Team combat 世界大会用リファレンス

・STL

0. ヘッダ

コンパイルが通らなかったら順番に突っ込んでみる

```
<iostream> <iomanip> <fstream> <sstream>
<vector> <list> <map> <set> <queue> <deque>
<algorithm> <numeric> <iterator> <functional>
<memory> <utility>
```

1. 入出力

```
cin>>x;
cin>>hex>>h; // 16進数
cin.get(); // 一文字
getline(cin,s); // string に読み込み

cout<<x;
cout<<setiosflags(ios::fixed)<<setprecision(5)<<d; // 小数点以下5桁

#include <sstream>
ostringstream oss; // 文字列ストリーム
oss<<foo; // 普通に出力
oss.str(); // 文字列(string)をゲット

istream_iterator<int,char> intstream(cin), eof;  // 入力ストリーム
ostream_iterator<int, char> output (cout, "\n"); // 出力ストリーム セパレータは毎回出力される
```

2. コンテナ

```
// 文字列
string s;
str=s.substr(pos[, length]);
pos=s.find(str[, startpos]); // 見つからなければstring::npos
pos=s.rfind(str[, startpos]);
pos=s.find_first_of(str[, startpos); // strのいずれかの文字を前から探す
pos=s.find_last_of(str[, startpos]); // 後ろから
pos=s.find_first_not_of(str[, startpos]); // 含まれないものを前から探す

s.insert(pos,str); // posのところにstrを挿入。破壊的
s.erase(pos,length); // posからlength文字を削除。破壊的
s.replace(pos,length,str); // s.erase(pos,length),s.insert(pos,str); と同じ

// その他コンテナ
vectot<T> v;
v[n] // 参照

list<T> l; // 挿入削除が早い。push_front(),pop_front()がある。
l.remove(x); // xと等しい要素を全部削除
l.remove_if(f); // 関数版
l.sort();
l.reverse();
l.unique(); // 隣接同一要素を削除

stack<T> queue<T>
```

```
deque<T> dq; // stack+queue
dq.push_back(x);
dq.push_front(x);
dq.pop_back();
dq.pop_front();

map<K,E> m;
multimap<K,E> m;
set<E> s;
multiset<E> s;
m[k]=v;
m.insert(make_pair(k,e));
s.insert(v);
// multimapはelementでソートされない！(たぶん)
// map/set は挿入や削除によりイテレータが無効にならない。
// vector/dequeは無効になる。
multiset<T> ms;
ms.lower_bound(e); // 一致するものの下限
ms.upper_bound(e); // 一致するものの上限
pair<it,it> ms.equal_range(e); // 上限と下限

// 以下の集合演算はすべてあらかじめソートされていること。
bool = includes(a.begin(),a.end(),b.begin(),b.end()); // bがaに含まれているか?
set_union(a.begin(),a.end(),b.begin(),b.end(),output); // 和集合
set_intersection(a.begin(),a.end(),b.begin(),b.end(),output); // 積集合
set_difference(a.begin(),a.end(),b.begin(),b.end(),output); // output=a-b
set_symmetric_difference(a.begin(),a.end(),b.begin(),b.end(),output);
// symmetric_differenceは(a∪b)-(a∩b)
// 出力をsetなどに挿入したい場合はoutputをinserter(c,c.begin())にせよ。

priority_queue<T,Container=vector<T>,Compare=less<T> > pq;
pq.push(),pq.pop(),pq.front(); // queue と同じ
// 優先順位をデータ構造の中に持たざるを得ないので、めんどくさい。
// 優先順位がデータとあんまり関係が無い場合はmultimapで代用できそう。
```

3. アルゴリズム

```
/* 初期化 */
fill(first,last,val);
fill_n(output,size,val);

copy(first,last,output);
copy_backward(first,last,result); // 逆にコピー,すべての引数がbidirectional

generate(first,last,GenFunc);
generate_n(output,size,GenFunc);

swap_ranges(first,last,first2); // 並列シーケンスをスワップ

/* 検索 */
iterator find(first,last,val); // 最初に一致するところ。線形検索。
iterator find_if(first,last,Pred); // 最初にpredがtrueを返すところ

adjacent_find(first,last[,BinPred]); // 最初に述語を満たす隣接二要素(デフォルト==)
find_first_of(first1,last1,first2,last2[,BinPred]); シーケンス2に含まれる最初のもの
search(first1,last1,first2,last2[,BinPred]); // サブシーケンスを探す
find_end(first1,last1,first2,last2[,BinPred]); // 最後のサブシーケンスの出現

max_element(first,last[,Comp]); // 最大要素(lessを指定)
min_element(first,last[,Comp]); // 最小要素(〃)

mismatch(first1,last1,first2,last2[,BinPred]); // シーケンスの最初の不一致要素のペア
```

```
/* インプレース変換 */
reverse(first,last); // 反転

replace(first,last,val,newval); // val を newval に置き換え
replace_if(first,last,Pred,newval); // Pred が true のところを置き換え
replace_copy(first,last,output,val,newval); // 純粋関数的。Output に結果が入る
replace_copy_if(first,last,output,Pred,newval); // 同じく

rotate(first,middle,last); // [first,middle) <-> [middle,last]
rotate_copy(first,middle,last,output); // 同じくコピー版

partition(first,last,Pred); // Pred を満たす要素を前方に集める。前方の終了反復子が返る
stable_partition(first,last,Pred); // stable 版。上より低速、メモリ消費量大

bool next_permutation(first,last[,Comp]); // 順列生成
bool prev_permutation(first,last[,Comp]); // 逆

inplace_merge(first,middle,last[,BinFunc]); // ソート済みシーケンスをマージする

random_shuffle(first,last[,Generator]); // シーケンスをランダムにする Gen::Int->Int

/* 削除 */

iterator remove(first,last,val); // 指定する要素を前に詰める。残存要素の終端を返す。
Iterator remove_if(first,last,Pred); // 関数版
iterator remove_copy(first,last,output,val); // 純粋関数版
iterator remove_copy_if(first,last,output,Pred); // 同関数版

iterator unique(first,last[,BinPred]); // 等価隣接グループの内最初以外を削除
iterator unique_copy(first,last,output[,BinPred]); // 純粋関数版

/* スカラー生成 */
cnt = count(first,last,val); // 一致する要素の数(一般に整数)を返す
cnt = count_if(first,last,Pred); // Pred を満たす要素の数を返す

#include <numeric> が要るかも…
val accumulate(first,last,initial[,BinFunc]); // foldl のことか。デフォルトでは+
val inner_product(first1,last1,first2,init-val[,add,times]);
// foldl add init-val $ zipWith times ls1 ls2 と同じ。デフォルトでは内積が計算される
bool equal(first1,last1,first2[,BinPred]); // 等価性テスト
bool lexicographical_compare(first1,last1,first2,last2[,Comp]); // 辞書順比較

/* シーケンス生成 */
iterator transform(first,last,output,UnaryFunc); // map
iterator transform(first1,last1,first2,output,BinFunc); // zipWith

iterator partial_sum(first,last,output[,BinFunc]); // scanl1
iterator adjacent_difference(first,last,output[,BinFunc]); // 初項+隣接二要素差

for_each(first,last,Func); // mapM_

/* 順序付きコレクション */

sort(first,last[,Comp]);
stable_sort(first,last[,Comp]);
nth_element(first,middle,last[,Comp]); // middle の場所に middle 番目の要素が入る

bool binary_search(first,last,val[,Comp]); // 二分探索を行う
iterator lower_bound(first,last,val[,Comp]); // 一致する下限を返す
iterator upper_bound(first,last,val[,Comp]); // 上限を返す
pair<it,it> equal_range(first,last,val[,Comp]); // 両方返す
```

```
iterator merge(first1,last1,first2,last2,output[,Comp]); //ソート済みリストのマージ
```

