TUTORIAL 10

1 Newton's iteration and integer roots

Newton's iteration is a fondamental tool in several topics (optimal transport, numerical analysis, ...). It can be seen as an iterative process to approximate a solution of $f(x) = 0$, where f is, *e.g.* , a given $\mathcal{C}^1$- real valued function. Starting from an initial condition x_0, one defines

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

Under some reasonable assumptions on f, it can be shown that the process converges quadratically fast to a solution. With a bit more work, one can show that Euclidean division of two integers of bitsize n can be done in $O(\mathrm{M}(n))$: we will use this result in the exercise.

We will also show that Newton's iteration can also be used with another notion of convergence to compute integer roots of integer polynomials.

1. Write down Newton's iteration to compute the inverse of a given $a \in \mathbb{Z}$. Use it to show that, if you start from an inverse of a modulo some integer m, the next iteration gives you an inverse of a modulo m^2. Deduce an algorithm to compute inverses modulo p^v, for some prime p and an even[1] integer $v \geq 1$, and give its complexity.

   A: *We want to solve $\frac{1}{x} - a = 0$, starting from some initial condition b_0. We then have $b_{i+1} = b_i - \frac{1/b_i - a}{-1/b_i^2} = 2b_i - b_i^2 a$. Now assuming $b_i \cdot a \equiv 1\ [m]$, we have $1 - b_{i+1}a = 1 - 2b_i a + b_i^2 a^2 = (1 - b_i a_i)^2 \equiv 0\ [m^2]$, which gives the claim.*

   We could compute inverse modulo p^v using EEA in $O(\log_2^2(p^v))$ operations. However it is less efficient when v is large than the following algorithm, based on Newton's iteration. In a first step, compute an inverse b_0 modulo p with EEA in $O(\log^2 p)$ operations. Then for i from 1 to $v-1$, let $b_0 \leftarrow (2b_0 - b_0^2 a) \bmod p^{i+1}$, which naively cost 3 multiplications and one modular reduction: everything can be done in $O(\mathrm{M}(\log_2(p^v)))$. This gives $O(\log_2^2 p + M(v \log_2 p))$ operations.

2. Let $P \in \mathbb{Z}[X]$, and assume that $a_i \in \mathbb{Z}$ such that $P(a_i) \equiv 0\ [m]$ and $P'(a_i) \in (\mathbb{Z}/m\mathbb{Z})^\times$ for some integer m. Define the next "modular" Newton iterate a_{i+1}, and show that it satisfies the following properties:

   - $a_{i+1} \equiv a_i\ [m]$;
   - $P(a_{i+1}) \equiv 0\ [m^2]$;
   - $P'(a_{i+1}) \in (\mathbb{Z}/m^2\mathbb{Z})^\times$.

[1]For the sake of simplicity. An analogous algorithm can be designed in the general case (we leave it as an exercise).

A: *The iteration defines $a_{i+1} = a_i - P(a_i)P'(a_i)^{-1} \bmod m^2$. This directly gives the first property, which also implies that $(a_{i+1} - a_i)^2 \equiv 0\ [m^2]$. From Taylor expansion for polynomials, we can write $P(a_{i+1}) = P(a_i) + P'(a_i)(a_{i+1} - a_i) + Q(a_i)(a_{i+1} - a_i)^2$, for some polynomial Q. By definition of the iterate, we have $-P'(a_i)(a_{i+1} - a_i) \equiv P(a_i)\ [m^2]$; we obtain the second claim using the previous remark. Lastly, the fact that $a_{i+1} \equiv a_i\ [m]$ and that the reductio modulo m is a ring homomorphism imply that, for any polynomial $Q \in \mathbb{Z}[X]$, we have $Q(a_{i+1}) \equiv Q(a_i)\ [m]$. In particular, $P'(a_{i+1}) \equiv P'(a_i)\ [m]$, so that $P'(a_{i+1})$ is invertible modulo m. In other words, it is coprime to m, hence coprime to m^2, and the last claim follows.*

The next question deals with the unicity of the solution given by the above Newton iteration process, assuming b_0 is a starting value (thus $P(b_0) \equiv 0\ [p]$ and $P'(b_0)$ is invertible modulo p).

