Efficient mapping functions

Denis TRYSTRAM

HomeWork

Maths for Computer Science – MOSIG 1 – 2023

Guidelines

This work can be done by groups of 3 to 4 students, but each one must send a **personal report** in pdf format at

Denis.Trystram@univ-grenoble-alpes.fr

Please, indicate the names of the other members of this group and detail clearly the credit of each (percentage in designing, proving, writing code, do experiments, etc.).

I don't encourage you to look at Internet, but in case, **indicate all sources you used**.

Use your professional e-mail address and clearly indicate:

subject: **Homework MCS**

The file should be named **Homework-name.pdf**

The strict deadline is **novembre 12, 23:59**, a penalty will be applied in case of delay.

The subject contains both explanations and questions.

It is mandatory to answer Questions 1 to 6. The further questions are harder and are only for those who wish to dive deeper into the subject.

Motivation

We propose here to investigate the mathematical analysis of coding functions motivated by the "real-life" challenge of devising *efficient* computer-storage mappings for arrays and tables that can be expanded and contracted dynamically.

The first part is a warm-up for understanding a generic construction paradigm on the example of diagonal pairing function in $N^+ \times N^+$ that will be applied in further sections.

Diagonal coding: Bringing linear order to tuple spaces

The problem is to encode complex structures via *ordered pairs*, that represent a myriad of complex structures. We illustrate below some of them.

(i) *(Ordered) tuples of integers.* Focus on the set of k-tuples of integers, for any integer $k > 1$. One way to encode this set using ordered pairs is by recursion as follows:

- the base case $k = 2$ consists of the *ordered pairs* themselves.
- For any $k > 2$, encode the k-tuple $\langle a_1, a_2, \ldots, a_k\rangle$ as the ordered pair whose *first* is the ordered $(k-1)$-tuple $\langle a_1, a_2, \ldots, a_{k-1}\rangle$

$$\langle a_1, a_2, \ldots, a_k\rangle \text{ is encoded as } \langle\langle a_1, a_2, \ldots, a_{k-1}\rangle, a_k\rangle$$

(ii) *Strings of integers.* One way to encode the string of integers $a_1 a_2 \cdots a_n$ using ordered pairs is as follows.

$$a_1 a_2 \cdots a_n \text{ is encoded as } \langle a_1, \langle a_2, \langle a_3, \ldots, \langle a_{n-2}, \langle a_{n-1}, a_n\rangle\rangle \cdots\rangle\rangle\rangle$$

(iii) *Binary trees.* One can use ordered pairs to encode any binary tree by an appropriate "parenthesization" of the sequence of the tree's leaves.

Question 1. Provide such a coding for binary trees with leaves a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8.

What does it really mean to *encode* one class of entities, A (integers, strings, trees, etc.), as another class, B?

For the mathematical perspective, there exists a *bijection* $f_{A,B}$ that maps A *one-to-one onto* B. In other words, when presented with an element $a \in A$, the function $f_{A,B}$ "produces" a unique element $b \in B$. And conversely, when presented with an element $b \in B$, the function $f_{A,B}^{-1}$ "produces" a unique element $a \in A$.

The Diagonal encoding of $N^+ \times N^+$ as N^+

Ordered pairs of integers play a special role in the study of encodings of structured sets of integers. These special bijections are called *pairing functions.*

One of the most valuable by-products of encodings via pairing functions is that such encodings provide a *linear/total ordering* of the set being encoded.[1] The orderings provided by pairing functions are particularly valuable when the structured sets being encoded as integers do not have their own "intrinsic" or "natural" orderings. Included in this category are structures such as tuples, strings, and trees.

Of course some structured sets do have natural, native linear orders: consider for instance strings under lexicographic ordering. Even for such sets, we often benefit from determining alternative orderings as we design and analyze algorithms on the sets.

We now describe the first explicit mapping function, namely, the *diagonal* encoding function $\delta(x, y)$.

$$(x, y) \;=\; \binom{x+y}{2} + (1-y) \tag{1}$$

Remark. δ of course has a twin that exchanges the roles of x and y.

δ's mapping of $N^+ \times N^+$ onto N^+ is depicted in Fig. 1. The figure

y \ x	1	2	3	4	5	6	7	8	
1	1	3	6	10	$\boxed{15}$	21	28	36	⋯
2	2	5	9	$\boxed{14}$	20	27	35	44	⋯
3	4	8	$\boxed{13}$	19	26	34	43	53	⋯
4	7	$\boxed{12}$	18	25	33	42	52	63	⋯
5	$\boxed{11}$	17	24	32	41	51	62	74	⋯
6	16	23	31	40	50	61	73	86	⋯
7	22	30	39	49	60	72	85	99	⋯
8	29	38	48	59	71	84	98	113	⋯
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Figure 1: The diagonal pairing function δ. The shell $x + y = 6$ is highlighted

exposes that we can view δ's mapping of $N^+ \times N^+$ as a two-step conceptual process:

1. partitioning $N^+ \times N^+$ into "diagonal shells".

   For each index $k \in N^+$, shell $\#k$ is the set

$$Shell_k \;=\; \{\langle x, y\rangle \mid x + y = k\}$$

[1] order within a number system is among one's biggest friends when reasoning about the numbers within the system.

The partitioning is an integral part of the specification of δ, as witnessed by the following subexpression below.

$$\frac{1}{2}(x+y)\cdot(x+y) \;=\; \binom{x+y}{2}$$

2. "climbing up" these shells in order

Shell-Based Methodology

The shell-oriented strategy that underlies the diagonal mapping function δ can be adapted to incorporate shell-shapes that are inspired by a variety of computational situations – and can be applied to computational advantage in such situations.

