Team Notebook

ICPCNextCentury - Damascus University

September 27, 2021

Contents

1 2-SAT

```
struct TwoSAT {
 int n;
 vector < vector<int> > g, gt;
 vector<bool> used;
 vector<int> order, comp;
 TwoSAT(int n = 0) :n(2 * n), g(2 * n), gt(2 * n), used(2 *
     n), comp(2 * n, -1) {}

 void dfs1(int u) {
  used[u] = true;
  for (int v : g[u])
   if (!used[v])
    dfs1(v);
  order.push_back(u);
 }
 void dfs2(int u, int c) {
  comp[u] = c;
  for (int v : gt[u])
   if (comp[v] == -1)
    dfs2(v, c);
 }
 bool solve(vector < int >& sol) {
  sol.clear();
  for (int i = 0; i < n; i++)
   if (!used[i])
    dfs1(i);
  for (int i = 0, j = 0; i < n; ++i) {
   int v = order[n - i - 1];
   if (comp[v] == -1) dfs2(v, j++);
  }
  for (int i = 0; i < n; i++)
   if (comp[i] == comp[i ^ 1])
    return 0;
  for (int i = 0; i < n; i += 2) {
   int ans = comp[i] > comp[i ^ 1] ? i : i ^ 1;
   sol.push_back(ans % 2);
  }
  return 1;
 }
 void add_edge(int a, int b) {
  g[a].push_back(b);
  gt[b].push_back(a);
 }
 void conv(int &a) {
  if (a < 0)
   a = (~a) * 2;
  else a = 2 * a + 1;
 }
 void setValue(int a) {
  conv(a); add_edge(a ^ 1, a);
 }
 void imply(int a, int b) {
  conv(a); conv(b);
  add_edge(a, b);
  add_edge(b ^ 1, a ^ 1);
 }
 void OR(int a, int b) {
  imply(~a, b);
  imply(~b, a);
 }
};
```

2 Aho Corasick

```
#define MAX_LETTERS 27
int ID = 1;
int getIdx(char x) {
 if (x >= 'a' && x <= 'z') return x - 'a';
 else return 26; //invalid char
}
struct Trie {
 Trie* Next[MAX_LETTERS];
 Trie* Fail;
 Trie* dp[MAX_LETTERS];
 vector<int> Chars;
 vector<Trie*> G;
 vector<int> patterns;
 int matchCnt;
 int id;
 Trie() {
  memset(Next, 0, sizeof(Next));
  memset(dp, 0, sizeof(dp));
  Fail = 0, matchCnt = 0;
  id = ID++;
 }

 void insert(string& str, int i, int patIndex) {
  if (i == str.size()) {
   matchCnt++;
   patterns.push_back(patIndex);
   return;
  }
  int idx = getIdx(str[i]);
  if (Next[idx] == 0) Chars.push_back(idx), Next[idx] = new
      Trie();
  Next[idx]->insert(str, i + 1, patIndex);
 }

};

Trie* getFail(Trie* cur, int ch) {
 if (cur->dp[ch] != NULL)
  return cur->dp[ch];
 if (cur->Next[ch] != NULL)
  return cur->dp[ch] = cur->Next[ch];
 return cur->dp[ch] = getFail(cur->Fail, ch);
}

struct AhoCorasick {
 Trie* Root;
 AhoCorasick(vector<string>& patterns) {
  Root = new Trie();
  for (int i = 0; i < (int)patterns.size(); i++) {
   Root->insert(patterns[i], 0, i);
  }
  BFS();
 }
 void BFS() {
  queue<Trie*> q;
  Root->Fail = Root;
  for (int i = 0; i < MAX_LETTERS; i++)
   if (Root->Next[i] == 0) Root->Next[i] = Root;
   else {
    Trie* temp = Root->Next[i];
    temp->Fail = Root;
    Root->G.push_back(temp);
    q.push(temp);
   }

  while (q.empty() == false) {
   Trie* cur = q.front(); q.pop();

   for (int ch : cur->Chars) {
    Trie* Fail = getFail(cur->Fail, ch);
    cur->Next[ch]->Fail = Fail;
    Fail->G.push_back(cur->Next[ch]);
    //cur->Next[ch]->patterns.insert(
    //cur->Next[ch]->patterns.end(), Fail->patterns.begin(),
        Fail->patterns.end());
    q.push(cur->Next[ch]);
   }
  }
 }
 int match(string& x) {
  int ret = 0;
  Trie* now = Root;
```

```
  for (int i = 0; i < (int)x.size(); i++) {
   int ch = getIdx(x[i]);
   now = getFail(now, ch);
   //ret += now->patterns.size();
   ret += now->matchCnt;
  }
  return ret;
 }
};
```

3 Bridge Tree

```
int tim = 0, grp = 1;
int tin[N], minAnc[N], comp[N];
bool vis[N], vis2[N], isBridge[M];
queue<int> Q[N];

// [node, edgeIndex]
vector<pair<int, int>> G[N];

// Tree stores Bridge Tree
// vertices stores the nodes in each component
vector<int> tree[N], vertices[N];

void DFS(int u, int par) {
 vis[u] = 1;
 tin[u] = ++tim;
 minAnc[u] = tin[u];
 for (auto [v, idx] : G[u]) {
  if (v == par) continue;
  if (vis[v]) {
   minAnc[u] = min(minAnc[u], tin[v]);
   continue;
  }
  DFS(v, u);
  minAnc[u] = min(minAnc[u], minAnc[v]);
  if (minAnc[v] > tin[u])
   isBridge[idx] = 1;
 }
}

void DFS2(int cur) {
 int comp = grp;
 Q[comp].push(cur);
 vis2[cur] = 1;
 while (!Q[comp].empty()) {
  int u = Q[comp].front();
  Q[comp].pop();
  vertices[comp].push_back(u);
  for (auto [v, edgeidx] : G[u]) {
   if (vis2[v]) continue;
   if (isBridge[edgeidx]) {
    grp++;
    tree[comp].push_back(grp);
    tree[grp].push_back(comp);
    DFS2(v);
   }
   else {
    Q[comp].push(v);
    vis2[v] = 1;
   }
  }
 }
}

int main() {
 DFS(1, 0);
 DFS2(1);
}
```

4 Bridges Art Point

```
int Time;
vector <int> G[N];
vector <int> tim, ret, artPoint;
vector <pair<int, int>> bridges;

void DFS(int u, int p = 0) {
 int children = 0;
 tim[u] = ret[u] = Time++;
 for (int v : G[u]) if (v != p) {
  if (tim[v] + 1)
   ret[u] = min(ret[u], tim[v]);
  else {
   DFS(v, u);
   children++;
   ret[u] = min(ret[u], ret[v]);

   if (tim[u] < ret[v])
    bridges.emplace_back(u, v);
   if ((!p && children >= 2) || (p && tim[u] <= ret[v]))
    artPoint.push_back(u);
  }
 }
}

int main() {
 tim = ret = vector<int>(n + 1, -1);
}
```

5 CHT (neal)

```
const long long LL_INF = (long long)2e18 + 5;
struct point {
    long long x, y;
    point() : x(0), y(0) {
    }
    point(long long _x, long long _y) : x(_x), y(_y) {
    }
};
// dp_hull enables you to do the following two operations in
      amortized O(log n) time:
// 1. Insert a pair (a_i, b_i) into the structure
// 2. For any value of x, query the maximum value of a_i * x
      + b_i
// All values a_i, b_i, and x can be positive or negative.
struct dp_hull {
    struct segment {
        point p;
        mutable point next_p;
        segment(point _p = {0, 0}, point _next_p = {0, 0}) :
             p(_p), next_p(_next_p) {
        }
        bool operator<(const segment &other) const {
            // Sentinel value indicating we should binary
                 search the set for a single x-value.
            if (p.y == LL_INF)
                return p.x * (other.next_p.x - other.p.x) <=
                     other.p.y - other.next_p.y;
            return make_pair(p.x, p.y) < make_pair(other.p.x,
                 other.p.y);
        }
    };
    set&lt;segment&gt; segments;
    int size() const {
        return segments.size();
    }
    set&lt;segment&gt;::iterator prev(set&lt;segment&gt;::iterator it)
         const {
        return it == segments.begin() ? it : --it;
    }
    set&lt;segment&gt;::iterator next(set&lt;segment&gt;::iterator it)
         const {
        return it == segments.end() ? it : ++it;
```

```
}
static long long floor_div(long long a, long long b) {
    return a / b - ((a ^ b) < 0 && a % b != 0);
}
static bool bad_middle(const point &a, const point &b,
     const point &c) {
    // This checks whether the x-value where b beats a
         comes after the x-value where c beats b.
    // It's fine to round down here if we will only query
          integer x-values. (Note: plain C++
    // division rounds toward zero)
    return floor_div(a.y - b.y, b.x - a.x) >= floor_div(b
         .y - c.y, c.x - b.x);
}
bool bad(set&lt;segment&gt;::iterator it) const {
    return it != segments.begin() && next(it) != segments
         .end() &&
           bad_middle(prev(it)->p, it->p, next(it)->p);
}
void insert(const point &p) {
    set&lt;segment&gt;::iterator next_it = segments.lower_bound
         (segment(p));
    if (next_it != segments.end() && p.x == next_it->p.x)
        return;
    if (next_it != segments.begin()) {
        set&lt;segment&gt;::iterator prev_it = prev(next_it);

        if (p.x == prev_it->p.x)
            segments.erase(prev_it);
        else if (next_it != segments.end() && bad_middle(
             prev_it->p, p, next_it->p))
            return;
    }
    // Note we need the segment(p, p) here for the single
          x-value binary search.
    set&lt;segment&gt;::iterator it = segments.insert(next_it,
         segment(p, p));
    while (bad(prev(it)))
        segments.erase(prev(it));
    while (bad(next(it)))
        segments.erase(next(it));
    if (it != segments.begin())
        prev(it)->next_p = it->p;
    if (next(it) != segments.end())
        it->next_p = next(it)->p;
}
void insert(long long a, long long b) {
    insert(point(a, b));
}
// Queries the maximum value of ax + b.
long long query(long long x) const {
    assert(size() > 0);
    set&lt;segment&gt;::iterator it = segments.upper_bound(
         segment(point(x, LL_INF)));
    return it->p.x * x + it->p.y;
}
};
```

6 CHT Li-Chao

```
template <typename TT = long long>
struct LiChaoTree
{
 const static TT INF = 1e18;
 const static TT MIN_Q = 1;
 const static TT MAX_Q = 1e6 + 1;

 struct SegNode {
  complex <TT> line;
  SegNode *left, *right;

  SegNode(complex <TT> _line = {0, INF}, SegNode* _left =
       NULL, SegNode* _right = NULL):
   line(_line), left(_left), right(_right){}
 } *root;

 LiChaoTree() {
  root = NULL;
 }

 TT evaluate(complex <TT> line, TT x) {
  return line.real() * x + line.imag();
 }

 void insert(complex <TT> line, SegNode* &node, TT L, TT R)
      {
  if (node == NULL) {
   node = new SegNode(line);
   return;
  }

  TT mid = (L + R) / 2;
  bool LP = evaluate(line, L) < evaluate(node->line, L);
  bool MP = evaluate(line, mid) < evaluate(node->line, mid);

  if (MP)
   swap(line, node->line);
  if (R - L == 1)
   return;
  else if (LP != MP)
   insert(line, node->left, L, mid);
  else
   insert(line, node->right, mid, R);
 }
 void insert(TT a, TT b) {
  insert(complex<TT>(a, b), root, MIN_Q, MAX_Q);
 }

 TT query(TT x, SegNode* &node, TT L, TT R) {
  if (node == NULL)
   return INF;

  TT mid = (L + R) / 2;
  TT res = evaluate(node->line, x);

  if (R - L == 1)
   return res;
  else if (x < mid)
   return min(res, query(x, node->left, L, mid));
  else
   return min(res, query(x, node->right, mid, R));
 }
 TT query(TT x) {
  return query(x, root, MIN_Q, MAX_Q);
 }
};
```

