1 データ構造

Union-Find

```
1  struct UnionFind{
2      vector<int> dat;
3
4      UnionFind(int sz){
5          dat.assign(sz, -1);
6      }
7
8      bool unite(int x, int y){
9          x = root(x), y = root(y);
10         if(x == y) return(false);
11         if(dat[x] > dat[y]) swap(x, y);
12         dat[x] += dat[y];
13         dat[y] = x;
14         return(true);
15     }
16
17     int root(int k){
18         if(dat[k] < 0) return(k);
19         return(dat[k] = root(dat[k]));
20     }
21
22     int size(int k){
23         return(-dat[root(k)]);
24     }
25 };
```

Binary Indexed Tree

```
1  template< class T >
2  struct BinaryIndexedTree {
3      vector< T > dat;
4
5      BinaryIndexedTree(int sz) {
6          dat.assign(++sz, 0);
7      }
8
9      T sum(int k){
10         T ret = 0;
11         for(++k; k > 0; k -= k & -k) ret += data[k];
12         return ret;
13     }
14
15     void add(int k, T x){
16         for(++k; k < data.size(); k += k & -k) data[k] += x;
17     }
18 };
```

Segment Tree

```cpp
/*
!) 0-indexed
!) [a, b)に対する演算
!) merge, updateNodeを書く
*/
template<typename Monoid>
struct Segtree{
    int n;
    vector<Monoid> dat;
    Monoid m0; // データの初期化値
    Segtree(int sz, Monoid m0) : m0(m0){
        n = 1;
        while(n < sz) n *= 2;
        dat.assign(2*n-1, m0);
    }

    Monoid merge(Monoid a, Monoid b) // 区間をマージする二項演算
    void updateNode(int k, Monoid x) // 区間を操作する二項演算

    void update(int k, Monoid x){
        k += n-1;
        updateNode(k, x);
        while(k > 0) {
            k = (k-1)/2;
            dat[k] = merge(dat[k*2+1], dat[k*2+2]);
        }
    }
    Monoid query(int a, int b){
        Monoid L = m0, R = m0;
        int A = a+n-1;
        int B = b+n-1;
        while(A < B) {
            if((A&1) == 0) L = merge(L, dat[A++]);
            if((B&1) == 0) R = merge(dat[--B], R);
            A >>= 1;
            B >>= 1;
        }
        return merge(L, R);
    }
    Monoid operator[](const int &k) const { return dat[k+n-1]; }
};

//########## 例 ##########
// Range min (AOJ DSL_2_A)
Monoid merge(Monoid a, Monoid b){ return min(a,b); }
void updateNode(int k, Monoid x){ dat[k] = x; }
Segtree<LL> a(n, (1LL<<31)-1);

// Range add (AOJ DSL_2_B)
Monoid merge(Monoid a, Monoid b){ return a + b; }
void updateNode(int k, Monoid x){ dat[k] += x; }
Segtree<LL> a(n, 0);
```

