0xE Team Reference

FFT

```
typedef complex<double> Complex;
const double pi = acos(-1.0);
void FFT(Complex P[], int n, int oper) {
    for (int i = 1, j = 0; i < n - 1; i++) {
        for (int s = n; j ^= s >>= 1, ~j & s;);
        if (i < j) swap(P[i], P[j]);
    }
    Complex wn, w;
    for (int d = 0; (1 << d) < n; d++) {
        int m = 1 << d, m2 = m << 1;
        double p0 = pi / m * oper;
        wn = Complex(cos(p0), sin(p0));
        for (int i = 0; i < n; i += m2) {
            w = 1;
            rp(j,m) {
                Complex &P1 = P[i + j + m], &P2 = P[i + j];
                Complex t = w * P1;
                P1 = P2 - t; P2 = P2 + t; w = w * wn;
            } } }
    if (oper == -1) rp(i,n) P[i] /= n;
}
// Complex aa[...], bb[...], cc[...];
// aa e bb devem ter o mesmo tamanho k (pot de 2), completa com 0
// while (k < n+m) k<<=1;
// FFT(aa,k,1) e FFT(bb,k,1), cc[i] = aa[i]*bb[i], FFT(cc,k,-1)
//resp em cc[i].real()
```

NTT *(depends on FFT)*

```
// n = 2 ** shift
// mod = n * k + 1
// root = generator(mod)
// wn = root ** k
// if (oper == -1) wn = inv(wn)
// Mod = 734003, root = 5
// Mod = 924844033, root = 5
// Mod = 2130706433, root = 3

int generator (int p) {
  vector<int> fact;
  int phi = p-1,  n = phi;
  for (int i=2; i*i<=n; ++i) {
    if (n % i == 0) {
      fact.push_back (i);
      while (n % i == 0) n /= i;
    } }
  if (n > 1) fact.push_back (n);
  for (int res=2; res<=p; ++res) {
    bool ok = true;
    for (size_t i=0; i<fact.size() && ok; ++i) {
      ok &= powmod (res, phi / fact[i], p) != 1; }
    if (ok)  return res;
  }
  return -1; }
```

Discrete Logarithm

```
// Given 3 positive integers x, z and k,
//find the smallest non-negative integer y,
//such that k%z = (x^y)%z. expr(x, y) = x^y
pii p[1 << 19];
int dis_log(int x, int k, int z) {
  k %= z;
  if (x % z == k) return 1;
  int raiz = (int) sqrt(z)+1;
  int n = z / raiz + 1;
  int xr = expr(x,raiz);
  int xa = 1;
  int res = 1 << 30;
  if (k % __gcd(x,z)) return -1;
  rp(i,n)
    p[i] = pii(xa, i*raiz),
    xa =(ll)xa*xr%z;
  sort(p, p+n);
  xa = k;
  rp(i, raiz + 100) {
    int q = lower_bound(p, p+n, pii(xa, 0)) - p;
    if (q < n && p[q].st == xa && p[q].nd >= i
        && expr(x, p[q].nd - i) == k) {
      res = min(res, p[q].nd - i);
    }
    xa = (ll)xa*x%z;
  }
  if (res == (1 << 30)) return -1;
  else return res; }
```

Preprocess Totients

```
for(int i = 1; i<N; i+=1) phi[i] = i;
for(int i = 2; i<N; i+=2) phi[i] >>= 1;
for(int j = 3; j<N; j+=2) if (phi[j] == j) {
  phi[j]--;
  for(int i = j+j; i<N; i+=j)
    phi[i] = phi[i]/j*(j-1);
}
```

Fast Sieve

```
#define MAXSIEVE 100000000
#define MAXSIEVEHALF (MAXSIEVE/2)
#define MAXSQRT 5001 // sqrt(MAXSIEVE)/2
char a[MAXSIEVE/16+2];
#define isprime(n) (a[(n)>>4]&(1<<(((n)>>1)&7)))
void crivo() {
  cl(a,0xff);
  a[0]=0xFE;
  for(int i=1;i<MAXSQRT;i++)
  if (a[i>>3]&(1<<(i&7)))
  if for(register int
j=(i*(i+1))<<1;j<MAXSIEVEHALF;j+=i+i+1)
  a[j>>3]&=~(1<<(j&7));
}
```

Extended Euclides

```
pii euclides(int a, int b) {
  if (b==0) return mp(1,0);
  pii rec = euclides(b, a%b);
  return mp(rec.nd, rec.st - (a/b) *
rec.nd); }
int invMod(int a, int n) {
  return euclides(a,n).st;
}
```

Modular Inverse (<O(N), O(1)>)

```
inv[1] = 1;
fr(i,2,maxn) {
  inv[i] = (MOD - (MOD / i) * inv[MOD % i]
% MOD) % MOD;
}
```

Chinese Remainder *(depends on Modular Inverse)*

```
int chinese(int n, int *a, int *p) {
  int M = 1, x = 0;
  rp(i,n) M *= p[i];
  rp(i,n) x += a[i] * invMod(M/p[i],p[i]) *
(M/p[i]);
  return (((x % M) + M) % M);
}
```

Floyd’s Cycle-Finding

```
int f(int x) { return (Z * x + I) % M; }

ii floydCycleFinding(int x0) {
  // 1st part: finding k*mu, hare is faster
  int tortoise = f(x0), hare = f(f(x0));
  while (tortoise != hare) {
    tortoise = f(tortoise); hare =
f(f(hare));
  }
  // 2nd part: finding mu, same speed
  int mu = 0; hare = x0;
  while (tortoise != hare) {
    tortoise = f(tortoise); hare = f(hare);
mu++;
  }
  // 3rd part: finding lambda, just hare
  int lambda = 1; hare = f(tortoise);
  while (tortoise != hare) {
    hare = f(hare); lambda++;
  }
  return ii(mu, lambda);
}
```

Dinic

```
int bfs(int source, int sink) {
  cl(level,-1);
  level[source] = 0;
  int front = 0, size = 0, v;
  fila[size++] = source;
  while (front < size) {
    v = fila[front++];
    for (int i=adj[v];i != -1;i = ant[i]) {
      if (cap[i] && level[to[i]] == -1) {
        level[to[i]] = level[v] + 1;
        fila[size++] = to[i];
      } } }
  return level[sink] != -1;
}
int dfs(int v, int sink, int flow) {
  if (v == sink) return flow;
  int f;
  for (int &i = copy_adj[v]; i != -1; i = ant[i]) {
    if (cap[i] && level[to[i]] == level[v]+1 && (f = dfs(to[i], sink, min(flow,
cap[i])))) {
      cap[i]-=f, cap[i^1]+=f;
      return f;
    }
  }
  return 0;
}
int maxflow(int source, int sink) {
  int ret = 0, flow;
  while (bfs(source, sink)) {
    memcpy(copy_adj, adj, sizeof adj);
    while ((flow = dfs(source, sink, 1<<30))) ret += flow;
  }
  return ret; }
```

Gomory-Hu *(depends on Dinic)*

```
void gomory_hu() {
  cl(parent,0); cl(mincut, 0x3f);
  rp(j,z) memcap[j] = cap[j];
  fr(i,1,n) {
    rp(j,z) cap[j] = memcap[j];
    int f = maxflow(i,parent[i]);
    fr(j,i+1,n) if ((~level[j]) && parent[i] == parent[j]) parent[j] = i;
    mincut[i][parent[i]] = mincut[parent[i]][i] = f;
    rp(j,i) mincut[i][j] = mincut[j][i] = min(f,mincut[parent[i]][j]);
 }}
```

Min-Cost Max-Flow

```
int dist[maxv], pot[maxv], pai[maxv];
set<pii> heap;
void update(int no, int ndist, int p) {
  if(ndist >= dist[no]) return;
  if(dist[no] < inf) heap.erase(pii(dist[no],no));
  dist[no] = ndist, pai[no] = p;
  heap.insert(pii(dist[no],no)); }
pii top() { pii ret = *heap.begin(); heap.erase(heap.begin()); return ret; }
int djikstra(int source, int sink) {
  heap.clear(); memset(dist,inf,sizeof dist);
  update(source,0,-1);
  while(heap.size()) {
    pii p = top();
    for (int i = adj[p.second]; i>=0; i = ant[i]) if (cap[i])
update(to[i],p.first+w[i]+pot[p.second]-pot[to[i]],i);
  }
  return dist[sink] < inf; }
pii mcmf(int source, int sink) {
  //need bellman-ford?
  //memset(pot,0x3f,sizeof pot), pot[source] = 0;
  // for(int k = 0; k < n; k++) for(int i = 0; i < n; i++)
  //  for(int j = adj[i]; j >= 0; j = ant[j]) if(cap[j])
  //    pot[to[j]] = min(pot[to[j]], pot[i] + w[j]);
  memset(pot,0,sizeof pot);
  pii p(0,0); // cost,flow
  while (djikstra(source,sink)) {
    int cost = 0, flow = inf;
    for (int x = sink; x != source; x = from[pai[x]])
      if (cap[pai[x]] < flow) flow = cap[pai[x]];
    for (int x = sink; x != source; x = from[pai[x]])
      cap[pai[x]] -= flow, cap[pai[x]^1] += flow, cost += w[pai[x]]*flow;
    for (int x = 0; x < n; x++) pot[x] += dist[x];
    p.first += cost, p.second += flow;
  }
  return p; }
```

Horse Distance in Infinite Grid

```
int steps(int x, int y) {
  x = abs(x); y = abs(y);
  if (!x && !y) return 0;
  if (x == 1 && y == 0 || x == 0 && y == 1) return 3;
  if (x == 2 && y == 2) return 4;
  int ret;
  if (2 * x < y || 2 * y < x) {
    ret = max(x,y) / 2; if (max(x,y) & 1) ++ret;
  } else {
    ret = (x + y) / 3; if ((x + y) % 3 != 0) ++ret;
  }
  if ((x + y) & 1) ret += (ret % 2 == 0);
  else ret += ret % 2;
  return ret; }
```

Heavy-light Decomposition (edges)

```
const int N = 50009, L = 20;
int an, adj[N], to[N+N], cost[N+N], ant[N+N];

int n, pai[N], h[N], dpai[N], sz[N], pref[N];
void dfs1(int u, int pp, int hh, int d) {
  pai[u] = pp; h[u] = hh;
  dpai[u] = d; sz[u] = 1; pref[u] = -1;
  fre(it, u) if (to[it] != pp) {
    int v = to[it];
    dfs1(v, u, hh+1, cost[it]);
    sz[u] += sz[v];
    if (pref[u] == -1 || sz[v] > sz[pref[u]])
      pref[u] = v;
}}

int *ps, sp[N], p_off, st[N][L];
//simple sparse table from page 22 (but for max)

int pilha[N], top;
int cmp[N], ind[N];
int qcmp, plen[N], head[N], *arr[N], off[N];
void buildhl(int u, int ci) {
  cmp[u] = qcmp; ind[u] = ci;
  pilha[top++] = u;
  if (ci == 0) head[qcmp] = u;
  if (pref[u] == -1) {
    int ln = plen[qcmp] = ci;
    arr[qcmp] = ps; ps += ln;
    off[qcmp] = p_off; p_off += ln;
    rp(i, ln) arr[qcmp][i] = dpai[pilha[i+1]];
    buildST(arr[qcmp], ln, st+off[qcmp]);
    top = 0; qcmp++;
  } else buildhl(pref[u], ci+1);
  fre(it, u) { int v = to[it];
    if (v != pai[u] && v != pref[u])
      buildhl(v, 0);
}}
void build() {
  dfs1(0, -1, 0, 0);
  qcmp = top = 0;
  ps = sp; p_off = 0;
  buildhl(0, 0);
}

int up(int u) {
  return head[cmp[u]] != u? head[cmp[u]]:
    pai[u] == -1? u: pai[u];
}
int lca(int u, int v) {
  while (cmp[u] != cmp[v]) {
    if (pai[u] == -1 ||
      h[up(u)] < h[up(v)]) swap(u, v);
    u = up(u);
  } return h[u] < h[v]? u: v;
}
int query(int u, int v) {// u descends from v
  int cp, r = 0;
  while (cmp[u] != cmp[v]) {
    cp = cmp[u];
    r = max(r, queryST(0, ind[u], st+off[cp]));
    r = max(r, dpai[head[cp]]);
    u = pai[head[cp]];
  } cp = cmp[u];
  r = max(r, queryST(ind[v], ind[u], st+off[cp]));
  return r;
}
```

Centroid Decomposition

```
const int N = 100009, L = 19;
int n, rank[N], sz[N], dist[L][N], head[N], pd[N];
int psize(int u, int pai) {
  sz[u] = 1;
  fre(it, u) { int v = to[it];
    if (v != pai && rank[v] == -1)
      sz[u] += psize(v, u);
  } return sz[u];
}
void pdist(int u, int pai, int d, int r) {
  dist[r][u] = d;
  fre(it, u) { int v = to[it];
    if (v != pai && rank[v] == -1)
      pdist(v, u, d+1, r);
}}

// any node, 0
int c_decomp(int u, int r, int hd) {
  int tot = psize(u, -1), pai = -1;
  while (1) {
    int big = -1;
    fre(it, u) { int v = to[it];
      if (v != pai && rank[v] == -1
        && 2*sz[v] > tot) {
        big = v; break;
    }}
    if (big == -1) break;
    pai = u; u = big;
  }
  rank[u] = r; pd[u] = oo;
  pdist(u, -1, 0, r);
  fre(it, u) { int v = to[it];
    if (rank[v] == -1) c_decomp(v, r+1, u);
  }
  head[u] = hd; return u;
}
// paint vertex U black
void paint(int u) {
  int v = u;
  while (v != -1) {
    pd[v] = min(pd[v], dist[rank[v]][u]);
    v = head[v];
}}
//nearest black vertex to U
int query(int u) {
  int ret = oo, v = u;
  while (v != -1) {
    ret = min(ret, dist[rank[v]][u] + pd[v]);
    v = head[v];
  } return ret;
}
//cl(rank, -1);
//int root = c_decomp(0, 0, -1);
```

