Contents

This is our team notebook for ACM-ICPC and other CP contests.

Regards,
Wiwit Rifa'i
Luqman A. Siswanto

1. Templates

```
import java.util.*;
import java.io.*;
import java.lang.*;
import java.math.BigInteger;

public class TEMPLATE {
  public static void main(String[] args) {
    InputStream inputStream = System.in;
    OutputStream outputStream = System.out;
    InputReader in = new InputReader(inputStream);
    PrintWriter out = new PrintWriter(outputStream);
    Task solver = new Task();
    solver.solve(1, in, out);
    out.close();
  }
}

class Task {
  public void solve(int testNumber, InputReader in,PrintWriter out)
{

  }
}

class InputReader {
  public BufferedReader reader;
  public StringTokenizer tokenizer;

  public InputReader(InputStream stream) {
    reader = new BufferedReader(new InputStreamReader(stream),
32768);
    tokenizer = null;
  }

  public String next() {
    while (tokenizer == null || !tokenizer.hasMoreTokens()) {
      try {
        tokenizer = new StringTokenizer(reader.readLine());
      } catch (IOException e) {
        throw new RuntimeException(e);
      }
    }
    return tokenizer.nextToken();
  }
```

```
  public int nextInt() {
    return Integer.parseInt(next());
  }
  public long nextLong() {
    return Long.parseLong(next());
  }
  public double nextDouble() {
    return Double.parseDouble(next());
  }
}

//stack resize
asm( "mov %0,%%esp\n" ::"g"(mem+10000000) );
//change esp to rsp if 64-bit system

//stack resize (linux)
#include <sys/resource.h>
void increase_stack_size() {
  const rlim_t ks = 64*1024*1024;
  struct rlimit rl;
  int res=getrlimit(RLIMIT_STACK, &rl);
  if(res==0){
    if(rl.rlim_cur<ks){
      rl.rlim_cur=ks;
      res=setrlimit(RLIMIT_STACK, &rl);
    }
  }
}

// optimizer in source code (or change O3 to O2)
#pragma GCC optimize("O3")

// improve cin-cout
ios_base::sync_with_stdio(false);

// fast IO, can change getchar to getchar_unlocked in linux
template <typename t>
t getnum()
{
  t res=0, mult=1;
  char c;
  while(1) {
    c=getchar(); if(c==' ' || c=='\n') continue; else break;
  }
  if(c=='-') mult*=-1; else res+=c-'0';
  while(1) {
    c=getchar();
    if(c>='0' && c<='9') { res*=10; res+=c-'0'; }
```

```
    else break;
  }
  return res*mult;
}
```

2. Const Big Prime Number

1e9 + **9**, 1e9 + **87**, 1e9 + **4207**, 2e9 + **89**, 2e9 + **143**, 2e9 + **11**, 2e9 + **1851**, 2e9 + **2153**,
252097800623, 1e15 - **11**, 1e15 + **37**,

3. Miller Rabin Big Primality Test and Pollard's Rho Factoring

```
vector<long long> A({2, 3, 5, 7, 11, 13, 17, 19, 23});
// if n < 3,825,123,056,546,413,051, it is enough to test a = 2, 3, 5,
7, 11, 13, 17, 19, and 23.

long long largemul(long long a, long long b, long long n) {
  // assert(0 <= a && a < n && 0 <= b && b < n);
  long long r = 0;
  for (; b; b >>= 1, a <<= 1) {
    if (a >= n) a -= n;
    if (b & 1) {
      r += a;
      if (r >= n) r -= n;
    }
  }
  return r;
}

long long fastexp(long long a, long long b, long long n) {
  // assert(0 <= a && a < n && b >= 0);
  long long ret = 1;
  for (; b; b >>= 1, a = largemul(a, a, n))
    if (b & 1) ret = largemul(ret, a, n);
  return ret;
}

bool mrtest(long long n) {
  if (n == 1) return false;
  long long d = n-1;
  int s = 0;
  while ((d & 1) == 0) {
    s++;
    d >>= 1;
  }
  s--;
  if (s < 0) s = 0;
  for (int j = 0; j < (int)A.size(); j++) {
    if (A[j] >= n) continue;
    long long ad = fastexp(A[j], d, n);
    if (ad == 1) continue;
    bool notcomp = false;
    long long a2rd = ad;
    for (int r = 0; r <= s; r++) {
      if(a2rd == n-1) {notcomp = true; break;}
      a2rd = largemul(a2rd, a2rd, n);
    }
    if (!notcomp) {
      return false;
    }
  }
  return true;
}

long long gcd(long long a, long long b) { return a ? gcd(b % a, a) :
b; }

long long pollard_rho(long long n) {
  int i = 0, k = 2;
  long long x = 3, y = 3; // random seed = 3, other values possible
  while (1) {
    i++;
    x = largemul(x, x, n)-1; // generating function
    if (x < 0) x += n;
    long long d = gcd(llabs(y - x), n); // the key insight
    if (d != 1 && d != n) return d;
    if (i == k) y = x, k <<= 1;
  }
}
```

4. Extended Euclidean Algorithm

```
long long x, y, d; // ax + by = d
void extendedEuclidean(long long a, long long b) {
  if(b == 0) { x = 1; y = 0; d = a; return; }
  extendedEuclidean(b, a % b);
  long long xx, yy;
  xx = y;
  yy = x - (a/b)*y;
```

```
  x = xx; y = yy;
}
```

5. Formulas and Theorems

$$\sum_{k=0}^{n} k\binom{n}{k} = n2^{n-1}$$

$$\sum_{k=0}^{n} k^2\binom{n}{k} = (n+n^2)2^{n-2}$$

$$\sum_{j=0}^{k} \binom{m}{j}\binom{n-m}{k-j} = \binom{n}{k}$$

$$\sum_{m=0}^{n} \binom{m}{j}\binom{n-m}{k-j} = \binom{n+1}{k+1},$$

$$\sum_{m=0}^{n} \binom{m}{k} = \binom{n+1}{k+1}.$$

$$\sum_{j=0}^{m} \binom{m}{j}^2 = \binom{2m}{m}.$$

$$\sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-k}{k} = F(n+1).$$

$$\sum_{i=0}^{n} i\binom{n}{i}^2 = \frac{n}{2}\binom{2n}{n}$$

$$\sum_{i=0}^{n} i^2\binom{n}{i}^2 = n^2\binom{2n-2}{n-1}.$$

$$\sum_{k=q}^{n} \binom{n}{k}\binom{k}{q} = 2^{n-q}\binom{n}{q}$$

Lucas' Theorem :

For non-negative integers *m* and *n* and a prime *p*, the following congruence relation holds:

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

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

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

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

Example : (combinatrics in small mod wheren mod < n && mod < k)

```
int comb[mod][mod];
int c(int n, int k) {
  return n == 0? 1 : comb[n%mod][k%mod] * c(n/mod, k/mod) % mod;
}
```

Faulhaber's Formula:

$$(n+1)^{k+1} - 1 = \sum_{m=1}^{n} \left((m+1)^{k+1} - m^{k+1}\right) = \sum_{p=0}^{k} \binom{k+1}{p} (1^p + 2^p + \cdots + n^p)$$

Examples:

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} = \frac{2n^3+3n^2+n}{6}$$

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4+2n^3+n^2}{4}$$

$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{6n^5+15n^4+10n^3-n}{30}$$

$$1^5 + 2^5 + 3^5 + \cdots + n^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12} = \frac{2n^6+6n^5+5n^4-n^2}{12}$$

$$1^6 + 2^6 + 3^6 + \cdots + n^6 = \frac{n(n+1)(2n+1)(3n^4+6n^3-3n+1)}{42} = \frac{6n^7+21n^6+21n^5-7n^3+n}{42}$$

Cayley's Formula: There are $n^{(n-2)}$ spanning trees of a complete graph with n label vertices.

Derangement: A permutation of the elements of a set that none of the elements appear in their original position. $d(n) = (n-1)\, x\, (d(n-1) + d(n-2))$ where $d(0) = 1,\ d(1) = 0.$

Erdos Gallai's Theorem: A sequence of non-negative numbers $d_1 \geq d_2 \geq \ldots \geq d_n$ can be the degree sequence of a **simple** graph on n vertices iff $\sum d_i$ is even

and $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$ is hold for $1 \leq k \leq n$.

