The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

Trie Joins

T. H. Merrett, McGill University

Objective: represent two relations as kd-tries and compute directly the kd-trie representing their natural join.

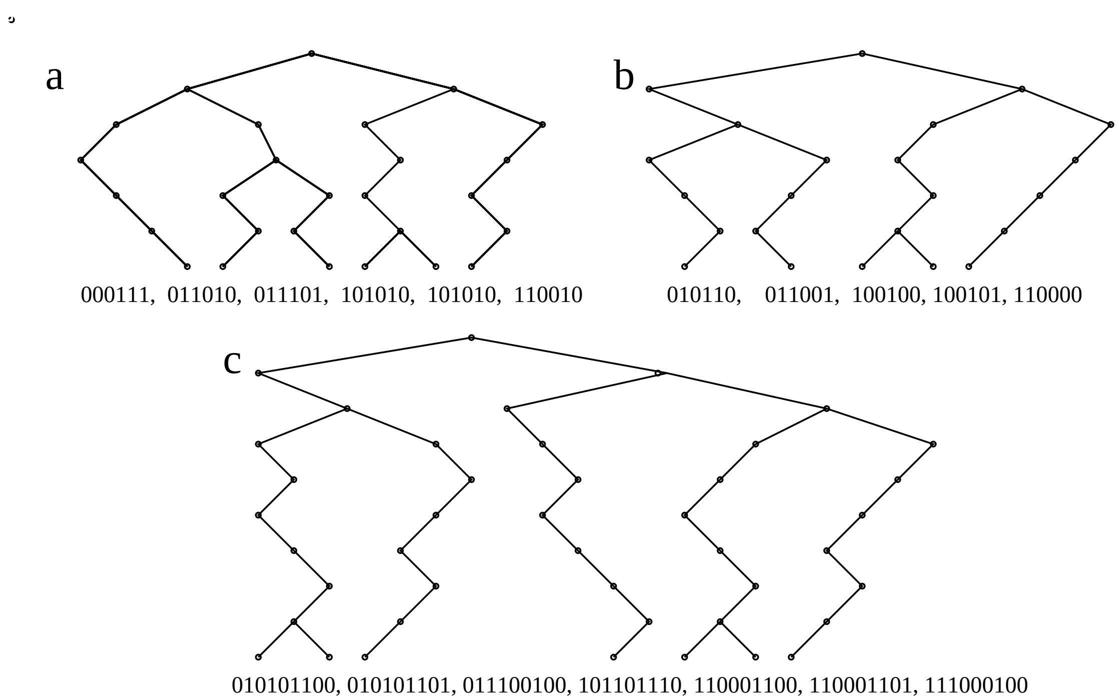
Benefit: work purely with tries, without decompressing the data.

