Foundations of Computer Algebra: Assignment 2 (version with elements of correction)

Marc Moreno Maza
University of Western Ontario
CS424 / CS556 – Winter 2006

Posted: Wednesday, February 8, 2006
Due: Friday, February 24, 2006

1 Guidelines

Date of postage. Wednesday, February 8, 2006.

Normal date for submission. Friday, February 24, 2006.

Due format. The assignment can be returned to the instructor by email in a form of a postscript, PDF or text file. Alternatively, a printed version can be delivered to his mail box in the Computer Science Department or given to him in class.

Collaboration. You are expected to do this assignment on your own without assistance from anyone else in the class. However, you can use the literature, and in particular the book *Modern Computer Algebra* and the references therein.

Marking. This assignment will be marked out of 40. Each exercise is worth 10 points.

2 Exercise 1

2.1 Statement

Let f and g in $\mathbb{Z}[x]$ be two polynomials with respective degrees $m \geq n > 0$. We assume that g is monic, that is, its leading coefficient is 1. Let B_f and B_g be upper bounds for the absolute value of a coefficient in f and g respectively. We aim at computing the quotient q and the remainder r in the division of f by g by means of a modular method.

2.2 Questions

(1) Give an upper bound B_q for the absolute value of a coefficient in the quotient q.

(2) Describe a modular algorithm for computing q and r. No complexity analysis is required.

2.3 Elements of correction

(1) Define:

$$f = f_m x^m + \cdots + f_1 x + f_0 \quad \text{and} \quad g = x^n + \cdots + g_1 x + g_0. \tag{1}$$

The following discussion is meant to give upper bounds for the quotient q and the remainder r in the division-with-remainder of f by g. We follow the classical algorithm (not the fast one) as stated at the beginning of the chapter on fast division.

Observe that the quotient q has degree $m - n$, so we define:

$$q = q_{m-n} x^{m-n} + \cdots + q_1 x + q_0. \tag{2}$$

Since g is monic, we clearly have:

$$q_{m-n} = f_m. \tag{3}$$

Let $r_0 = f$ be the "initial" remainder and let $C_0 = B_f$ be the largest absolute value of a coefficient in r_0. During the first iteration r_0 is replaced by

$$r_1 = r_0 - q_{m-n} g x^{m-n}. \tag{4}$$

The "intermediate" remainder r_1 is either null or C_1 the largest absolute value of a coefficient in r_1 satisfies:

$$C_1 \leq C_0 + C_0 B_g = C_0(1 + B_g) \tag{5}$$

If $r_1 \neq 0$, then the next intermediate remainder r_2 satisfies

$$r_2 = r_1 - \mathrm{lc}(r_1) g x^{\deg(r_1)-n} \tag{6}$$

where $\mathrm{lc}(r_1)$ is the leading coefficient of r_1 and, also, the next non-zero coefficient in q. If $r_2 \neq 0$, then C_2 the largest absolute value of a coefficient in r_2 satisfies:

$$C_2 \leq C_1 + C_1 B_g \leq C_0(1 + B_g)^2. \tag{7}$$

Then, an easy induction leads to:

$$B_q \leq B_f(1 + B_g)^{m-n}. \tag{8}$$

Observe that an upper bound for the absolute value of a coefficient in r is rather

$$B_r \leq B_f(1 + B_g)^{m-n+1}. \tag{9}$$

(2) We make a preliminary observation. Let $m \geq 1$ be an odd integer such that we have

$$m/2 > B_f(1 + B_g)^{m-n+1}. \tag{10}$$

Note that this implies

$$m/2 > B_f, \;\; m/2 > B_g \;\text{ and }\; m/2 > B_q. \tag{11}$$

We represent the elements of $\mathbb{Z}/m\mathbb{Z}$ with the symmetric range $-\frac{m-1}{2} \cdots \frac{m-1}{2}$. Let Ψ_m be the corresponding canonical ring homomorphism from $\mathbb{Z}[x]$ to $\mathbb{Z}/m\mathbb{Z}[x]$. In $\mathbb{Z}[x]$ we have:

$$f = qg + r \;\text{ and }\; \deg(r) < \deg(g). \tag{12}$$

This implies the following in $\mathbb{Z}/m\mathbb{Z}[x]$:

$$\Psi_m(f) = \Psi_m(q)\Psi_m(g) + \Psi_m(r) \;\text{ and }\; \deg(\Psi_m(r)) < \deg(\Psi_m(g)). \tag{13}$$

Observe that we have:

$$\Psi_m(f) = f,\ \Psi_m(g) = g,\ \Psi_m(q) = q \text{ and } \Psi_m(r) = r. \tag{14}$$

We shall describe now how to compute $\Psi_m(q)$ and $\Psi_m(r)$, and thus q and r.

