# C++11 CheatSheet

-Andrew

## Contents

**1 STL Useful Tips** **2**
1.1 General Template . . . 2
1.2 Compiling . . . 2
1.3 I/O . . . 2
1.4 C++ String operations . . . 3
1.5 Tricks in cmath . . . 3
1.6 Initialize array with predefined value . . . 3
1.7 2D Array Operations . . . 3
1.8 Searching . . . 4
1.9 Random algorithm . . . 4

**2 Computational Geom** **4**
2.1 C++ . . . 4
2.1.1 Constants . . . 4
2.1.2 Shape Structures . . . 4
2.1.3 General methods . . . 5
2.1.4 Angles . . . 6
2.1.5 Line Intersection . . . 6
2.1.6 Polygon related - Convex Hull . . . 6
2.1.7 Max Width Polygon . . . 7
2.1.8 Min Width Polygon . . . 8

**3 Number Theory** **8**
3.1 Prime number under 100 . . . 8
3.2 Max or min . . . 8
3.3 Greatest common divisor — GCD . . . 8
3.4 Least common multiple — LCM . . . 8
3.5 If prime number . . . 9
3.6 Prime factorization . . . 9
3.7 Leap year . . . 9
3.8 Binary exponiential . . . 9
3.9 $a^b$ `mod` $p$ . . . 10
3.10 Factorial `mod` . . . 10
3.11 Generate combinations . . . 10
3.12 Hungarian Algorithm . . . 11
3.13 Segment Trees . . . 12
3.14 10-ary to $m$-ary . . . 14
3.15 $m$-ary to 10-ary . . . 15
3.16 Binomial coefficient . . . 15
3.17 Catalan numbers . . . 15
3.18 Eulerian numbers . . . 15
3.19 Gauss Elimination . . . 16
3.20 Extras for 64 bit . . . 16
3.21 Pollard-Rho Factorization . . . 17
3.22 Polynomial Root . . . 17
3.23 Karatsuba algorithm in Java . . . 18
3.24 Euler's totient function . . . 19
3.25 Split plane . . . 19

**4 Searching Algorithms** **20**
4.1 Find rank $k$ in array . . . 20
4.2 KMP Algorithm . . . 20

**5 Graph Theory** **21**
5.1 Dikjstras C++11 . . . 21
5.2 SPFA — shortest path . . . 22
5.3 Tarjans – Strongly Connected . . . 22
5.4 Tarjans – Cut Vertex/Edge . . . 23
5.5 Bipartite Graph check . . . 24
5.6 Floyd-Warshall algorithm – shortest path of all pairs . . . 24
5.7 Prim — minimum spanning tree . . . 24
5.8 Eulerian circuit . . . 25
5.9 Topological sort . . . 26
5.10 3D convex . . . 27
5.11 Closest Pair Points . . . 29
5.12 2D base . . . 29
5.13 3D base . . . 31
5.14 Circle union . . . 32
5.15 Union Polygon . . . 33
5.16 Minimum circle . . . 36

# 1 STL Useful Tips

## 1.1 General Template

```cpp
#include <bits/stdc++.h>
using namespace std;
#define vi vector<int>
#define vii vector<vector<int>>
#define vpii vector<vector<pair<int, int>>>
#define pi pair<int, int>
#define add emplace_back
#define int long long

#undef int
int main(){
    #define int long long
  ios_base::sync_with_stdio(false); cin.tie(NULL);
  //freopen("input.in", "r", stdin);
  return 0;
}
```

## 1.2 Compiling

```
>g++ -std=c++11 X.cpp -o X
```

## 1.3 I/O

```
--
freopen("file.in", "r", stdin); //works with cin/cout
--

--
//Loads next n X's into the vector<X>
copy_n(istream_iterator<int>(cin), n, back_inserter(amps));
//where amps is a vector of type INT
--

--
//reads into a string
getline(cin, string_var);
```

```
//Can use like..
while (getline(cin, string_var)){} //reads till EOF

//CAREFUL -- If using cin >> before hand
//Make sure to trim /n.
//Can be achieved by cin >> std::ws;
--
```

## 1.4 C++ String operations

```
//For splitting a string with a single delim
//NOTE: be careful for empty elements
vector<string> split( std::string const& original, char separator )
{
    std::vector<std::string> results;
    auto start = original.begin();
    auto end = original.end();
    auto next = std::find( start, end, separator );
    while ( next != end ) {
        results.emplace_back( std::string( start, next ) );
        start = next + 1;
        next = std::find( start, end, separator );
    }
    results.emplace_back( std::string( start, next ) );
    return results;
}
```

## 1.5 Tricks in cmath

```
// when the number is too large. use powl instead of pow.
// will provide you more accuracy.
powl(a, b)
(int)round(p, (1.0/n)) // nth root of p
```

## 1.6 Initialize array with predefined value

```
// for 1d array, use STL fill_n or fill to initialize array
fill(a, a+size_of_a, value)
fill_n(a, size_of_a, value)
// for 2d array, if want to fill in 0 or -1
memset(a, 0, sizeof(a));
  // otherwise, use a loop of fill or fill_n through every a[i]
  fill(a[i], a[i]+size_of_ai, value) // from 0 to number of row.
```

## 1.7 2D Array Operations

```
//Rotate array (x,y) clockwise 90 deg
//n is size of array
for (int i=0; i<n/2; i++){
        for (int j=i; j<n-i-1; j++){
                tmp=a[i][j];
                a[i][j]=a[j][n-i-1];
                a[j][n-i-1]=a[n-i-1][n-j-1];
                a[n-i-1][n-j-1]=a[n-j-1][i];
                a[n-j-1][i]=tmp;
        }
```

```
}
```

```
\subsection{Permutations}
\begin{minted}[frame=lines, tabsize=2]{cpp}
bool next_permutation(iterator first, iterator last);
bool next_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
bool prev_permutation(iterator first, iterator last);
bool prev_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
```

### 1.8 Searching

```
// will return address of iterator, call result as *iterator;
iterator find(iterator first, iterator last, const T &value);
iterator find_if(iterator first, iterator last, const T &value, TestFunction test);
bool binary_search(iterator first, iterator last, const T &value);
bool binary_search(iterator first, iterator last, const T &value, LessThanOrEqualFunction comp);
```

### 1.9 Random algorithm

```
srand(time(NULL));
// generate random numbers between [a,b)
rand() % (b - a) + a;
// generate random numbers between [0,b)
rand() % b;
// generate random permutations
random_permutation(anArray, anArray + 10);
random_permutation(aVector, aVector + 10);
```

## 2 Computational Geom

### 2.1 C++

#### 2.1.1 Constants

```
const double eps = 1e-8;
const double PI = 3.14159265358979323846264433;
const double PIHALF = 1.57079632679489661923132;
```

#### 2.1.2 Shape Structures

```
struct point{
    double x, y;
    point(double x, double y): x(x), y(y){}
    point(){}
    bool operator == (const point& other)const {
        return x == other.x && y == other.y;
    }
    bool operator <(const point &p) const {
        return x < p.x || (x == p.x && y < p.y);
    }
    void operator -=(const point &other) {
        x -= other.x;
        y -= other.y;
    }
};
```

```
typedef pair<point,point> segment;

struct circle{
    point centre;
    double radius;
    circle(){}
    circle(double x, double y, double rad){
        centre.x = x;
        centre.y = y;
        radius = rad;
    }
    circle(point p, double rad): centre(p), radius(rad){}
};

struct line{
    point p1,p2;
    line(){}
    line(double x1, double y1, double x2, double y2){
        p1.x = x1;
        p1.y = y1;
        p2.x = x2;
        p2.y = y2;
    }
    line(point p1, point p2): p1(p1), p2(p2){}
};

typedef vector<point> polygon;
```

### 2.1.3 General methods

```
double cross(const point &O, const point &A, const point &B){
    return (A.x - O.x) * (double)(B.y - O.y) - (A.y - O.y) * (double)(B.x - O.x);
}

double dot(point a, point b){
    return a.x * b.x + a.y * b.y;
}

bool zero(double d){
    return d <= eps && d >= -eps;
}

bool smaller(double a, double b){
    return b - a > eps;
}

bool pointOnLine(point l1, point l2, point p){
    return zero(cross(l1,l2,p));
}

double dist(point a, point b){
  double dx = a.x - b.x;
  double dy = a.y - b.y;
  return sqrt(dx*dx + dy*dy);
}
```

