

1.0 概览

[列表 \(CSDN博客链接\)](#)

[代码检查](#)

[上机前](#)

[提交前](#)

[Rejected 后](#)

[缺省源](#)

[关闭同步流](#)

[字符串](#)

[随机打乱](#)

[linux 下对拍](#)

[bitset](#)

[Python](#)

[高精度](#)

1.0 概览

列表 (CSDN博客链接)

- [Algorithm Notes 1 字符串](#)
- [Algorithm Notes 2 数据结构](#)
- [Algorithm Notes 3 数论](#)
- [Algorithm Notes 4 动态规划 计算几何](#)
- [Algorithm Notes 5 图论](#)
- [Algorithm Notes 6 多项式](#)
- [Algorithm Notes 7 组合数学](#)
- [Algorithm Notes 8 分治](#)
- [Algorithm Notes 9 数学相关](#)

代码检查

上机前

- 写代码的人应当**完整阅读过题面**，清楚题目的**所有范围限制**。
- **避免出现没想清楚就上机**的情况。

提交前

- [Optional] 若题目的样例较弱，可以尝试构造一些 corner cases，力求测试样例**覆盖所有的分支情况**。
- **逐一检查所有变量的类型、数组大小、取模/模数、多测清空**，以及**空间限制、是否存在无解、是否关闭了同步流、读入是否超出 `int` 范围**。
- **确保所有函数都写了返回值**。
- 大致扫描一遍代码整体，不要出现一些**明显的符号或引用错误**。
- 一旦出现错误，对代码修改后，应**重新过一遍上述流程**。

Rejected 后

- 依情况顺序执行以下步骤：
 - 重新认真地读一遍题目。
 - 肉眼查错/小黄鸭调试法。
 - 构造小样例测试，极限数据测试，卡常。
 - （基本不要使用）对拍。

缺省源

```
1  #include <bits/stdc++.h>
2
3  template <class T>
4  inline void read(T &res)
5  {
6      char ch; bool flag = false; res = 0;
7      while (ch = getchar(), !isdigit(ch) && ch != '-');
8      ch == '-' ? flag = true : res = ch ^ 48;
9      while (ch = getchar(), isdigit(ch))
10         res = res * 10 + ch - 48;
11     flag ? res = -res : 0;
12 }
13
14 template <class T>
15 inline void nonnegative_put(T x)
16 {
17     if (x > 9)
18         nonnegative_put(x / 10);
19     putchar(x % 10 + 48);
20 }
21
22 template <class T>
23 inline void put(T x)
24 {
25     if (x < 0)
26         x = -x, putchar('-');
27     nonnegative_put(x);
28 }
29
30 template <class T>
31 inline void CkMin(T &x, T y) {x > y ? x = y : 0;}
32 template <class T>
33 inline void CkMax(T &x, T y) {x < y ? x = y : 0;}
34 template <class T>
35 inline T Min(T x, T y) {return x < y ? x : y;}
36 template <class T>
37 inline T Max(T x, T y) {return x > y ? x : y;}
38 template <class T>
39 inline T Abs(T x) {return x < 0 ? -x : x;}
40 template <class T>
41 inline T Sqr(T x) {return x * x;}
42 //call Sqr((ll)x) when the type of returned value is "long long".
43
44 using std::map;
45 using std::set;
46 using std::pair;
```

```

47 using std::bitset;
48 using std::string;
49 using std::vector;
50 using std::complex;
51 using std::multiset;
52 using std::priority_queue;
53
54 typedef long long ll;
55 typedef long double ld;
56 typedef complex<ld> com;
57 typedef pair<int, int> pir;
58 const ld pi = acos(-1.0);
59 const ld eps = 1e-8;
60 const int Maxn = 1e9;
61 const int Minn = -1e9;
62 const int mod = 998244353;
63 const int N = 1e5 + 5;
64 int T_data, n;
65
66 inline void solve()
67 {
68
69 }
70
71 int main()
72 {
73     read(T_data);
74     while (T_data--)
75         solve();
76 }

