

## AN EFFECTIVE METHOD FOR CONSTRUCTING DATA STRUCTURES SOLVING AN ARRAY MAINTENANCE PROBLEM

Adriana Toni, Angel Herranz, Juan Castellanos

**Abstract:** In this paper a constructive method of data structures solving an array maintenance problem is offered. These data structures are defined in terms of a family of digraphs which have previously been defined, representing solutions for this problem. We present as well a prototype of the method in Haskell

**Keywords:** array maintenance, average complexity, data structures, models of computation

### Introduction

The Range Query Problem of size  $N$  (N-RQP) deals with the analysis and design of data structures for the implementation of the operations Update and Retrieve: let  $A$  be an array of length  $N$  of elements of a commutative semigroup, Update( $i$ ,  $x$ ) increments  $A(i)$  ( $i$ -th element of  $A$ ) in  $x$  and Retrieve( $i$ ,  $j$ ) outputs the partial sum  $A(i)+\dots+A(j)$ .

In [4] we find the following definition of N-RQP design.

#### Definition N-RQP design

A N-RQP design is a triple  $(Z, U, R)$  where  $Z$  is an array of length  $M$  with  $N$  less or equal  $M$ ,  $U$  is a family of subsets of  $1..M$  indexed on  $1..N$  and  $R$  is a family of subsets of  $1..M$  indexed on  $1..N \times 1..N$ . Given a N-RQP design  $(Z, U, R)$ , the implementation of the operations Update and Retrieve is:

<pre> <b>procedure</b> Update   (<i>i</i>: 1..N, <i>x</i>:S) <b>is</b> <b>begin</b>   <b>for</b> <i>k</i> <b>in</b> <math>U_i</math> <b>loop</b>     <math>Z(k) := Z(k) + x;</math>   <b>end loop;</b> <b>end</b> Update; </pre>	<pre> <b>function</b> Retrieve   (<i>i</i>: 1..N, <i>j</i>: 1..N)   <b>return</b> S <b>is</b>   <b>begin</b>   <b>return</b> <math>\sum_{k \in R_{ij}} Z(k)</math>   <b>end</b> Retrieve; </pre>
--	--

It is a well known result that an N-RQP design  $(Z, U, R)$  is a N-RQP solution if and only if

$$\forall i, j, k \in 1..N \bullet |R_{ij} \cap U_k| = \begin{cases} 1 & \text{if } i \leq k \wedge k \leq j \\ 0 & \text{otherwise} \end{cases}$$

and a proof can be found in [1].

In [4] we find the three definitions below as well.

#### Definition N-RQP graph

An acyclic digraph  $G = (V, E)$  is a N-RQP graph if the following conditions hold:

- (1)  $V = 1..M$  with  $N \leq M$ .
- (2) For every vertex  $v \leq N$ , its out-degree is 0.
- (3) For every vertex  $v > N$ ,  $Successors(G, v) \cap 1..N \neq \emptyset$ .

#### Definition N-RQP design in terms of G

Given a N-RQP graph  $G = (V, E)$ , the N-RQP design  $(Z, U, R)$  is a N-RQP design in terms of  $G$  if it verifies the following properties:

- (1)  $|Z| = |V|$
- (2)  $U_i = Ancestors^*(G, i)$
- (3)  $R_{ij}$  is the set of vertices with the smallest cardinal that verifies:

$$\bigcup_{u \in R_{ij}} Successors^*(G, u) \cap 1..N = i..j$$

$$\bigcup_{u \in R_{ij}} \text{Successors}^*(G, u) \cap 1..N = 0$$

being

$$\text{Successors}^*(G, u) = \{u\} \cup \text{Successors}(G, u)$$

$$\text{Ancestors}^*(G, v) = \{v\} \cup \text{Ancestors}(G, v)$$

and in the same paper it has been proved that given a N-RQP graph, a N-RQP design in terms of G is a N-RQP solution.

### Definition 2<sup>K</sup> -RQP graph

Let K be a natural number. A 2<sup>K</sup> -RQP graph G<sup>K</sup> is defined inductively:

$$(1) \text{ If } K = 0 \text{ then } G^K = (\{(1,1)\}, 0)$$

$$(2) \text{ If } K > 0 \text{ then } G^K = \text{Duplicate}(G^{K-1})$$

where function Duplicate is defined as

```

function Duplicate (GK = (VK, EK) : Digraph) return Digraph is
  N : constant ℕ := 2K
  M : constant ℕ := |VK|
  V : {(i, j) ∈ 1..N × 1..N • i ≤ j} := 0;
  E : P(V×V) := 0;
  i, j : 1..(2N);
begin
  -- The ``cloning`` loops
  for (i, j) in VK loop
    V := V ∪ {(i, j), (i + N, j + N)};
  end loop;
  for (i, j) → (i', j') in EK loop
    E := E ∪ {(i, j) → (i', j'), (i + N, j + N) → (i' + N, j' + N)};
  end loop;
  -- (V,E) is a graph with two subgraphs which are just like G
  -- but with different node numbering
  for i in 1..(N-1) loop -- The ``left half`` loop
    j := i + 1;
    while (i, N) ∉ V ∧ j ≤ N loop
      if (i, j) ∈ V ∧ (j, N) ∈ V then
        V := V ∪ {(i, N)};
        E := E ∪ {(i, N) → (i, j), (i, N) → (j, N)};
      else
        j := j + 1;
      end if;
    end loop;
  end loop;
  for j in (N + 2)..(2N) loop -- The ``left half`` loop
    i := j - 1;
    while (N + 1, j) ∉ V ∧ i ≤ 2N loop
      if (N, i) ∈ V ∧ (i, j) ∈ V then
        V := V ∪ {(N, j)};
        E := E ∪ {(N, j) → (N, i), (N, j) → (i, j)};
      else
        i := i + 1;
      end if;
    end loop;
  end loop;
  return (V, E);
end Duplicate;

```

Obviously, the  $2^k$ -RQP design  $(Z,U,R)$  can be computed after the construction of the  $2^k$ -RQP graph as described by the following brute force algorithm:

**Algorithm 1** The following algorithm computes  $R_{ij}$  for a  $N$ -RQP design in terms of a  $N$ -RQP graph  $G=(V,E)$ :

```

R : P(1..|V|) := 1..N;
R' : P(1..|V|);
begin
  for R' in P(1..|V|) loop
    if |R'| ≤ |R|
      ∧  $\bigcup_{u \in R_{ij}} \text{Successors}^*(G,u) \cap 1..N = i..j$ 
      ∧  $\bigcup_{u \in R_{ij}} \text{Successors}^*(G,u) \cap 1..N = 0$  then
        R := R'
      end if;
    end loop;
  return R;
end;
```

The algorithm is correct for any  $N$ -RQP graph but in the case of  $2^k$ -RQP graphs a refinement can be applied by filtering those  $R'$  with a cardinal greater than 2 reducing the complexity drastically. Nevertheless, the user is just interested in the design and not in the graph so a direct constructive method that computes  $|Z|$ ,  $U$  and  $R$  would be welcome. In this section a method for calculating  $2^k$ -RQP designs is given..

As in the previous section,  $Z$  can be treated as a two dimensional array (where the variable  $Z(i, j)$  does not necessarily exist for all  $(i, j)$ ) that is isomorphic to a one dimensional array  $Z'$  and where the isomorphism is given by an injective partial map such that  $(i,j) \rightarrow i$  when  $i=j$ .

The method presented in the following definition is the result of a deep analysis of the properties of  $2^k$ -RQP graphs.

### Definition 1

Let be the  $2N$ -RQP with  $N = 2^K$  and  $K \in \mathbf{N}$  A  $2^{K+1}$ -RQP design  $(Z, U, R)$  is constructed in the following way:

$R_{i,j}$  ( $i \in 1..2N, j \in 1..2N$ ) is defined by the following cases:

- **A1.** If  $i = j$ ,

$$R_{i,j} = \{(i, j)\} \quad (4)$$

- **A2.** For every  $l \in 1..K$ ,
  - If  $i \in 1..2^{l-1}$ ,

$$R_{i,2^l} = \{(i, 2^l)\} \quad (5)$$

and for every  $c \in 1..(2^{K-l} - 1)$  and  $d = c2^{l+1}$ ,

$$R_{i+d, 2^l+d} = \{(i + d, 2^l + d)\} \quad (6)$$

- For every  $r \in 1..(l-2)$ ,
  - If  $i \in (2^l - 2^{l-r} + 2)..(2^l - 2^{l-r-1})$ ,

$$R_{i, 2^l} = \{(i, 2^l)\} \quad (7)$$

and for every  $c \in 1..(2^{K-l} - 1)$  and  $d = c2^{l+1}$ ,

$$R_{i+d \ 2^l+d} = \{(i + d, 2^l + d)\} \quad (8)$$

- **A3.** For every  $l \in 1..K$ ,
- If  $j \in (2^{l-1}3 + 1)..2^{l+1}$ ,

$$R_{2^{l+1} \ j} = \{(2^l + 1, j)\} \quad (9)$$

and for every  $c \in 1..(2^{K-l} - 1)$  and  $d = c2^{l+1}$ ,

$$R_{2^{l+1}+d \ j+d} = \{(2^l + 1 + d, j + d)\} \quad (10)$$

- For every  $r \in 1..(l - 2)$ ,
- If  $j \in (2^l + 1 + \frac{2^l}{2^{r+1}})..(2^l - 1 + \frac{2^l}{2^r})$ ,

