Contents

1 Basic

1.1 vimrc

```
colo ron
syn on
se ai ar nu rnu
se mouse=a bs=2 ts=4 sw=4 ttm=100
au BufNewFile *.cpp 0r ~/default.cpp | :1,$-7 fo
filetype indent on
```

1.2 IncreaseStackSize

```
//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);
    }
  }
}
```

1.3 Default Code

```
// #pragma GCC optimize ("-O2")
// #pragma GCC optimize ("unroll-loops")
#include<bits/stdc++.h>
using namespace std;
#define F first
#define S second
#define pb push_back
#define MP make_pair
#define ALL(x) begin(x),end(x)
#define SZ(x) ((int)(x).size())
typedef long long LL;
typedef double DB;
typedef long double LDB;
typedef pair<int, int> PII;
typedef pair<LL, LL> PLL;
// #define IOS ios_base::sync_with_stdio(0); cin.tie(0)
const int MXN = (int)1e6 + 7;
int main(){
  return 0;
}
```

1.4 Random

```
#include <random>
mt19937 rng(0x5EED);
// mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
int randint(int lb, int ub){
  return uniform_int_distribution<int>(lb, ub)(rng);
}
```

1.5 Intput Opt

```
const int bsz = 1048576;
inline int rc(){ //readchar
  static char buf[bsz];
```

```
  static char *ptr = buf, *end = buf;
  if(ptr == end){
    if((end = buf + fread(buf,1,sz,stdin)) == buf)
        return EOF;
    ptr = buf;
  }
  return *ptr++;
}
inline int ri(int &x) { //readint
    static char c, neg;
    while((c = rc()) < '-') if(c == EOF) return 0;
    neg = (c == '-') ? -1 : 1;
    x = (neg == 1) ? c-'0' : 0;
    while((c = rc()) >= '0')
        x = (x << 3) + (x << 1) + c-'0';
    x *= neg;
    return 1;
}
```

2 Data Structure

2.1 extc_heap

```
#include <bits/extc++.h>
typedef __gnu_pbds::priority_queue<int> heap_t;
heap_t a,b;

int main() {
  a.clear();
  b.clear();
  a.push(1);
  a.push(3);
  b.push(2);
  b.push(4);
  assert(a.top() == 3);
  assert(b.top() == 4);
  // merge two heap
  a.join(b);
  assert(a.top() == 4);
  assert(b.empty());

  return 0;
}
```

2.2 extc_balance_tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> set_t;
typedef cc_hash_table<int,int> umap_t;

int main(){
  // Insert some entries into s.
  set_t s;
  s.insert(12);
  s.insert(505);

  // The order of the keys should be: 12, 505.
  assert(*s.find_by_order(0) == 12);
  assert(s.find_by_order(2) == end(s));

  // The order of the keys should be: 12, 505.
  assert(s.order_of_key(12) == 0);
  assert(s.order_of_key(505) == 1);

  // Erase an entry.
  s.erase(12);

  // The order of the keys should be: 505.
  assert(*s.find_by_order(0) == 505);

  // The order of the keys should be: 505.
  assert(s.order_of_key(505) == 0);
}
```

2.3 Disjoint Set

```
struct DisjointSet {
  // save() is like recursive
  // undo() is like return
  int n, fa[MXN], sz[MXN];
  vector<pair<int*,int>> h;
  vector<int> sp;
  void init(int tn) {
    n=tn;
    for (int i=0; i<n; i++) {
      fa[i]=i;
      sz[i]=1;
    }
    sp.clear(); h.clear();
  }
  void assign(int *k, int v) {
    h.PB({k, *k});
    *k=v;
  }
  void save() { sp.PB(SZ(h)); }
  void undo() {
    assert(!sp.empty());
    int last=sp.back(); sp.pop_back();
    while (SZ(h)!=last) {
      auto x=h.back(); h.pop_back();
      *x.F=x.S;
    }
  }
  int f(int x) {
    while (fa[x]!=x) x=fa[x];
    return x;
  }
  void uni(int x, int y) {
    x=f(x); y=f(y);
    if (x==y) return ;
    if (sz[x]<sz[y]) swap(x, y);
    assign(&sz[x], sz[x]+sz[y]);
    assign(&fa[y], x);
  }
}djs;
```

2.4 DLX

```
int a[201][201];
struct DLX {
  int L[MXN], R[MXN], U[MXN], D[MXN];
  int rr[MXN], cc[MXN], S[MXN];
  int re[MXN], bst[MXN], ans;
  int n, m, cntp;
  void init() {
    for (int i = 0; i <= m; i++) {
      L[i] = i - 1; R[i] = i + 1;
      U[i] = D[i] = i;
      S[i] = 0;
    }
    L[0] = m; R[m] = 0;
    cntp = m + 1;
    for (int i = 1; i <= n; i++) {
      int f = -1;
      for (int j = 1; j <= m; j++) {
        if (!a[i][j]) continue ;
        if (f == -1) f = cntp;
        L[cntp] = cntp - 1; R[cntp] = cntp + 1;
        U[cntp] = U[j]; D[U[j]] = cntp;
        D[cntp] = j; U[j] = cntp;
        rr[cntp] = i; cc[cntp] = j;
        S[j]++;
        cntp++;
      }
```

```
      if (f != -1) {
        L[f] = cntp - 1;
        R[cntp-1] = f;
      }
    }
  }
  void cover(int c) {
    L[R[c]] = L[c];
    R[L[c]] = R[c];
    for (int i = D[c]; i != c; i = D[i]) {
      for (int j = R[i]; j != i; j = R[j]) {
        S[cc[j]]--;
        D[U[j]] = D[j]; U[D[j]] = U[j];
      }
    }
  }
  void uncover(int c) {
    for (int i = U[c]; i != c; i = U[i]) {
      for (int j = L[i]; j != i; j = L[j]) {
        S[cc[j]]++;
        D[U[j]] = j; U[D[j]] = j;
      }
    }
    R[L[c]] = c; L[R[c]] = c;
  }
  void dfs(int dep) {
    if (dep > ans) return ;
    if (R[0] == 0) {
      ans = min(ans, dep);
      return ;
    }
    int c = R[0];
    for (int i = R[0]; i != 0; i = R[i]) {
      if (S[i] < S[c]) c = i;
    }
    cover(c);
    for (int i = D[c]; i != c; i = D[i]) {
      re[dep] = rr[i];
      for (int j = R[i]; j != i; j = R[j]) {
        cover(cc[j]);
      }
      dfs(dep+1);
      for (int j = L[i]; j != i; j = L[j]) {
        uncover(cc[j]);
      }
    }
    uncover(c);
    return ;
  }
  int solve(int _n, int _m) {
    n = _n, m = _m;
    init(); ans = n + 1;
    dfs(0);
    if (ans == n + 1) return -1;
    return ans;
  }
} dlx;
```

2.5 Treap

```
const int MEM = 560004;
struct Treap {
  static Treap mem[MEM], *pmem;
  Treap *l, *r;
  LL val, mxn, sum, d;
  int size, pri;
  Treap () : l(NULL), r(NULL), size(0) {}
  Treap (LL _val) :
    l(NULL), r(NULL), val(_val), mxn(_val), sum(_val),
        d(0), size(1), pri(rand()){}
} Treap::mem[MEM], *Treap::pmem = Treap::mem;

int size(const Treap *t) {
    return t ? t->size : 0;
}
LL _mxn(Treap *t) {
    return t ? t->mxn : -INF;
}
LL _sum(Treap *t) {
    return t ? t->sum : 0;
}
void pull(Treap *t) {
  if (!t) return;
  t->size = size(t->l) + size(t->r) + 1;
  t->mxn = max(t->val, max(_mxn(t->l), _mxn(t->r)));
  t->sum = t->val + _sum(t->l) + _sum(t->r);
}
void pushdown(Treap *t) {
    if (!t) return ;
    if (t->l) {
        t->l->mxn += t->d;
        t->l->val += t->d;
        t->l->sum += size(t->l)*(t->d);
        t->l->d += t->d;
    }
    if (t->r) {
        t->r->mxn += t->d;
        t->r->val += t->d;
        t->l->sum += size(t->r)*(t->d);
        t->r->d += t->d;
    }
    t->d = 0;
}
Treap *merge(Treap *a, Treap *b) {
  if(!a || !b) return a ? a : b;
  if (a->pri > b->pri) {
    pushdown(a);
    a->r = merge(a->r, b);
    pull(a);
    return a;
  } else {
    pushdown(b);
    b->l = merge(a, b->l);
    pull(b);
    return b;
  }
}
void split_val(Treap *t, int k, Treap *&a, Treap *&b) {
  pushdown(t);
  if(!t) a = b = NULL;
  else if(t->val <= k){
    a = t;
    split_val(t->r, k, a->r, b);
    pull(a);
  } else{
    b = t;
    split_val(t->l, k, a, b->l);
    pull(b);
  }
}
void split_size(Treap *t, int k, Treap *&a, Treap *&b)
    {
  pushdown(t);
  if(!t) a = b = NULL;
  else if(size(t->l) + 1 <= k){
    a = t;
    split_size(t->r, k - size(t->l) - 1, a->r, b);
    pull(a);
  } else{
    b = t;
    split_size(t->l, k, a, b->l);
    pull(b);
  }
}
int main() {
  Treap *l = new (Treap::pmem++) Treap(5);
  return 0;
}
```

2.6 Persistent Treap

```
const int MEM = 16000004;
```

```
struct Treap {
  static Treap nil, mem[MEM], *pmem;
  Treap *l, *r;
  char val;
  int size;
  Treap () : l(&nil), r(&nil), size(0) {}
  Treap (char _val) :
    l(&nil), r(&nil), val(_val), size(1) {}
} Treap::nil, Treap::mem[MEM], *Treap::pmem = Treap::
    mem;

int size(const Treap *t) { return t->size; }
void pull(Treap *t) {
  if (!size(t)) return;
  t->size = size(t->l) + size(t->r) + 1;
}
Treap* merge(Treap *a, Treap *b) {
  if (!size(a)) return b;
  if (!size(b)) return a;
  Treap *t;
  if (rand() % (size(a) + size(b)) < size(a)) {
    t = new (Treap::pmem++) Treap(*a);
    t->r = merge(a->r, b);
  } else {
    t = new (Treap::pmem++) Treap(*b);
    t->l = merge(a, b->l);
  }
  pull(t);
  return t;
}
void split(Treap *t, int k, Treap *&a, Treap *&b) {
  if (!size(t)) a = b = &Treap::nil;
  else if (size(t->l) + 1 <= k) {
    a = new (Treap::pmem++) Treap(*t);
    split(t->r, k - size(t->l) - 1, a->r, b);
    pull(a);
  } else {
    b = new (Treap::pmem++) Treap(*t);
    split(t->l, k, a, b->l);
    pull(b);
  }
}

int nv;
Treap *rt[50005];

void print(const Treap *t) {
  if (!size(t)) return;
  print(t->l);
  cout << t->val;
  print(t->r);
}

int main(int argc, char** argv) {
  IOS;
  rt[nv=0] = &Treap::nil;
  Treap::pmem = Treap::mem;
  int Q, cmd, p, c, v;
  string s;
  cin >> Q;
  while (Q--) {
    cin >> cmd;
    if (cmd == 1) {
      // insert string s after position p
      cin >> p >> s;
      Treap *tl, *tr;
      split(rt[nv], p, tl, tr);
      for (int i=0; i<SZ(s); i++)
        tl = merge(tl, new (Treap::pmem++) Treap(s[i]))
            ;
      rt[++nv] = merge(tl, tr);
    } else if (cmd == 2) {
      // remove c characters starting at position
      Treap *tl, *tm, *tr;
      cin >> p >> c;
      split(rt[nv], p-1, tl, tm);
      split(tm, c, tm, tr);
      rt[++nv] = merge(tl, tr);
    } else if (cmd == 3) {
      // print c characters starting at position p, in
          version v
      Treap *tl, *tm, *tr;
      cin >> v >> p >> c;
      split(rt[v], p-1, tl, tm);
      split(tm, c, tm, tr);
      print(tm);
      cout << "\n";
    }
  }
  return 0;
}
```

2.7 Li Chao Segment Tree

```
struct LiChao_min{
  tructt line{
    LL m, c;
    line(LL _m=0, LL _c=0) { m = _m; c = _c; }
    LL eval(LL x) { return m * x + c; }
  };
  struct node{
    node *l, *r; line f;
    node(line v) { f = v; l = r = NULL; }
  };
  typedef node* pnode;
  pnode root; int sz;
#define mid ((l+r)>>1)
  void insert(line &v, int l, int r, pnode &nd){
    if(!nd) { nd = new node(v); return; }
    LL trl = nd->f.eval(l), trr = nd->f.eval(r);
    LL vl = v.eval(l), vr = v.eval(r);
    if(trl <= vl && trr <= vr) return;
    if(trl > vl && trr > vr) { nd->f = v; return; }
    if(trl > vl) swap(nd->f, v);
    if(nd->f.eval(mid) < v.eval(mid)) insert(v, mid +
        1, r, nd->r);
    else swap(nd->f, v), insert(v, l, mid, nd->l);
  }
  LL query(int x, int l, int r, pnode &nd){
    if(!nd) return LLONG_MAX;
    if(l == r) return nd->f.eval(x);
    if(mid >= x) return min(nd->f.eval(x), query(x, l,
        mid, nd->l));
    return min(nd->f.eval(x), query(x, mid + 1, r, nd->
        r));
  }
  /* -sz <= query_x <= sz */
  void init(int _sz){ sz = _sz + 1; root = NULL; }
  void add_line(LL m, LL c){ line v(m, c); insert(v, -
      sz, sz, root); }
  LL query(LL x) { return query(x, -sz, sz, root); }
};
```

2.8 HilbertCurve

```
long long hilbert(int n, int x, int y) {
    long long res = 0;
    for (int s = n / 2; s; s >>= 1) {
        int rx = (x & s) > 0;
        int ry = (y & s) > 0;
        res += s * 1ll * s * ((3 * rx) ^ ry);
        if (ry == 0) {
            if (rx == 1) {
                x = s - 1 - x;
                y = s - 1 - y;
            }
            swap(x, y);
        }
    }
    return res;
}
```

3 Graph

3.1 CentroidDecomposition