### 2.1.4 Angles

```
//2D
dot = x1*x2 + y1*y2      # dot product
det = x1*y2 - y1*x2      # determinant
angle = atan2(det, dot)  # atan2(y, x) or atan2(sin, cos)

//3D
dot = x1*x2 + y1*y2 + z1*z2
lenSq1 = x1*x1 + y1*y1 + z1*z1
lenSq2 = x2*x2 + y2*y2 + z2*z2
angle = acos(dot/sqrt(lenSq1 * lenSq2))
```

### 2.1.5 Line Intersection

```
int turn(segment s, point q2)
{
    point q0 = s.first;
    point q1 = s.second;
    point d1 (q1.x-q0.x, q1.y-q0.y);
    point d2 (q2.x-q0.x, q2.y-q0.y);
    if (d1.x*(long)d2.y > d1.y*(long)d2.x) return 1;
    if (d1.x*(long)d2.y < d1.y*(long)d2.x) return -1;
    if ((d1.x*(long)d2.x < 0)||(d1.y*(long)d2.y < 0)) return -1;
    if ((d1.x*(long)d1.x+d1.y*(long)d1.y) < (d2.x*(long)d2.x+d2.y*(long)d2.y)) return 1;
    return 0;
}



bool intersect(segment s1, segment s2)
{
    bool turn1 = turn(s1, s2.first)*turn(s1, s2.second) <= 0;
    bool turn2 = turn(s2, s1.first)*turn(s2, s1.second) <= 0;
    if ((s1.first == s1.second)||(s2.first == s2.second))
        return (turn1 || turn2);
    return (turn1 && turn2);
}
```

### 2.1.6 Polygon related - Convex Hull

```
typedef vector<point> polygon;

double area(polygon &p) {
  double total = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i + 1) % p.size();
    total += (p[i].x * p[j].y) -
      (p[j].x * p[i].y);
  }
  return total / 2.0;
}

point centroid(polygon& p){
  double a = area(p);
  double xsum = 0;
  double ysum = 0;
```

```
  for(int i=0;i<p.size();i++){
    int j = (i+1) % p.size();
    xsum += (p[i].x + p[j].x)
          * (p[i].x * p[j].y - p[j].x * p[i].y);
    ysum += (p[i].y + p[j].y)
          * (p[i].x * p[j].y - p[j].x * p[i].y);
  }
  xsum /= 6 * a;
  ysum /= 6 * a;
  return point(xsum, ysum);
}

// Returns a list of points on the convex hull in counter-clockwise order.
// Note: the last point in the returned list is the same as the first one.
//OTHER note: To strip co-linear points, simply make the eps POSITIVE

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

### 2.1.7 Max Width Polygon

```
double area(const point &a, const point &b, const point &c) {
    return abs((b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x));
}

double dist(const point &a, const point &b) {
    return hypot(a.x - b.x, a.y - b.y);
}

double max_width(polygon p) {
    polygon h = convex_hull(p);
    int m = h.size();
    if (m == 1)
        return 0;
    if (m == 2)
        return dist(h[0], h[1]);
    int k = 1;
    while (area(h[m - 1], h[0], h[(k + 1) % m]) > area(h[m - 1], h[0], h[k]))
        ++k;
    double res = 0;
```

```
    for (int i = 0, j = k; i <= k && j < m; i++) {
        res = max(res, dist(h[i], h[j]));
        while (j < m && area(h[i], h[(i + 1) % m], h[(j + 1) % m]) > area(h[i], h[(i + 1) % m], h[j])) {
            res = max(res, dist(h[i], h[(j + 1) % m]));
            ++j;
        }
    }
    return res;
}
```

### 2.1.8 Min Width Polygon

```
double min_width(polygon p, polygon ch) {
    double ret = 1e+60;
  ch.pop_back(); // remove last element
  int i,j,m=ch.size(),k, n=p.size();
  for(i = 0, j = m-1; i < m; j = i, i++) {
    double na = -(ch[i].y-ch[j].y), nb = ch[i].x-ch[j].x;
    double c = -(na*ch[i].x+nb*ch[i].y);
    double len = hypot(na, nb);
    double farth = 0;
    for(k = 0; k < n; k++)
      farth = max(farth, fabs(na*p[k].x+nb*p[k].y+c)/(len));
    ret = min(ret, farth);
  }
  return ret;
}
```

# 3 Number Theory

## 3.1 Prime number under 100

```
// there are 25 numbers
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
```

## 3.2 Max or min

```
int max(int a, int b) { return a>b ? a:b; }
int min(int a, int b) { return a<b ? a:b; }
```

## 3.3 Greatest common divisor — GCD

```
int gcd(int a, int b)
{
  if (b==0) return a;
  else return gcd(b, a%b);
}
```

## 3.4 Least common multiple — LCM

```
int lcm(int a, int b)
{
```

```
  return a*b/gcd(a,b);
}
```

### 3.5 If prime number

```
bool prime(int n)
{
  if (n<2) return false;
  if (n<=3) return true;
  if (!(n%2) || !(n%3)) return false;
  for (int i=5;i*i<=n;i+=6)
    if (!(n%i) || !(n%(i+2))) return false;

  return true;
}
```

### 3.6 Prime factorization

```
// smallest prime factor of a number.
function factor(int n)
{
  int a;
  if (n%2==0)
    return 2;
  for (a=3;a<=sqrt(n);a++++)
  {
    if (n%a==0)
    return a;
  }
  return n;
}

// complete factorization
int r;
while (n>1)
{
    r = factor(n);
    printf("%d", r);
    n /= r;
}
```

### 3.7 Leap year

```
bool isLeap(int n)
{
  if (n%100==0)
    if (n%400==0) return true;
    else return false;

  if (n%4==0) return true;
  else return false;
}
```

### 3.8 Binary exponiential

```
int binpow (int a, int n)
{
  int res = 1;
  while (n)
    if (n & 1)
    {
      res *= a;
      --n;
    }
    else
    {
      a *= a;
      n >>= 1;
    }
  return res;
}
```

### 3.9 $a^b$ mod $p$

```
long powmod(long base, long exp, long modulus) {
  base %= modulus;
  long result = 1;
  while (exp > 0) {
    if (exp & 1) result = (result * base) % modulus;
    base = (base * base) % modulus;
    exp >>= 1;
  }
  return result;
}
```

### 3.10 Factorial mod

```
//n! mod p
int factmod (int n, int p) {
  long long res = 1;
  while (n > 1) {
    res = (res * powmod (p-1, n/p, p)) % p;
    for (int i=2; i<=n%p; ++i)
      res=(res*i) %p;
    n /= p;
  }
  return int (res % p);
}
```

### 3.11 Generate combinations

```
// n>=m, choose M numbers from 1 to N.
void combination(int n, int m)
{
  if (n<m) return ;
  int a[50]={0};
  int k=0;

  for (int i=1;i<=m;i++) a[i]=i;
  while (true)
  {
```

```
    for (int i=1;i<=m;i++)
      cout << a[i] << " ";
    cout << endl;

    k=m;
    while ((k>0) && (n-a[k]==m-k)) k--;
    if (k==0) break;
    a[k]++;
    for (int i=k+1;i<=m;i++)
      a[i]=a[i-1]+1;
  }
}
```

## 3.12 Hungarian Algorithm

```
struct Network {
    struct Edge {
        int to;
        int pre_edge;
        int cap;
        int flow;
    };

    #define MAXNODE 405
    int last[MAXNODE];

    int nv; // total number of vertex, index range: [0, nv)
    vector<Edge> edge;
    void init(int _nv) {
        nv = _nv;
        edge.clear();
        fill(last, last + nv, -1);
    }

    void add_e(int x, int y, int cap, int r_cap = 0) {
        Edge e = {y, last[x], cap, 0};
        // Edge e{y, last[x], cap, 0};
        last[x] = edge.size();
        // edge.push_back(move(e));
        edge.push_back(e);

