Moscow Institute of Physics and Technology

My Pity

Fedor Alekseev, Dmitry Ivaschenko, Daria Kolodzey

Whatever contest

today

Contest (1)

template.cpp

13 lines

```
#include <bits/extc++.h>
using namespace std;

#define WHOLE(v) v.begin(),v.end()
#define sz(v) static_cast<int>(v.size())

using i64 = int64_t;

int main() {
  cin.sync_with_stdio(false);
  cin.tie(nullptr);
  cin.exceptions(cin.failbit);
}
```

.vimrc

3 lines

```
set nocp si aw ai is ts=2 sw=2 et tm=100 nu bg=dark
sy on
im jj <esc>
```

Numerical (2)

PolyRoots.h

Description: Finds the real roots to a polynomial.
Usage: `poly_roots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`
Time: $\mathcal{O}\left(n^2 \log(1/\epsilon)\right)$

"Polynomial.h" 23 lines

```
vector<double> poly_roots(Poly p, double xmin, double xmax) {
  if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Poly der = p;
  der.diff();
  auto dr = poly_roots(der, xmin, xmax);
  dr.push_back(xmin-1);
  dr.push_back(xmax+1);
  sort(all(dr));
  rep(i,0,sz(dr)-1) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign ^ (p(h) > 0)) {
      rep(it,0,60) { // while (h - l > 1e-8)
        double m = (l + h) / 2, f = p(m);
        if ((f <= 0) ^ sign) l = m;
        else h = m;
      }
      ret.push_back((l + h) / 2);
    }
  }
  return ret;
}
```

PolyInterpolate.h

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \ldots n-1$.
Time: $\mathcal{O}\left(n^2\right)$

13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}\left(n^2 m\right)$

40 lines

```
int solveLinear(vector<vector<double>& A, vector<double>& b,
    vector<double& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vector<int> col(m); iota(WHOLE(col), 0);

  for (int i = 0; i < n; ++i) {
    double v, bv = 0;
    for (int r = i; r < n; ++r) for (int c = i; c < m; ++c)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv < eps) {
      for (int j = 0; j < n; ++j)
        if (fabs(b[j]) > eps)
          return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    for (int j = 0; j < n; ++j)
      swap(A[j][i], A[j][bc]);
    bv = 1. / A[i][i];
    for (int j = i + 1; j < n; ++j) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      for (int k = i + 1; k < m; ++k)
        A[j][k] -= fac * A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    for (int j = 0; j < i; ++j)
      b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

"SolveLinear.h" 8 lines

```
for(int j = 0; j < n; ++j)if(j != i) // instead of for(j=i+1; j<n)
// ... then at the end:
x.assign(m, undefined);
for (int i = 0; i < rank; ++i) {
  for (int j = rank; j < m; ++j)
    if (fabs(A[i][j]) > eps) goto fail;
  x[col[i]] = b[i] / A[i][i];
fail:; }
```

FastFourierTransform.h

Description: Computes $\hat{f}(k) = \sum_x f(x) \exp(-2\pi ikx/N)$ for all k. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. a and b should be of roughly equal size. For convolutions of integers, consider using a number-theoretic transform instead, to avoid rounding issues.
Time: $\mathcal{O}\left(N \log N\right)$

<valarray> 29 lines

```
typedef valarray<complex<double> > carray;
void fft(carray& x, carray& roots) {
  int N = sz(x);
  if (N <= 1) return;
  carray even = x[slice(0, N/2, 2)];
  carray odd = x[slice(1, N/2, 2)];
  carray rs = roots[slice(0, N/2, 2)];
  fft(even, rs);
  fft(odd, rs);
  rep(k,0,N/2) {
    auto t = roots[k] * odd[k];
    x[k    ] = even[k] + t;
    x[k+N/2] = even[k] - t;
  }
}

typedef vector<double> vd;
vd conv(const vd& a, const vd& b) {
  int s = sz(a) + sz(b) - 1, L = 32-__builtin_clz(s), n = 1<<L;
  if (s <= 0) return {};
  carray av(n), bv(n), roots(n);
  rep(i,0,n) roots[i] = polar(1.0, -2 * M_PI * i / n);
  copy(all(a), begin(av)); fft(av, roots);
  copy(all(b), begin(bv)); fft(bv, roots);
  roots = roots.apply(conj);
  carray cv = av * bv; fft(cv, roots);
  vd c(s); rep(i,0,s) c[i] = cv[i].real() / n;
  return c;
}
```

NumberTheoreticTransform.h

Description: Can be used for convolutions modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For other primes/integers, use two different primes and combine with CRT. May return negative values.
Time: $\mathcal{O}(N \log N)$

"ModPow.h" 38 lines