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

Moser's Circle: Determine the number of pieces into a circle is divided if n points on its circumference are joined by chords with no three internally concurrent. Solution: $g(n) = {}^{n}C_4 + {}^{n}C_2 + 1$.

Pick's Theorem: provides a simple formula for calculating the area *A* of this polygon in terms of the number *i* of *lattice points in the interior*located in the polygon and the number *b* of *lattice points on the boundary* placed on the polygon's perimeter: $A = i + \frac{b}{2} - 1.$

The number of **spanning tree** of a **complete bipartite graph** $\mathbf{K_{n,\,m}}$ is $m^{n-1} x\, n^{m-1}$

Burnside's lemma can be used to count the number of combinations so that one representative is counted for each group of symmetric combinations. Burnside's lemma state that the number of combinations is

$\sum_{k=1}^{n} \frac{c(k)}{n}$ where there are n ways to change the position of a combination, and there are c(k) combinations that remain unchanged when *k*th way is applied

Catalan Number:

$$C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\binom{2n}{n} \quad \text{for } n \geq 0,$$

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^{n} C_i\, C_{n-i} \quad \text{for } n \geq 0; \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n,$$

Cat(n) can represents:

1. The number of distinct binary trees with n vertices.
2. The number of expressions containing n pairs of parentheses which are corrcetly matched.
3. The number of different ways n+1 factors can be completly parenthesized, e.g. for n = 3 and 3+1=4 factors:{a, b, c, d}, we have (ab)(cd), a(b(cd)), ((ab)c)d, (a(bc))d, and a((bc)d).
4. The number of ways a convex polygon of n+2 sides can be triangulated.
5. The number of monothonic paths along the edges of an nxn grid, which do not pass above the diagonal.

6. Strassen Matrix Multiplication Optimization

Complexity: $O([7+o(1)]^n) = O(N^{\log_2 7 + o(1)}) \approx O(N^{2.8074})$

$$\mathbf{C} = \mathbf{A}\mathbf{B} \qquad \mathbf{A}, \mathbf{B}, \mathbf{C} \in R^{2^n \times 2^n}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \ \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \ \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

$$\mathbf{M}_1 := (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \quad \mathbf{M}_2 := (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1}$$

$$\mathbf{M}_3 := \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \quad \mathbf{M}_4 := \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$

$$\mathbf{M}_5 := (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \quad \mathbf{M}_6 := (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$

$$\mathbf{M}_7 := (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})$$

$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \quad \mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$

$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4 \quad \mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

7. Fast Fourier Transform

```
/***************** FFT dengan complex *****************/
typedef complex<double> cd;
typedef vector< cd > vcd;

// asumsi ukuran as = 2^k, dengan k bilangan bulat positif
vcd fft(const vcd &as) {
  int n = (int)as.size();
  int k = 0;
  while((1<<k) < n) k++;
  vector< int > r(n);
  r[0] = 0;
  int h = -1;
  for(int i = 1; i<n; i++) {
    if((i & (i-1)) == 0)
      h++;
    r[i] = r[i ^ (1 << h)];
    r[i] |= (1<<(k-h-1));
  }
  vcd root(n);
  for(int i = 0; i<n; i++) {
    double ang = 2.0*M_PI*i/n;
    root[i] = cd(cos(ang), sin(ang));
  }

  vcd cur(n);
  for(int i = 0; i<n; i++)
    cur[i] = as[r[i]];

  for(int len = 1; len < n; len <<= 1 ) {
    vcd ncur(n);
    int step = n/(len << 1);
    for(int pdest = 0; pdest <n;) {
      for(int i = 0; i<len; i++) {
        cd val = root[i*step]*cur[pdest + len];
        ncur[pdest] = cur[pdest] + val;
        ncur[pdest + len] = cur[pdest] - val;
        pdest++;
      }
      pdest += len;
    }
    cur.swap(ncur);
  }
  return cur;
}

vcd inv_fft(const vcd& fa) {
  vcd res = fft(fa);
  for(int i = 0; i<nn; i++) {
    res[i] /= nn;
  }
  reverse(res.begin() + 1, res.end());
  return res;
}

/*************** FFT dengan Modular Aritmetic ***************/
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1<<20;

void fft (vector<int> & a, bool invert) {
  int n = (int) a.size();

  for (int i=1, j=0; i<n; ++i) {
    int bit = n >> 1;
    for (; j>=bit; bit>>=1)
      j -= bit;
    j += bit;
    if (i < j)
      swap (a[i], a[j]);
  }

  for (int len=2; len<=n; len<<=1) {
    int wlen = invert ? root_1 : root;
    for (int i=len; i<root_pw; i<<=1)
      wlen = int (wlen * 1ll * wlen % mod);
    for (int i=0; i<n; i+=len) {
      int w = 1;
      for (int j=0; j<len/2; ++j) {
        int u = a[i+j],  v = int (a[i+j+len/2] * 1ll * w % mod);
        a[i+j] = u+v < mod ? u+v : u+v-mod;
        a[i+j+len/2] = u-v >= 0 ? u-v : u-v+mod;
        w = int (w * 1ll * wlen % mod);
      }
    }
  }
  if (invert) {
    int nrev = reverse (n, mod);
    for (int i=0; i<n; ++i)
      a[i] = int (a[i] * 1ll * nrev % mod);
  }
}
```

Optimization Note: FFT for two polynomials simultaneously. Let $A(x), B(x)$ be the polynomials with real quotients. Consider $P(x) = A(x) + iB(x)$. Note that $\overline{P(\overline{x})} = A(x) - iB(x)$, thus $A(w_k) = \frac{P(w_k) + \overline{P(w_{n-k})}}{2}, B(w_k) = \frac{P(w_k) - \overline{P(w_{n-k})}}{2i}$.

Now backwards. Assume we know values of A, B and know they have real quotients. Calculate inverse FFT for $P = A + iB$. Quotients for *A* will be real part and quotients for *B* will be imaginary part.

8. Fast Walsh-Hadamart Transform

```
// Walsh-Hadamart Matrix: This is for polynom multiplication but with
custom operation on the power of x instead of addition.
// xor : 1/sqrt(2) * {{1, 1}, {1, -1}}, inverse : same
// and : {{0, 1}, {1, 1}}, inverse : {{-1, 1}, {1, 0}}
// or : {{1, 1}, {1, 0}}, inverse : {{0, 1}, {1, -1}}
poly FWHT(poly P, bool inverse) { // example: xor
  for (len = 1; 2 * len <= degree(P); len <<= 1) {
    for (i = 0; i < degree(P); i += 2 * len) {
      for (j = 0; j < len; j++) {
        u = P[i + j];
        v = P[i + len + j];
        if (!inverse) {
          P[i + j] = u + v; // xor's matrix
          P[i + len + j] = u - v;  // xor's matrix
        }
        else {
          P[i + j] = u + v; // use inverse matrix here
          P[i + len + j] = u - v;  // use inverse matrix here
        }
      }
    }
  }
  if (inverse) {
    for (i = 0; i < degree(P); i++)
      P[i] = P[i] / degree(P); // this for xor only
  }
  return P;
}
//source:
https://csacademy.com/blog/fast-fourier-transform-and-variations-of-it
```

9. Chinese Remainder Theorem

```
// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
// Return (z,M).  On failure, M = -1.
PII chinese_remainder_theorem(int x,int a,int y,int b){
  int s, t;
  int d = extended_euclid(x, y, s, t);
  if(a%d != b%d)return make_pair(0,-1);
  return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &x,const VI &a){
  PII ret = make_pair(a[0], x[0]);
  for(int i =1; i < x.size(); i++){
    ret = chinese_remainder_theorem(ret.second, ret.first, x[i],
a[i]);
    if(ret.second ==-1) break;
  }
  return ret;
}
```

10. Gaussian Elimination

```
// 1 equation/column, return {has answer, N variable}
pair<bool, vector<double>> gauss(vector< vector<double> >& A) {
  int n = A.size();
  for (int i=0; i<n; i++) {
    double maxEl = abs(A[i][i]);
    int maxRow = i;
    for (int k=i+1; k<n; k++) {
      if (abs(A[k][i]) > maxEl) {
        maxEl = abs(A[k][i]);
        maxRow = k;
      }
    }
    for (int k=i; k<n+1;k++) {
      double tmp = A[maxRow][k];
      A[maxRow][k] = A[i][k];
      A[i][k] = tmp;
    }
```

```
    for (int k=i+1; k<n; k++) {
      double c = -A[k][i]/A[i][i];
      for (int j=i; j<n+1; j++) {
        if (i==j) {
          A[k][j] = 0;
        } else {
          A[k][j] += c * A[i][j];
        }
      }
    }
  }
  vector<double> x(n);
  for (int i=n-1; i>=0; i--) {
    if (abs(A[i][i]) < eps) return {false, x};
    x[i] = A[i][n]/A[i][i];
    for (int k=i-1;k>=0; k--) {
      A[k][n] -= A[k][i] * x[i];
    }
  }
  return {true, x};
}
```

