Contents

1 General Problem Solving Strategies

- **Greedy** – Choose the most promising element based on "local information", remove from set of possibilities and recurse.
- **Divide and Conquer** – Break problem down into two smaller instances. Solve each smaller independently and then merge results.
- **Dynamic Programming** – Pick one element; consider solutions both using that element and not using it.

  The recursion will lead to many identical subcases. Use memoization to avoid duplicate work.

2 Common STL Methods

`T` = templated type used in declaration
`i` = integer, `b` = boolean, `c` = char
`IT` = iterator

Vector, `#include <vector>`

Iterators are invalidated by inserts/deletes
insertion $\mathcal{O}(1)$ at back, $\mathcal{O}$(N) front/middle

```
      push_back (T&)        pop_back ()
 b    empty ()          i   size ()
 T&   back ()           T&  front ()
      clear ()              resize (i)
 T&   operator[i]
```

Map, `#include <map>`

Iterators are *not* invalidated by inserts/deletes
insertion $\mathcal{O}$(lg N)
`K` = key type, `V` = data type; `map<K, V>`

```
 b    empty ()        i   size ()
      clear ()            insert (pair<K,V>)
 V&   operator[&K]        erase (IT)
 IT   find (&K)
```

Warning: `operator[&K]` will insert an element into the map if `K` is not already present. The value returned will be the default constructor for `V`.
Iterators for map point to `pair<K,V>`.

List, `#include <list>`

Iterators are *not* invalidated by insertion/deletion.
insertion $\mathcal{O}(1)$ at front/back. $\mathcal{O}$(N) middle

```
      push_back (T&)        push_front (T&)
      pop_back (T&)         pop_front (T&)
 T&   back ()           T&  front ()
 b    empty ()          i   size ()
      clear ()              insert (IT, &T)
      erase (IT)            unique ()
      sort ()               reverse ()
```

Note that list supports `unique ()`, `sort ()` and `reverse ()` as *member functions*. These versions are specialized to lists and should be used instead of the more generic versions found in the STL.

Deque, `#include <deque>`

insertion $\mathcal{O}(1)$ at front/back. $\mathcal{O}$(N) middle

```
      push_back (T&)    push_front (T&)
      pop_back ()       pop_front ()
 T&   back ()           T& front ()
 b    empty ()          clear ()
 i    size ()           resize (i)
 T&   operator[i]
```

stack Supports `push pop size empty top`

queue Supports `push pop size empty front back`

String, `#include <string>`

Iterators are invalidated by inserts/deletes

```
      push_back (c)          c*   c_str ()
 b    empty ()               i    size ()
 c&   back ()                c&   front ()
      clear ()                    resize (i)
 c    operator[i]
 i    find (string, pos)     i    find (c, pos)
```

```
erase([pos], [n])   append(n, c)
substr(pos, [n])    insert(pos, s)
insert(pos, c)      insert(pos, n, c)
find_first_of(s)    find_first_not_of(s)
```

`find` returns `string::npos` if the given string or character can't be found.

Char Predicates, `#include <cctype>`

`isalnum`	alpha or digit
`islower`	`isupper`
`alpha`	letter (upper or lower)
`digit`	0 to 9
`space`	any whitespace
`graph`	alnum or punct
`print`	not cntrl = graph or ' '
`punct`	print & !(space or alnum)
`xdigit`	hex digit

```
string lowercase(string s) {
  string rv(s); // could be in-place
  for(int i = 0; i < s.size(); ++i)
      rv[i] = tolower(s[i]);
  return rv;}

template<class N>
string to_str(N n) { stringstream s;
                     s << n; return s.str(); }
double to_num(string s) { stringstream ss(s);
              double n; ss >> n; return n; }
```

3 Summations

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \qquad \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=1}^{n} k^3 = \frac{n^2(n+1)^2}{4} \quad \sum_{k=1}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

4 Triangle Areas

If vertices of triangle, (x_i, y_i) are known,

$$A = \frac{1}{2}\begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{vmatrix}$$

This is a *signed* area. Counterclockwise around the triangle will give a positive area. Clockwise gives a negative area.

If the lengths of the sides are known,

$$A = \sqrt{s(s-a)(s-b)(s-c)} \quad s = \frac{1}{2}(a+b+c)$$

See also Intersection.java at the back of the packet.

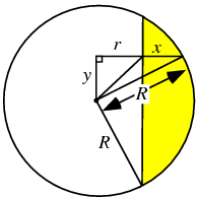
5 Polygon area

For an arbitrary polygon with N vertices, (x_i, y_i),

$$A = \frac{1}{2}\begin{vmatrix} x_{N-1} & x_0 \\ y_{N-1} & y_0 \end{vmatrix} + \frac{1}{2}\sum_{i=0}^{N-1}\begin{vmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{vmatrix}$$

This is a signed area, positive if you go counterclockwise around the polygon, negative if you go clockwise. The first term is needed to cover the last section of the polygon.

Formula for the area of a circular segment (shaded):



$$A = \frac{1}{2}R^2\cos^{-1}\left(\frac{r}{R}\right) - r\sqrt{R^2 - r^2}$$

This can also be computed by subtracting the area of the triangle (origin to each chord endpoint) from the area of the sector (which is $\frac{R^2}{2}\theta$).

Surface area of a sphere is $4\pi r^2$.

6 Combinatorics

n items, pick k of them such that order matters and any object may be used more than once. Number of ways is n^k.

n items, pick k of them such that order matters, but objects may only be used once. Number of ways

$$(n)_k = \frac{n!}{(n-k)!}$$

n items, pick k of them such that order does not matter and each object can be used only once. Number of different sets we can have is

$$(n)_k/k! = \binom{n}{k} = \frac{n!}{k!(n-k)!} = \binom{n}{n-k}$$

n items, pick k of them such that order does not matter and each object can be used more than once. Number of different sent we can have is

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

7 Binomial Coefficients

$$(x+y)^n = x^n + \binom{n}{1}x^{n-1}y + \binom{n}{2}x^{x-2}y^2 + \cdots + \binom{n}{n}y^n$$
$$= \sum_{k=0}^{n}\binom{n}{k}x^{n-k}y^k$$

$$\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$$

$$\sum_{k=0}^{n}\binom{n}{k} = 2^n$$

8 Catalan Numbers

$$C(n) = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} \quad n \geq 0$$

First few $C(n)$ are 1 1 2 5 14 42 132 ...
$C(n)$ gives a) number of matching () of length n, b) number of ordered binary trees with $n+1$ leaves, c) number of ways a convex polygon with $n+2$ sides can be cut into triangles, d) number of ways to tile a stairstep of height n with n rectangles.