```
void go(int u, int dpt, int d) {
  ck[u] = true;
  cdis[dpt][u] = d;
  for (int x : edge[u]) {
    if (!ck[x]) {
      go(x, dpt, d + 1);
    }
  }
}
void dfs(int u, int dpt) {
  st.clear();
  dfssz(u);
  int bst = -1;
  int k = (int)st.size();
  for (int v : st) {
    if (max(mxn[v], k - siz[v]) * 2 <= k) bst = v;
    ck[v] = false;
  }
  go(bst, dpt, 0);
  cdep[bst] = dpt;
  for (int v : st) {
    ck[v] = false;
    cprt[dpt][v] = bst;
    if (v > m) continue ;
    dd[bst].push_back(cdis[dpt][v]);
    if (dpt) re_dd[bst].push_back(cdis[dpt-1][v]);
  }
  sort(dd[bst].begin(), dd[bst].end());
  sort(re_dd[bst].begin(), re_dd[bst].end());
  ck[bst] = true;
  for (int v : edge[bst]) {
    if (!ck[v]) {
      dfs(v, dpt + 1);
    }
  }
}
int qy(int v, int d) {
  int res = upper_bound(dd[v].begin(), dd[v].end(), d)
      - dd[v].begin();
  for (int i = cdep[v]; i >= 1; i--) {
    int pa = cprt[i-1][v], u = cprt[i][v];
    res += upper_bound(dd[pa].begin(), dd[pa].end(), d
        - cdis[i-1][v]) - dd[pa].begin();
    res -= upper_bound(re_dd[u].begin(), re_dd[u].end()
        , d - cdis[i-1][v]) - re_dd[u].begin();
  }
  return res;
}
```

3.2 BCC Edge

```
struct BccEdge {
  static const int MXN = 100005;
  struct Edge { int v,eid; };
  int n,m,step,par[MXN],dfn[MXN],low[MXN];
  vector<Edge> E[MXN];
  DisjointSet djs;
  void init(int _n) {
    n = _n; m = 0;
    for (int i=0; i<n; i++) E[i].clear();
    djs.init(n);
  }
  void add_edge(int u, int v) {
    E[u].PB({v, m});
    E[v].PB({u, m});
    m++;
  }
  void DFS(int u, int f, int f_eid) {
    par[u] = f;
    dfn[u] = low[u] = step++;
    for (auto it:E[u]) {
      if (it.eid == f_eid) continue;
      int v = it.v;
      if (dfn[v] == -1) {
        DFS(v, u, it.eid);
        low[u] = min(low[u], low[v]);
      } else {
        low[u] = min(low[u], dfn[v]);
      }
    }
  }
  void solve() {
    step = 0;
    memset(dfn, -1, sizeof(int)*n);
    for (int i=0; i<n; i++) {
      if (dfn[i] == -1) DFS(i, i, -1);
    }
    djs.init(n);
    for (int i=0; i<n; i++) {
      if (low[i] < dfn[i]) djs.uni(i, par[i]);
    }
  }
}graph;
```

3.3 BCC Vertex

```
struct BccVertex{
  int n, nBcc, cntp, root, dfn[MXN], low[MXN];
  vector<int> E[MXN];
  vector<int> bcc[MXN];
  int top;
  int stk[MXN];
  bool is_cut[MXN];
  void init(int _n){
    n = _n;
    nBcc = cntp = 0;
    for(int i = 1; i <= n; ++i) E[i].clear();
  }
  void add_edge(int u, int v){
    E[u].pb(v);
    E[v].pb(u);
  }
  void dfs(int u, int pa){
    dfn[u] = low[u] = cntp++;
    stk[top++] = u;
    int son = 0;
    for(auto v : E[u]){
      if(v == pa) continue;
      if(dfn[v] == -1){
        son++;
        dfs(v, u);
        low[u] = min(low[u], low[v]);
        if(low[v] >= dfn[u]){
          is_cut[u] = 1;
          bcc[nBcc].clear();
          do{
            bcc[nBcc].pb(stk[--top]);
          } while(stk[top] != v);
          bcc[nBcc++].pb(u);
        }
      } else{
        low[u] = min(low[u], dfn[v]);
      }
    }
    if(u == root && son < 2) is_cut[u] = 0;
  }
  vector<vector<int>> solve(){
    vector<vector<int>> res;
    for(int i = 1; i <= n; ++i){
      dfn[i] = low[i] = -1;
      is_cut[i] = 0;
    }
    for(int i = 1; i <= n; ++i){
      if(dfn[i] == -1){
        top = 0;
        root = i;
```

```
        dfs(i, i);
      }
    }
    for(int i = 0; i < nBcc; ++i){
      res.pb(bcc[i]);
    }
    return res;
  }
}graph;
```

3.4 Maximum Clique

```
class MaxClique {
public:
    static const int MV = 210;

    int V;
    int el[MV][MV/30+1];
    int dp[MV];
    int ans;
    int s[MV][MV/30+1];
    vector<int> sol;

    void init(int v) {
        V = v; ans = 0;
        FZ(el); FZ(dp);
    }

    /* Zero Base */
    void addEdge(int u, int v) {
        if(u > v) swap(u, v);
        if(u == v) return;
        el[u][v/32] |= (1<<(v%32));
    }

    bool dfs(int v, int k) {
        int c = 0, d = 0;
        for(int i=0; i<(V+31)/32; i++) {
            s[k][i] = el[v][i];
            if(k != 1) s[k][i] &= s[k-1][i];
            c += __builtin_popcount(s[k][i]);
        }
        if(c == 0) {
            if(k > ans) {
                ans = k;
                sol.clear();
                sol.push_back(v);
                return 1;
            }
            return 0;
        }
        for(int i=0; i<(V+31)/32; i++) {
            for(int a = s[k][i]; a ; d++) {
                if(k + (c-d) <= ans) return 0;
                int lb = a&(-a), lg = 0;
                a ^= lb;
                while(lb!=1) {
                    lb = (unsigned int)(lb) >> 1;
                    lg ++;
                }
                int u = i*32 + lg;
                if(k + dp[u] <= ans) return 0;
                if(dfs(u, k+1)) {
                    sol.push_back(v);
                    return 1;
                }
            }
        }
        return 0;
    }

    int solve() {
        for(int i=V-1; i>=0; i--) {
            dfs(i, 1);
            dp[i] = ans;
        }
        return ans;
    }
};
```

3.5 MinimumMeanCycle

```
/* minimum mean cycle */
const int MAXE = 1805;
const int MAXN = 35;
const double inf = 1029384756;
const double eps = 1e-6;
struct Edge {
  int v,u;
  double c;
};
int n,m,prv[MAXN][MAXN], prve[MAXN][MAXN], vst[MAXN];
Edge e[MAXE];
vector<int> edgeID, cycle, rho;
double d[MAXN][MAXN];
inline void bellman_ford() {
  for(int i=0; i<n; i++) d[0][i]=0;
  for(int i=0; i<n; i++) {
    fill(d[i+1], d[i+1]+n, inf);
    for(int j=0; j<m; j++) {
      int v = e[j].v, u = e[j].u;
      if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
        d[i+1][u] = d[i][v]+e[j].c;
        prv[i+1][u] = v;
        prve[i+1][u] = j;
      }
    }
  }
}
double karp_mmc() {
  // returns inf if no cycle, mmc otherwise
  double mmc=inf;
  int st = -1;
  bellman_ford();
  for(int i=0; i<n; i++) {
    double avg=-inf;
    for(int k=0; k<n; k++) {
      if(d[n][i]<inf-eps) avg=max(avg,(d[n][i]-d[k][i])
          /(n-k));
      else avg=max(avg,inf);
    }
    if (avg < mmc) tie(mmc, st) = tie(avg, i);
  }
  for(int i=0; i<n; i++) vst[i] = 0;
  edgeID.clear(); cycle.clear(); rho.clear();
  for (int i=n; !vst[st]; st=prv[i--][st]) {
    vst[st]++;
    edgeID.PB(prve[i][st]);
    rho.PB(st);
  }
  while (vst[st] != 2) {
    int v = rho.back(); rho.pop_back();
    cycle.PB(v);
    vst[v]++;
  }
  reverse(ALL(edgeID));
  edgeID.resize(SZ(cycle));
  return mmc;
}
```

3.6 Dynamic MST

```
int cnt[maxn], cost[maxn], st[maxn], ed[maxn];
pair<int, int> qr[maxn];
// Dynamic MST O( Q lg^2 Q )
// qr[i].first = id of edge to be changed, qr[i].second
     = weight after operation
// cnt[i] = number of operation on edge i
// call solve(0, q - 1, v, 0), where v contains edges i
     such that cnt[i] == 0
```

```
void contract(int l, int r, vector<int> v, vector<int>
    &x, vector<int> &y) {
    sort(v.begin(), v.end(), [&](int i, int j) {
        if (cost[i] == cost[j]) return i < j;
        return cost[i] < cost[j];
    });
    djs.save();
    for (int i = l; i <= r; ++i) djs.merge(st[qr[i].
        first], ed[qr[i].first]);
    for (int i = 0; i < (int)v.size(); ++i) {
        if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
            x.push_back(v[i]);
            djs.merge(st[v[i]], ed[v[i]]);
        }
    }
    djs.undo();
    djs.save();
    for (int i = 0; i < (int)x.size(); ++i) djs.merge(
        st[x[i]], ed[x[i]]);
    for (int i = 0; i < (int)v.size(); ++i) {
        if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
            y.push_back(v[i]);
            djs.merge(st[v[i]], ed[v[i]]);
        }
    }
    djs.undo();
}

void solve(int l, int r, vector<int> v, long long c) {
    if (l == r) {
        cost[qr[l].first] = qr[l].second;
        if (st[qr[l].first] == ed[qr[l].first]) {
            printf("%lld\n", c);
            return;
        }
        int minv = qr[l].second;
        for (int i = 0; i < (int)v.size(); ++i) minv =
            min(minv, cost[v[i]]);
        printf("%lld\n", c + minv);
        return;
    }
    int m = (l + r) >> 1;
    vector<int> lv = v, rv = v;
    vector<int> x, y;
    for (int i = m + 1; i <= r; ++i) {
        cnt[qr[i].first]--;
        if (cnt[qr[i].first] == 0) lv.push_back(qr[i].
            first);
    }
    contract(l, m, lv, x, y);
    long long lc = c, rc = c;
    djs.save();
    for (int i = 0; i < (int)x.size(); ++i) {
        lc += cost[x[i]];
        djs.merge(st[x[i]], ed[x[i]]);
    }
    solve(l, m, y, lc);
    djs.undo();
    x.clear(), y.clear();
    for (int i = m + 1; i <= r; ++i) cnt[qr[i].first
        ]++;
    for (int i = l; i <= m; ++i) {
        cnt[qr[i].first]--;
        if (cnt[qr[i].first] == 0) rv.push_back(qr[i].
            first);
    }
    contract(m + 1, r, rv, x, y);
    djs.save();
    for (int i = 0; i < (int)x.size(); ++i) {
        rc += cost[x[i]];
        djs.merge(st[x[i]], ed[x[i]]);
    }
    solve(m + 1, r, y, rc);
    djs.undo();
    for (int i = l; i <= m; ++i) cnt[qr[i].first]++;
}
```

3.7 Kth shortest path

```
// time: O(|E| \lg |E| + |V| \lg |V| + K)
// memory: O(|E| \lg |E| + |V|)
struct KSP{ // 1-base
  struct nd{
    int u, v, d;
    nd(int ui = 0, int vi = 0, int di = INF)
    { u = ui; v = vi; d = di; }
  };
  struct heap{
    nd* edge; int dep; heap* chd[4];
  };
  static int cmp(heap* a,heap* b)
  { return a->edge->d > b->edge->d; }
  struct node{
    int v; LL d; heap* H; nd* E;
    node(){}
    node(LL _d, int _v, nd* _E)
    { d =_d; v = _v; E = _E; }
    node(heap* _H, LL _d)
    { H = _H; d = _d; }
    friend bool operator<(node a, node b)
    { return a.d > b.d; }
  };
  int n, k, s, t, dst[ N ];
  nd *nxt[ N ];
  vector<nd*> g[ N ], rg[ N ];
  heap *nullNd, *head[ N ];
  void init( int _n , int _k , int _s , int _t ){
    n = _n; k = _k; s = _s; t = _t;
    for( int i = 1 ; i <= n ; i ++ ){
      g[ i ].clear(); rg[ i ].clear();
      nxt[ i ] = head[ i ] = NULL;
      dst[ i ] = -1;
    }
  }
  void add_edge( int ui , int vi , int di ){
    nd* e = new nd(ui, vi, di);
    g[ ui ].push_back( e );
    rg[ vi ].push_back( e );
  }
  queue<int> dfsQ;
  void dijkstra(){
    while(dfsQ.size()) dfsQ.pop();
    priority_queue<node> Q;
    Q.push(node(0, t, NULL));
    while (!Q.empty()){
      node p = Q.top(); Q.pop();
      if(dst[p.v] != -1) continue;
      dst[ p.v ] = p.d;
      nxt[ p.v ] = p.E;
      dfsQ.push( p.v );
      for(auto e: rg[ p.v ])
        Q.push(node(p.d + e->d, e->u, e));
    }
  }
  heap* merge(heap* curNd, heap* newNd){
    if(curNd == nullNd) return newNd;
    heap* root = new heap;
    memcpy(root, curNd, sizeof(heap));
    if(newNd->edge->d < curNd->edge->d){
      root->edge = newNd->edge;
      root->chd[2] = newNd->chd[2];
      root->chd[3] = newNd->chd[3];
      newNd->edge = curNd->edge;
      newNd->chd[2] = curNd->chd[2];
      newNd->chd[3] = curNd->chd[3];
    }
    if(root->chd[0]->dep < root->chd[1]->dep)
      root->chd[0] = merge(root->chd[0],newNd);
    else
      root->chd[1] = merge(root->chd[1],newNd);
    root->dep = max(root->chd[0]->dep, root->chd[1]->
        dep) + 1;
    return root;
  }
```

