D_hokudai ICPC Library

Yuma Inoue, Hokkaido University

目次

1　基礎

1.1　チェックリスト

- 誤読、勘違いはないか？
- 計算量の見積もりは正しいか？
- 初期化忘れはないか？
- オーバフローしていないか？
- INF,EPS の値は適切か？
- 配列のサイズは適切か？
- 入力は正しく受け取れているか？
- 出力形式に準拠しているか？
- 最小・最大ケースは大丈夫か？
- n=0、負値のケースはないか？ 正しく処理できるか？
- 解が存在しない場合はあるか？ 例外処理が必要か？
- 変数名に重複はないか？
- コピペ後の修正忘れはないか？
- ライブラリの写し間違いはないか？

1.2　解法のヒント

- グラフを考える、同一視できる状態がないか考える
- ソートしてみる、逆順に考える
- 貪欲な手法を考える
- 無駄はないか？
- 楽でバグりづらい実装はないか？
- 「最小値の最大値」→二分探索
- 割当問題→フロー、重み付き割当問題→最小費用流

1.3　vim 設定ファイル

```
syntax on
set expandtab
set tabstop=4
set shiftwidth=4
set softtabstop=4
set number
set ruler
set autoindent
set smartindent
set cindent
"改行後に「」キーを押すと上行末尾に戻る Backspace"
"set backspace=1
set whichwrap=b,s,h,l,<,>,[,]
let loaded_matchparen = 1

inoremap <C-h> <Left>
inoremap <C-j> <Down>
inoremap <C-k> <Up>
inoremap <C-l> <Right>

inoremap { {}<LEFT>
inoremap [ []<LEFT>
inoremap ( ()<LEFT>
inoremap " ""<LEFT>
inoremap ' ''<LEFT>

cnoremap com :call CompileThisFile()

function! CompileThisFile()
  ! g++ -o "%:r" "%"
endfunction
```

1.4 テンプレート

```cpp
#include<algorithm>
#include<cassert>
#include<cctype>
#include<climits>
#include<cmath>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<iostream>
#include<iomanip>
#include<map>
#include<numeric>
#include<queue>
#include<vector>
#include<set>
#include<string>
#include<stack>
#include<sstream>
#include<complex>

#define pb push_back
#define clr clear()
#define sz size()
#define fs first
#define sc second

#define rep(i,a) for(int i=0;i<(int)(a);i++)
#define rrep(i,a) for(int i=(int)(a)-1;i>=0;i--)
#define all(a) (a).begin(),(a).end()
#define EQ(a,b) (abs((a)-(b)) < EPS)
#define INIT(a) memset(a,0,sizeof(a))

using namespace std;
typedef double D;
typedef pair<int,int> P;
typedef long long ll;
typedef vector<int> vi;
typedef vector<string> vs;

const D EPS = 1e-8;
const int INF = 1e8;
const D PI = acos(-1);
```

1.5 STL

1.5.1 algorithm

```cpp
sort( v.rbegin(), v.rend() );

max_element(v.begin(), v.end()); //最大要素のイテレータを返す
min_element(v.begin(), v.end()); //最小要素のイテレータを返す

lower_bound(v.begin(), v.end(), 10); //指定した値"以上"の要素が最初に現れる位置を返す
upper_bound(v.begin(), v.end(), 10); //指定した値より"大きい"の要素が最初に現れる位置を返す
```

1.5.2 map

```cpp
/* --- 宣言--- */
map<string, int> m1;
map<string, int>::iterator ite;

/* --- 操作--- */
m1.erase( "key");
m1["key"] = 100;
ite = m1.find( "key" ); //キーの位置を、存在しなければ end() を返す

for (ite = m1.begin(); ite != m1.end(); ite++){
    cout << ite->first << endl; // キーを出力
    cout << ite->second << endl; // 値を出力
}
```

1.5.3 set

```
/* --- 宣言--- */
set<int> s; //デフォルトでは照準ソート ( 小さい順)
set<int, greater<int> > s1; //降順ソート ( 大きい順)

/* --- 操作--- */
s.insert( 10 );
s.erase( 10 );

set<int>::iterator p = s.find(5);
if( nums.find( 10 ) == nums.end() ) cout << "Not Found" << endl;
```

1.5.4 pair

```
/* --- 宣言--- */
pair<string, string> sspair;

/* --- 操作--- */
sspair.first = "key";
sspair.second = "value";
cout << sspair.first << " " << sspair.second << endl;

/* --- よくやる--- */
vector< pair<string, int> > persons;
persons.push_back( make_pair("ishida", 1991) );
cout << persons[0].first << " " << persons[0].second << endl;
```

1.5.5 vector

```
/* --- 宣言--- */
vector<int> v1;
vector<int> v2(10);
vector<int> v3 = v1;
vector<int> v4(v1);

/* --- 操作--- */
v1.front(); //先頭の要素を取得
v1.back(); //末尾の要素を取得
v1.push_back(10); //末尾に要素を追加
v1.pop_back(); //末尾に要素を追加

v1.size(); //要素数を取得
v1.empty(); //bool 型で空なら true 、その他は false を返す
v1.clear(); //空のコンテナに初期化
v1.resize(n); //要素数を n にリサイズ
v1.resize(n, 0); //要素数を n にリサイズして 0 で初期化

vector<int>::iterator p;
advance( p, 5 ); //イテレータ p を 5 つ先に進める
iter_swap( it1, it2 ); // 2 つのイテレータが参照する値を交換する
for(p=vec.begin(); p<vec.end(); p++) cout << *p << endl;
```

1.5.6 queue

```
/* --- 宣言--- */
queue<int> q1;
queue<int> q2 = q1;
queue<int> q3(q1);

/* --- 操作--- */
q1.front(); //先頭の要素を取得
q1.back(); //末尾の要素を取得
q1.push(10); //先頭に要素を追加
q1.pop(); //先頭の要素を削除
q1.size(); //要素数を取得
q1.empty(); //bool 型で空なら true 、その他は false を返す
```

1.5.7 stack

```
/* ––– 宣言––– */
stack<int> st1;
stack<int> st2 = st1;
stack<int> st3(st1);

/* ––– 操作––– */
st1.top(); //先頭の要素を取得
st1.push(10); //先頭に要素を追加
st1.pop(); //先頭の要素を削除
st1.size(); //要素数を取得
st1.empty(); //bool 型で空なら true 、その他は false を返す
```

