C++补充板子

dev编译选项 `-std=c++11`

计时器

```
#include<bits/stdc++.h>
#include <windows.h>
#include <chrono>
using namespace std::chrono;//C++11 chrono库
using namespace std;
int main()
{
    /*double start = clock();
    Sleep(100);
    double end = clock();
    double last = end - start;
    cout << last << "ms" << endl;*/
    auto start = steady_clock::now();
    Sleep(100);
    auto end = steady_clock::now();
    auto last = duration_cast<microseconds>(end - start);
    cout <<  last.count() << "um";
    return 0;
}
```

快速（模）幂

```
long long binpow(long long a, long long b) {
  long long res = 1;
  while (b > 0) {
    if (b & 1) res = res * a;
    a = a * a;
    b >>= 1;
  }
  return res;
}

long long binpowmod(long long a, long long b, long long m) {
  a %= m;
  long long res = 1;
  while (b > 0) {
    if (b & 1) res = res * a % m;
    a = a * a % m;
    b >>= 1;
  }
```

```
    return res;
}
```

牛顿迭代

```
#include<iostream>
#include<cmath>
using namespace std;

double x,X;  //定义变量x//临时变量X，用于判断精度
double y(double x){          //定义关于x的表达式
    return x*x-10241024;     //y(x)=0
}
double dy(double x){     //定义导数公式y'
    return 2*x;
}
bool accuracy(double x0){
    if (fabs(X - x0) > 1e-8)return 1;  //定义精度
    else return 0;
}
void ND(double x0){     //牛顿迭代
    do{
        double _y = y(x0),_dy = dy(x0);
        X = x0;
        x0 = x0 - _y / _dy;
    } while (accuracy(x0));
    printf("%lf\n",x0);
}
void ND_optimize(double x1,double x0){//牛顿迭代法差商改进
    do{
        double _y1 = y(x0),_y2 = y(x1);
        X = x0;
        x0 = x0 - (_y1 / (_y1-_y2)) * (x0-x1);
    } while (accuracy(x0));
    printf("%lf\n",x0);
}
int main(){
    x = 10;              //定义x0初始值
  //ND(x);               //已知导数
  ND_optimize(x,x+0.1);//未知导数
  return 0;
}
```

__int128

仅 64 位 GCCG++ 支持，不在 C++ 标准中！有符号128位整数变量，最多表示**39位**整数。 支持基本的**加减乘除**运算，以及**按位与&、按位或|、按位异或^、左移<<、右移>>**。不需要快读时可以通过%*/完成读写。

```
inline __int128 read(){
    int x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9'){
        if(ch=='-') f=-1;
        ch=getchar();}
    while(ch>='0'&&ch<='9'){
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();}
    return x*f;
}
inline void write(__int128 x){
    if(x<0){putchar('-');x=-x;}
    if(x>9) write(x/10);
    putchar(x%10+'0');
}
```