Procedure Constructor(α)
/*Construct a shape-inspired pairing function α*/
begin

Step 1. Partition the set $N^+ \times N^+$ into finite sets called *shells*. Order these shells linearly in some way.

Step 2. Construct a pairing function from the shells as follows.

Step 2a. Enumerate $N^+ \times N^+$ shell by shell according to the ordering of the shells; i.e., list the pairs in shell #1, shell #2, shell #3, etc.

Step 2b. Enumerate each shell systematically.

end Constructor

Question 2. Provide a way to enumerate the pairs $\langle x, y\rangle$.

Question 3. Show that any function $N^+ \times N^+ \leftrightarrow N^+$ that is designed via Constructor is a bijection.

We now use Constructor to design two other functions which produce efficient storage mappings for extendible arrays and tables.

Question 4. Briefly analyze Stern's sequence as a diagonal mapping.

The Square-Shell function σ

One computational situation where pairing functions are useful is as storage-mappings for arrays/tables that can expand and/or contract dynamically.

In conventional programming systems, when one expands an $n \times n$ table into an $(n+1) \times (n+1)$ table, one allocates a new region of $(n+1)^2$ storage locations and copies the current table from its n^2-location region to the new region. Of course, this is very wasteful: one is moving $\Omega(n^2)$ items to make room for the anticipated $2n+1$ new items. On any given day, the practical impact of this waste depends on current technology.

Considering the mathematics perspective and not an engineering one, let us explore whether *in principle* we can avoid the waste.

If we employ a function $\varepsilon : N^+ \times N^+ \leftrightarrow N^+$ to allocate storage for tables, then to expand a table from dimensions $n \times n$ to $(n+1) \times (n+1)$, we need move only $O(n)$ items to accommodate the *new* table entries; the rest of the current entries need not be moved.

For square tables, the following *square-shell* function σ manages the described scenario perfectly.

y \ x	1	2	3	4	5	6	7	8	
1	1	2	5	10	$\boxed{17}$	26	37	50	$\cdots$
2	4	3	6	11	$\boxed{18}$	27	38	51	$\cdots$
3	9	8	7	12	$\boxed{19}$	28	39	52	$\cdots$
4	16	15	14	13	$\boxed{20}$	29	40	53	$\cdots$
5	$\boxed{25}$	$\boxed{24}$	$\boxed{23}$	$\boxed{22}$	$\boxed{21}$	30	41	54	$\cdots$
6	36	35	34	33	32	31	42	55	$\cdots$
7	49	48	47	46	45	44	43	56	$\cdots$
8	64	63	62	61	60	59	58	57	$\cdots$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Figure 2: *The square-shell* σ. *The shell* $\max(x, y) = 5$ *is highlighted*

$$\begin{array}{rcl} \sigma(x,y) & = & m^2 + m + y - x + 1 \\ & & \text{where} \quad m \;=\; \max(x-1, y-1). \end{array} \qquad (2)$$

Of course, σ has a twin that enumerates the shells in a counterclockwise direction...

Question 5. Verify the mapping σ and show that it follows the prescription of Constructor

Question 6. Show how to adapt σ to accommodate, with no wastage, arrays/tables of any fixed aspect ratio $an \times bn$ $(a, b \in N)$.

The Hyperbolic-Shell *hyp*

The previous diagonal and square-shell functions indicate that when the growth patterns of one's arrays/tables is very constrained, one can use pairing functions as storage mappings with very little wastage. In contrast, if one employs a pairing function such as δ without considering its wastage, then a storage map would show some $O(n)$-entry tables being “spread” over $\Omega(n^2)$ storage locations. In the worst-case, δ spreads the n-position $1 \times n$ array/table over more than $\frac{1}{2}n^2$ addresses, because: $(1,1) = 1$ and $(1,n) = \frac{1}{2}(n^2+n)$. This degree of wastefulness can be avoided via careful analysis, coupled with the use of Constructor. The target commodity to be minimized is the *spread* of a Pairing Function-based storage map, which we define as follows. Let denote PF in short for Pairing Function.

Note that an ordered pair of integers $\langle x, y\rangle$ appears as a position-index within an n-position table if, and only if, $xy \leq n$. Therefore, we define the spread of a PF-based storage map μ via the function

$$\mathbf{S}(n) \;=\; \max\{\mu(x,y) \mid xy \leq n\} \tag{3}$$

$\mathbf{S}(n)$ is the largest “address” that μ assigns to any position of a table that has n or fewer positions.

Question 7. Show that the following mapping *hyp* in Fig. 3 (within constant factors) has minimum worst-case spread

Let $d(k)$ be the number of divisors of the integer k
$hyp(x,y) = \sum_{k=1}^{xy-1} d(k) +$ the position of $\langle x, y\rangle$ among two-part factorizations of xy, in reverse lexicographic order.

y \ x	1	2	3	4	5	6	7	...
1	1	3	5	8	10	$\boxed{14}$	16	...
2	2	7	$\boxed{13}$	19	26	34	40	...
3	4	$\boxed{12}$	22	33	44	56	69	...
4	6	18	32	48	64	81	99	...
5	9	25	43	63	86	108	130	...
6	$\boxed{11}$	31	55	80	107	136	165	...
7	15	39	68	98	129	164	200	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Figure 3: *The hyperbolic function hyp where shell $xy = 6$ is highlighted*

Question 8. Show that the hyperbolic function *hyp* is a pairing function.

Question 9. The spread of *hyp* is $\mathbf{S}(n) \;=\; O(n \log n)$.

Question 10. No pairing function has better compactness than *hyp* by more than a constant factor.