```

关闭同步流

```

1 using std::ios;
2 using std::cin;
3 using std::cout;
4
5 int main()
6 {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     cout.tie(nullptr);
10 }

```

字符串

1. substr 函数：字符串截取函数，用于获取字符串的子串。

```

1 // str.substr(begin, length), 用于截取 str 中以 begin 为下标长度为 length 的字串
2 string s = "asd";
3 s = s.substr(0, 1); // 结果为 "a"

```

2. find 函数：查找字符串中是否存在该字符。

```
1 string s = "asd";
2 int a = s.find('e'); //如果找到就返回 1, 否则返回 -1
```

3. insert 函数：用于添加字符串。

```
1 string& insert(size_t pos, const string&str);
2 // 在位置 pos 处插入字符串 str
3 string& insert(size_t pos, const char *s);
4 // 在位置 pos 处插入字符串 s
5 string& insert(size_t pos, const char *s, size_t n);
6 // 在位置 pos 处插入字符串 s 的前 n 个字符
7 string& insert(size_t pos, size_t n, char c);
8 // 在位置 pos 处插入 n 个字符 c
```

4. erase 函数：用于作字符串删除操作。

```
1 // str.erase(begin, length), 用于删除 str 的从 begin 下标开始长度为 length 的字符串。
2 string str="abc";
3 str.erase(1, 1); // 结果为 "ac"
```

5. replace 函数：用来对字符串的字串作替换处理。

```
1 // str.replace(begin, length, string s), 用于把下标为 begin, 长度为 length 的字串替换为 s。
2 string str = "abc";
3 str = str.replace(1, 2, "lk"); // str = "alk"
4 // str.replace(s.begin(), s.begin() + 3, "aaa"); 给出两个起始位置和终点, 把其中的字符串替换为 "aaa"。
5 // str.replace(1, 3, "123456", 3, 5); 用 "123456" 子串的下标 [3,5] 替换 str 的 [1,3] 的位置。
6 // str.replace(1, 3, 5, 'a'); 用 5 个 'a' 代替 str 的 [1,3] 的位置。
```

6. to_string 函数：用于把非 string 类的数据类型转化为 string 类。

```
1 int a = 134;
2 string s = a.to_string(); // s = "134"
```

7. stoi 函数：把 string 类转化为 int 类型。

```
1 string str = "123";
2 int a = stoi(str);
```

9. 整行读取

```

1 // scanf/getchar()
2 scanf("%d\n", &k); // 注意这个换行符，也可以用单个 getchar 替换
3 while (ch = getchar(), ch != '\n') // gets 在新的 C++17 标准中已经不能使用
4     s[++n] = ch;
5
6 // cin/getline
7 cin >> k;
8 char ch; cin.get(ch);
9 getline(cin, s);

```

随机打乱

```

1 std::mt19937 rng(time(0));
2 std::shuffle(a + 1, a + n + 1, rng);

```

linux 下对拍

- 以 A + B problem 对拍为例，编译器环境为 codeblocks。
- test/main.cpp

```

1 #include <bits/stdc++.h>
2
3 using std::vector;
4
5 int main()
6 {
7     int a, b;
8     freopen("../test_gen/test.in", "r", stdin);
9     freopen("../test_check/test.out", "w", stdout);
10
11     scanf("%d%d", &a, &b);
12     printf("%d\n", a + b);
13
14     fclose(stdin); fclose(stdout);
15     return 0;
16 }

```

- test_bf/main.cpp

```

1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     freopen("../test_gen/test.in", "r", stdin);
8     freopen("../test_check/test_bf.out", "w", stdout);
9
10     int a, b;
11     std::cin >> a >> b;
12     std::cout << a + b << std::endl;
13
14     fclose(stdin); fclose(stdout);
15     return 0;