Number of lattice paths from $(0,0)$ to (a,b) with only east, north moves is

$$\binom{a+b}{a}$$

Number of lattice paths from $(0,0)$ to (b,a) with only east, north, northeast moves:

$$D(a,b) = D(a-1,b) + D(a,b-1) + D(a-1,b-1)$$

$$D(0,0) = 1$$

Dyck paths (staircase walks below diagonal y=x) of order n is $C(n)$

Number of nonattacking kings on a nxn chessboard:

$$K(n) = 1/4 * (n + (n \mod 2))^2$$

Number of bishops on a chessboard: min n, max $2n-2$. Number of rot + refl distinct:

$$2^{n-3} + 2^{\lfloor (n-1)/2 \rfloor - 1}$$

Pick's theorem: A is area of lattice polygon, B is number of lattice points on polygon I is number of points in polygon interior then $A = I + B/2 - 1$

Lattice points on line from origin to (a,b) is gcd(a,b) - 1

Fraction of lattice pts visible from $(0,0)$ is $6/(\pi^2)$

9 Config

vimrc

```
set nocp nu magic sm
set ww=b,s,[,]
set ve=onemore
set bs=indent,eol,start
set mouse=a
behave mswin
syntax on
filetype plugin indent on
set hls sc ru cin go-=a

noremap <C-S>   :update<CR>
inoremap <C-S>  <C-O>:update<CR>

# xmodmap -e "keycode 67 = Escape"
```

Makefile

```
# Don't forget to use tabs!
# $< would be "strings.cpp"
# $* would be "strings"
#

%: %.cpp
g++ -g $<

%.class: %.java
javac $<

%: %.class
java $* `cat`

.PRECIOUS: %.class
```

10 Vector and Map Utilities

```
template<class K, class V>
  pair<V,K>
invert(pair<K,V>& p) {
    return make_pair(p.second, p.first);
}

        // map to inverted vector
template<class K, class V>
  vector<pair<V,K> >
invert(map<K,V>& m) {
    vector< pair<V,K> > v;
    FOREACH(i, m) v.push_back(invert(*i));
    return v;
}
```

11 Algorithms and Code

11.1 GCD

```
int gcd (int b, int s) {
    int r;
    while ((r = (b % s))) {
        b = s;
        s = r;
    }
    return s;
}
```

Euler's totient, $\varphi(n)$, gives the number of positive integers less than or equal to n that are coprime to n (i.e., n and m are coprime if `gcd (n, m) = 1`).

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right) \quad \text{sum over primes } p \text{ dividing } n$$

$$G(n) + n = g(n) = \sum_{j=1}^{n} \gcd(j, n) = n \sum_{d|n} \frac{\phi(d)}{d}$$

$$G(a * b) = (a + G(a)) * (b + G(b)) - (a * b)$$

Quadratic formula is $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

11.2 Common Prelude Template File

```
// prelude.cpp
#include <bitset> // includes string too!
#include <iostream>
#include <fstream>
#include <sstream>
#include <map>
#include <vector>
#include <cstdio>

using namespace std;

ifstream fin("file.in"); // CHANGE FILENAME!
#define cin fin

#define ALL(x) x.begin(), x.end()

#define AUTO(v,e) typeof(e) v = e

    // loop with i as iterator for collection c
#define FOREACH(i,c) \
        for(AUTO(i, c.begin()); i != c.end(); ++i)

    // loop with i as index from 0 to n-1
#define FOR(i,n) for(int i = 0; i < n; ++i)

    // debugging function
#ifdef DEBUG
#define D(x) ( cout << "\t" << #x << " = " \
                << (x) << endl );
#else
#define D(x)
#endif

    // useful for "normalizing" angles
int mod(int n, int s) { return (s+(n%s)) % s; }

    // generic printing of maps, vectors, etc
namespace std {
  // Allows cout << make_pair(1,2) << endl;
  template<class A, class B>
  ostream& operator<<(ostream& out,
                      const pair<A,B>& x) {
      return out << "{" << x.first << ", "
                        << x.second << "}";
  }
}

template <class S>
string str(const S& v,
               const char* d = " ") {
 stringstream ss;
 FOREACH(i,v) ss << *i << d;
 string s = ss.str(); // rem trailing delim
 if(!s.empty())
   s.erase(s.size() - strlen(d));
 return s;
}
```

```
/*
Using common prelude functions:

Compile during testing with -DDEBUG

Liberally use D(var);
  and D(str(container));

Instead of
for(int i = 0;
  i < n; ++i) { blah; }
  use FOR(i,n) { blah; }

And instead of
  for(map<k,v>::iterator i = m.begin();
      i != m.end(); ++i) { blah; }
just use FOREACH(i,m) { blah; }
*/
```

11.3 Breadth First Search

```
BFS (state start, state end)
// worklist, use stack for a depth first search
queue wl
state s
wl.push (start)

while (!wl.empty ())
  s = wl.front () ; wl.pop ()
  if (s != end)
    foreach state s1 in expand (s)
       wl.push (s1)
```

BFS relies on you defining a "state" and providing an "expansion" operator. For a graph traversal, a state is node in the graph, expand (s) gives the list of nodes connected to s.

```
struct node_t {
  int prev_node; // prev. node in shortest path
  int cost;  // cost from s to this node
  // nodes connected to this node.
  // first = next node, second = edge weight
  vector<pair<int,int> > succ;
}
vector<node_t> nodes;

// initialize nodes
int shortest_path (int start, int end)
{
  int n;
  queue<int> wl;

// for each i in nodes ;
//   nodes [i].cost = very large number
//   nodes [i].prev_node = -1;

  wl.push (start);
  nodes [start].cost = 0;

  while (!wl.empty ()) {
    n = wl.front ();  wl.pop ();
    for (i = 0; i < nodes [n].succ.size (); i++) {
      int s = nodes [n].succ [i].first;
      int c = nodes [n].cost
               + nodes [n].succ [i].second;
      if (c < nodes [s].cost) {
        nodes [s].cost = c;
        nodes [s].prev_node = n;
        wl.push (s);
      }
    }  }
  return (nodes [end].cost);
  // can get path by reading backwards from
  //  nodes [end].prev_node;
}
```

