4. Batch evaluation and interpolation

CS-E4500 Advanced Course on Algorithms
Spring 2019

Petteri Kaski
Department of Computer Science
Aalto University

Lecture schedule

Tue 15 Jan:	1. Polynomials and integers
Tue 22 Jan:	2. The fast Fourier transform and fast multiplication
Tue 29 Jan:	3. Quotient and remainder
Tue 5 Feb:	4. Batch evaluation and interpolation
Tue 12 Feb:	5. Extended Euclidean algorithm and interpolation from erroneous data
Tue 19 Feb:	*Exam week — no lecture*
Tue 27 Feb:	6. Identity testing and probabilistically checkable proofs
Tue 5 Mar:	*Break — no lecture*
Tue 12 Mar:	7. Finite fields
Tue 19 Mar:	8. Factoring polynomials over finite fields
Tue 26 Mar:	9. Factoring integers

CS-E4500 Advanced Course in Algorithms (5 ECTS, III–IV, Spring 2019)

2019	KALENTERI	2019

	Tammikuu	Helmikuu	Maaliskuu	Huhtikuu	Toukokuu	Kesäkuu
1	Ti Uudenvuodenpäivä	Pe	Pe	Ma Vk 14 T9	Ke Vappu	La
2	Ke	La	La	Ti	To	Su
3	To	Su D3	Su	Ke	Pe	Ma Vk 23
4	Pe	Ma Vk 06 T3	Ma Vk 10 Break	To	La	Ti
5	La	Ti L4	Ti Laskiainen	Pe	Su	Ke
6	Su Loppiainen	Ke	Ke	La	Ma Vk 19	To
7	Ma Vk 02	To Q4	To	Su	Ti	Pe
8	Ti	Pe	Pe	Ma Vk 15	Ke	La
9	Ke	La	La	Ti	To	Su Helluntaipäivä
10	To	Su D4	Su D6	Ke	Pe	Ma Vk 24
11	Pe	Ma Vk 07 T4	Ma Vk 11 T6	To	La	Ti
12	La	Ti L5	Ti L7	Pe	Su Äitienpäivä	Ke
13	Su	Ke	Ke	La	Ma Vk 20	To
14	Ma Vk 03	To Q5	To Q7	Su Palmusunnuntai	Ti	Pe
15	Ti L1	Pe	Pe	Ma Vk 16	Ke	La
16	Ke	La	La	Ti	To	Su
17	To Q1	Su	Su D7	Ke	Pe	Ma Vk 25
18	Pe	Ma Vk 08 Exam week	Ma Vk 12 T7	To	La	Ti
19	La	Ti	Ti L8	Pe Pitkäperjantai	Su Kaatuneiden muistopäivä	Ke
20	Su D1	Ke	Ke Kevätpäiväntasaus	La	Ma Vk 21	To
21	Ma Vk 04 T1	To	To Q8	Su Pääsiäispäivä	Ti	Pe Kesäpäivänseisaus
22	Ti L2	Pe	Pe	Ma 2. pääsiäispäivä	Ke	La Juhannus
23	Ke	La	La	Ti	To	Su
24	To Q2	Su D5	Su D8	Ke	Pe	Ma Vk 26
25	Pe	Ma Vk 09 T5	Ma Vk 13 T8	To	La	Ti
26	La	Ti L6	Ti L9	Pe	Su	Ke
27	Su D2	Ke	Ke	La	Ma Vk 22	To
28	Ma Vk 05 T2	To Q6	To Q9	Su	Ti	Pe
29	Ti L3		Pe	Ma Vk 18	Ke	La
30	Ke		La	Ti	To Helatorstai	Su
31	To Q3		Su Kesäaika alkaa D9		Pe	

L = Lecture; hall T5, Tue 12–14

Q = Q & A session; hall T5, Thu 12–14

D = Problem set deadline; Sun 20:00

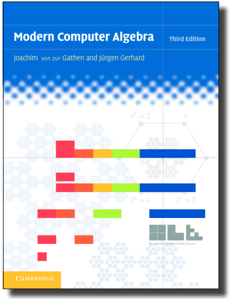
T = Tutorial (model solutions); hall T6, Mon 16–18

