```cpp
namespace strll_op {

  class strll {
  private:
    string val; // interior process is always reversed
    inline string ll_to_strll(ll x) {
      if (x == 0) return "0";
      bool neg_flag = false;
      if (x < 0) { neg_flag = true; x *= -1; }
      string ret = "";
      while (x > 0) {
        ret += '0' + (x % 10);
        x /= 10;
      }
      if (neg_flag) ret += '-';
      return ret;
    }
    inline string uadd_core(const string &s, const string &t) {
      int n = s.length();
      int m = t.length();
      string ret = "";
      int v_digits = max(n, m) + 1;
      vi v(v_digits, 0);
      Loop(i, v_digits - 1) {
        if (i < n) v[i] += s[i] - '0';
        if (i < m) v[i] += t[i] - '0';
        if (v[i] >= 10) {
          v[i] -= 10;
          v[i + 1] += 1;
        }
      }
      if (v[v_digits - 1] == 0) v_digits = max(1, v_digits - 1);
      Loop(i, v_digits) ret += '0' + v[i];
      return ret;
    }
    inline string usub_core(const string &s, const string &t) {
      int n = s.length();
      int m = t.length();
      string ret = "";
      int v_digits = 1;
      vi v(n, 0);
      Loop(i, n) {
        v[i] += s[i] - '0';
        if (i < m) v[i] -= (t[i] - '0');
        if (v[i] < 0) {
          v[i] += 10;
          v[i + 1] -= 1;
        }
        if (v[i] > 0) v_digits = i + 1;
      }
      Loop(i, v_digits) ret += '0' + v[i];
      return ret;
    }
    inline string umul_core(const string &s, const string &t) {
      int n = s.length();
      int m = t.length();
      string ret = "";
      vi v(n + m, 0);
      Loop(i, n) {
        Loop(j, m) {
          int z = (s[i] - '0') * (t[j] - '0');
          v[i + j] += z % 10;
          v[i + j + 1] += z / 10;
        }
      }
      int v_digits = 1;
      Loop(i, n + m - 1) {
        v[i + 1] += v[i] / 10;
        v[i] %= 10;
        if (v[i + 1] > 0) v_digits = i + 2;
      }
```

```
72        Loop(i, v_digits) ret += '0' + v[i];
73        return ret;
74      }
75      inline bool uge_core(const string &s, const string &t) const {
76        int n = s.length();
77        int m = t.length();
78        while (n > 1 && s[n - 1] == '0') n--;
79        while (m > 1 && t[m - 1] == '0') m--;
80        if (n > m) return true;
81        else if (n < m) return false;
82        else {
83          Loopr(i, n) {
84            if (s[i] > t[i]) return true;
85            if (s[i] < t[i]) return false;
86          }
87          return true;
88        }
89      }
90      inline string udiv_core(string s, const string &t, bool rem_flag) {
91        int n = s.length();
92        int m = t.length();
93        string ret = "0";
94        Loopr(i, n - m + 1) {
95          ret += '0';
96          string sbuf = s.substr(i, m + 1);
97          while (uge_core(sbuf, t)) {
98            sbuf = usub_core(sbuf, t);
99            ret.back()++;
100         }
101         Loop(j, min(m + 1, n)) {
102           s[i + j] = j < sbuf.size() ? sbuf[j] : '0';
103         }
104       }
105       reverse(ret.begin(), ret.end());
106       if (rem_flag) ret = s;
107       while (ret.size() > 1 && ret.back() == '0') ret.pop_back();
108       return ret;
109     }
110     inline string add(const string &s, const string &t) {
111       int n = int(s.length());
112       int m = int(t.length());
113       string ret = "";
114       int mode = (s[n - 1] == '-' ? 0b10 : 0) + (t[m - 1] == '-' ? 0b01 : 0);
115       switch (mode) {
116       case 0b00:
117         ret = uadd_core(s, t);
118         break;
119       case 0b01:
120         if (uge_core(s.substr(0, n), t.substr(0, m - 1))) ret = usub_core(s, t.substr(0, m - 1));
121         else ret = usub_core(t.substr(0, m - 1), s) + '-';
122         break;
123       case 0b10:
124         if (uge_core(s.substr(0, n - 1), t.substr(0, m))) ret = usub_core(s.substr(0, n - 1), t) + '-';
125         else ret = usub_core(t, s.substr(0, n - 1));
126         break;
127       case 0b11:
128         ret = uadd_core(s.substr(0, n - 1), t.substr(0, m - 1)) + '-';
129         break;
130       }
131       if (ret == "0-") ret.pop_back();
132       return ret;
133     }
134     inline string sub(const string &s, const string &t) {
135       string ret = "";
136       int n = s.length();
137       int m = t.length();
138       int mode = (s[n - 1] == '-' ? 0b10 : 0) + (t[m - 1] == '-' ? 0b01 : 0);
139       switch (mode) {
140       case 0b00:
141         if (uge_core(s.substr(0, n), t.substr(0, m))) ret = usub_core(s, t);
142         else ret = usub_core(t, s) + '-';
```

```
143         break;
144       case 0b01:
145         ret = uadd_core(s, t.substr(0, m - 1));
146         break;
147       case 0b10:
148         ret = uadd_core(s.substr(0, n - 1), t) + '-';
149         break;
150       case 0b11:
151         if (uge_core(s.substr(0, n - 1), t.substr(0, m - 1))) ret = usub_core(s.substr(0, n - 1), t.substr