Consider $p_1 < \cdots < p_r$ odd primes such that their product m satisfies (10). For all $1 \le i \le r$, let f_{p_i}, g_{p_i}, q_{p_i} and r_{p_i} be the respective images of f, g, q and r in $\mathbb{Z}/p_i\mathbb{Z}[x]$. Thus, for all $1 \le i \le r$, we have in $\mathbb{Z}/p_i\mathbb{Z}[x]$

$$f_{p_i} = g_{p_i} q_{p_i} + r_{p_i} \text{ and } \deg(r_{p_i}) < \deg(g_{p_i}). \tag{15}$$

By virtue of the uniqueness of the division by a monic polynomial, this implies that for all $1 \le i \le r$, the couple (q_{p_i}, r_{p_i}) is the couple quotient-remainder in the division of f_{p_i} by g_{p_i}. Applying coefficient-wise the Chinese Remaindering Algorithm (see course notes) we construct polynomials f_m, g_m, q_m and r_m in $\mathbb{Z}/m\mathbb{Z}[x]$ such that

$$f_m = g_m q_m + r_m \text{ and } \deg(r_m) < \deg(g_m). \tag{16}$$

Moreover, the Chinese Remaindering Theorem tells us that we have:

$$f_m = \Psi_m(f), g_m = \Psi_m(g), q_m = \Psi_m(q) \text{ and } r_m = \Psi_m(r). \tag{17}$$

Using the preliminary remark, we conclude that $q_m = q$ and $r_m = r$. This approach is interesting when fast division is available in each $\mathbb{Z}/p_i\mathbb{Z}[x]$.

To summarize the key ideas for this exercise are:

- $B_f(1+B_g)^{m-n}$ is an upper bound for (the absolute value of a coefficient in) the quotient
- for designing a modular method that reconstruct both q and r we can use the bound $B_f(1+B_g)^{m-n+1}$; alternatively, we can
  - reconstruct the quotient q only and then use the smaller bound $B_f(1+B_g)^{m-n}$,
  - then, get r as $f - qg$.

- the modular reconstruction relies on
  - the correct choice of the product m of the primes, which must be greater than twice the bound,
  - the Chinese Remaindering Theorem and Algorithm,
  - the uniqueness of the division-with-remainder by a monic polynomial.

3 Exercise 2

3.1 Statement

Let f and g in $\mathbb{Z}[x]$ be two polynomials of degree at most $d > 0$. Let B be an upper bound for the absolute value of a coefficient in f or g. Then $C = (d+1)B^2$ is an upper bound for the absolute value of a coefficient in the product fg. We assume that we have r primes numbers $2 < p_1 < \cdots < p_r$ such that their product m exceeds $2C$ In addition, we assume that

- every prime $p_1, \ldots, p_r$ fits in a machine word,
- computing the $fg \bmod p_i$, for all $i = 1, \ldots, r$, can be done $O(d\log(d)\text{loglog}(d))$ machine word operations.

3.2 Questions

(1) Justify the formula for C.

(2) Describe a modular algorithm for computing fg in $\mathbb{Z}[x]$.

(3) Give an upper bound for the number of machine word operations requires by this algorithm.

3.3 Elements of correction

(1) Define $h = fg$ in $\mathbb{Z}[x]$. Assume that f and g have degree d and define:

$$f = f_d x^d + \cdots + f_1 x + f_0 \quad \text{and} \quad g = g_d x^d + \cdots + g_1 x + g_0. \tag{18}$$

Then define:

$$h = c_{2d} x^{2d} + \cdots + c_1 x + c_0. \tag{19}$$

For all $0 \leq k \leq 2d$, we have:

$$c_k = \sum\nolimits_{i+j=k} f_i g_j. \tag{20}$$

Observe that, for a given k in the range $0 \cdots 2d$, the number of products $f_i g_j$ is at most $d+1$. Indeed, the polynomials f and g have $d+1$ terms. From there, we deduce that for all $0 \leq k \leq 2d$, we have:

$$\mid c_k \mid \leq \ (d+1)B^2. \tag{21}$$

(2) One can proceed as follows:

  (a) Let $p_1 < \cdots < p_r$ machine-word-size odd primes such that their product m exceeds $2C$

  (b) For all $i = 1 \cdots r$ compute

    ($b1$) the images f_i and g_i of f and g in $\mathbb{Z}/p_i\mathbb{Z}[x]$,

    ($b2$) the product $h_i = f_i\, g_i$ in $\mathbb{Z}/p_i\mathbb{Z}[x]$.

