Contents

1 準備 1
- 1.1 init.el 1
- 1.2 tpl.cpp 1

2 文字列 1
- 2.1 マッチング 1
  - 2.1.1 複数文字列マッチング (Aho-Corasick 法) 1
- 2.2 Suffix Array 2

3 グラフ 3
- 3.1 強連結成分分解 3
  - 3.1.1 関節点 3
  - 3.1.2 橋 3
  - 3.1.3 強連結成分分解 3
- 3.2 フロー 4
  - 3.2.1 最大流 4
  - 3.2.2 二部マッチング 4
  - 3.2.3 最小費用流 4
- 3.3 木 5
  - 3.3.1 木の直径 5
  - 3.3.2 最小シュタイナー木 5
- 3.4 包除原理 5
  - 3.4.1 彩色数 5

4 数学 5
- 4.1 整数 5
  - 4.1.1 拡張ユークリッドの互除法 5
  - 4.1.2 逆元 6
  - 4.1.3 冪剰余 6
  - 4.1.4 階乗 ($n!$ mod m) 6
  - 4.1.5 組み合わせ ($_nC_k$ mod m) 6
  - 4.1.6 カタラン数 6
- 4.2 多項式 6
  - 4.2.1 FFT(complex) 7
  - 4.2.2 FFT(modulo) 7
  - 4.2.3 積 (FMT) 7
  - 4.2.4 逆元 (FMT) 7
  - 4.2.5 平方根 (FMT) 7
- 4.3 行列 8
  - 4.3.1 単位行列 8
  - 4.3.2 積 8
  - 4.3.3 累乗 8
  - 4.3.4 線形方程式の解 (Givens 消去法) 8
  - 4.3.5 トレース 8

1 準備

1.1 init.el

linum は emacs24 のみ

```
;key
(keyboard-translate ?\C-h ?\C-?)
(global-set-key "\M-g" 'goto-line)

;tab
(setq-default indent-tabs-mode nil)
(setq-default tab-width 4)
(setq indent-line-function 'insert-tab)

;line number
(global-linum-mode t)
(setq linum-format "%4d ")
```

1.2 tpl.cpp

```
#include <bits/stdc++.h>
using namespace std;

#define rep(i,a) for(int i = 0;i < (a); i++)
#define repi(i,a,b) for(int i = (a); i < (b); i++)
#define repd(i,a,b) for(int i = (a); i >= (b); i--)
#define repit(i,a) for(__typeof((a).begin()) i = (a).begin(); i != (a).end(); i++)
#define all(u) (u).begin(),(u).end()
#define rall(u) (u).rbegin(),(u).rend()
#define UNIQUE(u) (u).erase(unique(all(u)),(u).end())
#define pb push_back
#define mp make_pair
const int INF = 1e9;
const double EPS = 1e-8;
const double PI = acos(-1.0);

typedef long long ll;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef pair<int,int> pii;

int main(){
}
```

2 文字列

2.1 マッチング

2.1.1 複数文字列マッチング (Aho-Corasick 法)

$O(N+M)$

```
struct PMA{
    PMA* next[256];     //0 is failure link
    vi matched;
    PMA(){memset(next, 0, sizeof(next));}
    ~PMA(){rep(i,256) if(next[i]) delete next[i];}
};
```

```
vi set_union(const vi &a,const vi &b){
    vi res;
    set_union(all(a), all(b), back_inserter(res));
    return res;
}
// patternからパターンマッチングオートマトンの生成
PMA *buildPMA(vector<string> pattern){
    PMA *root = new PMA, *now;
    root->next[0] = root;
    rep(i, patter.size()){
        now = root;
        rep(j, pattern[i].size()){
            if(now->next[(int)pattern[i][j]] == 0)
                now->next[(int)pattern[i][j]] = new PMA;
            now = now->next[(int)pattern[i][j]];
        }
        now->matched.push_back(i);
    }
    queue<PMA*> que;
    repi(i,1,256){
        if(!root->next[i]) root->next[i] = root;
        else {
            root->next[i]->next[0] = root;
            que.push(root->next[i]);
        }
    }
    while(!que.empty()){
        now = que.front(); que.pop();
        repi(i,1,256){
            if(now->next[i]){
                PMA *next = now->next[0];
                while(!next->next[i]) next = next->next[0];
                now->next[i]->next[0] = next->next[i];
                now->next[i]->matched = set_union(now->next[i]->matched, next->next[i]->matched);
                que.push(now->next[i]);
            }
        }
    }
    return root;
}
void match(PMA* &pma, const string s, vi &res){
    rep(i,s.size()){
        int c = s[i];
        while(!pma->next[c])
            pma = pma->next[0];
        pma = pma->next[c];
        rep(j,pma->matched.size())
            res[pma->matched[j]] = 1;
    }
}
```