```
  vector<heap*> V;
  void build(){
    nullNd = new heap;
    nullNd->dep = 0;
    nullNd->edge = new nd;
    fill(nullNd->chd, nullNd->chd+4, nullNd);
    while(not dfsQ.empty()){
      int u = dfsQ.front(); dfsQ.pop();
      if(!nxt[ u ]) head[ u ] = nullNd;
      else head[ u ] = head[nxt[ u ]->v];
      V.clear();
      for( auto&& e : g[ u ] ){
        int v = e->v;
        if( dst[ v ] == -1 ) continue;
        e->d += dst[ v ] - dst[ u ];
        if( nxt[ u ] != e ){
          heap* p = new heap;
          fill(p->chd, p->chd+4, nullNd);
          p->dep = 1;
          p->edge = e;
          V.push_back(p);
        }
      }
      if(V.empty()) continue;
      make_heap(V.begin(), V.end(), cmp);
#define L(X) ((X<<1)+1)
#define R(X) ((X<<1)+2)
      for( size_t i = 0 ; i < V.size() ; i ++ ){
        if(L(i) < V.size()) V[i]->chd[2] = V[L(i)];
        else V[i]->chd[2]=nullNd;
        if(R(i) < V.size()) V[i]->chd[3] = V[R(i)];
        else V[i]->chd[3]=nullNd;
      }
      head[u] = merge(head[u], V.front());
    }
  }
  vector<LL> ans;
  void first_K(){
    ans.clear();
    priority_queue<node> Q;
    if( dst[ s ] == -1 ) return;
    ans.push_back( dst[ s ] );
    if( head[s] != nullNd )
      Q.push(node(head[s], dst[s]+head[s]->edge->d));
    for( int _ = 1 ; _ < k and not Q.empty() ; _ ++ ){
      node p = Q.top(), q; Q.pop();
      ans.push_back( p.d );
      if(head[ p.H->edge->v ] != nullNd){
        q.H = head[ p.H->edge->v ];
        q.d = p.d + q.H->edge->d;
        Q.push(q);
      }
      for( int i = 0 ; i < 4 ; i ++ )
        if( p.H->chd[ i ] != nullNd ){
          q.H = p.H->chd[ i ];
          q.d = p.d - p.H->edge->d + p.H->chd[ i ]->
              edge->d;
          Q.push( q );
        }
    }
  }
  void solve(){
    dijkstra();
    build();
    first_K();
  }
} solver;
```

3.8 General Matching

```
const int N = MXN, E = (2e5) * 2;
struct Graph{
  int to[E],bro[E],head[N],e;
  int lnk[N],vis[N],stp,n;
  void init( int _n ){
    stp = 0; e = 1; n = _n;
    for( int i = 1 ; i <= n ; i ++ )
      lnk[i] = vis[i] = head[i] = 0;
  }
  void add_edge(int u,int v){
    to[e]=v,bro[e]=head[u],head[u]=e++;
    to[e]=u,bro[e]=head[v],head[v]=e++;
  }
  bool dfs(int x){
    vis[x]=stp;
    for(int i=head[x];i;i=bro[i]){
      int v=to[i];
      if(!lnk[v]){
        lnk[x]=v,lnk[v]=x;
        return true;
      }else if(vis[lnk[v]]<stp){
        int w=lnk[v];
        lnk[x]=v,lnk[v]=x,lnk[w]=0;
        if(dfs(w)){
          return true;
        }
        lnk[w]=v,lnk[v]=w,lnk[x]=0;
      }
    }
    return false;
  }
  int solve(){
    int ans = 0;
    for(int i=1;i<=n;i++)
      if(!lnk[i]){
        stp++; ans += dfs(i);
      }
    return ans;
  }
} G;
```

3.9 Directed Minimum Spanning Tree

```
pair<PII, int> edge[MXN];
int dis[MXN], fr[MXN];
int vis[MXN], id[MXN];
int sol(int n, int m, int root) {
    int ans = 0;
    while (true) {
        for (int i = 1; i <= n; i++)
            dis[i] = INF;
        for (int i = 1; i <= m; i++) {
            int u, v, w = edge[i].S;
            tie(u, v) = edge[i].F;
            if (dis[v] > w) {
                dis[v] = w;
                fr[v] = u;
            }
        }
        for (int i = 1; i <= n; i++) {
            if (i == root) continue ;
            ans += dis[i];
            if (dis[i] == INF) return -1;
        }
        for (int i = 1; i <= n; i++)
            id[i] = vis[i] = 0;

        int num = 0;
        for (int i = 1; i <= n; i++) {
            int v = i;
            while (v != root && vis[v] != i && !id[v])
                {
                vis[v] = i;
                v = fr[v];
            }
            if (v != root && !id[v]) {
                id[v] = ++num;
                for (int u = fr[v]; u != v; u = fr[u])
                    {
                    id[u] = num;
                }
```

```
            }
        }
        if (!num) break ;
        for (int i = 1; i <= n; i++)
            if (!id[i]) id[i] = ++num;
        int nm = 0;
        for (int i = 1; i <= m; i++) {
            int u, v; tie(u, v) = edge[i].F;
            if (id[u] == id[v]) continue ;
            nm++;
            edge[nm].F.F = id[u];
            edge[nm].F.S = id[v];
            edge[nm].S = edge[i].S - dis[v];
        }
        m = nm;
        n = num;
        root = id[root];
    }
    return ans;
}
```

3.10 Dominator Tree

```
const int MAXN = 100010;
struct DominatorTree{
  int n , m , s;
  vector< int > edge[ MAXN ] , re_edge[ MAXN ];
  vector< int > cov[ MAXN ];
  int dfn[ MAXN ] , nfd[ MAXN ] , cntp;
  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 ){
    cntp = 0; n = _n; m = _m; s = _s;
    for (int i = 1; i <= n; i++) {
      edge[ i ].clear();
      re_edge[ i ].clear();
    }
  }
  void add_edge( int u , int v ){
    edge[ u ].pb( v );
    re_edge[ v ].pb( u );
  }
  void dfs( int u ){
    dfn[ u ] = ++cntp;
    nfd[ cntp ] = u;
    for( int v : edge[ u ] ) if( dfn[ v ] == 0 ){
      par[ v ] = u;
      dfs( v );
    }
  }
  void solve(){
    for (int i = 1; i <= n; i++) {
      dfn[ i ] = nfd[ i ] = 0;
      cov[ i ].clear();
      mom[ i ] = mn[ i ] = sdom[ i ] = i;
    }
    dfs( s );
    for (int i = n; i >= 2; i--) {
      int u = nfd[ i ];
      if( u == 0 ) continue ;
      for( int v : re_edge[ 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();
    }
    for (int i = 2; i <= n; i++) {
      int u = nfd[ i ];
      if( u == 0 ) continue ;
      if( idom[ u ] != sdom[ u ] )
        idom[ u ] = idom[ idom[ u ] ];
    }
  }
} domT;
```

3.11 Minimum Steiner Tree

```
// Minimum Steiner Tree
// O(V 3^T + V^2 2^T)
struct SteinerTree{
#define V 33
#define T 8
#define INF 1023456789
  int n , dst[V][V] , dp[1 << T][V] , tdst[V];
  void init( int _n ){
    n = _n;
    for( int i = 0 ; i < n ; i ++ ){
      for( int j = 0 ; j < n ; j ++ )
        dst[ i ][ j ] = INF;
      dst[ i ][ i ] = 0;
    }
  }
  void add_edge( int ui , int vi , int wi ){
    dst[ ui ][ vi ] = min( dst[ ui ][ vi ] , wi );
    dst[ vi ][ ui ] = min( dst[ vi ][ ui ] , wi );
  }
  void shortest_path(){
    for( int k = 0 ; k < n ; k ++ )
      for( int i = 0 ; i < n ; i ++ )
        for( int j = 0 ; j < n ; j ++ )
          dst[ i ][ j ] = min( dst[ i ][ j ],
                dst[ i ][ k ] + dst[ k ][ j ] );
  }
  int solve( const vector<int>& ter ){
    int t = (int)ter.size();
    for( int i = 0 ; i < ( 1 << t ) ; i ++ )
      for( int j = 0 ; j < n ; j ++ )
        dp[ i ][ j ] = INF;
    for( int i = 0 ; i < n ; i ++ )
      dp[ 0 ][ i ] = 0;
    for( int msk = 1 ; msk < ( 1 << t ) ; msk ++ ){
      if( msk == ( msk & (-msk) ) ){
        int who = __lg( msk );
        for( int i = 0 ; i < n ; i ++ )
          dp[ msk ][ i ] = dst[ ter[ who ] ][ i ];
        continue;
      }
      for( int i = 0 ; i < n ; i ++ )
        for( int submsk = ( msk - 1 ) & msk ; submsk ;
                  submsk = ( submsk - 1 ) & msk )
          dp[ msk ][ i ] = min( dp[ msk ][ i ],
                               dp[ submsk ][ i ] +
                               dp[ msk ^ submsk ][ i ] );
      for( int i = 0 ; i < n ; i ++ ){
        tdst[ i ] = INF;
        for( int j = 0 ; j < n ; j ++ )
          tdst[ i ] = min( tdst[ i ],
                      dp[ msk ][ j ] + dst[ j ][ i ] );
      }
      for( int i = 0 ; i < n ; i ++ )
        dp[ msk ][ i ] = tdst[ i ];
```

```
    }
    int ans = INF;
    for( int i = 0 ; i < n ; i ++ )
      ans = min( ans , dp[ ( 1 << t ) - 1 ][ i ] );
    return ans;
  }
} solver;
```

3.12 De Bruijn sequence

```
unordered_set<string> seen;
vector<int> edges;

// Modified DFS in which no edge
// is traversed twice
void dfs(string node, int& k, string& A) {
    for (int i = 0; i < k; ++i) {
        string str = node + A[i];
        if (seen.find(str) == seen.end()) {
            seen.insert(str);
            dfs(str.substr(1), k, A);
            edges.push_back(i);
        }
    }
}

// Function to find a de Bruijn sequence
// of order n on k characters
string deBruijn(int n, int k, string A) {

    // Clearing global variables
    seen.clear();
    edges.clear();

    string startingNode = string(n - 1, A[0]);
    dfs(startingNode, k, A);

    string S;

    // Number of edges
    int l = pow(k, n);
    for (int i = 0; i < l; ++i)
        S += A[edges[i]];
    S += startingNode;

    return S;
}
int main()  {
    int n = 3, k = 2;
    string A = "01";
    cout << deBruijn(n, k, A);
    return 0;
}
```

3.13 Vizing Coloring

```
namespace vizing { // returns edge coloring in adjacent
     matrix G. 1 - based
int C[kN][kN], G[kN][kN];
void clear(int N) {
    for (int i = 0; i <= N; i++) {
        for (int j = 0; j <= N; j++) C[i][j] = G[i][j]
            = 0;
    }
}
void solve(vector<pair<int, int>> &E, int N) {
    int X[kN] = {}, a;
    auto update = [&](int u) {
        for (X[u] = 1; C[u][X[u]]; X[u]++);
    };
    auto color = [&](int u, int v, int c) {
        int p = G[u][v];
        G[u][v] = G[v][u] = c;
        C[u][c] = v, C[v][c] = u;
        C[u][p] = C[v][p] = 0;
        if (p) X[u] = X[v] = p;
        else update(u), update(v);
        return p;
    };
    auto flip = [&](int u, int c1, int c2) {
        int p = C[u][c1];
        swap(C[u][c1], C[u][c2]);
        if (p) G[u][p] = G[p][u] = c2;
        if (!C[u][c1]) X[u] = c1;
        if (!C[u][c2]) X[u] = c2;
        return p;
    };
    for (int i = 1; i <= N; i++) X[i] = 1;
    for (int t = 0; t < E.size(); t++) {
        int u = E[t].first, v0 = E[t].second, v = v0,
            c0 = X[u], c = c0, d;
        vector<pair<int, int>> L;
        int vst[kN] = {};
        while (!G[u][v0]) {
            L.emplace_back(v, d = X[v]);
            if (!C[v][c]) for (a = (int)L.size() - 1; a
                 >= 0; a--) c = color(u, L[a].first, c)
                ;
            else if (!C[u][d]) for (a = (int)L.size() -
                 1; a >= 0; a--) color(u, L[a].first, L
                [a].second);
            else if (vst[d]) break;
            else vst[d] = 1, v = C[u][d];
        }
        if (!G[u][v0]) {
            for (; v; v = flip(v, c, d), swap(c, d));
            if (C[u][c0]) {
                for (a = (int)L.size() - 2; a >= 0 && L
                    [a].second != c; a--);
                for (; a >= 0; a--) color(u, L[a].first
                    , L[a].second);
            } else t--;
        }
    }
}}
```

3.14 Graph Hash

$$F_t(i) = (F_{t-1}(i) \times A + \sum_{i \to j} F_{t-1}(j) \times B + \sum_{j \to i} F_{t-1}(j) \times C + D \times (i = a)) \bmod P$$

for each node i, iterate t times. t, A, B, C, D, P are hash parameter

4 Flow

4.1 Dinic

```
struct Dinic{
  static const int MXN = 10000;
  struct Edge{ int v,f,re; };
  int n,s,t,level[MXN];
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t){
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v, int f){
    E[u].pb({v,f,SZ(E[v])});
    E[v].pb({u,0,SZ(E[u])-1});
  }
  bool BFS(){
    for (int i=0; i<n; i++) level[i] = -1;
    queue<int> que;
    que.push(s);
    level[s] = 0;
    while (!que.empty()){
      int u = que.front(); que.pop();
      for (auto it : E[u]){
        if (it.f > 0 && level[it.v] == -1){
          level[it.v] = level[u]+1;
```

```
          que.push(it.v);
        }
      }
    }
    return level[t] != -1;
  }
  int DFS(int u, int nf){
    if (u == t) return nf;
    int res = 0;
    for (auto &it : E[u]){
      if (it.f > 0 && level[it.v] == level[u]+1){
        int tf = DFS(it.v, min(nf,it.f));
        res += tf; nf -= tf; it.f -= tf;
        E[it.v][it.re].f += tf;
        if (nf == 0) return res;
      }
    }
    if (!res) level[u] = -1;
    return res;
  }
  int flow(int res=0){
    while ( BFS() )
      res += DFS(s,2147483647);
    return res;
  }
}flow;
```

4.2 Cost Flow

