From cain.cs.curtin.edu.au!smyth Sat Nov 16 19:45:22 1996 To: bill@maccs.dcss.mcmaster.ca Subject: distinct.tex

Version 17/11/96

COUNTING DISTINCT STRINGS

Dennis Moore

School of Computing
Curtin University of Technology
e-mail: `moore@cs.curtin.edu.au`

W. F. Smyth

Department of Computer Science & Systems
McMaster University
e-mail: `smyth@mcmaster.ca`

School of Computing
Curtin University of Technology
e-mail: `smyth@cs.curtin.edu.au`

Dianne Miller

Department of Computer Science & Systems
McMaster University
e-mail: `dmiller@maccs.dcss.mcmaster.ca`

ABSTRACT

This paper discusses how to count and generate strings that are "distinct" in two senses: p-distinct and b-distinct. Two strings x on alphabet A and x' on alphabet A' are said to be p-distinct iff they represent distinct "patterns"; that is, iff there exists no 1-1 mapping $A \leftrightarrow A'$ that transforms $x \leftrightarrow x'$. Thus aab and baa are p-distinct while aab and ddc are p-equivalent. On the other hand, x and x' are said to be b-distinct iff they give rise to distinct border (failure function) arrays: thus aab with border array 010 is b-distinct from aba with border array 001. The number of p-distinct (respectively, b-distinct) strings of length n formed using exactly k different letters is the $[k,n]$ entry in an infinite p' (respectively, b') array. Column sums $p[n]$ and $b[n]$ in these arrays give the number of distinct strings of length n. We present algorithms to compute, in constant time per string, all p-distinct (respectively, b-distinct) strings of length n formed using exactly k letters, and we also show how to compute all elements $p'[k,n]$ and $b'[k,n]$. These ideas and results have application to the efficient generation of appropriate test data sets for many string algorithms.

1 INTRODUCTION

When is a string "distinct" from another? The answer to this question depends on how we intend to process the string. For some purposes we might choose to regard $x = abbcc$ and $x' = bccaa$ as distinct; if, however, we regard the letters of the alphabet as interchangeable, so that x and x' can be seen as conforming to the same "pattern", then we might prefer to think of x as being equivalent to x' in a well-defined sense.

This would be true, for example, if we were generating test data for an algorithm which recognized no ordering of the alphabet (say, an algorithm to compute all repetitions [ML84] in a string): in this case, if the algorithm executed correctly on input x, it would do so also on input x'.

To make this idea precise, let

$$x = x[1]x[2]\cdots x[n] = x[1..n], \quad x' = x'[1]x'[2]\cdots x'[n] = x'[1..n]$$

denote arbitrary finite strings of length $|x| = n \geq 1$. We say that x is *p-equivalent* to x' if and only if, for all integers i and j satisfying $1 \leq i \leq j \leq n$,

$$x[i] = x[j] \quad \Leftrightarrow \quad x'[i] = x'[j].$$

Clearly p-equivalence is an equivalence relation, breaking down the strings of length n into equivalence classes. Strings that are not p-equivalent are said to be *p-distinct.*

Another interpretation of "distinctness" is possible. Recall that a string x is said to have *border* u if and only if u is a proper prefix and suffix of x. For example, $x = abaabaab$ has borders $u = \epsilon$ (the empty string), ab and $abaab$, of lengths 0, 2 and 5, respectively. The *border array* $\beta_n = \beta[1..n]$ corresponding to $x_n = x[1..n]$ is a string defined on the integer alphabet $\{0, 1, \ldots, n-1\}$ in which, for every integer $j \in 1..n$, $\beta[j]$ is the length of the longest border of $x_j = x[1..j]$. ($\beta[j]$ is also referred to as the "failure function" of x_j [AHU74].)

We say that two strings are *b-equivalent* if and only if they give rise to identical border arrays. Strings that are not b-equivalent are said to be *b-distinct.* Thus, for example, even though $x_5 = ababb$ and $x'_5 = ababc$ are p-distinct, we find that they are nevertheless b-equivalent since both correspond to the border array $\beta_5 = 00120$. On the other hand, x_5 and $x''_5 = abacb$ are b-distinct since they give rise to distinct border arrays 00120 and 00100, respectively. It is clear then that each distinct valid border array determines an equivalence class of b-equivalent strings. Observe that two b-distinct strings are necessarily also p-distinct (so that p-equivalent strings are necessarily also b-equivalent); as we have just seen, the converse is not true.

In this paper we consider the two kinds of distinctness described above; for each, and for all positive integers k and n, we show how to

* generate (in only constant time per string) all distinct strings of length n formed using exactly k letters;
* count the number of all such strings.

In particular, we shall see that the number of p-distinct patterns of length n formed using exactly k letters is $\left\{ {n \atop k} \right\}$, a Stirling number of the second kind, a fact apparently not previously observed. We shall see therefore (equation (2.5)) that the total number of p-distinct strings of length n using at most k letters is reduced by an asymptotic factor of $1/k!$ from the number of such strings that are distinct in the ordinary sense. Moreover, the computation of b-distinct patterns leads to a sequence of integers that is apparently new [S73], and that represents a decline, by a further exponential factor, from the number of p-distinct patterns (Theorem 3.3(f) and equation (3.1)). Algorithms for generating distinct strings have been implemented in a software package for the testing of string algorithms [L96].

