```
////////////////////////////////////////////////////////////////////////////////
// Índice //////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
```



```
////////////////////////////////////////////////////////////////////////////////
// .vimrc //////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

syn on
set nocin ai noet ts=4 sw=4
mat Keyword "\<FOR\>"

////////////////////////////////////////////////////////////////////////////////
// Script de compilação C++ ////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

#!/bin/sh

clear
rm -f $1.out

if (g++ -o $1 $1.cpp -Wall -Wshadow -Wno-long-long -pedantic -g) then
  echo "### COMPILOU ###"
  if !(./$1 < $1.in > $1.out) then
    echo "### RUNTIME ERROR ###" >> $1.out
  fi
  less $1.out
fi

////////////////////////////////////////////////////////////////////////////////
// Script de compilação Java ///////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

#!/bin/sh

clear
rm -f $1.out

if (javac $1.java) then
  echo "### COMPILOU ###"
  if !(java $1 < $1.in > $1.out) then
    echo "### RUNTIME ERROR ###" >> $1.out
  fi
  less $1.out
fi
```

```
////////////////////////////////////////////////////////////////////////////////
// Modelo C++ //////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <ctype.h>

#include <algorithm>
#include <string>
#include <utility>
#include <iostream>

using namespace std;

#define TRACE(x...)
#define PRINT(x...) TRACE(printf(x))
#define WATCH(x) TRACE(cout << #x << " = " << x << endl)

#define all(v) (v).begin(), (v).end()
#define rall(v) (v).rbegin(), (v).rend()

#define FOR(it, b, e) for (typeof(b) it = (b); it != (e); ++it)
#define MSET(c, v) memset(c, v, sizeof(c)

const int INF = 0x3F3F3F3F; const int NEGINF = 0xC0C0C0C0;
const int NULO = -1;
const double EPS = 1e-10;

inline int cmp(double x, double y = 0, double tol = EPS) {
  return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}


int main() {
  TRACE(setbuf(stdout, NULL));
  return 0;
}

////////////////////////////////////////////////////////////////////////////////
// Modelo Java /////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

import java.util.*;
import java.math.*;

class modelo {
  static final double EPS = 1.e-10;
  static final boolean DBG = true;

  private static int cmp(double x, double y = 0, double tol = EPS) {
    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
  }

  public static void main(String[] argv) {
    Scanner s = new Scanner(System.in);
  }
}
```

```
////////////////////////////////////////////////////////////////////////////////
// Recomendações gerais ////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

# ANTES DA PROVA

- Revisar os algoritmos disponíveis na biblioteca.
- Revisar a referência STL.
- Reler este roteiro.
- Ouvir o discurso motivacional do técnico.

# ANTES DE IMPLEMENTAR UM PROBLEMA

- Quem for implementar deve relê-lo antes.
- Peça todas as clarifications que forem necessárias.
- Marque as restrições e faça contas com os limites da entrada.
- Teste o algoritmo no papel e convença outra pessoa de que ele funciona.
- Planeje a resolução para os problemas grandes: a equipe se junta para definir
  as estruturas de dados, mas cada pessoa escreve uma função.

# DEBUGAR UM PROGRAMA

- Ao encontrar um bug, escreva um caso de teste que o dispare.
- Reimplementar trechos de programas entendidos errados.
- Em caso de RE, procure todos os [, / e %.

////////////////////////////////////////////////////////////////////////////////
// Roteiro de prova ////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

# 300 MINUTOS: INÍCIO DE PROVA

- Fábio e Daniel começam lendo os problemas.
- Roberto começa digitando ".vimrc", "compila" e "modelo.cpp".
- Roberto gera todos os arquivos-fonte, copiando "modelo.cpp" com -i.
- Roberto apaga "modelo.cpp" com -i.
- Quando surgir um problema fácil, todos discutem se ele deve ser o primeiro
  a ser resolvido.
- Quando o primeiro problema for escolhido, Fábio o implementa, possivelmente
  tirando Roberto do computador e interrompendo as digitações.
- Se surgir um problema ainda mais fácil que o primeiro, Fábio passa a
  implementar esse novo problema.
- Enquanto Fábio resolve o primeiro problema, Daniel e Roberto lêem os demais.
- À medida que Roberto for lendo os problemas, ele os explica para Daniel.
- Daniel preenche a tabela com os problemas até então lidos:
  > AC: :)
  > Ordem: ordem de resolução dos problemas (pode ser infinito).
  > Escrito: se já há código escrito neste problema, mesmo que no papel.
  > Leitores: pessoas que já leram o problema.
  > Complexidade: complexidade da solução implementada.
  > Resumo: resumo sobre o problema.
- Assim que o primeiro problema começar a ser compilado, Fábio avisa Daniel e
  Roberto para escolherem o segundo problema mais fácil.
- Assim que o primeiro problema for submetido, Fábio sai do computador.
- Roberto entra no computador, e termina as digitações pendentes.
- Roberto implementa o segundo problema mais fácil.
- Fora do computador, Daniel e Fábio escolhem a ordem e os resolvedores
  dos problemas, com base no tempo de implementação.
- Se ninguém tiver alguma idéia para resolver um problema, empurre-o para
  o final (ou seja, a ordem desse problema será infinito).
- Quando Roberto submete o segundo problema e sai do computador, ele revê
  a ordenação dos problemas com quem ficou fora do computador.

# 200 MINUTOS: MEIA-PROVA

- A equipe deve resolver no máximo três problemas ao mesmo tempo.
- Escreva o máximo possível de código no papel.
- Depure com o código do problema e com a saída do TRACE impressos.
  > Explique seu código para outra pessoa da equipe.
  > Acompanhe o código linha por linha, anotando os valores das variáveis e
    redesenhando as estruturas de dados à medida que forem alteradas.
- Momentos nos quais quem estiver no computador deve avisar os outros membros
  da equipe:
  > Quando estiver pensando ou depurando.
  > Quando estiver prestes a submeter, para que os outros membros possam fazer
    testes extras e verificar o formato da saída.
- Submeta sempre em C++, com extensão .cpp.
- Logo após submeter, imprima o código.
- Jogue fora as versões mais antigas do código impresso de um programa.
- Jogue fora todos os papéis de um problema quando receber Accepted.
- Mantenha todos os papéis de um problema grampeados.

# 100 MINUTOS: FINAL DE PROVA

- A equipe deve resolver apenas um problema no final da prova.
- Use os balões das outras equipes para escolher o último problema:
  > Os problemas mais resolvidos por outras equipes provavelmente são mais
    fáceis que os outros problemas.
  > Uma equipe mais bem colocada só é informativa quando as demais não o forem.
- Como Fábio digita mais rápido, ele fica o tempo todo no computador.
- Daniel e Roberto sentam ao lado de Fábio e dão sugestões para o problema.

# 60 MINUTOS: PLACAR CONGELADO

- Preste atenção nos melancias e nas comemorações das outras equipes:
  os balões continuam vindo!

# __ MINUTOS: JUÍZES CALADOS

- Quando terminar um problema, teste com o exemplo de entrada, submeta
  e só depois pense em mais casos de teste.
- Nos últimos cinco minutos, faça alterações pequenas no código, remova o TRACE
  e submeta.

////////////////////////////////////////////////////////////////////////////////
// Os 1010 Mandamentos /////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

0. Não dividirás por zero.
1. Não alocarás dinamicamente.
2. Compararás números de ponto flutuante usando cmp().
3. Verificarás se o grafo pode ser desconexo.
4. Verificarás se as arestas do grafo podem ter peso negativo.
5. Verificarás se pode haver mais de uma aresta ligando dois vértices.
6. Conferirás todos os índices de uma programação dinâmica.
7. Reduzirás o branching factor da DFS.
8. Farás todos os cortes possíveis em uma DFS.
9. Tomarás cuidado com pontos coincidentes e com pontos colineares.
```

```
////////////////////////////////////////////////////////////////////////////////
// Limites de representação de dados //////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

          tipo      | bits |      mínimo .. máximo      | precisão decimal
    ----------------+------+----------------------------+------------------
     char           |  8   |            0 .. 127        |        2
     signed char    |  8   |         -128 .. 127        |        2
     unsigned char  |  8   |            0 .. 255        |        2
     short          |  16  |      -32.768 .. 32.767     |        4
     unsigned short |  16  |            0 .. 65.535     |        4
     int            |  32  |   -2 × 10**9 .. 2 × 10**9  |        9
     unsigned int   |  32  |            0 .. 4 × 10**9  |        9
     int64_t        |  64  | -9 × 10**18 .. 9 × 10**18  |       18
     uint64_t       |  64  |            0 .. 18 × 10**18 |       19

               tipo      | bits | expoente | precisão decimal
             ----------------+------+----------+------------------
              float          |  32  |     38   |        6
              double         |  64  |    308   |       15
              long double    |  80  |  19.728  |       18

////////////////////////////////////////////////////////////////////////////////
// Quantidade de números primos de 1 até 10**n ////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

pi(10**1) =          4
pi(10**2) =         25
pi(10**3) =        168
pi(10**4) =      1.229
pi(10**5) =      9.592
pi(10**6) =     78.498
pi(10**7) =    664.579
pi(10**8) =  5.761.455
pi(10**9) = 50.847.534

[É sempre verdade que n / ln(n) < pi(n) < 1.26 * n / ln(n).]