遅延伝搬 Segment Tree

```
1   /*
2   !) 0-indexed 遅延伝搬セグメント木
3   !) [a, b)に対する演算
4   !) M0, L0, merge, updateNode, propagateを書く
5   */
6   template <typename Monoid>
7   struct LazySegtree {
8       int n;
9       vector<Monoid> dat, lazy;
10
11      Monoid M0, L0 // データと遅延配列の初期化値 queryに合わせて選択する
12
13      LazySegtree(int sz, Monoid dat_init){
14          n = 1;
15          while (n < sz) n *= 2;
16          dat.assign(2*n-1, dat_init);
17          lazy.assign(2*n-1, L0);
18      }
19
20      Monoid merge(Monoid a, Monoid b) // 区間をマージする二項演算
21      void updateNode(int k, Monoid x) // 区間を操作する二項演算
22      void propagate(int k, int l, int r) // 遅延配列の伝搬のさせ方
23
24      void eval(int k, int l, int r) {
25          if(lazy[k] == L0) return;
26          propagate(k, l, r);
27          if(r-l > 1) {
28              updateNode(2*k+1, lazy[k]);
29              updateNode(2*k+2, lazy[k]);
30          }
31          lazy[k] = L0;
32      }
33
34      void update(int a, int b, Monoid x, int k, int l, int r) {
35          eval(k, l, r);
36          if (r <= a || b <= l) return;
37          if (a <= l && r <= b) {
38              updateNode(k, x);
39              eval(k, l, r);
40          }else{
41              update(a, b, x, k*2+1, l, (l+r)/2);
42              update(a, b, x, k*2+2, (l+r)/2, r);
43              dat[k] = merge(dat[2*k+1], dat[2*k+2]);
44          }
45      }
46
47      void update(int a, int b, Monoid x) {
48          update(a, b, x, 0, 0, n);
49      }
50
51      Monoid query(int a, int b, int k, int l, int r) {
52          eval(k, l, r);
53          if (r <= a || b <= l) return M0;
54          if (a <= l && r <= b) return dat[k];
55          Monoid L = query(a, b, k*2+1, l, (l+r)/2);
```

```
56          Monoid R = query(a, b, k*2+2, (l+r)/2, r);
57          return merge(L, R);
58      }
59
60      Monoid query(int a, int b){
61          return query(a, b, 0, 0, n);
62      }
63 };
64
65 //########## 例 ##########
66 // Range update - min (AOJ DSL_2_F)
67 Monoid M0 = LLINF, L0 = LLINF;
68 Monoid merge(Monoid a, Monoid b){ return min(a, b); }
69 void updateNode(int k, Monoid x){ lazy[k] = x; }
70 void propagate(int k, int l, int r){ dat[k] = lazy[k]; }
71 LazySegtree<LL> seg(n+1, (1LL<<31)-1);
72
73 // Range update - sum (AOJ DSL_2_I)
74 Monoid M0 = 0, L0 = LLINF;
75 Monoid merge(Monoid a, Monoid b){ return a + b; }
76 void updateNode(int k, Monoid x){ lazy[k] = x; }
77 void propagate(int k, int l, int r){ dat[k] = lazy[k]*(r-l); }
78 LazySegtree<LL> seg(n+1, 0);
79
80 // Range add - min (AOJ DSL_2_H)
81 Monoid M0 = LLINF, L0 = 0;
82 Monoid merge(Monoid a, Monoid b){ return min(a, b); }
83 void updateNode(int k, Monoid x){ lazy[k] += x; }
84 void propagate(int k, int l, int r){ dat[k] += lazy[k]; }
85 LazySegtree<LL> seg(n+1, 0);
86
87 // Range add - sum (AOJ DSL_2_G)
88 Monoid M0 = 0, L0 = 0;
89 Monoid merge(Monoid a, Monoid b){ return a + b; }
90 void updateNode(int k, Monoid x){ lazy[k] += x; }
91 void propagate(int k, int l, int r){ dat[k] += lazy[k]*(r-l); }
92 LazySegtree<LL> seg(n+1, 0);
```

2 グラフ

Grid 上での BFS

```
1  int W, H;
2  vector<vector<char>> s;
3  vector<vector<int>> cost;
4  int bfs(){
5      int dx[] = {0, 1, 0, -1}, dy[] = {1, 0, -1, 0};
6      queue<pair<int, int>> que;
7      que.push(make_pair(0, 0));
8      cost[0][0] = 0;
9
10     while(!que.empty()) {
11         pair<int, int> p = que.front();
12         que.pop();
13         if(p == make_pair(H-1, W-1)){
14             // ゴールに到達
15             return cost[p.first][p.second];
16         }
17         for(int i = 0; i < 4; i++) {
18             int ny = p.first + dy[i], nx = p.second + dx[i];
19             if(nx < 0 || ny < 0 || nx >= W || ny >= H) continue;
20             if(s[ny][nx] == '#') continue;
21             if(cost[ny][nx] != -1) continue;
22
23             cost[ny][nx] = cost[p.first][p.second] + 1;
24             que.push(make_pair(ny, nx));
25         }
26     }
27     return -1;
28 }
```