4. 関数オブジェクト

```
plus<T>       // +
minus<T>      // -
multiplies<T> // *
divides<T>    // /
modulus<T>    // %
negate<T>     // 単項 -

equal_to<T>      // ==
not_equal_to<T>  // !=
greater<T>       // >
less<T>          // <
greater_equal<T> // >=
less_equal<T>    // <=

logical_and<T> // &&
logical_or<T>  // ||
logical_not<T> // !

not1(UnaryPred);  // 一引数述語の論理を反転した関数を返す
not2(BinaryPred); // 二引数述語の論理を反転した関数を返す

bind1st(BinaryFunc,val); // 部分適用 : bind1st f v = \x -> f v x
bind2nd(BinaryFunc,val); // 同じく   : bind2nd f v = \x -> f x v
```

・グラフ

-ダイクストラ

開始点の距離を 0 とする。
(未確定頂点のある限り)
未確定頂点のうちで最短距離の点(これをヒープで探索するとよい)を確定点にする。
そこから到達できる頂点に距離を書き込む。必要ならルートも付与する。

-ウォーシャルフロイド

x->y のパスと y->j のパスが存在すれば、x->j のパスが存在することから導かれる。
最短ルートもほしい場合はどこで中継したかを覚えておけばいける。

```
for (int y=0;y<v;y++)
  for (int x=0;x<v;x++)
    if (a[x][y]!=0)
      for (int j=0;j<v;j++)
        if (a[y][j]!=0)
          if (a[x][j]==0 || a[x][y]+a[y][j]<a[x][j])
            a[x][j]=a[x][y]+a[y][j];
```

-最小全域木

```
エッジを短い順にサイクルができないものを足してゆく。
サイクルができるかどうかは union-find で調べると高速。
```

-union-find (動作確認済み)

```
class union_find{
public:
  union_find(int n){
    for (int i=0;i<n;i++)
      dat.push_back(i);
  }

  void unite(int a,int b){
    dat[find(a)]=find(b);
  }
  int find(int n){
    if (dat[n]==n) return n;
    else return dat[n]=find(dat[n]);
  }
  vector<int> dat;
};
```

-安定結婚 (動作確認済み)

```
// 安定結婚 それぞれの順位を渡す。
// 女性がどの男性と結婚するのかを返す。
vector<int> stable_match(vector<vector<int> > &m,vector<vector<int> > &w)
{
  int n=m.size();
  vector<int> next(n,-1);
  vector<vector<int> > rank(n,vector<int>(n+1,n));
  vector<int> fian(n,n);

  for (int i=0;i<n;i++)
    for (int j=0;j<n;j++)
      rank[i][w[i][j]]=j;

  for (int i=0;i<n;i++){
    for (int c=i;c!=n;){
      int t=m[c][++next[c]];
      if (rank[t][c]<rank[t][fian[t]])
        swap(c,fian[t]);
    }
  }
  return fian;
}
```

-深さ優先探索(最終手段、アドホックな最適化と組み合わせ)
-幅優先探索(探索ノード数がおおよそ 100 万以下なら)

-最良優先、反復深化

各ノードに対して優先度を設定。
Open 集合の中で最小のものを選択しながら探索。
優先度関数を深さの逆にすると深さ優先、
深さにすると幅優先探索になる。
最短路が見つかる保証は無い。

-A* (良いヒューリスティック関数があるときの計算量が多そうな問題向け、15 パズルなど)

最良優先と一点のみ異なる。
優先度関数では無くて、それに深さを足したものが小さいものから探索する。
関数の値が常に本当の距離よりも小さい値を返すのであれば
最短路が見つかることが保証される。
なるべく本当の距離に近い値を返すほうが探索量は少ない。

-Min-Max 探索 alpha-beta 探索 (二人完全情報零和ゲームの必勝手)

min-max は自分の手番なら最大のスコアになるように、
相手の手番なら最小のスコアになるように選ぶ。
Alpha-beta は min-max を大幅に枝刈りする。
各探索において、下限値(alpha)と上限値(beta)を用いて枝刈りを行う。
自分の phase では、beta 以上の値が出たら、その探索では beta 以上の値になるのが
確実なので、beta を結果として返せばよい。計算の途中では下限が更新されるので、
alpha をそれぞれの値との max で更新する。
相手の phase では逆。Alpha 以下のものが出れば計算終了。
beta が徐々に下がるのでそれを min で更新。

- 貪欲なアルゴリズムが使えないか?
見極めには注意せよ。証明できるとベスト。

最大フロー(動作保障あり)

```
// 有向グラフのときは size を正と負にして保持。
// 無向グラフのときは size を両方正にする。
class edge{
public:
  edge():size(0),flow(0){}
  edge(int size,int flow):size(size),flow(flow){}
  int size,flow;
};

typedef vector<vector<edge> > graph;
graph make_graph(int size){ return graph(size,vector<edge>(size)); }

// パスを見つける
bool search(graph &g,int source,int sink,vector<int> &val,vector<int> &par)
{
  for (int i=0;i<g.size();i++) val[i]=0;

  multimap<int,int> pq;
  pq.insert(make_pair(-0x7fffffff,source));
  par[source]=-1;
  val[source]=0x7fffffff;

  while(!pq.empty()){
    int v=-pq.begin()->first,t=pq.begin()->second;
    pq.erase(pq.begin());
    for (int i=0;i<g.size();i++){
      if (g[t][i].size==0)
        continue;
      //int iv=min(v,-g[t][i].flow+max(0,g[t][i].size)); // 有向グラフ
      int iv=min(v,g[t][i].size-g[t][i].flow); // 無向グラフ
      if (val[i]<iv){
        val[i]=iv;
        par[i]=t;
        if (i==sink) return true; // これを入れると微妙に速くなる?
        pq.insert(make_pair(-iv,i));
      }
    }
```

```
  }

  return val[sink]!=0;
}

// 最大フローを返す。gにフローを書き込む
int max_flow(graph &g,int source,int sink)
{
  vector<int> v(g.size()),p(g.size());

  while(search(g,source,sink,v,p)){
    //cout<<"route size: "<<v[sink]<<endl;
    int val=v[sink];
    for (int f=sink,t=p[f];t!=-1;f=t,t=p[f]){
      g[t][f].flow+=val;
      g[f][t].flow=-g[t][f].flow;
      //cout<<t<<"->"<<f<<" : "<<g[t][f].flow<<endl;
    }
  }

  int ret=0;
  for (int i=0;i<g.size();i++)
    ret+=g[i][sink].flow;
  return ret;
}
```