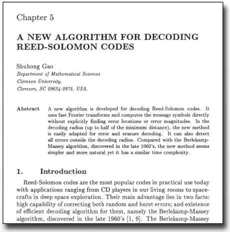
Recap of last week

- **Division** (**quotient** and **remainder**) for integers and polynomials
- Fast division by **reduction** to fast multiplication
- Integer division via **approximation** of the multiplicative inverse of the divisor
- The radix-point representation and approximation of rational numbers
- **Newton iteration**
- Newton iteration for the multiplicative inverse of the divisor
- **Convergence analysis** for Newton iteration
- Polynomial division via **reversal**
- Newton iteration for the inverse of the reverse of the divisor

Goal: Near-linear-time toolbox for univariate polynomials

- Multiplication
- Division (quotient and remainder)
- Batch evaluation (this week)
- Interpolation (this week)
- Extended Euclidean algorithm (gcd)
- Interpolation from partly erroneous data




Chapter 5

A NEW ALGORITHM FOR DECODING REED-SOLOMON CODES

Shuhong Gao
Department of Mathematical Sciences
Clemson University,
Clemson, SC 29634-0975, USA.

Abstract A new algorithm is developed for decoding Reed-Solomon codes. It uses fast Fourier transforms and computes the message symbols directly without explicitly finding error locations or error magnitudes. In the decoding radius (up to half of the minimum distance), the new method is easily adapted for error and erasure decoding. It can also detect all errors outside the decoding radius. Compared with the Berlekamp-Massey algorithm, discovered in the late 1960's, the new method seems simpler and more natural yet it has a similar time complexity.

1. Introduction

Reed-Solomon codes are the most popular codes in practical use today with applications ranging from CD players in our living rooms to spacecrafts in deep space exploration. Their main advantage lies in two facts: high capability of correcting both random and burst errors; and existence of efficient decoding algorithm for them, namely the Berlekamp-Massey algorithm, discovered in the late 1960's [1, 9]. The Berlekamp-Massey

Key content for Lecture 4

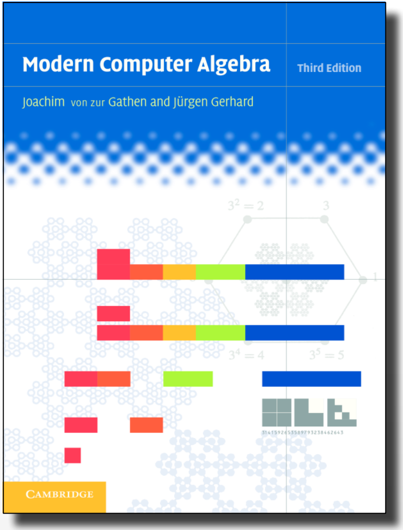
- Fast **batch evaluation** and **interpolation** of polynomials
- Reduction to fast quotient and remainder
  —divide-and-conquer recursive remaindering along a **subproduct tree**
- **Secret sharing** by randomization

Further motivation for this week

- The evaluation–interpolation duality for polynomials is the source of many algorithm designs and applications
- An application we encounter today:
  How to share a secret (Shamir [25])
- With further knowledge of algebra and algebraic structures (e.g. cf. Lang [18] and Cox, Little, and O'Shea [6]), considerable generalizations are possible

Batch evaluation and interpolation

(von zur Gathen and Gerhard [11], Sections 10.1–10.3 and 5.1–5.4)



Batch evaluation and interpolation

- To **evaluate** a polynomial $(\varphi_0, \varphi_1, \ldots, \varphi_d) \in F^{d+1}$ at ("a batch of") distinct points $\xi_0, \xi_1, \ldots, \xi_d \in F$, we multiply from the left with the Vandermonde matrix:

$$\begin{bmatrix} \xi_0^0 & \xi_0^1 & \cdots & \xi_0^d \\ \xi_1^0 & \xi_1^1 & \cdots & \xi_1^d \\ \vdots & \vdots & & \vdots \\ \xi_d^0 & \xi_d^1 & \cdots & \xi_d^d \end{bmatrix} \begin{bmatrix} \varphi_0 \\ \varphi_1 \\ \vdots \\ \varphi_d \end{bmatrix} = \begin{bmatrix} f(\xi_0) \\ f(\xi_1) \\ \vdots \\ f(\xi_d) \end{bmatrix}$$