3. (**Uniqueness property**) If b, b' are integers such that $b \equiv b_0 \equiv b'\ [p]$ and $P(b) \equiv P(b')\ [p^{2^i}]$, show that $b \equiv b'\ [p^{2^i}]$.

   A: *Using again Taylor expansion, we may find a polynomial Q such that $P(b') - P(b) = (b - b')(P'(b) - Q(b)(b - b'))$. By assumptions, $P'(b) + Q(b)(b - b') \equiv P'(b) \equiv P'(b_0)\ [p]$. In particular, the left hand side is coprime to p, hence invertible modulo p^{2^i}. Let u be said inverse, so that $u(P(b) - P(b')) \equiv b - b'\ [p^{2^i}]$. The claim follows as the left hand side vanishes.*

4. What is missing to design a general algorithm to compute "modular" roots of integer polynomials? Assuming this problem can be dealt with, write down an algorithm that compute modular roots of integer polynomials and give its complexity.

   A: *By CRT, the problem amounts to computing roots modulo prime powers. However, to start Newton's iteration, we require a root modulo p. When p is small, a root can be computed by "brute force", just as we did above. In general, there are well-known root finding algorithms over field as $\mathbb{Z}/p\mathbb{Z}$, that were not covered in the lectures this year. Go see Berlekamp's or Cantor-Zassenhaus'.*

   If we can find a starting root, we just iterate Newton's until the exponent is reached. At each step, we also update the current inverse for the derivative. This gives the following:

```
• $a_0 \leftarrow$ Root of $P$ mod $p$;
• $u_0 \leftarrow$ ModInverse$(P'(a_0), p)$;
• for i from $1$ to $v - 1$ do
    – $a_i \leftarrow a_{i-1} - P(a_{i-1})u_i \bmod p^{2^i}$, and $u_i \leftarrow 2u_i - u_i^2 P'(a_i) \bmod p^{2^i}$
• outputs $a_{v-1}$.
```

   To evaluate the complexity, we use the fact that, by assumption, reduction modulo p^v costs $O(\mathrm{M}(v \log p))$. In particular, this means that "computation mod p^v" can be done in this cost: reduce operands, do the needed operation, reduce the result. In particular, evaluating a polynomial of degree d cost $O(d\mathrm{M}(v \log p))$, which turns out to be a bound for any step in the above algorithm. There are $v = \log p$ steps, hence the total complexity is $O(\log p \cdot \deg P\mathrm{M}(v \log p))$.

2 Evaluating the derivatives of a polynomial

In this exercise we are give a degree n polynomial $P \in K[X]$, for some field K, and $a \in K$. Our goal is to evaluate all the derivative of P at a.

1. Give a naive algorithm to compute $P^{(i)}(a)$ for all $i \in \{0, \dots, n\}$. What is its complexity?

   A: *Compute $P^{(i)}$ for all i. It take $O(i)$ multiplications to obtain $P^{(i+1)}$ from $P^{(i)}$, making this $O(n^2)$ multiplications in total. Each evalutation can be done in $O(i)$ as well. The total complexity is therefore $O(n^2)$.*

2. If $a = 0$, give an algorithm that computes all $P^{(i)}(0)$'s in linear time.

   A: *We have $P^{(i)}(O) = i!p_i$. From $P^{(i-1)}(0)$, computing this costs 2 multiplications ($i! = (i-1)! \cdot i$ and $P^{(i)}(0) = i! \cdot p_i$.) We compute all needed evaluation in $O(n)$.*

3. Show that $Q(X) = P(X + a)$ can be computed in quasi-linear time. (Hint: Consider the Euclidean division $P(X) = Q(X)(X - a)^{\deg P/2} + R(X)$, and apply a recursive approach.)

   A: *We have $P(X + a) = Q(X + a)X^{\deg P/2} + R(X + a)$, with $\deg Q, \deg R < n/2$. A divide-and-conquer approach seems in order:*

   - *Compute Euclidan Division of P by $(X - a)^{\deg P/2}$ in $O(\mathrm{M}(n))$, and obtain Q, R.*
   - *Call recursively the algorithm on Q, R.*
   - *Return $Q(X + a) * X^{\deg P/2} + R(X + a)$.*

   The last step can be done in $O(n)$ by shifting-add. The complexity of the algorithm follows $T(n) = 2T(n/2) + O(\mathrm{M}(n))$, from wich we obtain $T(n) = O(\mathrm{M}(n) \log n)$.

