pattern[i].size()+1,0-indexed
 int cnt[N],path[N],ind[N],len;///for number of occurrences
 void clear()
 {
 for(int i = 0; i < psz; i++)
 is_end[i] = 0,cnt[i]=0,path[i]=0, ind[i]=0,link[i]
= 0,to[i].clear(),ending[i].clear(),oc[i].clear();
 psz = 1;
 is_end[0] = 1;
 len=0;
 }

```

```

aho_corasick() { psz = N - 2; clear(); }

void add_word(string s,int idx)
{
 int u = 0;
 for(char c: s)
 {
 if(!to[u].count(c)) to[u][c] = psz++;
 u = to[u][c];
 }

 is_end[u] = 1;
 ending[u].eb(idx);
 ind[idx]=u;
}

void populate(int cur)
{
 /// merging the occurrences of patterns ending at cur node in
 the trie
 for(auto occ: ending[link[cur]])
 ending[cur].eb(occ);
}

void push_links()
{
 queue<int> q;
 int u, v, j;
 char c;

 q.push(0);
 link[0] = -1;

 while(!q.empty())
 {
 u = q.front();
 q.pop();

 for(auto it: to[u])
 {
 v = it.second;
 c = it.first;
 j = link[u];

 while(j != -1 && !to[j].count(c)) j =
 link[j];
 if(j != -1) link[v] = to[j][c];
 else link[v] = 0;

 q.push(v);
 populate(v);
 path[len++]=v;
 }
 }
}

void populate(vi &en, int cur)
{
 /// occurrences of patterns in the given string
}

```

```

for(auto idx: en)
{
 oc[idx].eb(cur);
}
}

void traverse(string s)
{
 int n=s.size();
 int cur=0;///root

 for(int i=0;i<n;i++){
 char c=s[i];
 while(cur!=-1 && !to[cur].count(c)) cur=link[cur];
 if(cur!=-1) cur=to[cur][c];
 else cur=0;
 populate(ending[cur],i);
 cnt[cur]++;
 }
 for(int i=len-1;i>=0;i--) cnt[link[path[i]]]+=cnt[path[i]];
}

aho_corasick t;
string p[N];
int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 string s;
}

```

```

cin>>s;
cin>>m;
for(i=0;i<m;i++){
 cin>>p[i];
 t.add_word(p[i],i);
}
t.push_links();
t.traverse(s);
///print all occurrences
for(i=0;i<m;i++){
 cout<<t.oc[i].size()<<nl;
 for(auto x:t.oc[i]) cout<<x-p[i].size()+1<<' ';
 cout<<nl;
}
///print number of occurrences
for(i=0;i<m;i++) cout<<t.cnt[t.ind[i]]<<' ';
cout<<nl;
return 0;
}

```

### Aho Corasick Dynamic

```

//add strings and count sum of occurrences of all strings in a given
string dynamically
struct aho_corasick_static
{
 int cnt[N], link[N], psz;
 map<char, int> to[N];

 void clear()
}

```

```

{
 for(int i = 0; i < psz; i++)
 cnt[i] = 0, link[i] = -1, to[i].clear();

 psz = 1;
 link[0] = -1;
 cnt[0] = 0;
}

aho_corasick_static() { psz = N - 2; clear(); }

void add_word(string s)
{
 int u = 0;
 for(char c: s)
 {
 if(!to[u].count(c)) to[u][c] = psz++;
 u = to[u][c];
 }
 cnt[u]++;
}

void push_links()
{
 queue<int> Q;
 int u, v, j;
 char c;
}

Q.push(0);
link[0] = -1;

while(!Q.empty())
{
 u = Q.front();
 Q.pop();

 for(auto it: to[u])
 {
 v = it.second;
 c = it.first;
 j = link[u];

 while(j != -1 && !to[j].count(c)) j =
 link[j];
 if(j != -1) link[v] = to[j][c];
 else link[v] = 0;

 cnt[v] += cnt[link[v]];
 Q.push(v);
 }
}

int get_count(string p)
{
 int u = 0, ans = 0;
 for(char c: p)
}

```

```

 {
 while(u != -1 && !to[u].count(c)) u = link[u];
 if(u == -1) u = 0;
 else u = to[u][c];
 ans += cnt[u];
 }

 return ans;
};

struct aho_corasick
{
 vector<string> li[20];
 aho_corasick_static ac[20];

 void clear()
 {
 for(int i = 0; i < 20; i++)
 {
 li[i].clear();
 ac[i].clear();
 }
 }

 aho_corasick() { clear(); }

 void add_word(string s)
 {
 int pos = 0;
 for(int l = 0; l < 20; l++)
 if(li[l].empty())
 {
 pos = l;
 break;
 }

 li[pos].push_back(s);
 ac[pos].add_word(s);

 for(int bef = 0; bef < pos; bef++)
 {
 ac[bef].clear();
 for(string s2: li[bef])
 {
 li[pos].push_back(s2);
 ac[pos].add_word(s2);
 }

 li[bef].clear();
 }

 ac[pos].push_links();
 }

 ///sum of occurrences of all patterns in this string
 int get_count(string s)
 {
 int ans = 0;
 }
}

```

```

 for(int l = 0; l < 20; l++)
 ans += ac[l].get_count(s);

 return ans;
 }

};

string s[N];
aho_corasick aho;

int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin >> n;
 for(int i= 0; i < n; i++)
 cin >> s[i];
 aho.clear();
 for(int i = 0; i < n; i++)
 {
 aho.add_word(s[i]);
 cout << aho.get_count("aaaaaassssaaa") << endl;
 }
 return 0;
}

```

## A KMP Application

/\*\* You are given a text t and a pattern p. For each index of t, find how many prefixes of p ends in this position. Similarly, find how many

suffixes start from this position.  
While calculating the failure function, we can find for each position of the pattern p  
how many of its own prefixes end in that position. After calculating that in dp[i],  
we can just feel table[i] for text t.  
\*\*/

```

int pi[N], dp[N];
void prefix_function(string &p)
{
 int now;
 pi[0]=now=-1;

 dp[0]=1; // 0th character is a prefix ending in itself, base case

 for(int i=1; i<sz(p); i++)
 {
 while(now!=-1 && p[now+1]!=p[i])
 now=pi[now];

 if(p[now+1]==p[i]) pi[i]=++now;
 else pi[i]=now=-1;

 if(pi[i]!=-1) // calculate the number of prefixes end in
this position of p
 dp[i]=dp[pi[i]]+1;
 else dp[i]=1;
 }
}

```

```

}

int ans[N];//number of prefixes(not only proper)of p ending at t[i]
///for suffixes use reverse t and reverse p
int kmp(string &p, string &t)
{
 int now=-1;
 for(int i=0;i<sz(t);i++)
 {
 while(now!=-1 && p[now+1]!=t[i]) now=pi[now];
 if(p[now+1]==t[i])
 {
 ++now;
 ans[i]=dp[now]; // ans for text t
 }
 else now=-1;
 if(now+1==sz(p))
 {
 now=pi[now];
 }
 }
}

int32_t main()
{
 BeatMeScarf;
 int i,j,k,n,m;
 string a,b;
 cin>>a>>b;
}

```

```

prefix_function(b);
kmp(b,a);
for(i=0;i<sz(a);i++) cout<<ans[i]<<' ';
return 0;
}

```

### Aho Corasick All Pair Occurrence Relation

/\*\*

Suppose we have  $n \leq 1000$  strings. Total summation of the length of these strings

can be  $1e7$ . Now we are given queries. In each query, we are given indices of

two strings and asked if one of them occurs in another as a substring. We need to find this relation efficiently. We will use Aho-Corasick.

The solution is to build a graph where vertices denote indices of strings and an edge

from  $u$  to  $v$  denotes that  $\text{string}[v]$  occurs in  $\text{string}[u]$ .

\*\*/

```

#define ALPHABET_SIZE 26
#define MAX_NODE (int)1e5
int n; // number of strings
string in[N], p;//input string

int node[MAX_NODE][ALPHABET_SIZE];
int root, nnodes, link[MAX_NODE], terminal[MAX_NODE];
bool g[N][N];

```

```
/// termlink[u] = a link from node u to a node which is a terminal node
/// terminal node is a node where an ending of an input string occurs
/// terminal[node] = the index of the string which ends in node
```

/\*\* Solution:

For every node of the Aho-Corasick structure find and remember the nearest terminal node (termlink[u]) in the suffix-link path; Once again traverse

all strings through Aho-Corasick. Every time new symbol is added, add an arc from the node

corresponding to the current string (in the graph we build, not Aho-Corasick) to

the node of the graph corresponding to the nearest terminal in the suffix-link path;

The previous step will build all essential arcs plus some other arcs, but they do not affect the next step in any way;

Find the transitive closure of the graph.

\*\*/

```
void init()
{
 root=0;
 nnode=0;
 mem(terminal,-1);
 mem(termlink,-1);
}
```

```
void add_word(int idx)
```

```
{
 p=in[idx];
 int len=p.size();
 int now=root;
 for(int i=0;i<len;i++){
 int x=p[i]-'a';
 if(!node[now][x]) node[now][x]=++nnode;
 now=node[now][x];
 }
 terminal[now]=idx; // string with index idx ends in now
}

void push_links()
{
 queue<int>q;
 link[0]=-1;
 q.push(0);

 while(!q.empty())
 {
 int u=q.front();
 q.pop();

 for(int i=0; i<ALPHABET_SIZE; i++)
 {
 if(!node[u][i]) continue;

 int v=node[u][i];
 int j=link[u];
 if(j==v) continue;

 link[v]=j;
 q.push(v);
 }
 }
}
```

```

while(j!=-1 && !node[j][i]) j=link[j];
if(j!=-1) link[v]=node[j][i];
else link[v]=0;

 // finding nearest terminal nodes
if(terminal[link[v]]!=-1) termlink[v]=link[v];
else termlink[v]=termlink[link[v]];

q.push(v);
}

}

void build_graph()
{
 for(int i=0;i<n;i++)
 {
 int cur=root;

 for(int j=0;j<sz(in[i]);j++)
 {
 char ch=in[i][j];
 cur=node[cur][(int)ch-'a'];

 int st=cur;
 if(terminal[st]==-1) st=termlink[st];

 for(int k=st; k>=0; k=termlink[k])
 {
 if(terminal[k]==i) continue;
 if(g[i][terminal[k]]) break;
 g[i][terminal[k]]=1;
 }
 }
 }
}

/**/
Finally, find transitive closure of the graph. If O(n^3) is possible, we can use Floyd-Warshall. Otherwise, run dfs from each node and add an edge from current starting node to each reachable node. An edge in this transitive closure denotes the occurrence relation.

*/
bool has[N][N];//has[i][j] means string in[i] has the string in[j]
/// i.e. in[j] occurs in in[i]
vi adj[N];
///0-indexed
void dfs(int st,int nw)
{
 has[st][nw]=1;
 for(auto i:adj[nw]) if(!has[st][i]) dfs(st,i);
}
///O(n^2)
void relation()
{
 for(int i=0;i<n;i++) for(int j=0;j<n;j++) if(g[i][j]) adj[i].eb(j);
}

```

```

 for(int i=0;i<n;i++) dfs(i,i);
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,m,q;
 cin>>n>>q;
 init();
 for(i=0;i<n;i++){
 cin>>in[i];
 add_word(i);
 }
 push_links();
 build_graph();
 relation();
 while(q--){
 int u,v;
 cin>>u>>v;//if one of them occurs in another
 --u,--v;
 if(has[u][v] || has[v][u]) cout<<"YES\n";
 else cout<<"NO\n";
 }
 return 0;
}

```

## 64. String Hashing

### Notes

The probability that collision happens is  $\approx \frac{1}{mod}$

If we compare a string with N different strings then the probability of collision is  $\approx \frac{N}{mod}$

### Hashing

```

ll qpow(ll n,ll k,ll mod)
ans=1;assert(k>=0);n%=mod;while(k>0){if(k&1)
ans=(ans*n)%mod;n=(n*n)%mod;k>>=1;}return ans%mod;

```

```
const int MOD1=127657753,MOD2=987654319;
```

```
const int p1=137,p2=277;
```

```
int invp1,invp2;
```

```
pii pw[N],invpw[N];
```

```
void pre()
```

```
{
```

```
pw[0]={1,1};
```

```
for(int i=1;i<N;i++){
```

```
 pw[i].F=1LL*pw[i-1].F*p1%MOD1;
```

```
 pw[i].S=1LL*pw[i-1].S*p2%MOD2;
```

```
}
```

```
invp1=qpow(p1,MOD1-2,MOD1);
```

```
invp2=qpow(p2,MOD2-2,MOD2);
```

```
invpw[0]={1,1};
```

```
for(int i=1;i<N;i++){
```

```
 invpw[i].F=1LL*invpw[i-1].F*invp1%MOD1;
```

```
 invpw[i].S=1LL*invpw[i-1].S*invp2%MOD2;
```

```

}

}

///returns hash of string s
pii get_hash(string s)
{
 int n=s.size();
 pii ans={0,0};
 for(int i=0;i<n;i++){
 ans.F=(ans.F+1LL*pw[i].F*s[i]%MOD1)%MOD1;
 ans.S=(ans.S+1LL*pw[i].S*s[i]%MOD2)%MOD2;
 }
 return ans;
}
struct RollingHash
{
 int n;
 string s;//0-indexed
 vector<pii>hs;//1-indexed
 void init(string _s)
 {
 n=_s.size();
 s=_s;
 hs.eb(0,0);
 for(int i=0;i<n;i++){
 pii p;
 p.F=(hs[i].F+1LL*pw[i].F*s[i]%MOD1)%MOD1;
 p.S=(hs[i].S+1LL*pw[i].S*s[i]%MOD2)%MOD2;
 hs.pb(p);
 }
 }
 }

 ///returns hash of substring [l....r],1-indexed
 pii get_hash(int l,int r)
 {
 pii ans;
 ans.F=(hs[r].F-hs[l-1].F+MOD1)%MOD1*1LL*invpw[l-1].F%MOD1;
 ans.S=(hs[r].S-hs[l-1].S+MOD2)%MOD2*1LL*invpw[l-1].S%MOD2;
 return ans;
 }
 ///returns hash of total string
 pii get()
 {
 return get_hash(1,n);
 }
};

RollingHash h;
int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 ///never forget to initialize pre()
 pre();
 string s;
 cin>>s;
 h.init(s);
 while(1){
}

```

```

 || a,b,c,d;
 cin>>a>>b>>c>>d;
 cout<<(h.get_hash(a,b)==h.get_hash(c,d))<<nl;
}
return 0;
}

```

## Hashing 2D

```

struct Hash {
 vector<vi> hs;
 vi pwx;
 vi pwy;
 void init(vector<string>& s)
 {
 int primex = 3731;
 int primey = 2999;
 int mod=1e9+7;
 int n = (int)s.size();
 int m = (int)s[0].size();
 hs.assign(n + 1, vi(m + 1, 0));
 pwx.assign(n + 1, 1);
 pwy.assign(m + 1, 1);
 for(int i=0;i<n;i++) pwx[i + 1] = 1LL*pwx[i] * primex%mod;
 for(int i=0;i<m;i++) pwy[i + 1] = 1LL*pwy[i] * primey%mod;
 for(int i=0;i<n;i++){
 for(int j=0;j<m;j++){
 hs[i + 1][j + 1] = s[i][j] - 'a' + 1;
 }
 }
 }

```

```

 for(int i=0;i<=n;i++){
 for(int j=0;j<m;j++){
 hs[i][j + 1] =(hs[i][j+1]+ 1LL*hs[i][j] * primey%mod)%mod;
 }
 }
 for(int i=0;i<n;i++){
 for(int j=0;j<=m;j++){
 hs[i + 1][j] =(hs[i+1][j]+ 1LL*hs[i][j] * primex%mod)%mod;
 }
 }
 //0-indexed
 //het_hash of array (posx,posy,posx+dx,posx+dy)
 //for array(1,1,3,4) return get_hash(1,1,2,3)
 int get_hash(int posx, int posy, int dx, int dy) {
 return (1LL*(hs[posx + dx][posy + dy] - 1LL*hs[posx + dx][posy] *
 pwy[dy]%mod+mod)%mod -
 1LL*(hs[posx][posy + dy] - 1LL*hs[posx][posy] *
 pwy[dy]%mod+mod)%mod * pwx[dx]%mod+mod)%mod;
 }
}h1,h2;

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m,r,c;
 cin>>r>>c;
 string s;
 vs v;
}
```

```

for(i=0;i<r;i++) cin>>s,v.eb(s);
h1.init(v);
v.clear();
cin>>n>>m;
for(i=0;i<n;i++) cin>>s,v.eb(s);
h2.init(v);
int cnt=0;
for(i=0;i<n;i++){
 for(j=0;j<m;j++){
 if(i+r-1<n&&j+c-
1<m&&h1.get_hash(0,0,r,c)==h2.get_hash(i,j,r,c))
cout<<('<<i+1<<','<<j+1<<')<<nl,cnt++;
 }
}
if(!cnt) cout<<"NO MATCH FOUND...\n";
return 0;
}

```

## 65. String Suffix Structures

### Suffix Array O(n)

```

const int kinds = 128;//maximum ASCII value of any character of the
string
char str[N];
int K, buc[N], r[N], sa[N], X[N], Y[N], high[N];
bool cmp(int *r, int a, int b, int x)
{
 return (r[a] == r[b] && r[a+x] == r[b+x]);
}

```

```

void suffix_array_DA(int n, int m)
{
 int *x = X, *y = Y, i, j, k = 0, l;
 memset(buc, 0, sizeof(buc));
 for(i = 0; i < n; i++) buc[x[i]=str[i]]++;
 for(i = 1; i < m; i++) buc[i] += buc[i-1];
 for(i = n-1; i >= 0; i--) sa[--buc[x[i]]] = i;
 for(l = 1, j = 1; j < n; m = j, l <= 1)
 {
 j = 0;
 for(i = n-l; i < n; i++) y[j++] = i;
 for(i = 0; i < n; i++) if(sa[i] >= l) y[j++] = sa[i]-l;
 for(i = 0; i < m; i++) buc[i] = 0;
 for(i = 0; i < n; i++) buc[x[y[i]]]++;
 for(i = 1; i < m; i++) buc[i] += buc[i-1];
 for(i = n-1; i >= 0; i--) sa[--buc[x[y[i]]]] = y[i];
 for(swap(x, y), x[sa[0]] = 0, i = 1, j = 1; i < n; i++)
 x[sa[i]] = cmp(y, sa[i-1], sa[i], l) ? j-1 : j++;
 }
 for(i = 1; i < n; i++) r[sa[i]] = i;
 for(i = 0; i < n-1; high[r[i++]] = k)
 for(k ? k--: 0, j = sa[r[i]-1]; str[i+k] == str[j+k]; k++);
}
vector<int> suffix_array_construction(string s)
{
 int n=s.size();
 for(int i=0;i<n;i++) str[i]=s[i];
 str[n]='\0';
}

```

```

suffix_array_DA(n+1,kinds);
vector<int>saa;
for(int i=1;i<=n;i++) saa.eb(sa[i]);
return saa;
}
vector<int> lcp_construction(string const& s, vector<int> const& p)
{
 int n = s.size();
 vector<int> rank(n, 0);

 for (int i = 0; i < n; i++) rank[p[i]] = i;

 int k = 0;
 vector<int> lcp(n-1, 0);

 for (int i = 0; i < n; i++) {
 if (rank[i] == n - 1) {
 k = 0;
 continue;
 }
 int j = p[rank[i] + 1];
 while (i + k < n && j + k < n && s[i+k] == s[j+k]) k++;
 lcp[rank[i]] = k;
 if (k) k--;
 }
 return lcp;
}
const int MX = 18;
int st[N][MX];

```

```

int lg[N];

void pre()
{
 lg[1] = 0;
 for (int i=2; i<N; i++)
 lg[i] = lg[i/2]+1;
}

void build(vector<int> &lcp)
{
 int n = lcp.size();
 for (int i=0; i<n; i++)
 st[i][0] = lcp[i];

 for (int k=1; k<MX; k++)
 for (int i=0; i<n; i++)
 {
 st[i][k] = st[i][k-1];
 int nxt = i + (1<<(k-1));
 if (nxt >= n) continue;
 st[i][k] = min(st[i][k], st[nxt][k-1]);
 }
 ///minimum of lcp[l.....r]
 int get(int l, int r)
 {
 int k = lg[r-l+1];
 return min(st[l][k], st[r-(1<<k)+1][k]);
 }
}

```

```

}

int ra[N],sz;
///lcp of suffix starting from i and j
int lcp_(int i,int j)
{
 if(i==j) return sz-i;
 int l=ra[i];
 int r=ra[j];
 if(l>r) swap(l,r);
 return get(l,r-1);
}
string ss;
///lower bound of string t
int lb(string &t,vi &sa){
 int l=0,r=sz-1;
 int k=t.size();
 int ans=sz;
 while(l<=r){
 int mid = (l+r)/2;
 if(ss.substr(sa[mid],min(sz-sa[mid],k)) >= t) ans=mid,r=mid-1;
 else l = mid+1;
 }
 return ans;
}
///upper bound of string t
int ub(string &t,vi &sa){
 int l=0,r=sz-1;
 int k=t.size();
 int ans=sz;

```

```

 while(l<=r){
 int mid = (l+r)/2;
 if(ss.substr(sa[mid],min(sz-sa[mid],k)) > t) ans=mid,r=mid-1;
 else l = mid+1;
 }
 return ans;
}
int main()
{
 fast;
 int i,j,k,n,m,q;
 string s;
 cin>>s;
 n = s.size();
 vector<int> sa = suffix_array_construction(s);
 vector<int> lcp = lcp_construction(s, sa);
 sz=n;
 ss=s;
 for(i=0;i<n;i++) ra[sa[i]]=i;
 pre();
 build(lcp);
 for(i=0;i<n;i++) cout<<sa[i]<<' ';
 cout<<nl;
 for(i=0;i<n-1;i++) cout<<lcp[i]<<' ';
 cout<<nl;
 cin>>q;
 ///lcp of suffixes
 while(q--){
 cin>>i>>j;

```

```

cout<<lcp_(i,j)<<nl;
}
cin>>q;
//number of occurrences of a pattern, not in sorted order
while(q--){
 string t;
 cin>>t;
 int l=lb(t,sa);
 int r=ub(t,sa);
 debug(l,r);
 for(i=l;i<r;i++) cout<<sa[i]<<' ';
 cout<<nl;
}
return 0;
}

```

### Suffix Array O(nlogn)

```

/// Equivalence Class of every k-th step
vector<vector<int>>c;
vector<int>sort_cyclic_shifts(string const& s)
{
 int n = s.size();
 const int alphabet = 256;
 vector<int> p(n), cnt(alphabet, 0);

 c.clear();
 c.emplace_back();
 c[0].resize(n);

```

```

for (int i = 0; i < n; i++) cnt[s[i]]++;
for (int i = 1; i < alphabet; i++) cnt[i] += cnt[i-1];
for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;

c[0][p[0]] = 0;
int classes = 1;

for (int i = 1; i < n; i++) {
 if (s[p[i]] != s[p[i-1]]) classes++;
 c[0][p[i]] = classes - 1;
}

vector<int> pn(n), cn(n);
cnt.resize(n);

for (int h = 0; (1<<h) < n; h++) {
 for (int i = 0; i < n; i++) {
 pn[i] = p[i] - (1<<h);
 if (pn[i] < 0) pn[i] += n;
 }
 fill(cnt.begin(), cnt.end(), 0);

 /// radix sort
 for (int i = 0; i < n; i++) cnt[c[h][pn[i]]]++;
 for (int i = 1; i < classes; i++) cnt[i] += cnt[i-1];
 for (int i = n-1; i >= 0; i--) p[--cnt[c[h][pn[i]]]] = pn[i];

 cn[p[0]] = 0;
 classes = 1;
}
```

```

for (int i = 1; i < n; i++) {
 pii cur = {c[h][p[i]] , c[h][(p[i] + (1<<h))%n]};
 pii prev = {c[h][p[i-1]], c[h][(p[i-1] + (1<<h))%n]};
 if (cur != prev) ++classes;
 cn[p[i]] = classes - 1;
}
c.push_back(cn);
}
return p;
}

vector<int> suffix_array_construction(string s)
{
 s += "!";
 vector<int> sorted_shifts = sort_cyclic_shifts(s);
 sorted_shifts.erase(sorted_shifts.begin());
 return sorted_shifts;
}

/// compare two suffixes starting at i and j with length 2^k
int compare(int i, int j, int n, int k)
{
 pii a = {c[k][i], c[k][(i+1-(1<<k))%n]};
 pii b = {c[k][j], c[k][(j+1-(1<<k))%n]};
 return a == b ? 0 : a < b ? -1 : 1;
}

int lcp_(int i, int j)
{

```

```

 int log_n = c.size()-1;
 int ans = 0;

 for (int k = log_n; k >= 0; k--) {
 if (c[k][i] == c[k][j]) {
 ans += 1 << k;
 i += 1 << k;
 j += 1 << k;
 }
 }
 return ans;
}

vector<int> lcp_construction(string const& s, vector<int> const& p)
{
 int n = s.size();
 vector<int> rank(n, 0);

 for (int i = 0; i < n; i++) rank[p[i]] = i;

 int k = 0;
 vector<int> lcp(n-1, 0);

 for (int i = 0; i < n; i++) {
 if (rank[i] == n - 1) {
 k = 0;
 continue;
 }
 int j = p[rank[i] + 1];

```

```

while (i + k < n && j + k < n && s[i+k] == s[j+k]) k++;
lcp[rank[i]] = k;
if (k) k--;
}
return lcp;
}

const int K = 18;
int st[N][K];
int lg[N];

void pre()
{
 lg[1] = 0;
 for (int i=2; i<N; i++)
 lg[i] = lg[i/2]+1;
}

void build(vector<int> &lcp)
{
 int n = lcp.size();
 for (int i=0; i<n; i++)
 st[i][0] = lcp[i];

 for (int k=1; k<K; k++)
 for (int i=0; i<n; i++)
 {
 st[i][k] = st[i][k-1];
 int nxt = i + (1<<(k-1));
 if (nxt >= n) continue;
 st[i][k] = min(st[i][k], st[nxt][k-1]);
 }
 ///minimum of lcp[l.....r]
 int get(int l, int r)
 {
 int k = lg[r-l+1];
 return min(st[l][k], st[r-(1<<k)+1][k]);
 }
}

int main()
{
 int i,j,k,n,m;
 pre();
 string s;
 cin>>s;
 n = s.size();

 vector<int> sa = suffix_array_construction(s);
 vector<int> lcp = lcp_construction(s, sa);
 build(lcp);
 for(i=0;i<n;i++) cout<<sa[i]<<nl;
 return 0;
}

```

**Suffix Automaton**

/\*The simplest, and at the same time the most important property of the suffix automaton, is that it contains information about all the substrings  $s$ . Namely, any path from the initial state  $t_0$ , if we write out the labels of arcs along this path, necessarily forms a **substring of the string  $s$** . Conversely, to any substring of a string there  $s$  corresponds some path starting in the initial state  $t_0$ .

If we go from the initial state  $t_0$  along any path to any terminal state, and write down the labels of all the arcs, we get a string, which must be one of the suffixes of the string  $s$  \*/

```
///number of states or nodes in a suffix automaton is equal to the
/// number of equivalence classes i.e. endpos-equivalent classes
/// among all substrings
struct node
{
 int len; //largest string length of the corresponding endpos-
 equivalence class
 int link; //leads to the state that corresponds to the longest
 suffix of w
 //that is another endpos-equivalent class.
 int firstpos; //1-indexed end position of the first occurrence of
 the largest string length of the
 //corresponding endpos-equivalent class
 map<char,int>nxt;
};
//all suffix links of the last node are terminal nodes including the last
node
//minlen(v)=smallest string length of the corresponding endpos-
equivalent class=len(link(v))+1
//every node represents len-minlen+1 strings
```

```
const int MX=mxn*2;
node t[MX];
int sz,last;
void init()
{
 sz=last=0;
 t[0].len=0;
 t[0].firstpos=0;
 t[0].link=-1;
 sz++;
}
ll cnt[MX];//number of times i-th node occurs in the string
vpii v;
set<pii>nodes;
void add_letter(char ch)
{
 int cur=sz++;
 t[cur].len=t[last].len+1;
 t[cur].firstpos=t[cur].len;
 cnt[cur]=1;
 nodes.insert({t[cur].len,cur});
 int p;
 for(p=last;p!=-1&&!t[p].nxt.count(ch);p=t[p].link) t[p].nxt[ch]=cur;
 if(p==-1) t[cur].link=0;
 else{
 int q=t[p].nxt[ch];
 if(t[p].len+1==t[q].len) t[cur].link=q;
 else{
 int clone=sz++;
 t[clone].len=t[q].len;
 t[clone].firstpos=t[q].firstpos;
 t[clone].link=t[q].link;
 t[q].link=clone;
 t[clone].nxt=t[q].nxt;
 t[q].nxt.clear();
 nodes.insert({t[clone].len,clone});
 }
 }
}
```

```

t[clone].len=t[p].len+1;
t[clone].nxt=t[q].nxt;
t[clone].link=t[q].link;
t[clone].firstpos=t[q].firstpos;
cnt[clone]=0;
nodes.insert({t[clone].len,clone});
for(;p!=-1&&t[p].nxt[ch]==q;p=t[p].link) t[p].nxt[ch]=clone;
t[q].link=t[cur].link=clone;
}
}
last=cur;
}
// dcnt[MX]; //number of distinct substrings in the subtree of
node i
// dist_sub(int u) //number of distinct substrings of the string
{
 // ans=1;
 if(dcnt[u]) return dcnt[u];
 for(auto x:t[u].nxt){
 char ch=x.F;
 ans+=1LL*dist_sub(t[u].nxt[ch]);
 }
 return dcnt[u]=ans;
}
///returns the lexicographically k-th substring pos in 1-indexed
///O(n)
pii kth_Path(int k)
{
 int len = 0;

```

```

int cur = 0;
int pos = -1;
for(; k; --k){
 int s = 0, p = cur;
 for(auto it:t[cur].nxt){
 if(dcnt[it.S] + s < k) s += dcnt[it.S];
 else{
 len++, cur = it.second, pos = t[it.second].firstpos;
 break;
 }
 }
 if(cur == p) break;
 k -= s;
}
if(k == 0) return MP(pos - len + 1, pos);
else return MP(-1, -1);
}
int lcs(string s)
{
 int cur=0,ans=0,len=0,pos=0;
 for(int i=0;i<s.size();i++){
 while(cur&&!t[cur].nxt.count(s[i])){
 cur=t[cur].link;
 len=t[cur].len;
 }
 if(t[cur].nxt.count(s[i])){
 cur=t[cur].nxt[s[i]];
 len++;
 }
 }
}

```

```

 if(len>ans) ans=len,pos=i;
 }
 string sub=s.substr(pos-ans+1,ans);
 return ans;
}
int main()
{
 fast;
 int i,j,n,m,k,q;
 string s;
 cin>>s;
 n=s.size();
 init();
 for(i=0;i<n;i++) add_letter(s[i]);
 for(auto it=nodes.rbegin();it!=nodes.rend();++it)
 cnt[t[(*it).S].link]+=cnt[(*it).S];
 dist_sub(0);
 cout<<dcnt[0]-1<<nl;
 k=2;
 pii pos=kth_Path(k);
 if(pos.F==-1) cout<<"no such string\n";
 else cout<<s.substr(pos.F-1,pos.S-pos.F+1)<<nl;
 cout<<lcs("abc")<<nl;
 for(i=1;i<sz;i++) cout<<t[i].firstpos<<' '<<t[i].len<<' '<<cnt[i]<<nl;
 ///longest repeated substring
 ///for bababa ans is 4(baba)
 int ans=0;
 for(i=1;i<sz;i++) if(cnt[i]>1) ans=max(ans,t[i].len);
 cout<<ans<<nl;
}

```

```

 return 0;
 }

Suffix Tree
string s;
int n;

struct node {
 int l, r, par, link;
 map<char,int> next;

 node (int l=0, int r=0, int par=-1)
 : l(l), r(r), par(par), link(-1) {}

 int len() { return r - l; }

 int &get (char c) {
 if (!next.count(c)) next[c] = -1;
 return next[c];
 }
};

node t[N];
int sz;

struct state {
 int v, pos;
 state (int v, int pos) : v(v), pos(pos) {}

};

state ptr (0, 0);

state go (state st, int l, int r) {

```

```

while (l < r) {
 if (st.pos == t[st.v].len()) {
 st = state (t[st.v].get(s[l]), 0);
 if (st.v == -1) return st;
 }
 else {
 if (s[t[st.v].l + st.pos] != s[l])
 return state (-1, -1);
 if (r-l < t[st.v].len() - st.pos)
 return state (st.v, st.pos + r-l);
 l += t[st.v].len() - st.pos;
 st.pos = t[st.v].len();
 }
 return st;
}

int split (state st) {
 if (st.pos == t[st.v].len())
 return st.v;
 if (st.pos == 0)
 return t[st.v].par;
 node v = t[st.v];
 int id = sz++;
 t[id] = node (v.l, v.l+st.pos, v.par);
 t[v.par].get(s[v.l]) = id;
 t[id].get(s[v.l+st.pos]) = st.v;
 t[st.v].par = id;
 t[st.v].l += st.pos;
 return id;
}

} }

int get_link (int v) {
 if (t[v].link != -1) return t[v].link;
 if (t[v].par == -1) return 0;
 int to = get_link (t[v].par);
 return t[v].link = split (go (state(to,t[to].len()), t[v].l +
(t[v].par==0), t[v].r));
}

void tree_extend (int pos) {
 for (;;) {
 state nptr = go (ptr, pos, pos+1);
 if (nptr.v != -1) {
 ptr = nptr;
 return;
 }
 int mid = split (ptr);
 int leaf = sz++;
 t[leaf] = node (pos, n, mid);
 t[mid].get(s[pos]) = leaf;
 ptr.v = get_link (mid);
 ptr.pos = t[ptr.v].len();
 if (!mid) break;
 }
}

```

```

void build_tree() {
 sz = 1;
 for (int i=0; i<n; ++i)
 tree_extend (i);
}
void dfs(int i)
{
 cout<<i<<' '<<t[i].l<<' '<<t[i].r<<nl;
 for(auto x:t[i].next) dfs(x.S);
}
int main()
{
 BeatMeScanf;
 int i,j,k,m;
 s="banana";
 s+="\$";
 n=s.size();
 build_tree();
 dfs(0);
 return 0;
}

```

## 66. Palindromes

### Palindromic Tree

```

///There can be at most n unique palindromes for a string of size n
struct node //a node is a palindromic substring of the string
{
 int nxt[26]; //link to the palindrome which is formed by adding

```

```

 ///next[i] in both side of this palindrome
 int len; //length of the palindrome
 int st,en; //starting and ending index of the node
 int suflink; //link to the maximum proper suffix palindrome of
the node
 int cnt; //stores the length of the suffix link chain from it
(including this node)
 ///i.e. the number of palindromic suffix of this node
 int oc; //stores the number of occurrence of the node
};
string s;
node t[N];
int n; //size of string
int sz; //indicates size of the tree
int suf; //index of maximum suffix palindrome
void init()
{
 sz=2,suf=2;
 t[1].len=-1,t[1].suflink=1; //node 1- root with length -1
 t[2].len=0,t[2].suflink=1; //node 2- root with length 0 i.e null
palindrome
}
/// return if creates a new palindrome
int add_letter(int pos)
{
 //find the maximum suffix of the prefix+s[pos]
 int cur=suf,curlen=0;
 int ch=s[pos]-'a';
 while(1){

```

```

curlen=t[cur].len;
if(pos-1-curlen>=0&&s[pos-1-curlen]==s[pos]) break;
cur=t[cur].suflink;
}
//if the node is not created yet then create the new node
if(t[cur].nxt[ch]){
 suf=t[cur].nxt[ch];
 t[suf].oc++;
 return 0;
}
sz++;
suf=sz;
t[sz].oc=1;
t[sz].len=t[cur].len+2;
t[cur].nxt[ch]=sz;
t[sz].en=pos;
t[sz].st=pos-t[sz].len+1;
if(t[sz].len==1){
 t[sz].suflink=2;
 t[sz].cnt=1;
 return 1;
}
while(1){
 cur=t[cur].suflink;
 curlen=t[cur].len;
 if(pos-1-curlen>=0&&s[pos-1-curlen]==s[pos]){
 t[sz].suflink=t[cur].nxt[ch];
 break;
 }
}

```

```

 }
 t[sz].cnt=1+t[t[sz].suflink].cnt;
 return 1;
}
int main()
{
 fast;
 int i,j,k,n,m;
 ll ans=0;//number of palindromic substrings of a string (not unique
 palindrome)
 cin>>s;
 n=s.size();
 init();
 for(i=0;i<n;i++){
 add_letter(i);
 ans+=1LL*t[suf].cnt;
 }
 cout<<ans<<nl;
 for(i=sz;i>=3;i--) t[t[i].suflink].oc+=t[i].oc;
 for(i=3;i<=sz;i++) cout<<t[i].st<<' '<<t[i].en<<' '<<t[i].oc<<nl;
 //for multiple input clear all the (0..sz) nodes nxt array
 return 0;
}

```

### Manacher's Algorithm

```

int main()
{
 BeatMeScanf;
 int i,j,k,n,m;

```

```

string s;
cin>>s;
n=s.size();
vector<int> d1(n); //maximum odd length palindrome centered at
i
 //here d1[i]=the palindrome has d1[i]-1 right
characters from i
 //e.g. for aba, d1[1]=2;
for (int i = 0, l = 0, r = -1; i < n; i++) {
 int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
 while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
 k++;
 }
 d1[i] = k--;
 if (i + k > r) {
 l = i - k;
 r = i + k;
 }
}
vector<int> d2(n); //maximum even length palindrome centered
at i
 //here d2[i]=the palindrome has d2[i]-1 right
characters from i
 //e.g. for abba, d2[2]=2;
for (int i = 0, l = 0, r = -1; i < n; i++) {
 int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
 while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
 k++;
 }
}

```

```

d2[i] = k--;
if (i + k > r) {
 l = i - k - 1;
 r = i + k ;
}
for(i=0;i<n;i++) cout<<d1[i]<<' ';
cout<<nl;
for(i=0;i<n;i++) cout<<d2[i]<<' ';
cout<<nl;
///number of palindromes
ll ans=0;
for(i=0;i<n;i++){
 ans+=1LL*d1[i];
 ans+=1LL*d2[i];
}
cout<<ans<<nl;
///
return 0;
}

```

### Number of Palindromes in range

```

///Given a string and q queries of type (l,r) find number of odd and
even palindromes in substring [l...r]
vi d1,d2;
void manachers(string &s)
{
 int n=s.size();

```

```

d1=vector<int>(n); //maximum odd length palindrome centered
at i
 //here d1[i]=the palindrome has d1[i]-1 right
characters from i
 //e.g. for aba, d1[1]=2;
for (int i = 0, l = 0, r = -1; i < n; i++) {
 int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
 while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
 k++;
 }
 d1[i] = k--;
 if (i + k > r) {
 l = i - k;
 r = i + k;
 }
}
d2=vector<int>(n); //maximum even length palindrome centered
at i
 //here d2[i]=the palindrome has d2[i]-1 right
characters from i
 //e.g. for abba, d2[2]=2;
for (int i = 0, l = 0, r = -1; i < n; i++) {
 int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
 while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
 k++;
 }
 d2[i] = k--;
 if (i + k > r) {
 l = i - k - 1;
 }
}

```

```

r = i + k;
}
}
}
const int MAXN = (int)1e5+9;
const int MAXV = (int)1e5+9; //maximum value of any element in
array

//array values can be negative too, use appropriate minimum and
maximum value
struct wavelet_tree
{
 int lo, hi;
 wavelet_tree *l, *r;
 int *b, bsz, csz;
 ll *c; // c holds the prefix sum of elements

 wavelet_tree() { lo = 1; hi = 0; bsz = 0; csz = 0; l = NULL; r = NULL; }

 void init(int *from, int *to, int x, int y)
 {
 lo = x, hi = y;
 if (from >= to) return;
 int mid = (lo + hi) >> 1; auto f = [mid](int x) { return x
 <= mid; };
 b = (int*)malloc((to - from + 2) * sizeof(int)); bsz = 0;
 b[bsz++] = 0;
 }
}
```

```

c = (ll*)malloc((to - from + 2) * sizeof(ll)); csz = 0;
c[csz++] = 0;
 for(auto it = from; it != to; it++){
 b[bsz] = (b[bsz - 1] + f(*it));
 c[csz] = (c[csz - 1] + 1LL * (*it));
 bsz++;
 csz++;
 }
if(hi==lo) return;
 auto pivot = stable_partition(from, to, f);
 l = new wavelet_tree();
 l->init(from, pivot, lo, mid);
 r = new wavelet_tree();
 r->init(pivot, to, mid+1, hi);
}
///kth smallest element in [l, r]
///for array [1,2,1,3,5] 2nd smallest is 1 and 3rd smallest is 2
int kth(int l, int r, int k)
{
 if(l > r) return 0;
 if(lo == hi) return lo;
 int inLeft = b[r] - b[l - 1], lb = b[l - 1], rb = b[r];
 if(k <= inLeft) return this->l->kth(lb + 1, rb, k);
 return this->r->kth(l - lb, r - rb, k - inLeft);
}
///count of numbers in [l, r] Less than or equal to k
int LTE(int l, int r, int k)
{
 if(l > r || k < lo) return 0;
}

```

```

 if(hi <= k) return r - l + 1;
 int lb = b[l - 1], rb = b[r];
 return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r -
rb, k);
}
///count of numbers in [l, r] equal to k
int count(int l, int r, int k)
{
 if(l > r || k < lo || k > hi) return 0;
 if(lo == hi) return r - l + 1;
 int lb = b[l - 1], rb = b[r];
 int mid = (lo + hi) >> 1;
 if(k <= mid) return this->l->count(lb + 1, rb, k);
 return this->r->count(l - lb, r - rb, k);
}
///sum of numbers in [l ,r] less than or equal to k
int sum(int l, int r, int k) {
 if(l > r or k < lo) return 0;
 if(hi <= k) return c[r] - c[l-1];
 int lb = b[l-1], rb = b[r];
 return this->l->sum(lb+1, rb, k) + this->r->sum(l-lb, r-
rb, k);
}
~wavelet_tree()
{
 delete l;
 delete r;
}
};
```

```

|| get(int l,int r)
{
 return 1LL*r*(r+1)/2-1LL*(l-1)*l/2;
}
wavelet_tree oddl,oddr;
|| odd(int l,int r)
{
 int m=(l+r)/2;
 int c=1-l;
 int less_=oddl.LTE(l,m,c);
 || ansl=get(l,m)+oddl.sum(l,m,c)+1LL*(m-l+1-less_)*c;
 c=1+r;
 less_=oddr.LTE(m+1,r,c);
 || ansr=-get(m+1,r)+oddr.sum(m+1,r,c)+1LL*(r-m-less_)*c;
 return ansl+ansr;
}
wavelet_tree evenl,evenr;
|| even(int l,int r)
{
 int m=(l+r)/2;
 int c=-l;
 int less_=evenl.LTE(l,m,c);
 || ansl=get(l,m)+evenl.sum(l,m,c)+1LL*(m-l+1-less_)*c;
 c=1+r;
 less_=evenr.LTE(m+1,r,c);
 || ansr=-get(m+1,r)+evenr.sum(m+1,r,c)+1LL*(r-m-less_)*c;
 return ansl+ansr;
}
int a[N],b[N],c[N],d[N];

```

```

int main()
{
 BeatMeScanf;
 int i,j,k,n,m,q,l,r;
 string s;
 cin>>s;
 n=s.size();
 manachers(s);
 for(i=1;i<=n;i++) a[i]=d1[i-1]-i;
 oddl.init(a+1,a+n+1,-MAXV,MAXV);
 for(i=1;i<=n;i++) b[i]=d1[i-1]+i;
 oddr.init(b+1,b+n+1,-MAXV,MAXV);
 for(i=1;i<=n;i++) c[i]=d2[i-1]-i;
 evenl.init(c+1,c+n+1,-MAXV,MAXV);
 for(i=1;i<=n;i++) d[i]=d2[i-1]+i;
 evenr.init(d+1,d+n+1,-MAXV,MAXV);

 cin>>q;
 while(q--){
 cin>>l>>r;
 cout<<odd(l,r)+even(l,r)<<nl;
 }
 return 0;
}

```

### Minimum Palindrome Factorization

//There can be at most n unique palindromes for a string of size n  
 struct node        ///a node is a palindromic substring of the string

```

{
 int nxt[26]; ///link to the palindrome which is formed by adding
 ///next[i] in both side of this palindrome
 int len; ///length of the palindrome
 int st,en; ///starting and ending index of the node
 int suflink; ///link to the maximum proper suffix palindrome of
the node
 int cnt; ///stores the length of the suffix link chain from it
(including this node)
 ///i.e. the number of palindromic suffix of this node
 int oc; ///stores the number of occurrence of the node
 int diff; ///difference diff[v] = len[v]-len[suflink[v]]
 int serieslink; ///longest suffix-palindrome of v having the
difference unequal to diff[v].
};

string s;
node t[N];
int n; ///size of string
int sz; ///indicates size of the tree
int suf; ///index of maximum suffix palindrome
void init()
{
 t[1].diff=t[2].diff=0;
 sz=2,suf=2;
 t[1].len=-1,t[1].suflink=t[1].serieslink=1; ///node 1- root with
length -1
 t[2].len=0,t[2].suflink=1;t[2].serieslink=2; ///node 2- root with
length 0 i.e null palindrome
}

```

```

/// return if creates a new palindrome
int add_letter(int pos)
{
 ///find the maximum suffix of the prefix+s[pos]
 int cur=suf,curlen=0;
 int ch=s[pos]-'a';
 while(1){
 curlen=t[cur].len;
 if(pos-1-curlen>=0&&s[pos-1-curlen]==s[pos]) break;
 cur=t[cur].suflink;
 }
 ///if the node is not created yet then create the new node
 if(t[cur].nxt[ch]){
 suf=t[cur].nxt[ch];
 t[suf].oc++;
 return 0;
 }
 sz++;
 suf=sz;
 t[sz].oc=1;
 t[sz].len=t[cur].len+2;
 t[cur].nxt[ch]=sz;
 t[sz].en=pos;
 t[sz].st=pos-t[sz].len+1;
 if(t[sz].len==1){
 t[sz].diff=1;
 t[sz].serieslink=2;
 t[sz].suflink=2;
 t[sz].cnt=1;
 }
}

```

```

 return 1;
}
while(1){
 cur=t[cur].suflink;
 curlen=t[cur].len;
 if(pos-1-curlen>=0&&s[pos-1-curlen]==s[pos]){
 t[sz].suflink=t[cur].nxt[ch];
 break;
 }
 t[sz].cnt=1+t[t[sz].suflink].cnt;
 t[sz].diff=t[sz].len-t[t[sz].suflink].len;
 if(t[sz].diff==t[t[sz].suflink].diff)
 t[sz].serieslink=t[t[sz].suflink].serieslink;
 else t[sz].serieslink=t[sz].suflink;
 return 1;
}

int ans[N][2],dp[N][2],kfact[N];

int getmin(int pos,int v,int k)
{
 dp[v][k]=ans[pos-t[v].len][k];
 if (t[v].diff == t[t[v].suflink].diff){
 dp[v][k] = min(dp[v][k], dp[t[v].suflink][k]);
 dp[v][k] = min(dp[v][k], ans[pos - t[t[v].serieslink].len - t[v].diff][k]);
 }
 return dp[v][k] + 1;
}

}

void calc(int pos,int cur)
{
 pos++;
 ans[pos][0]=1e9;
 ans[pos][1]=1e9;
 for(int v=cur;t[v].len>0;v=t[v].serieslink){
 ans[pos][0]=min(ans[pos][0],getmin(pos,v,1));
 ans[pos][1]=min(ans[pos][1],getmin(pos,v,0));
 }
}

///ans[i][0]=minimal even k (or -2 if it doesn't exist) such that
///we can split string s[1.. i] into k palindromes.