```
const ll mod = (119 << 23) + 1, root = 3; // = 998244353
// For p < 2^30 there is also e.g. (5 << 25, 3), (7 << 26, 3),
// (479 << 21, 3) and (483 << 21, 5). The last two are > 10^9.

typedef vector<ll> vl;
void ntt(ll* x, ll* temp, ll* roots, int N, int skip) {
  if (N == 1) return;
  int n2 = N/2;
  ntt(x     , temp, roots, n2, skip*2);
  ntt(x+skip, temp, roots, n2, skip*2);
  rep(i,0,N) temp[i] = x[i*skip];
  rep(i,0,n2) {
    ll s = temp[2*i], t = temp[2*i+1] * roots[skip*i];
    x[skip*i] = (s + t) % mod; x[skip*(i+n2)] = (s - t) % mod;
  }
}
void ntt(vl& x, bool inv = false) {
  ll e = modpow(root, (mod-1) / sz(x));
  if (inv) e = modpow(e, mod-2);
  vl roots(sz(x), 1), temp = roots;
  rep(i,1,sz(x)) roots[i] = roots[i-1] * e % mod;
  ntt(&x[0], &temp[0], &roots[0], sz(x), 1);
}
vl conv(vl a, vl b) {
  int s = sz(a) + sz(b) - 1; if (s <= 0) return {};
  int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0, n = 1 << L;
  if (s <= 200) { // (factor 10 optimization for |a|,|b| = 10)
    vl c(s);
    rep(i,0,sz(a)) rep(j,0,sz(b))
      c[i + j] = (c[i + j] + a[i] * b[j]) % mod;
    return c;
  }
  a.resize(n); ntt(a);
  b.resize(n); ntt(b);
  vl c(n); ll d = modpow(n, mod-2);
  rep(i,0,n) c[i] = a[i] * b[i] % mod * d % mod;
  ntt(c, true); c.resize(s); return c;
}
```

Strings (3)

SuffixArray.h

57 lines

```
struct SuffixArray {
  string s;
  vector<int> order, rank, lcp;

  SuffixArray(const string& _s): s(_s + ’$’) {
    int n = sz(s);
    std::vector<int> count(n + 130), nextPos(count.size() + 1);
    std::vector<int> nextOrder(n), nextColor(n);
    std::vector<int> color(WHOLE(s));

    auto norm = [n](int i) {
      return i < 0 ? i + n : i >= n ? i - n : i;
    };

    order.resize(n);
    std::iota(WHOLE(order), 0);
    std::sort(WHOLE(order),
        [&](int aa, int bb) { return s[aa] < s[bb]; });

    for (int half = 1; half < n; half *= 2) {
      count.assign(count.size(), 0);
      for (auto col : color)
        ++count[col];

      nextPos[0] = 0;
      partial_sum(WHOLE(count), nextPos.begin() + 1);

      for (auto pos : order) {
        auto shifted = norm(pos - half);
        nextOrder[nextPos[color[shifted]]++] = shifted;
      }
      order.swap(nextOrder);

      nextColor[order[0]] = 0;
      for (int i = 1; i < n; ++i) {
        auto pos = order[i], prev = order[i - 1];
        nextColor[pos] = nextColor[prev] + (
            tie(color[pos], color[norm(pos + half)]) !=
            tie(color[prev], color[norm(prev + half)])
        );
      }
      color.swap(nextColor);
    }

    rank.resize(n);
    for (int i = 0; i < n; ++i)
      rank[order[i]] = i;

    lcp.resize(n);
    for (int i = 0; i < n; ++i) if (rank[i]) {
      for (int p0 = order[rank[i] - 1]; s[i + h] == s[p0 + h];)
        h++;
      lcp[rank[i]] = h;
      h -= h > 0;
    }
  }
};
```

Hashes.h

29 lines