2 DISTINCT PATTERNS

In this section we discuss p-distinct strings: how to count them and how to generate them. In order to do so, it is convenient to identify a unique representative of each p-distinct equivalence class. We therefore introduce a countably infinite *standard alphabet*

$$\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k, \ldots.\}, \qquad \ldots(2.1)$$

with subalphabets $\Lambda_k = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ for every integer $k \geq 1$. We suppose the letters of Λ to be naturally ordered according to $\lambda_1 < \lambda_2 < \cdots < \lambda_k < \cdots$. Then, given any string $x = x[1..n]$ on any alphabet A, we define the *p-canonical* string x^* corresponding to x to be the lexicographically least string on Λ that is p-equivalent to x. It is clear that x^* satisfies the following property:

(P) For every positive integer j, the first occurrence (if any) of λ_j in x^* precedes the first occurrence of λ_{j+1}.

We first concern ourselves with the problem of counting the number $p'[k, n]$ of p-canonical strings x^* of length n formed using exactly the letters of Λ_k. We imagine these values to be laid out in an infinite two-dimensional array called the p' array.

Theorem 2.1 For any positive integers n and k:

(a) $p'[1, n] = 1$;
(b) if $k > n$, $p'[k, n] = 0$;
(c) $p'[k, k] = 1$;
(d) if $k \geq 2$ and $n \geq 2$, $p'[k, n] = p'[k-1, n-1] + kp'[k, n-1]$.

Proof (a) For $k = 1$, the only p-canonical string is $x^* = \lambda_1^n$.

(b) By property (P), no p-canonical string x^* can contain a letter λ_k, $k > n$.

(c) Again by property (P), there exists exactly one p-canonical string of length k formed using exactly k distinct letters: $x^* = \lambda_1\lambda_2\cdots\lambda_k$.

(d) Let $\pi_1 = p'[k-1, n-1]$ denote the number of distinct p-canonical strings of length $n-1$ that include exactly the $k-1$ letters of Λ_{k-1}. Denote these strings by

$$S_1 = \{x_1, x_2, \ldots, x_{\pi_1}\}.$$

Then for every integer i satisfying $1 \leq i \leq \pi_1$, each string

$$x_i \lambda_k \qquad \ldots (2.2)$$

is distinct and p-canonical.

Similarly, let $\pi_2 = p'[k, n-1]$ denote the number of distinct p-canonical strings of length $n-1$ on exactly k distinct letters Λ_k. Denote these strings by

$$S_2 = \{y_1, y_2, \ldots, y_{\pi_2}\}.$$

Then for every integer i satisfying $1 \leq i \leq \pi_2$, the k strings

$$\{y_i\lambda_1, y_i\lambda_2, \ldots, y_i\lambda_k\} \qquad \ldots (2.3)$$

must all be distinct and p-canonical. Further, since the distinct final letter occurs at least twice in each string, each of these strings is distinct from any of the strings (2.2). Thus $p'[k, n] \geq p'[k-1, n-1] + kp'[k, n-1]$.

Suppose now that x^* is a p-canonical string of length n formed using exactly the letters Λ_k. Let $x^* = y^*\lambda_i$. If λ_i occurs in y^*, then $y^* \in S_2$ and therefore x^* is one of the strings (2.3). Otherwise, by property (P), λ_k cannot occur in y^* either, and so $i = k$, $y^* \in S_1$, and x^* is one of the strings (2.2). We conclude that $p'[k, n] \leq p'[k-1, n-1] + kp'[k, n-1]$, and so the result is proved. □

The recurrence relation of Theorem 2.1(d) is well-known; with the initial values specified by Theorem 2.1(a)-(c), it defines the Stirling numbers $\left\{ {n \atop k} \right\}$ of the second kind ([K68], [PTW83]). Hence

$$p'[k, n] = \left\{ {n \atop k} \right\} \qquad \ldots (2.4)$$

for all positive integers n and k. In fact, as we illustrate with an example, the correspondence between classical Stirling numbers and our $p'[k, n]$ values can be made in another way. A common definition [PTW83] of $\left\{ {n \atop k} \right\}$ is the number of ways that a set S of n elements can be decomposed into k nonempty nonintersecting subsets whose union is S. To see how this definition corresponds to $p'[k, n]$, consider the case $n = 4$, $k = 2$. If we write down the seven strings counted by $p'[2, 4]$ and collect into $k = 2$ subsets the *indices* of identical letters in these strings, we find that each pair of subsets is a unique (because each string is distinct) decomposition of $\{1, 2, 3, 4\}$ into nonempty (because each of the k letters occurs) nonintersecting (because each position contains exactly one letter) subsets:

1234

$aaab$	$\{1,2,3\}\ \{4\}$
$aaba$	$\{1,2,4\}\ \{3\}$
$aabb$	$\{1,2\}\ \{3,4\}$
$abaa$	$\{1,3,4\}\ \{2\}$
$abab$	$\{1,3\}\ \{2,4\}$
$abba$	$\{1,4\}\ \{2,3\}$
$abbb$	$\{1\}\ \{2,3,4\}$

The unions of the pairs of sets in the righthand column exhaust all the possible ways of forming $S = \{1,2,3,4\}$ from $k = 2$ nonempty nonintersecting subsets.

