ĐẠI HỌC KHOA HỌC TỰ NHIÊN

HUS.HAD

Hanoi University of Science

Mục lục

1 Miscellanous

1.0.1 Commands on Shell

```
# compile
g++ $1.cpp --std=c++17 -Wall -Wextra -O2 -o $1
diff output.txt answer.txt
# before running shell file
chmod 700 any_shell_file.sh
```

1.0.2 Template

```
#include <bits/stdc++.h>
using namespace std;
/*
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
↪  tree_order_statistics_node_update> ordered_set;
*/
typedef long long ll;
typedef long double ld;
template<class T>using PQMax=priority_queue<T>;
template<class T>using PQMin=priority_queue<T, vector<T>,
↪  greater<T>>;
template<class T1, class T2>
void maximize(T1 &a, T2 b){ if (b > a) a = b; }
template<class T1, class T2>
void minimize(T1 &a, T2 b){ if (b < a) a = b; }
template<class T> void read(T &number) {
  bool negative = false; register int c;
  number = 0; c = getchar();
  while (c!='-' && !isalnum(c)) c = getchar();
  if (c=='-') negative = true, c = getchar();
  for (; (c>47 && c<58); c=getchar())
    number = number*10 + c-48;
  if (negative) number *= -1;
}
template<class T, class ...Ts>
void read(T &a, Ts&... args){
  read(a); read(args...);
}

#define fi first
#define se second
#define FOR(type,i,a,b) for(type i=(a); i<=(b); i++)
#define REV(type,i,b,a) for(type i=(b); i>=(a); i--)
#define EFOR(type,i,a,b) for(type i=(a); i<(b); i++)
#define EREV(type,i,b,a) for(type i=(b); i>(a); i--)
#define testBit(n, bit) (((n) >> (bit)) & 1)
#define flipBit(n, bit) ((n) ^ (1ll << (bit)))
#define cntBit(n) __builtin_popcount(n)
#define cntBitll(n) __builtin_popcountll(n)
#define log2(n) (31 - __builtin_clz(n))
#define log2ll(n) (63 - __builtin_clzll(n))
#define CURRENT_TIMESTAMP
↪  chrono::steady_clock::now().time_since_epoch().count()
#define randomize mt19937_64 mt(CURRENT_TIMESTAMP)

// remember to fill in:
// #define MAX ???
// #define MOD ???

// int main()
// {ios_base::sync_with_stdio(0);cin.tie(0);}
```

2 Mathematics

2.1 Data structure

2.1.1 Modulo

```
typedef unsigned long long ull;
struct modint{
  inline static const ll MOD = 998244353;
  int v;
  static inline ll mod(ll num) {
    ll val=num-ull((__uint128_t(-1ULL/MOD)*num)>>64)*MOD;
    return val-(val>=MOD)*MOD;
  }
  modint inv() const{
    ll answer = 1, a = v, n = MOD - 2;
    while (n) {
      if (n & 1) answer = mod(answer * a);
      a = mod(a * a); n >>= 1;
    }
    return answer;
  }

  modint(ll a = 0)
    { v=(a<0)?(MOD-mod(-a)):mod(a);v-=(v>=MOD)*MOD; }
  inline modint& operator += (modint b)
    { v+=b.v; v-=(v>=MOD)*MOD; return *this; }
  inline modint& operator -= (modint b)
    { v+=MOD-b.v; v-=(v>=MOD)*MOD; return*this; }
  inline modint& operator *= (modint b)
    { v = mod(1ll * v * b.v); return *this; }
  inline modint& operator /= (modint b)
    { return (*this)*=b.inv(); }
  inline modint& operator ^= (ll n) {
    modint a = v; v = 1;
    while (n) {if (n & 1) *this *= a; a *= a, n >>= 1;}
    return *this;
  }
};
inline modint operator+(modint a, modint b){return a+=b;}
inline modint operator-(modint a, modint b){return a-=b;}
inline modint operator*(modint a, modint b){return a*=b;}
inline modint operator/(modint a, modint b){return a/=b;}
inline modint operator^(modint a, ll n){return a^=n;}
inline bool operator == (modint a, modint b)
  { return a.v==b.v; }
inline bool operator != (modint a, modint b)
  { return a.v!=b.v; }
inline bool operator < (modint a, modint b)
  { return a.v<b.v; }
inline bool operator > (modint a, modint b)
  { return a.v>b.v; }
inline bool operator <= (modint a, modint b)
  { return a.v<=b.v; }
inline bool operator >= (modint a, modint b)
  { return a.v>=b.v; }
inline istream& operator >> (istream& s, modint &i)
  { ll tmp; s >> tmp; i = tmp; return s; }
inline ostream& operator << (ostream& s, modint i)
  {return s << i.v;}
```