Dijkstra 法 (単一始点最短経路)

```
1  struct edge{
2      int to, cost;
3  };
4  using WeightedGraph = vector<vector<edge>>;
5
6  vector<int> dijkstra(WeightedGraph &G, int st){
7      vector<int> dist(G.size(), INF);
8      using pi = pair<int, int>;
9      priority_queue<pi, vector<pi>, greater<pi>> que;
10     dist[st] = 0;
11     que.push(mp(dist[st], st));
12     while(!que.empty()){
13         int cost, idx;
14         tie(cost, idx) = que.top(); que.pop();
15         if(dist[idx] < cost) continue;
16         for(auto &e: G[idx]){
17             if(dist[e.to] <= cost+e.cost) continue;
18             dist[e.to] = cost+e.cost;
19             que.push(mp(dist[e.to], e.to));
20         }
```

```
21      }
22      return dist;
23  }
```

Bellman-Ford 法 (負路あり単一始点最短経路)

```
1   struct edge{
2       int src, to, cost;
3   };
4   using Edges = vector<edge>;
5
6   vector<int> bellman_ford(Edges &E, int V, int st){
7       vector<int> dist(V, INF);
8       dist[st] = 0;
9       rep(i, V-1){
10          for(auto &e: E){
11              if(dist[e.src] == INF) continue;
12              dist[e.to] = min(dist[e.to], dist[e.src]+e.cost);
13          }
14      }
15      for(auto &e: E){
16          if(dist[e.src] == INF) continue;
17          if(dist[e.to] > dist[e.src]+e.cost){
18              // 負閉路が存在
19              return vector<int>();
20          }
21      }
22      return dist;
23  }
```

Warshall-Floyd 法 (全点対間最短経路)

```
1   using Graph = vector<vector<int>>;
2
3   void warshall_floyd(Graph &G){
4       int V = G.size();
5       rep(k, V)rep(i, V)rep(j, V){
6           if(G[i][k] == INF || G[k][j] == INF) continue;
7           G[i][j] = min(G[i][j], G[i][k]+G[k][j]);
8       }
9       // G[i][i] < 0 が存在 <=> 負閉路が存在
10  }
```

Kruskal 法 (最小全域木)

```
1   // UnionFind が必要
2
3   struct edge{
4       int src, to, cost;
5   };
6   using Edges = vector<edge>;
7
8   int kruskal(Edges &E, int V)
9   {
10      sort(all(E), [](const edge &a, const edge &b)
11      {
```

```
12        return (a.cost < b.cost);
13    });
14    UnionFind tree(V);
15    int res = 0;
16    for(auto &e : E) {
17        if(tree.unite(e.src, e.to)) res += e.cost;
18    }
19    return (res);
20 }
```

トポロジカルソート

```
1  struct edge{
2      int to, cost;
3  };
4  using WeightedGraph = vector<vector<edge>>;
5
6  vector<int> tsort(WeightedGraph &G){
7      vector<int> tsorted;
8      vector<int> used(G.size(), 0);
9      bool f = false;
10     function<void(int)> dfs = [&](int u){
11         if(used[u] > 0){
12             if(used[u] == 1) f = true;
13             return;
14         }
15         used[u] = 1;
16         for(auto &e : G[u]) dfs(e.to);
17         used[u] = 2;
18         tsorted.pb(u);
19     };
20     rep(i, G.size()) dfs(i);
21     if(f){
22         // 閉 路 が 存 在
23         return vector<int>();
24     }
25     reverse(all(tsorted));
26     return tsorted;
27 }
```