Theorem 2.1(d) provides an iterative method of computing $p'[k,n]$ and various formulæ for direct computation are available in the literature [R68]. Observe that, for any fixed value of k, the partial column sum $\sum_{i=1}^{k} p'[i,n]$ is the number of p-distinct strings of length n formed from at most k letters. Since for n large with respect to k almost all of these strings contain exactly k letters, it follows that

$$\lim_{n\to\infty}\left(\sum_{i=1}^{k} p'[i,n] \Big/ \frac{k^n}{k!}\right) = 1. \qquad \ldots(2.5)$$

In the usual meaning of distinctness in strings, the number of distinct strings of length n formed from at most k letters is k^n. Thus (2.5) tells us that using p-distinct strings on an alphabet of fixed size k reduces the number of strings that need to be generated by an asymptotic factor of $1/k!$. Of particular interest is the case

$$p[n] \equiv \sum_{i=1}^{n} p'[i,n],$$

the number of p-distinct strings of length n, known in the literature as Bell numbers [S73]. These numbers also can be computed directly or iteratively in various ways [R68, PBM86], in particular using

$$p[n] = \sum_{j=0}^{n-1} \binom{n-1}{j} p[j], \qquad \ldots(2.6)$$

$p[0] \equiv 0$, that avoids any reference to the p' values. The first few Bell numbers are $p[1] = 1$, $p[2] = 2$, $p[3] = 5$, $p[4] = 15$, $p[5] = 52$, $p[6] = 203$. By contrast, there are 46,656 distinct (in the ordinary sense) strings of length 6 on an alphabet of 6 letters.

We conclude this section with a discussion of the generation of p-canonical strings. It is clear from the proof of Theorem 2.1(d) that, in order to generate all the strings counted by $p'[k,n]$, we

* append λ_k to the strings counted by $p'[k-1, n-1]$;
* append $\lambda_1, \lambda_2, \ldots, \lambda_k$ to the strings counted by $p'[k, n-1]$.

This observation gives rise to straightforward recursive algorithms to generate either *all* the p-canonical strings x^* counted by $p'[k, n]$ or else *pseudorandom* strings x^*. The generation of each pseudorandom string will necessarily require $\Theta(n)$ time, but the generation of all p-canonical strings of length n can actually be accomplished in constant time per string by making use of a rooted tree structure T_n of height n, as described below.

The nodes of T_n may be thought of as pairs (λ, k), where λ is a letter of Λ and k is the number of distinct letters λ found in the nodes which lie on the path to the current node from the root. T_1 consists of the single root node $(\lambda_1, 1)$, and for every integer $n \geq 2$, T_n is formed by adding the following children to every leaf node (λ, k) of T_{n-1}:

$$(\lambda_1, k), (\lambda_2, k), \ldots, (\lambda_k, k), (\lambda_{k+1}, k+1).$$

It is easy to see that T_n has exactly $p[n]$ leaf nodes and that the letters found on the paths to these nodes from the root give exactly the $p[n]$ p-canonical strings x^* of length n. Thus the generation of these strings x^* is accomplished simply by generating T_n. Observe that, for every integer $n \geq 2$, T_n is formed from T_{n-1} by appending $p[n]$ leaf nodes, a task requiring $\Theta(p[n])$ time. Since by (2.6) $p[n] \geq 2p[n-1]$, it follows that T_n can be constructed in $\Theta(p[n])$ time.

Theorem 2.2 For every positive integer n, all $p[n]$ p-canonical strings of length n can be computed in $\Theta(p[n])$ time and represented in $\Theta(p[n])$ space. □

We may establish a similar result for the generation of all p-canonical strings counted by $p'[k, n]$. In this case we generate only the subtree of T_n whose paths of length n terminate at a vertex whose label is (λ, k) for any letter λ; these paths represent exactly the $p'[k, n]$ p-canonical strings of length n which contain exactly k letters. Thus in this case only the nodes on these paths need to be computed, and so we have

Theorem 2.3 For all positive integers k and $n \geq k$, all $p'[k, n]$ p-canonical strings of length n formed using exactly k letters can be computed in $O(kp'[k, n])$ time and represented in $O(kp'[k, n])$ space.

Proof The recurrence relation of Theorem 2.1(d) implies that, in order to compute the strings counted by $p'[k, n]$, k diagonal entries

$$p'[k, n-j], p'[k-1, n-j-1], \ldots, p'[1, n-j-k+1]$$

need to be computed for every integer $j = 0, 1, \ldots, n-k$. For every such integer j, let

$$D_{k,n-j} = \sum_{i=0}^{k-1} p'[k-i, n-i-j]$$

denote the sum of the terms in the $(n-j)^{\text{th}}$ diagonal. Observe that, since $p'[k, n-j]$ is the largest element in its diagonal, $kp'[k, n-j] \geq D_{k,n-j}$, with equality if and only if $j = n-k$. Further, it follows from the recurrence relation that

$$p'[k, n-j] > kp'[k, n-j-1] \geq D_{k,n-j-1},$$

provided $n-j > 1$. Hence

$$\begin{aligned}\sum_{j=0}^{n-k} D_{k,n-j} &\leq kp'[k,n] + p'[k,n](1 + 1/k + \cdots + 1/k^{n-k-1}) \\ &\leq (k+2)p'[k,n],\end{aligned}$$

and the result follows. □

We remark finally that the tree T_n may be traversed in various ways corresponding to various orderings of the p-canonical strings. For example, preorder traversal of T_n (or any subtree of it generated by $p'[k, n]$) yields the strings in lexicographic order; so also does postorder traversal if the empty letter is assumed to sort largest. In fact, if each string of T_n can be discarded after generation, then the strings determined by T_n can actually be generated using only $\Theta(n)$ storage, corresponding to either preorder or postorder traversal of T_n. Since by (2.6) $p[n] \leq 2^n$, this reduces the storage requirement to $\Theta(\log p[n])$.

