```
       444444444  77777777777777777777
      4::::::::4  7::::::::::::::::::7
     4:::::::::4  7::::::::::::::::::7
    4::::44::::4  777777777777:::::::7
   4::::4 4::::4             7::::::7
  4::::4  4::::4            7::::::7
 4::::4   4::::4           7::::::7
4::::444444::::444        7::::::7
4::::::::::::::::4       7::::::7
4444444444:::::444      7::::::7
          4::::4       7::::::7
          4::::4      7::::::7
          4::::4     7::::::7
        44::::::44  7::::::7
        4::::::::4 7::::::7
        444444444477777777
```

```
// 1. C++ template

#include <bits/stdc++.h>
using namespace std;

#define FOR(i,a,b) for (int i = (a); i < (b); i++)
#define RFOR(i,b,a) for (int i = (b) - 1; i >= (a); i--)
#define ITER(it,a) for (__typeof(a.begin()) it = a.begin(); it != a.end(); it++)
#define FILL(a,value) memset(a, value, sizeof(a))

#define SZ(a) (int)a.size()
#define ALL(a) a.begin(), a.end()
#define PB push_back
#define MP make_pair

typedef long long LL;
typedef vector<int> VI;
typedef pair<int, int> PII;

const double PI = acos(-1.0);
const int INF = 1000 * 1000 * 1000 + 7;
const LL LINF = INF * (LL) INF;

int main()
{
	// freopen("in.txt", "r", stdin);
	// ios::sync_with_stdio(false); cin.tie(0);
}
```

```
// 2. FFT with complex

// Don't use for long long values
// Except for some special cases
// Precalc roots of -1
typedef complex<double> base;
const int LEN  = 1<<20; // max length, power of 2
base PW[LEN];           // LEN-th roots of -1
void fft(vector<base>& a, bool invert)
{
	int lg = 0;
	while((1<<lg) < SZ(a)) lg++;
	FOR (i, 0, SZ(a))
	{
		int x=0;
		FOR (j, 0, lg)
			x |= ((i>>j)&1)<<(lg-j-1);
		if(i<x)
			swap(a[i], a[x]);
	}
	for (int len = 2; len <= SZ(a); len *= 2)
	{
		int diff = LEN / len;
		if (invert) diff = LEN - diff;
		for (int i = 0; i < SZ(a); i += len)
		{
			int pos = 0;
			FOR (j, 0, len/2)
			{
				base v = a[i+j];
				base u = a[i+j+len/2] * PW[pos];

				a[i+j] = v + u;
				a[i+j+len/2] = v - u;
				pos += diff;
				if (pos >= LEN) pos -= LEN;
			}
		}
	}
	if (invert)
		FOR (i, 0, SZ(a))
			a[i] /= SZ(a);
}
void initFFT()
{
	double angle = 2 * PI / LEN;
	FOR (i, 0, LEN)
	{
		double ca = angle * i;
		PW[i] = base(cos(ca), sin(ca));
	}
}
```

```
// 3. FFT with modulo

// GEN ^ LEN == 1 mod BASE
// GEN ^ (LEN / 2) != 1 mod BASE
// for prime modulo c2^k+1 generator for len 2^k exists always
const int LEN = 1<<20;    // max length, power of 2
const int BASE = 7340033; // modulo
const int GEN = 5;        // generator
int PW[LEN];              // powers of generator
void fft(vector<int>& a, bool invert)
{
	int lg = 0;
	while((1<<lg) < SZ(a)) lg++;

	FOR (i, 0, SZ(a))
	{
		int x=0;
		FOR (j, 0, lg)
			x |= ((i>>j)&1)<<(lg-j-1);
		if(i<x)
			swap(a[i], a[x]);

	}

	for (int len = 2; len <= SZ(a); len *= 2)
	{
		int diff = LEN / len;
		if (invert) diff = LEN - diff;
		for (int i = 0; i < SZ(a); i += len)
		{
			int pos = 0;
			FOR (j, 0, len/2)
			{
				int w = PW[pos];

				int v = a[i+j];
				int u = (a[i+j+len/2] * (LL) w) % BASE;

				int t = v + u;
				if (t >= BASE) t -= BASE;
				a[i+j] = t;

				t = v - u;
				if (t < 0)
					t += BASE;
```

```
                    a[i+j+len/2] = t;

                    pos += diff;
                    if (pos >= LEN) pos -= LEN;
                }
            }
        }

        if (invert)
        {
            int m = inv(SZ(a), BASE);
            FOR (i, 0, SZ(a))
                a[i] = (a[i] * (LL)m) % BASE;
        }
}
```

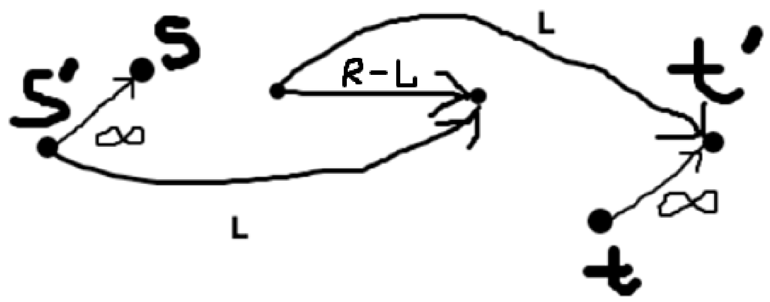
// 4. Min-cost max-flow with Levit

```
struct edge
{
    int x, y;
    int c, f;
    LL p;
};
vector<edge> E;
vector<int> g[MAX];
int N;
LL D[MAX];
int Par[MAX];
int T[MAX];
int Q[MAX];

void add_edge(int x, int y, int c, LL p)
{
    edge e;
    e.x = x; e.y = y;
    e.c = c; e.f = 0;
    e.p = p;
    g[x].PB(SZ(E));
    E.PB(e);

    e.x = y; e.y = x;
    e.c = 0; e.f = 0;
    e.p = -p;
    g[y].PB(SZ(E));
    E.PB(e);
}
pair<int, LL> Flow(int s, int t)
{
    int flow = 0;
    LL price = 0;
    while (true)
    {
        FOR(i, 0, N)
        {
            D[i] = LINF;
            Par[i] = -1;
            T[i] = 0;
        }

        T[s] = 1;
        D[s] = 0;
        Q[0] = s;
        int qh = 0, qt = 1;
        while (qh != qt)
        {
            int x = Q[qh++];
            if (qh == N) qh = 0;

            FOR(i, 0, SZ(g[x]))
            {
                int e = g[x][i];
                if (E[e].f == E[e].c) continue;
                int to = E[e].y;
                LL p = E[e].p;
                if (D[to] > D[x] + p)
                {
                D[to] = D[x] + p;
                Par[to] = e;

                if (T[to] == 0)
                {
                    Q[qt++] = to;
                    if (qt == N) qt = 0;
                }
                if (T[to] == 2)
                {
                    qh--;
                    if (qh == -1) qh = N - 1;
                    Q[qh] = to;
                }

                T[to] = 1;
                }
            }

            T[x] = 2;
        }

        if (Par[t] == -1) break;

        int x = t;
        int f = INF;
        LL p = 0;
        while (x != s)
        {
            int e = Par[x];
            p += E[e].p;
            f = min(f, E[e].c - E[e].f);
            x = E[e].x;
        }

        x = t;
        while(x != s)
        {
            int e = Par[x];
            E[e].f += f;
            E[e^1].f -= f;
            x = E[e].x;
        }

        flow += f;
        price += p * f;
    }

    return MP(flow, price);
}
```

```
 // 5. Dijkstra with potentials

// - each vertex has potential P[x]
// - initially potentials = distances from the source
// - for edge (x, y) the weight is D(x, y) + P[x] - P[y] >= 0
// - after each itration:
// - - if (D[x] < INF) P[x] += D[x];

 // 6. Kun algorithm
VI g[MAX];   // edges from left to right
int mt[MAX]; // matching vertex on the left
int P[MAX];  // matching vertex on the rigth
int U[MAX];
int iter;
bool kun(int x)
{
	if (U[x] == iter) return false;
	U[x] = iter;
	FOR (i, 0, SZ(g[x]))
	{
		int to = g[x][i];
		if (mt[to] == -1)
		{
			mt[to] = x;
			P[x] = to;
			return true;
		}
	}
	FOR (i, 0, SZ(g[x]))
	{
		int to = g[x][i];
		if (kun(mt[to]))
		{
			mt[to] = x;
			P[x] = to;
			return true;
		}
	}
	return false;
}
int doKun(int n)
{
	FILL(mt, -1);
	FILL(P, -1);
	FILL(U, -1);
	int res = 0;
	iter = 0;
	while(true)
	{
		iter++;
		bool ok = false;
		FOR (i, 0, n)
			if (P[i] == -1)
				if (kun(i))
				{
					ok = true;
					res++;
				}
		if (!ok) break;
	}
	return res;
}
```

```
 // 7. Dinic max-flow algorithm
struct edge
{
	int x, y;
	LL c, f;
};
vector<edge> E;
VI g[MAX];
int D[MAX];
int Q[MAX];
int Ptr[MAX];
int N; // number of vertices in the network (required)
void add_edge(int x, int y, LL c)
{
	edge e;
	e.x = x; e.y = y;
	e.c = c; e.f = 0;
	g[x].PB(SZ(E));
	E.PB(e);
	e.x = y; e.y = x;
	e.c = 0; e.f = 0;
	g[y].PB(SZ(E));
	E.PB(e);
}
int bfs(int s, int t)
{
	FOR (i, 0, N)
		D[i] = -1;
	D[s] = 0;
	Q[0] = s;
	int qh = 0, qt = 1;
	while(qh < qt && D[t] == -1)
	{
		int x = Q[qh++];
		FOR (i, 0, SZ(g[x]))
		{
			int e = g[x][i];
			if (E[e].f == E[e].c) continue;
			int to = E[e].y;

			if (D[to] == -1)
			{
				D[to] = D[x] + 1;
				Q[qt++] = to;
			}
		}
	}

	return D[t];
}
LL dfs(int x, int t, LL flow)
{
	if (x == t || flow == 0) return flow;
	for (; Ptr[x] < SZ(g[x]); Ptr[x]++)
	{
		int e = g[x][Ptr[x]];
		LL c = E[e].c;
		LL f = E[e].f;
		int to = E[e].y;
		if (c == f) continue;
		if (D[to] == D[x] + 1)
		{
			LL push = dfs(to, t, min(flow, c - f));
			if (push > 0)
```

```
                {
                        E[e].f += push;
                        E[e^1].f -= push;
                        return push;
                }
            }
        }
        return 0;
}
LL Flow(int s, int t)
{
        LL res = 0;
        while(bfs(s, t) != -1)
        {
                FOR (i, 0, N)
                {
                        Ptr[i] = 0;
                }
                while(true)
                {
                        LL push = dfs(s, t, LINF);
                        if (push == 0) break;
                        res += push;
                }
        }
        return res;
}
 // 8. Flow with lower bound
```




```
 // 9. Hungarian algorithm

int u[MAX];
int v[MAX];
int p[MAX];
int way[MAX];
int minv[MAX];
bool used[MAX];
// Complexity: O(N^2*M)
int Hungarian(vector<VI> a)
{
    int n = a.size();
    int m = a[0].size();
    FOR(i,0,n + 1)
        u[i] = 0;
    FOR(j,0,m + 1)
    {
        v[j] = 0;
        p[j] = n;
        way[j] = 0;
    }
    FOR(i,0,n)
    {
        p[m] = i;
        int j0 = m;
        FOR(j,0,m + 1)
        {
            minv[j] = INF;
            used[j] = 0;
        }
        while (p[j0] != n)
        {
            used[j0] = true;
            int i0 = p[j0];
            int j1;
            int delta = INF;
            FOR(j, 0, m)
            {
                if (!used[j])
                {
                    int cur = a[i0][j] - u[i0] - v[j];
                    if (cur < minv[j])
                    {
                        minv[j] = cur;
                        way[j] = j0;
                    }
                    if (minv[j] < delta)
                    {
                        delta = minv[j];
                        j1 = j;
                    }
                }
            }
            FOR(j,0,m + 1)
            {
                if (used[j])
                {
                    u[p[j]] += delta;
                    v[j] -= delta;
                }
                else
                    minv[j] -= delta;
            }
            j0 = j1;
        }
        while(j0 != m)
        {
            int j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        }
    }
    vector<int> ans (n + 1);
    FOR(j,0,m)
        ans[p[j]] = j;
    int res = 0;
    FOR(i,0,n)
        res += a[i][ans[i]];
    assert(res == -v[m]);
    return res;
}
```

```
// 10. Centroid decomposition

// dfsSz calculates sizes of subtrees
void build(int x)
{
	dfsSZ(x, -1);
	int szAll = SZ[x];
	int p = x;
	while(true)
	{
		int w = -1;
		FOR (i, 0, SZ(g[x]))
		{
			int to = g[x][i];
			if (to == p || U[to]) continue;
			if (SZ[to] * 2 > szAll)
			{
				w = to;
				break;
			}
		}
		if (w == -1) break;
		p = x;
		x = w;
	}
	U[x] = true;
	// do something
	FOR (i, 0, SZ(g[x]))
	{
		int to = g[x][i];
		if (!U[to]) build(to);
	}
}
// 11. Dominator tree

vector<int> g[MAX];
vector<int> gr[MAX];
int Par[MAX];  // parent in dfs
bool U[MAX];
int P[MAX];    // parent in dsu
int V[MAX];    // vertex with min sdom in dsu
int SDOM[MAX]; // min vertex with alternate path
int DOM[MAX];  // immediate dominator
vector<int> BKT[MAX]; // vertices with this sdom
int tin[MAX];
int timer;
int n;
int find(int x)
{
	if (P[x] == x) return x;
	int y = find(P[x]);
	if (P[y] == y) return x; // don't consider root
	if (tin[SDOM[V[P[x]]]] < tin[SDOM[V[x]]])
								V[x] = V[P[x]];
	P[x] = y;
	return y;
}
int get(int x)
{
	find(x);
	return V[x]; // return vertex with min sdom
}

vector<int> ord;

void dfs(int x, int p)
{
	tin[x] = timer++;
	U[x] = true;
	ord.PB(x);
	Par[x] = p;
	FOR (i, 0, SZ(g[x]))
	{
		int to = g[x][i];
		if (U[to]) continue;
		dfs(to, x);
	}
}

void build(int s)
{
	FOR (i, 0, n)
	{
		U[i] = false;
		SDOM[i] = i;
		DOM[i] = -1;
		P[i] = i;
		V[i] = i;
		BKT[i].clear();
	}
	ord.clear();
	timer = 0;

	dfs(s, -1);

	RFOR(i, SZ(ord), 0)
	{
		int x = ord[i];
		FOR (j, 0, SZ(gr[x]))
		{
			int frm = gr[x][j];
			if (!U[frm]) continue; // don't consider unreachable vertices
			if (tin[SDOM[x]] > tin[SDOM[get(frm)]]) // find min sdom
			{
				SDOM[x] = SDOM[get(frm)];
			}
		}
		if (x != s) BKT[SDOM[x]].PB(x);
		FOR (j, 0, SZ(BKT[x]))
		{
			int y = BKT[x][j];
			int v = get(y);
			if (SDOM[y] == SDOM[v]) DOM[y] = SDOM[y]; // if sdoms equals
then this is dom
			else DOM[y] = v; // else we will find it later
		}

		if (Par[x] != -1) P[x] = Par[x]; // add vertex to dsu
	}

	FOR (i, 0, SZ(ord))
	{
		int x = ord[i];
		if (x == s) continue;
		if (DOM[x] == -1) continue;
		if (DOM[x] != SDOM[x]) DOM[x] = DOM[DOM[x]];
	}
}
```

```
// 12. Kirchhoff theorem

// Calculates number of different spanning trees
// -- Take adjacency matrix multiplied by -1
// -- Replace elements on main diagonal by degrees of vertices
// -- Remove any row and column with same parity
// -- Calculate determinant

// 13. Tutte matrix
```