Dinic 法 (最大流)

```
1  template<typename flow_t>
2  struct Dinic{
3      const flow_t INF_flow_t = INF;   // WRITE HERE
4
5      struct edge{
6          int to;
7          flow_t cap;
8          int rev;
9      };
10     using WeightedGraph = vector<vector<edge>>;
11     int V;
12     WeightedGraph G;
13     vector<int> itr, level;
14
```

```
15        Dinic(int V) : V(V) { G.assign(V, vector<edge>()); }
16
17        void add_edge(int from, int to, int cap) {
18            G[from].push_back((edge){to, cap, (int)G[to].size()});
19            G[to].push_back((edge){from, 0, (int)G[from].size()-1});
20        }
21
22        void bfs(int s) {
23            level.assign(V, -1);
24            queue<int> que;
25            level[s] = 0;
26            que.push(s);
27            while (!que.empty()) {
28                int v = que.front(); que.pop();
29                for(auto &&e: G[v]){
30                    if (e.cap > 0 && level[e.to] < 0) {
31                        level[e.to] = level[v] + 1;
32                        que.push(e.to);
33                    }
34                }
35            }
36        }
37
38        flow_t dfs(int v, int t, flow_t f) {
39            if(v == t) return f;
40            for(int &i = itr[v]; i < (int)G[v].size(); i++) {
41                edge &e = G[v][i];
42                if (e.cap > 0 && level[v] < level[e.to]) {
43                    flow_t d = dfs(e.to, t, min(f, e.cap));
44                    if (d > 0) {
45                        e.cap -= d;
46                        G[e.to][e.rev].cap += d;
47                        return d;
48                    }
49                }
50            }
51            return 0;
52        }
53
54        flow_t max_flow(int s, int t) {
55            flow_t res = 0, f;
56            while(bfs(s), level[t] >= 0) {
57                itr.assign(V, 0);
58                while((f = dfs(s, t, INF_flow_t)) > 0) res += f;
59            }
60            return res;
61        }
62    };
63
64    // 最小流量制限付き最大流
65    // 各辺に[lb, ub]の容量の辺を張る
66    template<typename flow_t>
67    struct DinicWithLowerBound{
68        Dinic<flow_t> flow;
69        int S, T;
70        flow_t sum_lb;
71
```

8

```
72        DinicWithLowerBound(int V) : flow(V+2), S(V), T(V+1), sum_lb(0) {}
73
74        void add_edge(int from, int to, flow_t lb, flow_t ub) {
75            flow.add_edge(from, to, ub-lb);
76            flow.add_edge(S, to, lb);
77            flow.add_edge(from, T, lb);
78            sum_lb += lb;
79        }
80
81        flow_t max_flow(int s, int t) {
82            auto a = flow.max_flow(S, T);
83            auto b = flow.max_flow(s, T);
84            auto c = flow.max_flow(S, t);
85            auto d = flow.max_flow(s, t);
86            return (b == c && a + b == sum_lb) ? b+d : -1;
87        }
88  };
```

二部マッチング

```
1   struct BipartiteMatching {
2       using Graph = vector<vector<int>>;
3       Graph G;
4       vector<int> match, alive, used;
5       int timestamp;
6
7       BipartiteMatching(int n) : G(n), alive(n, 1),
8                                  used(n, 0), match(n, -1), timestamp(0) {}
9
10      void add_edge(int u, int v) {
11          G[u].push_back(v);
12          G[v].push_back(u);
13      }
14
15      int dfs(int idx) {
16          used[idx] = timestamp;
17          for(auto &&to : G[idx]) {
18              int w = match[to];
19              if(alive[to] == 0) continue;
20              if(w < 0 || (used[w] != timestamp && dfs(w))) {
21                  match[idx] = to;
22                  match[to] = idx;
23                  return 1;
24              }
25          }
26          return 0;
27      }
28
29      int bipartite_matching() {
30          int res = 0;
31          for(int i = 0; i < G.size(); i++) {
32              if(alive[i] == 0) continue;
33              if(match[i] == -1) {
34                  ++timestamp;
35                  res += dfs(i);
36              }
37          }
```

```
38        return res;
39     }
40
41     void output() {
42         for(int i = 0; i < G.size(); i++) {
43             if(i < match[i]) {
44                 cout << i << "-" << match[i] << endl;
45             }
46         }
47     }
48 };
```

