Notebook

vici

October 25, 2013

Contents

1 Basic

1.1 Header

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <string>
#include <sstream>
#include <vector>
#include <queue>
#include <set>
#include <map>
#include <ctime>

#define inf 0x3f3f3f3f
#define Inf 0x3FFFFFFFFFFFFFFFLL
#define rep(i, n) for (int i = 0; i < (n); ++i)
#define Rep(i, n) for (int i = 1; i <= (n); ++i)
#define clr(x, a) memset(x, (a), sizeof x)
typedef double db;
typedef long long ll;

using namespace std;

int main() {

  return 0;
}
```

2 Maths

2.1 Gcd & Lcm

```
int gcd(int a, int b) { return b ? gcd(b, a % b) : a;}
int lcm(int a, int b) { return a / gcd(a, b) * b; }
db fgcd(db a, db b) {
  if(b > -eps && b < eps) return a;
  else return fgcd( b, fmod(a, b) );
}
```

2.2 PowMod & MulMod

```
//powMod
ll powMod(ll a, ll b, ll c) {
  ll ret = 1 % c;
  for (; b; a = a * a % c, b >>= 1)
    if(b & 1) ret = ret * a % c;
  return ret;
}
//powMod plus
ll mulMod(ll a, ll b, ll c) {
  ll ret = 0;
  for (; b; a = (a << 1) % c, b >>= 1)
    if (b & 1) ret = (ret + a) % c;
  return ret;
}
ll powMod(ll a, ll b, ll c) {
  ll ret = 1 % c;
  for (; b; a = mulMod(a, a, c), b >>= 1)
    if (b & 1) ret = mulMod(ret, a, c);
  return ret;
}
// mulMod plus for a, b, c <= 2^40
ll mulMod(ll a, ll b, ll c) {
  return (((a*(b>>20)%c)<<20) + a*(b&((1<<20)-1)))%c;
}
```

2.3 Extgcd

```
ll ext_gcd(ll a, ll b, ll &x, ll &y) {
  ll t, ret;
  if (!b) {
    x = 1, y = 0;
    return a;
  }
  ret = ext_gcd(b, a % b, x, y);
  t = x, x = y, y = t - a / b * y;
  return ret;
}
```

2.4 Inverse

```
// (b / a) % c
ll inv(ll a, ll b, ll c) {
  ll x, y;
  ext_gcd(a, c, x, y);
  return (1LL * x * b % c + c) % c;
}
//sieve inv
int inv[N];
void sieve_inv() {
  inv[1] = 1;
  for (int i = 2; i < N; ++i)
    inv[i] = inv[mod % i] * (mod - mod / i) % mod;
}
```

2.5 Sieve Primes

```
//mark[i]: the minimum factor of i (for prime, mark[i] = i)
int pri[N], mark[N], cnt;
void sieve() {
  cnt = 0, mark[0] = mark[1] = 1;
  for (int i = 2; i < N; ++i) {
    if (!mark[i]) pri[cnt++] = mark[i] = i;
    for (int j = 0; pri[j] * i < N; ++j) {
      mark[i * pri[j]] = pri[j];
      if (!(i % pri[j])) break;
    }
  }
}
```

2.6 Phi

```
//phi
int phi(int n) {
  int ret = n;
  for (int i = 2; i * i <= n; i += (i != 2) + 1) {
    if (!(n % i)) {
      ret = ret / i * (i - 1);
      while (n % i == 0) n /= i;
    }
  }
  if (n > 1) ret = ret / n * (n - 1);
  return ret;
}
//phi plus (sieve() first & (n < N))
int phi(int n) {
  int ret = n, t;
  while ((t = mark[n]) != 1) {
    ret = ret / t * (t - 1);
    while (mark[n] == t) n /= mark[n];
  }
  return ret;
}
//sieve phi
int pri[N], phi[N], cnt;
void sieve_phi() {
```

```
  cnt = 0, phi[1] = 1;
  for (int i = 2; i < N; ++i) {
    if (!phi[i]) pri[cnt++] = i, phi[i] = i - 1;
    for (int j = 0; pri[j] * i < N; ++j) {
      if (!(i % pri[j])) {
        phi[i * pri[j]] = phi[i] * pri[j];
        break;
      } else {
        phi[i * pri[j]] = phi[i] * (pri[j] - 1);
      }
    }
  }
}
```

2.7 The Number of Divisors

```
//the number of divisors
int d_func(int n) {
  int ret = 1, t = 1;
  for (int i = 2; i * i <= n; i += (i != 2) + 1) {
    if (!(n % i)) {
      while (!(n % i)) ++t, n /= i;
      ret *= t, t = 1;
    }
  }
  return n > 1 ? ret << 1 : ret;
}
//sieve the number of divisors (O(nlogn))
int nod[N];
void sieve_nod() {
  for (int i = 1; i < N; ++i)
    for (int j = i; j < N; j += i)
      ++nod[j];
}
//sieve the number of divisors (O(n))
int pri[N], e[N], divs[N], cnt;
void sieve_nod() {
  cnt = 0, divs[0] = divs[1] = 1;
  for (int i = 2; i < N; ++i) {
    if (!divs[i]) divs[i] = 2, e[i] = 1, pri[cnt++] = i;
    for (int j = 0; i * pri[j] < N; ++j) {
      int k = i * pri[j];
      if (i % pri[j] == 0) {
        e[k] = e[i] + 1;
        divs[k] = divs[i] / (e[i] + 1) * (e[i] + 2);
        break;
      } else {
        e[k] = 1, divs[k] = divs[i] << 1;
      }
    }
  }
}
```

2.8 The Sum of All Divisors