2.2 Suffix Array

find_string() : $O(|T| \log |S|)$
S 中に T が含まれないなら-1, 含まれるならその先頭.
LCS() : $O(|S + T|)$
最長共通部分文字列. (先頭, 長さ) を返す.

```
const int MAX_N = 1000000;
int n, k;
int rank[MAX_N+1], tmp[MAX_N+1], sa[MAX_N+1], lcp[MAX_N+1];

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

void construct_sa(string S, int *sa) {
  n = S.length();
  for(int i = 0; i <= n; i++) {
    sa[i] = i;
    rank[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);
    }
    for(int i = 0; i <= n; i++) {
      rank[i] = tmp[i];
    }
  }
}

void construct_lcp(string S, int *sa, int *lcp) {
  int n = S.length();
  for(int i = 0; i <= n; i++) rank[sa[i]] = i;
  int h = 0;
  lcp[0] = 0;
  for(int i = 0; i < n; i++) {
    int j = sa[rank[i] - 1];
    if(h > 0) h--;
    for(; j + h < n && i + h < n; h++) {
      if(S[j+h] != S[i+h]) break;
    }
    lcp[rank[i] - 1] = h;
  }
}

//============ 使用例 =============//
// 文字列検索(蟻本p338 改) O(|T|log|S|)
// S中にTが含まれないなら-1, 含まれるならその先頭
int find_string(string S, int *sa, string T) {
  int a = 0, b = S.length();
  while(b - a > 1) {
    int c = (a + b) / 2;
    if(S.compare(sa[c], T.length(), T) < 0) a = c;
    else b = c;
  }
  return (S.compare(sa[b], T.length(), T) == 0)?sa[b]:-1;
}

// 最長共通部分文字列(蟻本p341 改) construct_sa以外はO(|S+T|)
// (先頭, 長さ)を返す
pair<int, int> LCS(string S, string T) {
  int sl = S.length();
  S += '\0' + T;
  construct_sa(S, sa);
  construct_lcp(S, sa, lcp);
  int len = 0, pos = -1;
  for(int i = 0; i < S.length(); i++) {
    if(((sa[i] < sl) != (sa[i+1] < sl)) && (len < lcp[i])) {
      len = lcp[i];
      pos = sa[i];
```

```
    }
  }
  return make_pair(pos, len);
}
```

3 グラフ

3.1 強連結成分分解

3.1.1 関節点

$O(E)$

ある関節点 u がグラフを k 個に分割するとき art には k-1 個の u が含まれる. 不要な場合は unique を忘れないこと.

```
vi G[MAX], art; // artに関節点のリストが入る
int num[MAX], low[MAX], t, V;

void visit(int v, int u){
    low[v] = num[v] = ++t;
    repit(e,G[v]){
        int w = *e;
        if (num[w] == 0) {
            visit(w, v);
            low[v] = min(low[v], low[w]);
            if ((num[v] == 1 && num[w] != 2) ||
                (num[v] != 1 && low[w] >= num[v])) art.pb(v);
        }
        else low[v] = min(low[v], num[w]);
    }
}
void art_point(){
    memset(low, 0, sizeof(low));
    memset(num, 0, sizeof(num));
    art.clear();
    rep(u,V) if (num[u] == 0) {
        t = 0;
        visit(u, -1);
    }
    /*
    sort(all(art));
    UNIQUE(art);
    */
}
```