11. Simplex

```
typedef long double LD;
typedef vector<LD> VD;
typedef vector<VD> VVD;

const LD EPS = 1e-9;

inline bool eq(LD a, LD b) { return fabs(a - b) < EPS; }
inline bool lt(LD a, LD b) { return a + EPS < b; }
inline bool le(LD a, LD b) { return a < b + EPS; }

struct simplex {
  VVD a;
  VD b, c, res;
  LD v;
  int n, m, status; // -2 : not started, -1 : fail, 0 : ok, 1 :
infinity;
  vector<int> left, up, pos;
  simplex() {}

  void set(VVD & A, VD & B, VD & C) {
    n = C.size(); m = A.size(); left.resize(m);
    up.resize(n); pos.resize(n); res.resize(n);
    status = -2; v = 0; a = A; b = B; c = C;
  }

  void pivot(int x, int y) {
    swap(left[x], up[y]);
    LD k = a[x][y];
    a[x][y] = 1;
    b[x] /= k;
    int cur = 0;
    for (int i = 0; i < n; i++) {
      a[x][i] = a[x][i] / k;
      if (!eq(a[x][i], 0))
        pos[cur++] = i;
    }

    for (int i = 0; i < m; i++) {
      if (i == x || eq(a[i][y], 0)) continue;
      LD cof = a[i][y];
      b[i] -= cof * b[x];
      a[i][y] = 0;
      for (int j = 0; j < cur; j++)
        a[i][pos[j]] -= cof * a[x][pos[j]];
    }
    LD cof = c[y];
    v += cof * b[x];
    c[y] = 0;
    for (int i = 0; i < cur; i++) {
      c[pos[i]] -= cof * a[x][pos[i]];
    }
  }

  void solve() {
    for (int i = 0; i < n; i++)
      up[i] = i;
    for (int i = 0; i < m; i++)
      left[i] = i + n;
    while (1) {
      int x = -1;
      for (int i = 0; i < m; i++)
        if (lt(b[i], 0) && (x == -1 || b[i] < b[x])) {
          x = i;
        }
      if (x == -1) break;
      int y = -1;
      for (int j = 0; j < n; j++)
        if (lt(a[x][j], 0)) {
          y = j;
          break;
        }
```

```
      if (y == -1) {
        status = -1;
        return;
        assert(false); // no solution
      }
      pivot(x, y);
    }
    while (1) {
      int y = -1;
      for (int i = 0; i < n; i++)
        if (lt(0, c[i])  && (y == -1 || (c[i] > c[y]))) {
            y = i;
        }
      if (y == -1) break;
      int x = -1;
      for (int i = 0; i < m; i++) {
        if (lt(0, a[i][y])) {
          if (x == -1 || (b[i] / a[i][y] < b[x] / a[x][y])) {
            x = i;
          }
        }
      }
      if (x == -1) {
        status = 1;
        return;
        assert(false); // infinite solution
      }
      pivot(x, y);
    }
    res.assign(n, 0);
    for (int i = 0; i < m; i++) {
      if (left[i] < n) {
        res[left[i]] = b[i];
      }
    }
    status = 0;
  }
  // HOW TO USE ::
  // -- call init(n, m)
  // -- call solve()
  // -- variables in "up" equals to zero
  // -- variables in "left" equals to b
  // -- max: c * x
  // -- b[i] >= a[i] * x
  // -- answer in "v"
  // -- sertificate in "res"
};
```

12. Maxflow Dinic

```
struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct Dinic {
  int N;
  vector<vector<Edge> > G;
  vector<Edge *> dad;
  vector<int> Q;

  Dinic(int N) : N(N), G(N), dad(N), Q(N) {}

  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }

  long long BlockingFlow(int s, int t) {
    fill(dad.begin(), dad.end(), (Edge *) NULL);
    dad[s] = &G[0][0] - 1;

    int head = 0, tail = 0;
    Q[tail++] = s;
    while (head < tail) {
      int x = Q[head++];
      for (int i = 0; i < G[x].size(); i++) {
    Edge &e = G[x][i];
    if (!dad[e.to] && e.cap - e.flow > 0) {
      dad[e.to] = &G[x][i];
      Q[tail++] = e.to;
    }
      }
    }
    if (!dad[t]) return 0;

    long long totflow = 0;
    for (int i = 0; i < G[t].size(); i++) {
      Edge *start = &G[G[t][i].to][G[t][i].index];
      int amt = INF;
      for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
    if (!e) { amt = 0; break; }
    amt = min(amt, e->cap - e->flow);
      }
```

```
      if (amt == 0) continue;
      for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
    e->flow += amt;
    G[e->to][e->index].flow -= amt;
      }
      totflow += amt;
    }
    return totflow;
  }

  long long GetMaxFlow(int s, int t) {
    long long totflow = 0;
    while (long long flow = BlockingFlow(s, t))
      totflow += flow;
    return totflow;
  }
};
```

13. Minimum Cost Max Flow (Negative Cost)

```
/** Max Flow Min Cost **/
/* complexity: O(min(E^2 V log V, E log V F)) */
const int maxnodes = 2010;


int nodes = maxnodes;
int prio[maxnodes], curflow[maxnodes], prevedge[maxnodes],
prevnode[maxnodes], q[maxnodes], pot[maxnodes];
bool inqueue[maxnodes];

const int INF = 1e9;
struct Edge {
  int to, f, cap, cost, rev;
};

vector<Edge> graph[maxnodes];

void addEdge(int s,int t,int cap,int cost){
  Edge a ={t,0, cap, cost, graph[t].size()};
  Edge b ={s,0,0,-cost, graph[s].size()};
  graph[s].push_back(a);
  graph[t].push_back(b);
}

void bellmanFord(int s,int dist[]){
  fill(dist, dist + nodes,1000000000);
  dist[s]=0;
  int qt =0;
  q[qt++]= s;
  for(int qh =0;(qh - qt)% nodes !=0; qh++){
    int u = q[qh % nodes];
    inqueue[u]=false;
    for(int i =0; i <(int) graph[u].size(); i++){
      Edge &e = graph[u][i];
      if(e.cap <= e.f)continue;
      int v = e.to;
      int ndist = dist[u]+ e.cost;
      if(dist[v]> ndist){
        dist[v]= ndist;
        if(!inqueue[v]){
          inqueue[v]=true;
          q[qt++% nodes]= v;
        }
      }
    }
  }
}


pair<int, int> minCostFlow(int s,int t,int maxf){
  // bellmanFord can be safely commented if edges costs are
non-negative
  bellmanFord(s, pot);
  int flow =0;
  int flowCost =0;
  while(flow < maxf){
    priority_queue<ll, vector<ll>, greater<ll>> q;
    q.push(s);
    fill(prio, prio + nodes, INF);
    prio[s]=0;
    curflow[s]= INF;
    while(!q.empty()){
      ll cur = q.top();
      int d = cur >>32;
      int u = cur;
      q.pop();
      if(d != prio[u])continue;
      for(int i =0; i <(int) graph[u].size(); i++){
        Edge &e = graph[u][i];
        int v = e.to;
        if(e.cap <= e.f)continue;
        int nprio = prio[u]+ e.cost + pot[u]- pot[v];
        if(prio[v]> nprio){
          prio[v]= nprio;
```

```
          q.push(((ll) nprio <<32)+ v);
          prevnode[v]= u;
          prevedge[v]= i;
          curflow[v]= min(curflow[u], e.cap - e.f);
        }
      }
    }
    if(prio[t] == INF)break;
    for(int i =0; i < nodes; i++) pot[i]+= prio[i];
    int df = min(curflow[t], maxf - flow);
    flow += df;
    for(int v = t; v != s; v = prevnode[v]){
      Edge &e = graph[prevnode[v]][prevedge[v]];
      e.f += df;
      graph[v][e.rev].f -= df;
      flowCost += df * e.cost;
    }
  }
  return make_pair(flow, flowCost);
}


/* usage example:
* addEdge (source, target, capacity, cost)
* minCostFlow(source, target, INF) -><flow, flowCost>
*/
```