7 CHT Linear

```
// this for min
// for max, mul all lines with -1
// and mul the result of query
struct LinearCHT {
#define m first
#define c second
#define queryDir 1 // 1 if queries are increasing, -1 if
     decreasing
    typedef long long ftype;
    typedef pair<ftype, ftype> line;
    ftype f(line l, ftype x) {
        return l.m * x + l.c;
    }
    bool bad(line l1, line l2, line l3) {
        return (l3.c - l1.c) * (l1.m - l2.m) <= (l2.c - l1.c)
             * (l1.m - l3.m);
```

```
    }
    deque<line> q;
    int curQ;
    void init() {
        q.clear();
        curQ = 0;
    }
    void add_line(ftype a, ftype b) {
        line l = line(a, b);
        if (q.size() == 0) {
            q.push_back(l);
            if (queryDir == -1)
                curQ++;
            return;
        }
        bool left = (l.m >= q.at(0).m);
        while (q.size() >= 2) {
            if (left) {
                line l1 = q.at(0), l2 = q.at(1);
                if (!bad(l, l1, l2))
                    break;
                q.pop_front();
                curQ -= queryDir;
            } else {
                line l1 = q.at(q.size() - 2), l2 = q.at(q.size
                    () - 1);
                if (!bad(l1, l2, l))
                    break;
                q.pop_back();
                curQ -= queryDir;
            }
        }
        if (left)
            q.push_front(l), curQ -= queryDir;
        else
            q.push_back(l), curQ -= queryDir;
    }
    ftype query(ftype x) {
        if (curQ < 0)
            curQ = 0;
        if (curQ >= q.size())
            curQ = (int)q.size() - 1;
        while (curQ + queryDir < q.size() && curQ + queryDir
             >= 0 &&
              f(q.at(curQ + queryDir), x) < f(q.at(curQ), x))
            curQ += queryDir;
        return f(q.at(curQ), x);
    }
} LCHT;
```

8 DSU on Trees

```
 const  int N = 1e5 + 100;

int sz[N], ans[N], big[N];

vector<int> Tree[N];

void preCalc(int u = 1, int p = 0) {
 sz[u] = 1;

 int maxChild = 0;
 for (auto v : Tree[u]) if (v != p) {
  preCalc(v, u);

  sz[u] += sz[v];
  if (sz[v] > maxChild || big[u] == -1) {
   maxChild = sz[v];
   big[u] = v;
  }
 }
}

void Add(int u, int p, int type) {
 if (type == 1) {

 }
 else {

 }

 for (auto v : Tree[u])
  if (v != p) Add(v, u, type);
}

void DFS(int u = 1, int p = 0, bool keep = 0) {
 for (auto v : Tree[u])
  if (v != p && v != big[u])
   DFS(v, u, 0);

 if (big[u] + 1)
  DFS(big[u], u, 1);

 // Add Node
 for (auto v : Tree[u])
  if (v != p && v != big[u])
   Add(v, u, 1);

 // answer Query
 if (keep == 0)
  Add(u, p, -1);
}

int main() {
 memset(big, -1, sizeof big);
 preCalc();
 DFS();
}
```

9 DiscreteLog

```
/// return 'x' that satisfies ( a ^ x = b % m ) or return -1
/// 'a' and 'm' must be coprime ( GCD(a, m) == 1 )
/// Algorithm : baby step giant step
/// Complexity : O( sqrt(m) * log(sqrt(m)) )
int SolveDiscreteLogarithm(int a, int b, const int& m) {
    assert(0 <= a && a < m && 0 <= b && b < m);
    assert(GCD(a, m) == 1);

    if (!a || !b)
        return -1;
    if (b == 1)
        return 0;

    int n = 1;
    while (n * n < m)
        n++;

    map<int, int> Max;
    for (int q = 0, x = b; q < n; q++, x = mul(x, a, m))
        Max[x] = q;

    int Ans = m + 1;

    int an = 1; /// a ^ n
    for (int i = 0; i < n; i++)
        an = mul(an, a, m);

    for (int p = 1, x = an; p * n - (n - 1) <= m; p++, x =
         mul(x, an, m))
        if (Max.count(x))
            Ans = min(Ans, n * p - Max[x]);

    if (Ans > m)
        Ans = -1;

    return Ans;
```

```
}
```

10 Dynamic SegTree (Vector)

```
 struct  SegNode {
    ll Sum, Lazy;
    int Left, Right;
    SegNode(){
        Sum = Lazy = 0;
        Right = Left = 0;
    }
};

vector<SegNode> Seg;

void PushLazy(int Node, ll Len){
    Seg[Node].Sum += Len * Seg[Node].Lazy;

    if (Len == 1){
        Seg[Node].Lazy = 0;
        return;
    }

    if (!Seg[Node].Left){
        Seg[Node].Left = (int) Seg.size();
        Seg.push_back(SegNode());
    }
    if (!Seg[Node].Right){
        Seg[Node].Right = (int) Seg.size();
        Seg.push_back(SegNode());
    }

    Seg[Seg[Node].Left].Lazy += Seg[Node].Lazy;
    Seg[Seg[Node].Right].Lazy += Seg[Node].Lazy;
    Seg[Node].Lazy = 0;
}

void Update(int i, int j, ll Val, int Node, int L = 1, int R
     = 1e9) {
    PushLazy(Node, R - L + 1);
    if (R < i || L > j) return;
    if (L >= i && R <= j){
        Seg[Node].Lazy += Val;
        PushLazy(Node, R - L + 1);
        return;
    }

    int Mid = L + R >> 1;

    Update(i, j, Val, Seg[Node].Left, L, Mid);
    Update(i, j, Val, Seg[Node].Right, Mid + 1, R);

    Seg[Node].Sum = Seg[Seg[Node].Left].Sum + Seg[Seg[Node].
         Right].Sum;
}

ll Query(int i, int j, int Node, int L = 1, int R = 1e9) {
    PushLazy(Node, R - L + 1);

    if (R < i || L > j) return 0;
    if (L >= i && R <= j)
        return Seg[Node].Sum;

    int Mid = L + R >> 1;

    ll Q1 = Query(i, j, Seg[Node].Left, L, Mid);
    ll Q2 = Query(i, j, Seg[Node].Right, Mid + 1, R);

    return Q1 + Q2;
}

int main() {
    Seg.clear();
    Seg.push_back(SegNode());
}
```

11 Extended Euclidean

```
int gcd(int a, int b, int &x, int &y) {
 if (b == 0) {
  return x = 1, y = 0, a;
 }
 int x1, y1;
 int d = gcd(b, a % b, x1, y1);
 x = y1;
 y = x1 - y1 * (a / b);
 return d;
}
bool find_any_solution(int a, int b, int c, int &x0, int &y0
     , int &g) {
 g = gcd(abs(a), abs(b), x0, y0);
 if (c % g) {
  return false;
 }
 x0 *= c / g;
 y0 *= c / g;
 if (a < 0)
  x0 = -x0;
 if (b < 0)
  y0 = -y0;
 return true;
}
void shift_solution(int &x, int &y, int a, int b, int cnt) {
 x += cnt * b;
 y -= cnt * a;
}
int find_all_solutions(int a, int b, int c, int minx, int
     maxx, int miny, int maxy) {
 int x, y, g;
 if (!find_any_solution(a, b, c, x, y, g))
  return 0;
 a /= g;
 b /= g;
 int sign_a = a > 0 ? +1 : -1;
 int sign_b = b > 0 ? +1 : -1;
 shift_solution(x, y, a, b, (minx - x) / b);
 if (x < minx)
  shift_solution(x, y, a, b, sign_b);
 if (x > maxx)
  return 0;
 int lx1 = x;
 shift_solution(x, y, a, b, (maxx - x) / b);
 if (x > maxx)
  shift_solution(x, y, a, b, -sign_b);
 int rx1 = x;
 shift_solution(x, y, a, b, -(miny - y) / a);
 if (y < miny)
  shift_solution(x, y, a, b, -sign_a);
 if (y > maxy)
  return 0;
 int lx2 = x;
 shift_solution(x, y, a, b, -(maxy - y) / a);
 if (y > maxy)
  shift_solution(x, y, a, b, sign_a);
 int rx2 = x;
 if (lx2 > rx2)
  swap(lx2, rx2);
 int lx = max(lx1, lx2);
 int rx = min(rx1, rx2);
 if (lx > rx)
  return 0;
 return (rx - lx) / abs(b) + 1;
}
```

12 FFT - NTT

```
namespace fft {
 struct num {
  double x, y;
  num() { x = y = 0; }
  num(double x, double y) : x(x), y(y) {}
 };
 inline num operator+(num a, num b) { return num(a.x + b.x,
      a.y + b.y); }
 inline num operator-(num a, num b) { return num(a.x - b.x,
      a.y - b.y); }
 inline num operator*(num a, num b) { return num(a.x * b.x -
       a.y * b.y, a.x * b.y + a.y * b.x); }
 inline num conj(num a) { return num(a.x, -a.y); }

 int base = 1;
 vector<num> roots = {{0, 0}, {1, 0}};
 vector<int> rev = {0, 1};
 const double PI = acosl(-1.0);

 void ensure_base(int nbase) {
  if (nbase <= base)
   return;
  rev.resize(1 << nbase);
  for (int i = 0; i < (1 << nbase); i++)
   rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
  roots.resize(1 << nbase);
  while (base < nbase) {
   double angle = 2 * PI / (1 << (base + 1));
   for (int i = 1 << (base - 1); i < (1 << base); i++) {
    roots[i << 1] = roots[i];
    double angle_i = angle * (2 * i + 1 - (1 << base));
    roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
   }
   base++;
  }
 }

 void fft(vector<num> &a, int n = -1) {
  if (n == -1)
   n = a.size();
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = base - zeros;
  for (int i = 0; i < n; i++)
   if (i < (rev[i] >> shift))
    swap(a[i], a[rev[i] >> shift]);
  for (int k = 1; k < n; k <<= 1) {
  for (int i = 0; i < n; i += 2 * k) {
   for (int j = 0; j < k; j++) {
    num z = a[i + j + k] * roots[j + k];
    a[i + j + k] = a[i + j] - z;
    a[i + j] = a[i + j] + z;
   }
  }
 }
}

vector<num> fa, fb;

vector<int> multiply(vector<int> &a, vector<int> &b) {
 int need = a.size() + b.size() - 1;
 int nbase = 0;
 while ((1 << nbase) < need)
  nbase++;
 ensure_base(nbase);
 int sz = 1 << nbase;
 if (sz > (int)fa.size())
  fa.resize(sz);
 for (int i = 0; i < sz; i++) {
  int x = (i < (int)a.size() ? a[i] : 0);
  int y = (i < (int)b.size() ? b[i] : 0);
  fa[i] = num(x, y);
 }
 fft(fa, sz);
 num r(0, -0.25 / sz);
 for (int i = 0; i <= (sz >> 1); i++) {
  int j = (sz - i) & (sz - 1);
  num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
  if (i != j)
   fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
  fa[i] = z;
 }
 fft(fa, sz);
 vector<int> res(need);
 for (int i = 0; i < need; i++)
  res[i] = fa[i].x + 0.5;
 return res;
}

vector<int> multiply_mod(vector<int> &a, vector<int> &b,
     int m, int eq = 0)
{
 int need = a.size() + b.size() - 1;
 int nbase = 0;
 while ((1 << nbase) < need)
  nbase++;
 ensure_base(nbase);
 int sz = 1 << nbase;
 if (sz > (int)fa.size())
  fa.resize(sz);
 for (int i = 0; i < (int)a.size(); i++) {
  int x = (a[i] % m + m) % m;
  fa[i] = num(x & ((1 << 15) - 1), x >> 15);
 }
 fill(fa.begin() + a.size(), fa.begin() + sz, num{0, 0});
 fft(fa, sz);
 if (sz > (int)fb.size())
  fb.resize(sz);
 if (eq)
  copy(fa.begin(), fa.begin() + sz, fb.begin());
 else {
  for (int i = 0; i < (int)b.size(); i++) {
   int x = (b[i] % m + m) % m;
   fb[i] = num(x & ((1 << 15) - 1), x >> 15);
  }
  fill(fb.begin() + b.size(), fb.begin() + sz, num{0, 0});
  fft(fb, sz);
 }
 double ratio = 0.25 / sz;
 num r2(0, -1), r3(ratio, 0), r4(0, -ratio), r5(0, 1);
 for (int i = 0; i <= (sz >> 1); i++) {
  int j = (sz - i) & (sz - 1);
  num a1 = (fa[i] + conj(fa[j]));
  num a2 = (fa[i] - conj(fa[j])) * r2;
  num b1 = (fb[i] + conj(fb[j])) * r3;
  num b2 = (fb[i] - conj(fb[j])) * r4;
  if (i != j) {
   num c1 = (fa[j] + conj(fa[i]));
   num c2 = (fa[j] - conj(fa[i])) * r2;
   num d1 = (fb[j] + conj(fb[i])) * r3;
   num d2 = (fb[j] - conj(fb[i])) * r4;
   fa[i] = c1 * d1 + c2 * d2 * r5;
   fb[i] = c1 * d2 + c2 * d1;
  }
  fa[j] = a1 * b1 + a2 * b2 * r5;
  fb[j] = a1 * b2 + a2 * b1;
 }
 fft(fa, sz);
 fft(fb, sz);
 vector<int> res(need);
 for (int i = 0; i < need; i++) {
  ll aa = fa[i].x + 0.5;
  ll bb = fb[i].x + 0.5;
  ll cc = fa[i].y + 0.5;
  res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
 }
 return res;
```

```
 }
 vector<int> square_mod(vector<int> &a, int m) {
  return multiply_mod(a, a, m, 1);
 }
};
using namespace fft;

const int Mod = 1e9 + 7;

vector <int> FastPower(vector <int> v, int k) {
 if (!k) return {1};

 vector <int> res = FastPower(v, k >> 1);
 res = multiply_mod(res, res, Mod);
 if (k & 1) res = multiply_mod(res, v, Mod);

 while (!res.empty() && res.back() == 0)
  res.pop_back();

 return res;
}
```

