Moscow Institute of Physics and Technology

My Pity

Fedor Alekseev, Dmitry Ivaschenko, Daria Kolodzey

Whatever contest

today

Numerical (1)

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: $\mathcal{O}(n^2 m)$

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:; }
```

Strings (2)

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};
    }
  }
```

```
}
```