14. Maximum Cardinality Bipartite Matching

```
// The code below finds a augmenting path:
bool dfs(int v){// v is in X, it returns true if and only if there is
an augmenting path starting from v
  if(mark[v])
    return false;
  mark[v] = true;
  for(auto &u : adj[v])
    if(match[u] == -1 or dfs(match[u])) // match[i] = the vertex i is
matched with in the current matching, initially -1
      return matched[v] = u, match[u] = v, true;
  return false;
}
```

An easy way to solve the problem is:

```
 for(int i = 0;i < n;i ++)if(matched[i] == -1){
    memset(mark, false, sizeof mark);
    dfs(i);
 }
```

But there is a faster way:

```
 while(true){
    memset(mark, false, sizeof mark);
    bool fnd = false;
    for(int i = 0;i < n;i ++) if(matched[i] == -1 && !mark[i])
      fnd |= dfs(i);
    if(!fnd)
      break;
 }
```

15. Blossom (Maximum Simple Graph Matching)

```
/*
GETS:
V->number of vertices
E->number of edges
pair of vertices as edges (vertices are 1..V)

GIVES:
output of edmonds() is the maximum matching
match[i] is matched pair of i (-1 if there isn't a matched pair)
*/

#include <bits/stdc++.h>
using namespace std;
const int M=505;
struct struct_edge{int v;struct_edge* n;};
typedef struct_edge* edge;
struct_edge pool[M*M*2];
edge top=pool,adj[M];
int V,E,match[M],qh,qt,q[M],father[M],base[M];
bool inq[M],inb[M],ed[M][M];
void add_edge(int u,int v) {
  top->v=v,top->n=adj[u],adj[u]=top++;
  top->v=u,top->n=adj[v],adj[v]=top++;
}
int LCA(int root,int u,int v) {
  static bool inp[M];
  memset(inp,0,sizeof(inp));
  while(1) {
    inp[u=base[u]]=true;
    if (u==root) break;
    u=father[match[u]];
```

```
  }
  while(1) {
    if (inp[v=base[v]]) return v;
    else v=father[match[v]];
  }
}
void mark_blossom(int lca,int u) {
  while (base[u]!=lca) {
    int v=match[u];
    inb[base[u]]=inb[base[v]]=true;
    u=father[v];
    if (base[u]!=lca) father[u]=v;
  }
}
void blossom_contraction(int s,int u,int v) {
  int lca=LCA(s,u,v);
  memset(inb,0,sizeof(inb));
  mark_blossom(lca,u);
  mark_blossom(lca,v);
  if (base[u]!=lca)
    father[u]=v;
  if (base[v]!=lca)
    father[v]=u;
  for (int u=0;u<V;u++)
    if (inb[base[u]]) {
      base[u]=lca;
      if (!inq[u])
        inq[q[++qt]=u]=true;
    }
}
int find_augmenting_path(int s) {
  memset(inq,0,sizeof(inq));
  memset(father,-1,sizeof(father));
  for (int i=0;i<V;i++) base[i]=i;
  inq[q[qh=qt=0]=s]=true;
  while (qh<=qt) {
    int u=q[qh++];
    for (edge e=adj[u];e;e=e->n) {
    int v=e->v;
    if (base[u]!=base[v]&&match[u]!=v)
      if ((v==s)||(match[v]!=-1 && father[match[v]]!=-1))
        blossom_contraction(s,u,v);
      else if (father[v]==-1) {
        father[v]=u;
        if (match[v]==-1)
          return v;
        else if (!inq[match[v]])
          inq[q[++qt]=match[v]]=true;
      }
    }
  }
  return -1;
}
int augment_path(int s,int t) {
  int u=t,v,w;
  while (u!=-1) {
    v=father[u];
    w=match[v];
    match[v]=u;
    match[u]=v;
    u=w;
  }
  return t!=-1;
}
int edmonds() {
  int matchc=0;
  memset(match,-1,sizeof(match));
  for (int u=0;u<V;u++)
    if (match[u]==-1)
      matchc+=augment_path(u,find_augmenting_path(u));
  return matchc;
}
int main() {
  int u,v;
  cin>>V>>E;
  while(E--) {
    cin>>u>>v;
    if (!ed[u-1][v-1]) {
      add_edge(u-1,v-1);
      ed[u-1][v-1]=ed[v-1][u-1]=true;
    }
  }
  cout<<edmonds()<<endl;
  for (int i=0;i<V;i++)
    if (i<match[i])
      cout<<i+1<<" "<<match[i]+1<<endl;
  return 0;
}
```

16. Minimum Cut Stoer - Wagner

```
// Adjacency matrix implementation of Stoer-Wagner min cut algorithm.
//
// Running time:
```

```
//  O(|V|^3)
//
// INPUT:
//   - graph, constructed using AddEdge()
//
// OUTPUT:
//   - (min cut value, nodes in half of min cut)

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

const int INF =1000000000;

pair<int, VI> GetMinCut(VVI &weights){
  int N = weights.size();
  VI used(N), cut, best_cut;
  int best_weight =-1;

  for(int phase = N-1; phase >=0; phase--){
    VI w = weights[0];
    VI added = used;
    int prev, last =0;
    for(int i =0; i < phase; i++){
      prev = last;
      last =-1;
      for(int j =1; j < N; j++)
        if(!added[j]&&(last ==-1|| w[j]> w[last])) last = j;
      if(i == phase-1){
        for(int j =0; j < N; j++) weights[prev][j]+=
weights[last][j];
        for(int j =0; j < N; j++) weights[j][prev]=
weights[prev][j];
        used[last]=true;
        cut.push_back(last);
        if(best_weight ==-1|| w[last]< best_weight){
          best_cut = cut;
          best_weight = w[last];
        }
      }else{
        for(int j =0; j < N; j++)
          w[j]+= weights[last][j];
        added[last]=true;
```

```
      }
    }
  }
  return make_pair(best_weight, best_cut);
}
```

17. Finding Cut Vertices & Cut Edges

```
// Tarjan version again, for undirected graph
void dfs(int v) {
  low[v]= num[v] = ++cntr;
  for(auto u : adj[v]) {
    if(num[u] == -1) {
      par[u] = v;
      if(v == Root) rootChild++;

      dfs(u);

      if(low[u] >= num[v])
        articulation_vertex[v] = true;
      if(low[u] > num[v])
        printf("Edge (%d %d) is a bridge\n", v, u);

      low[v] = min(low[v], low[u]);
    }
    else if(u != parent[v])
      low[v] = min(low[v],num[u]);//be careful!num[u] not low[u]
  }
}
// Inside Main
cntr = 0;
num.assign(n, -1);
low.assign(n, 0);
par.assign(n, -1);
articulation_vertex.assign(n, 0);
for(int i = 0; i<n; i++) if(num[i] == -1) {
  Root = i;
  rootChild = 0;
  dfs(i);
  articulation_vertex[i] = (rootChild > 1);
}
```

18. Biconnected Component

```
void dfs(int v, int bef = -1) {
```

```
  num[v] = low[v] = counter++;
  for (int u : adj[v]) {
    if (num[u] == -1) {
      edge.emplace_back(v, u);
      if (v == root)
        childroot++;
      dfs(u, v);
      if (childroot > 1 && v == root) {
        artp[v] = 1;
        while (edge.size() > 0) {
          auto it = edge.back(); edge.pop_back();
          block[nblock].push_back(it);
          if (it == make_pair(v, u))
            break;
        }
        nblock++;
      }
      if (low[u] >= num[v] && v != root) {
        artp[v] = 1;
        while (edge.size() > 0) {
          auto it = edge.back(); edge.pop_back();
          block[nblock].push_back(it);
          if (it == make_pair(v, u))
            break;
        }
        nblock++;
      }
      if (low[u] > num[v])
        bridge.emplace_back(u, v);
      low[v] = min(low[v], low[u]);
    }
    else if (bef != u && num[v] > num[u]) {
      low[v] = min(low[v], num[u]);
      edge.emplace_back(v, u);
    }
  }
}
int main() {
  for (int i = 0; i < gr.n; i++) if (gr.num[i] == -1) {
    root = i;
    childroot = 0;
    edge.clear();
    dfs(i);
    artp[i] = childroot > 1;
    if (edge.size() > 0) {
      while (edge.size() > 0) {
        auto it = edge.back(); edge.pop_back();
        block[nblock].push_back(it);
      }
      nblock++;
    }
  }
}
```