高精度

```
#pragma once
#include<bits/stdc++.h>
#include <algorithm>
#include <string>
#include <vector>

struct BigIntTiny {
    int sign;
    std::vector<int> v;
    BigIntTiny() : sign(1) {}
    BigIntTiny(const std::string &s) { *this = s; }
    BigIntTiny(int v) {
        char buf[21];
        sprintf(buf, "%d", v);
        *this = buf;
    }
    void zip(int unzip) {
        if (unzip == 0) {
            for (int i = 0; i < (int)v.size(); i++)
                v[i] = get_pos(i * 4) + get_pos(i * 4 + 1) * 10 + get_pos(i *
4 + 2) * 100 + get_pos(i * 4 + 3) * 1000;
        } else
            for (int i = (v.resize(v.size() * 4), (int)v.size() - 1), a; i >=
0; i--)
                a = (i % 4 >= 2) ? v[i / 4] / 100 : v[i / 4] % 100, v[i] = (i
& 1) ? a / 10 : a % 10;
        setsign(1, 1);
    }
    int get_pos(unsigned pos) const { return pos >= v.size() ? 0 : v[pos]; }
    BigIntTiny &setsign(int newsign, int rev) {
        for (int i = (int)v.size() - 1; i > 0 && v[i] == 0; i--)
```

```
            v.erase(v.begin() + i);
        sign = (v.size() == 0 || (v.size() == 1 && v[0] == 0)) ? 1 : (rev ?
newsign * sign : newsign);
        return *this;
    }
    std::string to_str() const {
        BigIntTiny b = *this;
        std::string s;
        for (int i = (b.zip(1), 0); i < (int)b.v.size(); ++i)
            s += char(*(b.v.rbegin() + i) + '0');
        return (sign < 0 ? "-" : "") + (s.empty() ? std::string("0") : s);
    }
    bool absless(const BigIntTiny &b) const {
        if (v.size() != b.v.size()) return v.size() < b.v.size();
        for (int i = (int)v.size() - 1; i >= 0; i--)
            if (v[i] != b.v[i]) return v[i] < b.v[i];
        return false;
    }
    BigIntTiny operator-() const {
        BigIntTiny c = *this;
        c.sign = (v.size() > 1 || v[0]) ? -c.sign : 1;
        return c;
    }
    BigIntTiny &operator=(const std::string &s) {
        if (s[0] == '-')
            *this = s.substr(1);
        else {
            for (int i = (v.clear(), 0); i < (int)s.size(); ++i)
                v.push_back(*(s.rbegin() + i) - '0');
            zip(0);
        }
        return setsign(s[0] == '-' ? -1 : 1, sign = 1);
    }
    bool operator<(const BigIntTiny &b) const {
        return sign != b.sign ? sign < b.sign : (sign == 1 ? absless(b) :
b.absless(*this));
    }
    bool operator==(const BigIntTiny &b) const { return v == b.v && sign ==
b.sign; }
    BigIntTiny &operator+=(const BigIntTiny &b) {
        if (sign != b.sign) return *this = (*this) - -b;
        v.resize(std::max(v.size(), b.v.size()) + 1);
        for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
            carry += v[i] + b.get_pos(i);
            v[i] = carry % 10000, carry /= 10000;
        }
        return setsign(sign, 0);
    }
    BigIntTiny operator+(const BigIntTiny &b) const {
        BigIntTiny c = *this;
        return c += b;
    }
    void add_mul(const BigIntTiny &b, int mul) {
        v.resize(std::max(v.size(), b.v.size()) + 2);
```

```
        for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
            carry += v[i] + b.get_pos(i) * mul;
            v[i] = carry % 10000, carry /= 10000;
        }
    }
    BigIntTiny operator-(const BigIntTiny &b) const {
        if (b.v.empty() || b.v.size() == 1 && b.v[0] == 0) return *this;
        if (sign != b.sign) return (*this) + -b;
        if (absless(b)) return -(b - *this);
        BigIntTiny c;
        for (int i = 0, borrow = 0; i < (int)v.size(); i++) {
            borrow += v[i] - b.get_pos(i);
            c.v.push_back(borrow);
            c.v.back() -= 10000 * (borrow >>= 31);
        }
        return c.setsign(sign, 0);
    }
    BigIntTiny operator*(const BigIntTiny &b) const {
        if (b < *this) return b * *this;
        BigIntTiny c, d = b;
        for (int i = 0; i < (int)v.size(); i++, d.v.insert(d.v.begin(), 0))
            c.add_mul(d, v[i]);
        return c.setsign(sign * b.sign, 0);
    }
    BigIntTiny operator/(const BigIntTiny &b) const {
        BigIntTiny c, d;
        BigIntTiny e=b;
        e.sign=1;

        d.v.resize(v.size());
        double db = 1.0 / (b.v.back() + (b.get_pos((unsigned)b.v.size() - 2) /
1e4) +
                           (b.get_pos((unsigned)b.v.size() - 3) + 1) / 1e8);
        for (int i = (int)v.size() - 1; i >= 0; i--) {
            c.v.insert(c.v.begin(), v[i]);
            int m = (int)((c.get_pos((int)e.v.size()) * 10000 +
c.get_pos((int)e.v.size() - 1)) * db);
            c = c - e * m, c.setsign(c.sign, 0), d.v[i] += m;
            while (!(c < e))
                c = c - e, d.v[i] += 1;
        }
        return d.setsign(sign * b.sign, 0);
    }
    BigIntTiny operator%(const BigIntTiny &b) const { return *this - *this / b
* b; }
    bool operator>(const BigIntTiny &b) const { return b < *this; }
    bool operator<=(const BigIntTiny &b) const { return !(b < *this); }
    bool operator>=(const BigIntTiny &b) const { return !(*this < b); }
    bool operator!=(const BigIntTiny &b) const { return !(*this == b); }
};
void testBigIntTiny() {
    // 测试构造函数
    BigIntTiny zero;
    std::cout << zero.to_str() << std::endl;
```

```
    BigIntTiny one(1);
    std::cout << one.to_str() << std::endl;
    BigIntTiny negOne(-1);
    std::cout << negOne.to_str() << std::endl;
    BigIntTiny fromStr("123456789");
    std::cout << fromStr.to_str() << std::endl;
    // 测试加法
    BigIntTiny a("12345");
    BigIntTiny b("67890");
    BigIntTiny sum = a + b;
    std::cout<<sum.to_str()<< std::endl;
    // 测试减法
    BigIntTiny diff = a - b;
    std::cout<<diff.to_str()<< std::endl;
    // 测试乘法
    BigIntTiny product = a * b;
    std::cout<<product.to_str()<<std::endl;
    // 测试除法
    BigIntTiny quotient = product / b;
    std::cout<<quotient.to_str()<<std::endl;
    // 测试取模
    BigIntTiny remainder = product % b;
    std::cout<<remainder.to_str()<<std::endl;
    // 测试比较运算符
    std::cout<<(a < b)<<std::endl;
    // 测试负数运算
    BigIntTiny negA = -a;
    std::cout<<negA.to_str()<<std::endl;
}
int main() {
    testBigIntTiny();
    return 0;
}
```