1.5.8 deque

```
/* ––– 宣言––– */
deque<int> deq1;
deque<int> deq2(10);
deque<int> deq3 = deq1;
deque<int> deq4(deq1);

/* ––– 操作––– */
deq1.front(); //先頭の要素を取得
deq1.back(); //末尾の要素を取得
deq1[3]; //3 番目の要素を取得
deq1.push_front(10); //先頭に要素を追加
deq1.pop_front(); //先頭の要素を削除
deq1.push_back(10); //末尾に要素を追加
deq1.pop_back(); //末尾に要素を削除
deq1.size(); //要素数を取得
deq1.empty(); //bool 型で空なら true 、その他は false を返す
deq1.clear(); //空のコンテナに初期化
```

1.5.9 priority_queue

```
/* ––– 宣言––– */
priority_queue<int> pq; //デフォルトでは大きい順にソート
priority_queue<int, vector<int>, greater<int> > pq1; //小さい順にソート

/* ––– 操作––– */
pq.top(); //先頭の要素を取得
pq.pop(); //先頭に要素を削除
pq.push( 10 ); //要素を追加
deq1.size(); //要素数を取得
deq1.empty(); //bool 型で空なら true 、その他は false を返す
deq1.clear(); //空のコンテナに初期化
```

1.5.10 string

```
/* ––– 宣言––– */
string s1;
string s2(s1);
string s3( "abc" );
string s4 = "abc";

/* ––– 操作––– */
s1.length(); //文字列の長さを取得
s1.size(); //文字列の長さを取得 ( 同じ )
s1.clear();

s3 = s1 + s2; //文字列の連結
s3 += s1; //末尾に文字列の追加

s4.find("cd"); //合致する文字列の最初のインデックスを返す 無い時は string::npos を返す

string str( "abcdefghijk" );
```

```
str.substr( 5 ); // "fghijk" 部分文字列
str.substr( 5, 3 ); // "fgh" 文字列のスライス

string::iterator it = s4.begin();
for (it = s4.begin(); it < s4.end(); it++ ) cout << *it << endl;
```

1.5.11 型変換

```
//----------- INT => CHAR -----------//
/* 8 ビット int 型と考え、'a' は 97 の別表記として考える*/
int i = 3;
printf("%c", '0' + i); // => 3

//----------- CHAR => INT -----------//
printf("%d", '9' - '0'); // => 9

//----------- C_IO -----------//
char c;
while( (c = getchar() ) != EOF && c != '#' ) { cout << c; }
```

2 グラフ

2.1 グラフの用語と性質

用語

- マッチング：グラフ G の辺の部分集合 M で、互いに端点を共有しない。
- 辺カバー：グラフ G の辺の部分集合 F で、G に含まれるどの頂点も、少なくとも 1 つの F の辺の端点である。
- 安定集合：グラフ G の頂点の部分集合 D で、D のどの 2 点も G 上で隣接していない。独立集合とも。
- 点カバー：グラフ G の頂点の部分集合 S で、G に含まれるどの辺も、少なくとも 1 つの S の頂点に接続している。

性質

1. 孤立点のないグラフに対し、| 最大マッチング | + | 最小辺カバー | = V
2. | 最大安定集合 | + | 最小点カバー | = $|V|$
3. 二部グラフならば、| 最大マッチング | = | 最小点カバー |
4. 強連結成分をつぶすと DAG になる

最大流の変形

1. ソース、シンクが複数の場合: スーパーソース・スーパーシンクを別に 1 つずつ作ればよい。ただし、特定のソースから特定のシンクに流さなければいけない問題は多品種フローとなり難しい。
2. 無向グラフの場合: 通常のグラフのように双方向辺を張ればよい。
3. 頂点にも容量がある場合: 頂点を流入・流出の 2 点に分解し、その間に頂点容量分の辺を張る。
4. 流量に下限制約がある場合: 辺 $e = (u, v)$ に対し、上限容量 $c(e)$、下限流量 $b(e)$ が設定されているとする。このとき、スーパーソース S、スーパーシンク T を用意し、$u \to T$ と $S \to v$ に容量 $b(e)$ の辺、$u \to v$ に容量 $c(e) - b(e)$ の辺、$S \to s$ と $t \to T$ に容量 INF の辺を張ったグラフ G' を作ると、G' の最大流 F' は元のグラフ G の最大量 F に対し、$F = F' - \sum b(e)$ となる。ただし、G が下限流量制約を満たせるとは限らない場合、$S \to s$、$t \to T$ の辺を張る前に $t \to s$ に容量 INF の辺を張ってフローを流し、$\sum b(e)$ が流れることを確認する必要がある。

2.2 グラフの構成

隣接リスト表現用の edge 型の定義。

```
//辺の定義。必要に応じて削る。
struct edge{
  int from,to,cost,cap,rev;
  bool operator<(const edge x)const{return cost<x.cost;}
};

vector<edge> G[V]; //グラフの隣接リスト
//辺の追加。2つ目の辺の追加はフローアルゴリズムの残余グラフ用。
void AddEdge(int s,int g,int c,int p){
  G[s].pb((edge){s,g,c,p,G[g].sz});
  G[g].pb((edge){g,s,-c,0,G[s].sz-1}); //for Max-Flow
}
```

2.3 最短路

頂点 s から t への最短パスの長さを求める問題。

2.3.1 ベルマンフォード

単一始点最短路問題を解く。$O(VE)$。更新回数が V 回以上かどうかで負の閉路を検出可。

```
int v; //頂点数
int d[V]; //s から i への最短路
vector<edge> G[V]; //グラフの隣接リスト表現

void BellmanFord(int s){
  fill(d,d+v,INF);
  d[s] = 0;
  for(;;){
    bool f = false;
    rep(i,v)rep(j,G[i].sz){
      edge e = G[i][j];
      if(d[i]<INF && d[e.to] > d[i] + e.cost){
        d[e.to] = d[i] + e.cost;
        f = true;
      }
    }
    if(!f)break;
  }
}
```

2.3.2 ダイクストラ

単一始点最短路問題を解く。負の辺があると使えない。$O(E \log V)$。$O(V^2)$ 実装も可。

```
int v; //頂点数
int d[V]; //s から i への最短路
vector<edge> G[V]; //グラフの隣接リスト表現

void dijkstra(int s){
  fill(d,d+v,INF);
  d[s] = 0;
  priority_queue<P ,vector<P> ,greater<P> > q;
  q.push(P(0,s));

  while(q.sz){
    P p = q.top();q.pop();
    int u = p.second;
    if(d[u] < p.first)continue;
    rep(i,G[u].sz){
      edge e = G[u][i];
      if(d[e.to] > d[u] + e.cost){
        d[e.to] = d[u] + e.cost;
        q.push(P(d[e.to],e.to));
      }
    }
  }
}
```