- To **interpolate** the coefficients of a polynomial with values $(f(\xi_0), f(\xi_1), \ldots, f(\xi_d)) \in F^{d+1}$ at distinct $\xi_0, \xi_1, \ldots, \xi_d \in F$, we multiply from the left with the inverse of the Vandermonde matrix:

$$\begin{bmatrix} \xi_0^0 & \xi_0^1 & \cdots & \xi_0^d \\ \xi_1^0 & \xi_1^1 & \cdots & \xi_1^d \\ \vdots & \vdots & & \vdots \\ \xi_d^0 & \xi_d^1 & \cdots & \xi_d^d \end{bmatrix}^{-1} \begin{bmatrix} f(\xi_0) \\ f(\xi_1) \\ \vdots \\ f(\xi_d) \end{bmatrix} = \begin{bmatrix} \varphi_0 \\ \varphi_1 \\ \vdots \\ \varphi_d \end{bmatrix}$$

Fast batch evaluation and interpolation?

- Can we go faster than working with the Vandermonde matrix in explicit form?
- Yes, for example, in the case when the points $\xi_0, \xi_1, \ldots, \xi_d$ are powers of a primitive root of unity of composite order $d + 1$ (recall fast Fourier transform from Lecture 2)
- But what about in general?
  That is, when $\xi_0, \xi_1, \ldots, \xi_d$ are arbitrary distinct points in a ring R
- *We now know how to multiply and divide fast, so maybe we could put these algorithms into use ...*

Polynomial division (quotient and remainder) recalled

- Let R be a ring (commutative and nontrivial, as usual)
- Let $a = \sum_i \alpha_i x^i \in R[x]$ and $b = \sum_i \beta_i x^i \in R[x]$ be given as input with $\deg a = n$, $\deg b = m$, and $n \geq m \geq 0$
- Let us also assume that $\beta_m = 1$ (that is, b is **monic**)
- We want to compute $q, r \in R[x]$ with $a = qb + r$ and $\deg r < m$
- That is, $q = a \operatorname{quo} b$ is the **quotient** and $r = a \operatorname{rem} b$ is the **remainder** in the polynomial division with **dividend** a and **divisor** b
- We now have a fast algorithm that divides in $O(M(n))$ operations in R by reduction to fast multiplication
- Let us now develop fast algorithms for batch evaluation and interpolation to by reduction to fast division

Fast batch evaluation by recursive remaindering

- Suppose we have a polynomial $f = \varphi_0 + \varphi_1 x + \varphi_2 x^2 + \ldots + \varphi_d x^d \in R[x]$ and we want to compute the values $f(\xi_0), f(\xi_1), \ldots, f(\xi_{e-1})$ at e given points $\xi_0, \xi_1, \ldots, \xi_{e-1} \in R$
- **Goal:** $O(M(d) + M(e) \log e)$ operations in R
- We reduce the multi-point (batch) evaluation task to recursive remaindering along a subproduct tree enabled by the following to lemmas (proofs: in the problem set)

Lemma 6 (Evaluation at a point via remainder)

For all $\xi \in R$ *and* $f \in R[x]$ *it holds that* $f(\xi) = f \operatorname{rem} (x - \xi)$

Lemma 7 (Recursive remaindering)

Let $a, b, c \in R[x]$, *with* b *and* c *monic, and suppose that* c *divides* b. *Then,*
$a \operatorname{rem} c = (a \operatorname{rem} b) \operatorname{rem} c$

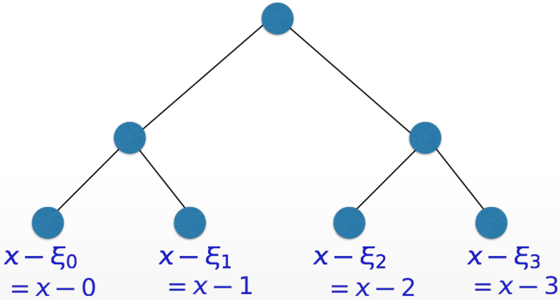
Example: Batch evaluation

- The algorithm for fast batch evaluation is perhaps best illustrated by starting with an example and then proceeding with the details