13 FenwickTree

```
template <typename T = int>
struct FenwickTree {
    int n;
    vector <T> bit;

    FenwickTree(){}
    FenwickTree(int n):
        n(n + 1), bit(n + 5){}

    void update(int idx, T val = 1) {
        for (int i = idx + 1; i <= n; i += (i & -i))
            bit[i] += val;
    }

    T query(int idx) {
        T res = 0;
        for (int i = idx + 1; i > 0; i -= (i & -i))
            res += bit[i];
        return res;
    }
    T query(int L, int R) {
        return (L <= R ? query(R) - query(L - 1) : 0);
    }
};
```

14 Functions

```
// KMP
void Match(){
 int j = 0;
 for (int i = 1; i < n; i++){
  while(j && t[i] != t[j])
   j = KMP[j - 1];
  if (t[i] == t[j])
   KMP[i] = ++j;
 }
}

// fast unordered map O(1)
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        srand(time(NULL));
        static const uint64_t FIXED_RANDOM = rand();
        return splitmix64(x + FIXED_RANDOM);
    }
};

unordered_map<int, int, custom_hash> mp;


//Fasting unordered_set/map
st.max_load_factor(0.25); st.reserve(512);
mp.max_load_factor(0.25); mp.reserve(1024); // 1024 / 512
```

15 Gaussian Elimination (MOD)

```
typedef vector<ll> vll;

ll fp(ll x, ll p,ll mod) {
 if (!p) return 1;
 ll y = fp(x, p >> 1, mod);
 y = y * y % mod;
 if (p & 1)
  y = y * x % mod;
 return y;
}

ll inverse(ll x, ll mod) {
 return fp(x, mod - 2, mod);
}

int Solve(vector<vll> a, vll& ans,ll mod) {
 int N = a.size();
 int M = a[0].size() - 1;
 int i, c, k;
 ll temp;
 for (c = 0; c < N; c++) {
  temp = -1;
  for(i = c;i<N;i++)
   if (a[i][c]) {
    temp = i;
    break;
   }
  if (temp == -1 )
   continue;
  if (temp != c)
   for (i = c; i <= M; i++)
    swap(a[temp][i], a[c][i]);
  ll inv = inverse(a[c][c], mod);
  for (i = 0; i < N; i++)
   if (a[i][c] && i != c) {
    temp = a[i][c] * inv% mod;
    for (k = c; k <= M; k++)
     a[i][k] = (a[i][k] - a[c][k] * temp % mod + mod) % mod;
   }
 }
 int cnt = 0;
 for (c = 0; c < M; c++) {
  if (a[c][c] == 0) {
   if (a[c][M] == 0)
    cnt++;
   else return ans.clear(), -1;
  }
  ans.push_back(a[c][M] * inverse(a[c][c], mod) % mod);
 }
 return cnt;
}
```

16 Gaussian Elimination

```
typedef vector<double> vd;
const double Eps = 1e-20;
int Solve(vector<vd> a, vd& ans) {
```

```
 int N = a.size();
 int M = a[0].size() - 1;
 int i, c, k, temp;
 double t;
 for (c = 0; c < N; c++) {
  temp = c;
  for (i = c + 1; i < N; i++)
   if (fabs(a[i][c]) > fabs(a[temp][c]))
    temp = i;
  for (k = c; k <= M; k++)
   swap(a[c][k], a[temp][k]);

  for (i = 0; i < N; i++) {
   if (i == c)
    continue;
   t = a[i][c] / a[c][c];
   for (k = c; k <= M; k++)
    a[i][k] -= a[c][k] * t;
  }
 }
 int cnt = 0;
 for (c = 0; c < M; c++)
 {
  if (fabs(a[c][c]) < Eps)
  {
   if (fabs(a[c][M]) < Eps)
    cnt++;
   else return ans.clear(), -1;
  }
  ans.push_back(a[c][M] / a[c][c]);
 }
 return cnt;
}
```

17 Gaussian elimination (XOR)

```
struct Base {
 int m;
 vector <ll> A;

 Base(int m = 0) : m(m) {
  A.assign(max(m, 1), 0);
 }

 void insert(ll x) {
  for (ll i = m - 1; ~i; --i)
   if (x & (1LL << i)) {
    if (A[i])
     x ^= A[i];
    else {
     A[i] = x;
     break;
    }
   }
 }

 vector <ll> Get() {
  vector <ll> Res;
  for (int i = 0; i < m; i++)
   if (A[i])
    Res.push_back(A[i]);
  return Res;
 }
};
```