3.1.2 橋

$O(V + E)$

```
vi G[MAX];
vector<pii> brdg; // brdgに橋のリストが入る
stack<int> roots, S;
int num[MAX], inS[MAX], t, V;

void visit(int v, int u){
    num[v] = ++t;
    S.push(v); inS[v] = 1;
    roots.push(v);
    repit(e, G[v]){
        int w = *e;
        if(!num[w]) visit(w, v);
        else if(u != w && inS[w])
            while(num[roots.top()] > num[w])
                roots.pop();
    }
    if(v == roots.top()){
        int tu = u, tv = v;
        if(tu > tv) swap(tu, tv);
        brdg.pb(pii(tu, tv));
        while(1){
            int w = S.top(); S.pop();
            inS[w] = 0;
            if(v == w) break;
        }
        roots.pop();
    }
}

void bridge(){
    memset(num, 0, sizeof(num));
    memset(inS, 0, sizeof(inS));
    brdg.clear();
    while(S.size()) S.pop();
    while(roots.size()) roots.pop();
    t = 0;
    rep(u,V) if(num[u] == 0){
        visit(u,V);
        brdg.pop_back();
    }
}
```

3.1.3 強連結成分分解

$O(V + E)$

```
vi G[MAX];
vvi scc; // ここに強連結成分分解の結果が入る
stack<int> S;
int inS[MAX], low[MAX], num[MAX], t, V;

void visit(int v){
    low[v] = num[v] = ++t;
    S.push(v); inS[v] = 1;
    repit(e,G[v]){
        int w = *e;
        if(num[w] == 0){
            visit(w);
            low[v] = min(low[v], low[w]);
        }
        else if(inS[w]) low[v] = min(low[v], num[w]);
    }
    if(low[v] == num[v]){
        scc.pb(vi());
        while(1){
            int w = S.top(); S.pop();
            inS[w] = 0;
            scc.back().pb(w);
            if(v == w) break;
        }
    }
}

void stronglyCC(){
    t = 0;
    scc.clear();
```

```
    memset(num, 0, sizeof(num));
    memset(low, 0, sizeof(low));
    memset(inS, 0, sizeof(inS));
    while(S.size()) S.pop();
    rep(u,V) if(num[u] == 0) visit(u);
}
```

3.2 フロー

3.2.1 最大流

$O(EV^2)$

```
struct edge{int to, cap, rev;};
vector<edge> G[MAX];
int level[MAX], itr[MAX];

void add_edge(int from, int to, int cap){
    G[from].push_back((edge){to, cap, int(G[to].size())});
    G[to].push_back((edge){from, 0, int(G[from].size()-1)});
}

void bfs(int s, int t){
    memset(level, -1, sizeof(level));
    queue<int> que; que.push(s);
    level[s] = 0;
    while(!que.empty()){
        int v = que.front(); que.pop();
        if(v == t) return;
        for(int i = 0; i < G[v].size(); i++){
            edge &e = G[v][i];
            if(e.cap <= 0 or level[e.to] != -1) continue;
            que.push(e.to);
            level[e.to] = level[v]+1;
        }
    }
}

int dfs(int v, int t, int f){
    if(v == t) return f;
    for(int &i = itr[v] ; i < G[v].size(); i++){
        edge &e = G[v][i];
        if(level[e.to] <= level[v] or e.cap <= 0) continue;
        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 max_flow(int s, int t){
    int flow = 0, f;
    while(1){
        bfs(s, t);
        if(level[t] == -1) return flow;
        memset(itr, 0, sizeof(itr));
        while((f = dfs(s, t, INF)) > 0) flow += f;
    }
}
```