```
int ds_func(int n) {
  int ret = 1, t;
  for (int i = 2; i * i <= n; i += (i != 2) + 1) {
    if (!(n % i)) {
      t = i * i, n /= i;
      while (!(n % i)) t *= i, n /= i;
      ret *= (t - 1) / (i - 1);
    }
  }
  return n > 1 ? ret * (n + 1) : ret;
}
```

2.9 Miller-Rabin

```
bool suspect(ll a, int s, ll d, ll n) {
  ll x = powMod(a, d, n);
  if (x == 1) return true;
  for (int r = 0; r < s; ++r) {
    if (x == n - 1) return true;
    x = mulMod(x, x, n);
  }
  return false;
}
// {2,7,61,-1} is for n < 4759123141 (2^32)
int const test[] = {2,3,5,7,11,13,17,19,23,-1}; // for n < 10^16
bool isPrime(ll n) {
  if (n <= 1 || (n > 2 && n % 2 == 0)) return false;
  ll d = n - 1, s = 0;
  while (d % 2 == 0) ++s, d /= 2;
  for (int i = 0; test[i] < n && ~test[i]; ++i)
    if (!suspect(test[i], s, d, n)) return false;
  return true;
}
```

2.10 Pollard-Rho

```
ll pollard_rho(ll n, ll c) {
  ll d, x = rand() % n, y = x;
  for (ll i = 1, k = 2; ; ++i) {
    x = (mulMod(x, x, n) + c) % n;
    d = gcd(y - x, n);
    if (d > 1 && d < n) return d;
    if (x == y) return n;
    if (i == k) y = x, k <<= 1;
  }
  return 0;
}
```

2.11 Find Factors

```
//find factors
int facs[N];
int find_fac(int n) {
  int cnt = 0;
  for(int i = 2; i * i <= n; i += (i != 2) + 1)
```

```
    while (!(n % i)) n /= i, facs[cnt++] = i;
  if (n > 1) facs[cnt++] = n;
  return cnt;
}
//find factors plus (sieve() first & (n < N))
int facs[N];
int find_fac(int n) {
  int cnt = 0;
  while (mark[n] != 1)
    facs[cnt++] = mark[n], n /= mark[n];
  return cnt;
}
```

2.12 Place n Balls into m Boxes

Balls	Boxes	Empty Boxes	Answer
Different	Different	Yes	m^n
Different	Different	No	$m!S(n,m)$
Different	Same	Yes	$S(n,1)+S(n,2)+\ldots+S(n,\min(n,m))$
Different	Same	No	$S(n,m)$
Same	Different	Yes	$C(n+m-1,n)$
Same	Different	No	$C(n-1,m-1)$
Same	Same	Yes	$F(n,m)$
Same	Same	No	$F(n-m,m)$

```
//+ mod if needed
ll C[N][N];
void Cinit() {
  for (int i = 0; i < N; ++i) {
    C[i][0] = 1;
    for (int j = 1; j <= i; ++j) {
      C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
    }
  }
}

ll S[N][N]; // Strling2[]
void Sinit() {
  S[0][0] = 1;
  for (int i = 1; i < N; ++i) {
    S[i][1] = 1;
    for (int j = 2; j <= i; ++j) {
      S[i][j] = S[i - 1][j - 1] + j * S[i - 1][j];
    }
  }
}

ll F[N][N];
void Finit() {
  for (int i = 0; i < N; ++i) F[i][1] = F[0][i] = 1;
  for (int i = 1; i < N; ++i) {
    for (int j = 2; j < N; ++j) {
      F[i][j] = F[i][j - 1];
      if (i >= j) F[i][j] += F[i - j][j];
    }
  }
}
```

3 Graph

3.1 Edges

```
int n, m; // |V|, |E|
int p[N], idx;
struct Edge {
  int u, w, next;
} e[M];

inline void addedge(int u, int v, int w) {
  e[idx].u = v;
  e[idx].w = w;
  e[idx].next = p[u];
  p[u] = idx++;
}
inline void init() {
  idx = 0;
  clr(p, 0xff);
}
```

3.2 Cut-Vertex