19. Strongly Connected Component

```
/****** Tarjan's SCC *******/ for directed graph
vector< int > num, low, S, vis;
int cntr, numCC;

void tarjanSCC(int v) {
  low[v] = num[v] = ++cntr;
  vis[v] = 1;
  S.push_back(v);
  for(auto u : adj[v]) {
    if(num[u] == -1)
      tarjanSCC(u);
    if(vis[u])
      low[v] = min(low[v], low[u]);
  }
  if(low[v] == num[v]) {
    printf("SCC %d :", ++numCC);
    while(1) {
      int u = S.back(); S.pop_back(); vis[u] = 0;
      printf(" %d", u);
      if(u == v)
        break;
    }
  }
}

// In MAIN();
  num.assign(n, -1);
  low.assign(n, 0);
  vis.assign(n, 0);
  cntr = numCC = 0;
  for(int i = 0; i<n; i++)
    if(num[i] == -1)
      tarjanSCC(i);
```

20. Dominator Tree

```
const int MAXN = 100010;
```

```
struct DominatorTree{
#define REP(i,s,e) for(int i=(s);i<=(e);i++)
#define REPD(i,s,e) for(int i=(s);i>=(e);i--)
  int n , m , s;
  vector< int > g[ MAXN ] , pred[ MAXN ];
  vector< int > cov[ MAXN ];
  int dfn[ MAXN ] , nfd[ MAXN ] , ts;
  int par[ MAXN ];
  int sdom[ MAXN ] , idom[ MAXN ];
  int mom[ MAXN ] , mn[ MAXN ];
  inline bool cmp( int u , int v )
  { return dfn[ u ] < dfn[ v ]; }
  int eval( int u ){
    if( mom[ u ] == u ) return u;
    int res = eval( mom[ u ] );
    if(cmp( sdom[ mn[ mom[ u ] ] ] , sdom[ mn[ u ] ] ))
      mn[ u ] = mn[ mom[ u ] ];
    return mom[ u ] = res;
  }
  void init( int _n , int _m , int _s ){
    ts = 0; n = _n; m = _m; s = _s;
    REP( i, 1, n ) g[ i ].clear(), pred[ i ].clear();
  }
  void addEdge( int u , int v ){
    g[ u ].push_back( v );
    pred[ v ].push_back( u );
  }
  void dfs( int u ){
    ts++;
    dfn[ u ] = ts;
    nfd[ ts ] = u;
    for( int v : g[ u ] ) if( dfn[ v ] == 0 ){
      par[ v ] = u;
      dfs( v );
    }
  }
  void build(){
    REP( i , 1 , n ){
      dfn[ i ] = nfd[ i ] = 0;
      cov[ i ].clear();
      mom[ i ] = mn[ i ] = sdom[ i ] = i;
    }
    dfs( s );
    REPD( i , n , 2 ){
      int u = nfd[ i ];
      if( u == 0 ) continue ;
      for( int v : pred[ u ] ) if( dfn[ v ] ){
        eval( v );
        if( cmp( sdom[ mn[ v ] ] , sdom[ u ] ) )
          sdom[ u ] = sdom[ mn[ v ] ];
      }
      cov[ sdom[ u ] ].push_back( u );
      mom[ u ] = par[ u ];
      for( int w : cov[ par[ u ] ] ){
        eval( w );
        if( cmp( sdom[ mn[ w ] ] , par[ u ] ) )
          idom[ w ] = mn[ w ];
        else idom[ w ] = par[ u ];
      }
      cov[ par[ u ] ].clear();
    }
    REP( i , 2 , n ){
      int u = nfd[ i ];
      if( u == 0 ) continue ;
      if( idom[ u ] != sdom[ u ] )
        idom[ u ] = idom[ idom[ u ] ];
    }
  }
} domT;
```

21. Geometry: Point

```
typedef long double LD;
const LD EPS = 1e-9, PI = acos(-1);
inline bool eq(LD a, LD b) { return fabs(a-b) < EPS; }
inline bool lt(LD a, LD b) { return a + EPS < b; }
inline bool le(LD a, LD b) { return a < b + EPS; }
inline int sign(LD x) { return eq(x, 0) ? 0 : (x < 0 ? -1 : 1); }

struct point {
  LD x, y;
  point(LD x = 0, LD y = 0) : 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); }
  point operator*(LD s) { return point(x*s, y*s); }
  point operator/(LD s) { return point(x/s, y/s); }
  LD operator*(const point& p) const { return x*p.x + y*p.y; } //
dot
  LD operator%(const point& p) const { return x*p.y - y*p.x; } //
cross
  LD norm_sq() { return *this * *this; }
  LD norm() { return sqrt(*this * *this); }
  point rotate(LD cs, LD sn) { return point(x*cs-y*sn, x*sn+y*cs); }
```

```
  point rotate(LD angle) { return rotate(cos(angle), sin(angle)); }
  bool operator==(const point& p) const { return eq(x, p.x) && eq(y,
p.y); }
  bool operator<(const point& p) const { return eq(y, p.y) ? x < p.x
: y < p.y; }
};
int ccw(point a, point b, point c) { return sign((b - a) % (c - b));
}
LD dist(point a, point b) { return (b-a).norm(); }
LD dist2(point a, point b) { return (b-a).norm_sq(); }
LD angle(point a, point o, point b) {
  point oa = a-o, ob = b-o;
  return acos(oa*ob/(oa.norm()*ob.norm()));
}
point bisector(point a, point b) { return a * b.norm() + b *
a.norm(); }
```

22. Geometry: Line

```
struct line {
  point a, ab; // p(t) = a + ab * t
  line(point ta, point tb) {
    if (tb < ta) swap(ta, tb);
    a = ta; ab = tb-ta;
  }
  point b() { return a + ab; }
  bool isLine() { return !(ab == point()); } // minor
  operator bool() { return !(ab == point()); } // minor
  // Line
  bool onLine(point p) {
    if (ab == point()) return a == p;
    return eq(ab % (p-a), 0);
  }
  LD distLine(point p) {
    if (ab == point()) return dist(a, p);
    return fabs((p-a) % ab)/ab.norm();
  }
  point projection(point p) {
    if (ab == point()) return a;
    return a + ab * ((p-a) * ab / ab.norm_sq());
  }
  point reflection(point p) {
    return projection(p) * 2.0 - p;
  }
  // Segment
  bool onSegment(point p) {
    if (ab == point()) return a == p;
    point pa = a-p, pb = b()-p;
    return eq(pa % pb, 0) && le(pa * pb, 0);
  }
  LD distSegment(point p) {
    if (le((p-a) * ab, 0)) return dist(a, p);
    if (le(0, (p-b()) * ab)) return dist(b(), p);
    return distLine(p);
  }
  point closestSegment(point p) {
    if (le((p-a) * ab, 0)) return a;
    if (le(0, (p-b()) * ab)) return b();
    return projection(p);
  }
  bool areParallel(line l) {
    return eq(ab % l.ab, 0);
  }
  bool areSame(line l) {
    return areParallel(l) && onLine(l.a) && l.onLine(a);
  }
};
bool areIntersect(line l1, line l2, point & res) {
  if (l1.areParallel(l2)) return 0;
  LD ls = (l2.a  - l1.a) % l2.ab, rs = l1.ab % l2.ab;
  res = l1.a + l1.ab * ls/rs;
  return 1;
}
```

