```
  int mid = (b + e) >> 1, l = node << 1, r = l | 1;
  upd(l, b, mid, i, x);
  upd(r, mid + 1, e, i, x);
  t[node] = merge(t[l], t[r]);
 }
 array<T, 2> query(int node, int b, int e, int i, int j) {
  if (b > j || e < i) return {T({0, 0}), T({0, 0})};
  if (b >= i && e <= j) return t[node];
  int mid = (b + e) >> 1, l = node << 1, r = l | 1;
  return merge(query(l, b, mid, i, j), query(r, mid + 1, e, i, j));
 }
 Hashing() {}
 Hashing(string _s) {
  n = _s.size();
  s = "." + _s;
  t.resize(4 * n + 1);
  build(1, 1, n);
 }
 void upd(int i, char c) {
  upd(1, 1, n, i, c);
  s[i] = c;
 }
 T get_hash(int l, int r) { // 1 - indexed
  return query(1, 1, n, l, r)[0] * ipw[l - 1];
 }
 T rev_hash(int l, int r) { // 1 - indexed
  return query(1, 1, n, l, r)[1] * ipw[n - r];
 }
 T get_hash() {
  return get_hash(1, n);
 }
 bool is_palindrome(int l, int r) {
  return get_hash(l, r) == rev_hash(l, r);
 }
};
```