```
//+ DSU to maintain the size of each component
//* Cut-Vertex: (u) if (sc[u] > 1)
//* Cut-Bridge: (u, v) if (low[v] > dfn[u])
struct CutVertex {
  int dfn[N], low[N], sc[N], cnt;
  void dfs(int u, int pre) {
    dfn[u] = low[u] = ++cnt;
    for (int i = p[u]; ~i; i = e[i].next) {
      int v = e[i].u;
      if (!dfn[v]) {
        dfs(v, pre);
        low[u] = min(low[u], low[v]);
        if (low[v] >= dfn[u]) ++sc[u];
      }
      else low[u] = min(low[u], dfn[v]);
    }
    if (u != pre) ++sc[u];
  }
  void solve() {
    cnt = 0, clr(dfn, 0), clr(sc, 0);
    Rep(i, n) if (!dfn[i]) dfs(i, i);
  }
};
```

3.3 SCC (tarjan)

```
struct SCC {
  int top, cnt, cc, t, v;
  int st[N], dfn[N], low[N], col[N]; bool vis[N];
  void tarjan(int u) {
    dfn[u] = low[u] = ++cnt;
    st[++top] = u, vis[u] = 1;
    for (int i = p[u]; ~i; i = e[i].next) {
      v = e[i].u;
      if (!dfn[v]) {
        tarjan(v);
        low[u] = min(low[u], low[v]);
      }
      else if (vis[v]) low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
      do {
        t = st[top--];
        col[t] = cc;
        vis[t] = 0;
      } while (t != u);
      ++cc;
    }
  }
  void solve() {
    top = cnt = cc = 0;
    clr(vis, 0), clr(col, 0), clr(dfn, 0);
    Rep(i, n) if (!dfn[i]) tarjan(i);
  }
};
```

3.4 Floyd

```
int n, mp[N][N]; // clr(mp, 0x3f); mp[i][i] = 0;
void floyd() {
  rep(k, n) rep(i, n) rep(j, n)
    mp[i][j] = min(mp[i][j], mp[i][k] + mp[k][j]);
}
```

3.5 Dijkstra

```
priority_queue<pair<int, int> > Q;
int dis[N]; bool vis[N];

void dijkstra(int s) {
  int u, v, w;
  while (!Q.empty()) Q.pop(); clr(vis, 0), clr(dis, 0x3f);
  Q.push(make_pair(0, s)), dis[s] = 0;
  while (!Q.empty()) {
    pair<int, int> tmp = Q.top(); Q.pop();
    if (vis[u = tmp.second]) continue;
    else vis[u] = true;
    for (int i = p[u]; ~i; i = e[i].next) {
      v = e[i].u, w = e[i].w;
      if (!vis[v] && dis[u] + w < dis[v]) {
        dis[v] = dis[u] + w;
        Q.push(make_pair(-dis[v], v));
      }
    }
```

```
  }
}
```

3.6 SPFA

```
int dis[N]; bool vis[N];
int Q[N * N];

void spfa(int s) {
  int u, v, w, l(0), h(0);
  clr(vis, 0), clr(dis, 0x3f);
  Q[h++] = s, dis[s] = 0;
  while (l < h) {
    u = Q[l++];
    vis[u] = 0;
    for (int i = p[u]; ~i; i = e[i].next) {
      v = e[i].u, w = e[i].w;
      if (dis[u] + w < dis[v]) {
        dis[v] = dis[u] + w;
        if (!vis[v]) {
          vis[v] = 1;
          Q[h++] = v;
        }
      }
    }
  }
}
```

3.7 Hungary

```
/* Matching
|Minimum Vertex Cover| = |Maximum Matching|
|Maximum Independent Set| = |V| - |Maximum Matching|
|Minimum Path Cover| = |V| - |Maximum Matching| (Directed Acyclic Graph)
|Minimum Edge Cover| = |V| - |Maximum Matching| / 2 (Undirected Graph)
*/
// O(|V|*|E|)
int n, m; // |V(x)|, |V(y)|
int mp[N][N], matx[N], maty[N]; bool fy[N];

int path(int u) {
  rep(v, m) if (mp[u][v] && !fy[v]) {
    fy[v] = 1;
    if (!~maty[v] || path(maty[v])) {
      matx[u] = v, maty[v] = u;
      return 1;
    }
  }
  return 0;
}
int hungary() {
  int ret = 0;
  clr(matx, 0xff), clr(maty, 0xff);
  rep(i, n) if (!~matx[i]) {
    clr(fy, 0);
    ret += path(i);
  }
  return ret;
}
```

3.8 Prim

```
int n; // |V|
int mp[N][N], ml[N]; bool vis[N];

int prim() {
  int ret(0); clr(vis, 0), clr(ml, 0x3f); ml[0] = 0;
  rep(i, n) {
    int id(-1);
    rep(j, n) if (!vis[j] && (!~id || ml[j] < ml[id])) id = j;
    vis[id] = 1, ret += ml[id];
    rep(j, n) if (!vis[j] && mp[j][id] < ml[j]) ml[j] = mp[j][id];
  }
  return ret;
}
```

4 Data Structure

4.1 UnionSet