///ans[i][1]=minimal odd k (or -1 if it doesn't exist) such that
///we can split string s[1.. i] into k palindromes.

///k-palindrome factorization=splitting the string into k non-empty
non-intersecting palindromes

///minimum palindrome factorization=minimum k such that k-
palindrome factorization exist

int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
}

```

```

cin>>s;
n=s.size();
init();
ans[0][1]=1e9;
dp[2][1]=1e9;
for(i=0;i<n;i++){
 add_letter(i);
 calc(i,suf);
}
for(i=1;i<=n;i++) cout<<(ans[i][1]==1e9?-1:ans[i][1])<<''
'<<(ans[i][0]==1e9?-2:ans[i][0])<<nl;

///minimum palindrome factorization
int res=min(ans[n][0],ans[n][1]);
cout<<res<<nl;

///if k-palindrome factorization is possible or not
for(i=ans[n][1];i<=n;i+=2) kfact[i]=1;
for(i=ans[n][0];i<=n;i+=2) kfact[i]=1;
for(i=1;i<=n;i++) cout<<(kfact[i]?"YES\n":"NO\n");
return 0;
}

```

## 67. Lyndon Factorization And Minimum Rotation

///Complexity:  $O(n)$   
 ///A string is called simple (or a Lyndon word), if it is strictly smaller than

```

///any of its own nontrivial suffixes. Examples of simple strings are: a,
b, ab, aab, abb, ababb,
///abcd. It can be shown that a string is simple, if and only if it is strictly
smaller
///than all its nontrivial cyclic shifts.
///Next, let there be a given string s. The Lyndon factorization of the
///string s is a factorization s=w1w2...wk, where all strings wi are
simple,
///and they are in non-increasing order w1>=w2>=....>=wk.
///It can be shown, that for any string such a factorization exists and
that it is unique.
///Here we use Duval algorithm for finding lyndon factorization

vector<string> duval(string const& s) {
 int n = s.size();
 int i = 0;
 vector<string> factorization;
 while (i < n) {
 int j = i + 1, k = i;
 while (j < n && s[k] <= s[j]) {
 if (s[k] < s[j])
 k = i;
 else
 k++;
 j++;
 }
 while (i <= k) {
 factorization.push_back(s.substr(i, j - k));
 i += j - k;
 }
 }
}
```

```

 }
}

return factorization;
}

string min_cyclic_string(string s) {
 s += s;
 int n = s.size();
 int i = 0, ans = 0;
 while (i < n / 2) {
 ans = i;
 int j = i + 1, k = i;
 while (j < n && s[k] <= s[j]) {
 if (s[k] < s[j])
 k = i;
 else
 k++;
 j++;
 }
 while (i <= k)
 i += j - k;
 }
 return s.substr(ans, n / 2);
}

int main()
{
 BeatMeScanf;
 || i,j,k,n,m;
 string s;
 cin>>s;
}

```

```

cout<<min_cyclic_string(s)<<nl;
return 0;
}

```

## 68. Expression Parsing

```

///returns precedence of operators
int precedence(char symbol)
{
 switch(symbol)
 {
 case '+':
 case '-':
 return 2;
 break;
 case '*':
 case '/':
 return 3;
 break;
 case '^':
 return 4;
 break;
 case '(':
 case ')':
 case '#':
 return 1;
 break;
 }
}

```

```

///check whether the symbol is operator?
int isOperator(char symbol)
{
 switch(symbol)
 {
 case '+':
 case '-':
 case '*':
 case '/':
 case '^':
 case '(':
 case ')':
 return 1;
 break;
 default:
 return 0;
 }
}

```

```

///converts infix expression to postfix
string convert(string infix)
{
 stack<char>st;
 string postfix="";
 st.push('#');
 for(auto ch:infix){
 if(isOperator(ch) == 0) postfix+=ch;
 }
}

```

```

else
{
 if(ch == '(') st.push(ch);
 else
 {
 if(ch == ')')
 {
 while(st.top() != '(')
 {
 postfix+= st.top();
 st.pop();
 }
 st.pop();
 }
 else
 {
 if(precedence(ch)>precedence(st.top())) st.push(ch);
 else
 {
 while(precedence(ch)<=precedence(st.top()))
 {
 postfix+= st.top();
 st.pop();
 }
 st.push(ch);
 }
 }
 }
}

```

```

}

while(st.top() != '#')
{
 postfix+=st.top();
 st.pop();
}
return postfix;
}

///evaluates postfix expression
int evaluate(string postfix)
{
 int op1,op2;
 stack<int>st;
 for(auto ch:postfix){
 if(isdigit(ch)) st.push(ch-'0');
 else
 {
 ///Operator,pop two operands
 op2 = st.top();
 st.pop();
 op1 = st.top();
 st.pop();
 switch(ch)
 {
 case '+':
 st.push(op1+op2);
 break;
 case '-':
 st.push(op1-op2);
 break;
 case '*':
 st.push(op1*op2);
 break;
 case '/':
 st.push(op1/op2);
 break;
 default:
 st.push((int)(pow(1.0*op1,1.0*op2)+eps));
 }
 }
 return st.top();
 }
 int main()
 {
 BeatMeScanf;
 int i,j,k,n,m;
 string infix= "(9/3)*1*(2+3)+7-9^2";
 string postfix=convert(infix);
 cout<<infix<<nl<<postfix<<nl<<evaluate(postfix)<<nl;
 return 0;
 }
}

```

## GRAPH THEORY

### 69. The Art of Problem Solving

- Try to think the question backward.
- Try to solve the question backward

### 70. Notes

- **Cayley's formula**

Cayley's formula states that the number of spanning trees in a complete labeled graph with  $n$  vertices is equal to:

$$n^{n-2}$$

- Sum of degrees of all nodes in any tree is:

$$2n - 2$$

- If a graph has  $n$  nodes and  $k$  connected components then number of ways to make the graph connected with minimum number of edges is:

$$s_1 \cdot s_2 \dots s_k \cdot n^{k-2}$$

where  $s_i$  is the size of  $i$ -th component.

Be aware for  $k=1$ . For  $k=1$  ans=1.

- **Number of paths of a fixed length**

We are given a directed, unweighted graph  $G$  with  $n$  vertices and we are given an integer  $k$ . The task is the following: for each pair of vertices  $(i,j)$  we have to find the number of paths of length  $k$  between these vertices. Paths don't have to be simple, i.e. vertices and edges can be visited any number of times in a single path.

We assume that the graph is specified with an adjacency matrix, i.e. the matrix  $G[][]$  of size  $n \times n$ , where each element  $G[i][j]$  equal to 1 if the vertex  $i$  is connected with  $j$  by an edge, and 0 is they are not connected by an edge. The following algorithm works also in the case of multiple edges: if some pair of vertices  $(i,j)$  is connected with  $m$  edges, then we can record this in the adjacency matrix by setting  $G[i][j]=m$ . Also the algorithm works if the graph contains loops (a loop is an edge that connect a vertex with itself).

Let  $C_k[i][j]$  be the answer for pair  $(i,j)$  of length  $k$ . then,

$$C_k = G^k$$

It remains to note that the matrix products can be raised to a high power efficiently using Binary exponentiation. This gives a solution with  $O(n^3 \log k)$  complexity.

- **If  $u$  is ancestor of  $v$ ?**

```
st[u]=++T,en[u]=T;
return st[u]<=st[v]&&en[u]>=en[v];
```

- Lca of node  $[l, l+1, \dots, r] = \text{lca}(\text{node with minimum start time}, \text{node with maximum start time})$
- If  $G$  is a graph with  $\delta(G) \geq 2$ , then the graph  $G$  must contain a cycle where  $\delta(G)$  refers to the minimum degree of a graph.
- A Graph  $G$  where each vertex has an even degree can be split into cycles by which no cycle has a common edge.
- a planar graph is a graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. In other words, it can be drawn in such a way that no edges cross each other.
- Euler's formula states that if a finite, connected, planar graph is drawn in the plane without any edge intersections, and  $v$  is

the number of vertices,  $e$  is the number of edges and  $f$  is the number of faces (regions bounded by edges, including the outer, infinitely large region), then  $v-e+f=2$

If  $G$  is planar then  $n - m + f = 2$ , so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree  $\leq 5$ .

- In an undirected graph, a widest path may be found as the path between the two vertices in the maximum spanning tree of the graph, and a minimax path(minimum of maximum edge in a path) may be found as the path between the two vertices in the minimum spanning tree.

### • König's theorem

it states that, in any bipartite graph, the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover.

### • Path from node to root query

Consider following problem: you have a tree and there are lots of queries of kind add number on subtree of some vertex or calculate sum on the path between some vertices. HLD? Damn, no! Let's consider two euler tours: in first we write the vertex when we enter it, in second we write it when we exit from it. We can see that difference between prefixes including subtree of  $v$  from first and second tours will exactly form vertices from  $v$  to the root. Thus problem is reduced to adding number on segment and calculating sum on prefixes. Worth mentioning that there are alternative approach which is to keep in each vertex linear function from its height and update

such function in all  $v$ 's children, but it is hard to make this approach more general.

- Given a bipartite graph, find out minimum number of colors to color all the edges such that no two adjacent edges have a same color assigned. Number of minimum colors equals the max degree of a vertex in the bipartite graph.

### • Line Graph

the line graph of an undirected graph  $G$  is another graph  $L(G)$  that represents the adjacencies between edges of  $G$ .

Given a graph  $G$ , its line graph  $L(G)$  is a graph such that-

- \*each vertex of  $L(G)$  represents an edge of  $G$ ; and
- \*two vertices of  $L(G)$  are adjacent if and only if their corresponding edges share a common endpoint ("are incident") in  $G$ .

Properties-

\*A line graph has an articulation point if and only if the underlying graph has a bridge for which neither endpoint has degree one

\*For a graph  $G$  with  $n$  vertices and  $m$  edges, the number of vertices of the line graph  $L(G)$  is  $m$ , and the number of edges of  $L(G)$  is half the sum of the squares of the degrees of the vertices in  $G$ , minus  $m$

\*A maximum independent set in a line graph corresponds to maximum matching in the original graph.

\*The edge chromatic number of a graph  $G$  is equal to the vertex chromatic number of its line graph  $L(G)$

\*If a graph  $G$  has an Euler cycle, that is, if  $G$  is connected and has an even number of edges at each vertex, then the line graph of  $G$  is Hamiltonian. However, not all Hamiltonian cycles

in line graphs come from Euler cycles in this way; for instance, the line graph of a Hamiltonian graph G is itself Hamiltonian, regardless of whether G is also Eulerian

- **Graph Coloring**

The chromatic polynomial counts the number of ways a graph can be colored using no more than a given number of colors.

**Chromatic polynomials for certain graphs**

|                        |                                        |
|------------------------|----------------------------------------|
| Triangle $K_3$         | $t(t - 1)(t - 2)$                      |
| Complete graph $K_n$   | $t(t - 1)(t - 2) \cdots (t - (n - 1))$ |
| Tree with $n$ vertices | $t(t - 1)^{n-1}$                       |
| Cycle $C_n$            | $(t - 1)^n + (-1)^n(t - 1)$            |

Chromatic polynomial of triangle graph is  $t(t-2)(t-3)$

- **Minimum Weight Cycle**

undirected graph: for each edge i we want the minimum weight cycle containing this edge. first remove this edge( $u_i, v_i, w_i$ ). find shortest distance between  $u_i$  to  $v_i$  in the modified graph. let it be  $W$ . answer for that edge is  $W+w_i$ . Ultimate answer is minimum of all of the edges.

directed graph: same to above

- **Number of ways to make a graph connected**

A graph with n nodes and k connected components each having size  $s_1, s_2, \dots, s_k$  can be connected with minimum number of edges in following ways:

$$s_1 \cdot s_2 \cdot \dots \cdot s_k \cdot (s_1 + s_2 + \dots + s_k)^{k-2} = s_1 \cdot s_2 \cdot \dots \cdot s_k \cdot n^{k-2}.$$

- **Negative Cycles**

Let us consider the situation where we have no negative cycle. Now, we can assign an integer  $p_i$  to each vertex so that the following condition is satisfied:

- ✓ For each edge  $e = (i \rightarrow j)$ ,  $p_j \leq p_i + \text{weight}(e)$  holds.

This can be achieved by, for example, letting  $p_i =$  (the shortest distance from Vertex 0 to Vertex i).

On the other hand, we can see that, if we can assign  $p_i$  to the vertices so that this condition is satisfied, the graph contains no negative cycle. (If we assume there is a negative cycle, adding  $p_j - p_i \leq +\text{weight}(e)$  along it would lead to contradiction.)

## 71. Strongly Connected Components

### Condensation Graph

//// A condensation graph is a graph containing every strongly connected component as one vertex

//this code also describes- given a directed graph return minimum number of edges to be added so the whole graph become a SCC  
bool vis[N];

vi g[N], r[N], cn[N], vec;

void dfs1(int u)

{

vis[u]=1;

for(auto v:g[u]) if(!vis[v]) dfs1(v);

vec.eb(u);

}

vi comp;

void dfs2(int u)

```

{
 comp.eb(u);
 vis[u]=1;
 for(auto v:r[u]) if(!vis[v]) dfs2(v);
}
int idx[N],in[N],out[N];
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v;
 cin>>n>>m;
 for(i=1;i<=m;i++){
 cin>>u>>v;
 g[u].eb(v);
 r[v].eb(u);
 }
 for(i=1;i<=n;i++) if(!vis[i]) dfs1(i);
 rev(vec);
 mem(vis,0);
 int scc=0;//number of SCC
 for(auto u:vec){
 if(!vis[u]){
 comp.clear();
 dfs2(u);
 //here we have all the nodes in this component
 scc++;
 for(auto x:comp) idx[x]=scc;
 }
 }
}

```

```

for(u=1;u<=n;u++){
 for(auto v:g[u]){
 if(idx[u]!=idx[v]){
 in[idx[v]]++,out[idx[u]]++;
 cn[idx[u]].eb(idx[v]);
 }
 }
}
int needed_in=0,needed_out=0;
for(i=1;i<=scc;i++){
 if(!in[i]) needed_in++;
 if(!out[i]) needed_out++;
}
int ans=max(needed_in,needed_out);
if(scc==1) ans=0;//corner case
//answer for the corresponding problem;
cout<<ans<<nl;
//output the condensation graph
for(u=1;u<=scc;u++){
 cout<<u<<": ";
 for(auto v:cn[u]) cout<<v<<' ';
 cout<<nl;
}
return 0;
}

```

## 72. Articulation Points

```

bool vis[N];
int art[N];

```

```

vi g[N];
int dis[N],low[N],T;
void dfs(int u,bool isroot)
{
 dis[u]=low[u]=++T;
 vis[u]=1;
 int child=0;
 for(auto v:g[u]){
 if(!vis[v]){
 dfs(v,0);
 if(dis[u]<=low[v]&&!isroot) art[u]=1;
 low[u]=min(low[u],low[v]);
 child++;
 }
 else low[u]=min(low[u],dis[v]);
 }
 if(isroot&&child>1) art[u]=1;
}
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v;
 cin>>n>>m;
 for(i=1;i<=m;i++){
 cin>>u>>v;
 g[u].eb(v);
 g[v].eb(u);
 }
 for(i=1;i<=n;i++) if(!vis[i]) dfs(i,1);
}

```

```

int ans=0;
for(i=1;i<=n;i++) if(art[i]) ans++;
cout<<ans<<nl;
return 0;
}

```

## 73. Articulation Bridges

### Articulation Bridges Standard

```

bool vis[N];
vi g[N];
int dis[N],low[N],T;
set<pii>bridge;
void dfs(int u,int pre)
{
 low[u]=dis[u]=++T;
 vis[u]=1;
 for(auto v:g[u]){
 if(!vis[v]){
 dfs(v,u);
 low[u]=min(low[u],low[v]);
 if(low[v]>dis[u]) bridge.insert({min(u,v),max(u,v)});
 }
 else{
 if(v!=pre) low[u]=min(low[u],dis[v]);
 }
 }
}
int main()

```

```
{
 BeatMeScanf;
 int i,j,k,n,m,u,v;
 cin>>n>>m;
 for(i=1;i<=m;i++){
 cin>>u>>v;
 g[u].eb(v);
 g[v].eb(u);
 }
 dfs(1,0);
 int ans=bridge.size();
 cout<<ans<<nl;
 return 0;
}
```

### Articulation Bridges Online

```
///Given number of nodes n and q queries
///add edge (u,v)
///output the bridges in current graph
int n, bridges, par[N], bl[N], comp[N], sz[N];

void init() {
 for (int i=0; i<n; ++i) {
 bl[i] = comp[i] = i;
 sz[i] = 1;
 par[i] = -1;
 }
 bridges = 0;
}
```

```
}
int get (int v) {
 if (v== -1) return -1;
 return bl[v]==v ? v : bl[v]=get(bl[v]);
}

int get_comp (int v) {
 v = get(v);
 return comp[v]==v ? v : comp[v]=get_comp(comp[v]);
}

void make_root (int v) {
 v = get(v);
 int root = v,
 child = -1;
 while (v != -1) {
 int p = get(par[v]);
 par[v] = child;
 comp[v] = root;
 child=v; v=p;
 }
 sz[root] = sz[child];
}

int cu, u[N];
```

```

void merge_path (int a, int b) {
 ++cu;

 vector<int> va, vb;
 int lca = -1;
 for(;;) {
 if (a != -1) {
 a = get(a);
 va.pb (a);

 if (u[a] == cu) {
 lca = a;
 break;
 }
 u[a] = cu;

 a = par[a];
 }

 if (b != -1) {
 b = get(b);
 vb.pb (b);

 if (u[b] == cu) {
 lca = b;
 break;
 }
 u[b] = cu;
 }
 }
}

```

```

 b = par[b];
}

for (int i=0; i<va.size(); ++i) {
 bl[va[i]] = lca;
 if (va[i] == lca) break;
 --bridges;
}
for (int i=0; i<vb.size(); ++i) {
 bl[vb[i]] = lca;
 if (vb[i] == lca) break;
 --bridges;
}

void add_edge (int a, int b) {
 a = get(a); b = get(b);
 if (a == b) return;

 int ca = get_comp(a), cb = get_comp(b);
 if (ca != cb) {
 ++bridges;
 if (sz[ca] > sz[cb]) {
 swap (a, b);
 swap (ca, cb);
 }
 make_root (a);
 }
}

```

```

 par[a] = comp[a] = b;
 sz[cb] += sz[a];
 }
 else merge_path(a, b);
}
///1-indexed
int main()
{
 BeatMeScanf;
 int i,j,k,m,q,u,v;
 cin>>n>>q;
 init();
 while(q--){
 cin>>u>>v;
 add_edge(u,v);
 cout<<bridges<<nl;
 }
 return 0;
}

```

## Articulation Bridge Tree

```

///diameter of the Articulation Bridge Tree
vi g[N],gr[N];
bool vis[N];
int T,low[N],dis[N],d[N],par[N];
set<pii>bridge;
void dfs(int u,int pre)
{
 low[u]=dis[u]=++T;

```

```

 vis[u]=1;
 for(auto v:g[u]){
 if(!vis[v]){
 dfs(v,u);
 low[u]=min(low[u],low[v]);
 if(low[v]>dis[u]) bridge.insert({min(u,v),max(u,v)});
 }
 else{
 if(v!=pre) low[u]=min(low[u],dis[v]);
 }
 }
 int find_(int x)
 {
 if(par[x]==x) return x;
 return par[x]=find_(par[x]);
 }
 void merge_(int x,int y)
 {
 int u=find_(x);
 int v=find_(y);
 if(u!=v){
 if(rand()%2) par[u]=v;
 else par[v]=u;
 }
 }
 int bfs(int s)
 {
 mem(d,-1);

```

```

queue<int>q;
q.push(s);
d[s]=0;
int u=s;
while(!q.empty()){
 u=q.front();
 q.pop();
 for(auto v:gr[u]){
 if(d[v]==-1){
 q.push(v);
 d[v]=d[u]+1;
 }
 }
}
return u;
}

int main()
{
 fast;
 int i,j,k,n,m,u,v,ans=0;
 cin>>n>>m;
 for(i=1;i<=m;i++) cin>>u>>v,g[u].eb(v),g[v].eb(u);
 dfs(1,0);
 for(i=1;i<=n;i++) par[i]=i;
 for(u=1;u<=n;u++){
 for(auto v:g[u]){
 if(bridge.find({min(u,v),max(u,v)})==bridge.end())
 merge_(u,v);
 }
 }
}

```

```

 }
 for(auto p:bridge){
 u=p.F;
 v=p.S;
 int x=find_(u);
 int y=find_(v);
 gr[x].eb(y);
 gr[y].eb(x);
 }
 u=bfs(find_(1));
 v=bfs(find_(u));
 cout<<d[v]<<nl;
 return 0;
}

```

## 74. Biconnected Components

```

/// biconnected component of a given graph is the maximal
connected subgraph
///which does not contain any articulation vertices.

///1 Based,no problem in multiple edge and self loop
int dis[N],low[N];
int T,n;
vector<int> g[N]; //only g should be cleared
stack<pii> st;
vector<set<pii>> e; //biconnected components
void calc_bcc(int u, int v)
{
 int i, j, uu, vv, cur;

```

```

pii now;
set<pii>se;
while(!st.empty())
{
 now = st.top();
 st.pop();
 uu = now.first, vv = now.second;
 se.insert({uu, vv});
 if(u==uu && v==vv)
 break;
 if(u==vv && v==uu)
 break;
}
e.eb(se);
return;
}
void bcc(int u,int pre) // pre=-1 dhore call dite hobe(root ar parent
nai)
{
 dis[u] = low[u] = ++T;
 for(int i = 0 ; i<g[u].size() ; i++)
 {
 int v = g[u][i];
 if(v==pre) continue;
 if(dis[v]==0)
 {
 st.push(make_pair(u, v));
 bcc(v,u);
 low[u] = min(low[u],low[v]);
 }
 if(low[v]>=dis[u])
 {
 calc_bcc(u, v);
 }
 else if(dis[v] < dis[u])
 {
 low[u] = min(low[u],dis[v]);
 st.push(make_pair(u, v));
 }
 }
 return;
}
int main()
{
 BeatMeScanf;
 int i,j,k,m,u,v;
 cin>>n>>m;
 while(m--)
 {
 cin>>u>>v;
 g[u].eb(v);
 g[v].eb(u);
 }
 T=0;
 memset(dis,0,sizeof dis);
 for(i = 1; i <= n; i++)if(!dis[i]) bcc(i,-1);
 for(auto se:e){

```

```

for(auto edge:se) cout<<edge.F<<' '<<edge.S<<' ';
cout<<nl;
}
///if two nodes u and v have at least two vertex disjoint path i.e. if
two paths have
///no common vertices except u and v
///check if they are in same biconnected components
}

```

## 75. Block Cut Tree

```

///Any connected graph decomposes into a tree of biconnected
///components called the block-cut tree of the graph
///1 Based,no problem in multiple edge and self loop
int dis[N],low[N];
int T,n;
vector<int> g[N]; ///only g should be cleared
stack<pii>st;
vector<set<pii>>e;///biconnected components
void calc_bcc(int u, int v)
{
 int i, j, uu, vv, cur;
 pii now;
 set<pii>se;
 while(!st.empty())
 {
 now = st.top();
 st.pop();
 uu = now.first, vv = now.second;
 se.insert({uu, vv});
 }
}

```

```

if(u==uu && v==vv)
 break;
if(u==vv && v==uu)
 break;
}
//if(vec.size()<=1) return;
e.eb(se);
return;
}
int art[N];
void bcc(int u,int pre) /// pre=-1 dhore call dite hobe(root ar parent
nai)
{
 dis[u] = low[u] = ++T;
 int child=0;
 for(int i = 0 ; i<g[u].size() ; i++)
 {
 int v = g[u][i];
 if(v==pre) continue;
 if(dis[v]==0)
 {
 st.push(make_pair(u, v));
 bcc(v,u);
 low[u] = min(low[u],low[v]);
 if(low[v]>=dis[u])
 {
 if(pre!=-1) art[u]=1;
 calc_bcc(u, v);
 }
 }
 }
}

```

```

 child++;
 }
 else if(dis[v] < dis[u])
 {
 low[u] = min(low[u],dis[v]);
 st.push(make_pair(u, v));
 }
 if(pre==-1&&child>1) art[u]=1;
 return;
}
set<int> bt[N];//block cut tree
int id[N];
int main()
{
 BeatMeScanf;
 int i,j,k,m,u,v;
 cin>>n>>m;
 while(m--)
 {
 cin>>u>>v;
 g[u].eb(v);
 g[v].eb(u);
 }
 for(i = 1; i <= n; i++)if(!dis[i]) bcc(i,-1);
 int sz=0;
 for(i=1;i<=n;i++){
 if(art[i]){

```

```

 id[i]=++sz;
 cout<<i<<' ';
 }
 }
 cout<<nl;
 for(auto se:e){
 bool nonart=0;
 for(auto edge:se){
 u=edge.F,v=edge.S;
 if(!art[u] || !art[v]){
 nonart=1;
 break;
 }
 }
 if(nonart) ++sz;
 for(auto edge:se){
 u=edge.F,v=edge.S;
 if(art[u]&&art[v]){
 bt[id[u]].insert(id[v]);
 bt[id[v]].insert(id[u]);
 continue;
 }
 int dummy;
 if(!art[u]) id[u]=sz;
 else bt[id[u]].insert(sz),bt[sz].insert(id[u]);
 int dumm;
 if(!art[v]) id[v]=sz;
 else bt[id[v]].insert(sz),bt[sz].insert(id[v]);
 }
 }
}
```

```

 }
 for(i=1;i<=n;i++) cout<<id[i]<<' ';
 cout<<nl;
 for(i=1;i<=sz;i++){
 for(auto x:bt[i]) cout<<x<<' ';
 cout<<nl;
 }
}

```

## 76. Path Intersection

//Now, we just have to find length of intersection of paths a to b and c to d.

//Let's break down each path into two parts because finding intersection

//between straight chains would be easier. So, let u= lca(a,b) and v = lca(c,d),

// so total intersection would be intersection(u,a,v,c) + intersection(u,a,v,d)

///+ intersection(u,b,v,c) + intersection(u,b,v,d).

//Now, let's see how we can find intersection between straight chains - and

///- which can be managed by the following logic:

```

/// returns common path between u->a and v->b
///where dep[u]<dep[a] and dep[v]<dep[b]
pair<int, int> getCommonPath(int u,int a,int v,int b)
{

```

if(!isAncestor(v,a)) return MP(0,0); // v has to be an ancestor of a to have any common path

int x=lca(a,b);

if(L[v]<L[u]) // v is an ancestor of u, so the beginning of common path should be from u

{

if(isAncestor(u,x)) // if u is not an ancestor of x no common path exists

return MP(u,x); // u->x is the common path

}

else // u is an ancestor of v, so the beginning of common path should be from v

{

if(isAncestor(v,x)) // if v is not an ancestor of x no common path exists

return MP(v,x); // v->x is the common path

}

return MP(0,0);

}

## 77. Spanning Tree

### Notes

- If each edge has a distinct weight then there will be only one, unique minimum spanning tree
- **Number of Spanning trees in a weight tree:**

You can calculate the number of "spanning trees" with Kirchhoff's theorem. I guess this was  $O(N^3)$ . I don't know this well, but you can always google and copy paste.

You can reduce this problem to counting the number of spanning trees. This works because you can think edges with different costs individually. To elaborate, suppose you are doing something for all edges with cost  $X$ . **No matter what MST you chose, the components formed with edge cost < X is consistent (Think Kruskal's Algorithm!)** Then you can think such components as a single node. You should now use some cost  $X$  edges, to merge the nodes as much as possible. This is exactly the number of spanning trees.

## Prim's Minimum Spanning Tree

```
int g[N][N];
struct edge {
 int w = 1e9, to = -1;
};
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v,w;
 cin>>n>>m;
 for(i=1;i<=n;i++) for(j=1;j<=n;j++) g[i][j]=1e9;
 for(i=1;i<=m;i++){
 cin>>u>>v>>w;
 g[u][v]=w;
 }
}
```

```
g[v][u]=w;
}
int ans = 0;
vector<bool> selected(n+1);
vector<edge> e(n+1);
e[1].w = 0;
vector<pii>edges;
for (i=1;i<=n;++i) {
 u=-1;
 for (j=1;j<=n;++j) {
 if (!selected[j] && (u == -1 || e[j].w < e[u].w))
 u = j;
 }
 if (e[u].w == 1e9) {
 cout<<"No MST!"<< endl;
 exit(0);
 }
 selected[u] = true;
 ans += e[u].w;
 if (e[u].to != -1) edges.eb(u,e[u].to);

 for (int to = 1; to <= n; ++to) {
 if (g[u][to] < e[to].w) e[to] = {g[u][to], u};
 }
}
cout<<ans<<endl;
for(auto x:edges) cout<<x.F<<' '<<x.S<<nl;
return 0;
}
```

## Directed Minimum Spanning Tree

*//Complexity:O(nlogn)*

//Directed MST is finding a spanning arborescence of minimum weight  
 //An arborescence is a directed graph in which,a vertex u  
 //called the root and for any other vertex v, there is exactly one  
 //directed path from u to v. An arborescence is thus  
 //the directed-graph form of a rooted tree,

```
#define INF 100100000
struct edge
{
 int u,v,w,idx;
 edge(int u=0,int v=0,int w=0)
 {
 this->u = u;
 this->v = v;
 this->w = w;
 }
 bool operator < (const edge &b) const
 {
 return w<b.w;
 }
};
//here value of N should be maximum number of vertices
int n,m; //n = number of vertex,m=number of edges
vector<edge> e[N]; //edge u->v inserted into list of v.
vector<edge> edges; //all edges ,needed if used edges required.
vector<int>g[N]; // to check the graph connectivity.
```

```
int par[N],color[N];
int weight[N],touse[N];
bool used[N+100];
vector<int>choosed;
int directed_mst(int root)
{
 int i,j,t,u,v;
 e[root].clear();
 for(i=0; i<n; i++)
 {
 par[i] = i;
 sort(e[i].begin(),e[i].end());
 }
 bool cycle_found = true;
 while(cycle_found)
 {
 cycle_found = false;
 mem(color,0);
 color[root] = -1;
 for(j=0,t=1; j<n; j++,t++)
 {
 u = par[j];
 if(color[u]) continue;
 for(v=u; !color[v]; v=par[e[v][0].u])
 {
 color[v] = t;
 choosed.push_back(e[v][0].idx);
 }
 if(color[v] != t) continue;
 }
 }
}
```

```

cycle_found = true;
int sum = 0, super = v;
for(; color[v]==t; v=par[e[v][0].u])
{
 color[v]++;
 sum+= e[v][0].w;
}
for(j=0; j<n; j++) weight[j] = INF;
for(; color[v]==t+1; v=par[e[v][0].u])
{
 color[v]--;
 for(j = 1; j<e[v].size(); j++)
 {
 int w = e[v][j].w+sum-e[v][0].w;
 if(w<weight[e[v][j].u])
 {
 weight[e[v][j].u] = w;
 touse[e[v][j].u]=e[v][j].idx;
 }
 }
 par[v] = super;
}
e[super].clear();
for(j=0; j<n; j++) if(par[j] != par[par[j]]) par[j] = par[par[j]];
for(j=0; j<n; j++){
 if(weight[j]<INF && par[j]!= super)
 {
 edge ed = edge(j,super,weight[j]);
 ed.idx = touse[j];
 e[super].push_back(ed);
 }
}
sort(e[super].begin(),e[super].end());
for(j=0; j<e[super].size(); j++)
{
 edge ed=e[super][j];
}
}
int sum = 0;
for(i=0; i<n; i++){
 if(i!=root && par[i]==i)
 {
 sum += e[i][0].w; // i'th node's zero'th edge contains the
minimum cost after directed_mst algo.
 }
}
return sum;
}
int ispossible(int root)
{
 int i,j,u,v;
 for(i=0; i<n; i++)
 {
 for(j=0; j<e[i].size(); j++)
 {
 g[e[i][j].u].push_back(e[i][j].v);
 }
 }
}

```

```

}

queue<int>q;
q.push(root);
mem(color,0);
color[root] = 1;
while(!q.empty()) //BFS to check graph connectivity.
{
 u = q.front();
 q.pop();
 for(i=0; i<g[u].size(); i++)
 {
 v = g[u][i];
 if(color[v]) continue;
 color[v] = 1;
 q.push(v);
 }
 for(i=0; i<n; i++) if(!color[i]) return -1;
 return directed_mst(root);
}
//Beware!!!0-indexed
int main()
{
 int i,j,k;
 edge ed;
 int root;
 cin>>n>>m;
 for(i=0; i<m; i++)
 {

```

```

 cin>>ed.u>>ed.v>>ed.w;
 ed.u--;
 ed.v--;
 ed.idx = i;
 e[ed.v].push_back(ed);
 edges.push_back(ed);
}
root=0;//select the root of the rooted minimum spanning tree
int res = ispossible(root);
if(res == -1) cout<<"impossible\n";
else
{
 mem(used,0);
 mem(color,0);
 for(i=(int)choosed.size()-1; i>=0; i--)
 {
 edge ed = edges[choosed[i]];
 if(color[ed.v]) continue;
 color[ed.v] = 1;
 used[choosed[i]] = true;
 }
 cout<<res<<nl;
 for(i=0; i<m; i++) if(used[i]) cout<<i+1<<' ';
 cout<<nl;
}
return 0;
}
```

## Steiner tree

```

//Given a graph and a subset of vertices in the graph,
///a steiner tree is a tree that contains at least all the nodes from the
given subset.
///The Steiner Tree may contain some vertices which are not
///in given subset but are used to connect the vertices of subset.
///The given set of vertices is called Terminal Vertices and
/// other vertices that are used to construct Steiner tree are called
Steiner vertices.
///The Steiner Tree Problem is to find the minimum cost Steiner Tree.

///Problem: Given a 2d matrix and k nodes ,find the steiner tree
///where cost is the sum of a[i][j]s where (i,j) is the nodes of the
steiner tree

///Complexity; min(3^k*n*m,2^k*(n*m)^2)
///must be in 0-indexed
const int M=8;//max number of terminal nodes
int ter[M][2];//terminal nodes
int a[N][N],dp[N][1<<M],n,m;
pii par[N][1<<M];
char s[N][N];

///tracking the steiner tree
void backtrack(int i,int mask)
{
 if(i<0 || mask<0) return;
 s[i/m][i%m]='X';
}

```

```

backtrack(par[i][mask].F,par[i][mask].S);
if(i==par[i][mask].F) backtrack(i,mask^par[i][mask].S);
}
int main()
{
 BeatMeScanf;
 int i,j,k,t;
 cin>>n>>m>>k;
 for (i=0; i<n; i++) for (int j=0; j<m; j++) cin>>a[i][j];
 for (i=0; i<k; i++)
 {
 int x,y;
 cin>>x>>y;
 ter[i][0]=x-1;
 ter[i][1]=y-1;
 }
 for (int i=0; i<n*m; i++) for (int j=0; j<(1<<k); j++)
 dp[i][j]=1e9,par[i][j]=MP(-1,-1);
 /// base case
 for (int i=0; i<k; i++)
 {
 int x=ter[i][0];
 int y=ter[i][1];
 dp[x*m+y][1<<i]=a[x][y];
 }
 for (int mask=1; mask<(1<<k); mask++)
 {
 for (int i=0; i<n*m; i++)
 {

```

```

for (int sub=((mask-1)&mask); sub>0 ; sub=((sub-1)&mask))
{
 int num1=dp[i][sub];//cost of having nodes of sub
 int num2=dp[i][sub^mask];//cost for tree without sub
nodes
 int num3=a[i/m][i%m];//remove cost for common node i
 if (num1+num2-num3<dp[i][mask]){
 dp[i][mask]=num1+num2-num3;
 par[i][mask]=MP(i,sub);
 }
}
//run spfa
queue<int> q;
vector<bool> inque(N*N,0);
for (int i=0; i<n*m; i++)
{
 inque[i]=true;
 q.push(i);
}
while (!q.empty())
{
 int node=q.front();
 q.pop();
 inque[node]=false;
 for (int i=0; i<4; i++)
 {
 int x=node/m;
 int y=node%m;

```

```

 int nx=x+dx[i];
 int ny=y+dy[i];
 if (nx>=0 && nx<n && ny>=0 && ny<m)
 {
 int num=dp[x*m+y][mask]+a[nx][ny];//add cost for
adjacent transitions
 if (num<dp[nx*m+ny][mask])
 {
 dp[nx*m+ny][mask]=num;
 par[nx*m+ny][mask]=MP(node,mask);
 if (inque[nx*m+ny]==false)
 {
 inque[nx*m+ny]=true;
 q.push(nx*m+ny);
 }
 }
 }
 }
 int ans=1e9,last=0;
 for(int i=0;i<n*m;i++){
 if(dp[i][(1<<k)-1]<ans){
 ans=dp[i][(1<<k)-1];
 last=i;
 }
 }
 cout<<ans<<nl;
 backtrack(last,(1<<k)-1);
}

```

```

for(i=0;i<n;i++){
 for(j=0;j<m;j++){
 if(s[i][j]!='X') s[i][j]='.';
 cout<<s[i][j];
 }
 cout<<nl;
}
return 0;
}

```

### Kirchhoff's Theorem

```

///number of spanning trees in an undirected graph
int g[N][N];
double determinant(vector<vector<double>>a)
{
 int n=a.size();
 double det = 1;
 for (int i=0; i<n; ++i)
 {
 int k = i;
 for (int j=i+1; j<n; ++j)
 if (abs (a[j][i]) > abs (a[k][i]))
 k = j;
 if (abs (a[k][i]) < eps)
 {
 det = 0;
 break;
 }
 swap (a[i], a[k]);
 }
}

```

```

if (i != k)
 det = -det;
det *= a[i][i];
for (int j=i+1; j<n; ++j)
 a[i][j] /= a[i][i];
for (int j=0; j<n; ++j)
 if (j != i && abs (a[j][i]) > eps)
 for (int k=i+1; k<n; ++k)
 a[j][k] -= a[i][k] * a[j][i];
}

return det;
}
int deg[N];
int main()
{
 fast;
 int i,j,k,n,m,u,v;
 cin>>n>>m;
 for(i=1;i<=m;i++){
 cin>>u>>v;
 g[u][v]++;
 g[v][u]++;
 deg[u]++;
 deg[v]++;
 }
 for(i=1;i<=n;i++){
 for(j=1;j<=n;j++){
 if(i==j) g[i][j]=deg[i];
 }
 }
}

```

```

 else g[i][j]*=-1;
 }
}
vector<vector<double>>a(n-1,vector<double>(n-1));
for(i=1;i<n;i++) for(j=1;j<n;j++) a[i-1][j-1]=1.0*g[i][j];
cout<<round(determinant(a))<<nl;
return 0;
}

```

### Manhattan MST

```

int n;
vpii g[N];

struct point {
 int x, y, index;
 bool operator<(const point &p) const { return x == p.x ? y < p.y
: x < p.x; }
} p[N];

struct node {
 int value, p;
} T[N];

struct UnionFind {
 int p[N];
 void init(int n) { for (int i = 1; i <= n; i++) p[i] = i; }
 int find(int u) { return p[u] == u ? u : p[u] = find(p[u]); }
 void Union(int u, int v) { p[find(u)] = find(v); }
} dsu;

```

```

struct edge {
 int u, v, c;
 bool operator < (const edge &p) const {
 return c < p.c;
 }
};
vector<edge> edges;

int query(int x) {
 int r = 2e9, p = -1;
 for (; x <= n; x += (x & -x)) if (T[x].value < r) r = T[x].value, p =
T[x].p;
 return p;
}

void modify(int x, int w, int p) {
 for (; x > 0; x -= (x & -x)) if (T[x].value > w) T[x].value = w, T[x].p
= p;
}

int dist(point &a, point &b) {
 return abs(a.x - b.x) + abs(a.y - b.y);
}

void add(int u, int v, int c) {
 edges.pb({u, v, c});
}

```

```

void kruskal() {
 dsu.init(n);
 srt(edges);
 for (edge e : edges) {
 int u = e.u, v = e.v, c = e.c;
 if (dsu.find(u) != dsu.find(v)) {
 g[u].push_back({v,c});
 g[v].push_back({u,c});
 dsu.Union(u, v);
 }
 }
}

int manhattan() {
 for (int i = 1; i <= n; ++i) p[i].index = i;
 for (int dir = 1; dir <= 4; ++dir) {
 if (dir == 2 || dir == 4) {
 for (int i = 1; i <= n; ++i) swap(p[i].x, p[i].y);
 } else if (dir == 3) {
 for (int i = 1; i <= n; ++i) p[i].x = -p[i].x;
 }
 sort(p + 1, p + 1 + n);
 vector<int> v; static int a[N];
 for (int i = 1; i <= n; ++i) a[i] = p[i].y - p[i].x,
 v.push_back(a[i]);
 sort(v.begin(), v.end());
 v.erase(unique(v.begin(), v.end()), v.end());
 for (int i = 1; i <= n; ++i) a[i] = lower_bound(v.begin(),
 v.end(), a[i]) - v.begin() + 1;
 }
}

```