18 Geometry 2D

```
#define TT double
#define point complex<TT>
#define x real()
#define y imag()

const TT eps = 1e-9;
const double PI = acos(-1.0);

istream& operator>> (istream& is, point& p) {
 int a, b; is >> a >> b;
 p = point(a, b);
 return is;
}

ostream& operator<< (ostream& os, point& p) {
 // return os << p.x << ' ' << p.y;
 return os << "(" << p.x << ", " << p.y << ")";
}

bool operator< (const point& a, const point& b) {
 if (abs(a.x - b.x) > eps)
  return a.x < b.x;
 return a.y < b.y;
}

int sign(TT v) { return (v > eps) - (v < -eps); }

TT cross(const point &a, const point &b) {
 return imag(conj(a) * b);
}

TT dot(const point &a, const point &b) {
 return real(conj(a) * b);
}

point perp(point p) {
 return point(-p.y, p.x);
}

// 1->left turn, 0->collinear, -1->right turn
int orient(point a, point b, point c) {
 return sign(cross(b-a, c-a));
}

double angle(point v, point w) {
 double cosTheta = dot(v, w) / abs(v) / abs(w);
 return acos(max(-1.0, min(1.0, cosTheta)));
}

// point inside circle with diameter AB
bool inDisk(point a, point b, point p) {
 return dot(a - p, b - p) <= 0;
}

bool inAngle(point a, point b, point c, point p) {
 assert(orient(a,b,c) != 0);
 if (orient(a,b,c) < 0) swap(b,c);
 return orient(a,b,p) >= 0 && orient(a,c,p) <= 0;
}

double orientedAngle(point a, point b, point c) {
 if (orient(a, b, c) >= 0)
  return angle(b-a, c-a);
 else return 2.*PI - angle(b-a, c-a);
}

bool half(point p, point center) {
 p -= center;
 return p.y > 0.0 || (p.y == 0.0 && p.x < 0.0);
}

// start polar sort from vec // vec can't be (0, 0) //
bool costum_half(point p, point vec) {
 return cross(vec, p) < 0 || (cross(vec, p) == 0 && dot(vec,
      p) < 0);
}

// sort points in counterclockwise
```

```
void polar_sort(vector <point> &v, point center = point(0,
    0)) {
 sort(v.begin(), v.end(), [center](point a, point b) {
  return make_tuple(half(a, center), 0.0, norm(a)) <
   make_tuple(half(b, center), cross(a, b), norm(b));
 });
}

struct cmp_line { // sort along line ab
 point a, b;
 cmp_line(point a, point b): a(a), b(b){}
 bool operator() (const point& p, const point& q) const {
  return dot(p - a, b - a) < dot(q - a, b - a);
 }
};

//>>----Line----<<//

struct Line {
 point v;
 TT c;

 // From direction vector v and offset c
 Line(point v, TT c): v(v), c(c){}

 // From equation ax + by = c
 Line(TT a, TT b, TT c): v({b, -a}), c(c){}

 // From points P and Q
 Line(point p, point q): v(q - p), c(cross(v, p)){}

 // get two point lie on the line
 pair <point, point> get_points() {
  double a = -v.y, b = v.x; // ax + by = -c
  if (sign(a) == 0)
   return {point(0, -c / b), point(1, -c / b)};
  else if (sign(b) == 0)
   return {point(-c / a, 0), point(-c / a, 1)};
  else
   return {point(0, -c / b), point(1, (-c - a) / b)};
 }

 // 1 if p on the left, -1 if p on the right, 0 if p on the
     line
 int side(point p) {
  return sign(cross(v, p) - c);
 }

 // minimum distance from point p to this line
 double dist(point p) {
```

```
  return abs(cross(v, p) - c) / abs(v);
 }

 // project point p to this line
 point proj(point p) {
  return p - perp(v) * (cross(v, p) - c) / dot(v, v);
 }

 // compare two points by their orthogonal projection on
     this line
 bool cmpProj(point p, point q) {
  return dot(v, p) < dot(v, q);
 }

 // translate the line by vector t i.e. shifting it by
     vector t
 Line translate(point t) {
  return Line(v, c + cross(v, t));
 }

 Line perpThrough(point p) {
  return {p, p + perp(v)};
 }

 // -1->coincide, 0->parallel, 1->intersected
 int intersect(Line l, point &p) {
  TT d = cross(v, l.v);
  if (abs(d) > eps) {
   p = (l.v * c - v * l.c) / d;
   return 1;
  }
  if (abs(cross(v, get_points().first - l.get_points().first
      )) > eps)
   return 0;
  return -1;
 }
};

// angle bisector is a line that forms equal angles with l1
    and l2
// interior points between the direction vectors of l1 and
    l2
Line bisector(Line l1, Line l2, bool interior) {
 assert(cross(l1.v, l2.v) != 0); // l1 and l2 cannot be
     parallel!
 TT sign = interior ? 1 : -1;
 return {l2.v / abs(l2.v) + l1.v / abs(l1.v) * sign,
   l2.c / abs(l2.v) + l1.c / abs(l1.v) * sign};
}
```

```
//>>----Segment----<<//

bool onSegment(point a, point b, point p) {
 return orient(a, b, p) == 0 && inDisk(a, b, p);
}

// -2 -> not parallel and no intersection
// -1 -> coincide with no common point
// 0 -> parallel and not coincide
// 1 -> intersected ('p' is intersection of segments)
// 2 -> coincide with common points ('p' is one of the end
// points lying on both segments)
int seg_seg_inter(point a, point b, point c, point d, point
     &p) {
 int s = Line(a, b).intersect(Line(c, d), p);
 if (s == 0) return 0;
 else if (s == -1) {
  // '< -eps' excludes endpoints in the coincide case
  if (dot(a - c, a - d) < eps)
   return p = a, 2;
  if (dot(b - c, b - d) < eps)
   return p = b, 2;
  if (dot(c - a, c - b) < eps)
   return p = c, 2;
  return -1;
 }
 // '< -eps' excludes endpoints in intersected case
 if (dot(p - a, p - b) < eps && dot(p - c, p - d) < eps)
  return 1;
 return -2;
}

double seg_point_dist(point a, point b, point p) {
 if (a != b) {
  Line l(a, b);
  if (l.cmpProj(a, p) && l.cmpProj(p, b))
   return l.dist(p);
 }
 return min(abs(p - a), abs(p - b));
}

//>>----Polygon----<<//

// using length of sides
double triangle_area(double a, double b, double c) {
 double s = (a + b + c) / 2.0;
 return sqrt(s * (s - a) * (s - b) * (s - c));
}

// other way using picks theorem
```

```
// Area = InsidePoints + 0.5*BoundaryPoints - 1
double polygon_area(vector<point>& pts) {
 double area = 0;
 for (int i = 0, n = pts.size(); i < n; i++)
  area += cross(pts[i], pts[i + 1 == n ? 0 : i + 1]);
 return abs(area) / 2.0;
}

// check if polygon is convex
bool isConvex(vector<point>& p) {
 bool hasPos = false, hasNeg = false;
 for (int i = 0, n = p.size(); i < n; i++) {
  int o = orient(p[i], p[(i + 1) % n], p[(i + 2) % n]);
  if (o > 0) hasPos = true;
  if (o < 0) hasNeg = true;
 }
 return !(hasPos && hasNeg);
}

// check if [PQ] crosses ray from A
bool crossesRay(point a, point p, point q) {
 auto above = [](point _a, point _b) {
  return _b.y >= _a.y;
 };
 return (above(a, q) - above(a, p)) * orient(a, p, q) > 0;
}

// amplitude travelled around point A, from P to Q
double angleTravelled(point a, point p, point q) {
 // remainder ensures the value is in [-pi,pi]
 return remainder(arg(q - a) - arg(p - a), 2 * PI);
}

int windingNumber(vector<point>& p, point a) {
 double ampli = 0;
 for (int i = 0, n = p.size(); i < n; i++)
  ampli += angleTravelled(a, p[i], p[i + 1 == n ? 0 : i +
      1]);
 return round(ampli / (2 * PI));
}

int inside_polygon(vector<point>& pts, point p, bool strict
    = true) {
 int can = 0;
 for (int i = 0, n = pts.size(); i < n; i++) {
  point a = pts[i], b = pts[i + 1 == n ? 0 : i + 1];
  if (onSegment(a, b, p)) return !strict;
  can ^= crossesRay(p, a, b);
 }
 return can;
}

vector<point> convex_hull(vector<point> pts) {
 if (pts.size() <= 1)
  return pts;

 sort(pts.begin(), pts.end(), [](point a, point b){ return a
      < b; });
 vector<point> up, dn;
 for (auto& p : pts) {
  // remove equal if all points on boarder needed
  while (up.size() > 1 && orient(up[up.size() - 2], up.back
      (), p) >= 0)
   up.pop_back();
  while (dn.size() > 1 && orient(dn[dn.size() - 2], dn.back
      (), p) <= 0)
   dn.pop_back();

  up.push_back(p);
  dn.push_back(p);
 }

 pts = dn;
 reverse(up.begin(), up.end());
 for (auto& p : up)
  if (p != dn[0] && p != dn.back())
   pts.push_back(p);

 return pts;
}

// left side of a convex polygon with respect to a line PQ
vector <point> convex_cut(vector<point> pts, point p, point
     q) {
 vector<point> qs;
 for (int i = 0, n = pts.size(); i < n; ++i) {
  int j = (i + 1 == n ? 0 : i + 1);
  if (sign(cross(p - pts[i], q - pts[i])) >= 0)
   qs.push_back(pts[i]);
  if (sign(cross(p - pts[i], q - pts[i])) *
    sign(cross(p - pts[j], q - pts[j])) < 0) {
   auto a = cross(pts[j] - pts[i], q - p);
   auto b = cross(p - pts[i], q - p);
   qs.push_back(pts[i] + b / a * (pts[j] - pts[i]));
  }
 }
 return qs;
}

// centroid of a (possibly non-convex) polygon,
// assuming that the coordinates are listed in a CW or CCW
// centroid is often known as the "center of mass".
point centroid(vector<point> &p) {
 int n = p.size(); point c(0, 0);
 double sum = 0;
 for (int i = 0; i < n; i++)
  sum += cross(p[i], p[(i + 1) % n]);
 double scale = 3.0 * sum;
 for (int i = 0; i < n; i++) {
  int j = (i + 1) % n;
  c = c + (p[i] + p[j]) * cross(p[i], p[j]);
 }
 return c / scale;
}

double polygon_diameter(vector<point> &p) {
 int n = (int)p.size();
 if (n == 1) return 0;
 if (n == 2) return abs(p[0] - p[1]);
 double ans = 0;
 for (int i = 0, j = 1; i < n; i++) {
  while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n] - p[j])
       >= 0) {
   ans = max(ans, norm(p[i] - p[j]));
   j = (j + 1) % n;
  }
  ans = max(ans, norm(p[i] - p[j]));
 }
 return sqrt(ans);
}

//>>----Circle----<<//

struct Circle {
 point o;
 double r;

 Circle(){}
 Circle(point c, double r): o(c), r(r){}

 // circle describe by three points
 Circle(point a, point b, point c) {
  b -= a, c -= a;
  assert(cross(b, c) != 0);
  this->o = a + perp(b * dot(c, c) - c * dot(b, b)) / cross(
      b, c) / 2.0;
  this->r = abs(o - a);
 }

 int intersect_line(Line l, vector<point> &inter) {
```

```
 double h2 = r * r - l.dist(o)*l.dist(o);
 if (h2 >= 0) {
  point p = l.proj(o);
  point h = l.v * sqrt(h2) / abs(l.v);
  inter.push_back(p + h);
  if (abs(h) > eps)
   inter.push_back(p - h);
 }
 return (int) inter.size();
}

int intersect_circle(Circle c, vector<point> &inter) {
 point d = c.o - o;
 double d2 = dot(d, d);
 if (d2 == 0) {
  assert(r != c.r); // same circle
  return 0; // lie inside
 }
 double pd = (d2 + r * r - c.r * c.r) / 2;
 double h2 = r * r - pd * pd / d2;
 if (h2 >= 0) {
  point p = o + d * pd / d2, h = perp(d) * sqrt(h2 / d2);
  inter.push_back(p + h);
  if (abs(h) > eps)
   inter.push_back(p - h);
 }
 return (int) inter.size();
}

// long double recommended
double circ_inter_area(Circle c) {
 double d = abs(c.o - o);
 if (d <= (c.r - r)) return r * r * PI;
 if (d <= (r - c.r)) return c.r * c.r * PI;
 if (d >= r + c.r) return 0;
 double alpha = acos((r * r + d * d - c.r * c.r) / (2 * r *
      d));
 double beta = acos((c.r * c.r + d * d - r * r) / (2 * c.r
     * d));
 return r * r * (alpha - 0.5 * sin(2 * alpha)) + c.r * c.r
     * (beta - 0.5 * sin(2 * beta));
}

int tangents(Circle c, bool inner, vector<Line> &out) {
 if (inner) c.r = -c.r;
 point d = c.o - o;
 double dr = r - c.r, d2 = dot(d, d), h2 = d2 - dr * dr;
 if (d2 == 0 || h2 < 0) {
  assert(h2 != 0);
  return 0;
 }
 for (double sign : {-1, 1}) {
  point v = (d * dr + perp(d) * sqrt(h2) * sign) / d2;
  point a = o + v * r, b = c.o + v * c.r;
  if (fabs(a.x - b.x) < eps && fabs(a.y - b.y) < eps) {
   out.push_back(Line(o, a).perpThrough(a));
   break;
  }
  out.push_back(Line(a, b));
 }
 return 1 + (h2 > 0);
 }
};

// find a circle of radius r that contains many points as
    possible
int maximum_circle_cover(vector<point> ps, double r) {
 const double dx[] = {1, -1, -1, 1}, dy[] = {1, 1, -1, -1};
 point best_p;
 int best = 0;
 function<void(point, double, vector<point>)>
 rec = [&](point p, double w, vector<point> ps) {
  w /= 2;
  point qs[4];
  vector<point> pss[4];
  for (int i = 0; i < 4; ++i) {
   qs[i] = p + w * point(dx[i], dy[i]);
   int lo = 0;
   for (point q : ps) {
    auto d = abs(qs[i] - q);
    if (sign(d - r) <= 0) ++lo;
    if (sign(d - w * sqrt(2) - r) <= 0)
     pss[i].push_back(q);
   }
   if (lo > best) {
    best = lo;
    best_p = qs[i];
   }
  }
  for (int i = 0, j; i < 4; ++i) {
   for (int j = i + 1; j < 4; ++j)
    if (pss[i].size() < pss[j].size())
     swap(pss[i], pss[j]), swap(qs[i], qs[j]);
   if (pss[i].size() <= best)
    break;
   rec(qs[i], w, pss[i]);
  }
 };
 TT w = 0;
 for (point p : ps)
  w = max(w, max(abs(p.x), abs(p.y)));
 rec({0, 0}, w, ps);
 return best; //best_p;
}

// given n points, find the minimum enclosing circle of the
    points
// call convex_hull() before this for faster solution
// expected O(n)
Circle minimum_enclosing_circle(vector<point> &p) {
 random_shuffle(p.begin(), p.end());
 int n = p.size();
 Circle c(p[0], 0);
 for (int i = 1; i < n; i++)
  if (sign(abs(c.o - p[i]) - c.r) > 0) {
   c = Circle(p[i], 0);
   for (int j = 0; j < i; j++)
    if (sign(abs(c.o - p[j]) - c.r) > 0) {
     c = Circle((p[i] + p[j]) / 2.0, abs(p[i] - p[j]) / 2.0);
     for (int k = 0; k < j; k++)
      if (sign(abs(c.o - p[k]) - c.r) > 0)
       c = Circle(p[i], p[j], p[k]);
    }
  }
 return c;
}

//>>----Union----<<//
//O(n^2 log n)
struct CircleUnion {
 int n;
 double x[2020], y[2020], r[2020];
 int covered[2020];
 vector<pair<double, double> > seg, cover;
 double arc, pol;
 inline int sign(double x) {return x < -eps ? -1 : x > eps;}
 inline int sign(double x, double y) {return sign(x - y);}
 inline double SQ(const double x) {return x * x;}
 inline double dist(double x1, double y1, double x2, double
     y2) {
  return sqrt(SQ(x1 - x2) + SQ(y1 - y2));
 }
 inline double angle(double A, double B, double C) {
  double val = (SQ(A) + SQ(B) - SQ(C)) / (2 * A * B);
  if (val < -1) val = -1;
  if (val > +1) val = +1;
  return acos(val);
 }
 CircleUnion() {
  n = 0;
```

```
 seg.clear(), cover.clear();
 arc = pol = 0;
}
void init() {
 n = 0;
 seg.clear(), cover.clear();
 arc = pol = 0;
}
void add(double xx, double yy, double rr) {
 x[n] = xx, y[n] = yy, r[n] = rr, covered[n] = 0, n++;
}
void getarea(int i, double lef, double rig) {
 arc += 0.5 * r[i] * r[i] * (rig - lef - sin(rig - lef));
 double x1 = x[i] + r[i] * cos(lef), y1 = y[i] + r[i] * sin
     (lef);
 double x2 = x[i] + r[i] * cos(rig), y2 = y[i] + r[i] * sin
     (rig);
 pol += x1 * y2 - x2 * y1;
}
double solve() {
 for (int i = 0; i < n; i++)
  for (int j = 0; j < i; j++)
   if (!sign(x[i] - x[j]) && !sign(y[i] - y[j]) && !sign(r[i
       ] - r[j])) {
    r[i] = 0.0;
    break;
   }
 for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
   if (i != j && sign(r[j] - r[i]) >= 0 && sign(dist(x[i], y
       [i], x[j], y[j]) - (r[j] - r[i])) <= 0) {
    covered[i] = 1;
    break;
   }
 for (int i = 0; i < n; i++) {
  if (sign(r[i]) && !covered[i]) {
   seg.clear();
   for (int j = 0; j < n; j++) {
    if (i != j) {
     double d = dist(x[i], y[i], x[j], y[j]);
     if (sign(d - (r[j] + r[i])) >= 0 || sign(d - abs(r[j] -
         r[i])) <= 0)
      continue;
     double alpha = atan2(y[j] - y[i], x[j] - x[i]);
     double beta = angle(r[i], d, r[j]);
     pair<double, double> tmp(alpha - beta, alpha + beta);
     if (sign(tmp.first) <= 0 && sign(tmp.second) <= 0)
      seg.push_back(pair<double, double>(2 * PI + tmp.first,
          2 * PI + tmp.second));
     else if (sign(tmp.first) < 0) {
      seg.push_back(pair<double, double>(2 * PI + tmp.first,
          2 * PI));
      seg.push_back(pair<double, double>(0, tmp.second));
     }
     else seg.push_back(tmp);
    }
   }
   sort(seg.begin(), seg.end());
   double rig = 0;
   for (vector<pair<double, double> >::iterator iter = seg.
       begin(); iter != seg.end(); iter++) {
    if (sign(rig - iter->first) >= 0)
     rig = max(rig, iter->second);
    else {
     getarea(i, rig, iter->first);
     rig = iter->second;
    }
   }
   if (!sign(rig))
    arc += r[i] * r[i] * PI;
   else
    getarea(i, rig, 2 * PI);
  }
 }
 return pol / 2.0 + arc;
 }
} CU;

// calculates the area of the union of n polygons (not
    necessarily convex).
// the points within each polygon must be given in CCW order
    .
// complexity: O(N^2), where N is the total number of points
double rat(point a, point b, point p) {
 return !sign(a.x - b.x) ? (p.y - a.y) / (b.y - a.y) : (p.x
     - a.x) / (b.x - a.x);
};
double polygon_union(vector<vector<point>> &p) {
 int n = p.size();
 double ans = 0;
 for (int i = 0; i < n; ++i) {
  for (int v = 0; v < (int)p[i].size(); ++v) {
   point a = p[i][v], b = p[i][(v + 1) % p[i].size()];
   vector<pair<double, int>> segs;
   segs.emplace_back(0, 0), segs.emplace_back(1, 0);
   for (int j = 0; j < n; ++j) {
    if (i != j) {
     for (size_t u = 0; u < p[j].size(); ++u) {
      point c = p[j][u], d = p[j][(u + 1) % p[j].size()];
      int sc = sign(cross(b - a, c - a)), sd = sign(cross(b -
          a, d - a));
      if (!sc && !sd) {
       if (sign(dot(b - a, d - c)) > 0 && i > j)
        segs.emplace_back(rat(a, b, c), 1), segs.emplace_back
            (rat(a, b, d), -1);
      }
      else {
       double sa = cross(d - c, a - c), sb = cross(d - c, b -
           c);
       if (sc >= 0 && sd < 0)
        segs.emplace_back(sa / (sa - sb), 1);
       else if (sc < 0 && sd >= 0)
        segs.emplace_back(sa / (sa - sb), -1);
      }
     }
    }
   }
   sort(segs.begin(), segs.end());
   double pre = min(max(segs[0].first, 0.0), 1.0), now, sum
       = 0;
   int cnt = segs[0].second;
   for (int j = 1; j < segs.size(); ++j) {
    now = min(max(segs[j].first, 0.0), 1.0);
    if (!cnt) sum += now - pre;
    cnt += segs[j].second;
    pre = now;
   }
   ans += cross(a, b) * sum;
  }
 }
 return ans * 0.5;
}

//>>----Cool Stuff----<<//

// minimum perimeter #note: define inf
double minimum_enclosing_rectangle(vector<point> &pts) {
 const double inf = 1e18;
 int n = pts.size();
 if (n <= 2)
  return abs(pts[0] - pts[1 % n]);

 int mndot = 0;
 double tmp = dot(pts[1] - pts[0], pts[0]);
 for (int i = 1; i < n; i++) {
  if (dot(pts[1] - pts[0], pts[i]) <= tmp) {
   tmp = dot(pts[1] - pts[0], pts[i]);
   mndot = i;
  }
```

```
 }
 double ans = inf;
 for (int i = 0, j = 1, mxdot = 1; i < n; i++) {
  point cur = pts[(i + 1) % n] - pts[i];
  while (cross(cur, pts[(j + 1) % n] - pts[j]) >= 0)
   j = (j + 1) % n;
  while (dot(pts[(mxdot + 1) % n], cur) >= dot(pts[mxdot],
       cur))
   mxdot = (mxdot + 1) % n;
  while (dot(pts[(mndot + 1) % n], cur) <= dot(pts[mndot],
       cur))
   mndot = (mndot + 1) % n;
  ans = min(ans, 2.0 * ((dot(pts[mxdot], cur) / fabs(cur) -
       dot(pts[mndot], cur) / fabs(cur)) + Line(pts[i], pts[(
       i + 1) % n]).dist(pts[j])));
 }
 return ans;
}

// it returns a point such that the sum of distances
// from that point to all points in p is minimum O(n log^2
     MX)
point geometric_median(vector<point> p) {
 const double MX = 1e5; // boundary
 auto tot_dist = [&](point z) {
  double res = 0;
  for (int i = 0; i < p.size(); i++)
   res += abs(p[i] - z);
  return res;
 };
 auto findY = [&](double _x) { // auto may not work
  double yl = -MX, yr = MX;
  for (int i = 0; i < 60; i++) {
   double ym1 = yl + (yr - yl) / 3;
   double ym2 = yr - (yr - yl) / 3;
   double d1 = tot_dist(point(_x, ym1));
   double d2 = tot_dist(point(_x, ym2));
   if (d1 < d2) yr = ym2;
   else yl = ym1;
  }
  return pair<double, double>(yl, tot_dist(point(_x, yl)));
 };
 double xl = -MX, xr = MX;
 for (int i = 0; i < 60; i++) {
  double xm1 = xl + (xr - xl) / 3;
  double xm2 = xr - (xr - xl) / 3;
  double y1, d1, y2, d2;
  tie(y1, d1) = findY(xm1);
  tie(y2, d2) = findY(xm2);
  if (d1 < d2) xr = xm2;
  else xl = xm1;
 }
 return point(xl, findY(xl).first);
}

double closest_pair(vector<point> pts) {
 double res = 1e18;
 struct compare {
  bool operator()(const point& a, const point& b) const {
   return make_pair(a.x, a.y) < make_pair(b.x, b.y);
  }
 };

 multiset<point, compare> curWin;
 sort(pts.begin(), pts.end(), [](point a, point b) {
  return make_pair(a.x, a.y) < make_pair(b.x, b.y);
 });

 for (int right = 0, left = 0; right < (int)pts.size();
      right++) {
  int d = ceil(sqrt(res));
  while (left < right && pts[right].x - pts[left].x >= res)
       {
   curWin.erase(curWin.find(point(pts[left].y, pts[left].x))
       );
   left++;
  }
  auto L = curWin.lower_bound(point(pts[right].y - d, pts[
       right].x));
  auto R = curWin.upper_bound(point(pts[right].y + d, pts[
       right].x));
  for (; L != R; L++) {
   point cur = point(L->y, L->x);
   res = min(res, fabs(pts[right] - cur));
  }
  curWin.insert(point(pts[right].y, pts[right].x));
 }
 return sqrt(res);
}
```

