Testing k-cover, k-partition and k-packing

Mats Rydberg

September 11, 2013

1 Introduction

In this text, I will present some results from my work in the form of benchmarks for a few variations of the algorithms examined in this Master's Thesis. The core of the algorithm is the Fast Zeta Transform in Linear Space, see 3, which makes the algorithms look very much alike. The algorithm is presented in this paper in three versions, each solving a well-known algorithmical problem. These are k-cover, k-partition and k-packing, further explained below.

One question that this test seeks to answer, or at least make an indication toward an answer for, is whether the complexity of the arithmetics performed with polynomials in the partition and packing problems is a significant time-consuming part of the algorithm. That is, would it be worthwhile to represent the polynomials in a non-explicit way in order to perform faster multiplication and addition? In this text, I have provided two different alternatives to polynomial representation (and arithmetics).

2 Problem statements

2.1 Fast Zeta Transform

Given a universe $U = \{1, 2, \dots, n\}$ of integers, a family $\mathcal{F}$ of subsets of U and a function $f : \mathcal{F} \to R$, where R is an algebraic ring, output the number of ways to pick k members of $\mathcal{F}$ such that their union equals U.

For a detailed description of the algorithm, see Björklund et al [**?**].

2.2 k-cover

Given a universe $U = \{1, 2, \dots, n\}$ of integers, a family $\mathcal{F}$ of subsets of U and an integer k, output $c_k(\mathcal{F})$, which is the number of ways to pick k members of $\mathcal{F}$ such that their union equals U.

Solution: Set f as the characteristic function of $\mathcal{F}$ ($f = 1$ for each $Y \in \mathcal{F}$) and calculate its FZT $f\zeta$. Then get c_k as

$$c_k = \sum_{X \subseteq U} (-1)^{|U \setminus X|} f\zeta(X)^k$$

2.3 k-partition

Given a universe $U = \{1, 2, \dots, n\}$ of integers, a family $\mathcal{F}$ of subsets of U and an integer k, output $d_k^n(\mathcal{F})$, which is the number of ways to pick k pairwise disjoint members of $\mathcal{F}$ such that their union equals U.

Solution: Set $f(Y) = z^{|Y|}$ for each $Y \in \mathcal{F}$. Calculate $f\zeta$ and get d_k^n as the coefficient of z^n in

$$d_k = \sum_{X \subseteq U} (-1)^{|U \setminus X|} f\zeta(X)^k$$

2.4 k-packing

Given a universe $U = \{1, 2, \ldots, n\}$ of integers, a family $\mathcal{F}$ of subsets of U and an integer k, output $p_k^n(\mathcal{F})$, which is the number of ways to pick k pairwise disjoint members of $\mathcal{F}$.

Solution: With the insight that a k-packing is indeed a $(k+1)$-partition with the $(k+1)$:th member being an arbitrary subset of U (all the remaining elements in U), we set f as we did with k-partition and get p_k^n as the coefficient of z^n in

$$p_k = \sum_{X \subseteq U} (-1)^{|U \setminus X|} (1+z)^{|X|} f\zeta(X)^k$$

3 Fast Zeta Transform in Linear Space

The core of the algorithm is the Fast Zeta Transform (FZT) in Linear Space, presented by Björklund et al in [**?**]. The FZT is directly translated into a solution for the three problems mentioned above, by a simple line of summation. But we perform tests on this core also, and try to determine its alleged linear increase in complexity as $|\mathcal{F}|$ increases.

4 Representation of polynomials

How to handle polynomial representation is an open question, relevant for k-partition and k-packing. Two approaches comes to mind; firstly the explicit, direct representation of polynomials as a vector of coefficients, where addition and multiplication is performed in time polynomial to the degree of the polynomial. Secondly, one could evaluate each polynomial at sufficiently many (constant number?) small integers, and recover the coefficients by interpolation and the Chinese Remainder Theorem [**?**].

5 First implications and limitations

My initial idea was to test many things. Too many, I soon realized, as some simple calculations showed that I would need over a hundred graphs to present the results. This is of course infeasible due to lack of space and time (no pun intended). So, instead of allowing both `std::vector`- and pointer-based list structures, I will resort to only keep pointer-based lists. Although it would be interesting to see whether `std::vector` provides a lot of overhead for large structures, this will have to be deduced elsewhere.

When performing intitial try-outs for my problem instances, I soon realized that it wasn't the sums of the FZT that would be the problem, as I initially had believed. Instead, the large sum is (of course) the result sum of the various problems, since that is the sum that has its terms raised to k. This is partly due to the fact that the algorithm considers the solutions S_1, S_2 and S_2, S_1 equal, the number of solutions is not just the every correct combination of sets, but also all permutations of every combinations (which means that a unique solution is multiplied by a factor $O(k!)$). Fact is, the numbers grow so fast that my plan of testing the algorithms using standard integral types as well as multiple precision types are of no use; a 32-bit integer rolls over even for $n = 12$ in one of the instances.

5.1 Complexity of arithmetics

6 Test structure

6.1 FZT in Linear Space

6.2 k-cover

6.3 k-partition

6.4 k-packing

7 Conclusions