        Edge r_e = {x, last[y], r_cap, 0};
        // Edge r_e{x, last[y], r_cap, 0};
        last[y] = edge.size();
        // edge.push_back(move(r_e));
        edge.push_back(r_e);
    }
    void show_edge() {
        for (int i = 0; i < nv; i++) {
            printf("v [%d]:", i);
            for (int ie = last[i]; ie != -1; ) {
                const Edge& e = edge[ie];
                ie = e.pre_edge;
                printf(" [%d]%d/%d", e.to, e.flow, e.cap);
            }
            printf("\n");
        }
        printf("\n");
```

```
    }

    //
    // bipartite match
    // O(V * E)
    int peer[MAXNODE];
    bool went[MAXNODE];
    int bipartite_match() {
        fill(peer, peer + nv, -1);
        int ans = 0;
        for (int i = 0; i < nv; i++) {
            if (last[i] == -1 || peer[i] != -1)
                continue;
            fill(went, went + nv, false);
            if (match(i))
                ans++;
        }
        return ans;
    }
    bool match(int cur) {
        for (int ie = last[cur]; ie != -1; ) {
            const Edge& e = edge[ie];
            ie = e.pre_edge;
            int to = e.to;
            if (went[to])
                continue;
            went[to] = true;
            if (peer[to] == -1 || match(peer[to])) {
                peer[to] = cur;
                peer[cur] = to;
                return true;
            }
        }
        return false;
    }
    void show_peer() {
        for (int i = 0; i < nv; i++)
            printf("%d peer-> %d\n", i, peer[i]);
    }
    // end of
    // bipartite match
    //
};
```

## 3.13 Segment Trees

```
#define MAXN 1024000
#define MAXH 21  // 1 + ceil(log2(MAXN))

// Flags to identify states. 0 is for "Nothing".
#define UP_SET 1
#define UP_CLR 2
#define UP_FLP 3

struct SegTree {
    vector<int> A;  // original array of integers
    vector<int> T;  // segment tree
    vector<int> U;  // segment tree for lazy propagation (the states)
```

```
int n;  // size of the array

SegTree(int N=0) : n(N) {
    A.resize(MAXN);
    T.resize(1 << MAXH);
    U.resize(1 << MAXH);
}

void init() { tree_init(1, 0, n-1); }
void tree_init(int x, int a, int b) {
    U[x] = 0;
    if (a == b) { T[x] = A[a]; return; }
    int lt = 2*x, rt = lt + 1, md = (a+b)/2;
    tree_init(lt, a, md);
    tree_init(rt, md + 1, b);
    T[x] = T[lt] + T[rt];
}

void set(int i, int j) { tree_set(i, j, 1, 0, n - 1); }
void tree_set(int i, int j, int x, int a, int b) {
    propagate(x, a, b);
    if (j < a || i > b) return;
    if (a == b) { T[x] = 1; return; }
    int lt = 2*x, rt = lt + 1, md = (a+b)/2;
    if (a >= i && b <= j) {
        T[x] = b - a + 1;
        U[lt] = U[rt] = UP_SET;
        return;
    }
    tree_set(i, j, lt, a, md);
    tree_set(i, j, rt, md + 1, b);
    T[x] = T[lt] + T[rt];
}

void clear(int i, int j) { tree_clear(i, j, 1, 0, n - 1); }
void tree_clear(int i, int j, int x, int a, int b) {
    propagate(x, a, b);
    if (j < a || i > b) return;
    if (a == b) { T[x] = 0; U[x] = 0; return; }
    int lt = 2*x, rt = lt + 1, md = (a+b)/2;
    if (a >= i && b <= j) {
        T[x] = 0;
        U[lt] = U[rt] = UP_CLR;
        return;
    }
    tree_clear(i, j, lt, a, md);
    tree_clear(i, j, rt, md + 1, b);
    T[x] = T[lt] + T[rt];
}

void flip(int i, int j) { tree_flip(i, j, 1, 0, n - 1); }
void tree_flip(int i, int j, int x, int a, int b) {
    propagate(x, a, b);
    if (j < a || i > b) return;
    if (a == b) {
        T[x] = T[x] == 1 ? 0 : 1;
        return;
    }
```

```
        int lt = 2*x, rt = lt + 1, md = (a+b)/2;
        if (a >= i && b <= j) {
            T[x] = (b - a + 1) - T[x];
            U[lt] = apply_flip(U[lt]);
            U[rt] = apply_flip(U[rt]);
            return;
        }
        tree_flip(i, j, lt, a, md);
        tree_flip(i, j, rt, md + 1, b);
        T[x] = T[lt] + T[rt];
    }

    int query(int i, int j) { return tree_query(i, j, 1, 0, n-1); }
    int tree_query(int i, int j, int x, int a, int b) {
        if (j < a || i > b) return -1;
        propagate(x, a, b);
        if (a >= i && b <= j) return T[x];
        int lt = 2*x, rt = lt + 1, md = (a+b)/2;
        int q1 = tree_query(i, j, lt, a, md);
        int q2 = tree_query(i, j, rt, md + 1, b);
        if (q1 < 0) return q2;
        if (q2 < 0) return q1;
        return q1 + q2;
    }

    int apply_flip(int v) {
        if (v == UP_SET) return UP_CLR;
        if (v == UP_CLR) return UP_SET;
        if (v == UP_FLP) return 0;
        return UP_FLP;
    }
    void propagate(int x, int a, int b) {
        if (U[x] == 0) return;
        if (U[x] == UP_SET)
            T[x] = b - a + 1;
        else if (U[x] == UP_CLR)
            T[x] = 0;
        else if (U[x] == UP_FLP)
            T[x] = (b - a + 1) - T[x];

        if (a != b) {
            int lt = 2*x, rt = lt + 1;
            if (U[x] == UP_SET || U[x] == UP_CLR)
                U[lt] = U[rt] = U[x];
            else
                U[lt] = apply_flip(U[lt]), U[rt] = apply_flip(U[rt]);
        }
        U[x] = 0;
    }
};
```

## 3.14 10-ary to $m$-ary

```
//takes in unsigned int and returns a string
//input is base 10.
//works from 2-32
string baseconvert(int num, int b){
  const string digits("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ");
```

```
    bool is_neg = num < 0;
    string result;
    for (; num; num /= b) {
        result.insert(result.begin(), digits[abs(num % b)]);
    }
    if (is_neg)
         result.insert(result.begin(), '-');
  return result;
}
```

### 3.15 $m$-ary to 10-ary

```
string num="0123456789ABCDE";

int mToTen(string n, int m)
{
  int multi=1;
  int result=0;

  for (int i=n.size()-1;i>=0;i--)
  {
    result+=num.find(n[i])*multi;
    multi*=m;
  }

  return result;
}
```

### 3.16 Binomial coefficient

```
#define MAXN 100 // largest n or m
long binomial_coefficient(n,m) // compute n choose m
int n,m;
{
    int i,j;
    long bc[MAXN][MAXN];
    for (i=0; i<=n; i++) bc[i][0] = 1;
    for (j=0; j<=n; j++) bc[j][j] = 1;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
           bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
    return bc[n][m];
}
```

### 3.17 Catalan numbers

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1}\binom{n}{k} \tag{1}$$

The first terms of this sequence are 2, 5, 14, 42, 132, 429, 1430 when $C_0 = 1$. This is the number of ways to build a balanced formula from $n$ sets of left and right parentheses. It is also the number of triangulations of a convex polygon, the number of rooted binary tress on $n+1$ leaves and the number of paths across a lattice which do not rise above the main diagonal.

### 3.18 Eulerian numbers

$$\left\langle {n \atop k} \right\rangle = k \left\langle {n-1 \atop k} \right\rangle + (n-k+1) \left\langle {n-1 \atop k-1} \right\rangle \tag{2}$$

```
// This is the number of permutations of length n with exactly k ascending sequences or runs.
// Basis: k=0 has value 1
#define MAXN 100 // largest n or k
long eularian(n,k)
int n,m;
{
    int i,j;
    long e[MAXN][MAXN];
    for (i=0; i<=n; i++) e[i][0] = 1;
    for (j=0; j<=n; j++) e[0][j] = 0;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
           e[i][j] = k*e[i-1][j] + (i-j+1)*e[i-1][j-1];
    return e[n][k];
}
```

### 3.19 Gauss Elimination