2.1.2 BigInteger

```
#include "../../miscellanous/template.hpp"
#include "../../math/numerical/ntt.hpp"

// potential MLE in reserve() && resize()
class bigint {
  using vc = vector<char>;
  private:
    bool nega = false;
    vc digs;
    inline bigint&flipNega(){nega=!nega;return *this;}
    inline void reformat() {
      while ((int)digs.size() > 1 && *(digs.rbegin()) ==
      ↪  0) digs.pop_back();
      if (digs.size() == 1 && digs[0] == 0) nega = false;
    }
    inline bigint& _plusD(const bigint&num, int p10=0) {
      resize(max(size(), num.size() + p10)); bool nho = 0;
```

```
   EFOR(int, i, p10, num.size() + p10) {
    digs[i] += nho + num.digs[i-p10];
    nho = digs[i] > 9; digs[i] -= 10 * nho;
   }
   EFOR(int, i, num.size() + p10, size()) {
    digs[i]+=nho; nho=digs[i]>9; digs[i]-=10*nho;
    if (nho == 0) break;
   }
   if (nho) push_back(nho); return *this;
  }
  inline bool ltAbs(const bigint&oth,bool xV,bool eqV)const{
   if (size()!=oth.size())return size()<oth.size();
   REV(int, i, size()-1, 0) if (digs[i] != oth.digs[i])
    return (digs[i] < oth.digs[i]) xor xV;
   return eqV;
  }
  inline bigint& _minusD(const bigint &num) {
   bool flip = ltAbs(num, false, false);
   const vc *big = &(flip ? num.digs : digs),
      *small = &(flip ? digs : num.digs);
   resize(big->size());

   bool nho = 0;
   EFOR(int, i, 0, small->size()) {
    digs[i] = (*big)[i] - (nho + (*small)[i]);
    nho = digs[i] < 0; digs[i] += 10 * nho;
   }
   EFOR(int, i, small->size(), size()) {
    digs[i] = (*big)[i] - nho; nho = digs[i] < 0;
    digs[i] += 10 * nho; if (nho == 0) break;
   }
   if (nho or flip) flipNega();
   if (nho) push_back(nho); else reformat();
   return *this;
  }

 public:
  inline static string _I;
  /* constructors */
  bigint() { digs = {0}; }
  bigint(ll num) {
   if(num == 0){digs={0};return;} digs.reserve(18);
   while (num) digs.push_back(num%10), num/=10;
  }
  bigint(string s) {
   digs.reserve(s.size());
   REV(int, j, (int)s.size()-1, 0)
    if (s[j] != '-') digs.push_back(s[j] - '0');
    else nega = true;
   reformat();
  }
  bigint(vc &digs, bool nega)
   : digs(digs), nega(nega) {reformat();}
  bigint(vc &&digs, bool nega)
   : digs(digs), nega(nega) {reformat();}
  /* access data */
  inline bool isNega() const { return nega; }
  inline char&operator[](int idx) { return digs[idx]; }
  /* copy vector api */
  inline int size() const { return digs.size(); }
  inline bool empty() const { return digs.empty(); }
  inline void pop_back(){ digs.pop_back(); }
  inline void reserve(int sz) { digs.reserve(sz); }
  inline void resize(int sz) { digs.resize(sz); }
  inline void push_back(char c){ digs.push_back(c); }
  /* comparator */
  inline bool operator == (const bigint& oth) const {
   if (isNega() != oth.isNega()) return false;
   if (size() != (int)oth.size()) return false;
   REV(int,i,size()-1,0)if(digs[i]!=oth.digs[i])return false;
   return true;
  }
  inline bool operator < (const bigint& oth) const {
   if (isNega() != oth.isNega()) return isNega();