```
int fa[N], v[N];
int Find(int a) {
  if (fa[a] < 0) return a;
  else {
    int t = fa[a];
    fa[a] = Find(fa[a]);
    v[a] += v[t];
    return fa[a];
  }
}
//addEdge(b, a, c) -> Union(ra, rb, v[a] - v[b] + c)
void Union(int a, int b, int c = 0) {
  if (fa[a] < fa[b]) {
    fa[a] += fa[b], fa[b] = a, v[b] += c;
  }
  else {
    fa[b] += fa[a], fa[a] = b, v[a] -= c;
  }
}
void init() { clr(fa, 0xff), clr(v, 0); }
```

4.2 FenwickTree

```
struct FenwickTree {
  ll a[N];
  inline void init() { clr(a, 0); }
  inline int lowbit(int x) { return x & -x; }
  void update(ll p, ll c) {
    while (p < N) {
      a[p] += c;
      p += lowbit(p);
    }
  }
  ll query(ll p) {
    ll ret = 0;
    while (p > 0) {
      ret += a[p];
      p -= lowbit(p);
    }
    return ret;
  }
  int get_kth(ll k) {
    int now = 0;
    for(int i = 20; ~i; --i) { // for N ~ 1e6
      now |= (1 << i);
      if (now >= N || a[now] >= k)
        now ^= (1 << i);
      else k -= a[now];
    }
    return now + 1;
  }
};
```

4.3 RMQ

```
//+ pos if needed
struct RMQ {
  int lg[N], dmx[M][N]; // M = lg[N] + 1
  void init() {
    lg[0] = -1;
    Rep(i, n) {
      lg[i] = lg[i - 1] + !(i & (i - 1));
      dmx[0][i] = a[i]; // the original array
    }
    for (int i = 1; (1 << i) <= n; ++i) {
      for (int j = 1; j + (1 << i) - 1 <= n; ++j) {
        dmx[i][j] = max(dmx[i - 1][j], dmx[i - 1][j + (1 << i - 1)]);
      }
    }
  }
  int get_mx(int a, int b) { // a <= b
    int k = lg[b - a + 1];
    return max(dmx[k][a], dmx[k][b - (1 << k) + 1]);
  }
};
```

4.4 HashMap

```
struct HashMap {
  int p[M], v[M], f[M], idx; ll a[M];
  void init() { idx = 0, clr(p, 0xff); }
  void insert(int u, ll t) { //add
    int x = u % M;
    for (int i = p[x]; ~i; i = f[i]) {
      if (v[i] == u) {
        a[i] += t;
        return;
      }
    }
    a[idx] = t;
    v[idx] = u, f[idx] = p[x], p[x] = idx++;
  }
};
```

4.5 Tree_Linear

```
int L[N], R[N], _;
void dfs(int u, int pre) {
  L[u] = ++_;
  for (int i = p[u]; ~i; i = e[i].next) {
    if (e[i].u != pre) dfs(e[i].u, u);
  }
  R[u] = _;
}
```

5 String

5.1 KMP

```
int fail[N], len;
void buildF(char *p) {
  for (int i = 1, j = fail[0] = -1; i < len; ++i) {
    while (~j && p[i] != p[j + 1]) j = fail[j];
    fail[i] = j += p[i] == p[j + 1];
  }
}
int kmp(char *s, char *p) {
  len = strlen(p); buildF(p);
  int ret (0);
  for (int i = 0, j = -1; s[i]; ++i) {
    while (~j && s[i] != p[j + 1]) j = fail[j];
    if (s[i] == p[j + 1]) ++j;
    if (j == len - 1) {
      ++ret;
      j = fail[j];
    }
  }
  return ret;
}
// all repetends
vector<int> r; //empty
void get_r() {
  int now = len - 1;
  while (~now) {
    r.push_back(len - 1 - fail[now]);
    now = fail[now];
  }
}
```

5.2 Trie

```
int const N = 100100; // size
int const M = 32;  // length
struct Trie {
  int idx, cnt;
  struct Trie_Node {
    int id;
    Trie_Node *next[26];
    void init() {
      id = -1;
      clr(next, NULL);
    }
  } trie[N*M], root;

  int insert(char* s) {
    Trie_Node *p = &root;
    for (int i = 0; s[i]; ++i) {
      int j = s[i] - 'a';
      if (p -> next[j] == NULL) {
        trie[idx].init();
        p -> next[j] = &trie[idx++];
      }
      p = p -> next[j];
    }
    if (p -> id == -1) p -> id = cnt++;
    return p -> id;
  }
  void init() {
    root.init();
    idx = cnt = 0;
  }
};
```

5.3 Min_Representation

```
int mins(char *s, int n) {
  int i = 0, j = 1, len = 0, x, y;
  while(i < n && j < n && len < n) {
    x = i + len; if (x >= n) x -= n;
    y = j + len; if (y >= n) y -= n;
    if (s[x] == s[y]) ++len;
    else if (s[x] < s[y]) j += len + 1, len = 0;
    else i = j++, len = 0;
  }
  return i;
}
```

5.4 Manacher