  (c) Using the Chinese Remaindering Algorithm coefficient-wise, reconstruct $f\,g$ from $h_1, \ldots, h_r$.

(3) We give complexity estimates (upper bounds for the number of machine-word operations) for each of the steps ($b1$), ($b2$) and (c) above. For the operations on integers, we use bounds based on classical quadratic algorithms (not fast ones, since they were not discussed in class). Let N be the length of a machine-word, let $\ell_B = \lfloor \log_2(B/N) \rfloor + 1$ be the number of machine-words to write B and let $\ell_C = \lfloor \log_2(C/N) \rfloor + 1$ to write C.

  ($b1$) $2\,r\,(d+1)\,O({\ell_B}^2)$ machine-word operations

  ($b2$) $r\,O(d \log(d) \mathrm{loglog}(d))$ machine-word operations

  (c) $(2d+1)O({\ell_C}^2)$ machine-word operations

4 Exercise 3

4.1 Statement

Let f and g in $\mathbb{Z}[x]$ be two polynomials with respective degrees $m \geq n > 0$. We assume that the leading coefficient $\mathrm{lc}(g)$ of g divides the leading coefficient

$\mathrm{lc}(f)$ of f. We want to decide whether g divides f exactly, that is, whether there exists a polynomial $q \in \mathbb{Z}[x]$ such that $f = qg$ holds. Moreover, if g divides f exactly, we want to compute the quotient q.

Another *obvious* necessary condition (for g to divide f exactly) is that the trailing coefficient of g (that is the coefficient of g of its non-zero term with smallest degree) divides that of f. So, we assume that this condition holds too.

If f and g are *random enough* the probability for g to divide f exactly is clearly low. So we aim at designing a modular method that will take advantage of this remark.

4.2 Questions

(1) Design a modular modular algorithm based on the following observation: if g divides f then $g(z)$ divides $f(z)$ for every $z \in \mathbb{Z}$.

(2) Assume that we are given an upper bound for the absolute value of a coefficient in any polynomial h dividing f. (Such upper bound exists and will be discussed in class.) Design a modular modular algorithm based on the following observation: if g divides f (in $\mathbb{Z}[x]$) then g divides f modulo any prime number p.

(3) No complexity analysis is required. However, you are asked to suggest why the second algorithm is likely to be more efficient that the first one.

4.3 Elements of correction

(1) Assume that g divides f and let q be the quotient. The degree of q is $m - n$. Viewing f, g and q as polynomials in $\mathbb{Q}[x]$ one can use Lagrange interpolation to construct q from its values at $0, 1, \ldots, m - n$. This idea leads to the following algorithm.

   (a) Let $i = 0$.

   (b) While $i \leq m - n$ repeat

      $(b1)$ Compute $f(i)$ and $g(i)$.

      $(b2)$ If $g(i)$ does not divide $f(i)$ in $\mathbb{Z}$ then return "g does not divide f" otherwise define $q(i) = f(i)/g(i)$.

($b3$) $i = i + 1$.

(c) Interpolate $(0, q(0)), \ldots, (m-n, q(m-n))$ in order to get q and return it.

(2) Assume again that g divides f and let q be the quotient. This means that $f = gq$ in $\mathbb{Z}[x]$. Let C be an upper for the absolute value of a coefficient in q. (Actually, one can adapt Exercise 1 in order to obtain such a bound). Let p be a prime number. Then, we have $f \equiv gq \mod p$. This idea leads to the following algorithm.

(a) Let $p_1 < \cdots < p_r$ be machine-word-size odd primes such that their product m exceeds $2C$

(b) For all $i = 1 \cdots r$ repeat:

($b1$) compute the images f_i and g_i of f and g in $\mathbb{Z}/p_i\mathbb{Z}[x]$,

($b2$) compute the quotient q_i and remainder r_i in the division of f_i by g_i in $\mathbb{Z}/p_i\mathbb{Z}[x]$,

($b3$) if $r_i \neq 0$ then return "g does not divide f"

(c) Use the Chinese Remainder Algorithm coefficient-wise in order to construct a polynomial q from $q_1, \ldots, q_r$.