3.2.2 二部マッチング

$O(EV)$

```
int  V;
vector<int> G[MAX_V];
int match[MAX_V];
bool used[MAX_V];

void add_edge(int u, int v){
    G[u].push_back(v);
    G[v].push_back(u);
}

bool dfs(int v){
    used[v] = 1;
    rep(i,G[v].size()){
        int u = G[v][i], w = match[u];
        if(w < 0 || !used[w] && dfs(w)){
            match[v] = u;
            match[u] = v;
            return 1;
        }
    }
    return 0;
}

int bi_matching(){
    int res = 0;
    memset(match, -1, sizeof(match));
    rep(v,V) if(match[v] < 0){
        memset(used, 0, sizeof(used));
        if(dfs(v)) res++;
    }
    return res;
}
```

3.2.3 最小費用流

$O(FE \log V)$

```
struct edge{ int to, cap, cost, rev;};

int V;
vector<edge> G[MAX_V];
int h[MAX_V];
int dist[MAX_V];
int prevv[MAX_V], preve[MAX_V];

void add_edge(int from, int to, int cap, int cost){
    G[from].push_back((edge){to, cap, cost, int(G[to].size())});
    G[to].push_back((edge){from, 0, -cost, int(G[from].size() - 1)});
}

int min_cost_flow(int s, int t, int f){
    int res = 0;
    fill(h, h + V, 0);
    while(f > 0){
        priority_queue<pii, vector<pii>, greater<pii> > que;
        fill(dist, dist + V, inf);
        dist[s] = 0;
        que.push(pii(0, s));
        while(!que.empty()){
            pii p = que.top(); que.pop();
```

```cpp
            int v = p.second;
            if(dist[v] < p.first) continue;
            rep(i,G[v].size()){
                edge &e = G[v][i];
                if(e.cap > 0 && dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]){
                    dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
                    prevv[e.to] = v;
                    preve[e.to] = i;
                    que.push(pii(dist[e.to], e.to));
                }
            }
        }
        if(dist[t] == inf) return -1;
        rep(v,V) h[v] += dist[v];
        int d = f;
        for(int v = t; v != s; v = prevv[v])
            d = min(d, G[prevv[v]][preve[v]].cap);
        f -= d;
        res += d * h[t];
        for(int v = t; v != s; v = prevv[v]){
            edge &e = G[prevv[v]][preve[v]];
            e.cap -= d;
            G[v][e.rev].cap += d;
        }
    }
    return res;
}
```

3.3 木

3.3.1 木の直径

ある点 (どこでもよい) から一番遠い点 a を求める. 点 a から一番遠い点までの距離がその木の直径になる.

3.3.2 最小シュタイナー木

$O(4^{|T|}V)$

g は無向グラフの隣接行列. T は使いたい頂点の集合.

```cpp
int minimum_steiner_tree(vi &T, vvi &g){
    int n = g.size(), t = T.size();
    if(t <= 1) return 0;
    vvi d(g);    // all-pair shortest
    rep(k,n)rep(i,n)rep(j,n)    //Warshall Floyd
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

    int opt[1 << t][n];
    rep(S,1<<t) rep(x,n)
        opt[S][x] = INF;

    rep(p,t) rep(q,n)   // trivial case
        opt[1 << p][q] = d[T[p]][q];

    repi(S,1,1<<t){   // DP step
        if(!(S & (S-1))) continue;
        rep(p,n) rep(E,S)
            if((E | S) == S)
                opt[S][p] = min(opt[S][p], opt[E][p] + opt[S-E][p]);
        rep(p,n) rep(q,n)
            opt[S][p] = min(opt[S][p], opt[S][q] + d[p][q]);
    }

    int ans = INF;
    rep(S,1<<t) rep(q,n)
        ans = min(ans, opt[S][q] + opt[((1<<t)-1)-S][q]);
    return ans;
}
```