23. Geometry: Circle and Triangle

```
// (Circle & Triangle)
vector<point> interCircle(point o1, LD r1, point o2, LD r2) {
  LD d2 = (o1 - o2).norm_sq();
  LD d = sqrt(d2);
  if (d < fabs(r1-r2)) return {};
  if (d > r1+r2) return {};
  point u = (o1+o2) * 0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2));
  LD A = sqrt((r1+r2+d) * (r1-r2+d) * (r1+r2-d) * (-r1+r2+d));
  point v = point(o1.y-o2.y, -o1.x+o2.x) * (A / (2*d2));
  return {u+v, u-v};
}
// Heron's formula
LD heron(LD a, LD b, LD c) {
  LD s = (a + b + c) * 0.5;
  return sqrt(s * (s - a)) * sqrt((s-b) * (s-c));
}
// area by cross
```

```
LD areaTriangle(point a, point b, point c) {
  return fabs((a-b) % (c-b)) * 0.5;
}
LD rInCircle(double ab, double bc, double ca) {
  return heron(ab, bc, ca) / (0.5 * (ab + bc + ca));
}
LD rInCircle(point a, point b, point c) {
  return rInCircle(dist(a, b), dist(b, c), dist(c, a));
}

LD rCircumCircle(double ab, double bc, double ca) {// = BC / 2
sin(<ABC)
  return ab * bc * ca / (4.0 * heron(ab, bc, ca));
}
LD rCircumCircle(point a, point b, point c) {
  return rCircumCircle(dist(a, b), dist(b, c), dist(c, a));
}

point inCenter(point &A, point &B, point &C) { // 内心
  LD a = (B-C).norm(), b = (C-A).norm(), c = (A-B).norm();
  return (A * a + B * b + C * c) / (a + b + c);
}
point circumCenter(point &a, point &b, point &c) { // 外心
  point bb = b - a, cc = c - a;
  LD db = (bb).norm_sq(), dc = (cc).norm_sq(), d= 2*(bb % cc);
  return a-point(bb.y*dc-cc.y*db, cc.x*db-bb.x*dc) / d;
}
point othroCenter(point &a, point &b, point &c) { // 垂心
  point ba = b - a, ca = c - a, bc = b - c;
  LD y = ba.y * ca.y * bc.y,
    A = ca.x * ba.y - ba.x * ca.y,
    x0= (y+ca.x*ba.y*b.x-ba.x*ca.y*c.x) / A,
    y0= -ba.x * (x0 - c.x) / ba.y + ca.y;
  return point(x0, y0);
}

point perp(const point& p) {
  return point(p.y, -p.x);
}
vector<pair<point, point> > tangent2Circle(point o1, double r1,
point o2, double r2){
  vector<pair<point, point> > ret;
  LD d_sq = (o1 - o2).norm_sq();
  if( d_sq < EPS ) return ret;
  LD d = sqrt( d_sq );
  point v = ( o2 - o1 ) / d;
  for( int sign1 = 1 ; sign1 >= -1 ; sign1 -= 2 ){
    LD c = ( r1 - sign1 * r2 ) / d;
    if( c * c > 1 ) continue;
    LD h = sqrt(max( (LD)0.0 , 1.0 - c * c ) );
    for( int sign2 = 1 ; sign2 >= -1 ; sign2 -= 2 ){
      point n;
      n.x = v.x * c - sign2 * h * v.y;
      n.y = v.y * c + sign2 * h * v.x;
      point p1 = o1 + n * r1;
      point p2 = o2 + n * ( r2 * sign1 );
      if( fabs( p1.x - p2.x ) < EPS and
          fabs( p1.y - p2.y ) < EPS )
        p2 = p1 + perp( o2 - o1 );
      ret.push_back( { p1 , p2 } );
    }
  }
  return ret;
}
```

24. Geometry:The Great-Circle Distance(SPHERES)

```
double gcDistance(double plat, double plong, double qlat, double,
qlong ,double radius) {
  plat *= PI/180; plong *= PI/180;
  qlat *= PI/180; qlong *= PI/180;
  return radius * acos(
                 cos(plat)*cos(plong)*cos(qlat)*cos(qlong) +
                 cos(plat)*sin(plong)*cos(qlat)*sin(qlong) +
                 sin(plat)*sin(qlat));
}
```

25. Geometry: Polygon

```
// (Polygon)
double area(const vector< point > & P) {
  double result = 0.0;
  for(int i = 0; i+1 < (int)P.size(); ++i) {
    result += P[i] % P[i+1]; // cross(P[i], P[i+1]);
  }
  return fabs(result)/2.0;
}

// check if point p inside (CONVEX/CONCAVE) polygon vp
// 0 on boundary, -1 inside, 1 outside
int pointVsPolygon(point p, const vector< point >& vp) {
  int wn = 0, n = (int)vp.size() - 1;
```

```
  for(int i = 0; i < n; i++) {
    int cs = ccw(vp[i+1], vp[i], p);
    if(cs == 0 && (vp[i+1]-p) * (vp[i]-p) <= 0)
      return 0; // between(vp[i], p, vp[i+1])
    if(vp[i].y <= p.y) {
      if(vp[i+1].y > p.y && cs > 0)
        wn++;
    }
    else {
      if(vp[i+1].y <= p.y && cs < 0)
        wn--;
    }
  }
  return wn == 0 ? 1 : -1;
}

// line segment p-q intersect with line A-B
point lineIntersectSeg(point p, point q, point A, point B) {
  double a = B.y - A.y;
  double b = A.x - B.x;
  double c = B.x * A.y - A.x * B.y;
  double u = fabs(a * p.x + b * p.y + c);
  double v = fabs(a * q.x + b * q.y + c);
  return point((p.x*v + q.x*u)/(u+v), (p.y*v + q.y*u)/(u+v));
}
// cuts polygon Q along the line formed by point a-> point b
// (note: the last point must be the same as the first point)
vector<point> cutPolygon(point a, point b, vector<point> Q) {
  vector<point> P;
  for(int i = 0; i<(int)Q.size(); i++) {
    double left1 = (b - a) % (Q[i] - a), left2 = 0.0;
    if(i != (int)Q.size()-1) left2 = (b - a) % (Q[i+1] - a);
    if(left1 > -EPS) P.push_back(Q[i]);
    if(left1 * left2 < -EPS)
        P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
    }

    if(P.empty()) return P;
    if(!(P.front() == P.back()))
        P.push_back(P.front());y
    return P;
}
```

26. Geometry: Convex hull

```
// Graham's Scan Algorithm
// need implement operator<,-,cross,ccw on Point's library
double dist2(point a, point b) {// norm_sq(b - a)
  return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}
point pivot;
bool angle_cmp(point a, point b) {
  if(ccw(pivot, a, b) == 0)
    return dist2(a, pivot) < dist2(b, pivot);
  return ccw(pivot, a, b) > 0;
}

// hasil convexHull tidak siklik(P[0] != P.back())
void convexHull(vector<point> & P) {
  int i, j, n = (int) P.size();
  if(n < 3) {
    return;
  }
  int PO = 0;
  for(i = 1; i < n; i++) {
    if(P[i] < P[PO]) {
      PO = i;
    }
  }
  swap(P[0], P[PO]);
  pivot = P[0];
  sort(++P.begin(), P.end(), angle_cmp);
  // if point on boundary is included then uncomment this:
  // int k = (int)P.size()-1; while (k-1 > 0 && ccw(P[0], P[k-1],
P.back()) == 0) k--;
  // reverse(P.begin() + k, P.end());
  vector<point> S;
  S.push_back(P[0]);
  S.push_back(P[1]);
  i = 2;
  while(i < n) {
    j = (int) S.size() - 1;
    // if point on boundary is included then ccw >= 0
    if(j < 1 || ccw(S[j-1], S[j], P[i]) > 0) S.push_back(P[i++]);
    else S.pop_back();
  }
  P = S;
}
```

27. Convex Hull Trick Dynamic

```
const ll is_query = -(1LL<<62);
struct Line {
```

```
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m; // min: reverse it
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x; // min: reverse it
    }
};
struct HullDynamic : public multiset<Line> { // will maintain upper
hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b; // min: reverse it
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b; // min:
reverse it
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m -
x->m); // beware overflow!
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

28. Knuth-Morris-pratt (Precompute & Checking)

Precompute :

```
Arrays.fill(a, 0);
for(int i = 1; i < n; i++) {
  int j = a[i - 1];
  while(j > 0 && s[i] != s[j]) j = a[j - 1];
  if(s[i] == s[j]) a[i] = j + 1;
}
```

Checking :

```
int[] b = computeKMP(pattern);
int j = 0;
for(int i = 0; i < text.length();) {
  if(pattern.charAt(j) == text.charAt(i)) {
    i++; j++;
  } else if(j > 0) {
    j = b[j - 1];
  } else {
    i++;
  }
  if(j == pattern.length()) {
    return i - pattern.length();
  }
}
return NOT_FOUND;
```

29. Manacher Algorithm (Palindrom)