- Let us work over $R = \mathbb{Z}$ for simplicity
- Let $f = x^5 - x^4 + 2x^3 + 4x - 5 \in \mathbb{Z}[x]$
- Let $\xi_0 = 0$, $\xi_1 = 1$, $\xi_2 = 2$, $\xi_3 = 3$

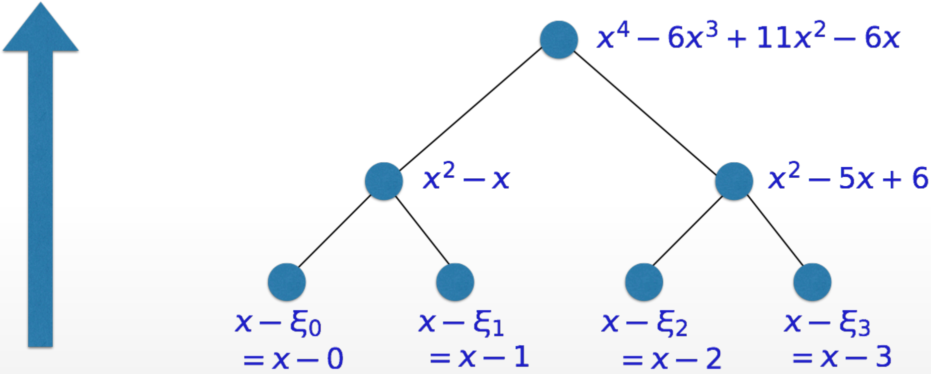
Example: Batch evaluation (0/4)

0. Place the linear polynomials $x - \xi_j$ for $j = 0, 1, \ldots, e - 1$ at the leaves of a perfect binary tree (can assume that e is a power of 2)



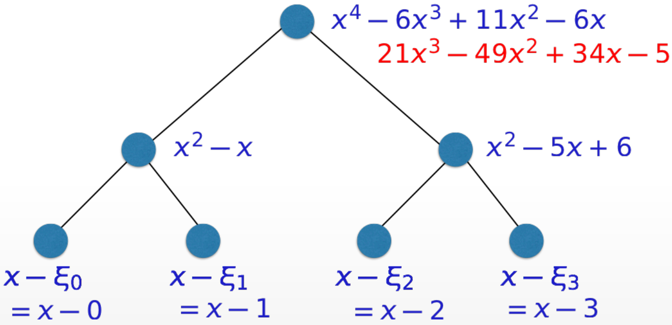
Example: Batch evaluation (1/4)

1. For each internal node in post-order, place the product of the two child nodes at the node



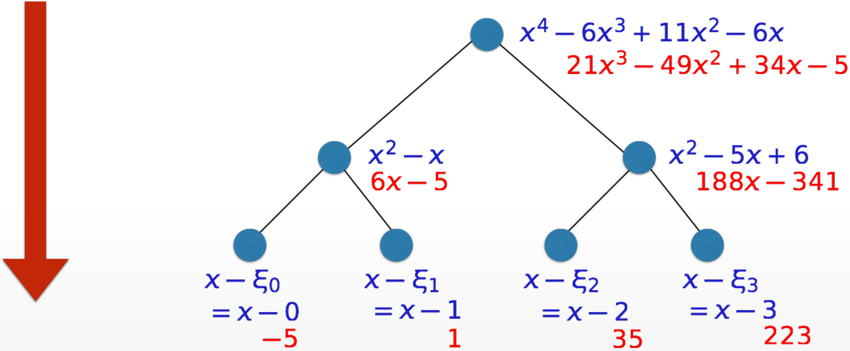
Example: Batch evaluation (2/4)