2.3.3 ワーシャルフロイド

全頂点間最短路問題を解く。$O(V^3)$。負の閉路があるとき、d[i][i]==0 となる i が存在する。

```
int v; //頂点数
int d[V][V]; //グラフの隣接行列
void WarshallFloyd(void){ rep(k,v)rep(i,v)rep(j,v)d[i][j] = min(d[i][j],d[i][k]+d[k][j]); }
```

2.4 無向最小全域木

辺の重みの和が最小となる連結グラフを求める問題。答えは全域木になる。

2.4.1 クラスカル

```
//Union-Find を書いておく。

int Kruskal(vector<edge> &G){
  init(v);
  priority_queue<edge> q;
  rep(i,v)rep(j,G[i].sz)q.push(G[i][j]);

  int res = 0;
  while(q.size()){
    edge e = q.top(); q.pop();
    if(!same(e.from,e.to)){
      res += e.cost;
      unite(e.from,e.to);
    }
  }
  return res;
}
```

2.5 最大流

ソース s からシンク t へ流せる最大の流量を求める問題。

2.5.1 Dinic

最大流を $O(EV^2)$ で解く。実用上、計算量見積よりも非常に高速であることが多い。

```
int v; //頂点数
vector<edge> G[V]; //グラフの隣接リスト表現
int level[V]; //までの距離 s
int iter[V]; //どこまで調べ終わったか

void bfs(int s){
  memset(level,-1,sizeof(level));
  level[s] = 0;
  queue<int> q; q.push(s);
  while(q.sz){
    int u = q.front(); q.pop();
    rep(i,G[u].sz){
      edge &e = G[u][i];
      if(e.cap > 0 && level[e.to] < 0){
        level[e.to] = level[u] + 1;
        q.push(e.to);
      }
    }
  }
}

int dfs(int u, int t, int f){
  if(u==t)return f;
  for(int &i = iter[u];i<G[u].sz;i++){
    edge &e = G[u][i];
    if(e.cap > 0 && level[u] < level[e.to]){
      int d = dfs(e.to,t,min(f,e.cap));
      if(d > 0){
        e.cap -= d;
        G[e.to][e.rev].cap += d;
```

```
            return d;
        }
    }
  }
  return 0;
}

int dinic(int s, int t){
  int res = 0;
  for(;;){
    bfs(s);
    if(level[t]<0)return res;
    INIT(iter);
    int f;
    while((f=dfs(s,t,INF))>0)res += f;
  }
}
```

2.6 最小費用流

ソース s からシンク t へ一定の流量を流す時、かかるコストを最小化する問題。

2.6.1 Primal-Dual

最小費用流を求めるアルゴリズム。$O(FVE)$。F は流量の最大値。

```
int v; //グラフの頂点数
vector<edge> G[V]; //グラフの隣接リスト表現
int d[V];
int pv[V],pe[V]; //直前の頂点と辺

int primal_dual(int s,int t, int f){
  int res = 0;
  while(f>0){
    fill(d,d+v,INF);
    d[s] = 0;
    bool update = true;
    while(update){
      update = false;
      rep(u,v){
        if(d[u]==INF)continue;
        rep(i,G[u].sz){
          edge &e = G[u][i];
          if(e.cap > 0 && d[e.to] > d[u] + e.cost){
            d[e.to] = d[u] + e.cost;
            pv[e.to] = u; pe[e.to] = i;
            update = true;
          }
        }
      }
    }

    if(d[t] ==INF)return -1;

    int x = f;
    for(int u=t;u!=s;u=pv[u]){
      x = min(x,G[pv[u]][pe[u]].cap);
    }
    f -= x;
    res += x*d[t];
    for(int u=t;u!=s;u=pv[u]){
      edge &e = G[pv[u]][pe[u]];
      e.cap -= x;
      G[u][e.rev].cap += x;
    }
  }
  return res;
}
```

2.6.2 ポテンシャル法

ポテンシャル法により最短路検索にダイクストラを用いている。$O(FE\log V)$。初期から負の辺があると使えない？

```
int v; //グラフの頂点数
vector<edge> G[V]; //グラフの隣接リスト表現
int d[V];
int h[V],pv[V],pe[V];

int MinCostFlow(int s,int t,int f){
  int res = 0;
  fill(h,h+v,0);
  while(f>0){
    priority_queue<P ,vector<P> ,greater<P> > q;
    fill(d,d+v,INF);
    d[s] = 0; q.push(P(0,s));
    while(q.sz){
      P p = q.top();q.pop();
      int u = p.second;
      if(d[u] > p.first)continue;
      rep(i,G[u].sz){
        edge &e = G[u][i];
        if(e.cap>0 && d[e.to] > d[u] + e.cost + h[u] - h[e.to]){
          d[e.to] = d[u] + e.cost + h[u] - h[e.to];
          pv[e.to] = u; pe[e.to] = i;
          q.push(P(d[e.to],e.to));
        }
      }
    }
    if(d[t]==INF)return -1;
    rep(u,v)h[u] += d[u];

    int x = f;
    for(int u=t;u!=s;u=pv[u])x = min(x,G[pv[u]][pe[u]].cap);
    f -= x;
    res += x*h[t];
    for(int u=t;u!=s;u=pv[u]){
      edge &e = G[pv[u]][pe[u]];
      e.cap -= x; G[u][e.rev].cap += x;
    }
  }
  return res;
}
```

2.7 二部グラフのマッチング

最大流問題に帰着できる。O(VE)。上記最大流アルゴリズムを使ってもよいが、以下のように簡易実装もできる。

```
int v; //グラフの頂点数
vector<int> G[V]; //グラフの隣接リスト表現、重みがないので行き先だけで覚える int
int match[V]; //マッチングのペア
bool use[V];

bool dfs(int u){
  use[u] = true;
  rep(i,G[u].sz){
    int t = G[u][i], w = match[t];
    if(w<0 || (!use[w] && dfs(w))){
      match[u] = t;
      match[t] = u;
      return true;
    }
  }
  return false;
}

int biparticle_matching(){
  int res = 0;
  memset(match,-1,sizeof(match));
  rep(u,v){
    if(match[u]<0){
      INIT(use);
      if(dfs(u))res++;
    }
  }
  return res;
}
```