3.4 包除原理

3.4.1 彩色数

$O(2^V V)$

N[i] := i と隣接する頂点の集合 (i も含む)

```cpp
const int MAX_V=16;
const int mod = 10009;
int N[MAX_V], I[1<<MAX_V], V;
inline int mpow(int a, int k){ return k==0? 1: k%2? a*mpow(a,k-1)%mod: mpow(a*a%mod,k
    /2);}

bool can(int k){
    int res = 0;
    rep(S, 1<<V){
        if(__builtin_popcountll(S)%2) res -= mpow(I[S], k);
        else res += mpow(I[S],k);
    }
    return (res%mod+mod)%mod;
}

int color_number(){
    memset(I, 0, sizeof(I));
    I[0] = 1;
    repi(S,1,1<<V){
        int v = 0;
        while(!(S&(1<<v))) v++;
        I[S] = I[S-(1<<v)] + I[S&(~N[v])];
    }
    int lb = 0, ub = V, mid;
    while(ub-lb>1){
        mid = (lb+ub)/2;
        if(can(mid)) ub = mid;
        else lb = mid;
    }
    return ub;
}
```

4 数学

4.1 整数

4.1.1 拡張ユークリッドの互除法

$O(\log min(a,b))$

$ax + by = gcd(a,b)$ を求める. 解がある場合は 1 を返す.

```cpp
// a x + b y = gcd(a, b)
ll extgcd(ll a, ll b, ll &x, ll &y) {
    ll g = a; x = 1; y = 0;
    if (b != 0) g = extgcd(b, a % b, y, x), y -= (a / b) * x;
    return g; // 1なら解あり
```

```
}
```

4.1.2 逆元

mod_inverse()	*gen_mod_inv*()
$O(\log n)$	$O(n)$
	extgcd()

gen_mod_inv() は N 未満の全ての数の逆元を生成する.

```
ll mod_inverse(ll a, ll m){
    ll x, y;
    if(extgcd(a, m, x, y) != 1) return 0; // unsolvable
    return (m + x % m) % m;
}
ll mod_inv[MAX];
void gen_mod_inv(int n, ll mod){
    repi(i,2,n) mod_inv[i] = mod_inv[mod%i] * (mod - mod / i) % mod;
}
```

4.1.3 冪剰余

$O(\log k)$

```
int pow_mod(int x, int k, int m) {
    int ret = 1;
    for(x%=m; k>0; x=1LL*x*x%m,k>>=1) if(k&1) ret = 1LL*ret*x%m;
    return ret;
}
```

4.1.4 階乗 ($n!$ mod m)

gen_fact()	*mod_fact*()
$O(m)$	$O(\log_m n)$

m は素数.

```
ll fact[MAX];
void gen_fact(ll m){
    fact[0] = 1;
    repi(i,1,m) fact[i] = fact[i-1] * i % m;
}
ll mod_fact(ll n, ll m, ll& e){
    e = 0;
    if(!n) return 1;
    ll res = mod_fact(n / m, m, e);
    e += n / m;
    if(n / m % 2) return res * (m - fact[n % m]) % m;
    return res * fact[n % m] % m;
}
```

4.1.5 組み合わせ (${}_nC_k$ mod m)

$O(\log n)$
mod_fact() と *mod_inverse*() が必要.

```
/* nCk mod m */
ll mod_combi(ll n, ll k, ll m){
    if(n < 0 || k < 0 || n < k) return 0;
    ll e1, e2, e3;
    ll a1 = mod_fact(n, m, e1), a2 = mod_fact(k, m, e2), a3 = mod_fact(n - k, m, e3);
    if(e1 > e2 + e3) return 0; // m で割り切れる
    return a1 * mod_inverse(a2 * a3 % m, m) % m;
}
```

4.1.6 カタラン数

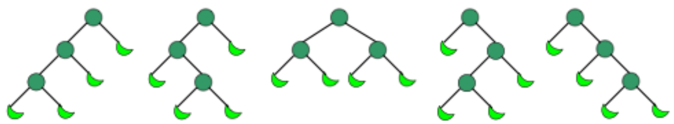
$n \leq 16$ 程度が限度. $n \geq 1$ について以下が成り立つ.