3 DISTINCT BORDER ARRAYS

In this section we consider how to generate and how to count b-distinct strings. We begin with a series of lemmas that show how b-distinct strings of length $n+1$ can be derived from those of length n.

Among any class of b-equivalent strings, it will again be convenient to identify one *b-canonical* string x^* as a representative of its class: as with p-canonical strings, we choose this string to be the lexicographically least among those strings on the standard alphabet that are in the class. Every class of b-equivalent strings on Λ is of infinite cardinality, but we can simplify matters without loss of generality by restricting such classes only to strings that are also p-canonical. Then, for example, the class of p-canonical b-equivalent strings on Λ corresponding to $\beta_6 = 001000$ is

$$S_6 = \{\lambda_1\lambda_2\lambda_1\lambda_3\lambda_2\lambda_2, \lambda_1\lambda_2\lambda_1\lambda_3\lambda_2\lambda_3, \lambda_1\lambda_2\lambda_1\lambda_3\lambda_3\lambda_2, \lambda_1\lambda_2\lambda_1\lambda_3\lambda_3\lambda_3\},$$

with b-canonical element $x_6^* = \lambda_1\lambda_2\lambda_1\lambda_3\lambda_2\lambda_2$.

In order to establish a recurrence to compute a b-canonical string $x_{n+1}^* = x^*[1..n+1]$ from a b-canonical string $x_n^* = x^*[1..n]$, we need to understand how β_{n+1} is computed from β_n. Let $\beta^i[n]$, $i \geq 1$, denote $\beta[\beta^{i-1}[n]]$, where $\beta^0[n] \equiv n$. We state without proof a lemma on which the standard failure function algorithm [AHU74] is based:

Lemma 3.1 Let β_n denote the border array of some string x_n of length $n \geq 1$, and let $k < n$ be the least integer such that $\beta^k[n] = 0$. Then for integers $i \in 1..k$,

(a) the borders of x_n are exactly $x_{\beta^i[n]} = x[1..\beta^i[n]]$;

(b) for any string x_{n+1} with proper prefix x_n, $\beta_{n+1} = \beta_n\beta[n+1]$, where $\beta[n+1] \in \{0;\ \beta^i[n]+1\}$. □

This result describes the values that may possibly be assumed by $\beta[n+1]$, given $\beta_n = \beta[1..n]$. We now prove a much stronger result, that the set of values *actually* assumed by $\beta[n+1]$ is independent of the underlying string x_n.

Lemma 3.2 For $n \geq 1$, the values assumed by $\beta[n+1]$ depend only on β_n and the size of the alphabet.

Proof Suppose that there exist two strings x_n and y_n, both defined on alphabets of size α, and both giving rise to border array β_n. Suppose further that for some letter λ, $x_{n+1} = x_n\lambda$ gives rise to border array $\beta_{n+1} = \beta_n m$, but that there exists no letter μ such that $y_{n+1} = y_n\mu$ gives rise to β_{n+1}. Then $\beta[n+1] = m$ is one of the values specified in Lemma 3.1(b).

First consider the case $m = \beta^i[n]+1$ for some integer $i \in 1..k$. Since m does not occur for any y_{n+1}, it follows that

$$y[\beta^i[n]+1] = y[\beta^{i'}[n]+1]$$

for some least $i' < i$, while on the other hand

$$x[\beta^i[n]+1] \neq x[\beta^{i'}[n]+1].$$

Let $n' = \beta^{i'}[n]$. Then

$$y[n'+1] = y[\beta^{i''}[n']+1],$$

where $i'' = i - i'$, so that $\beta[n'+1] = \beta[\beta^{i''}[n']+1]$. However, since

$$x[n'+1] \neq x[\beta^{i''}[n']+1],$$

we conclude $\beta[n'+1] \neq \beta[\beta^{i''}[n']+1]$, a contradiction since we assumed that both x_n and y_n gave rise to β_n. Thus the lemma holds for every $m = \beta^i[n]+1$.

Now suppose that $m = 0$. Then every one of the α possible choices $y[n+1] = \mu$ yields a unique value $\beta[n+1] \neq 0$, while at least one choice $x[n+1] = \lambda$ gives rise to $\beta[n+1] = 0$. Hence there exists $m' > 0$ such that $y[n+1]$ yields $\beta[n+1] = m'$ while $x[n+1]$ does not yield $\beta[n+1] = m'$, in contradiction to the previous case.