19 Geometry 3D

```
#define TT double

const double eps = 1e-9;
const double PI = acos(-1.0);

int sign(TT v) { return (v > eps) - (v < -eps); }

//>>----Point----<<//
struct point {
    TT x, y, z;

    point(TT x = 0, TT y = 0, TT z = 0):
        x(x), y(y), z(z){}

    point operator+ (point p) { return point(x + p.x, y + p.y
         , z + p.z); }
    point operator- (point p) { return point(x - p.x, y - p.y
         , z - p.z); }
    point operator* (double d) { return point(x * d, y * d, z
          * d); }
    point operator/ (double d) { return point(x / d, y / d, z
          / d); }
    bool operator== (point p) { return tie(x, y, z) == tie(p.
         x, p.y, p.z); }
    bool operator!= (point p) { return !operator==(p); }

    TT operator| (const point& p) { // dot
        return x * p.x + y * p.y + z * p.z;
    }

    point operator* (const point& p) { // cross
        return {y * p.z - z * p.y,
                z * p.x - x * p.z,
                x * p.y - y * p.x};
    }

    TT sq() { return (*this | *this); }
    double abs() { return sqrt(sq()); }
    point unit() { return *this / abs(); }

} zero(0, 0, 0);

double angle(point v, point w) {
    double cosTheta = (v | w) / abs(v) / abs(w);
    return acos(max(-1.0, min(1.0, cosTheta)));
}

double smallAngle(point v, point w) {
    return acos(min(abs(v | w) / v.abs() / w.abs(), 1.0));
}

// 1->S above PQR, 0->S on PQR, -1->S below PQR
TT orient(point p, point q, point r, point s) {
    return sign((q - p)*(r - p)|(s - p));
}
```

```
// 1->left turn, 0->collinear, -1->right turn
TT orientByNormal(point p, point q, point r, point n) {
    return sign((q - p) * (r - p) | n);
}

//>>----Plane----<<//
struct Plane {
    point n;
    TT d;

    // From normal n and offset d
    Plane(point n, TT d): n(n), d(d) {}
    // From normal n and point P
    Plane(point n, point p): n(n), d(n | p) {}
    // From three non-collinear points P,Q,R
    Plane(point p, point q, point r): Plane((q - p) * (r - p)
        , p) {}

    // >0 -> above, =0 -> on, <0 -> below
    TT side(point p) {
        return (n | p) - d;
    }

    double dist(point p) {
        return abs(side(p)) / n.abs();
    }

    Line perpThrough(point p) {
        return Line(p, p + n);
    }

    Plane translate(point t) {
        return Plane(n, d + (n | t));
    }

    Plane shiftUp(double dist) {
        return Plane(n, d + dist * n.abs());
    }

    point proj(point p) { return p - n * side(p) / n.sq(); }
    point refl(point p) { return p - n * 2 * side(p) / n.sq()
        ; }

    bool isParallel(Plane p) {
        return n * p.n == zero;
    }
    bool isPerp(Plane p) {
        return (n | p.n) == 0;
    }
};
```

```
double angle(Plane p1, Plane p2) {
    return smallAngle(p1.n, p2.n);
}

//>>----Line----<<//
struct Line {
    point d, o;

    // From two points P, Q
    Line(point p, point q): d(q - p), o(p) {}

    // From two planes p1, p2 (requires T = double)
    Line(Plane p1, Plane p2) {
        d = p1.n * p2.n;
        o = (p2.n * p1.d - p1.n * p2.d) * d / d.sq();
    }

    double sqDist(point p) { return (d * (p - o)).sq() / d.sq
        (); }
    double dist(point p) { return sqrt(sqDist(p)); }

    bool cmpProj(point p, point q) {
        return (d | p) < (d | q);
    }

    point proj(point p) { return o + d * (d | (p - o)) / d.sq
        (); }
    point refl(point p) { return proj(p) * 2 - p; }

    point plane_interect(Plane p) {
        return o - d * p.side(o) / (d | p.n);
    }

    double dist_line(Line l) {
        point n = d * l.d;
        if (n == zero) // parallel
            return dist(l.o);
        return abs((l.o - o) | n) / n.abs();
    }

    // nearest point to l that lie on this Line
    point closestOnLine(Line l) {
        point n2 = l.d * (d * l.d);
        return o + d * ((l.o - o) | n2) / (d | n2);
    }

    bool isParallel(Line l) {
        return d * l.d == zero;
    }
```

```
    bool isPerp(Line l) {
        return (d | l.d) == 0;
    }
};

double angle(Line l1, Line l2) {
    return smallAngle(l1.d, l2.d);
}

double angle(Plane p, Line l) {
    return PI / 2 - smallAngle(p.n, l.d);
}

//>>----Polyhedrons----<<//

point vectorArea2(vector<point> pts) {
    point S = zero;
    for (int i = 0, n = pts.size(); i < n; i++)
        S = S + pts[i] * pts[(i + 1) % n];
    return S;
}
double polyhedron_area(vector<point> pts) {
    return vectorArea2(pts).abs() / 2.0;
}

double polyhedron_volume(vector<vector<point>> fs) {
    double vol6 = 0.0;
    for (vector<point> f : fs)
        vol6 += (vectorArea2(f) | f[0]);
    return abs(vol6) / 6.0;
}

//>>----Spherical----<<//
// latitude goes up
// longitude goes left

point sph(double r, double lat, double lon) {
    lat *= PI / 180, lon *= PI / 180;
    return {r * cos(lat) * cos(lon), r * cos(lat) * sin(lon),
         r * sin(lat)};
}

int sphere_line_inter(point o, double r, Line l, vector<
    point> &inter) {
    double h2 = r * r - l.sqDist(o);
    if (h2 < 0) return 0;
    point p = l.proj(o);
    point h = l.d * sqrt(h2) / l.d.abs();
    inter.push_back(p - h);
    if (h.abs() > eps)
```

```cpp
        inter.push_back(p + h);
    return 1 + (h2 > 0);
}

double greatCircleDist(point o, double r, point a, point b)
    {
    return r * angle(a - o, b - o);
}

bool validSegment(point a, point b) {
    return a * b != zero || (a | b) > 0;
}

bool properInter(point a, point b, point c, point d, point &
    out) {
    point ab = a * b, cd = c * d; // normals of planes OAB
        and OCD
    int oa = sign(cd | a),
        ob = sign(cd | b),
        oc = sign(ab | c),
        od = sign(ab | d);
    out = ab * cd * od; // four multiplications => careful
        with overflow!
    return (oa != ob && oc != od && oa != oc);
}

bool onSphSegment(point a, point b, point p) {
    point n = a * b;
    if (n == zero)
        return a * p == zero && (a | p) > 0;
    return (n | p) == 0 && (n | a * p) >= 0 && (n | b * p) <=
         0;
}

struct directionSet : vector<point> {
    using vector::vector; // import constructors
    void insert(point p) {
        for (point q : *this)
            if (p * q == zero) return;
        push_back(p);
    }
};
directionSet intersSph(point a, point b, point c, point d) {
    assert(validSegment(a, b) && validSegment(c, d));
    point out;
    if (properInter(a, b, c, d, out)) return {out};
    directionSet s;
    if (onSphSegment(c, d, a)) s.insert(a);
    if (onSphSegment(c, d, b)) s.insert(b);
    if (onSphSegment(a, b, c)) s.insert(c);
    if (onSphSegment(a, b, d)) s.insert(d);
    return s;
}

// A B C on the sphere surface
double angleSph(point a, point b, point c) {
    return angle(a * b, a * c);
}

double orientedAngleSph(point a, point b, point c) {
    if ((a * b | c) >= 0)
        return angleSph(a, b, c);
    else
        return 2 * PI - angleSph(a, b, c);
}

// area of polygon on sphere
double areaOnSphere(double r, vector<point> p) {
    int n = p.size();
    double sum = -(n - 2) * PI;
    for (int i = 0; i < n; i++)
        sum += orientedAngleSph(p[(i + 1) % n], p[(i + 2) % n
            ], p[i]);
    return r * r * sum;
}

int windingNumber3D(vector<vector<point>> fs) {
    double sum = 0;
    for (vector<point> f : fs)
        sum += remainder(areaOnSphere(1, f), 4 * PI);
    return round(sum / (4 * PI));
}
```