```

for (int i = 1; i <= n; ++i) T[i].value = 2e9, T[i].p = -1;
for (int i = n; i >= 1; --i) {
 int pos = query(a[i]);
 if (pos != -1) add(p[i].index, p[pos].index,
dist(p[i], p[pos]));
 modify(a[i], p[i].x + p[i].y, i);
}
}

int main()
{
 scanf("%d", &n);
 // points
 for(int i=1;i<=n;i++) scanf("%d%d", &p[i].x, &p[i].y);
 manhattan();
 kruskal();
 // g = manhattan mst adjacency list and corresponding costs
 for(int i=1;i<=n;i++){
 cout<<i<<" ";
 for(auto v:g[i]) cout<<v.F<<' ';
 cout<<nl;
 }
 return 0;
}

```

### Minimum Diameter MST

```

/** Find a minimum diameter spanning tree

```

i.e. farthest distance between two nodes must be minimal  
( $\Leftrightarrow$  absolute center of a graph)

Complexity:

$O(n^2 \log n + n m)$  time,

$O(n^2)$  space.

\*\*/

///0-indexed

typedef int T; // type of weight

const T inf = 1 << 28;

struct graph

{

    struct edge

{

        int s, t;

        T w;

        edge(int s, int t, T w) : s(s), t(t), w(w) {}

};

    int n;

    vector<edge> ed; // edge list

    vector<vector<T>> A; // adj matrix

    vector<vector<int>> adj;

    vector<vector<pair<int,T>>> gr;

graph(int n) : n(n), A(n, vector<int>(n, inf)), adj(n), gr(n)

{

    for(int u=0; u<n; u++)

        A[u][u] = 0;

}

void add\_edge(int s, int t, T w)

{

```
w *= 2; // because of half integrality
if (w >= A[s][t])
 return;
A[s][t] = A[t][s] = w;
ed.push_back(edge(s, t, w));
adj[s].push_back(t);
adj[t].push_back(s);
gr[s].eb(t,w);
gr[t].eb(s,w);
}

vector<pii>mst;//edges of the MST
vector<int> R; // radius from the absolute center
vector<bool> visited;
void traverse(int u, int sh = 0) // explicit construction of sp-tree
{
 visited[u] = true;
 for(int v=0; v<n; v++)
 if (R[v] == R[u] + A[u][v])
 {
 if (!visited[v])
 {
 mst.eb(u,v);
 traverse(v, sh+2);
 }
 }
}
vector<vector<T>> d;
vector<vector<int>> L;
//for weighted edges
```

```

//this part is O(n^2logn)
void farthestOrdering ()
{
 L.assign(n, vector<int>(n));
 d = A;
 for(int i=0; i<n; i++)
 {
 PQ<pair<T,int>,vector<pair<T,int>>,greater<pair<T,int>>>q;
 vector<T> dis(n,inf);
 vi par(n,0);
 vector<bool> vis(n,0);
 q.push({0,i});
 dis[i]=0;
 while(!q.empty())
 {
 int u=q.top().S;
 q.pop();
 if(vis[u])
 continue;
 vis[u]=1;
 for(auto x:gr[u])
 {
 int v=x.F,w=x.S;
 if(dis[u]+w<dis[v])
 {
 par[v]=u;
 dis[v]=dis[u]+w;
 q.push({dis[v],v});
 }
 }
 }
 for(int j=0; j<n; j++)
 d[i][j]=min(d[i][j],dis[j]);
 }
 for(int x=0; x<n; x++)
 {
 vector<pii> aux;
 for(int y=0; y<n; y++)
 aux.push_back(pii(-d[x][y], y));
 sort(all(aux));
 for(int k=0; k<n; k++)
 L[x][k] = aux[k].S; // farthest ordering
 }
}
//for unweighted edges i.e. constant edges
//this part is O(n^2)
// void farthestOrdering () {
// L.assign(n, vector<int>());
// d.assign(n, vector<int>(n, inf));
// for(int s=0;s<n;s++) {
// d[s][s] = 0;
// queue<int> Q; Q.push(s);
// while (!Q.empty()) {
// int u = Q.front(); Q.pop();
// L[s].push_back(u);
// for(int i=0;i<adj[u].size();i++) {
// int v = adj[u][i];
// if(d[s][v]>=d[s][u]+1) {
// d[s][v] = d[s][u]+1;
// L[s].push_back(v);
// }
// }
// }
// }
// }

```

```

// if (d[s][v] > d[s][u] + 2) { // half integrality//2=2*constant
edge,here constant edge weight=1
// d[s][v] = d[s][u] + 2;
// Q.push(v);
// }
// }
// reverse(all(L[s]));
// }
// }

T MDST()
{
 if(n==1) return 0;
 farthestOrdering(); // preprocessing
 Th, D = inf;//D is minimum diameter weight of the spanning tree
 edge &e = ed[0];
 for(auto it=ed.begin(); it!=ed.end(); ++it)
 {
 int s = it->s, t = it->t; // for simplicity
 T w = it->w;
 if (d[s][L[t][0]] + d[t][L[s][0]] + w > 2*D)
 continue; // Halpern bound
 if (L[s][0] == L[t][0])
 continue; // no-coincide condition

 vector<int> &v = L[s];
 int k = 0; // last active constraint
 T x = 0, y = min(d[s][v[0]], d[t][v[0]] + w), xi, yi;
 for (int i = 1; i < n; ++i)

```

```

 {
 if (d[t][v[k]] < d[t][v[i]])
 {
 xi = (d[t][v[k]] - d[s][v[i]] + w) / 2;
 yi = xi + d[s][v[i]];
 if (yi < y)
 {
 y = yi;
 x = xi;
 }
 k = i;
 }
 yi = min(d[s][v[k]]+w, d[t][v[k]]);
 if (yi < y)
 {
 y = yi;
 x = 1;
 }
 if (y < D)
 {
 D = y;
 h = x;
 e = *it;
 }
 }
 R.resize(n); // explicit reconstruction by DFS
 for(int u=0; u<n; u++)
 R[u] = min(d[e.s][u]+h, d[e.t][u]+e.w-h);
}

```

```

visited.assign(n, false);
if (h > 0) traverse(e.t);
if (e.w > h)
{
 traverse(e.s);
 if (h > 0) mst.eb(e.s,e.t);
}
return D;
}

int main () {
 BeatMeScanf;
 cin.tie(0);
 int i,j,k,n,m,t,w;
 cin>>t;
 while(t--){
 cin>>n;
 graph g(n);
 for(i=1;i<=n;i++){
 cin>>k>>k;
 while(k--){
 cin>>j>>w;
 g.add_edge(i-1,j-1,w);
 }
 }
 cout<<g.MDST()<<nl;
 //print mst
 /**

```

```

 for(auto x:g.mst) cout<<x.F+1<<' '<<x.S+1<<nl;
 */
 }
 return 0;
}

```

## 78. Down Trick On Tree

### Notes

When we need to do path queries without any update and if path query has following characteristics,

$\text{path}(u,v)=\text{path}(u,\text{lca})+\text{path}(v,\text{lca})-\text{path}(\text{root},\text{lca})-\text{path}(\text{root},\text{par}[\text{lca}])$   
or similar formula, Then these types of problems can be solved using down trick on tree.

Again all pair path sum or similar problems can be solved with this technique.

### Down Trick Standard

//There's a tree, with each vertex assigned a number. For each query (a, b, c), you are asked whether there is a vertex on the path from a to b, which is assigned number c?

```

vi g[N];
int dep[N],par[N][20],a[N];
void dfs(int u,int pre=0)
{
 dep[u]=dep[pre]+1;
 par[u][0]=pre;
 for(int i=1;i<18;i++) par[u][i]=par[par[u][i-1]][i-1];
 for(auto v:g[u]){
 if(v==pre) continue;
 dfs(v,u);
 }
}

```

```

 }
}

int lca(int u,int v)
{
 if(dep[u]<dep[v]) swap(u,v);
 for(int i=17;i>=0;i--) if(dep[par[u][i]]>=dep[v]) u=par[u][i];
 if(u==v) return u;
 for(int i=17;i>=0;i--) if(par[u][i]!=par[v][i]) u=par[u][i],v=par[v][i];
 return par[u][0];
}
int cnt[N],ans[N];
vpii q[N];
void down(int u,int pre=0)
{
 cnt[a[u]]++;
//now here in cnt array we have all information from the path root to
u
 for(auto x:q[u]){
 int idx=x.F,val=x.S;
 if(idx<0) ans[-idx]-=cnt[val];
 else ans[idx]+=cnt[val];
 }
 for(auto v:g[u]){
 if(v==pre) continue;
 down(v,u);///Don't make any silly mistake by typing here dfs(v,u)
 }
 cnt[a[u]]--;
}
int main()

```

```

{
BeatMeScanf;
int i,j,k,n,m,u,v,c,que;
while(cin>>n>>m){
 for(i=1;i<=n;i++) cin>>a[i];
 for(i=1;i<n;i++) cin>>u>>v,g[u].eb(v),g[v].eb(u);
 dfs(1);
 for(i=1;i<=m;i++){
 cin>>u>>v>>c;
 q[u].eb(i,c);
 q[v].eb(i,c);
 int lc=lca(u,v);
 q[lc].eb(-i,c);
 q[par[lc][0]].eb(-i,c);
 }
 down(1);
 for(i=1;i<=m;i++){
 if(ans[i]) cout<<"Find\n";
 else cout<<"NotFind\n";
 }
 cout<<nl;
 for(i=1;i<=n;i++) g[i].clear();
 for(i=1;i<=m;i++) q[i].clear(),ans[i]=0;
}
return 0;
}

79. Lowest Common Ancestor
vi g[N];

```

```

int par[N][20],dep[N],sz[N];
void dfs(int u,int pre)
{
 par[u][0]=pre;
 dep[u]=dep[pre]+1;
 sz[u]=1;
 for(int i=1;i<=18;i++) par[u][i]=par[par[u][i-1]][i-1];
 for(auto v:g[u]){
 if(v==pre) continue;
 dfs(v,u);
 sz[u]+=sz[v];
 }
}
int lca(int u,int v)
{
 if(dep[u]<dep[v]) swap(u,v);
 for(int k=18;k>=0;k--) if(dep[par[u][k]]>=dep[v]) u=par[u][k];
 if(u==v) return u;
 for(int k=18;k>=0;k--) if(par[u][k]!=par[v][k])
 u=par[u][k],v=par[v][k];
 return par[u][0];
}
int kth(int u,int k)
{
 for(int i=0;i<=18;i++) if(k&(1<<i)) u=par[u][i];
 return u;
}
int dist(int u,int v)
{

```

```

 int lc=lca(u,v);
 return dep[u]+dep[v]-2*dep[lc];
}
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v,q;
 cin>>n;
 for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
 dfs(1,0);
 cin>>q;
 while(q--){
 cin>>u>>v;
 cout<<dist(u,v)<<nl;
 }
 return 0;
}

```

## 80. Shortest Paths

### 0-1 BFS

```

///minimum weight from (0,0) to (n-1,m-1) where weight (x1,y1) to
(x2,y2)=(s[x1][y1]!=s[x2][y2])
int ans[N][N];
string s[N];
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,t,x,y;

```

```

cin>>t;
while(t--){
 cin>>n>>m;
 for(i=0;i<n;i++) cin>>s[i];
 deque<pii>d;
 for(i=0;i<n;i++) for(j=0;j<m;j++) ans[i][j]=1e9;
 ans[0][0]=0;
 d.push_front({0,0});
 while(!d.empty()){
 tie(x,y)=d.front();
 d.pop_front();
 for(i=0;i<4;i++){
 int nx=x+dx[i];
 int ny=y+dy[i];
 if(valid(nx,ny)){
 int w=(s[nx][ny]!=s[x][y]);
 if(ans[x][y]+w<ans[nx][ny]){
 ans[nx][ny]=ans[x][y]+w;
 if(w==0) d.push_front({nx,ny});
 else d.push_back({nx,ny});
 }
 }
 }
 cout<<ans[n-1][m-1]<<nl;
 }
 return 0;
}

```

### Dijkstra's Algorithm

```

vpll g[N];
int main()
{
 BeatMeScanf;
 ll i,j,k,n,m,u,v,w;
 cin>>n>>m;
 for(i=0;i<m;i++) cin>>u>>v>>w,g[u].eb(v,w),g[v].eb(u,w);
 PQ<pll,vpll,greater<pll>>q;
 vll d(n+1,inf);
 vll par(n+1,0);
 q.push({0,1});
 d[1]=0;
 while(!q.empty()){
 tie(w,u)=q.top();
 q.pop();
 for(auto x:g[u]){
 v=x.F,w=x.S;
 if(d[u]+w<d[v]){
 par[v]=u;
 d[v]=d[u]+w;
 q.push({d[v],v});
 }
 }
 }
 if(d[n]==inf) return cout<<-1<<nl,0;
 vll path;
 for(ll nw=n;nw!=0;nw=par[nw]) path.eb(nw);
 rev(path);
}

```

```

for(auto x:path) cout<<x<<' ';
return 0;
}

```

## Bellman-Ford Algorithm

```

#define INF 2e9
struct st
{
 int u,v,w;
}e[N];
///1-based
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,s,t;
 cin>>n>>m;
 for(i=1;i<=m;i++){
 cin>>e[i].u>>e[i].v>>e[i].w;
 }
 cin>>s>>t;
 vector<int> d (n+1, INF);
 d[s] = 0;
 vector<int> p (n+1,-1);
 int cnt=0;
 for (;;) {
 bool any = false;
 for (i = 1; i <=m; ++i){
 if (d[e[i].u] < INF){

```

```

 if (d[e[i].v] > d[e[i].u] + e[i].w)
 {
 d[e[i].v] = d[e[i].u] + e[i].w;
 p[e[i].v] = e[i].u;
 any = true;
 }
 }
 }
 if (!any) break;
 ++cnt;
 if(cnt>=n){
 cout<<"Negative cycle detected\n";
 return 0;
 }
 }
 if (d[t] == INF) cout << "No path from " << s << " to " << t << ".";
 else
 {
 cout<<d[t]<<nl;
 vector<int> path;
 for (int cur = t; cur != -1; cur = p[cur]) path.push_back (cur);
 reverse (path.begin(), path.end());
 cout << "Path from " << s << " to " << t << ":" ;
 for (i=0; i<path.size(); ++i) cout << path[i] << ' ';
 }
 return 0;
}

```

## Johnson's Algorithm

```
/// Johnson's algorithm for all pair shortest paths in sparse graphs
with negative edges
/// Complexity: O(N * M) + O(N * M * log(N))
```

```
const long long INF = (1LL << 60) - 666;
const int MAX=1010;

struct edge{
 int u, v;
 long long w;
 edge(){}
 edge(int u, int v, long long w) : u(u), v(v), w(w){}
}

void print(){
 cout << "edge " << u << " " << v << " " << w << endl;
}
};
```

```
bool bellman_ford(int n, int src, vector <struct edge> ed, vector <long
long>& dis){
 dis[src] = 0;
 for (int i = 0; i <= n; i++){
 int flag = 0;
 for (auto e: ed){
 if ((dis[e.u] + e.w) < dis[e.v]){
 flag = 1;
 dis[e.v] = dis[e.u] + e.w;
 }
 }
 }
}
```

```
}
if (flag == 0) return true;
}
return false;
}

vector <long long> dijkstra(int n, int src, vector <struct edge> ed,
vector <long long> potential){
 set<pair<long long, int> > S;
 vector <long long> dis(n + 1, INF);
 vector <long long> temp(n + 1, INF);
 vector <pair<int, long long> > adj[n + 1];

 dis[src] = temp[src] = 0;
 S.insert(make_pair(temp[src], src));
 for (auto e: ed){
 adj[e.u].push_back(make_pair(e.v, e.w));
 }

 while (!S.empty()){
 pair<long long, int> cur = *(S.begin());
 S.erase(cur);

 int u = cur.second;
 for (int i = 0; i < adj[u].size(); i++){
 int v = adj[u][i].first;
 long long w = adj[u][i].second;

 if ((temp[u] + w) < temp[v]){

```

```

 S.erase(make_pair(temp[v], v));
 temp[v] = temp[u] + w;
 dis[v] = dis[u] + w;
 S.insert(make_pair(temp[v], v));
 }
}
return dis;
}

void johnson(int n, long long ar[MAX][MAX], vector <struct edge> ed){
 vector <long long> potential(n + 1, INF);
 for (int i = 1; i <= n; i++) ed.push_back(edge(0, i, 0));

 assert(bellman_ford(n, 0, ed, potential));
 for (int i = 1; i <= n; i++) ed.pop_back();

 for (int i = 1; i <= n; i++){
 vector <long long> dis = dijkstra(n, i, ed, potential);
 for (int j = 1; j <= n; j++){
 ar[i][j] = dis[j];
 }
 }
}

long long ar[MAX][MAX];

int main(){
 vector <struct edge> ed;

```

```

 ed.push_back(edge(1, 2, 2));
 ed.push_back(edge(2, 3, -15));
 ed.push_back(edge(1, 3, -10));

 int n = 3;
 johnson(n, ar, ed);
 for (int i = 1; i <= n; i++){
 for (int j = 1; j <= n; j++){
 printf("%d %d = %lld\n", i, j, ar[i][j]);
 }
 }
 return 0;
}

```

## Shortest Path Faster Algorithm

**Complexity:**

Average: $O(m)$

Worst: $O(nm)$

//it works for graph with negative edges

#define INF 2e9

vpii g[N];

//0-based

//directed graph

int main()

{

BeatMeScnf;

int i,j,k,n,m,u,v,w;

cin>>n>>m;

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

```

cin>>u>>v>>w;
--u;
--v;
g[u].eb(v,w);
}
vector<int>d;
d.assign(n, INF);
vector<int> cnt(n, 0);
vector<bool> inqueue(n, false);
queue<int> q;
int s=0;
d[s] = 0;
q.push(s);
inqueue[s] = true;
while (!q.empty()) {
 int v = q.front();
 q.pop();
 inqueue[v] = false;

 for (auto edge : g[v]) {
 int to = edge.first;
 int len = edge.second;

 if (d[v] + len < d[to]) {
 d[to] = d[v] + len;
 if (!inqueue[to]) {
 q.push(to);
 inqueue[to] = true;
 cnt[to]++;
 }
 }
 }
}

```

```

if (cnt[to] > n){
 cout<<"Negative cycle detected\n";
 return 0;
}
}
}
}
}
for(i=0;i<n;i++) cout<<d[i]<<' ';
return 0;
}

```

### Floyd-Warshall Algorithm

```

#define INF 2e9
int d[N][N];
///0-based
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v,w;
 cin>>n>>m;
 for(i=0;i<n;i++) for(j=0;j<n;j++) d[i][j]=INF;
 for(i=0;i<n;i++) d[i][i]=0;
 for(i=0;i<m;i++){
 cin>>u>>v>>w;
 --u;
 --v;
 d[u][v]=w;
 d[v][u]=w;
 }
}

```

```

 }
 for (int k = 0; k < n; ++k) {
 for (int i = 0; i < n; ++i) {
 for (int j = 0; j < n; ++j) {
 if (d[i][k] < INF && d[k][j] < INF) d[i][j] = min(d[i][j], d[i][k] +
d[k][j]);
 }
 }
 }
 //For a undirected graph
 //The graph has a negative cycle if at the end of the algorithm,
 //the distance from a vertex v to itself is negative.
 //for directed graph d[u][v]=w hobe only
 //not d[u][v]=w and d[v][u]=w eksathe
 return 0;
}

```

### Shortest Path with Fixed Length

```

///given an undirected graph of n nodes and m edges find the
shortest path
///from node u to node v that has exactly k edges
int sz;
///make sure to erase mod if not needed

void matrix_mul_min(vector<vll> a,vector<vll> b,vector<vll> &sol)
{
 for(int i=0;i<sz;i++){
 for(int j=0;j<sz;j++){
 sol[i][j]=inf;

```

```

 for(int k=0;k<sz;k++){
 sol[i][j]=min(sol[i][j],a[i][k]+b[k][j]);
 }
 }
 }
}

void mat_expo(vector<vll> a,ll n,vector<vll> &res)
{
 res=a;
 while(n){
 if(n&1) matrix_mul_min(res,a,res);
 matrix_mul_min(a,a,a);
 n>>=1;
 }
}

int main()
{
 BeatMeScanf;
 ll i,j,k,n,m,u,v,w;
 cin>>n>>m>>k;
 vector<vll>g(n,vll(n,inf));
 for(i=1;i<=m;i++){
 cin>>u>>v>>w;
 --u,--v;
 g[u][v]=min(w,g[u][v]);
 g[v][u]=min(w,g[v][u]);
 }
}
```

```

vector<vll>res;
sz=n;
mat_expo(g,k-1,res);
for(i=0;i<n;i++){
 for(j=0;j<n;j++){
 cout<<res[i][j]<<' ';
 }
 cout<<nl;
}
return 0;
}

```

### Generalization of the problems for paths with length up to k

The above solution and number of paths with fixed length problem solve the problems for a fixed k. However the solutions can be adapted for solving problems for which the paths are allowed to contain no more than k edges.

This can be done by slightly modifying the input graph.

We duplicate each vertex: for each vertex v we create one more vertex v' and add the edge (v,v') and the loop (v',v'). The number of paths between i and j with at most k edges is the same number as the number of paths between i and j' with exactly k+1 edges.

The same trick can be applied to compute the shortest paths with at most k edges. We again duplicate each vertex and add the two mentioned edges with weight 0.

## 81. Finding Negative Cycle

```

struct st
{
 int a,b,cost;
}e[N];
const int INF=2e9;
//0-based
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,s;
 cin>>n>>m;
 for(i=0;i<m;i++) cin>>e[i].a>>e[i].b>>e[i].cost;
 cin>>s;//is there any negative cycle from s?
 vector<int> d (n, INF);
 d[s] = 0;
 vector<int> p (n, -1);
 int x;
 for (int i=0; i<n; ++i)
 {
 x = -1;
 for (int j=0; j<m; ++j){
 if (d[e[j].a] < INF){
 if (d[e[j].b] > d[e[j].a] + e[j].cost)
 {
 d[e[j].b] = max (-INF, d[e[j].a] + e[j].cost);///for overflow
 p[e[j].b] = e[j].a;
 x = e[j].b;
 }
 }
 }
 }
}

```

```

 }
}
}

if (x == -1) cout << "No negative cycle from" << s;
else
{
 int y = x;
 for (int i=0; i<n; ++i) y = p[y];

 vector<int> path;
 for (int cur=y; ; cur=p[cur])
 {
 path.push_back (cur);
 if (cur == y && path.size() > 1) break;
 }
 reverse (path.begin(), path.end());

 cout << "Negative cycle: ";
 for (int i=0; i<path.size(); ++i)
 cout << path[i] << ' ';
}
return 0;
}

```

## 82. Dominator Tree

```

const int N = int(1e5)+10;
const int M = int(5e5)+10;
vi g[N];

```

```

vi t[N],rg[N],bucket[N];
int sdom[N],par[N],dom[N],dsu[N],label[N];
int arr[N],rev[N],T;
int find_(int u,int x=0)
{
 if(u==dsu[u])return x?-1:u;
 int v = find_(dsu[u],x+1);
 if(v<0)return u;
 if(sdom[label[dsu[u]]] < sdom[label[u]])
 label[u] = label[dsu[u]];
 dsu[u] = v;
 return x?v:label[u];
}
void union_(int u,int v) ///Add an edge u-->v
{
 dsu[v]=u;
}
void dfs(int u)
{
 T++;arr[u]=T;rev[T]=u;
 label[T]=T;sdom[T]=T;dsu[T]=T;
 for(int i=0;i<g[u].size();i++)
 {
 int w = g[u][i];
 if(!arr[w])dfs(w),par[arr[w]]=arr[u];
 rg[arr[w]].eb(arr[u]);
 }
}
int main()

```

```

{
 int i,j,k,n,m,u,v,w;
 cin>>n>>m;
 for(int i=0;i<m;i++)
 {
 cin>>u>>v;
 g[u].eb(v);
 }
 //Build Dominator tree
 dfs(1);
 n=T;
 for(i=n;i>=1;i--)
 {
 for(j=0;j<rg[i].size();j++)
 sdom[i] = min(sdom[i],sdom[find_(rg[i][j])]);
 if(i>1)bucket[sdom[i]].eb(i);
 for(j=0;j<bucket[i].size();j++)
 {
 w = bucket[i][j];
 v = find_(w);
 if(sdom[v]==sdom[w])dom[w]=sdom[w];
 else dom[w] = v;
 }
 if(i>1)union_(par[i],i);
 }
 for(int i=2;i<=n;i++)
 {
 if(dom[i]!=sdom[i])
 dom[i]=dom[dom[i]];
 }
}

```

```

t[rev[i]].eb(rev[dom[i]]);
t[rev[dom[i]]].eb(rev[i]);
}
///make sure to use t[] for the the dominator tree
return 0;
}

83. 2-SAT
///Zero Indexed
///we have vars variables
///F=(x_0 XXX y_0) and (x_1 XXX y_1) and ... (x_{vars-1} XXX y_{vars-1})
///here {x_i,y_i} are variables
///and XXX belongs to {OR,XOR}
///is there any assignment of variables such that F=true

struct twosat {
 int n; // total size combining +, -. must be even.
 vector<vector<int>> g, gt; // graph, transpose graph
 vector<bool> vis, res; // visited and resulting assignment
 vector<int> comp; // component number
 stack<int> ts; // topsort

 twosat(int vars=0) {
 n = (vars << 1);
 g.resize(n);
 gt.resize(n);
 }
}
```

```

//zero indexed, be careful

//if you want to force variable a to be true in OR or XOR
combination
 //add addOR (a,1,a,1);
 //if you want to force variable a to be false int OR or XOR
combination
 //add addOR (a,0,a,0);

///(x_a or (not x_b))-> af=1,bf=0
void addOR(int a, bool af, int b, bool bf) {
 a += a+(af^1);
 b += b+(bf^1);
 g[a^1].push_back(b); /// !a => b
 g[b^1].push_back(a); /// !b => a
 gt[b].push_back(a^1);
 gt[a].push_back(b^1);
}

///(x_a xor x_b)-> af=0,bf=0
void addXOR(int a,bool af,int b,bool bf)
{
 addOR(a,af,b,bf);
 addOR(a,!af,b,!bf);
}

//add this type of condition->
//add(a,af,b,bf) means if a is af then b must need to be bf
void add(int a,bool af,int b,bool bf)
{
 a += a+(af^1);
}

```

```

b += b+(bf^1);
g[a].push_back(b);
gt[b].push_back(a);
}

void dfs1(int u) {
 vis[u] = true;
 for(int v : g[u]) if(!vis[v]) dfs1(v);
 ts.push(u);
}

void dfs2(int u, int c) {
 comp[u] = c;
 for(int v : gt[u]) if(comp[v] == -1) dfs2(v, c);
}
bool ok() {
 vis.resize(n, false);
 for(int i=0; i<n; ++i) if(!vis[i]) dfs1(i);

 int scc = 0;
 comp.resize(n, -1);
 while(!ts.empty()) {
 int u = ts.top();
 ts.pop();
 if(comp[u] == -1) dfs2(u, scc++);
 }

 res.resize(n/2);
 for(int i=0; i<n; i+=2) {
 if(comp[i] == comp[i+1]) return false;
 }
}

```

```

 res[i/2] = (comp[i] > comp[i+1]);
 }
 return true;
}

int main() {
 BeatMeScanf;
 int i,j,k,n,m,a,b,u,v;
 cin>>n>>m;
 twosat ts(n);
 for(i=0;i<m;i++){
 cin>>u>>v>>k;
 --u;
 --v;
 if(k)
 ts.add(u,0,v,0),ts.add(u,1,v,1),ts.add(v,0,u,0),ts.add(v,1,u,1);
 else
 ts.add(u,0,v,1),ts.add(u,1,v,0),ts.add(v,0,u,1),ts.add(v,1,u,0);
 }
 k=ts.ok();
 if(!k) cout<<"Impossible\n";
 else{
 vi v;
 for(i=0;i<n;i++) if(ts.res[i]) v.eb(i);
 cout<<sz(v)<<nl;
 for(auto x:v) cout<<x+1<<' ';
 cout<<nl;
 }
}

```

```

 return 0;
}

```

## 84. Maximum Clique(BronKerbosch)

//cliques are subsets of vertices, all adjacent to each other, also called complete subgraphs  
 //maximum clique is a clique with the largest possible number of vertices

//Complexity ( $3^{(n/3)}$ ) , where n is number of nodes  
 int g[51][51];  
 //for each i every nodes j!=i must need to be set 1 if  
 //there is any edge between i and j  
 //1-indexed  
 int res;  
 long long edges[51];  
 void BronKerbosch(int n, long long R, long long P, long long X) {  
 if (P == OLL && X == OLL) {  
 int t = 0;  
 for (int i = 0; i < n; i++) if ((1ll << i) & R) t++;  
 res = max(res, t);  
 return;  
 }  
 long long u = 0;  
 while (!((1ll << u) & (P | X))) u++;  
 for (int v = 0; v < n; v++) {  
 if (((1ll << v) & P) && !((1ll << v) & edges[u])) {  
 BronKerbosch(n, R | (1ll << v), P & edges[v], X & edges[v]);  
 P -= (1ll << v);
 }
 }
 }
}

```

 X |= (1ll << v);
 }
}

int max_clique (int n) {
 res = 0;
 for (int i = 1; i <= n; i++) {
 edges[i] = 0;
 for (int j = 1; j <= n; j++) if (g[i][j]) edges[i - 1] |= (1ll << (j - 1));
 }
 BronKerbosch(n, 0, (1ll << n) - 1, 0);
 return res;
}

int main()
{
 BeatMeScanf;
 int i,j,k,n;
 n=5;
 for(i=1;i<=n;i++) for(j=1;j<=n;j++) g[i][j]=1;
 for(i=1;i<=n;i++) g[i][i]=0;
 cout<<max_clique(n)<<nl;
 return 0;
}

```

## 85. Number of Different Cliques

//number of cliques in a graph including null clique  
//meet in the middle

```

///Complexity: O((M/2)*2^(M/2))
const int M=42;
int dp[(1<<(M/2))];
ll g[M];
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v;
 cin>>n>>m;
 for(i=0;i<m;i++){
 cin>>u>>v;
 --u;--v;///0-indexed
 g[u]|=(1LL<<v);
 g[v]|=(1LL<<u);
 }
 for(i=0;i<n;i++) g[i]|=(1LL<<i);
 k=n/2;
 dp[0]=1;
 for(i=1;i<(1<<k);i++){
 ll nw=(1LL<<n)-1;
 for(j=0;j<k;j++){
 if((i>>j)&1) nw&=g[j];
 }
 if((nw&i)==i) dp[i]=1;
 }
 for(i=0;i<k;i++){
 for(int mask=0;mask<(1<<k);mask++)
 if((mask>>i)&1) dp[mask]+=dp[mask^(1<<i)];
 }
}

```

```

|| ans=dp[(1<<k)-1];
k=n-k;
for(i=1;i<(1<<k);i++){
 || nw=(1LL<<n)-1;
 for(j=0;j<k;j++){
 if((i>>j)&1) nw&=g[n/2+j];
 }
 || p=(1LL*i)<<(n/2);
 if((nw&p)==p){
 || x=nw&((1LL<<(n/2))-1);
 ans+=dp[x];
 }
}
cout<<ans<<nl;
return 0;
}

```

## 86. Maximum Independent Set(Anticlique)

```

///an independent set or anticlique is a set of
/// vertices in a graph, no two of which are adjacent.
///Maximum Independent Set is an independent set with maximum
number of vertices

///0-indexed
/// O(1.38 ^ n) worst case,n<=60 is appreciable, maybe more
const int MAXN = (52);

int g[MAXN][MAXN], n;

```

```

int mn_deg, comp_size;
bitset<MAXN> st;
vector<int> adj[MAXN];
bool visited[MAXN];

int get_deg(int u)
{
 int res = 0;
 for(int v = 0; v < n; v++)
 if(st[v]) res += g[u][v];

 return res;
}

void dfs(int u)
{
 visited[u] = true;
 mn_deg = min(mn_deg, (int)adj[u].size());
 comp_size++;
 for(int v: adj[u])
 if(!visited[v])
 dfs(v);
}

int brute()
{
 for(int u = 0; u < n; u++) if(st[u]) visited[u] = false,
adj[u].clear();
 for(int u = 0; u < n; u++)

```

```

 if(st[u]) for(int v = 0; v < n; v++)
 if(g[u][v] && st[v]) adj[u].push_back(v);

 int res = 0;
 for(int u = 0; u < n; u++)
 if(st[u] && !visited[u])
 {
 mn_deg = MAXN;
 comp_size = 0;
 dfs(u);

 if(mn_deg <= 1) res += ((comp_size + 1) / 2);
 else res += (comp_size / 2);
 }

 return res;
}

int rec()
{
 if(!st.count()) return 0;

 int d = -1;
 for(int v = 0; v < n; v++)
 if(st[v] && (d == -1 || get_deg(v) > get_deg(d)))
 d = v;

 if(get_deg(d) <= 2) return brute();
}

```

```

 int ret = 0;
 bitset<MAXN> prv = st;

 st[d] = 0;
 ret = max(ret, rec());

 st = prv;
 st[d] = 0;
 for(int u = 0; u < n; u++)
 if(g[u][d]) st[u] = 0;

 ret = max(ret, 1 + rec());

 st = prv;
 return ret;
}

int max_anticlique(int k)
{
 n=k;
 for(int i = 0; i < n; i++) st[i] = 1;
 return rec();
}

map<string,int>mp;
int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 set<int>se;
}

```

```

cin>>n>>m;
for(i=1;i<=n+1;i++){
 int ty;
 if(i<=n) cin>>ty;
 if(ty==1||i>n){
 for(auto x:se) for(auto y:se) g[x-1][y-1]=1,g[y-1][x-1]=1;
 se.clear();
 }
 else{
 string s;
 cin>>s;
 if(mp.find(s)==mp.end()) mp[s]=mp.size();
 int p=mp[s];
 se.insert(p);
 }
}
for(i=0;i<m;i++) g[i][i]=0;
cout<<max_anticlique(m)<<nl;
return 0;
}

```

## 87. Euler Path

```

/*
An Eulerian path is a path that goes through each edge exactly once.
A Hamiltonian path is a path that visits each node exactly once.
#Existence->
#In an undirected graph, euler path:
->the degree of each node is even or

```

->the degree of exactly two nodes is odd, and the degree of all other nodes is even.

#In an undirected graph, euler circuit:

->the degree of each node is even

#In a directed graph, euler path:

- > all the edges belong to the same connected component
- > in each node, the indegree equals the outdegree, or
- > in one node, the indegree is one larger than the outdegree, in another node, the outdegree is one larger than the indegree, and in all other nodes, the indegree equals the outdegree

#In a directed graph, euler circuit:

- > all the edges belong to the same connected component
- > in each node, the indegree equals the outdegree

#Hamiltonian Path or Circuit:

- >NP-Hard

\*/

```

///euler path in a directed graph
///if also checks circuit too
vi ans,g[N];
int done[N],in[N],out[N],n;
void dfs(int u)
{
 while(done[u]<sz(g[u])) dfs(g[u][done[u]++]);
 ans.eb(u);
}
vi path()
{

```

```

int m=0;
for(int u=1;u<=n;u++){
 for(auto v:g[u]){
 in[v]++;
 out[u]++;
 m++;
 }
}
int pos=1,cnt1=0,cnt2=0,root=1;
for(int i=1;i<=n;i++){
 if(in[i]-out[i]==1) cnt1++;
 if(out[i]-in[i]==1) cnt2++,root=i;
 if(abs(in[i]-out[i])>1) pos=0;
}
if(cnt1>1 || cnt2>1) pos=0;
if(!pos) return ans;
dfs(root);
rev(ans);
if(sz(ans)!=m+1){
 ans.clear();
 return ans;
}
return ans;
}
int main()
{
 BeatMeScanf;
 int i,j,k,u,v,m;
 cin>>n>>m;
}

```

```

for(i=1;i<=m;i++) cin>>u>>v,g[u].eb(v);
vi ans=path();
if(ans.empty()) cout<<"NO\n";
else for(auto x:ans) cout<<x<<' ';
return 0;
}

```

## 88. Path Intersection

```

int kth(int u,int k)
{
 for(int i=0;i<=18;i++) if(k&(1<<i)) u=par[u][i];
 return u;
}

int isanc(int u, int g) {
 int k = dep[u] - dep[g];
 return k>=0 && kth(u, k) == g;
}

///merging two paths
pii merge(pii x, pii y) {
 if (x.first == 0) return y;//where {0,0} is a null path
 if (y.first == 0) return x;
 if (x.first===-1 || y.first===-1) return {-1, -1};

 vector<int> can = {x.first, x.second, y.first, y.second};
 int a = can[0];

 for (int u: can)

```

```

if (dep[u] > dep[a])
 a = u;

int b = -1;
for (int u: can)
 if (!isanc(a, u)) {
 if (b == -1) b = u;
 if (dep[b] < dep[u]) b = u;
 }

if (b == -1) {
 b = can[0];
 for (int u: can)
 if (dep[u] < dep[b])
 b = u;
 return {a, b};
}

int g = lca(a, b);
for (int u: can) {
 if (u == a || u == b) continue;
 if (dep[u] < dep[g] || (!isanc(a, u) && !isanc(b, u))) return {-1, -1};
}
return {a, b};
}

```

## 89. Virtual Tree

```
int isanc(int u, int v) {
```

```

 return (st[u] <= st[v]) && (en[v] <= en[u]);
}
vi t[N];
///given specific nodes, construct a compressed directed tree with
these vertices(if needed some other nodes included)
///returns the nodes of the tree
///t[] is the specific tree
vi buildtree(vi v)
{
 ///sort by entry time
 sort(all(v), [](int x, int y){
 return st[x] < st[y];
 });
 ///finding all the ancestors, there are few of them
 int s = sz(v);
 for (int i = 0; i < s - 1; i++) {
 int lc = lca(v[i], v[i + 1]);
 v.eb(lc);
 }
 /// removing duplicated nodes
 sort(all(v));
 v.erase(unique(all(v)), v.end());
 ///again sort by entry time
 sort(all(v), [](int x, int y){
 return st[x] < st[y];
 });
 stack<int> st;
 st.push(v[0]);
 for (int i = 1; i < sz(v); i++) {

```

```

 while(!isanc(st.top(),v[i])) st.pop();
 t[st.top()].pb(v[i]);
 st.push(v[i]);
 }
 return v;
}
int ans;
int imp[N];
int yo(int u)
{
 vi nw;
 for(auto v:t[u]) nw.eb(yo(v));
 if(imp[u]){
 for(auto x:nw) if(x) ans++;
 return 1;
 }
 else{
 int cnt=0;
 for(auto x:nw) cnt+=x>0;
 if(cnt>1){
 ans++;
 return 0;
 }
 return cnt;
 }
}
int32_t main()
{
 BeatMeScnf;
}

```

```

 cin.tie(0);
 int i,j,k,n,m,q,u,v;
 cin>>n;
 for(i=1;i<n;i++) cin>>u>>v,g[u].eb(v),g[v].eb(u);
 dfs(1,0);
 cin>>q;
 while(q--){
 cin>>k;
 vi v;
 for(i=0;i<k;i++) cin>>m,v.eb(m),imp[m]=1;
 int fl=1;
 for(auto x:v) if(imp[par[x][0]]) fl=0;
 ans=0;
 vi nodes;
 if(fl) nodes=buildtree(v);
 if(fl) yo(nodes.front());
 if(!fl) ans=-1;
 cout<<ans<<nl;
 //clear the tree
 for(auto x:nodes) t[x].clear();
 for(auto x:v) imp[x]=0;
 }
 return 0;
}

```

## 90. Max Flow

### Notes

In a normal flow network the flow of an edge is only limited by the capacity  $c(e)$  from above and by 0 from below. In this article we will discuss flow networks, where we additionally require the flow of each edge to have a certain amount, i.e. we bound the flow from below by a **demand** function  $d(e)$ :

$$d(e) \leq f(e) \leq c(e)$$

So next each edge has a minimal flow value, that we have to pass along the edge.

This is a generalization of the normal flow problem, since setting  $d(e)=0$  for all edges  $e$  gives a normal flow network. Notice, that in the normal flow network it is extremely trivial to find a valid flow, just setting  $(e)=0$  is already a valid one. However if the flow of each edge has to satisfy a demand, than suddenly finding a valid flow is already pretty complicated.

We will consider two problems:

1. finding an arbitrary flow that satisfies all constraints
2. finding a minimal flow that satisfies all constraints

#### Finding an arbitrary flow

We make the following changes in the network. We add a new source  $s'$  and a new sink  $t'$ , a new edge from the source  $s'$  to every

other vertex, a new edge for every vertex to the sink  $t'$ , and one edge from  $t$  to  $s$ . Additionally we define the new capacity function  $c'$  as:

- $c'((s', v)) = \sum_{u \in V} d((u, v))$  for each edge  $(s', v)$ .
- $c'((v, t')) = \sum_{w \in V} d((v, w))$  for each edge  $(v, t')$ .
- $c'((u, v)) = c((u, v)) - d((u, v))$  for each edge  $(u, v)$  in the old network.
- $c'((t, s)) = \infty$

If the new network has a saturating flow (a flow where each edge outgoing from  $s'$  is completely filled, which is equivalent to every edge incoming to  $t'$  is completely filled), then the network with demands has a valid flow, and the actual flow can be easily reconstructed from the new network. Otherwise there doesn't exist a flow that satisfies all conditions. Since a saturating flow has to be a maximum flow, it can be found by any maximum flow algorithm, like the Edmonds-Karp algorithm or the Push-relabel algorithm

#### Minimal Flow

Note that along the edge  $(t, s)$  (from the old sink to the old source) with the capacity  $\infty$  flows the entire flow of the corresponding old network. I.e. the capacity of this edge effects the flow value of the old network. By giving this edge a sufficient large capacity (i.e.  $\infty$ ), the flow of the old network is unlimited. By limiting this edge by smaller capacities, the flow value will decrease. However if we limit this edge by a too small value, than the network will not have a saturated solution, e.g. the corresponding solution for the original network will not satisfy the demand of the edges. Obviously here can use a binary search to find the lowest value with which all constraints are still satisfied. This gives the minimal flow of the original network.

#### Hall's Theorem

Hall's theorem can be used to find out whether a bipartite graph has a matching that contains all left or right nodes. If the number of left and right nodes is the same, Hall's theorem tells us if it is possible to construct a perfect matching that contains all nodes of the graph. Assume that we want to find a matching that contains all left nodes. Let  $X$  be any set of left nodes and let  $f(X)$  be the set of their neighbors. According to Hall's theorem, a matching that contains all left nodes exists exactly when for each  $X$ , the condition  $|X| \leq |f(X)|$  holds. where  $|X|$  is the size of the set  $X$ .

### Konig's Theorem

A minimum node cover of a graph is a minimum set of nodes such that each edge of the graph has at least one endpoint in the set. In a general graph, finding a minimum node cover is a NP-hard problem. However, if the graph is bipartite, Konig's theorem tells us that the size of a minimum node cover and the size of a maximum matching are always equal. The nodes that do not belong to a minimum node cover form a maximum independent set. This is the largest possible set of nodes such that no two nodes in the set are connected with an edge. Once again, finding a maximum independent set in a general graph is a NP-hard problem, but in a bipartite graph maximum independent set= node count - maximum matching

### Path cover

A path cover is a set of paths in a graph such that each node of the graph belongs to at least one path. It turns out that in directed, acyclic graphs, we can reduce the problem of finding a minimum path cover to the problem of finding a maximum flow in another graph.

### Node-disjoint path cover

In a node-disjoint path cover, each node belongs to exactly one path. We can find a minimum node-disjoint path cover by constructing a matching graph where each node of the original graph is represented by two nodes: a left node and a right node. There is an edge from a left node to a right node if there is such an edge in the original graph. In addition, the matching graph contains a source and a sink, and there are edges from the source to all left nodes and from all right nodes to the sink. A maximum matching in the resulting graph can be used to find a minimum node-disjoint path cover in the original graph. The size of the minimum node-disjoint path cover is  $n - c$ , where  $n$  is the number of nodes in the original graph and  $c$  is the size of the maximum matching.

### General path cover

A general path cover is a path cover where a node can belong to more than one path. A minimum general path cover may be smaller than a minimum node-disjoint path cover, because a node can be used multiple times in paths.

A minimum general path cover can be found almost like a minimum node-disjoint path cover. It suffices to add some new edges to the matching graph so that there is an edge  $a \rightarrow b$  always when there is a path from  $a$  to  $b$  in the original graph (possibly through several edges).

### Dilworth's theorem

An antichain is a set of nodes of a graph such that there is no path from any node to another node using the edges of the graph. Dilworth's theorem states that in a directed acyclic graph, the size of

a minimum general path cover equals the size of a maximum antichain.

### Dinic's Algorithm

```

struct edge
{
 int to, rev, flow, w;
};

struct dinic
{
 int d[N], done[N], s, t;
 const int INF=2e9;
 vector<edge> g[N];
 /// N equals to node_number

 void addedge(int u, int v, int w)
 {
 edge a={v,(int)g[v].size(),0,w};
 edge b={u,(int)g[u].size(),0,0};