```
struct Manacher {
  int p[N<<1], len; char str[N<<1];
  int id, ret; // maxPalindrome_idx, maxPalindrome_length
  void func() {
    int mx (0);
    rep(i, len) {
      if (mx > i) p[i] = min(p[id + id - i], mx - i);
      else p[i] = 1;
      for (; str[i + p[i]] == str[i - p[i]]; ++p[i]);
      ret = max(ret, p[i]);
      if (p[i] + i > mx) {
        mx = p[i] + i;
        id = i;
      }
    }
    --ret;
  }
  void cal(char *s) {
    // "aaa" -> "!#a#a#a#"
    len = 0; str[len++]= '!', str[len++] = '#';
    for (int i = 0; s[i]; ++i) {
      str[len++] = s[i];
      str[len++] = '#';
```

```
    }
    str[len] = 0;
    ret = 0;
    func();
  }
};
```

6 Geometry

6.1 Basic

```
db const pi = 3.141592653589793;
db const eps = 1e-10;
db const dnf = 1e+10;

inline int sgn(db x) { return (x > eps) - (x < -eps); }
inline db sqr(db x) { return x * x; }
inline db rtoa(db x) { return x * pi / 180.; }
inline db ator(db x) { return x * 180. / pi; }
```

6.2 Point

```
struct Point {
  db x, y;
  Point (db x = 0., db y = 0.): x(x), y(y) {}
  void in() {
    scanf("%lf␣%lf", &x, &y);
  }
  void out() { //+ eps to avoid -0.000
    printf("%.3lf␣%.3lf\n", x, y);
  }
  friend Point operator+ (Point const &a, Point const &b) {
    return Point(a.x + b.x, a.y + b.y);
  }
  friend Point operator- (Point const &a, Point const &b) {
    return Point(a.x - b.x, a.y - b.y);
  }
  friend Point operator* (Point const &a, db const &b) {
    return Point(a.x * b, a.y * b);
  }
  friend Point operator/ (Point const &a, db const &b) {
    return Point(a.x / b, a.y / b);
  }
  // det
  friend db operator^ (Point const &a, Point const &b) {
    return a.x * b.y - a.y * b.x;
  }
  // dot
  friend db operator* (Point const &a, Point const &b) {
    return a.x * b.x + a.y * b.y;
  }
  friend bool operator== (Point const &a, Point const &b) {
    return !sgn(a.x - b.x) && !sgn(a.y - b.y);
  }
  friend bool operator!= (Point const &a, Point const &b) {
    return sgn(a.x - b.x) || sgn(a.y - b.y);
  }
  friend bool operator< (Point const &a, Point const &b) {
    int ca = sgn(a.x - b.x), cb = sgn(a.y - b.y);
    return !ca ? !~cb : !~ca ;
  }
  // rotate (anti-clockwise)
  friend Point operator& (Point const &a, db const &b) {
    return Point(a.x * cos(b) - a.y * sin(b),
                 a.x * sin(b) + a.y * cos(b));
  }
  // rotate (clockwise)
  friend Point operator% (Point const &a, db const &b) {
    return Point(a.x * cos(b) + a.y * sin(b),
                -a.x * sin(b) + a.y * cos(b));
  }
  Point &operator+= (Point const &b) {
    x += b.x, y += b.y;
    return *this;
  }
  Point &operator-= (Point const &b) {
    x -= b.x, y -= b.y;
    return *this;
  }
  Point &operator*= (db const &b) {
    x *= b, y *= b;
    return *this;
  }
  Point &operator/= (db const &b) {
    x /= b, y /= b;
    return *this;
  }
  // rotate (anti-clockwise)
  Point &operator&= (db const &b) {
    db t = x;
      x = x * cos(b) - y * sin(b), y = t * sin(b) + y * cos(b);
    return *this;
  }
  // rotate (clockwise)
  Point &operator%= (db const &b) {
    db t = x;
      x = x * cos(b) + y * sin(b), y = -t * sin(b) + y * cos(b);
    return *this;
  }
  Point stz() {
    return *this / len();
  }
  db len() {
    return sqrt(sqr(x) + sqr(y));
  }
  db lens() {
    return sqr(x) + sqr(y);
  }
  db ang() {
    return atan2(y, x);
  }
  db ang2(Point b) {
```

```
    return acos((*this * b) / len() / b.len());
  }
};

// collinearity
bool collinear(Point a, Point b, Point c) {
  return !sgn((b - a) ^ (c - a));
}
```

6.3 Line

```
struct Line {
  Point a, b;
  Line () {}
  Line (Point a, Point b) : a(a), b(b) {}
  void in() {
    scanf("%lf %lf %lf %lf", &a.x, &a.y, &b.x, &b.y);
  }
  void out() {
    printf("%.3lf %.3lf %.3lf %.3lf\n", a.x, a.y, b.x, b.y);
  }
  db len() {
    return (a - b).len();
  }
  bool parallel(Line const &t) {
    return !sgn((b - a) ^ (t.b - t.a));
  }
  bool vertical(Line const &t) {
    return !sgn((b - a) * (t.b - t.a));
  }
  bool intersect_seg(Line const &t) {
    db A = (b - a) ^ (t.b - t.a);
    db B = (b - a) ^ (b - t.a);
    db C = (t.b - t.a) ^ (b - a);
    db D = (t.b - t.a) ^ (t.b - a);
    if (!sgn(A)) {
      return sgn(B) ? 0 : sgn((t.a - a) * (t.a - b)) <= 0
                      || sgn((t.b - a) * (t.b - b)) <= 0
                      || sgn((a - t.a) * (a - t.b)) <= 0;
    }
    else return sgn(B / A) >= 0      //t-
            && sgn(B / A - 1) <= 0  //t+
            && sgn(D / C) >= 0      //s-
            && sgn(D / C - 1) <= 0; //s+
  }