```
typedef pair<long long, long long> pll;
struct CostFlow {
  static const int MXN = 205;
  static const long long INF = 102938475610293847LL;
  struct Edge {
    int v, r;
    long long f, c;
  };
  int n, s, t, prv[MXN], prvL[MXN], inq[MXN];
  long long dis[MXN], fl, cost;
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t) {
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
    fl = cost = 0;
  }
  void add_edge(int u, int v, long long f, long long c)
       {
    E[u].PB({v, SZ(E[v])  , f,  c});
    E[v].PB({u, SZ(E[u])-1, 0, -c});
  }
  pll flow() {
    while (true) {
      for (int i=0; i<n; i++) {
        dis[i] = INF;
        inq[i] = 0;
      }
      dis[s] = 0;
      queue<int> que;
      que.push(s);
      while (!que.empty()) {
        int u = que.front(); que.pop();
        inq[u] = 0;
        for (int i=0; i<SZ(E[u]); i++) {
          int v = E[u][i].v;
          long long w = E[u][i].c;
          if (E[u][i].f > 0 && dis[v] > dis[u] + w) {
            prv[v] = u; prvL[v] = i;
            dis[v] = dis[u] + w;
            if (!inq[v]) {
              inq[v] = 1;
              que.push(v);
            }
          }
        }
      }
      if (dis[t] == INF) break;
      long long tf = INF;
      for (int v=t, u, l; v!=s; v=u) {
        u=prv[v]; l=prvL[v];
        tf = min(tf, E[u][l].f);
      }
      for (int v=t, u, l; v!=s; v=u) {
        u=prv[v]; l=prvL[v];
        E[u][l].f -= tf;
        E[v][E[u][l].r].f += tf;
      }
      cost += tf * dis[t];
      fl += tf;
    }
    return {fl, cost};
  }
}flow;
```

4.3 Kuhn Munkres

```
struct KM{
// Maximum Bipartite Weighted Matching (Perfect Match)
  static const int MXN = 650;
  static const int INF = 2147483647; // long long
  int n,match[MXN],vx[MXN],vy[MXN];
  int edge[MXN][MXN],lx[MXN],ly[MXN],slack[MXN];
  // ^^^^ long long
  void init(int _n){
    n = _n;
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        edge[i][j] = 0;
  }
  void add_edge(int x, int y, int w){ // long long
    edge[x][y] = w;
  }
  bool DFS(int x){
    vx[x] = 1;
    for (int y=0; y<n; y++){
      if (vy[y]) continue;
      if (lx[x]+ly[y] > edge[x][y]){
        slack[y] = min(slack[y], lx[x]+ly[y]-edge[x][y
            ]);
      } else {
        vy[y] = 1;
        if (match[y] == -1 || DFS(match[y])){
          match[y] = x;
          return true;
        }
      }
    }
    return false;
  }
  int solve(){
    fill(match,match+n,-1);
    fill(lx,lx+n,-INF);
    fill(ly,ly+n,0);
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        lx[i] = max(lx[i], edge[i][j]);
    for (int i=0; i<n; i++){
      fill(slack,slack+n,INF);
      while (true){
        fill(vx,vx+n,0);
        fill(vy,vy+n,0);
        if ( DFS(i) ) break;
        int d = INF; // long long
        for (int j=0; j<n; j++)
          if (!vy[j]) d = min(d, slack[j]);
        for (int j=0; j<n; j++){
          if (vx[j]) lx[j] -= d;
          if (vy[j]) ly[j] += d;
          else slack[j] -= d;
        }
      }
```

```
    }
    int res=0;
    for (int i=0; i<n; i++)
      res += edge[match[i]][i];
    return res;
  }
}graph;
```

4.4 Maximum Simple Graph Matching

```
const int MAX = 300;

int V, E;
int el[MAX][MAX];
int mtp[MAX];
int djs[MAX];
int bk[MAX], pr[MAX], vt[MAX];
queue<int> qu;

int ffa(int a){
  return (djs[a] == -1) ? a : djs[a] = ffa(djs[a]);
}

void djo(int a, int b){
  int fa = ffa(a), fb = ffa(b);
  if (fa != fb) djs[fb] = fa;
}

int lca(int u, int v){
  static int ts = 0;
  ts ++;
  while(1){
    if( u != -1 ){
      u = ffa(u);
      if(vt[u] == ts) return u;
      vt[u] = ts;
      if(pr[u] != -1) u = bk[pr[u]];
      else u = -1;
    }
    swap(u, v);
  }
  return u;
}

void flower(int u, int w){
  while(u != w){
    int v1 = pr[u], v2 = bk[v1];
    if(ffa(v2) != w) bk[v2] = v1;
    if(mtp[v1] == 1){
      qu.push(v1);
      mtp[v1] = 0;
    }
    if(mtp[v2] == 1){
      qu.push(v2);
      mtp[v2] = 0;
    }
    djo(v1, w);
    djo(v2, w);
    djo(u, w);
    u = v2;
  }
}
bool flow(int s){
  memset(mtp, -1, sizeof(mtp));
  while(qu.size()) qu.pop();
  qu.push(s);
  mtp[s] = 0; bk[s] = pr[s] = -1;

  while(qu.size() && pr[s] == -1){
    int u = qu.front(); qu.pop();
    for(int v=0; v<V; v++){

      if (el[u][v] == 0) continue;
      if (ffa(v) == ffa(u)) continue;

      if(pr[v] == -1){
        do{
          int t = pr[u];
          pr[v] = u; pr[u] = v;
          v = t; u = t==-1?-1:bk[t];
        }while( v != -1 );
        break;
      }else if(mtp[v] == 0){
        int w = lca(u, v);
        if(ffa(w) != ffa(u)) bk[u] = v;
        if(ffa(w) != ffa(v)) bk[v] = u;
        flower(u, w);
        flower(v, w);
      }else if(mtp[v] != 1){
        bk[v] = u;
        mtp[v] = 1;
        mtp[pr[v]] = 0;
        qu.push(pr[v]);
      }
    }
  }
  return pr[s] != -1;
}


int match(){

  memset(pr, -1, sizeof(pr));
  int a = 0;
  for (int i=0; i<V; i++){
    if (pr[i] == -1){
      if(flow(i)) a++;
      else mtp[i] = i;
    }
  }
  return a;
}
```

4.5 Minimum Weight Matching (Clique version)

```
struct Graph {
  // Minimum General Weighted Matching (Perfect Match)
      0-base
  static const int MXN = 105;

  int n, edge[MXN][MXN];
  int match[MXN],dis[MXN],onstk[MXN];
  vector<int> stk;

  void init(int _n) {
    n = _n;
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        edge[i][j] = 0;
  }
  void add_edge(int u, int v, int w) {
    edge[u][v] = edge[v][u] = w;
  }
  bool SPFA(int u){
    if (onstk[u]) return true;
    stk.PB(u);
    onstk[u] = 1;
    for (int v=0; v<n; v++){
      if (u != v && match[u] != v && !onstk[v]){
        int m = match[v];
        if (dis[m] > dis[u] - edge[v][m] + edge[u][v]){
          dis[m] = dis[u] - edge[v][m] + edge[u][v];
          onstk[v] = 1;
          stk.PB(v);
          if (SPFA(m)) return true;
          stk.pop_back();
          onstk[v] = 0;
        }
      }
    }
    onstk[u] = 0;
```

```
    stk.pop_back();
    return false;
  }

  int solve() {
    // find a match
    for (int i=0; i<n; i+=2){
      match[i] = i+1;
      match[i+1] = i;
    }
    while (true){
      int found = 0;
      for (int i=0; i<n; i++)
        dis[i] = onstk[i] = 0;
      for (int i=0; i<n; i++){
        stk.clear();
        if (!onstk[i] && SPFA(i)){
          found = 1;
          while (SZ(stk)>=2){
            int u = stk.back(); stk.pop_back();
            int v = stk.back(); stk.pop_back();
            match[u] = v;
            match[v] = u;
          }
        }
      }
      if (!found) break;
    }
    int ret = 0;
    for (int i=0; i<n; i++)
      ret += edge[i][match[i]];
    ret /= 2;
    return ret;
  }
}graph;
```

4.6 Bounded max flow

```
// Max flow with lower/upper bound on edges
// source = 1 , sink = n
const int N = 1005;
const int M = 3005;
int in[ N ] , out[ N ];
int l[ M ] , r[ M ] , a[ M ] , b[ M ];
int n, m;
int solve (int n, int m) {
  int st = 0, ed = n + 1;
  flow.init(n + 2, st, ed);
  for (int i = 1; i <= n; i++) {
    in[i] = out[i] = 0;
  }
  for (int i = 1; i <= m; i++) {
    in[ r[ i ] ] += a[ i ];
    out[ l[ i ] ] += a[ i ];
    flow.add_edge( l[ i ] , r[ i ] , b[ i ] - a[ i ] );
    // flow from l[i] to r[i] must in [a[ i ], b[ i ]]
  }
  int nd = 0;
  for (int i = 1; i <= n; i++) {
    if (in[ i ] < out[ i ]) {
      flow.add_edge( i , ed , out[ i ] - in[ i ] );
      nd += out[ i ] - in[ i ];
    }
    if (out[ i ] < in[ i ])
      flow.add_edge( st , i , in[ i ] - out[ i ] );
  }
  // original sink to source
  flow.add_edge( n , 1 , INF );
  if( flow.flow() != nd )
    // no solution
    return -1;
  int ans = flow.E[ 1 ].back().f; // source to sink
  flow.E[ 1 ].back().f = flow.E[ n ].back().f = 0;
  // take out super source and super sink
  for (int i = 0; i < SZ(flow.E[ st ]); i++) {
    flow.E[ st ][ i ].f = 0;
    Dinic::Edge &e = flow.E[ st ][ i ];
    flow.E[ e.v ][ e.re ].f = 0;
  }
  for (int i = 0; i < SZ(flow.E[ ed ]); i++){
    flow.E[ ed ][ i ].f = 0;
    Dinic::Edge &e = flow.E[ ed ][ i ];
    flow.E[ e.v ][ e.re ].f = 0;
  }
  flow.add_edge( st , 1 , INF );
  flow.add_edge( n , ed , INF );
  return ans + flow.flow();
}
```

4.7 SW-mincut

```
// global min cut
struct SW{ // O(V^3)
  static const int MXN = 514;
  int n,vst[MXN],del[MXN];
  int edge[MXN][MXN],wei[MXN];
#define FZ(x) memset(x, 0, sizeof(x))
  void init(int _n){
    n = _n; FZ(edge); FZ(del);
  }
  void add_edge(int u, int v, int w){
    edge[u][v] += w; edge[v][u] += w;
  }
  void search(int &s, int &t){
    FZ(vst); FZ(wei);
    s = t = -1;
    while (true){
      int mx=-1, cur=0;
      for (int i=0; i<n; i++)
        if (!del[i] && !vst[i] && mx<wei[i])
          cur = i, mx = wei[i];
      if (mx == -1) break;
      vst[cur] = 1;
      s = t; t = cur;
      for (int i=0; i<n; i++)
        if (!vst[i] && !del[i]) wei[i] += edge[cur][i];
    }
  }
  int solve(){
    int res = 2147483647;
    for (int i=0,x,y; i<n-1; i++){
      search(x,y);
      res = min(res,wei[y]);
      del[y] = 1;
      for (int j=0; j<n; j++)
        edge[x][j] = (edge[j][x] += edge[y][j]);
    }
    return res;
  }
}graph;
```

4.8 Flow Method

```
Maximize c^T x subject to Ax ≤ b, x ≥ 0;
with the corresponding symmetric dual problem,
Minimize b^T y subject to A^T y ≥ c, y ≥ 0.

Maximize c^T x subject to Ax ≤ b;
with the corresponding asymmetric dual problem,
Minimize b^T y subject to A^T y = c, y ≥ 0.


Minimum vertex cover on bipartite graph =
Maximum matching on bipartite graph =
Max flow with source to one side, other side to sink

To reconstruct the minimum vertex cover, dfs from each
unmatched vertex on the left side and with unused edges
only. Equivalently, dfs from source with unused edges
only and without visiting sink. Then, a vertex is
    chosen
```

```
iff. it is on the left side and without visited or on
the right side and visited through dfs.


Maximum density subgraph ( \sum W_e + \sum W_v ) / |V|

Binary search on answer:
For a fixed D, construct a Max flow model as follow:
Let S be Sum of all weight( or inf)
1. from source to each node with cap = S
2. For each (u,v,w) in E, (u->v,cap=w), (v->u,cap=w)
3. For each node v, from v to sink with cap = S + 2 * D
     - deg[v] - 2 * (W of v)
where deg[v] = \sum weight of edge associated with v
If maxflow < S * |V|, D is an answer.

