План лекции про FFT

Комплексные числа, полярные координаты

Многочлен <= (n - 1)-й степени однозначно задается значениями в n точках.
Хотим перемножить два многочлена p(x) и q(q).
Вычислим их значения в нескольких точках. Чтобы получить значения g(x) = p(x) * q(x) в этих же точках, нужно перемножить полученные значения. Теперь обратно по значениям в точках нужно получить коэффициенты многочлена.

Скажем, что n = 2^k > deg(p) + deg(q).
В качестве точек возьмем корни n-й степени из 1. Это точки вида root[j] = cos(2PI * j / n) + sin(2PI * j / n) * i, 0 <= j < n.

val[j] = p(root[j]) = sum_l p[l] * root[j]^l

Скажем, что p_0(x) = p[0] + p[2] * x + p[4] * x^2 + p[6] * x^3 ... (оставляем коэффициенты с четными номерами)
p_1(x) = p[1] + p[3] * x + p[5] * x^2 + ... (оставляем коэффициенты с нечетными номерами)

p(x) = p_0(x ^ 2) + p_1(x ^ 2) * x

root'[j] = cos(2PI * j / (n / 2)) + sin(2PI * j / (n / 2)) * i = root[j * 2] (в root' в два раза меньше точек)
val_0[j] = p_0(root'[j])
val_1[j] = p_1(root'[j])

val[j] =
= p(root[j]) =
= p_0(root[j] ^ 2) + p_1(root[j] ^ 2) * root[j] =
= p_0(root[j * 2]) + p_1(root[j * 2]) * root[j] =
= p_0(root'[j]) + p_1(root'[j]) * root[j] =
= val_0[j] + val_1[j] * root[j]

Вопрос. Что такое val_0[j] при j >= n / 2? На самом деле, это val_0[j % (n / 2)], потому что корни зацикленны.

Таким образом, для того, чтобы вычислить все val[j], можно разбиться на две задачи в два раза меньше, вычислить val_0[j] и val_1[j], а затем за линию пересчитать val[j]. Время работы O(nlogn).

Теперь нужно научиться по val[j] обратно получать p[j]. Давайте сделаем то же самое преобразование, передав ему в качестве коэффициентов многочлена значения val[j]. Получим:
val'[j] = sum_l val[l] * root[j] ^ l = (раскроем val[l])
= sum_l ((sum_h p[h] * root[l] ^ h) * root[j] ^ l) =
= sum_l sum_h (p[h] * root[l] ^ h * root[j] ^ l) =
= sum_h sum_l (p[h] * root[l] ^ h * root[j] ^ l) =
= sum_h ((sum_l root[l] ^ h * root[j] ^ l) * p[h]) =
= sum_h ((sum_l root[l * h + j * l]) * p[h]) =
= sum_h ((sum_l root[h + j] ^ l) * p[h]) =

Если h = −j (mod n), то
sum_l root[h + j] ^ l = n
Иначе,
sum_l root[h + j] ^ l = 0

= p[(n − j) % n] ∗ n

Magic!
Несложно понять, что осталось сделать, чтобы получить именно коэффициенты. Нужно поделить все значения на n, и развернуть отрезок [1, n).

Получилась рекурсивная реализация. Ее можно дополнительно ускорить, избавившись от рекурсии и сделав in-place.

Во-первых, сделаем алгоритм in-place. Пусть у нас есть массив arr, изначально arr[j] = p[j], а после работы алгоритма arr[j] = val[j]. Рекурсия будет принимать l и r -- границы полуинтервала. Она должна взять arr[l .. r] как коэффициенты, а в результаты записать в arr[l .. r] значения. Пусть m = (l + r) / 2. Сначала функция кладет arr[l], arr[l + 2], arr[l + 4], ... в arr[l .. m]; arr[l + 1], arr[l + 3], arr[l + 5], ... в arr[m .. r]. Затем вызывает rec(l, m) и rec(m, r). Затем пересчитывает значения.

Заметим, что сначала элементы как-то перемещаются, пока не достигнут нижнего слоя рекурсии. Давайте сразу применим эту перестановку. Тогда можно будет сразу идти от нижних слоев рекурсии вверх, пропустив фазу спуска и переупорядочивания элементов. Это позволит превратить рекурсию в цикл.

Заметки:
Можно использовать std::complex<double>. Однако, самописная структура будет быстрее.

Можно делать все то же самое не в комплексных числах, а по простому модулю. Для этого, должен существовать корень n-й степени по этому модулю. Корень степени n по простому модулю MOD существует, если (MOD − 1) делится на n. Искать корень можно линейным пробегом и проверкой. Для корня должно выполняться root^(n / 2) = −1 (mod MOD)

Можно перемножать длинные числа. Рассмотрим число как многочлен с коэффициентами равными цифрам. Тогда, значение числа равно p(10). Перемножим эти два многочлена и вычислим g(10). Для этого нужно будет идти от младших коэффициентов к старшим и поддерживать carry.