 /// If the graph has bidirectional edges
 /// Capacity for the edge b will be equal to w
 /// For directed, it is 0

 g[u].emplace_back(a);
 g[v].emplace_back(b);
 }
}

```

```

bool bfs()
{
 mem(d,-1);
 d[s]=0;
 queue<int>q;
 q.push(s);
 while(!q.empty()){
 int u=q.front();
 q.pop();
 for(auto &e: g[u])
 {
 int v=e.to;
 if(d[v]==-1 && e.flow<e.w)
 {
 d[v]=d[u]+1;
 q.push(v);
 }
 }
 }
 return d[t]!=-1;
}

int dfs(int u, int flow)
{
 if(u==t) return flow;
 for(int &i=done[u]; i<(int)g[u].size(); i++)
 {
 edge &e=g[u][i];
 if(e.w<=e.flow) continue;

```

```

int v=e.to;
if(d[v]==d[u]+1)
{
 int nw=dfs(v,min(flow,e.w-e.flow));
 if(nw>0)
 {
 e.flow+=nw;
 g[v][e.rev].flow-=nw;
 return nw;
 }
}
return 0;
}

int max_flow(int _s, int _t)
{
 s=_s;
 t=_t;
 int flow=0;
 while(bfs())
 {
 mem(done,0);
 while(int nw=dfs(s,INF)) flow+=nw;
 }
 return flow;
}
};

dinic flow;

```

```

int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v,w;
 cin>>n>>m;
 for(i=1;i<=m;i++) cin>>u>>v>>w,flow.addedge(u,v,w);
 cout<<flow.max_flow(1,n)<<nl;
 //print used edges for the flow
 for(i=1;i<=n;i++) for(auto e:flow.g[i]) if(e.flow>0) cout<<i<<'<<e.to<<'<<e.flow<<nl;
 return 0;
}

Min Cost Max Flow
//works for negative edges too
//if gets TLE for positive edges use dijkstra instead of spfa
//Complexity: O(min(E^2 * V log V, E logV * flow))

struct edge{
 int u, v;
 long long cap, cost;
}

edge(int _u, int _v, long long _cap, long long _cost){
 u = _u; v = _v; cap = _cap; cost = _cost;
}
};

//Works for both directed and undirected and with negative cost too

```

```

///for undirected just make the directed flag false
struct PrimalDual{
 int n, s, t;
 long long flow, cost;
 vector<vector<int>> g;
 vector<edge> e;
 vector<long long> dist, potential;
 vector<int> parent;
 bool negativeCost;
 PrimalDual(int _n){
 /// 0-based indexing
 n = _n;
 g.assign(n, vector<int> ());
 negativeCost = false;
 }

 void addedge(int u, int v, long long cap, long long cost, bool
directed = true){
 if(cost < 0) negativeCost = true;
 g[u].push_back(e.size());
 e.push_back(edge(u, v, cap, cost));

 g[v].push_back(e.size());
 e.push_back(edge(v, u, 0, -cost));

 if(!directed) addedge(v, u, cap, cost, true);
 }

 //returns {maxflow,mincost}
}

```

```

pair<long long, long long> mincost_maxflow(int _s, int _t){
 s = _s; t = _t;
 flow = 0, cost = 0;

 potential.assign(n, 0);
 if(negativeCost){
 /// run Bellman-Ford to find starting potential
 dist.assign(n, 1LL<<62);
 for(int i = 0, relax = false; i < n && relax; i++, relax = false){
 for(int u = 0; u < n; u++){
 for(int k = 0; k < g[u].size(); k++){
 int eldx = g[u][i];
 int v = e[eldx].v, cap = e[eldx].cap, w = e[eldx].cost;

 if(dist[v] > dist[u] + w && cap > 0){
 dist[v] = dist[u] + w;
 relax = true;
 }
 }
 }
 }
 }

 for(int i = 0; i < n; i++){
 if(dist[i] < (1LL<<62)){
 potential[i] = dist[i];
 }
 }
}

```

```

while(dijkstra()){
 flow += sendFlow(t, 1LL<<62);
}

return make_pair(flow, cost);
}

bool dijkstra(){
 parent.assign(n, -1);
 dist.assign(n, 1LL<<62);
 priority_queue<pll, vector<pll>, greater<pll> > pq;

 dist[s] = 0;
 pq.push(pll(0, s));

 while(!pq.empty()){
 int u = pq.top().second;
 long long d = pq.top().first;
 pq.pop();

 if(d != dist[u]) continue;

 for(int i = 0; i < g[u].size(); i++){
 int eldx = g[u][i];
 int v = e[eldx].v, cap = e[eldx].cap;
 int w = e[eldx].cost + potential[u] - potential[v];

 if(dist[u] + w < dist[v] && cap > 0){
 dist[v] = dist[u] + w;
 parent[v] = eldx;

 pq.push(pll(dist[v], v));
 }
 }
 }

 /// update potential
 for(int i = 0; i < n; i++){
 if(dist[i] < (1LL<<62))
 potential[i] += dist[i];
 }

 return dist[t] != (1LL<<62);
}

long long sendFlow(int v, long long curFlow){
 if(parent[v] == -1) return curFlow;
 int eldx = parent[v];
 int u = e[eldx].u, w = e[eldx].cost;

 long long f = sendFlow(u, min(curFlow, e[eldx].cap));

 cost += f*w;
 e[eldx].cap -= f;
 e[eldx^1].cap += f;

 return f;
}

```

```

int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v,w,cost;
 cin>>n>>m;
 PrimalDual flow(2*n+10);
 for(i=1;i<=m;i++) cin>>u>>v>>w>>cost,flow.addedge(u-1,v-
1,w,cost);
 cout<<flow.mincost_maxflow(0,n-1).F<<nl;
 return 0;
}

```

### Bipartite Matching Unweighted

//1 indexed Hopcroft-Karp Matching in O(E sqrtV)

```

struct Hopcroft_Karp
{
 static const int inf = 1e9;

 int n;
 vector<int> matchL, matchR, dist;
 vector<vector<int> > g;

 Hopcroft_Karp(int n) :
 n(n), matchL(n+1), matchR(n+1), dist(n+1), g(n+1) {}

 void addEdge(int u, int v)
 {

```

```

 g[u].push_back(v);///left part(1..n indexed)-right
 part(1..n indexed)
 }

 bool bfs()
 {
 queue<int> q;
 for(int u=1;u<=n;u++)
 {
 if(!matchL[u])
 {
 dist[u]=0;
 q.push(u);
 }
 else
 dist[u]=inf;
 }
 dist[0]=inf;

 while(!q.empty())
 {
 int u=q.front();
 q.pop();
 for(auto v:g[u])
 {
 if(dist[matchR[v]] == inf)
 {
 dist[matchR[v]] = dist[u] + 1;
 q.push(matchR[v]);

```

```

 }
 }

 return (dist[0]!=inf);
}

bool dfs(int u)
{
 if(!u)
 return true;
 for(auto v:g[u])
 {
 if(dist[matchR[v]] == dist[u]+1
&&dfs(matchR[v]))
 {
 matchL[u]=v;
 matchR[v]=u;
 return true;
 }
 }
 dist[u]=inf;
 return false;
}

int max_matching()
{
 int matching=0;
 while(bfs())

```

```

 {
 for(int u=1;u<=n;u++)
 {
 if(!matchL[u])
 if(dfs(u))
 matching++;
 }
 }
 return matching;
};

Bipartite matching Weighted
/// Complexity: O(n^3) but optimized
/// It finds minimum cost maximal matching.
/// For finding maximum cost maximal matching
/// add -cost and return -matching()
/// 1-indexed
struct HungarianMatching{
 long long c[N][N], fx[N], fy[N], d[N];
 int mx[N], my[N], arg[N], trace[N];
 queue<int> q;
 int start, finish, n;
 const long long inf = 1e18;

 HungarianMatching() {}
 HungarianMatching(int n):n(n) {
 for (int i = 1; i <= n; ++i) {
 fy[i] = mx[i] = my[i] = 0;

```

```

 for (int j = 1; j <= n; ++j) c[i][j] = inf;
 }

}

void addedge(int u, int v, long long cost) {
 c[u][v] = min(c[u][v], cost);
}

inline long long getC(int u, int v) {
 return c[u][v] - fx[u] - fy[v];
}

void initBFS() {
 while (!q.empty()) q.pop();
 q.push(start);
 for (int i = 0; i <= n; ++i) trace[i] = 0;
 for (int v = 1; v <= n; ++v) {
 d[v] = getC(start, v);
 arg[v] = start;
 }
 finish = 0;
}

void findAugPath() {
 while (!q.empty()) {
 int u = q.front(); q.pop();
 for (int v = 1; v <= n; ++v) if (!trace[v]) {
 long long w = getC(u, v);

```

```

 if (!w) {
 trace[v] = u;
 if (!my[v]) {
 finish = v;
 return;
 }
 q.push(my[v]);
 }
 if (d[v] > w) {
 d[v] = w;
 arg[v] = u;
 }
 }
 }
 }

void subX_addY() {
 long long delta = inf;
 for (int v = 1; v <= n; ++v) if (trace[v] == 0 && d[v] < delta) {
 delta = d[v];
 }
 // Rotate
 fx[start] += delta;
 for (int v = 1; v <= n; ++v) if(trace[v]) {
 int u = my[v];
 fy[v] -= delta;
 fx[u] += delta;
 } else d[v] -= delta;
 for (int v = 1; v <= n; ++v) if (!trace[v] && !d[v]) {

```

```

trace[v] = arg[v];
if (!my[v]) {
 finish = v; return;
}
q.push(my[v]);
}

void Enlarge() {
 do {
 int u = trace[finish];
 int nxt = mx[u];
 mx[u] = finish;
 my[finish] = u;
 finish = nxt;
 } while (finish);
}

long long matching() {
 for (int u = 1; u <= n; ++u) {
 fx[u] = c[u][1];
 for (int v = 1; v <= n; ++v) {
 fx[u] = min(fx[u], c[u][v]);
 }
 }
 for (int v = 1; v <= n; ++v) {
 fy[v] = c[1][v] - fx[1];
 for (int u = 1; u <= n; ++u) {
 fy[v] = min(fy[v], c[u][v] - fx[u]);
 }
 }
}

 }

}

for (int u = 1; u <= n; ++u) {
 start = u;
 initBFS();
 while (!finish) {
 findAugPath();
 if (!finish) subX_addY();
 }
 Enlarge();
}

long long ans = 0;
for (int i = 1; i <= n; ++i) if(c[i][mx[i]]!=inf) ans += c[i][mx[i]];
return ans;
};

int a[N],b[N],t[N];
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v,w,cost;
 cin>>n;
 for(i=0;i<n;i++) cin>>a[i]>>b[i]>>t[i];
 HungarianMatching M(n);
 for(i=0;i<n;i++){

```

```

 for(j=0;j<n;j++)
 1LL*min(i,t[j])*b[j]));
 }
 cout<<-M.matching()<<nl;
 return 0;
}

```

## Matching Unweighted

/\*

GETS:

V->number of vertices

E->number of edges

pair of vertices as edges (vertices are 1..V)

GIVES:

output of edmonds() is the maximum matching

match[i] is matched pair of i (-1 if there isn't a matched pair)

\*/

```

#include <bits/stdc++.h>
using namespace std;
const int M=500;
struct struct_edge{int v;struct_edge* n;};
typedef struct_edge* edge;
struct_edge pool[M*M*2];
edge top=pool,adj[M];
int V,E,match[M],qh,qt,q[M],father[M],base[M];
bool inq[M],inb[M],ed[M][M];
void add_edge(int u,int v)

```

```

{
 top->v=v,top->n=adj[u],adj[u]=top++;
 top->v=u,top->n=adj[v],adj[v]=top++;
}
int LCA(int root,int u,int v)
{
 static bool inp[M];
 memset(inp,0,sizeof(inp));
 while(1)
 {
 inp[u=base[u]]=true;
 if (u==root) break;
 u=father[match[u]];
 }
 while(1)
 {
 if (inp[v=base[v]]) return v;
 else v=father[match[v]];
 }
}
void mark_blossom(int lca,int u)
{
 while (base[u]!=lca)
 {
 int v=match[u];
 inb[base[u]]=inb[base[v]]=true;
 u=father[v];
 if (base[u]!=lca) father[u]=v;
 }
}
```

```

}

void blossom_contraction(int s,int u,int v)
{
 int lca=LCA(s,u,v);
 memset(inb,0,sizeof(inb));
 mark_blossom(lca,u);
 mark_blossom(lca,v);
 if (base[u]!=lca)
 father[u]=v;
 if (base[v]!=lca)
 father[v]=u;
 for (int u=0;u<V;u++)
 if (inb[base[u]])
 {
 base[u]=lca;
 if (!inq[u])
 inq[q[++qt]=u]=true;
 }
}
int find_augmenting_path(int s)
{
 memset(inq,0,sizeof(inq));
 memset(father,-1,sizeof(father));
 for (int i=0;i<V;i++)
 base[i]=i;
 inq[q[qh=qt=0]=s]=true;
 while (qh<=qt)
 {
 int u=q[qh++];
 for (edge e=adj[u];e;e=e->n)

```

```

{
 int v=e->v;
 if (base[u]!=base[v]&&match[u]!=v)
 if ((v==s) || (match[v]!=-1 && father[match[v]]!=-1))
 blossom_contraction(s,u,v);
 else if (father[v]==-1)
 {
 father[v]=u;
 if (match[v]==-1)
 return v;
 else if (!inq[match[v]])
 inq[q[++qt]=match[v]]=true;
 }
}
return -1;
}
int augment_path(int s,int t)
{
 int u=t,v,w;
 while (u!=-1)
 {
 v=father[u];
 w=match[v];
 match[v]=u;
 match[u]=v;
 u=w;
 }
 return t!=-1;
}

```

```

}

int edmonds()
{
 int matchc=0;
 memset(match,-1,sizeof(match));
 for (int u=0;u<V;u++)
 if (match[u]==-1)
 matchc+=augment_path(u,find_augmenting_path(u));
 return matchc;
}
int main()
{
 int u,v;
 cin>>V>>E;
 while(E--)
 {
 cin>>u>>v;
 if (!ed[u-1][v-1])
 {
 add_edge(u-1,v-1);
 ed[u-1][v-1]=ed[v-1][u-1]=true;
 }
 }
 cout<<edmonds()<<endl;
 for (int i=0;i<V;i++)
 if (i<match[i])
 cout<<i+1<<" "<<match[i]+1<<endl;
}

```

## Matching Weighted

```

const int MAXN = 100 + 10, inf = 0x3f3f3f3f;

int G[MAXN][MAXN], ID[MAXN];
int match[MAXN], stk[MAXN];
int vis[MAXN], dis[MAXN];
int K, top;//K->node number

bool spfa(int u) {
 stk[top ++] = u;
 if (vis[u]) return true;
 vis[u] = true;
 for (int i = 1; i <= K; ++ i) {
 if (i != u && i != match[u] && !vis[i]) {
 int v = match[i];
 if (dis[v] < dis[u] + G[u][i] - G[i][v]) {
 dis[v] = dis[u] + G[u][i] - G[i][v];
 if (spfa(v)) return true;
 }
 }
 }
 top --; vis[u] = false;
 return false;
}

///maximum weight matching in general graph
///O(n^3)
///add negative edge and return -MaxWeightMatch() for minimum
weight matching

```

```

///1-indexed
int MaxWeightMatch() {
 for (int i = 1; i <= K; ++ i) ID[i] = i;
 for (int i = 1; i <= K; i += 2) match[i] = i + 1, match[i + 1] = i;
 for (int times = 0, flag; times < 3;) {
 memset(dis, 0, sizeof(dis));
 memset(vis, 0, sizeof(vis));
 top = 0; flag = 0;
 for (int i = 1; i <= K; ++ i) {
 if (spfa(ID[i])) {
 flag = 1;
 int t = match[stk[top - 1]], j = top - 2;
 while (stk[j] != stk[top - 1]) {
 match[t] = stk[j];
 swap(t, match[stk[j]]);
 -- j;
 }
 match[t] = stk[j]; match[stk[j]] = t;
 break;
 }
 }
 if (!flag) times++;
 if (!flag) random_shuffle(ID + 1, ID + K + 1);
 }
 int ret = 0;
 for (int i = 1; i <= K; ++ i)
 if (i < match[i]) ret += G[i][match[i]];
 return ret;
}

```

```

int w[N][N];
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v,cost,t,cs=0;
 cin>>t;
 while(t--){
 cin>>n>>m>>K;
 for(i=1;i<=n;i++) for(j=1;j<=n;j++) w[i][j]=1e9,w[i][i]=0;
 for(i=1;i<=m;i++){
 cin>>u>>v>>j;
 w[u][v]=min(w[u][v],j);
 w[v][u]=min(w[v][u],j);
 }
 _case;
 if(K&1){
 cout<<"Impossible\n";
 continue;
 }
 for(k=1;k<=n;k++) for(i=1;i<=n;i++) for(j=1;j<=n;j++)
 w[i][j]=min(w[i][j],w[i][k]+w[k][j]);
 for(i=1;i<=K;i++){
 for(j=1;j<=K;j++) G[i][j]=(i==j?-inf:-w[i][j]);
 }
 cout<<-MaxWeightMatch()<<nl;
 }
 return 0;
}

```

## Maximum Closure

```
//a closure of a directed graph is a set of vertices with no outgoing edges.
///That is, the graph should have no edges that start within the closure and
///end outside the closure. The closure problem is the task of finding the maximum-weight
///or minimum-weight closure in a vertex-weighted directed graph
template<typename FlowT>
struct max_flow
{
 const static FlowT finf = 1e18 + 42 + 17;
 const static FlowT feps = 0;

 struct edge
 {
 FlowT flow, cap;
 int idx, rev, to;
 edge() { flow = 0; cap = 0; rev = 0; idx = 0; to = 0; }
 edge(int _to, int _rev, FlowT _flow, FlowT _cap, int _idx)
 {
 to = _to; rev = _rev;
 flow = _flow; cap = _cap;
 idx = _idx;
 }
 };
 vector<edge> G[N];
 int n, dist[N], po[N];
};
```

```
bool bfs(int s, int t)
{
 dist[s] = -1, po[s] = 0;
 dist[t] = -1, po[t] = 0;
 for(int v = 0; v <= n; v++)
 dist[v] = -1, po[v] = 0;

 queue<int> Q;
 Q.push(s);
 dist[s] = 0;

 while(!Q.empty())
 {
 int u = Q.front();
 Q.pop();

 for(edge e: G[u])
 if(dist[e.to] == -1 && e.flow < e.cap)
 {
 dist[e.to] = dist[u] + 1;
 Q.push(e.to);
 }
 }

 return dist[t] != -1;
}

FlowT dfs(int u, int t, FlowT fl = finf)
```

```

{
 if(u == t)
 return fl;

 for(; po[u] < G[u].size(); po[u]++)
 {
 auto &e = G[u][po[u]];
 if(dist[e.to] == dist[u] + 1 && e.flow < e.cap)
 {
 FlowT f = dfs(e.to, t, min(fl, e.cap - e.flow));

 e.flow += f;
 G[e.to][e.rev].flow -= f;

 if(f > 0)
 return f;
 }
 }

 return 0;
}

void init(int _n) { n = _n; for(int i = 0; i <= n; i++) G[i].clear(); }

void add_edge(int u, int v, FlowT w, int idx = -1)
{
 G[u].push_back(edge(v, G[v].size(), 0, w, idx));
 G[v].push_back(edge(u, G[u].size() - 1, 0, 0, -1));
}

```

```

FlowT flow(int s, int t)
{
 if(s == t) return finf;

 FlowT ret = 0, to_add;
 while(bfs(s, t))
 while((to_add = dfs(s, t)))
 ret += to_add;

 return ret;
};

//for minimum closure make weights negative
template<typename T>
struct maximum_closure
{
 int n;
 T w[N];
 max_flow<T> mf;
 vector<int> adj[N];

 void init(int _n)
 {
 n = _n;
 for(int i = 1; i <= n; i++)
 w[i] = 0, adj[i].clear();
 }
}
```

```

void add_clause(int i, int j) { adj[i].push_back(j); }

int dfs_time, cnt_comp, comp[N], disc[N], low[N];
bool in_stack[N];
stack<int> st;

void dfs_tarjan(int u)
{
 disc[u] = low[u] = ++dfs_time;
 in_stack[u] = 1;
 st.push(u);

 for(int v: adj[u])
 if(disc[v] == -1)
 {
 dfs_tarjan(v);
 low[u]=min(low[u], low[v]);
 }
 else if(in_stack[v])
 low[u]=min(low[u], disc[v]);

 if(low[u] == disc[u])
 {
 cnt_comp++;
 while(st.top() != u)
 {
 in_stack[st.top()] = 0;
 comp[st.top()] = cnt_comp;
 }
}

```

```

 st.pop();
 }

 comp[u] = cnt_comp;
 in_stack[u] = 0;
 st.pop();
}

T solve()
{
 for(int i = 1; i <= n; i++)
 disc[i] = -1;

 dfs_time = 0, cnt_comp = 0;
 for(int i = 1; i <= n; i++)
 if(disc[i] == -1)
 dfs_tarjan(i);

 int s = cnt_comp + 1, t = cnt_comp + 2;
 mf.init(cnt_comp + 3);

 vector<T> new_w;
 new_w.assign(cnt_comp + 1, 0);
 for(int i = 1; i <= n; i++)
 new_w[comp[i]] += w[i];

 for(int i = 1; i <= n; i++)
 for(int j: adj[i])

```

```

 mf.add_edge(comp[i], comp[j],
max_flow<T>::finf);

 T sum = 0;
 for(int i = 1; i <= cnt_comp; i++)
 {
 if(new_w[i] < 0) mf.add_edge(i, t, -new_w[i]);
 else mf.add_edge(s, i, new_w[i]), sum += new_w[i];
 }

 return sum - mf.flow(s, t);
 }

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v;
 cin>>n>>m;
 maximum_closure<ll>M;
 M.init(n);
 for(i=1;i<=n;i++) cin>>M.w[i];//weight of the vertices
 for(i=1;i<=m;i++) cin>>u>>v,M.add_clause(u,v);//u nile v nite
 hobei
 cout<<M.solve()<<nl;
 return 0;
}

```

## Gomory-Hu Tree

```

using F = ll; using W = ll; /// types for flow and weight/cost
struct S{
 const int v; // neighbour
 const int r; // index of the reverse edge
 F f; // current flow
 const F cap; // capacity
 const W cost; // unit cost
 S(int v, int ri, F c, W cost = 0) :
 v(v), r(ri), f(0), cap(c), cost(cost) {}
 inline F res() const { return cap - f; }
};

struct FlowGraph : vector<vector<S>> {
 FlowGraph(size_t n) : vector<vector<S>>(n) {}
 void add_arc(int u, int v, F c, W cost = 0){ auto &t = *this;
 t[u].emplace_back(v, t[v].size(), c, cost);
 t[v].emplace_back(u, t[u].size()-1, 0, -cost);
 }
 void add_edge(int u, int v, F c, W cost = 0) { auto &t = *this;
 t[u].emplace_back(v, t[v].size(), c, cost);
 t[v].emplace_back(u, t[u].size()-1, c, -cost);
 }
 void clear() { for (auto &ed : *this) for (auto &e : ed) e.f = 0LL;
 }
};

///0-indexed
struct Dinic{
 FlowGraph &edges; int V,s,t;

```

```

vi l; vector<vector<S>>::iterator> its; // levels and iterators
Dinic(FlowGraph &edges, int s, int t) :
 edges(edges), V(edges.size()), s(s), t(t), l(V,-1), its(V) {}
|| augment(int u, F c) { // we reuse the same iterators
 if (u == t) return c;
 for(auto &i = its[u]; i != edges[u].end(); i++){
 auto &e = *i;
 if (e.cap > e.f && l[u] < l[e.v]) {
 auto d = augment(e.v, min(c, e.cap - e.f));
 if (d > 0) { e.f += d; edges[e.v][e.r].f -= d;
return d; }
 }
 }
 return 0;
}
|| run() {
 || flow = 0, f;
 while(true) {
 fill(l.begin(), l.end(), -1); l[s]=0; // recalculate
the layers
 queue<int> q; q.push(s);
 while(!q.empty()){
 auto u = q.front(); q.pop();
 for(auto &&e : edges[u]) if(e.cap > e.f
&& l[e.v]<0)
 l[e.v] = l[u]+1, q.push(e.v);
 }
 if (l[t] < 0) return flow;
 }
}

```

```

for (int u = 0; u < V; ++u) its[u] =
edges[u].begin();
while ((f = augment(s, 2e9)) > 0) flow +=
f;///take care of inf
}
};

//For a given weighted graph the Gomory-Hu tree has the following
properties:
//The vertex set of the tree and the graph is the same.
//The maximum flow between vertices u and v in the tree(i.e.
minimum edge from u to v)
//is equal to the maximum flow in the graph.
//O-Indexed
//O(n*maxflow+n^2)
struct edge { int u, v; || w; };
struct GomoryHuTree {
 int V;
 vi p, w, c;
 vector<edge> tree;
 void dfs(int u, const FlowGraph &fg) {
 c[u] = 1;
 for (const auto &e : fg[u])
 if (!c[e.v] && e.res())
 dfs(e.v, fg);
 }
 GomoryHuTree(int n, const vector<edge> &ed) : V(n), p(V),
w(V), c(V) {

```

```

FlowGraph fg(V);
for (const edge &e : ed) fg.add_edge(e.u, e.v, e.w);
p[0] = -1, std::fill(p.begin() + 1, p.end(), 0);
for (int i = 1; i < V; ++i) {
 w[i] = Dinic(fg, i, p[i]).run();
 std::fill(c.begin(), c.end(), 0);
 dfs(i, fg);
 for (int j = i+1; j < V; ++j)
 if (c[j] && p[j] == p[i]) p[j] = i;
 if (p[p[i]] >= 0 && c[p[p[i]]]) { // Is this needed?
 :-))))))
 int pi = p[i];
 swap(w[i], w[pi]);
 p[i] = p[pi];
 p[pi] = i;
 }
 fg.clear();
}
}

const vector<edge> &get_tree() {
 if (tree.empty())
 for (int i = 0; i < V; ++i) {
 if (p[i] >= 0)
 tree.push_back(edge{i,
(int)p[i], w[i]}));
 }
 return tree;
}
};

```

```

struct dsu {
 vi par, rank, size; int c;
 dsu(int n) : par(n), rank(n,0), size(n,1), c(n) {
 for (int i = 0; i < n; ++i) par[i] = i;
 }
 int find(int i) { return (par[i] == i) ? i : (par[i] = find(par[i])); }
 bool same(int i, int j) { return find(i) == find(j); }
 int get_size(int i) { return size[find(i)]; }
 int count() { return c; }
 int merge(int i, int j) {
 if ((i = find(i)) == (j = find(j))) return -1; else --c;
 if (rank[i] > rank[j]) swap(i, j);
 par[i] = j; size[j] += size[i];
 if (rank[i] == rank[j]) rank[j]++;
 return j;
 }
};

///find a permutation such that sum of max flow(p[i],p[i+1]) is
maximum
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin >> n >> m;

 vector<edge> ed(m);
 for (edge &e : ed) cin >> e.u >> e.v >> e.w, --e.u, --e.v;
}

```

```

GomoryHuTree gt(n, ed);
vector<edge> t = gt.get_tree();

sort(t.begin(), t.end(), [](const edge &l, const edge &r) {
 return l.w > r.w;
});

dsu d(n);
vector<vi> perm(n);
ll ans = OLL;
for (int i = 0; i < n; ++i) perm[i].push_back(i);
for (const auto &e : t) {
 int l = d.find(e.u), r = d.find(e.v);
 if (l != d.merge(l, r)) swap(l, r);
 ans += e.w;
 for (int j : perm[r]) perm[l].push_back(j);
 perm[r].clear();
}

cout << ans << endl;
for (int i = 0; i < n; ++i)
 cout << perm[d.find(0)][i]+1 << " \n"[i+1==n];
return 0;
}

```

## 91. POSET

### Partially Ordered Sets

A set of elements that have relation on an operator, say  $\leq$  for example. So if there are any two elements  $x$  and  $y$ , their relation can be  $x \leq y$ . Let's assume we have such a poset,  $P$ .

It has:

Reflexivity—if  $x$  is an element in the set, then  $x \leq x$ .

Antisymmetry—if  $x \leq y$  and  $y \leq x$ , then  $x = y$ .

Transitivity—if  $x \leq y$ ,  $y \leq z$  then  $x \leq z$ . The term itself talks about partiality, so not

all elements are related on the particular operator.

The term itself talks about partiality, so not all elements are related on the particular operator.

#### Chain and Anti-chain

Chain—so we have elements in our poset  $P$ . We take a subset  $C$  from  $P$ . If these elements

can be compared on some particular sequence one after another, then this subset is a chain.

Suppose, we have taken  $a, b, c, d$  from our poset. If we can place the relational operator

in between these elements, like  $a \leq b \leq c \leq d$ , then it is a chain.

Antichain—similarly, if we take a subset  $A$  from  $P$  in such a way that no two of them

can be compared on the relational operator, we call such a subset an antichain.

Posets can be reduced to DAG:

- A chain is a path.
- An antichain is an independent set. (An independent set is a set of nodes such that there is no edge between any two of them.)

### Problem Type 1

Given a poset, which is reduced to DAG, we need to find the minimum number of partitions of the elements into antichains or independent sets. This minimum number is found by Mirsky's Theorem.

According to this theorem, the minimum number of partitions is equal to the length of the maximum sized chain a.k.a length of the longest path in the corresponding DAG.

If we need the actual partition, we define a value  $\text{maxend}[u]$  = the length of the longest path that ends in vertex u.

We have already noted that the minimum number of partition is the length of the longest path, let this length is  $\text{maxLen}$ .

To find the actual partition, we iterate from  $i=1$  to  $\text{maxLen}$ , and find such nodes u that have  $\text{maxend}[u]=i$ . All such nodes fall in one partition.

This works because it can be proved that if there are two nodes u and v for which  $\text{maxend}[u]=\text{maxend}[v]$ , then it is impossible that these two nodes have an edge between them. So they can be in the same independent set.

So we get that minimum partition into antichains = maximum size of a chain.

### Problem Type 2

In this type, we are asked to find the partition into minimum number of chains. This corresponds to taking minimum number of paths from the DAG to cover all the vertices such that no vertex is covered twice (edge disjoint paths).

This problem can be mapped to Dilworth's Theorem. According to Dilworth, minimum number partition into chains is equal to the

maximum size of the antichain. In terms of DAG, we need to find the size of the maximum independent set.

(Maximum Independent Set—it is an independent set such that adding one more vertex to this set will destroy its independent set property).

This can be solved using bipartite matching. The idea is, we take two copies of all of our vertices and put them into two sets. Then we add an edge between u from one set and v from the other set if and only if there is an edge between u to v in the original graph.

We then find the maximum matching in this graph. According to Konig's Theorem,

this maximum matching equals to the minimum vertex cover of the graph, and minimum vertex cover is the dual of the maximum independent set. So, minimum number of partition into independent set equals to the ( $\text{number of nodes} - \text{maximum bipartite matching}$ ) in the constructed graph.

Vertex Cover—a vertex cover (sometimes node cover) of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set

### Problem Type 3

What if it is asked that the edges in the previous problem do not have to be disjoint?

In such case, we find the closure of the original DAG. Then apply the previous solution.

Closure of a DAG is such a graph where for any three vertices u, v and w, if there is an edge from u to v, and v to w in the original graph, then in the closure graph, we will add an edge from u to w

## 92. Betweenness Centrality

```

///Betweenness centrality of undirected unweighted graph (Brandes)
///Compute betweenness centrality, defined by
///f(v) := sum(S(u,w,v)/s(u,w)) for all u,w and u!=v&&w!=v
///Here S(u,w)=number of shortest paths from u to w
///S(u,w,v)= number of shortest paths from u to w that contains node
///v
///Brandes's algorithm, O(nm) time, O(m) space.
vi g[N];
int n;
//0-based
vector<double> betweenness_centrality()
{
 vector<double> centrality(n);

 for (size_t s = 0; s < n; ++s)
 {
 vector<size_t> S;
 vector<double> sigma(n);
 sigma[s] = 1;
 vector<int> dist(n, -1);
 dist[s] = 0;
 queue<size_t> que;
 que.push(s);
 while (!que.empty())
 {
 size_t u = que.front();
 S.push_back(u);
 for (auto v: g[u])
 if (dist[v] < 0)
 dist[v] = dist[u] + 1;
 que.push(v);
 if (dist[v] == dist[u] + 1)
 sigma[v] += sigma[u];
 }
 vector<double> delta(n);
 while (!S.empty())
 {
 size_t u = S.back();
 S.pop_back();
 for (auto v: g[u])
 if (dist[v] == dist[u] + 1)
 delta[u] += sigma[u] / sigma[v] * (1 + delta[v]);
 if (u != s)
 centrality[u] += delta[u];
 }
 }
}

```

```

que.pop();
for (auto v: g[u])
{
 if (dist[v] < 0)
 {
 dist[v] = dist[u] + 1;
 que.push(v);
 }
 if (dist[v] == dist[u] + 1)
 {
 sigma[v] += sigma[u];
 }
}
vector<double> delta(n);
while (!S.empty())
{
 size_t u = S.back();
 S.pop_back();
 for (auto v: g[u])
 {
 if (dist[v] == dist[u] + 1)
 {
 delta[u] += sigma[u] / sigma[v] * (1 + delta[v]);
 }
 }
 if (u != s)
 centrality[u] += delta[u];
}

```

```

 }
 return centrality;
}

int main() {
 int i,j,k,m,u,v;
 cin>>n>>m;
 for(i=0;i<m;i++) cin>>u>>v,>-u,>-v,g[u].eb(v),g[v].eb(u);
 vector<double> ans=betweenness_centrality();
 for(int i=1;i<=n;i++) cout<<ans[i-1]<<' ';
}

```

### 93. ST-numbering

```

```

Let  $G = (V, E)$  be an undirected graph with  $n = |V|$  vertices. An orientation of  $G$  is an assignment of a direction to each edge of  $G$ , making it into a directed graph. It is an acyclic orientation if the resulting directed graph has no directed cycles. Every acyclically oriented graph has at least one source (a vertex with no incoming edges) and at least one sink (a vertex with no outgoing edges); it is a bipolar orientation if it has exactly one source and exactly one sink. In some situations,  $G$  may be given together with two designated vertices  $s$  and  $t$ ; in this case, a bipolar orientation

for  $s$  and  $t$  must have  $s$  as its unique source and  $t$  as its unique sink.

An st-numbering of  $G$  (again, with two designated vertices  $s$  and  $t$ ) is an assignment of

the integers from 1 to  $n$  to the vertices of  $G$ , such that

1. each vertex is assigned a distinct number,
2.  $s$  is assigned the number 1,
3.  $t$  is assigned the number  $n$ , and
4. if a vertex  $v$  is assigned the number  $i$  with  $1 < i < n$ , then at least one neighbor of  $v$  is assigned a smaller number than  $i$  and at least one neighbor of  $v$  is assigned a larger number than  $i$

To form the DAG add edges from lower numbered vertex to higher numbered vertex

```
vector<int> adj[N];
int low[N], disc[N], dfs_time = 0, par[N], sign[N];
vector<int> preorder;
```

```
bool tarjan_check(int u, int pr = -1)
{
 low[u] = disc[u] = ++dfs_time;

 int child_cnt = 0;
 for(int v: adj[u])
 if(v != pr)
 {
 if(disc[v] == -1)
```

```

 {
 child_cnt++;
 if(!tarjan_check(v, u)) return false;
 low[u]=min(low[u],low[v]);
 if(pr != -1 && low[v] >= disc[u]) return
false;
 }
 else low[u]=min(low[u], disc[v]);
}

if(pr == -1 && child_cnt > 1) return false;
return true;
}

void tarjan(int u, int pr = -1)
{
 low[u] = disc[u] = ++dfs_time;
 for(int v: adj[u])
 if(v != pr)
 {
 if(disc[v] == -1)
 {
 preorder.push_back(v);
 tarjan(v, u);
 low[u]=min(low[u], low[v]);
 par[v] = u;
 }
 else low[u]=min(low[u], disc[v]);
 }
}
}

list<int> st_li;
list<int>::iterator it_ver[N];

vector<int> st_numbering(int n,int s,int t)
{
 /// additional edge
 adj[s].push_back(t);
 adj[t].push_back(s);

 dfs_time = 0;
 preorder.clear();
 for(int i = 1; i <= n; i++) disc[i] = low[i] = -1, sign[i] = 0;

 if(!tarjan_check(t))
 return vector<int>(); // no bipolar orientation

 for(int i = 1; i <= n; i++)
 if(disc[i] == -1)
 return vector<int>(); // no bipolar orientation

 for(int i = 1; i <= n; i++) disc[i] = low[i] = -1, sign[i] = 0;

 dfs_time = 0;
 preorder.clear();
 disc[s] = low[s] = ++dfs_time;
 sign[disc[s]] = -1;
 tarjan(t);
}

```

```

st_li.clear();
st_li.push_back(s);
st_li.push_back(t);

it_ver[disc[s]] = st_li.begin();
it_ver[disc[t]] = next(st_li.begin());

for(int v: preorder)
{
 if(sign[low[v]] == -1) it_ver[disc[v]] =
st_li.insert(it_ver[disc[par[v]]], v);
 else it_ver[disc[v]] =
st_li.insert(next(it_ver[disc[par[v]]]), v);
 sign[disc[par[v]]] = -sign[low[v]];
}

vector<int> ret(st_li.begin(), st_li.end());
return ret;
}

int st[N];
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m,s,t;
 cin >> n >> m >> s >> t;
 for(int i = 0; i < m; i++)
 {
 int u, v;

```

```

 cin >> u >> v;
 adj[u].push_back(v);
 adj[v].push_back(u);
 }

 vector<int> li = st_numbering(n,s,t);
 if(sz(li)==0) return cout<<-1<<n<0;
 for(i=0;i<n;i++) st[li[i]]=i+1;
 for(i=1;i<=n;i++) cout<<st[i]<<' ';
 cout<<nl;
 cout << endl;
 return 0;
}

```

## 94. Stable Marriage Problem

/\*\* Given n men and n women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. When there are no such pairs of people, the set of marriages is deemed stable. There is always a solution boy[][][],girl[][] lists partners in order of their preferences sorted according to boy's preference complexity O( $n^2$ ) \*\*/ int n,boy[N][N],girl[N][N],firstof[N],egirl[N],ans[N],wboy[N][N];

```

void stablemarriage()
{
 queue<int>q;
 for(int i=1;i<=n;i++) {q.push(i);firstof[i]=1;}
 mem(egirl,-1);
 for(int i=1;i<=n;i++)
 {
 for(int j=1;j<=n;j++) wboy[i][boy[i][j]]=j;
 }
 while(!q.empty())
 {
 int curgirl=q.front();
 q.pop();
 int curboy=girl[curgirl][firstof[curgirl]];
 if(egirl[curboy]!=-1)
 {
 if(wboy[curboy][egirl[curboy]]>wboy[curboy][curgirl])
 {
 firstof[egirl[curboy]]++;
 q.push(egirl[curboy]);
 egirl[curboy]=curgirl;
 }
 else
 {
 firstof[curgirl]++;
 q.push(curgirl);
 }
 }
 else egirl[curboy]=curgirl;
 }
}

```

```

 }
 for(int i=1;i<=n;i++) ans[i]=egirl[i];
}
int32_t main()
{
 BeatMeScanf;
 cin.tie(0);
 int i,j,k,m,t;
 cin>>t;
 while(t--){
 cin>>n;
 for(i=1;i<=n;i++){
 cin>>k;
 for(j=1;j<=n;j++) cin>>girl[i][j];
 }
 for(i=1;i<=n;i++){
 cin>>k;
 for(j=1;j<=n;j++) cin>>boy[i][j];
 }
 stablemarriage();
 for(i=1;i<=n;i++) cout<<i<<' '<<ans[i]<<nl;
 }
 return 0;
}

```

## 95. Hall Theorem

/\*\*  
Theorem:

Suppose G is a bipartite graph with bipartition (A, B). There is a matching that covers A if and only if for every subset (X belongs to A),  $N(X) \geq |X|$  where  $N(X)$  is the number of neighbors of X  
ans  $|X|$  is the size of subset X

\*\*/

/\*\*

**Problem:**

You are given a string s consisting of characters from a to f, and also some conditions. Each condition is of the form (p, t) where p is an index, t is a string, possibly of length 1. Each condition

means that on position p, you have to place one character from t, which can only consist of letters a to f. All the indices are distinct.

If there is no condition for a particular position, then any letter from a to f can be placed there.

Reorganize string s such that each condition is met and the resulting string is lexicographically smallest.

\*\*/

/\*\*

**Solution:**

The problem can be solved using Hall's Theorem.

We consider a bipartite graph where each partition B consists of the allowed characters and partition

A is the indices (1 to n). An edge from A to B means that we can place this character. Of course if

there is no edge for a particular position of A, we have to add an edge to every character of partition

B.

There are 64 possible subsets for those the 6 allowed characters. For each subset X, we calculate  $N(X)$ .

Here note that a neighbour will be counted in  $N(X)$  if it is connected to any one or more of the element of the subset.

Now, we try to build the reorganized string one character at a time. We try to place the smallest allowed character on our current position. For this we check if the suffix starting from the next position can be constructed. This checking can be done using Hall's Theorem.

How? After placing the current character, we decrement its occurrence count by 1 as we are considering it placed. Now for each of the subsets of the allowed letters, we count total number

of occurrence we are still left with for current subset. If the  $N(X)$  for subset X in the remaining positions is less than the calculated number of occurrence, we are sure that the suffix starting from this position cannot be built.

\*\*/

```
const int K=6;
string s;
int oc[6], p, q, n, cnt[N][64];
bool g[N][K];
void calcSubset()
```

```

{
 for(int i=0;i<n;i++){
 for(int j=0;j<(1<<K);j++){
 bool flag=0;
 for(int k=0;k<K;k++){
 if(j&(1<<k))
 {
 if(g[i][k])
 {
 flag=1;
 break;
 }
 }
 }
 cnt[i][j]+=flag;
 if(i) cnt[i][j]+=cnt[i-1][j];
 }
 }
 bool canBuildSuffix(int pos)
 {
 for(int i=0;i<(1<<K);i++){
 int val=0;
 for(int j=0;j<6;j++){
 if(i&(1<<j)) val+=oc[j];
 }
 if(val>cnt[n-1][i]-cnt[pos-1][i]) return 0;
 }
 return 1;
 }
}

```

```

}
void solve()
{
 for(int i=0;i<n;i++){
 int t=0;
 for(int j=0;j<K;j++) if(g[i][j]) t++;
 if(t==0) for(int j=0;j<K;j++) g[i][j]=1;
 }
 calcSubset();
 string ans="";
 bool canbuild=0;
 for(int i=0;i<n;i++){
 for(int j=0;j<6;j++){
 if(!g[i][j] || oc[j]==0) continue;
 oc[j]--;
 if(canBuildSuffix(i+1))
 {
 canbuild=1;
 ans+=char(j+'a');
 break;
 }
 oc[j]++;
 }
 if(!canbuild) break;
 }
 if(!canbuild) cout<<"Impossible"<<nl;
 else cout<<ans<<nl;
}
int main()

```

```

{
 cin>>s;
 for(int i=0;i<sz(s);i++) oc[s[i]-'a']++;
 n=s.size();
 cin>>q;
 while(q--){
 cin>>p>>s;
 for(int i=0;i<sz(s);i++) g[p-1][s[i]-'a']=1;
 }
 solve();
 return 0;
}

```

## 96. Chromatic Number

/\*\*

Exact Algorithm for Chromatic Number

Description:

A vertex coloring is an assignment of colors to the vertices such that no adjacent vertices have a same color. The smallest number of colors for a vertex coloring is called the chromatic number. Computing the chromatic number is NP-hard.

We can compute the chromatic number by the inclusion-exclusion principle. The complexity is  $O(\text{poly}(n) 2^n)$ . The following implementation runs in  $O(n 2^n)$  but is a Monte-Carlo algorithm since it takes modulus to avoid multiprecision numbers.

Complexity:  $O(n 2^n)$

\*\*/

```

///0-indexed
struct graph {
 int n;
 vector<vector<int>> adj;
 graph(int n) : n(n), adj(n) { }
 void add_edge(int u, int v) {
 adj[u].push_back(v);
 adj[v].push_back(u);
 }
};

int chromatic_number(graph g) {
 const int N = 1 << g.n;
 vector<int> nbh(g.n);
 for (int u = 0; u < g.n; ++u)
 for (int v: g.adj[u])
 nbh[u] |= (1 << v);

 int ans = g.n;
 for (int d: {7}) { /// ,11,21,33,87,93} ///for high probability, return
maximum of each ans
 long long mod = 1e9 + d;
 vector<long long> ind(N), aux(N, 1);
 ind[0] = 1;
 for (int S = 1; S < N; ++S) {
 int u = __builtin_ctz(S);
 ind[S] = ind[S^(1<<u)] + ind[(S^(1<<u))&~nbh[u]];
 }
 for (int k = 1; k < ans; ++k) {

```

```

long long chi = 0;
for (int i = 0; i < N; ++i) {
 int S = i ^ (i >> 1); // gray-code
 aux[S] = (aux[S] * ind[S]) % mod;
 chi += (i & 1) ? aux[S] : -aux[S];
}
if (chi % mod) ans = k;
}
return ans;
}

int main() {
 int n = 6;
 graph g(n);
 g.add_edge(0,1);
 g.add_edge(1,2);
 g.add_edge(2,3);
 g.add_edge(0,2);
 g.add_edge(3,4);
 g.add_edge(4,5);
 g.add_edge(5,0);
 cout << chromatic_number(g) << endl;
}

```

## 97. Transitive Closure

```

///O(n^2)
bool t[N][N];//transitive closure i.e. reachability check for every i to j
vi g[N];

```

```

void dfs(int st,int u)
{
 t[st][u]=1;
 for(auto v:g[u]) if(!t[st][v]) dfs(st,v);
}
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v;
 cin>>n;
 for(i=1;i<=n;i++) cin>>u>>v,g[u].eb(v);
 for(i=1;i<=n;i++) dfs(i,i);
 return 0;
}

```

## 98. Transitive Reduction

/\*\*  
a transitive reduction of a directed graph D is another directed graph  
with the  
same vertices and as few edges as possible, such that if there is a  
(directed) path  
from vertex v to vertex w in D, then there is also such a path in the  
reduction.  
The transitive reduction of a directed acyclic graph is unique and is  
a subgraph of the given graph. However, uniqueness fails for graphs  
with (directed) cycles  
and is NP-Hard to find one such transitive reduction  
a subgraph of a graph is another graph, formed from a subset of the  
vertices of

the graph and all of the edges connecting pairs of vertices in that subset.

