Online FFT

Tanuj Khattar

January 18, 2018

Problem Statement

- Let $G[1...N]$ be sequence of length N that is known to us. We wish to compute all N terms of sequence F defined as follows:

$$F[n] = \sum_{i=1}^{n-1} F[i] * G[n-i]$$

- A naive implementation would take $O(N^2)$ time to compute N terms of the sequence F.
- We present a way to do this in $\mathcal{O}(N \log^2 N)$ using FFT. Since it is a convolution where n'th term of the resulting sequence depends on all the previous terms, we call the technique Online FFT.

General Idea

- Let $P(x)$ and $Q(x)$ be two polynomials of degree $N-1$ each. FFT helps us compute the polynomial $F(x) = P(x) \times Q(x)$ in $\mathcal{O}(N \log N)$.
- Now, we will see another slower method to compute $F(x)$, but which will turn out to be useful.
- Assume N is a power of 2. If not, add extra terms with coeff $= 0$.
- Let $P(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$,
  $Q(x) = b_0 + b_1 x + \cdots + b_{n-1}x^{n-1}$.
- Divide $P(x)$ into blocks of size 2^i such that we can write $P(x)$ as:

$$P(x) = a_0 + x * B_0 + x^2 * B_1 + \cdots + x^{2^i} * B_i + \cdots + x^{n/2} * B_{\log(N)-1}$$

where

$$B_i = a_{2^i} + a_{2^i+1} * x + a_{2^i+2} * x^2 + \cdots + a_{2^{i+1}-1} * x^{2^i-1}$$

such that $|B_i| = 2^i$.

General Idea

- $F(x) = P(x) \times Q(x)$ can now be written as

$$F(x) = \left(a_0 + x * B_0 + x^2 * B_1 + \cdots + x^{2^i} * B_{\log(N)-1}\right) \times Q(x)$$

$$\implies F(x) = a_0 \times Q(x) \quad + \sum_{i=0}^{\log(N)-1} x^{2^i} \times B_i \times Q(x)$$

- The first term $a_0 \times Q(x)$ can be computed in linear time. Hence we consider only the second summation now.
- Let $G_i(x) = B_i \times Q(x)$. To compute $G_i(x)$ we first partition N terms of $Q(x)$ into consecutive blocks of size 2^i since $|B_i| = 2^i$ (note that N is a power of 2).

General Idea

- Partitioning $Q(x)$ as mentioned earlier, we get

$$Q(x) = \sum_{j=0}^{N/2^i-1} x^{2^i \times j} * C_{i,j}$$

where

$$C_{i,j} = b_{2^i \times j} + b_{2^i \times j+1} * x + \cdots + b_{2^i \times (j+1)-1} * x^{2^i-1}$$

- Substituting in $G_i(x)$ we get,

$$G_i(x) = B_i \times Q(x)$$

$$\implies G_i(x) = B_i \times \sum_{j=0}^{N/2^i-1} x^{j \cdot 2^i} \cdot C_{i,j}$$

General Idea

$$\implies G_i(x) = \sum_{j=0}^{N/2^i-1} x^{j\cdot 2^i} \cdot (B_i \times C_{i,j})$$

- Note that $|B_i| = |C_{i,j}| = 2^i$. We can compute the product $B_i \times C_{i,j}$ using FFT in $\mathcal{O}(|B_i| \log |B_i|)$
- Time required to compute $G_i(x)$ is given as

$$\mathcal{O}((N/|B_i|) \times |B_i| \log |B_i|)$$

$$\implies \mathcal{O}(N \log |B_i|)$$

General Idea

- Time required to compute $F(x)$ is given as

$$\mathcal{O}(N \times \sum_{i=0}^{\log(N)-1} \log|B_i|)$$

$$\implies \mathcal{O}(N \times \sum_{i=0}^{\log(N)-1} i)$$

$$\implies \mathcal{O}(N \log^2 N)$$

- Instead of directly computing $F(x) = P(x) \times Q(x)$ in $\mathcal{O}(N \log N)$, the above method gives us an alternate approach to compute $F(x)$ in $\mathcal{O}(N \log^2 N)$. We will soon see why this approach is helpful.

Important Observations

$$F(x) = a_0 \times Q(x) \quad + \sum_{i=0}^{\log(N)-1} \sum_{j=0}^{N/2^i-1} x^{(j+1)\cdot 2^i} \cdot (B_i \times C_{i,j})$$

- Note that to compute the coefficient of x^k in $F(x)$, it is sufficient to compute $x^{(j+1)\cdot 2^i} \cdot (B_i \times C_{i,j})$ $\forall (i,j)$ s.t.

$$2^i \times (j+1) \leq k$$

- Also, we know that the last term of $C_{i,j} = b_{2^i \times (j+1)-1} * x^{2^i-1}$
- Therefore, to compute the coefficient of x^k in F(x), it is sufficient to compute $x^{(j+1)\cdot 2^i} \cdot (B_i \times C_{i,j})$ $\forall (i,j)$ such that all terms of $C_{i,j}$ are among the first k terms of $Q(x)$.

Solution Idea

- With the above observations, we now look at the original problem of computing the first N terms of sequence F where

$$F[n] = \sum_{i=1}^{n-1} F[i] * G[n-i]$$

  and sequence G is known to us in advance.
- In the previous derivation, let $P(x) = G(x)$ and let $Q(x) = F(x)$. Only change is that now, we do not know $Q(x)$ in advance. However, as shown earlier, to compute the coefficient of x^k in the resulting product(i.e. $F[k]$) we only need to know the first $k-1$ terms of $Q(x)$.
- After computing $F[k]$, we look at all the $C_{i,j}$ that end at index k and add their contribution $(x^{(j+1)\cdot 2^i} \cdot (B_i \times C_{i,j}))$ to all the remaining terms ahead of k. Since whenever we reach $F[k]$, contribution of all previous terms would have already been added to it, $F[k]$ would store the required answer.

Solution Pseudocode

```
//For solving recurrences of the form F_i=sum(1 <=j<i)F_j*G_n-j
void convolve(int l1, int r1, int l2, int r2){
  A = F[l1 .. r1]; B = G[l2 ... r2];//0-based polynomials
  C = A * B;//multiplication of two polynomials.
  for(int i = 0; i < C.size(); ++i)
    F[l1 + l2 + i] += C[i];
}//in main function.
F[1] = 1;//some base case.
for(int i = 1;i <= n - 1; i++){
  //We have computed till F_i and want to add its contribution.
  F[i + 1] += F[i] * G[1]; F[i + 2] += F[i] * G[2];
  for(int pw = 2; i % pw == 0 && pw + 1 <= n; pw = pw * 2){
    //iterate over every power of 2 untill 2 ^ i divides i.
    convolve(i - pw, i - 1, pw + 1, min(2 * pw, n));
  }
}
```

Summary

- Since number and degrees of polynomial multiplications using FFT remains exactly the same as the previous analysis, the complexity of the algorithm is $\mathcal{O}(N \log^2 N)$
- The trick of dividing the given polynomial into blocks of size 2^i helped us compute online convolution with only $\mathcal{O}(\log N)$ extra overhead.
- Such a trick is useful to convert many static data structures into dynamic data structures with little overhead. For further reading, refer : `http://repository.cmu.edu/cgi/viewcontent.cgi?article=3453&context=compsci`