  bool dot_online(Point const &p) {
    return !sgn((a - p) ^ (b - p)) && !sgn((p - a) * (p - b));
  }
  bool dot_inline(Point const &p) {
    return dot_online(p) && p != a && p != b;
  }
  Point intersect(Line const &t) { // check parallel first
    db A = (a - t.a) ^ (t.b - t.a), B = (b - t.a) ^ (t.b - t.a);
    return (b * A - a * B) / (A - B);
  }

  Point proj(Point const &p) {
    return (b - a) * ((b - a) * (p - a)) / (b - a).lens() + a;
  }

  db dis_ptoseg(Point const &p) {
    if (sgn((p - a) ^ (b - a)) < 0) return (p - a).len();
    else if (sgn((p - b) ^ (a - b)) < 0) return (p - b).len();
    else return fabs((a - p) ^ (b - p)) / len();
  }
  db dis_ptoline(Point const &p) {
    return fabs((a - p) ^ (b - p)) / len();
  }

  Line move(db const &d) {
    Point p = (b - a).stz() & (0.5 * pi);
    return Line(a + p * d, b + p * d);
  }
};
```

6.4 Triangle

```
struct Triangle {
  Point a, b, c;
  Triangle() { }
  Triangle(Point a, Point b, Point c): a(a), b(b), c(c) { }
  db area() { return 0.5 * fabs((b - a) ^ (c - a)); }
  // Mass_Center
  Point mass() {
    return (a + b + c) / 3.;
  }
  // Circum_Center
  Point circum() {
    db a1 = b.x - a.x, b1 = b.y - a.y, c1 = .5 * (sqr(a1) + sqr(b1));
    db a2 = c.x - a.x, b2 = c.y - a.y, c2 = .5 * (sqr(a2) + sqr(b2));
    db d = a1 * b2 - a2 * b1;
    return Point(a.x + (c1*b2 - c2*b1) / d, a.y + (a1*c2 - a2*c1) / d);
  }
  // Ortho_Center
  Point ortho() {
    return mass() * 3.0 - circum() * 2.0;
  }
  // Inner_Center
  Point inner() {
    db la = (b - c).len(), lb = (c - a).len(), lc = (a - b).len();
    return Point((la*a.x + lb*b.x + lc*c.x), (la*a.y + lb*b.y + lc*c.y))
            / (la + lb + lc);
  }
} ;
```

6.5 Graham

```
int n; Point p[N], ans[N];
bool xmult(Point a, Point b, Point c) {
  return sgn((b - a) ^ (c - a)) <= 0;
}

int Graham() {
  int now = 1, top; sort(p, p + n); p[n] = p[0];
  rep(i, 3) {
    if (n == i) return i;
    ans[i] = p[i];
  }
  for (int i = 2; i < n; ++i) {
    while (now && xmult(ans[now - 1], ans[now], p[i])) --now;
    ans[++now] = p[i];
  }
  top = now, ans[++now] = p[n - 2];
  for (int i = n - 3; ~i; --i) {
    while (now != top && xmult(ans[now - 1], ans[now], p[i])) --now;
    ans[++now] = p[i];
  }
  ans[now] = ans[0];
  return now;
};
```

6.6 N Circles cover [1-K] times

```
int n; Point p[N]; db r[N];

struct CIRUT {
  pair<db, int> e[N << 1];
  db ans[N]; int cnt;
  inline int rlt(int a, int b) {
    db d = (p[a] - p[b]).len();
    int s1 = sgn(d - r[a] + r[b]), s2 = sgn(d - r[b] + r[a]);
    if (s1 < 0 || !s1 && (d > eps || a > b)) return 0;
    if (s2 < 0 || !s2 && (d > eps || a < b)) return 1;
    return d < r[a] + r[b] - eps ? 2 : 3;
  }
  inline db areaArc(Point o, db r, db ang1, db ang2) {
    Point a(o.x + r * cos(ang1), o.y + r * sin(ang1));
    Point b(o.x + r * cos(ang2), o.y + r * sin(ang2));
    db dif = ang2 - ang1;
    return 0.5 * ((a ^ b) + (dif - sin(dif)) * r * r);
  }
  void cal() {
    rep(i, n + 1) ans[i] = 0.;
    db last; Point x, y;
    rep(i, n) if (r[i] > eps) {
      int acc = 0; cnt = 0;
      e[cnt++] = make_pair(-pi, 1);
      e[cnt++] = make_pair(pi, -1);
      rep(j, n) if (i != j && r[j] > eps) {
        int rel = rlt(i, j);
        if (rel == 1) {
          e[cnt++] = make_pair(-pi, 1);
          e[cnt++] = make_pair(pi, -1);
        }
        else if (rel == 2) {
          db center = atan2(p[j].y - p[i].y, p[j].x - p[i].x);
          db ds = (p[i] - p[j]).lens();
          db ang = acos((sqr(r[i]) - sqr(r[j]) + ds)
                / (2.0 * r[i] * sqrt(ds)));
          db angX = center + ang, angY = center - ang;
          if (angX > pi) angX -= 2.0 * pi;
          if (angY < -pi) angY += 2.0 * pi;
          if (angX < angY) ++acc;
          e[cnt++] = make_pair(angX, -1);
          e[cnt++] = make_pair(angY, 1);
        }
      }
      sort(e, e + cnt); last = -pi;
      rep(j, cnt) {
        db tmp = areaArc(p[i], r[i], last, e[j].first);
        ans[acc] += tmp;
        ans[acc - 1] -= tmp;
        acc += e[j].second;
        last = e[j].first;
      }
    }
  }
} ;
```

6.7 Volume of a Tetrahedron

