# CS 260: PROJECT PROPROSAL
# RELAXED MULTIPLICATION

**Hani Al Majed, Junchen Deng, Reem Alzahrani, Rihan Huang, Salman Shaikh, Tianqi Xu**
Division of Computer, Electrical and Mathematical Science and Engineering

## 1 INTRODUCTION

To introduce our problem, which we'll refer to as "relaxed multiplication," we begin with a counting problem. Given the number of vertices denoted as $n$, how many labeled undirected graphs $G$ that are connected are there? Figure 1 illustrates the four eligible graphs when there are three vertices. Various methods can be employed to calculate it, but in this context, we will solve it by dynamic programming.

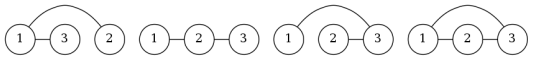


Figure 1: There are four graphs that meet the specified conditions with three vertices. These graphs are considered distinct solutions even if they share the same topology but have different vertex labels.

We'll denote the required number of connected graphs with $n$ vertices as $f_n$ and the total number of graphs regardless of whether they are connected or not as $g_n$. To determine the count of connected graphs, we'll subtract the number of disconnected graphs from $g_n$. Let's begin by discussing how to calculate $g_n$, and then we'll proceed to count the number of disconnected graphs.

We can consider the set of all possible edges of the graph, which amounts to $\binom{n}{2}$ edges. Since any labeled graph is uniquely determined by its edges, the number of labeled graphs with $n$ vertices is equal to $g_n = 2^{\binom{n}{2}}$.

Next, let's consider the scenario of illegitimacy, wherein the graph comprises two or more connected components. To prevent duplication, we enumerate the size of the connected component containing vertex 1 as $m$, while the remaining vertices are interconnected arbitrarily but assuredly not connected to the component containing vertex 1. To label the vertices within this component, we must choose $m-1$ labels from the set of $n-1$ labels, subtracting one because label 1 is obligated to appear within it.

By combining the aforementioned procedures, we arrive at the transfer equation for dynamic programming:

$$f_i = g_i - \sum_{j=1}^{i-1} f_j g_{i-j} \binom{i-1}{j-1} \tag{1}$$

The starting point for dynamic programming is set with an initial value of $f_1 = g_1 = 1$. With this setup, we are capable of computing $f_n$ with a time complexity of $O(n^2)$.

In order to accelerate the computation, we choose to expand the combinatorial number and apply certain identity transformations:

$$f_i = g_i - \sum_{j=1}^{i-1} f_j g_{i-j} \binom{i-1}{j-1} \tag{2}$$

$$f_i = g_i - \sum_{j=1}^{i-1} f_j g_{i-j} \frac{(i-1)!}{(j-1)!(i-j)!} \tag{3}$$

$$\frac{f_i}{(i-1)!} = \frac{g_i}{(i-1)!} - \sum_{j=1}^{i-1} \frac{f_j}{(j-1)!} \frac{g_{i-j}}{(i-j)!} \tag{4}$$

$$f'_i = i g'_i - \sum_{j=1}^{i-1} f'_j g'_{i-j} \tag{5}$$

where $f'_i = \frac{f_i}{(i-1)!}, g'_i = \frac{g_i}{i!}$.

Given that $\{g_i\}$ is a known quantity and can be preprocessed, if we are able to efficiently calculate the second term, we can likewise perform fast computations for both $\{f'_i\}$ and $\{f_i\}$. In the next chapter, we will give a formal formulation for this kind of problem.

## 2 Problem Formulation

Our problem is that given $f_0$ and $\{g_i\}_{i=1}^{n-1}$, compute $\{f_i\}_{i=1}^{n-1}$ by

$$f_i = \sum_{j=1}^{i} f_{i-j} g_j \tag{6}$$

Indeed, this problem bears a strong resemblance to convolution in its mathematical form:

$$f_i = \sum_{j=0}^{i} h_{i-j} g_j \tag{7}$$

However, a key distinction lies in the fact that in typical convolution problems, we can have values for both arrays $g$ and $h$. In our specific problem, one of the "convolution" arrays remains as $f$ itself. Consequently, the computation of $f_i$ necessitates that we calculate all elements before it. This kind of algorithms are so called lazy algorithms, in which the coefficient $h$ and $g$ may actually depend on "previous" coefficient of $f$ ($h_i = f_i$ in our case), as long as they are computed before they are needed in the multiplication algorithm. For this reason, lazy algorithms are extremely useful for the resolution of functional equations. van der Hoeven (2002)

The similarity in form to convolution is precisely why we have two algorithms that can significantly enhance the time complexity from $O(n^2)$ to $O(n \log^2 n)$ and even further to $O(n \log n)$ by leveraging the Fast Fourier Transform.