Three relations as kd-tries

$R(A$	$B)$
7	0
7	1
1	3
3	4
5	4
2	7

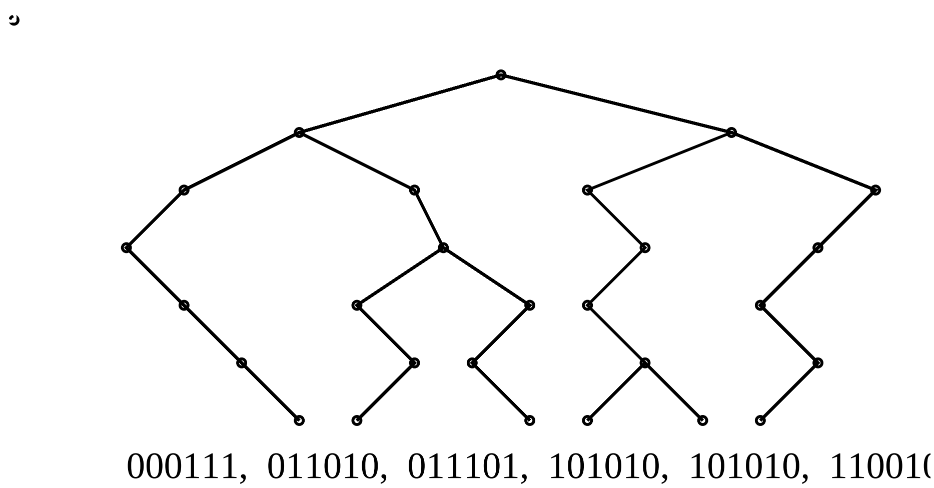
$S(B$	$C)$
1	6
2	5
4	2
4	3
4	4

$T(A$	B	$C)$
7	1	6
3	4	2
3	4	3
3	4	4
5	4	2
5	4	3
5	4	4



a) $R(A,B)$ b) $S(B,C)$ c) $T(A,B,C)$

$R(A, B)$ as bitpairs



11

11 11

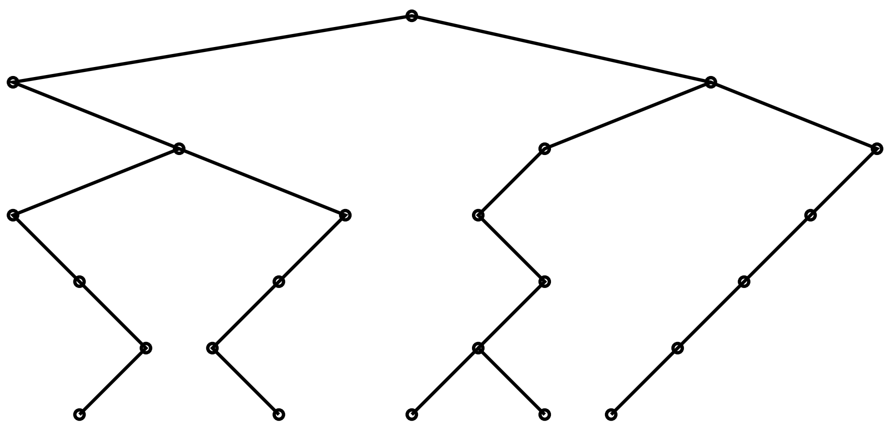
10 01 01 10

01 11 10 10

01 01 10 01 01

01 10 01 11 10

$S(B, C)$ **as bitpairs**



010110, 011001, 100100, 100101, 110000

11

01 11

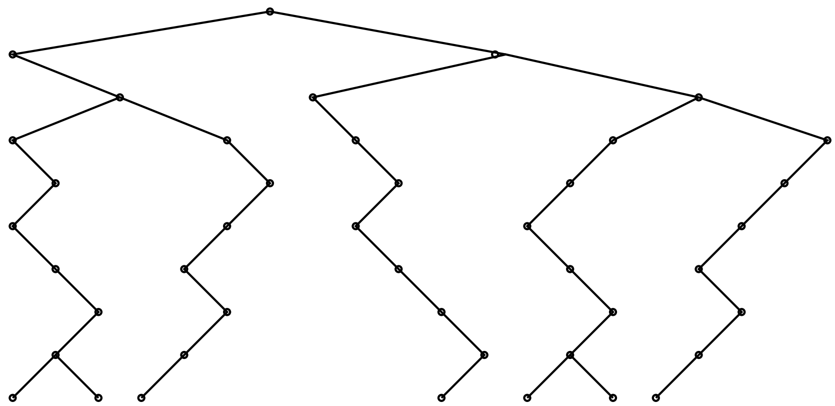
11 10 10

01 10 01 10

01 10 10 10

10 01 11 10

$T(A, B, C)$ as bitpairs



010101100, 010101101, 011100100, 101101110, 110001100, 110001101, 1110001

11

01 11

11 01 11

01 01 01 10 10

10 10 10 10 10

01 10 01 01 10

01 01 01 01 01

10 10 01 10 10

11 10 10 11 10

$T <- R$ **ijoin** S

	$R(A,B)$	$S(B,C)$	$T(A,B,C)$
	11		11
J	11 11	11	01 11
		01 11	11 01 11
	10 01 01 10		01 01 01 10 10
J	01 11 10 10	11 10 10	10 10 10 10 10
		01 10 01 10	01 10 01 01 10
	01 01 10 01 01		01 01 01 01 01
J	01 10 01 11 10	01 10 10 10	10 10 01 10 10
		10 01 11 10	11 10 10 11 10

Note that the rows cycle:

- left (R)
- both (R, S; join attribute)
- right (S)

The algorithm

1. Use the paths in the result so far to predict all possible next steps.
2. See whether and whence these come from the source(s): left, both, right.