Lazy Propagation

```
#define pm(v,b,e) v+v, b, (b+e)/2
#define sm(v,b,e) v+v+1, (b+e)/2, e
ll pd[V], inc[V];
void prop(int v, int b, int e) {
  pd[v] += ll(e-b)*inc[v];
  if (e-b > 1) {
    inc[v+v] += inc[v]; inc[v+v+1] += inc[v];
  } inc[v] = 0;
}
ll query(int v, int b, int e) {
  if (j <= b || e <= i) return 0;
  prop(v, b, e);
  if (i <= b && e <= j) return pd[v];
  return query(pm(v, b, e)) + query(sm(v, b, e)); }
void incr(int v, int b, int e, int add) {
  if (j <= b || e <= i) return prop(v, b, e);
  if (i <= b && e <= j) {
    inc[v] += add;
    return prop(v, b, e);
  } prop(v, b, e);
  incr(pm(v, b, e), add); incr(sm(v, b, e), add);
  pd[v] = pd[v+v] + pd[v+v+1];
}
```

Min (Max) Mean Weight Cycle

```
double karp() {
  rp(i,n) if (~adj[i]) add(n, i, 0);
  ++n;
  rp(i,n) rp(k,n+1) dist[i][k] = inf; // -inf if
maximum
  dist[n - 1][0] = 0;
  fr(k,1,n+1) rp(u,n) if (dist[u][k - 1] != inf) {
// -inf if maximum
    for (int i = adj[u]; ~i; i = ant[i]) {
      int v = to[i];
      dist[v][k] = min(dist[v][k], dist[u][k - 1] +
cost[i]); // max if maximum
    }
  }
  double ans = 1e15; // -1e15 if maximum
  rp(u,n-1) if (dist[u][n] != inf) { // -inf if
maximum
    double w = -1e15; // 1e15 if maximum
    bool ok = false;
    rp(k,n) if (dist[u][k] != inf) { // -inf if
maximum
      ok = true;
      w = max(w, (double)(dist[u][n] -
dist[u][k])/(n - k)); // min if maximum
    }
    if (ok) ans = min(ans, w); // max if maximum
  }
  return ans; }
```

Hackenbush

```
nim(u) = (self_loops[u] & 1) ^ XOR(nim(v) + 1)
```

Range BIT

```
//1-based, [left, right], [1, at], lsone(a) = a&-a
void _update(int i, ll mul, ll add) {
  for(; i <= n; i += lsone(i))
    ftmul[i] += mul, ftadd[i] += add; }
void update(int lf, int rt, ll by) {
  _update(lf, by, -by*(lf-1)); _update(rt, -by,
by*rt); }
ll query(int i) {
  ll mul = 0, add = 0, start = i;
  for(; i > 0; i -= lsone(i))
    mul += ftmul[i], add += ftadd[i];
  return mul*start + add; }
```

Offline Dynamic Connectivity

```
const int N = 50009, M = 50009;
Const int E = 150009, H = 3000009;
int rep[N], sz[N], comp;
bool used[M];

const int MARK_EDGE = 1, SET_REP = 2, INC_SZ = 3;
pii hist[H];
int h;
void save(int type, int a, int b) {
  int st = (type == MARK_EDGE)? -1:
      (type == SET_REP)? a+a+1: a+a;
  hist[h++] = pii(st, b); }
void rollback(int version) {
  while (h > version) {
    pii p = hist[--h];
    if (p.st == -1) {
      used[p.nd] = 0; comp++;
    } else {
      int id = p.st/2;
      if (p.st&1) rep[id] = p.nd;
      else sz[id] -= p.nd;
}}}

void set_rep(int a, int b) {
  save(SET_REP, a, rep[a]); rep[a] = b; }
int find(int a) {
  if (rep[a] == a) return a;
  int r = find(rep[a]);
  if (rep[a] != r) set_rep(a, r);
  return r; }
struct edge {
  int a, b, c, id;
  edge() {}
  edge(int a, int b, int c, int id):
    a(a), b(b), c(c), id(id) {}
  bool operator<(const edge &e) const
    { return c < e.c; }
} edges[M];
//sort(edges, edges+m);
//rp(i, m) pos[edges[i].id] = i;
int pos[M];
void join(int id) {
  edge ed = edges[pos[id]];
  int a = find(ed.a), b = find(ed.b);
  if (a == b) return;
  if (sz[a] > sz[b]) swap(a, b);
  set_rep(a, b);
  sz[b] += sz[a];
  save(INC_SZ, b, sz[a]);
  used[ed.id] = 1; comp--;
  save(MARK_EDGE, 0, ed.id);
}

const int ADD = 0, QUERY = 1, REMOVE = 2;
int bpos[M], epos[M], ec;
pii evt[E];
bool dynamic(int l, int r) {
  if (r-l == 1)
    return (evt[l].st == QUERY && comp == 1);
  int m = (r+l)/2, version = h;
  fr(i, m, r) {
    int tp = evt[i].st, id = evt[i].nd;
    if (tp == REMOVE && bpos[id] < l) join(id);
  }
  if (dynamic(l, m)) return 1;
  rollback(version);
  fr(i, l, m) {
    int tp = evt[i].st, id = evt[i].nd;
    if (tp == ADD && epos[id] >= r) join(id);
  }
  if (dynamic(m, r)) return 1;
  rollback(version);
  return 0; }
void put(int type, int id) {
  if (type == ADD) bpos[id] = ec;
  if (type == REMOVE) epos[id] = ec;
  evt[ec++] = pii(type, id); }

int n, m;
// is there a spanning tree with
// max weight - min weight <= dif?
bool can_connect(int dif) {
  rp(i, n) { rep[i] = i; sz[i] = 1; }
  cl(used, 0); comp = n;
  ec = 0; int p = 0, q = 0;
  while (1) {
    while (q < m && edges[q].c - edges[p].c <= dif)
      put(ADD, edges[q++].id);
    put(QUERY, 0);
    if (q == m) break;
    while (p < q && edges[q].c - edges[p].c > dif)
      put(REMOVE, edges[p++].id);
  }
  while (p < q) put(REMOVE, edges[p++].id);
  h = 0;
  return dynamic(0, ec); }
```

Manacher

```
int manacher() {
  n = strlen(s);
  for(int i = 0, l = 0, r = -1;
      i < n; i++) { // even
    int k = (i > r ? 0 :
      min(even[r + l - i - 1], r - i));
    while( 0<= i-k && i+k+1 < n &&
      s[i-k] == s[i+k+1] ) k++;
    even[i] = k;
    if(i + k > r) l = i - k + 1, r = i + k;
  }
  for(int i = 0, l = 0, r = -1;
      i < n; i++) { // odd
    int k = (i > r ? 0 :
      min(odd[r + l - i],r - i));
    while(0<= i-k-1 && i+k+1<n &&
      s[i-k-1] == s[i+k+1]) k++;
    odd[i] = k;
    if(i + k > r) l = i - k, r = i + k;
  }
  int palindromes = 0;
  for(int i = 0; i < n; i++)
    palindromes += even[i],
      palindromes += (odd[i] + 1);
  return palindromes; }
```

Z-Algorithm

```
fz[0] = n = strlen(str);
for (int i = 1, a = 0, b = 0; i < n; i++) {
  if (a && i + fz[i-a] < b) fz[i] = fz[i-a];
  else {
    int j = min(a ? fz[i-a] : 0, i > b ? 0 :
b-i);
    while (str[i+j] == str[j] && ++j);
    fz[i] = j, a = i, b = i + j;
}}
```

KMP

```
void kmpPreprocess() {
  int i = 0, j = -1; b[0] = -1;
  while (i < m) {
    while (j >= 0 && P[i] != P[j]) j = b[j];
    b[++i] = ++j;
}}
void kmpSearch() {
  int i = 0, j = 0;
  while (i < n) {
    while (j >= 0 && T[i] != P[j]) j = b[j];
    i++; j++;
    if (j == m) printf("%d\n", i - j), j =
b[j]; }}
```

Suffix Array O(N log N)

```
char str[N];
int n, sa[N], rank[N], s[N], r[N], cnt[N], lcp[N];
bool cmp(int a, int b) {
  return str[a] != str[b]? str[a] < str[b]: a > b;
}
void criarSA() {
  n = strlen(str);
  rp(i, n) sa[i] = i;
  sort(sa, sa+n, cmp);
  rp(i, n) rank[i] = str[i];
  for(int l = 1, dif = 1; dif && l < n; l *= 2) {
    memcpy(r, rank, n*sizeof(int));
    rp(i, n) rank[sa[i]] =
      (i > 0 && r[sa[i-1]] == r[sa[i]] &&
      sa[i-1] + l < n &&
      r[sa[i-1] + l/2] == r[sa[i] + l/2])?
      rank[sa[i-1]]: i;
    rp(i, n) cnt[i] = i;
    memcpy(s, sa, n*sizeof(int));
    dif = 0;
    rp(i, n) {
      int s1 = s[i]-l, pos;
      if (s1 >= 0) {
        pos = cnt[rank[s1]]++;
        sa[pos] = s1;
        if (sa[pos] != s[pos]) dif = 1;
}}}}
void criarLCP() {
  int h = 0, j;
  rp(i, n) rank[sa[i]] = i;
  rp(i, n) if (rank[i] > 0) {
    j = sa[rank[i]-1];
    while (i+h < n && j+h < n
      && str[i+h] == str[j+h]) h++;
    lcp[rank[i]] = h;
    if (h > 0) h--;
}}
```

Aho-Corasick

```
struct No {
  int fail, next, adj[CHILDREN], mark;
  vector<ii> out;
  void init() {
    fail = next = -1; mark = 0;
    cl(adj, -1); out.clear(); }
} aho[NODES];
// lembrar de fazer aho[0].init(); no_cnt = 1;
int no_cnt, fila[NODES];
inline int get(char c) { return c - 'a'; }
void add_pattern(char* w, int id) {
  int no = 0, len = 0;
  for (int i = 0; w[i]; ++i, ++len) {
    int to = get(w[i]);
    if (aho[no].adj[to] == -1) {
      aho[no_cnt].init();
      aho[no].adj[to] = no_cnt++;
    }
    no = aho[no].adj[to];
  }
  aho[no].out.push_back(ii(id, len));
}
void build_failure() {
  int ini = 0, fim = 0;
  rp(i,CHILDREN) if (aho[0].adj[i] != -1) {
    int ch = aho[0].adj[i];
    aho[ch].fail = 0;
    aho[ch].next = aho[0].out.size() ? 0 : -1;
    fila[fim++] = ch;
  }
  while (ini < fim) {
    int u = fila[ini++];
    rp(i,CHILDREN) if (aho[u].adj[i] != -1) {
      int v = aho[u].adj[i], f = aho[u].fail;
      fila[fim++] = v;
      while (f != 0 && aho[f].adj[i] == -1) f = aho[f].fail;
      if (aho[f].adj[i] != -1) f = aho[f].adj[i];
      aho[v].fail = f;
      aho[v].next = aho[f].out.size() ? f : aho[f].next;
      // update aho[v] based on aho[f]
    }
  }
}
void search(char* text) {
  int no = 0;
  for (int i = 0; text[i]; ++i) {
    int to = get(text[i]);
    while (no != 0 && aho[no].adj[to] == -1) no = aho[no].fail;
    if (aho[no].adj[to] != -1) no = aho[no].adj[to];
    int aux = no;
    while (aux != -1 /* && aho[aux].mark == 0 */) {
      aho[aux].mark = 1;  // decision
      rp(j,aho[aux].out.size()) {
        found[aho[aux].out[j].st] = passo;
      }
      aux = aho[aux].next;
    }
  } }
```

Convex Hull Trick

```
//b[j] >= b[j+1] (opt: a[i] <= a[i+1])
typedef pair<ll,ll> pll;
//line = st*x + nd, beg of line = st/nd
pll line[N], beg[N];
int pcmp(pll i, pll j) {
  ll ri = i.st*j.nd, rj = j.st*i.nd;
  return ri == rj? 0: ri < rj? -1: 1; }
pll inter(pll u, pll v) {
  ll num = u.nd - v.nd, den = v.st - u.st;
  return den < 0? pll(-num, -den): pll(num, den); }

int h, best, ind[N];//h = best = 0;
void insert(pll p, int i) {
  if (h == 0) beg[h] = pll(-oo, 1);
  else {
    if (p >= line[h-1]) return;
    for(;; h--) {
      pll begi = inter(p, line[h-1]);
      if (pcmp(beg[h-1], begi) < 0) {
        beg[h] = begi; break;
  }}} line[h] = p; ind[h++] = i; }
int query(ll x) {
  while (best < h-1 &&
    pcmp(beg[best+1], pll(x, 1)) <= 0) best++;
  return best;
}

//pd[0] = 0, ans = pd[n-1]
//pd[i] = min(j < i) (b[j]*a[i] + pd[j])
int n, a[N], b[N];
ll pd[N];
void process() {
  h = best = 0;
  pd[0] = 0;
  insert(pll(b[0], pd[0]), 0);
  fr(i, 1, n) {
    int j = query(a[i]);
    pd[i] = line[j].st*a[i] + line[j].nd;
    insert(pll(b[i], pd[i]), i);
}}

/*pd[i][j] = min(j < k < n)
(pd[i-1][k] + acc[j] - acc[k] - (k-j)*sum[k])
pd[i][j] = min(j < k < n)
(pd[i-1][k] - acc[k] - k*sum[k]) + j*sum[k] + acc[j]
ans = pd[K][0]*/
int n, K, w[N];
ll pd[15][N], acc[N], sum[N];
void process() {
  rp(j, n) pd[1][j] = acc[j];
  fr(i, 2, K+1) {
    h = best = 0;
    rp(k, n) insert(pll(sum[k],
        pd[i-1][k] - (acc[k] + k*sum[k])), k);
    rp(j, n) {
      while (best < h-1 && ind[best] <= j) best++;
      int k = query(j);
      pd[i][j] = acc[j] + line[k].st*j + line[k].nd;
}}}

//----------------------online---------------------

int h;//h = 0; cnj.clear();
pll inter(int i, int j) {
  return inter(line[i], line[j]); }
bool inserting;
bool cmp(int i, int j) {
  if (inserting) return line[i] > line[j];
  else return pcmp(beg[i], beg[j]) < 0; }
set<int, bool(*)(int, int)> cnj(cmp);

void insert(ll a, ll b) {
  inserting = true;
  int i = h++;
  line[i] = pll(a, ll(oo)*oo);
  auto i1 = cnj.lower_bound(i);
  if (i1 != cnj.end() && line[*i1].st == a) {
    if (line[*i1].nd <= b) return;
    cnj.erase(i1); }
  --WARNING: new line still can be rejected
  line[i] = pll(a, b);
  auto i2 = i1 = cnj.lower_bound(i);
  while (i1 != cnj.begin()) { i1--;
    if (pcmp(beg[*i1], inter(*i1, i)) < 0) {
      i1++; break;
  }}
  if (i2 != cnj.end()) { i2++;
    while (i2 != cnj.end() &&
      pcmp(beg[*i2], inter(i, *i2)) <= 0) i2++;
    i2--; }
  cnj.erase(i1, i2); i1 = i2 = cnj.insert(i).st;
  beg[i] = (i1 == cnj.begin())? pll(-oo, 1):
    inter(*(--i1), i);
  if (++i2 != cnj.end()) beg[*i2] = inter(i, *i2);
}
int query(ll x) {//cnj cant be empty
  inserting = false;
  const int POS = N-3; beg[POS] = pll(x, 1);
  auto i1 = cnj.upper_bound(POS);
  return i1 == cnj.begin()? *i1: *(--i1); }
```