```
void eliminate(int n, double g[maxN][maxN]) { // n*(n+1), Ax=b
  for (int i = 1; i <= n; ++i) {
    double temp = 0;
    int pos = -1;
    for (int j = i; j <= n; ++j) {
      if (fabs(g[j][i]) > temp) temp = fabs(g[j][i]), pos = j;
    }
    if (pos == -1) continue;
    for (int k = 1; k <= n + 1; ++k) swap(g[pos][k], g[i][k]);
    temp = g[i][i];
    for (int k = 1; k <= n + 1; ++k) g[i][k] /= temp;
    for (int j = i + 1; j <= n; ++j) {
      temp = g[j][i];
      for (int k = 1; k <= n + 1; ++k) g[j][k] -= temp * g[i][k];
    }
  }
  for (int i = n; i >= 1; --i) {
    for (int j = 1; j < i; ++j) {
      g[j][n + 1] -= g[i][n + 1] * g[j][i];
      g[j][i] = 0;
    }
  }
}
```

### 3.20 Extras for 64 bit

```
LL mul64(LL a, LL b, LL m) { // 64-bit long long multiplication
  a %= m, b %= m;
  LL ret = 0;
  for (; b; b >>= 1) {
    if (b & 1) ret = (ret + a) % m;
    a = (a + a) % m;
  }
  return ret;
}

int fpow(int a, LL p, int mod) { // fast power-modulo
  int res = 1;
  for (; p; p >>= 1) {
```

```
    // use mul64 if there is overflow
    if (p & 1) res = (1ll * res * a) % mod;
    a = (1ll * a * a) % mod;
  }
  return res;
}
```

## 3.21 Pollard-Rho Factorization

```
LL pollard_rho(LL n, LL seed) { // always call factorize
  LL x, y;
  x = y = rand() % (n - 1) + 1;
  int head = 1, tail = 2;
  while (true) {
    x = mul64(x, x, n);
    x = (x + seed) % n;
    if (x == y) return n;
    LL d = gcd(max(x - y, y - x), n);
    if (1 < d && d < n) return d;
    if (++head == tail) y = x, tail <<= 1;
  }
}

void factorize(LL n, vector<LL> &divisor) { // pollard-rho factorization
  if (n == 1) return;
  if (ptest(n)) divisor.push_back(n);
  else {
    LL d = n;
    while (d >= n) d = pollard_rho(n, rand() % (n - 1) + 1);
    factorize(n / d, divisor);
    factorize(d, divisor);
  }
}

vector<LL> divisors(vector<LL> d) {
  vector<LL> ret = {1};
  sort(d.begin(), d.end());
  for (int i = 0, count = 1; i < d.size(); ++i) {
    if (i + 1 == d.size() || d[i] != d[i + 1]) {
      int c = ret.size();
      ret.resize(ret.size() * (count+1));
      LL n = 1;
      for (int j = 1; j <= count + 1; ++j) {
        for (int k = 0; k < c; ++k) {
          ret[(j-1)*c+k] = ret[k]*n;
        }
        n *= d[i];
      }
      count = 1;
    } else count += 1;
  }
  sort(ret.begin(), ret.end());
  return ret;
}
```

## 3.22 Polynomial Root

```cpp
double cal(const vector<double> &coef, double x) {
  double e = 1, s = 0;
  for (int i = 0; i < coef.size(); ++i) s += coef[i] * e, e *= x;
  return s;
}

double find(const vector<double> &coef, double l, double r, int sl, int sr) {
  int sl = dblcmp(cal(coef, l)), sr = dblcmp(cal(coef, r));
  if (sl == 0) return l;
  if (sr == 0) return r;
  for (int tt = 0; tt < 100 && r - l > eps; ++tt) {
    double mid = (l + r) / 2;
    int smid = dblcmp(cal(coef, mid));
    if (smid == 0) return mid;
    if (sl * smid < 0) r = mid;
    else l = mid;
  }
  return (l + r) / 2;
}

vector<double> rec(const vector<double> &coef, int n) { // c[n]==1
  vector<double> ret;
  if (n == 1) {
    ret.push_back(-coef[0]);
    return ret;
  }
  vector<double> dcoef(n);
  for (int i = 0; i < n; ++i) dcoef[i] = coef[i + 1] * (i + 1) / n;
  double b = 2;
  for (int i = 0; i <= n; ++i) b = max(b, 2 * pow(fabs(coef[i]), 1.0 / (n - i)));
  vector<double> droot = rec(dcoef, n - 1);
  droot.insert(droot.begin(), -b);
  droot.push_back(b);
  for (int i = 0; i + 1 < droot.size(); ++i) {
    int sl = dblcmp(cal(coef, droot[i])), sr = dblcmp(cal(coef, droot[i + 1]));
    if (sl * sr > 0) continue;
    ret.push_back(find(coef, droot[i], droot[i + 1], sl, sr));
  }
  return ret;
}

// solve c[0]+c[1]*x+c[2]*x^2+...+c[n]*x^n==0
vector<double> solve(vector<double> coef) {
  int n = coef.size() - 1;
  while (coef.back() == 0) coef.pop_back(), --n;
  for (int i = 0; i <= n; ++i) coef[i] /= coef[n];
  return rec(coef, n);
}
```

### 3.23 Karatsuba algorithm in Java

```java
// fast algorithm to find multiplication of two big numbers.
import java.math.BigInteger;
import java.util.Random;

class Karatsuba {
  private final static BigInteger ZERO = new BigInteger("0");
  public static BigInteger karatsuba(BigInteger x, BigInteger y)
```

```
  {
    int N = Math.max(x.bitLength(), y.bitLength());
    if (N <= 2000) return x.multiply(y);
    N=(N/2)+(N %2);
    BigInteger b = x.shiftRight(N);
    BigInteger a = x.subtract(b.shiftLeft(N));
    BigInteger d = y.shiftRight(N);
    BigInteger c = y.subtract(d.shiftLeft(N));
    BigInteger ac = karatsuba(a, c);
    BigInteger bd = karatsuba(b, d);
    BigInteger abcd = karatsuba(a.add(b), c.add(d));
    return ac.add(abcd.subtract(ac).subtract(bd).shiftLeft(N)).add(bd.shiftLeft(2*N));
  }

  public static void main(String[] args)
  {
    long start, stop, elapsed;
    Random random = new Random();
    int N = Integer.parseInt(args[0]);
    BigInteger a = new BigInteger(N, random);
    BigInteger b = new BigInteger(N, random);
    start = System.currentTimeMillis();
    BigInteger c = karatsuba(a, b);
    stop = System.currentTimeMillis();
    System.out.println(stop - start);
    start = System.currentTimeMillis();
    BigInteger d = a.multiply(b);
    stop = System.currentTimeMillis();
    System.out.println(stop - start);
    System.out.println((c.equals(d)));
  }
}
```

## 3.24 Euler's totient function

```
// the positive integers less than or equal to n that are relatively prime to n.
int phi (int n)
{
  int result = n;
  for (int i=2; i*i<=n; ++i)
    if(n %i==0)
    {
      while(n %i==0)
        n /= i;
      result -= result / i;
    }
  if (n > 1)
    result -= result / n;
  return result;
}
```

## 3.25 Split plane

$n$ lines can split a plane in $\frac{(n+1)n}{2} + 1$ sub-regions.

# 4 Searching Algorithms

## 4.1 Find rank $k$ in array