We conclude that β_{n+1} is a border array of some x_{n+1} if and only if it is a border array of some y_{n+1}. □

This fundamental result raises the possibility, discussed below, that β_{n+1} can be computed from β_n without reference to any specific string. We can use the result immediately, however, to show that every b-canonical string x^*_{n+1} must have a b-canonical string as a prefix:

Lemma 3.3 For $n \geq 1$, every b-canonical string $x^*_{n+1} = x^*_n\lambda$, where x^*_n is also b-canonical and λ is some letter of the standard alphabet.

Proof Suppose $x^*_{n+1} = x_n\lambda$ with associated border array β_{n+1}, where x_n is a string of length n that is not b-canonical. Suppose that x_n has border array β_n. Then there exists a string $y_n < x_n$ with border array β_n. Hence by Lemma 3.2 there also exists $y_{n+1} = y_n\lambda'$ with border array β_{n+1}, where $y_{n+1} < x^*_{n+1}$. But then x^*_{n+1} is not b-canonical, a contradiction. □

It is thus clear that *all* of the b-canonical strings x^*_{n+1} can be formed from b-canonical strings x^*_n — no other strings need be considered. This foreshadows a tree structure similar to that of Section 2, where strings x^*_{n+1} are children of strings x^*_n. The next lemma provides more exact information about how to generate distinct border arrays β_{n+1} from a given β_n, and also about the form of the associated b-canonical strings x^*_{n+1}.

Lemma 3.4 Suppose a border array β_n corresponds to a b-canonical string x^*_n on the standard alphabet Λ. Then β_n gives rise to exactly κ distinct border arrays β_{n+1} if and only if $x^*_n\lambda_\kappa$ is a b-canonical string that corresponds to $\beta^{(0)}_{n+1} = \beta_n 0$.

Proof Suppose first that $x_{n+1} = x^*_n\lambda_\kappa$ is b-canonical and has only the empty border. Then, since every b-canonical string corresponding to a given border array must be lexicographically least, it follows that there exists no λ_i, $i < \kappa$, such that $x^*_n\lambda_i$ has only the empty border; that is, for every $i \in 1..\kappa-1$, every $x^*_n\lambda_i$ has a distinct nonempty border.

Now suppose that for some integer $i > \kappa$, the b-canonical string $x^*_n\lambda_i$ has a longest border of length $m > 0$, so that $\beta_{n+1} = \beta_n m$. (Note that in fact, since $m \geq i > \kappa \geq 2$, $m \geq 3$.) It follows from Lemma 3.3 that x^*_n has a b-canonical prefix $x^*_m = x^*_{m-1}\lambda_i$ for some b-canonical string x^*_{m-1}. Moreover, since $x^*_n\lambda_\kappa$ has only the empty border, it follows that the prefix $x^*_{m-1}\lambda_\kappa$ also has only the empty border. Then for some positive integer $\kappa' \leq \kappa$, $x^*_{m-1}\lambda_{\kappa'}$ is a b-canonical string with only the empty border while $x^*_{m-1}\lambda_i$, $i > \kappa'$, is a b-canonical string with a nonempty border. In other words, we have reduced an instance of a problem for finite positive integers n and κ to an instance of exactly the same problem for finite positive integers $m-1$ and κ'. This reduction can therefore be continued indefinitely, an impossibility which persuades us that there exists no $i > \kappa$ such that $x_n^*\lambda_i$ has a nonempty border. Thus there are exactly κ distinct border arrays β_{n+1}, and sufficiency is proved.

To prove necessity, suppose that there exist exactly κ distinct border arrays β_{n+1}. But then one of them must be $\beta_n 0$ and, as we have just seen, must correspond to $x_n^*\lambda_\kappa$. □

It is noteworthy that Lemma 3.4 does not necessarily hold on a finite alphabet Λ_k; in other words, it holds only if the alphabet is sufficiently large. For example, on the alphabet $\Lambda_3 = \{\lambda_1, \lambda_2, \lambda_3\}$, the b-canonical string $x_7^* = \lambda_1\lambda_2\lambda_1\lambda_3\lambda_1\lambda_2\lambda_1$ has border array $\beta_7 = 0010123$, but there is no $x_8^* = x_7^*\lambda$ on Λ_3 with border array $\beta_8 = 00101230$.

Lemmas 3.2-3.4 suggest an algorithm for generating all b-canonical strings of length n: for every integer $j = 1, 2, \ldots, n-1$, append to each b-canonical string x_j^* single standard letters $\lambda_1, \lambda_2, \ldots,$, until for some integer $\kappa \geq 2$, $x_j^*\lambda_\kappa$ has only the empty border. Then the strings $x_j^*\lambda_1, x_j^*\lambda_2, \ldots, x_j^*\lambda_\kappa$ will be exactly the b-canonical strings derived from x_j^*.

