DIGRAPHS DEFINITION FOR AN ARRAY MAINTENANCE PROBLEM

Angel Herranz, Adriana Toni

Abstract: *In this paper we present a data structure which improves the average complexity of the operations of updating and a certain type of retrieving information on an array. The data structure is devised from a particular family of digraphs verifying conditions so that they represent solutions for this problem.*

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

Introduction

Let A be an array of fixed length N with elements belonging to a commutative semigroup an let us consider two operations, Update and Retrieve, with the following intended effect:

- Update(i, x) increments the i-th element of A in x (A(i):= A(i)+ x;).
- Retrieve(i, j) outputs A(i)+ A(i + 1) +... + A(j).

The less space consuming and, likely, the most natural data structure for implementing both operations is the array itself (from now on, expression i..j denotes the set $\{k \in N \bullet i \leq k \wedge k \leq j\}$):

Example 1

```
Update(i,x):                          Retrieve(i,j):
begin                                 begin
  A(i) := A(i) + x;                     return $\sum_{k \in i..j} A(i)$
end;                                  end;
```

Running in a random access memory machine, the complexity of Update(i, x) is constant whilst, in the worst case, the complexity of Retrieve(i, j) is linear on N. To improve the complexity of Retrieve the data structure can be reified as an array S of length N + 1 with the property $S(i) = \sum_{k \in 1..N} A(i)$. Then programs must be adapted:

Example 2

```
Update(i,x):                          Retrieve(i,j):
for k in i..N loop                    begin
  S(k) := S(k) + x;                     return S(i)-S(j+1);
end loop;                             end;
```

For this implementation the complexity of Retrieve is constant whereas, in the worst case, the complexity of Update is linear on N. The design in Example 2 assumes the existence of - (the inverse of +) in the model. This consideration aside, under any execution sequence of operations Update and Retrieve, both implementations are indistinguishable from a functional point of view. This means that Example 1 and Example 2 are different solutions to the same problem definition.

In this paper we are interested in designs with a good average complexity of Update and Retrieve operations when the program variables store elements of a commutative semigroup. Obviously, programs must yield the correct result irrespective of the particular semigroup. Uniform probability of Update and Retrieve execution in programs is assumed in order to improve what we have called average complexity, in other words, we are trying to minimise the sumof the costs of all executions.

In the next Section the RQP (Range Query Problem) and its solutions in terms of digraphs are formalised. Then a particular family of digraphs that represent solutions to the RQP will be presented in an informal way.

Range query Problem

In this section, the range query problem and its solutions are formalized. In this formalization, arrays store elements that belong to a commutative semigroup S. Let us start with the definition of arrays used in this paper.

Definition 1. An array A of length N is a total function from 1..N into S.

Criterion 2. Let A be an array of length N interpreted as a function from 1..N into S: A. 1..N. S. |A| denotes N, dom A denotes 1..N and ran A denotes S.

Definition 3. The Range Query Problem of size N (N-RQP) is the analysis and design of data structures for the implementation of the operations Update and Retrieve where both operations are interpreted as higher order functions:

$$Update : (1..N \to \mathbf{S}) \times 1..N \times \mathbf{S} \to (1..N \to \mathbf{S})$$

$$Update(A, i, x)(j) = \begin{cases} A(j) + x \ \textbf{\textit{if}}\ i = j \\ A(j) \qquad \textbf{\textit{otherwise}} \end{cases}$$

$$Retrieve : (1..N \to \mathbf{S}) \times 1..N \times 1..N \to \mathbf{S}$$

$$Retrieve(A, i, j) = \sum_{k \in i..j} A(k)$$

Definition 4. 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 × 1..N. Given a N-RQP design (Z, U, R), the implementation of the operations Update and Retrieve is:

```
procedure $Update$
   ($i$ : $1..N$, $x$ : $\mathbf{S}$) is
begin
   for $k$ in $U_i$ loop
      $Z(k)$ := $Z(k) + x$;
   end loop;
end $Update$;
```

```
function $Retrieve$
   ($i$ : $1..N$, $j$ : $1..N$)
   return $\mathbf{S}$ is
begin
   return $\sum_{k \in R_{ij}} Z(i)$
end $Retrieve$;
```

Lemma 5. The complexity of the implementation of Update(i, x) and Retrieve(i, j) in Definition 4 is linear on the cardinal of Ui and Rij, respectively, when running on a random access memory machine.