   return ltAbs(oth, isNega(), false);
  }
  inline bool operator <= (const bigint& oth) const {
   if (isNega() != oth.isNega()) return isNega();
   return ltAbs(oth, isNega(), true);
  }
  inline bool operator != (const bigint &num) const
   { return not operator==(num); }
  inline bool operator >= (const bigint &num) const
   { return not operator<(num); }
  inline bool operator > (const bigint &num) const
   { return not operator<=(num); }
  // fix rlvalue error/improve performance by:
  // inline bool operator ...  (bigint &&num) const
  //   { return operator ... (num); }

  /* operator + - * / */
  inline bigint operator - () const
   { return bigint(*this).flipNega(); }
  inline bigint& operator += (const bigint& num) {
   if (num.isNega() == isNega()) _plusD(num);
   else _minusD(num); return *this;
  }
  inline bigint& operator -= (const bigint& num) {
   if (num.isNega() == isNega()) _minusD(num);
   else _plusD(num); return *this;
  }
  inline bigint operator + (const bigint& num)
   { bigint res(*this); res += num; return res; }
  inline bigint operator - (const bigint& num)
   { bigint res(*this); res -= num; return res; }
  inline bigint operator * (const bigint &num) const {
   if (*this==0||num==0||empty()||num.empty())return 0;
   if (size() == 1 or num.size() == 1) {
    const vector<char> &D =(size()==1)?num.digs:digs;
    char mul = (size() == 1) ? digs[0] : num.digs[0];
    bigint ans; ans.resize(D.size()); char nho = 0;
    EFOR(int, i, 0, D.size()) {
     ans[i] = D[i] * mul + nho; nho = ans[i] / 10;
     ans[i] -= 10 * nho;
    }
    if (nho) ans.push_back(nho);
    if (isNega() xor num.isNega()) ans.flipNega();
    return ans;
   }
   /* FFT */
   FFT::buildRoot();
   vector<int> a(digs.begin(), digs.end());
   vector<int> b(num.digs.begin(), num.digs.end());
   vector<modint> newAns = FFT::multiply(a, b);
   bigint ans; ans.resize(newAns.size());
   int nho = 0;
   EFOR(int, i, 0, ans.size()){
    int tmp = newAns[i].v + nho;
    nho = tmp / 10;
    ans[i] = tmp - nho * 10;
   }
   while (nho>0) {ans.push_back(nho%10); nho/=10;}
   /* Karatsuba */
   // int mxSz = max(size(), num.size()), B = mxSz / 2;
   // bigint a0(vector<char>(digs.begin(), digs.begin() + min(B, size())), false),
   // a1(vector<char>(digs.begin()+min(B, size()), digs.end() ), false),
   // b0(vector<char>(num.digs.begin(), num.digs.begin()+min(B, num.size())), false),
   // b1(vector<char>(num.digs.begin()+min(B, num.size()), num.digs.end() ), false);
   //ans=a0*b0;bigint z2=a1*b1,z1=(a0+a1)*(b0+b1)-(z2+ans);
   // ans._plusD(z2, B*2)._plusD(z1, B);
   if (isNega() xor num.isNega()) ans.flipNega();
   ans.reformat(); return ans;
  }
  friend istream& operator>>(istream&inp, bigint&num)
   {inp>>bigint::_I;num=bigint(bigint::_I);return inp;}
  friend ostream&operator<<(ostream&out,const bigint&num){
   if (num.isNega()) out << '-';
   REV(int,i,(int)num.size()-1,0)out<<char(num.digs[i]+'0');
   return out;
  }
  friend ostream& operator << (ostream& out, bigint&& num) {
   return out << num;
  }
};
```