////////////////////////////////////////////////////////////////////////////////
// Triângulo de Pascal /////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

    n \ p |  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |  9  |  10
   -------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+------
        0 |   1 |_____
        1 |   1 |   1 |_____
        2 |   1 |   2 |   1 |_____
        3 |   1 |   3 |   3 |   1 |_____
        4 |   1 |   4 |   6 |   4 |   1 |_____
        5 |   1 |   5 |  10 |  10 |   5 |   1 |_____
        6 |   1 |   6 |  15 |  20 |  15 |   6 |   1 |_____
        7 |   1 |   7 |  21 |  35 |  35 |  21 |   7 |   1 |_____
        8 |   1 |   8 |  28 |  56 |  70 |  56 |  28 |   8 |   1 |_____
        9 |   1 |   9 |  36 |  84 | 126 | 126 |  84 |  36 |   9 |   1 |_____
       10 |   1 |  10 |  45 | 120 | 210 | 252 | 210 | 120 |  45 |  10 |   1 |

C(33, 16) =              1.166.803.110 [limite do int]
C(34, 17) =              2.333.606.220 [limite do unsigned int]
C(66, 33) =  7.219.428.434.016.265.740 [limite do int64_t]
C(67, 33) = 14.226.520.737.620.288.370 [limite do uint64_t]
```

```
////////////////////////////////////////////////////////////////////////////////
// Fatorial ////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

 0! =                         1
 1! =                         1
 2! =                         2
 3! =                         6
 4! =                        24
 5! =                       120
 6! =                       720
 7! =                     5.040
 8! =                    40.320
 9! =                   362.880
10! =                 3.628.800
11! =                39.916.800
12! =               479.001.600 [limite do (unsigned) int]
13! =             6.227.020.800
14! =            87.178.291.200
15! =         1.307.674.368.000
16! =        20.922.789.888.000
17! =       355.687.428.096.000
18! =     6.402.373.705.728.000
19! =   121.645.100.408.832.000
20! = 2.432.902.008.176.640.000 [limite do (u)int64_t]

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

p(n) ~ exp(pi * sqrt(2 * n / 3))/(4 * n * sqrt(3))

Os números pentagonais generalizados são os números da for a n*(3*n-1)/2, onde
n = ..., -3, -2, -1, 0, 1, 2, 3, ...

p(n) - p(n-1) - p(n-2) + p(n-5) + p(n-7) - p(n-12) - p(n-15) + ... = 0.

A soma é feita sobre p(n-k), k pentagonal generalizado, e o sinal de p(n-k) é
(-1)**int((k+1)/2). p(0) é definido como 1.
```

```
////////////////////////////////////////////////////////////////////////////////
// Primos até 10.000 ///////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
```

[Existem 1.229 números primos até 10.000.]

2	3	5	7	11	13	17	19	23	29	31
37	41	43	47	53	59	61	67	71	73	79
83	89	97	101	103	107	109	113	127	131	137
139	149	151	157	163	167	173	179	181	191	193
197	199	211	223	227	229	233	239	241	251	257
263	269	271	277	281	283	293	307	311	313	317
331	337	347	349	353	359	367	373	379	383	389
397	401	409	419	421	431	433	439	443	449	457
461	463	467	479	487	491	499	503	509	521	523
541	547	557	563	569	571	577	587	593	599	601
607	613	617	619	631	641	643	647	653	659	661
673	677	683	691	701	709	719	727	733	739	743
751	757	761	769	773	787	797	809	811	821	823
827	829	839	853	857	859	863	877	881	883	887
907	911	919	929	937	941	947	953	967	971	977
983	991	997	1009	1013	1019	1021	1031	1033	1039	1049
1051	1061	1063	1069	1087	1091	1093	1097	1103	1109	1117
1123	1129	1151	1153	1163	1171	1181	1187	1193	1201	1213
1217	1223	1229	1231	1237	1249	1259	1277	1279	1283	1289
1291	1297	1301	1303	1307	1319	1321	1327	1361	1367	1373
1381	1399	1409	1423	1427	1429	1433	1439	1447	1451	1453
1459	1471	1481	1483	1487	1489	1493	1499	1511	1523	1531
1543	1549	1553	1559	1567	1571	1579	1583	1597	1601	1607
1609	1613	1619	1621	1627	1637	1657	1663	1667	1669	1693
1697	1699	1709	1721	1723	1733	1741	1747	1753	1759	1777
1783	1787	1789	1801	1811	1823	1831	1847	1861	1867	1871
1873	1877	1879	1889	1901	1907	1913	1931	1933	1949	1951
1973	1979	1987	1993	1997	1999	2003	2011	2017	2027	2029
2039	2053	2063	2069	2081	2083	2087	2089	2099	2111	2113
2129	2131	2137	2141	2143	2153	2161	2179	2203	2207	2213
2221	2237	2239	2243	2251	2267	2269	2273	2281	2287	2293
2297	2309	2311	2333	2339	2341	2347	2351	2357	2371	2377
2381	2383	2389	2393	2399	2411	2417	2423	2437	2441	2447
2459	2467	2473	2477	2503	2521	2531	2539	2543	2549	2551
2557	2579	2591	2593	2609	2617	2621	2633	2647	2657	2659
2663	2671	2677	2683	2687	2689	2693	2699	2707	2711	2713
2719	2729	2731	2741	2749	2753	2767	2777	2789	2791	2797
2801	2803	2819	2833	2837	2843	2851	2857	2861	2879	2887
2897	2903	2909	2917	2927	2939	2953	2957	2963	2969	2971
2999	3001	3011	3019	3023	3037	3041	3049	3061	3067	3079
3083	3089	3109	3119	3121	3137	3163	3167	3169	3181	3187
3191	3203	3209	3217	3221	3229	3251	3253	3257	3259	3271
3299	3301	3307	3313	3319	3323	3329	3331	3343	3347	3359
3361	3371	3373	3389	3391	3407	3413	3433	3449	3457	3461
3463	3467	3469	3491	3499	3511	3517	3527	3529	3533	3539
3541	3547	3557	3559	3571	3581	3583	3593	3607	3613	3617
3623	3631	3637	3643	3659	3671	3673	3677	3691	3697	3701
3709	3719	3727	3733	3739	3761	3767	3769	3779	3793	3797
3803	3821	3823	3833	3847	3851	3853	3863	3877	3881	3889
3907	3911	3917	3919	3923	3929	3931	3943	3947	3967	3989
4001	4003	4007	4013	4019	4021	4027	4049	4051	4057	4073
4079	4091	4093	4099	4111	4127	4129	4133	4139	4153	4157
4159	4177	4201	4211	4217	4219	4229	4231	4241	4243	4253
4259	4261	4271	4273	4283	4289	4297	4327	4337	4339	4349
4357	4363	4373	4391	4397	4409	4421	4423	4441	4447	4451
4457	4463	4481	4483	4493	4507	4513	4517	4519	4523	4547
4549	4561	4567	4583	4591	4597	4603	4621	4637	4639	4643
4649	4651	4657	4663	4673	4679	4691	4703	4721	4723	4729
4733	4751	4759	4783	4787	4789	4793	4799	4801	4813	4817
4831	4861	4871	4877	4889	4903	4909	4919	4931	4933	4937
4943	4951	4957	4967	4969	4973	4987	4993	4999	5003	5009
5011	5021	5023	5039	5051	5059	5077	5081	5087	5099	5101
5107	5113	5119	5147	5153	5167	5171	5179	5189	5197	5209
5227	5231	5233	5237	5261	5273	5279	5281	5297	5303	5309
5323	5333	5347	5351	5381	5387	5393	5399	5407	5413	5417
5419	5431	5437	5441	5443	5449	5471	5477	5479	5483	5501
5503	5507	5519	5521	5527	5531	5557	5563	5569	5573	5581
5591	5623	5639	5641	5647	5651	5653	5657	5659	5669	5683
5689	5693	5701	5711	5717	5737	5741	5743	5749	5779	5783
5791	5801	5807	5813	5821	5827	5839	5843	5849	5851	5857
5861	5867	5869	5879	5881	5897	5903	5923	5927	5939	5953
5981	5987	6007	6011	6029	6037	6043	6047	6053	6067	6073
6079	6089	6091	6101	6113	6121	6131	6133	6143	6151	6163
6173	6197	6199	6203	6211	6217	6221	6229	6247	6257	6263
6269	6271	6277	6287	6299	6301	6311	6317	6323	6329	6337
6343	6353	6359	6361	6367	6373	6379	6389	6397	6421	6427
6449	6451	6469	6473	6481	6491	6521	6529	6547	6551	6553
6563	6569	6571	6577	6581	6599	6607	6619	6637	6653	6659
6661	6673	6679	6689	6691	6701	6703	6709	6719	6733	6737
6761	6763	6779	6781	6791	6793	6803	6823	6827	6829	6833
6841	6857	6863	6869	6871	6883	6899	6907	6911	6917	6947
6949	6959	6961	6967	6971	6977	6983	6991	6997	7001	7013
7019	7027	7039	7043	7057	7069	7079	7103	7109	7121	7127
7129	7151	7159	7177	7187	7193	7207	7211	7213	7219	7229
7237	7243	7247	7253	7283	7297	7307	7309	7321	7331	7333
7349	7351	7369	7393	7411	7417	7433	7451	7457	7459	7477
7481	7487	7489	7499	7507	7517	7523	7529	7537	7541	7547
7549	7559	7561	7573	7577	7583	7589	7591	7603	7607	7621
7639	7643	7649	7669	7673	7681	7687	7691	7699	7703	7717
7723	7727	7741	7753	7757	7759	7789	7793	7817	7823	7829
7841	7853	7867	7873	7877	7879	7883	7901	7907	7919	7927
7933	7937	7949	7951	7963	7993	8009	8011	8017	8039	8053
8059	8069	8081	8087	8089	8093	8101	8111	8117	8123	8147
8161	8167	8171	8179	8191	8209	8219	8221	8231	8233	8237
8243	8263	8269	8273	8287	8291	8293	8297	8311	8317	8329
8353	8363	8369	8377	8387	8389	8419	8423	8429	8431	8443
8447	8461	8467	8501	8513	8521	8527	8537	8539	8543	8563
8573	8581	8597	8599	8609	8623	8627	8629	8641	8647	8663
8669	8677	8681	8689	8693	8699	8707	8713	8719	8731	8737
8741	8747	8753	8761	8779	8783	8803	8807	8819	8821	8831
8837	8839	8849	8861	8863	8867	8887	8893	8923	8929	8933
8941	8951	8963	8969	8971	8999	9001	9007	9011	9013	9029
9041	9043	9049	9059	9067	9091	9103	9109	9127	9133	9137
9151	9157	9161	9173	9181	9187	9199	9203	9209	9221	9227
9239	9241	9257	9277	9281	9283	9293	9311	9319	9323	9337
9341	9343	9349	9371	9377	9391	9397	9403	9413	9419	9421
9431	9433	9437	9439	9461	9463	9467	9473	9479	9491	9497
9511	9521	9533	9539	9547	9551	9587	9601	9613	9619	9623
9629	9631	9643	9649	9661	9677	9679	9689	9697	9719	9721
9733	9739	9743	9749	9767	9769	9781	9787	9791	9803	9811
9817	9829	9833	9839	9851	9857	9859	9871	9883	9887	9901
9907	9923	9929	9931	9941	9949	9967	9973			

```
////////////////////////////////////////////////////////////////////////////////
// Utilidades do lar ///////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

template <class T>
struct index_lt {
  T& v;
  index_lt(T& v): v(v) {}
  inline bool operator ()(int i, int j) {
    return (v[i] != v[j]) ? (v[i] < v[j]) : (i < j);
  }
};
template <class T> index_lt<T> make_index_lt(T& v) { return index_lt<T>(v); }

bool cmp_eq(double x, double y) { return cmp(x, y) == 0; }
bool cmp_lt(double x, double y) { return cmp(x, y) < 0; }

int safe_gets(char*& s) { // depois de usar, free(s);
  return scanf("%a[^\r\n]%*[\r\n]", &s);
}

////////////////////////////////////////////////////////////////////////////////
// Geometria ///////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

#include <vector>

struct point {
  double x, y;
  point(double x = 0, double y = 0): x(x), y(y) {}

  point operator +(point q) { return point(x + q.x, y + q.y); }
  point operator -(point q) { return point(x - q.x, y - q.y); }
  point operator *(double t) { return point(x * t, y * t); }
  point operator /(double t) { return point(x / t, y / t); }
  double operator *(point q) { return x * q.x + y * q.y; }
  double operator %(point q) { return x * q.y - y * q.x; }

  int cmp(point q) const {
    if (int t = ::cmp(x, q.x)) return t;
    return ::cmp(y, q.y);
  }
  bool operator ==(point q) const { return cmp(q) == 0; }
  bool operator !=(point q) const { return cmp(q) != 0; }
  bool operator < (point q) const { return cmp(q) <  0; }

  friend ostream& operator <<(ostream& o, point p) {
    return o << "(" << p.x << ", " << p.y << ")";
  }

  static point pivot;
};

point point::pivot;

double abs(point p) { return hypot(p.x, p.y); }
double arg(point p) { return atan2(p.y, p.x); }

typedef vector<point> polygon;

inline int ccw(point p, point q, point r) {
  return cmp((p - r) % (q - r));
}

inline double angle(point p, point q, point r) {
  point u = p - q, v = r - q;
  return atan2(u % v, u * v);
}

////////////////////////////////////////////////////////////////////////////////
// Decide se q está sobre o segmento fechado pr.