$$\begin{pmatrix} 0 & x_{12} & x_{13} & \cdots & x_{1(n-1)} & x_{1n} \\ -x_{12} & 0 & x_{23} & \cdots & x_{2(n-1)} & x_{2n} \\ -x_{13} & -x_{23} & 0 & \cdots & x_{3(n-1)} & x_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -x_{1(n-1)} & -x_{2(n-1)} & -x_{3(n-1)} & \cdots & 0 & x_{(n-1)n} \\ -x_{1n} & -x_{2n} & -x_{3n} & \cdots & -x_{(n-1)n} & 0 \end{pmatrix}$$

// -- $x_{ij} := rand()$, if there is an edge between i and j.

```
// -- Determinant equals 0 iff there is no perfect matching

// 14. 3-SAT (O((4/3) ^ n) on average)

// Repeat following:
// -- Generate n random values
// -- Repeat n times:
// -- -- If all clauses are satisfied then OK
// -- -- Take any (e.g. first) unsatisfied clause and invert any of the 3
variables

// 16. MST in directed graph

// Repeat following:
// -- For each vertex find minimal incoming edge and for all incoming edges
subtract this value
// -- If edges with 0 value form DAG then they form mst
// -- Otherwise find any cycle compress it and repeat

// To restore mst:
// -- For each step store ids of edges that form cycle
// -- For each step and for each vertex store id of the component it is in
// -- Add edges from DAG that form mst to the answer
// -- For each cycle in reverse order:
// -- -- Let ID be the edge in current answer that goes to vertex #1 (component
with cycle on previous step) on current step
// -- -- Add all edges from the cycle except the one that goes to the same
component as ID on the previous step

// 17. Heavy-light decomposition

// to run:
// dfsSZ(0, -1, 0);
// dfsHLD(0, -1, 0);
// [tin[x], tout[x]] --- subtree of x
// [tin[TO[x]], tin[x]] --- path from TO[x] to x
VI g[MAX];     // graph
int SZ[MAX];   // size of the subtree
int H[MAX];    // height of the vertex
int P[MAX];    // parent of the vertex
int TO[MAX];   // top node of the heavy path
int tin[MAX];
int tout[MAX];
int timer = 0;
void dfsSZ(int x, int p, int h)
{
      SZ[x] = 1;
      H[x] = h;
      P[x] = p;
      FOR (i, 0, SZ(g[x]))
      {
            int to = g[x][i];
            if (to == p) continue;
            dfsSZ(to, x, h + 1);
            SZ[x] += SZ[to];
            if (g[x][0] == p || SZ[to] > SZ[g[x][0]])
                  swap(g[x][0], g[x][i]);
      }
}
void dfsHLD(int x, int p, int v)
{
      tin[x] = timer++;
      TO[x] = v;
      FOR (i, 0, SZ(g[x]))
      {
            int to = g[x][i];
            if (to == p) continue;
            if (i == 0) dfsHLD(to, x, v);
            else dfsHLD(to, x, to);
      }
      tout[x] = timer - 1;
}
int get(int x, int y) // query on path x-y
{
      int res = 0;
      while(true)
      {
            int tx = TO[x];
            int ty = TO[y];
            if (tx == ty) // same heavy path
            {
                  int t1 = tin[x];
                  int t2 = tin[y];
                  if (t1 > t2) swap(t1, t2);
                  if (t1 < t2)
                        res = max(res, R.get(t1 + 1, t2)); // lca not considered
                  break;
            }
            if (H[tx] < H[ty])
            {
                  swap(tx, ty);
                  swap(x, y);
            }
            res = max(res, R.get(tin[tx], tin[x]));
            x = P[tx];
      }
      return res;
}
```

```
// 18. Gauss algorithm

const double EPS = 1e-7;
double A[MAX][MAX]; // Input matrix  (n x m)
double B[MAX];      // Input vector  (n)
double X[MAX];      // Output values (m)
int P[MAX];         // -1 if a free variable; row index otherwise
// solves A * X = B
// returns:
//  - 0 if no solution
//  - 1 is one solution
//  - -(number of free variables) if multiple solutions
// Complexity O(N^2 * M)
// Beware of precision errors!!
int gauss(int n, int m)
{	int ind = 0;
	FOR (j, 0, m)
	{
		pair<double, int> mx = MP(-1e47, -1);
		FOR (i, ind, n)
			mx = max(mx, MP(abs(A[i][j]), i));
		if (mx.second == -1 || abs(mx.first) < EPS)
		{
			P[j] = -1;
			continue;
		}
		if (mx.second != ind)
		{
			int x = mx.second;
			FOR (i, j, m)
				swap(A[ind][i], A[x][i]);
			swap(B[ind], B[x]);
		}
		FOR (i, ind + 1, n)
		{
			double c = A[i][j] / A[ind][j];
			FOR (k, j, m)
				A[i][k] -= A[ind][k] * c;
			B[i] -= B[ind] * c;
		}
		P[j] = ind;
		ind++;
	}
	FOR (i, ind, n)
		if (abs(B[i]) > EPS) return 0;
	int res = 1;
	RFOR(j, m, 0)
	{
		if (P[j] == -1)
		{
			X[j] = 0;
			if (res == 1) res = 0;
			res--;
			continue;
		}
		int ind = P[j];
		double sum = B[ind];
		FOR (k, ind+1, m)
			sum -= A[ind][k] * X[k];
		sum /= A[ind][j];
		X[j] = sum;
	}
	return res;
}
```

```
// 19. Chinese reminder theorem

// X = A[i] (mod P[i])
// Complexity O(n^2).
LL Chinese()
{
	bool ok = true;
	FOR(j,1,N)
	{
		int a = 1;
		int b = 0;
		RFOR(k,j,0)
		{
			b = (b * P[k] + A[k]) % P[j];
			a = a * P[k] % P[j];
		}
		b = (A[j] - b + P[j]) % P[j];
		int c = P[j];
		int g = gcd(a , c);
		if (b % g != 0)
		{
			ok = false;
			break;
		}
		a /= g;
		b /= g;
		c /= g;
		b = b * bpow(a , Phi[c] - 1, c) % c;
		A[j] = b;
		P[j] = c;
	}
	if (ok)
	{
		LL res = A[N - 1];
		RFOR(j,N - 1,0)
		{
			res *= P[j];
			res += A[j];
		}
		return res;
	}
	else  cout << "No solution" << endl;
}
//if lcm(P[i]) < 10^18 can be done in O(n)
LL Chinese2()
{
	LL aa = P[0];
	LL bb = A[0];
	FOR(j,1,N)
	{
		int b = (A[j] - bb % P[j] + P[j]) % P[j];
		int a = aa % P[j];
		int c = P[j];
		int g = gcd(a , c);
		if (b % g != 0)
		{
			ok = 0;
			break;
		}
		a /= g;
		b /= g;
		c /= g;
		b = b * bpow(a , Phi[c] - 1, c) % c;
```

```
            bb = aa * b + bb;
            aa = aa * c;
      }
      if (ok)
      {
            LL res = bb;
            return res;
      }
      else  cout<<"No solutions"<<endl;
}
```