3 木

木の直径

```
1  struct edge{
2      int to, cost;
3  };
4  using WeightedGraph = vector<vector<edge>>;
5  using pi = pair<int, int>;
6
7  pi dfs(WeightedGraph &G, int idx, int src){
8      pi res(0, idx);
9      for(auto &e : G[idx]) {
10         if(e.to == src) continue;
11         pi cost = dfs(G, e.to, idx);
12         cost.first += e.cost;
13         res = max(res, cost);
14     }
15     return res;
16 }
17
18 int tree_diameter(WeightedGraph &G)
19 {
20     auto far = dfs(G, 0, -1);
21     auto res = dfs(G, far.second, -1);
22     return (res.first);
23 }
```

4 数学

GCD・LCM

```
1  LL gcd(LL a, LL b){
2      if(a < b) swap(a, b);
3      if(b == 0) return a;
4      return gcd(b, a%b);
5  }
6
7  LL lcm(LL a, LL b){
8      return a*b/gcd(a,b);
9  }
```

ModInt

```
1   template<int mod>
2   struct ModInt {
3       int x;
4
5       ModInt() : x(0) {}
6       ModInt(long long y) : x(y >= 0 ? y % mod : (mod - (-y) % mod) % mod) {}
7
8       ModInt &operator+=(const ModInt &p) {
9           if((x += p.x) >= mod) x -= mod;
10          return *this;
11      }
12      ModInt &operator-=(const ModInt &p) {
13          if((x += mod - p.x) >= mod) x -= mod;
14          return *this;
15      }
16      ModInt &operator*=(const ModInt &p) {
17          x = (int)(1LL * x * p.x % mod);
18          return *this;
19      }
20      ModInt &operator/=(const ModInt &p) {
21          *this *= p.inverse();
22          return *this;
23      }
24
25      ModInt operator-() const { return ModInt(-x); }
26      ModInt operator+(const ModInt &p) const { return ModInt(*this) += p; }
27      ModInt operator-(const ModInt &p) const { return ModInt(*this) -= p; }
28      ModInt operator*(const ModInt &p) const { return ModInt(*this) *= p; }
29      ModInt operator/(const ModInt &p) const { return ModInt(*this) /= p; }
30
31      bool operator==(const ModInt &p) const { return x == p.x; }
32      bool operator!=(const ModInt &p) const { return x != p.x; }
33
34      ModInt inverse() const{
35          int a = x, b = mod, u = 1, v = 0, t;
36          while(b > 0) {
37              t = a / b;
38              a -= t * b;
39              swap(a, b);
```

```
40              u -= t * v;
41              swap(u, v);
42          }
43          return ModInt(u);
44      }
45
46      ModInt pow(int e){
47          long long a = 1, p = x;
48          while(e > 0) {
49              if(e%2 == 0) {p = (p*p) % mod; e /= 2;}
50              else {a = (a*p) % mod; e--;}
51          }
52          return ModInt(a);
53      }
54
55      friend ostream &operator<<(ostream &os, const ModInt<mod> &p) {
56          return os << p.x;
57      }
58      friend istream &operator>>(istream &is, ModInt<mod> &a) {
59          long long x;
60          is >> x;
61          a = ModInt<mod>(x);
62          return (is);
63      }
64  };
65
66  const int mod = 1e9 + 7;
67  using modint = ModInt<mod>;
```