11.4 Convex Hulls - Graham Scan, Melkman

```
import java.awt.geom.*;
import java.util.*;
import java.awt.*;
import java.io.*;

public class ConvexHull {
  private class PDD extends Point2D.Double {
    PDD(double x, double y) { super(x,y); }
  }

 public static void main(String[] args)
                 throws IOException {
   new ConvexHull().run();
 }

 public void run() throws IOException {
   ArrayList<PDD> hull,
                  pts = new ArrayList<PDD>();
   Scanner fin = new Scanner(System.in);

   int datasets = fin.nextInt();
   for(int ds=0; ds < datasets; ++ds) {
     pts.clear();
     int N = fin.nextInt();

     for(int i = 0; i < N; ++i) {
       double px = fin.nextDouble();
       double py = fin.nextDouble();
       pts.add(new PDD(px,py));
     }

     hull = convexHull(pts);

     Polygon p = new Polygon();
     for(PDD pt : hull) {
      p.addPoint(pt.x, pt.y);
     }
     Area a = new Area(p);
     System.out.println(
       "Area of convex hull is: "
         + areaOf(a));
   }
 }

// Melkman's algorithm (for polylines): 38 lines
  double area( PDD a, PDD b, PDD c) {
      return (b.x-a.x)*(c.y-a.y) -
             (c.x-a.x)*(b.y-a.y);
  }

  public ArrayList<PDD>
  MconvexHull(ArrayList<PDD> v) {
      int n = v.size();
      PDD[] D = new PDD[2*n + 1];
      int bot = n-2, top = bot+3;
      D[bot] = D[top] = v.get(2);
      if(area( v.get(0), v.get(1),
                         v.get(2) ) > 0) {
          D[bot+1] = v.get(0);
          D[bot+2] = v.get(1);
      } else {
          D[bot+1] = v.get(1);
          D[bot+2] = v.get(0);
      }
      for(int i = 3; i < n; ++i) {
          PDD x = v.get(i);
          if((area(D[bot],
                   D[bot+1], x) > 0) &&
             (area(D[top-1],
                   D[top], x) > 0) )
                 continue;
          while(area(D[bot],
                     D[bot+1], x) <= 0)
              ++bot;
          D[--bot] = x;
          while(area(D[top-1],
                     D[top], x) <= 0)
              --top;
          D[++top] = x;
      }
      ArrayList<PDD> rv =
           new ArrayList<PDD>();
      int h;
      for(h=0; h < (top-bot); ++h)
         rv.add(D[bot+h]);
      return rv;
  }

// Graham Scan, for set of points: 53 lines
  private class PtCmp
          implements Comparator<PDD> {
    public int compare(PDD a, PDD b) {
      if(a.y < b.y) return -1;
      if(a.y > b.y) return 1;
      if(a.x < b.x) return -1;
      if(a.x > b.x) return 1;
      return 0;
    }
  }

  class AngleCmp implements Comparator<PDD> {
    PDD p;
    AngleCmp(PDD pt) { p = pt; }
    public int compare(PDD a, PDD b) {
      // clockwise -> right turns
      return -DIRECTION * ccw(p, a, b);
    }
  }
  // -1 means ccw/left
  static final int DIRECTION = -1;

  // c is on line ab, close to a: -1;
  //     after b: 1. between = 0
  public int ccw( PDD a, PDD b, PDD c) {
    return Line2D.relativeCCW(a.x, a.y,
                              b.x, b.y, c.x, c.y);
  }

  static double areaOf(Shape s) {
   PathIterator i = s.getPathIterator(null);
   double[] coords = new double[6];
```

```
  int r;
  double area = 0.;
  double mx, my, lx, ly, x, y;
  mx= my= lx= ly= 0.;
  while(!i.isDone()) {
    r = i.currentSegment(coords);
    x = coords[0]; y = coords[1];

   if(r == PathIterator.SEG_LINETO) {
      area += (lx*y - x*ly);
   } else if(r == PathIterator.SEG_CLOSE) {
      area += (lx*my - mx*ly);
   } else if(r == PathIterator.SEG_MOVETO) {
      mx = x;
      my = y;
   }
    lx = x; ly = y;
    i.next();
  }
  return Math.abs(area/2);
 }

 public ArrayList<PDD>
 convexHull(ArrayList<PDD> l) {
   PDD p, last, next2last;

   // find pivot
   p = Collections.min(l, new PtCmp());

   l.remove(p); // remove pivot from sort
   Collections.sort(l, new AngleCmp(p));

   Stack<PDD> stack = new Stack<PDD>();
   stack.push(p); // add pivot back
   stack.push(l.get(0));
   for(int i = 1; i < l.size(); ++i) {
     boolean done = false;
     while(!done && stack.size() > 1) {
       last = stack.pop();
       next2last = stack.peek();
       stack.push(last);
       // "good" ccw pt means left turn
       if( DIR == ccw(next2last, last,
                      l.get(i))) {
         done = true;
       } else  stack.pop();
     }
     stack.push(l.get(i));
   }

   return new ArrayList<PDD>(stack);
 }
}
```

End Graham Scan for Convex Hulls

11.5 BigInteger Example

```
//Catalan numbers
import java.awt.geom.*;
import java.util.*;
import java.awt.*;
import java.io.*;
import java.math.BigInteger;

class Catalan {
public static void main(String[] args)
          throws IOException {
    new Catalan().run();
 }

 TreeMap<Integer, BigInteger> facts;

 public void run() throws IOException {
  Scanner cin = new Scanner(System.in);
  facts = new TreeMap<Integer,BigInteger>();

  while(true) {
   int n = cin.nextInt();
   if (n == 0) break;
    System.out.println( catalan(n).toString());
    //System.out.println( fact(n).toString());
  }
 }

 BigInteger catalan(int n) {
    return fact(2*n).divide(
                  fact(n+1).multiply(fact(n)));
 }

 BigInteger fact(int n) {
    BigInteger rv = BigInteger.valueOf(1);
    for(int i = 2; i <= n; ++i)
        rv = rv.multiply(BigInteger.valueOf(i));
    return rv;
 }
}
```

11.6 Graph Algorithm - All-pairs shortest Path

```
#include "prelude.cpp"

void floyd(vector<edge>& edgepairs,
            map<edge, weight>& weights,
            map<edge, weight>& spath,
            map<edge, vector<edge> >& path,
            int n) {
  weight MAX_PATH = 999999;
  FOR(k,n) {
      FOR(i,n) {
          spath[edge(i,k)] = MAX_PATH;
      }
  }

  FOREACH(it, edgepairs) {
      spath[*it] = weights[*it];
      path[*it].push_back( *it );
  }

  FOR(k,n) {
    FOR(i,n) {
      FOR(j,n) {
        edge ij = edge(i,j);
        edge ik = edge(i,k);
        edge kj = edge(k,j);
        int w = spath[ik] + spath[kj];
        if( w < spath[ij] ) {
           // record path weight info
          spath[ij] = w;
          // also record actual path info
          path[ij].clear();
          path[ij].insert(path[ij].end(),
                          ALL(path[ik]));
          path[ij].insert(path[ij].end(),
                          ALL(path[kj]));
        }
      }
    }
  }
}
```

11.7 Stable Marriage (Pair Matching)

Problem: Find matching between group of men and women such that no pair of people would rather be in a different pair.