//

bool between(point p, point q, point r) {
  return ccw(p, q, r) == 0 && cmp((p - q) * (r - q)) <= 0;
}

////////////////////////////////////////////////////////////////////////////////
// Decide se os segmentos fechados pq e rs têm pontos em comum.
//

bool seg_intersect(point p, point q, point r, point s) {
  point A = q - p, B = s - r, C = r - p, D = s - q;
  int a = cmp(A % C) + 2 * cmp(A % D);
  int b = cmp(B % C) + 2 * cmp(B % D);
  if (a == 3 || a == -3 || b == 3 || b == -3) return false;
  if (a || b || p == r || p == s || q == r || q == s) return true;
  int t = (p < r) + (p < s) + (q < r) + (q < s);
  return t != 0 && t != 4;
}

////////////////////////////////////////////////////////////////////////////////
// Calcula a distância do ponto r ao segmento pq.
//

double seg_distance(point p, point q, point r) {
  point A = r - q, B = r - p, C = q - p;
  double a = A * A, b = B * B, c = C * C;
  if (cmp(b, a + c) >= 0) return sqrt(a);
  else if (cmp(a, b + c) >= 0) return sqrt(b);
  else return fabs(A % B) / sqrt(c);
}

////////////////////////////////////////////////////////////////////////////////
// Classifica o ponto p em relação ao polígono T.
//
// Retorna 0, -1 ou 1 dependendo se p está no exterior, na fronteira
// ou no interior de T, respectivamente.
//

int in_poly(point p, polygon& T) {
  double a = 0; int N = T.size();
  for (int i = 0; i < N; i++) {
    if (between(T[i], p, T[(i+1) % N])) return -1;
    a += angle(T[i], p, T[(i+1) % N]);
  }
  return cmp(a) != 0;
}
```

```
//////////////////////////////////////////////////////////////////////////////
// Comparação radial.
//

bool radial_lt(point p, point q) {
  point P = p - point::pivot, Q = q - point::pivot;
  double R = P % Q;
  if (cmp(R)) return R > 0;
  return cmp(P * P, Q * Q) < 0;
}

//////////////////////////////////////////////////////////////////////////////
// Determina o fecho convexo de um conjunto de pontos no plano.
//
// Destrói a lista de pontos T.
//

polygon convex_hull(vector<point>& T) {
  int j = 0, k, n = T.size(); polygon U(n);

  point::pivot = *min_element(all(T));
  sort(all(T), radial_lt);
  for (k = n-2; k >= 0 && ccw(T[0], T[n-1], T[k]) == 0; k--);
  reverse((k+1) + all(T));

  for (int i = 0; i < n; i++) {
    // troque o >= por > para manter pontos colineares
    while (j > 1 && ccw(U[j-1], U[j-2], T[i]) >= 0) j--;
    U[j++] = T[i];
  }
  U.erase(j + all(U));
  return U;
}

//////////////////////////////////////////////////////////////////////////////
// Calcula a área orientada do polígono T.
//

double poly_area(polygon& T) {
  double s = 0; int n = T.size();
  for (int i = 0; i < n; i++)
    s += T[i] % T[(i+1) % n];
  return s / 2;
}

//////////////////////////////////////////////////////////////////////////////
// Encontra o ponto de interseção das retas pq e rs.
//

point line_intersect(point p, point q, point r, point s) {
  point a = q - p, b = s - r, c = point(p % q, r % s);
  return point(point(a.x, b.x) % c, point(a.y, b.y) % c) / (a % b);
}

//////////////////////////////////////////////////////////////////////////////
// Encontra o menor círculo que contém todos os pontos dados.
//

typedef pair<point, double> circle;

bool in_circle(circle C, point p){
  return cmp(abs(p - C.first), C.second) <= 0;
}

point circumcenter(point p, point q, point r) {
  point a = p - r, b = q - r, c = point(a * (p + r) / 2, b * (q + r) / 2);
  return point(c % point(a.y, b.y), point(a.x, b.x) % c) / (a % b);
}

circle spanning_circle(vector<point>& T) {
  int n = T.size();
  random_shuffle(all(T));
  circle C(point(), -INFINITY);
  for (int i = 0; i < n; i++) if (!in_circle(C, T[i])) {
    C = circle(T[i], 0);
    for (int j = 0; j < i; j++) if (!in_circle(C, T[j])) {
      C = circle((T[i] + T[j]) / 2, abs(T[i] - T[j]) / 2);
      for (int k = 0; k < j; k++) if (!in_circle(C, T[k])) {
        point o = circumcenter(T[i], T[j], T[k]);
        C = circle(o, abs(o - T[k]));
      }
    }
  }
  return C;
}

//////////////////////////////////////////////////////////////////////////////
// Determina o polígono interseção dos dois polígonos convexos P e Q.
//
// Tanto P quanto Q devem estar orientados positivamente.
//

polygon poly_intersect(polygon& P, polygon& Q) {
  int m = Q.size(), n = P.size();
  int a = 0, b = 0, aa = 0, ba = 0, inflag = 0;
  polygon R;
  while ((aa < n || ba < m) && aa < 2*n && ba < 2*m) {
    point p1 = P[a], p2 = P[(a+1) % n], q1 = Q[b], q2 = Q[(b+1) % m];
    point A = p2 - p1, B = q2 - q1;
    int cross = cmp(A % B), ha = ccw(p2, q2, p1), hb = ccw(q2, p2, q1);
    if (cross == 0 && ccw(p1, q1, p2) == 0 && cmp(A * B) < 0) {
      if (between(p1, q1, p2)) R.push_back(q1);
      if (between(p1, q2, p2)) R.push_back(q2);
      if (between(q1, p1, q2)) R.push_back(p1);
      if (between(q1, p2, q2)) R.push_back(p2);
      if (R.size() < 2) return polygon();
      inflag = 1; break;
    } else if (cross != 0 && seg_intersect(p1, p2, q1, q2)) {
      if (inflag == 0) aa = ba = 0;
      R.push_back(line_intersect(p1, p2, q1, q2));
      inflag = (hb > 0) ? 1 : -1;
    }
    if (cross == 0 && hb < 0 && ha < 0) return R;
    bool t = cross == 0 && hb == 0 && ha == 0;
    if (t ? (inflag == 1) : (cross >= 0) ? (ha <= 0) : (hb > 0)) {
      if (inflag == -1) R.push_back(q2);
      ba++; b++; b %= m;
    } else {
      if (inflag == 1) R.push_back(p2);
      aa++; a++; a %= n;
```

```
    }
  }
  if (inflag == 0) {
    if (in_poly(P[0], Q)) return P;
    if (in_poly(Q[0], P)) return Q;
  }
  R.erase(unique(all(R)), R.end());
  if (R.size() > 1 && R.front() == R.back()) R.pop_back();
  return R;
}

////////////////////////////////////////////////////////////////////////////////
// Campos ////////////////////////////////////////////////////////// 15 min. //
////////////////////////////////////////////////////////////////////////////////

#include <list>
#include <set>

const int TAM = 2000;

typedef pair<point, point> segment;
typedef pair<int, int> barrier;

struct field {
  int n, m;
  point v[TAM];
  barrier b[TAM];
  list<int> e[TAM];

  field(): n(0), m(0) {}

  void clear() {
    for (int i = 0; i < n; i++) e[i].clear();
    n = m = 0;
  }

  inline int ccw(int a, int b, int c) { return ::ccw(v[a], v[b], v[c]); }

  void make_barrier(int i, int j) {
    e[i].push_back(m); e[j].push_back(m);
    b[m++] = barrier(i, j);
  }

  //////////////////////////////////////////////////////////////////////////////
  // Remove os casos degenerados de um campo.