Requiring subgraph: all vertex can be reached from
    source with
edge whose cap > 0.
```

5 Math

5.1 Bigint

```
struct Bigint{
  static const int LEN = 60;
  static const int BIGMOD = 10000;

  int s;
  int vl, v[LEN];
  //  vector<int> v;
  Bigint() : s(1) { vl = 0; }
  Bigint(long long a) {
    s = 1; vl = 0;
    if (a < 0) { s = -1; a = -a; }
    while (a) {
      push_back(a % BIGMOD);
      a /= BIGMOD;
    }
  }
  Bigint(string str) {
    s = 1; vl = 0;
    int stPos = 0, num = 0;
    if (!str.empty() && str[0] == '-') {
      stPos = 1;
      s = -1;
    }
    for (int i=SZ(str)-1, q=1; i>=stPos; i--) {
      num += (str[i] - '0') * q;
      if ((q *= 10) >= BIGMOD) {
        push_back(num);
        num = 0; q = 1;
      }
    }
    if (num) push_back(num);
    n();
  }

  int len() const {
    return vl;
    //    return SZ(v);
  }
  bool empty() const { return len() == 0; }
  void push_back(int x) {
    v[vl++] = x;
    //    v.PB(x);
  }
  void pop_back() {
    vl--;
    //    v.pop_back();
  }
  int back() const {
    return v[vl-1];
    //    return v.back();
  }
  void n() {
    while (!empty() && !back()) pop_back();
  }
  void resize(int nl) {
    vl = nl;
    fill(v, v+vl, 0);
    //    v.resize(nl);
    //    fill(ALL(v), 0);
  }

  void print() const {
    if (empty()) { putchar('0'); return; }
    if (s == -1) putchar('-');
    printf("%d", back());
    for (int i=len()-2; i>=0; i--) printf("%.4d",v[i]);
  }
  friend std::ostream& operator << (std::ostream& out,
      const Bigint &a) {
    if (a.empty()) { out << "0"; return out; }
    if (a.s == -1) out << "-";
    out << a.back();
    for (int i=a.len()-2; i>=0; i--) {
      char str[10];
      snprintf(str, 5, "%.4d", a.v[i]);
      out << str;
    }
    return out;
  }

  int cp3(const Bigint &b)const {
    if (s != b.s) return s - b.s;
    if (s == -1) return -(-*this).cp3(-b);
    if (len() != b.len()) return len()-b.len();//int
    for (int i=len()-1; i>=0; i--)
      if (v[i]!=b.v[i]) return v[i]-b.v[i];
    return 0;
  }

  bool operator < (const Bigint &b)const{ return cp3(b)
      <0; }
  bool operator <= (const Bigint &b)const{ return cp3(b
      )<=0; }
  bool operator == (const Bigint &b)const{ return cp3(b
      )==0; }
  bool operator != (const Bigint &b)const{ return cp3(b
      )!=0; }
  bool operator > (const Bigint &b)const{ return cp3(b)
      >0; }
  bool operator >= (const Bigint &b)const{ return cp3(b
      )>=0; }

  Bigint operator - () const {
    Bigint r = (*this);
    r.s = -r.s;
    return r;
  }
  Bigint operator + (const Bigint &b) const {
    if (s == -1) return -(-(*this)+(-b));
    if (b.s == -1) return (*this)-(-b);
    Bigint r;
    int nl = max(len(), b.len());
    r.resize(nl + 1);
    for (int i=0; i<nl; i++) {
      if (i < len()) r.v[i] += v[i];
      if (i < b.len()) r.v[i] += b.v[i];
      if(r.v[i] >= BIGMOD) {
        r.v[i+1] += r.v[i] / BIGMOD;
        r.v[i] %= BIGMOD;
      }
    }
    r.n();
    return r;
  }
  Bigint operator - (const Bigint &b) const {
    if (s == -1) return -(-(*this)-(-b));
    if (b.s == -1) return (*this)+(-b);
    if ((*this) < b) return -(b-(*this));
    Bigint r;
```

```
    r.resize(len());
    for (int i=0; i<len(); i++) {
      r.v[i] += v[i];
      if (i < b.len()) r.v[i] -= b.v[i];
      if (r.v[i] < 0) {
        r.v[i] += BIGMOD;
        r.v[i+1]--;
      }
    }
    r.n();
    return r;
  }
  Bigint operator * (const Bigint &b) {
    Bigint r;
    r.resize(len() + b.len() + 1);
    r.s = s * b.s;
    for (int i=0; i<len(); i++) {
      for (int j=0; j<b.len(); j++) {
        r.v[i+j] += v[i] * b.v[j];
        if(r.v[i+j] >= BIGMOD) {
          r.v[i+j+1] += r.v[i+j] / BIGMOD;
          r.v[i+j] %= BIGMOD;
        }
      }
    }
    r.n();
    return r;
  }
  Bigint operator / (const Bigint &b) {
    Bigint r;
    r.resize(max(1, len()-b.len()+1));
    int oriS = s;
    Bigint b2 = b; // b2 = abs(b)
    s = b2.s = r.s = 1;
    for (int i=r.len()-1; i>=0; i--) {
      int d=0, u=BIGMOD-1;
      while(d<u) {
        int m = (d+u+1)>>1;
        r.v[i] = m;
        if((r*b2) > (*this)) u = m-1;
        else d = m;
      }
      r.v[i] = d;
    }
    s = oriS;
    r.s = s * b.s;
    r.n();
    return r;
  }
  Bigint operator % (const Bigint &b) {
    return (*this)-(*this)/b*b;
  }
};
```

5.2 ax+by=gcd

```
PLL ex_gcd(LL a, LL b){
  if(b == 0) return MP(1, 0);
  else{
    LL p = a / b;
    PLL q = ex_gcd(b, a % b);
    return MP(q.S, q.F - q.S * p);
  }
}
```

5.3 Linear Prime Sieve

```
int ck[MXN];
vector<int> pr;
void linear_sieve(){
  for (int i = 2; i < MXN; i++) {
    if(!ck[i]) pr.pb(i);
    for (int j = 0; i*pr[j] < MXN; j++){
      ck[ i*pr[j] ] = pr[j];
      if(i % pr[j] == 0) break;
    }
  }
}
```

5.4 Chinese Remainder

```
template <typename T> tuple<T, T, T> extgcd(T a, T b) {
    if (!b) return make_tuple(a, 1, 0);
    T d, x, y;
    tie(d, x, y) = extgcd(b, a % b);
    return make_tuple(d, y, x - (a / b) * y);
}
long long crt(vector<int> mod, vector<int> a) {
    long long mult = mod[0];
    int n = (int)mod.size();
    long long res = a[0];
    for (int i = 1; i < n; ++i) {
        long long d, x, y;
        tie(d, x, y) = extgcd(mult, mod[i] * 1ll);
        if ((a[i] - res) % d) return -1;
        long long new_mult = mult / __gcd(mult, 1ll *
            mod[i]) * mod[i];
        res += x * ((a[i] - res) / d) % new_mult * mult
             % new_mult;
        mult = new_mult;
        ((res %= mult) += mult) %= mult;
    }
    return res;
}
```

5.5 Fast Fourier Transform

```
#define rep(i, a) for (int i = 0; i < a; i++)
#define rep1(i, a, b) for(int i = a; i < b; i++)
struct cp{
  double a,b;
  cp(){};
  cp(double _a, double _b){
    a = _a, b = _b;
  }
  cp operator +(const cp &o){ return cp(a+o.a, b+o.b);
      }
  cp operator -(const cp &o){ return cp(a-o.a, b-o.b);
      }
  cp operator *(const cp &o){ return cp(a*o.a-b*o.b, b*
      o.a+a*o.b); }
  cp operator *(const double &o){ return cp(a*o, b*o);
      }
  cp operator !(){ return cp(a, -b); }
}w[MXN];
int pos[MXN];
void fft_init(int len){
  int j = 0;
  while((1<<j) < len) j++;
  j--;
  rep(i, len) pos[i] = pos[i>>1]>>1 | ((i&1)<<j);
  return ;
}
void fft(cp *x, int len, int sta){
  rep(i, len) if(i < pos[i]) swap(x[i], x[pos[i]]);
  w[0] = cp(1, 0);
  for(unsigned i = 2; i <= len; i <<= 1){
    cp g = cp(cos(2*PI/i), sin(2*PI/i)*sta);
    for(int j = i>>1; j >= 0; j -= 2) w[j] = w[j>>1];
    for(int j = 1; j < (i>>1); j += 2) w[j] = w[j-1]*g;
    for(int j = 0; j < len; j += i){
      cp *a = x+j, *b = a+(i>>1);
      rep(l, i>>1){
          cp o = b[l]*w[l];
          b[l] = a[l]-o;
          a[l] = a[l]+o;
      }
    }
```

```
  }
  if(sta == -1) rep(i, len) x[i].a /= len, x[i].b /=
      len;
  return ;
}
cp xt[MXN], yt[MXN], zt[MXN];
LL st[3][MXN];
void FFT(int a, int b, int l1, int l2, int c){
  int len = 1;
  while(len <= (l1+l2)>>1) len <<= 1;
  fft_init(len);
  rep1(i, l1>>1, len) xt[i].a = xt[i].b = 0;
  rep(i, l1) (i&1 ? xt[i>>1].b : xt[i>>1].a) = st[a][i
      ];
  fft(xt, len, 1);

  rep1(i, l2>>1, len) yt[i].a = yt[i].b = 0;
  rep(i, l2) (i&1 ? yt[i>>1].b : yt[i>>1].a) = st[b][i
      ];
  fft(yt, len, 1);


  rep(i, len>>1){
    int j = len - 1&len - i;
    zt[i] = xt[i]*yt[i] - (xt[i]-!xt[j])*(yt[i]-!yt[j])
        *(w[i]+cp(1,0))*0.25;
  }
  rep1(i, len>>1, len){
    int j = len - 1&len - i;
    zt[i] = xt[i]*yt[i] - (xt[i]-!xt[j])*(yt[i]-!yt[j])
        *(cp(1,0)-w[i^len>>1])*0.25;
  }
  fft(zt, len, -1);
  rep(i, l1 + l2 - 1){
    if(i&1) st[c][i] = (LL)(zt[i>>1].b+0.5);
    else st[c][i] = (LL)(zt[i>>1].a+0.5);
  }
  return ;
}
```

5.6 Kth Residue

```
/*
 * find x for x^t = m (mod p) p is prime
 * 1. find PrimitiveRoot of p (assume it is v) O( sqrt(
     n)log(n) )
 * 2. v^(at) = v^b
 * 3. use Bsgs to find b O( sqrt(n) + m*log(m) )
 * 4. use ex_gcd to find a(ax + by = gcd, at + b(p-1) =
      gcd) O(log(n))
 */
struct KthResidue{
  struct PriRoot{
    int a[MXN], cntp;
    LL phi(LL n){
      int h = sqrt(n);
      LL res = n, v = n;
      for (int i = 2; i <= h; i++) {
        if (v % i == 0) {
          res = res / i * (i - 1);
          while (v % i == 0) v /= i;
        }
      }
      if (v != 1) res = res / v * (v - 1);
      return res;
    }
    int solve(LL n){
      LL num = phi(n); // if n is prime, num = n - 1
      LL v = num;
      int h = sqrt(num);
      cntp = 0;
      for (int i = 2; i <= h; i++) {
        if (v % i == 0) {
          a[++cntp] = i;
          while (v % i == 0) v /= i;
        }
      }
      if (v != 1) a[++cntp] = v;
      v = num;
      for (int i = 2; i < n; i++) {
        if (gcd(n, i) != 1) continue;
        bool ok = 1;
        for (int j = 1; j <= cntp; j++) {
          if (mypow(i, v / a[j], n) == 1) {
            ok = 0; break;
          }
        }
        if (ok) return i;
      }
      return -1;
    }
  }root;
  struct Bsgs{
    map<LL, int>mp;
    LL solve(LL v, LL m, LL p){
      mp.clear();
      int h = ceil( sqrt(p + 0.5) );
      LL cv = 1;
      for (int i = 0; i < h; i++) {
        mp[cv] = i;
        cv = cv*v%p;
      }
      cv = mypow(cv, p - 2, p);
      int ok = 0, ans = 0;
      for (int i = 0; i <= h; i++) {
        if (mp.find(m) != mp.end()) {
          ans += mp[m];
          ok = 1; break;
        }
        ans += h;
        m = m*cv%p;
      }
      return ok ? ans : -1;
    }
  }bsgs;
  LL solve(LL t, LL m, LL p){
    LL v = root.solve(p);
    LL gd = bsgs.solve(v, m, p);
    if (gd == -1) return -1;

    PLL res = ex_gcd(t, p-1);
    LL val = (t*res.F + (p-1)*res.S);
    if (gd % val) return -1;
    LL num = (res.F*(gd / val)%(p-1) + p - 1) % (p-1);
    return mypow(v, num, p);
  }
}residue;
```

5.7 Gauss Elimination

```
const int MAX = 300;
const double EPS = 1e-8;

double mat[MAX][MAX];
void Gauss(int n) {
  for(int i=0; i<n; i++) {
    bool ok = 0;
    for(int j=i; j<n; j++) {
      if(fabs(mat[j][i]) > EPS) {
        swap(mat[j], mat[i]);
        ok = 1;
        break;
      }
    }
    if(!ok) continue;

    double fs = mat[i][i];
    for(int j=i+1; j<n; j++) {
      double r = mat[j][i] / fs;
      for(int k=i; k<n; k++) {
        mat[j][k] -= mat[i][k] * r;
      }
```

```
    }
  }
}
```

5.8 NTT(eddy ver.)

```
typedef long long LL;
// Remember coefficient are mod P
/* p=a*2^n+1
   n    2^n         p          a    root
   16   65536       65537      1    3
   20   1048576     7340033    7    3 */
// (must be 2^k)
template<LL P, LL root, int MAXN>
struct NTT{
  static LL bigmod(LL a, LL b) {
    LL res = 1;
    for (LL bs = a; b; b >>= 1, bs = (bs * bs) % P)
      if(b&1) res=(res*bs)%P;
    return res;
  }
  static LL inv(LL a, LL b) {
    if(a==1)return 1;
    return (((LL)(a-inv(b%a,a))*b+1)/a)%b;
  }
  LL omega[MAXN+1];
  NTT() {
    omega[0] = 1;
    LL r = bigmod(root, (P-1)/MAXN);
    for (int i=1; i<=MAXN; i++)
      omega[i] = (omega[i-1]*r)%P;
  }
  // n must be 2^k
  void tran(int n, LL a[], bool inv_ntt=false){
    int basic = MAXN / n , theta = basic;
    for (int m = n; m >= 2; m >>= 1) {
      int mh = m >> 1;
      for (int i = 0; i < mh; i++) {
        LL w = omega[i*theta%MAXN];
        for (int j = i; j < n; j += m) {
          int k = j + mh;
          LL x = a[j] - a[k];
          if (x < 0) x += P;
          a[j] += a[k];
          if (a[j] > P) a[j] -= P;
          a[k] = (w * x) % P;
        }
      }
      theta = (theta * 2) % MAXN;
    }
    int i = 0;
    for (int j = 1; j < n - 1; j++) {
      for (int k = n >> 1; k > (i ^= k); k >>= 1);
      if (j < i) swap(a[i], a[j]);
    }
    if (inv_ntt) {
      LL ni = inv(n,P);
      reverse( a+1 , a+n );
      for (i = 0; i < n; i++)
        a[i] = (a[i] * ni) % P;
    }
  }
};
const LL P=2013265921,root=31;
const int MAXN=4194304;
NTT<P, root, MAXN> ntt;
```

5.9 FWT