// 20. Mult long long modulo long long

```
LL mult(LL a, LL b, LL mod)
{
      LL k = (long double)a * b / mod;
      LL res = a * b - k * mod;
      if (res < 0 || res >= mod)
            res %= mod;
      if (res < 0) res += mod;
      return res;
}
```

// 21. Miller-Rabin test

```
bool MillerRabin(LL n, int k) //n > 3, n - odd
{
      LL d = n - 1;
      int s = 0;
      while (d % 2 == 0)
      {
            d /= 2;
            ++s;
      }
      FOR(it,0,k)
      {
            LL a = rand() % (n - 3) + 2; // use custom rand
            LL x = bpow(a, d, n); // == a ^ d % n
            if (x == 1 || x == n - 1) continue;
            bool ok = 0;
            FOR(i,0,s - 1)
            {
                  x = mult(x, x, n); // == x * x % n
                  if (x == 1) return 0;
                  if (x == n - 1)
                  {
                        ok = 1;
                        break;
                  }
            }
            if (!ok) return 0;
      }
      return 1;
}
```

// 22. Inverse of numbers 1..m-1 modulo m (m prime)

```
r[1] = 1;
FOR (i, 2, m)
      r[i] = (m - (m/i) * r[m%i] % m) % m;
```

// 23. Extended gcd

```
int gcd(int a, int b, int& x, int& y)
{
      int ax = 1, ay = 0;
      int bx = 0, by = 1;
      while (b)
      {
            int k = a / b;
            ax -= k * bx;
            ay -= k * by;
            a %= b;
            swap(a, b);
            swap(ax, bx);
            swap(ay, by);
      }
      x = ax;
      y = ay;
      return a;
}
```

// 24. Segments with equal n/x

```
void segms()
{
      LL x = 1;
      LL prev = 1;
      while (x <= n)
      {
            LL y = n / x;
            x = n / y + 1;
            cout << prev << ' ' << x - 1 << endl;
                                          // segment [prev , x - 1]
            prev = x;
      }
}
```

// 25. Golden ration (ternary search)

```
const double phi = (3. - sqrt(5.0)) / 2.;
double get(double L, double R)
{
      double M1, M2, v1, v2;
      M1 = L + (R - L) * phi;
      M2 = R - (R - L) * phi;
      v1 = get(M1);
      v2 = get(M2);
      FOR (it, 0, 80)
      {
            if (v1 > v2) // for minimum
            {
                  L = M1;
                  M1 = M2;
                  v1 = v2;
                  M2 = R - (R - L) * phi;
                  v2 = get(M2);
            }else{
                  R = M2;
                  M2 = M1;
                  v2 = v1;
                  M1 = L + (R - L) * phi;
                  v1 = get(M1);
            }
      }
      return L; // or F(L)
}
```

```
// 26. Recurrence in O(k^2 log k)

int k;      // number of elements
VI C;       // coefficients of recurrence
VI T;       // initial values (k items)
// T_i = T_(i-k)*C_0 + T_(i-k+1)*C_1 + ... + T_(i-1)*C_(k-1)
VI mult(VI a, VI b)
{
      VI v(SZ(a) + SZ(b));
      FOR(i, 0, SZ(a))
      FOR(j, 0, SZ(b))
            v[i + j] = (v[i + j] + a[i] * (LL)b[j]) % MOD;
      while (SZ(v) > k)
      {
            FOR(j,0,k)
                  v[SZ(v) - 2 - j] = (v[SZ(v) - 2 - j] + v.back() * (LL)C[k - 1
- j]) % MOD;
            v.pop_back();
      }
      return v;
}
int get(int n)
{
      VI A(k);
      A[1] = 1;
      VI res(k);
      res[0] = 1;
      while (n)
      {
            if (n & 1)
                  res = mult(res , A);
            A = mult(A , A);
            n /= 2;
      }
      int val = 0;
      FOR(i,0,k)
            val = (val + T[i] * (LL)res[i]) % MOD;
      return val;
}
```

// 28. Polya's theorem

$$ClassesCount = \frac{1}{|G|}\sum_{\pi\epsilon G} k^{C(\pi)}$$

$$ClassesCount = \frac{1}{|G|}\sum_{\pi\epsilon G} f\big(C(\pi)\big)$$

```
// C(pi) --- number of cycles in permutation
// f(C(pi)) --- number of ways to paint elements such that elements
// of each cycle of the permutation are painted in the same color
```

// 29. Mobius inversion formulas

```
int mn[MAX];
void calcmn(int n)
{
    mn[1] = 1;
    FOR (i,1,n)
    {
        if (!mn[i]) continue;
        for(int j = 2*i; j<n; j+=i)
        {
            mn[j] -= mn[i];
        }
    }
}
```

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

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

$$M(n) \equiv \sum_{k=1}^{n} \mu(k), \quad \sum_{n=1}^{x} M\left(\frac{x}{n}\right) = 1$$

// 30. Catalan numbers formulas

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

$$C_n = \frac{1}{n+1} C_{2n}^n$$

$$C_n = C_{2n}^n - C_{2n}^{n-1}$$

// 31. Binomial coefficients formulas

$$\sum_{k=0}^{n} C_n^k = 2^n$$

$$\sum_{m=0}^{n} C_m^k = C_{n+1}^{k+1}$$

$$\sum_{k=0}^{m} C_{n+k}^k = c_{n+m+1}^m$$

$$(C_n^0)^2 + (c_n^1)^2 + \cdots + (c_n^n)^2 = C_{2n}^n$$

$$1C_n^1 + 2C_n^2 + \cdots + nC_n^n = n2^{n-1}$$

$$C_n^0 + C_{n-1}^1 + \cdots + C_{n-k}^k + \cdots + C_0^n = F_{n+1}$$

// 32. Fibonacci numbers formulas

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

$$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$$

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

$$\gcd(F_m, F_n) = F_{\gcd(m,n)}$$

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

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

// 33. Stirling numbers

$$x^n = \sum_{k=0}^{n} S(n,k) * (x)_k$$

$$S(n,n) = 1, \quad n \geq 0$$

$$S(n,0) = 0, \quad n > 0$$

$$S(n,k) = S(n-1,k-1) + S(n-1,k) * k$$

$$(x)_n = x(x-1)(x-2)\ldots(x-n+1)$$

```
First Bell numbers (starting from 0):
1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437,
190899322, 1382958545, ...
```

// 34. Euler formula

```
// if (a, m) != 1:
//              | a ^ b mod m, if b < phi(m)
// a ^ b mod m=|
//              | a ^ (phi(m) + b mod phi(m)) mod m, if b >= phi(m)
```

// 35. Simpson integration

$$\int_a^b f(x)dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

// 36. Hook length formula

$$d_\lambda = \frac{n!}{\Pi h_\lambda(i,j)}$$

$h_\lambda(i,j)$ --- number of cells under or right to the cell (i, j) in Young diagram including itself

// 37. Generators

```
// g is a generator for modulo m if the set {g^0, g^1, ..., g^(phi(m)-1)}
//          equals to the set of all numbers coprime with m
// generator exists for:
// -- m = p ^ k (p -- odd prime, k >= 1)
// -- m = 2 * p ^ k (p -- prime, k >= 1)
// -- m = 1, 2, 4
// To find generator:
// -- find phi(m) and p_1, p_2, ..., p_k -- prime factors of phi(m)
// -- find g such that g ^ (phi(m) / p_j) != 1 for each prime factor
// -- check g = 2, 3, 4, ..., p-2, p-1
```

// 38. Some integer sequences

```
// Arithmetic progression:
// -- a_n = a_(n-1) + d
// -- a_n = a_1 + d * (n - 1)
// -- S_n = (a_1 + a_n) * n / 2
// Geometric progression:
// -- b_n = b_(n-1) * d
// -- b_n = b_1 * d ^ (n-1)
// -- S_n = b_1 * (1 - d ^ n) / (1 - d)
// -- if |d| < 1:
// -- -- S_inf = b_1 / (1 - d)
// Sum of squares of natural numbers:
// -- sum[i = 1..n](i^2) = n*(n+1)*(2*n+1)/6
// Sum of cubes of natural numbers:
// -- sum[i = 1..n](i^3) = n^2 * (n+1)^2 / 4
```