```
using Hash = array<ui64, 3>;
#define HOP(op) \
  inline Hash operator op (Hash a, Hash b) { \
    return {a[0] op b[0], a[1] op b[1], a[2] op b[2]}; \
  }
HOP(+)HOP(-)HOP(*)HOP(%)
inline Hash makeHash(ui64 val) { return {val, val, val}; }

const Hash Multiplier{{228227, 227223, 22823}};
const Hash Modulus{{424242429, 2922827, 22322347}};

vector<Hash> pows(1);
struct Hashes {
  explicit Hashes(const string& s) {
    pows.front().fill(1);
    while (pows.size() <= s.size())
      pows.push_back(pows.back() * Multiplier % Modulus);
    prefs.push_back(makeHash(0));
    for (auto c : s)
      prefs.push_back((prefs.back() * Multiplier + makeHash(c))
          % Modulus);
  }
  Hash get(size_t begin, size_t end) const {
    return (prefs[end] - prefs[begin] * pows[end - begin]
        % Modulus + Modulus) % Modulus;
  }
private:
  vector<Hash> prefs;
};
```

AhoCorasick.h

Description: on-line tracking of the set of suffixes of a text that are prefixes of some words from a dictionary.

44 lines

```
struct AhoCorasick {
  AhoCorasick(): n(1) {
    n.reserve(TrieSize);
  }

  void addWord(const string& word, int id) {
    int v = 0;
    for (int ch : word) {
      ch -= ’a’;
      auto& u = n[v].trans[ch];
      if (!u) {
        u = int(n.size());
        n.emplace_back();
      }
      v = u;
    }
    n[v].termId = id;
  }

  void build() {
    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
      auto v = q.front();
      for (Char ch = 0; ch < Alph; ++ch) {
        auto& u = n[v].trans[ch];
        if (!u) {
          u = n[n[v].link].trans[ch];
          continue;
        }
        q.push(u);
        auto i = n[u].link = (v ? n[n[v].link].trans[ch] : 0);
        n[u].nextTerm = (n[i].termId >= 0 ? i : n[i].nextTerm);
      }
    }
  }

private:
  struct Node {
    int trans[Alph]{};
    int nextTerm = -1, termId = -1, link = 0;
  };

  vector<Node> n;
};
```

ZFunction.h

Description: z[x] is max L: s[x:x+L] == s[:L]

11 lines

```
vector<size_t> zFun(const string& s) {
  vector<size_t> z(s.size(), 0);
  for (size_t left = 0, right = 0, i = 1; i < s.size(); ++i) {
    z[i] = (i < right ? min(right - i, z[i - left]) : 0);
    while (i + z[i] < s.size() && s[i + z[i]] == s[z[i]])
      ++z[i];
    if (i + z[i] > right)
      tie(left, right) = {i, i + z[i]};
  }
  return z;
}
```

PrefixFunction.h

Description: pi[x] is the length of the longest prefix of s that ends at x, other than s[0..x] itself

10 lines

```
vector<size_t> pi(const string& s) {
  vector<size_t> p(s.size(), 0);
  for (size_t i = 1; i < s.size(); ++i) {
    auto px = p[i - 1];
    while (px && s[i] != s[px])
      px = p[px - 1];
    p[i] = px + (s[i] == s[g]);
  }
  return p;
}
```

Manacher.h

Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).
Time: $\mathcal{O}(N)$

17 lines

```
void manacher(const string& s) {
  auto n = int(s.size());
  vector<int> p[2];
  p[0].resize(n + 1);
  p[1].resize(n);
  for (int z = 0; z < 2; ++z) {
    for (int i=0, l=0, r=0; i < n; ++i){
      int t = r - i + !z;
      if (i<r) p[z][i] = min(t, p[z][l + t]);
      int L = i - p[z][i], R = i + p[z][i] - !z;
      while (L >= 1 && R + 1 < n && s[L - 1] == s[R + 1])
        p[z][i]++, L--, R++;
      if (R > r)
        tie(l, r) = {L, R};
    }
  }
}
```