・DP

典型的な問題:
最長共通部分列
edit distance
行列の積のコスト
…
たいていはアドホックなアルゴリズム考案による

・線形計画

目的関数: $\boldsymbol{c}^T\boldsymbol{x}$ → 最小
制約条件: $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b},\boldsymbol{x}\geq\boldsymbol{0}$ (任意の線形計画問題をこの形に変形可能)
ごくごくナイーブには全実行可能基底解を列挙すれば何とかなる。

シンプレックス法
(0)初期実行可能基底解 $(\boldsymbol{x}_B,\boldsymbol{x}_N)=(\boldsymbol{B}^{-1}\boldsymbol{b},\boldsymbol{0})$ を選ぶ。 $\bar{\boldsymbol{b}}=\boldsymbol{B}^{-1}\boldsymbol{b}$ とおく。
(1)シンプレックス乗数 $\pi=(\boldsymbol{B}^T)^{-1}\boldsymbol{c}_B$ を計算する。
(2) $\boldsymbol{c}_N^T-\pi^T\boldsymbol{N}\geq\boldsymbol{0}$ が満たされているなら、最適基底解が得られている。
そうでなければ $c_k-\pi^T\boldsymbol{a}_k<0$ であるような非基底変数 x_k を一つえらぶ。
(3)ベクトル $\boldsymbol{y}=\boldsymbol{B}^{-1}\boldsymbol{a}_k$ を計算する。
(4)ベクトル $\boldsymbol{y}$ に正の要素が無ければ問題は有界で無いので終了。そうでなければ
$\Delta=min\left\{\bar{b}_i/y_i,y_i>0(i=1,\ldots,m)\right\}$ および、最小を取るiを求める。
(5)非基底変数 x_k の値を Δ 、それ以外の非基底変数の値を0と置く。基底変数 $\boldsymbol{x}_B$ を
$\boldsymbol{x}_B=\bar{\boldsymbol{b}}-\Delta\boldsymbol{y}$ と置く。非基底変数 x_k を基底変数とし、ステップ(4)で求めたiに対応する基

底変数を非基底変数として基底を更新する。ステップ(1)へ戻る。

・数論

素数
ふるいの際にどれにふるわれたかを記憶しておくことにより
超高速に(連続数の)素因数分解が可能。

フェルマーの小定理

・p が素数の場合、整数 a に対し
a^p ≡ a (mod p)
・特に a が p の倍数でなければ
a^(p - 1) ≡ 1 (mod p)

gcd

```
int gcd(int a,int b){ return b==0?a:gcd(b,a%b); }
```

拡張 gcd

x, y を 0 でない自然数とし, c=GCD(x,y)とする。このとき,
ax+by=c となる整数 a,b が存在する。そして, この a,b は実際に計算することが出来る。

```
// a>0,b>0 に対して ax+by=g なる x,y,g=gcd(a,b) を求める
int exgcd(int a,int b,int &x,int &y)
{
  if (b==0)
    return x=1,y=0,a;
  else{
    int d=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return d;
  }
}
```

ヨセフス数(Josephus) (未テスト)

```
概要
n 人の輪から k 人ごとに殺していったとき、s 番目に殺される人の番号(1〜n)を求める。

入力仕様
n>=1
k>=1
1<=s<=n

int Josephus(int n,int k,int s)
{
  int x = k*s ;
   while(x>n) x = ((x-n)*k-1)/(k-1) ;
   return x ;
}
```

・中国の剰余定理

m_1, m_2 を互いに素な正整数(即ち最大公約数が1)とする。a_1, a_2 を整数する。このとき,

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$

を満たす整数 x が $m_1 m_2$ を法にして唯ひとつ存在する。

```
int chinese_reminder(int a1,int a2,int m1,int m2)
{
  int x,y;
  if (exgcd(m1,m2,x,y)!=1) return -1; // 入力がおかしい。
  x=a1+(a2-a1)*x*m1;
  while(x<0) x+=m1*m2;
  return x;
}
```

n 乗数の和

powersum(n,m) = $\sum_{i}^{n} i^m$ を計算する。

```
long_int combination(long_int n,long_int r)
{
  long_int ret=1;
  for (long_int i=n;i>r;i-=1)       ret=ret*i;
  for (long_int i=1;(n-r+1)>i;i+=1) ret=ret/i;
  return ret;
}

long_int power(long_int n,int m)
{
  if(m==0) return 1;
  if(m==1) return n;
  return sq(power(n,m/2))*power(n,m%2);
}

long_int powersum(long_int n,int m)
{
  if(m==0) return n;
  vector<long_int> mem(m+1);
  mem[0]=n;
  for(int k=1;k<=m;k++){
    long_int sum = power(n+1,k+1)-1;
    for(int i=0;i<k;i++)
      sum -= mem[i]*combination(k+1,i);
    mem[k]=sum/long_int(k+1);
  }
  return mem[m];
}

long_int powersum_wise(long_int n,int m){
  if(n==0) return 0;
  if(m==0) return n;
  if(m==1) return n * (n+1) / 2;
  if(m==2) return n * (n+1) * (n*2+1) / 6;
  if(m==3) return sq(n*(n+1)/2);
  if(m==4) return n * (n+1) * (n*2+1) * (sq(n)*3 + n*3 - 1)/30;
  if(m==5) return sq(n*(n+1))*(sq(n)*2 + n*2 - 1)/12;
  if(m==6) return n * (n+1) * (n*2+1) * (power(n,4)*3+power(n,3)*6-n*3+1)/42;
  if(m==7) return sq(n*(n+1)) * (power(n,4)*3+power(n,3)*6-sq(n)-n*4+2)/24;
```

```
  return powersum(n,m);
}
```

剰余の和

$\sum_{i=1}^{n} n \bmod i$ を $O(\sqrt{n})$ で計算できる。

n mod i を並べると等差数列に分割できることによる。
差 i の等差数列が floor(n/(i+1))+1 ～ floor(n/i) の区間に存在する。

```
int mod_sum(int n)
{
  int ret=0,up=n,cur=1;
  for (;;){
    int st=(n/(cur+1))+1;
    if (st>up) break;
    int ofs=n%up;
    int range=up-st+1;
    ret+=range*(range-1)/2*cur+ofs*range;
    cur++;
    up=st-1;
  }
  for (int i=1;i<=up;i++)
    ret+=n%i;
}
```

・幾何

平面幾何
以下では typedef complex<double> pt; を前提とする。

＊注意！！
complex には大小比較演算子が定義されていないが、定義する際には namespace std で囲うように(namespace std にあるクラス全般)。そうしないと、max や min やがちゃんとコンパイルできない。

atan2 相当

```
arg(v); // 返す値の範囲は (-PI,PI] であるので、注意すること!
```

内積 u・v

```
real(conj(u)*v);
```

行列式 det[u,v]

```
imag(conj(u)*v);
```

三角形の符号付面積

```
double area(pt a,pt b,pt c)
{
  return imag(conj(b-a)*(c-a))*0.5;
}
```