```
//AB, AC, AD, CD, BD, BC.
db calc(db a, db b, db c, db r, db p, db q) {
  a *= a, b *= b, c *= c, r *= r, p *= p, q *= q;
  db P1 = a * p * (-a + b + c - p + q + r);
  db P2 = b * q * (a - b + c + p - q + r);
  db P3 = c * r * (a + b - c + p + q - r);
  db P = a * b * r + a * c * q + b * c * p + p * q * r;
  return sqrt((P1 + P2 + P3 - P)) / 12.;
}
```

6.8 Ellipse's Circumference

```
db ellcir(db a, db b) {
  if (a < b) swap(a, b);
  db e2 = 1.0 - b * b / a / a, e = e2;
  db ret = 1.0, x = 1.0, y = 2.0, t = 0.25;
  rep(i, 10000) {
    ret -= t * e;
    t = t * x * (x + 2.0) / (y + 2.0) / (y + 2.0);
    x += 2.0, y += 2.0, e *= e2;
  }
  return 2.0 * pi * a * ret;
```

```
}
```

7 Appendix

7.1 Primes

n	Value
<= 1000	2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997
large	9973 99991 999983 9999991 9999989 999999937 9999999967 99999999977 999999999989 9999999999971 99999999999973 999999999999989 9999999999999937 99999999999999997

7.2 Other Constants

Name	Symbol	Value
Pi	π	3.14159265358979323846
Base of Natural Logarithms	e	2.71828182845904523536
Eulers Constant	γ	0.57721566490153286061
Golden Mean	ϕ	1.61803398874989484820
Square Root of 2	$\sqrt{2}$	1.41421356237309504880
Square Root of 3	$\sqrt{3}$	1.73205080756887729353

7.3 $C(n, m)$

```
// C[i][0] = 1;
// C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
```

n	Value
0	1
1	1 1
2	1 2 1
3	1 3 3 1
4	1 4 6 4 1
5	1 5 10 10 5 1
6	1 6 15 20 15 6 1
7	1 7 21 35 35 21 7 1
8	1 8 28 56 70 56 28 8 1
9	1 9 36 84 126 126 84 36 9 1
10	1 10 45 120 210 252 210 120 45 10 1

7.4 $S(n, m)$