```
int find(int l, int r, int k)
{
  int i=0,j=0,x=0,t=0;

  if (l==r) return a[l];
  x=a[(l+r)/2];
  t=a[x]; a[x]=a[r]; a[r]=t;
  i=l-1;

  for (int j=l; j<=r-1;j++)
    if (a[j]<=a[r])
    {
      i++;
      t=a[i]; a[i]=a[j]; a[j]=t;
    }
  i++;
  t=a[i]; a[i]=a[r]; a[r]=t;
  if (i==k) return a[i];
  if (i<k) return find(i+1, r,k);

  return find(l, i-1, k);
}
```

## 4.2 KMP Algorithm

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

typedef vector<int> VI;

void buildTable(string& w, VI& t)
{
  t = VI(w.length());
  int i = 2, j = 0;
  t[0] = -1; t[1] = 0;

  while(i < w.length())
  {
    if(w[i-1] == w[j]) { t[i] = j+1; i++; j++; }
    else if(j > 0) j = t[j];
    else { t[i] = 0; i++; }
  }
}

int KMP(string& s, string& w)
{
  int m = 0, i = 0;
  VI t;

  buildTable(w, t);
  while(m+i < s.length())
```

```
  {
    if(w[i] == s[m+i])
    {
      i++;
      if(i == w.length()) return m;
    }
    else
    {
      m += i-t[i];
      if(i > 0) i = t[i];
    }
  }
  return s.length();
}

int main(void)
{
  string a = (string) "The example above illustrates the general technique for assembling "+
    "the table with a minimum of fuss. The principle is that of the overall search: "+
    "most of the work was already done in getting to the current position, so very "+
    "little needs to be done in leaving it. The only minor complication is that the "+
    "logic which is correct late in the string erroneously gives non-proper "+
    "substrings at the beginning. This necessitates some initialization code.";

  string b = "table";

  int p = KMP(a, b);
  cout << p << ": " << a.substr(p, b.length()) << " " << b << endl;

  return 0;
}
```

# 5 Graph Theory

## 5.1 Dikjstras C++11

```
struct edge { int to, length; };

int dijkstra(const vector< vector<edge> > &graph, int source, int target) {
    vector<int> min_distance( graph.size(), INT_MAX );
    min_distance[ source ] = 0;
    set< pair<int,int> > active_vertices;
    active_vertices.insert( {0,source} );

    while (!active_vertices.empty()) {
        int where = active_vertices.begin()->second;
        if (where == target) return min_distance[where];
        active_vertices.erase( active_vertices.begin() );
        for (auto edge : graph[where])
            if (min_distance[edge.to] > min_distance[where] + edge.length) {
                active_vertices.erase( { min_distance[edge.to], edge.to } );
                min_distance[edge.to] = min_distance[where] + edge.length;
                active_vertices.insert( { min_distance[edge.to], edge.to } );
            }
    }
    return INT_MAX;
}
```

## 5.2 SPFA — shortest path

```
vector<pair<int,int>> g[MAX];
int s;

//UVA: 558 - Wormholes
bool solve(int source){
  vector<int> dist(s, INT_MAX); dist[source] = 0;
  deque<int> q; q.push_front(source);
  vector<int> in_q(s, 0); in_q[source] = 1;
  vector<int> counter(s, 0); //if = V-1, then cycle
  int u,j,v,w;
  while (!q.empty()){
    u = q.front(); q.pop_front();
    in_q[u] = 0;
    for (j = 0; j < (int)g[u].size(); j++){
      v = g[u][j].first, w=g[u][j].second;
      if (dist[u] + w < dist[v]){
        dist[v] = dist[u] + w;
        if (!in_q[v]){
          //Check V-1 (vertex count) for cycle check
          if (counter[v]++ > s-1) //cycle
            return true;
          if (!q.empty() && dist[v] < dist[q.front()])
            q.push_front(v);
          else //Optimization, however only push_back(v) is needed
            q.push_back(v);
          in_q[v] = 1;
        }
      }

    }
  }
  //if this was a DIST, we could return dist[to]
  //However currently finds cycles
  return false;
}
```

## 5.3 Tarjans – Strongly Connected

```
/*
S.clear();
sccs.clear();
dfs.assign(n, 0); dfs_low.assign(n, 0), v.assign(n, 0);
dfs_n = 0;
for (i = 0; i < n; i++){
  if (!dfs[i])
    tarjans(i);
}
*/
vector<int> g[MAX];
vector<int> dfs, dfs_low, S, v;
int dfs_n = 0;
vector<vector<int>> sccs;
void tarjans(int u){
  dfs[u] = dfs_low[u] = ++dfs_n;
  S.emplace_back(u); v[u] = 1;
  for (int j = 0; j < g[u].size(); j++){
```

```
    if (!dfs[g[u][j]]){
      //set all dfs to -1 before calling this
      tarjans(g[u][j]);
    }
    if (v[g[u][j]]){
      dfs_low[u] = min(dfs_low[u], dfs_low[g[u][j]]);
    }
  }
  if (dfs_low[u] == dfs[u]){
    vector<int> scc;
    while (1){
      int vv = S.back();S.pop_back(); v[vv] = 0;
      scc.emplace_back(vv);
      if (u == vv) break;
    }
    sccs.emplace_back(scc);
  }
}
```

## 5.4 Tarjans – Cut Vertex/Edge

```
/*
REMEMBER THIS ONLY WORKS ON UNDIRECTED GRAPHS
b.clear();
dfs_num.assign(n, 0); dfs_low.assign(n, 0); dfs_parent.assign(n, 0);
a_v.assign(n, 0); dfs_cnt = 0;
int ans = 0;
for (int i = 0; i < n; i++){
  if (!dfs_num[i]){
    dfs_root = i; rC = 0; ap(i);
    a_v[dfs_root] = (rC > 1);
  }
}
*/
vpii g;
vi dfs_num, dfs_low, dfs_parent, a_v;
int rC, dfs_root, dfs_cnt;
vector<pair<int,int>> b;
void ap(int u){
  dfs_low[u] = dfs_num[u] = ++dfs_cnt;
  for (int j = 0; j < (int)g[u].size(); ++j){
    pi v = g[u][j];
    if (!dfs_num[v.first]) {
      dfs_parent[v.first] = u;
      if (u == dfs_root) rC++;
      ap(v.first);
      if (dfs_low[v.first] >= dfs_num[u]){
        //a_v[u]++;
        //count will show how many connected components
        //it splits.
      }
      if (dfs_low[v.first] > dfs_num[u]){
        //bridge
        //b.add(v.first, u), etc.
      }
      dfs_low[u] = min(dfs_low[u], dfs_low[v.first]);
    }
    else if (v.first != dfs_parent[u])
```

```
      dfs_low[u] = min(dfs_low[u], dfs_num[v.first]);
  }
}
```

### 5.5 Bipartite Graph check

```
vii g;
//set this to INT_MAX initially.
vi color;
int solve(int u){
  queue<int> bfs; bfs.push(u);
  color[u] = 0;
  while (!bfs.empty()){
    int u = bfs.front(); bfs.pop();
    for (int i = 0; i < (int)g[u].size(); i++){
      if (color[g[u][i]] == INT_MAX){
        color[g[u][i]] = 1 - color[u];
        bfs.push(g[u][i]);
      }
      else if (color[g[u][i]] == color[u])
        return -1;
    }
  }
  return 0;
}
```

### 5.6 Floyd-Warshall algorithm – shortest path of all pairs

```
// map[i][j]=infinity at start
void floyd()
{
  for (int k=1; k<=n; k++)
    for (int i=1; i<=n; i++)
      for (int j=1; j<=n; j++)
        if (i!=j && j!=k && i!=k)
          if (map[i][k]+map[k][j]<map[i][j])
            map[i][j]=map[i][k]+map[k][j];
}
```

### 5.7 Prim — minimum spanning tree

```
int d[1001]={0};
bool v[1001]={0};
int a[1001][1001]={0};

int main(void)
{
  int n=0;
  cin >> n;
  for (int i=1;i<=n;i++)
  {
    int x=0, y=0, z=0;
    cin >> x >> y >> z;
    a[x][y]=z;
  }
  for (int i=1;i<=n;i++)
```

```
    for (int j=1;j<=n;j++)
      if (a[i][j]==0) a[i][j]=INT_MAX;

  cout << prim(1,n) << endl;
}
int prim(int u, int n)
{
  int mst=0,k;

  for (int i=0;i<d.length;i++) d[i]=INT_MAX;
  for (int i=0;i<v.length;i++) v[i]=false;

  d[u]=0;
  int i=u;

  while (i!=0)
  {
    v[i]=true;k=0;
    mst+=d[i];
    for (int j=1;j<=n;j++)
      if (!v[j])
      {
        if (a[i][j]<d[j]) d[j]=a[i][j];
        if (d[j]<d[k]) k=j;
      }
    i=k;
  }
  return mst;
}
```