4. Conclude with a quasi-linear time algorithm to compute all $P^{(i)}(a)$'s.

   A: *First compute $Q(X) = P(X + a)$ in quasi-linear time, then evaluate $Q^{(i)}(0)$ for all i in linear time.*

3 Time to read your lecture notes

1. Let K be a field. Fill the following table **without** using fast algorithms.

	$\mathbb{Z}$	$K[X]$
Unit operation in complexity		
Input size		
Addition/subtraction		
Multiplication		
Euclidean division		
Extended Euclidean Algorithm		
Multipoint evaluation		
Interpolation/CRT		

2. Let $\mathrm{M}(n)$ be the complexity of multiplying two integers of size n, and $\mathrm{P}(n)$ be the complexity of multiplying two polynomials of degree n. What are the best values you know for each?

A: M$(n) = n\log n \log\log n$ *and* P$(n) = n\log n$.

3. What is the complexity of adding/multiplying polynomials in $\mathbb{Z}_p[X]$ in terms of bit operations (and using naive algorithms)?

A: *The bitsize of a degree n polynomial is $O(n\log p)$. Addition therefore costs $O(n\log p)$ (n additions in $\mathbb{Z}_p$), and multiplication of two polynomials of degree n is $O(n^2\log^2 p)$.*

4. Same as question 1, but fast algorithms are allowed this time.

	$\mathbb{Z}$	$K[X]$
Unit operation in complexity		
Input size		
Addition/subtraction		
Multiplication		
Euclidean division		
Extended Euclidean Algorithm		
Multipoint evaluation		
Interpolation/CRT		

A:

	$\mathbb{Z}$	$K[X]$
Unit operation in complexity	size of integers	degree of polynomials
Input size	$\log p$ (number of bits)	$\max\{\deg f, \deg g\}$
Addition/subtraction	$\mathcal{O}(\log p)$	$\mathcal{O}(\max\{\deg f, \deg g\})$
Multiplication	$\mathcal{O}(\log p \log(\log p)\log(\log\log p))$	$\mathcal{O}(\deg f \log(\deg f)\log\log(\deg f))$
Euclidean division	$\mathcal{O}(M(\log p))$	$4M(m) + M(n) + \mathcal{O}(n)$
Extended Euclidean Algorithm	$\mathcal{O}(M(\log p)\log^2(\log p))$	$\mathcal{O}(M(n)\log^2 n)$
Multipoint evaluation		$\mathcal{O}(M(n)\log n)$
Interpolation/CRT	$\mathcal{O}(M(\log p)\log(\log p))$	$\mathcal{O}(M(n)\log n)$

Remarks: in Euclidean division for polynomials, we assume that we divide a polynomial of degree m by a polynomial of degree n. For fast EEA, we assume that n is the degree of the dividend.

5. Fill the following table for linear algebra.

	$\mathcal{M}_n(K)$
Unit operation in complexity	
Input size	
Addition/subtraction	
Matrix $\times$ vector	
Matrix multiplication	
Gaussian elimination	
Determinant	
Linear systems	
Inversion	
Characteristic polynomial	

A:

	$\mathcal{M}_n(K)$
Unit operation in complexity	*matrix dimension*
Input size	$\mathcal{O}(n^2)$
Addition/subtraction	$\mathcal{O}(n^2)$
Matrix $\times$ vector	$\mathcal{O}(n^2)$
Matrix multiplication	$\mathcal{O}(n^\omega)$ *(Strassen)*
Gaussian elimination	$\mathcal{O}(n^\omega)$
Determinant	$\mathcal{O}(n^\omega)$
Linear systems	$\mathcal{O}(n^\omega)$
Inversion	$\mathcal{O}(n^\omega)$
Characteristic polynomial	$\mathcal{O}(n^\omega)$

Remark: in class, we've seen how to compute the characteristic polynomial in time $\mathcal{O}(n^3)$ and, for special cases, $\mathcal{O}(n^\omega \log n)$