```

```
16 | }
```

- test_gen/main.cpp

```
1  #include <iostream>
2
3  using namespace std;
4  const int mod = 1e9;
5  int main()
6  {
7      freopen("test.in", "w", stdout);
8
9      srand(time(0));
10     printf("%d %d\n", rand() % mod + 1, rand() % mod + 1); // 64bit 的
ubunutu rand 的最大值一般是  $2^{31} - 1$ 
11     return 0;
12 }
```

- test_check/main.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      while (1)
8      {
9          system("g++ ../test_gen/main.cpp -o test_gen");
10         system("g++ ../test/main.cpp -o test");
11         system("g++ ../test_bf/main.cpp -o test_bf");
12
13         system("./test_gen");
14         system("./test");
15         system("./test_bf");
16
17         if (system("diff test.out test_bf.out"))
18             break ;
19         puts("checking");
20     }
21     return 0;
22 }
```

bitset

C++中__builtin内置函数^Q是GCC、Clang等编译器^Q所提供的一系列高效的内联函数，其中包括许多与二进制^Q相关的函数。下面是所有与二进制相关的__builtin函数：

- __builtin_popcount(x): 返回x的二进制表示中1的个数。

```
unsigned int x = 65535u;
int count = __builtin_popcount(x); // count的值为16
```

- __builtin_clz(x): 返回x的二进制表示中从最高位开始连续0的个数，如果x的值为0，则返回所在类型的位宽。

```
unsigned int x = 0xf00000u;
int count = __builtin_clz(x); // count的值为8
```

- __builtin_ctz(x): 返回x的二进制表示中从最低位开始连续0的个数，如果x的值为0，则返回所在类型的位宽。

```
unsigned int x = 0xf0u;
int count = __builtin_ctz(x); // count的值为4
```

- __builtin_parity(x): 返回x的二进制表示中1的个数是否为奇数^Q，是则返回1，否则返回0。

```
unsigned int x = 0xfu;
int parity = __builtin_parity(x); // parity的值为0
```

CSDN @Log_x

- __builtin_bswap16(x): 将x的二进制表示中的16位进行字节交换。

```
unsigned short x = 0xaabb;
unsigned short y = __builtin_bswap16(x); // y的值为0xbbaa
```

- __builtin_bswap32(x^Q): 将x的二进制表示中的32位进行字节交换。

```
unsigned int x = 0xaabbccddu;
unsigned int y = __builtin_bswap32(x); // y的值为0xddccbbaa
```

- __builtin_bswap64(x): 将x的二进制表示中的64位进行字节交换。

```
unsigned long long x = 0xaabbccddeeff1122ull;
unsigned long long y = __builtin_bswap64(x); // y的值为0x2211ffeeddccbbaa
```

CSDN @Log_x

Python

- 读入多个以空间间隔的整数:

```
1 string = input().split() #a list of strings
2 n = int(string[0])
3 k = int(string[1])
```

- 取整数长度:

```
1 Len = len(str(n))
```

- 格式化输出:

```
1 print(f'hello world {n} {k}')
```

- 数组的简单使用:

```
1 ex = [1]
2 for i in range(1, 101):
3     ex.append(ex[i - 1] * 10)
```

- range(l,r,d) 分别表示起点、终点和步幅:

```
1 >>> list(range(-10, -100, -30))
2 [-10, -40, -70]
```

- 按格式输出

```
1 print('%.2f' % a)
```

高精度

- 仅供参考, 且仅支持非负整数, 有需求优先考虑 Python。

```
1 #include<bits/stdc++.h>
2
3 using std::pair;
4 using std::make_pair;
5 const int N_bigint = 2500;
6 char s[N_bigint * 10];
7
8 template <class T>
9 inline T Max(T x, T y) {return x > y ? x : y;}
10
11 int askLenx(int x)
12 {
13     x = abs(x);
14     int cnt = 0;
15     while (x)
16     {
17         x /= 10;
18         ++cnt;
19     }
```



