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

Mats Rydberg

September 12, 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 **??**, 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 $U = \{1, 2, \ldots, n\}$ of integers, a family $\mathcal{F}$ of subsets of U and a function $f : \mathcal{F} \to R$, where R is an algebraic ring[1], output the Fast Zeta Transform $f\zeta$ of f, given by

$$f\zeta(X) = \sum_{Y \subseteq X} f(Y)$$

For a detailed description of the algorithm, see Björklund et al [1].

2.2 k-cover

Given a universe $U = \{1, 2, \ldots, n\}$ 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 $U = \{1, 2, \ldots, n\}$ 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

[1]In this paper, R is either the set of integers modulo 2 $\mathbb{Z}_2$ or the ring of polynomials $\mathbb{Z}_2[z]$ of degree n with coefficients from $\mathbb{Z}_2$ (for some indeterminant z).

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

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 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 [**?**].

4 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.

There are two kinds of integer types used in my program. First there are the index variables for my list structures. They need to be large enough to hold all of the members of $\mathcal{F}$. Since $|\mathcal{F}|$ grows as $O^*(2^n)$, it is the size of n that sets the limitation on our index variables. For convenience, I choose to let my program support $n < 31$, allowing me to use a standard 32-bit sized integer. For the instances provided here, maximum input size is $\sim 10^6$ lines, which means that for $n = 30$ our maximum density[2] of subsets is around $1‰$. For $n < 21$, there is no limitation on subset density.

Second, there are the summation variables needed to output the solution for the k-problems. In order to produce correct results, these do naturally need to be arbitrarily large. My programs uses GMP, the GNU Multiple Precision library [2], to support this. An initial idea was to test also the performance of GMP as compared to any standard data type (such as `int`), but as 32 bits is too small even for $n = 12$ (for some k), this was abandoned. The main reason as to why the sums grow so incredibly fast is due

[2]In this paper, the *density df* of $\mathcal{F}$ is defined as $|\mathcal{F}| = df \cdot 2^n$.

to the fact that the algorithm consider ordering of subsets in a k-sized selection to be critical. That is, for k distinct chosen members of $\mathcal{F}$ that solve the problem, there are $k!$ solutions counted.

4.1 Complexity of arithmetics

Since the FZT is running in time $O^*(2^n)$ and space $O^*(|\mathcal{F}|)$, so do the k-problem algorithms as well, assuming that the arithmetics in the ring R take constant time and that each ring element uses constant space. But what about the summation variables of the k-problems? As mentioned, they need to be stored in an arbitrarily large data type, but how large? How fast does it grow and what implications does this have on its use as a term or a factor?

For k-cover, we have an integer that in each step of the inner-most loop (there are $O^*(2^n)$ such steps) take the FZT $f\zeta(X)$ for some $X \in \mathcal{F}$ and raise it to the k. The largest possible value of $f\zeta(X)$ is 2^n. This happens when the density of subsets is at 100% and only for $X = U$ (which has all other sets in $\mathcal{F}$ as subsets). This (and every other) number is taken to the power of k. In k-cover, a set can be picked more than once for a given selection that solves the problem, leaving us with no bound at all on k. This means that the maximum value of the largest term in the sum of c_k is 2^{nk}, where k may approach ∞. In my programs, I have decided to let $k < |\mathcal{F}|$ hold, effectively bounding k by 2^n. Our maximum term is now $(2^n)^{2^n}$.

TODO: Not sure where I am going with this... relevance? What's my point? How hard is it exactly to sum with this number?

5 Test structure

All problem instances are stored in a file named `*p_n_df` where `*` is `k` for the k-problems and `f` for the FZT, `n` is the size of n and `df` is the density *df* of $\mathcal{F}$ in parts of thousands (‰).

My measures are performed by pausing the process just before normal termination and reading from the process' stat file in `/proc/[pid]/stat` (see the linux man pages for `proc` [3]). The information gathered is the amount of time used (both user and system time), the peak size of the resident set and the peak virtual memory.

The tests are as follows (k is of course not a parameter for the FZT).

1. Sets n and k constant and varies *df* over 20 measuring points, from 1‰ to 1000‰.

2. Sets $|\mathcal{F}|$ and k constant and varies n (this means *df* also varies, but that's not relevant) over 7 measuring points, from 14 to 20.

3. Sets n and $|\mathcal{F}|$ constant and varies k over 100 measuring points.

5.1 FZT in Linear Space

5.2 k-cover

5.3 k-partition

5.4 k-packing

6 Conclusions

References

[1] http://fileadmin.cs.lth.se/cs/Personal/Thore_Husfeldt/papers/lsfzt.pdf

[2] http://gmplib.org/

[3] http://man7.org/linux/man-pages/man5/proc.5.html