ポリゴンの符号付面積

```
double polygon_area(vector<pt> &p)
```

```
{
  double ret=0;
  for (int i=1;i<p.size()-1;i++)
    ret+=area(p[0],p[i],p[i+1]);
  return ret;
}
```

直線 a+su,b+tv (a,b,u,v はベクトル、s,t はスカラ変数) の交点 (s,t を返す)

```
pair<double,double> cross_lines(pt a,pt u,pt b,pt v)
{
  double det=imag(conj(u)*v);
  if (abs(det)<epsilon) throw "交差しない";
  return make_pair(
    -imag(conj(v)*(b-a))/det,
    -imag(conj(u)*(b-a))/det);
}
```

直線 x=a+su と円 |x-c|=r の交点 (s を二つ返す)

```
pair<double,double> cross_circle_and_line(pt a,pt u,pt c,double r)
{
  // norm(u)<ε の時は別に判定すること!
  double b=real(u*conj(a-c));
  double d=b*b-norm(u)*(norm(a-c)-r*r);
  if (d<0) throw "交差しない";
  return make_pair(
    (-b+sqrt(d))/norm(u),
    (-b-sqrt(d))/norm(u));
}
```

円 |x-c|=r と |x-d|=s の交点

```
pair<pt,pt> cross_circles(pt c,double r,pt d,double s)
{
  double di=abs(c-d);
  if (di>r+s || di<abs(r-s))
    throw "交差しない";
  if (di<epsilon && r==s)
    throw "同一円";
  double l=((r*r-s*s)/di+di)/2.0;
  double n=sqrt(r*r-l*l);
  pt e=c+(d-c)*l/di;
  pt p=(d-c)*n/di*pt(0,-1);
  return make_pair(e+p,e-p);
}
```

点 a を通る|x-b|=r への接線

```
// 点 a を通る|x-b|=r への接線をもとめ、
// u, unum, z,znum を返す。ここで

//   \s -> a + s * u*(b-a)
//   \s -> a + s * conj(u)*(b-a)
// が接線であり、

//   a + z * (b-a)
//   a + conj(z) * (b-a)
// が接点である。
void point_passing_circle_touching_line(
  cplx a,cplx b,double r,int& unum,cplx& u,int& znum,cplx& z)
{
  if(r*r > norm(b-a)){
```

```
    unum = znum = 0;
    return;
  }
  if(r*r == norm(b-a)){
    if(r==0){
      unum = 3;znum = 1;
      z = 0;
      return;
    }
    u = cplx(0,1);
    z = 0;
    unum = 2; znum = 1;
    return;
  }
  u = cplx(sqrt(norm(b-a)-r*r),r);
  z = u / abs(b-a);
  if(r==0){
    unum=znum=1;
    return;
  }
  unum=znum=2;
  return;
}
```

凸包 (O(nlogn) 動作保証一応あり)

```
// 凸包を生成する。出来る凸包は反時計回り。最初と最後は同じ座標。
vector<pt> convex_hull(vector<pt> &v)
{
  // 最も左下のものを選ぶ。そういうように<演算子を定義せよ
  pt o=*min_element(v.begin(),v.end());

  multimap<double,pt> m;
  for (int i=0;i<v.size();i++) // 仰角でソート
    if (abs(v[i]-o)>=epsilon)
      m.insert(make_pair(arg(v[i]-o),v[i]));
  m.insert(make_pair(10,o)); // 最後に初期点を追加する

  // 順番に突っ込みながら凹角を削る。微小に凸でもほぼ直線なら削る。
  // 仰角が同一の複数の点があった場合もうまいこと働いて最後には正しい結果が得られるはずである。
  // 同一直線上の点もうまいこと最長辺のみになるはず。
  // 重複な点があった場合もうまく削られるはず。
  vector<pt> ret(1,o);
  for (multimap<double,pt>::iterator p=m.begin();p!=m.end();p++){
    ret.push_back(p->second);
    for (int n=ret.size();n>=3&&area(ret[n-3],ret[n-2],ret[n-1])<epsilon;n--)
      ret[n-2]=ret[n-1],ret.pop_back();
  }
  return ret;
}

bool operator<(const pt &a,const pt &b)
{
  if (abs(imag(a)-imag(b))>=epsilon) return imag(a)<imag(b);
  return real(a)<real(b);
}
```

最近点対問題(点の集合の中で最も短い距離の点対を求める)

・分割統治で。
x 座標でソート。

左半分と右半分に対してそれぞれ最短(これを m とする)を求める。
領域にまたがる最短距離であるが、
境界から m の距離にあるものを抜き出し、y 座標でソートし
(分割しているのでマージソート可能)
下から 6 個までの範囲にあるものの距離の中で最短を選べばよい。

・数値計算

ニュートン法 (収束は早いが安定性に欠ける)

$f(x)=0$ なる x を求める漸化式

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$$

微分方程式の数値的解法

-オイラー法

$$\frac{dy}{dt}=f(t,y),\ y(a)=\alpha$$

なる微分方程式に対して y(t)を、t における傾きが dy/dt であることを用いて

$$w_0=\alpha$$
$$w_{i+1}=w_i+hf(t_i,w_i)$$

により数値的に求めることが出来る。ただし h は微小な値。
かなり誤差がでかい。(刻み幅の一次に比例?)

```
// dy/dt=f(t,y), y(a)=alp のとき y(b)を求める。分割数 n。
double eular_method(double a,double b,double alp,int n)
{
  double h=(b-a)/n,t=a,w=alp;
  for (int i=0;i<n;i++){
    w+=h*f(t,w);
    t+=h;
  }
  return w;
}
```

-ルンゲクッタ法
4 次のルンゲクッタ法

$$w_0=\alpha$$
$$k_1=hf(t_i,w_i)$$
$$k_2=hf(t_i+\frac{h}{2},w_i+\frac{k_1}{2})$$
$$k_3=hf(t_i+\frac{h}{2},w_i+\frac{k_2}{2})$$

$$k_4 = hf(t_{i+1}, w_i + k_3)$$
$$w_{i+1} = w_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

オイラー法よりもかなり精度が良い。
(誤差は刻み幅の 4 次に比例?)

```
// dy/dt=f(t,y), y(a)=alp のとき y(b)を求める。分割数 n。
double runge_kutta(double a,double b,double alp,int n)
{
  double h=(b-a)/n,t=a,w=alp;
  for (int i=0;i<n;i++){
    double k1=h*f(t,w);
    double k2=h*f(t+h/2,w+k1/2);
    double k3=h*f(t+h/2,w+k2/2);
    double k4=h*f(t+h,w+k3);
    w+=(k1+2*k2+2*k3+k4)/6;
    t+=h;
  }
  return w;
}
```