```
/**/
///O(nm)
///0-indexed
///original graph changes into the minimal graph
struct graph /// DAG
{
 int n;
 vector<vector<int>> adj;
 graph(int n) : n(n), adj(n) { }
 void add_edge(int i, int j)
 {
 adj[i].push_back(j);
 }

 void transitive_reduction()
 {
 vector<int> ord, d(n, -1);
 function<void(int)> rec = [&](int u)
 {
 d[u] = 0;
 for (int v: adj[u])
 if (d[v] < 0)
 rec(v);
 ord.push_back(u);
 for (int v: ord)
 d[v] = 0;
 for (int i = ord.size()-1; i >= 0; --i)
```

```
for (int w: adj[ord[i]])
 d[w] = max(d[w], d[ord[i]] + 1);

adj[u].erase(
 remove_if(all(adj[u]), [&](int v)
{
 return d[v] > 1;
}),
adj[u].end()
);
for (int u = 0; u < n; ++u)if (d[u] < 0) rec(u);
};

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 n = 10000, m = 100 * n;
 set<pair<int, int>> edges;
 while (edges.size() < m)
 {
 int u = rand() % n, v = rand() % n;
 if (u == v)
 continue;
 if (u > v)
 swap(u, v);
 edges.insert({u, v});
 }
}
```

```

graph g(n);
for (auto p: edges)
 g.add_edge(p.F, p.S);

g.transitive_reduction();
int mm = 0; // new edges
for (int u = 0; u < n; ++u) mm += g.adj[u].size();
cout << m << ' ' << mm << nl;
return 0;
}

```

## 99. DAG Reachability Dynamic

```

/**
It is a data structure that admits the following operations:
add_edge(s, t): insert edge (s,t) to the network if
it does not make a cycle

is_reachable(s, t): return true iff there is a path s --> t
Complexity: amortized O(n) per update
Space: O(n^2)
*/
///0-indexed
struct dag_reachability {
 int n;
 vector<vector<int>> parent;
 vector<vector<vector<int>>> child;
 dag_reachability(int n) : n(n), parent(n, vector<int>(n, -1)), child(n,
 vector<vector<int>>(n)) {}
```

```

bool is_reachable(int src, int dst) {
 return src == dst || parent[src][dst] >= 0;
}
bool add_edge(int src, int dst) {
 if (is_reachable(dst, src)) return false; // break DAG condition
 if (is_reachable(src, dst)) return true; // no-modification
 performed
 for (int p = 0; p < n; ++p)
 if (is_reachable(p, src) && !is_reachable(p, dst))
 meld(p, dst, src, dst);
 return true;
}
void meld(int root, int sub, int u, int v) {
 parent[root][v] = u;
 child[root][u].push_back(v);
 for (int c: child[sub][v])
 if (!is_reachable(root, c))
 meld(root, sub, v, c);
}
// add edges one by one
// if it breaks DAG law then print it
int32_t main()
{
 BeatMeScnf;
 int i,j,k,n,m;
 cin >> n >> m;
 dag_reachability t(n);
 while(m--){

```

```

cin>>i>>j;
--i;
--j;
if(t.is_reachable(j,i)) cout<<i+1<<' '<<j+1<<nl;
else t.add_edge(i,j);
}
cout<<0<<' '<<0<<nl;
return 0;
}

```

## 100. Minimum Mean Weight Cycle

/\*\*

Description:

Given a directed graph  $G = (V, E)$  with edge length  $w$ .

Find a minimum mean cycle  $C$ , i.e.,  $\min w(C)/|C|$ .

Algorithm:

Karp's algorithm. Fix a vertex  $s$ .

By a dynamic programming, we can compute  
the shortest path from  $s$  to  $v$ , with "exactly"  $k$  edges.  
We write  $d(s,u;k)$  for this value.

Then, we can show that

$$\min_u \max_k [d(s,u;n) - d(s,u;k)]/(n-k)$$

is the length of minimum mean cycle.

(prf. Note that  $d(s,u;k)$  consists of cycle + path.

Subtract the path, we obtain a length of cycle.)

Complexity:

$O(nm)$  time,  $O(n^2)$  space.

Remark:

The algorithm can be applied to a "directed" network.

For an undirected network, MMC problem can be solved by  
b-matching/T-join.

```

*/
struct graph
{
 typedef int T;
 const T INF = 99999999;
 struct edge
 {
 int src, dst;
 T weight;
 };
 int n;
 vector<vector<edge>> adj;
 graph(int n) : n(n), adj(n) { }
 void add_edge(int src, int dst, T weight)
 {
 adj[src].push_back({src, dst, weight});
 }
 typedef pair<T, int> fraction;
 fraction min_mean_cycle()
 {
 vector<vector<T>> dist(n+1, vector<T>(n));
 vector<vector<int>> prev(n+1, vector<int>(n, -1));
 fill(all(prev[0]), 0);///starting vertex s=0

```

```

for (int k = 0; k < n; ++k)
{
 for (int u = 0; u < n; ++u)
 {
 if (prev[k][u] < 0)
 continue;
 for (auto e: adj[u])
 {
 if (prev[k+1][e.dst] < 0 ||
 dist[k+1][e.dst] > dist[k][e.src] + e.weight)
 {
 dist[k+1][e.dst] = dist[k][e.src] + e.weight;
 prev[k+1][e.dst] = e.src;
 }
 }
 }
 int v = -1;
 fraction opt = {1, 0}; // +infty
 for (int u = 0; u < n; ++u)
 {
 fraction f = {-1, 0}; // -infty
 for (int k = n-1; k >= 0; --k)
 {
 if (prev[k][u] < 0)
 continue;
 fraction g = {dist[n][u] - dist[k][u], n - k};
 if (f.F * g.S < f.S * g.F)
 f = g;
 }
 }
}

```

```

 }
 if (opt.F * f.S > f.F * opt.S)
 {
 opt = f;
 v = u;
 }
}
if (v >= 0) // found a loop
{
 vector<int> p; // path
 for (int k = n; p.size() < 2 || p[0] != p.back(); v = prev[k--][v])
 p.push_back(v);
 reverse(all(p));
}
return opt;
};

int main()
{
 int n = 4;
 graph g(n);
 for (int i = 0; i < n; ++i)
 {
 for (int j = 0; j < n; ++j)
 {
 if (i == j)

```

```

 continue;
 int c = (rand() % (2*n)) - n;
 g.add_edge(i, j, c);
}
}
auto p = g.min_mean_cycle();
cout << p.F << "/" << p.S << endl;
}

```

## 101. Prufer Code

```

/*
prufer code is a sequence of length n-2 to uniquely determine a
labeled tree with n vertices
Each time take the leaf with the lowest number and add the node
number the leaf is connected to
the sequence and remove the leaf.break the algo after n-2 iterations
*/
//0-indexed
int n;
vector<int> g[N];
int parent[N], degree[N];

void dfs (int v) {
 for (size_t i=0; i<g[v].size(); ++i) {
 int to = g[v][i];
 if (to != parent[v]) {
 parent[to] = v;
 dfs (to);
 }
 }
}
auto p = g.min_mean_cycle();
cout << p.F << "/" << p.S << endl;
}

```

```

 }
 }

///O(n)
vector<int> prufer_code() {
 parent[n-1] = -1;
 dfs (n-1);

 int ptr = -1;
 for (int i=0; i<n; ++i) {
 degree[i] = (int) g[i].size();
 if (degree[i] == 1 && ptr == -1)
 ptr = i;
 }

 vector<int> result;
 int leaf = ptr;
 for (int iter=0; iter<n-2; ++iter) {
 int next = parent[leaf];
 result.push_back (next);
 --degree[next];
 if (degree[next] == 1 && next < ptr)
 leaf = next;
 else {
 ++ptr;
 while (ptr<n && degree[ptr] != 1)
 ++ptr;
 leaf = ptr;
 }
 }
}
```

```

 }
}

return result;
}

/*
prufer code to labeled tree:
Each vertex is found in the Prufer code a certain number of times
equal to its degree minus one
O(n)
*/
vector< pair<int,int>> prufer_to_tree(const vector<int> &
prufer_code) {
 int n = (int) prufer_code.size() + 2;
 vector<int> degree(n, 1);
 for (int i=0; i<n-2; ++i)
 ++degree[prufer_code[i]];

 int ptr = 0;
 while (ptr < n && degree[ptr] != 1)
 ++ptr;
 int leaf = ptr;

 vector< pair<int,int>> result;
 for (int i=0; i<n-2; ++i) {
 int v = prufer_code[i];
 result.push_back (make_pair (leaf, v));

 --degree[leaf];
 if (--degree[v] == 1 && v < ptr)

```

```

 leaf = v;
 } else {
 ++ptr;
 while (ptr < n && degree[ptr] != 1)
 ++ptr;
 leaf = ptr;
 }
 }

 for (int v=0; v<n-1; ++v)
 if (degree[v] == 1)
 result.push_back (make_pair (v, n-1));
 return result;
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k;
 n=4;
 g[0].eb(1);
 g[1].eb(0);
 g[1].eb(2);
 g[2].eb(1);
 g[1].eb(3);
 g[3].eb(1);
 vi ans=prufer_code();
 printv(ans);
 auto p=prufer_to_tree(ans);
 for(auto x:p) cout<<x.F<<' '<<x.S<<nl;
}

```

```
return 0;
}
```

## 102. Tree isomorphism

```
/*
Rooted trees (S,s) and (T,t) are isomorphic
iff there is a bijection between childs of s and childs of t
such that s.child[i] and t.child[pi[i]] is isomorphic.//pi[i] is a
permutation of the child indexes
```

Two trees are isomorphic iff these are isomorphic with some roots.  
Again Two trees T1 and T2 are isomorphic if there is a bijection f between the

vertex sets of T1 and T2 such that any two vertices u and v of T1 are adjacent  
in T1 if and only if f(u) and f(v) are adjacent in T2.

Algorithm:

Aho-Hopcroft-Ullmann's algorithm for rooted isomorphism.  
It simultaneously scans the vertices in the two trees from bottom to root,  
and assigns unique id (among the level) for each subtrees.

For unrooted isomorphism, it first finds centers of tree,  
and try all possibility of rooted tree isomorphism.  
Since the number of centers in a tree is at most two,  
it can be performed as the same complexity as rooted isomorphism.

```
*/
//O(nlogn)
//0-indexed
```

```
struct Tree
{
 int n;
 vector<vector<int>> adj;
 Tree(int n) : n(n), adj(n) { }
 void add_edge(int src, int dst)
 {
 adj[src].push_back(dst);
 adj[dst].push_back(src);
 }
 vector<int> centers()
 {
 vector<int> prev;
 int u = 0;
 for (int k = 0; k < 2; ++k) // double sweep
 {
 queue<int> que;
 prev.assign(n, -1);
 que.push(prev[u] = u);
 while (!que.empty())
 {
 u = que.front();
 que.pop();
 for (auto v: adj[u])
 {
 if (prev[v] >= 0)
 continue;
 que.push(v);
 prev[v] = u;
 }
 }
 }
 }
};
```

```

 }
}
vector<int> path = {u}; // median on a path
while (u != prev[u])
 path.push_back(u = prev[u]);
int m = path.size();
if (m % 2 == 0)
 return {path[m/2-1], path[m/2]};
else
 return {path[m/2]};
}

vector<vector<int>> layer;
vector<int> prev;
int levelize(int r) // split vertices into levels
{
 prev.assign(n,-1);
 prev[r] = n;
 layer = {{r}};
 while (1)
 {
 vector<int> next;
 for (int u: layer.back())
 {
 for (int v: adj[u])
 {
 if (prev[v] >= 0)
 continue;
 prev[v] = u;
 next.push_back(v);
 }
 }
 if (next.empty())
 break;
 layer.push_back(next);
 }
 return layer.size();
};

///rooted isomorphism
bool isomorphic(Tree S, int s, Tree T, int t)
{
 if (S.n != T.n)
 return false;
 if (S.levelize(s) != T.levelize(t))
 return false;

 vector<vector<int>> longcodeS(S.n+1), longcodeT(T.n+1);
 vector<int> codeS(S.n), codeT(T.n);
 for (int h = S.layer.size()-1; h >= 0; --h)
 {
 map<vector<int>, int> bucket;
 for (int u: S.layer[h])
 {
 sort(all(longcodeS[u]));
 bucket[longcodeS[u]] = 0;
 }
 }
}
```

```

 }
 for (int u: T.layer[h])
 {
 sort(all(longcodeT[u]));
 bucket[longcodeT[u]] = 0;
 }
 int id = 0;
 for (auto &p: bucket)
 p.S = id++;
 for (int u: S.layer[h])
 {
 codeS[u] = bucket[longcodeS[u]];
 longcodeS[S.prev[u]].push_back(codeS[u]);
 }
 for (int u: T.layer[h])
 {
 codeT[u] = bucket[longcodeT[u]];
 longcodeT[T.prev[u]].push_back(codeT[u]);
 }
}
return codeS[s] == codeT[t];
}

///unrooted isomorphism
bool isomorphic(Tree S, Tree T)
{
 auto x = S.centers(), y = T.centers();
 if (x.size() != y.size())
 return false;
 if (isomorphic(S, x[0], T, y[0]))
 return true;
 return x.size() > 1 && isomorphic(S, x[1], T, y[0]);
}

void solve()
{
 int n;
 scanf("%d", &n);
 Tree S(n), T(n);
 for (int i = 0; i < n-1; ++i)
 {
 int u, v;
 scanf("%d %d", &u, &v);
 S.add_edge(u-1, v-1);
 }
 for (int i = 0; i < n-1; ++i)
 {
 int u, v;
 scanf("%d %d", &u, &v);
 T.add_edge(u-1, v-1);
 }
 if (isomorphic(S, T))
 printf("YES\n");
 else
 printf("NO\n");
}

int main()
{
 int cs;

```

```

scanf("%d", &cs);
for (int i = 0; i < cs; ++i)
 solve();
}

```

### 103. 3-CYCLE and 4-CYCLE

```

///given a simple undirected graph with n nodes and m edges
///with no self loops or multiple edges
///find the number of 3 and 4 length cycles
///two cycles are different if there edge collections are different
vi g[N],G[N];
int val[N];
int deg[N];//degrees
//m.sqrt(m)
ll cycle3(int n) {
 int i, x;
 ll w = 0;
 for(i = 1; i <= n; i++) G[i].clear();
 for(i = 1; i <= n; i++) val[i] = 0;
 for(i = 1; i <= n; i++) {
 for(auto e : g[i]) {
 if(e < i) continue;
 if(deg[i] <= deg[e]) G[i].push_back(e);
 else G[e].push_back(i);
 }
 }
 for(i = 1; i <= n; i++) {
 for(auto e : g[i]) {
 if(e < i) continue;

```

```

x = deg[i] + deg[e] - 3;
for(auto u : G[i]) val[u] = i;
for(auto v : G[e]) {
 if(val[v] == i) w+=x + deg[v] - 2;
}
}
return w;
}
///m.sqrt(m)
ll cycle4(int n) {
 int i, x;
 ll w = 0;
 for(i = 1; i <= n; i++) G[i].clear();
 for(i = 1; i <= n; i++) val[i] = 0;
 for(i = 1; i <= n; i++) {
 for(auto e : g[i]) {
 if(e < i) continue;
 if(deg[i] <= deg[e]) G[i].push_back(e);
 else G[e].push_back(i);
 }
 }
 for(i = 1; i <= n; i++) {
 for(auto u:g[i]) {
 for(auto v:G[u]) {
 if(deg[v] > deg[i] || (deg[v] == deg[i] && v > i)) w += val[v]++;
 }
 }
 for(auto u:g[i]) {

```

```

 for(auto v:G[u]) val[v] = 0;
 }
}
return w;
}
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m,u,v;
 cin>>n>>m;
 for(i=0;i<m;i++) cin>>u>>v,g[u].eb(v),g[v].eb(u);
 for(i=1;i<=n;i++) deg[i]=sz(g[i]);
 cout<<cycle3(n)<<' '<<cycle4(n)<<nl;
 return 0;
}

```

## POLYNOMIAL

### 104. Polynomial Structure

```

///Beware!!! never forget to call initNTT() initially
///only works for ntt friendly primes
const int MT=18,T=(1<<MT)|1;//maximum size of array,always make
it a power of 2
const int root=3;//primitive root of K
const int K=998244353;//prime modulo
int inv[T], w[T];

```

```

namespace NTT{
 int power(int a,int b)
 {
 int t=1;
 for (; b>0; b>>=1,a=1ll*a*a%K)
 b&1? t=1ll*t*a%K:0;
 return t;
 }

 void initNTT(){
 w[0]=1,w[1]=power(root,K>>MT);
 for(int i=2;i<=T-1;i++) w[i]=1ll*w[i-1]*w[1]%K;
 inv[1]=1;
 for(int i=2;i<=T-1;i++) inv[i]=(K-K/i)*1ll*inv[K%i]%K;
 }

 int getmx(int n)
 {
 int mx=1;
 for (; mx<n; mx<<=1);
 return mx;
 }

 vector<int> ntt(vector<int> &a, int mx)
 {
 vector<int> b=a;
 static int rev[T];
 b.resize(mx);
 for(int i=0;i<=mx-1;i++)

```

```

rev[i]=rev[i>>1]>>1|(i&1)*(mx>>1),rev[i]>i?
swap(b[i],b[rev[i]]):void();
for (int k=1,_=T/2; k<mx; k<=1,_>=1)
 for (int j=0; j<mx; j+=k<<1)
 for (int t,i=j,p=0; i<j+k; ++i,p+=_)
 t=1ll*b[i+k]*w[p]%K,b[i+k]=(b[i]-t+K)%K,b[i]=(b[i]+t)%K;

 return b;
}

vector<int> trans(vector<int> &a, int n){
 const int mx=a.size();
 if(mx<=n) n=mx-1;
 vector<int> b=a;
 vector<int> rev(mx);
 for(int i=0;i<=mx-1;i++)
 rev[i]=rev[i>>1]>>1|(i&1)*(mx>>1),rev[i]>i?
 swap(b[i],b[rev[i]]):void();
 for (int k=1,_=T/2; k<mx; k<=1,_>=1)
 for (int j=0; j<mx; j+=k<<1)
 for (int t,i=j,p=T-1; i<j+k; ++i,p+=_)
 t=1ll*b[i+k]*w[p]%K,b[i+k]=(b[i]-t+K)%K,b[i]=(b[i]+t)%K;
 int t=power(mx,K-2);
 vector<int> c;
 int te=1;
 for(int i=n;i>=0;i--) if (b[i]) {c.pb(b[i]*1ll*t%K);te=i; break;}
 for(int i=te-1;i>=0;i--) c.pb(b[i]*1ll*t%K);
 reverse(all(c));
 return c;
}

```

```

}

vector<int> multiply(vector<int> &a, vector<int> &b, int n){
 int N=a.size(); if (N<b.size()) N=b.size();
 int mx=getmx((N<<1));
 vector<int> c=ntt(a,mx);
 vector<int> d=ntt(b,mx);
 for(int i=0;i<=mx-1;i++) c[i]=c[i]*1ll*d[i]%K;
 int u=a.size()+b.size()-2 ;
 if (n>u) n=u;
 c=trans(c,n);
 return c;
}
///return f(x)*g(x) modulo K
inline vector<int> multiply(vector<int> &a, vector<int> &b){
 return multiply(a,b,T-1);
}
///returns 1/f(x) modulo K
vector<int> inverse(vector<int> &a, int n){
 if (n==0){
 vector<int> b; b.pb(power(a[0],K-2)); return b;
 }
 vector<int> b=inverse(a,n>>1);
 n++;
 vector<int> t;
 if (a.size()>=n) t=vector<int> (a.begin(),a.begin()+n);
 else t=a;
 int mx=getmx(n<<1);
 t=ntt(t,mx);

```

```

b=ntt(b,mx);
for(int i=0;i<=mx-1;i++)
 b[i]=b[i]*(2ll-t[i]*1ll*b[i]%K+K)%K;
n--;
vector<int> C=trans(b,n);
return trans(b,n);
}
/// Polynomial division: divide P (deg n) by MONIC M (deg m)
/// P(x) = Q(x)M(x) + R(x)
/// O(n log n)
vector<int> divide(vector<int> P, vector<int> M, vector<int> &R) {
 if (P.size() < M.size()) {
 R = P;
 return {};
 }
 if (M.size() <= 1) {
 R.clear();
 return P;
 }
 int n = P.size() - 1, m = M.size() - 1;
 vector<int> PP = P, tmp = M, Q;
 reverse(PP.begin(), PP.end());
 reverse(tmp.begin(), tmp.end());
 int nn = 1 << (33 - __builtin_clz(n));
 PP.resize(nn);
 tmp.resize(nn);
 vector<int> invM;
 invM=inverse(tmp, nn / 2);
 invM.resize(nn);
}

```

```

fill(invM.begin() + n - m + 1, invM.end(), 0);
tmp=multiply(invM, PP, nn);
tmp.resize(n - m + 1);
reverse(tmp.begin(), tmp.end());
Q = tmp;
tmp.resize(nn / 2);
M.resize(nn / 2);
tmp=multiply(tmp, M, nn / 2);
R.resize(m);
for (int i = 0; i < m; ++i) {
 R[i] = P[i] - tmp[i];
 if (R[i] < 0) R[i] +=K;
}
while (!R.empty() && !R.back()) R.pop_back();
return Q;
}
///returns d/dx(f(x)) modulo K
inline vector<int> derivative(vector<int> &a, int m){
 vector<int> b;
 for (int i=0; i+1<a.size(); i++){
 b.pb(a[i+1]*1ll*(i+1)%K);
 }
 return b;
}
///returns integration of f(x)dx modulo K
inline vector<int> integrate(vector<int> &a, int m){
 vector<int> b;
 b.pb(0);
 for (int i=1; i-1<a.size() && i<=m; i++){

```

```

 b.pb(a[i-1]*1ll*inv[i]%K);
}
return b;
}
///returns log(e)(f(x)) modulo K
///it uses this formula-> ln(f(x))=integration(d/dx(f(x))/f(x))
vector<int> ln(vector<int> &a, int n){
 int mx=getmx((n+1)<<1);
 vector<int> u=derivative(a,n);
 u=ntt(u,mx);
 vector<int> d=inverse(a,n);
 d=ntt(d,mx);
 vector<int> b(mx);
 for(int i=0;i<=mx-1;i++) b[i]=u[i]*1ll*d[i]%K;
 b=trans(b,n);
 return integrate(b,n);
}
///returns e^f(x) modulo K
vector<int> exp(vector<int> &a, int n){
 if (!n){
 vector<int> b; b.pb(1); return b;
 }
 vector<int> b=exp(a,n>>1);
 n++;
 vector<int> l,t;
 int mx=getmx(n<<1);
 if (a.size()>=n) t=vector<int> (a.begin(),a.begin()+n);
 else t=a;
 t=ntt(t,mx);
}

l=ln(b,n-1); l=ntt(l,mx);
b=ntt(b,mx);
for(int i=0;i<=mx-1;i++)
 b[i]=b[i]*1ll*(1ll-l[i]+t[i]+K)%K;
n--;
return trans(b,n);
}
///returns sqrt(f(x)) modulo K
vector<int> sqrt(vector<int>&a, int n){
 // this only works with a[0]=0,1
 if (n==0){
 vector<int> b; b.pb(a[0]); return b;
 }
 int i2=inv[2];
 vector<int> b=sqrt(a,n>>1);
 n++;
 vector<int> ib,t;
 int mx=getmx(n<<1);
 if (a.size()>=n) t=vector<int> (a.begin(),a.begin()+n);
 else t=a;
 t=ntt(t,mx);
 ib=inverse(b,n-1);
 ib=ntt(ib,mx);
 b=ntt(b,mx);
 for(int i=0;i<=mx-1;i++)
 b[i]=(b[i]*1ll*b[i]+t[i])%K*1ll*i2%K*ib[i]%K;
 n--;
 return trans(b,n);
}

```

```

//returns f(x)^n modulo K
vector<int> pw(vector<int> &a, int n, int m){
 vector<int> res;
 if (n==0){
 res.pb(1);
 return res;
 }
 int A0=power(a[0],n);
 res=ln(a,m);
 for (int i=0; i<res.size(); i++) res[i]=res[i]*1ll*n%K;
 res=exp(res,m);
 for (int i=0; i<res.size(); i++) res[i]=res[i]*1ll*A0%K;
 return res;
}
using namespace NTT;
vector<int> a,b;
int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 initNTT();///always call it initially

 cin>>n>>m;
 a.resize(m+1);
 a[0]=1;
 for(i=1;i<=n;i++) {
 int x;
 cin>>x;
 }
}

```

```

 if (x<=m) a[x]=K-4;
}
b=sqrt(a,m);
if (b.size()==1) {
 for(i=1;i<=m;i++) cout<<0<<endl;
 return 0;
}
b[0]++;
a=inverse(b,m);
a.resize(m+1);
for(i=1;i<=m;i++) cout<<a[i]*2ll%K<<endl;
return 0;
}

```

## 105. Polynomial Root

```

int dblcmp(double d)
{
 if (fabs(d)<eps) return 0;
 return d>eps?1:-1;
}

double cal(const vector<double> &coef, double x) {
 double e = 1, s = 0;
 for (int i = 0; i < coef.size(); ++i) s += coef[i] * e, e *= x;
 return s;
}

double find(const vector<double> &coef, double l, double r, int sl, int
sr) {

```

```

sl = dblcmp(cal(coef, l)), sr = dblcmp(cal(coef, r));
if (sl == 0) return l;
if (sr == 0) return r;
for (int tt = 0; tt < 100 && r - l > eps; ++tt) {
 double mid = (l + r) / 2;
 int smid = dblcmp(cal(coef, mid));
 if (smid == 0) return mid;
 if (sl * smid < 0) r = mid;
 else l = mid;
}
return (l + r) / 2;
}

vector<double> rec(const vector<double> &coef, int n) { // c[n]==1
vector<double> ret;
if (n == 1) {
 ret.push_back(-coef[0]);
 return ret;
}
vector<double> dcoef(n);
for (int i = 0; i < n; ++i) dcoef[i] = coef[i + 1] * (i + 1) / n;
double b = 2;
for (int i = 0; i <= n; ++i) b = max(b, 2 * pow(fabs(coef[i]), 1.0 / (n - i)));
vector<double> droot = rec(dcoef, n - 1);
droot.insert(droot.begin(), -b);
droot.push_back(b);
for (int i = 0; i + 1 < droot.size(); ++i) {
 /*
 */
}
}

```

```

int sl = dblcmp(cal(coef, droot[i])), sr = dblcmp(cal(coef, droot[i + 1]));
if (sl * sr > 0) continue;
ret.push_back(find(coef, droot[i], droot[i + 1], sl, sr));
}
return ret;
}

/// solve c[0]+c[1]*x+c[2]*x^2+...+c[n]*x^n==0
/// returns all possible real roots
/// O(n^2)
vector<double> solve(vector<double> coef) {
int n = coef.size() - 1;
while (coef.back() == 0) coef.pop_back(), --n;
for (int i = 0; i <= n; ++i) coef[i] /= coef[n];
return rec(coef, n);
}
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 vector<double>ans=solve({6,5,1});
 printv(ans);
 return 0;
}

```

## 106. Polynomial Interpolation Standard

/\*
Given n points (x[i], y[i]), computes an n-1-degree polynomial p that

passes through them:  $p(x) = a[0]*x^0 + \dots + a[n-1]*x^{n-1}$ .

Time:  $O(n^2)$

To work modulo a prime, replace the divisions with modinverse  
\*/

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
 vd res(n,0), temp(n,0);
 for(int k=0;k<n;k++) for(int i=k+1;i<n;i++)
 y[i] = (y[i] - y[k]) / (x[i] - x[k]);
 double last = 0; temp[0] = 1;
 for(int k=0;k<n;k++) for(int i=0;i<n;i++) {
 res[i] += y[k] * temp[i];
 swap(last, temp[i]);
 temp[i] -= last * x[k];
 }
 return res;
}
```

## 107. Lagrange Interpolation

“For a given set of points  $(x_i, y_i)$  with no two  $x_i$  values equal, the Lagrange polynomial is the polynomial of lowest degree that assumes at each value  $x_i$  the corresponding value  $y_i$  (i.e. the functions coincide at each point). The interpolating polynomial of the least degree is unique”

The Largange polynomial have the following form:  $P(x) = \sum_{i=1}^n y_i \prod_{j=1, j \neq i}^n \frac{x-x_j}{x_i-x_j}$ .

**Code:**

```
// global, pos, neg;
void shift()
```

```
{
 pos++;
 global = (global * pos) % mod;
 ll d = qpow(neg + mod, mod-2);
 global = (global * d) % mod;
 neg++;

}
struct point
{
 ll x, y;
 ll hor;
 point() {}
 point(ll x, ll y): x(x), y(y) {}
 ll get_val(ll val)
 {
 ll d = (val - x);
 if(d < 0) d += mod;
 ll now = (global * qpow(d,mod-2)) % mod;
 return (now * hor) % mod;
 }
};

//this is useful when the degree n is constant but you
//ought to bruteforce over x to find f(x).
//In that case bruteforce all x's upto n and find f(x) in O(n) for big
values of x
struct lagrange
{
 //polynomial degree n
```

```

//there must be n+1 relation points (xi,yi) for each xi from 0 to n
in the vector v
//careful about the 0-degree polynomials
int n;
vector<point> v;
lagrange() {}
lagrange(int n, vector<point> v): n(n), v(v) {}
///O(nlog(mod))
void init()
{
 assert(v.size() == n + 1);
 ll ret = 1;
 for(int i = 1; i <= n; i++)
 {
 ll now = (-i + mod);
 ret = (ret * now) % mod;
 }
 global = ret;
 pos = 0;
 neg = -n;
 for(int i = 0; i <= n; i++)
 {
 v[i].hor = qpow(global,mod-2);
 shift();
 }
}
///O(nlog(mod))
ll get_val(ll val)
{

```

```

 if(val <= n) return v[val].y;
 global = 1;
 for(int i = 0; i <= n; i++) global = (global * (val - v[i].x)) % mod;
 if(global < 0) global += mod;

 ll ret = 0;
 for(int i = 0; i <= n; i++)
 {
 ll now = v[i].get_val(val);
 ret = (ret + now * v[i].y) % mod;
 }
 return ret;
};

lagrange l;

///given k and x,return f(x) where f(x)=(1^k+2^k+3^k+...x^k)%mod
int32_t main()
{
 BeatMeScanf;
 int i,j,n,k,m;
 vector<point> v;
 cin>>k;
 ll ret = 0;
 for(int i = 0; i <= k + 1; i++)
 {
 ll now = qpow(i, k);
 ret = (ret + now) % mod;
 }
}
```

```

 v.pb(point(i, ret));//relation points
}
l = lagrange(k + 1, v);
l.init();
while(cin>>n) cout<<l.get_val(n)<<nl;
return 0;
}

```

## 108. MultiPoint Evaluation

///You are given a polynomial of degree N with integer coefficients.  
 ///Your task is to find the value of this polynomial at some K different integers, modulo 786433.

```

const int mod=786433;
const int N=1<<18;
const int r=10 ;///primitive root of mod

```

```

int a[N],b[N],c[N],R[N],W[N],res[N],ans[mod] ;
void ntt(int *f)
{
 int n=N,i,j,len,w,wn ;
 for(i=0;i<N;i++)if(res[i]>i)swap(f[i],f[res[i]]);
 for(len=2;len<=N;len<<=1)
 {
 for(i=0,wn=W[n/len];i<n;i+=len)
 {
 for(j=0,w=1;j<len/2;j++,w=w*1ll*wn%mod)
 {
 int u=f[i+j],v=f[i+j+len/2]*1ll*w %mod ;
 f[i+j]=(u+v)%mod , f[i+j+len/2]=(mod+u-v)%mod;
 }
 }
 }
}

```

```

 }
}
}
}

int main()
{
 int i,j,n,tmp,T ;
 W[0]=R[0]=1;
 for(i=1;i<N;i++)
 {
 R[i]=r*1ll*R[i-1] %mod ;
 W[i]=W[i-1]*1000ll%mod ;
 }
 for(int len=N>>1;len;len>>=1)
 {
 int i=N/len >> 1 ;
 for(j=0;j<i;j++)res[j+i]=res[j]+len ;
 }
 scanf("%d",&n);
 for(i=0;i<=n;i++)
 {
 scanf("%d",&a[i]);///the polynomial
 b[i]=1ll*a[i]*R[i]%mod;
 c[i]=1ll*b[i]*R[i]%mod;
 }
 ans[0]=a[0];
 ntt(a),ntt(b),ntt(c);
 for(i=0;i<N;i++)

```

```

{
 tmp=W[i];
 ans[tmp]=a[i];
 tmp=1ll*tmp*r %mod ;
 ans[tmp]=b[i];
 tmp=1ll*tmp*r %mod;
 ans[tmp]=c[i];
}
scanf(" %d",&T);
while(T--)
{
 scanf("%d",&tmp);
 printf("%d\n",ans[tmp]);
}
}

```

## 109. Polynomial with more features

```
/*
Polynomial with integer coefficient (mod M)
```

Implemented routines:

- 1) addition
- 2) subtraction
- 3) multiplication Karatsuba  $O(n^{1.5..})$ , FFT  $O(n \log n)$
- 4) division Newton  $O(M(n))$
- 5) gcd
- 6) multipoint evaluation (divide conquer:  $O(M(n) \log |X|)$ )
- 7) interpolation (divide conquer  $O(M(n) \log n)$ )
- 8) polynomial shift

\*)  $n! \bmod M$  in  $O(n^{1/2} \log n)$  time  
here  $M(n) \sim n \log n$   
Beware!!! heavy constant factors

```
*/
ll add(ll a, ll b, ll M) // a + b (mod M)
{
 return (a += b) >= M ? a - M : a;
}
ll sub(ll a, ll b, ll M) // a - b (mod M)
{
 return (a -= b) < 0 ? a + M : a;
}
ll mul(ll a, ll b, ll M) // a * b (mod M)
{
 ll r = a*b - ((long double)(a)*b/M+.5)*M;
 return r < 0 ? r + M : r;
}
ll div(ll a, ll b, ll M) // solve b x == a (mod M)
{
 ll u = 1, x = 0, s = b, t = M;
 while (s) // extgcd for b x + M s = t
 {
 ll q = t / s;
 swap(x -= u * q, u);
 swap(t -= s * q, s);
 }
 if (a % t)
```

```

return -1; // infeasible
return mul(x < 0 ? x + M : x, a / t, M); // b (xa/t) == a (mod M)
}
// pow(|| a, || b, || M)
{
 || x = 1;
 for (; b > 0; b >>= 1)
 {
 if (b & 1)
 x = (a * x) % M;
 a = (a * a) % M;
 }
 return x;
}

/// p(x) = p[0] + p[1] x + ... + p[n-1] x^n-1
/// assertion: p.back() != 0
typedef vector<ll> poly;
ostream& operator<<(ostream &os, const poly &p)
{
 const double EPS = 1e-4;
 bool head = true;
 for (int i = 0; i < p.size(); ++i)
 {
 if (p[i] == 0)
 continue;
 if (!head)
 os << " + ";
 os << p[i];
 }
}

```

```

head = false;
if (i >= 1)
 os << " x";
if (i >= 2)
 os << " ^ " << i;
}
return os;
}
// adding two polynomials,p+q
poly add(poly p, const poly &q, || M)
{
 if (p.size() < q.size())
 p.resize(q.size());
 for (int i = 0; i < q.size(); ++i)
 p[i] = add(p[i], q[i], M);
 while (!p.empty() && !p.back())
 p.pop_back();
 return p;
}
// p-q
poly sub(poly p, const poly &q, || M)
{
 if (p.size() < q.size())
 p.resize(q.size());
 for (int i = 0; i < q.size(); ++i)
 p[i] = sub(p[i], q[i], M);
 while (!p.empty() && !p.back())
 p.pop_back();
 return p;
}

```

```

}

/// Karatsuba multiplication; this works correctly for M in [long long]
/// n^(1.5)
poly mul_k(poly p, poly q, ll M)
{
 int n = max(p.size(), q.size()), m = p.size() + q.size() - 1;
 for (int k:
 {
 1,2,4,8,16
 })
 n |= (n >> k);
 ++n; // n is power of two
 p.resize(n);
 q.resize(n);
 poly r(6*n);
 function<void(ll*, ll*, int, ll*)> rec = [&](ll *p0, ll *q0, int n, ll *r0)
 {
 if (n <= 4) // 4 is the best threshold
 {
 fill_n(r0, 2*n, 0);
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < n; ++j)
 r0[i+j] = add(r0[i+j], mul(p0[i], q0[j], M), M);
 return;
 }
 }
}

```

```

 ||
*p1=p0+n/2,*q1=q0+n/2,*r1=r0+n/2,*r2=r0+n,*u=r0+5*n,*v=u+n/
2,*w=r0+2*n;
for (int i = 0; i < n/2; ++i)
{
 u[i] = add(p0[i], p1[i], M);
 v[i] = add(q0[i], q1[i], M);
}
rec(p0, q0, n/2, r0);
rec(p1, q1, n/2, r2);
rec(u, v, n/2, w);
for (int i = 0; i < n; ++i)
 w[i] = sub(w[i], add(r0[i], r2[i], M), M);
for (int i = 0; i < n; ++i)
 r1[i] = add(r1[i], w[i], M);
};
rec(&p[0], &q[0], n, &r[0]);
r.resize(m);
return r;
}

/// FFT-based multiplication: this works correctly for M in [int]
/// assume: size of a/b is power of two, mod is predetermined
template <int mod, int sign>
void fmt(vector<ll>& x)
{
 const int n = x.size();
 int h = pow(3, (mod-1)/n, mod);
 if (sign < 0)

```

```

h = div(1, h, mod);
for (int i = 0, j = 1; j < n-1; ++j)
{
 for (int k = n >> 1; k > (i ^= k); k >>= 1);
 if (j < i)
 swap(x[i], x[j]);
}
for (int m = 1; m < n; m *= 2)
{
 ll w = 1, wk = pow(h, n / (2*m), mod);
 for (int i = 0; i < m; ++i)
 {
 for (int s = i; s < n; s += 2*m)
 {
 ll u = x[s], d = x[s + m] * w % mod;
 if ((x[s] = u + d) >= mod)
 x[s] -= mod;
 if ((x[s + m] = u - d) < 0)
 x[s + m] += mod;
 }
 w = w * wk % mod;
 }
}
if (sign < 0)
{
 ll inv = div(1, n, mod);
 for (auto &a: x)
 a = a * inv % mod;
}

}

/// assume: size of a and b is power of two, mod is predetermined
template <int mod>
vector<ll> conv(vector<ll> a, vector<ll> b)
{
 fmt<mod,+1>(a);
 fmt<mod,+1>(b);
 for (int i = 0; i < a.size(); ++i)
 a[i] = a[i] * b[i] % mod;
 fmt<mod,-1>(a);
 return a;
}

/// general convolution where mod < 2^31.
vector<ll> conv(vector<ll> a, vector<ll> b, ll mod)
{
 int n = a.size() + b.size() - 1;
 for (int k:
 {
 1,2,4,8,16
 })
 n |= (n >> k);
 ++n;
 a.resize(n);
 b.resize(n);
 const int A = 167772161, B = 469762049, C = 1224736769, D =
 (ll)(A) * B % mod;
 vector<ll> x = conv<A>(a,b), y = conv(a,b), z = conv<C>(a,b);
 for (int i = 0; i < x.size(); ++i)
 {
}

```

```

|| X = (y[i] - x[i]) * 104391568;
if ((X %= B) < 0)
 X += B;
|| Y = (z[i] - (x[i] + A * X) % C) * 721017874;
if ((Y %= C) < 0)
 Y += C;
x[i] += A * X + D * Y;
if ((x[i] % mod) < 0)
 x[i] += mod;
}
x.resize(n);
return x;
}

poly mul(poly p, poly q, || M)
{
 poly pq = conv(p, q, M);///if M is prime use faster conv
 pq.resize(p.size() + q.size() - 1);
 while (!pq.empty() && !pq.back())
 pq.pop_back();
 return pq;
}

/// Newton division: O(M(n)); M is the complexity of multiplication
/// fast when FFT multiplication is used
/// Note: complexity = M(n) + M(n/2) + M(n/4) + ... <= 2 M(n).
/// returns {result,remainder}
pair<poly,poly> divmod(poly p, poly q, || M)
{
 if (p.size() < q.size())
 return { {}, p };
 reverse(all(p));
 reverse(all(q));
 poly t = {div(1, q[0], M)};
 if (t[0] < 0)
 return { {}, {} }; // infeasible
 for (int k = 1; k <= 2*(p.size()-q.size())+1; k *= 2)
 {
 poly s = mul(mul(t, q, M), t, M);
 t.resize(k);
 for (int i = 0; i < k; ++i)
 t[i] = sub(2*t[i], s[i], M);
 }
 t.resize(p.size() - q.size() + 1);
 t = mul(t, p, M);
 t.resize(p.size() - q.size() + 1);
 reverse(all(t));
 reverse(all(p));
 reverse(all(q));
 while (!t.empty() && !t.back())
 t.pop_back();
 return {t, sub(p, mul(q, t, M), M) };
}

/// polynomial GCD: O(M(n) log n);
poly gcd(poly p, poly q, || M)
{
 for (; !p.empty(); swap(p, q = divmod(q, p, M).S));
 return p;
}

```

```

/// value of p(x)
|| eval(poly p, || x, || M)
{
 || ans = 0;
 for (int i = p.size()-1; i >= 0; --i)
 ans = add(mul(ans, x, M), p[i], M);
 return ans;
};

/// faster multipoint evaluation
/// fast if |x| >= 10000.
/// algo:
/// evaluate(p, {x[0], ..., x[n-1]})
/// = evaluate(p mod (X-x[0])...(X-x[n/2-1]), {x[0], ..., x[n/2-1]}),
/// + evaluate(p mod (X-x[n/2])...(X-x[n-1]), {x[n/2], ..., x[n-1]}),
/// f(n) = 2 f(n/2) + M(n) ==> O(M(n) log n)
/// x->points to evaluate
/// O(M(n)log(|x|))
vector<||> evaluate(poly p, vector<||> x, || M)
{
 vector<poly> prod(8*x.size()); // segment tree
 function<poly(int,int,int)> run = [&](int i, int j, int k)
 {
 if (i == j)
 return prod[k] = (poly)
 {
 1
 };
 if (i+1 == j)