PROOF. Trivial

Definition 6. A N-RQP design (Z, U, R) is a N-RQP solution if and only if for every $i, j, k \in N$ and $x, y \in S$ the following triplets in the programming logic (annotated programs) are totally correct:

```
--  $i \le k \ \wedge \ k \le j$
--  $\wedge \ Retrieve(i,j) = x$
$Update(k,y)$;
--  $Retrieve(i,j) = x + y$
```

```
--  $(k < i \ \vee \ k > j)$
--  $\wedge \ Retrieve(i,j) = x$
$Update(k,y)$;
--  $Retrieve(i,j) = x$
```

Lemma 7. A N-RQP design (Z, U, R) is a N-RQP solution if and only if

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

PROOF. This is a well known result and a proof can be found in [1].

Average Complexity in RQP Solutions

As we mentioned in previous sections, we will try to minimize the sum of the costs of all different executions of Update and Retrieve. A uniform probability distribution for each Update possible execution (N operations) and Retrieve possible execution ($\left(\frac{N+1}{2}\right)$ operations) is assumed.

Definition 8. The average complexity of a N-RQP design (Z, U, R) is

$$\frac{\sum_{i=1,j=i}^{N,N} |R_{ij}| + \sum_{i=1}^{N} |U_i|}{N + \binom{N+1}{2}}$$

Minimizing function ϕ below, is enough to minimize the average complexity function above.

$$\phi(N) = \sum_{i=1,j=i}^{N,N} |R_{ij}| + \sum_{i=1}^{N} |U_i|$$

RQP Solutions as Graphs

N-RQP designs can be described in terms of graphs where the content of Rij and Ui are represented as graph vertices and edges (Definition 14). Let us start with some basic definitions.

Definition 9. A digraph, or directed graph, G is a pair (V, E), where V is a finite set (vertex set) and E is a binary relation on V (edge set).

Criterion 10. Notation $u \to v$, instead of (u, v), is used to denote the edges

Definition 11. Let G =(V, E) be a digraph, the out-degree of a vertex u is $|\{v \in V \mid u \to v \in E\}|$ and the in-degree of a vertex v is $|\{u \in V \mid u \to v \in E\}|$.

Definition 12. If there is a path from v1 to v2 in a digraph G =(V, E) we say that v2 is reachable from v1. Functions Successors and Ancestors are defined as:

$$Successors(G,u) = \{v \in V \mid v \text{ is reachable from } u\}$$
$$Ancestors(G,v) = \{u \in V \mid v \text{ is reachable from } u\}$$
$$Successors^*(G,u) = \{u\} \cup Successors(G,u)$$
$$Ancestors^*(G,v) = \{v\} \cup Ancestors(G,v)$$

Definition 13. 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 14. 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:*

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

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

The existence of Rij is guaranteed for every i, j such that $1 \leq i \leq j \leq N$ because in the absence of a set Rij with a cardinal less than j. i +1 we would end up with Rij = i..j. With respect to the uniqueness of Rij several sets could exist with a smallest cardinal verifying the conditions in Definition 16 so an arbitrary criterion should be given (lexicographic order, for instance).

The following theorem states that N-RQP graphs represent N-RQP solutions:

Theorem 15. Let G =(V, E) be a N-RQP graph, a N-RQP design in terms of G is a N-RQP solution.

PROOF. Let us consider the execution of an arbitrary program:

r := Retrieve(i,j);
Update(k,x);
r' := Retrieve(i,j);

with $i, k \in 1..N$ and $j \in i..2N$. We have to prove that if $i \le k \le j$ then $r' = r + x$; otherwise $r' = r$. The proof is based on the following obvious fact: $r' = r + |U_k \cap R_{ij}|\, x$ (observe that *Retrieve*(i, j) is $Z(u_1)+ \dots +$ $Z(u_n)$ where $R_{ij} = \{u_1,\dots,u_n\}$).

- Case $k < i \vee j < k$: in this case $U_k \cap R_{ij} = \varnothing$ therefore $r' = r$.
- Case $i \le k \le j$: in this case $|U_k \cap R_{ij}| = 1$ therefore $r' = r + x$.

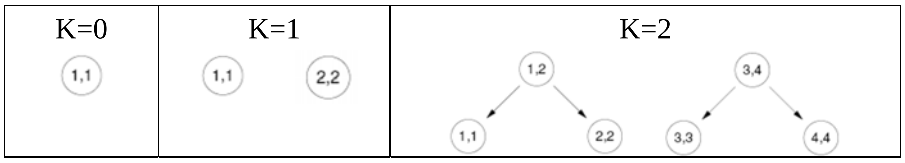

Fig. 1. 2^K-RQP graphs for $K \in 0..2$

Constructing RQP Solutions

The inspiration of our approach comes from a particular family of N-RQP graphs where N is a power of 2. In the solution designs in terms of graphs of this kind, the cost of Retrieve operations is less or equal to 2.

