LECTURE 12

FAST FOURIER TRANSFORM

Po Hao Chen

4/15 2022

BOSTON UNIVERSITY

Fast Fourier Transform

Objectives

- What is FFT? Black Magic
- Transforming our problems into polynomials
- FFT Variants

What is FFT?

- It is the fast algorithm for Fourier Transform, $O(nlogn)$
- Without being overly technical, it is just fast polynomial multiplication
- We can use it to do fast convolution with two arrays.
- Convolution: $C_p = (A * B)_p = \sum_{i*j=p} A_i B_j$

Applications

- If we can formulate our problem into polynomials, we can potentially solve the problem more efficiently with FFT
- Let's illustrate with an example.

Example: Find all possible sums of two arrays

Naive Solution $O(n^2)$:

- Example:
  $A = [1,2,3]$
  $B = [2,4]$
- $1 + 2 = 3$
  $1 + 4 = 5$
  $2 + 2 = 4$
  $2 + 4 = 6$
  $3 + 2 = 5$
  $3 + 4 = 7$

```
set<int> s;
for(int i  = 0 ;i < 3 ; i++){
    for(int j = 0 ; j < 2 ; j++){
        s.insert(A[i] + B[j]);
    }
}
```

Transforming our input

- Our input:
  A = [1,2,3]
  B = [2,4]
- Let us represent A and B as polynomials by taking the number to the exponents:
  $A = x^1 + x^2 + x^3$
  $B = x^2 + x^4$
- Take A as example, the coefficient represents the number of 1s, 2s, 3s in the original array. (i.e $1 * x^1 + 1 * x^2 + 1 * x^3$)

$O(nlogn)$ optimization

- Take our polynomial A and B and multiply them
  $(x^1 + x^2 + x^3) * (x^2 + x^4) =$
  $(x^3 + x^4 + 2x^5 + x^6 + x^7)$
- This says:

  3 can be formed in 1 way

  4 ..... 1 way

  5 ..... 2 ways

  6 ..... 1 way

  7 ..... 1 way
- FFT occurs at the multiplication step
  We can do CONV([0,1,1,1,0], [0,0,1,0,1]) $\rightarrow$ [0,0,0,1,1,2,1,1]

Example: With a single array

- What if we want to find all the possible sums of a single array?
- A = [1,2,3]
- Output:

  1

  2

  3

  $1 + 3 = 4$

  $2 + 3 = 5$
- We can simply perform CONV(A_p,A_p) then take care of the duplicates.

  Denote A_p as A's polynomial form.

Example: All Substrings Hamming Distance

Given two binary strings A and B with the length of N and M respectively. You need to calculate the Hamming distance between B and **every sub-strings** of length M of A

- Example:
  1010 A
  100 B
- Output:
  1, 2

Consider a simpler version of the problem

- If we want to calculate the hamming distance of two binary strings
  In C++:

  ```
  __builtin_popcnt(a^b)
  ```

  Complexity: $O(n)$
- Doing it for all substring will take $O(n^2)$

Using FFT

- Given string A (100101) B (110)
- f: conv(A, B_r)
- Denote B_r as the reversal of B. Pad 0s to match A's length.
- compute f with conv([1,0,0,1,0,1], [0.1,1,0,0,0])
  $= [0,1,1,0,1,1]$
- f[2] contains the number of 1 that matches between 100, 110
- Observation:
  - We are taking a stride of 3 and convolving B against A.
  - f computes the 1's that matches
  - if we compute the #0's that matches in some way we can find the hamming distance

Complement

Find the #0's that match

- A (100101) B (110)
- f^c: conv(A^c, B^c_r)
- $A^c = 011010$
- $B^c = 001 \rightarrow_{reverse+pad} \rightarrow B^c_r = 100000$
- $f^c = [0, 1, 1, 0, 1, 0]$

Hamming Distance between A and B:
$N - max(f[i] + f^c[i])$ for $m - 1 \leq i \leq n - 1$
For each substring of A w/ length M:
$M - (f[i] + f^c[i])$ for $0 \leq i \leq n - 1$
Complexity: O(nlogn)

FFT Variants

Variants

- FFT can be extended and be more powerful
- Number theoretic transform allows you to compute coefficients modulo some prime number p (only works with integer)
- Fast Walsh Hadamard Transforms allows you to do bitwise xor, or, and convolution