20 HLD

```cpp
 const  int N = 1e5 + 100;
vector <int> Tree[N];
struct HLD {
 const int ON_EDGE = 0;

 int cur_pos;
 vector<int> parent, depth, heavy, head, pos, rpos;

 void init(int n) {
  cur_pos = 0;
  parent.resize(++n);
  head.resize(n);
  depth.resize(n);
  pos.resize(n);
  heavy.resize(n, -1);
  rpos.resize(n); // reverse mapping
  DFS(1, 0);
  decompose(1, 1);
 }

 int DFS(int u, int p) {
  parent[u] = p;
  int size = 1, mx = 0;
  for (int v : Tree[u]) if (v != p) {
   depth[v] = depth[u] + 1;
   int x = DFS(v, u);
   size += x;
   if (x > mx)
    mx = x, heavy[u] = v;
  }
  return size;
 }

 void decompose(int u, int h) {
  head[u] = h;
  rpos[pos[u] = ++cur_pos] = u;
  if (heavy[u] != -1)
   decompose(heavy[u], h);
  for (auto v : Tree[u])
   if (v != parent[u] && v != heavy[u])
    decompose(v, v);
 }

 vector <pair<int, int>> getRanges(int u, int v) {
  int swaps = 0;
  vector <pair<int, int>> a, b;
  for (; head[u] != head[v]; v = parent[head[v]]) {
   if (depth[head[u]] > depth[head[v]])
    swap(u, v), swap(a, b), swaps ^= 1;
   b.emplace_back(pos[head[v]], pos[v]);
  }

  if (depth[u] > depth[v])
   swap(u, v), swap(a, b), swaps ^= 1;

  if (!(ON_EDGE && u == v))
   b.emplace_back(pos[u] + ON_EDGE, pos[v]);

  if (swaps)
   swap(a, b);

  a.insert(a.end(), b.rbegin(), b.rend());
  return a;
```

```
 }
} hld;
```

21 Hash

```
// Big: 976791889 998260643 953367143 915397319
// Small: 1093 2683 1151 1193 2957 1289
ll const SEED1 = 1337;
ll const SEED2 = 3623;
ll const MOD1 = 1515151531;
ll const MOD2 = 1e9 + 7;

struct Hash {
 ll SEED, MOD;
 vector<ll> pre;
 vector<ll> power;

 Hash(){}
 Hash(const string& s, ll SEED = SEED1, ll MOD = MOD1):
  SEED(SEED), MOD(MOD) {
  pre = vector<ll>(s.size());
  power = vector<ll>(s.size());
  build(s);
 }

 void build(const string& s) {
  power[0] = 1;
  for (int i = 1; i < (int) s.size(); i++)
   power[i] = (SEED * power[i - 1]) % MOD;

  pre[0] = s[0];
  for (int i = 1; i < (int) s.size(); i++)
   pre[i] = (SEED * pre[i - 1] + s[i]) % MOD;
 }

 ll get(int i, int j) {
  ll res = pre[j];
  if (i) res -= pre[i - 1] * power[j - i + 1];
  res = (res % MOD + MOD) % MOD;
  return res;
 }
};
```

22 LCA O(1)

```
struct LCA {
    vector<vector<int>> Dp;
    vector<int> Euler, Bit, F, H;
    void DFS(int u, int p, vector<vector<int>> &G) {
        H[u] = H[p] + 1;
        F[u] = Euler.size();
        Euler.push_back(u);
        for (auto v : G[u])
        {
            if (v == p)
                continue;
            DFS(v, u, G);
            Euler.push_back(u);
        }
    }
    int MIN(int &a, int &b) {
        if (a == 0)
            return b;
        if (b == 0)
            return a;
        return H[a] > H[b] ? b : a;
    }
    void Build(vector<vector<int>> &G, int root) {
        H = F = vector<int>(G.size(), 0);
        DFS(root, 0, G);
        int n = Euler.size();
        Dp = vector<vector<int>>(n, vector<int>(20, 0));
        for (int i = 0; i < n; i++)
            Dp[i][0] = Euler[i];
        Bit = vector<int>(1 << 20, 0);
        for (int i = 1; i < (1 << 20); i++)
            for (int j = 0; j < 20; j++)
                if (i & 1 << j)
                    Bit[i] = j;
        for (int j = 1; j < 20; j++)
            for (int i = 0; i < n; i++) {
                if (i + (1 << j) >= n)
                    break;
                Dp[i][j] = MIN(Dp[i][j - 1], Dp[i + (1 << j -
                    1)][j - 1]);
            }
    }
    int FindLCA(int u, int v) {
        if (F[v] < F[u])
            swap(u, v);
        int f1 = F[u];
        int f2 = F[v];
        int b = Bit[f2 - f1 + 1];
        return MIN(Dp[f1][b], Dp[f2 - (1 << b) + 1][b]);
    }
    LCA() {}
    LCA(vector<int> G[], int n, int root = 1) {
        vector<vector<int>> GG;
        for (int i = 0; i <= n; i++)
            GG.push_back(G[i]);
        Build(GG, root);
    }
    LCA(vector<vector<int>> &G, int root = 1) {
        Build(G, root);
    }
} cc;
```

23 Manacher

```
string s;
int rad[2 * LEN], n; // WARNING: n = 2 * strlen(s)

void build_rad() {
 for (int i = 0, j = 0, k; i < n; i += k, j = max(j - k, 0))
      {
  for (; i >= j && i + j + 1 < n && s[(i - j) / 2] == s[(i +
       j + 1) / 2]; ++j);
  rad[i] = j;
  for (k = 1; i >= k && rad[i] >= k && rad[i - k] != rad[i]
      - k; ++k)
   rad[i + k] = min(rad[i - k], rad[i] - k);
 }
}

// WARNING: n = s.size() [0..n-1]
bool is_palindrome(int L, int R) {
 return L >= 0 && R < n && rad[L + R] >= R - L + 1;
}

int main() {
 n = 2 * s.size();
 build_rad();
}
```

24 Math

```
   // SumOfPowers
ll SumPows(ll a, ll k) { // a^1 + a^2 + a^3 + .. a^k
 if (!k) return 0;
 if (k&1) return a * (1 + SumPows(a, k - 1));
```

```
 ll Half = SumPows(a, k >> 1);
 return Half * (1 + Half - SumPows(a, k/2 - 1));
}

// Linear Sieve
const int N = 10000000;
int lp[N + 1];
vector<int> pr;
for (int i = 2; i <= N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;
        pr.push_back(i);
    }
    for (int j = 0; j < (int)pr.size() && pr[j] <= lp[i] && i
         * pr[j] <= N; ++j)
        lp[i * pr[j]] = pr[j];
}

// Phi function O(N * logN)
unsigned long long p[5000005];
for (int i = 2; i < Max; i++) {
 p[i] = 1;
 for(int i=2;i<Max;i++)
  if (p[i] == 1) {
   for (ll j = i; j < Max; j *= i) {
    int p1 = j - j / i;
    int p2 = j / i - j / i / i;
    for (int k = j; k < Max; k += j)
     p[k] = p[k] / p2 * p1;
   }
  }
}

// Stirling Number
ll S(int n, int k) {
 ll res = 0;
 for (int i = 0; i <= k; i++)
  res += ((k + i) % 2 ? -1 : 1) * C(k, i) * fp(i, n) % Mod;
 return (res % Mod * inv_fact[k] % Mod + Mod) % Mod;
}

// Precalc Inverse
void precalc_mod() {
    inv[1] = 1;
    for (int i = 2; i < N; i++) {
        inv[i] = sub(0, mul(MOD / i, inv[MOD % i]));
    }

    fact[0] = ifact[0] = 1;
    for (int i = 1; i < N; i++) {
        fact[i] = mul(i, fact[i - 1]);
        ifact[i] = mul(inv[i], ifact[i - 1]);
    }
}

// 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;
    // d = s * x + t * y
    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;
}
```

25 Matrix

```
 struct  Matrix {
    int row, col;
    ll M[MAX_N][MAX_N];

    Matrix () {}
    Matrix (int n, int m) {
        this->row = n;
        this->col = m;
        memset(M, 0, sizeof M);
    }

    ll* operator[](int i) {
        return this->M[i];
    }

    Matrix Identity (int n) {
        Matrix Res = Matrix(n, n);
        for (int i = 0; i < n; i++)
            Res[i][i] = 1;
        return Res;
    }

    Matrix operator* (Matrix& a) {
        assert(col == a.row);
        Matrix Res = Matrix(row, a.col);
        for (int i = 0; i < row; i++)
            for (int j = 0; j < a.col; j++)
                for (int k = 0; k < col; k++)
                    Res[i][j] = (Res[i][j] + M[i][k] * a[k
                        ][j]) % Mod;
        return Res;
    }

    Matrix Power (ll K) {
        Matrix Res = Identity(row);
        Matrix cur = (*this);
        while (K > 0) {
            if (K&1LL) Res = Res * cur;
            K >>= 1;
            cur = cur * cur;
        }
        return Res;
    }
};
```

