```
    Hs.second = (suffix[l].second - (suffix[r + 1].second * po[r - l + 1].second % mod2) + mod2) % mod2;
    return Hs;
  }
  pair<ll, ll> concat(pair<ll, ll> &hash1, pair<ll, ll> &hash2, int len) //len = 2nd string size
  {
    return {((hash1.first * po[len].first) + hash2.first) % mod1, ((hash1.second * po[len].second) +
                                                                   hash2.second) % mod2};
  }
  void build(string &s)  {
    n = s.size();
    prefix.resize(n), suffix.resize(n);
    generatePrefixHash(s);
    // generateSuffixHash(s);
    if (!isCalPow) generatePower(), isCalPow = 1;
  }
} Hash;

void solve() {
  int n, m;
  string s1, s2;
  s1 = "abcababababc", s2 = "abc";
  // cin >> s1 >> s2;
  n = s1.size();
  Hash.build(s1);

  pair<ll, ll> hashOfS2 = Hash.generateHash(s2);
  for (int i = 0; i + s2.size() <= s1.size(); i++) {
    if (Hash.getPrefixRangeHash(i, i + s2.size() - 1) == hashOfS2) {
      cout << i << "\n";
    }
  }
  return;
}
```

String Hashing With Updates and Reverse:

```
const int N = 1e5 + 9;

int power(long long n, long long k, const int mod) {
 int ans = 1 % mod;
 n %= mod;
 if (n < 0) n += mod;
 while (k) {
  if (k & 1) ans = (long long) ans * n % mod;
  n = (long long) n * n % mod;
  k >>= 1;
 }
 return ans;
}

using T = array<int, 2>;
const T MOD = {127657753, 987654319};
const T p = {137, 277};
```