The first cycle

left				right				result				final
lev	pos	path	bp	lev	pos	path	bp	lev	pos	path	bp	
0	0		11					0	0		11	
1	1	l	11	0	0		11	1	1	l	11	01
	2	r	11						2	r	11	
				1	1	l	01	2	3	ll	01	X
					2	r	11		4	lr	11	
									5	rl	01	
									5	rr	11	

left

- 11 from left ⟶ result.

both

- (1) Result level 1 will have l node and r node.
- (2) Left level 1 has 2 nodes; right level 1 has 1 node: **and** the Cartesian product, giving result 11, 11.
- (The l node 11 will eventually be corrected to 01 but we don't know this yet.)

right

- (1) Result level 2 will have 4 nodes: ll, lr, rl rr. (ll removed: later.)
- (2) The ll result must come from left l, both l, so copy over right l: 01.
- (2) The lr result must come from left l, both r, so copy over right r: 11.
- (2) The rl result must come from left r, both l, so copy over right l: 01.
- (2) The rr result must come from left r, both r, so copy over right r: 11.

The second cycle

left				right				result				final
lev	pos	path	bp	lev	pos	path	bp	lev	pos	path	bp	
2	3	ll	10					3	7	llr	10	×
	4	lr	01						8	lrl	01	
	5	rl	01						9	lrr	01	
	6	rr	10						10	rlr	01	
									11	rrl	10	
									12	rrr	10	
3	7	lll	01	2	3	lr	11	4	13	llrl	01	×
	8	lrr	11		4	rl	10		14	lrlr	10	
	9	rlr	10		5	rr	10		15	lrrr	10	
	10	rrl	10						16	rlrr	10	
									17	rrll	10	
									18	rrrl	10	
				3	6	lrl	01	5	19	llrlr	10	×
					7	lrr	10		20	lrlrl	01	
					8	rll	01		21	lrrrl	10	
					9	rrr	10		22	rlrrl	01	
									23	rrlll	01	
									24	rrrll	10	

left

- (1) Result level 3 will have 6 nodes: llr, lrl, lrr, rlr, rrl, rrr.
- (2) The llr result must come from left l, both l, right r so copy over 10 (the left ll node). And so on: both lrl and lrr from left lr (01), rlr from left rl (01), and both rrl and rrr from left rr (10).

both

- (1) Result level 4 will have 6 nodes: llrl, lrlr, lrrr, rlrr, rrll, rrrl.
- (2) The llrl result must come from left l, both l, right r and left l, so **and** left lll (01) with right lr (11) giving 01.
  Similarly, lrlr comes from left lrr and right rl, etc.

The third cycle (last for this ex.)

left				right				result				final
lev	pos	path	bp	lev	pos	path	bp	lev	pos	path	bp	
4	11	lllr	01					6	25	llrlrl	01	X
	12	lrrl	01						26	lrlrlr	01	
	13	lrrr	10						27	lrrrll	01	
	14	rlrl	01						28	rlrrlr	01	
	15	rrll	01						29	rrlllr	01	
									30	rrrlll	01	
5	16	lllrr	01	4	10	lrlr	01	7	31	llrlrlr		X
	17	lrrlr	10		11	lrrl	10		31	lrlrlrr	10	
	18	lrrrl	01		12	rllr	10		32	lrrrllr	10	
	19	rlrlr	11		13	rlll	10		33	rlrrlrr	01	
	20	rrllr	10						34	rrlllrr	10	
									35	rrrlllr	10	
				5	14	lrlrr	10	8	36	lrlrlrrl	11	
					15	lrrll	01		37	lrrrllrl	10	
					16	rllrl	11		38	rlrrlrrr	10	
					17	rrlll	10		39	lrrrllrl	11	
									40	rrrllrll	10	

both

- (1) Result level 7 will have 6 nodes: llrlrlr, lrlrlrr, lrrrllr, rlrrlr, rrlllr, rrrlll.
- (2) The llrlrlr result must come from left lllrr and right lrrl, but the latter doesn't exist.

  So this is the end of a false trail, and the path llrlrlr must be removed (i.e., all entries that are prefixes of llrlrlr).

  The earlier pos numbers will also change, but we just continue without the llrlrlr entry.

Fixing up the algorithm

By expanding the paths as lrlrlrrl, etc, we have lost all the compression, so just use the original and growing tries instead.

Analyzing the algorithm

Natural join complexity is $\mathcal{O}(n^2)$ for two operands of size n.

So note the double contributions of the operands to the result, in all three phases of the cycle (left, both, right). The doubling starts with a 11 in the common attribute ("both" phase of the cycle), right in cycle 1 for the example. It may double again with further common 11s, and again, and so on. Thus the algorithm is super-linear.