```
/* Stirling number of the second kind:
   The number of ways to partition a set of
   n objects into k non-empty subsets
   S[i][1] = 1;
   S[i][j] = S[i - 1][j - 1] + j * S[i - 1][j];
*/
```

n	Value
1	1
2	1 1
3	1 3 1
4	1 7 6 1
5	1 15 25 10 1
6	1 31 90 65 15 1
7	1 63 301 350 140 21 1
8	1 127 966 1701 1050 266 28 1
9	1 255 3025 7770 6951 2646 462 36 1
10	1 511 9330 34105 42525 22827 5880 750 45 1

7.5 $F(n, m)$

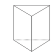
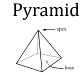
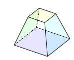
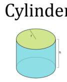
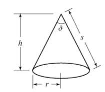
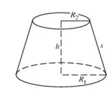
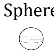
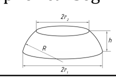
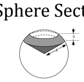
```
/* Euler's table:
   Number of partitions of n into at most m parts.
   F[i][1] = F[0][i] = 1;
   F[i][j] = F[i][j - 1] + (i >= j) ? F[i - j][j] : 0;
*/
```

n	Value
1	1
2	1 2
3	1 2 3
4	1 3 4 5
5	1 3 5 6 7
6	1 4 7 9 10 11
7	1 4 8 11 13 14 15
8	1 5 10 15 18 20 21 22
9	1 5 12 18 23 26 28 29 30
10	1 6 14 23 30 35 38 40 41 42

7.6 Geometry Formulas 2D

Shape	Property	Formula
Triangle	Semi-perimeter	$p = 0.5\,(a+b+c)$
	Area	$S = 0.5aH$ $S = ab\sin C$ $S = \sqrt{p\,(p-a)\,(p-b)\,(p-c)}$ $S = \frac{abc}{4R}$
	Midline	$M_a = 0.5\sqrt{2\,(b^2+c^2)-a^2}$ $M_a = 0.5\sqrt{b^2+c^2+2bccos(A)}$
	Bisector	$T_a = \frac{\sqrt{bc((b+c)^2-a^2)}}{b+c}$ $T_a = \frac{2bc\cos(0.5A)}{b+c}$
	Height	$H_a = bsin(C)$ $H_a = csin(B)$ $H_a = \sqrt{b^2 - \left(\frac{a^2+b^2-c^2}{2a}\right)^2}$
	Radius(Inscribed Circle)	$r = \frac{S}{p}$ $r = \arcsin\frac{B}{2}\sin\frac{C}{2}\sin\left(\frac{B+C}{2}\right)$ $r = 4R\sin\frac{A}{2}\sin\frac{B}{2}\sin\frac{C}{2}$ $r = \sqrt{\frac{(p-a)(p-b)(p-c)}{p}}$ $r = p\tan\frac{A}{2}\tan\frac{B}{2}\tan\frac{C}{2}$
	Radius(Circumcircle)	$R = \frac{abc}{4S} = \frac{a}{2\sin A} = \frac{b}{2\sin B} = \frac{c}{2\sin C}$
Quadrilateral	D_1, D_2: Diagonal A: Angle of D_1 and D_2 M: Dist of two Midpoints	$a^2+b^2+c^2+d^2 = {D_1}^2+{D_2}^2+4M^2$
	Area	$S = 0.5D_1D_2\sin A$
	For Concyclic	$ac+bd = D_1D_2$ $S = \sqrt{(p-a)\,(p-b)\,(p-c)\,(p-d)}$
N-gon	Central Angle	$A = \frac{2\pi}{n}$
	Interior Angle	$C = \frac{(n-2)\pi}{n}$
	Side Length	$a = 2\sqrt{R^2-r^2} = 2R\sin\frac{A}{2} = 2r\tan\frac{A}{2}$
	Area	$S = 0.5nar = nr^2\tan\frac{A}{2} = nR^2\sin\frac{A}{2} = \frac{na^2}{4\tan\frac{A}{2}}$
Circle	Arc	$l = rA$
	Chord 	$a = 2\sqrt{2hr-h^2} = 2r\sin\frac{A}{2}$
	Height of Bow-Shaped 	$h = r - \sqrt{r^2-\frac{a^2}{4}} = r\left(1-\cos\frac{A}{2}\right) = 0.5\arctan\frac{A}{4}$
	Area of the Fan-Shaped	$S_1 = 0.5rl = 0.5r^2A$
	Area of the Bow-Shaped	$S_2 = 0.5\,(rl - a\,(r-h)) = 0.5r^2\,(A-\sin A)$

7.7 Geometry Formulas 3D

Shape	Property	Formula
Prism 	Volume	$p = 0.5\,(a+b+c)$
	Curved Surface Area	$S = hp$
	Total Surface Area	$T = S + 2A$
Pyramid 	Volume	$V = \frac{Ah}{3}$
	Curved Surface Area	$S = 0.5lp$
	Total Surface Area	$T = S + A$
Frustum 	Volume	$V = \frac{1}{3}\left(A_1+A_2+\sqrt{A_1A_2}\right)h$
	Curved Surface Area	$S = 0.5\,(p_1+p_2)\,l$
	Total Surface Area	$T = S + A_1 + A_2$
Cylinder 	Volume	$V = \pi r^2 h$
	Curved Surface Area	$S = 2\pi rh$
	Total Surface Area	$T = 2\pi r\,(h+r)$
Cone 	Generating Line	$l = \sqrt{h^2+r^2}$
	Volume	$V = \frac{1}{3}\pi r^2 h$
	Curved Surface Area	$S = \pi rl$
	Total Surface Area	$T = \pi r\,(l+r)$
Truncated Cone 	Generating Line	$l = \sqrt{h^2+(r_1-r_2)^2}$
	Volume	$V = \frac{1}{3}\pi h\left({r_1}^2+{r_2}^2+r_1r_2\right)$
	Curved Surface Area	$S = \pi\,(r_1+r_2)\,l$
	Total Surface Area	$T = \pi r_1\,(l+r_1) + \pi r_2\,(l+r_2)$
Sphere 	Volume	$V = \frac{4}{3}\pi r^3$
	Total Surface Area	$T = 4\pi r^2$
Spherical Segment 	Volume	$V = \frac{1}{6}\pi h\left(3\left({r_1}^2+{r_2}^2\right)+h^2\right)$
	Curved Surface Area	$S = 2\pi rh$
	Total Surface Area	$T = \pi\left(2rh+{r_1}^2+{r_2}^2\right)$
Sphere Sector 	Volume	$V = \frac{2}{3}\pi r^2 h$
	Total Surface Area	$T = \pi r\,(2h+r_0)$