// 39. Walsh-Hadamard transform

```
// to multiply two polynomes having x^i * x^j = x^(i xor j)
// conv_xor(a);
// conv_xor(b);
// a[i] *= b[i];
// conv_xor(a);
// a[i] /= n;
void conv_xor(VI& a, int k)
{
      FOR (i, 0, k)
            FOR (j, 0, 1<<k)
                  if ((j & (1<<i)) == 0)
                  {
                        int u = a[j];
                        int v = a[j + (1<<i)];
                        a[j] = u + v;
                        a[j + (1<<i)] = u - v;
                  }
}
```

```
// to multiply two polynomes having x^i * x^j = x^(i or j)
// conv_or(a);
// conv_or(b);
// a[i] *= b[i];
// inverse_or(a);
void conv_or(VI& a, int k)
{
      FOR (i, 0, k)
            FOR (j, 0, 1<<k)
                  if ((j & (1<<i)) == 0)
                        a[j + (1<<i)] += a[j];
}
void inverse_or(VI& a, int k)
{
      FOR (i, 0, k)
            FOR (j, 0, 1<<k)
                  if ((j & (1<<i)) == 0)
                        a[j + (1<<i)] -= a[j];
}
// to multiply two polynomes having x^i * x^j = x^(i and j)
// conv_and(a);
// conv_and(b);
// a[i] *= b[i];
// inverse_and(a);
void conv_and(VI& a, int k)
{
      FOR (i, 0, k)
            FOR (j, 0, 1<<k)
                  if ((j & (1<<i)) == 0)
                        a[j] += a[j + (1<<i)];
}
void inverse_and(VI& a, int k)
{
      FOR (i, 0, k)
            FOR (j, 0, 1<<k)
                  if ((j & (1<<i)) == 0)
                        a[j] -= a[j + (1<<i)];
}
```

// 40. Newton interpolation

```
double X[MAX]; // interpolation nodes
double Y[MAX]; // function values in corresponding nodes
double D[MAX][MAX];
int n; // number of interpolation nodes
void addPt(double pt, double val)
{
      X[n]=pt;
      Y[n]=val;
      D[0][n] = Y[n];
      FOR(i,1,n+1)
      {
            int j = n-i;
            D[i][j] = (D[i-1][j+1]-D[i-1][j]) / (X[j+i]-X[j]);
      }
      ++n;
}
```

```
double Newton(double x)
{
      double res = 0;
      double mul = 1;
      FOR(i,0,n)
      {
            res += D[i][0] * mul;
            mul *= x-X[i];
      }
      return res;
}
```

// 41. Simplex method

```
// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
// OUTPUT: value of the optimal solution (infinity if
//         unbounded above, nan if infeasible)
typedef long double LD;
typedef vector<LD> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
const LD EPS = 1e-12;
struct LPSolver
{
    int m, n;
    VI B, N;
    VVD D;
    LPSolver(const VVD &A, const VD &b, const VD &c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
        FOR (i,0,m) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
        FOR (i,0,m) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i]; }
        FOR (j,0,n) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m + 1][n] = 1;
    }
    void Pivot(int r, int s) {
        double inv = 1.0 / D[r][s];
        FOR (i,0,m+2) if (i != r)
            FOR (j,0,n+2) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] * inv;
        for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
        for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv;
        swap(B[r], N[s]);
    }

    bool Simplex(int phase) {
        int x = phase == 1 ? m + 1 : m;
        while (true) {
            int s = -1;
            FOR (j,0,n+1) {
                if (phase == 2 && N[j] == -1) continue;
                if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] <
N[s]) s = j;
            }
            if (D[x][s] > -EPS) return true;
            int r = -1;
            FOR (i,0,m) {
                if (D[i][s] < EPS) continue;
                if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
                    (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] <
B[r]) r = i;
            }
            if (r == -1) return false;
            Pivot(r, s);
        }
    }
    LD Solve(VD &x) {
        int r = 0;
        FOR (i,1,m) if (D[i][n + 1] < D[r][n + 1]) r = i;
        if (D[r][n + 1] < -EPS) {
            Pivot(r, n);
            if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -
numeric_limits<LD>::infinity();
            FOR (i,0,m) if (B[i] == -1) {
                int s = -1;
                FOR (j,0,n+1)
                if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j] <
N[s]) s = j;
                Pivot(i, s);
            }
        }
        if (!Simplex(2)) return numeric_limits<LD>::infinity();
        x = VD(n);
        FOR (i,0,m) if (B[i] < n) x[B[i]] = D[i][n + 1];
        return D[m][n + 1];
    }
};
LPSolver solver(A, b, c);
VD x;
LD value = solver.Solve(x);
```

// 42. Segment tree with adding on range

```
// assignment on segment
// sum on segment
struct RMQ
{
      LL A[MAX * 4]; // sum
      LL P[MAX * 4]; // push
      int n;
      void pull(int v) // combine values of two vertices to parent
      {
            A[v] = A[v*2] + A[v*2+1];
      }
      void build(int tl, int tr, int v, LL* a)
      {
            P[v] = -1;
            if (tl == tr)
            {
                  A[v] = a[tl];
                  return;
            }
            int tm = (tl + tr) / 2;
            build(tl, tm, v*2, a);
            build(tm+1, tr, v*2+1, a);
            pull(v);
      }
```

```
	void init(int n, LL* a)
	{
		this->n = n;
		build(0, n-1, 1, a);
	}
	void upd(int tl, int tr, int v, LL val) // update vertex with value
	{
		A[v] = val * (LL)(tr - tl + 1);
		P[v] = val;
	}
	void push(int tl, int tr, int v) // push changes to children
	{
		if (tl == tr || P[v] == -1) return;
		int tm = (tl + tr) / 2;
		upd(tl, tm, v*2, P[v]);
		upd(tm + 1, tr, v*2+1, P[v]);
		P[v] = -1;
	}
	void add(int tl, int tr, int v, int l, int r, LL val)
	{
		if (l > r) return;
		push(tl, tr, v);
		if (l == tl && r == tr)
		{
			upd(tl, tr, v, val);
			return;
		}
		int tm = (tl + tr) / 2;
		add(tl, tm, v*2, l, min(tm, r), val);
		add(tm+1, tr, v*2+1, max(tm+1, l), r, val);
		pull(v);
	}
	void add(int l, int r, LL val)
	{
		add(0, n-1, 1, l, r, val);
	}
	LL get(int tl, int tr, int v, int l, int r)
	{
		if (l > r) return 0;
		push(tl, tr, v);
		if (l == tl && r == tr)
			return A[v];
		int tm = (tl + tr) / 2;
		LL res = get(tl, tm, v*2, l, min(tm, r));
		res += get(tm+1, tr, v*2+1, max(tm+1, l), r);
		return res;
	}
	LL get(int l, int r)
	{
		return get(0, n-1, 1, l, r);
	}
} R;
```

// 43. Set with count on segment

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
// example: ordered_set s; s.insert(47);
// s.order_of_key(k); -- returns number of elements less then k
// s.find_by_order(k); - returns iterator to k-th element or s.end()
```

// 44. Cartesian tree with push

```
int mrand()
{
	return abs((rand() << 16) ^ rand());
}
struct node
{
	int l, r;	// children
	int y;		// priority (should be random and different)
	int cnt;	// size of subtree
	int par;	// parent of the vertex
	int val;	// value of the vertex
	int rev;	// reverse push
	int mn;		// minimum of subtree

void init(int val) // init with value
	{
		l = r = -1;
		y = mrand();
		cnt = 1;
		par = -1;
		this->val = val;
		mn = val;
		rev = 0;
	}
};
// Minimum on subtree + reverse
struct Treap
{
	node A[MAX];
	int sz;
	int getCnt(int x)
	{
		if (x == -1) return 0;
		return A[x].cnt;
	}
	int getMn(int x)
	{
		if (x == -1) return INF;
		return A[x].mn;
	}
	int newNode(int val)
	{
		A[sz].init(val);
		sz++;
		return sz - 1;
	}
	int PB(int root, int val)
	{
		return merge(root, newNode(val));
	}
	void upd(int x)
	{
		if (x == -1) return;
		A[x].cnt = getCnt(A[x].l) + getCnt(A[x].r) + 1;
	A[x].mn = min(A[x].val, min(getMn(A[x].l), getMn(A[x].r)));
	}
	void reverse(int x)
	{
		if (x == -1) return;
		swap(A[x].l, A[x].r);
		A[x].rev ^= 1;
	}
```

```
	void push(int x)
	{
		if (x == -1 || A[x].rev == 0) return;
		reverse(A[x].l);
		reverse(A[x].r);
		A[x].rev = 0;
	}
	PII split(int x, int cnt)
	{
		if (x == -1) return MP(-1, -1);
		if (cnt == 0) return MP(-1, x);
		push(x);
		int left = getCnt(A[x].l);
		PII res;
		if (left >= cnt)
		{
			if (A[x].l != -1) A[A[x].l].par = -1;
			res = split(A[x].l, cnt);
			A[x].l = res.second;
			if (res.second != -1) A[res.second].par = x;
			res.second = x;
		}
		else
		{
			if (A[x].r != -1) A[A[x].r].par = -1;
			res = split(A[x].r, cnt - left - 1);
			A[x].r = res.first;
			if (res.first != -1) A[res.first].par = x;
			res.first = x;
		}
		upd(x);
		return res;
	}
	int merge(int x, int y)
	{
		if (x == -1) return y;
		if (y == -1) return x;
		int res;
		//if (mrand() % (getCnt(x) + getCnt(y)) < getCnt(x))
		if (A[x].y > A[y].y)
		{
			push(x);
			if (A[x].r != -1) A[A[x].r].par = -1;
			res = merge(A[x].r, y);
			A[x].r = res;
			if (res != -1) A[res].par = x;
			res = x;
		}
		else
		{
			push(y);
			if (A[y].l != -1) A[A[y].l].par = -1;
			res = merge(x, A[y].l);
			A[y].l = res;
			if (res != -1) A[res].par = y;
			res = y;
		}
		upd(res);
		return res;
	}} T;
```

```
 // 45. Fenwick tree

struct Fen
{
	int A[MAX];
	int n;
	void init(int n)
	{
		this->n = n;
	}
	void add(int x, int val)
	{
		for (; x < n; x = x | (x + 1)) // ascending
			A[x] += val;
	}
	int get(int x)
	{
		int res = 0;
		for (; x >= 0; x = (x & (x + 1)) - 1) // descending
			res += A[x];
		return res;
	}
};
// Fenwick tree:
// A_i = sum {j = F(i)..i} V_j
// F(i) = i & (i+1) --- removes the last block of ones in i
//
// 7        |
// 6       ||
// 5      | |
// 4     || |
// 3    |   |
// 2   ||   |
// 1  | |   |
// 0 || |   |
//   01234567
// Descent on Fenwick tree:
//  - Consider bits in order from largest to smallest.
//  - For each bit determine whether is should be set in the answer or not.
//  - Use single array cell for each check.