  //

  void normalize() {
    set&lt;segment&gt; T; set<point> U;
    for (int i = 0; i < n; i++) make_barrier(i, i);
    for (int i = 0; i < m; i++) {
      point p = v[b[i].first], q = v[b[i].second];
      set<point> S;
      S.insert(p); S.insert(q);
      for (int j = 0; j < m; j++) {
        point r = v[b[j].first], s = v[b[j].second];
        if (r == p || r == q || s == p || s == q) continue;
        if (cmp((q - p) % (s - r)) == 0) {
          if (between(p, r, q)) S.insert(r);
          if (between(p, s, q)) S.insert(s);
        } else if (seg_intersect(p, q, r, s)) {
          S.insert(line_intersect(p, q, r, s));
        }
      }
      foreach (st, all(S)) {
        if (st != S.begin()) T.insert(segment(p, *st));
        U.insert(p = *st);
      }
    }
    clear();
    foreach (it, all(U)) v[n++] = *it;
    foreach (it, all(T)) {
      int i = lower_bound(v, v+n, it->first) - v;
      int j = lower_bound(v, v+n, it->second) - v;
      make_barrier(i, j);
    }
  }

  //////////////////////////////////////////////////////////////////////////////
  // Algoritmo de Poggi-Moreira-Fleischman-Cavalcante.
  //
  // Determina um grafo que contém todas as arestas de um eventual menor
  // caminho entre pontos do grafo.
  //

  void pmfc(graph& G) {
    int sel[TAM][2], active[TAM];
    for (int i = 0; i < n; i++) {
      vector< pair<double, int> > T;
      foreach (it, all(e[i])) {
        int j = b[*it].first + b[*it].second - i;
        T.push_back(make_pair(arg(v[j] - v[i]), j));
      }
      sort(all(T));
      if (T.empty()) T.push_back(make_pair(0, i));
      active[i] = 0;
      int p = T.size();
      for (int j = 0; j < p; j++) {
        sel[i][0] = T[j].second; sel[i][1] = T[(j+1) % p].second;
        if (ccw(sel[i][0], sel[i][1], i) <= 0) {
          active[i] = 1; break;
        }
      }
    }
    G.init(n);
    for (int i = 0; i < n; i++) for (int j = 0; j < i; j++) {
      if (!active[i] || !active[j]) continue;
      if (ccw(i, j, sel[i][0]) * ccw(i, j, sel[i][1]) == -1 || \
          ccw(i, j, sel[j][0]) * ccw(i, j, sel[j][1]) == -1)
        continue;
      for (int k = 0; k < m; k++) {
        int org = b[k].first, dest = b[k].second;
        if (org == i || org == j || dest == i || dest == j) continue;
        if (seg_intersect(v[i], v[j], v[org], v[dest])) goto PROX;
      }
      G.aresta(i, j, 1, abs(v[j] - v[i]));
PROX: ;
    }
  }
};
```

```
//////////////////////////////////////////////////////////////////////////////
// Triangulações ///////////////////////////////////////////////////// 15 min. //
//////////////////////////////////////////////////////////////////////////////

// Depende da struct field.

#include <map>

typedef pair<int, int> edge;

struct triangulation: public map<edge, int> {
  edge sym(edge e)   { return edge(e.second, e.first); }
  edge lnext(edge e) { return edge(e.second, (*this)[e]); }
  edge lprev(edge e) { return edge((*this)[e], e.first); }
  edge dnext(edge e) { return lprev(sym(lprev(e))); }
  edge dprev(edge e) { return lnext(sym(lnext(e))); }

  void new_tri(edge e, int r) {
    if (count(e)) { erase(lnext(e)); erase(lprev(e)); }
    (*this)[e] = r; (*this)[lnext(e)] = e.first; (*this)[lprev(e)] = e.second;
  }

  // Digite esta função apenas para triangulações com restrições.
  void bdsm(field& F, edge e) {
    int a, b, c, d, org = e.first, dest = e.second, topo = 0;
    edge xt;
    edge pilha[TAM];
    if (count(e)) return;
    for (iterator it = lower_bound(edge(org, 0)); ; it++) {
      xt = lnext(it->first); a = xt.first, b = xt.second;
      if (b < 0) continue;
      if (seg_intersect(F.v[a], F.v[b], F.v[org], F.v[dest])) break;
    }
    while (xt.second != dest) {
      pilha[topo++] = xt; xt = sym(xt);
      xt = F.ccw(org, dest, (*this)[xt]) >= 0 ? lnext(xt) : lprev(xt);
      while (topo > 0) {
        edge ee = pilha[topo-1];
        a = ee.first; b = ee.second;
        c = (*this)[ee]; d = (*this)[sym(ee)];
        if (F.ccw(d, c, b) >= 0 || F.ccw(c, d, a) >= 0) break;
        erase(ee); erase(sym(ee)); xt = edge(d, c);
        new_tri(xt, a); new_tri(sym(xt), b);
        topo--;
        xt = F.ccw(org, dest, d) >= 0 ? lprev(xt) : lnext(sym(xt));
      }
    }
  }

  void triangulate(field& F) {
    int J[TAM], i, k, topo = 0;
    edge pilha[TAM];
    clear();
    for (int i = 0; i < F.n; i++) J[i] = i;
    sort(J, J + F.n, make_index_lt(F.v));
    for (i = 2; i < F.n; i++) if (k = F.ccw(J[0], J[1], J[i])) break;
    if (i >= F.n) return;
    for (int j = 1; j < i; j++) {
      edge e(J[j-1], J[j]);
      new_tri(e, (k > 0) ? J[i] : -1);
      new_tri(sym(e), (k > 0) ? -1 : J[i]);
    }
    edge lb(J[i], J[(k > 0) ? i-1 : 0]), ub(J[(k > 0) ? 0 : i-1], J[i]);
    for (i++; i < F.n; i++) {
      while (F.ccw(lb.first, lb.second, J[i]) >= 0) lb = dprev(lb);
      while (F.ccw(ub.first, ub.second, J[i]) >= 0) ub = dnext(ub);
      for (edge e = dnext(lb); e != ub; e = dnext(e)) pilha[topo++] = e;
      while (topo > 0) new_tri(pilha[--topo], J[i]);
      edge e(-1, J[i]);
      new_tri(e, lb.first); new_tri(sym(e), ub.second);
      lb = lnext(e); ub = dnext(lb);
    }
    // Digite esta linha somente para triangulações com restrições.
    for (i = 0; i < F.m; i++) {
      bdsm(F, edge(F.b[i].first, F.b[i].second));
    }
  }
};
```

```
////////////////////////////////////////////////////////////////////////////////
// Inteiros de precisão arbitrária /////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

#include <sstream>

const int DIG = 4;
const int BASE = 10000; // BASE**3 < 2**51
const int TAM = 2048;

struct bigint {
  int v[TAM], n;
  bigint(int x = 0): n(1) {
    memset(v, 0, sizeof(v));
    v[n++] = x; fix();
  }
  bigint(char *s): n(1) {
    memset(v, 0, sizeof(v));
    int sign = 1;
    while (*s && !isdigit(*s)) if (*s++ == '-') sign *= -1;
    char *t = strdup(s), *p = t + strlen(t);
    while (p > t) {
      *p = 0; p = max(t, p - DIG);
      sscanf(p, "%d", &v[n]);
      v[n++] *= sign;
    }
    free(t); fix();
  }
  bigint& fix(int m = 0) {
    n = max(m, n);
    int sign = 0;
    for (int i = 1, e = 0; i <= n || e && (n = i); i++) {
      v[i] += e; e = v[i] / BASE; v[i] %= BASE;
      if (v[i]) sign = (v[i] > 0) ? 1 : -1;
    }
    for (int i = n - 1; i > 0; i--)
      if (v[i] * sign < 0) { v[i] += sign * BASE; v[i+1] -= sign; }
    while (n && !v[n]) n--;
    return *this;
  }

  int cmp(const bigint& x = 0) const {
    int i = max(n, x.n), t = 0;
    while (1) if ((t = ::cmp(v[i], x.v[i])) || i-- == 0) return t;
  }
  bool operator <(const bigint& x) const { return cmp(x) < 0; }
  bool operator ==(const bigint& x) const { return cmp(x) == 0; }
  bool operator !=(const bigint& x) const { return cmp(x) != 0; }

  operator string() const {
    ostringstream s; s << v[n];
    for (int i = n - 1; i > 0; i--) {
      s.width(DIG); s.fill('0'); s << abs(v[i]);
    }
    return s.str();
  }
  friend ostream& operator <<(ostream& o, const bigint& x) {
    return o << (string) x;
  }
  bigint& operator +=(const bigint& x) {
    for (int i = 1; i <= x.n; i++) v[i] += x.v[i];
    return fix(x.n);
  }
  bigint operator +(const bigint& x) { return bigint(*this) += x; }
  bigint& operator -=(const bigint& x) {
    for (int i = 1; i <= x.n; i++) v[i] -= x.v[i];
    return fix(x.n);
  }
  bigint operator -(const bigint& x) { return bigint(*this) -= x; }
  bigint operator -() { bigint r = 0; return r -= *this; }
  void ams(const bigint& x, int m, int b) { // *this += (x * m) << b;
    for (int i = 1, e = 0; (i <= x.n || e) && (n = i + b); i++) {
      v[i+b] += x.v[i] * m + e; e = v[i+b] / BASE; v[i+b] %= BASE;
    }
  }
  bigint operator *(const bigint& x) const {
    bigint r;
    for (int i = 1; i <= n; i++) r.ams(x, v[i], i-1);
    return r;
  }
  bigint& operator *=(const bigint& x) { return *this = *this * x; }
  // cmp(x / y) == cmp(x) * cmp(y); cmp(x % y) == cmp(x);
  bigint div(const bigint& x) {
    if (x == 0) return 0;
    bigint q; q.n = max(n - x.n + 1, 0);
    int d = x.v[x.n] * BASE + x.v[x.n-1];
    for (int i = q.n; i > 0; i--) {
      int j = x.n + i - 1;
      q.v[i] = int((v[j] * double(BASE) + v[j-1]) / d);
      ams(x, -q.v[i], i-1);
      if (i == 1 || j == 1) break;
      v[j-1] += BASE * v[j]; v[j] = 0;
    }
    fix(x.n); return q.fix();
  }
  bigint& operator /=(const bigint& x) { return *this = div(x); }
  bigint& operator %=(const bigint& x) { div(x); return *this; }
  bigint operator /(const bigint& x) { return bigint(*this).div(x); }
  bigint operator %(const bigint& x) { return bigint(*this) %= x; }
  bigint pow(int x) {
    if (x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
    bigint r = 1;
    for (int i = 0; i < x; i++) r *= *this;
    return r;
  }
  bigint root(int x) {
    if (cmp() == 0 || cmp() < 0 && x % 2 == 0) return 0;
    if (*this == 1 || x == 1) return *this;
    if (cmp() < 0) return -(-*this).root(x);
    bigint a = 1, d = *this;
    while (d != 1) {
      bigint b = a + (d /= 2);
      if (cmp(b.pow(x)) >= 0) { d += 1; a = b; }
    }
    return a;
  }
};
```

```
//////////////////////////////////////////////////////////////////////////////
// Teoria dos números ////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
// Calcula o maior divisor comum dos números x e y.