## 5.8 Eulerian circuit

```
// USACO Fence
#include<iostream>

using namespace std;

int f[100]={0}, ans[100]={0};
bool g[100][100]={0}, v[100]={0};
int n=0, m=0, c=0;

void dfs(int k)
{
  for (int i=1;i<=n;i++)
    if (g[k][i])
    {
      g[k][i]=false;
      g[i][k]=false;
      dfs(i);
    }
  m++;
  ans[m]=k;
}

int main(void)
{
  cin >> n >> m;
```

```
  for (int i=1;i<=m;i++)
  {
    int x=0, y=0;
    g[x][y]=true;
    g[y][x]=true;
    f[x]++;
    f[y]++;
  }

  m=0;
  int k1=0;
  for (int i=1;i<=n;i++)
  {
    if (f[i]%2==1) k1++;
    if (k1>2)
    {
      cout << "error" << endl;
      return 0;
    }
    if (f[i]%2 && c==0) c=i;
  }

  if (c==0) c=1;
  dfs(x);

  for (int i=m;i>=1;i--) cout << ans[i] << endl;
  return 0;
}
```

## 5.9 Topological sort

```
// Find any solution of topological sort.
#include<iostream>

using namespace std;

int f[100]={0}, ans[100]={0};
bool g[100][100]={0}, v[100]={0};
int n=0, m=0;

void dfs(int k)
{
  int i=0;
  v[k]=true;
  for (int i=1;i<=n;i++)
    if (g[k][i] && !v[i]) dfs(i);

  m++;
  ans[m]=k;
}

int main(void)
{
  cin >> n >> m;

  for (int i=1;i<=m;i++)
  {
    int x=0, y=0;
```

```
    cin >> x >> y;
    g[y][x]=true;
  }

  m=0;
  for (int i=1;i<=n;i++)
    if (!v[i]) dfs(i);

  for (int i=1;i<=n;i++) cout << ans[i] << endl;
  return 0;
}
```

```
// Find the order of topological sort is dictionary minimum
#include<iostream>

using namespace std;

int f[100]={0}, ans[100]={0};
bool g[100][100]={0}, v[100]={0};
int n=0, m=0;

int main(void)
{
  cin >> n >> m;

  for (int i=1;i<=m;i++)
  {
    int x=0, y=0;
    cin >> x >> y;
    g[x][y]=true;
    f[y]++;
  }
  for (int i=1;i<=n;i++)
  {
    for (int j=1;j<=n;j++)
    {
      if (f[j]==0 && !v[j]) break;

      if (f[j]!=0)
      {
        cout << "error" << endl;
        return 0;
      }

      ans[i]=j;
      v[j]=true;
      for (int k=1;k<=n;k++)
        if (g[j][k]) f[k]--;
    }
  }
  for (int i=1;i<=n;i++) cout << ans[i] << endl;
  return 0;
}
```

## 5.10 3D convex

```
int n, bf[maxN][maxN], fcnt;
point3_t pt[maxN];
```

```
struct face_t {
  int a, b, c;
  bool vis;
} fc[maxN << 5];

bool remove(int p, int b, int a) {
  int f = bf[b][a];
  face_t ff;
  if (fc[f].vis) {
    if (dblcmp(volume(pt[p], pt[fc[f].a], pt[fc[f].b], pt[fc[f].c])) >= 0) {
      return true;
    } else {
      ff.a = a, ff.b = b, ff.c = p;
      bf[ff.a][ff.b] = bf[ff.b][ff.c] = bf[ff.c][ff.a] = ++fcnt;
      ff.vis = true;
      fc[fcnt] = ff;
    }
  }
  return false;
}

void dfs(int p, int f) {
  fc[f].vis = false;
  if (remove(p, fc[f].b, fc[f].a)) dfs(p, bf[fc[f].b][fc[f].a]);
  if (remove(p, fc[f].c, fc[f].b)) dfs(p, bf[fc[f].c][fc[f].b]);
  if (remove(p, fc[f].a, fc[f].c)) dfs(p, bf[fc[f].a][fc[f].c]);
}

void hull3d() {
  for (int i = 2; i <= n; ++i) {
    if (dblcmp((pt[i] - pt[1]).length()) > 0) swap(pt[i], pt[2]);
  }
  for (int i = 3; i <= n; ++i) {
    if (dblcmp(fabs(area(pt[1], pt[2], pt[i]))) > 0) swap(pt[i], pt[3]);
  }
  for (int i = 4; i <= n; ++i) {
    if (dblcmp(fabs(volume(pt[1], pt[2], pt[3], pt[i]))) > 0) swap(pt[i], pt[4]);
  }
  mset0(fc); fcnt = 0; mset0(bf);
  for (int i = 1; i <= 4; ++i) {
    face_t f;
    f.a = i + 1, f.b = i + 2, f.c = i + 3;
    if (f.a > 4) f.a -= 4;
    if (f.b > 4) f.b -= 4;
    if (f.c > 4) f.c -= 4;
    if (dblcmp(volume(pt[i], pt[f.a], pt[f.b], pt[f.c])) > 0) swap(f.a, f.b);
    f.vis = true;
    bf[f.a][f.b] = bf[f.b][f.c] = bf[f.c][f.a] = ++fcnt;
    fc[fcnt] = f;
  }
  random_shuffle(pt + 5, pt + 1 + n);
  for (int i = 5; i <= n; ++i) {
    for (int j = 1; j <= fcnt; ++j) {
      if (!fc[j].vis) continue;
      if (dblcmp(volume(pt[i], pt[fc[j].a], pt[fc[j].b], pt[fc[j].c])) >= 0) {
        dfs(i, j);
        break;
      }
    }
```

```
  }
  for (int i = 1; i <= fcnt; ++i) if (!fc[i].vis) swap(fc[i--], fc[fcnt--]);
}
```

## 5.11 Closest Pair Points

```
double dac(point_t *p, int l, int r) {
  double d = 10e100;
  if (r - l <= 3) {
    for (int i = l; i <= r; ++i) {
      for (int j = i + 1; j <= r; ++j) {
        d = min(d, dist2(p[i], p[j]));
      }
    }
    sort(p + l, p + r + 1, cmpY);
  } else {
    int mid = (l + r) / 2;
    d = min(dac(p, l, mid), dac(p, mid + 1, r));
    inplace_merge(p + l, p + mid + 1, p + r + 1, cmpY);
    static point_t tmp[maxN]; int cnt = 0;
    for (int i = l; i <= r; ++i) {
      if ((p[i].x - p[mid].x) * (p[i].x - p[mid].x) <= d) tmp[++cnt] = p[i];
    }
    for (int i = 1; i <= cnt; ++i) {
      for (int j = 1; j <= 8 && j + i <= cnt; ++j) {
        d = min(d, dist2(tmp[i], tmp[j + i]));
      }
    }
  }
  return d;
}

double cal(point_t *p, int n) {
  sort(p + 1, p + 1 + n, cmpX);
  return sqrt(dac(p, 1, n));
}
```