Simplex

```
typedef long double ld;
#define N 21
#define M 21
#define K M+N+1
ld eps = 1e-9, a[M][K], b[M], c[K], res[N];
int n, m, kt[M], nn[K];
inline void pivot(int k, int l, int e) {
  int x = kt[l]; ld p = a[l][e];
  rp(i, k) a[l][i] /= p; b[l] /= p; nn[e] = 0;
  rp(i, m) if (i != l)
    b[i] -= a[i][e]*b[l],
    a[i][x] = a[i][e]*-a[l][x];
  rp(j, k) if (nn[j]) {
    c[j] -= c[e]*a[l][j];
    rp(i, m)
      if (i != l) a[i][j] -= a[i][e]*a[l][j];
  }
  kt[l] = e; nn[x] = 1; c[x]=c[e]*-a[l][x];
}
void doit(int k) {
  ld best;
  while (1) {
    int e = -1, l = -1;
    rp(i, k) if (nn[i] && c[i] > eps)
    { e = i; break; }
    if (e == -1) break;
    rp(i, m) if (a[i][e] > eps
      && (l == -1 || best > b[i]/a[i][e]))
      best = b[l=i]/a[i][e];
    if (l == -1) return;// ILIMITADO
    pivot(k, l, e);
  }
  rp(i, k) res[i] = 0;
  rp(i, m) res[kt[i]] = b[i];
}
void simplex(ld aa[M][N], ld bb[M], ld cc[N]) {
  int k = n+m+1;
  memcpy(b, bb, m*sizeof(ld));
  memcpy(c, cc, n*sizeof(ld));
  rp(i, m) memcpy(a[i], aa[i], n*sizeof(ld));
  rp(i, m) {
    a[i][n+i] = 1; a[i][k-1] = -1; kt[i] = n+i;
  }
  rp(i, k) nn[i] = 1;
  rp(i, m) nn[kt[i]] = 0;
  int pos = min_element(b, b+m) - b;
  if (b[pos] < -eps) {
    rp(i, k) c[i] = 0; c[k-1] = -1;
    pivot(k, pos, k-1); doit(k);
    if (res[k-1] > eps) return;// IMPOSSIVEL
    rp(i, m) if (kt[i] == k-1)
```

```
      rp(j, k-1) if (nn[j] &&
        (a[i][j] < -eps || eps < a[i][j])) {
        pivot(k, i, j); break;
      }
    memcpy(b, bb, m*sizeof(ld));
    fr(i, n, k) c[i] = 0;
    rp(i, m) rp(j, k)
      if (nn[j]) c[j] -= c[kt[i]]*a[i][j];
  }
  doit(k-1);
}//maximize c*x, a*x <= b, x >= 0
int main() {
  ld a[M][N], b[M], c[N];
  while (sc2(n, m) == 2) {
    rp(i, n) scanf("%Lf", c+i);
    rp(i, m) {
      rp(j, n) scanf("%Lf", a[i]+j);
      scanf("%Lf", b+i);
    }
    simplex(a, b, c);
    ld ans = 0; rp(i, n) ans += res[i]*c[i];
    printf("Nasa can spend %d taka.\n",
      (int) ceil(ans*m));
  }
  return 0;
}
```

Stoer-Wagner

```
// cl(foi, 0), cl(adjmat, 0)
// rp(i,n-1) res = min(res, mincut());
int adjmat[N][N], mark[N], cap[N], foi[N];
int mincut() {
  int ret, S, T;
  memcpy(mark, foi, sizeof foi);
  rp(i,n) if (!foi[i]) {
    rp(j,n) cap[j] = adjmat[i][j];
    mark[i] = true;
    S = i;
    break;
  }
  while (true) {
    int x, y = 0;
    rp(i,n) if (!mark[i] && cap[i] > y) {
      x = i, y = cap[i];
    }
    if (!y) break;
    ret = y; T = S; S = x;
    mark[S] = true;
    rp(i,n) if (!mark[i] && adjmat[S][i]) {
      cap[i] += adjmat[S][i];
    }
  }
  foi[S] = true;
  rp(i,n) {
    adjmat[i][T] += adjmat[i][S];
    adjmat[T][i] += adjmat[S][i];
  }
  return ret;
}
```

SCC & 2-SAT

```
// memset(adj,-1,sizeof adj); z = 0;
// memset(idx,-1,sizeof idx); ind = 1;
int adj[N], to[E], ant[E], z;
int st[N], idx[N], low[N], comp[N], ind, stp = 0, n,
ncomp = 0;
int dfs(int x) {
  if (~idx[x]) return idx[x] ? idx[x] : ind;
  low[x] = idx[x] = ind++;
  st[stp++] = x;
  for (int w = adj[x] ; ~w ; w = ant[w])
    low[x] = min(low[x], dfs(to[w]));
  if (idx[x] == low[x]) {
    ++ncomp;
    while (idx[st[--stp]] = 0, st[stp] != x) {
      low[st[stp]] = low[x], comp[st[stp]] = ncomp; }
      comp[x] = ncomp; }
  return low[x];
}
bool tarjan() {
  fr(i,0,n) dfs(i);
  // no final, low[v] indica qual o componente de v
  fr(i,0,n) if (low[i] == low[i^1]) return 0;
  return 1; }

// Operações comuns de 2-sat
// traduz de forma que se possa escrever "não v" como
"~v"
#define trad(v) (v<0?((~v)*2)^1:v*2)
void addImp(int a, int b){ add(trad(a), trad(b)); }
void addOr(int a,int b) { addImp(~a,b); addImp(~b,a); }
void addEqual(int a, int b) { addOr(a,~b); addOr(~a,b); }
void addDiff(int a, int b) { addEqual(a,~b); }
// valoração: value[i] = (comp[i] < comp[i + 1]) ? 1 : 0;
```

Digits Counting

```
vi C(int x, int BASE = 10) {
  int a,b,d; vi count = vi(BASE);
  int k = 1, l=x;
  while(x) {
    a=x/BASE, b=x%BASE, d=l%k;
    rp(i,b) count[i]+=k*(a+(i?1:0));
    count[b]+=k*(a-(b?0:1))+d+1;
    fr(i,b+1,BASE) count[i]+=k*a;
    k*=BASE, x/=BASE; }
  return count; }
```

Articulation Points and Bridges

```
//dfsNumberCounter = 0; dfs_num.assign(V, -1);
//dfs_low.assign(V, 0);
//dfs_parent.assign(V, -1); art_vertex.assign(V, 0);

vi dfs_num, dfs_low, art_vertex;
int dfsNumberCounter, dfsRoot, rootChildren;

void artPointAndBridge(int u) {
  dfs_low[u] = dfs_num[u] = dfsNumberCounter++;
  for (int i = adj[u]; ~i; i = ant[i]) {
    int v = to[i];
    if (dfs_num[v] == -1) {
        dfs_parent[v] = u;
        if (u == dfsRoot) rootChildren++;
        artPointAndBridge(v);
        if (dfs_low[v] >= dfs_num[u]) art_vertex[u] =
true;
        if (dfs_low[v] > dfs_num[u]) printf(" Edge (%d,
%d) is a bridge\n", u, v);
        dfs_low[u] = min(dfs_low[u], dfs_low[v]);
    } else if (v != dfs_parent[u])
        dfs_low[u] = min(dfs_low[u], dfs_num[v]);
  }
}

printf("Bridges:\n");
fr(i,0,V) if (dfs_num[i] == -1) {
  dfsRoot = i; rootChildren = 0;
  artPointAndBridge(i);
  art_vertex[dfsRoot] = (rootChildren > 1);
}
printf("art Points:\n");
rp(i,V) if (art_vertex[i]) printf("Vertex %d\n", i);
```

Pape

```
int mate[N], gf[N], n;
bool inner[N];

int func() {
  cl(mate, -1);
  rp(i,n) {
    if (mate[i] != -1) continue;
    for (int k = adj[i]; ~k; k = ant[k]) {
        int j = to[k];
        if (mate[j] != -1) continue;
        mate[i] = j, mate[j] = i;
        break;
    }
  }
```

```
  rp(i,n) {
    if (mate[i] != -1) continue;
    cl(inner, 0);
    queue<int> q;
    q.push(i); inner[i] = 1; gf[i] = -1;
    while (!q.empty()) {
      int u = q.front(), v; q.pop();
      for (int k = adj[u]; ~k; k = ant[k]) {
        int prox = to[k];
        if (inner[prox]) continue;
        if (mate[prox] == -1) {
          while (u >= 0) {
            int old = mate[u];
            mate[u] = prox; mate[prox] = u;
            prox = old; u = gf[u];
          }
          goto next;
        }
        for (v = u; v >= 0; v = gf[v]) {
          if (v == prox) break;
        }
        if (v != prox) {
          inner[prox] = 1;
          q.push(mate[prox]); gf[mate[prox]] = u;
    }}}
    next:;
  }
  int tot = 0;
  rp(i,n) tot += (mate[i] != -1);
  return tot / 2;
}
```