Basic idea:

```
Put all men onto a free list #1
while (no change)
 foreach m on free list #1
   order all women according to m's preference
   foreach w on m's pref. list
    if w is not engaged
      engage (m, w)
      change = true
    if w is engaged to m'
      if w prefers m to m'
        put m' on free list #2
        engage (m, w)
        change = true
   endfor
   if m was not matched
     put m on free list #2
  endfor
  swap free list #2 and #1
end while
```

If the number of men equals the number of women, all men will be matched. If there's more men than women, free list #1 will have the list of unmatched men.

```
#include "prelude.cpp"
#include <list>
#include <deque>

using namespace std;

// Assumptions: men and women are both
// identified by integers, starting from 0
typedef int manid;
typedef int womanid;

struct man {
    int id; int people; int minutes;
    man(int i, int p, int m):
      id(i), people(p), minutes(m) {}
};

struct woman {
    int id; int seats; int minutes;
    woman(int i, int p, int m):
        id(i), seats(p), minutes(m) {}
};

vector<man> men;
vector<woman> women;

// representation of fit between man/woman
typedef pair<int,int> score;
// how much does a given man like this woman?
typedef pair<score, womanid> prefPair;
```

```
score calc_score(man m, woman w) {
    // (extraseats, extraminutes)
    return score(w.seats - m.people,
                 w.minutes - m.minutes);
}

bool sortPrefList(const prefPair& a,
                  const prefPair& b) {
    return a.first < b.first;
}

void stable_marriage(vector<man>& M,
                     vector<woman>& W) {
 // initially all men are free
 deque<man> q(ALL(M));

 map<womanid, bool> womantaken;
 map<manid, bool> mantaken;
 map<womanid, manid> husband;
 map<manid, list<prefPair> > prefs;

 FOR(i,M.size()) {
   prefs[i] = list<prefPair>();
   FOR(j,W.size()) {
    score s = calc_score(M[i], W[j]);
    // are the given man and woman
    //   even potentially compatible?
    // Not if the man is bigger than the woman
    if(s.first >= 0 && s.second >= 0) {
      prefs[i].push_back( prefPair(s, j) );
    }
   }
   prefs[i].sort(sortPrefList);
 }

 while(!q.empty()) {
   man aMan = q.front();
   q.pop_front();

   manid m = aMan.id;

   // if the man has "proposed"
   //   to every woman, he's SOL
   if(prefs[m].empty()) { continue; }

   // Pick the highest-ranked woman
   //   left for the man
   prefPair scorepair = prefs[m].front();
   prefs[m].pop_front();

   womanid w = scorepair.second;

   // if woman is free, "engage" m and the woman
  if(!womantaken[w]) {
    womantaken[w] = true;
    mantaken[m] = true;
    husband[w] = m;
  } else {
    // w is currently engaged to another man, "m2"
    manid m2 = husband[w];
    score s1 = calc_score(M[m],  W[w]);
    score s2 = calc_score(M[m2], W[w]);
    if(s1 < s2) {
      // m is better match than m2
      husband[w] = m;
      mantaken[m] = true;

      // m2 returns to the dating pool
      q.push_back(M[m2]);
      mantaken[m2] = false;
    } else { // m is still free
      q.push_back(aMan);
    }
  }
 }

 // by now, all women have taken the man
 //    that fits them best.
 int bachelors = 0;
 int people_in_tents = 0;
 FOR(i,M.size()) {
   if(!mantaken[i]) {
     bachelors++;
     people_in_tents += men[i].people;
   }
 }
 printf("%2d %d\n", bachelors,
        people_in_tents);
}

int main(void) {
int p, d, s, h, m, w, r;
 int casenumber = 0;
 while(true) {
   w = 0;
   cin >> w;
   if(w == 0) return 0;

   ++casenumber;
   men.clear();
   women.clear();

   FOR(i,w) {
     cin >> p >> d;
     men.push_back(man(i,p,d));
   }

   cin >> r;
   string line;
   getline(cin, line); // discard endline
   FOR(i,r) {
     getline(cin, line);
     string::size_type c = line.find(':', 0);
     if(c != string::npos) line[c] = ' ';
     stringstream ss(line);
     ss >> s >> h >> m;
     m += 60 * (h - 14);
     women.push_back(woman(i,s,m));
   }
   cout << "Trial " << casenumber
        << "\n";
   stable_marriage(men, women);
 }
}
```

11.8 bitset example

```
#include <bitset>
#include <iostream>
#include <sstream>

using namespace std;

// removes leading c characters from string s
string strip(char c, string s) {
  return s.substr(
         s.find_first_not_of(c) );
}

template<typename B>
string strip(B b) {
    stringstream ss; ss << b;
    return strip('0', ss.str());
}

int main() {
        // left-padded with zeroes
    cout << bitset<16>(43) << endl;
        // remove left-padding
    cout << strip(bitset<64>(43)) << endl;

    bitset<10> b(43);
    b;             // 0000101011
    b.count();     // 4
    ~b;            // 1111010100
    (~b).count(); // 6
    (b << 2);      // 0010101100
    b.set();       // 1111111111
    b.reset();     // 0000000000
    b.set(2);      // 0000000100
    b[3] = 1;      // 0000001100
    b[3];          // 1
    b[1];          // 0
    b.size();      // 10

    cout << "\"1101001001\" as number: "
         << bitset<64>(string("1101001001"))
                    .to_ulong() << endl;
}
```

11.9 Min-cut / Max Flow

Problem: Given a graph with capacities along edges, what is the maximum flow from a start node, s, to an end node, t? For a flow graph, multiple edges from u to v and from v to u can be combined into one edge with positive capacity.