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n-1}$$

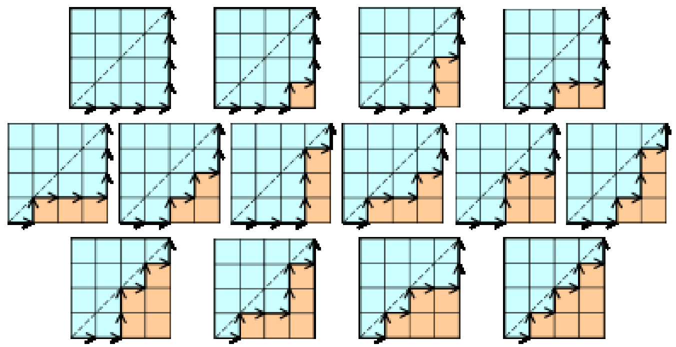
n が十分大きいとき, カタラン数は以下に近似できる.

$$C_n = \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

() を正しく並べる方法, 二分木, 格子状の経路の数え上げ, 平面グラフの交差などに使われる.
$C_3 = 5$



$C_4 = 14$



4.2 多項式

FFT は基本定数重めなので TLE に注意する.

4.2.1 FFT(complex)

$O(N \log N)$
複素数を用いた FFT. 変換する vector のサイズは 2 の冪乗にすること.

```
typedef complex<double> cd;
vector<cd> fft(vector<cd> f, bool inv){
    int n, N = f.size();
    for(n=0;;n++) if(N == (1<<n)) break;
    rep(m,N){
        int m2 = 0;
        rep(i,n) if(m&(1<<i)) m2 |= (1<<(n-1-i));
        if(m < m2) swap(f[m], f[m2]);
    }

    for(int t=1;t<N;t*=2){
        double theta = acos(-1.0) / t;
        cd w(cos(theta), sin(theta));
        if(inv) w = cd(cos(theta), -sin(theta));
        for(int i=0;i<N;i+=2*t){
            cd power(1.0, 0.0);
            rep(j,t){
                cd tmp1 = f[i+j] + f[i+t+j] * power;
                cd tmp2 = f[i+j] - f[i+t+j] * power;
                f[i+j] = tmp1;
                f[i+t+j] = tmp2;
                power = power * w;
            }
        }
    }
    if(inv) rep(i,N) f[i] /= N;
    return f;
}
```

4.2.2 FFT(modulo)

$O(N \log N)$
剰余環を用いた FFT(FMT). 変換する vector のサイズは 2 の冪乗にすること. mod は $a*2^e+1$ の形.

```
const int mod = 7*17*(1<<23)+1;
vector<int> fmt(vector<int> f, bool inv){
    int e, N = f.size();
    assert((N&(N-1))==0 and "f.size() must be power of 2");
    for(e=0;;e++) if(N == (1<<e)) break;
    rep(m,N){
        int m2 = 0;
        rep(i,e) if(m&(1<<i)) m2 |= (1<<(e-1-i));
        if(m < m2) swap(f[m], f[m2]);
    }
    for(int t=1; t<N; t*=2){
        int r = pow_mod(3,(mod-1)/(t*2),mod);
        if(inv) r = mod_inverse(r,mod);
        for(int i=0; i<N; i+=2*t){
            int power = 1;
            rep(j,t){
                int x = f[i+j], y = 1LL*f[i+t+j]*power%mod;
                f[i+j] = (x+y)%mod;
                f[i+t+j] = (x-y+mod)%mod;
                power = 1LL*power*r%mod;
            }
        }
    }
    if(inv) for(int i=0,ni=mod_inverse(N,mod);i<N;i++) f[i] = 1LL*f[i]*ni%mod;
    return f;
}
```

4.2.3 積 (FMT)

$O(N \log N)$
poly_mul() が必要.

```
vector<int> poly_mul(vector<int> f, vector<int> g){
    int N = max(f.size(),g.size())*2;
    f.resize(N); g.resize(N);
    f = fmt(f,0); g = fmt(g,0);
    rep(i,N) f[i] = 1LL*f[i]*g[i]%mod;
    f = fmt(f,1);
    return f;
}
```