・長整数

```
class long_int{
public:
  long_int(){}
  long_int(string s){
    for (int i=0;i<s.length();i++)
      d.push_front(s[i]-'0');
    normalize();
  }

  long_int &normalize(){
    for (int i=d.size()-1;i>=0;i--){
      if (d[i]!=0) break;
      d.pop_back();
    }
    return *this;
  }
  int operator[](int i){ return i>=d.size()?0:d[i]; }
  int size(){ return normalize(),d.size(); }

  long_int &operator+=(long_int n){
    int l=max(size(),n.size());
    d.resize(l+1,0);
    for (int c=0,i=0;i<=l;i++)
      d[i]=(c=d[i]+n[i]+c/10)%10;
    return normalize();
  }
  long_int &operator-=(long_int n){
    assert(size()>=n.size());
    for (int c=0,i=0;i<d.size();i++)
      d[i]=((c=d[i]-n[i]+(c-9)/10)+10)%10;
    return normalize();
  }
  long_int &operator*=(int n){
    d.push_back(0);
    for (int c=0,i=0;i<d.size();i++)
```

```
    d[i]=(c=d[i]*n+c/10)%10;
    return normalize();
  }

  long_int operator+(long_int n){ return long_int(*this)+=n; }
  long_int operator-(long_int n){ return long_int(*this)-=n; }
  long_int operator*(int      n){ return long_int(*this)*=n; }

  long_int operator*(long_int n){
    long_int ret;
    int ka=size(),kb=n.size();
    ret.d.resize(ka+kb,0);
    for (int i=0;i<ka;i++)
      for (int c=0,j=0;j<=kb;j++)
        ret.d[i+j]=(c=ret.d[i+j]+d[i]*n[j]+c/10)%10;
    return ret.normalize();
  }
  long_int &operator*=(long_int n){ return (*this)=(*this)*n; }

  // 単精度除算
  int div(int n){
    int c=0;
    for (int i=size()-1;i>=0;i--)
      d[i]=(c=c%n*10+d[i])/n;
    return c%n;
  }

  // 左にシフト (10^n 倍)
  long_int &operator<<=(int n){
    for (int i=0;i<n;i++) d.push_front(0);
    return normalize();
  }
  long_int operator<<(int n){ return long_int(*this)<<=n; }

  // 比較
  bool operator>(long_int n){
    if (size()!=n.size()) return size()>n.size();
    for (int i=size()-1;i>=0;i--)
      if (d[i]!=n[i]) return d[i]>n[i];
    return false;
  }

  deque<int> d;
};

pair<long_int,long_int> ldiv(long_int a,long_int b)
{
  long_int ret;
  for (int i=a.size()-1;i>=0;i--){
    int n;
    for (n=1;n<10;n++)
      if (((b*n)<<i)>a) break;
    a-=(b*(n-1))<<i;
    ret.d.push_front(n-1);
  }
  return make_pair(ret.normalize(),a.normalize());
}

ostream &operator<<(ostream &os,const long_int &l)
{
  long_int n=l; // GCCのバグ?対策
  if (n.size()==0) os<<0;
```

```
  else for (int i=n.d.size()-1;i>=0;i--) os<<n.d[i];
  return os;
}
```

任意制度浮動少数(いらんかも)

・行列演算

以下を前提とする

```
typedef vector<vector<double> > matrix;
int height(const matrix m) { return m.size();    }
int width (const matrix m) { return m[0].size(); }
```

積

```
matrix operator*(const matrix &a,const matrix &b)
{
  matrix ret(height(a),vector<double>(width(b),0.0));
  assert(width(a)==height(b));
  int num = width(a);

  for(int y=0;y<height(a);y++)
    for(int x=0;x<width(b);x++)
      for(int i=0;i<num;i++)
        ret[y][x]+=a[y][i]*b[i][x];

  return ret;
}
```

連立方程式の解

```
// this method solves AX=B where
// A is an n,n matrix and B,X are n vectors.
// destroies A,B while computation.
// costs O(n^3) computation for 1 equation solving.
// simple code, but slow.

vector<double> gaussian_method(matrix a,vector<double> b)
{
  int n=b.size();

  for(int i=0;i<n;i++){
    int champ=i;double score=0;
    for(int cand=i;cand<n;cand++)
      if(abs(a[cand][i])>score)
        champ=cand,score=abs(a[cand][i]);

    swap(a[champ],a[i]);
    swap(b[champ],b[i]);

    for(int j=i+1;j<n;j++){
      a[j][i]/=a[i][i];
      for(int k=i+1;k<n;k++)
        a[j][k] -= a[i][k]*a[j][i];
      b[j] -= b[i]*a[j][i];
    }
  }

  for(int i=n-1;i>=0;i--){
    for(int j=i+1;j<n;j++)
      b[i] -= a[i][j]*b[j];
    b[i] /= a[i][i];
```

```
  }
  return b;
}
```

高速版・逆行列

```
//this method analyse the matrix A and returns sufficient
//set of information to solve AX=B.
//with these infomation, AX=B can be solved in O(n^2),
//not in O(n^3).
void make_solver(matrix a,vector<double>& solver,vector<int>& swapinfo)
{
  int n=a.size();
  solver.clear();
  swapinfo=vector<int>(n);
  for(int i=0;i<n;i++){
    int champ=i;double score=0;
    for(int cand=i;cand<n;cand++)
      if(abs(a[cand][i])>score)
        champ=cand,score=abs(a[cand][i]);
    swap(a[champ],a[i]);
    swapinfo[i]=champ;
    for(int j=i+1;j<n;j++){
      a[j][i]/=a[i][i];
      for(int k=i+1;k<n;k++)
       a[j][k] -= a[i][k]*a[j][i];
      solver.push_back(a[j][i]);
    }
  }
  for(int i=n-1;i>=0;i--){
    for(int j=i+1;j<n;j++)
      solver.push_back(a[i][j]);
    solver.push_back(a[i][i]);
  }
}

// with infomation created by makeSolver,
// solves AX=b in O(n^2).
vector<double> solve(const vector<double>& solver,const vector<int>&
swapinfo,vector<double> b)
{
  int n=b.size(),ptr=0;
  for(int i=0;i<n;i++){
    swap(b[swapinfo[i]],b[i]);
    for(int j=i+1;j<n;j++)
      b[j] -= b[i]*solver[ptr++];
  }
  for(int i=n-1;i>=0;i--){
    for(int j=i+1;j<n;j++)
      b[i] -= b[j]*solver[ptr++];
    b[i] /= solver[ptr++];
  }
  return b;
}

matrix inverse(const matrix &a)
{
  int n=a.size();
  matrix ret(n);
  vector<double> solver;vector<int> sinfo;
  make_solver(a,solver,sinfo);
  for(int i=0;i<n;i++){
    vector<double> e(n,0.0); e[i]=1;
```

```
    vector<double> x=solve(solver,sinfo,e);
    for(int j=0;j<n;j++)
      ret[j].push_back(x[j]);
  }
  return ret;
}
```

区間演算
ゼロでの掛け算に注意せよ。
例:(1,2) * [0,2] = [0,4) ((0,4) ではない)

・その他

部分配列の和の最大を求める

(1)O(nlogn)
　中央で分割。左右の解と、両側にまたがる最大値を計算。
　両側にまたがるので、境界の要素を含む。
　境界の要素を含む左右の部分列の最大値は O(n)で求まる。
　左右、またがるものの最大値が答え。