2. Compute the remainder of $f = x^5 - x^4 + 2x^3 + 4x - 5$ and the root node and place it at the root node



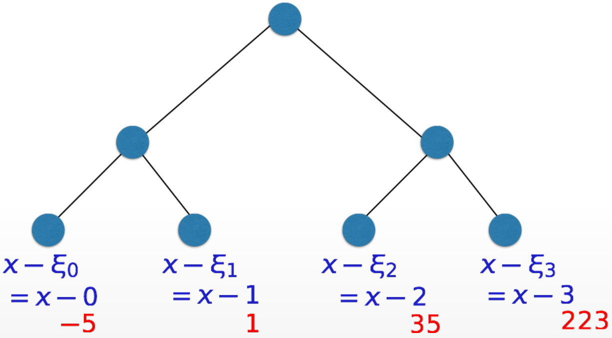
Example: Batch evaluation (3/4)

3. For each nonroot node in preorder, compute the remainder of the parent node and the subproduct at the node



Example: Batch evaluation (4/4)

4. The remainders at the leaf nodes are the evaluations $f(\xi_j)$



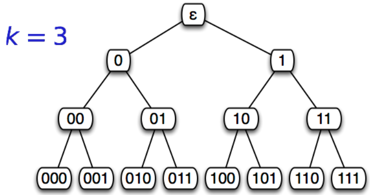
Nodes of a perfect binary tree and binary strings (1/2)

- Let us now present the algorithm in detail
- Without loss of generality we can assume that $e = 2^k$ for some $k \in \mathbb{Z}_{\geq 0}$ (for example, insert new points of evaluation until e is a power of 2)
- We will structure the recursion along a perfect binary tree with 2^k leaves
- Let us write $\{0,1\}^{\underline{k}}$ for the set of all binary strings of length at most k, including the empty string ϵ
- For $u \in \{0,1\}^{\underline{k}}$ let us write $0 \leq |u| \leq k$ for the length of u
- *Example.* For $k = 3$, we have

$$\{0,1\}^{\underline{k}} = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111\}$$

Nodes of a perfect binary tree and binary strings (2/2)

- The $2^{k+1} - 1 = \sum_{j=0}^{k} 2^j$ strings in $\{0, 1\}^{\underline{k}}$ are in a natural one-to-one correspondence with the nodes of a perfect binary tree with 2^k leaves, with the empty string ϵ corresponding to the root and the strings of length k corresponding to the leaves
- Indeed, to navigate from a non-root node to its parent node, simply delete the last bit from the corresponding string
- Dually, to navigate from a non-leaf node to one of its two children, append either the bit 0 (to go the left child) or the bit 1 (to go the right child) to the string



A subproduct tree for batch evaluation

- Let us work with a perfect binary tree with 2^k leaves and nodes indexed by the binary strings in $\{0,1\}^{\underline{k}}$
- Associate with each leaf $v \in \{0,1\}^k$ the linear polynomial

$$s_v = x - \xi_v \tag{26}$$

- Associate with each internal node $u \in \{0,1\}^{\underline{k-1}}$ the product of the children of u by

$$s_u = s_{u0} s_{u1} \tag{27}$$

- We observe that s_u is a monic polynomial of degree $2^{k-|u|}$ for all $u \in \{0,1\}^{\underline{k}}$

Fast batch evaluation using a subproduct tree

- To perform batch evaluation, first compute and store the polynomials s_u for all $u \in \{0,1\}^{\underline{k}}$ using (26) and (27)
- Then, associate the remainder

$$r_\epsilon = f \operatorname{rem} s_\epsilon \tag{28}$$

  with the root ϵ of the binary tree
- For each nonroot $u \in \{0,1\}^{\underline{k}} \setminus \{\epsilon\}$, associate with u the remainder

$$r_u = r_p \operatorname{rem} s_u \tag{29}$$

  where $p \in \{0,1\}^{\underline{k-1}}$ is the parent of u in the binary tree
- For each leaf $v \in \{0,1\}^k$, the remainder r_v satisfies $r_v = f(\xi_v)$

Analysis

- Recall that s_u is a monic polynomial of degree $2^{k-|u|}$ for all $u \in \{0,1\}^{\underline{k}}$
- From (26) and (27) we have that each s_u can be prepared in $O(M(2^{k-|u|}))$ operations in R using fast multiplication
- There are in total 2^j binary strings $u \in \{0,1\}^j$, implying that the total cost of level $j = k, k-1, \ldots, 0$ is $O(2^j M(2^{k-j}))$ operations in R, which is $O(M(2^k)) = O(M(e))$ by at-least-linear and at-most-polynomial growth of M
- The root remainder (28) takes $O(M(d) + M(e))$ operations in R using fast division
- Below the root, each level $j = 0, 1, \ldots, k$ similarly takes $O(M(e))$ operations in R using (29) and fast division
- Since there are $k = O(\log e)$ levels, we obtain that that batch evaluation runs in total $O(M(d) + M(e) \log e)$ operations in R