// Fenwick tree for minimum on segment:
//  - Use two Fenwick trees with n = 2^k
//  - One tree for normal array and one for reversed
//  - When querying for minimum on the segment only consider
//    segments from trees that are COMPLETELY inside the segment

// Fenwick tree for adding on segment (prefixes)
//  - Use two Fenwick trees:
//  -- F#1: for the actual sums on segments
//  -- F#2: for values that should be added to all elements of the segment
//  - To add value on prefix:
//  -- Add value*(R-F(i)+1) to F#1 to all segments that contain this prefix (use
code for ascending)
//  -- Add value to F#2 to segments that cover the prefix (use code for
descending)
//  - To get sum on prefix:
//  -- Sum all values from F#1 that cover the prefix (use code for descending)
//  -- Sum all values*(i-F(i)+1) from F#2 that cover the prefix (use code for
descending)
//  -- Sum all values*(R-F(i)+1) from F#2 from all segments that coves the
prefix (use code for ascending)
```

```
// 46. Suffix automaton

// To find number of occurrences of each class:
// -- Set cnt=1 if node is created
// -- Set cnt=0 if node is cloned
// -- Sum up  cnt[link(v)] += cnt[v] in reverse topsort order.

struct node
{
   int g[26];
   int link, len;
   void init()
   {
         FILL(g, -1);
         link = len = -1;
   }
};
struct automaton
{
    node A[MAX * 2];
    int sz, head;
    void init()
    {
        sz = 1; head = 0;
        A[0].init();
    }
    void add(char ch)
    {
        ch = ch - 'a';
        int nhead = sz++;
        A[nhead].init();
        A[nhead].len = A[head].len + 1;
        int cur = head;
        head = nhead;
        while(cur != -1 && A[cur].g[ch] == -1)
        {
            A[cur].g[ch] = head;
            cur = A[cur].link;
        }
        if (cur == -1)
        {
            A[head].link = 0;
            return ;
        }
        int p = A[cur].g[ch];
        if (A[p].len == A[cur].len + 1)
        {
            A[head].link = p;
            return ;
        }
        int q = sz++;
        A[q] = A[p];
        A[q].len = A[cur].len + 1;
        A[p].link = A[head].link = q;
        while(cur != -1 && A[cur].g[ch] == p)
        {
            A[cur].g[ch] = q;
            cur = A[cur].link;
        }
    }
};
```

```
// 47. Suffix array

const int MAX = 100100;
const int LEN = 18;
const int ALP = 128; // size of the alphabet. 128 for all chars in ASCII order
char S[MAX];     // input string
int Q[MAX];      // indexes for each class
int C[LEN][MAX]; // classes of equivalence
int P[LEN][MAX]; // permutations
int CNT[MAX];   // number of occurrences of each equivalence class
void buildArray(int n)
{
      FOR (i, 0, n)
            CNT[(int)S[i]]++;
      int sum = 0;
      FOR (i, 0, ALP)
      {
            Q[i] = sum;
            sum += CNT[i];
      }
      FOR (i, 0, n)
      {
            P[0][Q[(int)S[i]]] = i;
            Q[(int)S[i]]++;
      }
      C[0][P[0][0]] = 0;
      FOR (i, 1, n)
      {
            C[0][P[0][i]] = C[0][P[0][i-1]];
            if (S[P[0][i]] != S[P[0][i-1]]) C[0][P[0][i]]++;
      }
      FOR (it, 1, LEN)
      {
            int* Ccur = C[it];
            int* Cprev = C[it-1];
            int* Pcur = P[it];
            int* Pprev = P[it-1];
            int len = (1<<(it - 1));
            if (len >= n)
            {
                  FOR (i, 0, n)
                  {
                        Ccur[i] = Cprev[i];
                        Pcur[i] = Pprev[i];
                  }
                  continue;
            }
            FOR (i, 0, n)
                  CNT[i] = 0;
            FOR (i, 0, n)
                  CNT[Cprev[i]]++;
            int sum = 0;
            FOR (i, 0, n)
            {
                  Q[i] = sum;
                  sum += CNT[i];
            }
            FOR (i, 0, n)
            {
                  int cur = Pprev[i];
                  int prev = cur - len;
                  if (prev < 0) prev += n;
                  Pcur[Q[Cprev[prev]]++] = prev;
            }
```

```
		Ccur[Pcur[0]] = 0;
		FOR (i, 1, n)
		{
			int cur = Pcur[i];
			int prev = Pcur[i-1];
			Ccur[cur] = Ccur[prev];
			if (Cprev[cur] != Cprev[prev])
			{
				Ccur[cur]++;
				continue;
			}
			int cc = cur;
			cur += len;
			if (cur >= n) cur -= n;
			prev += len;
			if (prev >= n) cur -= n;
			if (Cprev[cur] != Cprev[prev])
			{
				Ccur[cc]++;
				continue;
			}
		}
	}
}
```

// 48. Aho-corasick

```
const int MAX = 100100; // total length of all strings +1
const int AL = 30;            // size of alphabet
struct node
{
	int p;              // parent node
	int c;              // character on incoming edge
	int g[AL];   // go on automaton
	int nxt[AL]; // go on bor
	int link;    // node corresponding to longest suffix
	void init()
	{
		c = -1;
		p = -1;
		FILL(g, -1);
		FILL(nxt, -1);
		link = -1;
	}
};
struct AC
{
	node A[MAX];
	int sz;
	void init()
	{
		A[0].init();
		sz = 1;
	}
	void addStr(string& s)
	{
		int x = 0;
		FOR (i, 0, SZ(s))
		{
			int c = s[i] - 'a';
			if (A[x].nxt[c] == -1)
			{
				A[x].nxt[c] = sz;
				A[sz].init();
				A[sz].c = c;
				A[sz].p = x;
				sz++;
			}
			x = A[x].nxt[c];
		}
	}
	int go(int x,int c)
	{
		if (A[x].g[c] == -1)
		{
			if (A[x].nxt[c] != -1)
				A[x].g[c] = A[x].nxt[c];
			else if (x != 0)
				A[x].g[c] = go(getLink(x), c);
			else
				A[x].g[c] = 0;
		}
		return A[x].g[c];
	}
	int getLink(int x)
	{
		if (A[x].link == -1)
		{
			if (x == 0 || A[x].p == 0) return 0;
			A[x].link = go(getLink(A[x].p), A[x].c);
		}
		return A[x].link;
	}
};
```

// 49. Pi function

```
void Pi(string& S)
{
	P[0] = 0;
	FOR (i, 1, SZ(S))
	{
		int j = P[i-1];
		while(j != 0 && S[i] != S[j]) j = P[j-1];

		if (S[i] == S[j]) j++;
		P[i] = j;
	}
}
```

// 50. Z-function

```
void Zf(string& s)
{
	int n = SZ(s);

	for(int i=1,l=0,r=0;i<n;i++)
	{
		Z[i] = 0;
		if(i<=r)
			Z[i] = min(r-i+1,Z[i-l]);

		while(i+Z[i]<n && s[i+Z[i]] == s[Z[i]]) ++Z[i];
		if(i+Z[i]-1>r)
		{
			r=i+Z[i]-1;
			l=i;
		}
	}
}
```

```
 // 51. Manaker algorithm

int d1[MAX],d2[MAX];
void manaker(string s)
{
    int n = SZ(s);
    for(int i=0,l=-1,r=-1;i<n;i++)
    {
        if(i<=r)
            d1[i] = min(r-i+1,d1[l+(r-i)]);
        while(i+d1[i]<n && i-d1[i]>=0 && s[i+d1[i]]==s[i-d1[i]])++d1[i];
        if(i+d1[i]-1>r)
        {
            r=i+d1[i]-1;
            l=i-(d1[i]-1);
        }
    }
    for(int i=0,l=-1,r=-1;i<n;i++)
    {
        if(i<=r)
            d2[i] = min(r-i+1,d2[l+(r-i)+1]);
        while(i+d2[i]<n && i-(d2[i]+1)>=0 && s[i+d2[i]] == s[i-
(d2[i]+1)])++d2[i];
        if(i+d2[i]>r)
        {
            r = i+d2[i]-1;
            l = i-d2[i];
        }
    }
}

 // 52. Minimal cyclic shift

 // -- Double string B
 // -- After execution s is a minimal cyclic shift
int s = 0;
FOR (i, 1, m)
{
      int j = F[i-1-s];
      while (j > 0 && B[s+j] != B[i])
      {
            if (B[s+j] > B[i])
                  s = i-j;
            j = F[j-1];
      }
      if (j == 0 && B[s] != B[i])
            if (B[s] > B[i])
                  s = i;
      else
            ++ j;
      F[i-s] = j;
}

// 53. Palindromic tree
struct node
{
      int to[2]; // size of the alphabet (can be changed to map)
      int link;
      int len;
      void clear()
      {     FILL(to, -1);
            link = -1;
            len = -1;
      }
};
char S[MAX]; // the string
struct PalTree
{     node A[MAX];
      int sz;
      int last;
      void init()
      {     A[0].clear(); // root of odd-length palindromes
            A[1].clear(); // root of even-length palindromes
            A[1].len = 0;
            A[1].link = 0;
            sz = 2;
            last = 1;
      }
      void add(int ind)
      {     int cur = last;
            while(cur != -1)
            {
                  int pos = ind - A[cur].len - 1;
                  if (pos >= 0 && S[pos] == S[ind]) break;
                  cur = A[cur].link;
            }
            if (cur == -1) throw -1;
            if (A[cur].to[S[ind] - 'a'] == -1)
            {
                  A[cur].to[S[ind] - 'a'] = sz;
                  A[sz].clear();
                  A[sz].len = A[cur].len + 2;
                  int link = A[cur].link;
                  while(link != -1)
                  {
                        int pos = ind - A[link].len - 1;
                        if (pos >= 0 && S[pos] == S[ind]) break;
                        link = A[link].link;
                  }
                  if (link == -1) link = 1;
                  else link = A[link].to[S[ind] - 'a'];
                  A[sz].link = link;
                  sz++;
            }
            last = A[cur].to[S[ind] - 'a'];
      }
} PT;

 // 54. Two closest points

// Iterate over points in order of increasing of X-coordinate:
// -- Let D be the best distance so far, (x, y) be the current point
// -- Insert the point into the set that stores points by Y-coordinate
// -- Iterate over all points with Y in range [y - D, y + D] and update answer
(use single lower_bound)
// -- Remove all points (x', y') that have x' < x - D
// -- Maintain pointer to the point that should be removed next in the array of
points sorted by X-coordinate
```