```

```

 return prod[k] = (poly)
 {
 M-x[i], 1
 };
 return prod[k] = mul(run(i,(i+j)/2,2*k+1), run((i+j)/2,j,2*k+2), M);
 };
 run(0, x.size(), 0);
 vector<||> y(x.size());
 function<void(int,int,int,poly)> rec = [&](int i, int j, int k, poly p)
 {
 if (j - i <= 8)
 {
 for (; i < j; ++i)
 y[i] = eval(p, x[i], M);
 }
 else
 {
 rec(i, (i+j)/2, 2*k+1, divmod(p, prod[2*k+1], M).S);
 rec((i+j)/2, j, 2*k+2, divmod(p, prod[2*k+2], M).S);
 }
 };
 rec(0, x.size(), 0, p);
 return y;
}

/// faster algo to find a poly p such that
/// p(x[i]) = y[i] for each i
/// O(M(n)logn)
poly interpolate(vector<||> x, vector<||> y, || M)

```

```

{
vector<poly> prod(8*x.size()); // segment tree
function<poly(int,int,int)> run = [&](int i, int j, int k)
{
 if (i == j)
 return prod[k] = (poly)
 {
 1
 };
 if (i+1 == j)
 return prod[k] = (poly)
 {
 M-x[i], 1
 };
 return prod[k] = mul(run(i,(i+j)/2,2*k+1), run((i+j)/2,j,2*k+2), M);
};
run(0, x.size(), 0); // preprocessing in O(n log n) time

poly H = prod[0]; // newton polynomial
for (int i = 1; i < H.size(); ++i)
 H[i-1] = mul(H[i], i, M);
do
 H.pop_back();
while (!H.empty() && !H.back());

vector<ll> u(x.size());
function<void(int,int,int,poly)> rec = [&](int i, int j, int k, poly p)
{
 if (j - i <= 8)

```

```

{
 for (; i < j; ++i)
 u[i] = eval(p, x[i], M);
 }
 else
 {
 rec(i, (i+j)/2, 2*k+1, divmod(p, prod[2*k+1], M).S);
 rec((i+j)/2, j, 2*k+2, divmod(p, prod[2*k+2], M).S);
 }
 rec(0, x.size(), 0, H); // multipoint evaluation

 for (int i = 0; i < x.size(); ++i)
 u[i] = div(y[i], u[i], M);

function<poly(int,int,int)> f = [&](int i, int j, int k)
{
 if (i >= j)
 return poly();
 if (i+1 == j)
 return (poly)
 {
 u[i]
 };
 return add(mul(f(i,(i+j)/2,2*k+1), prod[2*k+2], M),
 mul(f((i+j)/2,j,2*k+2), prod[2*k+1], M), M);
};
return f(0, x.size(), 0);
}

```

```

/// faster algorithm for computing p(x + a)
/// algo: p(x+a) = p_h(x) (x+a)^m + q_h(x)
/// complexity:
/// preproc: O(M(n))
/// div-con: O(M(n) log n)
poly shift(poly p, ll a, ll M)
{
 vector<poly> pow(p.size());
 pow[0] = {1};
 pow[1] = {a,1};
 int m = 2;
 for (; m < p.size(); m *= 2)
 pow[m] = mul(pow[m/2], pow[m/2], M);
 function<poly(poly,int)> rec = [&](poly p, int m)
 {
 if (p.size() <= 1)
 return p;
 while (m >= p.size())
 m /= 2;
 poly q(p.begin() + m, p.end());
 p.resize(m);
 return add(mul(rec(q, m), pow[m], M), rec(p, m), M);
 };
 return rec(p, m);
}

/// overperform when n >= 134217728 lol

```

```

//sqrt(n)logn
ll factmod(ll n, ll M)
{
 if (n <= 1)
 return 1;
 ll m = sqrt(n);
 function<poly(int,int)> get = [&](int i, int j)
 {
 if (i == j)
 return poly();
 if (i+1 == j)
 return (poly)
 {
 i,1
 };
 return mul(get(i, (i+j)/2), get((i+j)/2, j), M);
 };
 poly p = get(0, m); // = x (x+1) (x+2) ... (x+(m-1))
 vector<ll> x(m);
 for (int i = 0; i < m; ++i)
 x[i] = 1 + i * m;
 vector<ll> y = evaluate(p, x, M);
 ll fac = 1;
 for (int i = 0; i < m; ++i)
 fac = mul(fac, y[i], M);
 for (ll i = m*m+1; i <= n; ++i)
 fac = mul(fac, i, M);
 return fac;
}

```

```

int main()
{
 const ll M = 1000000007;

 poly p = {0, 0, 0, 0, 0, 1};
 poly q = shift(p, 1, M);
 cout << q << endl;
 return 0;
}

```

## GEOMETRY

### 110. Geometry 2D

```

const int mod=1e9+7;
const int mxn=3e5+9;
const double eps=1e-9;
const double PI=acos(-1.0);
const int mxp=2100;
//ll gcd(ll a,ll b){while(b){ll x=a%b;a=b;b=x;}return a;}
//ll lcm(ll a,ll b){return a/gcd(a,b)*b;}
//ll qpow(ll n,ll k) {ll ans=1;assert(k>=0);n%=mod;while(k>0){if(k&1)
ans=(ans*n)%mod;n=(n*n)%mod;k>>=1;}return ans%mod;}
int sign(double d)
{
 if (fabs(d)<eps) return 0;
}

```

```

 return d>eps?1:-1;
 }
 inline double sqr(double x){return x*x;}
 struct PT
 {
 double x,y;
 PT() {}
 PT(double x, double y) : x(x), y(y) {}
 PT(const PT &p) : x(p.x), y(p.y) {}
 void in() {
 sf("%lf %lf",&x,&y);
 }
 void out() {
 pf("%.10f %.10f\n",x,y);
 }
 PT operator + (const PT &a) const{
 return PT(x+a.x,y+a.y);
 }
 PT operator - (const PT &a) const{
 return PT(x-a.x,y-a.y);
 }
 PT operator * (const double a) const{
 return PT(x*a,y*a);
 }
 friend PT operator * (const double &a, const PT &b)
 {
 return PT(a * b.x, a * b.y);
 }
 PT operator / (const double a) const{

```

```

 return PT(x/a,y/a);
}
bool operator==(PT a)const
{
 return sign(a.x-x)==0&&sign(a.y-y)==0;
}
bool operator<(PT a)const
{
 return sign(a.x-x)==0?sign(y-a.y)<0:x<a.x;
}
bool operator>(PT a)const
{
 return sign(a.x-x)==0?sign(y-a.y)>0:x>a.x;
}
double val()
{
 return sqrt(x*x+y*y);
}
double val2()
{
 return (x*x+y*y);
}
double arg()
{
 return atan2(y,x);
}
//return point that is truncated the distance from center to r
PT trunc(double r){
 double l=val();

```

```

 if (!sign(l)) return *this;
 r/=l;
 return PT(x*r,y*r);
 }
 istream& operator >> (istream& is,PT &a)
 {
 return is>>a.x>>a.y;
 }
 ostream& operator << (ostream& os,const PT &a)
 {
 return os<<fixed<<setprecision(10)<<a.x<<' '<<a.y;
 }
 double dist(PT a,PT b)
 {
 return sqrt(sqr(a.x-b.x)+sqr(a.y-b.y));
 }
 double dist2(PT a,PT b)
 {
 return sqr(a.x-b.x)+sqr(a.y-b.y);
 }
 double dot(PT a,PT b)
 {
 return a.x*b.x+a.y*b.y;
 }
 double cross(PT a,PT b)
 {
 return a.x*b.y-a.y*b.x;
 }

```

```

PT rotateccw90(PT a)
{
 return PT(-a.y,a.x);
}
PT rotatecw90(PT a)
{
 return PT(a.y,-a.x);
}
PT rotateccw(PT a,double th)
{
 return PT(a.x*cos(th)-a.y*sin(th),a.x*sin(th)+a.y*cos(th));
}
PT rotatecw(PT a,double th)
{
 return PT(a.x*cos(th)+a.y*sin(th),-a.x*sin(th)+a.y*cos(th));
}
double angle_between_vectors(PT a, PT b)
{
 double costheta=dot(a,b)/a.val()/b.val();
 return acos(fmax(-1.0,fmin(1.0,costheta)));
}
double rad_to_deg(double r) {
 return (r * 180.0 / PI);
}

double deg_to_rad(double d) {
 return (d * PI / 180.0);
}

```

```

// a line is defined by two points
struct line
{
 PT a,b;
 line(){}
 line(PT _a,PT _b)
 {
 a=_a;
 b=_b;
 }
 //ax+by+c=0
 line(double _a,double _b,double _c)
 {
 if (sign(_a)==0)
 {
 a=PT(0,-_c/_b);
 b=PT(1,-_c/_b);
 }
 else if (sign(_b)==0)
 {
 a=PT(-_c/_a,0);
 b=PT(-_c/_a,1);
 }
 else
 {
 a=PT(0,-_c/_b);
 b=PT(1,(-_c-_a)/_b);
 }
 }
}
```

```

 }
}

void in()
{
 a.in();
 b.in();
}
PT vec() const
{
 return (b-a);
}
bool operator==(line v)
{
 return (a==v.a)&&(b==v.b);
}
PT cross_point(line v){
 double a1=cross(v.b-v.a,a-v.a);
 double a2=cross(v.b-v.a,b-v.a);
 return PT((a.x*a2-b.x*a1)/(a2-a1),(a.y*a2-b.y*a1)/(a2-a1));
}
istream &operator>>(istream &is, line &a) {
 return is >> a.a >> a.b;
}
ostream &operator<<(ostream &os, line &p) {
 return os << p.a << " to " << p.b;
}

// find a point from 'a' through 'b' with distance d
PT point_along_line(PT a,PT b,double d) {
 return a + (((b-a) / (b-a).val()) * d);
}

// projection point c onto line through a and b assuming a != b
PT project_from_point_to_line(PT a, PT b, PT c) {
 return a + (b-a)*dot(c-a, b-a)/(b-a).val2();
}

// reflection point c onto line through a and b assuming a != b
PT reflection_from_point_to_line(PT a, PT b, PT c) {
 PT p=project_from_point_to_line(a,b,c);
 return point_along_line(c,p,2*dist(c,p));
}

//minimum distance from point c to line through a and b
double dist_from_point_to_line(PT a,PT b,PT c)
{
 return fabs(cross(b-a,c-a)/(b-a).val());
}

//return 1 if point c is on line segment ab
bool is_point_on_seg(PT a,PT b,PT c)
{
 double d1=dist(a,c)+dist(c,b);
 double d2=dist(a,b);
 if(fabs(d1-d2)<eps) return 1;
 else return 0;
}

//minimum distance point from point c to segment ab that lies on
segment ab

```

```

PT project_from_point_to_seg(PT a, PT b, PT c)
{
 double r = dist2(a,b);
 if (fabs(r) < eps) return a;
 r = dot(c-a, b-a)/r;
 if (r < 0) return a;
 if (r > 1) return b;
 return a + (b-a)*r;
}
//minimum distance from point c to to segment ab
double dist_from_point_to_seg(PT a, PT b, PT c)
{
 return dist(c, project_from_point_to_seg(a, b, c));
}
//returns a parallel line of line ab in counterclockwise direction with
//d distance from ab
line get_parallel_line(PT a,PT b,double d)
{
 return line(point_along_line(a,rotateccw90(b-
a)+a,d),point_along_line(b,rotatecw90(a-b)+b,d));
}
//Return a tangent line of line ab which intersects
//with it at point c in counterclockwise direction
line get_perpendicular_line(PT a,PT b,PT c)
{
 return line(c+rotateccw90(a-c),c+rotateccw90(b-c));
}
//relation of point p with line ab
//return

```

```

//1 if point is ccw with line
//2 if point is cw with line
//3 if point is on the line
int point_line_relation(PT a,PT b,PT p){
 int c=sign(cross(p-a,b-a));
 if (c<0) return 1;
 if (c>0) return 2;
 return 3;
}
//return
//0 if not parallel
//1 if parallel
//2 if collinear
bool is_parallel(PT a,PT b,PT c,PT d)
{
 double k=fabs(cross(b-a,d-c));
 if(k<eps){
 if(fabs(cross(a-b,a-c))<eps&&fabs(cross(c-d,c-a))<eps) return 2;
 else return 1;
 }
 else return 0;
}
double area_of_triangle(PT a,PT b,PT c)
{
 return fabs(cross(b-a,c-a)/2.0);
}
//radian angle of <bac
double angle(PT b,PT a,PT c)
{

```

```

 return angle_between_vectors(b-a,c-a);
}

//orientation of point a,b,c
double orient(PT a,PT b,PT c)
{
 return cross(b-a,c-a);
}
//is point p within angle <bac
bool is_point_in_angle(PT b,PT a,PT c,PT p)
{
 assert(fabs(orient(a,b,c)-0.0)>0);
 if(orient(a,c,b)<0) swap(b,c);
 return orient(a,c,p)>=0&&orient(a,b,p)<=0;
}
//equation of bisector line of <bac
line bisector(PT b,PT a,PT c)
{
 PT unit_ab=(b-a)/(b-a).val();
 PT unit_ac=(c-a)/(c-a).val();
 PT l=unit_ab+unit_ac;
 return line(l.y,-l.x,l.x*a.y-l.y*a.x);
}
//sort points in counterclockwise;
bool half(PT p)
{
 return p.y>0.0||(p.y==0.0&&p.x<0.0);
}
void polar_sort(vector<PT>&v)

```

```

 {
 sort(all(v),[](PT a,PT b){return make_tuple(half(a),0.0)<make_tuple(half(b),cross(a,b));});
 }
 //intersection point between ab and cd
 //assuming unique intersection exists
 bool line_line_intersection(PT a,PT b,PT c,PT d,PT &out)
 {
 double a1=a.y-b.y;
 double b1=b.x-a.x;
 double c1=cross(a,b);
 double a2=c.y-d.y;
 double b2=d.x-c.x;
 double c2=cross(c,d);
 double det=a1*b2-a2*b1;
 if(det==0) return 0;
 out=PT((b1*c2-b2*c1)/det,(c1*a2-a1*c2)/det);
 return 1;
 }
 //intersection point between segment ab and segment cd
 //assuming unique intersection exists
 bool seg_seg_intersection(PT a,PT b,PT c,PT d,PT &out)
 {
 double oa=orient(c,d,a);
 double ob=orient(c,d,b);
 double oc=orient(a,b,c);
 double od=orient(a,b,d);
 //proper intersection exists iff opposite temps
 if(oa*ob<0&&oc*od<0){

```

```

 out=(a*ob-b*oa)/(ob-oa);
 return 1;
}
else return 0;
}
//intersection point between segment ab and segment cd
//assuming unique intersection may not exists
//se.size()==0 means no intersection
//se.size()==1 means one intersection
//se.size()==2 means range intersection
set<PT> seg_seg_intersection_inside(PT a, PT b, PT c, PT d)
{
 PT out;
 if (seg_seg_intersection(a,b,c,d,out)) return {out};
 set<PT> se;
 if (is_point_on_seg(c,d,a)) se.insert(a);
 if (is_point_on_seg(c,d,b)) se.insert(b);
 if (is_point_on_seg(a,b,c)) se.insert(c);
 if (is_point_on_seg(a,b,d)) se.insert(d);
 return se;
}
//intersection between segment ab and line cd
//return
//0 if do not intersect
//1 if proper intersect
//2 if segment intersect
int seg_line_relation(PT a,PT b,PT c,PT d)
{
 double p=orient(c,d,a);

```

```

 double q=orient(c,d,b);
 if(sign(p)==0&&sign(q)==0) return 2;
 //proper intersection exists iff opposite tmps
 else if(p*q<0) return 1;
 else return 0;
}
//minimum distance from segment ab to segment cd
double dist_from_seg_to_seg(PT a,PT b,PT c,PT d)
{
 PT dummy;
 if(seg_seg_intersection(a,b,c,d,dummy)) return 0.0;
 else return min({dist_from_point_to_seg(a,b,c),
 dist_from_point_to_seg(a,b,d),
 dist_from_point_to_seg(c,d,a),
 dist_from_point_to_seg(c,d,b)});
}

//a circle is defined by a center and radius
struct circle
{
 PT p;
 double r;
 circle(){}
 circle(PT _p,double _r): p(_p),r(_r){};
 //center (x,y) and radius r
 circle(double x,double y,double _r): p(PT(x,y)),r(_r){};
 //compute circle given three points i.e. circumcircle of a triangle

```

```

circle(PT a,PT b,PT c){
 b=(a+b)/2.0;
 c=(a+c)/2.0;
 line_line_intersection(b,b+rotatecw90(a-b),c,c+rotatecw90(a-
c),p);
 r=dist(a,p);
}
circle(PT a,PT b,PT c,bool t){
 line u,v;
 double m=atan2(b.y-a.y,b.x-a.x),n=atan2(c.y-a.y,c.x-a.x);
 u.a=a;
 u.b=u.a+(PT(cos((n+m)/2.0),sin((n+m)/2.0)));
 v.a=b;
 m=atan2(a.y-b.y,a.x-b.x),n=atan2(c.y-b.y,c.x-b.x);
 v.b=v.a+(PT(cos((n+m)/2.0),sin((n+m)/2.0)));
 line_line_intersection(u.a,u.b,v.a,v.b,p);
 r=dist_from_point_to_seg(a,b,p);
}
void in(){
 p.in();scanf("%lf",&r);
}
void out(){
 printf("%.10f %.10f %.10f\n",p.x,p.y,r);
}
bool operator==(circle v){
 return ((p==v.p)&&sign(r-v.r)==0);
}
bool operator<(circle v)const{
 return ((p<v.p)|| (p==v.p)&&sign(r-v.r)<0);
}
}

double area(){return PI*sqr(r);}
double circumference(){return 2.0*PI*r;}
};

istream &operator>>(istream &is, circle &a) {
 return is >> a.p >> a.r;
}
ostream &operator<<(ostream &os, circle &a) {
 return os << a.p << " " << a.r;
}
//if point is inside circle
//return
//0 outside
//1 on circumference
//2 inside circle
int circle_point_relation(PT p,double r,PT b)
{
 double dst=dist(p,b);
 if (sign(dst-r)<0) return 2;
 if (sign(dst-r)==0) return 1;
 return 0;
}
//if segment is inside circle
//return
//0 outside
//1 on circumference
//2 inside circle
int circle_seg_relation(PT p,double r,PT a,PT b)
{
}

```

```

double dst=dist_from_point_to_seg(a,b,p);
if (sign(dst-r)<0) return 2;
if (sign(dst-r)==0) return 1;
return 0;
}
//if line cross circle
//return
//0 outside
//1 on circumference
//2 inside circle
int circle_line_relation(PT p,double r,PT a,PT b)
{
 double dst=dist_from_point_to_line(a,b,p);
 if (sign(dst-r)<0) return 2;
 if (sign(dst-r)==0) return 1;
 return 0;
}
//compute intersection of line through points a and b with
//circle centered at c with radius r > 0
vector<PT> circle_line_intersection(PT c, double r ,PT a, PT b)
{
 vector<PT> ret;
 b = b-a;
 a = a-c;
 double A = dot(b, b);
 double B = dot(a, b);
 double C = dot(a, a) - r*r;
 double D = B*B - A*C;
 if (D < -eps) return ret;
 ret.pb(c+a+b*(-B+sqrt(D+eps))/A);
 if (D > eps) ret.pb(c+a+b*(-B-sqrt(D))/A);
 return ret;
}
//return
//5 - outside and do not intersect
//4 - intersect outside in one point
//3 - intersect in 2 points
//2 - intersect inside in one point
//1 - inside and do not intersect
int circle_circle_relation(PT a,double r,PT b,double R)
{
 double d=dist(a,b);
 if (sign(d-r-R)>0) return 5;
 if (sign(d-r-R)==0) return 4;
 double l=fabs(r-R);
 if (sign(d-r-R)<0&&sign(d-l)>0) return 3;
 if (sign(d-l)==0) return 2;
 if (sign(d-l)<0) return 1;
}
// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> circle_circle_intersection(PT a,double r,PT b,double R)
{
 if(a==b&&sign(r-R)==0) return {PT(1e18,1e18)};
 vector<PT> ret;
 double d = sqrt(dist2(a, b));
 if (d > r+R || d+min(r, R) < max(r, R)) return ret;
 double x = (d*d-R*R+r*r)/(2*d);
}

```

```

double y = sqrt(r*r-x*x);
PT v = (b-a)/d;
ret.pb(a+v*x + rotateccw90(v)*y);
if (y > 0) ret.pb(a+v*x - rotateccw90(v)*y);
return ret;
}
// returns two circle c1,c2 through points a,b of radius r
// returns 0 if there is no such circle
// 1 if one circle
// 2 if two circle
int getcircle(PT a,PT b,double r,circle &c1,circle &c2)
{
 vector<PT> v=circle_circle_intersection(a,r,b,r);
 int t=v.size();
 if(!t) return 0;
 c1.p=v[0],c1.r=r;
 if(t==2) c2.p=v[1],c2.r=r;
 return t;
}
// returns two circle c1,c2 which is tangent to line u, goes through
// point q and has radius r1
// returns 0 for no circle ,1 if c1=c2 ,2 if c1!=c2
int getcircle(line u,PT q,double r1,circle &c1,circle &c2)
{
 double dis=dist_from_point_to_line(u.a,u.b,q);
 if (sign(dis-r1*2)>0) return 0;
 if (sign(dis)==0)
 {
 c1.p=(q+rotateccw90(u.vec())).trunc(r1);
 c2.p=(q+rotatecw90(u.vec())).trunc(r1);
 c1.r=c2.r=r1;
 return 2;
 }
 line
 u1=line((u.a+rotateccw90(u.vec())).trunc(r1),(u.b+rotateccw90(u.vec())).trunc(r1));
 line
 u2=line((u.a+rotatecw90(u.vec())).trunc(r1),(u.b+rotatecw90(u.vec())).trunc(r1));
 circle cc=circle(q,r1);
 PT p1,p2;
 vector<PT>v;
 v=circle_line_intersection(q,r1,u1.a,u1.b);
 if (!v.size()) v=circle_line_intersection(q,r1,u2.a,u2.b);
 v.eb(v[0]);
 p1=v[0],p2=v[1];
 c1=circle(p1,r1);
 if (p1==p2)
 {
 c2=c1;
 return 1;
 }
 c2=circle(p2,r1);
 return 2;
}
//returns area of intersection between two circles
double circle_circle_area(PT a,double r1,PT b,double r2)
{
}

```

```

circle u(a,r1),v(b,r2);
int rel=circle_circle_relation(a,r1,b,r2);
if (rel>=4) return 0.0;
if (rel<=2) return min(u.area(),v.area());
double d=dist(u.p,v.p);
double hf=(u.r+v.r+d)/2.0;
double ss=2*sqrt(hf*(hf-u.r)*(hf-v.r)*(hf-d));
double a1=acos((u.r*u.r+d*d-v.r*v.r)/(2.0*u.r*d));
a1=a1*u.r*u.r;
double a2=acos((v.r*v.r+d*d-u.r*u.r)/(2.0*v.r*d));
a2=a2*v.r*v.r;
return a1+a2-ss;
}
//tangent lines i.e. sportholes from point q to circle with center p and
radius r
int tangent_lines_from_point(PT p,double r,PT q,line &u,line &v)
{
 int x=circle_point_relation(p,r,q);
 if (x==2) return 0;
 if (x==1)
 {
 u=line(q,q+rotateccw90(q-p));
 v=u;
 return 1;
 }
 double d=dist(p,q);
 double l=sqr(r)/d;
 double h=sqrt(sqr(r)-sqr(l));
 u=line(q,p+((q-p).trunc(l)+(rotateccw90(q-p).trunc(h))));
```

```

v=line(q,p+((q-p).trunc(l)+(rotatecw90(q-p).trunc(h))));

return 2;

}

//returns outer tangents line of two circles

// if inner==1 it returns inner tangent lines

int tangents_lines_from_circle(PT c1, double r1, PT c2, double r2,

bool inner, line &u,line &v)

{
 if (inner)
 r2 = -r2;
 PT d = c2-c1;
 double dr = r1-r2, d2 = d.val(), h2 = d2-dr*dr;
 if (d2 == 0 || h2 < 0)
 {
 assert(h2 != 0);
 return 0;
 }
 vector<pair<PT,PT>>out;
 for (int tmp :{-1,1}){
 PT v = (d*dr + rotateccw90(d)*sqrt(h2)*tmp)/d2;
 out.pb({c1 + v*r1, c2 + v*r2});
 }
 u=line(out[0].F,out[0].S);
 if(out.size()==2) v=line(out[1].F,out[1].S);
 return 1 + (h2 > 0);
}

//a polygon is defined by n points
```

```

//here l[] array stores lines of the polygon
struct polygon
{
 int n;
 PT p[mxp];
 line l[mxp];
 void in(int _n){
 n=_n;
 for (int i=0;i<n;i++) p[i].in();
 }
 void add(PT q){p[n++]=q;}
 void getline(){
 for (int i=0;i<n;i++)
 l[i]=line(p[i],p[(i+1)%n]);
 }
 double getcircumference()
 {
 double sum=0;
 int i;
 for (i=0;i<n;i++)
 {
 sum+=dist(p[i],p[(i+1)%n]);
 }
 return sum;
 }
 double getarea()
 {
 double sum=0;
 int i;

```

```

 for (i=0;i<n;i++)
 {
 sum+=cross(p[i],p[(i+1)%n]);
 }
 return fabs(sum)/2;
 }
 //returns 0 for cw, 1 for ccw
 bool getdir()
 {
 double sum=0;
 int i;
 for (i=0;i<n;i++)
 {
 sum+=cross(p[i],p[(i+1)%n]);
 }
 if (sign(sum)>0) return 1;
 return 0;
 }
 struct cmp{
 PT p;
 cmp(const PT &p0){p=p0;}
 bool operator()(const PT &aa,const PT &bb){
 PT a=aa,b=bb;
 int d=sign(cross(a-p,b-p));
 if (d==0) return sign(dist(a,p)-dist(b,p))<0;
 return d>0;
 }
 };
 //sorting in convex_hull order

```

```

void norm(){
 PT mi=p[0];
 for (int i=1;i<n;i++)mi=min(mi,p[i]);
 sort(p,p+n,cmp(mi));
}
//returns convex hull in convex (monotone chain)
void getconvex(polygon &convex)
{
 int i,j,k;
 sort(p,p+n);
 convex.n=n;
 for (i=0;i<min(n,2);i++)
 {
 convex.p[i]=p[i];
 }
 if (n<=2) return;
 int &top=convex.n;
 top=1;
 for (i=2;i<n;i++)
 {
 while (top&&cross(convex.p[top]-p[i],convex.p[top-1]-p[i])<=0)
 top--;
 convex.p[++top]=p[i];
 }
 int temp=top;
 convex.p[++top]=p[n-2];
 for (i=n-3;i>=0;i--)
 {

```

```

 while (top!=temp&&cross(convex.p[top]-p[i],convex.p[top-1]-p[i])<=0)
 top--;
 convex.p[++top]=p[i];
 }
}
//checks if convex
bool isconvex()
{
 bool s[3];
 memset(s,0,sizeof(s));
 int i,j,k;
 for (i=0;i<n;i++)
 {
 j=(i+1)%n;
 k=(j+1)%n;
 s[sign(cross(p[j]-p[i],p[k]-p[i]))+1]=1;
 if (s[0]&&s[2])return 0;
 }
 return 1;
}
//used for later function
double xmult(PT a, PT b, PT c)
{
 return cross(b - a,c - a);
}
// returns
// 3 - if q is a vertex
// 2 - if on a side

```

```

// 1 - if inside
// 0 - if outside
int relation_point(PT q){
 int i,j;
 for (i=0;i<n;i++){
 if (p[i]==q) return 3;
 }
 getline();
 for (i=0;i<n;i++){
 if (is_point_on_seg(l[i].a,l[i].b,q)) return 2;
 }
 int cnt=0;
 for (i=0;i<n;i++){
 j=(i+1)%n;
 int k=sign(cross(q-p[j],p[i]-p[j]));
 int u=sign(p[i].y-q.y);
 int v=sign(p[j].y-q.y);
 if (k>0&&u<0&&v>0)cnt++;
 if (k<0&&v<0&&u>0)cnt--;
 }
 return cnt!=0;
}
//returns minimum enclosing circle
void find_(int st,PT tri[],circle &c){
 if (!st) c=circle(PT(0,0),-2);
 if (st==1) c=circle(tri[0],0);
 if (st==2) c=circle((tri[0]+tri[1])/2.0,dist(tri[0],tri[1])/2.0);
 if (st==3) c=circle(tri[0],tri[1],tri[2]);
}

```

```

void solve(int cur,int st,PT tri[],circle &c){
 find_(st,tri,c);
 if (st==3) return;
 int i;
 for (i=0;i<cur;i++){
 if (sign(dist(p[i],c.p)-c.r)>0){
 tri[st]=p[i];
 solve(i,st+1,tri,c);
 }
 }
}
circle minimum_enclosing_circle(){
 random_shuffle(p,p+n);
 PT tri[4];
 circle c;
 solve(n,0,tri,c);
 return c;
};

//stores some polygons
struct polygons{
 vectorp;
 polygons(){p.clear();}
 void clear(){p.clear();}
 void push(polygon q){if (sign(q.getarea()))p.pb(q);}
 vector<pair<double,int> >e;
}

```

```

//used for later use
void ins(PT s,PT t,PT X,int i){
 double r=fabs(t.x-s.x)>eps?(X.x-s.x)/(t.x-s.x):(X.y-s.y)/(t.y-s.y);
 r=fmin(r,1.0);r=fmax(r,0.0);
 e.pb(MP(r,i));
}
double polyareaunion(){
 double ans=0.0;
 int c0,c1,c2,i,j,k,w;
 for (i=0;i<p.size();i++)
 if (p[i].getdir()==0)reverse(p[i].p,p[i].p+p[i].n);
 for (i=0;i<p.size();i++){
 for (k=0;k<p[i].n;k++){
 PT &s=p[i].p[k],&t=p[i].p[(k+1)%p[i].n];
 if (!sign(cross(s,t)))continue;
 e.clear();
 e.pb(MP(0.0,1));
 e.pb(MP(1.0,-1));
 for (j=0;j<p.size();j++)if (i!=j){
 for (w=0;w<p[j].n;w++){
 PT a=p[j].p[w],b=p[j].p[(w+1)%p[j].n],c=p[j].p[(w-
1+p[j].n)%p[j].n];
 c0=sign(cross(t-s,c-s));
 c1=sign(cross(t-s,a-s));
 c2=sign(cross(t-s,b-s));
 if (c1*c2<0)ins(s,t,line(s,t).cross_point(line(a,b)),c2);
 else if (!c1&&c0*c2<0)ins(s,t,a,-c2);
 else if (!c1&&!c2){
 int c3=sign(cross(t-s,p[j].p[(w+2)%p[j].n]-s));

```

```

 int dp=sign(dot(t-s,b-a));
 if (dp&&c0)ins(s,t,a,dp>0?c0*((j>i)&(c0<0)):(c0>0));
 if (dp&&c3)ins(s,t,b,dp>0?-c3*((j>i)&(c3<0)):(c3>0));
 }
 }
 }
 sort(e.begin(),e.end());
 int ct=0;
 double tot=0.0,last;
 for (j=0;j<e.size();j++){
 if (ct==2)tot+=e[j].first-last;
 ct+=e[j].second;
 last=e[j].first;
 }
 ans+=cross(s,t)*tot;
 }
 }
 return fabs(ans)*0.5;
}
int main()
{
 fast;
 PT a(1,0),b(2,0),c(1,2),d(3,4);
 circle x(a,b,c,0);
 x.out();
 return 0;
}

```

## 111. Convex Hull

```

int sign(double d)
{
 if (fabs(d)<eps) return 0;
 return d>eps?1:-1;
}
struct PT
{
 double x,y;
 PT() {}
 PT(double x, double y) : x(x), y(y) {}
 PT operator + (const PT &a) const{
 return PT(x+a.x,y+a.y);
 }
 PT operator - (const PT &a) const{
 return PT(x-a.x,y-a.y);
 }
 PT operator * (const double a) const{
 return PT(x*a,y*a);
 }
 bool operator==(PT a) const
 {
 return sign(a.x-x)==0&&sign(a.y-y)==0;
 }
 bool operator<(PT a) const
 {
 return sign(a.x-x)==0?sign(y-a.y)<0:x<a.x;
 }
}

```

```

bool operator>(PT a) const
{
 return sign(a.x-x)==0?sign(y-a.y)>0:x>a.x;
}
double val()
{
 return sqrt(x*x+y*y);
}
double cross(PT a,PT b)
{
 return a.x*b.y-a.y*b.x;
}
bool cmp(PT p,PT q)
{
 return mt(p.x,p.y)<mt(q.x,q.y);
}
vector<PT> hull(vector<PT> a) {
 sort(all(a),cmp);
 a.resize(unique(a.begin(), a.end()) - a.begin());
 if((int)a.size()==1) return a;
 vector<PT> res;
 int l = 0;
 for(int i=0;i<2;i++) {
 for(auto & C : a) {
 while((int) res.size() >= l + 2) {
 PT A = res[(int) res.size() - 2];
 PT B = res.back();
 if(cross((C-A),(B-A)) >= 0) break;
 }
 res.push_back(C);
 }
 }
 return res;
}

```

```

 res.pop_back();
 }
 res.pb(C);
}
res.pop_back();
reverse(a.begin(), a.end());
l = (int) res.size();
}
return res;
}
int main()
{
 fast;
 ll i,j,k,n,m;
 cin>>n;
 vector<PT> v(n);
 for(i=0;i<n;i++) cin>>v[i].x>>v[i].y;
 vector<PT>a=hull(v);
 ll sz=a.size();
 double ans=0;
 for(i=0;i<sz;i++) ans+=(a[(i+1)%sz]-a[i]).val();
 cout<<fout(10)<<ans<<nl;
 return 0;
}

```

## 112. Pick's Theorem

Given a certain lattice polygon with non-zero area.

We denote its area by  $S$ , the number of points with integer coordinates lying strictly inside the polygon by  $I$  and the number of points lying on polygon sides by  $B$ .

Then, the Pick's formula states:

$$S = I + \frac{B}{2} - 1$$

In particular, if the values of  $I$  and  $B$  for a polygon are given, the area can be calculated in  $O(1)$  without even knowing the vertices.

## 113. Closest Pair of Points

```

struct PT
{
 double x,y;
 int idx;
 PT(double xt=0,double yt=0,int zt=0)
 {
 x=xt,y=yt,idx=zt;
 }
};
struct comp
{
 bool operator ()(const PT &a,const PT &b)
 const
 {
 if(a.y!=b.y) return a.y<b.y;
 return a.x<b.x;
 }
};

```

```

bool cmp(PT a,PT b)
{
 return(a.x!=b.x)?(a.x<b.x):(a.y<b.y);
}
double SD(PT a,PT b)
{
 return SQ(a.x-b.x)+SQ(a.y-b.y);
}
///returns the indexes of the closest points.
pii ClosestPair(vector<PT>p)
{
 int l,r,ci,cj,i;
 int n=p.size();
 double dis,m;
 set<PT,comp>se;
 PT tmp;
 for(i=0; i<n; i++) p[i].idx = i;
 sort(all(p),cmp);
 ci=p[0].idx;
 cj=p[1].idx;
 m = SD(p[0],p[1]);
 se.insert(p[0]);
 se.insert(p[1]);
 l=0;
 r=2;
 while(r<n)
 {
 while(l<r&&SQ(p[l].x-p[r].x)>=m)
 {
 se.erase(p[l]);
 l++;
 }
 dis=sqrt(m);
 auto itl = se.lower_bound(PT(p[r].x,p[r].y-dis));
 auto itr = se.upper_bound(PT(p[r].x,p[r].y+dis));
 while(itl!=itr)
 {
 dis = SD(*itl,p[r]);
 if(dis<m)
 {
 m=dis;
 ci=itl->idx;
 cj = p[r].idx;
 }
 itl++;
 }
 se.insert(p[r]);
 r++;
 }
 return pii(ci,cj);
}
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin>>n;
 vector<PT>v;
 int x,y;

```

```

for(i=0;i<n;i++) cin>>x>>y,y.eb(PT(1.0*x,1.0*y,0));
pii ans=ClosestPair(v);
if(ans.F>ans.S) swap(ans.F,ans.S);
cout<<ans.F<<'<<ans.S<<
'<<fout(6)<<sqrt(SD(v[ans.F],v[ans.S]))<<nl;
return 0;
}

```

## MATRIX RELATED ALGORITHM

### 114. The Art of Problem Solving

- Found something like  $c[i][j] = \min_{1 \leq k \leq j} (a[i][k] + b[k][j])?$

Think about matrix!

### 115. Matrix Structure

```

struct matrix
{
 int n;
 vector<vector<int> > t;

 matrix() { }
 matrix(int _n, int val) {n = _n; t.assign(n, vector<int>(n, val)); }
 matrix(int _n) {n = _n; t.assign(n, vector<int>(n, 0)); }
 matrix(vector<vi> v){n=v.size();t=v;}
 matrix unit(int _n)
 {
 matrix ans=matrix(_n,0);
 for(int i=0;i<_n;i++) ans.t[i][i]=1;
 }
}

```

```

 return ans;
}
//make sure to erase mod if not needed
matrix operator * (matrix b)
{
 matrix c = matrix(n, 0);
 for(int i = 0; i < n; i++)
 for(int k = 0; k < n; k++)
 for(int j = 0; j < n; j++)
 c.t[i][j] = (c.t[i][j] + 1LL*t[i][k] *
b.t[k][j]) % mod;
 return c;
}
matrix operator | (matrix b)
{
 matrix c = matrix(n, -inf);
 for(int i = 0; i < n; i++)
 for(int k = 0; k < n; k++)
 for(int j = 0; j < n; j++)
 c.t[i][j] = max(c.t[i][j], t[i][k] +
b.t[k][j]);
 return c;
}
matrix pow(ll k)
{
 if(k==0) return unit(n);
 k--;
 matrix a=*this,ans=a;
 while(k){

```

```

if(k&1) ans=ans*a;
a=a*a;
k>>=1;
}
return ans;
}
matrix pow_max(ll k)
{
if(k==0) return unit(n);
k--;
matrix a=*this,ans=a;
while(k){
if(k&1) ans=ans|a;
a=a|a;
k>>=1;
}
return ans;
}
matrix operator + (matrix a)
{
matrix ans=matrix(n,n);
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
ans.t[i][j]=t[i][j]+a.t[i][j];
return ans;
}

matrix operator - (matrix a)
{

```

```

matrix ans=matrix(n,n);
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
ans.t[i][j]=t[i][j]-a.t[i][j];
return ans;
}
bool operator == (const matrix& a)
{
return t==a.t;
}
bool operator != (const matrix& a)
{
return t!=a.t;
}
void print()
{
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
cout<<t[i][j]<<"\n"[j==n-1];
}
};
```

## 116. Gaussian Elimination

### Standard

```
/**
```

The input to the function gauss is the system matrix a.

The last column of this matrix is vector b.

The function returns the number of solutions of the system (0,1,or 2).

