```
T operator + (T a, int x) {return {(a[0] + x) % MOD[0], (a[1] + x) % MOD[1]};}
T operator - (T a, int x) {return {(a[0] - x + MOD[0]) % MOD[0], (a[1] - x + MOD[1]) % MOD[1]};}
T operator * (T a, int x) {return {(int)((long long) a[0] * x % MOD[0]), (int)((long long) a[1] * x %
MOD[1])};}
T operator + (T a, T x) {return {(a[0] + x[0]) % MOD[0], (a[1] + x[1]) % MOD[1]};}
T operator - (T a, T x) {return {(a[0] - x[0] + MOD[0]) % MOD[0], (a[1] - x[1] + MOD[1]) %
MOD[1]};}
T operator * (T a, T x) {return {(int)((long long) a[0] * x[0] % MOD[0]), (int)((long long) a[1] * x[1]
% MOD[1])};}
ostream& operator << (ostream& os, T hash) {return os << "(" << hash[0] << ", " << hash[1] << ")";}

T pw[N], ipw[N];
void prec() {
 pw[0] =  {1, 1};
 for (int i = 1; i < N; i++) {
  pw[i] = pw[i - 1] * p;
 }
 ipw[0] =  {1, 1};
 T ip = {power(p[0], MOD[0] - 2, MOD[0]), power(p[1], MOD[1] - 2, MOD[1])};
 for (int i = 1; i < N; i++) {
  ipw[i] = ipw[i - 1] * ip;
 }
}
struct Hashing {
 int n;
 string s; // 1 - indexed
 vector<array<T, 2>> t; // (normal, rev) hash
 array<T, 2> merge(array<T, 2> l, array<T, 2> r) {
  l[0] = l[0] + r[0];
  l[1] = l[1] + r[1];
  return l;
 }
 void build(int node, int b, int e) {
  if (b == e) {
   t[node][0] = pw[b] * s[b];
   t[node][1] = pw[n - b + 1] * s[b];
   return;
  }
  int mid = (b + e) >> 1, l = node << 1, r = l | 1;
  build(l, b, mid);
  build(r, mid + 1, e);
  t[node] = merge(t[l], t[r]);
 }
 void upd(int node, int b, int e, int i, char x) {
  if (b > i || e < i) return;
  if (b == e && b == i) {
   t[node][0] = pw[b] * x;
   t[node][1] = pw[n - b + 1] * x;
   return;
  }
```