Graphs of this family are called 2^k- RQP graphs, with $K \in N$. The construction method is described by induction on K and Figure 1 presents the trivial examples for K=0,1,2.

The reader will observe that, strictly speaking, graphs presented in this section are not N-RQP graphs because their vertices are not positive integers but pairs (i, j) of positive integers where $(i \le j \wedge j \le N)$. This is not an important problem, as pairs can be trivially encoded as positive integers 1 and an isomorphic N-RQP graph would be obtained. Authors pursue elegance in the presentation so vertices as pairs (i, j) are maintained.

The main characteristic of the construction of N-RQP solutions is that our graphs have the following property:

$$1 \le |R_{ij}| \wedge |R_{ij}| \le 2$$

Intuitively, the 2^{K+1}-RQP graph can be built by cloning twice the 2^K-RQP graph and then adding new vertices and edges that maintain the above mentioned property. To achieve this aim, after cloning the 2^K-RQP graph, new vertices and edges will be added taking into account that the property on Rij holds if j ≤ 2^K or i> 2^K.

Let us show an example; the 8-RQP graph in Figure 2 is the result of cloning the 2^2-RQP graph in Figure 1. In the 8-RQP design (Z, U, R) in terms of that graph, the values of all Ui and those Rij such that |Rij| > 2 are:

$U_1 = \{(1, 1), (1, 2)\}$	$U_2 = \{(2, 2), (1, 2)\}$	$U_3 = \{(3, 3), (3, 4)\}$	$U_4 = \{(4, 4), (3, 4)\}$
$U_5 = \{(5, 5), (5, 6)\}$	$U_6 = \{(6, 6), (5, 6)\}$	$U_7 = \{(7, 7), (7, 8)\}$	$U_8 = \{(8, 8), (7, 8)\}$

$R_{15} = \{(1, 2), (3, 4), (5, 5)\}$	$R_{16} = \{(1, 2), (3, 4), (5, 6)\}$
$R_{17} = \{(1, 2), (3, 4), (5, 6), (7, 7)\}$	$R_{18} = \{(1, 2), (3, 4), (5, 6), (7, 8)\}$
$R_{25} = \{(2, 2), (3, 4), (5, 5)\}$	$R_{26} = \{(2, 2), (3, 4), (5, 6)\}$
$R_{27} = \{(2, 2), (3, 4), (5, 6), (7, 7)\}$	$R_{28} = \{(2, 2), (3, 4), (5, 6), (7, 8)\}$
$R_{37} = \{(3, 4), (5, 6), (7, 7)\}$	$R_{38} = \{(3, 4), (5, 6), (7, 8)\}$

$R_{47} = \{(4, 4), (5, 6), (7, 7)\}$ $R_{48} = \{(4, 4), (5, 6), (7, 8)\}$

The idea is that new vertices and edges have to be added in order to decrease the cardinal of R_{i4} and R_{5j} to 1. R_{ij} is then obtained as the union of R_{i4} and R_{5j} with a resulting cardinal of 2. In the example $|R_{14}|=|R_{24}|=2$ so a pair of vertices are added representing $R_{14}=R_{12}\cup R_{34}$ and $R_{24}=R_{22}\cup R_{34}$. Reasoning symmetrically with R_{57} and R_{58} we get the 2^3-RQP graph in Figure 3. The application of the idea is shown in the left half of the 16-RQP graph (after clowning the 2^3-RQP) in Figure 4.