If at least one solution exists, then it is returned in the vector ans.  
returns-

```

0 if no solution
1 if one solution
2 if infinity solution
*/
int gauss(vector<vector<double>>a,vector<double>&ans)
{
 int n=(int)a.size();
 int m=(int)a[0].size()-1;
 vi where(m,-1);
 for(int col=0,row=0;col<m&&row<n;++col){
 int mx=row;
 for(int i=row;i<n;i++) if(fabs(a[i][col])>fabs(a[mx][col])) mx=i;
 if(fabs(a[mx][col])<eps) continue;
 for(int i=col;i<=m;i++) swap(a[row][i],a[mx][i]);
 where[col]=row;
 for(int i=0;i<n;i++){
 if(i!=row){
 double c=a[i][col]/a[row][col];
 for(int j=col;j<=m;j++) a[i][j]-=a[row][j]*c;
 }
 }
 ++row;
 }
 ans.assign(m,0);
 for(int i=0;i<m;i++){
 if(where[i]!=-1){
 ans[i]=a[where[i]][m]/a[where[i]][i];
 }
 }
}

```

```

 }
}

for(int i=0;i<n;i++){
 double sum=0;
 for(int j=0;j<m;j++) sum+=ans[j]*a[i][j];
 if(fabs(sum-a[i][m])>eps) return 0;
}

for(int i=0;i<m;i++) if(where[i]==-1) return 2;
return 1;
}

int main()
{
 fast;
 ll i,j,k,n,m;
 double x;
 cin>>n;
 vector<vector<double>>v(n);
 for(i=0;i<n;i++){
 for(j=0;j<n+1;j++){
 cin>>x;
 v[i].eb(x);
 }
 }
 vector<double>ans;
 k=gauss(v,ans);
 if(k) for(i=0;i<n;i++) cout<<fout(5)<<ans[i]<<' ';
 else cout<<"no solution\n";
 return 0;
}
```

**Modular**

```
#define MAXROW 1010
#define MAXCOL 1010
int ar[MAXROW][MAXCOL];
/***
Gaussian elimination in field MOD (MOD should be a prime)
n is number of equations and m is number of variables
last column is the non variable column
returns free_var where number of solutions=MOD^free_var
if free_var== -1 then there is no solution
res stores any solution of the equations
**/>
int gauss(int n, int m, int MOD, vector<int>& res){
 res.assign(m, 0);
 vector<int> pos(m, -1);
 int i, j, k, l, p, d, free_var = 0;
 const long long MODSQ = (long long)MOD * MOD;

 for (j = 0, i = 0; j < m && i < n; j++){
 for (k = i, p = i; k < n; k++){
 if (ar[k][j] > ar[p][j]) p = k;
 }
 if (ar[p][j]){
 pos[j] = i;
 for (l = j; l <= m; l++) swap(ar[p][l], ar[i][l]);

 d = qpow(ar[i][j], MOD - 2, MOD);
 for (k = 0; k < n && d; k++){
 if (k != i && ar[k][j]){
 int x = ((long long)ar[k][j] * d) % MOD;
 for (l = j; l <= m && x; l++){
 if (ar[i][l]) ar[k][l] = (MODSQ + ar[k][l] - ((long long)ar[i][l] * x)) % MOD;
 }
 }
 }
 i++;
 }
 }
}
```

```
if (k != i && ar[k][j]){
 int x = ((long long)ar[k][j] * d) % MOD;
 for (l = j; l <= m && x; l++){
 if (ar[i][l]) ar[k][l] = (MODSQ + ar[k][l] - ((long long)ar[i][l] * x)) % MOD;
 }
}
}
i++;
}

for (i = 0; i < m; i++){
 if (pos[i] == -1) free_var++;
 else res[i] = ((long long)ar[pos[i]][m] * qpow(ar[pos[i]][i], MOD - 2, MOD)) % MOD;
}

for (i = 0; i < n; i++) {
 long long val = 0;
 for (j = 0; j < m; j++) val = (val + ((long long)res[j] * ar[i][j])) % MOD;
 if (val != ar[i][m]) return -1;
}
return free_var;
}

int32_t main()
{
```

```

BeatMeScanf;
int i,j,k,n,m;
cin>>n>>m;
for(i=0;i<n;i++) for(j=0;j<=m;j++) cin>>ar[i][j];
vi res;
k=gauss(n,m,mod,res);
if(k==-1) cout<<"no solution\n";
else printv(res);
return 0;
}

```

## Modulo 2

```

///32 times faster for modulo 2
int gauss(vector < bitset<N> > &a, bitset<N> &ans, int n, int m)
{
 int Rank=0,Det=1;
 vi where(N,-1);
 for(int col = 0, row = 0; col < m && row < n; ++col) {
 int sel = row;
 for(int i = row; i < n; ++i) if(a[i][col]) { sel = i; break; }
 if(!a[sel][col]) { Det = 0; continue; }
 swap(a[sel], a[row]);
 if(row != sel) Det = -Det;
 Det&=a[row][col];
 where[col] = row;

 for(int i = 0; i < n; ++i) if (i != row && a[i][col] > 0) a[i] ^= a[row];
 ++row, ++Rank;
 }
}

```

```

for(int i = 0; i < m; ++i) ans[i] = (where[i] == -1) ? 0 : a[where[i]][m];
for(int i = Rank; i < n; ++i) if(a[i][m]) return 0;///no solution
int free_var=0;
for(int i = 0; i < m; ++i) if(where[i] == -1) free_var++;
int number_of_solution=qpow(2,free_var);
return 1;///has solution
}

int main()
{
 fast;
 ll i,j,k,n,m;
 return 0;
}

XOR Basis
vi basis;
int add(int x)
{
 for(auto &it:basis) if((x^it)<x) x^=it;
 for(auto &it:basis) if((it^x)<it) it^=x;
 if(x) basis.pb(x);
}
///if any subset xor equal to x
bool pos(int x)
{
 for(auto &it:basis) if((x^it)<x) x^=it;
 return (x==0);
}

```

```

int kth(int k)
{
 srt(basis);
 k--;
 int ans=0;
 //k th subset xor is the xor value of positions where k has one in its
 binary representation
 for(int i=0;i<basis.size();i++) if((k>>i)&1) ans^=basis[i];
 return ans;
 //maximum xor value is the xor of all values of basis vector
}
int main()
{
 BeatMeScanf;
 cin.tie(NULL);
 int i,j,k,n,m;
 cin>>n;
 while(n--){
 int ty;
 cin>>ty>>m;
 if(ty==1) add(m);///add value m to the set
 else cout<<kth(m)<<nl;///find k-th subset xor of the set
 }
 return 0;
}

```

### Determinant

```

double determinant(vector<vector<double>>a)
{

```

```

 int n=a.size();
 double det = 1;
 for (int i=0; i<n; ++i)
 {
 int k = i;
 for (int j=i+1; j<n; ++j)
 if (abs (a[j][i]) > abs (a[k][i]))
 k = j;
 if (abs (a[k][i]) < eps)
 {
 det = 0;
 break;
 }
 swap (a[i], a[k]);
 if (i != k)
 det = -det;
 det *= a[i][i];
 for (int j=i+1; j<n; ++j)
 a[i][j] /= a[i][i];
 for (int j=0; j<n; ++j)
 if (j != i && abs (a[j][i]) > eps)
 for (int k=i+1; k<n; ++k)
 a[j][k] -= a[i][k] * a[j][i];
 }
 return det;
}
int main()
{

```

```

fast;
int n;
vector < vector<double> > a (n, vector<double> (n));
cout<<determinant(a)<<n;
return 0;
}

```

### Determinant Modular

```

void Egcd (int a, int b, int &x, int &y) //extended gcd
{
 if (b == 0)
 {
 x = 1, y = 0;
 return ;
 }
 Egcd (b, a%b, x, y);
 int tp = x;
 x = y;
 y = tp - a/b*y;
}

/// Returns 0 if the matrix is singular or degenerate (hence no
determinant exists)
int det (vector < vector < long long > > a,int mod) //determinant of
a square matrix
{
 int n=(int) a. size ();
 int i, j, k, ans = 1, x, y, flg = 1;
 for (i = 0; i < n; i++)

```

```

 {
 if (a[i][i] == 0)
 {
 for (j = i+1; j < n; j++)
 if (a[j][i])
 break;
 if (j == n)
 return -1;
 flg = !flg;
 for (k = i; k < n; k++)
 swap (a[i][k], a[j][k]);
 }
 ans = 1LL*ans * a[i][i] % mod;
 Egcd (a[i][i], mod, x, y); //inverse modulo
 x = (x%mod + mod) % mod;
 for (k = i+1; k < n; k++)
 a[i][k] = a[i][k] * x % mod;
 for (j = i+1; j < n; j++)
 if (a[j][i] != 0)
 for (k = i+1; k < n; k++)
 a[j][k] = ((a[j][k] - a[i][k]*a[j][i])%mod + mod) % mod;
 }
 if (flg)
 return ans;
 return mod-ans;
}

int main()
{

```

```

BeatMeScanf;
ll i,j,k,n,m;
cin>>n;
vector<vector<ll>>v(n,vector<ll>(n,0));
for(i=0;i<n;i++) for(j=0;j<n;j++) cin>>v[i][j];
cout<<det(v,mod)<<nl;
return 0;
}

```

## Gauss-Jordan Elimination

```

// Gauss-Jordan elimination with full pivoting.
//
// Uses:
// (1) solving systems of linear equations (AX=B)
// (2) inverting matrices (AX=I)
// (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT: a[][] = an nxn matrix
// b[][] = an nxm matrix
//
// OUTPUT: X = an n xm matrix (stored in b[][])
// A^{-1} = an nxn matrix (stored in a[][])
// returns determinant of a[][]
//
// Example used: LightOJ Snakes and Ladders

#include <iostream>

```

```

#include <vector>
#include <cmath>

using namespace std;

const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {
 const int n = a.size();
 const int m = b[0].size();
 VI irow(n), icol(n), ipiv(n);
 T det = 1;

 for (int i = 0; i < n; i++) {
 int pj = -1, pk = -1;
 for (int j = 0; j < n; j++) if (!ipiv[j])
 for (int k = 0; k < n; k++) if (!ipiv[k])
 if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
 if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl;
 exit(0); }
 ipiv[pk]++;
 swap(a[pj], a[pk]);
 swap(b[pj], b[pk]);
 if (pj != pk) det *= -1;
 }
}

```

```

irow[i] = pj;
icol[i] = pk;

T c = 1.0 / a[pk][pk];
det *= a[pk][pk];
a[pk][pk] = 1.0;
for (int p = 0; p < n; p++) a[pk][p] *= c;
for (int p = 0; p < m; p++) b[pk][p] *= c;
for (int p = 0; p < n; p++) if (p != pk) {
 c = a[p][pk];
 a[p][pk] = 0;
 for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
 for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
}
}

for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
 for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
}

return det;
}

int main() {
 const int n = 4;
 const int m = 2;
 double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
 double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
 VVT a(n), b(n);
}

```

```

for (int i = 0; i < n; i++) {
 a[i] = VT(A[i], A[i] + n);
 b[i] = VT(B[i], B[i] + m);
}

double det = GaussJordan(a, b);

// expected: 60
cout << "Determinant: " << det << endl;

// expected: -0.233333 0.166667 0.133333 0.0666667
// 0.166667 0.166667 0.333333 -0.333333
// 0.233333 0.833333 -0.133333 -0.0666667
// 0.05 -0.75 -0.1 0.2
cout << "Inverse: " << endl;
for (int i = 0; i < n; i++) {
 for (int j = 0; j < n; j++)
 cout << a[i][j] << ' ';
 cout << endl;
}

// expected: 1.63333 1.3
// -0.166667 0.5
// 2.36667 1.7
// -1.85 -1.35
cout << "Solution: " << endl;
for (int i = 0; i < n; i++) {
 for (int j = 0; j < m; j++)
 cout << b[i][j] << ' ';
}

```

```

cout << endl;
}

Thomas Algorithm
/// Equation of the form: (x_prev * l) + (x_cur * p) + (x_next * r) = rhs
/***
n equations of form:

a_i*x_{i-1}+b_i*x_i+c_i*x_{i+1}=d_i, for i=1 to n
where a_1=0,c_n=0

matrix form:
b1 c1 0 0 0 | x1 d1
a2 b2 c2 0 0 | x2 d2
0 a3 b3 c3 0 | x3 = d3
0 0 a4 b4 c4 | x4 d4
0 0 0 a5 b5 | x5 d5
*/
struct equation{
 long double l, p, r, rhs;

 equation(){}
 equation(long double l, long double p, long double r, long double
rhs = 0.0):
 l(l), p(p), r(r), rhs(rhs){}
};

/// Thomas algorithm to solve tri-diagonal system of equations in O(n)

```

```

vector <long double> thomas_algorithm(int n, vector <struct
equation> ar){
 ar[0].r = ar[0].r / ar[0].p;
 ar[0].rhs = ar[0].rhs / ar[0].p;
 for (int i = 1; i < n; i++){
 long double v = 1.0 / (ar[i].p - ar[i].l * ar[i - 1].r);
 ar[i].r = ar[i].r * v;
 ar[i].rhs = (ar[i].rhs - ar[i].l * ar[i - 1].rhs) * v;
 }
 for (int i = n - 2; i >= 0; i--) ar[i].rhs = ar[i].rhs - ar[i].r * ar[i + 1].rhs;
 vector <long double> res;
 for (int i = 0; i < n; i++) res.push_back(ar[i].rhs);
 return res;
}
int a[N],b[N],c[N],d[N];
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin>>n;
 //make sure that a[1]=0 and c[n]=0
 for(i=0;i<n;i++) cin>>a[i]>>b[i]>>c[i]>>d[i];
 vector<equation>v;
 for(i=0;i<n;i++) v.pb(equation(1.0*a[i],1.0*b[i],1.0*c[i],1.0*d[i]));
 vector<long double>ans=thomas_algorithm(n,v);
 for(i=0;i<n;i++) cout<<ans[i]<<' ';
 return 0;
}

```

## 117. Mat Expo

### Notes

#### Odd even variation

\* Odd/even conditional function

The function  $f$  may behave differently according to the parity of the argument. For example, if  $i$  is even, then  $f(i) = f(i - 1)/2$ , otherwise (if  $i$  is odd)  $f(i) = 3f(i - 1) + 1$ .

How to solve this variant? It is possible to split the equation into even and odd parts. We construct two matrices  $T_{\text{even}}$  and  $T_{\text{odd}}$  such that:

- $T_{\text{even}}F_i = F_{i+1}$ ,  $i$  is even
- $T_{\text{odd}}F_i = F_{i+1}$ ,  $i$  is odd

We also construct matrix  $T = T_{\text{even}} \cdot T_{\text{odd}}$ . Now, we can compute  $F_N$  with a single formula:

- $F_N = T^{N/2} \cdot F_1$ , if  $N$  is odd,
- $F_N = T_{\text{odd}} \cdot T^{(N-1)/2} \cdot F_1$ , otherwise.

Note that  $T_{\text{even}}$  and  $T_{\text{odd}}$  must be of the same size, so in this example, convert  $f(i) = f(i - 1)/2$  into  $f(i) = \frac{1}{2}f(i - 1) + 0$  to make it look like the odd part, and then apply the standard method.

here  $F_i$ =all  $a[i][k]$ s of given 2d array  $a$ .

$T_{\text{even}}$ =transition for odd  $n$  and vice versa.

#or, try to convert  $f[\text{odd } i]$  and  $f[\text{even } i]$  into two functions  $g[i]=f[2*i]$  and  $h[i]=f[2*i+1]$  each having mutual transition

2D

Let there be  $K$  states,  $s_1, s_2 \dots s_K$ .

For each state  $s_i$  where  $i = 1$  to  $K$  recurrence is defined as:

$$f(n, s_i) = \sum_{j=1}^K c_{i,j} f(n - 1, s_j).$$

We define a transition matrix  $M$  where  $M[i][j] = c_{i,j}$ .

Let's consider  $M^{N-1}$ . Sum of all elements in row  $i$  will give us  $f(N, s_i)$  assuming  $f(0, i) = 0 \forall i$ .

It uses the number of paths in a graph of length  $N$ . (Exponentiate the adjacency matrix of the graph to  $N$  and the value of the  $(i, j)$ th element in the exponentiated matrix is the number of paths of length  $N$  from the  $i$ th vertex to  $j$ th vertex).

Now, in each of these problems, consider the states as vertices and keep on adding edges from each vertex and then find the answer. :)

Like, in "ASCI Nice Patterns Strike Back", each bitmask is a vertex, from each bitmask, find to which all bitmasks you can go by checking all of the others. The graph is made! Now simply exponentiate and find the answer! :)

for much more dimensional DP, say 4 dimensional transition, we can still solve it.

for this, We map each triplet of  $(i, j, k)$  to a new number which will denote a new state and create the transition matrix. Now the problem is similar as a 2-D DP matrix exponentiation problem.

#### Mat Expo Standard

\*use matrix structure here\*

```
int main()
{
 int n,i,j,t,f0,f1,m,cs=0;
 cin>>t;
 while(t--){
 cin>>f0>>f1>>n;
 vector<vi>a(2,vi(2,0));
```

```

a[0][0]=1;
a[0][1]=1;
a[1][0]=1;
matrix res=matrix(a).pow(n-1);
ll ans=(1LL*res.t[0][0]*f1%mod+1LL*res.t[0][1]*f0%mod)%mod;
pf("Case %d: %lld\n",++cs,ans);
}
return 0;
}

```

## 118. Linear Recurrence

### Slow

```

///O(k2logn)
///transition-> for(i=0;i<x;i++) f[n]=tr[i]*f[n-i-1]
///S=starting recurrence
///k is 0-indexed
long long linearRec(vector<long long> S, vector<long long> tr, long
long k) {
 long long n = S.size();
 if(n==0) return 0;
 auto combine = [&](vector<long long> a, vector<long long> b)
{
 vector<long long> res(n * 2 + 1);
 for (long long i=0; i<n+1; i++) for (long long j=0; j<n+1;
j++)
 res[i + j] = (res[i + j] + a[i] * b[j]) % Mod;
 for (long long i = 2 * n; i > n; --i) for (long long j=0; j<n;
j++)
 res[i + j] = (res[i + j] + a[i] * b[j]) % Mod;
 return res;
}

```

```

 res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % Mod;
 res.resize(n + 1);
 return res;
};

vector<long long> pol(n + 1), e(pol);
pol[0] = e[1] = 1;

for (++k; k /= 2) {
 if (k % 2) pol = combine(pol, e);
 e = combine(e, e);
}

long long res = 0;
for (long long i=0; i<n; i++) res = (res + pol[i + 1] * S[i]) % Mod;
return res;
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;

 return 0;
}

Fast
///Ai = (Ai-1.P1 + Ai-2.P2 + ... + Ai-k.Pk) mod K, for i = k+1, k+2...
///returns An

```

```

///Complexity: k(log k)(log n) with heavy constant factor (for
k=30000,n=10^18) it takes 4s
#include <bits/stdc++.h>
/*
 * FFT implementation with double by dacin21
 * splits digits into the imaginary part to cope with bigger numbers
*/
using namespace std;
using ll = long long;

namespace fft{
// floored base 2 logarithm
int log2i(unsigned long long a){
 return __builtin_clzll(1) - __builtin_clzll(a);
}
const double PI = 3.1415926535897932384626;
vector<complex<double>> roots;
// pre-calculate complex roots, log(N) calls to sin/cos
void gen_roots(int N){
 if((int)roots.size()!=N){
 roots.clear();
 roots.resize(N);
 for(int i=0;i<N;++i){
 if((i&-i) == i){
 roots[i] = polar(1.0, 2.0*PI*i/N);
 } else {
 roots[i] = roots[i&-i] * roots[i-(i&-i)];
 }
 }
 }
}

```

```

 }
}
void fft(complex<double> const*a, complex<double> *to, int n, bool
isInv = false){
 to[0] = a[0];
 for (int i=1, j=0; i<n; ++i) {
 int m = n >> 1;
 for (; j>=m; m >>= 1)
 j -= m;
 j += m;
 to[i] = a[j];
 }
 gen_roots(n);
 for(int iter=1, sh=log2i(n)-1;iter<n;iter*=2, --sh){
 for(int x=0;x<n;x+=2*iter){
 for(int y=0;y<iter;++y){
 complex<double> ome = roots[y<<sh];
 if(isInv) ome = conj(ome);
 complex<double> v = to[x+y], w=to[x+y+iter];
 to[x+y] = v+ome*w;
 to[x+y+iter] = v-ome*w;
 }
 }
 }
}
template<ll mod, typename int_t>
vector<int_t> poly_mul(vector<int_t> const&a, vector<int_t>
const&b){

```

```

int logn = log2i(a.size()+b.size()-1)+1;
int n = 1<<logn;
vector<complex<double> > x(n), y(n), xx(n), yy(n);
// split digit into real and imaginary part
for(int i=0;i<(int)a.size();++i) x[i] = complex<double>(a[i]&((1<<15)-1), a[i]>>15);
for(int i=0;i<(int)b.size();++i) y[i] = complex<double>(b[i]&((1<<15)-1), b[i]>>15);

fft(x.data(), xx.data(), n, false);
fft(y.data(), yy.data(), n, false);
// use that fft(conj(x)) = reverse(conj(fft(x)))
// to recover fft(real(x)) and fft(imag(x))
for(int i=0;i<n;++i){
 int j = (n-i)&(n-1); //reverse index
 complex<double> rx = (xx[i] + conj(xx[j]))*0.5;
 complex<double> ix = (xx[i] - conj(xx[j]))*complex<double>(0, -0.5);
 complex<double> ry = (yy[i] + conj(yy[j]))*0.5;
 complex<double> iy = (yy[i] - conj(yy[j]))*complex<double>(0, -0.5);
 x[i] = (rx*ry + ix*iy*complex<double>(0, 1.0))/(double)n;
 y[i] = (rx*iy + ix*ry)/(double)n;
}
fft(x.data(), xx.data(), n, true);
fft(y.data(), yy.data(), n, true);
vector<int_t> ret(a.size()+b.size()-1);
for(int i=0;i<(int)ret.size();++i){

```

```

 ll l = llround(xx[i].real()), m = llround(yy[i].real());
 r=llround(xx[i].imag());
 ret[i] = (l + (m%mod<<15) + (r%mod<<30))%mod;
}
return ret;
}
template<ll mod>
struct NT{
 static int add(int const&a, int const&b){
 ll ret = a+b;
 if(ret>=mod) ret-=mod;
 return ret;
 }
 static int& xadd(int& a, int const&b){
 a+=b;
 if(a>=mod) a-=mod;
 return a;
 }
 static int sub(int const&a, int const&b){
 return add(a, mod-b);
 }
 static int& xsub(int& a, int const&b){
 return xadd(a, mod-b);
 }
 static int mul(int const&a, int const&b){
 return a*(ll)b%mod;
 }
 static int& xmul(int &a, int const&b){

```

```

 return a=mul(a, b);
}
static int inv_rec(int const&a, int const&m){
 assert(a!=0);
 if(a==1) return 1;
 int ret = m+(1-inv_rec(m%a, a)*(ll)m)/a;
 return ret;
}
// this is soooo great, can even be used for a sieve
static int inv_rec_2(int const&a, int const&m){
 assert(a!=0);
 if(a==1) return 1;
 int ret = m-NT<mod>::mul((m/a), inv_rec_2(m%a, m));
 return ret;
}
static int inv(int const&a){
 return inv_rec_2(a, mod);
}
};

template<ll mod>
struct poly : vector<int>{
 poly(size_t a):vector<int>(a){}
 poly(size_t a, int b):vector<int>(a, b){}
 poly(vector<int> const&a):vector<int>(a){}
 poly& normalize(){
 while(size()>1 && back()==0) pop_back();
 return *this;
 }
}

```

```

poly substr(int l, int r)const{
 if(r>(int)size()) r=size();
 if(l>(int)size()) l=size();
 return poly(vector<int>(begin()+l, begin()+r));
}
poly reversed()const{
 return poly(vector<int>(rbegin(), rend()));
}
poly operator+(poly const&o)const{
 poly ret(max(size(), o.size()));
 copy(begin(), end(), ret.begin());
 for(int i=0;i<(int)o.size();++i)
 NT<mod>::xadd(ret[i], o[i]);
 return ret.normalize();
}
poly operator-(poly const&o)const{
 poly ret(max(size(), o.size()));
 copy(begin(), end(), ret.begin());
 for(int i=0;i<(int)o.size();++i)
 NT<mod>::xsub(ret[i], o[i]);
 return ret.normalize();
}
poly operator*(poly const&o)const{
 poly ret(fft::poly_mul<mod, int>(*this, o));
 return ret.normalize();
}
friend ostream& operator<<(ostream&o, poly const&p){
 for(int i=(int)p.size()-1;i>=0;--i){
 o << (abs(p[i]-mod)<p[i] ? p[i]-mod:p[i]);
 }
}

```

```

if(j){
 o << "*x";
 if(i>1) o << "^" << i;
 o << "+";
}
return o;
}
// inverse mod x^n
poly inv(int n)const{
 assert(size() && operator[](0));
 if((int)size()>n) return poly(vector<int>(begin(),
begin() + n)).inv(n);

 poly ret(1, NT<mod>::inv(operator[](0)));
 ret.reserve(2*n);
 for(int i=1;i<n;i*=2){
 poly l = substr(0, i) * ret; // l[0:i] will be 0
 poly r = substr(i, 2*i) * ret; // r[i:2*i] will be irrelevant
 poly up = (l.substr(i, 2*i) + r.substr(0, i)) * ret;
 ret.resize(2*i);
 for(int j=0;j<i;++j){
 ret[i+j] = NT<mod>::sub(0, up[j]);
 }
 }
 ret.resize(n);
 return ret.normalize();
}
pair<poly, poly> div(poly const&o, poly const& oinvrev)const{
 if(o.size()>size()) return {poly(1, 0), *this};
 int rsize = size()-o.size()+1;
 poly q = (reversed()*oinvrev.substr(0, rsize));
 q.resize(rsize);
 reverse(q.begin(), q.end());
 poly r = *this - q*(o);
 return make_pair(q, r.normalize());
}
pair<poly, poly> div(poly const&o)const{
 return div(o, o.reversed().inv(size()+2));
}
/// n-th term of linear recurrence in O(K log K log N)
signed codechef_rng(){
 const int mod = 104857601;
 int k;
 ll n;
 cin >> k >> n;
 --n;
 vector<int> a(k), c(k);
 for(auto &e:a) cin >> e;
 for(auto &e:c) cin >> e;
 poly<mod> p(k+1, 1);
 for(int i=0;i<k;++i){
 p[k-i-1] = (mod-c[i])%mod;;
 }
 poly<mod> b(1, 1), x(vector<int>({0, 1}));
 poly<mod> previnv = p.reversed().inv(p.size()+2);
 for(ll i=1;i<<61;i>>=1){

```

```

if(2*i<=n){
 b = (b*b).div(p, previnv).second;
}
if(n&i){
 b = (b*x).div(p, previnv).second; // could be optimized
}
b.resize(k, 0);
int res = 0;
for(int i=0;i<k;++i){
 NT<mod>::xadd(res, NT<mod>::mul(b[i], a[i]));
}
cout << res << "\n";
return 0;
}

signed main(){
 cin.tie(0); ios_base::sync_with_stdio(false);
 codechef_rng();
// const int mod = 1e9+7;
//
// int n, m;
// cin >> n;
// poly<mod> a(n);
// for(auto &e:a) cin >> e;
// cin >> m;
// poly<mod> b(m);
}

```

```

// for(auto &e:b) cin >> e;
// cout << "a = " << a << "\n";
// cout << "b = " << b << "\n";
// cout << "a+b = " << a+b << "\n";
// cout << "a-b = " << a-b << "\n";
// cout << "a*b = " << a*b << "\n";
// cout << "a/b = " << a.div(b).first << "\n";
// cout << "a%b = " << a.div(b).second << "\n";

 return 0;
}

```

## 119. FFT and NTT

### Standard FFT and NTT

```

namespace ntt
{
 struct num
 {
 double x,y;
 num() {x=y=0;}
 num(double x,double y):x(x),y(y){}
 };
 inline num operator+(num a,num b) {return num(a.x+b.x,a.y+b.y);}
 inline num operator-(num a,num b) {return num(a.x-b.x,a.y-b.y);}
 inline num operator*(num a,num b) {return num(a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x);}
 inline num conj(num a) {return num(a.x,-a.y);}
}

```

```

int base=1;
vector<num> roots={{0,0},{1,0}};
vector<int> rev={0,1};
const double PI=acos(-1.0);

void ensure_base(int nbase)
{
 if(nbase<=base) return;
 rev.resize(1<<nbase);
 for(int i=0;i<(1<<nbase);i++)
 rev[i]=(rev[i]>>1)>>1+((i&1)<<(nbase-1));
 roots.resize(1<<nbase);
 while(base<nbase)
 {
 double angle=2*PI/(1<<(base+1));
 for(int i=1<<(base-1);i<(1<<base);i++)
 {
 roots[i<<1]=roots[i];
 double angle_i=angle*(2*i+1-(1<<base));
 roots[(i<<1)+1]=num(cos(angle_i),sin(angle_i));
 }
 base++;
 }
}

void fft(vector<num> &a,int n=-1)
{
 if(n== -1) n=a.size();
}

```

```

assert((n&(n-1))==0);
int zeros=__builtin_ctz(n);
ensure_base(zeros);
int shift=base-zeros;
for(int i=0;i<n;i++)
 if(i<(rev[i]>>shift))
 swap(a[i],a[rev[i]>>shift]);
for(int k=1;k<n;k<<=1)
{
 for(int i=0;i<n;i+=2*k)
 {
 for(int j=0;j<k;j++)
 {
 num z=a[i+j+k]*roots[j+k];
 a[i+j+k]=a[i+j]-z;
 a[i+j]=a[i+j]+z;
 }
 }
}
vector<num> fa,fb;

vector<int> multiply(vector<int> &a, vector<int> &b)
{
 int need=a.size()+b.size()-1;
 int nbase=0;
 while((1<<nbase)<need) nbase++;
 ensure_base(nbase);
}

```

```

int sz=1<<nbase;
if(sz>(int)fa.size()) fa.resize(sz);
for(int i=0;i<sz;i++)
{
 int x=(i<(int)a.size())?a[i]:0;
 int y=(i<(int)b.size())?b[i]:0;
 fa[i]=num(x,y);
}
fft(fa,sz);
num r(0,-0.25/sz);
for(int i=0;i<=(sz>>1);i++)
{
 int j=(sz-i)&(sz-1);
 num z=(fa[j]*fa[j]-conj(fa[i]*fa[i]))*r;
 if(i!=j) fa[j]=(fa[i]*fa[i]-conj(fa[j]*fa[j]))*r;
 fa[i]=z;
}
fft(fa,sz);
vector<int> res(need);
for(int i=0;i<need;i++) res[i]=fa[i].x+0.5;
return res;
}

vector<int> multiply(vector<int> &a,vector<int> &b,int m,int eq=0)
{
 int need=a.size()+b.size()-1;
 int nbase=0;
 while((1<<nbase)<need) nbase++;
 ensure_base(nbase);
}

```

```

int sz=1<<nbase;
if(sz>(int)fa.size()) fa.resize(sz);
for(int i=0;i<(int)a.size();i++)
{
 int x=(a[i]%m+m)%m;
 fa[i]=num(x&((1<<15)-1),x>>15);
}
fill(fa.begin()+a.size(),fa.begin()+sz,num{0,0});
fft(fa,sz);
if(sz>(int)fb.size()) fb.resize(sz);
if(eq) copy(fa.begin(),fa.begin()+sz,fb.begin());
else
{
 for(int i=0;i<(int)b.size();i++)
 {
 int x=(b[i]%m+m)%m;
 fb[i]=num(x&((1<<15)-1),x>>15);
 }
 fill(fb.begin()+b.size(),fb.begin()+sz,num{0,0});
 fft(fb,sz);
}
double ratio=0.25/sz;
num r2(0,-1),r3(ratio,0),r4(0,-ratio),r5(0,1);
for(int i=0;i<=(sz>>1);i++)
{
 int j=(sz-i)&(sz-1);
 num a1=(fa[i]+conj(fa[j]));
 num a2=(fa[i]-conj(fa[j]))*r2;
 num b1=(fb[i]+conj(fb[j]))*r3;
 num b2=(fb[i]-conj(fb[j]))*r4;
 num c1=(a1*a2+b1*b2)*r5;
 num c2=(a1*a2-b1*b2)*r3;
 num c3=(a1+b1)*(a2+b2)*r4;
 num c4=(a1+b1)*(a2-b2)*r5;
 fa[i]=c1+c2*c3*c4;
 fb[i]=c2*c3*c4;
}

```

```

num b2=(fb[i]-conj(fb[j]))*r4;
if(i!=j)
{
 num c1=(fa[j]+conj(fa[i]));
 num c2=(fa[j]-conj(fa[i]))*r2;
 num d1=(fb[j]+conj(fb[i]))*r3;
 num d2=(fb[j]-conj(fb[i]))*r4;
 fa[i]=c1*d1+c2*d2*r5;
 fb[i]=c1*d2+c2*d1;
}
fa[j]=a1*b1+a2*b2*r5;
fb[j]=a1*b2+a2*b1;
}
fft(fa,sz);fft(fb,sz);
vector<int> res(need);
for(int i=0;i<need;i++)
{
 || aa=fa[i].x+0.5;
 || bb=fb[i].x+0.5;
 || cc=fa[i].y+0.5;
 res[i]=(aa+((bb%m)<<15)+((cc%m)<<30))%m;
}
return res;
}
vector<int> square(vector<int> &a,int m)
{
 return multiply(a,a,m,1);
}
};

```

```

using namespace ntt;

int mod;
vi pow(vi& a,int p)
{
 vi res;
 res.eb(1);
 while(p){
 if(p&1) res=multiply(res,a,mod);
 a=square(a,mod);
 p>>=1;
 }
 return res;
}
int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 mod=998244353;
 cin>>n>>k;
 vi a(10,0);
 while(k--){
 cin>>m;
 a[m]=1;
 }
 vi ans=pow(a,n/2);
 int res=0;
 for(auto x:ans){
 res=(res+1LL*x*x%mod)%mod;
 }
}
```

```

 }
 cout<<res<<nl;
 return 0;
}

```

## Notes

- **String Matching 2D**

/\*\*

You are given two grids s of size ( $n \times m$ ) and t of size ( $r \times c$ ). Each each grid contains character 'G' and 'L'. Now, find the upper-left most position of the grid s on which if you place the (0, 0) of grid t, maximum amount of characters match. Output such position and number of characters match for each letter in the alphabet. Grid t should not cross boundary of s.

Solution: Flatten both the grids, using dummy character for flattening t.

\*\*/

```

void solve()
{
 string sflat="", tflat="";
 for(int i=0;i<n;i++) for(int j=0;j<m;j++) sflat+=s[i][j];
 for(int i=0;i<n;i++)
 {
 for(int j=0;j<m;j++)
 {
 if(i>=r || j>=c)
 tflat+='x'; //Important
 }
 }
}

```

```

 else
 tflat+=t[i][j];
 }
}
reverse(sflat);
int sz=n*m;
vi ga(sz), la(sz), gb(sz), lb(sz);
convert(sflat,ga,'G');//convert to binary polynomial
convert(sflat,la,'L');
convert(tflat,gb,'G');
convert(tflat,lb,'L');
vi gout=mult(ga,gb);
vi lout=mult(la,lb);
int j=0, k=0, mxval=0, sx=0, sy=0;
int grain=0, lives=0;
for(int i=sz-1;i>=0;i--)
{
 if(j+r-1<n && k+c-1<m)
 {
 if(gout[i]+lout[i]>mxval)
 {
 mxval=gout[i]+lout[i];
 sx=j, sy=k;
 grain=gout[i];
 lives=lout[i];
 }
 }
 k++;
 if(k==m)

```

```

{
 j++;
 k=0;
}
printf("Case #%d: %d %d %d %d\n", cases++, sx+1, sy+1,
grain, lives);//'G' and 'L'
}

● Three Sum
/*
You're given a sequence s of N distinct integers. Consider all
the possible sums of three integers from the sequence
at three different indices. For each obtainable sum output the
number of different triples of indices that generate
it.
*/
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 scanf("%d", &n);
 vector<ll> a(N), b(N), c(N);
 for(int i=0;i<n;i++)
 {
 scanf("%d", &in[i]);
 in[i]+=offset; // make all integers non-negative
 a[in[i]]++;
 b[in[i]*2]++;
 }
 printf("Case #%d: %d %d %d %d\n", cases++, sx+1, sy+1,
grain, lives);//'G' and 'L'
}

```

```

c[in[i]*3]++;
}
/// From all possible ways, subtract ways where 2 integers
are on same index
/// and one is on different. This can be done in 3C2 ways
because we are multiplying
/// three polynomials. While counting this subtraction, we
subtracted all three indices
/// same case thrice, but we have to subtract once. So we
add 2*c[i] where c[i] is the
/// case where all three are same. Finally, divide by 6
because each different combination
/// three indices occurs 3! times.
vector<ll> cube=mult(a,a);
cube=mult(cube,a);
b=mult(b,a);
for(int i=0;i<N;i++) cube[i]=cube[i]-3LL*b[i];
for(int i=0;i<N;i++)
{
 cube[i]=(cube[i]+2*c[i])/6;
 if(cube[i])
 printf("%d : %lld\n", i-offset*3, cube[i]);
}
return 0;
}

● Hamming Distance Queries
/*
You are given two bitstrings and queries. In each query, you
are given two starting position and a length. You


```

have to output the hamming distance between the two strings starting from the given two positions with given length.

Solution:

Suppose we are given two strings S and T. We want to find all hamming distances between S and T if we place T in all positions of S without crossing the boundary.

This can be solved easily by FFT.

```
*/
vi hamming(string &s, string &t)
{
 vi a, b;
 int n=s.size(), m=t.size();
 for(int i=0;i<n;i++)
 {
 if(s[i]=='1')
 a.pb(1);
 else
 a.pb(-1);
 }
 for(int i=m-1;i>=0;i--)
 {
 if(t[i]=='1')
 b.pb(1);
 else
 b.pb(-1);
 }
 vi c=mult(a,b), ret;
```

```
for(int i=0; (i+m)<=n; i++)
 ret.pb(m-(c[i+m-1]+m)/2);
return ret;
}
```

Now for the original problem, we do

square-root decomposition. We take blocks from string T and multiply each

block with S as mentioned above. Now while answering queries, we take each block of S that falls inside the current query and add the value of the corresponding position from the block's multiplication values. For the rest of the positions that do

not fall inside this box, we can do linear checking. But depending on the constraints, we may need to apply more block-based operation to do this linear checking more efficiently. Here is the solution:

```
*/
///nsqrt(n)logn
void solve()
{
 int m=t.size(), last=0;
 int blockcnt=(m-1)/blocksz;
 for(int i=0; i<m; i+=blocksz)
 {
 /// find the hamming distance for each block with string s
 blocks[i/blocksz]=hamming(i,min(m-1,i+blocksz-1)); //range of t
 last=min(i+blocksz-1,m-1);
```

```

/// prnt(last);
}
if(last!=m-1)
 blocks[blockcnt]=hamming(last,m-1);
/// Taken chunks of BLOCK from each position to answer
/// the linear check efficiently
for(int i=0; i+BLOCK-1<s.size(); i++)
{
 for(int j=i; j<=i+BLOCK-1; j++)
 {
 if(s[j]=='1')
 chunks[i].set(BLOCK-1-j+i); // bitsets
 }
}
for(int i=0; i+BLOCK-1<t.size(); i++)
{
 for(int j=i; j<=i+BLOCK-1; j++)
 {
 if(t[j]=='1')
 chunkt[i].set(BLOCK-1-j+i);
 }
}
int handleBlocks(int sx, int sy, int tx, int l, int r)
{
 int ret=0;
 int pos=sx+l*blocksz-tx;
 for(int i=l; i<=r; i++)
 {
 if(pos+blocksz-1>sy)
 break;
 ret+=blocks[i][pos];
 pos+=blocksz;
 }
 return ret;
}
int handle(int sx, int sy, int tx, int ty)
{
 int ret=0, i, j;
 for(i=sx, j=tx; ; i+=BLOCK, j+=BLOCK)
 {
 if(i+BLOCK-1>sy)
 break;
 bitset<BLOCK> temp=chunks[i]^chunkt[j];
 ret+=temp.count();
 }
 if(i<=sy)
 {
 for(int k=i, l=j; k<=sy, l<=ty; k++, l++)
 ret+=(s[k]!=t[l]);
 }
 return ret;
}
/// (sx,sy) - string s position, (tx,ty) - string t positions
/// (sy-sx+1)==(ty-tx+1)
int findAns(int sx, int sy, int tx, int ty)
{

```

```

int ret=0;
int l=tx/blocksz+1;
int r=ty/blocksz-1;
if(l+1<r)
{
 int pos1=sx+l*blocksz-tx;
 int pos2=sy-(ty-(r+1)*blocksz);
 /// handle the left part out of the blocks
 ret=handle(sx,pos1-1,tx,l*blocksz-1);
 /// handle the right part
 ret+=handle(pos2,sy,(r+1)*blocksz,ty);
 /// handle the blocks
 ret+=handleBlocks(sx,sy,tx,l,r);
}
else
{
 ret=handle(sx,sy,tx,ty);
}
return ret;
}

```

- **Number of Triangles**

```

/***
Given at most 10^5
sticks. Pick 3 sticks from there. What is the probability that it
forms a triangle? The length
of the sticks can be same, but indices will be different.
Solution: First, find the number of occurrences of different
summations of pair of numbers taken from the
input. In this case, we will only consider different indices

```

and distinct pairs, i.e (1,2) and (2,1) will be counted twice.

```

*/
int main()
{
 int n;
 scanf("%d", &n);
 int sz = 0;
 for(int i=0;i<n;i++)
 {
 scanf("%d", &a[i]);
 p[a[i]]++;
 sz = max(sz, a[i]);
 }
 sz++;
 vector<int>out=multiply(p,p);
 int len=sz(out);
 for(int i=0;i<n;i++)
 {
 out[a[i] + a[i]]--;
 }
 for(int i=0;i<len;i++) out[i] /= 2;
 for(int i=1;i<len;i++) sum[i] = sum[i - 1] + out[i];
 sort(a, a + n);
 ll ans = 0;
 for(int i=0;i<n;i++)
 {
 //let a[i] is the largest side of the triangle so add

```

```

//the number of ways we can take other two sides where
their summation
//is greater than a[i].
ans += sum[len - 1] - sum[a[i]];
//subtract cases where one side was less a[i], other was
greater, as a[i] largest
ans -= (ll)(n - i - 1) * i;
//subtract cases where two sides are a[i], and other is
some other
ans -= (n - 1);
//one is a[i], other two are larger than a[i]
ans -= ((ll)(n - i - 1) * (n - i - 2)) / 2;
}
cout<<ans<<nl;
return 0;
}

```

## 120. Online FFT

/\*\*

Given G[1...n] and F[0], find every

$F[n] = \sum_{i=1}^{n-1} F[i] * G[n-i]$

So, value of  $F[n]$  depends on previous  $n-1$  values. How can we compute it efficiently?

The idea is to divide the  $G[1...N]$  into blocks.

First a block of size 1, then 1, then 2, then 4, then 8 etc. So we divide  $G[1...N]$  in blocks of power of 2 where first two blocks are of size 1.

Now, we do following for each  $F[i]$ :

\* When we get an  $F[i]$ , first we convolve it with first two blocks. And add those to the appropriate entries.

\* Now suppose  $i$  is non-zero multiple of  $2^k$  for some  $k$ . Then we convolve  $F[i-2^k]$  to  $F[i-1]$  with the block in  $G$  of the same size for each  $k$ .

\*\*/

/\*\*

Following problem:

Given  $A[0...n-1]$

$f[n] = \sum_{i=0}^{n-1} A[n-1-i] * f[i]$

\*\*/

///O( $n \log^2 n$ )

int LM; //size of the array f

int f[N], A[N];

void convolve(int l1, int r1, int l2, int r2)

{

int n=max(r1-l1+1, r2-l2+1);

int t=1;

while(t<n) t<=1;

n=t;

vector<int> a(n), b(n);

for(int i=l1; i<=r1; i++) a[i-l1]=f[i];

for(int i=l2; i<=r2; i++) b[i-l2]=A[i];

vector<int> ret=multiply(a,b,mod);

for(int i=0; i<ret.size(); i++)

{

int idx=i+l1+l2+1;

```

if(idx>LM) break;
// adding to the appropriate entry
f[idx]+=ret[i];
f[idx]%=mod;
}
}
void solve()
{
 f[0]=1; // base case, change it if you want
 for(int i=0; i<=LM; i++)
 {
 // Doing the part 1
 f[i+1]=(f[i+1]+1LL*f[i]*A[0]%mod)%mod;
 f[i+2]=(f[i+2]+1LL*f[i]*A[1]%mod)%mod;
 if(!i) continue;
 // part 2
 int limit=(i& -i);
 for(int p=2; p<=limit; p*=2)
 {
 convolve(i-p,i-1,p,min(2*p-1,LM));
 }
 }
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
}

```

```

cin>>n;
for(i=0;i<n;i++) cin>>A[i];
LM=n;
solve();
for(i=0;i<=n;i++) cout<<f[i]<<' ';
cout<<nl;
return 0;
}

```

## 121. Fast Walsh Hadamard Transformation

```

#define MX (1 << 16)
#define OR 0
#define AND 1
#define XOR 2
int in=qpow(2,mod-2);
/// Fast Walsh-Hadamard Transformation in n log n
/// Beware!!! after the convolution the arrays will not be the same
again
/// array sizes must be same and powers of 2
struct fwht{
 int P1[MX], P2[MX];

 void walsh_transform(int* ar, int n, int flag = AND){
 if (n == 0) return;

 int i, m = n/2;
 walsh_transform(ar, m, flag);
 walsh_transform(ar+m, m, flag);
 }
}

```

```

for (i = 0; i < m; i++){ // Don't forget to remove modulo if not
required
 int x = ar[i], y = ar[i + m];
 if (flag == OR) ar[i] = x, ar[i + m] = (x + y)%mod;
 if (flag == AND) ar[i] = (x + y)%mod, ar[i + m] = y;
 if (flag == XOR) ar[i] = (x + y)%mod, ar[i + m] = (x -
y+mod)%mod;
}
}

void inverse_walsh_transform(int* ar, int n, int flag = AND){
if (n == 0) return;

int i, m = n/2;
inverse_walsh_transform(ar, m, flag);
inverse_walsh_transform(ar+m, m, flag);

for (i = 0; i < m; i++){ // Don't forget to remove modulo if not
required
 int x = ar[i], y = ar[i + m];
 if (flag == OR) ar[i] = x, ar[i + m] = (y - x+mod)%mod;
 if (flag == AND) ar[i] = (x - y+mod)%mod, ar[i + m] = y;
 if (flag == XOR) ar[i] = 1LL*(x + y)%mod *in%mod, ar[i + m] =
1LL*(x - y+mod)%mod *in%mod; /// replace modular inverse(in) by
>>1 if not required
}
}

```

```

/// For i = 0 to n - 1, j = 0 to n - 1
/// v[i flag j] += A[i] * B[j]
vi convolution(int n, int* A, int* B, int flag = AND){
 assert(__builtin_popcount(n) == 1); // n must be a power of 2
 for (int i = 0; i < n; i++) P1[i] = A[i];
 for (int i = 0; i < n; i++) P2[i] = B[i];

 walsh_transform(P1, n, flag);
 walsh_transform(P2, n, flag);
 for (int i = 0; i < n; i++) P1[i] = 1LL*P1[i] * P2[i]%mod;
 inverse_walsh_transform(P1, n, flag);
 return vi(P1,P1+n);
}

///compute A^k where A*A=A convolution A
vi pow(int n,int* A,ll k,int flag=AND)
{
 walsh_transform(A,n,flag);
 for(int i=0;i<n;i++) A[i]=qpow(A[i],k);
 inverse_walsh_transform(A,n,flag);
 return vi(A,A+n);
}

int a[MX];
int32_t main()
{
 BeatMeScnf;
 int i,j,k,n,m;
 cin>>n>>k;
}

```

```

for(i=0;i<=k;i++) a[i]=1;
vi v=t.pow(MX,a,n,XOR);
//Beware!!! the array a will not be the same again
int ans=0;
for(i=1;i<MX;i++) ans=(ans+v[i])%mod;
cout<<ans<<nl;
return 0;
}

```

## 122. Freivalds Algorihm

```

struct matrix
{
 int n,m;
 vector<vector<int> > t;

 matrix() { }
 matrix(int _n,int _m, int val) {n = _n;m=_m; t.assign(n,
vector<int>(m, val));}
 matrix(int _n,int _m) {n=_n;m=_m;t.assign(n, vector<int>(m,
0));}
 matrix(vector<vi> v){n=v.size();m=n?v[0].size():0;t=v;}

 //remove mod if not needed
 matrix operator * (matrix a)
 {
 assert(m==a.n);
 matrix ans = matrix(n, a.m);
 for(int i = 0; i < n; i++)
 for(int k = 0; k < m; k++)

```

```

 for(int j = 0; j < a.m; j++)
 ans.t[i][j] = ans.t[i][j] + t[i][k] *
a.t[k][j];
 }

 matrix operator + (matrix a)
 {
 assert(MP(n,m)==MP(a.n,a.m));
 matrix ans=matrix(n,m);
 for(int i=0;i<n;i++)
 for(int j=0;j<m;j++)
 ans.t[i][j]=t[i][j]+a.t[i][j];
 return ans;
 }

 matrix operator - (matrix a)
 {
 assert(MP(n,m)==MP(a.n,a.m));
 matrix ans=matrix(n,m);
 for(int i=0;i<n;i++)
 for(int j=0;j<m;j++)
 ans.t[i][j]=t[i][j]-a.t[i][j];
 return ans;
 }

 bool operator == (const matrix& a)
 {
 return t==a.t;
 }
}

```

```

bool operator !=(const matrix& a)
{
 return t!=a.t;
}
void print()
{
 for(int i=0;i<n;i++)
 for(int j=0;j<m;j++)
 cout<<t[i][j]<<"\n"[j==m-1];
}
};

///check if two n*n matrix a*b=c within complexity (iteration*n^2)
///probability of error 2^(-iteration)
int check(matrix a,matrix b,matrix c)
{
 int n=a.n;
 int iteration=40;
 matrix zero(n,1),r(n,1);
 while(iteration--){
 for(int i=0;i<n;i++) r.t[i][0]=rand()%2;
 matrix ans=(a*(b*r))-(c*r);
 if(ans!=zero) return 0;
 }
 return 1;
}
int main()
{
 BeatMeScanf;
}

```

```

int i,j,k,n,m,q;
cin>>n>>q;
matrix a(n,n),c(n,n);
for(i=0;i<n;i++){
 for(j=0;j<n;j++) cin>>a.t[i][j];
}
while(q--){
 int ty;
 cin>>ty;
 if(ty==1){
 int x1,y1,x2,y2,v;
 cin>>x1>>y1>>x2>>y2>>v;
 for(i=x1;i<=x2;i++) for(j=y1;j<=y2;j++) a.t[i][j]+=v;
 }
 else{
 for(i=0;i<n;i++) for(j=0;j<n;j++) cin>>c.t[i][j];
 cout<<(check(a,a,c)?"yes\n":"no\n");
 }
}
return 0;
}

```

## GAME THEORY

### 123. Hackenbush

```

// Green Hackenbush
//
// Description:

```

```

// Consider a two player game on a graph with a specified vertex
// (root).
// In each turn, a player eliminates one edge.
// Then, if a subgraph that is disconnected from the root, it is
// removed.
// If a player cannot select an edge (i.e., the graph is singleton),
// he will lose.
//
// Compute the Grundy number of the given graph.
//
// Algorithm:
// We use two principles:
// 1. Colon Principle: Grundy number of a tree is the xor of
// Grundy number of child subtrees.
// (Proof: easy).
//
// 2. Fusion Principle: Consider a pair of adjacent vertices u, v
// that has another path (i.e., they are in a cycle). Then,
// we can contract u and v without changing Grundy number.
// (Proof: difficult)
//
// We first decompose graph into two-edge connected components.
// Then, by contracting each components by using Fusion Principle,
// we obtain a tree (and many self loops) that has the same Grundy
// number to the original graph. By using Colon Principle, we can
// compute the Grundy number.
//
// Complexity:
// O(m + n).