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

using namespace std;

typedef vector<int> Vec;
typedef vector<Vec> VV;
VV C; // capacity
VV F; // flow
VV E; // adj. list

int min_cut(int n, int s, int t) {
  while(true) {
    Vec P(n, -1); // parent table
    // capcity of path to node
    Vec M(n, 0);
    P[s] = s;
    M[s] = INT_MAX;

    queue<int> q;
    q.push(s);
    while(!q.empty()) {
      int u = q.front(); q.pop();
      for(int i = 0; i < E[u].size(); ++i) {
        int v = E[u][i];
        int w = C[u][v] - F[u][v];
        if( w > 0 && P[v] == -1) {
          P[v] = u;
          M[v] = min(M[u], w);
          if(v != t) {
            q.push(v);
          } else {
            while(P[v] != v) {
              u = P[v];
              F[u][v] += M[t];
              F[v][u] -= M[t];
              v = u;
            }
            q = queue<int>();
            break;
          }
        }
      }
    }
    // No more augmenting flows
    if(P[t] == -1) {
      int rv = 0;
      for(int i = 0; i < n; ++i)
        rv += F[s][i];
      return rv;
    }
```

```
  }
}

int main() {
 int V, Edges, a, b, c;
 int source, sink;
 char type;
 string line;
 while(cin >> type && cin.good()) {
   if(type == 'c') {
     getline(cin,line);
     continue;
   } else if(type == 'p') {
     cin >> line; // "max"
     cin >> V >> Edges;
     F = VV(V);
     C = VV(V);
     E = VV(V);
     for(int i = 0; i< V; ++i) {
       F[i] = Vec(V, 0);
       C[i] = Vec(V, 0);
       E[i] = Vec();
     }
   } else if(type == 'a') {
     // Input goes from 1-n,
     cin >> a >> b >> c;
     --a; --b;
     C[a][b] = c;
     E[a].push_back(b);
     E[b].push_back(a);
   } else if(type == 'n') {
     char s_or_t;
     cin >> a >> s_or_t;
     // Input goes from 1-n,
     --a;
     if(s_or_t == 's') source = a;
     else sink = a;
   } else {
     getline(cin,line);
     continue;
   }
 }
 cout << min_cut(V, source, sink) << endl;
}
```

11.10 Minimum Spanning Tree

Problem: Given a *connected*, *undirected* graph, G, find the minimum spanning tree for G. The minimal spanning tree, T, is a connected graph with no cycles such that the vertices of T is the same as G. The vertices in T are connected by exactly one path.

Kruskal's algorithm:

```
using namespace std;

typedef int node;
typedef vector<node> nodes;
typedef pair<node,node> edge;

nodes p; // union/find parents
nodes rank; // union/find rank

template<class K, class V>
pair<V,K> invert(pair<K,V>& p) {
    return make_pair(p.second, p.first);
}

// map to inverted vector
template<class K, class V>
vector<pair<V,K> > invert(map<K,V>& m) {
    vector< pair<V,K> > v;
    FOREACH(i, m)
        v.push_back(invert(*i));
    return v;
}

map<edge,int> G;

int find(int);
int find_union(int,int);

void Kruskal(int V, vector<edge>& T) {
    vector<pair<int,edge> > Q = invert(G);
    sort(ALL(Q));

    int i = 0;
    while(T.size() < V-1) {
        edge e = Q[i++].second;
        int Cv = find(e.first);
        int Cu = find(e.second);
        if(Cv != Cu) {
            T.push_back(e);
            find_union(Cv, Cu);
        }
    }
}

int main() {
    while(true) {
        int V,E,u,v,w;
        cin >> V >> E;
        if(V == 0) break;

        G.clear();
        p = nodes(V);
        rank = nodes(V,0);
```

```
        for(int i = 0; i < V; ++i) {
            p[i] = i;
        }

        for(int i=0; i < E; ++i) {
            cin >> u >> v >> w;
            G[edge(u,v)] = w;
        }

        vector<edge> T;
        Kruskal(V, T);

        // do something with MST T

        return 0;
    }
}

int find(int x) {
    if(p[x] != x)
        p[x] = find(p[x]);
    return p[x];
}

int find_union(int x, int y) {
    int xr = find(x);
    int yr = find(y);
    int xrr = rank[xr];
    int yrr = rank[yr];
     if(xrr > yrr) p[yr] = xr;
    else if(xrr < yrr) p[xr] = yr;
    else if(xr != yr) {
        p[yr] = xr;
        rank[xr] = xrr + 1;
    }
}
```

11.11 Backtracking

Backtracking allows you to try all the possible configurations of a problem to find a solution using global data structures. General idea:

```
BT (state s) {
  if (s is the goal state) {
      // process
  }
  if (goal state is unreachable from s)
     return

  foreach value v for state s {
     mark (s, v)
     BT (next-state)
     unmark (s, v)
  }
}
```

`mark`/`unmark` adjust the global data structure to account for `s1`. For a problem like solving sudoku, the global data structure would the playing grid. A state would a row and column number for a particular cell and the possible values that cell could take. Expanding the state tries each possible cell value. mark would adjust the playing grid so that all other cells in the row, column and region cannot use the value `v` anymore. unmark undoes the adjustments of mark.

For placing n queens on a nxn chessboard:

```
#include <cstdio>

const int N = 8;
 // row r's queen is in columm col[r]
int col[N],
    s = 0; // solution number

#define FOR(i,n) for(int i = 0; i < n; ++i)

bool safe(int x, int r) {
  FOR(z,r) {
   int i = z + 1;
   int c = col[r - i];
   if (c == x - i || c == x
     || c == x + i) return false;
   }
   return true;
}

void find(int r) {
 FOR(x,N) {
  if (safe(col[r] = x, r))
   if (r+1 < N) find(r+1);
   else putboard();
 }
}
int main() {
 find(0);
 return 0;
}
```

11.12 Sudoku with bitsets

```
#include <queue>
#include <set>
#include <cassert>
#include "prelude.cpp"

using namespace std;

ifstream in("sudoku.in");

typedef bitset<9> cell;
typedef pair<int,int> pos;
typedef int val;
typedef map<pos,cell> aBoard;
queue<pos> q; // positions with exactly one value
// possible values the cell could be
vector<int> values(cell c) {
  vector<int> v;
  FOR(i,c.size())
    if(c.test(i) != 0) v.push_back(i+1);
  return v;
}

int valueof(cell c) { return values(c)[0]; }
string str(cell c) {
  return "[" + str(values(c)) + "]"; }

// set "0" positions to be any number,
// and add known values to the queue
aBoard parseBoard() {
  aBoard board;
  string line;
  char num;
  FOR(i,9) {
    getline(in, line);
    stringstream ss(line);
    FOR(j,9) {
      pos ij(i,j);
      ss >> num;
      board[ij].reset(); // all off
      if(num == '0') {
        board[ij].set(); // all on
      } else {
        board[ij].set(num-'1');
        q.push(ij);
      }
    }
  }
  return board;
}