2.2 Number Theory

2.2.1 Mildly-optimized sieve(Factorable)

```
// remember to call sieve() before any factorize()
namespace Eratos_Factorable {
  constexpr ll MAX = 100000000;
  constexpr ll EST = 53000000; // 1.1*MAX/ln(MAX)
  int smallDiv[MAX + 1] = {};
  int primes[EST], cntPrime = 0;
  inline void sieve(int size = MAX) {
    memset(smallDiv, false, sizeof(smallDiv));
    primes[++cntPrime] = 2; primes[++cntPrime] = 3;
    for (int i=2; i<=size; i += 2) smallDiv[i] = 2;
    for (int i=3; i<=size; i += 6) smallDiv[i] = 3;
    for (int mul = 1; 6 * mul - 1 <= size; mul++) {
      bool pass = false;
      for(int i:{6*mul-1,6*mul+1})if(!smallDiv[i]){
        primes[++cntPrime] = i; smallDiv[i] = i;
        for (ll j = 1ll*i*i; j <= size; j += i*2)
          if (not smallDiv[j]) smallDiv[j] = i;
        if (1ll * i * i > size) pass = true;
      }
      if (pass) break;
    }
  }
  inline vector<int> factorize(ll number) {
    vector<int> ans;
    while (number > 1) {
      int d = smallDiv[number];
      while (number % d == 0)
        ans.push_back(d), number /= d;
    }
    return ans;
  }
}
```

2.2.2 Primality check

```
#define MAX 1000001
#define MOD 1000000007

namespace MillerRabin{
  typedef __int128_t i128;
  constexpr int SMALL_PRIMES[12] =
  ↪  {2,3,5,7,11,13,17,19,23,29,31,37};

  inline ll _power(ll a, ll n, ll mod) {
    ll ans = 1;
    while (n) {
      if (n & 1) ans = (i128)ans * a % mod;
      a = (i128)a * a % mod; n >>= 1;
    }
    return ans;
  }
  inline bool _fermatCheck(ll n, ll a, ll pw, int p2) {
    ll num = _power(a, pw, n);
    if (num == 1 || num == n - 1) return true;
    FOR(int, i, 1, p2-1) {
      num = (i128) num * num % n;
      if (num == n - 1) return true;
    }
    return false;
  }

  inline bool checkPrime(const ll n) {
    if (n == 2 || n == 3 || n == 5 || n == 7) return true;
    if (n < 11 || (n & 1) == 0) return false;
    ll d = n-1; int p2 = 0;
    while ((d & 1) == 0) d >>= 1, p2++;
    for (int a: SMALL_PRIMES)
      if (n == a) return true;
      else if (not _fermatCheck(n, a, d, p2))
        return false;
    return true;
  }
}
```

2.2.3 Prime factorization

```
#include "millerrabin.hpp"
randomize;
namespace Pollards{
  typedef __int128_t i128;
  #define sqp1(a, b, mod) (((i128(a)*(a)+(b)))%(mod))
  #define rand (mt()%1'000'000'000 + 1)
  inline vector<ll> rho(ll n) {
    if (n == 1) return {};
    if (MillerRabin::checkPrime(n)) return {n};
    vector<ll> ans;
    if (n % 2 == 0) {
      while (n % 2 == 0) ans.push_back(2), n /= 2;
      vector<ll> others = rho(n);
      ans.insert(ans.end(), others.begin(), others.end());
      return ans;
    }
    ll x = 2, y = 2, g = 1, b = 1;
    while (g == 1) {
      x=sqp1(x,b,n); y=sqp1(y,b,n); y=sqp1(y,b,n);
      g = __gcd(abs(x-y), n);
      if (g == n) x=y=rand, b=rand, g=1;
    }
    vector<ll> tmp1 = rho(g), tmp2 = rho(n / g);
    ans.insert(ans.end(), tmp1.begin(), tmp1.end());
    ans.insert(ans.end(), tmp2.begin(), tmp2.end());
    return ans;
  }
}
```