```

```

#include <iostream>
#include <vector>
#include <cstdio>
#include <algorithm>
#include <functional>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define TEST(s) if (!(s)) { cout << __LINE__ << " " << #s << endl; exit(-1); }

struct hackenbush {
 int n;
 vector<vector<int>> adj;

 hackenbush(int n) : n(n), adj(n) { }
 void add_edge(int u, int v) {
 adj[u].push_back(v);
 if (u != v) adj[v].push_back(u);
 }
};

// r is the only root connecting to the ground
int grundy(int r) {
 vector<int> num(n), low(n);
 int t = 0;
 function<int(int,int)> dfs = [&](int p, int u) {

```

```

num[u] = low[u] = ++t;
int ans = 0;
for (int v: adj[u]) {
 if (v == p) { p += 2*n; continue; }
 if (num[v] == 0) {
 int res = dfs(u, v);
 low[u] = min(low[u], low[v]);
 if (low[v] > num[u]) ans ^= (1 + res) ^ 1; // bridge
 else ans ^= res; // non bridge
 } else low[u] = min(low[u], num[v]);
}
if (p > n) p -= 2*n;
for (int v: adj[u])
 if (v != p && num[u] <= num[v]) ans ^= 1;
return ans;
};
return dfs(-1, r);
}
};

int main() {
 int cases; scanf("%d", &cases);
 for (int icase = 0; icase < cases; ++icase) {
 int n; scanf("%d", &n);
 vector<int> ground(n);
 int r;
 for (int i = 0; i < n; ++i) {
 scanf("%d", &ground[i]);
 if (ground[i] == 1) r = i;
 }
 }
}

```

```

}

int ans = 0;
hackenbush g(n);
for (int i = 0; i < n-1; ++i) {
 int u, v;
 scanf("%d %d", &u, &v);
 --u; --v;
 if (ground[u]) u = r;
 if (ground[v]) v = r;
 if (u == v) ans ^= 1;
 else g.add_edge(u, v);
}
int res = ans ^ g.grundy(r);
printf("%d\n", res != 0);
}
}

```

## MISCELLANEOUS

### 124. Permutation and Inversion

- sum of number of inversions for all permutations of size  $k$  is equal to  $sumAll[k] = k! \cdot \frac{k(k-1)}{4}$ .
- 

### 125. Minimum K Length Inverses To Make all 0

```

///Given a binary array
///in one operation you can xor any k length subarray with 1
///find minimum operations to make all the array having only 0s

```

```

int a[N],b[N];
int main()
{
 BeatMeScanf;
 int i,j,k,n,m,sum=0,ans=0;
 cin>>n>>k;
 for(i=1;i<=n;i++) cin>>a[i];
 for(i=1;i+k-1<=n;i++){
 if((sum+a[i])&1) b[i]=1,ans++;
 sum+=b[i];
 if(i-k+1>=1) sum-=b[i-k+1];
 }
 for(i=n-k+2;i<=n;i++){
 if((sum+a[i])&1) ans=-1;///not possible
 if(i-k+1>=1) sum-=b[i-k+1];
 }
 cout<<ans<<nl;
 return 0;
}

```

## 126. Divide and Conquer on Queries

```

///maximum subset xor in range
///(n+q) logn^2
struct st{
 vi basis;
 int add(int x)
 {
 for(auto &it:basis) if((x^it)<x) x^=it;
 for(auto &it:basis) if((it^x)<it) it^=x;
 }
};

```

```

if(x) basis.eb(x);
}
int get()
{
 int ans=0;
 for(auto x:basis) ans^=x;
 return ans;
}
void clear()
{
 basis.clear();
}
int a[N];
int input[N][2]; /// inputs queries
st dp_left[N], dp_right[N];
int ans[N];
void solve(int L, int R, vi all)
{
 if(L>R || all.empty()) return;
 /// initialize only this range
 for(int i=L;i<=R;i++) dp_left[i].clear(),dp_right[i].clear();

 int mid = (L+R)/2;

 ///get answer for [i....mid-1]
 for(int i=mid-1; i>=L; i--)
 {
 if(i+1<mid) dp_left[i]=dp_left[i+1];
 }
}

```

```

 dp_left[i].add(a[i]);
 }
 //get answer for [mid....i]
 for(int i=mid; i<=R; i++)
 {
 if(i-1>=mid) dp_right[i]=dp_right[i-1];
 dp_right[i].add(a[i]);
 }

 vi ls, rs;
 for(auto idx: all)
 {
 int l = input[idx][0], r = input[idx][1];

 if(l>mid) rs.pb(idx);
 else if(r<mid) ls.pb(idx);
 else
 {
 if(l==r && l==mid) /// query is just on mid,
specially handled
 {
 ans[idx] = a[mid];
 }
 else if(l==mid) /// starts from mid
 {
 ans[idx] = dp_right[r].get();
 }
 else if(r==mid) /// ends in mid
 {

```

```

 st nw=dp_left[l];
 nw.add(a[mid]);///add mid element
 ans[idx]=nw.get();
 }
 else
 {
 /// merge both sides and calculate
 answer for current query

 if(dp_right[r].basis.size()>dp_left[l].basis.size())
 st nw=dp_right[r];
 for(auto x:dp_left[l].basis) nw.add(x);
 ans[idx]=nw.get();
 }
 else{
 st nw=dp_left[l];
 for(auto x:dp_right[r].basis) nw.add(x);
 ans[idx]=nw.get();
 }
 }
 }
 /// find answer for other queries by divide and conquer
 solve(L,mid,ls);
 solve(mid+1,R,rs);
 }
 int main()
 {
 BeatMeScnf;

```

```

cin.tie(NULL);
int i,j,k,n,m,l,r,q;
cin>>n;
for(i=1;i<=n;i++) cin>>a[i];
cin>>q;
vi v;
for(i=1;i<=q;i++){
 cin>>input[i][0];
 cin>>input[i][1];
 v.eb(i);
}
solve(1,n,v);
for(i=1;i<=q;i++) cout<<ans[i]<<nl;
return 0;
}

```

## 127. Longest Common Subsequence

"Well, you can decrease it to a LIS problem. Lets say you have strings S1 and S2. You can take S1 and for every character put in a list for that character the indexes where the character occurs. These lists should be sorted in decreasing order. For example if you have string "abacba" you have these lists

```

'a': 5,2,0
'b': 1,4
'c': 3

```

Now, if you look at S2 and for each character put in a sequence its list(note that it can be empty), you have a sequence in which you

should search for LIS. Here's with the example if S2 is 'baaxac' you have sequence 4,1, 5,2,0, 5,2,0, ,5,2,0, 3. The LIS of this sequence is 3 as the answer would be. If you have to retrieve some string as an answer you can just retrieve the corresponding letters. Now that we have a LIS problem, it can be solved in  $n\log n$ . The only problem is that the sequence can get big, but it is in rare situations, so you have an average case of  $n\log n$  and a worst case of  $O(NM)$  or something like that, but it is still a good idea, which is worth the thoughts, especially if you have some restrictions about the type of the input"

## 128. Fraction Structure

```

struct Fraction {
 ll n, d;
 Fraction() { n = 0, d = 1; }
 Fraction(ll _n, ll _d) : n(_n), d(_d) { }
 Fraction operator+(const Fraction& p) const{
 ll g = gcd(d, p.d);
 ll td = d * (p.d / g);
 ll tn = n * (td/d) + p.n * (td/p.d);
 g = gcd(tn, td);
 tn /= g, td /= g;
 return Fraction(tn, td);
 }
 Fraction operator-(const Fraction& p) const{
 ll g = gcd(d, p.d);
 ll td = d * (p.d / g);
 ll tn = n * (td/d) - p.n * (td/p.d);
 g = gcd(tn, td);
 tn /= g, td /= g;
 }
}

```

```

 return Fraction(tn, td);
 }

Fraction operator*(const Fraction& p) const{
 ll g1 = gcd(n, p.d), g2 = gcd(p.n, d);
 ll tn = (n/g1) * (p.n/g2);
 ll td = (d/g2) * (p.d/g1);
 ll g = gcd(tn, td);
 tn /= g, td /= g;
 return Fraction(tn, td);
}

Fraction operator/(const Fraction& p) const{
 ll g1 = gcd(n, p.n), g2 = (d, p.d);
 ll tn = (n/g1) * (p.d/g2);
 ll td = (d/g2) * (p.n/g1);
 ll g = gcd(tn, td);
 tn /= g, td /= g;
 return Fraction(tn, td);
}

bool operator==(const Fraction& p) const{
 return n == p.n and d == p.d;
}

bool operator!=(const Fraction& p) const{
 return !(*this == p);
}

bool operator<(const Fraction& b) const
{
 if(n == b.n)
 return d < b.d;
 return n < b.n;
}

```

```

 }

ostream& operator<<(ostream &out, Fraction p) {
 out << p.n << "/" << p.d;
}

int main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 Fraction p=Fraction(2,3),q=Fraction(3,4),ans;
 ans=p+q;
 cout<<ans<<nl;
 return 0;
}

```

## 129. Berlekamp-Messey Algorithm

```

///Description: n-order recurrence has following recurrence a[i]=f(i-1)*a[i-1]+f(i-2)*a[i-2]+...f(i-n)*a[i-n]
///this algo Recovers any n-order linear recurrence relation from the first
///2n terms of the recurrence.
///Useful for guessing linear recurrences after brute-forcing the first terms.
///Should work on any field, but numerical stability for floats is not guaranteed.
///Output will have size <=2*n.

```

```

vector<ll> BerlekampMassey(vector<ll> s) {
 int n = sz(s), L = 0, m = 0;
 vector<ll> C(n), B(n), T;
 C[0] = B[0] = 1;

 ll b = 1;
 for(int i=0;i<n;i++) {
 ++m;
 ll d = s[i] % mod;
 for(int j=1;j<=L;j++) d = (d + C[j] * s[i - j]) % mod;
 if (!d) continue;
 T = C; ll coef = d * qpow(b, mod-2) % mod;
 for(int j=m;j<n;j++) C[j] = (C[j] - coef * B[j - m]) % mod;
 if (2 * L > i) continue;
 L = i + 1 - L; B = T; b = d; m = 0;
 }

 C.resize(L + 1); C.erase(C.begin());
 for(auto& x:C) x = (mod - x) % mod;
 return C;
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 vll v({1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6});
 vll ans=BerlekampMassey(v);
 for(auto x:ans) cout<<x<<' ';
 return 0;
}

```

```
}
```

## 130. Ternary Search

/\*It's impossible to run ternary search on integer if we allow  $f(i)=f(i+1)$ \*/

### On Double

```

ll a[mxn],n;
ld b[mxn];
ld yo(ld x)
{
 ld mx=-1e9,mn=1e9,summn=0,summx=0;
 for(ll i=1;i<=n;i++){
 summn+=a[i]-x;
 if(summn>0) summn=0;
 mn=min(mn,summn);
 summx+=a[i]-x;
 if(summx<0) summx=0;
 mx=max(mx,summx);
 }
 return fmax(fabs(mx),fabs(mn));
}

```

```

int main()
{
 fast;
 ll i,j,k,m,t=200;
 cin>>n;
 for(i=1;i<=n;i++) cin>>a[i];
 ld l=-1e9,r=(ld)1e9;
 ld ans1,ans2;
}

```

```

while(t--){
 ld mid1=l+(r-l)/3.0;
 ld mid2=r-(r-l)/3.0;
 ans1=yo(mid1);
 ans2=yo(mid2);
 if(ans1<ans2) r=mid2;
 else l=mid1;
}
cout<<dout(10)<<ans1<<n;
return 0;
}

```

### On Integer

///f[x] increases and then decreases

```

int lo = -1, hi = n;
while (hi - lo > 1){
 int mid = (hi + lo)>>1;
 if (f(mid) > f(mid + 1))
 hi = mid;
 else
 lo = mid;
}
//lo + 1 is the answer

```

## 131. Histogram

///Find the largest rectangular area possible in a given histogram  
 ///where the largest rectangle can be made of a number of  
 ///contiguous bars. For simplicity, assume that all bars have same  
 ///width and the width is 1 unit.

```
int a[N];
```

```

|| histogram(int hist[],int n)
{
 stack < int > st;
 || ans = 0, i = 1;
 while(i <= n) {
 if(st.empty() || hist[st.top()] <= hist[i]) {
 st.push(i);
 i++;
 continue;
 }
 int top = st.top();
 st.pop();
 || area = (||)(hist[top]*((st.empty())? (i-1) : (i - st.top() - 1)));
 ans = max(ans,area);
 }
 while(!st.empty()) {
 int top = st.top(); st.pop();
 || area = hist[top]*((st.empty())? (i - 1) : (i - st.top() - 1));
 ans = max(ans,area);
 }
 return ans;
}

int main()
{
 fast;
 || i,j,k,n,m;
 cin>>n;
 for(i=1;i<=n;i++) cin>>a[i];
}
```

```

cout<<histogram(a,n)<<n;
return 0;
}

```

## 132. Linear Programming

```
/**
```

You have a weighted tree, consisting of n vertices.

Each vertex is either painted black or is painted red.

A red and black tree is called beautiful, if for any its vertex we can find a black vertex at distance at most x. You have a red and black tree. Your task is to make it beautiful in the minimum number of color swap operations.

In one color swap operation, you can choose two vertices of different colors and paint

each of them the other color. In other words, if you choose a red vertex p and a black

vertex q, then in one operation you are allowed to paint p black and paint q red.

Print the minimum number of required actions.

```
**/
```

```

ll n,k,a[N],dis[N][N];
vector<ll> g[N],cost[N];
bool ok[N][N];

```

```

void dfs(ll u,ll p,ll d,ll cur)
{

```

```

 dis[cur][u] = d;
 ll len = sz(g[u]), i;
 for(int i=0;i<len;i++)

```

```

 {
 ll v = g[u][i];
 if(v == p) continue;
 dfs(v,u,d+cost[u][i],cur);
 }
}
```

```
#define MAXC 1010
```

```
#define MAXV 1010
```

```
#define EPS 1e-13
```

```
#define MINIMIZE -1
```

```
#define MAXIMIZE +1
```

```
#define LESSEQ -1
```

```
#define EQUAL 0
```

```
#define GREATEQ 1
```

```
#define INFEASIBLE -1
```

```
#define UNBOUNDED 666
```

```
/***
```

1. Simplex Algorithm for Linear Programming

2. Maximize or minimize  $f_0*x_0 + f_1*x_1 + f_2*x_2 + \dots + f_{n-1}*x_{n-1}$  subject to some constraints

3. Constraints are of the form,  $c_0x_0 + c_1x_1 + c_2x_2 + \dots + c_{n-1}x_{n-1} (\leq \text{ or } \geq \text{ or } =) \text{ lim}$

4. m is the number of constraints indexed from 1 to m, and n is the number of variables indexed from 0 to n-1

5. ar[0] contains the objective function f, and ar[1] to ar[m] contains the constraints,  $\text{ar}[i][n] = \text{lim}_i$

6. It is assumed that all variables satisfies non-negativity constraint, i.e,  $x_i \geq 0$

7. If non-negativity constraint is not desired for a variable  $x$ , replace each occurrence

by difference of two new variables  $r_1$  and  $r_2$  (where  $r_1 \geq 0$  and  $r_2 \geq 0$ , handled automatically by simplex).

That is, replace every  $x$  by  $r_1 - r_2$  (Number of variables increases by one,  $-x, +r_1, +r_2$ )

8. `solution_flag = INFEASIBLE` if no solution is possible and `UNBOUNDED` if no finite solution is possible

9. Returns the maximum/minimum value of the linear equation satisfying all constraints otherwise

10. After successful completion, `val[]` contains the values of  $x_0, x_1 \dots, x_n$  for the optimal value returned

\*\*\* If  $\text{ABS}(X) \leq M$  in constraints, Replace with  $X \leq M$  and  $-X \leq M$

\*\*\* Fractional LP:

max/min

$$3x_1 + 2x_2 + 4x_3 + 6$$

-----

$$3x_1 + 3x_2 + 2x_3 + 5$$

constraint:

$$2x_1 + 3x_2 + 5x_3 \geq 23$$

$$3x_2 + 5x_2 + 4x_3 \leq 30$$

$$x_1, x_2, x_3 \geq 0$$

Replace with:

max/min

$$3y_1 + 2y_2 + 4y_3 + 6t$$

constraint:

$$3y_1 + 3y_2 + 2y_3 + 5t = 1$$

$$2y_1 + 3y_2 + 5y_3 - 23t \geq 0$$

$$3y_1 + 5y_2 + 4y_3 - 30t \leq 0$$

$$y_1, y_2, y_3, t \geq 0$$

\*\*\*/

//Complexity: O( $n^3$ ) or faster

//MAXV=1e5,MAXC=2 works within 80ms

//MAXV=1e5,MAXC=200 works within 1.5s  
namespace lp

{

long double val[MAXV], ar[MAXC][MAXV];

int m, n, solution\_flag, minmax\_flag, basis[MAXC], index[MAXV];

/// nvars = number of variables, f = objective function, flag = MINIMIZE or MAXIMIZE

inline void init(int nvars, long double f[], int flag){

solution\_flag = 0;

ar[0][nvars] = 0.0;

m = 0, n = nvars, minmax\_flag = flag;

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

ar[0][i] = f[i] \* minmax\_flag; // Negating sign of objective function when minimizing

}

}

```

/// C[] = co-efficients of the constraints (LHS), lim = limit in RHS
/// cmp = EQUAL for C[] = lim, LESSEQ for C[] <= lim, GREATERQ for
C[] >= lim
inline void add_constraint(long double C[], long double lim, int
cmp){
 m++, cmp *= -1;
 if (cmp == 0){
 for (int i = 0; i < n; i++) ar[m][i] = C[i];
 ar[m++][n] = lim;
 for (int i = 0; i < n; i++) ar[m][i] = -C[i];
 ar[m][n] = -lim;
 }
 else{
 for (int i = 0; i < n; i++) ar[m][i] = C[i] * cmp;
 ar[m][n] = lim * cmp;
 }
}

inline void init(){ // Initialization
 for (int i = 0; i <= m; i++) basis[i] = -i;
 for (int j = 0; j <= n; j++){
 ar[0][j] = -ar[0][j], index[j] = j, val[j] = 0;
 }
}

inline void pivot(int m, int n, int a, int b){ // Pivoting and exchanging
a non-basic variable with a basic variable
 for (int i = 0; i <= m; i++){

```

```

 if (i != a){
 for (int j = 0; j <= n; j++){
 if (j != b){
 ar[i][j] -= (ar[i][b] * ar[a][j]) / ar[a][b];
 }
 }
 }
 for (int j = 0; j <= n; j++){
 if (j != b) ar[a][j] /= ar[a][b];
 }
 for (int i = 0; i <= m; i++){
 if (i != a) ar[i][b] = -ar[i][b] / ar[a][b];
 }
 ar[a][b] = 1.0 / ar[a][b];
 swap(basis[a], index[b]);
 }
}

inline long double solve(){ // simplex core
 init();
 int i, j, k, l;
 for (; ;){
 for (i = 1, k = 1; i <= m; i++){
 if ((ar[i][n] < ar[k][n]) || (ar[i][n] == ar[k][n] && basis[i] <
basis[k] && (rand() & 1))) k = i;
 }
 if (ar[k][n] >= -EPS) break;
 for (j = 0, l = 0; j < n; j++){

```

```

 if ((ar[k][j] < (ar[k][l] - EPS)) || (ar[k][j] < (ar[k][l] - EPS) &&
index[i] < index[j] && (rand() & 1))){
 l = j;
 }
 }
 if (ar[k][l] >= -EPS){
 solution_flag = INFEASIBLE; /// No solution is possible
 return -1.0;
 }
 pivot(m, n, k, l);
}
for (; ;){
 for (j = 0, l = 0; j < n; j++){
 if ((ar[0][j] < ar[0][l]) || (ar[0][j] == ar[0][l] && index[j] <
index[l] && (rand() & 1))) l = j;
 }
 if (ar[0][l] > -EPS) break;
 for (i = 1, k = 0; i <= m; i++){
 if (ar[i][l] > EPS && (!k || ar[i][n] / ar[i][l] < ar[k][n] / ar[k][l] -
EPS || (ar[i][n] / ar[i][l] < ar[k][n] / ar[k][l] + EPS && basis[i] <
basis[k]))){
 k = i;
 }
 }
 if (ar[k][l] <= EPS){
 solution_flag = UNBOUNDED; /// Solution is infinity, no finite
solution exists
 return -666.0;
 }
}

```

```

pivot(m, n, k, l);
}
for (i = 1; i <= m; i++){
 if (basis[i] >= 0) val[basis[i]] = ar[i][n];
}
solution_flag = 1; /// Successful completion
return (ar[0][n] * minmax_flag); /// Negate the output for
MINIMIZE since the objective function was negated
}
}
long double obj[N], cons[N];

int32_t main()
{
 BeatMeScanf;
 // i,x,y,w,j,black = 0;
 cin>>n>>k;
 for(i=0;i<n;i++) cin>>a[i], black += a[i];
 for(i=1;i<n;i++)
 {
 cin>>x>>y>>w;
 x--;
 y--;
 g[x].pb(y);
 g[y].pb(x);
 cost[x].pb(w);
 cost[y].pb(w);
 }
 for(i=0;i<n;i++) dfs(i,-1,0,i);
}

```

```

for(i=0;i<n;i++)
{
 for(j=0;j<n;j++)
 {
 if(dis[i][j] <= k)ok[i][j] = 1;
 }
}
for(i=0;i<n;i++) obj[i] = (long double)1 - a[i];
lp::init(n,obj,MINIMIZE);
for(i=0;i<n;i++)cons[i] = 1.0;
lp::add_constraint(cons,black,EQUAL);
for(i=0;i<n;i++)
{
 for(j=0;j<n;j++)
 {
 cons[j] = (long double)ok[i][j];
 }
 lp::add_constraint(cons,1.0,GREATEQ);
}
long double ret = lp::solve();
if(ans < 0)ans = -1;
cout<<ans<<nl;
return 0;
}

// Permutation Hash

```

```

//
// Description:
// hash_perm gives one-to-one correspondence between
// permutations over [0,n) and the integer less than n!.
// unhash_perm is the inverse function of hash_perm.
//
// Algorithm:
// The idea is based on the Fisher-Yates shuffle algorithm:
// while n > 1:
// swap(x[n-1], x[rand() % n]);
// --n;
// For an integer given by a factorial number system:
// hash = d_0 (n-1)! + d_1 (n-2)! + ... + d_{n-1} 0!
// The algorithm computes
// while n > 1:
// swap(x[n-1], x[d_{n-1}])
// --n;
//
// Complexity:
// O(n) time, O(n) space.
//
// Verification:
// self.


```

```

#include <iostream>
#include <vector>
#include <cstdio>
#include <algorithm>

```

### 133. Permutation Hash

```

#include <numeric>
#include <functional>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())

typedef long long ll;

vector<int> unhash_perm(ll r, int n) {
 vector<int> x(n);
 iota(all(x), 0);
 for (; n > 0; --n) {
 swap(x[n-1], x[r % n]);
 r /= n;
 }
 return x;
}

ll hash_perm(vector<int> x) {
 int n = x.size();
 vector<int> y(n);
 for (int i = 0; i < n; ++i) y[x[i]] = i;
 ll c = 0, fac = 1;
 for (; n > 1; --n) {
 c += fac * x[n-1]; fac *= n;
 swap(x[n-1], x[y[n-1]]);
 swap(y[n-1], y[x[y[n-1]]]);
 }
 return c;
}

int main() {
 int n = 9;
 vector<int> x(n);
 iota(all(x), 0);
 do {
 ll r = hash_perm(x);
 cout << r << ": ";
 auto a = unhash_perm(r, n);
 for (int i = 0; i < n; ++i) {
 cout << a[i] << " ";
 if (a[i] != x[i]) exit(-1);
 }
 cout << endl;
 } while (next_permutation(all(x)));
 return 0;
}

```

## 134. GP Hash Table

```

#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

// For integer
gp_hash_table<int, int> table;

// Custom hash function approach is better

```

```

const int RANDOM = // so table[i][j][k] is storing an integer for corresponding k as hash
chrono::high_resolution_clock::now().time_since_epoch().count();
struct hash {
 int operator()(int x) const { return x ^ RANDOM; }
};

gp_hash_table<int, int, hash> table;

const ll TIME = // For pairs
chrono::high_resolution_clock::now().time_since_epoch().count();
const ll SEED = (ll)(new ll);
const ll RANDOM = TIME ^ SEED;
const ll MOD = (int)1e9+7;
const ll MUL = (int)1e6+3;
struct hash{
 ll operator()(ll x) const { return std::hash<ll>{}((x ^ RANDOM) %
MOD * MUL); }
};
gp_hash_table<ll, int, hash> table;

unsigned hash_f(unsigned x) {
 x = ((x >> 16) ^ x) * 0x45d9f3b;
 x = ((x >> 16) ^ x) * 0x45d9f3b;
 x = (x >> 16) ^ x;
 return x;
}
struct hash {
 int operator()(ll x) const { return hash_f(x); }
};
gp_hash_table<ll, int, hash> table[N][N]; //this is faster
 // The better the hash function, the less collisions
 // Note that hash function should not be costly
struct hash {
 int operator()(pii x) const { return x.first* 31 + x.second; }
};
gp_hash_table<pii, int, hash> table;

// Another recommended hash function by neal on CF
struct custom_hash {
 static uint64_t splitmix64(uint64_t x) {
 // http://xorshift.di.unimi.it/splitmix64.c
 x += 0x9e3779b97f4a7c15;
 x = (x ^ (x >> 30)) * 0xb58476d1ce4e5b9;
 x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
 return x ^ (x >> 31);
 }

 size_t operator()(uint64_t x) const {
 static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
 return splitmix64(x + FIXED_RANDOM);
 }
};
gp_hash_table<ll,int,custom_hash> safe_gp_hash_table;
unordered_map<ll,int,custom_hash> safe_umap;

```

```

typedef gp_hash_table<int, int, hash<int>,
equal_to<int>, direct_mod_range_hashing<int>, linear_probe_fn<>,
hash_standard_resize_policy<hash_prime_size_policy,
hash_load_check_resize_trigger<true>, true>>
gp;
gp Tree;
// Now Tree can probably be used for fenwick, indices can be long
long
// S is an offset to handle negative value
// If values can be >= -1e9, S=1e9+1
// maxfen is the MAXN in fenwick, this case it was 2e9+2;
// Note that it was okay to declare gp in integer as the values were
// still in the range of int.
void add(long long p, int v) {
 for (p += S; p < maxfen; p += p & -p)
 Tree[p] += v;
}
int sum(int p) {
 int ans = 0;
 for (p += S; p; p ^= p & -p)
 ans += Tree[p];
 return ans;
}

```

### 135. All Pair Max and 2<sup>nd</sup> Max Generator

```

///find $\max_{1 \leq l \leq r \leq n} (mx(a[l..r]) ^ 2nd mx(a[l..r]))$
ll a[N];
int main()

```

```

{
fast;
ll i,j,k,n,m,t;
cin>>n;
for(i=1;i<=n;i++) cin>>a[i];
ll ans=0;
stack<ll>st;
for(i=1;i<=n;i++){
 while(!st.empty()){
 ans=max(ans,a[i]^st.top());
 if(st.top()<a[i]) st.pop();
 else break;
 }
 st.push(a[i]);
}
cout<<ans<<nl;
return 0;
}

```

### 136. Huffman Coding

```

/**
assign each character a binary code so that no code is a prefix of
another code
and sum of frequency*(code length) is minimum

```

The version implemented here analyses a given ASCII character string in order to obtain optimal codes based on character frequencies in the string.

Complexity:  $O(n \log n)$  time to construct

$O(H)$  time (on average) to encode/decode, where  $H$  is the height of the huffman tree

\*\*/

// The string to analyse

string str;

// Maps the encountered characters to their relative frequencies  
map<char, double> P;

// The Huffman Tree

struct HuffTreeNode

{

double p;

bool leaf;

char letter;

HuffTreeNode \*parent;

HuffTreeNode \*l;

HuffTreeNode \*r;

// nonleaf node

HuffTreeNode(double p, HuffTreeNode \*l, HuffTreeNode \*r)

{

this -> p = p;

this -> leaf = false;

this -> letter = '\0';

this -> parent = NULL;

this -> l = l;

this -> r = r;

l -> parent = this;

r -> parent = this;

}

// leaf node

HuffTreeNode(double p, char c)

{

this -> p = p;

this -> leaf = true;

this -> letter = c;

this -> parent = this -> l = this -> r = NULL;

}

};

// Comparator of two node pointers

struct CmpNodePtrs

{

// As priority\_queue is a max heap rather than min-heap,

// invert the meaning of the < operator,

// in order to get lower probabilities at the top

bool operator()(const HuffTreeNode\* lhs, const HuffTreeNode\* rhs) const

{

return (lhs -> p) > (rhs -> p);

}

};

```

// the root of the tree
HuffTreeNode *root;

// mapping each character to its leaf node (for quick encoding)
map<char, HuffTreeNode*> leaf;

// Produces the probability distribution (may be omitted if known in
// advance)
inline void analyse()
{
 for (int i=0;i<str.length();i++)
 {
 P[str[i]]++;
 }
 for (auto it = P.begin();it!=P.end();it++)
 {
 P[it -> first] = it -> second / str.length();
 }
}

// Construct the Huffman Tree using the probability distribution
inline void build_tree()
{
 priority_queue<HuffTreeNode*, vector<HuffTreeNode*>, CmpNodePtrs> pq;

 // First construct the leaves, and fill the priority queue
 for (auto it = P.begin();it!=P.end();it++)

```

```

 {
 leaf[it -> first] = new HuffTreeNode(it -> second, it -> first);
 pq.push(leaf[it -> first]);
 }

 while (pq.size() > 1)
 {
 HuffTreeNode* L = pq.top(); pq.pop();
 HuffTreeNode* R = pq.top(); pq.pop();

 // Spawn a new node generating the children
 HuffTreeNode* par = new HuffTreeNode((L -> p) + (R -> p), L, R);
 pq.push(par);
 }

 root = pq.top(); pq.pop();
}

// Huffman-encode a given character
inline string encode(char c)
{
 string ret = "";

 HuffTreeNode* curr = leaf[c];
 while (curr -> parent != NULL)
 {
 if (curr == curr -> parent -> l) ret += "0";
 else if (curr == curr -> parent -> r) ret += "1";
 }
}

```

```

curr = curr -> parent;
}

reverse(ret.begin(), ret.end());
return ret;
}

// Huffman-encode the given string
inline string encode(string s)
{
 string ret = "";

 for (int i=0;i<s.length();i++)
 {
 ret += encode(s[i]);
 }

 return ret;
}

// Huffman-decode the given string
inline string decode(string s)
{
 string ret = "";

 int i = 0;
 HuffTreeNode* curr;

 while (i < s.length())

```

```

 {
 curr = root;
 while (!(curr -> leaf))
 {
 if (s[i++] == '0') curr = curr -> l;
 else curr = curr -> r;
 }
 ret += curr -> letter;
 }
 return ret;
}

int main()
{
 str = "this is an example of a huffman tree";

 analyse();
 build_tree();

 string test = "aefhimnstloprux";
 for (int i=0;i<test.length();i++)
 {
 cout << "Encode(" << test[i] << ") = " << encode(test[i]) << endl;
 }

 cout << encode("this is real") << endl;
 cout << decode("110000101011001111101100111110010010111100") << endl;
}

```

```

cout << decode(encode("this is real")) << endl;

return 0;
}

```

## 137. All Nearest Smaller Values

```

/*
linear time all nearest smaller values, standard stack-based algorithm.
get_left stores indices of nearest smaller values to the left in res. -1
means no smaller value was found.
get_right likewise looks to the right. v.size() means no smaller value
was found.
*/
vi get_left(vector<int>& v) {
 stack<pair<int, int> > st;
 st.push(MP(INT_MIN, sz(v)));
 vi res;
 res.resize(sz(v));
 for (int i = sz(v)-1; i >= 0; i--) {
 while (st.top().F > v[i]) {
 res[st.top().S] = i;
 st.pop();
 }
 st.push(MP(v[i], i));
 }
 while (st.top().S < sz(v)) {
 res[st.top().S] = -1;
 st.pop();
 }
}

```

```

 }
 return res;
}

vi get_right(vector<int>& v) {
 stack<pair<int, int> > st;
 st.push(MP(INT_MIN, -1));
 vi res;
 res.resize(sz(v));
 for (int i = 0; i < sz(v); i++) {
 while (st.top().F > v[i]) {
 res[st.top().S] = i;
 st.pop();
 }
 st.push(MP(v[i], i));
 }
 while (st.top().S > -1) {
 res[st.top().S] = sz(v);
 st.pop();
 }
 return res;
}

```

## 138. Fraction Binary Search

```

/*
Given a function f and n, finds the smallest fraction p/q in [0, 1] or
[0,n]
such that f(p/q) is true, and p, q<=n.
Time: O(log(n))
*/

```

```

**/

struct frac { ll p, q; };
bool f(frac x)
{
 return 6+8*x.p>=17*x.q+12;
}
frac fracBS(ll n) {
 bool dir = 1, A = 1, B = 1;
 frac lo{0, 1}, hi{1, 0}; // Set hi to 1/0 to search within [0, n]
and {1,1} to search within [0,1]
 if (f(lo)) return lo;
 assert(f(hi));///checking if any solution exists or not
 while (A || B) {
 ll adv = 0, step = 1; // move hi if dir, else lo
 for (int si = 0; step; (step *= 2) >>= si) {
 adv += step;
 frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
 if (abs(mid.p) > n || mid.q > n || dir == !f(mid))
{
 adv -= step; si = 2;
 }
 hi.p += lo.p * adv;
 hi.q += lo.q * adv;
 dir = !dir;
 swap(lo, hi);
 A = B; B = !adv;
 }
 }
}

```

```

 return dir ? hi : lo;
}
```

```

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 frac ans=fracBS(10);
 cout<<ans.p<<' '<<ans.q<<nl;
 return 0;
}
```

### 139. Longest Zigzag Subsequence

```

/***
A sequence xs is zigzag if $x[i] < x[i+1], x[i+1] > x[i+2]$, for all i

(initial direction can be arbitrary). The maximum length zigzag

subsequence is computed in $O(n)$ time by a greedy method.
***/

int lzs(vector<int> v) {
 int n = v.size(), len = 1, prev = -1;
 for (int i = 0, j = n; i < n; i = j) {
 for (j = i+1; j < n && v[i] == v[j]; ++j);
 if (j < n) {
 int sign = (v[i] < v[j]);
 if (prev != sign) ++len;
 prev = sign;
 }
 }
 return len;
}
```

```

}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cout<<lzs({1,3,1,3,4,2})<<nl;
 return 0;
}

```

## 140. Bit Hacks

///well, it describes itself

```

unsigned int reverse_bits(unsigned int v){
 v = ((v >> 1) & 0x55555555) | ((v & 0x55555555) << 1);
 v = ((v >> 2) & 0x33333333) | ((v & 0x33333333) << 2);
 v = ((v >> 4) & 0x0F0F0F0F) | ((v & 0x0F0F0F0F) << 4);
 v = ((v >> 8) & 0x00FF00FF) | ((v & 0x00FF00FF) << 8);
 return ((v >> 16) | (v << 16));
}

```

/// Returns i if  $x = 2^i$  and 0 otherwise

```

int bitscan(unsigned int x){
 __asm__ volatile("bsf %0, %0" : "=r" (x) : "0" (x));
 return x;
}

```

/// Only for non-negative integers

/// Returns the immediate next number with same count of one bits,  
-1 on failure

```

long long next_one(long long n){

```

```

 if (n == 0) return -1;
 long long x = (n & -n);
 long long left = (x + n);
 long long right = ((n ^ left) / x) >> 2;
 long long res = (left | right);
 return res;
}

```

/// Returns the immediate previous number with same count of one bits, -1 on failure

```

long long prev_one(long long n){
 if (n == 0 || n == 1) return -1;
 long long res = ~next_one(~n);
 return (res == 0) ? -1 : res;
}

```

///generate all subsets of mask in descending order

```

for (int sub = mask; ; sub = (sub - 1) & mask) {
 ...
 if (sub == 0) break;
}

```

///generate all subsets of mask in ascending order

```

for (int sub = 0; ; sub = (sub - mask) & mask) {
 ...
 if (sub == mask) break;
}

```

///generate all supersets of mask in ascending order

```

for (int super = mask; super < (1 << n); super = (super + 1) | mask) {
 ...
}

///generate all masks such that popcount(mask)= k in ascending order
for (int mask = (1 << k) - 1; mask < (1 << n);) {
 ...
 int tmp = mask | (mask - 1);
 mask = (tmp + 1) | (((~tmp & - ~tmp) - 1) >> (__builtin_ctz(mask)
+ 1));
}

///generate all submasks of mask having only one bit set in ascending
order
for (int sub = mask & - mask; sub; sub = mask & (~ mask + (sub << 1)))
{
 ...
}

```

## 141. Fibonacci Faster

```

int fib(LL n, int mod) {
 assert (3 * pow(mod, 2) < pow(2, 63));
 assert (n >= 0);
 if (n <= 1) return n;
 int a = 0;
 int b = 1;
 LL i = 1LL << (63 - __builtin_clzll(n) - 1);
 for (; i; i >>= 1) {
 int na = (a *(LL) a + b *(LL) b) % mod;

```

```

 int nb = (2LL * a + b) * b % mod;
 a = na;
 b = nb;
 if (n & i) {
 int c = a + b; if (c >= mod) c -= mod;
 a = b;
 b = c;
 }
 }
 return b;
}

int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cout<<fib(1e18,mod)<<nl;
 return 0;
}

```

## 142. System Of Difference Constraints

/\*\*Problem:

Given some inequality on some variable ( $x_i, x_j, \dots$ ) in form  $x_j - x_i \leq w$ , we need to determine whether we can assign values to the variables so that

all the given inequalities are satisfiable or not. If satisfiable, then output a solution.

Solution:

\* For each variable we create a vertex.

\* For each inequality,  $x_j - x_i \leq w$ , we give a directed edge  $(v_i, v_j)$  with cost  $w$ .

\* Create a source vertex  $S$  and give an edge  $(S, v_i)$  for all vertices with cost 0. Can be solved without source vertex if we use SPFA.

The SPFA code for determining existence of negative cycle:

```
*/
bool spfa()
{
 queue<int> Q;
 for(int i=1; i<=n; i++)
 {
 Q.push(i);
 dist[i] = inf;
 inq[i] = true;
 cntr[i] = 1;
 }
 dist[1] = 0;
 while(!Q.empty())
 {
 int u = Q.front();
 Q.pop();
 inq[u] = false;
 for(auto it: graph[u])
 {
 int v = it[0], w = it[1];
 if(dist[v] > dist[u] + w)
 {
 dist[v] = dist[u] + w;
```

```
 if(!inq[v])
 {
 inq[v] = true;
 cntr[v]++;
 Q.push(v);
 if(cntr[v]>n)
 return false;
 }
 }
 }
 return true;
 }
}

If the constraint graph contains a negative cycle, then the system of differences is unsatisfiable.

Determining a Possible Solution:
* If there is no negative cycle in the constraint graph, then there is a solution for the system.
* For each variable x_i , x_i = shortest path distance of v_i from the source vertex in constraint graph.
* Let $x = x_1, x_2, \dots, x_n$ be a solution to a system of difference constraints and let d be any constant. Then $x + d = x_1 + d, x_2 + d, \dots, x_n + d$ is a solution as well.
* Shortest Path can be calculated from Bellman-Ford algorithm.
* Bellman-Ford maximizes $x_1 + x_2 + \dots + x_n$ subject to the constraints $x_j - x_i \leq w_{ij}$ and $x_i \leq 0$
* Bellman-Ford also minimizes $\max(x_i) - \min(x_i)$
```

```
**/
```

## 143. Slope

```
struct slope
{
 int x, y;
 slope(int _x, int _y) { x = _x; y = _y; }
 slope() { x = y = 0; }
 void reduce()
 {
 int g=__gcd(x,y);
 if(g) x/=g,y/=g;
 if(y < 0) x *= -1, y *= -1;
 }
 //the slope
 pair<int, int> get() { return {x, y}; }
 ///perpendicular/orthogonal line slope on this line
 slope ort()
 {
 slope ret;
 ret = slope(-y, x);
 ret.reduce();
 return ret;
 }
};
///slope of the line (x1,y1) to (x2,y2)
slope line(int x1, int y1, int x2, int y2)
{
 int dx = x1 - x2;
```

```
 int dy = y1 - y2;
 slope ret = slope(dy, dx);
 ret.reduce();
 return ret;
}
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 slope p(6,-4);
 p.reduce();
 cout<<p.x<<' '<<p.y;
 return 0;
}
```

## 144. Sequence Merge

```
///given n sequences seq0,seq1,...seq(n-1), and an integer m
///find the largest m values that can be made using exactly one value
from each sequence
///Complexity O(mlogm+K) where sum of size(seq_i)=K
struct node
{
 int v, x, y;
 bool sp;
 bool operator < (node const &t) const
 {
 return v < t.v;///v>t.v for m smallest values
 }
};
```

```

///n=seq.size();
vector<int> sequence_merge(int n, int m, vector<vector<int>>seq) {
 vector<int> ret;
 ret.reserve(m);

 int base = 0;///largest value
 vector<vector<int>>a;
 int id=0;
 for(int i = 0; i < n; ++i) {
 assert(sz(seq[i])>0);
 sort(seq[i].begin(), seq[i].end(),
greater<int>());///check for long long, increasing sort for m smallest
values
 base += seq[i][0];
 if(sz(seq[i])>1) a.eb(seq[i]);
 }
 n=sz(a);
 sort(a.begin(),a.end(), [&](vector<int> const &u, vector<int>
const &v) {
 return u[0] - u[1] < v[0] - v[1];///(u[1]-u[0]<v[1]-v[0])
for m smallest values
 });
 ///rest of the code is same for m smallest values
 priority_queue<node> Q;
 Q.push((node){base, 0, 0, 0});
 for(int i = 0; i < m && !Q.empty(); ++i) {
 node cur = Q.top();
 Q.pop();
 ret.push_back(cur.v);
 }
}

```

```

if(cur.y + 1 < (int)a[cur.x].size())
 Q.push((node){cur.v - a[cur.x][cur.y] +
a[cur.x][cur.y + 1], cur.x, cur.y + 1, 0});
 if(cur.x + 1 < n)
 Q.push((node){cur.v - a[cur.x + 1][0] + a[cur.x +
1][1], cur.x + 1, 1, 1});
 if(cur.sp)
 Q.push((node){cur.v - a[cur.x][1] +
a[cur.x][0] - a[cur.x + 1][0] + a[cur.x + 1][1], cur.x + 1, 1, 1});
}
return ret;
}
vector<vi>a;
int32_t main()
{
 BeatMeScanf;
 int i,j,k,n,m;
 cin>>n;
 for(i=0;i<n;i++){
 cin>>k;
 vi p;
 for(j=0;j<k;j++){
 cin>>m;
 p.eb(m);
 }
 a.eb(p);
 }
 cin>>k;
}

```

```
vi ans=sequence_merge(n,k,a);
printv(ans);
return 0;
}
```