4.2.4 逆元 (FMT)

$O(N \log N)$
extgcd(), *mod_inverse*(), *poly_mul*(), *fmt*() が必要.

```
vector<int> poly_inv(vector<int> f){
    int N = f.size();
    vector<int> r(1,mod_inverse(f[0],mod));
    for(int k = 2; k <= N; k <<= 1){
        vector<int> nr = poly_mul(poly_mul(r,r), vector<int>(f.begin(),f.begin()+k));
        nr.resize(k);
        rep(i,k/2) {
            nr[i] = (2*r[i]-nr[i]+mod)%mod;
            nr[i+k/2] = (mod-nr[i+k/2])%mod;
        }
        r = nr;
    }
    return r;
}
```

4.2.5 平方根 (FMT)

$O(NlogN)$
extgcd(), *mod_inverse*(), *poly_inv*(), *poly_mul*(), *fmt*() が必要.

```
const int inv2 = (mod+1)/2;
vector<int> poly_sqrt(vector<int> f){
    int N = f.size();
    vector<int> s(1,1); // s[0] = sqrt(f[0])
    for(int k = 2; k <= N; k <<= 1){
        s.resize(k);
        vector<int> ns = poly_mul(poly_inv(s), vector<int>(f.begin(),f.begin()+k));
        ns.resize(k);
        rep(i,k) s[i] = 1LL*(s[i]+ns[i])*inv2%mod;
    }
    return s;
}
```

4.3 行列

C++11 だと array という名前では衝突するので arr にしている.

```
typedef double number;
typedef vector<number> arr;
typedef vector<arr> mat;
```

4.3.1 単位行列

$O(N)$

```
mat identity(int n) {
    mat A(n, arr(n));
    rep(i,n) A[i][i] = 1;
    return A;
}
```

4.3.2 積

arr*arr	mat*arr	mat*mat
$O(N)$	$O(N^2)$	$O(N^3)$

```
number inner_product(const arr &a, const arr &b) {
    number ans = 0;
    rep(i,a.size()) ans += a[i] * b[i];
    return ans;
}

arr mul(const mat &A, const arr &x) {
    arr y(A.size());
    rep(i,A.size()) rep(j,A[0].size())
        y[i] = A[i][j] * x[j];
    return y;
}

mat mul(const mat &A, const mat &B) {
    mat C(A.size(), arr(B[0].size()));
    rep(i,C.size()) rep(j,C[i].size()) rep(k,A[i].size())
        C[i][j] += A[i][k] * B[k][j];
    return C;
}
```

4.3.3 累乗

$O(N^3 \log e)$
単位行列と積 (mat*mat) が必要.

```
mat pow(const mat &A, int e) {
    return e == 0 ? identity(A.size())  :
    e % 2 == 0 ? pow(mul(A, A), e/2) : mul(A, pow(A, e-1));
}
```

4.3.4 線形方程式の解 (Givens 消去法)

$O(N^3)$

```
#define mkrot(x,y,c,s) {double r = sqrt(x*x+y*y); c = x/r; s = y/r;}
#define rot(x,y,c,s) {double u = c*x+s*y; double v = -s*x+c*y; x = u; y = v;}
arr givens(mat A, arr b){
    int n = b.size();
    rep(i,n) repi(j,i+1,n){
        double c, s;
        mkrot(A[i][i], A[j][i], c, s);
        rot(b[i], b[j], c, s);
        repi(k,i,n) rot(A[i][k],A[j][k],c,s);
    }
    repd(i,n-1,0){
        repi(j,i+1,n)
            b[i] -= A[i][j] * b[j];
        b[i] /= A[i][i];
    }
    return b;
}
```

4.3.5 トレース

$O(N)$

```
number trace(const mat &A) {
    number ans = 0;
    rep(i,A.size()) ans += A[i][i];
    return ans;
}
```