```
// 55. Geometry

struct point
{
    double x, y;
    point() {}
    point(double x, double y) : x(x), y(y) {};
    point operator-(const point& p)const
    {
        return point (x - p.x, y - p.y);
    }
    point operator+(const point& p)const
    {
        return point (x + p.x, y + p.y);
    }
    double operator*(const point & p) const
    {
        return x * p.y - y * p.x;
    }
    point operator*(double k) const
    {
        return point(k * x, k * y);
    }
    double d2() const
    {
        return x * x + y * y;
    }
    double len() const
    {
        return sqrt(d2());
    }
    bool operator==(const point & p) const
    {
        return abs(x - p.x) < EPS && abs(y - p.y) < EPS;
    }
    bool operator<(const point & p) const
    {
        if (abs(x - p.x) > EPS)return x < p.x;
        if (abs(y - p.y) > EPS)return y < p.y;
        return 0;
    }
    point rotate(double cosx, double sinx) const //ccw
    {
        double xx = x * cosx - y * sinx;
        double yy = x * sinx + y * cosx;
        return point(xx, yy);
    }
    point rotate(double ang) const //ccw
    {
        return rotate(cos(ang), sin(ang));
    }
    point scale(double l) const  //assuming len of vector > 0
    {
        l /= len();
        return point(l * x, l * y);
    }
    double dot(const point& p) const
    {
        return x * p.x + y * p.y;
    }
    double polar() const // (-PI; PI]
    {
        double ang = atan2(y, x);
        //if (ang < -EPS)ang += 2 * PI; // if need [0; 2 * PI)
        return ang;
    }
    int hp() const //halfpalne relative to X-axis
    {
        return y < -EPS || (abs(y) < EPS && x < -EPS);
    }
};

bool cmpVec(const point& a, const point& b) //sort by polar angle [0; 2*PI)
{
    if (a.hp() != b.hp())return a.hp() < b.hp();
    return a * b > EPS;
}
int sign(double x)
{
    if (abs(x) < EPS)return 0;
    return x > 0 ? 1 : -1;
}

struct line
{
    point n;
    double c;
    line() {}
    line(double a, double b, double c)
    {
        n = point(a, b);
        this->c = c;
    }
    line(point a, point b) // assuming a != b
    {
        double A = b.y - a.y;
        double B = a.x - b.x;
        double C = -a.x * A - a.y * B;
        n = point(A, B);
        c = C;
    }
    double dist(const point & p) const //oriented
    {
        return (n.dot(p) + c) / n.len();
    }
    point clothestPoint(const point& p) const
    {
        return p + n.scale(-dist(p));
    }
    bool paralel(const line& l) const
    {
        return abs(n * l.n) < EPS;
    }
    point intersect(const line &l) const
    {
        //assuming that lines are not parallel
        double z = n * l.n;
        double x = - (c * l.n.y - n.y * l.c) / z;
        double y = - (n.x * l.c - c * l.n.x) / z;
        return point(x, y);
    }
};

struct segment
{
    point a, b;
    segment() {}
    segment(point a, point b)
```

```
    {
        this -> a = a;
        this -> b = b;
    }
    line getLine() const
    {
        return line(a, b);
    }
    bool contains(const point& p) const
    {
        return abs((a - b).len() - (a - p).len() - (b - p).len()) < EPS;
    }
    bool intersect(const segment& s) const
    {
        if (min(s.a.x, s.b.x) > max(a.x, b.x))return false;
        if (min(s.a.y, s.b.y) > max(a.y, b.y))return false;
        if (max(s.a.x, s.b.x) < min(a.x, b.x))return false;
        if (max(s.a.y, s.b.y) < min(a.y, b.y))return false;

        int s1 = sign((a - s.a) * (b - s.a));
        int s2 = sign((a - s.b) * (b - s.b));
        int s3 = sign((s.a - a) * (s.b - a));
        int s4 = sign((s.a - b) * (s.b - b));

        return s1 * s2 <= 0 && s3 * s4 <= 0;
    }
    double dist(const point& p) const //not oriented
    {
        point q = getLine().clothestPoint(p);
        if (contains(q))return (p-q).len();
        return min((a-p).len(), (b-p).len());
    }
    double dist(const segment& s) const
    {
        if (intersect(s))return 0;
        double ans = min(dist(s.a), dist(s.b));
        ans = min(ans, s.dist(a));
        ans = min(ans, s.dist(b));
        return ans;
    }
};

bool triangleContains(const point& a, const point& b, const point& c, const
point& p)
{
    int s1 = sign((b - a) * (p - a));
    int s2 = sign((c - b) * (p - b));
    int s3 = sign((a - c) * (p - c));
    return (s1 >= 0 && s2 >= 0 && s3 >= 0) || (s1 <= 0 && s2 <= 0 && s3 <= 0);
}

struct polygon
{
    vector<point> p;
    polygon() {}
    polygon(const vector<point> &a)
    {
        p = a;
        if (SZ(a))p.PB(a[0]);
    }
    int sz() const
    {
        return max(SZ(p) - 1, 0);
    }
    polygon convex() const //returns ccw-ordered
    {
        vector<point> pp = p;
        if (SZ(pp))pp.pop_back();
        sort(ALL(pp));
        vector<point> U, D;
        FOR(i, 0, SZ(pp))
        {
            while(SZ(D) > 1 && sign((D.back() - D[SZ(D) - 2]) * (pp[i] - D[SZ(D)
- 2])) <= 0)D.pop_back();
            while(SZ(U) > 1 && sign((U.back() - U[SZ(U) - 2]) * (pp[i] - U[SZ(U)
- 2])) >= 0)U.pop_back();
            U.PB(pp[i]);
            D.PB(pp[i]);
        }
        reverse(ALL(U));
        FOR(i, 1, SZ(U)-1)
        D.PB(U[i]);
        return polygon(D);
    }
    //randomised, could be used for not convex polygons
    bool contains(const point& x) const
    {
        double MX = sqrt(3) * PI / 47.7 + 123.23424;
        double MY = sqrt(2) * acos(0.47747) + 4 * PI;
        point v = point(MX, MY).scale(1e8); //v should be strictly outside
polygon;
        segment S = segment(x, v);
        int cnt = 0;
        FOR(i, 0, SZ(p)-1)
        {
            segment seg = segment(p[i], p[i+1]);
            if (seg.contains(x))return 1;
            if (seg.intersect(S))cnt++;
        }
        return cnt % 2;
    }
    //only for convex polygons
    //requires ccw-order
    //duplicated points are not allowed
    bool contains2(const point& q) const
    {
        if (!sz())return false;
        if (sz() == 1)return p[0] == q;
        int l = 1, r = sz()-1;
        int s1 = sign((p[l] - p[0]) * (q - p[0]));
        int s2 = sign((p[r] - p[0]) * (q - p[0]));
        if (s1 == -1)return 0;
        if (s2 == 1)return 0;
        while(r - l > 1)
        {
            int m = (l + r) / 2;
            int s = sign((p[m] - p[0]) * (q - p[0]));
            if (s <= 0)r = m;
            else l = m;
        }
        return triangleContains(p[0], p[l], p[r], q);
    }
    int _minVertex() const
    {
        //assuming point coords are integers, otherwise some EPSs should be
added
        int id = 0;
        FOR(i, 1, sz())
```

```
        if (p[i].y < p[id].y || (p[i].y == p[id].y && p[i].x < p[id].x))
            id = i;
        return id;
    }
    //assuming both polygons are convex and ccw-ordered
    polygon minkowskySum(const polygon& a) const
    {
        int i = _minVertex();
        int j = a._minVertex();
        int n = sz(), m = a.sz();
        vector<point> res;
        if (sz() == 0 || a.sz() == 0)return res;
        int ci = 0, cj = 0;
        while(ci < n || cj < m)
        {
            res.PB(p[i] + a.p[j]);
            int ii = i == n - 1 ? 0 : i + 1;
            int jj = j == m - 1 ? 0 : j + 1;
            if (ci == n)
            {
                j = jj, cj++;
                continue;
            }
            if (cj == m)
            {
                i = ii, ci++;
                continue;
            }
            if (cmpVec(p[ii] - p[i], a.p[jj] - a.p[j]))i = ii, ci++;
            else j = jj, cj++;
        }
        return res;
    }

    //TANGENTS TO CONVEX POLYGON
    //No three points should be collinear
    //Polygon should be convex and given in ccw order
    //Polygon should contain at least 3 vertices
    bool _visible(int idx, const point& x) const
    {
        if (idx >= sz())idx -= sz();
        return (p[idx] - x) * (p[idx+1] - x) < 0; //change to <= if colinear
edges should be visible
    }
    int _findOppositeToFirst(const point& x) const
    {
        int v = _visible(0, x);
        int l = 1, r = sz() - 1;
        int s1 = sign((p[l] - x) * (p[0] - x));
        int s2 = sign((p[r] - x) * (p[0] - x));
        if (s1 * s2 >= 0)
        {
            if (_visible(l, x) != v)return l;
            if (_visible(r, x) != v)return r;
            if (_visible(l + 1, x) != v)return l + 1;
            if (_visible(r - 1, x) != v)return r - 1;
            return -1;
        }
        while(r - l > 1)
        {
            int m = (l + r) / 2;
            if (sign((p[m] - x) * (p[0] - x)) == s1)l = m;
            else r = m;
        }
```

```
        if (_visible(l, x) == v)return -1;
        return l;
    }
    int _findChangePoint(int l, int r, const point& x) const
    {
        int v = _visible(l, x);
        while(r - l > 1)
        {
            int m = (l + r) / 2;
            if (_visible(m, x) == v)l = m;
            else r = m;
        }
        return l;
    }
    //all vertices in cyclic segment [first, second] are visible
    //if edge is colinear to point of view, only closer vertice is visible
    //(-1, -1) if point is inside polygon
    PII findTangents(const point& x) const
    {
        int l = 0;
        int r = _findOppositeToFirst(x);
        if (r == -1)return MP(-1, -1);
        int p1 = _findChangePoint(l, r, x);
        int p2 = _findChangePoint(r, l + sz(), x);
        if (p2 >= sz())p2 -= sz();
        if (_visible(l, x))
            swap(p2, p1);
        p1++; p2++;
        if (p1 >= sz())p1 -= sz();
        if (p2 >= sz())p2 -= sz();
        return MP(p1, p2);
    }
};

struct circle
{
    point O;
    double r;
    circle() {}
    circle(point O, double r)
    {
        this->O = O;
        this->r = r;
    }
    vector<point> intersect(const line& l) const
    {
        vector<point> ans;
        double d = l.dist(O);
        if (abs(d) > r + EPS)return ans;
        double cosx = r < EPS ? 1.0 : -d/r;
        double sinx = sqrt(abs(1.0 - cosx * cosx));
        ans.PB(O + l.n.rotate(cosx, sinx).scale(r));
        if (abs(sinx) > EPS)ans.PB(O + l.n.rotate(cosx, -sinx).scale(r));
        return ans;
    }
    vector<point> intersect(const circle& c) const
    {
        point v = c.O - O;
        double A = -2.0 * v.x;
        double B = -2.0 * v.y;
        double C = v.d2() + r * r - c.r * c.r;
        line l = line(A, B, C);
        vector<point> ans = circle(point(0, 0), r).intersect(l);
        FOR(i, 0, SZ(ans))
```