2D Geometry - Primitives

```
double operator^(const PT &q) const {
  return atan2(*this%q,*this*q); }

PT rotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t),
p.x*sin(t)+p.y*cos(t)); }

PT projPtLine(PT a, PT b, PT c) {
  b = b - a; c = c - a;
  return a + b*(c*b)/(b*b); }

PT projPtSeg(PT a, PT b, PT c) {
  b = b - a; c = c - a;
  double r = b * b;
  if (cmp(r) == 0) return a;
  r = c * b / r;
  if (cmp(r,0) < 0) return a;
  if (cmp(r,1) > 0) return a + b;
  return a + b * r; }
```

```
double distPtSeg(PT a, PT b, PT c) {
  return !(c - projPtSeg(a, b, c));
}

// compute distance between point (x,y,z) and plane
ax+by+cz=d
double distPtPlane(double x, double y, double z,
double a, double b, double c, double d) {
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

bool parallel(PT a, PT b, PT c, PT d) {
  return cmp((b-a)%(c-d))==0;
}

bool collinear(PT a, PT b, PT c, PT d) {
  return parallel(a, b, c, d)
      && cmp((a-b)%(a-c))==0
      && cmp((c-d)%(c-a))==0;
}

bool inSeg(PT a, PT b, PT c) {
  return cmp((c-a)%(c-b)) == 0 && cmp((c-a)*(c-b))
<= 0;
}

bool segInter(PT a, PT b, PT c, PT d) {
  if (collinear(a, b, c, d)) {
    if (cmp(!(a-c)) == 0 || cmp(!(a-d)) == 0 ||
        cmp(!(b-c)) == 0 || cmp(!(b-d)) == 0) return
true;
    if (cmp((c-a)*(c-b)) > 0 && cmp((d-a)*(d-b)) > 0
&& cmp((c-b)*(d-b)) > 0)
      return false;
    return true;
  }
  if (cmp(((d-a)%(b-a)) * ((c-a)%(b-a))) > 0) return
false;
  if (cmp(((a-c)%(d-c)) * ((b-c)%(d-c))) > 0) return
false;
  return true;
}

// assume que é única
// para segmentos checar se intersecta
PT lineLine(PT a, PT b, PT c, PT d) {
  b = b - a; d = c - d; c = c - a;
  return a + b * (c % d) / (b % d);
}
```

```
PT circleCenter(PT a, PT b, PT c) {
  b = (a+b) / 2;
  c = (a+c) / 2;
  return lineLine(b, b + rotateCW90(a-b), c, c +
rotateCW90(a-c));
}

vector<PT> circleLine(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b - a;
  a = a - c;
  double A = b*b;
  double B = a*b;
  double C = a*a - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c + a + b*(-B + sqrt(D + EPS)) / A);
  if (D > EPS)
    ret.push_back(c + a + b*(-B - sqrt(D)) / A);
  return ret;
}

double rIncircle(PT a, PT b, PT c) {
  double ab = !(a-b), bc = !(b-c), ca = !(c-a);
  return fabs(((b-a)%(c-a))/(ab+bc+ca));
}

bool segSegIntersect(PT a, PT b, PT c, PT d, PT &p)
{
  if (cmp((d-c)%(b-a)) == 0) return 0;
  p = c + (d-c)*(((b-a)%(c-a))/((d-c)%(b-a)));
  return inSeg(p,a,b) && inSeg(p,c,d); }

vector<PT> circleCircle(PT a, double r, PT b, double
R) {
  vector<PT> ret;
  double d = !(a-b);
  if (d > r + R || d + min(r, R) < max(r, R)) return
ret;
  double x = (d*d - R*R + r*r) / (2*d);
  double y = sqrt(r*r - x*x);
  PT v = (b - a)/d;
  ret.push_back(a + v*x + rotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a + v*x - rotateCCW90(v)*y);
  return ret;
}
```

```
PT centroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * signedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y -
p[j].x*p[i].y);
  }
  return c / scale;
}

typedef pair<PT,double> CIRCLE;
void circleCircleTangents(CIRCLE p, CIRCLE q,
vector< pair<PT,PT> > &vec) {
  double d = !(p.F-q.F);
  double ang = asin((q.S-p.S)/d);
  bool trocou = 0;
  if (cmp(p.S,q.S) < 0) swap(p,q), trocou = 1;
  PT i1, i2;
  if (cmp(d+q.S,p.S) == 0) {
    i1 = p.F + ((q.F-p.F)/d*p.S);
    vec.pb(mp(i1,i1));
  } else if (cmp(d+q.S,p.S) > 0) {
    if (trocou) swap(p,q);
      i1 = p.F + ((q.F-p.F)/d*p.S)[ang+pi/2.0];
      i2 = q.F + ((p.F-q.F)/d*q.S)[ang-pi/2.0];
      vec.pb(mp(i1,i2));
      i1 = p.F + ((q.F-p.F)/d*p.S)[-ang-pi/2.0];
      i2 = q.F + ((p.F-q.F)/d*q.S)[-ang+pi/2.0];
      vec.pb(mp(i1,i2));
   }
}

void circleCircleOpTangents(CIRCLE p, CIRCLE q,
vector< pair<PT,PT> > &vec) {
  double d = !(p.F-q.F);
  double ang = asin((q.S+p.S)/d);
  PT i1, i2;
  if (cmp(d,p.S+q.S) == 0) {
    i1 = p.F + ((q.F-p.F)/d*p.S);
    vec.pb(mp(i1,i1));
  } else if (cmp(d,p.S+q.S) > 0) {
    i1 = p.F + ((q.F-p.F)/d*p.S)[pi/2.0-ang];
    i2 = q.F + ((p.F-q.F)/d*q.S)[pi/2.0-ang];
    vec.pb(mp(i1,i2));
    i1 = p.F + ((q.F-p.F)/d*p.S)[-pi/2.0+ang];
    i2 = q.F + ((p.F-q.F)/d*q.S)[-pi/2.0+ang];
    vec.pb(mp(i1,i2));
}}
```

```
bool circle2PtsRad(PT p1, PT p2, double r, PT &c) {
  double d2 = !(p1 - p2);
  d2 *= d2;
  double det = r * r / d2 - 0.25;
  if (det < 0.0) return false;
  double h = sqrt(det);
  c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
  c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
  return true;
}   // to get the other center, reverse p1 and p2
```

Minimum Enclosing Circle

```
//PT deve ser de doubles!
circle spanning_circle(PT *T, int n) {
  random_shuffle(T, T + n);
  circle C(PT(), 0);
  rp(i,n) if (!in_circle(C, T[i])) {
    C = circle(T[i], 0);
    rp(j,i) if (!in_circle(C, T[j])) {
      C = circle((T[i] + T[j]) / 2, T[i].dist(T[j])
/ 2);
      rp(k,j) if (!in_circle(C, T[k])) {
        PT o = circleCenter(T[i], T[j], T[k]);
        C = circle(o, T[k].dist(o));
      } } }
  return C;
}
```

Convex Hull 2D

```
vector<PT> ConvexHull(vector<PT> P) {
  int n = P.size(), k = 0; vector<PT> H(2*n);
  sort(P.begin(), P.end());
  for (int i = 0; i < n; i++) {
    while (k >= 2 && (H[k-1]-H[k-2])%(P[i]-H[k-2])
<= 0) k--;
    H[k++] = P[i];
  }
  for (int i = n-2, t = k+1; i >= 0; i--) {
    while (k >= t && (H[k-1]-H[k-2])%(P[i]-H[k-2])
<= 0) k--;
    H[k++] = P[i];
  }
  H.resize(k);
  return H;
}
```

Point in Convex Polygon (O(log n))

```
bool ptInSegment(PT a, PT b, PT p) {
  bool x = min(a.x,b.x) <= p.x && p.x <=
max(a.x,b.x);
  bool y = min(a.y,b.y) <= p.y && p.y <=
max(a.y,b.y);
  return x && y && ((b - a) % (p - a) == 0); }

bool ptInsideTriangle(PT p, PT a, PT b, PT c) {
  if ((b - a) % (c - b) < 0) swap(a, b);
  ll x = (b - a) % (p - b);
  ll y = (c - b) % (p - c);
  ll z = (a - c) % (p - a);
  if (x > 0 && y > 0 && z > 0) return true;
  if (!x) return ptInSegment(a,b,p);
  if (!y) return ptInSegment(b,c,p);
  if (!z) return ptInSegment(c,a,p);
  return false; }

bool inPolygon(PT q, const vector<PT> &P) {
  PT pivot = P[0];
  int X = 1, Y = P.size();
  while (Y - X != 1) {
    int Z = (X+Y)/2;
    PT diagonal = pivot - P[Z];
    if(((P[X] - pivot) % (q - pivot))*((q - pivot) %
(P[Z] -pivot)) >= 0) Y = Z;
    else X = Z;
  }
  return ptInsideTriangle(q, P[X], P[Y], pivot); }
```

Intersection of Half-planes

```
bool outside(Line q, Line i, Line j) {
  PT inter = lineLine(i, j);
  double det = (q.q - q.p) % (inter - q.p);
  return cmp(det) < 0; }

bool intersection_is_polygon(double mid) {
  rep(i,n) A[i] = lines[i].move(mid);
  deq[0] = A[0]; deq[1] = A[1];
  int ini = 0, fim = 1;
  fr(i,2,n) {
    while (ini < fim and outside(A[i], deq[fim - 1],
deq[fim])) --fim;
    while (ini < fim and outside(A[i], deq[ini],
deq[ini + 1])) ++ini;
    deq[++fim] = A[i];
  }
  while (ini < fim and outside(deq[ini], deq[fim -
1], deq[fim])) --fim;
  while (ini < fim and outside(deq[fim], deq[ini],
deq[ini + 1])) ++ini;
  return fim - ini > 1; }
```

Cut Polygon with Line

```
// line segment p-q intersect with line A-B.
PT lineIntersectSeg(PT p, PT q, PT A, PT B) {
  double a = B.y - A.y;
  double b = A.x - B.x;
  double c = B.x*A.y - A.x*B.y;
  double u = fabs(a*p.x + b*p.y + c);
  double v = fabs(a*q.x + b*q.y + c);
  return PT((p.x*v+q.x*u) / (u+v), (p.y*v+q.y*u) /
(u+v));
}

// cuts polygon Q along the line formed by point a
-> point b
// (note: the last point must be the same as the
first point)
vector<PT> cutPolygon(PT a, PT b, const vector<PT>&
Q) {
  vector<PT> P;
  fr(i,0,Q.size()) {
    double left1 = (Q[i]-a) % (b-a), left2=0.0;
    if (i!= Q.size()-1) left2 = (Q[i+1]-a) % (b-a);
    if (left1 > -EPS) P.push_back(Q[i]);
    if (left1*left2 < -EPS)
      P.push_back(lineIntersectSeg(Q[i], Q[i+1], a,
b));
  }
  if (P.empty()) return P;
  if (fabs(P.back().x - P.front().x) > EPS ||
      fabs(P.back().y - P.front().y) > EPS)
        P.push_back(P.front());
  return P;
}
```

Lower-bound Dinic *(depends on Dinic)*

```
// N >= num nodes in graph + 2, E >= num edges in
graph + 2 * N

// add arrays from Dinic's algorithm
int low[E], delta[N];

// low[a -> b] = l, low[b -> a] = 0
// other arrays exactly as Dinic
void add_edge(int a, int b, int l, int u);

// add bfs & dns from Dinic
// add maxflow from Dinic as method "dinic"

// sink + 1 & sink + 2 CANNOT BE USED AS VERTICES
// hint: put sink as the vertex with highest index
// actual flow in edge i is cap[i ^ 1] + low[i]
int lb_max_flow(int source, int sink, int sum = 0) {
  memset(delta, 0, sizeof delta);

  for (int i = 0; i < inde; i += 2) {
    delta[to[i]] += low[i];
    delta[to[i ^ 1]] -= low[i];
    cap[i] -= low[i]; }

  // sink = last node
  for (int i = 0; i <= sink; ++i) {
    if (delta[i] > 0) {
      add_edge(sink + 1, i, 0, delta[i]);
      sum += delta[i];
    } else if (delta[i] < 0) {
      add_edge(i, sink + 2, 0, -delta[i]); } }

  // don't add if there is no source or sink
(circulating problem)
  add_edge(sink, souce, 0, INT_MAX);

  int f = dinic(sink + 1, sink + 2);
  if (f != sum) return -1; // impossible

  // if circulating problem (no source and sink)
  // maxflow is equal to f

  // if not circulating problem
  return dinic(source, sink);
}
```

MCBM

```
int match[maxn], vis[maxn];
int aug(int u) {
  if (vis[u]) return 0;
  vis[u] = 1;
  fre(j,u) {
    int v = to[j];
    if (match[v] == -1 || aug(match[v])) {
      match[v] = u;
      return 1;
    }
  }
  return 0;
}
```

Biconnected Components

```
void generateBC(int no){
  while(pilha.top() != no){
        // pilha.top() is in BC
    pilha.pop();
  }
  // no is in BC
  pilha.pop();
}
// reset dfs_num to -1 and art to 0
int dfs_num[maxn], contador, nbc;
bool art[maxn];
int dfs(int u) {
  int low = dfs_num[u] = contador++;
  pilha.push(u);
  int child = 0;
  fre(i,u) {
    int v = to[i];
    if(dfs_num[v] == -1) {
      child++;
      int temp = dfs(v);
      low = min(low,temp);
      if(temp >= dfs_num[u]) {
        if (u > 0) art[u] = 1;
        nbc++;
        pilha.push(u);
        generateBC(v);
      }
    } else if(dfs_num[v] < low) low = dfs_num[v];
  }
  if (u == 0) art[u] = (child > 1);
  return low;
}
```

Hu-Tucker

```
#define PQ(x) priority_queue< x, vector<x>, greater<x> >
PQ(pii) heap;
int cst[MAXN];
PQ(int) hpq[MAXN];
int lef[MAXN], rig[MAXN];
int one(PQ(int) &s) {
  return s.size() < 1 ? INF : s.top(); }
int two(PQ(int) &s) {
  if(s.size() < 2) return INF;
  int r = s.top(); s.pop();
  r += s.top(), s.push(r - s.top());
  return r; }
int merge(int x, int y) {
  rig[y] = rig[x];
  lef[rig[x]] = y;
  if(hpq[y].size() < hpq[x].size())
    swap(hpq[x], hpq[y]);
  while(!hpq[x].empty())
    hpq[y].push( hpq[x].top() ), hpq[x].pop();
  lef[x] = rig[x] = -1;
  return y; }
int tuck(int n, int *A) {
  while(!heap.empty()) heap.pop();
  rp(i,n) while(!hpq[i].empty()) hpq[i].pop();
  rp(i,n) lef[i] = i - 1, rig[i] = i + 1;
  lef[0] = rig[n - 1] = -1;
  fr(i,0,n - 1) cst[i] = A[i] + A[i + 1];
  fr(i,0,n - 1) heap.push(MP(cst[i], i));
  int ans = 0;
  fr(_i,1,n) {
    int v; int x;
    do {
      v = heap.top().F; x = heap.top().S; heap.pop();
    } while(rig[x] == -1 || cst[x] != v);
    bool l = false, r = false;
    if(A[x] + A[rig[x]] == v) l = r = true;
    else if(A[x] + one(hpq[x]) == v) l = true;
    else if(A[rig[x]] + one(hpq[x]) == v) r = true;
    else if(two(hpq[x]) == v);
    ans += v;
    if(l) A[x] = INF;
    else if(hpq[x].size() > 0) hpq[x].pop();
    if(r) A[rig[x]] = INF;
    else if(hpq[x].size() > 0) hpq[x].pop();
    if(l && x > 0) x = merge(x, lef[x]);
    if(r && rig[x] < n - 1) merge(rig[x], x);
    hpq[x].push(v);
    cst[x] = min(A[x] + A[rig[x]],
             min(A[x], A[rig[x]]) + one(hpq[x]));
    cst[x] = min(cst[x], two(hpq[x]));
    heap.push(mp(cst[x], x)); }
    return ans; }
```