To implement this algorithm, we generate a rooted tree T_n', similar to the tree employed in Section 2. Here each node of T_n' is a pair (λ, β), where $\lambda \in \Lambda$ and β denotes the border array entry for λ in the string defined by the labels in the nodes on the path from the root of T_n' to the current node. Thus T_1' consists of the root node $(\lambda_1, 0)$, and for every integer $n \geq 2$, T_n' is formed by adding the children

$$(\lambda_1, \beta_1), (\lambda_2, \beta_2), \ldots, (\lambda_\kappa, 0)$$

to every leaf node of T_{n-1}'. Hence each node of T_n' determines a b-canonical string together with its border array. Denoting by $b[n]$ the number of b-canonical strings of length exactly n, we see that T_n' has exactly $b[n]$ leaf nodes. Thus all $b[n]$ b-canonical strings (and their corresponding border arrays) can be represented simply by appending $b[n]$ children to the leaf nodes of T_{n-1}', a task requiring $\Theta(b[n])$ time since the border array element contained in each new child can be computed in amortized constant time using the standard failure function algorithm [AHU74]. Since by Lemma 3.1 every non-leaf node of T_n' has at least two children, it follows that the number of nodes in each level of T_n' exceeds the number of nodes in all previous levels, hence that T_{n-1}' contains fewer than $b[n]$ nodes, and so can be constructed in $O(b[n])$ time. We have then the analogue to Theorem 2.2:

Theorem 3.1 For every positive integer n, all $b[n]$ b-canonical strings of length n can be computed in $\Theta(b[n])$ time and represented in $\Theta(b[n])$ space. □

We remark that trivial modification to the algorithm outlined above yields an algorithm to compute all the b-canonical strings of length n defined on Λ_k: in computing the children of each node, it is necessary only, as indicated above, to ensure that every child $(\lambda_\kappa, 0) = (\lambda_{k+1}, 0)$ is omitted from the tree. Note also that it is straightforward, using the tree T'_n, to compute b-canonical strings that are "random" in the sense that, at each step, a child x^*_j of x^*_{j-1} is pseudorandomly selected.

It is clear from Lemma 3.4 that there always exist at least two border arrays $\beta^{(0)}_{n+1} = \beta_n 0$ and $\beta^{(m+1)}_{n+1} = \beta_n(m+1)$, where $m = \beta[n]$. The next result shows how to determine whether or not there exists $\beta^{(i)}_{n+1}$, $1 \leq i \leq m$, and so provides a basis for an algorithm which, given all distinct border arrays β_n, computes all distinct border arrays β_{n+1} without any knowledge of x^*_n. Thus Theorem 3.2 establishes the interesting and nonobvious fact that distinct border arrays of length n can be computed by constructing a tree T''_n whose nodes contain border array elements only. In fact, as observed by a referee, T''_n can like T'_n be constructed in $\Theta(b[n])$ time, but only at a cost of introducing an extra pointer into each node i. Thus no storage is saved using T''_n and it turns out that the algorithm for its construction is considerably more complicated than the one given above for T'_n. The algorithm is therefore not described here in detail. In the following theorem, the notation $j' \to j$ is used to mean that $\beta^i[j'] = j$ for some $i > 0$.

Theorem 3.2 Let $m = \beta[n] \geq 1$. For every integer $i \in 1..m$, there exists a valid border array $\beta^{(i)}_{n+1} = \beta_n i$ if and only if the following conditions all hold:
(a) $\beta[m+1] \not\to i$;
(b) $\beta[m] \to i-1$;
(c) there exists no integer $i' \to i$ such that $\beta^{(i')}_{n+1} = \beta_n i'$ is valid.

Proof To prove the necessity of the three conditions, suppose first that $\beta_n i$ is a valid border array. Then there exists a b-canonical string $x^*_{n+1} = x^*_n \lambda$ with a longest border $x^*_i = x^*[1..i]$, where x^*_n has a longest border $x^*_m = x^*[1..m]$, $m \geq i$. Thus $\lambda \equiv x^*[n+1] = x^*[i]$ while $\lambda \neq x^*[m+1]$, since otherwise it would follow that x^*_{n+1} would have a longest border x^*_{m+1}. We conclude that $x^*[m+1] \neq x^*[i]$, from which (a) follows.

To prove (b), observe first that for $i = 1$, (b) is true. Suppose therefore that $i > 1$. But then the fact that $\lambda = x^*[i]$ leads to the conclusion that $x^*[n] = x^*[m] = x^*[i-1]$, hence that $\beta[m] \to i-1$.

To prove (c), suppose on the contrary that for some $i' \to i$, $\beta_n i'$ is a valid border array. But then in order to form a border x^*_i of x^*_{n+1}, a longer border $x^*_{i'}$ is necessarily formed, contradicting the assumption that $\beta_n i$ is a valid border array. Thus (c) also must be true.

To prove sufficiency, suppose that (a), (b) and (c) all hold. Since $\beta[m] \rightarrow i-1$, we may choose $\lambda = x^*[i]$ to ensure that x^*_{n+1} has a border of length at least i. Since $\beta[m+1] \not\rightarrow i$, we are assured that $x^*[m+1] \neq x^*[i]$, hence that x^*_{n+1} does not have a border of length m. Since by (c) i is a leaf node in B_{n+1}, we are further assured that x^*_{n+1} has no border longer than i. Thus $\beta^{(i)}_{n+1} = \beta_n i$ is a valid border array, as required. □

We turn now to consideration of a b' array analogous to the p' array of Section 2: for positive integers k and n, $b'[k,n]$ denotes the number of b-canonical strings of length n formed using exactly the k standard letters of Λ_k. Then the already-defined quantities $b[n]$ are the column sums in the b' array:

$$b[n] = \sum_{k \geq 1} b'[k,n].$$

As we shall see below (Theorem 3.3(a)), all terms in the n^{th} column of the b' array are zero for $k > \lceil \log_2(n+1) \rceil$; that is, the k^{th} letter of the alphabet does not appear in b-canonical strings of length $n < 2^{k-1}$. For $k \leq \log_2(n+1)$, computation of the elements $b'[k,n]$ requires generation of a tree T'''_n in which each node takes the form of a triple (λ, β, i), where as in Section 2 the additional term i counts the number of distinct letters in the b-canonical string represented by the path from the root. Using T'''_n a straightforward algorithm allows $b'[k,n]$ to be computed in $O(b[n])$ time.

