STREAMS OF COMBINATORIAL OBJECTS IN JAVA USING VIRTUAL COLLECTIONS

RALPH FREESE

ABSTRACT. Methods are given for constructing streams (and parallel streams) of combinatorial objects using immutable lists indexed by `long`'s with no backing structure. Examples include all tuples of a fixed length from a list of objects, all such tuples with the minimum entry at least m, for a given m, all subsets of fixed size of a given set, all permutations of a set, etc.

INTRODUCTION

The Java language has introduced streams which, among other things, are useful for parallel computing. Given a list, `mylist`, in Java we can obtain a stream simply using the stream method: `mylist.stream()`. But suppose, for example, we want to create a stream of all 3-tuples of elements from our list. So, if our list is $(a_0, a_1, \ldots, a_{n-1})$, then we would like to create the stream

$$(a_0, a_0, a_0), (a_1, a_0, a_0), \ldots, (a_{n-1}, a_{n-1}, a_{n-1}). \tag{1}$$

This steam will have n^3 elements so we need to create the stream without creating the list of all 3-tuples. We represent lists such as (1) with a simple interface we call `LongList`. These lists are immutable and have no backing collection so we sometimes refer to them as *virtual lists*. The elements are indexed by `long`'s; hence the name.

```
public interface LongList<E> {
  public long size();
  public E get(long k);
}
```

This interface also has methods `stream()` and `parallelStream()` which have default methods. For example,

```
default public Stream<E> parallelStream() {
  return LongStream.range(0, size()).parallel().
     mapToObj(i -> get(i));
}
```

Since Java's `LongStream.range` produces easily parallelized streams, `LongList`'s `parallelStream()` method can be effectively used in parallel algorithms.

But the key is the `get` method: it needs be efficient and thread-safe. The main goal of this paper is to give stateless (or at least thread-safe), efficient getters (implementations of `get(long k)`) for various long lists of combinatorial objects.

Date: October 4, 2015.

Besides r-tuples, we show how to write getters for all subsets, for all subsets of size r, for all permutations, etc.

Returning to 3-tuples, we first observe it is enough, given n, to produce the long list of `int` arrays

$$[0,0,0],\ [1,0,0]\,,\ldots,\ [n-1,n-1,n-1]. \tag{2}$$

since we can create the stream (1) by using the triples from (2) as indices of triples from our list. The `LongList` interface includes static methods for constructing the list (2) and other combinatorial lists as a `LongList`. For example,

```
public static LongList<int[]> intTuples(int
    tupleLength, int base)

public static LongList<int[]> intTuplesWithMin(int
    tupleLength, int base, int min)

public static LongList<int[]> permutations(int n)
```

The Java code is available at

`https://github.com/UACalc/CombinatorialStreams`.

Getters for combinatorial lists

r-tuples. As in the introduction we want a getter for the set T of all `int` arrays of length r whose entries are nonnegative integers less than n. The getter is essentially the algorithm used for a change of arithmetical basis: given $0 \leq k < n^r$ divide k by n, giving a quotient q and remainder s, so that, $k = nq + s$. Put the remainder into the first entry of an array of length r. Now divide the quotient by n and put the remainder of that division into the second position. And continue.

r-tuples with a specified minimum. Let $0 \leq m < n$. Let $S \subseteq T$ be those elements of T with at least one entry at least m. Such a stream would be useful if, for example, an earlier part of a program has already dealt with all r-tuples with entries less than m and the program now needs deal with r-tuples with entries less than n.

Writing a getter for S is more difficult than the previous example. Note

$$|S| = n^r - m^r = (n-m)\sum_{i=0}^{r-1} n^{i-1}m^{r-i}.$$

This suggests we write S as the disjoint union of sets S_i of size $n^{i-1}(n-m)m^{r-i}$, for $i = 1, \ldots, r-1$. To do this, we let

$$S_i = \{[a_0, \ldots, a_{r-1}] \in T : a_i \geq m, \text{ and } a_j < m, \text{ for } j = i+1, \ldots, r-1\},$$

so the i^{th} entry is at least m, the entries to the right are less than m and the entries to the left are arbitrary. Of course $|S_i| = n^{i-1}(n-m)m^{r-i}$. For $0 \leq t < r$ define

$$s_t := \sum_{i=1}^{t} n^{i-1}(n-m)m^{r-i}.$$

Given $0 \leq k < n^r$, our getter first finds t with $s_t \leq k < s_{t+1}$. Set $k' = k - s_t$. We now construct our array as in the previous example. First we divide k' by n and put the remainder in the first coordinate of our array. Then we divide the quotient of the previous division by n and put the remainder in the second coordinate. We repeat this t times. Next we divide the current quotient by $n - m$ and put the remainder plus m into the t^{th} coordinate. Then we divide the current quotient by m, putting the remainder into the next entry and repeat until we have filled the array.

Of course another option would be to produce the stream in this example would be to just stream all n^r tuples and filter out unwanted ones. But if m is nearly equal to n, then n^r can be close to n/r times as large as $n^r - m^r$.

Subsets of a set. We represent subsets of $\{0, 1, \ldots, n-1\}$ as array of `int`'s $[a_0, a_1, \ldots, a_{r-1}]$ with $a_0 < a_1 < \cdots < a_{r-1} < n$. In our list of subsets we put all subsets of $\{0, 1, \ldots, n-2\}$ first, before any subset containing $n - 1$. ???

Subsets of a fixed size r. We represent subsets of $\{0, 1, \ldots, r-1\}$ by arrays $[a_0, \ldots, a_{r-1}]$ with $0 \leq a_0 < \cdots < a_{r-1} < n$. The order of our list of all of these will effectively be lexicographic starting at the right end of the list. So all subsets whose maximal (right most) entry is t come before all subsets whose maximal element is less that t. So the last entry of the k^{th} subset is t, where

$$\binom{t}{r} \leq k < \binom{t+1}{r}. \tag{3}$$

Letting $k' = k - \binom{t}{r}$, the second to the last entry is t', where

$$\binom{t'}{r-1} \leq k' < \binom{t'+1}{r-1},$$

etc.

Approximations of binomial coefficients suggest that t in (3), given k and r, will approximately be

$$t = \lfloor (r!k)^{1/r} \rfloor + \lfloor r/2 \rfloor,$$

So we start with this an adjust t if necessary so that it satisfies (3).

Department of Mathematics, University of Hawaii at Manoa, U.S.A.
E-mail address: `ralph@math.hawaii.edu`