## 3 Algorithms

### 3.1 Algorithm 1

#### 3.1.1 Algorithm Description

The first algorithm uses divide and conquer with Fast Fourier Transform (FFT). Given a subsequence of $\{f\}$ from index $l$ to index $r$ to be computed, i.e. $\{f_i\}_{i=l}^{r}$, we divide the sequence into two subsequence from the middle, so $mid$ equals $\lfloor \frac{l+r}{2} \rfloor$. First, we compute the first subsequence, i.e. $\{f_i\}_{i=l}^{mid}$. Then, we compute the second subsequence based on the results of the first subsequence, i.e. $\{f_i\}_{i=mid+1}^{r}$.

To compute $\{f_i\}_{i=l}^{mid}$, we apply the divide and conquer algorithm recursively, since their results are not related to the value of $\{f_i\}_{i=mid+1}^{r}$.

To compute $\{f_i\}_{i=mid+1}^{r}$, we compute the contribution from $\{f_i\}_{i=l}^{mid}$ to each $f_i$ in $i \in [mid+1, r]$, where the contribution from $f_i$ to $f_k$ means $f_i g_{k-i}$. It is obvious that $f_i = \sum_{j=1}^{i} f_{i-j} g_i$ equals to the sum of the contributions from $\{f_i\}_{i=0}^{i-1}$. Let $w_i$ denote the contribution from the computed sequence $\{f\}_{i=1}^{mid}$ to $f_i$. We have $f_i = w_i$ if $i \leq mid + 1$.

To compute $\{w_i\}_{i=mid+1}^{r}$ from $\{f_i\}_{i=l}^{mid}$, we construct a new sequence $\{h_i\}_{i=l}^{r-1}$, where

$$h_i = \begin{cases} f_i, \ i \in [l, mid] \\ 0, \ i \in [mid+1, r-1] \end{cases}$$

Therefore, the contribution to $f_k$ from $\{f_i\}_{i=1}^{mid}$ is $\sum_{i=l}^{k-1} h_i g_{k-i}$, which is in the form of convolution of $\{h_i\}_{i=l}^{r-1}$ and $\{g_j\}_{j=1}^{l-r}$ that can be computed by Fast Fourier Transform. Then we add this value to the previous $w_i$ as the new $w_i$ of the current $mid$.

The termination condition is $l$ equals $r$. All contributions from the left side of $f_l$ to $f_l$ is computed, so $f_l = w_l$.

The pseudo-code is shown in 1.

**Algorithm 1** Divide and Conquer + FFT

**Require:** $f_0$ and $\{g_i\}_{i=1}^{n}$

$\{w_i\}_{i=0}^{n} \leftarrow \{0\}_{i=0}^{n}$
$\{f_i\}_{i=1}^{n} \leftarrow \{0\}_{i=1}^{n}$
$g_0 \leftarrow 0$

**function** SOLVEFX($l$, $r$, $\{f_i\}_{i=l}^{r}$, $\{w_i\}_{i=l}^{r}$, $\{g_i\}_{i=l}^{r}$)
  **if** $l = r$ **then**
    $f_l \leftarrow w_l$
    **return**
  $mid \leftarrow \lfloor \frac{l+r}{2} \rfloor$
  SolveFx($l$, $mid$, $\{f_i\}_{i=l}^{mid}$, $\{w_i\}_{i=l}^{mid}$, $\{g_i\}_{i=l}^{mid}$)
  $\{h_i\}_{i=l}^{mid} \leftarrow \{f\}_{i=l}^{mid}$
  $\{h_i\}_{i=mid+1}^{r} \leftarrow \{0\}_{i=mid+1}^{r}$
  $\{contribution_i\} \leftarrow$ FFTConvolution($\{h_i\}_{i=l}^{r-1}$,$\{g_i\}_{i=1}^{l-r}$)
  **for** $i \leftarrow mid + 1$ to $r$ **do**
    $w_i \leftarrow w_i + contribution_i$
  SolveFx($mid + 1$, $r$, $\{f_i\}_{i=mid+1}^{r}$, $\{w_i\}_{i=mid+1}^{r}$, $\{g_i\}_{i=mid+1}^{r}$)

SolveFx(0, $n$, $\{f_i\}_{i=0}^{n}$, $\{w_i\}_{i=0}^{n}$, $\{g_i\}_{i=0}^{n}$)
**return** $\{f_i\}_{i=1}^{n}$