Suffix Automaton

```
struct State {
  int len, link;
  ll cnt;
  int next[30];
  State() { cnt = 0; }
} st[2000005];
int sz, last;
void sa_init() {
  sz = 1; last = st[0].len = st[0].cnt = 0;
  st[0].link = -1;
  memset(st[0].next, -1, sizeof st[0].next);
}
void sa_extend(int c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  memset(st[cur].next, -1, sizeof st[cur].next);
  st[cur].cnt = 1;
  int p;
  for (p = last; p != -1 && st[p].next[c] == -1; p = st[p].link) {
    st[p].next[c] = cur;
  }
  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;
      memcpy(st[clone].next, st[q].next, sizeof st[q].next);
      st[clone].link = st[q].link;
      for (; p != -1 && st[p].next[c] == q; p = st[p].link) {
        st[p].next[c] = clone;
      }
      st[q].link = st[cur].link = clone;
    }
  }
  last = cur;
}

void build_dfa(char *s) {
  sa_init();
  for (int i = 0; s[i]; ++i) {
    sa_extend(i, s[i] - 'a');
  }
}
```

```
void prep_num_times_substr() {
  rp(i,sz) len[st[i].len].push_back(i);
  for (int l = n; l; --l) {
    rp(k,len[l].size()) {
      int i = len[l][k];
      int j = st[i].link;
      st[j].cnt += st[i].cnt; } }
}
ll num_diff_substr() {
  ll ret = 0;
  fr(i,1,sz) ret += st[i].len - st[st[i].link].len;
  return ret; }

int LCSubstr(const string& s, const string& t) {
  sa_init();
  rp(i,s.size()) sa_extend(s[i] - 'a');
  int at = 0, tam = 0, ret = 0;
  rp(i,t.size()) {
    while (at && st[at].next[t[i] - 'a'] == -1) {
      at = st[at].link;
      tam = st[at].len;
    }
    if (st[at].next[t[i] - 'a'] != -1) {
      at = st[at].next[t[i] - 'a'];
      ++tam;
    }
    ret = max(ret, tam);
  }
  return ret; }

int longest_repeated_susbtring(int vezes) {
  int r = 0;
  fr(i,1,sz) if (st[i].cnt >= vezes) {
    r = max(r, st[i].len);
  }
  return r; }

ll num_times(const string& pattern) {
  ll ans = 0;
  int cur = 0, mylen = 0;
  rp(i,len) nxt_node(cur, mylen, pattern[i] - 'a');
  if (mylen == pattern.size()) ans += st[cur].cnt;
  rp(i,per - 1) { // per == period of pattern
    if (mylen == pattern.size()) {
      --mylen;
      if (mylen <= st[st[cur].link].len) cur =
st[cur].link;
    }
    nxt_node(cur, mylen, pattern[i] - 'a');
    if (mylen == len) ans += st[cur].cnt;
  }
  return ans; }
```

```
// number of times a rotation of pattern appears
void nxt_node(int& cur, int& len, int c) {
  while (cur && st[cur].next[c] == -1) {
    cur = st[cur].link;
    len = st[cur].len;  }
  if (st[cur].next[c] != -1) {
    cur = st[cur].next[c];
    ++len;
  } }
```

Link-Cut Tree

```
struct Node {
  int id;
  Node *left, *right, *parent;
  bool revert;

  Node() {}
  Node(int id): id(id), left(NULL), right(NULL),
parent(NULL), revert(false) {}

  bool isRoot() {
    return parent == NULL || (parent->left != this
&& parent->right != this);
  }
  void push() {
    if (revert) {
      revert = false;
      swap(left, right);
      if (left != NULL) left->revert ^= 1;
      if (right != NULL) right->revert ^= 1;
    }
  }
} nodes[NODES];

void connect(Node *ch, Node *p, bool isLeftChild) {
  if (ch != NULL) ch->parent = p;
  if (isLeftChild) p->left = ch;
  else p->right = ch; }

void rotate(Node *x) {
  Node *p = x->parent;
  Node *g = p->parent;
  bool isRoot = p->isRoot();
  bool leftChild = x == p->left;

  connect(leftChild ? x->right : x->left, p,
leftChild);
  connect(p, x, !leftChild);
  if (!isRoot) connect(x, g, p == g->left);
  else x->parent = g; }
void splay(Node *x) {
  while (!x->isRoot()) {
    Node *p = x->parent, *g = p->parent;
    if (!p->isRoot()) g->push();
    p->push(); x->push();
    if (!p->isRoot()) {
      rotate((x == p->left) == (p == g->left) ? p :
x);
    }
    rotate(x);
  }
  x->push(); }
```

```
Node* expose(Node *x) {
  Node *last = NULL, *y;
  for (y = x; y != NULL; y = y->parent) {
    splay(y);
    y->left = last;
    last = y;
  }
  splay(x);
  return last;
}

void makeRoot(Node *x) {
  expose(x);
  x->revert ^= 1;
}

bool connected(Node *x, Node *y) {
  if (x == y) return true;
  expose(x);
  expose(y);
  return x->parent != NULL;
}

bool link(Node *x, Node *y) {
  if (connected(x,y)) return false;
  makeRoot(x);
  x->parent = y;
  return true;
}

bool cut(Node *x, Node *y) {
  makeRoot(x);
  expose(y);
  if (y->right != x || x->left != NULL || x->right
!= NULL) return false;
  y->right->parent = NULL;
  y->right = NULL;
  return true;
}
```

Xor Gauss

```
void xorgauss() {
  int k = 0;
  for (int i=60, j; i>=0; i--) {
    for(j=k;j<ans.size();j++)
if((ans[j]&(1LL<<i))!=0) break;
    if (j==ans.size()) continue;
    if (j!=k) swap(ans[k],ans[j]);
    for (j=k+1;j<ans.size();j++)
if((ans[j]&(1LL<<i))!=0) ans[j]^=ans[k];
    k++; } }
```

Gauss

```
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
T GaussJordan(VVT &a, VVT &b) {
  const int n = a.size();
  const int m = b[0].size();
  vi irow(n), icol(n), ipiv(n);
  T det = 1;
  rp(i,n) {
    int pj = -1, pk = -1;
    rp(j,n) if (!ipiv[j]) rp(k,n) if (!ipiv[k]) {
      if (pj == -1 || fabs(a[j][k]) >
fabs(a[pj][pk])) {
        pj = j; pk = k;
      }
    }
    // matrix is singular, exit!
    if (fabs(a[pj][pk]) < eps) return 1./0.;
    ipiv[pk]++;
    swap(a[pj], a[pk]); swap(b[pj], b[pk]);
    if (pj != pk) det *= -1;
    irow[i] = pj; icol[i] = pk;
    T c = 1.0 / a[pk][pk];
    det *= a[pk][pk];
    a[pk][pk] = 1.0;
    rp(p,n) a[pk][p] *= c;
    rp(p,m) b[pk][p] *= c;
    rp(p,n) if (p != pk) {
      c = a[p][pk];
      a[p][pk] = 0;
      rp(q,n) a[p][q] -= a[pk][q] * c;
      rp(q,m) b[p][q] -= b[pk][q] * c;
    }
  }
  for (int p = n-1; p >= 0; p--) if (irow[p] !=
icol[p])
    rp(k,n) swap(a[k][irow[p]], a[k][icol[p]]);
  return det;
}
```

Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

// Value should be null_type if using as set
typedef tree<Key, Value, less<Key>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;

// order_of_key(x) -> number of elements < x
// *find_by_order(k) -> kth element in set
(0-based)
```

Polynomials

```
typedef complex<double> cdouble;
int cmp(cdouble x, cdouble y = 0) {
  return cmp(abs(x), abs(y)); }
const int TAM = 200;
struct poly {
  cdouble poly[TAM]; int n;
  poly(int n = 0): n(n) { cl(p, 0); }
  cdouble& operator [](int i) { return p[i]; }
  poly operator ~() {
    poly r(n-1);
    fr(i,1,n+1) r[i-1] = p[i] * cdouble(i);
    return r; }
  pair<poly, cdouble> ruffini(cdouble z) {
    if (n == 0) return mp(poly(), 0);
    poly r(n-1);
    for(int i=n;i>0;i--) r[i-1]=r[i] * z + p[i];
    return mp(r, r[0] * z + p[0]); }
  cdouble operator ()(cdouble z) {
    return ruffini(z).nd; }
  cdouble find_one_root(cdouble x) {
    poly p0 = *this, p1 = ~p0, p2 = ~p1;
    int m = 1000;
    while (m--) {
      cdouble y0 = p0(x);
      if (cmp(y0) == 0) break;
      cdouble G = p1(x) / y0;
      cdouble H = G * G - p2(x) - y0;
      cdouble R = sqrt(cdouble(n-1)*(H * cdouble(n) -
G*G));
      cdouble D1 = G + R, D2 = G - R;
      cdouble a = cdouble(n) / (cmp(D1, D2) > 0 ? D1 :
D2);
      x -= a;
      if (cmp(a) == 0) break;
    }
    return x;
  }
  vector<cdouble> roots() {
    poly q = *this;
    vector<cdouble> r;
    while (q.n > 1) {
      cdouble z(rand()/double(RAND_MAX),
              rand()/double(RAND_MAX));
      z = q.find_one_root(z); z = find_one_root(z);
      q = q.ruffini(z).first;
      r.push_back(z);
    }
    return r;
  }
};
```

Polynomial Division

```
pair<P,P> operator / (const P &rhs) const {
  P quo = P(), ret = P(); // quociente, resto
  quo.grau = grau - rhs.grau;
  memcpy(ret.v, v, sizeof v);
  ret.grau = grau;
  int ind = grau;
  while (ind >= rhs.grau) {
    ll aux = ret.v[ind] / rhs.v[rhs.grau];
    quo.v[ind-rhs.grau] = aux;
    fr(i,0,rhs.grau+1) {
      ret.v[ind-rhs.grau+i] = ret.v[ind-rhs.grau+i] - aux *
rhs.v[i];
    }
    --ind;
  }
  while (quo.grau >= 0 && quo.v[quo.grau] == 0) quo.grau--;
  while (ret.grau >= 0 && ret.v[ret.grau] == 0) ret.grau--;
  return pair<P,P>(quo, ret);
}
```

Hungarian

```
struct hungarian {
  int cost[maxn][maxn];
  int n, max_match;
  int lx[maxn], ly[maxn];
  int xy[maxn], yx[maxn];
  bool S[maxn], T[maxn];
  int slack[maxn], slackx[maxn];
  int pre[maxn];
  void init_labels() {
    cl(lx,0); cl(ly,0);
    rp(x,n) rp(y,n) lx[x] = max(lx[x], cost[x][y]); }
  void add_to_tree(int x, int prex) {
    S[x] = true; pre[x] = prex;
    rp(y,n) {
      if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
        slack[y] = lx[x] + ly[y] - cost[x][y];
        slackx[y] = x;
      } } }
  void update_labels() {
    int x, y, delta = oo;
    rp(y,n) if (!T[y]) delta = min(delta, slack[y]);
    rp(x,n) if (S[x]) lx[x] -= delta;
    rp(y,n) if (T[y]) ly[y] += delta;
    rp(y,n) if (!T[y]) slack[y] -= delta; }
  void augment() {
    if (max_match == n) return;
    int x, y, root;
    int q[maxn], wr = 0, rd = 0;
    cl(S,0); cl(T,0); cl(pre,-1);
    rp(x,n) if (xy[x] == -1) {
      q[wr++] = root = x;
      pre[x] = -2;
      S[x] = true;
      break; }
    rp(y,n) {
      slack[y] = lx[root] + ly[y] - cost[root][y];
      slackx[y] = root; }
    while (true) {
      while (rd < wr) {
        x = q[rd++];
        for (y = 0; y < n; y++) {
          if (cost[x][y] == lx[x] + ly[y] && !T[y]){
            if (yx[y] == -1) break;
            T[y] = true;
            q[wr++] = yx[y];
            add_to_tree(yx[y], x); } }
        if (y < n) break;
      }
      if (y < n) break;
      update_labels();
      wr = rd = 0;
      for (y = 0; y < n; y++) {
        if (!T[y] && slack[y] == 0) {
          if (yx[y] == -1) {
            x = slackx[y];
            break;
          } else {
            T[y] = true;
            if (!S[yx[y]]) {
              q[wr++] = yx[y];
              add_to_tree(yx[y], slackx[y]);
            } } } }
      if (y < n) break;
    }
    if (y < n) {
      max_match++;
      for (int cx = x, cy = y, ty; cx != -2; cx =
pre[cx], cy = ty) {
        ty = xy[cx];
        yx[cy] = cx;
        xy[cx] = cy; }
      augment(); }  }
  int go() {
    int ret = 0;
    max_match = 0;
    cl(xy, -1); cl(yx,-1);
    init_labels();
    augment();
    rp(x,n) ret += cost[x][xy[x]];
    return ret;
  }
};
```

Edmonds's Blossom O(n^3)

```
#define MAXN 110
#define MAXM MAXN*MAXN
int n,m;
int mate[MAXN], first[MAXN], label[MAXN];
int adj[MAXN][MAXN], nadj[MAXN], from[MAXM],
to[MAXM];
queue<int> q;
#define OUTER(x) (label[x] >= 0)
void L(int x, int y, int nxy) {
  int join, v, r = first[x], s = first[y];
  if (r == s) return;
  nxy += n + 1;
  label[r] = label[s] = -nxy;
  while (1) {
    if (s != 0) swap(r,s);
    r = first[label[mate[r]]];
    if (label[r] != -nxy) label[r] = -nxy;
    else {
      join = r;
      break; } }
  v = first[x];
  while (v != join) {
    if (!OUTER(v)) q.push(v);
    label[v] = nxy; first[v] = join;
    v = first[label[mate[v]]]; }
  v = first[y];
  while (v != join) {
    if (!OUTER(v)) q.push(v);
    label[v] = nxy; first[v] = join;
    v = first[label[mate[v]]]; }
  for (int i = 0; i <= n; i++) {
    if (OUTER(i) && OUTER(first[i]))
      first[i] = join; }}
void R(int v, int w) {
  int t = mate[v]; mate[v] = w;
  if (mate[t] != v) return;
  if (label[v] >= 1 && label[v] <= n) {
    mate[t] = label[v];
    R(label[v],t);
    Return; }
  int x = from[label[v]-n-1], y =
to[label[v]-n-1];
  R(x,y); R(y,x); }
int E() {
  memset(mate,0,sizeof(mate));
  int r = 0; bool e7;
  for (int u = 1; u <= n; u++) {
    memset(label,-1,sizeof(label));
    while (!q.empty()) q.pop();
    if (mate[u]) continue;
    label[u] = first[u] = 0;
```