FFT

```
1   // https://ei1333.github.io/luzhiled/snippets/math/fast-fourier-transform.html
2   namespace FastFourierTransform {
3       using real = double;
4
5       struct C {
6           real x, y;
7           C() : x(0), y(0) {}
8           C(real x, real y) : x(x), y(y) {}
9           inline C operator+(const C &c) const { return C(x + c.x, y + c.y); }
10          inline C operator-(const C &c) const { return C(x - c.x, y - c.y); }
11          inline C operator*(const C &c) const {
12              return C(x * c.x - y * c.y, x * c.y + y * c.x);
13          }
14          inline C conj() const { return C(x, -y); }
15      };
16
17      const real PI = acosl(-1);
18      int base = 1;
19      vector< C > rts = { {0, 0},
20                          {1, 0} };
21      vector< int > rev = {0, 1};
22
23
24      void ensure_base(int nbase) {
25          if(nbase <= base) return;
26          rev.resize(1 << nbase);
```

```
27          rts.resize(1 << nbase);
28          for(int i = 0; i < (1 << nbase); i++) {
29              rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
30          }
31          while(base < nbase) {
32          real angle = PI * 2.0 / (1 << (base + 1));
33          for(int i = 1 << (base - 1); i < (1 << base); i++) {
34              rts[i << 1] = rts[i];
35              real angle_i = angle * (2 * i + 1 - (1 << base));
36              rts[(i << 1) + 1] = C(cos(angle_i), sin(angle_i));
37          }
38              ++base;
39          }
40      }
41
42      void fft(vector< C > &a, int n) {
43          assert((n & (n - 1)) == 0);
44          int zeros = __builtin_ctz(n);
45          ensure_base(zeros);
46          int shift = base - zeros;
47          for(int i = 0; i < n; i++) {
48              if(i < (rev[i] >> shift)) {
49                  swap(a[i], a[rev[i] >> shift]);
50              }
51          }
52          for(int k = 1; k < n; k <<= 1) {
53              for(int i = 0; i < n; i += 2 * k) {
54                  for(int j = 0; j < k; j++) {
55                  C z = a[i + j + k] * rts[j + k];
56                  a[i + j + k] = a[i + j] - z;
57                  a[i + j] = a[i + j] + z;
58                  }
59              }
60          }
61      }
62
63      vector< int64_t > multiply(const vector< int > &a, const vector< int > &b) {
64          int need = (int) a.size() + (int) b.size() - 1;
65          int nbase = 1;
66          while((1 << nbase) < need) nbase++;
67          ensure_base(nbase);
68          int sz = 1 << nbase;
69          vector< C > fa(sz);
70          for(int i = 0; i < sz; i++) {
71              int x = (i < (int) a.size() ? a[i] : 0);
72              int y = (i < (int) b.size() ? b[i] : 0);
73              fa[i] = C(x, y);
74          }
75          fft(fa, sz);
76          C r(0, -0.25 / (sz >> 1)), s(0, 1), t(0.5, 0);
77          for(int i = 0; i <= (sz >> 1); i++) {
78              int j = (sz - i) & (sz - 1);
79              C z = (fa[j] * fa[j] - (fa[i] * fa[i]).conj()) * r;
80              fa[j] = (fa[i] * fa[i] - (fa[j] * fa[j]).conj()) * r;
81              fa[i] = z;
82          }
83          for(int i = 0; i < (sz >> 1); i++) {
```

```
84            C A0 = (fa[i] + fa[i + (sz >> 1)]) * t;
85            C A1 = (fa[i] - fa[i + (sz >> 1)]) * t * rts[(sz >> 1) + i];
86            fa[i] = A0 + A1 * s;
87          }
88        fft(fa, sz >> 1);
89        vector< int64_t > ret(need);
90        for(int i = 0; i < need; i++) {
91          ret[i] = llround(i & 1 ? fa[i >> 1].y : fa[i >> 1].x);
92        }
93        return ret;
94      }
95  };
96  // auto C = FastFourierTransform::multiply(A, B);
```