```
/* xor convolution:
 * x = (x0,x1) , y = (y0,y1)
 * z = ( x0y0 + x1y1 , x0y1 + x1y0 )
 * =>
 * x' = ( x0+x1 , x0-x1 ) , y' = ( y0+y1 , y0-y1 )
 * z' = ( ( x0+x1 )( y0+y1 ) , ( x0-x1 )( y0-y1 ) )
 * z = (1/2) * z''
 * or convolution:
 * x = (x0, x0+x1), inv = (x0, x1-x0) w/o final div
 * and convolution:
 * x = (x0+x1, x1), inv = (x0-x1, x1) w/o final div */
typedef long long LL;
const int MAXN = (1<<20)+10;
const LL MOD = 1e9+7;
inline LL pw( LL x , LL k ) {
  LL res = 1;
  for( LL bs = x ; k ; k >>= 1, bs = (bs * bs)%MOD )
    if( k&1 ) res = ( res * bs ) % MOD;
  return res;
}
inline LL inv( LL x ) {
  return pw( x , MOD-2 );
}
inline void fwt( LL x[ MAXN ] , int N , bool inv=0 ) {
  for( int d = 1 ; d < N ; d <<= 1 ) {
    int d2 = d<<1;
    for( int s = 0 ; s < N ; s += d2 )
      for( int i = s , j = s+d ; i < s+d ; i++, j++ ){
        LL ta = x[ i ] , tb = x[ j ];
        x[ i ] = ta+tb;
        x[ j ] = ta-tb;
        if( x[ i ] >= MOD ) x[ i ] -= MOD;
        if( x[ j ] < 0 ) x[ j ] += MOD;
      }
  }
  if( inv )
    for( int i = 0 ; i < N ; i++ ) {
      x[ i ] *= inv( N );
      x[ i ] %= MOD;
    }
}
```

5.10 Miller Rabin

```
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
// you want to use magic.
long long mult(LL a, LL b, LL mod) {
  a %= mod, b %= mod;
  LL res = 0;
  while (b) {
    if(b & 1) res = (res + a) % mod;
    b >>= 1;
    a = (a<<1)%mod;
  }
  return res;
}
long long power(long long x,long long p,long long mod){
  long long s=1,m=x;
  while (p) {
    if(p&1) s=mult(s,m,mod);
    p>>=1;
    m=mult(m,m,mod);
  }
  return s;
}
bool witness(long long a,long long n,long long u,int t)
    {
  long long x=power(a,u,n);
  for(int i = 0;i < t; i++) {
    long long nx=mult(x,x,n);
    if(nx==1&&x!=1&&x!=n-1) return 1;
    x=nx;
  }
  return x!=1;
}
bool miller_rabin(long long n,int s=100) {
  // iterate s times of witness on n
```

```
  // return 1 if prime, 0 otherwise
  if(n<2) return 0;
  if(!(n&1)) return n==2;
  long long u=n-1;
  int t=0;
  // n-1 = u*2^t
  while(!(u&1)) {
    u>>=1;
    t++;
  }
  while(s--) {
    long long a=randll()%(n-1)+1;
    if(witness(a,n,u,t)) return 0;
  }
  return 1;
}
```

5.11 Pollard Rho

```
// does not work when n is prime
LL f(LL x, LL mod){ return add(mul(x,x,mod),1,mod); }
LL pollard_rho(LL n) {
  if(!(n&1)) return 2;
  while(true){
    LL y=2, x=rand()%(n-1)+1, res=1;
    for(int sz=2; res==1; sz*=2) {
      for(int i=0; i<sz && res<=1; i++) {
        x = f(x, n);
        res = __gcd(abs(x-y), n);
      }
      y = x;
    }
    if (res!=0 && res!=n) return res;
  }
}
```

5.12 Algorithms about Primes

```
/*
 * 12721
 * 13331
 * 14341
 * 75577
 * 123457
 * 222557
 * 556679
 * 999983
 * 1097774749
 * 1076767633
 * 100102021
 * 999997771
 * 1001010013
 * 1000512343
 * 987654361
 * 999991231
 * 999888733
 * 98789101
 * 987777733
 * 999991921
 * 1010101333
 * 1010102101
 * 1000000000039
 * 1000000000000037
 * 2305843009213693951
 * 4611686018427387847
 * 9223372036854775783
 * 18446744073709551557
 */

int mu[MX],p_tbl[MX];
vector<int> primes;
void sieve() {
  mu[1] = p_tbl[1] = 1;
  for (int i=2; i<MX; i++) {
    if (!p_tbl[i]) {
      p_tbl[i] = i;
      primes.PB(i);
      mu[i] = -1;
    }
    for (auto p : primes) {
      int x = i*p;
      if (x >= M) break;
      p_tbl[x] = p;
      mu[x] = -mu[i];
      if (i%p==0) {
        mu[x] = 0;
        break;
      }
    }
  }
}

vector<int> factor(int x) {
  vector<int> fac{1};
  while (x > 1) {
    int fn=SZ(fac), p=p_tbl[x], pos=0;
    while (x%p == 0) {
      x /= p;
      for (int i=0; i<fn; i++)
        fac.PB(fac[pos++]*p);
    }
  }
  return fac;
}
```

5.13 Faulhaber

```
/* faulhaber's formula -
 * cal power sum formula of all p=1~k in O(k^2) */
#define MAXK 2500
const int mod = 1000000007;
int b[MAXK]; // bernoulli number
int inv[MAXK+1]; // inverse
int cm[MAXK+1][MAXK+1]; // combinactories
int co[MAXK][MAXK+2]; // coeeficient of x^j when p=i
inline int getinv(int x) {
  int a=x,b=mod,a0=1,a1=0,b0=0,b1=1;
  while(b) {
    int q,t;
    q=a/b; t=b; b=a-b*q; a=t;
    t=b0; b0=a0-b0*q; a0=t;
    t=b1; b1=a1-b1*q; a1=t;
  }
  return a0<0?a0+mod:a0;
}
inline void pre() {
  /* combinational */
  for(int i=0;i<=MAXK;i++) {
    cm[i][0]=cm[i][i]=1;
    for(int j=1;j<i;j++)
      cm[i][j]=add(cm[i-1][j-1],cm[i-1][j]);
  }
  /* inverse */
  for(int i=1;i<=MAXK;i++) inv[i]=getinv(i);
  /* bernoulli */
  b[0]=1; b[1]=getinv(2); // with b[1] = 1/2
  for(int i=2;i<MAXK;i++) {
    if(i&1) { b[i]=0; continue; }
    b[i]=1;
    for(int j=0;j<i;j++)
      b[i]=sub(b[i],
               mul(cm[i][j],mul(b[j], inv[i-j+1])));
  }
  /* faulhaber */
  // sigma_x=1~n {x^p} =
  //   1/(p+1) * sigma_j=0~p {C(p+1,j)*Bj*n^(p-j+1)}
  for(int i=1;i<MAXK;i++) {
    co[i][0]=0;
    for(int j=0;j<=i;j++)
      co[i][i-j+1]=mul(inv[i+1], mul(cm[i+1][j], b[j]))
          ;
  }
}
```

```
/* sample usage: return f(n,p) = sigma_x=1~n (x^p) */
inline int solve(int n,int p) {
  int sol=0,m=n;
  for(int i=1;i<=p+1;i++) {
    sol=add(sol,mul(co[p][i],m));
    m = mul(m, n);
  }
  return sol;
}
```

5.14 Poly operation

```
struct Polyop {
  NTT<P, root, MAXN> ntt;
  static int nxt2k(int x) {
    int i = 1; for (; i < x; i <<= 1); return i;
  }
  void Mul(int n, LL a[], int m, LL b[], LL c[]) {
    static LL aa[MAXN], bb[MAXN];
    int N = nxt2k(n+m);
    copy(a, a+n, aa); fill(aa+n, aa+N, 0);
    copy(b, b+m, bb); fill(bb+m, bb+N, 0);
    ntt.tran(N, aa); ntt.tran(N, bb);
    for(int i = 0; i < N; i++) c[i] = aa[i] * bb[i] % P
        ;
    ntt.tran(N, c, true);
  }
  void Inv(int n, LL a[], LL b[]) {
    // ab = aa^-1 = 1 mod x^(n/2)
    // (b - a^-1)^2 = 0 mod x^n
    // bb + a^-2 - 2 ba^-1 = 0
    // bba + a^-1 - 2b = 0
    // -bba + 2b = a^-1
    static LL tmp[MAXN];
    if (n == 1) {
      b[0] = mypow(a[0], P-2);
      return;
    }
    Inv((n+1)/2, a, b);
    int N = nxt2k(n*2);
    copy(a, a+n, tmp2);
    fill(tmp+n, tmp+N, 0);
    fill(b+n, b+N, 0);
    ntt.tran(N, tmp); ntt.tran(N, b);
    for (int i = 0; i < N; i++) {
      LL t1 = (2 - b[i] * tmp[i]) % P;
      if (t1 < 0) t1 += P;
      b[i] = b[i] * t1 % P;
    }
    ntt.tran(N, b, true);
    fill(b+n, b+N, 0);
  }
  void Sqrt(int n, LL a[], LL b[]) {
    // (a+(b')^2) / 2b'
    static LL tmp[MAXN], tmp2[MAXN];
    if (n == 1) {
      b[0] = sqrt(a[0]); // if (!exist b[0]) => false
      return ;
    }
    int N = nxt2k(n*2);
    int m = (n+1)>>1;
    Sqrt(m, a, b);
    Inv(n, b, tmp);
    fill(tmp + n, tmp + N, 0);
    copy(a, a + n, tmp2);
    fill(tmp2 + n, tmp2 + N, 0);
    ntt.tran(N, tmp2); ntt.tran(N, tmp);
    for (int i = 0; i < N; i++)
      tmp2[i] = tmp2[i]*tmp[i]%P;
    ntt.tran(N, tmp2, true);
    for (int i = 0; i < n; i++)
      b[i] = (b[i] + tmp2[i]) * inv2 % P;
    fill(tmp2, tmp2 + N, 0);
    fill(tmp, tmp + N, 0);
    fill(b + n, b + N, 0);
  }
} op;
```

5.15 Theorem

5.15.1 Lucas' Theorem

For non-negative integer n, m and prime p, $\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod{p}$ where m_i is the i-th digit of m in base p.

5.15.2 Sum of Two Squares Thm (Legendre)

For a given positive integer n, let
$D_1 = (\#$ of positive integers d dividing N that $1 \equiv d \pmod 4)$
$D_3 = (\#$ of positive integers d dividing N that $3 \equiv d \pmod 4)$
then n can be written as a sum of two squares in exactly
$R(n) = 4(D_1 - D_3)$ ways.

5.15.3 Difference of D1-D3 Thm

let $n = 2^t \cdot (p_1^{e_1} \cdot ... \cdot p_r^{e_r}) \cdots (q_1^{f_1} \cdot ... \cdot q_s^{f_s})$
where p_i, q_i are primes and $1 \equiv p_i \pmod 4), 3 \equiv q_i \pmod 4$
then $D_1 - D_3 = \begin{cases} (e_1+1)(e_2+1)...(e_r+1), & \text{if } f_i \text{ all even} \\ 0, & \text{if any } f_i \text{ is odd} \end{cases}$

5.15.4 Krush–Kuhn–Tucker Conditions

Stationarity
For maximizing $f(x)$: $\nabla f(x^*) = \sum_{i=1}^{m} \mu_i \nabla g_i(x^*) + \sum_{j=1}^{l} \lambda_j \nabla h_j(x^*)$
For minimizing $f(x)$: $-\nabla f(x^*) = \sum_{i=1}^{m} \mu_i \nabla g_i(x^*) + \sum_{j=1}^{l} \lambda_j \nabla h_j(x^*)$

Primal feasibility
$g_i(x^*) \le 0$, for all $i = 1, \dots, m$
$h_j(x^*) = 0$, for all $j = 1, \dots, l$

Dual feasibility
$\mu_i \ge 0$, for all $i = 1, \dots, m$

Complementary slackness
$\mu_i g_i(x^*) = 0$, for all $i = 1, \dots, m$

5.15.5 Chinese remainder theorem

$x \equiv r_i \mod p_i$
$N = \prod p_i$
$N_i = N/p_i$
$x \equiv \sum r_i N_i (N_i)_{p_i}^{-1} \mod N$

5.15.6 Stirling Numbers(permutation $|P| = n$ with k cycles)

$S(n, k) =$ coefficient of x^k in $\Pi_{i=0}^{n-1}(x + i)$

5.15.7 Stirling Numbers(Partition n elements into k non-empty set)

$S(n, k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$

5.15.8 Pick' s Theorem

$A = I + O/2 - 1$

5.15.9 Kirchhoff's theorem

$A_{ii} = deg(i), A_{ij} = (i, j) \in E \ ? -1 : 0$, Deleting any one row, one column, and cal the det(A)

6 Geometry

6.1 Intersection of two circles

```
vector<Pt> interCircle(Pt o1, D r1, Pt o2, D r2) {
  if( norm(o1 - o2) > r1 + r2 ) return {};
  if( norm(o1 - o2) < max(r1, r2) - min(r1, r2) )
      return {};
  D d2 = (o1 - o2) * (o1 - o2);
  D d = sqrt(d2);
  if(d > r1 + r2) return {};
  Pt u = (o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2));
  D A = sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2+d));
  Pt v = Pt(o1.Y - o2.Y, -o1.X + o2.X) * A / (2*d2);
  return {u+v, u-v};
}
```

6.2 Intersection of two lines

```
Pt interPnt(Pt p1, Pt p2, Pt q1, Pt q2){
  double f1 = (p2 - p1) ^ (q1 - p1);
  double f2 = (p2 - p1) ^ (p1 - q2);
  double f = (f1 + f2);
  if( fabs( f ) < eps ) return Pt( nan(""), nan("") );
  return q1 * (f2 / f) + q2 * (f1 / f);
}
```

6.3 Intersection of two segments

```
int ori(const Pt &o, const Pt &a, const Pt &b) {
  LL ret = (a - o) ^ (b - o);
  return (ret > 0) - (ret < 0);
}
// p1 == p2 || q1 == q2 need to be handled
bool banana(const Pt &p1, const Pt &p2,
            const Pt &q1, const Pt &q2) {
  if( ( (p2 - p1) ^ (q2 - q1) ) == 0 ) { // parallel
    if( ori(p1, p2, q1) ) return false;
    return ( (p1 - q1) * (p2 - q1) ) <= 0 ||
           ( (p1 - q2) * (p2 - q2) ) <= 0 ||
           ( (q1 - p1) * (q2 - p1) ) <= 0 ||
           ( (q1 - p2) * (q2 - p2) ) <= 0;
  }
  return (ori(p1, p2, q1) * ori(p1, p2, q2) <= 0) &&
         (ori(q1, q2, p1) * ori(q1, q2, p2) <= 0);
}
```

6.4 Intersection of circle and line

