RABIN KARP

```
vector<int> rabin_karp(string const& s, string const& t)
{
    const int p = 31;
    const int mod = 1e9+9;

    vector<long long> p_pow(max(s.size(),t.size()));
    p_pow[0]=1;
    for (int i=1;i<p_pow.size();i++)
        p_pow[i]=(p_pow[i-1]*p)%mod;
    vector<long long> hash_T(t.size()+1,0);
    for (int i=0;i<t.size();i++)
    {
        hash_T[i+1]=(hash_T[i]+(t[i]-'a'+1)*p_pow[i])%mod;
    }
    long long hash_s=0;
    for (int i=0;i<s.size();i++)
    {
        hash_s=(hash_s+(s[i]-'a'+1)*p_pow[i])%mod;
    }
    vector<int> ocurrences;
    for (int i=0;(i+(int)s.size()-1)<t.size();i++)
    {
        long long cur_t =(hash_T[i+s.size()]+mod-hash_T[i])%mod;
        if ((hash_s*p_pow[i])%mod==cur_t)
            ocurrences.push_back(i);
    }
    return ocurrences;
}
```

HASH FUNCTION

```
// optionn 1 (slower - don't know why)
struct chash{
 ll operator()(ll x) const {return std::hash<ll>{}((x^RANDOM)%MOD *
MUL);}
};

// Option 2
struct chash{
   int operator()(int x) const{return x^RANDOM;}
};

// Option 3 (fastest)
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
gp_hash_table<int,int, chash> inQ;
```