2.8 強連結成分分解

グラフ G の部分集合 S において、任意の 2 点 $u, v \in S$ の間に常にパスが存在する時、S は強連結であるといい、任意の $a \in G \setminus S$ について $\{a\} \cup S$ が強連結でないとき、S は G の強連結成分であるという。2 回の DFS を利用して強連結成分に分解する。$O(V+E)$。

```
vi g[V], rg[V]; //グラフとその逆辺グラフ。
vi vo; //頂点の帰りがけ順保持。
bool use[V];
int ord[V];
int n;

void fdfs(int v){
   use[v] = true;
   rep(i,g[v].sz){
      if(!use[g[v][i]])fdfs(g[v][i]);
   }
   vo.pb(v);
}

void rdfs(int v,int k){
   use[v] = true;
   ord[v] = k;
   rep(i,rg[v].sz){
      if(!use[rg[v][i]])rdfs(rg[v][i],k);
   }
}

int scc(){
   INIT(use);
   vo.clr;
   rep(i,n)if(!use[i])fdfs(i);

   INIT(use);
   int k = 0;
   rrep(i,vo.sz){
      if(!use[vo[i]])rdfs(vo[i],k++);
   }
   return k;
}
```

2.9 LCA(最近共通祖先)

根付き木上の 2 点 u, v の LCA をダブリングによって二分探索で求める。$O(\log V)$。

```
vector<int> g[N];
int root; //根ノードの番号
int par[LOG_N][N];
int depth[N];

void dfs(int v,int p,int d){
   par[0][v] = p;
   depth[v] = d;
   rep(i,g[v].sz){
      if(g[v][i] != p)dfs(g[v][i],v,d+1);
   }
}

void init(){
   dfs(root,-1,0);
   rep(k,LOG_N-1){
      if(par[k][v] < 0)par[k+1][v] = -1;
      else par[k+1][v] = par[k][par[k][v]];
   }
}

int lca(int u,int v){
   if(depth[u] > depth[v])swap(u,v);
   rep(k,LOG_N){
      if((depth[v] - depth[u]) >> k & 1)v = par[k][v];
   }
   if(u==v)return u;

   rrep(k,LOG_V){
      if(par[k][u] != par[k][v]){
         u = par[k][u]; v = par[k][v];
      }
   }
   return par[0][u];
}
```

3 データ構造

3.1 Union-Find 木

集合の管理ができる。２つの要素 a,b に対し、

1. a と b が同じ集合か (= same(a,b))
2. a と b が含まれる集合を併合する (= unite(a,b))

ができる。$O(\alpha(n))$。

```
int par[N]; //親ノード
int r[N]; //木の高さ

void init_uf(int n){
  for(int i=0;i<n;i++){par[i] = i;r[i] = 0;}
}

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

void unite(int x,int y){
  x = find(x);
  y = find(y);
  if(x==y)return;

  if(r[x] < r[y])par[x] = y;
  else par[y] = x;
  if(r[x] == r[y])r[x]++;
}

bool same(int x,int y){return find(x)==find(y);}
```

3.2 重み付き Union-Find

a と b の相対的距離も込みで連結を管理する。a,b が同じ集合に含まれるとき、dis(a,b) で相対距離を測れる。

```
P par[N];
int r[N];

void init(int n){
  rep(i,n)par[i] = P(i,0);
  INIT(r);
}

P find(int a){
  if(par[a].fs == a)return par[a];
  P tmp = find(par[a].fs);
  return par[a] = P(tmp.fs,tmp.sc + par[a].sc);
}

bool same(int a,int b){
  return (find(a).fs == find(b).fs);
}

bool unite(int a,int b,int cost){
  P x = find(a);
  P y = find(b);

  if(same(x.fs,y.fs)){
    if(y.sc - x.sc != cost)return false;
  }else{
    if(r[x.fs]<r[y.fs]){
      par[x.fs] = P(y.fs,y.sc-x.sc-cost);
    }else{
      par[y.fs] = P(x.fs,x.sc-y.sc+cost);
      if(r[x.fs] == r[y.fs])r[x.fs]++;
    }
  }
  return true;
}

int dis(int a, int b){
  return find(b).sc - find(a).sc;
}
```

3.3 セグメント木

RMQ(Range Minimum Query) と値の更新に $O(\log n)$ で対応するセグメント木。min を変更して他のクエリに対応可能。

```
int n,dat[2*N-1];

void init(int n_){
  n=1;
  while(n<n_)n*=2;
  rep(i,2*n-1)dat[i] = INF;
}

void update(int k,int a){
  k += n-1;
  dat[k] = a;
  while(k>0){
    k = (k-1)/2;
    dat[k] = min(dat[k*2+1],dat[k*2+2]);
  }
}

//return minimam value in [a,b). ( [l,r) is interval in which k is.)
int query(int a,int b,int k,int l,int r){
  if(r<=a || b<=l)return INF;
  if(a<=l && r<=b)return dat[k];
  int vl = query(a,b,2*k+1,l,(l+r)/2);
  int vr = query(a,b,2*k+2,(l+r)/2,r);
  return min(vl,vr);
}
```

3.4 BIT(Fenwick 木)

部分和クエリと値の加算クエリに $O(\log n)$ で答えるデータ構造。

```
//区間 [1,n] なので注意!!
ll bit[N_MAX];

void init_bit(int n){
  for(int i=0;i<=n;i++)bit[i]=0;
}

ll sum(int i){
  ll res = 0;
  while(i > 0){
    res += bit[i];
    i -= i & -i;
  }
  return res;
}

void add(int i, ll x){
  while(i <= n){
    bit[i] += x;
    i += i & -i;
  }
}
```

4 数学

4.1 素数

4.1.1 素数判定

$O(\sqrt{n})$。

```
bool isPrime(ll n){
  if(n<2)return false;
  for(ll i=2;i*i<=n;i++)if(!(n%i))return false;
  return true;
}
```

4.1.2　エラトステネスの篩

$O(n \log \log n)$ で素数表を作成。

```
bool pt[N+1];

void Eratosthenes(int n){
   pt[0] = pt[1] = false;
   for(int i=2;i<=n;i++)pt[i] = true;

   int i = 3;
   while(i*i<=n){
      for(int j=i+i;j<=n;j+=i)pt[j] = false;
      while(i*i<=n && !pt[++i]);
   }
}
```

4.2　約数・mod

4.2.1　繰り返し二乗法

$a^n \ (mod\ m)$ を $O(\log n)$ で計算する。

```
ll pow_mod(ll a, ll n, ll m){
   ll res = 1;
   while(n){
      if(n&1)res = res * x % m;
      x = x * x % m;
      n>>=1;
   }
   return res;
}
```

4.2.2　GCD,LCM

A と B の最大公約数と最小公倍数を求める。ユークリッドの互除法。$O(\log max(A, B))$。組み込み関数__gcd(A,B) を使ってもよい。

```
ll gcd(ll a, ll b){return b?gcd(b,a%b):a;}
ll lcm(ll a, ll b){return a/gcd(a,b)*b;}
```