```
            ans[i] = ans[i] + O;
        return ans;
    }
    vector<line> tangents(const point& p) const
    {
        point v = p - O;
        vector<line> ans;
        double d = v.len();
        if (d < r + EPS)return ans;
        double cosx = r/d;
        double sinx = sqrt(abs(1.0 - cosx * cosx));
        point p1 = O + v.rotate(cosx, sinx).scale(r);
        point p2 = O + v.rotate(cosx, -sinx).scale(r);
        ans.PB(line(p, p1));
        if (!(p2 == p1))ans.PB(line(p, p2));
        return ans;
    }
    void _add_tan(const point& c, double r1, double r2, vector<line>& res) const
    {
        double rr = r2 - r1;
        double z = c.d2();
        double d = z - rr * rr;
        if (d < -EPS)return ;
        d = sqrt(abs(d));
        double a = (c.x * rr + c.y * d) / z;
        double b = (c.y * rr - c.x * d) / z;
        res.PB(line(a, b, r1 - a * O.x - b * O.y));
    }
    vector<line> common_tangents(const circle& C) const
    {
        vector<line> ans;
        if (O == C.O)return ans;
        point OO = C.O - O;
        _add_tan(OO, -r, -C.r, ans);
        _add_tan(OO, -r, C.r, ans);
        _add_tan(OO, r, -C.r, ans);
        _add_tan(OO, r, C.r, ans);
        return ans;
    }
    double distOnCircle(const point& p1, const point& p2) const
    {
        //assuming that both points are on circle
        double a1 = (p1 - O).polar();
        double a2 = (p2 - O).polar();
        if(a1 > a2) swap(a1,a2);
        return min(a2 - a1, 2 * PI - (a2 - a1)) * r;
    }
    bool contains(const point& p) const
    {
        return (O-p).d2() < r * r + EPS;
    }
};

//HALFPLANES INTERSECTION
struct halfplane
{
    point v, x; //x - point on line, v - vector along line
    line l;
    halfplane() {}
    halfplane(const point& p1, const point& p2) //to the left hand from vector
between p1 and p2
    {
        v = p2 - p1;
        x = p1;
        l = line(p1, p2);
    }
    bool contains(const point& p) const
    {
        return sign(v * (p - x)) >= 0;
    }
};

bool cmpHP(const halfplane& a, const halfplane& b)
{
    if (a.v.hp() != b.v.hp())return a.v.hp() < b.v.hp();
    int s = sign(a.v * b.v);
    if (s)return s > 0;
    if (a.contains(b.x) && b.contains(a.x))return 0;
    return b.contains(a.x);
}
bool eqlAngHP(const halfplane& a, const halfplane& b)
{
    return abs(a.v * b.v) < EPS;
}
halfplane Q[MAX * 2];
//assuming bounding box planes are allready added
//returns ccw-ordered convex polygon without duplicated points
polygon intersectHP(vector<halfplane> hp)
{
    sort(ALL(hp), cmpHP);
    hp.resize(unique(ALL(hp), eqlAngHP) - hp.begin());
    int l = 0, r = 0;
    FOR(i, 0, SZ(hp))
    {
        while(r - l > 1 && !hp[i].contains(Q[r-1].l.intersect(Q[r-2].l)))
            r--;
        while(r - l > 1 && !hp[i].contains(Q[l].l.intersect(Q[l+1].l)))
            l++;
        if (r - l > 0 && sign(Q[r-1].v * hp[i].v) <= 0)
            return vector<point>();
        if (r - l < 2 || Q[l].contains(hp[i].l.intersect(Q[r-1].l)))
            Q[r++] = hp[i];
    }
    vector<point> ans;
    FOR(i, l, r - 1)
        ans.PB(Q[i].l.intersect(Q[i+1].l));
    if (r - l > 2)
        ans.PB(Q[r-1].l.intersect(Q[l].l));

    ans.resize(unique(ALL(ans)) - ans.begin());
    if (ans.size() > 1 && ans[0] == ans.back())
        ans.pop_back();

    return polygon(ans);
}

//MINIMUM CICRLCE THAT CONTAINS A SET OF POINTS
point P[MAX]; //WARNING :: algorithm reorders original array
circle getCircumscribedCircle(const point& a, const point& b, const point& c)
{
    if (sign((b - a) * (c - a)) == 0)
    {
        point p = min(a, min(b, c));
        point q = max(a, max(b, c));
        return circle((p + q) * .5 , (p - q).len() * .5);
    }
    double A = a.x - b.x;
    double B = a.y - b.y;
```

```
    point p = (a + b) * .5;
    double C = -p.x * A - p.y * B;
    line l1 = line(A, B, C);

    A = b.x - c.x;
    B = b.y - c.y;
    p = (b + c) * .5;
    C = -p.x * A - p.y * B;
    line l2 = line(A, B, C);

    point O = l1.intersect(l2);
    double r = (O - a).len();
    return circle(O, r);
}
circle _minimumContainingCircle_2(int n, const point& p, const point& q)
{
    circle C = circle((p + q) * .5, (p - q).len() * .5);
    FOR(i, 0, n)
    if (!C.contains(P[i]))
        C = getCircumscribedCircle(p, q, P[i]);
    return C;
}
circle _minimumContainingCircle_1(int n, const point& p)
{
    random_shuffle(P, P + n);
    circle C = circle((P[0] + p) * .5, (P[0] - p).len() * .5);
    FOR(i, 1, n)
    if (!C.contains(P[i]))
        C = _minimumContainingCircle_2(i, P[i], p);
    return C;
}
circle minimumContainingCircle(int n)
{
    if (n == 0)return circle(point(0, 0), 0);
    if (n == 1)return circle(P[0], 0);
    random_shuffle(P, P + n);
    circle C = circle((P[0] + P[1]) * .5, (P[0] - P[1]).len() * .5);
    FOR(i, 2, n)
    if (!C.contains(P[i]))
        C = _minimumContainingCircle_1(i, P[i]);
    return C;
}

//STAINER ELLIPSE - unique ellipse inscribed in the triangle and tangent to the
sides at their midpoints.
//center of ellipse is an intersection of medians
//this is an inscribed ellipse with maximum area
//area is PI/3/sqrt(3) * S, where S - area of triangle
void stainer_ellipse(point A, point B, point C)
{
    point O = (A + B + C) * (1.0/3.0);
    point T = (A + B) * .5;
    A = A - T, B = B - T, C = C - T;

    double ang = A.polar();

    A = A.rotate(-ang);
    B = B.rotate(-ang);
    C = C.rotate(-ang);

    double k = 1/A.x;
    A = A * k; B = B * k; C = C * k;

    double AA = (B - C).d2();
    double BB = (A - C).d2();
    double CC = (A - B).d2();

    double K = 2 * (AA * AA + BB * BB + CC * CC - AA * BB - AA * CC - BB * CC);
    double a = sqrt (AA + BB + CC + K) / 3.0;
    double b = sqrt (AA + BB + CC - K) / 3.0;
    double c = sqrt(K * .5) * 2.0 / 3.0;

    complex<double> alph = complex<double>(4 * (C.x * C.x - C.y * C.y + 3), 8 *
C.x * C.y);
    alph = sqrt(alph);

    point F1 = point(2 * C.x + alph.real(), 2 * C.y + alph.imag()) * (1.0/6.0);
    F1 = F1 * (1.0/k);
    F1 = F1.rotate(ang);
    F1 = F1 + T;
    point F2 = O * 2 - F1;