```

19     }
20     return cnt;
21 }
22
23 struct bigint
24 {
25     typedef long long ll;
26     const ll base = 1e8;
27     ll a[N_bigint];
28     int len;
29
30     void clear() {memset(a, 0, sizeof(a)); a[len = 1] = 0;}
31     bigint() {clear();}
32     bigint(ll x) {*this = x;}
33     bigint operator = (const bigint &b)
34     {
35         memset(a, 0, sizeof(a));
36         len = b.len;
37         for (int i = 1; i <= len; ++i)
38             a[i] = b.a[i];
39         return *this;
40     }
41
42     bigint operator + (const bigint &b) const
43     {
44         int L = Max(len, b.len);
45         bigint tmp;
46         for (int i = 1; i <= L; ++i)
47         {
48             if (i > len)
49                 tmp.a[i] += b.a[i];
50             else if (i > b.len)
51                 tmp.a[i] += a[i];
52             else
53             {
54                 tmp.a[i] += a[i] + b.a[i];
55                 if (tmp.a[i] >= base)
56                 {
57                     tmp.a[i] -= base;
58                     ++tmp.a[i + 1];
59                 }
60             }
61         }
62         if (tmp.a[L + 1]) tmp.len = L + 1;
63         else tmp.len = L;
64         return tmp;
65     }
66     bigint operator - (const bigint &b) const
67     {
68         int L = Max(len, b.len);
69         bigint tmp;
70         for (int i = 1; i <= L; ++i)
71         {
72             tmp.a[i] += a[i] - b.a[i];
73             if (tmp.a[i] < 0)
74             {
75                 tmp.a[i] += base;
76                 --tmp.a[i + 1];

```

```

77     }
78 }
79 while (L > 1 && !tmp.a[L]) --L;
80 tmp.len = L;
81 return tmp;
82 }
83 bigint operator * (const bigint &b) const
84 {
85     int L = len + b.len;
86     bigint tmp;
87     for (int i = 1; i <= len; ++i)
88         for (int j = 1; j <= b.len; ++j)
89         {
90             tmp.a[i + j - 1] += a[i] * b.a[j];
91             if (tmp.a[i + j - 1] >= base)
92             {
93                 tmp.a[i + j] += tmp.a[i + j - 1] / base;
94                 tmp.a[i + j - 1] %= base;
95             }
96         }
97     tmp.len = len + b.len;
98     while (tmp.len > 1 && !tmp.a[tmp.len])
99         --tmp.len;
100    return tmp;
101 }
102 pair<bigint, bigint> Divide(const bigint &a, bigint b) const
103 {
104     int L = a.len; bigint c, d;
105     for (int i = L; i; --i)
106     {
107         c.a[i] = 0;
108         d = d * base;
109         d.a[1] = a.a[i];
110         ll l = 0, r = base - 1, mid;
111         while (l < r)
112         {
113             mid = (l + r + 1) >> 1;
114             if (b * mid <= d) l = mid;
115             else r = mid - 1;
116         }
117         c.a[i] = l;
118         d -= b * l;
119     }
120     while (L > 1 && !c.a[L])
121         --L;
122     c.len = L;
123     return make_pair(c, d);
124 }
125 bigint operator / (ll x) const
126 {
127     ll d = 0; bigint tmp;
128     for (int i = len; i; --i)
129     {
130         d = d * base + a[i];
131         tmp.a[i] = d / x;
132         d %= x;
133     }
134     tmp.len = len;

```