(2)O(n)
　i 番目まで操作したときの最大値(sofer)と、そこで終わる部分列の最大値(max_ending_here)を保持しておく。正当性は自明ではないが、コードは単純。

```
  int sofar=0,ending_here=0;
  for (int i=0;i<x.size();i++){
    ending_here=max(ending_here+x[i],0);
    sofar=max(sofar,ending_here);
  }
```

文字列検索

(1)ボイヤー・ムーア法
　計算量の上限 O(M+N) 、アルファベットの種類が十分に多ければ O(N/M)
　現時点で最速の文字列検索アルゴリズム。

```
  // a より p を探す。m=p.length, n=a.length
  int bmsearch(const char *p,int m,const char *a,int n)
  {
    static int skip[256];
    for (int i=0;i<256;i++) skip[i]=m;
    for (int i=0;i<m;i++) skip[p[i]]=m-i-1;

    int i,j;
    for (i=m-1,j=m-1;j>0;i--,j--)
      while(a[i]!=p[j]){
        int t=skip[a[i]];
        i+=(m-j>t)?m-j:t;
        if (i>=n) return -1;
        j=m-1;
      }
    return i;
  }
```

(2)ラビン・カープ法
　ほとんど確実に O(M+N) のオーダで計算できる。ハッシュ関数を用いる。
　キーのハッシュ値と同じハッシュ値を取る箇所を調べる。簡潔な割に高速。
　…でも 10 万文字から 1000 文字を検索、でタイムリミットになる程度の性能。

```
int rksearch(const string &p,const string &a)
{
  const long long q=33554393; // (d+1)*q がオーバーフローしないような大きい素数
  const long long d=256;

  long long dm=1,h1=0,h2=0;
  int m=p.length(),n=a.length();
  if (m>n) return -1;

  for (int i=1;i<m;i++) dm=(d*dm)%q;
  for (int i=0;i<m;i++){
    h1=(h1*d+p[i])%q;
    h2=(h2*d+a[i])%q;
  }
  for (int i=0;i<=n-m;i++){
    if (h1==h2 && equal(p.begin(),p.end(),a.begin()+i))
      return i;
    h2=(h2+d*q-a[i]*dm)%q;
    h2=(h2*d+a[(i+m)%n])%q;
  }
  return -1;
}
```

再帰的降下型パーザー
- 覚えてるから良い?

・数学公式

整数論

・ $p^k q^m r^n$ の約数の個数は $(k+1)(m+1)(n+1)$

・ $p^k q^m r^n$ の約数の総和は $\frac{1-p^{k+1}}{1-p}\frac{1-q^{m+1}}{1-q}\frac{1-r^{n+1}}{1-r}$

・ $x=p^k q^m r^n$ とすると、[1..x]の範囲で、x と互いに素な自然数の個数は $x(1-\frac{1}{p})(1-\frac{1}{q})(1-\frac{1}{r})$

・ $n!$ に含まれる素因数 p の個数は $[\frac{n!}{p}]+[\frac{n!}{p^2}]+\cdots$

・ $p=x^4+1$ をみたす素数 p は 5 しかない
　$p=(x^2+2x+2)(x^2-2x+2)$ なる素因数分解のすくなくとも一方は 1 だからである。

・ウィルソンの定理
　p が素数のとき、$(p-1)!\equiv -1(mod\ p)$

・フェルマーの定理

p が素数、a は p の倍数でないとき、 $a^{p-1}\equiv 1 \pmod{p}$

代数学

・3次方程式の解

$ax^3+bx^2+cx+d=0$ の解は

$x=X-\frac{b}{3}a$ where

$X=A-\frac{p}{3aA}$

$p=\frac{-b^2}{3a}+c$

$q=\frac{2b^3}{27a^2}-\frac{bc}{3a}+d$

$A^3=B$

$aB^2+qB-\frac{p^3}{27a^2}=0$

B が 2 通り出るが、同じ **X** をもたらす。

三角法

・基本公式 $\cos t=real(\exp it)$ = real(polar(1,t))

$\sin t=imag(\exp it)$ = imag(polar(1,t))

$\sin t=\cos(t-\frac{\pi}{2})$

・余弦定理 $a^2=b^2+c^2-2bc\cos(A)$

$\cos A=\frac{b^2+c^2-a^2}{2bc}$

$\cos\frac{A}{2}=\sqrt{\frac{s(s-a)}{bc}}$

・正弦定理 $\frac{a}{\sin A}=\frac{b}{\sin B}=\frac{c}{\sin C}=2R$

$\sin A=\frac{2}{bc}\sqrt{s(s-a)(s-b)(s-c)}$

$\sin\frac{A}{2}=\sqrt{\frac{(s-b)(s-c)}{bc}}$

・正接定理 $\tan\frac{A}{2}=\sqrt{\frac{(s-b)(s-c)}{s(s-a)}}$

・三角形の面積と接円の半径 $S=\frac{abc}{4R}=\frac{2r}{a+b+c}=\sqrt{s(s-a)(s-b)(s-c)}$

R,r は外接、内接円の半径。S は三角形の面積。

s は三角形の半周長

・加法定理

$$\sin(a+b)=\sin a\cos b+\cos a\sin b$$
$$\cos(a+b)=\cos a\cos b-\sin a\sin b$$
$$\tan(a+b)=\frac{\tan a+\tan b}{1+\tan a\tan b}$$

・倍角、半角

$$\sin 2t=2\sin t\cos t$$
$$\cos 2t=\cos^2 t-\sin^2 t=2\cos^2 t-1=1-2\sin^2 t$$
$$\tan 2t=\frac{2\tan t}{1-\tan^2 t}$$
$$\sin 3t=3\sin t-4\sin^3 t$$
$$\cos 3t=4\cos^3 t-3\cos t$$

・和積の公式

$$\sin a\cos b=\frac{1}{2}(\sin(a+b)+\sin(a-b))$$
$$\cos a\cos b=\frac{1}{2}(\cos(a+b)+\cos(a-b))$$
$$\sin a\sin b=\frac{1}{2}(\sin(a+b)+\sin(a-b))$$

$$\sin a+\sin b=2\sin\frac{a+b}{2}\cos\frac{a-b}{2}$$
$$\sin a-\sin b=2\cos\frac{a+b}{2}\sin\frac{a-b}{2}$$
$$\cos a+\cos b=2\cos\frac{a+b}{2}\cos\frac{a-b}{2}$$
$$\cos a-\cos b=-2\sin\frac{a+b}{2}\sin\frac{a+b}{2}$$

積分法

・不定積分

$$\int \sin x\,dx=-\cos x$$
$$\int \cos x\,dx=\sin x$$
$$\int \tan x\,dx=-\log(\cos x)$$
$$\int \cot x\,dx=\int\frac{dx}{\tan x}=\log(\sin x)$$
$$\int sec\,x\,dx=\int\frac{dx}{\cos x}=\log\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right|$$
$$\int cosec\,x\,dx=\int\frac{dx}{\sin x}=\log\left|\tan\left(\frac{x}{2}\right)\right|$$
$$\int sec^2 x\,dx=\int\frac{dx}{\cos^2 x}=\tan x$$
$$\int cosec^2 x\,dx=\int\frac{dx}{\sin^2 x}=-\frac{1}{\tan x}$$