26 Max Flow (Dinic)

```
//warning : make sure that c++14 is working otherwise
     variables wont be initialized
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(
         cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
```

```
    vector<FlowEdge> edges;
    vector< vector<int> > adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.push_back(FlowEdge(v, u, cap));
        edges.push_back(FlowEdge(u, v, 0));
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid
             ++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap -
                 edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap -
                 edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

27 Min Cost Max Flow

```
//Works for negative costs, but does not work for negative
     cycles
//Complexity: O(min(E^2 *V log V, E logV * flow))
struct edge {
 int to, flow, cap, cost, rev, id;
};
struct MinCostMaxFlow {
 int nodes;
 vector<int> prio, curflow, prevedge, prevnode, q, pot;
 vector<bool> inqueue;
 vector<vector<edge> > graph;
 MinCostMaxFlow() {}
 MinCostMaxFlow(int n): nodes(n), prio(n, 0), curflow(n, 0),
 prevedge(n, 0), prevnode(n, 0), q(n, 0), pot(n, 0), inqueue
      (n, 0), graph(n) {}
 void addEdge(int source, int to, int capacity, int cost,
      int id = -1) {
  edge a = {to, 0, capacity, cost, (int)graph[to].size(),id
       };
  edge b = {source, 0, 0, -cost, (int)graph[source].size()
       ,-1};
  graph[source].push_back(a);
  graph[to].push_back(b);
 }
 void bellman_ford(int source, vector<int> &dist) {
  fill(dist.begin(), dist.end(), INT_MAX);
  dist[source] = 0;
  int qt=0;
  q[qt++] = source;
  for(int qh=0;(qh-qt)%nodes!=0;qh++) {
   int u = q[qh%nodes];
   inqueue[u] = false;
   for(auto &e : graph[u]) {
    if(e.flow >= e.cap)
     continue;
    int v = e.to;
    int newDist = dist[u] + e.cost;
    if(dist[v] > newDist) {
     dist[v] = newDist;
     if(!inqueue[v]) {
      inqueue[v] = true;
      q[qt++ % nodes] = v;
     }
    }
   }
  }
 }
 pair<int, int> minCostFlow(int source, int dest, int
      maxflow) {
  bellman_ford(source, pot);
  int flow = 0;
  int flow_cost = 0;
  while(flow < maxflow) {
   priority_queue<pair<int, int>, vector<pair<int, int> >,
        greater<pair<int, int> > > q;
   q.push({0, source});
   fill(prio.begin(), prio.end(), INT_MAX);
   prio[source] = 0;
   curflow[source] = INT_MAX;
   while(!q.empty()) {
    int d = q.top().first;
    int u = q.top().second;
    q.pop();
    if(d != prio[u])
     continue;
    for(int i=0;i<graph[u].size();i++) {
     edge &e=graph[u][i];
```

```
      int v = e.to;
      if(e.flow >= e.cap)
       continue;
      int newPrio = prio[u] + e.cost + pot[u] - pot[v];
      if(prio[v] > newPrio) {
       prio[v] = newPrio;
       q.push({newPrio, v});
       prevnode[v] = u;
       prevedge[v] = i;
       curflow[v] = min(curflow[u], e.cap - e.flow);
      }
     }
    }
    if(prio[dest] == INT_MAX)
     break;
    for(int i=0;i<nodes;i++)
     pot[i]+=prio[i];
    int df = min(curflow[dest], maxflow - flow);
    flow += df;
    for(int v=dest;v!=source;v=prevnode[v]) {
     edge &e = graph[prevnode[v]][prevedge[v]];
     e.flow += df;
     graph[v][e.rev].flow -= df;
     flow_cost += df * e.cost;
    }
   }
   return {flow, flow_cost};
  }
};
```

28 MinRotation

```
 // Description: Finds the lexicographically smallest
      rotation of a string.
 // Time: O(N)

#define rep(i, l, r) for (int i = l; i < r; i++)

int min_rotation(string s) {
 int a = 0, N = s.size(); s += s;
 rep(b,0,N) rep(i,0,N) {
  if (a+i == b || s[a+i] < s[b+i]) {b += max(0, i-1); break
      ;}
  if (s[a+i] > s[b+i]) { a = b; break; }
 }
 return a;
}

int main() {
 string v; cin >> v;
 rotate(v.begin(), v.begin() + min_rotation(v), v.end());
 cout << v << '\n';
 return 0;
}
```

29 Mo's

```
const int sz = sqrt(Max) + 1;

struct query {
    int l, r, id;
    query() {}
    query(int l, int r, int id) : l(l), r(r), id(id) {}
    bool operator<(query &q) {
        if (l / sz != q.l / sz)
            return l < q.l;
        return r < q.r;
    }
};

void Add(int i) {
    // add this element
}

void Remove(int i) {
    // remove this element
}

void Build(vector<query> &q) {
    sort(q.begin(), q.end());
    int Left = 0, Right = -1;
    for (auto i : q) {
        int id = i.id;
        int l = i.l, r = i.r;
        while (Right < r)
            Add(++Right);
        while (Left > l)
            Add(--Left);
        while (Right > r)
            Remove(Right--);
        while (Left < l)
            Remove(Left++);
        // here you can answer the q's query
    }
}
```

30 Persistent Segment Tree

```
struct Treap {
 int val;

 Treap() { }
 Treap(int val) : val(val) { }

 Treap operator+(const Treap& a) const {
  Treap ans;
  ans.val = this->val + a.val;
  return ans;
 }
};

vector<int> Root;
int T = 0;
Treap seg[N * 100];
int LC[N * 100];
int RC[N * 100];

void build(int p, int L = 1, int R = n) {
 if (L == R) {
  seg[p].val = 0;
  return;
 }
 LC[p] = T++, RC[p] = T++;
 int mid = L + R >> 1;
 build(LC[p], L, mid);
 build(RC[p], mid + 1, R);
 seg[p] = seg[LC[p]] + seg[RC[p]];
}

int update(int i, int x, int p, int L = 1, int R = n) {
 if (i < L || i > R) return p;
 int newP = T++;
 seg[newP] = seg[p], LC[newP] = LC[p], RC[newP] = RC[p];
 if (L == R) {
  seg[newP].val = x;
  return newP;
 }

 int mid = L + R >> 1;
 LC[newP] = update(i, x, LC[newP], L, mid);
 RC[newP] = update(i, x, RC[newP], mid + 1, R);
 seg[newP] = seg[LC[newP]] + seg[RC[newP]];
 return newP;
}

Treap query(int i, int j, int p, int L = 1, int R = n) {
```

```
 if (i > R || j < L) return Treap(0);
 if (L >= i && R <= j) return seg[p];

 int mid = L + R >> 1;
 Treap c1 = query(i, j, LC[p], L, mid);
 Treap c2 = query(i, j, RC[p], mid + 1, R);
 return c1 + c2;
}

int main() {
 Root.push_back(T++);
 build(root[0]);
 for (int i = 1; i <= n; i++)
  Root.push_back(update(i, a[i], Root[i - 1]));

 return 0;
}
```

31 SegmentTree Persistent Lazy

```
template <typename type>
struct SegmentTreePersistentLazy {
    #define Mid ((L + R) >> 1)

    private : const type E = 0;  /// identity element
              /// Check this
    private : const type EL = 0; /// identity element for
        lazy /// Check this
    private : static const int Nax = 100 * 100100;
                 /// Check this

    private : int n;      /// Array length
    private : int Counter; /// Counter for nodes

    private : int Left[Nax];
    private : int Right[Nax];
    private : type Tree[Nax];
    private : type Lazy[Nax];
    private : vector <int> Roots;

    private : int NewNode() {
        Tree[Counter] = E;
        Lazy[Counter] = EL;
        Left[Counter] = -1;
        Right[Counter] = -1;
        return Counter++;
    }
private : int NewNode(int Copy) {
    Tree[Counter] = Tree[Copy];
    Lazy[Counter] = Lazy[Copy];
    Left[Counter] = Left[Copy];
    Right[Counter] = Right[Copy];
    return Counter++;
}

private : type Unite(type x, type y) { /// Check this
    return max(x, y);
}

private : void Merge(int Node) {
    Tree[Node] = Unite(Tree[Left[Node]], Tree[Right[Node
        ]]);
}

private : void PushLazy(int Node, int Len) {
    if(Lazy[Node] != EL) {
        Tree[Node] += Lazy[Node]; /// Check this
        if(Left[Node] != -1) {
            Left[Node] = NewNode(Left[Node]);
            Lazy[Left[Node]] += Lazy[Node]; /// Check this
        }
        if(Right[Node] != -1) {
            Right[Node] = NewNode(Right[Node]);
            Lazy[Right[Node]] += Lazy[Node]; /// Check
                this
        }
        Lazy[Node] = EL;
    }
}

public : int Build(const vector <type>& a) {
    Clear();
    n = a.size();
    Roots.push_back(NewNode());
    Build(a, Roots[0], 0, n - 1);
    return (int) Roots.size() - 1;
}

private : void Build(const vector <type>& a, int Node,
     int L, int R) {
    Tree[Node] = E, Lazy[Node] = EL;
    if(L == R) {
        return void(Tree[Node] = a[L]);
    }
    Build(a, Left[Node] = NewNode(), L, Mid);
    Build(a, Right[Node] = NewNode(), Mid + 1, R);
    Merge(Node);
}

public : int Update(int i, int j, type x) {
    if(i < 0) {
        i = 0;
    }
    if(j >= n) {
        j = n - 1;
    }
    Roots.push_back(i <= j ? Update(i, j, x, Roots.back()
        , 0, n - 1) : Roots.back());
    return (int) Roots.size() - 1;
}

private : int Update(int i, int j, type x, int Node, int
     L, int R) {
    PushLazy(Node, R - L + 1);
    if(i > j || L > R) {
        return Node;
    }
    if(j < L || R < i) {
        return Node;
    }
    Node = NewNode(Node);
    if(i <= L && R <= j) {
        return Lazy[Node] = x, PushLazy(Node, R - L + 1),
             Node;
    }
    Left[Node] = Update(i, j, x, Left[Node], L, Mid);
    Right[Node] = Update(i, j, x, Right[Node], Mid + 1, R
        );
    return Merge(Node), Node;
}

public : type Query(int Version, int L, int R) {
    if(L < 0)  L = 0;
    if(R >= n) R = n - 1;
    if(L > R)  return E;
    return Query(L, R, Roots[Version], 0, n - 1);
}

private : type Query(int i, int j, int Node, int L, int R
    ) {
    PushLazy(Node, R - L + 1);
    if(j < L || R < i)    return E;
    if(i <= L && R <= j)  return Tree[Node];
    return Unite(Query(i, j, Left[Node], L, Mid), Query(i
        , j, Right[Node], Mid + 1, R));
}
```

```
    public : void Clear() {
        n = 0;
        Counter = 0;
        Roots.clear();
    }

    #undef Mid
};
```

32 Suffix Array

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

namespace SuffixArray {
   vector<int> build(vector<int>& s, int alphabet_size=256)
        {
  s.push_back(0);
  int n = s.size();
  int m = max(n, alphabet_size); // First iteration hash
       range
  vector<int> p(n), c(n), pn(n), cn(n), cnt(m);

  for(int i=0 ; i<n ; i++) cnt[s[i]]++;
  for(int i=1 ; i<m ; i++) cnt[i]+=cnt[i-1];
  for(int i=0 ; i<n ; i++) p[--cnt[s[i]]] = i;

  c[p[0]]=0; m=1;
  for(int i=1 ; i<n ; i++)
   if(s[p[i]] != s[p[i-1]]) c[p[i]] = m++;
   else c[p[i]] = m-1;

  for(int j=0 ; 1<<j < n ; j++){
   fill(cnt.begin(), cnt.begin()+m, 0);
   for(int i=0 ; i<n ; i++) pn[i]=p[i]-(1<<j)+(1<<j>p[i]?n
        :0);
   for(int i=0 ; i<n ; i++) cnt[c[pn[i]]]++;
   for(int i=1 ; i<m ; i++) cnt[i]+=cnt[i-1];
   for(int i=n ; i-- ;  ) p[--cnt[c[pn[i]]]] = pn[i];
   cn[p[0]] = 0; m = 1;
   for(int i=1 ; i<n ; i++)
    if(c[p[i]]!=c[p[i-1]] || c[(p[i]+(1<<j))%n]!=c[(p[i
         -1]+(1<<j))%n]) cn[p[i]]=m++;
    else cn[p[i]] = m-1;
   swap(c, cn);
  }
  p.erase(p.begin());
  s.pop_back();
  return p;
 }

 vector<int> kasaiLcp(vector<int>& str, vector<int>& suf){
  int n = suf.size();
  vector<int> rnk(n), lcp(n);
  int k=0;
  for(int i=0 ; i<n ; i++) rnk[suf[i]]=i;
  for(int i=0 ; i<n ; i++)
   if(rnk[i]==n-1) k=0;
   else {
    int j = suf[rnk[i]+1];
    while( i+k<n && j+k<n && str[i+k]==str[j+k] ) k++;
    lcp[rnk[i]]=k;
    if(k) k--;
   }
  return lcp;
 }
}

int main() {
 string s = "banana";
 vector<int> v(s.begin(), s.end());
 auto suf = SuffixArray::build(v);
 auto lcp = SuffixArray::kasaiLcp(v, suf);
 // ...
}
```