```

135         while (tmp.len > 1 && !tmp.a[tmp.len])
136             --tmp.len;
137         return tmp;
138     }
139     ll operator % (ll x) const
140     {
141         ll d = 0;
142         for (int i = len; i; --i) d = (d * base + a[i]) % x;
143         return d;
144     }
145     bigint operator / (const bigint &b) const {return Divide(*this,
146 b).first;}
147     bigint operator % (const bigint &b) const {return Divide(*this,
148 b).second;}
149     bigint &operator += (const bigint &b) {*this = *this + b; return
150 *this;}
151     bigint &operator -= (const bigint &b) {*this = *this - b; return
152 *this;}
153     bigint &operator *= (const bigint &b) {*this = *this * b; return
154 *this;}
155     bigint &operator ++() {bigint T; T = 1; *this = *this + T; return
156 *this;} //前缀++
157     bigint &operator --() {bigint T; T = 1; *this = *this - T; return
158 *this;} //前缀--
159     bigint operator ++(int) {bigint T, tmp = *this; T = 1; *this = *this +
160 T; return tmp;} //后缀++
161     bigint operator --(int) {bigint T, tmp = *this; T = 1; *this = *this -
162 T; return tmp;} //后缀--
163     bigint operator + (ll x) const {bigint T; T = x; return *this + T;}
164     bigint operator - (ll x) const {bigint T; T = x; return *this - T;}
165     bigint operator * (ll x) const {bigint T; T = x; return *this * T;}
166     bigint operator *= (ll x) {*this = *this * x; return *this;}
167     bigint operator += (ll x) {*this = *this + x; return *this;}
168     bigint operator -= (ll x) {*this = *this - x; return *this;}
169     bigint operator /= (ll x) {*this = *this / x; return *this;}
170     bigint operator %= (ll x) {*this = *this % x; return *this;}
171     bool operator == (ll x) const {bigint T; T = x; return *this == T;}
172     bool operator != (ll x) const {bigint T; T = x; return *this != T;}
173     bool operator <= (ll x) const {bigint T; T = x; return *this <= T;}
174     bool operator >= (ll x) const {bigint T; T = x; return *this >= T;}
175     bool operator < (ll x) const {bigint T; T = x; return *this < T;}
176     bool operator > (ll x) const {bigint T; T = x; return *this > T;}
177     bigint operator = (ll x)
178     {
179         len = 0;
180         while (x)
181             a[++len] = x % base, x /= base;
182         if (!len) a[++len] = 0;
183         return *this;
184     }
185     bool operator < (const bigint &b) const
186     {
187         if (len < b.len) return 1;
188         if (len > b.len) return 0;
189         for (int i = len; i; --i)
190         {
191             if (a[i] < b.a[i]) return 1;
192             if (a[i] > b.a[i]) return 0;
193         }
194         return 0;
195     }

```

```

184     }
185     return 0;
186 }
187 bool operator == (const bigint &b) const
188 {
189     if (len != b.len) return 0;
190     for (int i = len; i; --i)
191         if (a[i] != b.a[i]) return 0;
192     return 1;
193 }
194 bool operator != (const bigint &b) const {return !(*this == b);}
195 bool operator > (const bigint &b) const {return !(*this < b || *this ==
196 b);}
197 bool operator <= (const bigint &b) const {return (*this < b) || (*this
198 == b);}
199 bool operator >= (const bigint &b) const {return (*this > b) || (*this
200 == b);}
201
202 void str(char *s)
203 {
204     int l = strlen(s);
205     ll x = 0, y = 1; len = 0;
206     for (int i = l - 1; i >= 0; --i)
207     {
208         x = x + (s[i] - '0') * y;
209         y *= 10;
210         if (y == base)
211         {
212             a[++len] = x;
213             x = 0;
214             y = 1;
215         }
216     }
217     if (!len || x)
218         a[++len] = x;
219 }
220 void read()
221 {
222     scanf("%s", s);
223     this->str(s);
224 }
225 void print() const
226 {
227     printf("%d", (int)a[len]);
228     for (int i = len - 1; i; --i)
229     {
230         for (int j = base / 10; j >= 10; j /= 10)
231         {
232             if (a[i] < j) putchar('0');
233             else break;
234         }
235         printf("%d", (int)a[i]);
236     }
237     putchar('\n');
238 }
239 int askLen() const
240 {
241     return (len - 1) * 8 + askLenx(a[len]);

```

```
239     }
240 }a, b;
241
242 int main()
243 {
244     a.read();
245     b.read();
246     (a + b).print();
247     if (a >= b)
248         (a - b).print();
249     else
250     {
251         putchar('-');
252         (b - a).print();
253     }
254     (a * b).print();
255     pair<bigint, bigint> t = a.Divide(a, b);
256     t.first.print(); t.second.print();
257 }
```