#### 3.1.2 TIME COMPLEXITY ANALYSIS

### 3.2 ALGORITHM 2

For a recursive equation $f_i = \sum_{j=1}^{i} f_{i-j} g_i, i \geq 1$. We first consider two polynomials with infinite orders $F(x) = \sum_{i=0}^{\infty} f_i x^i, G(x) = \sum_{i=0}^{\infty} g_i x^i$. Then $\{f_0, f_1, \ldots, f_i\}$ are the coefficients of $F(x)$ for all $i \geq 0$. Let $g_0 = 0$, then $\{g_0, g_1, \ldots, g_i\}$ are the coefficients of $G(x)$. We can perform

polynomial multiplication between $F(x)$ and $G(x)$:

$$\begin{aligned}
F(x)G(x) &= (\sum_{i=0}^{\infty} f_i x^i)(\sum_{i=0}^{\infty} g_i x^i) \\
&= \sum_{i=0}^{\infty}(x^i \sum_{j=0}^{i} f_{i-j} g_i) \\
&= \sum_{i=0}^{\infty}(x^i (f_i g_0 + \sum_{j=1}^{i} f_{i-j} g_i)) \\
&= \sum_{i=0}^{\infty}(x^i (\sum_{j=1}^{i} f_{i-j} g_i)) \\
&= \sum_{i=1}^{\infty}(x^i (\sum_{j=1}^{i} f_{i-j} g_i)) \\
&= F(x) - f_0 x^0
\end{aligned}$$

It is worth noting that when $i = 0$, $\sum_{j=1}^{i} f_{i-j} g_i = 0$. Thus we get $F(x)G(x) = F(x) - f_0 x^0$, which yields:

$$F(x) = \frac{f_0 x^0}{1 - G(x)} \tag{8}$$

In Equation 8, the numerator is a constant while the denominator is a polynomial. Let polynomial $A = 1 - G(x)$, then $F(x) = f_0 x^0 A^{-1}$ and our goal becomes to find the inverse of polynomial $A$.

To derive the inverse of A, we first denote $A^{-1}$ using the notation $B$. It is clear that $AB = 1$. Since we only focus on the value of $f_n$, which indicates that our problem is defined under $x^n$. The equation $AB = 1$ holds if we apply modulus on both sides of the equal sign by $x^n$: $AB \equiv 1 \pmod{x^n}$.

We start with $n = 1$. $A = a_0 \pmod{x^1}$, so we have $B = \frac{1}{a_0}$ where $a_0$ denotes the coefficients of $x^0$ in polynomial $A$. Suppose that we get $A, B$ under the condition $AB \equiv 1 \pmod{x^m}$. Next we will find $B'$ such that $AB' \equiv 1 \pmod{x^{2m}}$ holds.

**Algorithm 2** Divide and Conquer + Inverse Polynomial Iteration

**Require:** $f_0$ and $\{g_i\}_{i=1}^n$

$\{f_i\}_{i=1}^n \leftarrow \{0\}_{i=1}^n$
$g_0 \leftarrow 0$

**function** SOLVEINVP($A$,$B$,$m$,$n$)
  **if** $m = n$ **then**
    **return**
  $A' = A \mod x^m$
  $B = A'B^2 - 2B$
  SolveInvP($A$, $B$, $2m$, $n$)

$n = 2^{\lceil \log_2 n \rceil}$
$a_0 = -g_0$
$B = \frac{1}{a_0}$
SolveInvP($1 - G$, $B$, $1$, $n$)
$F = f_0 x^0 B$
set the $i$th coefficient of $F$ to $\{f_i\}_{i=1}^n$
**return** $\{f_i\}_{i=1}^n$

Since $AB \equiv 1 \pmod{x^m}$, we can get $A^2B^2 \equiv AB \mod x^{2m}$ by multiplying both sides of the equation with $AB$. Then we have:

$$\begin{aligned} AB' &\equiv 1 \pmod{x^{2m}} \\ A \cdot AB^2 &\equiv 1 \pmod{x^{2m}} \\ A(AB^2 - 2B) &\equiv 1 \pmod{x^{2m}} \end{aligned}$$

Thus, $B' = (AB^2 - 2B)$, reveals how $B$ is updated when the order of a polynomial doubles. By finding $B$, we can get polynomial $F(x)$ and find the coefficients $f_0, f_1, \ldots, f_n$.

The pseudo-code of this algorithm is shown in 2.

## REFERENCES

J. van der Hoeven. Relax, but don't be too lazy. *JSC*, 34:479–542, 2002.