```
// p1, p2 should not be zero vector
bool Inter(const Pt &p1, const Pt &p2 , Circ &cc) {
  Pt dp = p2 - p1;
  double a = dp * dp;
  double b = 2 * ( dp * (p1 - cc.O) );
  double c = cc.O * cc.O + p1 * p1 - 2 * ( cc.O * p1 )
      - cc.R * cc.R;
  double bb4ac = b * b - 4 * a * c;
  return !( fabs( a ) < eps || bb4ac < 0 );
}
```

6.5 Intersection of circle and line

```
bool Inter(Pt p1, Pt p2, Circ cc) {
    D d1 = norm(cc.O - p1);
    D d2 = norm(cc.O - p2);
    if (min(d1, d2) <= cc.R - eps) return true;
    if ( ((cc.O - p1) * (p2 - p1)) < 0 ) return false;
    if ( ((cc.O - p2) * (p1 - p2)) < 0 ) return false;
    Pt d3 = cc.O - p1;
    Pt d4 = (p2 - p1) / norm(p2 - p1) ;
    return fabs(d3 ^ d4) < cc.R;
}
```

6.6 Intersection of circle and polygon

```
Pt ORI , info[ N ];
D r; int n;
// Divides into multiple triangle, and sum up
// oriented area
D area2(Pt pa, Pt pb){
  if( norm(pa) < norm(pb) ) swap(pa, pb);
  if( norm(pb) < eps ) return 0;
  D S, h, theta;
  D a = norm( pb ), b = norm( pa ), c = norm(pb - pa);
  D cosB = (pb * (pb - pa)) / a / c, B = acos(cosB);
  D cosC = (pa * pb) / a / b, C = acos(cosC);
  if(a > r){
    S = (C/2)*r*r;
    h = a*b*sin(C)/c;
    if (h < r && B < PI/2) S -= (acos(h/r)*r*r - h*sqrt
        (r*r-h*h));
  }else if(b > r){
    theta = PI - B - asin(sin(B)/r*a);
    S = .5*a*r*sin(theta) + (C-theta)/2*r*r;
  }else S = .5*sin(C)*a*b;
  return S;
}
D area() {
  D S = 0;
  //info[n] = info[0]
  for(int i = 0; i < n; ++i)
    S += abs( area2(info[i], info[i + 1]) ) * sign( (
        info[i] ^ info[i + 1]) );
  return fabs(S);
}
```

6.7 Tangent line of two circles

```
vector<Line> go( const Cir& c1 , const Cir& c2 , int
    sign1 ){
  // sign1 = 1 for outer tang, -1 for inter tang
  vector<Line> ret;
  double d_sq = norm2( c1.O - c2.O );
  if( d_sq < eps ) return ret;
  double d = sqrt( d_sq );
  Pt v = ( c2.O - c1.O ) / d;
  double c = ( c1.R - sign1 * c2.R ) / d;
  if( c * c > 1 ) return ret;
  double h = sqrt( max( 0.0 , 1.0 - c * c ) );
  for( int sign2 = 1 ; sign2 >= -1 ; sign2 -= 2 ){
    Pt n = { v.X * c - sign2 * h * v.Y ,
             v.Y * c + sign2 * h * v.X };
    Pt p1 = c1.O + n * c1.R;
    Pt p2 = c2.O + n * ( c2.R * sign1 );
    if( fabs( p1.X - p2.X ) < eps and
        fabs( p1.Y - p2.Y ) < eps )
      p2 = p1 + perp( c2.O - c1.O );
    ret.push_back( { p1 , p2 } );
  }
  return ret;
}
```

6.8 Circle cover

```
#define N 1021
struct CircleCover{
  int C; Circ c[ N ];
  bool g[ N ][ N ], overlap[ N ][ N ];
  // Area[i] : area covered by at least i circles
  // O(n*nlog(n))
  D Area[ N ];
  void init( int _C ){ C = _C; }
  bool CCinter(Circ &a, Circ &b, Pt &p1, Pt &p2){
    Pt o1 = a.O, o2 = b.O;
    D r1 = a.R , r2 = b.R;
    if (norm(o1 - o2) > r1 + r2) return {};
```

```
  if (norm(o1 - o2) < max(r1, r2) - min(r1, r2))
      return {};
  D d2 = (o1 - o2)*(o1 - o2);
  D d = sqrt(d2);
  if( d > r1 + r2 ) return false;
  Pt u = (o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2))
      ;
  D A = sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2+d)
      );
  Pt v = Pt(o1.Y - o2.Y, -o1.X + o2.X) * A / (2*d2);
  p1 = u + v; p2 = u - v;
  return true;
}
struct Teve {
  Pt p; D ang; int add;
  Teve() {}
  Teve(Pt _a, D _b, int _c) :
    p(_a), ang(_b), add(_c) {}
  bool operator < (const Teve &a) const {
    return ang < a.ang;
  }
}eve[ N * 2 ];
// strict: x = 0, otherwise x = -1
bool disjuct( Circ& a, Circ &b, int x ) {
  return sign( norm(a.O - b.O) - a.R - b.R ) > x;
}
bool contain( Circ& a, Circ &b, int x ) {
  return sign( a.R - b.R - norm(a.O - b.O) ) > x;
}
bool contain(int i, int j){
  /* c[j] is non-strictly in c[i]. */
  return (sign(c[i].R - c[j].R) > 0 ||
          (sign(c[i].R - c[j].R) == 0 && i < j) ) &&
              contain(c[i], c[j], -1);
}
void solve(){
  for(int i = 0; i <= C + 1; i++)
    Area[ i ] = 0;
  for(int i = 0; i < C; i++)
    for(int j = 0; j < C; j++)
      overlap[i][j] = contain(i, j);
  for(int i = 0; i < C; i++)
    for(int j = 0; j < C; j++)
      g[i][j] = !(overlap[i][j] || overlap[j][i] ||
                  disjuct(c[i], c[j], -1));
  for(int i = 0; i < C; i++){
    int E = 0, cnt = 1;
    for(int j = 0; j < C; j++)
      if(j != i && overlap[j][i])
        cnt++;
    for(int j = 0; j < C; j++)
      if(i != j && g[i][j]){
        Pt aa, bb;
        CCinter(c[i], c[j], aa, bb);
        D A = atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X)
            ;
        D B = atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X)
            ;
        eve[E++] = Teve(bb, B, 1);
        eve[E++] = Teve(aa, A, -1);
        if(B > A) cnt++;
      }
    if(E == 0) Area[cnt] += pi * c[i].R * c[i].R;
    else{
      sort(eve , eve + E);
      eve[E] = eve[0];
      for(int j = 0; j < E; j++) {
        cnt += eve[j].add;
        Area[cnt] += (eve[j].p ^ eve[j + 1].p) * 0.5;
        D theta = eve[j + 1].ang - eve[j].ang;
        if (theta < 0) theta += 2.0 * PI;
        Area[cnt] +=
          (theta - sin(theta)) * c[i].R*c[i].R * 0.5;
      }
    }
  }
}
```

6.9 Half plane intersection

```
Pt interPnt(Line l1, Line l2, bool &res){
  Pt p1, p2, q1, q2;
  tie(p1, p2) = l1; tie(q1, q2) = l2;
  double f1 = (p2 - p1) ^ (q1 - p1);
  double f2 = (p2 - p1) ^ (p1 - q2);
  double f = (f1 + f2);
  if(fabs(f) < eps){
    res = 0; return {0, 0};
  }
  res = true;
  return q1 * (f2 / f) + q2 * (f1 / f);
}
bool isin(Line l0, Line l1, Line l2){
  // Check inter(l1, l2) in l0
  bool res; Pt p = interPnt(l1, l2, res);
  return ( (l0.S - l0.F) ^ (p - l0.F) ) > eps;
}
/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (l.S - l.F) ^ (p - l.F) > 0
 */
/* --^-- Line.FI --^-- Line.SE --^-- */
vector<Line> halfPlaneInter(vector<Line> lines){
  int sz = SZ(lines);
  vector<double> ata(sz), ord(sz);
  for(int i = 0; i < sz; i++) {
    ord[i] = i;
    Pt d = lines[i].S - lines[i].F;
    ata[i] = atan2(d.Y, d.X);
  }
  sort( ALL(ord), [&](int i, int j) {
    if(fabs(ata[i] - ata[j]) < eps)
      return ( (lines[i].S - lines[i].F) ^
               (lines[j].S - lines[i].F) ) < 0;
    return ata[i] < ata[j];
  });
  vector<Line> fin;
  for (int i = 0; i < sz; i++)
    if (!i || fabs(ata[ord[i]] - ata[ord[i-1]]) > eps)
      fin.pb(lines[ord[i]]);
  deque<Line> dq;
  for (int i = 0; i < SZ(fin); i++) {
    while (SZ(dq) >= 2 && !isin(fin[i], dq[SZ(dq) - 2],
          dq[SZ(dq) - 1]))
      dq.pop_back();
    while (SZ(dq) >= 2 && !isin(fin[i], dq[0], dq[1]))
      dq.pop_front();
    dq.pb(fin[i]);
  }
  while (SZ(dq) >= 3 && !isin(dq[0], dq[SZ(dq) - 2], dq
      [SZ(dq) - 1]))
    dq.pop_back();
  while (SZ(dq) >= 3 && !isin(dq[SZ(dq) - 1], dq[0], dq
      [1]))
    dq.pop_front();
  vector<Line> res(ALL(dq));
  return res;
}
```

6.10 Poly union area

```
#define eps 1e-8
class PY{ public:
  int n;
  Pt pt[5];
  Pt& operator[](const int x){ return pt[x]; }
  void input(){
    int n = 4;
    for(int i = 0;i < n; i++)
      scanf("%lf %lf", &pt[i].x, &pt[i].y);
};
```

```
  }
  double getArea(){
    double s = pt[n-1]^pt[0];
    for(int i = 0;i < n-1; i++)
      s += pt[i]^pt[i+1];
    return s/2;
  }
};
PY py[500];
pair<double,int> c[5000];
inline double segP(Pt &p, Pt &p1, Pt &p2) {
  if(SG(p1.x-p2.x) == 0) return (p.y-p1.y)/(p2.y-p1.y);
  return (p.x-p1.x) / (p2.x-p1.x);
}
double polyUnion(int n){
  int i,j,ii,jj,ta,tb,r,d;
  double z,w,s,sum,tc,td;
  for(i=0;i<n;i++) py[i][py[i].n]=py[i][0];
  sum=0;
  for(i=0;i<n;i++){
    for(ii=0;ii<py[i].n;ii++){
      r=0;
      c[r++]=make_pair(0.0,0);
      c[r++]=make_pair(1.0,0);
      for(j=0;j<n;j++){
        if(i==j) continue;
        for(jj=0;jj<py[j].n;jj++){
          ta=SG(tri(py[i][ii],py[i][ii+1],py[j][jj]));
          tb=SG(tri(py[i][ii],py[i][ii+1],py[j][jj+1]))
              ;
          if(ta==0 && tb==0){
            if((py[j][jj+1]-py[j][jj])*(py[i][ii+1]-py[
                i][ii])>0 && j<i){
              c[r++]=make_pair(segP(py[j][jj],py[i][ii
                  ],py[i][ii+1]),1);
              c[r++]=make_pair(segP(py[j][jj+1],py[i][
                  ii],py[i][ii+1]),-1);
            }
          }else if(ta>=0 && tb<0){
            tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
            td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
            c[r++]=make_pair(tc/(tc-td),1);
          }else if(ta<0 && tb>=0){
            tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
            td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
            c[r++]=make_pair(tc/(tc-td),-1);
          }
        }
      }
      sort(c,c+r);
      z=min(max(c[0].first,0.0),1.0);
      d=c[0].second; s=0;
      for(j=1;j<r;j++){
        w=min(max(c[j].first,0.0),1.0);
        if(!d) s+=w-z;
        d+=c[j].second; z=w;
      }
      sum+=(py[i][ii]^py[i][ii+1])*s;
    }
  }
  return sum/2;
}
int main(){
  int n,i,j,k;
  double sum,ds;
  int n; scanf("%d", &n); sum = 0;
  for (int i = 0; i < n; i++) {
    py[i].input();
    ds = py[i].getArea();
    if(ds<0){
      for(j=0,k=py[i].n-1;j<k;j++,k--) swap(py[i][j],
          py[i][k]);
      ds=-ds;
    } sum+=ds;
  } printf("%.9f\n",sum/polyUnion(n));
}
```

6.11 2D Convex hull

```
double cross(Pt o, Pt a, Pt b) {
  return (a - o) ^ (b - o);
}
vector<Pt> convex_hull(vector<Pt> pt) {
  sort( ALL(pt) );
  int top = 0;
  vector<Pt> stk(2*SZ(pt));
  for (int i = 0; i < SZ(pt); i++) {
    while (top >= 2 && cross(stk[top-2], stk[top-1], pt
        [i])<= 0)
      top--;
    stk[top++] = pt[i];
  }
  for (int i = SZ(pt) - 2, t = top + 1; i >= 0; i--) {
    while (top >= t && cross(stk[top-2], stk[top-1], pt
        [i]) <= 0)
      top--;
    stk[top++] = pt[i];
  }
  stk.resize(top-1);
  return stk;
}
```

6.12 Convex hull trick