2.3 Numerical

2.3.1 FFT

```
#include "../../miscellanous/template.hpp"

#define MAX 1000001
#define MOD 1000000007

typedef complex<ld> cd;
namespace FFT{
  const ld TAU = acos(-1) * 2;
    // 3.14159'26535'89793'23846'26433'83279'50288'41971
    // '69399'37510'58 * 2;

  constexpr int BIT = 20, MAX_LEN = 1 << BIT;
  vector<int> _rev[BIT + 1];
  cd _root[MAX_LEN + 1];
  void buildRoot() {
    _root[0] = _root[MAX_LEN] = 1;
    for (int i=BIT-1, dist=1<<(BIT-1); i>=0; i--,
    ↪  dist>>=1) {
      cd w = polar(ld(1.0), TAU * dist / MAX_LEN);
      for (int pos = 0; pos < MAX_LEN; pos += 2 * dist)
        _root[pos + dist] = _root[pos] * w;
    }
    _rev[0] = {0};
    for (int bit=1, len=2; len<=MAX_LEN; bit++, len*=2) {
      _rev[bit].resize(len, 0);
      for (int i = 0; i < len; i++)
        _rev[bit][i]=((i&1)<<(bit-1))|_rev[bit-1][i/2];
    }
  }
  void dft(vector<cd> &poly, bool invert = false) {
    assert((cntBit((int)poly.size()) == 1));
    const int n = poly.size(), coef = MAX_LEN / n;
    for (int dist=n/2, span=n; dist>0; dist/=2, span/=2)
    for (int pos = 0; pos < dist; pos++)
    for (int i=pos, k=0; i<n; i+=span, k+=dist) {
      int len = log2(n / span);
      int newK = _rev[len][k / dist] * dist;
      int tmp = (invert ? (n - newK) : newK) * coef;
      cd a = poly[i], b = _root[tmp] * poly[i + dist];
      poly[i] = a + b, poly[i + dist] = a - b;
    }
    if (invert) for (cd &x: poly) x /= n;
  }
  template<class T>vector<T>conv(vector<T>_a,vector<T>_b){
    int len = int(_a.size() + _b.size()) - 1;
    int bit = log2(len)+(cntBit(len)>1); len=1<<bit;
    vector<cd> a(len);
    for (int i = 0; i < len; i++)
      a[i] = cd((i < _a.size())?_a[i]:0,
      ↪  (i<_b.size())?_b[i]:0);
    dft(a, false);
    for (int i = 0; i < len; i++)
      if (i < _rev[bit][i]) swap(a[i], a[_rev[bit][i]]);
    for (int i = 0; i < len; i++) a[i] *= a[i];
    vector<cd> b(a.begin(), a.end());
    for (int i = 0; i < len; i++)
      b[i] = a[i] - conj(a[-i & (len-1)]); //(n-i)%n
    dft(b, true);
    for (int i = 0; i < len; i++)
      if (i < _rev[bit][i]) swap(b[i], b[_rev[bit][i]]);

    while (len > 0 && b[len - 1].imag() < 1e-9) len--;
    vector<T> ans(len);
    FOR(int, i, 0, len-1) ans[i]=round(b[i].imag()/4.0);
    return ans;
  }
}
```

2.3.2 NTT

k-th root of unity, mod p: ω_k satisfies $\begin{cases}\omega_k^k \equiv 1 \\ \omega_k^i \not\equiv \omega_k^j & (i \neq j)\end{cases}$