$$R_{2^{l+1} \ j} = \{(2^l + 1, j)\} \quad (11)$$

and for every  $c \in 1..(2^{K-l} - 1)$  and  $d = c2^{l+1}$ ,

$$R_{2^{l+1}+d \ j+d} = \{(2^l + 1 + d, j + d)\} \quad (12)$$

- **B1.** If  $i \in 1..N$  and  $j \in N + 1..2N$ ,

$$R_{i \ j} = \{(i, N), (N + 1, j)\} \quad (13)$$

- **B2.** For every  $l \in 1..(K - 1)$ ,
- If  $i \in 1..2^l$  and  $j \in (2^l + 1)..(2^{l+1} - 1)$ ,

$$R_{i \ j} = \{(i, 2^l), (2^l + 1, j)\} \quad (14)$$

$$R_{2N-j+1 \ 2N-i+1} = \{(2N - j + 1, 2N - 2^l), (2N - 2^l + 1, 2N - i + 1)\} \quad (15)$$

- If  $i \in 2..2^l$  and  $j \in (2^l + 1)..(2^{l+1} - 2)$ , and for every  $c \in 1..(2^{K-l} - 1)$  and  $d = c2^{l+1}$ ,

$$R_{i+d \ j+d} = \{(i + d, 2^l + d), (2^l + d + 1, j + d)\} \quad (16)$$

$U_k$  ( $k \in 1..2N$ ) is defined by the following comprehension set:

$$U_k = \{(i, j) \bullet i \in 1..2N \wedge j \in i..2N \wedge i \leq k \wedge k \leq j \wedge |R_{ij}| = 1\}$$

$|Z|$ , the number of variables  $Z(i, j)$  of the design is the number of  $R_{ij}$  of size 1:

$$|Z| = \sum_{|R_{ij}|=1} 1$$

---

## Implementation in Haskell

---

The following Haskell [7] program implements the constructive method given in Definition 1.

This prototype implementation has been tested for  $N=2^k$  being  $K$  less or equal 25.

Given an integer  $K$ , most functions compute information of the solutions of the  $2^{K+1}$ -RQP:  $|Z|$ ,  $U_i$  and  $R_{ij}$ .

```
pow2 :: Integer -> Integer
pow2 0 = 1
pow2 n = 2 * (pow2 (n-1))

a1 :: Integer -> [(Integer, Integer)]
a1 k = [(i, i) | i <- [1..(pow2 (k+1))]]
```

```

a2 :: Integer -> [(Integer,Integer)]
a2 k = [(i, pow2 l)]
      | l <- [1 .. k],
        i <- [1 .. pow2 (l-1)]]
++
[(i+d, pow2 l + d)]
      | l <- [1 .. k],
        i <- [1 .. pow2 (l-1)],
        c <- [1 .. pow2 (k-1) - 1],
        let d = c * pow2 (l+1)]
++
[(i, pow2 l)]
      | l <- [1 .. k],
        r <- [1 .. l-2],
        i <- [pow2 l - pow2 (l-r) + 2 .. pow2 l - pow2 (l-r-1)]]
++
[(i+d, pow2 l + d)]
      | l <- [1 .. k],
        r <- [1 .. l-2],
        i <- [pow2 l - pow2 (l-r) + 2 .. pow2 l - pow2 (l-r-1)],
        c <- [1 .. pow2 (k-1) - 1],
        let d = c * pow2 (l+1)]

a3 :: Integer -> [(Integer,Integer)]
a3 k = [(pow2 l + 1, j)]
      | l <- [1 .. k],
        j <- [3 * pow2 (l-1) + 1 .. pow2 (l+1)]]
++
[(pow2 l + 1 + d, j + d)]
      | l <- [1 .. k],
        j <- [3 * pow2 (l-1) + 1 .. pow2 (l+1)],
        c <- [1 .. pow2 (k-1) - 1],
        let d = c * pow2 (l+1)]
++
[(pow2 l + 1, j)]
      | l <- [1 .. k],
        r <- [1 .. l-2],
        j <- [pow2 l + 1 + pow2 l `div` (pow2 (r+1))
              ..pow2 l - 1 + pow2 l `div` pow2 r]]
++
[(pow2 l + 1 + d, j +d)]
      | l <- [1 .. k],
        r <- [1 .. l-2],
        j <- [pow2 l + 1 + pow2 l `div` (pow2 (r+1))
              ..pow2 l - 1 + pow2 l `div` pow2 r],
        c <- [1 .. pow2 (k-1) - 1],
        let d = c * pow2 (l+1)]

r1 :: Integer -> [(Integer,Integer)]
r1 k = a1 k ++ a2 k ++ a3 k

b1 :: Integer -> [(Integer,Integer)]
b1 k = [(i,pow2 k),(pow2 k + 1,j)]
      | i <- [1 .. pow2 k],
        j <- [pow2 k + 1 .. pow2 (k+1)]]

b2 :: Integer -> [(Integer,Integer)]
b2 k = [(i,pow2 l),(pow2 l + 1,j)]
      | l <- [1..k-1],
        i <- [1..pow2 l],
        j <- [pow2 l + 1..pow2 (l+1) - 1]]
++
[(pow2 (k+1) - j + 1, pow2 (k+1) - pow2 l),
 (pow2 (k+1) - pow2 l + 1,pow2 (k+1) - i + 1)]
      | l <- [1..k-1],
        i <- [1..pow2 l],
        j <- [pow2 l + 1..pow2 (l+1) - 1]]
++
[(i+d,pow2 l + d),(pow2 l + d + 1,j+d)]