// coords of row, column, and block positions
map<pos, set<pos> > peers;

void init() {
  FOR(i,9) FOR(j,9) {
    pos p(i,j);
    int base_i = i - i%3;
    int base_j = j - j%3;
    FOR(k,9) {
      if(k != i) // add row
        from[p].insert( pos(k,j) );
      if(k != j) // add column
        from[p].insert( pos(i,k) );

      pos block( base_i + k/3,
                 base_j + k%3 );
      if(block != p)
        from[p].insert(block);
    }
  }
}

bool isFinished(aBoard& b) {
  FOREACH(p, b) // p is <pos,cell>
    if(p->second.count() != 1)return false;
  return true;
}

void printBoard(aBoard& b) {
  FOR(i,9) {
    FOR(j,9)
      printf("%13s",
        str(b[pos(i,j)]).c_str());
    printf("\n");
  }
}

// returns true if contradiction
bool expand(aBoard& b) {
  while(!q.empty()) {
    pos p = q.front(); q.pop();
    assert(b[p].count() == 1);

    // remove c's bit from each neighbor
    FOREACH(nbr, peers[p]) {
      if(b[*nbr].count() == 1)
        continue;
      b[*nbr] &= ~b[p];
      switch(b[*nbr].count()) {
        case 0: return true;
        case 1: q.push(*nbr);
      }
    }
  }
  return false; // no contradictions!
}

// returns true if we found a solution
bool backtrack(aBoard b) {
  bool failed = expand(b);
  if(failed) return false;
  else if(isFinished(b)) {
    printBoard(b);
    return true;
  } else { // try every value for some cell
    vector<pair<int,pos> > choices;
    FOR(i,9) FOR(j,9) {
      pos ij = pos(i,j);
      int count = b[ij].count();
      if(count <= 1)
        continue;
```

```
      choices.push_back(
        make_pair(count, ij));
    } // find cell with smallest # choices
    sort(ALL(choices));
    pos ij = choices[0].second;
    vector<int> poss = values(b[ij]);
    FOREACH(num, poss) {
      b[ij].reset();
      b[ij].set(*num-1);
      q.push(ij);
      if(backtrack(b))
        return true;
    }
  } // should not ever get here...
  return false;
}

int main() {
  init();
  while(in.good()) {
    aBoard b = parseBoard();
    backtrack(b);
    return 0;
  }
}
```

11.13 GSM and Geometry Code

```
#include <complex>
#include <queue>
#include <set>
#include <limits>
#include "prelude.cpp"

using namespace std;

typedef complex<double> P;
namespace std {
  // a < b if: a has smaller y value, or,
  // if they have equal y value, smaller x
  bool operator<(const P& a, const P& b) {
    return imag(b-a) > 0. ||
           imag(b-a) == 0. &&
           real(b-a) > 0.; }
}

double e = 1e-9; // roundoff error fuzz

double cross(P a, P b)
  { return -imag(a*conj(b)); }
double   dot(P a, P b)
  { return  real(a*conj(b)); }
P       proj(P v, P u)
  { return u * P(dot(u,v)/dot(u,u)); }

#define angle arg
#define len abs
P unit(P v) { return v/len(v); }

typedef pair<P,P> Seg;

Seg from_pts(P a, P b) {
    return Seg(b-a, a); }

P tail(Seg s) { return s.second; }
P  pos(Seg s) { return s.second; }
P  dir(Seg s) { return s.first; }
P head(Seg s) { return pos(s) + dir(s); }

// angle between two lines
double angle(Seg a, Seg b) {
  return abs(angle(dir(b))
           - angle(dir(a))); }

namespace cis {
  template<typename NUM>
  bool between(NUM a, NUM b, NUM c) {
    if(a > c) swap(a,c);
    return a <= b && b <= c;
  }
}

bool in_seg(Seg l, P p) {
    bool in_x = cis::between(
        real(tail(l)), real(p),
        real(head(l)));
    bool in_y = cis::between(
        imag(tail(l)), imag(p),
```

```
        imag(head(l)));
    return in_x && in_y;
}

// general poly code -- not needed here
int ix_status;
// return true if segs intersect
bool intersect(Seg a, Seg b, P& p) {
  P ab = pos(b) - pos(a);
  double r0 = cross(ab, dir(b));
  double s0 = cross(ab, dir(a));
  double d = cross(dir(a), dir(b));
  double r =  r0/d, s = s0/d;
  p = pos(a) + dir(a)*P(r);

  bool on_a = cis::between(0.,r,1.),
       on_b = cis::between(0.,s,1.);
  ix_status = 0; // 4 = par
  if(d == 0) ix_status |= 4;
  if(r0 == 0) ix_status |= 8;
  if(on_a) ix_status |= 1;
  if(on_b) ix_status |= 2;
  return on_a && on_b; // 8 = coll
}

int ccw(P a, P b, P c,
                bool coll = false) {
    P d = b-a, f = c-a;
    double diff = cross(d,f);
    if (diff < -e) return -1;
    if (diff >  e) return 1;
    if(coll) return 0;
    if( real(d)*real(f) < 0 ||
        imag(d)*imag(f) < 0) return -1;
    if(dot(d,d) < dot(f,f)) return 1;
    return 0;
}

bool onseg(P p, Seg l)
    { return 0 == ccw(tail(l), head(l), p); }

// consider segs_i([a,b], [c5,c2]) above:
// intersect() returns false (which is right),
//  Still, 2 of the 4 possible
// endpoint-on-other-seg will always be true,
//  so we can just test three of them
bool segs_intersect(Seg ab, Seg cd, P& i) {
    return intersect(ab,cd, i) ||
         ((ix_status & 12) == 12 &&
        (onseg(tail(cd), ab) ||
         onseg(head(cd), ab) ||
         onseg(tail(ab), cd)));
}
/// end infrastructure ////////

vector<P> towers;
vector<P> cities;

typedef int node;
typedef int weight;
typedef pair<node,node> edge;
typedef pair<edge,weight> nw;

vector<edge> roads; // vector of city pairs
    // map of city pairs to bool
map<edge, bool> roadmap;

    // map of city pairs to weight
map<edge, weight> roadweights;


P closest_point(Seg l, P p) {
    P x1 = tail(l);
    P r = p - x1;
    P q = x1 + proj(r, dir(l));
    // if q is off the line segment,
    // return the closest endpoint
    if(!in_seg(l, q)) {
        if(abs(q-head(l)) < abs(q-tail(l)))
            return head(l);
        return tail(l);
    }
    return q;
}

P closest_tower(P q) {
    double min_dist = 1000000;
    P closest = q;
    FOR(i,towers.size()) {
        double dist = abs(q - towers[i]);
        if(dist < min_dist) {
            min_dist = dist;
            closest = towers[i];
        }
    }
    return closest;
}

int B,C,R,Q;

int main() {
  double x,y;
    int m,n;
    int casenumber = 0;
    while(true) {
    // init data structures
    roads.clear();
    cities.clear();
    towers.clear();
    roadweights.clear();

    casenumber++;