4.2.3　拡張ユークリッドの互除法

$ax + by = 1$ となる整数 x,y を求める。gcd(a,b)$\neq$ 1 のとき解はなし。

```
ll extgcd(ll a, ll b, ll &x, ll &y){
   ll d = a;
   if(b){
      d = extgcd(b,a%b, y, x);
      y -= (a/b) * x;
   }else{
      x = 1; y = 0;
   }
   return d;
}
```

4.2.4　逆元

m を法としたときの a の逆元 a^{-1} を求める。拡張ユークリッドの互除法を用いるため、gcd(a,m)$\neq$ 1 $\Leftrightarrow$ a と m が互いに素である必要がある。ちなみに m が素数の場合、後述のフェルマーの小定理により $a^{-1} \equiv a^{m-2} \ (mod\ m)$ である。

```
ll mod_inverse(ll a, ll m){
   ll x,y;
   extgcd(a,m,x,y);
   return (m+x%m) %m;
}
```

4.2.5 オイラーの ϕ 関数

m が素数でないとき、$m = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ とすると、$\phi(m) = m \times \prod(p_i - 1)/p_i$ は「m 以下の m と互いに素な自然数の個数」に等しい。このとき、m と互いに素な x について、$x^{\phi(m)} \equiv 1\ (mod\ m)$ が成り立つ。$O(\sqrt{m})$ で ϕ 関数を個別に求める方法と、エラトステネスの篩を応用して $O(m)$ 程度で ϕ 関数のテーブルを作る方法がある。

```
int euler_phi(int n){
  int res = n;
  for(int i=2;i*i<=n;i++){
    if(n%i==0){
      res = res / i * (i-1);
      for(;n%i==0;n/=i);
    }
  }
  if(n!=1)res = res / n * (n-1);
  return res;
}

//テーブル作成
int euler[N];
void euler_phi2(){
  for(int i=0;i<N;i++)euler[i] = i;
  for(int i=2;i<N;i++){
    if(euler[i] === i){
      for(int j=i;j<N;j+=i)euler[j] = euler[j] / i * (i-1);
    }
  }
}
```

4.2.6 メビウスの反転公式

周期的なものの数え上げに用いる。周期が n の約数であるものの総数を $f(n)$、周期がちょうど n であるものの個数を $g(n)$ とすると、以下のメビウスの反転公式が成り立つ。

$$f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) f(d)$$

ただし、$a|b$ は b の任意の約数 a、$\mu(n)$ はメビウス関数である。メビウス関数 $\mu(n)$ は、以下のように求められる。

- n が 1 でない平方数で割り切れるとき、$\mu(n) = 0$
- そうでないとき、n の素因数の個数を k とすると、$\mu(n) = (-1)^k$

以下はすべての要素が x 種類あり、かつそれが等確率で現れるとしたときの $g(n)$ を求める。メビウス関数の計算は $O(n$ の約数の個数 $\times \sqrt{n})$ である。

```
map<int,int> moebius(int n){
  map<int,int> res;
  vector<int> primes;

  //n の素因数の列挙
  for(int i=2;i*i<=n;i++){
    if(n%i == 0){
      primes.pb(i);
      while(n%i==0)n/=i;
    }
  }
  if(n!=1)primes.pb(n);

  int m = primes.sz;
  rep(i,1<<m){
    int mu = 1, d = 1;
    rep(j,m){
      if(i>>j & 1){
        mu *= -1;
        d *= primes[j];
      }
    }
    res[d] = m;
  }
  return res;
}

int calc_gn(int n, int x, int mod){
  int res = 0;
```

```
  map<int,int> mu = moebius(n);
  for(map<int,int>::iterator it = mu.begin();it!=mu.end();it++){
    res += it->second * mod_pow(x,n / it->first); //f(d) = x^d
    res = (res % mod + mod) % mod;
  }
  return res;
}
```

4.3 行列

ガウスジョルダンの消去法により連立 1 次方程式を解ける。$O(n^3)$。

```
typedef vector<double> vec;
typedef vector<vec> mat;

struct matrix{
  mat m;
  int r,c;

  matrix(void){r=c=0;m.clr;}

  matrix(mat a){
    r = a.sz;c = a[0].sz;
    m.resize(r);
    rep(i,r)m[i].resize(c);
    rep(i,r)rep(j,c)m[i][j] = a[i][j];
  }

  matrix operator+(matrix a){
    if(r==a.r && c==a.c){
      rep(i,r)rep(j,c)a.m[i][j] += m[i][j];
    }
    return a;
  }

  matrix operator-(matrix a){
    rep(i,a.r)rep(j,a.c)a.m[i][j] *= -1;
    return *this+a;
  }

  matrix operator*(matrix a){
    matrix x;
    if(c==a.r){
      x.r = r;x.c = a.c;
      x.m.resize(r);
      rep(i,r)x.m[i].resize(a.c);
      rep(i,r)rep(j,a.c){
        x.m[i][j] = 0;
        rep(k,c)x.m[i][j] += m[i][k] * a.m[k][j];
      }
    }
    return x;
  }
};

vec gauss_jordan(const mat& A, const vec& b){
  int n = A.size();
  mat B(n,vec(n+1));
  rep(i,n)rep(j,n)B[i][j] = A[i][j];
  rep(i,n)B[i][n] = b[i];

  rep(i,n){
    int p = i;
    for(int j=i;j<n;j++)
      if(abs(B[j][i]) > abs(B[p][i]))p = j;
    swap(B[i],B[p]);

    //解がないか、一意でない
    if(abs(B[i][i]) < EPS)return vec();

    for(int j=i+1;j<=n;j++)B[i][j] /= B[i][i];
    rep(j,n)
      if(i != j)
        for(int k=i+1;k<=n;k++)B[j][k] -= B[j][i] * B[i][k];
  }
  vec x(n);
  rep(i,n)x[i] = B[i][n];
  return x;
}
```

4.4 grundy 数

Nim を一般化した概念。Nim の勝敗判定は、各山の石の数が a_i であるとき、$a_1\ XOR\ \cdots\ XOR\ a_n = 0 \Leftrightarrow$負け状態。同様に、grundy 数の XOR が 0 であれば負け状態。 DP により O(状態数×遷移数) で前計算できる。

```
int N,K,X[N],A[N];
int grundy[X+1];

//石の個数が X[i] の山が N 個, 一度に取ることができる石の個数 A[i] が K 種類
void calc_grundy(){
    grundy[0] = 0;

    int max_x = *max_element(X,X+N);
    for(int j=1;j<=max_x;j++){
        set<int> s;
        rep(i,K){
            if(A[i]<=j)s.insert(grundy[j-A[i]]);
        }

        int g = 0;
        while(s.count(g) != 0)g++;
        grundy[j] = g;
    }
}
```