    //F1, F2 - focuses
}
```

// 56. Tables of integrals and derivatives

Differentiation Formulas:

1. $\frac{d}{dx}(x) = 1$
2. $\frac{d}{dx}(ax) = a$
3. $\frac{d}{dx}(x^n) = nx^{n-1}$
4. $\frac{d}{dx}(\cos x) = -\sin x$
5. $\frac{d}{dx}(\sin x) = \cos x$
6. $\frac{d}{dx}(\tan x) = \sec^2 x$
7. $\frac{d}{dx}(\cot x) = -\csc^2 x$
8. $\frac{d}{dx}(\sec x) = \sec x \tan x$
9. $\frac{d}{dx}(\csc x) = -\csc x(\cot x)$
10. $\frac{d}{dx}(\ln x) = \frac{1}{x}$
11. $\frac{d}{dx}(e^x) = e^x$
12. $\frac{d}{dx}(a^x) = (\ln a)a^x$
13. $\frac{d}{dx}(\sin^{-1} x) = \frac{1}{\sqrt{1-x^2}}$
14. $\frac{d}{dx}(\tan^{-1} x) = \frac{1}{1+x^2}$
15. $\frac{d}{dx}(\sec^{-1} x) = \frac{1}{|x|\sqrt{x^2-1}}$

Integration Formulas:

1. $\int 1\,dx = x + C$
2. $\int a\,dx = ax + C$
3. $\int x^n dx = \frac{x^{n+1}}{n+1} + C, n \neq -1$
4. $\int \sin x\,dx = -\cos x + C$
5. $\int \cos x\,dx = \sin x + C$
6. $\int \sec^2 x\,dx = \tan x + C$
7. $\int \csc^2 x\,dx = -\cot x + C$
8. $\int \sec x(\tan x)\,dx = \sec x + C$
9. $\int \csc x(\cot x)\,dx = -\csc x + C$
10. $\int \frac{1}{x}dx = \ln|x| + C$
11. $\int e^x dx = e^x + C$
12. $\int a^x dx = \frac{a^x}{\ln a} + C\ a > 0,\ a \neq 1$
13. $\int \frac{1}{\sqrt{1-x^2}}dx = \sin^{-1} x + C$
14. $\int \frac{1}{1+x^2}dx = \tan^{-1} x + C$
15. $\int \frac{1}{|x|\sqrt{x^2-1}}dx = \sec^{-1} x + C$

Inverse Trig Functions

$$\int \frac{1}{\sqrt{a^2-u^2}}du = \sin^{-1}\left(\frac{u}{a}\right)+c \qquad \int \sin^{-1}u\,du = u\sin^{-1}u + \sqrt{1-u^2}+c$$

$$\int \frac{1}{a^2+u^2}du = \frac{1}{a}\tan^{-1}\left(\frac{u}{a}\right)+c \qquad \int \tan^{-1}u\,du = u\tan^{-1}u - \frac{1}{2}\ln\left(1+u^2\right)+c$$

$$\int \frac{1}{u\sqrt{u^2-a^2}}du = \frac{1}{a}\sec^{-1}\left(\frac{u}{a}\right)+c \qquad \int \cos^{-1}u\,du = u\cos^{-1}u - \sqrt{1-u^2}+c$$

Hyperbolic Trig Functions

$$\int \sinh u\,du = \cosh u + c \qquad \int \cosh u\,du = \sinh u + c \qquad \int \operatorname{sech}^2 u\,du = \tanh u + c$$

$$\int \operatorname{sech}\tanh u\,du = -\operatorname{sech} u + c \qquad \int \operatorname{csch}\coth u\,du = -\operatorname{csch} u + c \qquad \int \operatorname{csch}^2 u\,du = -\coth u + c$$

$$\int \tanh u\,du = \ln(\cosh u) + c \qquad \int \operatorname{sech} u\,du = \tan^{-1}|\sinh u| + c$$

Miscellaneous

$$\int \frac{1}{a^2-u^2}du = \frac{1}{2a}\ln\left|\frac{u+a}{u-a}\right|+c \qquad \int \frac{1}{u^2-a^2}du = \frac{1}{2a}\ln\left|\frac{u-a}{u+a}\right|+c$$

$$\int \sqrt{a^2+u^2}\,du = \frac{u}{2}\sqrt{a^2+u^2}+\frac{a^2}{2}\ln\left|u+\sqrt{a^2+u^2}\right|+c$$

$$\int \sqrt{u^2-a^2}\,du = \frac{u}{2}\sqrt{u^2-a^2}-\frac{a^2}{2}\ln\left|u+\sqrt{u^2-a^2}\right|+c$$

$$\int \sqrt{a^2-u^2}\,du = \frac{u}{2}\sqrt{a^2-u^2}+\frac{a^2}{2}\sin^{-1}\left(\frac{u}{a}\right)+c$$

$$\int \sqrt{2au-u^2}\,du = \frac{u-a}{2}\sqrt{2au-u^2}+\frac{a^2}{2}\cos^{-1}\left(\frac{a-u}{a}\right)+c$$

// 57. Trigonometry formulas

$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$

$\cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$

$\sin(\alpha-\beta) = \sin\alpha\cos\beta - \cos\alpha\sin\beta$

$\cos(\alpha-\beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta$

$\tan(\alpha+\beta) = \frac{\tan\alpha+\tan\beta}{1-\tan\alpha\tan\beta}$

$\sin 2\alpha = 2\sin\alpha\cos\alpha,\ \cos 2\alpha = \cos^2\alpha - \sin^2\alpha$

$\cos^2\alpha = \frac{1}{2}(1+\cos 2\alpha)$

$\sin^2\alpha = \frac{1}{2}(1-\cos 2\alpha)$

$\sin\alpha + \sin\beta = 2\sin\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$

$\cos\alpha + \cos\beta = 2\cos\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$

$\sin\alpha - \sin\beta = 2\sin\frac{\alpha-\beta}{2}\cos\frac{\alpha+\beta}{2}$

$\cos\alpha - \cos\beta = -2\sin\frac{\alpha+\beta}{2}\sin\frac{\alpha-\beta}{2}$

$\tan\alpha + \tan\beta = \frac{\sin(\alpha+\beta)}{\cos\alpha\cos\beta}$

$\cot\alpha + \cot\beta = \frac{\sin(\alpha+\beta)}{\sin\alpha\sin\beta}$

$\sin\alpha\sin\beta = \frac{1}{2}[\cos(\alpha-\beta) - \cos(\alpha+\beta)]$

$\cos\alpha\cos\beta = \frac{1}{2}[\cos(\alpha-\beta) + \cos(\alpha+\beta)]$

$\sin\alpha\cos\beta = \frac{1}{2}[\sin(\alpha+\beta) + \sin(\alpha - beta)]$

$\sin' x = \cos x,\ \cos' x = -\sin x$

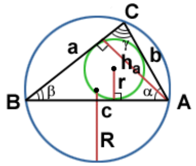
// 58. Planimetry and stereometry formulas

// triangle:

$S = \frac{1}{2} h_a * a = \frac{1}{2} b * c * sin(\alpha) = \sqrt{p(p-a)(p-b)(p-c)}$

$r = \frac{S}{p}$ $\quad R = \frac{a*b*c}{4S}$ $\quad p = \frac{a+b+c}{2}$

$a^2 = b^2 + c^2 - 2 * b * c * cos(\alpha)$ $\quad \frac{a}{\sin(\alpha)} = \frac{b}{\sin(\beta)} = \frac{c}{\sin(\gamma)} = 2R$

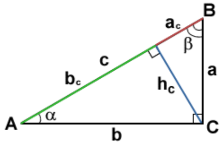


// right triangle:

$S = \frac{1}{2} * a * b = \frac{1}{2} c * h_c$ $\quad r = \frac{(a+b-c)}{2}$ $\quad R = \frac{c}{2}$

$\frac{a_c}{h_c} = \frac{h_c}{b_c}$ $\quad \frac{a_c}{a} = \frac{a}{c}$ $\quad \frac{b_c}{b} = \frac{b}{c}$

$a = c * \sin(\alpha) = c * \cos(\beta) = b * tg(\alpha) = \frac{b}{tg(\beta)}$

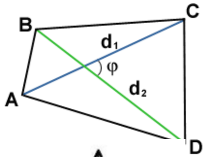


// regular triangle:

$S = \frac{a^2\sqrt{3}}{4}$ $\quad r = a\frac{\sqrt{3}}{6}$ $\quad R = a\frac{\sqrt{3}}{3}$

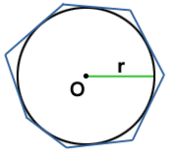
// quadrilateral:

$S = \frac{1}{2} * d_1 * d_2 * \sin(\phi)$



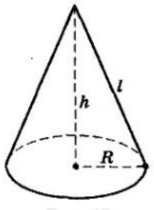
// circumscribed polygon:

$S = p * r$ (p – polygon perimeter)



// cone:

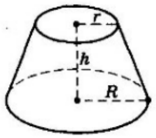
$V = \frac{1}{3}\pi r^2 h$ $\quad S_{lateral} = \pi * R * l$



// frustum (truncated cone):

$V = \frac{1}{3}\pi h\,(R^2 + R * r + r^2)$

$S_{lateral} = \pi * l * (R + r)$

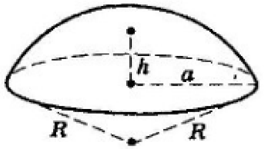


// ball segment:

$a = \sqrt{h * (2 * R - h)}$

$S_{lateral} = 2 * \pi * R * h = \pi * (a^2 + h^2)$

$V = \pi * h^2 \left(R - \frac{h}{3}\right)$

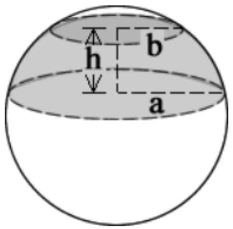


// ball layer:

$V = \frac{1}{6}\pi h\,(3 * a^2 + 3 * b^2 + h^2)$

$S_{lateral} = 2 * \pi * R * h$

$R = \sqrt{\frac{[(a-b)^2 + h^2][(a+b)^2 + h^2]}{4h^2}}$



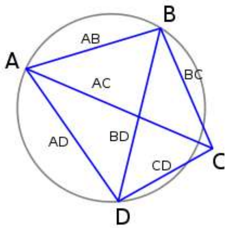
// 59. Ptolemey theorem

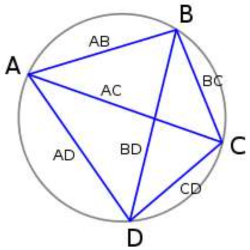
// Добуток довжин діагоналей вписаного в коло чотирикутника дорі
добутків довжин його протилежних сторін.

$|AC| * |BD| = |AB| * |CD| + |BC| * |AD|$

// Для довільного чотирикутника справджується:

$|AB| * |CD| + |BC| * |DA| \geq |AC| * |BD|$





// Рівність досягається
лише для чотирикутника
вписаного в коло

// 60. NP-complete problems

// Bipartite graphs:
// - Minimum vertex cover = size of maximum matching.
// - Maximum independent set + size of maximum matching = number of vertices.

// In ANY graph without isolated vertices the size of the minimum edge cover + size of a maximum matching = number of vertices.

// Dominating set for a graph G = (V, E) is a subset D of V such that every vertex not in D is adjacent to at least one member of D. Finding dominating set is NP-complete on bipartite graphs.

// 61. Min-cut restoring

// To restore mincut:
// - run dfs from source to sink by direct and reversed edges
// - use only edges that have flow != capacity
// - for each direct edge (u, v):
// -- if u is visited and v is not visited add (u, v) to cut

// 62. Java

```
//for fast input use
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
//if need to read fast several numbers given in one line
StringTokenizer tokenizer = new StringTokenizer(reader.readLine());
//for fast output use
PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(System.out)));
```

// 63. Sherman-Lehman factorization

```
// Retuans ordered list of prime factors of the number
// Complexity: O(N^1/3 * log(N))
vector<LL> Lehman(LL n)
{
	vector<LL> res;
	LL i, j;
	for(i = 2; i*i*i <= n; ++i)
		while(n % i == 0)
		{
			res.PB(i);
			n /= i;
		}
	if(n == 1)
		return res;
	LL Min, Max;
	LL x = 0;
	LL y = 0;
	for(i = 1; i*i*i <= n; ++i)
	{
		y += 4*n;
		Min = 0;
		Max = 1;
		while(Max*(x + x + 1) + Max*(Max - 1) <= y)
			Max *= 2;
		while(Max - Min > 1)
		{
			LL Mid = (Max + Min) / 2;
			if(Mid*(x + x + 1) + Mid*(Mid - 1) <= y)
```

```
                        Min = Mid;
                else
                        Max = Mid;
            }
            y -= Min*(x + x + 1) + Min*(Min - 1);
            x += Min;
            LL z = -y;
            for(j = 0; ; ++j)
            {
                if(z >= 0)
                {
                    Min = 0;
                    Max = 1;
                    while(Max*Max <= z)
                        Max <<= 1;
                    while(Max - Min > 1)
                    {
                        LL Mid = (Max + Min) / 2;
                        if(Mid*Mid <= z)
                            Min = Mid;
                        else
                            Max = Mid;
                    }
                    if(Min*Min == z)
                    {
                        LL a = x + j + Min;
                        LL g = gcd(a, n);
                        if(g != 1 && g != n)
                        {
                            res.PB(min(g, n/g));
                            res.PB(max(g, n/g));
                            return res;
                        }
                        a = x + j - Min;
                        g = gcd(a < 0 ? -a : a, n);
                        if(g != 1 && g != n)
                        {
                            res.PB(min(g, n/g));
                            res.PB(max(g, n/g));
                            return res;
                        }
                    }
                }
                z += j + j + x + x + 1;
                if(((j*j*j*j*j*j) << 12) > n/(i*i*i))
                    break;
            }
        }
        res.PB(n);
        return res;
}
```