//

int gcd(int x, int y) { return y ? gcd(y, x % y) : abs(x); }

//////////////////////////////////////////////////////////////////////////////
// Calcula o mínimo múltiplo comum de a e b.
//

uint64_t lcm(int x, int y) {
  if (x && y) return abs(x) / gcd(x, y) * uint64_t(abs(y));
  else return uint64_t(abs(x | y));
}

//////////////////////////////////////////////////////////////////////////////
// Decide se o inteiro n é primo.
//

bool is_prime(int n) {
  if (n < 0) return is_prime(-n);
  if (n < 5 || n % 2 == 0 || n % 3 == 0) return (n == 2 || n == 3);
  int maxP = sqrt(n) + 2;
  for (int p = 5; p < maxP; p += 6)
    if (n % p == 0 || n % (p+2) == 0) return false;
  return true;
}

//////////////////////////////////////////////////////////////////////////////
// Retorna a fatoração em números primos de abs(n).
//

#include <map>

typedef map<int, int> prime_map;

void squeeze(prime_map& M, int& n, int p) { for (; n % p == 0; n /= p) M[p]++; }

prime_map factor(int n) {
  prime_map M;
  if (n < 0) return factor(-n);
  if (n < 2) return M;
  squeeze(M, n, 2); squeeze(M, n, 3);
  int maxP = sqrt(n) + 2;
  for (int p = 5; p < maxP; p += 6) {
    squeeze(M, n, p); squeeze(M, n, p+2);
  }
  if (n > 1) M[n]++;
  return M;
}
```

```
//////////////////////////////////////////////////////////////////////////////
// Teorema de Bézout /////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
// Determina a e b tais que a * x + b * y == gcd(x, y).
//

typedef pair<int, int> bezout;

bezout find_bezout(int x, int y) {
  if (y == 0) return bezout(1, 0);
  bezout u = find_bezout(y, x % y);
  return bezout(u.second, u.first - (x/y) * u.second);
}

//////////////////////////////////////////////////////////////////////////////
// Acha a menor solução não-negativa de a*x = b (mod m).
//
// Retorna -1 se a congruência for impossível.
//

int mod(int x, int m) { return x % m + (x < 0) ? m : 0; }

int solve_mod(int a, int b, int m) {
  if (m < 0) return solve_mod(a, b, -m);
  if (a < 0 || a >= m || b < 0 || b >= m)
    return solve_mod(mod(a, m), mod(b, m), m);
  bezout t = find_bezout(a, m);
  int d = t.first * a + t.second * m;
  if (b % d) return -1;
  else return mod(t.first * (b / d), m);
}

//////////////////////////////////////////////////////////////////////////////
// Triângulo de Pascal ///////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////

const int TAM = 30;

int C[TAM][TAM];

void calc_pascal() {
  memset(C, 0, sizeof(C));
  for (int i = 0; i < TAM; i++) {
    C[i][0] = C[i][i] = 1;
    for (int j = 1; j < i; j++)
      C[i][j] = C[i-1][j-1] + C[i-1][j];
  }
}
```

```
//////////////////////////////////////////////////////////////////////////////
// Álgebra Linear ////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////

const int TAM = 100;

struct ivet {
  int m, u[TAM];

  ivet(int m = 0): m(m) {
    for (int i = 0; i < m; i++) u[i] = i;
  }
  int& operator [](int i) { return u[i]; }

  ivet operator ~() {
    ivet v(m);
    for (int i = 0; i < m; i++) v[u[i]] = i;
    return v;
  }
};

struct dvet {
  int m; double u[TAM];

  dvet(int m = 0): m(m) {
    memset(u, 0, sizeof(u));
  }
  double& operator [](int i) { return u[i]; }

  dvet operator %(ivet p) {
    dvet r(p.m);
    for (int i = 0; i < p.m; i++) r[i] = u[p[i]];
    return r;
  }

  dvet& operator +=(dvet v) {
    for (int i = 0; i < m; i++) u[i] += v[i];
    return *this;
  }
  dvet operator +(dvet v) { dvet r = v; return r += *this; }
  dvet& operator -=(dvet v) {
    for (int i = 0; i < m; i++) u[i] -= v[i];
    return *this;
  }
  dvet operator *(double t) {
    dvet r(m);
    for (int i = 0; i < m; i++) r[i] = u[i] * t;
    return r;
  }
  dvet operator -() {
    dvet r = *this;
    for (int i = 0; i < m; i++) r[i] = -r[i];
    return r;
  }
  double operator *(dvet v) {
    double r = 0;
    for (int i = 0; i < m; i++) r += u[i] * v[i];
    return r;
  }
};

struct mat {
  int m, n; dvet u[TAM];

  mat(int m = 0, int n = 0): m(m), n(n) {
    for (int i = 0; i < m; i++) u[i] = dvet(n);
  }
  dvet& operator [](int i) { return u[i]; }

  mat operator %(ivet p) {
    mat r(p.m, n);
    for (int i = 0; i < p.m; i++) r[i] = u[p[i]];
    return r;
  }
  mat operator ~() {
    mat r(n, m);
    for (int j = 0; j < n; j++)
      for (int i = 0; i < m; i++)
        r[j][i] = u[i][j];
    return r;
  }

  dvet operator *(dvet v) {
    dvet r(m);
    for (int i = 0; i < m; i++) r[i] = u[i] * v;
    return r;
  }
};
```

```
//////////////////////////////////////////////////////////////////////////////
// Sistemas lineares /////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////

struct linsys {
  ivet P, Q; dvet D; mat L, U;
  int m, n, r;

  void compile (const mat& A) {
    m = A.m; n = A.n;
    P = ivet(m); L = mat(m); D = dvet(); U = A; Q = ivet(n);
    for (r = 0; r < min(m, n); r++) {
      double best = 0; int p, q;
      for (int i = r; i < m; i++) for (int j = r; j < n; j++)
        if (cmp(fabs(U[i][j]), best) > 0)
          { p = i; q = j; best = fabs(U[i][j]); }
      if (cmp(best) == 0) break;
      swap(P[r], P[p]); swap(U[r], U[p]); swap(L[r], L[p]);
      swap(Q[r], Q[q]);
      for (int i = 0; i < m; i++) swap(U[i][r], U[i][q]);
      D[r] = 1 / U[r][r];
      U[r] = U[r] * D[r];
      for (int i = r + 1; i < m; i++) {
        L[i][r] = U[i][r] * D[r];
        U[i] -= U[r] * U[i][r];
      }
      for (int i = r; i < m; i++) U[i][r] = 0;
    }
    for (int i = 0; i < m; i++) L[i].m = r;
    L.n = D.m = U.m = r;
  }

  ////////////////////////////////////////////////////////////////////////
  // Encontra uma solução do sistema A * x = b.
  //
  // x.m = 0 caso o sistema seja impossível.
  //

  dvet solve(dvet b) {
    dvet x = b % P;
    for (int i = 0; i < m; i++) x[i] -= L[i] * x;
    for (int i = 0; i < r; i++) x[i] *= D[i];
    for (int i = r; i < m; i++) if (cmp(x[i]) != 0) return dvet();
    x.m = n;
    for (int i = r - 1; i >= 0; i--) x[i] -= U[i] * x;
    x = x % ~Q;
    return x;
  }

  ////////////////////////////////////////////////////////////////////////
  // Retorna a fatoração LU de ~A.
  //

  linsys operator ~() {
    linsys F;
    F.P = Q; F.Q = P; F.D = D; F.L = ~U; F.U = ~L;
    F.m = n; F.n = m; F.r = r;
    return F;
  }
};
```

```
//////////////////////////////////////////////////////////////////////////////
// Simplex ///////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////

struct simplex {
  int m, n, p, q;
  double s;
  dvet x, y, sx, sy, c;
  ivet N, B;
  mat AT;
  linsys F;

  simplex() {}
  simplex(dvet c): c(c), m(0), n(c.m), y(-c) {
    N.m = sy.m = AT.m = n;
    for (int j = 0; j < n; j++) {
      sy[j] = 1. + rand() / double(RAND_MAX);
      N[j] = j;
    }
  }

  ////////////////////////////////////////////////////////////////////////
  // Adiciona a restrição A*x <= b.