    // read in lines
    cin >>B >> C >> R >> Q;

    if(B == 0 && C == 0
    && R == 0 && Q == 0) {
      return 0;
    }

    FOR(i,B) {
      cin >> x >> y;
      towers.push_back(P(x,y));
```

```
    }

    FOR(i,C) {
      cin >> x >> y;
      cities.push_back(P(x,y));
    }

    FOR(i,R) {
      cin >> m >> n;
      // decrease m and n by one,
      // to account for indexing
      --m; --n;
      roads.push_back(edge(m,n));
      roadmap[edge(m,n)] = true;
    }

    // compute weights for all roads
    // to do so, we'll take each line and
    // compute closest-point-on-lines (Q-points)
    // for each tower to the line.
    // Then, find closest tower for each Q-pt.
    // For a given line, the cardinality
    // of the set of closest towers,
    // minus one, is the weight of the line.
    FOR(road_no,R) {
      edge citypair = roads[road_no];
      Seg road = from_pts(cities[citypair.first],
                cities[citypair.second]);
      vector<P> qs;
      set<P> closest_towers = set<P>();
      FOR(tower_no, B) {
        P tower = towers[tower_no];
        P q = closest_point(road, tower);
        P ct = closest_tower(q);
        closest_towers.insert(ct);
      }
      int weight = closest_towers.size() - 1;
      // duplicate weights for city pairs
      roadweights[citypair] = weight;
      // no matter which city comes first
      roadweights[invert(citypair)] = weight;
    }

    map<edge, weight> spath;
    floyd(roads, roadweights, spath, C);

    cout << "Case " << casenumber << ":"<<endl;
    FOR(i,Q) {
      int s,d;
      cin >> s >> d;
      --s; --d;
      edge sd = edge(s,d);
      // max path can't be greater
      // than number of cities/nodes!
      if(spath[sd] > C) cout << "Impossible";
      else cout << spath[sd];
      cout << endl;
    }
  }
  return 0;
}
```

12 printf() Guide

```
#include <iomanip>
#include <cstdio>

/*            printf/sprintf is your friend!
    Most of the time, problems say "two decimal places", and you want "%.2f"
printf codes:
  %d      signed integer            %s    C-style string
  %u    unsigned integer            %c    character
  %e/E  scientific notation         %%    percent sign
  %f    floating-point              %lf   long float (double)
  %g/G  shorter of %e or %f         %#f   force decimal place to be printed
  %x/X  unsigned hex                %#x   print with 0x prefix

"%12d"    means field of width 12, padded with spaces
"%012d"   means field of width 12, padded with zeroes

"%12.6f"  means at least 12 spaces wide, w/ 6 decimal places
"%-12.6f" means same as above, but left-aligned instead of right-aligned

"%5.10s"   means string printed with min 5 & max 10 chars
*/
   printf("|%s|\n", "hello");       // |hello|
   printf("|%.4s|\n", "hello");     // |hell|
   printf("|%10.4s|\n", "hello");   // |      hell|

   printf("|%lf|\n", big);         |654321.123456|
   printf("|%-20lf|\n", big);      |654321.123456       |
   printf("|%20lf|\n", big);       |       654321.123456|
   printf("|%020lf|\n", big);      |0000000654321.123456|

   printf("|%.10lf|\n", big);      |654321.1234560000|
   printf("|%.4lf|\n", big);       |654321.1235|

// printf will perform rounding! That's usually good, but it MAY not be what you want!
   printf("%.2f\n", 0.145);  // 0.14
   printf("%.2f\n", 0.1451); // 0.15

   trunc(2, .146); // 0.14

// Also, you don't have to embed the precision in the printf string:
   printf("%.*lf\n", 2, val); /* == */ printf("%.2lf\n", val);


// truncates strings to the specified number of places after the decimal
string trunc(int places, double val) {
  char buf[256];
  sprintf(buf, "%lf", val);
  string rv(buf);
  if(rv.find('.') != string::npos)
      rv.erase(rv.find('.') + 1 + places);
  return rv;
}

// Things you can do with cout that are not as easy to do with printf:
//        BOOL FORMATTING
  bool t = true;
  cout << t << endl; // 1
  cout << boolalpha;
  cout << t << endl; // true
```

```
//         NON-FIXED PRINTING
  cout << 0.1 << ", " << 1.0 << endl; // gives 0.1, 1
  printf("%f, %f\n", 0.1f, 1.0f); // gives 0.100000, 1.000000

//         INTERNAL ALIGNMENT
  cout << setw(10) << setfill('*') << internal << -1.0 << "." << endl;
          // gives -********1

    const float tenth = 0.1;
    const float one   = 1.0;
    const float big   = 1234567890.0;
    const float neg1  = -1.0;

    printf("%f\n", tenth);
    printf("%f\n", one);

    cout << "A. "                    << tenth << ", " << one << ", " << big << endl;
    cout << "B. " << fixed           << tenth << ", " << one << ", " << big << endl;
    cout << "C. " << scientific      << tenth << ", " << one << ", " << big << endl;
    cout << "D. " << fixed << setprecision(3) << tenth << ", " << one << ", " << big << endl;
    cout << "E. " << setprecision(20) << tenth << endl;
    cout << "F. " << setw(8) << setfill('*') << 34 << 45 << endl;
    cout << "G. " << setw(8) << 34 << setw(8) << 45 << endl;
/*
A. 0.1, 1, 1.23457e+09
B. 0.100000, 1.000000, 1234567936.000000
C. 1.000000e-01, 1.000000e+00, 1.234568e+09
D. 0.100, 1.000, 1234567936.000
E. 0.10000000149011611938
F. ******3445
G. ******34******45
*/
   cout.unsetf(ios::floatfield);
```

13 Java Regex Quickref

```
Char classes:
        [abc] [^abc] [a-zA-Z] [a-z&&[^bc]]==[ad-z]
Predefined char classes:
        \d      digit           \D      not digit
        \s      whitespace      \S      blackspace
        \w      word char: [a-zA-Z_0-9]
POSIX classes:
        \p{Lower}, \p{Upper}, \p{Alnum} etc
Boundaries:
        ^       start of line
        $       end of line
        \b      word boundary   \B non-word boundary
        \A      start of input
        \G      end of prev. match
        \Z      end of input (ignoring trailing newline)

(?i:__whatever__) -- matches __whatever__ case-insensitive
Flag for dotall mode is s
```

14 Java Misc Class Methods

Methods of the `Area` class:

```
Rectangle getBounds(), exclusiveOr, intersect,
add, subtract, contains(PDD)
```

Methods of the `Scanner` class:

```
TYPEs:  BigDecimal, BigInteger,
        Int, Double, Line

Scanner(String),
hasNext(), hasNext(String pattern),
hasNextTYPE(),
String next(), next(String pattern),
skip(pattern),
useDelimiter(pattern), useRadix(int)
```

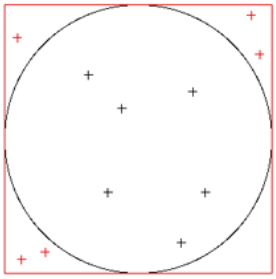
BigInteger also has:

```
testBit, setBit, shiftLeft, mod,
remainder, xor (and friends), bitCount,
bitLength, clearBit, flipBit, gcd,
getLowestSetBit, isProbablyPrime(int)
pow, negate, abs, and signum
```

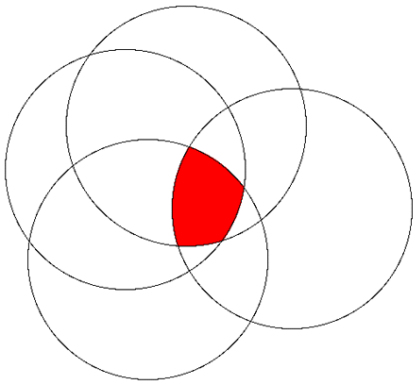
File input, Regular Expressions:

```
class Scan {
   Scanner fin;
   private void run() {
       try {
           fin = new Scanner(
                 new File("test.in"));
       } catch(Exception e) {
           System.exit(2);
       }
       while(fin.hasNextInt()) {
           System.out.println(
               fin.nextInt());
       }
   }
    static void main(String[] args) {
       new Scan().run();
       String s1 = "Hello World";
       System.out.println( // false
               s1.matches("o"));
       System.out.println( // true
               s1.matches("^.*o.*$"));
       System.out.println(
               s1.replaceAll("([A-Z])",
                             "O\'$1"));
       s1.toLowerCase();
       String[] splits = s1.split("");
       System.out.println(splits.length);
       for(String s : splits)
           System.out.println("|"+s+"|");
   }
}
```

15 Monte Carlo circle area



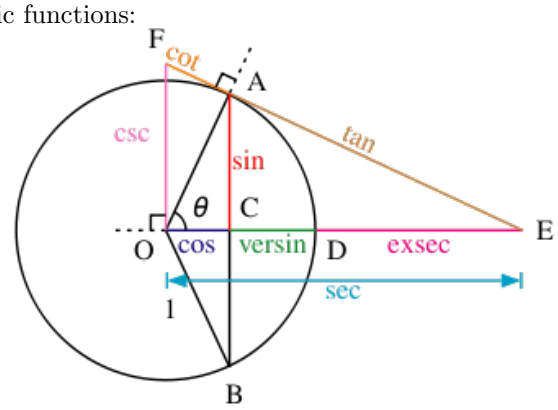
A circle with radius r and area $\pi/4r^2$ is bounded by a square of area $2r * 2r = 4r^2$. Randomly select n points within the square, of which k are within the circle. The ratio of points inside each figure should be the same as the ratio of areas of the two figures. $\frac{k}{n} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$.



A simple extension can be used to estimate the area of any number of intersecting circles. Select random points within one circle's bounding box, and count how many fall inside every circle. The area of the intersections is then $\frac{k}{n}4r^2$

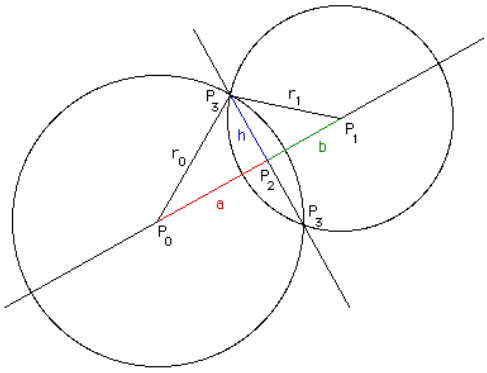
16 Geometry/Trigonometry Reference

Illustration of geometric construction of trigonometric functions:



Ranges		of trig functions:
any	cos	-1 to 1
any	sin	-1 to 1
any	tan	any
-1 to +1	*acos*	0 to *pi*
-1 to +1	*asin*	$-pi/2$ to $pi/2$
any	*atan*	$-pi/2$ to $pi/2$

Finding point(s) of intersection of two circles:



$$x_3 = x_2 + -h(y_1 - y_0)/d$$

$$y_3 = y_2 - +h(x_1 - x_0)/d$$

Number of lattice points inside a circle of radius r centered at the origin is

$$N(r) = 1 + 4\lfloor r \rfloor + 4\sum_{i=1}^{\lfloor r \rfloor} \lfloor \sqrt{r^2 - i^2} \rfloor$$

First few values are 1, 5, 13, 29, 49, 81...

16.1 Java Geometry

```
P dir(P a, P b) {
    return new P(b.x - a.x,
                 b.y - a.y); }
P perp(P a) {
    return new P(-a.y, a.x); }

// a dot b = |a||b| cos theta
// a dot a = |a|^2
double dot(P a, P b) {
    return a.x*b.x +
           a.y*b.y; }

// a x b = |a||b| sin theta = 2x triangle area
// >0 if b is right of a, <0 if b is right of a
double cross(P a, P b) {
    return dot(perp(a), b);
}
// Computes intersection of lines ab and cd.
P intersect(P a, P b, P c, P d) {
   P ab = dir(a,b), cd = dir(c,d);
   double r0 = cross(dir(a,c), cd);
   double r = r0 / cross(ab, cd);
   return new P(a.x + r*ab.x,
                a.y + r*ab.y);
}
// Closest point to C on line AB
P closest(P a, P b, P c) {
   P ab = dir(a,b), ac = dir(a,c);
   double r = dot(ab,ac)/dot(ab,ab);
   return new P(a.x + r*ab.x,
                a.y + r*ab.y);
}
double area(P a, P b, P c) {
   return cross(dir(a,b), dir(a,c));
}
double length(double[] v) {
   return Math.sqrt(dot(v,v));
}
double dot(double[] v, double[] z) {
    return v[0]*z[0]
         + v[1]*z[1]
         + v[2]*z[2];
}

// len of result = |v||z| sin theta
double[] cross(double[] v,
               double[] z) {
    return new double[] {
        v[1]*z[2] - v[2]*z[1],
        v[2]*z[0] - v[0]*z[2],
        v[0]*z[1] - v[1]*z[0] };
}
double[] dir(double[] a,
             double[] b) {
    return new double[] {
        b[0] - a[0],
        b[1] - a[1],
        b[2] - a[2] }; }
```