級数の和

・ $\sum_{k=0}^{n} a+kd=\frac{n+1}{2}(2a+nd)$

・ $\sum_{k=0}^{n} ar^k=\frac{a(r^{n+1}-1)}{r-1} \quad (r\neq 1)$

・ $\sum_{k=0}^{n} (a+kd)r^k=\frac{a-(a+nd)r^{n+1}}{1-r}+\frac{dr(1-r^n)}{(1-r)^2}$

・ $\sum_{k=0}^{n} \binom{n}{k} a^{n-k}b^k=(a+b)^n$

立体図形

・直線 $(\boldsymbol{a},\boldsymbol{u})$ は、点 $\boldsymbol{a}$ を通りベクトル $\boldsymbol{u}$ の方向の直線。
・平面 $(\boldsymbol{a},\boldsymbol{n})$ は、点 $\boldsymbol{a}$ を通り法線 $\boldsymbol{n}$ の平面。
・接線や法線は規格化されていなくても良い。
・ドット・は内積、クロス×は外積。
・割り算とか平方根ができないときは正当な原因で解が存在しない場合である。
・点 $\boldsymbol{x}$ と点 $\boldsymbol{y}$ の距離
$|\boldsymbol{y}-\boldsymbol{x}|=\sqrt{(\boldsymbol{y}-\boldsymbol{x})\cdot(\boldsymbol{y}-\boldsymbol{x})}$

・直線 $(\boldsymbol{a},\boldsymbol{u})$ 上で、点 $\boldsymbol{b}$ にもっとも近い点は $\boldsymbol{a}+s\boldsymbol{u}$ で、その距離はｈである。
$s=\frac{\boldsymbol{u}\cdot(\boldsymbol{b}-\boldsymbol{a})}{\boldsymbol{u}\cdot\boldsymbol{u}} \qquad h=\frac{|\boldsymbol{u}\times\boldsymbol{b}|}{|\boldsymbol{u}|}$

・平面 $(\boldsymbol{a},\boldsymbol{n})$ 上で、点 $\boldsymbol{b}$ に最も近い点は $\boldsymbol{b}-s\boldsymbol{n}$ で、その距離はｈである。
$s=\frac{\boldsymbol{n}\cdot(\boldsymbol{b}-\boldsymbol{a})}{\boldsymbol{n}\cdot\boldsymbol{n}} \qquad h=|s\boldsymbol{n}|$

・直線 $(\boldsymbol{a},\boldsymbol{u})$ と直線 $(\boldsymbol{b},\boldsymbol{v})$ の最短距離はｈで、その両端の点は $\boldsymbol{a}+s\boldsymbol{u}$ および $\boldsymbol{b}+t\boldsymbol{v}$ である。
$s=\frac{(\boldsymbol{b}-\boldsymbol{a})\cdot\boldsymbol{n}_b}{\boldsymbol{u}\cdot\boldsymbol{n}_b} \qquad t=\frac{(\boldsymbol{a}-\boldsymbol{b})\cdot\boldsymbol{n}_a}{\boldsymbol{v}\cdot\boldsymbol{n}_a}$
$\boldsymbol{n}_a=\boldsymbol{u}\times\boldsymbol{w} \qquad \boldsymbol{n}_b=\boldsymbol{v}\times\boldsymbol{w} \qquad \boldsymbol{w}=\boldsymbol{u}\times\boldsymbol{v}$
$\boldsymbol{h}=\frac{(\boldsymbol{b}-\boldsymbol{a})\cdot\boldsymbol{w}}{\boldsymbol{w}\cdot\boldsymbol{w}}\boldsymbol{w}=(\boldsymbol{b}+t\boldsymbol{v})-(\boldsymbol{a}+s\boldsymbol{u})$
$h=|\boldsymbol{h}|$

・直線 $(\boldsymbol{a},\boldsymbol{u})$ と、平面 $(\boldsymbol{b},\boldsymbol{n})$ は点 $\boldsymbol{a}+s\boldsymbol{u}$ で交わる。
$s=\frac{(\boldsymbol{b}-\boldsymbol{a})\cdot\boldsymbol{n}}{\boldsymbol{u}\cdot\boldsymbol{n}}$

・平面 $(\boldsymbol{a},\boldsymbol{u})$ と平面 $(\boldsymbol{b},\boldsymbol{v})$ の交線は直線 $(\boldsymbol{c},\boldsymbol{w})$ である。
$\boldsymbol{w}=\boldsymbol{u}\times\boldsymbol{v}$
$\boldsymbol{c}$ は適当に求めれば・・・

・直線 $(\boldsymbol{a},\boldsymbol{u})$ と球 $(\boldsymbol{b},r)$ の交点は $\boldsymbol{a}+s\boldsymbol{u}$ である。

$$t=\frac{\boldsymbol{u}\cdot(\boldsymbol{b}-\boldsymbol{a})}{\boldsymbol{u}\cdot\boldsymbol{u}} \qquad h=\frac{|\boldsymbol{u}\times\boldsymbol{b}|}{|\boldsymbol{u}|} \qquad s=t\pm\sqrt{r^2-h^2}$$

・平面 $(\boldsymbol{a},\boldsymbol{n})$ と球 $(\boldsymbol{b},r)$ の共通部分は、 $\boldsymbol{n}$ を法線とする平面にあって、 $\boldsymbol{a}+t\boldsymbol{u}$ を中心とし、半径が $\sqrt{r^2-h^2}$ の円である。

$$t=\frac{\boldsymbol{u}\cdot(\boldsymbol{b}-\boldsymbol{a})}{\boldsymbol{u}\cdot\boldsymbol{u}} \qquad h=\frac{|\boldsymbol{u}\times\boldsymbol{b}|}{|\boldsymbol{u}|}$$

$$\boldsymbol{u}=\boldsymbol{n}\times(\boldsymbol{n}\times(\boldsymbol{b}-\boldsymbol{a}))$$

・実際の例

最大長方形の面積
　0 1 の二次元配列が与えられる。1 のみを含む最大の長方形は?
　ナイーブには 上左端、下右端、その領域をスキャンして O(n^6)。
　うまくすると O(n^2)のやり方がある。(良く分からん)
　サイズが 5*5 までであることを利用すると全長方形パターンが 225 個しかないことを利用して
　あらかじめ全パターンを整数として生成して、全パターンとスキャンすればいける。

複数の文字を含む最小の区間
　100 万文字の列より、与えられた複数の文字を含む最短の部分列を求める。
　O(n)を要求。キーの最近の出現位置を考える。これで O(kn)。
　双方向リストを用い、出現位置の更新を O(1)で行えば O(n)になる。
　別解、尺取虫戦略。すべてのキーの出現回数が 1 以上になるまで先頭を右に。
　どれかのキーの出現回数が 0 になるまで終端を右に。その時点での区間が解の候補。

