Trie & Hash

- Trie pra pegar o max XOR:

```
const int ms = 1e5*32*2, sigma = 2;
using namespace std;

int trie[ms][sigma], terminal[ms], z;

string toBit(int x){
  string ret = "00000000000000000000000000000000";
  int bound = ceil(log2(x));
  for (int i = bound; i >= 0; i--){
      if (1<<i & x)
          ret[31-i] = '1';
  }
  return ret;
}

int get_id(char c){
  return c - '0';
}

void init(){
  memset(trie[0], -1, sizeof(trie[0]));
  z = 1;
}

void insert(string &p){
  int cur = 0;
  for (int i = 0; i < int(p.size()); i++){
      int id = get_id(p[i]);
      if (trie[cur][id] == -1){
          memset(trie[z], -1, sizeof(trie[z]));
          trie[cur][id] = z++;
      }
      cur = trie[cur][id];
  }
  terminal[cur]++;
}

int ans(string &p, int n){
  int cur = 0, num = 0;
  for (int i = 0; i < int(p.size()); i++){
      int id = get_id(p[i]);
      if (trie[cur][(id+1)%2] != -1){//!id
          cur = trie[cur][(id+1)%2];
          num += ((id+1)%2)<<(31-i);
      }else{// if (trie[cur][id] != -1){
          cur = trie[cur][id];
          num += id<<(31-i);
      }
  }
  return (n^num);
}

int count(string p){
  int cur = 0;
  for (int i = 0; i < int(p.size()); i++){
```

```
        int id = get_id(p[i]);
        if (trie[cur][id] == -1)
                return false;
        cur = trie[cur][id];
    }
    return terminal[cur];
}

bool remove(string &p, int pos, int cur){
    int id = get_id(p[pos]);
    if (pos == int(p.size())){
        if(terminal[cur] > 0)
                terminal[cur]--;

        return (terminal[cur] == 0);
    }
    bool posso_remover = true;
    if (trie[cur][id] != -1)
        posso_remover = remove(p, pos+1, trie[cur][id]);

    if (posso_remover){
        trie[cur][id] = -1;
        if(trie[cur][(id+1)%2] == -1){
                return true;
        }
    }
    return false;
}

int main(){
    ios::sync_with_stdio(false), cin.tie(0);
    init();
    string zero = toBit(0);
    insert(zero);

    int n; cin >> n;
    while(n--){
        string cmd; int num; cin >> cmd >> num;
        string cadeia = toBit(num);
        if (cmd == "+")
                insert(cadeia);
        else if (cmd == "-")
                remove(cadeia, 0, 0);
        else
                cout << ans(cadeia, num) << endl;
    }
    return 0;
}
```

- Pegar o maior prefix/sufix que nao seja a palavra inteira:

```
const ll mod = 1e9 + 5, base = 11, tamanho = 1e6 + 5;
string str;
vi abc = {3, 5, 7, 11, 13,
          17, 19, 23, 29, 31,
          37, 41, 43, 47, 53,
          59, 61, 67, 71, 73,
          79, 83, 89, 97, 101,
          103, 107, 109};
ll pot[tamanho], ahash[tamanho];
int get_id(int i){
  return abc[str[i]-'a'];
}
void build(){
  pot[0] = 1;
  ahash[0] = get_id(0);
  for (int i = 1; i < int(str.size()); i++){
      pot[i] = (pot[i-1] * base) % mod;
      ahash[i] = ((ahash[i-1]*base) + get_id(i)) % mod;
  }
}

ll getkey(int l, int r){
  ll res = ahash[r];
  if (l > 0) res = (res - ((pot[r-l+1] * ahash[l-1]) % mod) + mod) % mod;
  return res;
}

int main(){
  ios::sync_with_stdio(false), cin.tie(0);
  cin >> str; int n = str.size();
  build();
  set<ll> Hashes;
  vi sizes;
  for (int i = 0; i < n; i++){
      ll prefix = getkey(0, i), sufix = getkey(n-1-i, n-1);
      if (prefix == sufix){
          if (i+1 == n) continue;
          sizes.pb(i+1); Hashes.insert(prefix);
      }
  }
  sort(sizes.begin(), sizes.end(), greater<int>());
  int ans = -1; string answ;
  for (auto x : sizes){
      for (int i = 1; i < n-x; i++){
          if (Hashes.count(getkey(i, i+x-1))){
                          if (x > ans) answ = str.substr(i, x);
                          ans = max(ans, x);
                          cout << answ << endl; return 0;
          }
      }
  }
  (ans == -1) ? cout << "Just a legend\n" : cout << answ << endl;
  return 0;
}
```