  //
  void constraint(dvet a, double b) {
    for (int j = 0; j < n; j++) AT[j][m] = a[j];
    AT[AT.m++][AT.n++] = 1;
    for (int k = 0; k < AT.m; k++) AT[k].m = AT.n;
    for (int i = 0; i < B.m; i++) b -= a[B[i]] * x[i];
    x[x.m++] = b;
    sx[sx.m++] = 1e-2 * (1. + rand() / double(RAND_MAX));
    B[B.m++] = n + m++;
  }

  void find_entering(int m, dvet& x, dvet& sx, int& p, int& q) {
    for (int i = 0; i < m; i++) {
      double t = - x[i] / sx[i];
      if (cmp(sx[i]) > 0 && cmp(t, s) > 0) {
        s = t; p = -1; q = i;
      }
    }
  }

  int find_leaving(int m, dvet& x, dvet& dx, dvet& sx) {
    int k = -1;
    double r = 0.;
    for (int i = 0; i < m; i++) {
      if (cmp(x[i]) == 0 && cmp(dx[i]) == 0) continue;
      double t = dx[i] / (x[i] + s * sx[i]);
      if (cmp(t, r) > 0) { r = t; k = i; }
    }
    return k;
  }

  void pivot(dvet& x, dvet& dx, int q) {
    double t = x[q] / dx[q]; x -= dx * t; x[q] = t;
  }

  double solve(dvet& r) {
    dvet dx, dy;
```

```
    while (true) {
      s = 0.; p = -1; q = -1;
      find_entering(m, x, sx, p, q);
      find_entering(n, y, sy, q, p);
      if (cmp(s) == 0) break;
      F.compile(AT % B);
      if (p != -1) {
        dx = (~F).solve(AT[N[p]]);
        q = find_leaving(m, x, dx, sx);
        if (q == -1) return INFINITY;
        dvet eq(m);
        eq[q] = -1;
        dy = (AT % N) * F.solve(eq);
      } else {
        dvet eq(m);
        eq[q] = -1;
        dy = (AT % N) * F.solve(eq);
        p = find_leaving(n, y, dy, sy);
        if (p == -1) return -INFINITY;
        dx = (~F).solve(AT[N[p]]);
      }
      pivot(x, dx, q); pivot(sx, dx, q);
      pivot(y, dy, p); pivot(sy, dy, p);
      swap(N[p], B[q]);
    }
    r = dvet(n);
    for (int i = 0; i < m; i++)
      if (B[i] < n) r[B[i]] = x[i];
    return c * r;
  }
};
```

```
////////////////////////////////////////////////////////////////////////////////
// Poker ///////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

const char* rank_names = "**23456789TJQKA";
const char* suit_names = "CDHS";

struct card {
  int rank, suit;
  int read() {
    char ch[2];
    if (scanf(" %c%c", &ch[0], &ch[1]) == EOF) return 0;
    for (rank = 0; rank_names[rank] != ch[0]; rank++);
    for (suit = 0; suit_names[suit] != ch[1]; suit++);
    return 1;
  }
  void print() { printf("%c%c", rank_names[rank], suit_names[suit]); }
};

struct freq_lt {
  int* freq;
  freq_lt(int* freq): freq(freq) {}
  bool operator ()(const card A, const card B) const {
    if (int t = freq[A.rank] - freq[B.rank]) return t > 0;
    else return A.rank > B.rank;
  }
};

struct hand {
  card C[5];
  int type() {
    int freq[15]; memset(freq, 0, sizeof(freq));

    sort(C, C+5, freq_lt(freq));
    bool flush = true, straight = true;
    for (int i = 0; i < 5; i++) {
      if (i && C[i].suit != C[i-1].suit) flush = false;
      if (i && !(C[i].rank == 5 && C[i-1].rank == 14) && \
          C[i].rank != C[i-1].rank - 1) straight = false;
      freq[C[i].rank]++;
    }
    sort(C, C+5, freq_lt(freq));

    int kind[5]; memset(kind, 0, sizeof(kind));
    for (int i = 2; i <= 14; i++) kind[freq[i]]++;
    if (straight && flush) return 8;
    else if (kind[4]) return 7;
    else if (kind[3] && kind[2]) return 6;
    else if (flush) return 5;
    else if (straight) return 4;
    else if (kind[3]) return 3;
    else return kind[2];
  }
  bool operator <(hand H) {
    if (int t = type() - H.type()) return t < 0;
    for (int i = 0; i < 5; i++)
      if (int t = C[i].rank - H.C[i].rank) return t < 0;
    return false;
  }
};
```

```
//////////////////////////////////////////////////////////////////////////////
// Grafos ////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////

#include <queue> // Apenas para Fluxos

const int VT = 1010;
const int AR = VT * VT;

struct grafo {

  ////////////////////////////////////////////////////////////////////////////
  // Definições compartilhadas.
  //

  int dest[2 * AR]; // "2 *" apenas para CFC.
  int adj[VT][2 * VT]; // "2 *" apenas para Fluxos e CFC.
  int nadj[VT], nvt, nar;

  inline int inv(int a) { return a ^ 0x1; } // Apenas para Fluxos e PP.

  ////////////////////////////////////////////////////////////////////////////
  // Definições específicas para Fluxos.
  //

  int cap[AR], fluxo[AR], ent[VT];

  inline int orig(int a) { return dest[inv(a)]; }
  inline int capres(int a) { return cap[a] - fluxo[a]; }

  ////////////////////////////////////////////////////////////////////////////
  // Definições específicas para Fluxo Máximo.
  //

  int padj[VT], lim[VT], nivel[VT], qtd[VT];

  ////////////////////////////////////////////////////////////////////////////
  // Definições específicas para Fluxo a Custo Mínimo.
  //

  int imb[VT], marc[VT], delta;
  double custo[AR], pot[VT], dist[VT];

  inline double custores(int a) {
    return custo[a] - pot[orig(a)] + pot[dest[a]];
  }

  ////////////////////////////////////////////////////////////////////////////
  // Definição específica para Conexidade.
  //

  int prof[VT];

  ////////////////////////////////////////////////////////////////////////////
  // Definições específicas para Pontos de Articulação e Pontes.
  //

  char part[VT], ponte[AR];
  int menor[VT], npart, nponte;

  ////////////////////////////////////////////////////////////////////////////
  // Definições específicas para Componentes Fortemente Conexas.
  //

  int ord[VT], comp[VT], repcomp[VT], nord, ncomp;

  inline int transp(int a) { return (a & 0x1); }

  ////////////////////////////////////////////////////////////////////////////
  // Definições específicas para 2 SAT.
  //

  inline int verd(int v) { return 2 * v + 1; }
  inline int falso(int v) { return 2 * v; }

  ////////////////////////////////////////////////////////////////////////////
  // Funções compartilhadas.
  //

  ////////////////////////////////////////////////////////////////////////////
  // Inicializa o grafo.
  //

  void inic(int n = 0) {
    nvt = n;
    nar = 0;
    memset(nadj, 0, sizeof(nadj));
    memset(imb, 0, sizeof(imb)); // Apenas para FCM
  }

  ////////////////////////////////////////////////////////////////////////////
  // Adiciona uma aresta ao grafo.
  //
  // "int u" apenas para Fluxos; "double c" apenas para FCM.
  //

  int aresta(int i, int j, int u = 0, double c = 0) {
    int ar = nar;

    custo[nar] = c; // Apenas para FCM.
    cap[nar] = u; // Apenas para Fluxos.
    dest[nar] = j;
    adj[i][nadj[i]++] = nar++;

    custo[nar] = -c; // Apenas para FCM.
    cap[nar] = 0; // Apenas para Fluxos.
    dest[nar] = i;
    adj[j][nadj[j]++] = nar++;

    return ar;
  }

  ////////////////////////////////////////////////////////////////////////////
  // Funções específicas para Fluxo Máximo.
  //

  void revbfs(int ini, int fim) {
    int i, no, viz, ar;

    queue<int> fila;
```

```
  memset(nivel, NULO, sizeof(nivel));
  memset(qtd, 0, sizeof(qtd));

  nivel[fim] = 0; fila.push(fim);

  while (!fila.empty()) {
    no = fila.front(); fila.pop();
    qtd[nivel[no]]++;

    for (i = 0; i < nadj[no]; i++) {
      ar = adj[no][i]; viz = dest[ar];
      if (cap[ar] == 0 && nivel[viz] == NULO) {
        nivel[viz] = nivel[no] + 1; fila.push(viz);
      }
    }
  }
}

int admissivel(int no) {
  while (padj[no] < nadj[no]) {
    int ar = adj[no][padj[no]];
    if (nivel[no] == nivel[dest[ar]] + 1 && capres(ar) > 0) return ar;
    padj[no]++;
  }
  padj[no] = 0;
  return NULO;
}

int retrocede(int no) {
  int i, ar, viz, menor = NULO;
  if (--qtd[nivel[no]] == 0) return NULO;

  for (i = 0; i < nadj[no]; i++) {
    ar = adj[no][i]; viz = dest[ar];
    if (capres(ar) <= 0) continue;
    if (menor == NULO || nivel[viz] < nivel[menor]) menor = viz;
  }

  if (menor != NULO) nivel[no] = nivel[menor];
  qtd[++nivel[no]]++;
  return ((ent[no] == NULO) ? no : orig(ent[no]));
}

int avanca(int no, int ar) {
  int viz = dest[ar];
  ent[viz] = ar;
  lim[viz] = min(lim[no], capres(ar));
  return viz;
}

int aumenta(int ini, int fim) {
  int ar, no = fim, fmax = lim[fim];
  while (no != ini) {
    fluxo[ar = ent[no]] += fmax;
    fluxo[inv(ar)] -= fmax;
    no = orig(ar);
  }
  return fmax;
}
```

```
////////////////////////////////////////////////////////////////////////////////
// Função específica para Fluxo a Custo Mínimo.
//
// Algoritmo de Dijkstra: O(m * log n)
//

void dijkstra(int ini) {
  int i, j, k, a;
  double d;

  priority_queue<pair<double, int> > heap;
  memset(ent, NULO, sizeof(ent));
  memset(marc, 0, sizeof(marc));

  for (i = 0; i < nvt; i++) dist[i] = INFINITY;
  heap.push(make_pair(dist[ini] = 0.0, ini));

  while (!heap.empty()) {
    i = heap.top().second; heap.pop();
    if (marc[i]) continue; marc[i] = 1;
    for (k = 0; k < nadj[i]; k++) {
      a = adj[i][k]; j = dest[a]; d = dist[i] + custores(a);
      if (capres(a) >= delta && cmp(d, dist[j]) < 0) {
        heap.push(make_pair( -(dist[j] = d), j));
        ent[j] = a;
      }
    }
  }
}

////////////////////////////////////////////////////////////////////////////////
// Função específica para Pontos de Articulação e Pontes.
//

int dfs_partponte(int no, int ent) {
  int i, ar, viz, nf = 0;

  for (i = 0; i < nadj[no]; i++) {
    ar = adj[no][i]; viz = dest[ar];

    if (prof[viz] == NULO) {
      menor[viz] = prof[viz] = prof[no] + 1;
      dfs_partponte (viz, ar); nf++;

      if (menor[viz] >= prof[no]) {
        part[no] = 1;
        if (menor[viz] == prof[viz]) ponte[ar] = ponte[inv(ar)] = 1;
      }
      else menor[no] = min(menor[no], menor[viz]);
    }
    else if (inv(ar) != ent) menor[no] = min(menor[no], prof[viz]);
  }

  return nf;
}
```

```
/////////////////////////////////////////////////////////////////////////////
// Funções específicas para Componentes Fortemente Conexas.