Interpolation

- Let R be a ring
- Let $\xi_0, \xi_1, \ldots, \xi_{e-1} \in R$ and $\eta_0, \eta_1, \ldots, \eta_{e-1} \in R$ such that $\xi_i - \xi_j$ is a unit in R for all $0 \le i < j \le e-1$
- We seek to compute the coefficients of the Lagrange interpolation polynomial

$$\ell = \sum_{i=0}^{e-1} \left(\eta_i \prod_{\substack{j=0 \\ j \neq i}}^{e-1} (\xi_i - \xi_j)^{-1} \right) \prod_{\substack{j=0 \\ j \neq i}}^{e-1} (x - \xi_j) \in R[x]$$

  that satisfies $\ell(\xi_i) = \eta_i$ for all $i = 0, 1, \ldots, e-1$

Fast interpolation with subproduct trees

- The form
$$\ell = \sum_{i=0}^{e-1}\left(\eta_i \prod_{\substack{j=0\\ j\neq i}}^{e-1}(\xi_i-\xi_j)^{-1}\right)\prod_{\substack{j=0\\ j\neq i}}^{e-1}(x-\xi_j) \in R[x]$$
suggests that one should first seek to construct the coefficients of the polynomial
$$\ell = \sum_{i=0}^{e-1}\lambda_i \prod_{\substack{j=0\\ j\neq i}}^{e-1}(x-\xi_j) \in R[x]$$
from e given scalars $\lambda_0, \lambda_1, \ldots, \lambda_{e-1} \in R$

- A strategy based on subproduct-trees works also here and leads to an algorithm that runs in $O(M(e)\log e)$ operations in R (exercise)

Application: How to share a secret

"In this paper we show how to divide data D into n pieces in such a way that D is easily reconstructible from any k pieces, but even complete knowledge of $k-1$ pieces reveals absolutely no information about D. This technique enables the construction of robust key management schemes for cryptographic systems that can function securely and reliably even when misfortunes destroy half the pieces and security breaches expose all but one of the remaining pieces."

(Shamir [25])

Application: How to share a secret (1/5)

- Let us work over a finite field F (for example, $F = \mathbb{Z}_p$ for p prime)
- Let $f = \varphi_0 + \varphi_1 x \in F[x]$ be a line (polynomial of degree at most 1)
- How much do we know about the constant φ_0 of the line f if we know the value $f(\xi)$ for a **nonzero** $\xi \in F$?

Application: How to share a secret (2/5)

- Let us work over a finite field F (for example, $F = \mathbb{Z}_p$ for p prime)
- Let $f = \varphi_0 + \varphi_1 x + \varphi_2 x^2 + \ldots + \varphi_d x^d \in F[x]$ be a polynomial of degree at most d
- How much do we know about the constant φ_0 of the polynomial f if we know $(\xi_j, f(\xi_j))$ for exactly d **nonzero** distinct values $\xi_j \in F$ for $j = 1, 2, \ldots, d$?

Application: How to share a secret (3/5)

- Let $f = \varphi_0 + \varphi_1 x + \varphi_2 x^2 + \ldots + \varphi_d x^d \in F[x]$ be a polynomial of degree at most d
- How much do we know about the constant φ_0 of the polynomial f if we know $(\xi_j, f(\xi_j))$ for exactly d **nonzero** distinct values $\xi_j \in F$ for $j = 0, 1, \ldots, d$?
- *We claim that this knowledge reveals no information about* φ_0; indeed, let us set $\xi_0 = 0$ and recall the interpolation identity

$$\begin{bmatrix} \xi_0^0 & \xi_0^1 & \cdots & \xi_0^d \\ \xi_1^0 & \xi_1^1 & \cdots & \xi_1^d \\ \vdots & \vdots & & \vdots \\ \xi_d^0 & \xi_d^1 & \cdots & \xi_d^d \end{bmatrix}^{-1} \begin{bmatrix} f(\xi_0) \\ f(\xi_1) \\ \vdots \\ f(\xi_d) \end{bmatrix} = \begin{bmatrix} \varphi_0 \\ \varphi_1 \\ \vdots \\ \varphi_d \end{bmatrix}$$