```

```

| l <- [1..k-1],
  i <- [2..pow2 l],
  j <- [pow2 l + 1..pow2 (l+1) - 1],
  c <- [1..pow2 (k-1) - 2],
  let d = c * pow2 (l+1)]

r2 :: Integer -> [(Integer, Integer)]
r2 k = b1 k ++ b2 k

r :: Integer -> [(Integer, Integer)]
r k = r1 k ++ r2 k

u :: Integer -> [(Integer, Integer)]
u k = [(i,j) | i <- [1 .. pow2 (k+1)],
              j <- [i .. pow2 (k+1)],
              i <= k, k <= j,
              (i,j) `elem` concat (a1 k ++ a2 k ++ a3 k)]

zCard :: Integer -> Integer
zCard k = fromIntegral (length (r1 k))

```

We can prove that given a N-RQP solution (Z,U,R) obtained by applying the method in Definition 1, we have:

1. The number of program variables required is

$$|Z| = N \log_2 N - 2N + 2 \log_2 N + 2$$

2. The sum of costs of all update operations is

$$\frac{N^2}{2} - \frac{N}{2} \log_2 N + \frac{3N}{2} - 2$$

3. The sum of costs of all retrieve operations is

$$N^2 + N(3 - \log_2 N) - 2 \log_2 N - 2$$

4. The average complexity of the Update and Retrieve operations is constant (this is a consequence of 2 and 3 above)

---

## Bibliography

- [1] D.J. Volper, M.L. Fredman, *Query Time Versus Redundancy Trade-offs for Range Queries*, Journal of Computer and System Sciences 23, (1981) pp.355--365.
- [2] W.A. Burkhard, M.L. Fredman, D.J.Kleitman, *Inherent complexity trade-offs for range query problems*, Theoretical Computer science, North Holland Publishing Company 16, (1981) pp.279--290.
- [3] M.L. Fredman, *The Complexity of Maintaining an Array and Computing its Partial Sums*, J.ACM, Vol.29, No.1 (1982) pp.250--260.
- [4] A. Herranz, A. Toni, *Digraphs Definition for an Array Maintenance Problem*, Preprint.
- [5] A. Toni, *Lower Bounds on Zero-one Matrices*, Linear Algebra and its Applications, 376 (2004) 275--282.
- [6] A. Toni, *Complejidad y Estructuras de Datos para el problema de los rangos variables*, Doctoral Thesis, Facultad de Informática, Universidad Politécnica de Madrid, 2003.
- [7] S. P. Jones, J. Hughes, *Report on the Programming Language Haskell 98. A Non-strict Purely Functional Language*, (February 1999).

---

## Authors' Information

**Adriana Toni** – Grupo de Validacion y Aplicaciones Industriales, Facultad de Informática, Universidad Politécnica de Madrid; 28660-Boadilla del Monte, Madrid, SPAIN; e-mail: [atoni@fi.upm.es](mailto:atoni@fi.upm.es)

**Ángel Herranz Nieva** – Assistant Professor; Departamento de Lenguajes y Sistemas Informáticos; Facultad de Informática; Universidad Politécnica de Madrid; e-mail: [aherranz@fi.upm.es](mailto:aherranz@fi.upm.es)

**Juan Castellanos** – Departamento de Inteligencia Artificial, Facultad de Informática – Universidad Politécnica de Madrid (Campus de Montegancedo) – 28660 Boadilla de Monte – Madrid – Spain; e-mail: [jcastellanos@fi.upm.es](mailto:jcastellanos@fi.upm.es)