//
// Ordenação Topológica (duas primeiras funções).
//

void dfs_topsort(int no) {
  for (int i = 0; i < nadj[no]; i++) {
    int ar = adj[no][i], viz = dest[ar];
    if (!transp(ar) && prof[viz] == NULO) {
      prof[viz] = prof[no] + 1; dfs_topsort(viz);
    }
  }
  ord[--nord] = no;
}

void topsort() {
  memset(prof, NULO, sizeof(prof));
  nord = nvt;

  for (int i = 0; i < nvt; i++)
    if (prof[i] == NULO) {
      prof[i] = 0; dfs_topsort(i);
    }
}

void dfs_compfortcon(int no) {
  comp[no] = ncomp;
  for (int i = 0; i < nadj[no]; i++) {
    int ar = adj[no][i], viz = dest[ar];
    if (transp(ar) && comp[viz] == NULO) dfs_compfortcon(viz);
  }
}

/////////////////////////////////////////////////////////////////////////////
// Função específica para 2 SAT.
//
// Adiciona ao grafo as arestas correspondentes a clausula
// ((x = valx) ou (y = valy))
//

void clausula(int x, bool valx, int y, bool valy) {
  int hipA, teseA, hipB, teseB;

  if (valx) { hipA = falso(x); teseB = verd(x); }
  else { hipA = verd(x); teseB = falso(x); }

  if (valy) { hipB = falso(y); teseA = verd(y); }
  else { hipB = verd(y); teseA = falso(y); }

  aresta(hipA, teseA);
  aresta(hipB, teseB);
}

/////////////////////////////////////////////////////////////////////////////
// Fluxo Máximo: O(n^2 * m)
//

int maxflow(int ini, int fim) {
  int ar, no = ini, fmax = 0;

  memset(fluxo, 0, sizeof(fluxo));
  memset(padj, 0, sizeof(padj));

  revbfs(ini, fim);
  lim[ini] = INF; ent[ini] = NULO;

  while (nivel[ini] < nvt && no != NULO) {
    if ((ar = admissivel(no)) == NULO) no = retrocede(no);
    else if ((no = avanca(no, ar)) == fim) {
      fmax += aumenta(ini, fim);
      no = ini;
    }
  }
  return fmax;
}

/////////////////////////////////////////////////////////////////////////////
// Fluxo a Custo Mínimo: O(m^2 * log n * log U)
//
// Parametro global específico: imb
//

double mincostflow() {
  int a, i, j, k, l, U = 0;
  double C = 0.;

  memset(pot, 0, sizeof(pot));
  memset(fluxo, 0, sizeof(fluxo));

  for (a = 0; a < nar ; a++) {
    if (cmp(custo[a]) > 0) C += custo[a];
    U = max(cap[a], U);
  }
  for (i = 0; i < nvt; i++) U = max(imb[i], max(-imb[i], U));
  for (delta = 0x40000000; delta > U; delta /= 2);

  imb[nvt] = nadj[nvt] = 0; U *= 2 * nvt; C *= 2;
  for (i = 0; i < nvt; i++) {
    aresta(i, nvt, U, C);
    aresta(nvt, i, U, C);
  }
  nvt++;

  while (delta >= 1) {
    for (a = 0; a < nar ; a++) {
      i = orig(a); j = dest[a];
      if (delta <= capres(a) && capres(a) < 2 * delta &&
          cmp(custores(a)) < 0) {
        fluxo[inv(a)] -= capres(a);
        imb[i] -= capres(a); imb[j] += capres(a);
        fluxo[a] = cap[a];
      }
    }
```

```
    while (true) {
      for (k = 0 ; k < nvt && imb[k] < delta; k++);
      for (l = nvt - 1 ; l >= 0 && imb[l] > -delta; l--);
      if (k == nvt || l < 0) break;

      dijkstra(k);
      for (i = 0 ; i < nvt ; i++) pot[i] -= dist[i];
      for (a = ent[l]; a != NULO; a = ent[orig(a)])  {
        fluxo[a] += delta; fluxo[inv(a)] -= delta;
      }
      imb[k] -= delta; imb[l] += delta;
    }
    delta /= 2;
  }

  C = 0.;
  for (a = 0; a < nar; a++) if (fluxo[a] > 0) C += fluxo[a] * custo[a];
  return C;
}

//////////////////////////////////////////////////////////////////////////////
// Encontra os Pontos de Articulação e as Pontes.
//

void partponte() {
  memset(part, 0, sizeof(part));
  memset(ponte, 0, sizeof(ponte));
  memset(prof, NULO, sizeof(prof));
  memset(menor, NULO, sizeof(menor));
  npart = nponte = 0;

  for (int i = 0; i < nvt; i++)
    if (prof[i] == NULO) {
      menor[i] = prof[i] = 0;
      if (dfs_partponte(i, NULO) < 2) part[i] = 0;
    }
  for (int i = 0; i < nvt; i++) if (part[i]) npart++;
  for (int i = 0; i < nar; i++) if (ponte[i]) nponte++;
  nponte /= 2;
}

//////////////////////////////////////////////////////////////////////////////
// Encontra as Componentes Fortemente Conexas.
//

int compfortcon() {
  memset(comp, NULO, sizeof(comp));
  ncomp = 0;
  topsort();

  for (int i = 0; i < nvt; i++)
    if (comp[ord[i]] == NULO) {
      repcomp[ncomp] = ord[i];
      dfs_compfortcon(ord[i]);
      ncomp++;
    }

  return ncomp;
}

  //////////////////////////////////////////////////////////////////////////////
  // Decide se a conjunção das cláusulas pode ser satisfeita.
  //

  int twosat(int nvar) {
    compfortcon();
    for (int i = 0; i < nvar; i++)
      if (comp[verd(i)] == comp[falso(i)])
        return 0;
    return 1;
  }
};
```

```
////////////////////////////////////////////////////////////////////////////////
// Aho-Corasick ////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

#include <map>
#include <list>
#include <queue>
#include <string>

const int MAX_NO = 100010;
const int MAX_PAD = 1010;

typedef map<char, int> mapach;
typedef map<string, int> mapastr;

struct automato {
  mapach trans[MAX_NO];
  mapastr pad;
  list<int> pos[MAX_PAD];
  int falha[MAX_NO], final[MAX_NO], tam[MAX_PAD], numNos;

  automato(): numNos(0) {}

  //////////////////////////////////////////////////////////////////////////////
  // Função de inicialização.
  //
  // Uma chamada por instância, antes de todas as outras funções.
  //

  void inic() {
    memset(falha, NULO, sizeof(falha));
    memset(final, NULO, sizeof(final));
    for (int i = 0; i < numNos; i++) trans[i].clear();
    pad.clear(); numNos = 1;
  }

  //////////////////////////////////////////////////////////////////////////////
  // Função que adiciona um padrão ao autômato reconhecedor.
  //
  // Uma chamada por padrão, depois da inicialização.
  // Retorna o índice de acesso a variável global pos.
  //

  int adiciona_padrao(char* s) {
    pair<mapach::iterator, bool> pch;
    int i, no = 0, numPad = pad.size();

    if (pad.count(s)) return pad[s];
    else pad.insert(make_pair(s, numPad));

    for (i = 0; s[i]; i++) {
      if ((pch = trans[no].insert(make_pair(s[i], numNos))).second) numNos++;
      no = pch.first->second;
    }

    tam[numPad] = i ? i : 1;
    return final[no] = numPad;
  }

  //////////////////////////////////////////////////////////////////////////////
  // Função que gera o tratamento de falhas.
  //
  // Uma chamada por instância, depois da adição de todos os padrões.
  //

  void gera_falhas() {
    queue<int> fila;
    int filho;
    foreach (it, all(trans[0])) {
      falha[filho = it->second] = 0;
      fila.push(filho);
    }

    while (!fila.empty()) {
      int atual = fila.front(); fila.pop();

      foreach (it, all(trans[atual])) {
        char c = it->first; filho = it->second; int ret = falha[atual];

        while (ret != NULO && !trans[ret].count(c))
          ret = falha[ret];

        if (ret != NULO) {
          falha[filho] = trans[ret][c];
          if (final[filho] == NULO && final[falha[filho]] != NULO)
            final[filho] = final[falha[filho]];
        } else if (trans[0].count(c)) falha[filho] = trans[0][c];

        fila.push(filho);
      }
    }
  }

  //////////////////////////////////////////////////////////////////////////////
  // Função que busca os padrões em uma cadeia de consulta.
  //
  // Uma chamada por consulta, depois da geração do tratamento de falhas.
  // Preenche a variável global pos.