Lagrange 補間

```
1   template< typename T >
2   T lagrange_polynomial(const vector< T > &y, int64_t t) {
3     int N = y.size() - 1;
4     Combination< T > comb(N);
5     if(t <= N) return y[t];
6     T ret(0);
7     vector< T > dp(N + 1, 1), pd(N + 1, 1);
8     for(int i = 0; i < N; i++) dp[i + 1] = dp[i] * (t - i);
9     for(int i = N; i > 0; i--) pd[i - 1] = pd[i] * (t - i);
10    for(int i = 0; i <= N; i++) {
11      T tmp = y[i] * dp[i] * pd[i] * comb.rfact(i) * comb.rfact(N - i);
12      if((N - i) & 1) ret -= tmp;
13      else ret += tmp;
14    }
15    return ret;
16  }
17
18  int main() {
19    int N, T;
20    cin >> N;
21    vector< modint > A(N + 1);
22    cin >> A;
23    cin >> T;
24    cout << lagrange_polynomial(A, T) << endl;
25  }
```

5   文字列

KMP 法

文字列 $S[0, i-1]$ の接頭辞と接尾辞が最大何文字一致しているかを記録した配列を $O(|S|)$ で構築する

```
1   vector<int> A(s.size()+1);
2   A[0] = -1;
3   int j = -1;
4   for (int i = 0; i < s.size(); i++) {
5       while (j >= 0 && s[i] != s[j]) j = A[j];
6       j++;
7       A[i+1] = j;
8   }
```

ローリングハッシュ

```cpp
struct RollingHash {
    using T = long long;
    using hash_type = pair<T, T>;
    const T mod1 = 1000000007, mod2 = 1000000009;
    T base1, base2;
    vector<T> hash1, hash2, pow1, pow2;

    RollingHash(const string &s){
        int n = s.size();
        std::random_device rnd;
        std::mt19937 mt(rnd());
        base1 = mt()%(mod1-2)+2;
        base2 = mt()%(mod2-2)+2;

        hash1.assign(n+1,0);
        hash2.assign(n+1,0);
        pow1.assign(n+1,1);
        pow2.assign(n+1,1);
        for(int i=0; i<n; i++){
            hash1[i+1] = (hash1[i]*base1+s[i])%mod1;
            hash2[i+1] = (hash2[i]*base2+s[i])%mod2;
            pow1[i+1]=pow1[i]*base1%mod1;
            pow2[i+1]=pow2[i]*base2%mod2;
        }
    }

    // [l, r)
    hash_type get(int l,int r){
        T res1 = ((hash1[r] - hash1[l]*pow1[r-l]) % mod1 + mod1) % mod1;
        T res2 = ((hash2[r] - hash2[l]*pow2[r-l]) % mod2 + mod2) % mod2;
        return make_pair(res1, res2);
    }

    hash_type concat(hash_type h1, hash_type h2, int h2len) {
        T res1 = (h1.first*pow1[h2len] + h2.first) % mod1;
        T res2 = (h1.second*pow2[h2len] + h2.second) % mod2;
        return make_pair(res1, res2);
    }
    // LCPは長さで二分探索する
};
```

6 テクニック

座標圧縮

```cpp
vector<int> unzip = a;
map<int, int> zip;
sort(all(unzip));
unzip.erase(unique(all(unzip)), unzip.end());
for(int i=0; i<unzip.size(); i++) zip[unzip[i]] = i;
```

スライド最小値

$[i-k, i]$ の最小値を格納した vector を返す

```cpp
vector<int> slide_min(vector<int> &a, int k){
    deque<int> deq;
    vector<int> b;
    rep(i, a.size()){
        // maxはここの不等号の向きを変える
        while(!deq.empty() && a[deq.back()] >= a[i]) deq.pop_back();
        deq.push_back(i);
        b.push_back(a[deq.front()]);
        if(i-k+1 >= 0 && deq.front() == i-k+1) deq.pop_front();
    }
    return b;
}
```