4.5 公式集

1. フェルマーの小定理: 素数 p、任意の整数 x に対し、$x^p \equiv x (mod p)$
2. 中国剰余定理: k 個の整数 m_i がどの 2 つも互いに素ならば、任意に与えられる k 個の整数 a_i に対し、$x \equiv a_i (mod m_i)$ である x が一意に定まる。
3. ポリアの数え上げ定理: すべてのパターンをちょうど同じ回数だけ数え上げ、重複回数で割ることで数え上げが可能
4. シンプトン公式: 数値積分の公式。本来は近似値だが、$f(x)$ が二次以下であれば厳密値が得られる。

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

5 文字列

5.1 KMP 法

単一のパターン文字列 P がテキスト文字列 T に部分文字列として含まれるか判定。$O(|P| + |T|)$。パターンマッチングオートマトン (PMA) は複数テキストに使い回せる。

```
int Knuth_Morris_Pratt(string text,string pattern){
    int n = text.size(), m = pattern.size(), res = 0;
    vector<int> pma(m+1); //Pattern Matching Automaton

    int j = pma[0] = -1;
    for(int i=1;i<=m;++i){
        while(j>=0 && pattern[i-1] != pattern[j])j = pma[j];
        pma[i] = ++j;
    }

    for(int i=0,k=0;i<n;++i){
        while(k>=0 && pattern[k] != text[i])k = pma[k];
        if(++k >= m){
            return i-m+1;
            //res++; k = pma[k]; //for counting the number of the occurences
        }
    }
    return n;
}
```

5.2 ラビンカープ法 (ローリングハッシュ)

複数のパターン文字列 P がテキスト文字列 T に部分文字列として含まれるか判定。$O(\sum |P| + |T|)$。ハッシュ値の衝突に注意。

```
#define BASE 1000000007ULL //適当な素数
typedef unsigned long long ull;

set<ull> hash;
ull acc_h[N]; //hash value for interval [-1,i)
ull power[N]; //BASE のi 乗

void init_roll(string s,int n){
  acc_h[0] = 0;
  power[0] = 1;
  for(int i=1;i<=n;i++){
    power[i] = power[i-1] * BASE;
    acc_h[i] = acc_h[i-1] * BASE + s[i-1];
  }
}

ull rolling_hash(string s,int l,int r){
  return acc_h[r] - acc_h[l] * power[r-l];
}

ull roll_inc_nxt(string s,ull val,int r){
  return val*BASE + s[r-1];
}

ull roll_dec_prv(string s,ull val,int l,int r){
  return val - s[l] * power[r-l];
}
```

5.3 Suffix Array

Manber & Myers の構築法。$O(n \log^2 n)$。

```
int n,k;
int r[N+1];
int tmp[N+1];

bool compare_sa(int i,int j){
  if(r[i] !=r[j])return r[i]<r[j];
  int ri = i+k <= n ? r[i+k]:-1;
  int rj = j+k <= n ? r[j+k]:-1;
  return ri<rj;
}

void construct_sa(string S, int *sa){
  n = S.sz;
  rep(i,n+1){
    sa[i] = i;
    r[i] = i<n ? s[i]:-1;
  }

  for(k=1;k<=n;k*=2){
    sort(sa,sa+n+1,compare_sa);

    tmp[sa[0]] = 0;
    for(int i=1;i<=n;i++){
      tmp[sa[i]] = tmp[sa[i-1]] + (compare_sa(sa[i-1],sa[i]) ? 1:0);
    }
    rep(i,n+1)r[i] = tmp[i];
  }
}
```

5.4 LCP 配列

Longest Common Prefix の長さを表す配列を作る。$O(n)$。

```
int r[N+1];

void construct_lcp(string S, int *sa, int *lcp){
  int n = S.sz;
  rep(i,n+1)r[sa[i]] = i;

  int h = 0;
```

```
  lcp[0] = 0;
  rep(i,n){
    int j = sa[r[i]-1];

    if(h>0)h--;
    for(;j+h<n && i+h<n;h++){
      if(S[j+h] != S[i+h])break;
    }
    lcp[r[i]-1] = h;
  }
}
```

6 二次元幾何

6.1 ベクトル

点、およびベクトルは complex 型 (複素数型) を用いて表現する。

```
typedef complex<D> P;

//点の比較関数 (x 座標優先、同じなら y 座標を比較)
namespace std{
  bool operator<(const P &a,const P &b){
    return real(a)==real(b)?imag(a)<imag(b):real(a)<real(b);
  }
}

//p と同方向の単位ベクトルを返す
P unit(P p){return p / abs(p);}

//p と同じ長さの法線ベクトルを 2 つとも返す
pair<P,P> norm(P p){return make_pair(p*P(0,1),p*P(0,-1));}

//x,y の内積を返す
D dot(P x,P y){return real(conj(x)*y);}

//x,y の内積を返す
D cross(P x,P y){return imag(conj(x)*y);}

//rotate a point counter-clockwise on the origin
P rotate(P v,double s){
  return P(real(v)*cos(s) - imag(v)*sin(s), real(v)*sin(s) + imag(v)*cos(s) );
}

//3 点 a,b,c に対し、角 bac の角度を求める
D arg(P a,P b,P c){return acos(dot(b-a,c-a)/(abs(b-a)*abs(c-a)));}
//各辺の長さが a,b,c である三角形に対し、a の対角の角度を求める
D arg(D a,D b,D c){return acos( (b*b+c*c-a*a)/(2*b*c) );}
```

6.2 点集合

6.2.1 凸包

点集合に対する凸包を求める。$O(n \log n)$。

```
vector<P> convex_hull(vector<P> v){
  int n = v.sz, k = 0;
  sort(all(v));
  vector<P> r(2*n);
  for(int i=0;i<n;i++){
    while(k>1 && cross(r[k-1]-r[k-2],v[i]-r[k-1]) <= EPS)k--;
    r[k++] = v[i];
  }
  for(int i=n-2,t=k;i>=0;i--){
    while(k>t && cross(r[k-1]-r[k-2],v[i]-r[k-1]) <= EPS)k--;
    r[k++] = v[i];
  }
  r.resize(k-1);
  return r;
}
```

6.2.2 最遠点対

凸多角形に対し、その頂点対の中で最も長い距離を返す。$O(n)$。上記の凸包の後に使うべし。

```
D caliper(Poly p){
  int n = p.sz;
  if(n==2)return abs(p[0]-p[1]);

  int i = 0, j = 0;
  rep(k,n){
    if(!(p[i]<p[k]))i = k;
    if(p[j]<p[k])j = k;
  }

  D res = 0;
  int si = i, sj = j;
  while(i != sj || j != si){
    res = max(res,abs(p[i]-p[j]));
    if( cross(p[(i+1)%n] - p[i], p[(j+1)%n] - p[j]) < 0)i = (i+1)%n;
    else j = (j+1)%n;
  }
  return res;
}
```