$p = c \times 2^k + 1 \Rightarrow w_{2^k} = g^c$ w/ any $g : \gcd(g, p) = 1$

only works for p with big k (e.g. 998244353 w/ $k = 23$ below)

```
#include "../../math/ds/modint.hpp"

// MOD=998244353=c*2^k+1 => ROOT=g^c, any g: gcd(g,MOD)=1
namespace FFT{
  constexpr int K_MOD = 23, BIT = 22, MAX_LEN = 1 << BIT;
  constexpr int ROOT = 15311432;

  modint _root[MAX_LEN + 1];
  vector<int> _rev[BIT + 1];
  bool _built = false;
  void buildRoot() {
    if (_built) return; _built = true;
    _root[0] = 1; modint mul=ROOT;
    FOR(int, i, 1, K_MOD-BIT) mul*=mul;
    FOR(int, i, 1, MAX_LEN) _root[i] = _root[i-1] * mul;
    for (int bit=0, len=1; len<=MAX_LEN; bit++, len*=2) {
      _rev[bit].resize((1 << bit), 0);
      for (int i = 1; i < (1 << bit); i++)
        _rev[bit][i]=((i&1)<<(bit-1))|_rev[bit-1][i/2];
    }
  }

  void transform(vector<modint> &v, bool invert){
    const int len = v.size(), coef = MAX_LEN / len;
    const int bit = log2(len);
    FOR(int, i, 0, len-1) if (i < _rev[bit][i])
      swap(v[i], v[_rev[bit][i]]);

    for (int jmp=1, span=2; span<=len; jmp*=2, span*=2)
    for (int beg = 0; beg < len; beg += span)
    for (int i = 0; i < jmp; i++) {
      int k=coef*len/jmp*i/2; if (invert) k=MAX_LEN-k;
      modint a = v[beg+i], b = _root[k] * v[beg+i+jmp];
      v[beg + i] = a + b; v[beg + i + jmp] = a - b;
    }
    if (invert) FOR(int, i, 0, len-1) v[i] /= len;
  }
  template<class T>
  vector<modint> multiply(vector<T> &_a, vector<T> &_b) {
    int len = int(_a.size() + _b.size()) - 1;
    len = 1 << (log2(len) + (cntBit(len) > 1));
    vector<modint>a(_a.begin(),_a.end()); a.resize(len,0);
    vector<modint>b(_b.begin(),_b.end()); b.resize(len,0);
    transform(a, false); transform(b, false);
    FOR(int, i, 0, len - 1) a[i] *= b[i];
    transform(a, true); return a;
  }
}
```