```
Sumber : http://e-maxx.ru/algo/palindromes_count
vector<int> d1 (n);
int l=0, r=-1;
for (int i=0; i<n; ++i) {
  int k = (i>r ? 0 : min (d1[l+r-i], r-i)) + 1;
  while (i+k < n && i-k >= 0 && s[i+k] == s[i-k])  ++k;
  d1[i] = k--;
  if (i+k > r)
    l = i-k,  r = i+k;
}
vector<int> d2 (n);
l=0, r=-1;
for (int i=0; i<n; ++i) {
  int k = (i>r ? 0 : min (d2[l+r-i+1], r-i+1)) + 1;
  while (i+k-1 < n && i-k >= 0 && s[i+k-1] == s[i-k])  ++k;
  d2[i] = --k;
  if (i+k-1 > r)
    l = i-k,  r = i+k-1;
}

Sumber : http://codeforces.com/blog/entry/12143
vector< vector<int> > p(2, vector<int>(n,0)); //p[1][i] even,
p[0][i] odd palindrom center i
for (int z=0, l=0, r=0; z < 2; z++, l=0, r=0) {
  for (int i = 0; i < n; i++) {
    if (i < r) p[z][i] = min(r-i+!z, p[z][l+r-i+!z]);
    int L = i-p[z][i], R = i+p[z][i]-!z;
    while (L-1 >= 0 && R+1 < n && s[L-1] == s[R+1]) p[z][i]++, L--,
```

```
R++;
    if(R > r) l = L,r = R;
  }
}
```

30. Z Algorithm

```
// z[k] = p tells us that s[0...p-1] equals s[k...k+p-1]
string s;
cin >> s;
int L = 0, R = 0;
int n = s.size();
for (int i = 1; i < n; ++i) {
  if (i > R) {
    L = R = i;
    while (R < n && s[R] == s[R-L]) ++R;
    Z[i] = R-L; --R;
  }
  else {
    int k = i-L;
    if (Z[k] < R-i+1) Z[i] = Z[k];
    else {
      L = i;
      while (R < n && s[R] == s[R-L]) ++R;
      Z[i] = R-L; --R;
    }
  }
}
```

31. Smallest String Rotation O(n)

```
string mcp(string s){
  int n = s.length();
  s += s;
  int i=0, j=1;
  while (i<n && j<n){
    int k = 0;
    while (k < n && s[i+k] == s[j+k]) k++;
    if (s[i+k] <= s[j+k]) j += k+1;
    else i += k+1;
    if (i == j) j++;
  }
  int ans = i < n ? i : j;
  return s.substr(ans, n);
}
```

32. Suffix Array + LCP

```
// suffix array
const int N = 1e5 + 5;
string s;
int sa[N], pos[N], lcp[N], tmp[N], gap, n;

bool cmp_sa(int a, int b) {
  if(pos[a] - pos[b])
    return pos[a] < pos[b];
  a += gap; b += gap;
  return (a < n && b < n) ? pos[a] < pos[b] : a > b;
}

void build_sa() {
  n = s.size();
  for(int i = 0; i<n; i++)
    sa[i] = i, pos[i] = s[i];
  for(gap = 1;; gap <<= 1) {
    sort(sa, sa + n, cmp_sa);
    for(int i = 1; i<n; i++) tmp[i] = tmp[i-1] + cmp_sa(sa[i-1], 
sa[i]);
    for(int i = 0; i<n; i++) pos[sa[i]] = tmp[i];
    if(tmp[n-1] == n-1) break;
  }
}

void build_lcp() {
  for(int i = 0, k = 0; i<n; i++) if(pos[i] - n + 1) {
    for(int j = sa[pos[i] + 1]; s[j + k] == s[i + k]; k++);
    lcp[pos[i]] = k;
    if(k) k--;
  }
}
```

33. Aho-Corasick

```
/** Aho-Corasick Dictionary Matching **/
const int NALPHABET = 26;

struct Node {
  Node** children, go;
  bool leaf;
  char charToParent;
  Node* parent, suffLink, dictSuffLink;
  int count, value;
```

```
  Node(){
    children = new Node*[NALPHABET];
    go = new Node*[NALPHABET];
    for(int i = 0; i < NALPHABET;++i){
      children[i] = go[i] = NULL;
    }
    parent = suffLink = dictSuffLink = NULL;
    leaf = false;
    count = 0;
  }
};

Node* createRoot() {
  Node* node = new Node();
  node->suffLink = node;
  return node;
}

void addString(Node* node, const string& s, int value =-1) {
  for(int i = 0; i < s.length(); ++i){
    int c = s[i] - 'a';
    if(node->children[c] == NULL){
      Node* n = new Node();
      n->parent = node;
      n->charToParent = s[i];
      node->children[c] = n;
    }
    node = node->children[c];
  }
  node->leaf = true;
  node->count++;
  node->value = value;
}

Node* suffLink(Node* node);
Node* dictSuffLink(Node* node);
Node* go(Node* node, char ch);
int calc(Node* node);

Node* suffLink(Node* node) {
  if (node->suffLink == NULL){
    if (node->parent->parent == NULL){
      node->suffLink = node->parent;
    } else {
      node->suffLink =
go(suffLink(node->parent),node->charToParent);
    }
  }
  return node->suffLink;
}

Node* dictSuffLink(Node* node) {
  if(node->dictSuffLink == NULL){
    Node* n = suffLink(node);
    if (node == n){
      node->dictSuffLink = node;
    } else {
      while (!n->leaf && n->parent != NULL){
        n = dictSuffLink(n);
      }
      node->dictSuffLink = n;
    }
  }
  return node->dictSuffLink;
}

Node* go(Node* node, char ch) {
  int c = ch -'a';
  if (node->go[c] == NULL){
    if (node->children[c] != NULL) {
      node->go[c]= node->children[c];
    } else {
      node->go[c]= node->parent == NULL? node : go(suffLink(node),
ch);
    }
  }
  return node->go[c];
}

int calc(Node* node) {
  if (node->parent == NULL) {
    return 0;
  } else {
    return node->count + calc(dictSuffLink(node));
  }
}

int main() {
  Node* root = createRoot();
  addString(root,"a",0);
  addString(root,"aa",1);
  addString(root,"abc",2);

  string s("abcaadc");
  Node* node = root;
  for (int i = 0; i < s.length(); ++i){
```

```
    node = go(node, s[i]);
    Node* temp = node;
    while (temp != root) {
      if (temp->leaf) {
        printf("string (%d) occurs at position %d\n", temp->value,
i);
      }
      temp = dictSuffLink(temp);
    }
  }
  return 0;
}
```

34. Palindromic Tree

```
/*
 * sfail: compressed fail links with same diff
 * O(lgn): length of sfail link path
 */
const int MAXN = 1e6+10;
struct PalT{
  int tot,lst;
  int nxt[MAXN][26], len[MAXN];
  int fail[MAXN], diff[MAXN], sfail[MAXN];
  char* s;
  int newNode(int l, int _fail) {
    int res = ++tot;
    fill(nxt[res], nxt[res]+26, 0);
    len[res] = l, fail[res] = _fail;
    diff[res] = l - len[_fail];
    if (diff[res] == diff[_fail])
      sfail[res] = sfail[_fail];
    else
      sfail[res] = _fail;
    return res;
  }
  void push(int p) {
    int np = lst;
    int c = s[p]-'a';
    while (p-len[np]-1 < 0 || s[p] != s[p-len[np]-1])
      np = fail[np];
    if ((lst=nxt[np][c])) return;
    int nq_f = 0;
    if (len[np]+2 == 1) nq_f = 2;
    else {
      int tf = fail[np];
      while (p-len[tf]-1 < 0 || s[p] != s[p-len[tf]-1])
        tf = fail[tf];
      nq_f = nxt[tf][c];
    }
    int nq = newNode(len[np]+2, nq_f);
    nxt[np][c] = nq;
    lst=nq;
  }
  void init(char* _s){
    s = _s;
    tot = 0;
    newNode(-1, 1);
    newNode(0, 1);
    diff[2] = 0;
    lst = 2;
  }
} palt;
```

35. KD-Tree

```
const int MXN = 100005;

struct KDTree {
  struct Node {
    int x,y,x1,y1,x2,y2;
    int id,f;
    Node *L, *R;
  }tree[MXN];
  int n;
  Node *root;

  long long dis2(int x1, int y1, int x2, int y2) {
    long long dx = x1-x2;
    long long dy = y1-y2;
    return dx*dx+dy*dy;
  }
  static bool cmpx(Node& a, Node& b){ return a.x<b.x; }
  static bool cmpy(Node& a, Node& b){ return a.y<b.y; }
  void init(vector<pair<int,int>> ip) {
    n = ip.size();
    for (int i=0; i<n; i++) {
      tree[i].id = i;
      tree[i].x = ip[i].first;
      tree[i].y = ip[i].second;
    }
    root = build_tree(0, n-1, 0);
  }
  Node* build_tree(int L, int R, int dep) {
    if (L>R) return nullptr;
    int M = (L+R)/2;
```