            (0, m - 1)) + '-';
152         else ret = usub_core(t.substr(0, m - 1), s.substr(0, n - 1));
153         break;
154       }
155       if (ret == "0-") ret.pop_back();
156       return ret;
157     }
158     inline string mul(const string &s, const string &t) {
159       string ret;
160       int n = s.length();
161       int m = t.length();
162       int mode = (s[n - 1] == '-' ? 0b10 : 0) + (t[m - 1] == '-' ? 0b01 : 0);
163       switch (mode) {
164       case 0b00:
165         ret = umul_core(s, t);
166         break;
167       case 0b01:
168         ret = umul_core(s, t.substr(0, m - 1)) + '-';
169         break;
170       case 0b10:
171         ret = umul_core(s.substr(0, n - 1), t) + '-';
172         break;
173       case 0b11:
174         ret = umul_core(s.substr(0, n - 1), t.substr(0, m - 1));
175         break;
176       }
177       if (ret == "0-") ret.pop_back();
178       return ret;
179     }
180     inline bool ge(const string &s, const string &t) const {
181       bool ret;
182       int n = s.length();
183       int m = t.length();
184       int mode = (s[n - 1] == '-' ? 0b10 : 0) + (t[m - 1] == '-' ? 0b01 : 0);
185       switch (mode) {
186       case 0b00:
187         ret = uge_core(s, t);
188         break;
189       case 0b01:
190         ret = true;
191         break;
192       case 0b10:
193         ret = false;
194         break;
195       case 0b11:
196         if (s == t) ret = true;
197         else ret = !uge_core(s.substr(0, n - 1), t.substr(0, m - 1));
198         break;
199       }
200       return ret;
201     }
202     inline string div(const string &s, const string &t, bool rem_flag) {
203       string ret;
204       int n = s.length();
205       int m = t.length();
206       int mode = (s[n - 1] == '-' ? 0b10 : 0) + (t[m - 1] == '-' ? 0b01 : 0);
207       switch (mode) {
208       case 0b00:
209         ret = udiv_core(s, t, rem_flag);
210         break;
211       case 0b01:
212         ret = udiv_core(s, t.substr(0, m - 1), rem_flag);
```

```
213         if (!rem_flag) ret += '-';
214         break;
215       case 0b10:
216         ret = udiv_core(s.substr(0, n - 1), t, rem_flag) + '-';
217         break;
218       case 0b11:
219         ret = udiv_core(s.substr(0, n - 1), t.substr(0, m - 1), rem_flag);
220         if (rem_flag) ret += '-';
221         break;
222       }
223       if (ret == "0-") ret.pop_back();
224       return ret;
225     }
226   public:
227     strll(string init = "0") { reverse(init.begin(), init.end()); val = init; return; }
228     strll(ll init) { val = ll_to_strll(init); return; }
229     strll(const strll& another) { val = another.val; return; }
230     inline strll& operator=(const strll &another) { val = another.val; return *this; }
231     inline strll operator+(const strll &x) { strll ret; ret.val = add(val, x.val); return ret; }
232     inline strll operator-(const strll &x) { strll ret; ret.val = sub(val, x.val); return ret; }
233     inline strll operator*(const strll &x) { strll ret; ret.val = mul(val, x.val); return ret; }
234     inline strll operator/(const strll &x) { strll ret; ret.val = div(val, x.val, false); return ret; }
235     inline strll operator%(const strll &x) { strll ret; ret.val = div(val, x.val, true); return ret; }
236     inline strll& operator+=(const strll &x) { val = add(val, x.val); return *this; }
237     inline strll& operator-=(const strll &x) { val = sub(val, x.val); return *this; }
238     inline strll& operator*=(const strll &x) { val = mul(val, x.val); return *this; }
239     inline strll& operator/=(const strll &x) { val = div(val, x.val, false); return *this; }
240     inline strll& operator%=(const strll &x) { val = div(val, x.val, true); return *this; }
241     inline bool operator>=(const strll &x) { return ge(val, x.val); }
242     inline bool operator>(const strll &x) { return ge(val, x.val) && val != x.val; }
243     inline bool operator<=(const strll &x) { return ge(x.val, val); }
244     inline bool operator<(const strll &x) { return ge(x.val, val) && val != x.val; }
245     inline bool operator==(const strll &x) { return val == x.val; }
246     inline bool operator!=(const strll &x) { return val != x.val; }
247     inline bool operator<(const strll &x) const { return !ge(val, x.val); }
248     friend inline istream& operator >> (istream &is, strll& x) { is >> x.val; reverse(x.val.begin(),
        x.val.end()); return is; }
249     friend inline ostream& operator << (ostream &os, const strll& x) { os << x.get_val(); return os; }
250     string get_val() const { string ret = val; reverse(ret.begin(), ret.end()); return ret; }
251   };
252 }
253
254 using namespace strll_op;
255 typedef vector<strll> vstrll;
256 typedef vector<vector<strll>> vvstrll;
```