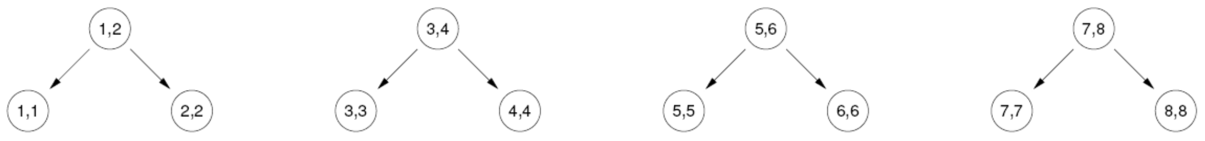

Fig. 2. A 8-RQP graph resulting after cloning the 2^2-RQP graph

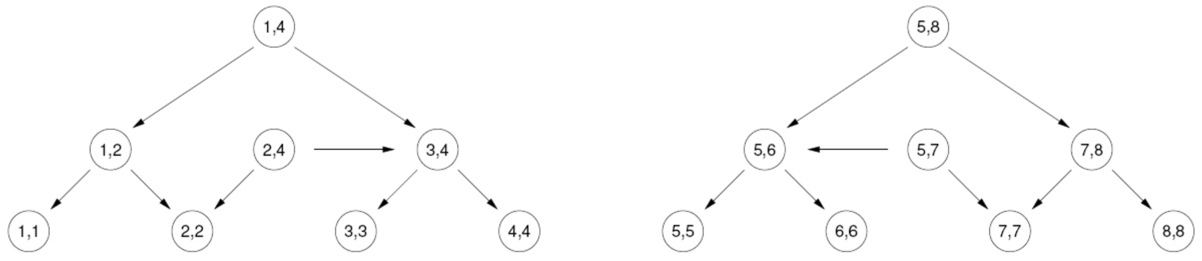

Fig. 3. The 2^3-RQP graph

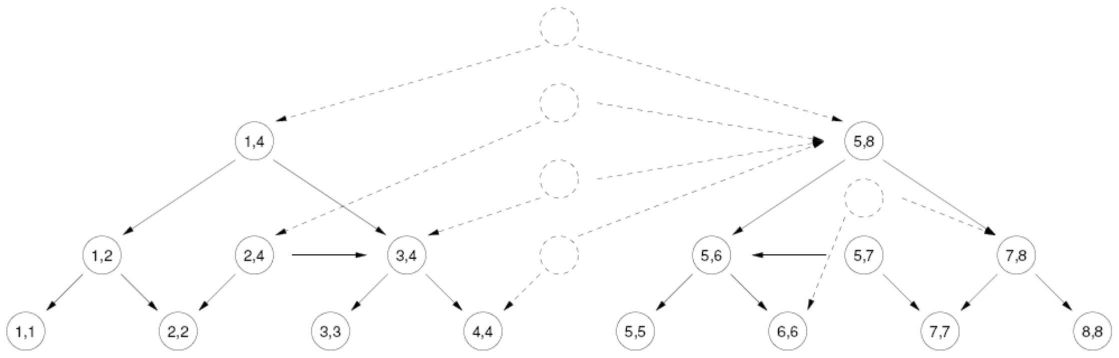

Fig. 4. Left half of the 2^4-RQP graph

Next we present now the formalization of 2^k-RQP graphs.

Definition 16

Let K be a natural number. A 2^K -RQP graph GK is defined inductively:

$$(1) \quad if \quad K = 0 \quad then \quad G^K = (\{1,1)\},0)$$

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

where function Duplicate is defined as

```
function Duplicate (G^K = (V^K,E^K) : Digraph) return Digraph is
  N : constant := 2^K
  M : constant := |V^K|
  V : {(i, j) ∈ 1..N × 1..N • i ≤ j} := ∅;
  E : P(V×V) := 0;
  i, j : 1..(2N);
begin
```

```
  -- The ''cloning'' loops
  for (i, j) in $V^K$ loop
    V := V $\cup${(i, j), (i + N, j + N)};
  end loop;
  for (i, j) $\to$ (i', j') in $E^K$ loop
    E := E $\cup$ {(i, j) $\to$ (i', j'), (i + N, j + N) $\to$ (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) $\notin$ V $\wedge$ j $\le$ N loop
      if (i, j) $\in$ V $\wedge$ (j, N) $\in$ V then
        V := V $\cup$ {(i, N)};
        E := E $\cup$ {(i, N) $\to$ (i, j), (i,N) $\to$ (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) $\notin$ V $\wedge$ i $\le$ 2N loop
      if (N, i) $\in$ V $\wedge$ (i, j) $\in$ V then
        V := V $\cup$ {(N, j)};
        E := E $\cup$ {(N, j) $\to$ (N, i), (N, j) $\to$ (i, j)};
      else
        i := i + 1;
      end if;
    end loop;
  end loop;
  return (V,E);
end Duplicate;
```

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. Toni, *Lower Bounds on Zero-one Matrices*, Linear Algebra and its Applications, 376 (2004) 275--282.

[5] A. Toni, Complejidad y Estructuras de Datos para e*l problema de los rangos variables*, Doctoral Thesis, Facultad de Informática, Universidad Politécnica de Madrid, 2003.

[6] A. Toni, *Matricial Model for the Range Query Problem and Lower Bounds on Complexity*, submitted.

Authors' Information

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

Adriana Toni – Facultad de Informática, Universidad Politécnica de Madrid, Spain; e-mail: atoni@fi.upm.es