```
  tree[M].f = dep%2;
  nth_element(tree+L, tree+M, tree+R+1, tree[M].f ? cmpy : cmpx);
  tree[M].x1 = tree[M].x2 = tree[M].x;
  tree[M].y1 = tree[M].y2 = tree[M].y;

  tree[M].L = build_tree(L, M-1, dep+1);
  if (tree[M].L) {
    tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
    tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
    tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
    tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
  }

  tree[M].R = build_tree(M+1, R, dep+1);
  if (tree[M].R) {
    tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
    tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
    tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
    tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
  }

  return tree+M;
}
int touch(Node* r, int x, int y, long long d2){
  long long dis = sqrt(d2)+1;
  if (x<r->x1-dis || x>r->x2+dis || y<r->y1-dis || y>r->y2+dis)
    return 0;
  return 1;
}
void nearest(Node* r, int x, int y, int &mID, long long &md2){
  if (!r || !touch(r, x, y, md2)) return;
  long long d2 = dis2(r->x, r->y, x, y);
  if (d2 < md2 || (d2 == md2 && mID < r->id)) {
    mID = r->id;
    md2 = d2;
  }
  // search order depends on split dim
  if ((r->f == 0 && x < r->x) ||
      (r->f == 1 && y < r->y)) {
    nearest(r->L, x, y, mID, md2);
    nearest(r->R, x, y, mID, md2);
  } else {
    nearest(r->R, x, y, mID, md2);
    nearest(r->L, x, y, mID, md2);
  }
}
int query(int x, int y) {
  int id = 1029384756;
```

```
    long long d2 = 102938475612345678LL;
    nearest(root, x, y, id, d2);
    return id;
  }
}tree;
```

36. Link-Cut Tree / Splay Tree

```
const int MXN = 100005;
const int MEM = 100005;
struct Splay {
  static Splay nil, mem[MEM], *pmem;
  Splay *ch[2], *f;
  int val, rev, size;
  Splay () : val(-1), rev(0), size(0)
  { f = ch[0] = ch[1] = &nil; }
  Splay (int _val) : val(_val), rev(0), size(1)
  { f = ch[0] = ch[1] = &nil; }
  bool isr()
  { return f->ch[0] != this && f->ch[1] != this; }
  int dir()
  { return f->ch[0] == this ? 0 : 1; }
  void setCh(Splay *c, int d){
    ch[d] = c;
    if (c != &nil) c->f = this;
    pull();
  }
  void push(){
    if( !rev ) return;
    swap(ch[0], ch[1]);
    if (ch[0] != &nil) ch[0]->rev ^= 1;
    if (ch[1] != &nil) ch[1]->rev ^= 1;
    rev=0;
  }
  void pull(){
    size = ch[0]->size + ch[1]->size + 1;
    if (ch[0] != &nil) ch[0]->f = this;
    if (ch[1] != &nil) ch[1]->f = this;
  }
} Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::mem;
Splay *nil = &Splay::nil;
void rotate(Splay *x){
  Splay *p = x->f;
  int d = x->dir();
  if (!p->isr()) p->f->setCh(x, p->dir());
  else x->f = p->f;
      p->setCh(x->ch[!d], d);
  x->setCh(p, !d);
```

```
      p->pull(); x->pull();
}
vector<Splay*> splayVec;
void splay(Splay *x){
  splayVec.clear();
  for (Splay *q=x;; q=q->f){
    splayVec.push_back(q);
    if (q->isr()) break;
  }
  reverse(begin(splayVec), end(splayVec));
  for (auto it : splayVec) it->push();
  while (!x->isr()) {
    if (x->f->isr()) rotate(x);
    else if (x->dir()==x->f->dir())
      rotate(x->f),rotate(x);
    else rotate(x),rotate(x);
  }
}
Splay* access(Splay *x){
  Splay *q = nil;
  for (;x!=nil;x=x->f){
    splay(x);
    x->setCh(q, 1);
    q = x;
  }
  return q;
}
void evert(Splay *x){
  access(x); splay(x); x->rev ^= 1; x->push(); x->pull();
}
void link(Splay *x, Splay *y){
//  evert(x);
  access(x); splay(x); evert(y); x->setCh(y, 1);
}
void cut(Splay *x, Splay *y){
//  evert(x);
  access(y); splay(y); y->push(); y->ch[0] = y->ch[0]->f = nil;
}
int N, Q;
Splay *vt[MXN];
int ask(Splay *x, Splay *y){
  access(x); access(y); splay(x);
  int res = x->f->val; if (res == -1) res=x->val;
  return res;
}
int main(int argc, char** argv){
  scanf("%d%d", &N, &Q);
  for (int i=1; i<=N; i++)
    vt[i] = new (Splay::pmem++) Splay(i);
  while (Q--) {
    char cmd[105];
    int u, v;
    scanf("%s", cmd);
    if (cmd[1] == 'i') {
      scanf("%d%d", &u, &v);
      link(vt[v], vt[u]);
    } else if (cmd[0] == 'c') {
      scanf("%d", &v);
      cut(vt[1], vt[v]);
    } else {
      scanf("%d%d", &u, &v);
      int res=ask(vt[u], vt[v]);
      printf("%d\n", res);
    }
  }
}
```

37. Implicit Treap

```
/**
 * Treap uses implicit key
 * This Implementation : maintain array, can insert and delete in any
position, can reverse interval
 */
#include <bits/stdc++.h>
using namespace std;
typedef struct item * pitem;
struct item
{
  int cnt, value, prior;
  bool rev;
  pitem l, r;
  item(int prior, int value) : cnt(1), rev(false), prior(prior),
value(value), l(NULL), r(NULL) {}
};
int cnt(pitem t) {
  return t ? t->cnt : 0;
}
void upd_cnt(pitem it) {
  if (it)
    it->cnt = cnt(it->l) + cnt(it->r) + 1;
}
void push(pitem it) {
  if (it && it->rev) {
    it->rev = false;
    swap(it->l, it->r);
```

```
    if (it->l) it->l->rev ^= true;
    if (it->r) it->r->rev ^= true;
  }
}
void merge(pitem & t, pitem l, pitem r) {
  push(l);
  push(r);
  if (!l || !r)
    t = l ? l : r;
  else if (l->prior > r->prior)
    merge(l->r, l->r, r), t = l;
  else
    merge(r->l, l, r->l), t = r;
  upd_cnt(t);
}
void split(pitem t, pitem & l, pitem & r, int key, int add = 0) {
  if (!t)
    return void (l = r = 0);
  int cur_key = cnt(t->l) + add;
  if (key <= cur_key)
    split(t->l, l, t->l, key, add), r = t;
  else
    split(t->r, t->r, r, key, add + cnt(t->l) + 1), l = t;
  upd_cnt(t);
}
void reverse(pitem t, int l, int r) {
  pitem t1, t2, t3;
  split(t, t1, t2, l);
  split(t2, t2, t3, r-l+1);
  t2->rev ^= true;
  merge(t, t1, t2);
  merge(t, t, t3);
}
int main() {
  int n;
  scanf("%d", &n);
  srand(time(NULL));
  pitem root = NULL;
  for (int i = 0; i < n; i++) {
    int a;
    scanf("%d", &a);
    pitem cur =  new item(rand(), a);
    if (root)
      merge(root, root, cur);
    else
      root = cur;
  }
  int m;
```

```
  scanf("%d", &m);
  for (int i = 0; i < m; i++) {
    int l, r;
    scanf("%d %d", &l, &r);
    reverse(root, l, r);
    output(root);
  }
}
```

38. Policy-based Data Structure

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <bits/stdc++.h>
using namespace std;
using namespace __gnu_pbds;
typedef
tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_
update> ordered_set;
ordered_set X;

int main() {
  cout<<*X.find_by_order(1)<<endl;           // array index ke-1
  cout<<(end(X)==X.find_by_order(6))<<endl; // end(X) = pointer
  cout<<X.order_of_key(400)<<endl;           // idx lower_bound 400
}
```