(d) If $f - qg \neq 0$ then return "g does not divide f" otherwise return q.

The *checking statement* (d) can be avoided but this requires additional material outside of the scope of this course.

(3) The second approach is probably better since it is a modular method relying on modular computations over machine-word-size finite fields. But, of course, the bound C has to be sharp.

5 Exercise 4

5.1 Statement

Let $\mathbb{K}$ be a field. We assume that we have at our disposal a fast algorithm for multiplications in $\mathbb{K}[x_1]$. Let $M(d)$ be an upper bound for the number of operations in $\mathbb{K}$ required for computing the product of two polynomials $\mathbb{K}[x_1]$ with degree at most d.

We would like to derive a fast algorithm for multiplying bivariate polynomials in $\mathbb{K}[x_1, x_2]$. To do so, we consider the following transformation.

Let $\alpha_2 > 0$ be a positive integer. Given a bivariate polynomial f we replace every monomial $x_1^{e_1} x_2^{e_2}$ of f by $x_1^{e_1+\alpha_2 e_2}$ obtaining a univariate polynomial u_f. Let d_1 be the degree of f w.r.t. x_1 and d_2 its degree w.r.t. x_2,

5.2 Questions

(1) Let $f = x_1 x_2 + x_2^2 + x_1^2$. Compute u_f for $\alpha_2 = 2$, and then for $\alpha_2 = 3$.

(2) More generally, how to choose α_2 such that f is the only pre-image of u_f ?

(3) Consider now two bivariate polynomials f and g with respective x_1-degrees d_1 and c_1. How to choose α_2 such that fg is the only pre-image of u_{fg} ?

(4) Describe an algorithm reducing the computation of the product fg to a multiplication in $\mathbb{K}[x_1]$. Give an upper bound for its running time in terms of number of operations in $\mathbb{K}$.

5.3 Elements of correction

(1) With $\alpha_2 = 2$, we have $u_f = x_1^3 + x_1^4 + x_1^2$ and with $\alpha_2 = 3$, we have $u_f = x_1^4 + x_1^6 + x_1^2$.

(2) $\alpha_2 \geq \deg(f, x_1) + 1$.

(3) $\alpha_2 \geq d_1 + c_1 + 1$.

(4) Using $\alpha_2 = d_1 + c_1 + 1$ compute u_f and u_g the respective images of f and g, then the product $u_f u_g$ in $\mathbb{K}[x_1]$, then its (unique) preimage fg. It is easy to check that the transformations $f \to u_f$, $g \to u_g$, and $u_f u_g \to fg$ have a linear cost in the number of terms of their results. So the dominant cost is the computation of the product $u_f u_g$ which is clearly in $M((\deg(fg, x_1)+1)\deg(fg, x_2)) = M(\alpha_2 \deg(fg, x_2))$ operations in $\mathbb{K}$.

6 Hints

- For both Exercises 1 and 2, looking at Exercise 5.34 from [GG99] should be helpful.
- For computing the bound B_q, looking at Exercise 2 of Quiz 1 of last year can be helpful too. This bound should look like $B_f B_g^{m-n+1}$.
- For Exercise 1, it is important to remember the following fact. Let a and b be two polynomials in $\mathbb{Z}[x]$ or $\mathbb{Z}/m\mathbb{Z}[x]$ (for some integer $m > 1$, prime or not) such that the leading coefficient of b is one. Then, there exists a **unique** couple (q, r) such that $a = bq + r$ and $\deg(r) < \deg(b)$. The algorithm for the division is in the first section of the Chapter *From Newton to Hensel* on our web site.

References

[BL82] Bruno Buchberger and Rüdiger Georg Konrad Loos. *Algebraic Simplification*, pages 11–43. Springer Verlag, Wien–New York, 1982.

[GG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[Moe76] Robert T. Moenck. Practical fast polynomial multiplication. In *SYMSAC '76: Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 136–148, New York, NY, USA, 1976. ACM Press.

[TCS02] R.L. Rivest T.H. Cormen, C.E. Leiserson and C. Stein. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2002.