```
    q.push(u); e7 = false;
    while (!q.empty() && !e7) {
      int x = q.front(); q.pop();
      for (int i = 0; i < nadj[x]; i++) {
        int y = from[adj[x][i]];
        if (y == x) y = to[adj[x][i]];
        if (!mate[y] && y != u) {
          mate[y] = x; R(x,y);
          r++; e7 = true;
          break; }
        else if (OUTER(y)) L(x,y,adj[x][i]);
        else {
          int v = mate[y];
          if (!OUTER(v)) {
            label[v] = x; first[v] = y;
            q.push(v);
          } } } }
    label[0] = -1; }
  return r; }
/*Exemplo simples de uso*/
cl(nadj,0);
rp(i,m) { //arestas
  sc2(a,b); a++, b++;//nao utilizar o vertice 0
  adj[a][nadj[a]++] = i;
  adj[b][nadj[b]++] = i;
  from[i] = a; to[i] = b; }
printf("O emparelhamento tem tamanho %d\n",E());
for (int i = 1; i <= n; i++) {
  if (mate[i] > i)
    printf("%d com %d\n",i-1,mate[i]-1); }
```

Stable Marriage

```
struct S {//sorted list of indices by preference
  int lista[maxn], pref[maxn];
} h[maxn], m[maxn];
int prox[maxn], quem[maxn], n;
void find(int H, int M) {
  if (quem[M] == -1) {
    quem[M] = H;
  } else if (m[M].pref[H] < m[M].pref[quem[M]])
{
    int now = quem[M];
    quem[M] = H;
    find(now, h[now].lista[prox[now]++]);
  } else {
    find(H, h[H].lista[prox[H]++]); } }
void process() {
  cl(prox,0); cl(quem,-1);
  rp(i,n) if (i) find(i, h[i].lista[prox[i]++]);
  rp(i,n) if (i) // quem[i] with i
}
```

Markov Chain DP

```
#define ANS -1
map<int, double> pd[N];
int mark[N];
void proc(int v) {
  if (mark[v] == 0) {
    pd[v].clear();
    int i = v/(n+1), j = v%(n+1);//specific
    if (max(i, j) == n) {//base case: put answer
      pd[v][ANS] = (i == n)? 1.0: 0.0;
      mark[v] = 2; return;
    }
    //has transitions: fill transition map
    create_adj(i, j);//specific
    fre(it, v) pd[v][to[it]] = 0.5;
  }
  mark[v] = 1;
  vi novo; novo.reserve(pd[v].size());
  fore(it, pd[v])
    if (it->st != ANS && mark[it->st] != 1)
      novo.pb(it->st);
  rp(i, novo.size()) {
    int w = novo[i];
    double fac = pd[v][w];
    pd[v].erase(w); proc(w);
    fore(it, pd[w]) {
      if (pd[v].count(it->st))
        pd[v][it->st] += fac*(it->nd);
      else pd[v][it->st] = fac*(it->nd);
  }}
  if (pd[v].count(v)) {
    double fac = 1.0 - pd[v][v];
    pd[v].erase(v);
    fore(it, pd[v]) it->nd /= fac;
  } mark[v] = 2; }

cl(mark, 0); proc(0);
assert(pd[0].size() == 1);
printf("%.3lf\n", pd[0][ANS]);
```

Dates

```
string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri",
"Sat", "Sun"};
string intToDay(int jd) {
  return dayOfWeek[jd % 7];
}

// converts Julian date to Julian day number
int julianToInt(int dia, int mes, int ano) {
  int id = ano * 365 + (ano + 3) / 4;
  rp(i,mes-1) {
    if (i == 2 && ano % 4 == 0) ++id;
    id += dias_mes[i];
  }
  id += dia + 1721057;
  return id;
}
// converts Gregorian date to integer
//(Julian day number)
int dateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}
// converts integer (Julian day number)
//to Gregorian date: month/day/year
void intToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;
  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}
```

Hybrid RMQ <O(N log log N), O(1)>

```
const int N = 100009, BN = 6259, LBN = 14;
int bs, bn, block[BN];
int st[N][LBN], bst[BN][LBN];
void buildST(int *a, int n, int pd[][LBN]) {
  rp(i, n) pd[i][0] = a[i];
  for(int j = 1; (1<<j) <= n; j++)
    rp(i, n+1-(1<<j))
      pd[i][j] = min(pd[i][j-1],
                     pd[i+(1<<(j-1))][j-1]);
}
int logg[N];//logg[0] = -1;
//fr(i, 1, N) logg[i] = 1+logg[i>>1];
int queryST(int b, int e, int pd[][LBN]) {
  int l = logg[e-b];
  return (l < 0)? oo: min(pd[b][l],
pd[e-(1<<l)][l]);
}
void build(int *a, int n) {
  bs = max(1, logg[n]); bn = (n+bs-1)/bs;
  rp(i, bn) {
    int beg = i*bs, end = min((i+1)*bs, n);
    buildST(a+beg, end-beg, st+beg);
    block[i] = queryST(beg, end, st);
  } buildST(block, bn, bst);
}
int query(int b, int e) {
  int p1 = (b-1)/bs+1, p2 = e/bs;
  if (p1 <= p2) return min(
      queryST(p1, p2, bst),
      min(queryST(b, p1*bs, st),
          queryST(p2*bs, e, st)));
  else return queryST(b, e, st);
}
```

Pollard-Rho

```
inline bool overflow(ull a, ull b, ll LINF =
(1LL<<62)) {
  return b && (a >= LINF / b);
}
ull mulMod(ull a, ull b, ull c) { // (a * b) % c
  if (!overflow(a, b)) return (a * b) % c;
  ull x = 0, y = a % c;
  for (; b; y = (y << 1) % c, b >>= 1) if(b & 1) x =
(x + y) % c;
  return x % c;
}
ull potMod(ull a, ull b, ull c) { // (a ^ b) % c
  ull x = 1, y = a;
  for (; b; y = mulMod(y, y, c), b >>= 1) if(b & 1) x
= mulMod(x, y, c);
  return x;
}
bool miller(ull p, int iteracao){
  if (p < 2) return false;
  if (p % 2 == 0) return (p == 2);
  ll s = p - 1;
  while( s % 2 == 0) s >>= 1;
  for (int i = 0; i < iteracao; i++) {
    ull a = rand() % (p - 1) + 1, temp = s;
    ull mod = potMod(a, temp, p);
    while (temp != p - 1 && mod != 1 && mod != p-1)
      mod = mulMod(mod, mod, p), temp <<= 1;
    if (mod != p - 1 && temp % 2 == 0) return false;
  }
  return true;
}
ull func(ull x, ull n, ull c) { return (mulMod(x, x,
n) + c) % n; }
ull gcd(ull x, ull y) { while(y) x %= y, swap(x,y);
return x; }
ull rho(ull n) {
  if (miller(n,20)) return n;
  ull x = 2, y = 2, d = 1, c;
  do {
    c = rand() % n;
  } while(c == 0 || (c + 2) % n == 0);
  ll pot = 1, lam = 1;
  while (d != n) { // Brent
    if (pot == lam) { x = y, pot <<= 1, lam = 0; }
    y = func(y, n, c), lam++;
    d = gcd((x >= y ? x - y : y - x), n);
    if (d != 1) return d; }
  return n; }
// not needed for pollard rho
bool isPrime(ll n) {
  if (n <= 1) return 0;
  if (n <= 3) return 1;
  if (!(n&1)) return 0;
  return miller(n,2) && miller(n,3) &&
  miller(n,5) && miller(n,7) &&
  (n < 3215031751LL || miller(n,11)) &&
  (n < 2152302898747LL || miller(n,13)) &&
  (n < 3474749660383LL || miller(n,17)) &&
  (n < 341550071728321LL || miller(n,23));
}
Const int MAXN = 50, MAXL = 22, LIM = 7;
pair<ull,int> f[MAXL][MAXN];
int cnt[MAXL];
void go(ull n, int lvl = 0) {
  cnt[lvl] = 0;
  pair<ull,int> *g = f[lvl];
  do {
    ull x = n;
    int lim = 0;
    while (x == n && ++lim < LIM) x = rho(n);
    if (x == n) { //prime
      g[cnt[lvl]++] = MP(x, 1);
      Break; }
    int q = 1; n /= x;
    while(!(n % x)) ++q, n /= x;
    go(x, lvl + 1);
    rp(i,cnt[lvl + 1])
      g[cnt[lvl]++] = MP(f[lvl + 1][i].F, f[lvl +
1][i].S * q);
  } while(n > 1);
  sort(g, g + cnt[lvl]);
  int p = 1;
  fr(i,1,cnt[lvl]) {
    if(g[i].F == g[p - 1].F) g[p - 1].S += g[i].S;
    else g[p++] = f[lvl][i]; }
  cnt[lvl] = p; }
```

```
ull n;
int main() {
  srand(time(0));
  int t; scanf("%d", &t);
  while(t--) {
    scanf("%llu", &n);
    printf("%llu", n);

    go(n);
    rp(i,cnt[0]) {
      printf(" %c %llu", i > 0 ? '*' : '=', f[0][i].F);
      if(f[0][i].S > 1) printf("^%d", f[0][i].S);
    }
    puts("");
  }

  return 0;
}
```

Theorems

Dilworth's theorem

In any partially ordered set, the maximum number of elements in any antichain equals the minimum number of chains in any partition of the set into chains.

Derangements

Number of permutations of the elements of a set such that none of the elements appear in their original position. {1, 0, 1, 2, 9, 44, 265, 1854, 14833, …}

D(n) = (n - 1) * (D(n - 1) + D(n - 2)) = n * D(n - 1) + $(-1)^n$.

Erdos Gallai's Theorem

Gives a necessary and sufficient condition for a finite sequence of natual numbers to be the degree sequence of a simple graph. A sequence of non-negative integers

$d1 \geq d2 \geq \ \dots \ \geq dn$ can be the degree sequence of a simple graph on n vertices iff $\sum_{i=1}^{n} di$ is even and $\sum_{i=1}^{k} di \leq k*(k-1) + \sum_{i=k+1}^{n} min(di,k)$ holds for every $1 \leq k \leq n$.

```
// graus em arr
bool valid() {
  sort(arr, arr+n, greater<int>());
  sum[0] = 0;
  fr(i,1,n+1) sum[i] = sum[i-1] + arr[i-1];
  if (sum[n] & 1) return false;
  for (ll k = 1, ind = n-1, aux; k <= n; ++k) {
    while (ind && arr[ind] < k) --ind;
    if (ind+1 > k) {
      aux = k * (k-1) + k*(ind+1-k) + sum[n]-sum[ind+1];
    } else {
      aux = k * (k-1) + sum[n]-sum[k];
    }
    if (sum[k] > aux) return false;
  }
  return true;
}
```

Fermat primes

A Fermat prime is a prime of form 2^(2^n) + 1. The only known Fermat primes are 3, 5, 17, 257, 65537. A number of form 2^n + 1 is prime only if it is a Fermat prime.

Narayana Number

Narayana is the number of paths from (0, 0) to (2n, 0), with steps only northeast and southeast, not straying below the x-axis, with k peaks.

$$N(n,k) = 1/n * \binom{n}{k} * \binom{n}{k-1} \quad N(n,1)+N(n,2)+\dots+N(n,n) = C(n)$$

Perfect numbers

$n > 1$ is called perfect if it equals sum of its proper divisors and 1. Even n is perfect i $n = 2^{p-1}(2^p - 1)$ and $2^p - 1$ is prime (Mersenne's). No odd perfect numbers are yet found.

Pitagorean Triples

Integer solutions of $x^2 + y^2 = z^2$. All relatively prime triples are given by: $x = 2mn$, $y = m^2 - n^2$ where $m > n > 0$, $gcd(m,n) = 1$ and $m \not\equiv n \bmod 2$.

Konig Theorem

Em qualquer grafo bipartido, o número de arestas no maximum matching é igual ao número de vértices no minimum vertex cover. Complemento de um minimum vertex cover é um maximum independent set.

Stirling numbers

Stirling numbers *of the first kind* count permutations according to their number of cycles (counting fixed points as cycles of length one)

Recorrência: $\begin{bmatrix} n+1 \\ k \end{bmatrix} = n \begin{bmatrix} n \\ k \end{bmatrix} + \begin{bmatrix} n \\ k-1 \end{bmatrix}$

Casos base: $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1, \begin{bmatrix} n \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ n \end{bmatrix} = 0$

Stirling number *of the second kind* (or Stirling partition number) is the number of ways to partition a set of n objects into k non-empty subsets

Recorrência: $\begin{Bmatrix} n+1 \\ k \end{Bmatrix} = k \begin{Bmatrix} n \\ k \end{Bmatrix} + \begin{Bmatrix} n \\ k-1 \end{Bmatrix}$

Casos base: $\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = 1, \begin{Bmatrix} n \\ 0 \end{Bmatrix} = \begin{Bmatrix} 0 \\ n \end{Bmatrix} = 0$

SuperCatalan Numbers

{1,1,3,11,45,197,...}

$$S(n) = \frac{3(2n-3)S(n-1) - (n-3)S(n-2)}{n}$$

- S(n) counts the total number of bracketing of n items.

Lucas' Theorem
For non-negative integers m and n and a prime p

$$\binom{n}{m} = \prod \binom{ni}{mi} \ (mod\ p)$$

Matrix-tree theorem (Kirchhoff's theorem)
Let matrix T = [tij], where ti_j is the number of multiedges between i and j, for i!=j, and tii = −degi. Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T.

Pick's Theorem
Area of a polygon in terms of the number i of lattice points in the interior located in the polygon and the number b of lattice points on the boundary placed on the polygon's perimeter.

$A = i + b/2 - 1$

Sum of squares stuff
A number n can be written as a sum of:
2 squares: iff all prime numbers in its factorization of the form 4*k + 3 appear an even number of times.
3 squares: n is not of the form 4^a * (8 * b + 7), for some a and b
4 squares: all numbers can be written as a sum of 4 squares

Wilson's Theorem
A natural number n > 1 is prime if and only if $(n - 1)! \equiv -1\ (mod\ n)$.

Sequences

Carmichael numbers
A positive composite n is a Carmichael number ($an{-}1 \equiv 1$ (mod n) for all a: gcd(a, n) = 1), i n is square-free, and for all prime divisors p of n, p − 1 divides n − 1.

Catalan Numbers
{ 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, ...}
Catalan numbers are defined by the recurrence:

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

$$C_n = \frac{2n \times (2n-1)}{(n+1) \times n} C_{n-i}$$

A closed formula for Catalan numbers is:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$$

- Cat(n) counts the number of distinct binary trees with n vertices.
- Cat(n) counts the number of expressions containing n pairs of parentheses which are correctly matched.
- Cat(n) counts the number of different ways n + 1 factors can be completely parenthesized.
- Cat(n) counts the number of ways a convex polygon of n + 2 sides can be triangulated.
- Cat(n) counts the number of monotonic paths along the edges of an n x n grid, which do not pass above the diagonal. A monotonic path is one which starts in the lower left corner, finishes in the upper right, and consists entirely of edges pointing rightwards or upwards.