6.2.3 最小包含球

点集合に対し、それらをすべて含むような最小の円を求めるアルゴリズム。dim を変えることで 3 次元以上にも対応するはずだが、貰い物のソースコードで理解していないので下手なことはしないこと。

```
//Minimam Cover Ball
//Using: Call for "solve" with vector<P> v, which is a point set, as argument
const int dim = 2;
struct min_ball{
  P center,v[dim+1],c[dim+1];
  double radius2,z[dim+1],r[dim+1];
  list<P> ps;
  list<P>::iterator sup;
  int m;

  void solve(vector<P> v){
    m = 0;
    center = P(0,0);
    radius2 = -1;
    ps.clear();
    for(int i=0;i<v.sz;i++)ps.pb(v[i]);
    make_ball(ps.end());
  }

  void push(const P& p){
    if(!m){
      c[0] = p; r[0] = 0;
    }else{
      double e = dot(p-c[m-1],p-c[m-1]) - r[m-1];
      P d = v[m] = p-c[0];
      for(int i=1;i<m;i++)v[m] -= v[i]*dot(v[i],d)/z[i];
      z[m] = dot(v[m],v[m]);
      c[m] = c[m-1] + e*v[m]*(1.0/z[m]/2.0);
      r[m] = r[m-1] + e*e/z[m]/4.0;
    }
    center = c[m];
    radius2 = r[m]; m++;
  }
  void make_ball(list<P>::iterator i){
    sup = ps.begin();
    if(m == dim + 1)return;
    for(list<P>::iterator k = ps.begin();k!=i;){
      list<P>::iterator j = k++;
      if(dot(*j-center,*j-center) > radius2){
        push(*j);
        make_ball(j); m--;
        if(sup == j)++sup;
        ps.splice (ps.begin(),ps,j);
      }
    }
  }
};
```

6.3 直線・線分

```
typedef pair<P,P> L;

//頂点 a,b,c の位置関係を判定
int ccw(P a,P b,P c){
  b -= a;c -= a;
  if (cross(b,c)>EPS) return 1; //counter clockwise
  if (cross(b,c)<-EPS) return -1; //clockwise
  if (dot(b, c)<-EPS) return 2; //c--a--b on line
  if (abs(b)<abs(c)) return -2; //a--b--c on line
  return 0; //on segment
}

//直交判定
bool orth(L a,L b){return abs(dot(a.fs-a.sc,b.fs-b.sc))<EPS;}

//平行判定
bool para(L a,L b){return abs(cross(a.fs-a.sc,b.fs-b.sc))<EPS;}

//直線と点との距離
D line_dis(L a,P x){return abs(cross(a.sc-a.fs,x-a.fs))/abs(a.sc-a.fs);}

//2 直線の交点
P line_cp(L a,L b){
  return a.fs+(a.sc-a.fs)*cross(b.sc-b.fs,b.fs-a.fs)/cross(b.sc-b.fs,a.sc-a.fs);
}

//線分と点との距離
D seg_p_dis(L a,P x){
  if(dot(a.sc-a.fs,x-a.fs)<EPS)return abs(x-a.fs);
  if(dot(a.fs-a.sc,x-a.sc)<EPS)return abs(x-a.sc);
  return abs(cross(a.sc-a.fs,x-a.fs))/abs(a.sc-a.fs);
}

//2 線分間の距離
D seg_seg_dis(L a,L b){
  D res = 1e10;
  res = min(res,seg_p_dis(a,b.fs));
  res = min(res,seg_p_dis(a,b.sc));
  res = min(res,seg_p_dis(b,a.fs));
  res = min(res,seg_p_dis(b,a.sc));
  return res;
}

//2 線分が交点を持つかどうか判定
bool is_cp(L a,L b){
  if(ccw(a.fs,a.sc,b.fs)*ccw(a.fs,a.sc,b.sc)<=0)
    if(ccw(b.fs,b.sc,a.fs)*ccw(b.fs,b.sc,a.sc)<=0)return true;
  return false;
}

//上記判定で交点を持つなら、2 線分の交点を求める
P seg_cp(L a,L b){
  D d = abs(cross(b.sc-b.fs,a.fs-b.fs));
  return a.fs + (a.sc-a.fs)*( d /(d + abs(cross(b.sc-b.fs,a.sc-b.fs))) );
}
```

6.4 多角形

6.4.1 ヘロンの公式

三角形の面積を求めるヘロンの公式。

```
D heron(D a,D b,D c){
  D s = (a+b+c)/2;
  return sqrt(s*(s-a)*(s-b)*(s-c));
}
```

6.4.2 一般の多角形 (凸に限らない)

```
typedef vector<P> Poly;

//多角形の面積を返す。反時計周りに整列されている必要がある。
```

```
D area(Poly p){
  if(p.sz<3)return 0;
  D res = cross(p[p.sz−1],p[0]);
  for(int i=1;i<p.sz;i++)res += cross(p[i−1],p[i]);
  return res/2;
}

//点 x が多角形 p に含まれているか。反時計周りに整列されている必要がある。
bool inter_cp(Poly p,P x){
  if(p.empty())return false;

  int s = p.sz;
  double max = p[0].real();
  for(int i=1;i<s;i++)if(max < p[i].real())max = p[i].real();
  L h = L( x,P(max+1.0,x.imag()) );

  int c = 0;
  rep(i,s){
    L l = L(p[i],p[(i+1)%s]);
    if(para(h,l) && abs(ccw(h.fs,h.sc,l.fs))!=1)c++;
    else if(is_cp(h,l))c++;
  }

  if(c&1)return true;
  return false;
}
```

6.4.3 凸カット

凸多角形 p を直線 l でカットし、左手にできる多角形を返す。反時計周りに整列されている必要がある。

```
Poly convex_cut(Poly p,L l){
  Poly res;
  int n = p.size();
  for(int i=0;i<n;i++){
    int nxt = (i+1)%n;
    if(ccw(l.fs,l.sc,p[i]) != −1)res.pb(p[i]);
    if(ccw(l.fs,l.sc,p[i]) * ccw(l.fs,l.sc,p[nxt]) < 0){
      res.pb( line_cp(l, L(p[i],p[nxt]) ) );
    }
  }
  return res;
}
```