円周上の点による最大面積多角形
　半径 1 の円周上に n 個の円。それらを用いて出来る面積最大の多角形。
　I 番目までに j 個の点を使ったときの最大値を考えると DP。

正方形の破壊
　格子状に置かれたマッチ棒からいくつかを取り除いて正方形が一つも無いようにする。
　何本取り除く必要があるか?NP 困難。枝刈りをがんばる。
　マッチを取り除いたときに破壊できる正方形の集合を考える。包含関係を利用して枝刈りする。

点を包含する球
　三次元空間上の点の集合をすべて含む最小の球を求める。
　(1)代数的に。4 点の外接球 or 3 点の外接円を大円とする球 or 二点を直径とする球のどれか
　が解。(2)ヒューリスティック的に。中心を最も遠い点に適当な距離移動させる。

六角形の組み合わせ
　ヘクス上の二つの軌跡が同一か？

$$T_1=\begin{pmatrix}2 & 0\\ 0 & \frac{2\sqrt{3}}{3}\end{pmatrix}, T_2=\begin{pmatrix}1 & \frac{\sqrt{3}}{3}\\ 0 & \frac{2\sqrt{3}}{3}\end{pmatrix}, R_1=\begin{pmatrix}\frac{1}{2} & \frac{-3}{2}\\ \frac{1}{2} & \frac{1}{2}\end{pmatrix}, R_2=\begin{pmatrix}1 & -1\\ 1 & 0\end{pmatrix}$$

　上のは T1,T2 は直行行列から整数座標への変換行列、
　R1,R2 はそれぞれの座標系での 60 度回転。

倉庫番

この問題では荷物が一つのものを扱う。一般の倉庫番は PSPACE 完全問題。
盤面サイズが小さけりゃ幅優先で解ける。あるいはマンハッタン距離による A*で解ける。

三角形の分割
直角二等辺三角形上にいくつか点があるとき、三角形上に点をとり、そこから各三角形の頂点への辺を引いたとき、その三直線により分割される領域に等しい数の点が入るようにしたい。
山登り法で。三領域の点の数を測定し、その偏り具合により微妙に中心の位置をずらす。
(解けないかもしれない)
まともな方法はややこしい。

倍数の和による整数の表現
整数 n,a,b(<=100 万)に対して、1～n の中で ai+bj(i,j>=0)の形に表せないものの個数を求めよ。
ai+bj は gcd(a,b)の倍数になる。それ以外のものははじめから除外(しても良いがしなくても良い)。
Hamming の問題を利用する。(例： $2^i \times 3^j \times 5^k$ の形の整数を小さいほうから 700 個生成せよ)
概念的には無限列のマージ。C でうまく書く方法がある。(set を使うと相当楽だが)
Hamming の問題のべき乗を乗算に、乗算を加算にすれば同じ問題である。
エラトステネスのふるいと同じような解き方でも解ける。さらに、マークを工夫すると同じ箇所に二回以上マークしないような方法ができ、二回以上無いのであればマークする必要すらなくなる。何箇所マークするかを計算し、足し合わせればよい。そのときの最悪計算量は O(n^1/2)。

どこで会える？
路線データが与えられて、二地点からスタートする二人が一定時間以上合ってからまた帰るのに必要な最小金額を求める。時間と駅で DP。むずい。

大小の贈り物
多角形を包含する最小・最大の長方形を求める。多角形の回転のみで大きさはきまる。
素朴な解法としては多角形を微小角度づつ回転させて値を調べる。任意の問題に対して十分な精度をもつ delta の値がキモ。
上下左右が入れ替わる部分に着目すると O(n^2)で解ける(ややややこしい)。
あらかじめ凸包を作ってから解くと大きな問題でも安心。

ユークリッドの書斎
部屋のマップが与えられ、部屋の中の二点間をケーブルでつなぐ最短距離を求める。
可能な辺を列挙してダイクストラ。辺を作るのがやや面倒か？

嘘つき島の問題
正直者 p1 人、嘘つき p2 人の島の人がいる。x 番目の人に y 番目の人は正直か？という質問をいくつかしたときに各個人が正直者か嘘つきか判定する。
グループ併合の問題。({1},{}) ～ ({n},{}) のグループ対の集合より始めて併合を繰り返す。
再帰的なアルゴリズムでグループわけを行うことも出来る。
グルーピングが終われば後は DP で解ける。

丸い紙ふぶき
さまざまな大きさの円形の紙が床に散らばっているときにいくつの紙が上から見えているか。
限られた平面を調べればよいのだが、水平分割を使うか。交点と上下をソートし、その中間点を調べれば OK。円の交点に高々 3 個の円が見えていることを利用しても可。
平面を分割して考えた場合、この問題は円であるためうまいこと計算量がカットできないようだ。

代数式を比較する
与えられた二つの代数式が同じか？パーズ+正規化して比較。

円の集まりをロープで囲む
円の集まりをロープで囲む。
凸包からの類推。ぐるっと囲えば出来る。任意の接点を列挙してそれらの凸包をとり、同一円な

ら弧長を加算…な方法でも可能。

多角形の面積の近似
座標平面に与えられた多角形が少しでも入っている正方形の個数を求める。
X 座標ごとにスライスして交線を考えると解ける。

無理数を近似する分数
整数 n,素数 p に対して $\sqrt{p}<\frac{u}{v}$ (u,v<=n)を満たす最小のものと $\frac{u}{v}<\sqrt{p}$ を満たす最大のものを求めよ。素朴な解、分母 1～n まですべてについて $\sqrt{p}\times i$ を分子とするもので max,min をとる。スターンブロコット木を使うとうまく解ける。

Silly Sort
二つの数を入れ替えることによりソートを行うとき、コストを移動させた要素の和としてコストの最小を求める。各数字が本来あるべき場所にある数へエッジを張ることによりリングがいくつか出来る。各リングを正しい整列にするのに必要なコストはリング中の数の和+(転送回数-2)*リング中の数の最小値、あるいは、リングの外から全体の最少数を借りてきてそれをたらいまわしにしてソートした場合の値、これらの小さいほうである。すべてのリングに対してこの値を求め、それらの和がコストの最小値になる。証明は難しい。いけそうならいけ、という場合もあるということか。

ケーブルマスタ
いくつかの長さの異なるケーブルからある等しい長さのケーブルを n 本とりたい。取れるケーブルの最長値を求めよ。
ドント方式で解ける。DP でも解ける。s 番目までを使って p 本作るときの最大値は、s-1 番目までを使って i 本つくり、s 番目から(p-i)本切り出す(0<=i<=p) …の最大値。

点の密集区域
平面上の n 個の点を最もたくさん囲う半径 1 の円を求めよ。
任意の二点をそれぞれ中心とする半径 1 の円同士の交点すべてを調べれば OK。

・実行環境チェック項目

long long ちゃんと 64 ビット?
abort()時のメッセージ
タイムリミット計測
メモリリミット計測
PE 出してくれるか?
ゼロ割エラーの扱い (どんなエラー返ってくる?)
実体渡しとかにコンパイラのバグが無いか?
スタックあふれたらどうなる?
(Java)長整数とか、いろいろ使えるか?