- Longest Palindromic Substring:

```
const ll maxn = 1e5*5 + 5, base = 31, mod = 1e9 + 9;
string str1, str2; int n; ll pot[2][maxn], ahash[2][maxn];
int get_id(int i, int m){
  (!m) ? return str1[i]-'a'+1 :  return str2[i]-'a'+1;
}
void build(int m){
  pot[m][0] = 1;
  ahash[m][0] = get_id(0, m);
  for (int i = 1; i < n; i++){
      pot[m][i] = (pot[m][i-1] * base) % mod;
      ahash[m][i] = ((ahash[m][i-1]*base) + get_id(i, m)) % mod;
  }
}
ll getkey(int l, int r, int m){
  ll res = ahash[m][r];
  if (l > 0) res = (res - ((pot[m][r-l+1] * ahash[m][l-1]) % mod) + mod) % mod;
  return res;
}
bool checkPal(int k){
  for (int i = 0; i <= n-k; i++){
      ll hash1 = getkey(i, i+k-1, 0);
      ll hash2 = getkey(n-i-k, n-1-i, 1);
      if (hash1 == hash2)
          return true;
  }
  return false;
}
int par = 1, impar = 1;
vi vPar, vImp;
void bSearchP(int l, int r){
  while(l < r){
      int m = (l+r)/2;
      if(!checkPal(vPar[m])){
          r = m;
      }else{
          par = max(par, vPar[m]);
          l = m+1;
      }
  }
}
void bSearchI(int l, int r){
  while(l < r){
      int m = (l+r)/2;
      if(!checkPal(vImp[m])){
          r = m;
      }else{
          impar = max(impar, vImp[m]);
          l = m+1;
      }
  }
}
int main(){
  ios::sync_with_stdio(false), cin.tie(0);
  cin >> n; cin >> str1; n = str1.size(); str2 = "";
  for (auto x : str1) str2 = x + str2;
  if (str1 == str2){cout << n << endl; return 0;}
  build(0); build(1);
  for (int i = 1; i <= n; i++) (i%2) ? vImp.pb(i) : vPar.pb(i);
  bSearchP(0, vPar.size()-1); bSearchI(0, vImp.size()-1);
  cout << max(par, impar) << endl; return 0;}
```

- Watto and the Mechanism

Given string *s*, determine if the memory of the mechanism contains string *t* that consists of the same number of characters as *s* and differs from *s* in exactly one position

```
const ll mod = 1e18 + 5, base = 11;
set<ll> hashes;
vi abc = {3, 5, 7};//abc[c - 'a']
int main(){
  ios::sync_with_stdio(false), cin.tie(0);

  int n, Q; cin >> n >> Q;
  while(n--){
      string str; cin >> str;
      ll hash = 0;
      for (int i = str.size()-1; i >= 0; i--){
          hash += (abc[str[i]-'a'] * fExp(base, (str.size()-1-i))) % mod;
      }
      hashes.insert(hash);
  }
  while(Q--){
      string str; cin >> str; bool valid = false;
      ll hash = 0;
      for (int i = str.size()-1; i >= 0; i--)
          hash += (abc[str[i]-'a'] * fExp(base, (str.size()-1-i))) % mod;
      for (int i = 0; i < int(str.size()); i++){
          ll modified1 = hash, modified2;
          ll exp = fExp(base, (str.size()-1-i));
          modified1 -= (abc[str[i]-'a'] * exp) % mod;
          modified2 = modified1;
          modified1 += (abc[(str[i]-'a'+1)%3]*exp) % mod;
          modified2 += (abc[(str[i]-'a'+2)%3]*exp) % mod;
          if (hashes.count(modified1) || hashes.count(modified2)){
                  valid = true;
                  break;
          }
      }
      (valid) ? cout << "YES\n" : cout << "NO\n";
  }
  return 0;
}
```