```
ll catalan(int n) { return n ? catalan(n-1)*(4*n-2)/(n+1): 1; }
```

Largest primes
The largest prime smaller than

```
-> 10    is 7.
-> 10^2  is 97.
-> 10^3  is 997.
-> 10^4  is 9973.
-> 10^5  is 99991. (4 9's)
-> 10^6  is 999983. (4 9's)
-> 10^7  is 9999991. (6 9's)
-> 10^8  is 99999989. (6 9's)
-> 10^9  is 999999937. (7 9's)
-> 10^10 is 9999999967. (8 9's)
-> 10^11 is 99999999977. (9 9's)
-> 10^12 is 999999999989. (10 9's)
-> 10^13 is 9999999999971. (11 9's)
-> 10^14 is 99999999999973. (12 9's)
-> 10^15 is 999999999999989. (13 9's)
-> 10^16 is 9999999999999937. (14 9's)
-> 10^17 is 99999999999999997. (16 9's)
-> 10^18 is 999999999999999989. (16 9's)
```

Formulae

Cayley's formula

There are n^{n-2} spanning trees of a complete graph with n labeled vertices.

Combinatorics

$$\binom{m+n}{r}=\sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k} \qquad \binom{n}{k}=\prod_{i=1}^{k}\frac{n-k+i}{i}$$

$$\binom{n}{k}=\frac{n!}{(n-k)!k!} \qquad \binom{n}{k}=\binom{n-1}{k}+\binom{n-1}{k-1}$$

$$\binom{n}{k}=\frac{n}{n-k}\binom{n-1}{k} \qquad \binom{n}{k}=\frac{n-k+1}{k}\binom{n}{k-1}$$

$$\binom{n+1}{k}=\frac{n+1}{n-k+1}\binom{n}{k} \qquad \binom{n}{k+1}=\frac{n-k}{k+1}\binom{n}{k}$$

Moser's Circle

Determine the number of pieces into which a circle is divided if n points on its circumference are joined by chords with no three internally concurrent.

$$g(n)=\binom{n}{4}+\binom{n}{2}+1$$

Number of spanning trees $K_{n,m}$

$$m^{n-1}\times n^{m-1}$$

Sums

$\sum_{k=0}^{n} k = n(n+1)/2$ $\qquad$ $\sum_{k=a}^{b} k = (a+b)(b-a+1)/2$

$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$ $\qquad$ $\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$

$\sum_{k=0}^{n} k^4 = (6n^5+15n^4+10n^3-n)/30$ $\qquad$ $\sum_{k=0}^{n} k^5 = (2n^6+6n^5+5n^4-n^2)/12$

$\sum_{k=0}^{n} x^k = (x^{n+1}-1)/(x-1)$ $\qquad$ $\sum_{k=0}^{n} kx^k = (x-(n+1)x^{n+1}+nx^{n+2})/(x-1)^2$

$$\sum_{k=1}^{n} k\binom{n}{k}=n2^{n-1} \qquad \sum_{k=1}^{n} k^2\binom{n}{k}=(n+n^2)2^{n-2}$$

General case

$$\sum_{i=1}^{n} i^p=\frac{1}{p+1}\left[(n+1)^{p+1}-1-\sum_{i=1}^{n}\sum_{j=0}^{p-1}\binom{p+1}{j}i^j\right]$$

```
// sum[i] = somatorio de k = 0 ate n-1 (de k^i)
// sum[0] = n - 1
sum[0].grau = 1;
sum[0].v[0] = F(-1,1);
sum[0].v[1] = F(1,1);
// sum[1] = n*(n-1)/2 = n²/2 - n/2
sum[1].grau = 2;
sum[1].v[0] = F(0,1);
sum[1].v[1] = F(-1,2);
sum[1].v[2] = F(1,2);
fr(i,2,n+1) {
  sum[i].grau = i+1;
  sum[i].v[0] = F(-1,1);
  sum[i].v[i+1] = F(1,1);
  fr(j,0,i) {
    sum[i] = sum[i] - sum[j] * C[i+1][j];
  }
  sum[i] = sum[i] / C[i+1][i]; // divide por (i+1)
}

sum[n].v[n] = sum[n].v[n] + F(1,1);
```

Geometric Progressions

$$S_n=\frac{a_1(q^n-1)}{q-1}$$

$$S_\infty=\sum_{n=1}^{\infty}a_1q^{n-1}=\frac{a_1}{1-q}$$

Josephus

```
int josephus (int n, int m, int k) {
  int x = -1;
  fr(i,n-k+1,n+1) x = (x+m)%i;
  return x; }
```

Mo stuff

```
if (st[U] > st[V]) swap(U, V);
int l = lca(U, V);
if (l == U) queries[i] = Query(st[U] + 1, st[V] + 1, i, U, V);
else queries[i] = Query(fin[U], st[V] + 1, i, U, V);
//---------------------
int l = 0, r = 0;
for (int i = 0; i < q; ++i) {
  while (l > Q[i].l) add(--l);
  while (r < Q[i].r) add(r++);
  while (l < Q[i].l) remove(l++);
  while (r > Q[i].r) remove(--r);
  ans[Q[i].id] = get_answer();
}
```

Faster Ternary Search

```
const double phi = 0.618;
double minimum(double lo, double hi) {
  double m1, m2, v1, v2;
  bool b1 = 0, b2 = 0;
  while (hi-lo > eps) {
    if (!b1) { m1 = (lo + phi*hi)*phi; v1 = f(m1); }
    if (!b2) { m2 = (lo*phi + hi)*phi; v2 = f(m2); }
    b1 = b2 = 1;
    if (v1 < v2) { hi = m2; m2 = m1; v2 = v1; b1 = 0; }
    else { lo = m1; m1 = m2; v1 = v2; b2 = 0; }
  }} return (hi+lo)/2.0; }
```

Treap

```
struct TNode {
  int x, y, z; char c;
  TNode *L, *R;
  TNode() {}
  TNode(int x, TNode *L, TNode *R, char c = 0) {
    this->x = x; this->y = rand(); this->z = 0;
    this->c = c; this->L = L; this->R = R; }
} *nil = new TNode(0,NULL,NULL), node[maxn];
typedef TNode* Node;
void fix(Node &P) { if (P != nil) P->z = P->L->z + P->R->z + 1; }
Node merge(Node L, Node R) {
  if (L == nil) return R; else if (R == nil) return L;
  if (L->y >= R->y) return L->R = merge(L->R,R), fix(L), L;
  else return R->L = merge(L,R->L), fix(R), R; }
void split(Node P, Node &L, Node &R, int x) {
  if (P == nil) return L = nil, R = nil, void();
  if (P->x <= x) return L = P, split(P->R, L->R, R, x), fix(L);
  return R = P, split(P->L, L, R->L, x), fix(R); }
void insert(Node &P, Node novo) {
  if (P == nil || novo->y >= P->y) split(P, novo->L, novo->R, novo->x), P
= novo;
  else if (novo->x < P->x) insert(P->L, novo); else insert(P->R, novo);
fix(P); }
void remove(Node &P, int x) {
  if (P->x == x) return P = merge(P->L, P->R), fix(P);
  if (x < P->x) remove(P->L,x), fix(P); else remove(P->R,x), fix(P); }
int kth(Node P, int x) {
  int myx = P->L->z + 1;
  if (x < myx) return kth(P->L, x);
  if (x > myx) return kth(P->R, x-myx);
  return P->x; }
// root = nil; node[..] = TNode(i, NULL, NULL, ...);
// lazy-like stuff: split(root, t1, t3, b); split(t1, t1, t2, a - 1);
root = merge(merge(t1, t2), t3);
```

Identities

Identities:

$$\sin x = \frac{1}{\csc x},\qquad \cos x = \frac{1}{\sec x},\qquad \sin 2x = 2\sin x\cos x,$$

$$\tan x = \frac{1}{\cot x},\qquad \sin^2 x + \cos^2 x = 1,\qquad \cos 2x = \cos^2 x - \sin^2 x,$$

$$1 + \tan^2 x = \sec^2 x,\qquad 1 + \cot^2 x = \csc^2 x,\qquad \cos 2x = 1 - 2\sin^2 x,$$

$$\sin x = \cos\left(\frac{\pi}{2} - x\right),\qquad \sin x = \sin(\pi - x),\qquad \tan 2x = \frac{2\tan x}{1 - \tan^2 x},$$

$$\cos x = -\cos(\pi - x),\qquad \tan x = \cot\left(\frac{\pi}{2} - x\right),\qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$

$$\cot x = -\cot(\pi - x),\qquad \csc x = \cot\frac{x}{2} - \cot x,\qquad \cos 2x = 2\cos^2 x - 1,$$

$$\sin(x \pm y) = \sin x\cos y \pm \cos x\sin y,\qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$

$$\cos(x \pm y) = \cos x\cos y \mp \sin x\sin y,\qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$

$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x\tan y},$$

$$\cot(x \pm y) = \frac{\cot x\cot y \mp 1}{\cot x \pm \cot y},$$

$$\sin(x + y)\sin(x - y) = \sin^2 x - \sin^2 y,$$

$$\cos(x + y)\cos(x - y) = \cos^2 x - \sin^2 y.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

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

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

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

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

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

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

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

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

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

Law of sines: $\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R_{out}$.

Law of cosines: $c^2 = a^2 + b^2 - 2ab\cos C$.

Law of tangents: $\frac{a+b}{a-b} = \frac{\tan[\frac{1}{2}(A+B)]}{\tan[\frac{1}{2}(A-B)]}$

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

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

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

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

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

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

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

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

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

Inscribed/outscribed circles: $R_{out} = \frac{abc}{4S}$, $R_{in} = \frac{2S}{a+b+c}$

Heron: $\sqrt{s(s-a)(s-b)(s-c)}$, $s = \frac{a+b+c}{2}$.

Δ's area, given side and adjacent angles: $\frac{c^2}{2(\cot\alpha + \cot\beta)}$

Wavelet Tree

```
struct WaveletTree {
  vector<vector<int>> C; int s;
  vector<int> S;

  WaveletTree(vector<int> A, int sigma) : S(A), C(sigma * 2), s(sigma) {
    build_tree(A.begin(), A.end(), 1, 0, s - 1); }

  void build_tree(iter b, iter e, int node, int l, int r) {
    if (l == r) return;
    int m = (l + r) / 2;

    C[node].reserve(e - b + 1); C[node].push_back(0);
    for (iter it = b; it != e; ++it)
      C[node].push_back(C[node].back() + (*it <= m));

    iter p = stable_partition(b, e, [m](int i) { return i <= m; });
    build_tree(b, p, node * 2    , l, m);
    build_tree(p, e, node * 2 + 1, m + 1, r);  }

  int kth(int k, int i, int j, int node, int l, int r) {
    if (l == r) return l;
    int m = (l + r) / 2;
    int ci = C[node][i], cj = C[node][j];
    if (k <= cj - ci) return kth(k, ci, cj, node * 2, l, m);
    else return kth(k - (cj - ci), i - ci, j - cj, node * 2 + 1, m + 1, r); }

  int kth(int k, int i, int j) { // [i, j), k starts from 1
    return kth(k, i, j, 1, 0, s - 1);  }

  void swap(int i, int a, int b, int node, int l, int r) {
    if (l == r) return;
    int m = (l + r) / 2;

    if (a <= m && b > m) { C[node][i + 1]--; return; }
    if (b <= m && a > m) { C[node][i + 1]++; return; }

    if (a <= m) swap(C[node][i], a, b, 2 * node, l, m);
    else swap(i - C[node][i], a, b, 2 * node + 1, m + 1, r);
  }

  void swap(int i) {
    if (S[i] == S[i + 1]) return;
    swap(i, S[i], S[i + 1], 1, 0, s - 1);
    std::swap(S[i], S[i + 1]);
  }
};
```

Minimum Cost Arborescence