- Since $f(\xi_0) = f(0) = \varphi_0$, we have that for each choice $\varphi_0 \in F$ the values $f(\xi_1), f(\xi_2), \ldots, f(\xi_d)$ are consistent with exactly one choice $(\varphi_0, \varphi_1, \ldots, \varphi_d) \in F^{d+1}$
- Thus, the values $f(\xi_1), f(\xi_2), \ldots, f(\xi_d)$ reveal no information about φ_0

Application: How to share a secret (4/5)

- Let $f = \varphi_0 + \varphi_1 x + \varphi_2 x^2 + \ldots + \varphi_d x^d \in F[x]$ be a polynomial of degree at most d
- How much do we know about the constant φ_0 of the polynomial f if we know $(\xi_j, f(\xi_j))$ for exactly e **nonzero** distinct values $\xi_j \in F$ for $j = 1, 2, \ldots, e$?
- For $e \leq d$, we obtain no information about φ_0
- For $e \geq d + 1$, we have full information about φ_0 since we can interpolate all the coefficients of f from any $d + 1$ evaluations at distinct points

Application: How to share a secret (5/5)

- Suppose $\varphi_0 \in F$ is a **secret** that you want to split into s **shares** so that
  - knowledge of any k shares enables recovery of the secret
  - knowledge of any $k-1$ or fewer shares reveals no information about the secret

1. Let $\xi_1, \xi_2, \ldots, \xi_s \in F$ be distinct and **nonzero**
2. Select elements $\varphi_1, \varphi_2, \ldots, \varphi_{k-1} \in F$ independently and uniformly at random
3. Let $f = \varphi_0 + \varphi_1 x + \varphi_2 x^2 + \ldots + \varphi_{k-1} x^{k-1} \in F[x]$
4. For $j = 1, 2, \ldots, s$, share j is the pair $(\xi_j, f(\xi_j)) \in F^2$

- Using fast batch evaluation and interpolation, preparing the shares takes $O(M(s) \log s)$ operations in F, and recovering the secret takes $O(M(k) \log k)$ operations in F

Randomization and primal–dual

- The secret $\varphi_0 \in F$ resides in the primal (coefficient representation)
- Selecting $\varphi_1, \varphi_2, \ldots, \varphi_{k-1} \in F$ independently and uniformly at random masks the secret in the dual (evaluation representation) unless we know k shares
- This is our first example of the use of **randomization** during this course
- The evaluation–interpolation duality enables us to spread the information in the coefficient representation uniformly to evaluations in the evaluation representation
- The following lectures will explore both randomization as a tool in algorithm design and the aforementioned "uniformity" further, the latter in particular as regards error-correcting codes and error-tolerant computation

Recap of Lecture 4

- Fast **batch evaluation** and **interpolation** of polynomials
- Reduction to fast quotient and remainder
  —divide-and-conquer recursive remaindering along a **subproduct tree**
- **Secret sharing** by randomization

Learning objectives (1/2)

- Terminology and objectives of modern algorithmics, including elements of algebraic online, and randomised algorithms
- Ways of coping with uncertainty in computation, including error-correction and proofs of correctness
- The art of solving a large problem by reduction to one or more smaller instances of the same or a related problem
- (Linear) independence, dependence, and their abstractions as enablers of efficient algorithms

Learning objectives (2/2)

- Making use of duality
  - Often a problem has a corresponding **dual** problem that is obtainable from the original (the **primal**) problem by means of an easy transformation
  - The primal and dual control each other, enabling an algorithm designer to use the interplay between the two representations
- Relaxation and tradeoffs between objectives and resources as design tools
  - Instead of computing the exact optimum solution at considerable cost, often a less costly but principled approximation suffices
  - Instead of the complete dual, often only a randomly chosen partial dual or other relaxation suffices to arrive at a solution with high probability