33 Suffix Automaton

```
 const  int N = 5e5 + 100;
const int Alpha = 28;

struct State {
    int Link, Len;
    int FirstPos;
    int Next[Alpha];
    bool isClone;

    vector<int> invLink;

    State() {
        Link = Len = 0;
        isClone = 0;
        FirstPos = -1;
        memset(Next, -1, sizeof Next);
    }
};

State st[2 * N];
bool Terminal[2 * N];

class SuffixAutomaton {
public:
    int Sz, Last;
    vector<State> st;
    vector<bool> Terminal;

    void init(int n) {
        st.clear();      st.resize(2 * n);
        Terminal.clear(); Terminal.resize(2 * n);

        st[0].Len = 0;
        st[0].Link = -1;

        Sz = 1; Last = 0;
    }

    int First_Occurrence(string &s, int u) {
        return st[u].FirstPos - s.size() + 1;
    }

    void Add(char ch) {
        int c = ch - 'a';
        int Cur = Sz++;

        st[Cur].Len = st[Last].Len + 1;
        st[Cur].FirstPos = st[Cur].Len - 1;

        int p = Last;
        while(p != -1 && st[p].Next[c] == -1) {
            st[p].Next[c] = Cur;
            p = st[p].Link;
        }

        if (p == -1){ st[Cur].Link = 0; }
        else {
            int q = st[p].Next[c];
            if (st[p].Len + 1 == st[q].Len) {
                st[Cur].Link = q;
            } else {
                int Clone = Sz++;
                st[Clone].Len = st[p].Len + 1;
                st[Clone].Link = st[q].Link;
                st[Clone].FirstPos = st[q].FirstPos;
                st[Clone].isClone = true;

                for (int i = 0; i < Alpha; i++)
```

```
                st[Clone].Next[i] = st[q].Next[i];

            while(p != -1 && st[p].Next[c] == q) {
                st[p].Next[c] = Clone;
                p = st[p].Link;
            }

            st[q].Link = st[Cur].Link = Clone;
        }
    }
    Last = Cur;
}

void GetTerminal() {
    int p = Last;
    while(p > 0) {
        Terminal[p] = 1;
        p = st[p].Link;
    }
}

void Build(string &s) {
    init(s.size());
    for (auto i : s) Add(i);
    GetTerminal();
}
} SA;

int main() {
    // after constructing the automaton
    for (int v = 1; v < SA.Sz; v++)
        st[st[v].Link].invLink.push_back(v);

    return 0;
}

// Functions for Suffix Automaton

// Get Node of substring
int Get_Node(string &s, int idx, int u) {
    if (idx == s.size())
        return u;

    if (st[u].Next[s[idx] - 'a'] == -1)
        return -1;

    return Get_Node(s, idx + 1, st[u].Next[s[idx] - 'a']);
}
// Get First Occurrenece of substring
int First_Occurrence(int u, int P_length) {
    return st[u].FirstPos - P_length + 1;
}

// All occurrence positions of substring
void Get_All_Occurrence(int u, int P_length, vector<int>&
     Positions) {
    if (!st[u].isClone)
        Positions.push_back(First_Occurrence(u, P_length));

    for (auto i : st[u].invLink)
        Get_All_Occurrence(i, P_length, Positions);
}
```

34 Sum of Kth Powers

```
// sum of 1^k + 2^k + .. + n^k
// O(k * log(k)) using Lagrange
// f(x) = Sum(Yi*Mul((x-Xj)/(Xi-Xj)))
// 1 <= i, j <= n, i != j
int SumOfKthPowers(int n, int k) {
    int m = k + 1;
    if (n <= m) {
        int Ans = 0;
        for (int i = 1; i <= n; i++)
            add_self(Ans, fp(i, k));
        return cout << Ans << endl, 0;
    }
    vector<int> Y;
    for (int x = 0, Sum = 0; x <= m; x++, add_self(Sum, fp(x,
         k)))
        Y.push_back(Sum);
    int Num = 1, Den = 1;
    for (int x = 0; x <= m; x++)
        mul_self(Num, sub(n, x));
    for (int x = 1; x <= m; x++)
        mul_self(Den, sub(0, x));
    int Ans = 0;
    for (int x = 0; x <= m; x++) {
        int Cur = mul(Num, inv(sub(n, x)));
        add_self(Ans, mul(Y[x], mul(Cur, inv(Den))));
        if (x < m) {
            mul_self(Den, inv(sub(x, m)));
            mul_self(Den, add(x, 1));
        }
    }
    cout << Ans << endl;
}
```

35 Treap

```
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().
     count());

struct Treap {
 int val;
 int pr, sz; // priority, size
 Treap *l, *r;
 Treap(int val = 0):
  val(val), pr(rnd()), sz(1), l(NULL), r(NULL){}
};

inline int get_size(Treap* p) {
 return p ? p->sz : 0;
}

void update_treap(Treap* p) {
 if (!p) return;
 p->sz = get_size(p->l) + get_size(p->r) + 1;
}

Treap* merge(Treap* a, Treap* b) {
 if (!a || !b)
  return a ? a : b;

 if (a->pr < b->pr) {
  // push_lazy(a)
  a->r = merge(a->r, b);
  update_treap(a); // update node
  return a;
 }
 else {
  // push_lazy(b)
  b->l = merge(a, b->l);
  update_treap(b);
  return b;
 }
}

// first : [L, idx] | second : ]idx, R]
// split(root,idx) => [0 ... idx],[idx+1... N-1]
pair<Treap*, Treap*> split(Treap* p, int idx) {
 if (!p)  return {NULL, NULL};
 if (!idx) return {NULL, p};
```

```
 // push_lazy(p)
 int cur = get_size(p->l) + 1;
 if (cur <= idx) { // p & p->l goes in first
  Treap *a = p, *b;
  tie(a->r, b) = split(p->r, idx - cur);
  update_treap(a);
  return {a, b};
 }
 else {
  Treap *a, *b = p;
  tie(a, b->l) = split(p->l, idx);
  update_treap(b);
  return {a, b};
 }
}

tuple<Treap*, Treap*, Treap*> split(Treap *root, int l, int
     r) {
 Treap *L, *R, *mid, *_mid;
 tie(_mid, R) = split(root, r);
 tie(L, mid) = split(_mid, l - 1);
 return {L, mid, R};
}

Treap* merge(Treap* L, Treap* mid, Treap* R) {
 mid = merge(L, mid);
 mid = merge(mid, R);
 return mid;
}
```

36 WaveletTree

```
#include <bits/stdc++.h>

using namespace std;

struct WaveletTree {
 #define T int
 typedef typename vector<T>::iterator iter;

 T low, high;
 vector <T> B;
 WaveletTree *L, *R;

 WaveletTree(iter left, iter right, int low, int high) {
  this->low = low;
  this->high = high;
  if (low == high || left >= right)
   return;

  T mid = low + (high - low) / 2;

  B.push_back(0); // 1-base
  for (iter i = left; i != right; ++i)
   B.push_back(B.back() + (*i <= mid));

  iter pivot = stable_partition(left, right, [mid](int x){
       return x <= mid; });
  L = new WaveletTree(left, pivot, low, mid);
  R = new WaveletTree(pivot, right, mid + 1, high);
 }

 // kth smallest element in [i, j]
 T kth(int i, int j, int k) {
  if (i > j) return 0;
  if (low == high) return low;

  int inLeft = B[j] - B[i - 1];
  if (k <= inLeft)
   return L->kth(B[i - 1] + 1, B[j], k);
  else
   return R->kth(i - B[i - 1], j - B[j], k - inLeft);
 }

 // count of elements in [i, j] equal to k
 int count(int i, int j, T k) {
  if (i > j || k < low || k > high)
   return 0;
  if (low == high)
   return j - i + 1;

  T mid = low + (high - low) / 2;
  if (k <= mid)
   return L->count(B[i - 1] + 1, B[j], k);
  else return R->count(i - B[i - 1], j - B[j], k);
 }

 // count of elements in [i, j] Less Than or Equal to k
 int LTE(int i, int j, T k) {
  if (i > j || k < low) return 0;
  if (high <= k) return j - i + 1;
  return L->LTE(B[i - 1] + 1, B[j], k) + R->LTE(i - B[i -
       1], j - B[j], k);
 }
};
```

Theoretical Computer Science Cheat Sheet

Definitions

$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)\ \forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0\ \forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim\limits_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\left[{n \atop k}\right]$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\left\{{n \atop k}\right\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\left\langle{n \atop k}\right\rangle$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\ldots\pi_n$ on $\{1, 2, \ldots, n\}$ with k ascents.
$\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \qquad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n}\frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \qquad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\displaystyle\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\displaystyle\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\displaystyle\binom{n}{k} = \binom{n}{n-k}$,

4. $\displaystyle\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\displaystyle\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\displaystyle\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\displaystyle\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\displaystyle\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\displaystyle\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\displaystyle\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\displaystyle\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$,

12. $\displaystyle\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, **13.** $\displaystyle\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$,

14. $\displaystyle\left[{n \atop 1}\right] = (n-1)!$, **15.** $\displaystyle\left[{n \atop 2}\right] = (n-1)!H_{n-1}$, **16.** $\displaystyle\left[{n \atop n}\right] = 1$, **17.** $\displaystyle\left[{n \atop k}\right] \ge \left\{{n \atop k}\right\}$,

18. $\displaystyle\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, **19.** $\displaystyle\left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$, **20.** $\displaystyle\sum_{k=0}^{n}\left[{n \atop k}\right] = n!$, **21.** $\displaystyle C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\displaystyle\left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1$, **23.** $\displaystyle\left\langle{n \atop k}\right\rangle = \left\langle{n \atop n-1-k}\right\rangle$, **24.** $\displaystyle\left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$,

25. $\displaystyle\left\langle{0 \atop k}\right\rangle = \begin{cases}1 & \text{if } k = 0,\\ 0 & \text{otherwise}\end{cases}$ **26.** $\displaystyle\left\langle{n \atop 1}\right\rangle = 2^n - n - 1$, **27.** $\displaystyle\left\langle{n \atop 2}\right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $\displaystyle x^n = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{x+k}{n}$, **29.** $\displaystyle\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $\displaystyle m!\left\{{n \atop m}\right\} = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{k}{n-m}$,

31. $\displaystyle\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{n}\left\{{n \atop k}\right\}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\displaystyle\left\langle\!\left\langle{n \atop 0}\right\rangle\!\right\rangle = 1$, **33.** $\displaystyle\left\langle\!\left\langle{n \atop n}\right\rangle\!\right\rangle = 0 \quad \text{for } n \ne 0$,

34. $\displaystyle\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle{n-1 \atop k}\right\rangle\!\right\rangle + (2n-1-k)\left\langle\!\left\langle{n-1 \atop k-1}\right\rangle\!\right\rangle$, **35.** $\displaystyle\sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\displaystyle\left\{{x \atop x-n}\right\} = \sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\displaystyle\left\{{n+1 \atop m+1}\right\} = \sum_{k}\binom{n}{k}\left\{{k \atop m}\right\} = \sum_{k=0}^{n}\left\{{k \atop m}\right\}(m+1)^{n-k}$,