```
const int N = 1000009, M = 10000009;
int u[M], v[M], cost[M], rep[M], orig[M], used[M];
int pre[N], id[N], vis[N], in[N], my[N];
int arbor(int root, int n, int m) {
  int ret = 0, bn = 0, bm = 0;
  cl(in, 0x3f); cl(id, -1); cl(vis, -1);
  fr(i, bm, m) {
    if (cost[i] < in[v[i]] && u[i] != v[i]) {
      pre[v[i]] = i; in[v[i]] = cost[i];
  }}
  while (1) {
    fr(i, bn, n) if (i != root && in[i] == oo) return oo;
    int n2 = n, m2 = m;
    in[root] = 0; pre[root] = -1;
    fr(i, bn, n) {
      int v = i; ret += in[v];
      while (vis[v] != i && id[v] == -1 && v != root) {
        vis[v] = i; v = u[pre[v]]; }
      if (v != root && id[v] == -1) {
        for(int x = u[pre[v]]; x != v; x = u[pre[x]])
          id[x] = n2;
        id[v] = n2++;
    }}
    if (n2 == n) break;
    fr(i, bn, n) if (id[i] == -1) id[i] = n2++;
    fr(i, bm, m) if (id[u[i]] != id[v[i]]) {
      u[m2] = id[u[i]]; v[m2] = id[v[i]];
      cost[m2] = cost[i] - in[v[i]];
      if (cost[m2] < in[v[m2]]) {
        pre[v[m2]] = m2; in[v[m2]] = cost[m2];
      } rep[i] = rep[m2] = m2; orig[m2++] = i;
    } root = id[root]; bn = n; n = n2; bm = m; m = m2;
  }
  rp(ii, m) {
    int e = m-ii-1;
    if (rep[e] != e) used[e] = used[rep[e]];
    else {
      used[e] = (e == pre[v[e]]);
      int w = id[v[e]];
      if (used[e] && w != -1) {
        int e2 = orig[my[w]];
        used[e] = (v[e2] != v[e]);
    }}
    if (used[e]) my[v[e]] = e;
  } return ret;
}
```

Integrals

Integrals:

1. $\int cu\,dx = c\int u\,dx$, 2. $\int (u+v)\,dx = \int u\,dx + \int v\,dx$,

3. $\int x^n\,dx = \frac{1}{n+1}x^{n+1}, \quad n \neq -1$, 4. $\int \frac{1}{x}dx = \ln x$, 5. $\int e^x\,dx = e^x$,

6. $\int \frac{dx}{1+x^2} = \arctan x$, 7. $\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$,

8. $\int \sin x\,dx = -\cos x$, 9. $\int \cos x\,dx = \sin x$,

10. $\int \tan x\,dx = -\ln|\cos x|$, 11. $\int \cot x\,dx = \ln|\cos x|$,

12. $\int \sec x\,dx = \ln|\sec x + \tan x|$, 13. $\int \csc x\,dx = \ln|\csc x + \cot x|$,

14. $\int \arcsin \frac{x}{a}dx = \arcsin \frac{x}{a} + \sqrt{a^2 - x^2}, \quad a > 0$,

15. $\int \arccos \frac{x}{a}dx = \arccos \frac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0$,

17. $\int \sin^2(ax)dx = \frac{1}{2a}(ax - \sin(ax)\cos(ax))$,

16. $\int \arctan \frac{x}{a}dx = x\arctan \frac{x}{a} - \frac{a}{2}\ln(a^2 + x^2), \quad a > 0$,

18. $\int \cos^2(ax)dx = \frac{1}{2a}(ax + \sin(ax)\cos(ax))$,

19. $\int \sec^2 x\,dx = \tan x$, 20. $\int \csc^2 x\,dx = -\cot x$,

$$\int_a^b f(x)\,dx \approx (b-a)\left(\frac{f(a)+f(b)}{2}\right).$$

$$\int_a^b f(x)\,dx \approx \frac{b-a}{n}\left(\frac{f(a)}{2} + \sum_{k=1}^{n-1}\left(f\left(a + k\frac{b-a}{n}\right)\right) + \frac{f(b)}{2}\right),$$

Derivatives

Derivatives:

1. $\frac{d(cu)}{dx} = c\frac{du}{dx}$, 2. $\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}$, 3. $\frac{d(uv)}{dx} = u\frac{dv}{dx} + v\frac{du}{dx}$,

4. $\frac{d(u^n)}{dx} = nu^{n-1}\frac{du}{dx}$, 5. $\frac{d(u/v)}{dx} = \frac{v(\frac{du}{dx}) - u(\frac{dv}{dx})}{v^2}$, 6. $\frac{d(e^{cu})}{dx} = ce^{cu}\frac{du}{dx}$,

7. $\frac{d(c^u)}{dx} = (\ln c)c^u\frac{du}{dx}$, 8. $\frac{d(\ln u)}{dx} = \frac{1}{u}\frac{du}{dx}$,

9. $\frac{d(\sin u)}{dx} = \cos u\frac{du}{dx}$, 10. $\frac{d(\cos u)}{dx} = -\sin u\frac{du}{dx}$,

11. $\frac{d(\tan u)}{dx} = \sec^2 u\frac{du}{dx}$, 12. $\frac{d(\cot u)}{dx} = \csc^2 u\frac{du}{dx}$,

13. $\frac{d(\sec u)}{dx} = \tan u \sec u\frac{du}{dx}$, 14. $\frac{d(\csc u)}{dx} = -\cot u \csc u\frac{du}{dx}$,

15. $\frac{d(\arcsin u)}{dx} = \frac{1}{\sqrt{1-u^2}}\frac{du}{dx}$, 16. $\frac{d(\arccos u)}{dx} = \frac{-1}{\sqrt{1-u^2}}\frac{du}{dx}$,

17. $\frac{d(\arctan u)}{dx} = \frac{1}{1+u^2}\frac{du}{dx}$, 18. $\frac{d(\operatorname{arccot} u)}{dx} = \frac{-1}{1+u^2}\frac{du}{dx}$,

19. $\frac{d(\operatorname{arcsec} u)}{dx} = \frac{1}{u\sqrt{1-u^2}}\frac{du}{dx}$, 20. $\frac{d(\operatorname{arccsc} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}}\frac{du}{dx}$,

21. $\frac{d(\sinh u)}{dx} = \cosh u\frac{du}{dx}$, 22. $\frac{d(\cosh u)}{dx} = \sinh u\frac{du}{dx}$,

23. $\frac{d(\tanh u)}{dx} = \operatorname{sech}^2 u\frac{du}{dx}$, 24. $\frac{d(\coth u)}{dx} = -\operatorname{csch}^2 u\frac{du}{dx}$,

25. $\frac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u \tanh u\frac{du}{dx}$, 26. $\frac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \coth u\frac{du}{dx}$,

27. $\frac{d(\operatorname{arcsinh} u)}{dx} = \frac{1}{\sqrt{1+u^2}}\frac{du}{dx}$, 28. $\frac{d(\operatorname{arccosh} u)}{dx} = \frac{1}{\sqrt{u^2-1}}\frac{du}{dx}$,

29. $\frac{d(\operatorname{arctanh} u)}{dx} = \frac{1}{1-u^2}\frac{du}{dx}$, 30. $\frac{d(\operatorname{arccoth} u)}{dx} = \frac{1}{u^2-1}\frac{du}{dx}$,

31. $\frac{d(\operatorname{arcsech} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}}\frac{du}{dx}$, 32. $\frac{d(\operatorname{arccsch} u)}{dx} = \frac{-1}{|u|\sqrt{1+u^2}}\frac{du}{dx}$,

DP Optimizations

Name	Original Recurrence	Sufficient Condition of Applicability	Original Complexity	Optimized Complexity
Convex Hull Optimization1	$dp[i] = min_{j<i}\{dp[j] + b[j] \star a[i]\}$	$b[j] \geq b[j+1]$ optionally $a[i] \leq a[i+1]$	$O(n^2)$	$O(n)$
Convex Hull Optimization2	$dp[i][j] = min_{k<j}\{dp[i-1][k] + b[k] * a[j]\}$	$b[k] \geq b[k+1]$ optionally $a[j] \leq a[j+1]$	$O(kn^2)$	$O(kn)$
Divide and Conquer Optimization	$dp[i][j] = min_{k<j}\{dp[i-1][k] + C[k][j]\}$	$A[i][j] \leq A[i][j+1]$	$O(kn^2)$	$O(knlogn)$
Knuth Optimization	$dp[i][j] = min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$	$A[i,j-1] \leq A[i,j] \leq A[i+1,j]$	$O(n^3)$	$O(n^2)$

Notes:

- `A[i][j]` - the smallest k that gives optimal answer, for example in `dp[i][j] = dp[i - 1][k] + C[k][j]`
- `C[i][j]` — some given cost function
- We can generalize a bit in the following way: $dp[i] = minj < i\{F[j] + b[j] * a[i]\}$, where $F[j]$ is computed from $dp[j]$ in constant time.
- It looks like **Convex Hull Optimization2** is a special case of **Divide and Conquer Optimization**.
- It is claimed (in the references) that **Knuth Optimization** is applicable if $C[i][j]$ satisfies the following 2 conditions:
- **quadrangle inequality**: $C[a][c] + C[b][d] \leq C[a][d] + C[b][c],\ a \leq b \leq c \leq d$
- **monotonicity**: $C[b][c] \leq C[a][d],\ a \leq b \leq c \leq d$
- It is claimed (in the references) that the recurrence $dp[j] = mini < j\{dp[i] + C[i][j]\}$ can be solved in $O(nlogn)$ (and even $O(n)$) if $C[i][j]$ satisfies **quadrangle inequality**.

Divide & Conquer O(N^2)

```
// condition: L(N, K - 1) ≤ L(N, K) ≤ L(N + 1, K)
inline void proc(int j) {
  ll &ret = pd[act][j];
  if (j <= 1) {
    opt[act][j] = ret = 0;
    return; }
  ret = oo;
  int l = opt[ant][j];
  int r = j != n ? opt[act][j + 1] : n;
  fr(k, l, r+1) {
    ll op = pd[ant][k] + C[k][j];
    if (op < ret) {
      ret = op;
      opt[act][j] = k; } } }

pd[0][0] = opt[0][0] = 0;
fr(j, 1, n+1) {
  pd[0][j] = oo;
  opt[0][j] = 0; }

act = 1, ant = 0;
fr(i, 1, n) {
  for (int j = n; j >= 1; --j) {
    proc(j); }
  // look at pd[act][n]
  swap(ant, act); }

proc(n);
// look at pd[act][n]
```

Divide & Conquer O(NK log N)

```
calculaF(min_N, max_N, K, min_i, max_i):
mid = (min_N + max_N)/2
calcula F(mid, K) considerando i entre min_i e max_i
opt = i ótimo para mid entre min_i e max_i
calculaF(min_N, mid-1, min_i, opt)
calculaF(mid+1, max_N, opt, max_i)
```

.vimrc

```
set ai ts=4 sw=2 st=2 et nu rnu hls acd
syntax enable
filetype plugin indent on

map <F4> :w<CR>:!for x in *.in; do echo $x; ./a.out <
$x; echo; done<CR>
```

Convex Hull 3D

```
struct P {
  double x, y, z;
  P() {}
  P(double x, double y, double z) : x(x), y(y), z(z) {}
  P operator - (const P& p) { return P(x - p.x, y - p.y, z - p.z); }
  P operator + (const P& p) { return P(x + p.x, y + p.y, z + p.z); }
  P operator * (double c) { return P(x * c, y * c, z * c); }
  P operator % (const P& p) {
    return P(y * p.z - z * p.y,
             z * p.x - x * p.z,
             x * p.y - y * p.x);
  }
  double operator * (const P& p) { return x * p.x + y * p.y + z * p.z; }
  double operator ! () { return sqrt(x * x + y * y + z * z); }
  bool operator == (const P& p) const {
    return make_tuple(x, y, z) == make_tuple(p.x, p.y, p.z);
  }
  bool operator < (const P& p) const {
    return make_tuple(x, y, z) < make_tuple(p.x, p.y, p.z);
}};

const int M = 1010;
P p[M], co;
vector<int> face[M << 3];
int aresta[M][M], f, n;

bool comp2(P a, P b) {
  double d = a.x * b.y - a.y * b.x;
  return d > 0 || d == 0 && a.x * a.x + a.y * a.y < b.x * b.x + b.y * b.y;
}
bool comp3(P a, P b) {
  double d = a.x * b.y - a.y * b.x;
  return d >= 0; }

void ch2d(vector<int>& w, P X, P Y) {
  int n = w.size(); vector<P> q;
  fr(i,0,n) q.emplace_back(p[w[i]] * X, p[w[i]] * Y, w[i]);
  P o = *min_element(q.begin(), q.end());
  o.z = 0;
  fr(i,0,n) q[i] = q[i] - o;
  sort(q.begin(), q.end(), comp2);
  int m = 0;
  for (int i = 1; i < n; ++i) {
    while (m && comp3(q[i] - q[m - 1], q[m] - q[m - 1])) m--;
    q[++m] = q[i];
  }
  w.resize(++m);
  fr(i,0,m) w[i] = q[i].z + 0.5;
}
```

```
void go(int a, int b) {
  if (~aresta[a][b]) return;
  P A = p[a], v = p[b] - A, w = co - A;
  vector<int> plano;
  fr(i,0,n) if (i != a && i != b) {
    P u = v % (p[i] - A);
    if (u * w > 0) plano = vector<int>(1, i), w = p[i] - A;
    else if (u * w == 0) plano.push_back(i);
  }
  plano.push_back(a);
  plano.push_back(b);
  ch2d(plano, v, (co - A) % v);
  face[f++] = plano;
  int m = plano.size();
  fr(i,0,m) aresta[plano[i]][plano[(i+1)%m]] = f;
  fr(i,0,m) go(plano[(i+1)%m], plano[i]);
}

bool meo(P a, P b) {
  P v = (a) % (b);
  return v.z != 0 ? v.z < 0
                  : v.y != 0 ? v.y < 0 : v.x != 0 ? v.x > 0 : !(a) > !(b);
}

void convex() {
  sort(p, p + n); n = unique(p, p + n) - p;
  int a = min_element(p, p + n) - p, b = (a + 1) % n;
  co = p[a];
  fr(i,0,n) if (i != a) {
    co = co + p[i];
    if (meo(p[i]-p[a], p[i]-p[b])) b = i;
  }
  co = co * (1. / n);
  fr(i,0,n) fr(j,0,n) aresta[i][j] = -1, f = 0;
  go(a,b); go(b,a);
}

// to get area and volume
convex();
double area = 0, vol = 0;
fr(i,0,f) {
  int m = face[i].size();
  P a = p[face[i][0]];
  fr(j,2,m) {
    P b = p[face[i][j-1]];
    P c = p[face[i][j]];
    area += !((b - a) % (c - a)) / 2;
    vol += c % b * a / 6;
  }
}
```