COSC160: Data Structures

Jeremy Bolton, PhD
Assistant Teaching Professor

GEORGETOWN UNIVERSITY

Outline

I. Polynomial Discussion

Project: Polynomials

- Reminder: With the sparse matrix structure, *we faced many structural design questions and subsequent algorithmic design questions, both of which affected efficiency.* You will face similar design questions in your polynomial project.

- Design a Representation (Data Structure) for Polynomials
  - Goals:
    - Polynomial evaluation
    - Polynomial arithmetic

- Class Project: Design Questions and Goals.
  - Linked Chain vs Array Implementation?
  - Goal: An efficient Solution (time and space)
    - Algorithmic improvements for basic operations
    - How can we increase efficiency: reduce computational complexity?
    - When you make a design decision, document the reason why and justify in the cover letter.
    - Use average and worst cases to make design decisions (not best case)

Speed up Examples: Polynomial Evaluation

- Evaluating a polynomial of form
  $P(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + \cdots + \alpha_0 x^0 = \Sigma_{\mathrm{i=1}}^{\mathrm{n}} \alpha_i x^i$

- A simple direct interpretation of this computation may result in the following. Is the result correct – YES. Is the computation efficient –NO. There are many unnecessary steps – the steps taken can be reorganized for efficiency.

```
val := coeff[0]
for i from 1 to n
    val := val + coeff[i]*exp(x,i);
```

- What is the time complexity here?

Polynomial Evaluation (cont)

- Note computing exp(x,i) during each iteration is excessively repetitive: intermediate results for each iterations result are computed in the previous iteration. Thus we can simply build this term *dynamically* during each iteration rather than re-computing in full during each iteration.

$$P(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + \cdots + \alpha_0 x^0 = \Sigma_{\mathrm{i}=1}^{\mathrm{n}} \alpha_i x^i$$

```
val := coeff[0]
for i from 1 to n
    val := val + coeff[i]*exp(x,i);
```

```
val := coeff[0]
y := 1
for i from 1 to n
    y := y * x  // build exp(x,i) dynamically
    val := val + coeff[i]*y;
```

- Result: number of additions is n; number of multiplications 2n.

Polynomial Evaluation (cont)

- Another “speed-up” scheme: Horner’s rule
  - Capitalizes on the factoring of the common factor x in the repeated sums.

$$
\begin{aligned}
&P(x) = \alpha_n x^n + \alpha_{n-1}x^{n-1} + \cdots + \alpha_0 x^0 = \\
&\alpha_n x^n + \alpha_{n-1}x^{n-1} + \cdots + \alpha_2 x^2 + \alpha_1 x^1 + \alpha_0 = \\
&[\alpha_n x^{n-1} + \alpha_{n-1}x^{n-1-1} + \cdots + \alpha_2 x^{2-1} + \alpha_1]x + \alpha_0 = \\
&[\alpha_n x^{n-1} + \alpha_{n-1}x^{n-2} + \cdots + \alpha_2 x^1 + \alpha_1]x + \alpha_0 = \\
&[[\alpha_n x^{n-1-1} + \alpha_{n-1}x^{n-2-1} + \cdots + \alpha_2]x + \alpha_1]x + \alpha_0 = \\
&[[\alpha_n x^{n-2} + \alpha_{n-1}x^{n-3} + \cdots + \alpha_2]x + \alpha_1]x + \alpha_0 = \\
&\ldots \\
&[[\ldots[\alpha_n x \quad + \alpha_{n-1}]x \quad + \alpha_{n-2}]x + \cdots + \alpha_2]x + \alpha_1]x + \alpha_0 =
\end{aligned}
$$

```
val := coeff[n]
for i from 1 to n
        val := val*x + coeff[n-i];
```

Another Example: Fast Exponentiation

- Makes use of binary encoding and mathematical properties of exponentiation to efficiently evaluate exponential terms
  - You may also use mathematical properties to your advantage!
  - Naïve approach: multiply base n times, for a total of n-1 multiplications
  - Or use squaring approach also used in modular exponentiation

Example: Evaluate a^{16}
  - One solution: Multiply a times itself 15 times.
  - A faster solution: $a^{16} = a^{2^4} = (a^2)^8 = ((a^2)^2)^4 = (((a^2)^2)^2)^2$
    - Only 4 multiplications !

- In general

$$x^n = \begin{cases} x(x^2)^{\frac{n-1}{2}}, & if\ n\ is\ odd \\ (x^2)^{\frac{n}{2}}, & if\ n\ is\ even \end{cases}$$

GEORGETOWN UNIVERSITY