3 String

3.1 Data Structures

3.1.1 HashStr

```
class Hash {
  #define op operator
private:
  inline static constexpr int SZ = 2, MOD[6] = {
    (int) 1e9+7, (int) 1e9+20041203, (int) 1e9+2277,
    (int) 1e9+5277, (int) 1e9+8277, (int) 1e9+9277,
  };
  int val[SZ] = {};
  static ll pw(ll a, ll n, ll mod) {
    ll answer = 1;
    for (; n > 0; a = a * a % mod, n >>= 1)
      if (n & 1) answer = answer * a % mod;
    return answer;
  }
public:
  Hash(ll num){EFOR(int,i,0,SZ)val[i]=num%MOD[i];}
  Hash(): Hash(0) {}
  int& op[] (int idx) {return val[idx];}
  bool op == (const Hash &oth) const {
    EFOR(int,i,0,SZ)if(oth.val[i]!=val[i])return false;
    return true;
  }
  bool op < (const Hash &oth) const {
    EFOR(int,i,0,SZ)
      if(oth.val[i]!=val[i]) return val[i]<oth.val[i];
    return false;
  }
  bool op <= (const Hash &oth) const {
    EFOR(int, i, 0, SZ)
      if(oth.val[i]!=val[i])return val[i]<oth.val[i];
    return true;
  }
  bool op != (const Hash&oth)const{return not op==(oth);}
  bool op > (const Hash&oth)const{return not op<=(oth);}
  bool op >= (const Hash&oth)const{return not op<(oth);}
  Hash& op += (Hash &oth) {
    EFOR(int,i,0,SZ)val[i]=(val[i]+oth[i])%MOD[i];
    return *this;
  }
  Hash& op -= (Hash &oth) {
    EFOR(int,i,0,SZ)val[i]=(val[i]+MOD[i]-oth[i])%MOD[i];
    return *this;
  }
  Hash& op *= (Hash &oth) {
    EFOR(int,i,0,SZ)val[i]=(1ll*val[i]*oth[i])%MOD[i];
    return *this;
  }
  Hash inv() const {
    Hash ans; EFOR(int, i, 0, SZ)
      ans[i] = pw(val[i], MOD[i]-2, MOD[i]);
    return ans;
  }
  Hash& op/=(const Hash&oth){return op*=(oth.inv());}
  Hash& op+=(Hash&&oth){ return op+=(oth);}
  Hash& op-=(Hash&&oth){ return op-=(oth);}
  Hash& op*=(Hash&&oth){ return op*=(oth);}
  Hash& op/=(Hash&&oth){ return op/=(oth);}
  friend Hash op+(Hash a,Hash b){return(a += b);}
  friend Hash op-(Hash a,Hash b){return(a -= b);}
  friend Hash op*(Hash a,Hash b){return(a *= b);}
  friend Hash op/(Hash a,Hash b){return(a /= b);}
  friend ostream& op << (ostream& out, const Hash &num) {
    out << "(";
    EFOR(int,i,0,SZ)out<<num.val[i]<<",)"[i==SZ-1];
    return out;
  }
};

class HashString {
private:
  inline static constexpr int MAX = 1e6, BASE = 311;
  inline static Hash _p[MAX + 1], _i[MAX + 1];
  Hash *sF, *sB; int len;
  inline static bool _init = false;
public:
  static void init() {
    _init = true; _p[0] = 1;
    FOR(int, i, 1, MAX) _p[i] = _p[i-1] * BASE;
    _i[MAX] = _p[MAX].inv();
    REV(int, i, MAX-1, 0) _i[i] = _i[i + 1] * BASE;
    FOR(int, i, 0, MAX) assert(_i[i] * _p[i] == 1);
  }
  HashString(string s = "") {
    if (not _init) init(); len = s.size();
    sF=new Hash[len+1],sB=new Hash[len+2];
    sF[0]=sB[len+1]=0;
    FOR(int,i,1,len)sF[i]=sF[i-1]+s[i-1]*_p[i];
    REV(int,i,len,1)sB[i]=sB[i+1]+_p[len-i+1]*s[i-1];
  }
  ~HashString() {}

  int getLength() const {return len;}
  Hash getF(int l,int r){return(sF[r]-sF[l-1])*_i[l-1];}
  Hash getB(int l,int r){return(sB[l]-sB[r+1])*_i[len-r];}
  Hash getF(int idx) {return sF[idx];}
  Hash getB(int idx) {return sB[idx];}
};
```

4 Graph

4.1 Shortest Path

4.1.1 SPFA

Somehow this passed yosupo's Library Checker.

```
#define MAX 1000001
struct Node
{
  int node, len;
  Node() {node = len = 0;}
  Node(int node, int len) {this -> node = node, this -> len = len;}
};
namespace SPFA{
int n; vector<Node> graph[MAX];
constexpr ll oo = (1e18+1);
bool in[MAX]; int cnt[MAX];
ll d[MAX], trace[MAX];
deque<int> q;
randomize;

void spfa(int root){
  FOR(int, i, 0, n)
    shuffle(graph[i].begin(), graph[i].end(), mt);
  memset(d, 0x3f, sizeof(d));

  d[root] = 0; q.push_back(root);
  ll qSum = 0;
  while (!q.empty()) {
    while (d[q.front()] > d[q.back()])
      { q.push_back(q.front()); q.pop_front(); }
    int node = q.front(); q.pop_front();
    in[node] = false;
    const int LIM = sqrt(n);
    qSum -= d[node];
    for (Node ch: graph[node]) {
      int child=ch.node;ll chDist=max(-oo,d[node]+ch.len);
      if (chDist >= d[child]) continue;
      qSum=max(-oo,qSum+chDist-in[child]*d[child]);
      d[child] = chDist, trace[child] = node;
      if (in[child]) continue;
      if (++cnt[child] == n)
        d[child] = chDist = -oo;
      in[child] = true;
      if (cnt[child] > LIM or d[child] * q.size() > qSum)
        q.push_back(child);
      else if (q.empty() or d[child] <= d[q.front()])
        q.push_front(child);
      else
        q.push_back(child);
    }
  }
}
}
```