In general, it appears to be much more difficult to find well-known expressions for the elements of the b' array than for those of the p' array. However, the following theorem provides enough information to allow useful upper bounds to be stated on $b'[k,n]$ and $b[n]$. It also illustrates the difficulty of expressing these values in closed form.

Theorem 3.3 Given positive integers k and n:

(a) $b'[k,n] = 0,\ k > \lceil \log_2(n+1) \rceil$.

(b) $b'[1,n] = b'[k, 2^{k-1}] = 1$.

(c) $b'[2,n] = p'[2,n] = 2^{n-1} - 1$.

(d) Let $\hat{b}[k,n]$ denote the number of strings counted by $b'[k,n]$ which contain λ_k only in position n. Then

$$\hat{b}[3,n] = 2^{\lceil n/2 \rceil - 2}(2^{\lfloor n/2 \rfloor - 1} - 1) - 2^{n-4} \sum_{j=0}^{\lfloor n/2 \rfloor - 2} \hat{b}[3, j+2]/2^{2j}$$

for every $n \geq 2$.

(e) Let $\tilde{b}[k,n] = b'[k,n] - \hat{b}[k,n]$. Then for every $k \geq 3$ and $n \geq 3$,

$$\tilde{b}[k,n] \geq 2b'[k,n-1] + b'[k,n-2],$$

with equality holding for $k = 3$ and $5 \le n \le 8$.

(f) For every nonnegative integer j,

$$b'[k, 2^{k-1} + j] \le p'[k, k + j],$$

with equality holding for $1 \le k \le 2$.

Proof (a) The proof is by induction. Observe that the result holds for $n = 1$. We suppose then that it holds for every n satisfying $2^{k-1} \le n \le 2^k - 1$ for some positive integer k, and we show that therefore it must hold for values n' satisfying $2^k \le n' \le 2^{k+1} - 1$.

By the definition of the b' array, the inductive assumption is equivalent to supposing that over the range of values n, at most k letters $\lambda_1, \lambda_2, \ldots, \lambda_k$ (in ascending order) are required in order to form the b-canonical string x_n corresponding to every border array β_n. Thus the letter λ_{k+1} does not occur in any position less than 2^k of any b-canonical string $x^*_{n'}$, $n' \ge 2^k$.

We need to show that for every n' satisfying $2^k \le n' \le 2^{k+1} - 1$, no b-canonical string $x^*_{n'}$ contains λ_{k+2}. Suppose on the contrary that some such $x^*_{n'}$ contains λ_{k+2} as its final letter: $x^*_{n'} = x^*_{n'-1}\lambda_{k+2}$. This can occur only if each of the strings

$$\{x^*_{n'-1}\lambda_1, x^*_{n'-1}\lambda_2, \ldots, x^*_{n'-1}\lambda_{k+1}\}$$

is b-canonical and has a nonempty border. In particular, let $x^*_{n'} = x^*_{n'-1}\lambda_{k+1}$, and let j denote the position of the first occurrence of λ_{k+1} in $x^*_{n'}$. By the inductive hypothesis, $j \ge 2^k$, and so the length of the longest border of $x^*_{n'}$ must exceed $n'/2$. But this implies that $x^*_{n'}[j-(n'-j)] = \lambda_{k+1}$, contradicting the assumption that j is the first occurrence of λ_{k+1}. We conclude that $x^*_{n'-1}\lambda_{k+1}$ cannot have a nonempty border, hence by Lemma 3.4 that no b-canonical string $x^*_{n'}$ contains λ_{k+2}, as required.

(b) $b'[1, n] = 1$ corresponding to the strings λ_1^n, while $b'[k, 2^{k-1}] = 1$ corresponding to the strings

$$\{\lambda_1, \lambda_1\lambda_2, \lambda_1\lambda_2\lambda_1\lambda_3, \lambda_1\lambda_2\lambda_1\lambda_3\lambda_1\lambda_2\lambda_1\lambda_4, \ldots\}.$$

(c) Follows from the observation that for $n = 2$ every p-canonical string is also b-canonical.

(d) To improve readability we make the substitution $\{a, b, c\} \leftarrow \{\lambda_1, \lambda_2, \lambda_3\}$. Then observe that every b-canonical string $x^*_{n-1} = ab * a$ gives rise to a b-canonical string $x^*_n = x^*_{n-1}c$. (Here $ab * a$ denotes a string with prefix ab, suffix a, and zero or more "don't-care" letters in between.) There are 2^{n-4} such b-canonical strings.