6.5 円

```
typedef pair<P,D> C;

//円の面積を返す
D area_cir(C c){return PI * c.sc * c.sc;}
//点 x が円 c の内部にあるかどうか判定する
bool in_cir(C c,P x){return (abs(x−c.fs) +EPS < c.sc);}
//点 x が円 c の周上にあるかどうか判定する
bool on_cir(C c,P x){return EQ(abs(x−c.fs),c.sc);}

//2 円の関係
int cpr(C a,C b){
  double d = abs(a.fs−b.fs);
  if(a.sc+b.sc + EPS < d)return −1; //no cross point (outside)
  if(b.sc+d + EPS < a.sc)return 1; //no cross point (inside,B in A)
  if(a.sc+d + EPS < b.sc)return 2; //no cross point (inside,A in B)
  //−−−−−−−−−−−−above verified−−−−−−−−−−−−−−−−//
  if(abs(a.sc+b.sc − d) < EPS)return −3; //one cross point (outside)
  if(abs(b.sc+d − a.sc) < EPS)return 3; //one cross point (inside,B in A)
  if(abs(a.sc+d − b.sc) < EPS)return 4; //one cross point (inside,A in B)
  return 0; //two cross point
}

//2 円の共通部分の面積を求める
D intersection_area(C a, C b){
  D d = abs(a.fs−b.fs);
  D ar = a.sc, br = b.sc;
  if(a.sc + b.sc < d + EPS)return 0;
  if(a.sc < b.sc)swap(a,b);
  if(b.sc + d < a.sc + EPS || b.sc < EPS)return area_cir(b);

```

```
    D t1 = arg(b.sc,a.sc,d), t2 = arg(a.sc,b.sc,d);
    D tri = ( a.sc*a.sc*sin(t1*2) + b.sc*b.sc*sin(t2*2) )/2.0;
    return a.sc*a.sc*t1 + b.sc*b.sc*t2 - tri;
}

//2 円の交点を列挙する
vector<P> cp_cir_to_cir(C a, C b){
    vector<P> v;
    int pos = cpr(a,b);
    if(pos==0){
        D s = arg(b.sc,abs(b.fs-a.fs),a.sc);
        P x = a.sc * unit(b.fs - a.fs);
        v.pb(a.fs + rotate(x,s));
        v.pb(a.fs + rotate(x,-s));
    }else if(abs(pos) >= 3){
        v.pb(a.fs + a.sc * unit(b.fs-a.fs));
    }
    return v;
}

//not verified. 円と直線の交点
vector<P> cp_cir_to_line(C a, L l){
    vector<P> v;
    D d = line_dis(l,a.fs);
    if(d < a.sc + EPS){
        P x = a.sc*unit(l.sc - l.fs);
        if(ccw(l.fs,l.sc,a.fs) == 1)x = a.fs + x*P(0,-1);
        else x = a.fs + x*P(0,1);
        if(d + EPS < a.sc){
            D y = sqrt(a.sc*a.sc - d*d);
            D s = arg(y,d,a.sc);
            v.pb(rotate(x,s));
            v.pb(rotate(x,-s));
        }else if(EQ(d,a.sc))v.pb(x);
    }
    return v;
}

//点 p を通るような c の接線を求める
vector<L> adj_line(C c,P p){
    vector<L> res;
    if(in_cir(c,p))return res;
    if(on_cir(c,p)){
        pair<P,P> n = norm(c.fs-p);
        res.pb(L(n.fs+p,p));
        return res;
    }
    D x = c.sc, z = abs(c.fs-p);
    D y = sqrt(z*z-x*x);
    D s = arg(y,x,z);
    P v = unit(p-c.fs)*c.sc;

    res.pb(L(rotate(v,s)+c.fs,p));
    res.pb(L(rotate(v,-s)+c.fs,p));
    return res;
}

//2 円の共通接線を求める
vector<L> common_adj_line(C a,C b){
    vector<L> res;
    if(a.sc+EPS<b.sc)return common_adj_line(b,a);
    if(EQ(real(a.fs),real(b.fs)) && EQ(imag(a.fs),imag(b.fs)) && EQ(a.sc,b.sc))return res;

    P pos = (b.fs-a.fs)*a.sc/(a.sc+b.sc)+a.fs;
    if(!in_cir(a,pos))res = adj_line(a,pos);

    if(EQ(a.sc,b.sc)){
        pair<P,P> n = norm(unit(b.fs-a.fs)*a.sc);
        res.pb(L(a.fs+n.fs,b.fs+n.fs));
        res.pb(L(a.fs+n.sc,b.fs+n.sc));
    }else{
        D c = abs(b.fs-a.fs);
        pos = unit(b.fs-a.fs)*( (a.sc*c)/(a.sc-b.sc) )+a.fs;
        if(!in_cir(a,pos)){
            vector<L> tmp = adj_line(a,pos);
            rep(i,tmp.sz)res.pb(tmp[i]);
        }
    }
    return res;
}
```

7 その他

7.1 ビット演算

7.1.1 sideways addition

1 が立っているビット数を数える。64bit にするときは__builtin_popcountl(long long x) を使う。

```
__builtin_popcount(int x);
```

7.1.2 n choose k の列挙

n アイテムから k アイテムを選ぶ部分集合を bit 表現で列挙する。

```
int comb = (1<<k) -1;
while(comb < (1<<n)){
  //処理を記述
  int x = comb & -comb, y = comb + x;
  comb = ( ( (comb & ~y) /x ) >> 1) | y;
}
```

7.2 日付

7.2.1 Fliegel の公式

```
ll Fliegel(void){
  int a = (14-m)/12, Y = y+4800-a, M = m+12*a-3;
  return (ll)d + (153*M+2)/5 + 365*Y + Y/4 - Y/100 + Y/400 - 32045;
}
```

7.2.2 Zeller の公式

```
int Zeller(void){
  int Y = y, M = m;
  if(M<3){Y--; M+=12;}
  return (Y + Y/4 - Y/100 + Y/400 + (13*M+8)/5 + d)%7;
}
```

7.3 さいころ

```
/*
.....+-+.....
......| 1 |......
+-+-+-+
.| 4 | 2 | 3 |
+-+-+-+
......| 6 |......
.....+-+.....
......| 5 |......
.....+-+.....
*/

//up-side = 1, north-side = 2, west-side = 3,
//rotation {North, West, East, South}
int r[4][6] = { {4,0,2,3,5,1},
{3,1,0,5,4,2},
{2,1,5,0,4,3},
{1,5,2,3,0,4} };

//key: up-side and north-side, value: west-side
int dice[6][6] = { {0,2,4,1,3,0},
{3,0,0,5,0,2},
{1,5,0,0,0,4},
{4,0,0,0,5,1},
{2,0,5,0,0,3},
{0,3,1,4,2,0} };
```