## 5.12 2D base

```
struct point_t {
  double x, y;
  point_t() { }
  point_t(double tx, double ty) : x(tx), y(ty) { }
  point_t operator-(const point_t &r) const { return point_t(x - r.x, y - r.y); }
  point_t operator+(const point_t &r) const { return point_t(x + r.x, y + r.y); }
  point_t operator*(double r) const { return point_t(x * r, y * r); }
  point_t operator/(double r) const { return point_t(x / r, y / r); }
  point_t rot90() const { return point_t(-y, x); }
  double l() const { return sqrt(x * x + y * y); }
  void read() { scanf("%lf%lf", &x, &y); }
};

int dblcmp(double x) {
  return (x < -eps ? -1 : x > eps);
}

double dist(point_t p1, point_t p2) {
```

```
  return (p2 - p1).l();
}

double cross(point_t p1, point_t p2) {
  return p1.x * p2.y - p2.x * p1.y;
}

double dot(point_t p1, point_t p2) {
  return p1.x * p2.x + p1.y * p2.y;
}

// count-clock wise is positive direction
double angle(point_t p1, point_t p2) {
  double x1 = p1.x, y1 = p1.y, x2 = p2.x, y2 = p2.y;
  double a1 = atan2(y1, x1), a2 = atan2(y2, x2);
  double a = a2 - a1;
  while (a < -pi) a += 2 * pi;
  while (a >= pi) a -= 2 * pi;
  return a;
}

bool onSeg(point_t p, point_t a, point_t b) {
  return dblcmp(cross(a - p, b - p)) == 0 && dblcmp(dot(a - p, b - p)) <= 0;
}

// 1 normal intersected, -1 denormal intersected, 0 not intersected
int testSS(point_t a, point_t b, point_t c, point_t d) {
  if (dblcmp(max(a.x, b.x) - min(c.x, d.x)) < 0) return 0;
  if (dblcmp(max(c.x, d.x) - min(a.x, b.x)) < 0) return 0;
  if (dblcmp(max(a.y, b.y) - min(c.y, d.y)) < 0) return 0;
  if (dblcmp(max(c.y, d.y) - min(a.y, b.y)) < 0) return 0;
  int d1 = dblcmp(cross(c - a, b - a));
  int d2 = dblcmp(cross(d - a, b - a));
  int d3 = dblcmp(cross(a - c, d - c));
  int d4 = dblcmp(cross(b - c, d - c));
  if ((d1 * d2 < 0) && (d3 * d4 < 0)) return 1;
  if ((d1 * d2 <= 0 && d3 * d4 == 0) || (d1 * d2 == 0 && d3 * d4 <= 0)) return -1;
  return 0;
}

vector<point_t> isLL(point_t a, point_t b, point_t c, point_t d) {
  point_t p1 = b - a, p2 = d - c;
  vector<point_t> ret;
  double a1 = p1.y, b1 = -p1.x, c1;
  double a2 = p2.y, b2 = -p2.x, c2;
  if (dblcmp(a1 * b2 - a2 * b1) == 0) return ret; // colined <=> a1*c2-a2*c1=0 && b1*c2-b2*c1=0
  else {
    c1 = a1 * a.x + b1 * a.y;
    c2 = a2 * c.x + b2 * c.y;
    ret.push_back(point_t((c1 * b2 - c2 * b1) / (a1 * b2 - a2 * b1), (c1 * a2 - c2 * a1) / (b1 * a2 - b2 * a1
    return ret;
  }
}

point_t angle_bisector(point_t p0, point_t p1, point_t p2) {
  point_t v1 = p1 - p0, v2 = p2 - p0;
  v1 = v1 / dist(v1) * dist(v2);
  return v1 + v2 + p0;
}
```

```cpp
point_t perpendicular_bisector(point_t p1, point_t p2) {
  point_t v = p2 - p1;
  swap(v.x, v.y);
  v.x = -v.x;
  return v + (p1 + p2) / 2;
}

point_t circumcenter(point_t p0, point_t p1, point_t p2) {
  point_t v1 = perpendicular_bisector(p0, p1);
  point_t v2 = perpendicular_bisector(p1, p2);
  return isLL((p0 + p1) / 2, v1, (p1 + p2) / 2, v2);
}

point_t incenter(point_t p0, point_t p1, point_t p2) {
  point_t v1 = angle_bisector(p0, p1, p2);
  point_t v2 = angle_bisector(p1, p2, p0);
  return isLL(p0, v1, p1, v2);
}

point_t orthocenter(point_t p0, point_t p1, point_t p2) {
  return p0 + p1 + p2 - circumcenter(p0, p1, p2) * 2;
}

// count-clock wise is positive direction
point_t rotate(point_t p, double a) {
  double s = sin(a), c = cos(a);
  return point_t(p.x * c - p.y * s, p.y * c + p.x * s);
}

bool insidePoly(point_t *p, int n, point_t t) {
  p[0] = p[n];
  for (int i = 0; i < n; ++i) if (onSeg(t, p[i], p[i + 1])) return true;
  point_t r = point_t(2353456.663, 5326546.243); // random point
  int cnt = 0;
  for (int i = 0; i < n; ++i) {
    if (testSS(t, r, p[i], p[i + 1]) != 0) ++cnt;
  }
  return cnt & 1;
}

bool insideConvex(point_t *convex, int n, point_t t) { // O(logN), convex polygen, cross(p[2] - p[1], p[3] -
  if (n == 2) return onSeg(t, convex[1], convex[2]);
  int l = 2, r = n;
  while (l < r) {
    int mid = (l + r) / 2 + 1;
    int side = dblcmp(cross(convex[mid] - convex[1], t - convex[1]));
    if (side == 1) l = mid;
    else r = mid - 1;
  }
  int s = dblcmp(cross(convex[l] - convex[1], t - convex[1]));
  if (s == -1 || l == n) return false;
  point_t v = convex[l + 1] - convex[l];
  if (dblcmp(cross(v, t - convex[l])) >= 0) return true;
  return false;
}
```

## 5.13 3D base

```
double dot(point3_t p1, point3_t p2) {
  return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
}

point3_t cross(point3_t p1, point3_t p2) {
  return point3_t(p1.y * p2.z - p1.z * p2.y, p1.z * p2.x - p1.x * p2.z, p1.x * p2.y - p1.y * p2.x);
}

double volume(point3_t p1, point3_t p2, point3_t p3, point3_t p4) {
  point3_t v1 = cross(p2 - p1, p3 - p1);
  p4 = p4 - p1;
  return dot(v1, p4) / 6;
}

double area(point3_t p1, point3_t p2, point3_t p3) {
  return cross(p2 - p1, p3 - p1).length() / 2;
}

pair<point3_t, point3_t> isFF(point3_t p1, point3_t o1, point3_t p2, point3_t o2) {
  point3_t e = cross(o1, o2), v = cross(o1, e);
  double d = dot(o2, v);
  if (fabs(d) < eps) throw - 1;
  point3_t q = p1 + (v * (dot(o2, p2 - p1) / d));
  return make_pair(q, q + e);
}

double distLL(point3_t p1, point3_t u, point3_t p2, point3_t v) {
  double s = dot(u, v) * dot(v, p1 - p2) - dot(v, v) * dot(u, p1 - p2);
  double t = dot(u, u) * dot(v, p1 - p2) - dot(u, v) * dot(u, p1 - p2);
  double deno = dot(u, u) * dot(v, v) - dot(u, v) * dot(u, v);
  if (dblcmp(deno) == 0) return dist(p1, p2 + v * (dot(p1 - p2, u) / dot(u, v)));
  s /= deno; t /= deno;
  point3_t a = p1 + u * s, b = p2 + v * t;
  return dist(a, b);
}
```

## 5.14 Circle union

```
/* O(n^2logn), remove coincided circles first. */
point_t center[maxN];
double radius[maxN], cntarea[maxN];

pair<double, double> isCC(point_t c1, point_t c2, double r1, double r2) {
  double d = dist(c1, c2);
  double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
  double mid = atan2(y2 - y1, x2 - x1);
  double a = r1, c = r2;
  double t = acos((a * a + d * d - c * c) / (2 * a * d));
  return make_pair(mid - t, mid + t);
}

struct event_t {
  double theta;
  int delta;
  event_t(double t, int d) : theta(t), delta(d) { }
  bool operator<(const event_t &r) const {
    if (fabs(theta - r.theta) < eps) return delta > r.delta;
    return theta < r.theta;
```

```
  }
};
vector<event_t> e;

void add(double begin, double end) {
  if (begin <= -pi) begin += 2 * pi, end += 2 * pi;
  if (end > pi) {
    e.push_back(event_t(begin, 1));
    e.push_back(event_t(pi, -1));
    e.push_back(event_t(-pi, 1));
    e.push_back(event_t(end - 2 * pi, -1));
  } else {
    e.push_back(event_t(begin, 1));
    e.push_back(event_t(end, -1));
  }
}

double calc(point_t c, double r, double a1, double a2) {
  double da = a2 - a1;
  double aa = r * r * (da - sin(da)) / 2;
  point_t p1 = point_t(cos(a1), sin(a1)) * r + c;
  point_t p2 = point_t(cos(a2), sin(a2)) * r + c;
  return cross(p1, p2) / 2 + aa;
}

void circle_union() {
  for (int c = 1; c <= n; ++c) {
    int cvrcnt = 0;
    e.clear();
    for (int i = 1; i <= n; ++i) {
      if (i != c) {
        int r = testCC(center[c], center[i], radius[c], radius[i]);
        if (r == 2) ++cvrcnt;
        else if (r == 0) {
          pair<double, double> paa = isCC(center[c], center[i], radius[c], radius[i]);
          add(paa.first, paa.second);
        }
      }
    }
    if (e.size() == 0) {
      double a = pi * radius[c] * radius[c];
      cntarea[cvrcnt] -= a;
      cntarea[cvrcnt + 1] += a;
    } else {
      e.push_back(event_t(-pi, 1));
      e.push_back(event_t(pi, -2));
      sort(e.begin(), e.end());
      for (int i = 0; i < int(e.size()) - 1; ++i) {
        cvrcnt += e[i].delta;
        double a = calc(center[c], radius[c], e[i].theta, e[i + 1].theta);
        cntarea[cvrcnt - 1] -= a;
        cntarea[cvrcnt] += a;
      }
    }
  }
}
```