For any integer $j \geq 0$, let y_j denote a substring of length j on $\{a, b\}$. Then observe further that every b-canonical string $x^*_{n-1} = ay_1b * ay_1$ gives rise to a b-canonical string $x^*_n = x^*_{n-1}c$: there are $2(2^{n-6})$ such strings.

Next consider $x^*_{n-1} = ay_2b * ay_2$ giving rise to $x^*_n = x^*_{n-1}c$. Here y_2 can take the values aa, ab and bb, but not ba, since the string $ab * a$ has already been counted. Thus in this case there are $(2^2 - 1)2^{n-8}$ new distinct b-canonical strings. Similarly for $x^*_{n-1} = ay_3b * ay_3$: here y_3 omits the values baa and bba, again since $ab*a$ has already been omitted. Thus we count $(2^4 - 2)2^{n-10}$ new distinct strings.

We see in general that corresponding to each $x^*_{n-1} = ay_jb * ay_j$, there are

$$(2^j - \hat{b}[3, j + 2])2^{n-2j-4}$$

distinct b-canonical strings which give rise to $x^*_n = x^*_{n-1}c$. Thus

$$\hat{b}[3, n] = \sum_{j=0}^{\lfloor n/2 \rfloor - 2} (2^j - \hat{b}[3, j + 2])2^{n-2j-4},$$

a sum which after simplification reduces to the form given in the statement of the theorem.

(e) Observe that the b-canonical strings counted by $\tilde{b}[k, n]$ include at least the following:

- strings $x^*_{n-1}\lambda_1$ and $x^*_{n-1}\lambda_2$, where x^*_{n-1} is a b-canonical string counted by $b'[k, n - 1]$;
- strings $x^*_{n-2}\lambda_1\lambda_3$, where x^*_{n-2} is a b-canonical string counted by $b'[k, n-2]$.

It is straightforward to verify that equality holds in the cases claimed.

(f) A consequence of (a) and the fact that every b-canonical string is also p-canonical. □

These results provide us with some capability to estimate the size of the entries in the b' array. It appears from Theorem 3.3(d) that exact computation of these entries is in general extremely complicated. Theorem 3.3(f) shows that, for every fixed $k \geq 3$, the entries $b'[k, n]$ are asymptotically less, by a factor exponential in k, than the corresponding entries $p'[k, n]$. This result can easily be applied to yield an upper bound on $b[n]$ expressed in terms of entries in the p' array: for every positive integer n,

$$b[n] \leq \sum_{k=1}^{k^*} p'[k, n - 2^k + k], \qquad \ldots(3.1)$$

where $k^* = \lceil \log_2(n+1) \rceil$. Note that by reducing the value of k^*, we can also use (3.1) to bound the partial column sums in the b' array.

We conclude by displaying some of the smaller values in the b' array:

Non-Zero Elements $b'[k,n]$, $n \leq 10$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
1	1	1	1	1	1	1	1	1	1	1	
2		1	3	7	15	31	63	127	255	511	
3				1	2	6	12	27	54	114	$(\hat{b}[3,n])$
					2	9	34	107	316	883	$(\tilde{b}[3,n])$
4								1	2	7	$(\hat{b}[4,n])$
									2	9	$(\tilde{b}[4,n])$
$b[n]$	1	2	4	9	20	47	110	263	630	1525	

Table 3.1

4 CONCLUSION

In this paper we have shown how "distinct" strings of length n formed using exactly k letters can be efficiently computed and counted, according to two definitions of distinctness. Both of these definitions lead to algorithms that are considerably more economical than the computation or counting of $\Theta(k^n)$ strings.

REFERENCES

[**AHU74**] Alfred V. Aho, John E. Hopcroft & Jeffrey D. Ullman, *The Design & Analysis of Computer Algorithms*, Addison-Wesley (1974).

[**K68**] Donald E. Knuth, *The Art of Computer Programming I — Fundamental Algorithms*, Addison-Wesley (1968).

[**L96**] Yin Li, *A Windows-Based String Algorithm Testing System*, Undergraduate Computer Science Project, Department of Computer Science & Systems, McMaster University (1996).

[**ML84**] Michael G. Main & Richard J. Lorentz, **An $O(n \log n)$ algorithm for finding all repetitions in a string**, *J. Algs. 5* (1984) 422-432.

[**MS95**] Dennis Moore & W. F. Smyth, **Correction to: An optimal algorithm to compute all the covers of a string**, *IPL 54* (1995) 101-103.

[**PBM86**] A. P. Prudnikov, Yu. A. Brychkov & O. L. Marichev, *Integrals and Series I* (transl. N. M. Queen) Gordon & Breach (1986).

[**PTW83**] George Pólya, Robert E. Tarjan & Donald R. Woods, *Notes on Introductory Combinatorics*, Birkhäuser (1983).

[**R68**] John Riordan, *Combinatorial Identities*, John Wiley (1968).
[**S73**] N. J. A. Sloane, *A Handbook of Integer Sequences*, Academic Press (1973).

ACKNOWLEDGEMENTS

The work of the second author was supported in part by Grant No. A8180 of the Natural Sciences & Engineering Research Council of Canada and by Grant No. GO-12778 of the Medical Research Council of Canada. The authors are pleased to acknowledge the programming assistance of Yin Li. They are also express their gratitude to an anonymous referee whose comments have substantially improved the quality of the paper.