```
/* Given a convexhull, answer querys in O(\lg N)
CH should not contain identical points, the area should
be > 0, min pair(x, y) should be listed first */
double det( const Pt& p1 , const Pt& p2 )
{ return p1.X * p2.Y - p1.Y * p2.X; }
struct Conv{
  int n;
  vector<Pt> a;
  vector<Pt> upper, lower;
  Conv(vector<Pt> _a) : a(_a){
    n = a.size();
    int ptr = 0;
    for(int i=1; i<n; ++i) if (a[ptr] < a[i]) ptr = i;
    for(int i=0; i<=ptr; ++i) lower.push_back(a[i]);
    for(int i=ptr; i<n; ++i) upper.push_back(a[i]);
    upper.push_back(a[0]);
  }
  int sign( LL x ){ // fixed when changed to double
    return x < 0 ? -1 : x > 0; }
  pair<LL,int> get_tang(vector<Pt> &conv, Pt vec){
    int l = 0, r = (int)conv.size() - 2;
    for( ; l + 1 < r; ){
      int mid = (l + r) / 2;
      if(sign(det(conv[mid+1]-conv[mid],vec))>0)r=mid;
      else l = mid;
    }
    return max(make_pair(det(vec, conv[r]), r),
               make_pair(det(vec, conv[0]), 0));
  }
  void upd_tang(const Pt &p, int id, int &i0, int &i1){
    if(det(a[i0] - p, a[id] - p) > 0) i0 = id;
    if(det(a[i1] - p, a[id] - p) < 0) i1 = id;
  }
  void bi_search(int l, int r, Pt p, int &i0, int &i1){
    if(l == r) return;
    upd_tang(p, l % n, i0, i1);
    int sl=sign(det(a[l % n] - p, a[(l + 1) % n] - p));
    for( ; l + 1 < r; ) {
      int mid = (l + r) / 2;
      int smid=sign(det(a[mid%n]-p, a[(mid+1)%n]-p));
      if (smid == sl) l = mid;
      else r = mid;
    }
    upd_tang(p, r % n, i0, i1);
  }
  int bi_search(Pt u, Pt v, int l, int r) {
    int sl = sign(det(v - u, a[l % n] - u));
    for( ; l + 1 < r; ) {
```

```
      int mid = (l + r) / 2;
      int smid = sign(det(v - u, a[mid % n] - u));
      if (smid == sl) l = mid;
      else r = mid;
    }
    return l % n;
  }
  // 1. whether a given point is inside the CH
  bool contain(Pt p) {
    if (p.X < lower[0].X || p.X > lower.back().X)
        return 0;
    int id = lower_bound(lower.begin(), lower.end(), Pt
        (p.X, -INF)) - lower.begin();
    if (lower[id].X == p.X) {
      if (lower[id].Y > p.Y) return 0;
    }else if(det(lower[id-1]-p,lower[id]-p)<0)return 0;
    id = lower_bound(upper.begin(), upper.end(), Pt(p.X
        , INF), greater<Pt>()) - upper.begin();
    if (upper[id].X == p.X) {
      if (upper[id].Y < p.Y) return 0;
    }else if(det(upper[id-1]-p,upper[id]-p)<0)return 0;
    return 1;
  }
  // 2. Find 2 tang pts on CH of a given outside point
  // return true with i0, i1 as index of tangent points
  // return false if inside CH
  bool get_tang(Pt p, int &i0, int &i1) {
    if (contain(p)) return false;
    i0 = i1 = 0;
    int id = lower_bound(lower.begin(), lower.end(), p)
        - lower.begin();
    bi_search(0, id, p, i0, i1);
    bi_search(id, (int)lower.size(), p, i0, i1);
    id = lower_bound(upper.begin(), upper.end(), p,
        greater<Pt>()) - upper.begin();
    bi_search((int)lower.size() - 1, (int)lower.size()
        - 1 + id, p, i0, i1);
    bi_search((int)lower.size() - 1 + id, (int)lower.
        size() - 1 + (int)upper.size(), p, i0, i1);
    return true;
  }
  // 3. Find tangent points of a given vector
  // ret the idx of vertex has max cross value with vec
  int get_tang(Pt vec){
    pair<LL, int> ret = get_tang(upper, vec);
    ret.second = (ret.second+(int)lower.size()-1)%n;
    ret = max(ret, get_tang(lower, vec));
    return ret.second;
  }
  // 4. Find intersection point of a given line
  // return 1 and intersection is on edge (i, next(i))
  // return 0 if no strictly intersection
  bool get_intersection(Pt u, Pt v, int &i0, int &i1){
   int p0 = get_tang(u - v), p1 = get_tang(v - u);
   if(sign(det(v-u,a[p0]-u))*sign(det(v-u,a[p1]-u))<0){
     if (p0 > p1) swap(p0, p1);
     i0 = bi_search(u, v, p0, p1);
     i1 = bi_search(u, v, p1, p0 + n);
     return 1;
   }
   return 0;
  }
};
```

6.13 KDTree (Nearest Point)

```
struct KDTree {
  static const int MXN = (int)1e5 + 7;
  struct Node {
    int x, y, x1, y1, x2, y2;
    int id, f;
    Node *L, *R;
  }tree[MXN];
  int n;
  Node *root;
  LL dis2(int x1, int y1, int x2, int y2) {
    LL dx = x1 - x2;
    LL 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<PII> ip) {
    n = SZ(ip);
    for (int i = 0; i < n; i++) {
      tree[i].id = i;
      tree[i].x = ip[i].F;
      tree[i].y = ip[i].S;
    }
    root = build_tree(0, n-1, 0);
  }
  Node *build_tree(int L, int R, int dep) {
    if (L > R) return NULL;
    int M = (L + R) >> 1;
    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, LL d2){
    LL 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, LL &md2){
    if (!r || !touch(r, x, y, md2)) return;
    LL 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;
    LL d2 = 102938475612345678LL;
    nearest(root, x, y, id, d2);
    return id;
  }
}tree;
```

6.14 Triangle

```
Pt inCenter( Pt &A,  Pt &B,  Pt &C) { // 內心
  double a = norm(B-C), b = norm(C-A), c = norm(A-B);
  return (A * a + B * b + C * c) / (a + b + c);
}
Pt circumCenter( Pt &a,  Pt &b,  Pt &c) { // 外心
  Pt bb = b - a, cc = c - a;
  double db=norm2(bb), dc=norm2(cc), d=2*(bb ^ cc);
  return a-Pt(bb.Y*dc-cc.Y*db, cc.X*db-bb.X*dc) / d;
}
Pt othroCenter( Pt &a,  Pt &b,  Pt &c) { // 垂心
  Pt ba = b - a, ca = c - a, bc = b - c;
  double 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 Pt(x0, y0);
}
```

7 Stringology

7.1 SAIS

```
struct SA{
#define REP(i,n) for ( int i=0; i<int(n); i++ )
#define REP1(i,a,b) for ( int i=(a); i<=int(b); i++ )
    static const int MXN = 300010;
    bool _t[MXN*2];
    int _s[MXN*2], _sa[MXN*2], _c[MXN*2], x[MXN], _p[
        MXN], _q[MXN*2], hei[MXN], r[MXN];
    int operator [] (int i){ return _sa[i]; }
    void build(int *s, int n, int m){
        memcpy(_s, s, sizeof(int) * n);
        sais(_s, _sa, _p, _q, _t, _c, n, m);
        mkhei(n);
    }
    void mkhei(int n){
        REP(i,n) r[_sa[i]] = i;
        hei[0] = 0;
        REP(i,n) if(r[i]) {
            int ans = i>0 ? max(hei[r[i-1]] - 1, 0) :
                0;
            while(_s[i+ans] == _s[_sa[r[i]-1]+ans]) ans
                ++;
            hei[r[i]] = ans;
        }
    }
    void sais(int *s, int *sa, int *p, int *q, bool *t,
         int *c, int n, int z){
        bool uniq = t[n-1] = true, neq;
        int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s +
             n, lst = -1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))
#define MAGIC(XD) MS0(sa, n); \
        memcpy(x, c, sizeof(int) * z); \
        XD; \
        memcpy(x + 1, c, sizeof(int) * (z - 1)); \
        REP(i,n) if(sa[i] && !t[sa[i]-1]) sa[x[s[sa[i
            ]-1]]++] = sa[i]-1; \
        memcpy(x, c, sizeof(int) * z); \
        for(int i = n - 1; i >= 0; i--) if(sa[i] && t[
            sa[i]-1]) sa[--x[s[sa[i]-1]]] = sa[i]-1;
        MS0(c, z);
        REP(i,n) uniq &= ++c[s[i]] < 2;
        REP(i,z-1) c[i+1] += c[i];
        if (uniq) { REP(i,n) sa[--c[s[i]]] = i; return;
             }
        for(int i = n - 2; i >= 0; i--) t[i] = (s[i]==s
            [i+1] ? t[i+1] : s[i]<s[i+1]);
        MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1]) sa[--x[
            s[i]]]=p[q[i]=nn++]=i);
        REP(i, n) if (sa[i] && t[sa[i]] && !t[sa[i]-1])
             {
            neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i
                ]]+1]-sa[i])*sizeof(int));
            ns[q[lst=sa[i]]]=nmxz+=neq;
        }
        sais(ns, nsa, p + nn, q + n, t + n, c + z, nn,
            nmxz + 1);
        MAGIC(for(int i = nn - 1; i >= 0; i--) sa[--x[s
            [p[nsa[i]]]]] = p[nsa[i]]);
    }
}sa;

void suffix_array(int* ip, int len) {
    // should padding a zero in the back
    // s is int array, n is array length
    // s[0..n-1] != 0, and s[n] = 0
    // resulting SA will be length n+1
    ip[len++] = 0;
    sa.build(ip, len, 128);
    // original 1-base
    for (int i=0; i<len; i++) {
        hei[i] = sa.hei[i + 1];
        sa[i] = sa._sa[i + 1];
    }
}
```

7.2 SAM

```
const int MAXM = 1000010;
struct SAM{
  int tot, root, lst, mom[MAXM], mx[MAXM];
  int acc[MAXM], nxt[MAXM][33];
  int newNode(){
    int res = ++tot;
    fill(nxt[res], nxt[res]+33, 0);
    mom[res] = mx[res] = acc[res] = 0;
    return res;
  }
  void init(){
    tot = 0;
    root = newNode();
    mom[root] = 0, mx[root] = 0;
    lst = root;
  }
  void push(int c){
    int p = lst;
    int np = newNode();
    mx[np] = mx[p]+1;
    for(; p && nxt[p][c] == 0; p = mom[p])
      nxt[p][c] = np;
    if(p == 0) mom[np] = root;
    else{
      int q = nxt[p][c];
      if(mx[p]+1 == mx[q]) mom[np] = q;
      else{
        int nq = newNode();
        mx[nq] = mx[p]+1;
        for(int i = 0; i < 33; i++)
          nxt[nq][i] = nxt[q][i];
        mom[nq] = mom[q];
        mom[q] = nq;
        mom[np] = nq;
        for(; p && nxt[p][c] == q; p = mom[p])
          nxt[p][c] = nq;
      }
    }
    lst = np;
  }
  void push(char *str){
    for(int i = 0; str[i]; i++)
      push(str[i]-'a'+1);
  }
} sam;
```

7.3 Aho-Corasick Algorithm

```
struct ACautomata{
  struct Node{
    int cnt,dp;
    Node *go[26], *fail;
    Node (){
      cnt = 0;
      dp = -1;
      memset(go,0,sizeof(go));
      fail = 0;
    }
  };

  Node *root, pool[1048576];
  int nMem;

  Node* new_Node(){
    pool[nMem] = Node();
    return &pool[nMem++];
  }
  void init(){
    nMem = 0;
    root = new_Node();
  }
  void add(const string &str){
    insert(root,str,0);
  }
  void insert(Node *cur, const string &str, int pos){
    if (pos >= (int)str.size()){
      cur->cnt++;
      return;
    }
    int c = str[pos]-'a';
    if (cur->go[c] == 0){
      cur->go[c] = new_Node();
    }
    insert(cur->go[c],str,pos+1);
  }
  void make_fail(){
    queue<Node*> que;
    que.push(root);
    while (!que.empty()){
      Node* fr=que.front();
      que.pop();
      for (int i=0; i<26; i++){
        if (fr->go[i]){
          Node *ptr = fr->fail;
          while (ptr && !ptr->go[i]) ptr = ptr->fail;
          if (!ptr) fr->go[i]->fail = root;
          else fr->go[i]->fail = ptr->go[i];
          que.push(fr->go[i]);
        }
      }
    }
  }
};
```

7.4 KMP

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

void build_fail_function(string B, int *fail) {
    int len = B.length(), pos;
    pos = fail[0] = -1;
    for (int i = 1; i < len; i ++) {
        while (pos != -1 and B[pos + 1] != B[i])
            pos = fail[pos];
        if (B[pos + 1] == B[i]) pos ++;
        fail[i] = pos;
    }
}

void match(string A, string B, int *fail) {
    int lenA = A.length(), lenB = B.length();
    int pos = -1;
    for (int i = 0; i < lenA; i ++) {
        while (pos != -1 and B[pos + 1] != A[i])
            pos = fail[pos];

        if (B[pos + 1] == A[i]) pos ++;

        if (pos == lenB - 1) {
            // Match ! A[i - lenB + 1, i] = B
            pos = fail[pos];
        }
    }
}
```

7.5 Z value

```
void Z_value(char *s, int len, int *z) {
  int i,j,left,right;
  left=right=0; z[0]=len;
  for(i=1;i<len;i++) {
    j=max(min(z[i-left],right-i),0);
    for(;i+j<len&&s[i+j]==s[j];j++);
    z[i]=j;
    if(i+z[i]>right) {
      right=i+z[i];
      left=i;
    }
  }
}
```

7.6 Z value (palindrome ver.)

```
// z[i] means that the longest odd palindrom centered
    at
// i is [i-z[i] .. i+z[i]]
int len, zv[MAX*2];
char ip[MAX], op[MAX*2];
int main(){
  cin >> ip; len = strlen(ip);
  int l2 = len*2 - 1;
  for(int i=0; i<l2; i++)
    if(i&1) op[i] = '@';
    else op[i] = ip[i/2];
  int l=0, r=0; zv[0] = 1;
  for(int i=1; i<l2; i++){
    if( i > r ){
      l = r = i;
      while( l>0 && r<l2-1 && op[l-1] == op[r+1] )
        l --, r ++;
      zv[i] = (r-l+1);
    }else{
      int md = (l+r)/2, j = md + md - i;
      zv[i] = zv[j];
      int q = zv[i] / 2, nr = i + q;
      if( nr == r ){
        l = i + i - r;
        while( l>0 && r<l2-1 && op[l-1] == op[r+1] )
          l --, r ++;
        zv[i] = r - l + 1;
      }else if( nr > r )
        zv[i] = (r - i) * 2 + 1;
    }
  }
}
```

7.7 Lexicographically Smallest Rotation

```
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);
}
```