## 5.15 Union Polygon

```cpp
// modified from syntax_error's code
bool operator<(const point_t &a, const point_t &b) {
  if (dblcmp(a.x - b.x) == 0) return a.y < b.y;
  return a.x < b.x;
}

bool operator==(const point_t &a, const point_t &b) {
  return dblcmp(a.x - b.x) == 0 && dblcmp(a.y - b.y) == 0;
}

struct segment_t {
  point_t a, b;
  segment_t() { a = b = point_t(); }
  segment_t(point_t ta, point_t tb) : a(ta), b(tb) { }
  double len() const { return dist(a, b); }
  double k() const { return (a.y - b.y) / (a.x - b.x); }
  double l() const { return a.y - k() * a.x; }
};

struct line_t {
  double a, b, c;
  line_t(point_t p) { a = p.x, b = -1.0, c = -p.y; }
  line_t(point_t p, point_t q) {
    a = p.y - q.y;
    b = q.x - p.x;
    c = a * p.x + b * p.y;
  }
};

bool ccutl(line_t p, line_t q) {
  if (dblcmp(p.a * q.b - q.a * p.b) == 0) return false;
  return true;
}

point_t cutl(line_t p, line_t q) {
  double x = (p.c * q.b - q.c * p.b) / (p.a * q.b - q.a * p.b);
  double y = (p.c * q.a - q.c * p.a) / (p.b * q.a - q.b * p.a);
  return point_t(x, y);
}

bool onseg(point_t p, segment_t s) {
  if (dblcmp(p.x - min(s.a.x, s.b.x)) < 0 || dblcmp(p.x - max(s.a.x, s.b.x)) > 0) return false;
  if (dblcmp(p.y - min(s.a.y, s.b.y)) < 0 || dblcmp(p.y - max(s.a.y, s.b.y)) > 0) return false;
  return true;
}

bool ccut(segment_t p, segment_t q) {
  if (!ccutl(line_t(p.a, p.b), line_t(q.a, q.b))) return false;
  point_t r = cutl(line_t(p.a, p.b), line_t(q.a, q.b));
  if (!onseg(r, p) || !onseg(r, q)) return false;
  return true;
}

point_t cut(segment_t p, segment_t q) {
  return cutl(line_t(p.a, p.b), line_t(q.a, q.b));
}

struct event_t {
  double x;
```

```
  int type;
  event_t() { x = 0, type = 0; }
  event_t(double _x, int _t) : x(_x), type(_t) { }
  bool operator<(const event_t &r) const {
    return x < r.x;
  }
};

vector<segment_t> s;

double solve(const vector<segment_t> &v, const vector<int> &sl) {
  double ret = 0;
  vector<point_t> lines;
  for (int i = 0; i < v.size(); ++i) lines.push_back(point_t(v[i].k(), v[i].l()));
  sort(lines.begin(), lines.end());
  lines.erase(unique(lines.begin(), lines.end()), lines.end());
  for(int i = 0; i < lines.size(); ++i) {
    vector<event_t> e;
    vector<int>::const_iterator it = sl.begin();
    for(int j = 0; j < s.size(); j += *it++) {
      bool touch = false;
      for (int k = 0; k < *it; ++k) if (lines[i] == point_t(s[j + k].k(), s[j + k].l())) touch = true;
      if (touch) continue;
      vector<point_t> cuts;
      for (int k = 0; k < *it; ++k) {
        if (!ccutl(line_t(lines[i]), line_t(s[j + k].a, s[j + k].b))) continue;
        point_t r = cutl(line_t(lines[i]), line_t(s[j + k].a, s[j + k].b));
        if (onseg(r, s[j + k])) cuts.push_back(r);
      }
      sort(cuts.begin(), cuts.end());
      cuts.erase(unique(cuts.begin(), cuts.end()), cuts.end());
      if (cuts.size() == 2) {
        e.push_back(event_t(cuts[0].x, 0));
        e.push_back(event_t(cuts[1].x, 1));
      }
    }
    for (int j = 0; j < v.size(); ++j) {
      if (lines[i] == point_t(v[j].k(), v[j].l())) {
        e.push_back(event_t(min(v[j].a.x, v[j].b.x), 2));
        e.push_back(event_t(max(v[j].a.x, v[j].b.x), 3));
      }
    }
    sort(e.begin(), e.end());
    double last = e[0].x;
    int cntg = 0, cntb = 0;
    for (int j = 0; j < e.size(); ++j) {
      double y0 = lines[i].x * last + lines[i].y;
      double y1 = lines[i].x * e[j].x + lines[i].y;
      if (cntb == 0 && cntg) ret += (y0 + y1) * (e[j].x - last) / 2;
      last = e[j].x;
      if (e[j].type == 0) ++cntb;
      if (e[j].type == 1) --cntb;
      if (e[j].type == 2) ++cntg;
      if (e[j].type == 3) --cntg;
    }
  }
  return ret;
}
```

```
double polyUnion(vector<vector<point_t>> polys) {
  s.clear();
  vector<segment_t> A, B;
  vector<int> sl;
  for (int i = 0; i < polys.size(); ++i) {
    double area = 0;
    int tot = polys[i].size();
    for (int j = 0; j < tot; ++j) {
      area += cross(polys[i][j], polys[i][(j + 1) % tot]);
    }
    if (dblcmp(area) > 0) reverse(polys[i].begin(), polys[i].end());
    if (dblcmp(area) != 0) {
      sl.push_back(tot);
      for (int j = 0; j < tot; ++j) s.push_back(segment_t(polys[i][j], polys[i][(j + 1) % tot]));
    }
  }
  for (int i = 0; i < s.size(); ++i) {
    int sgn = dblcmp(s[i].a.x - s[i].b.x);
    if (sgn == 0) continue;
    else if (sgn < 0) A.push_back(s[i]);
    else B.push_back(s[i]);
  }
  return solve(A, sl) - solve(B, sl);
}
```

## 5.16 Minimum circle

```
void set_circle(point_t &p, double &r, point_t a, point_t b) {
  r = dist(a, b) / 2;
  p = (a + b) / 2;
}

void set_circle(point_t &p, double &r, point_t a, point_t b, point_t c) {
  if (dblcmp(cross(b - a, c - a)) == 0) {
    if (dist(a, c) > dist(b, c)) {
      r = dist(a, c) / 2;
      p = (a + c) / 2;
    } else {
      r = dist(b, c) / 2;
      p = (b + c) / 2;
    }
  } else {
    p = circumcenter(a, b, c);
    r = dist(p, a);
  }
}

bool in_circle(point_t &p, double &r, point_t x) {
  return dblcmp(dist(x, p) - r) <= 0;
}

pair<point_t, double> minimum_circle(int n, point_t *p) {
  point_t c = point_t(0, 0);
  double r = 0;
  random_shuffle(p + 1, p + 1 + n);
  set_circle(c, r, p[1], p[2]);
  for (int i = 3; i <= n; ++i) {
    if (in_circle(c, r, p[i])) continue;
```

```
    set_circle(c, r, p[i], p[1]);
    for (int j = 2; j < i; ++j) {
      if (in_circle(c, r, p[j])) continue;
      set_circle(c, r, p[i], p[j]);
      for (int k = 1; k < j; ++k) {
        if (in_circle(c, r, p[k])) continue;
        set_circle(c, r, p[i], p[j], p[k]);
      }
    }
  }
  return make_pair(c, r);
}
```