  //

  void consulta(char* s) {
    int ret, atual = 0, i = 0;

    int N = pad.size();
    for (int j = 0; j < N; j++) pos[j].clear();

    do {
      while (atual != NULO && !trans[atual].count(s[i]))
        atual = falha[atual];
      atual = (atual == NULO) ? 0 : trans[atual][s[i]];

      for (ret = atual; ret != NULO && final[ret] != NULO; ret = falha[ret]) {
        pos[final[ret]].push_back(i - tam[final[ret]] + 1);
        while (falha[ret] != NULO && final[falha[ret]] == final[ret])
          ret = falha[ret];
      }
    } while (s[i++]);
  }
};
```

```
////////////////////////////////////////////////////////////////////////////////
// Árvore de Segmentos /////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

struct segtree {
  int B, E, C;
  segtree *L, *R;
  double len;

  int a, lbd, rbd; // só para union_perimeter()

  segtree(int b, int e): B(b), E(e), len(0), C(0), a(0), lbd(0), rbd(0) {
    if (E - B > 1) {
      int M = (B + E) / 2;
      L = new segtree(B, M);
      R = new segtree(M, E);
    } else if (E - B == 1) {
      L = new segtree(B, B);
      R = new segtree(E, E);
    } else L = R = NULL;
  }
  ~segtree() { delete L; delete R; }
  void insert(int b, int e) {
    if (e <= B || E <= b || B == E) return;
    if (b <= B && E <= e) C++;
    else { L->insert(b, e); R->insert(b, e); }
    update();
  }
  void erase(int b, int e) {
    if (e <= B || E <= b || B == E) return;
    if (b <= B && E <= e) C--;
    else { L->erase(b, e); R->erase(b, e); }
    update();
  }
  void update();
};

struct rect {
  double x1, y1, x2, y2;
  rect(double x1 = 0, double y1 = 0, double x2 = 0, double y2 = 0): \
    x1(x1), y1(y1), x2(x2), y2(y2) {}
};

const int TAM = 110;

double y[2 * TAM];

void segtree::update() {
  if (C) {
    len = y[E] - y[B];
    a = 2;
    lbd = rbd = 1;
  } else {
    len = L->len + R->len;
    a = L->a + R->a - 2 * L->rbd * R->rbd;
    lbd = L->lbd; rbd = R->rbd;
  }
}
```

```
double union_area(vector<rect>& R) {
  int n = R.size(); if (n == 0) return 0;
  vector< pair<double, int> > E;
  int m = 0;
  for (int i = 0; i < n; i++) {
    E.push_back(make_pair(R[i].x1, i));
    E.push_back(make_pair(R[i].x2, ~i));
    y[m++] = R[i].y1;
    y[m++] = R[i].y2;
  }
  sort(all(E)); sort(y, y + m); m = unique(y, y + m, cmp_eq) - y;
  double last = E[0].first, r = 0;
  segtree T(0, m-1);
  for (int i = 0; i < 2*n; i++) {
    int k = E[i].second; bool in = (k >= 0); if (!in) k = ~k;

    double dx = E[i].first - last, dy = T.len;
    r += dx * dy;

    int a = lower_bound(y, y + m, R[k].y1, cmp_lt) - y;
    int b = lower_bound(y, y + m, R[k].y2, cmp_lt) - y;
    if (in) T.insert(a, b);
    else T.erase(a, b);

    last += dx;
  }
  return r;
}

double union_perimeter(vector<rect>& R) {
  int n = R.size(); if (n == 0) return 0;
  vector< pair<double, int> > E;
  int m = 0;
  for (int i = 0; i < n; i++) {
    E.push_back(make_pair(R[i].x1, i));
    E.push_back(make_pair(R[i].x2, ~i));
    y[m++] = R[i].y1;
    y[m++] = R[i].y2;
  }
  sort(all(E)); sort(y, y + m); m = unique(y, y + m, cmp_eq) - y;
  double last = E[0].first, r = 0, dy = 0;
  segtree T(0, m-1);
  for (int i = 0; i < 2*n; i++) {
    int k = E[i].second; bool in = (k >= 0); if (!in) k = ~k;

    double dx = E[i].first - last;
    r += dx * T.a;

    int a = lower_bound(y, y + m, R[k].y1, cmp_lt) - y;
    int b = lower_bound(y, y + m, R[k].y2, cmp_lt) - y;
    if (in) T.insert(a, b);
    else T.erase(a, b);

    r += fabs(dy - T.len);
    dy = T.len;
    last += dx;
  }
  return r;
}
```

```
//////////////////////////////////////////////////////////////////////////////
// Polinômios ////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////

#include <complex>
#include <vector>

typedef complex<double> cdouble;

int cmp(cdouble x, cdouble y = 0, double tol = EPS) {
  return cmp(abs(x), abs(y), tol);
}

const int TAM = 200;

struct poly {
  cdouble poly[TAM]; int n;
  poly(int n = 0): n(n) { memset(p, 0, sizeof(p)); }
  cdouble& operator [](int i) { return p[i]; }
  poly operator ~() {
    poly r(n-1);
    for (int i = 1; i <= n; i++)
      r[i-1] = p[i] * cdouble(i);
    return r;
  }
  pair<poly, cdouble> ruffini(cdouble z) {
    if (n == 0) return make_pair(poly(), 0);
    poly r(n-1);
    for (int i = n; i > 0; i--) r[i-1] = r[i] * z + p[i];
    return make_pair(r, r[0] * z + p[0]);
  }
  cdouble operator ()(cdouble z) { return ruffini(z).second; }
  cdouble find_one_root(cdouble x) {
    poly p0 = *this, p1 = ~p0, p2 = ~p1;
    int m = 1000;
    while (m--) {
      cdouble y0 = p0(x);
      if (cmp(y0) == 0) break;
      cdouble G = p1(x) / y0;
      cdouble H = G * G - p2(x) - y0;
      cdouble R = sqrt(cdouble(n-1) * (H * cdouble(n) - G * G));
      cdouble D1 = G + R, D2 = G - R;
      cdouble a = cdouble(n) / (cmp(D1, D2) > 0 ? D1 : D2);
      x -= a;
      if (cmp(a) == 0) break;
    }
    return x;
  }
  vector<cdouble> roots() {
    poly q = *this;
    vector<cdouble> r;
    while (q.n > 1) {
      cdouble z(rand() / double(RAND_MAX), rand() / double(RAND_MAX));
      z = q.find_one_root(z); z = find_one_root(z);
      q = q.ruffini(z).first;
      r.push_back(z);
    }
    return r;
  }
};
```

```
//////////////////////////////////////////////////////////////////////////////
// Karatsuba (mult. polinomios) //////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////

#include <string.h>

int aux[16][3][100010]; // a primeira dimensao deve respeitar (lg GRAU_MAX)
                        // e a terceira (2 * GRAU_MAX)
int MOD; // definir o modulo!
const int LIM = 35;

// Assume que r[] ja esta todo zerado, e p1[] e p2[] normalizados em MOD
// g1 eh o grau de p1[] e g2 o grau de p2[]. _lvl nao deve ser usado
// o x^0 estah no indice 0, e o x^n no indice n do vetor passado

// Complexidade: O(n ^ lg 3) = O(n ^ 1.58)
void karatsuba(int r[], int g1, int p1[], int g2, int p2[], int _lvl = 0) {
  if (g1 <= LIM || g2 <= LIM) { // otimizacoes constantes
    for (int i = 0; i <= g1; i++)
      for (int j = 0; j <= g2; j++)
        r[i+j] = (r[i+j] + p1[i]*p2[j])%MOD;
    return;
  }
  int grau = max(g1, g2);
  int grau2 = grau/2;

  // A0*B0
  karatsuba(r, grau2, p1, grau2, p2, _lvl+1);

  // A1*B1
  karatsuba(r+2*grau2+2, g1-grau2-1, p1+grau2+1, g2-grau2-1, p2+grau2+1, _lvl+1);

  // (A0+A1)*(B0+B1)
  for (int i = 0; i <= grau2; i++)
    aux[_lvl][0][i] = (p1[i] + ((i + grau2 + 1 <= g1)?p1[i+grau2+1]:0)) % MOD;
  for (int i = 0; i <= grau2; i++)
    aux[_lvl][1][i] = (p2[i] + ((i + grau2 + 1 <= g2)?p2[i+grau2+1]:0)) % MOD;
  memset(aux[_lvl][2], 0, (grau + 2)*sizeof(int));
  karatsuba(aux[_lvl][2], grau2, aux[_lvl][0], grau2, aux[_lvl][1], _lvl+1);

  // poe resultado em r[]
  for (int i = 0; i <= 2*grau2; i++)
    aux[_lvl][2][i] = (aux[_lvl][2][i] + 2 * MOD - r[i] - r[i+2*grau2+2]) % MOD;
  for (int i = 0; i <= 2*grau2; i++)
    r[grau2 + i + 1] = (r[grau2 + i + 1] + aux[_lvl][2][i]) % MOD;
}
```

```
//////////////////////////////////////////////////////////////////////////////////
// RMQ (Range Minimum Query) /////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////

#include <vector>

struct RMQ {
  vector<int> M, R;
  int N,I,F;

  RMQ(vector<int>& _R) {
    R=_R; N=R.size();
    M.resize(8*N+10);
    _makeTree(1,0,N);
  }

  // Retorna o minimo no intervalo [a, b) em O(log N)
  int getMin(int a, int b) {
    I=a; F=b;
    return _find(1, 0, N);
  }

  void update(int pos, int num) { // O(log N)
    R[pos] = num;
    I=pos; F=pos+1;
    _update(1, 0, N);
  }

  int _find(int node, int a, int b) { // O(log N)
    if (a >= I && b <= F) return M[node];
    if (a >= F || b <= I) return -1;
    int left = _find(2*node, a, (a+b)/2);
    int right = _find(2*node+1, (a+b)/2, b);
    if(left == -1 || right == -1) return max(left,right);
    if (R[left] <= R[right]) return left; // (*)
    return right;
  }

  void _makeTree(int node, int a, int b) { // O(4*N)
    if (a+1 == b) { M[node] = a; return; }
    _makeTree(2*node, a, (a+b)/2);
    _makeTree(2*node+1, (a+b)/2, b);
    if (R[M[2*node]] <= R[M[2*node+1]]) M[node] = M[2*node]; // (*)
    else M[node] = M[2*node+1];
  }

  void _update(int node, int a, int b) { // O(log N)
    if (a == I && b == F) { M[node] = a; return; }
    if (a >= F || b <= I) return;
    _update(2*node, a, (a+b)/2);
    _update(2*node+1, (a+b)/2, b);
    if (R[M[2*node]] <= R[M[2*node+1]]) M[node] = M[2*node]; // (*)
    else M[node] = M[2*node+1];
  }

  // (*) trocar <= por >= para MAX query
};
```

```
//////////////////////////////////////////////////////////////////////////////////
// BIT (Binary Index Tree) ///////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////

#include <vector>

struct BIT {
  vector<int> tree;
  int N;

  BIT(int _N) {
    N = _N+1;
    tree.resize(N,0);
  }

  void update(int ind, int value) { // O(log n)
    ind++;
    while (ind < N) {
      tree[ind] += value;
      ind += (ind & -ind);
    }
  }

  int get(int ind) { // O(log n)
    int sum=0;
    while (ind > 0) {
      sum += tree[ind];
      ind -= (ind & -ind);
    }
    return sum;
  }

  int get(int ind1, int ind2)  { // O(log n)
    if (ind1 >= ind2) return 0;
    return getFreq(ind2) - getFreq(ind1);
  }
};
```