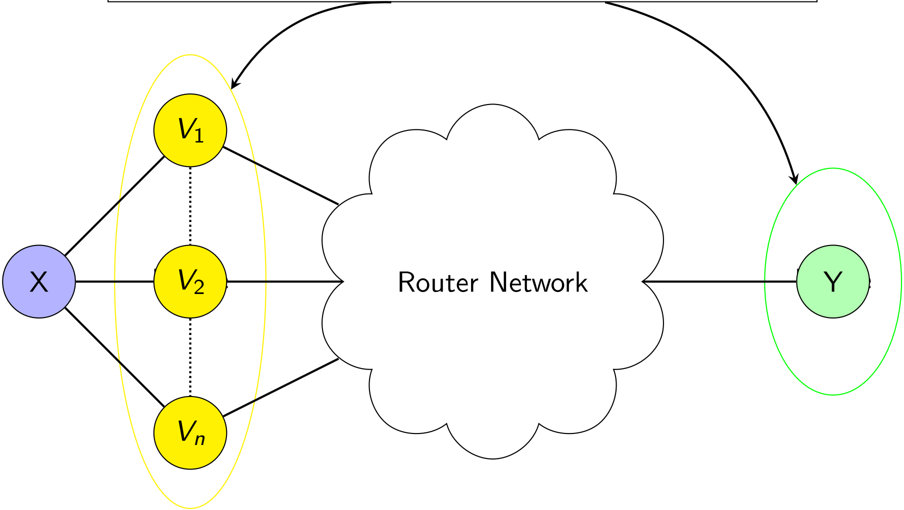
Distance-Vector Routing With SQL

Alexej Onken

27.01.2023

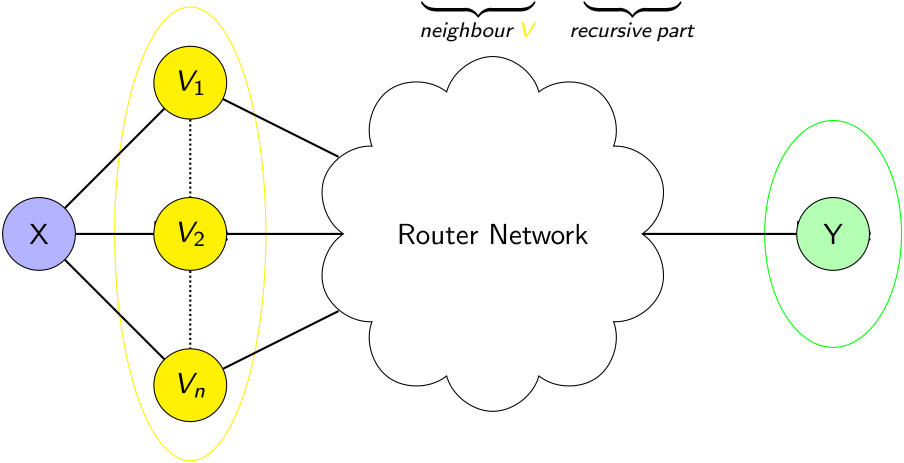
Motivating Example

What is the fastest path from X to Y via neighbour V_i?



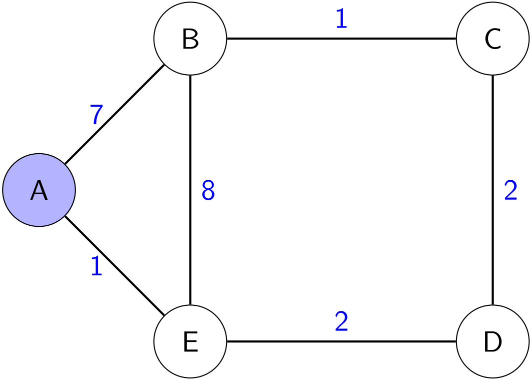
Motivating Example

$$d_X(Y) = \min_V \{\underbrace{c(X, V)}_{\textit{neighbour } V} + \underbrace{d_V(Y)}_{\textit{recursive part}}\}$$



What Are We Dealing With?

Given a router network consisting of routing tables of each node

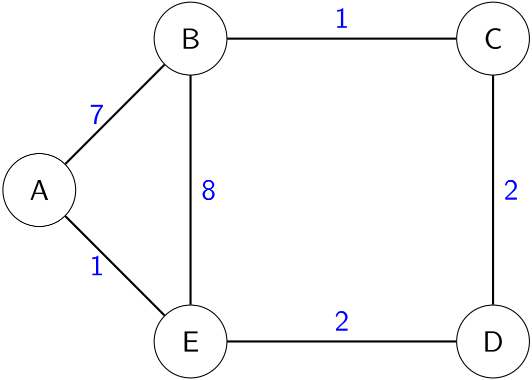


Node A's routing table

	TO			
VIA	B	C	D	E
B	7	–	–	–
C	–	–	–	–
D	–	–	–	–
E	–	–	–	1

What Are We Dealing With?

Distance-Vector Routing is a dynamic protocol from network technology



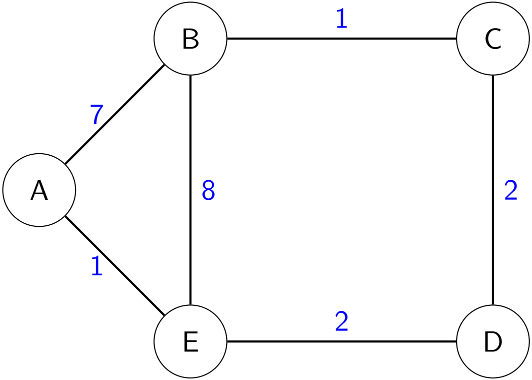
Initial configuration

FROM \ TO	A	B	C	D	E
A	0	7	-	-	1
B	7	0	1	-	8
C	-	1	0	2	-
D	-	-	2	0	2
E	1	8	-	2	0

What Are We Dealing With?

Bellman-Ford in an undirected graph without negative edge weights!

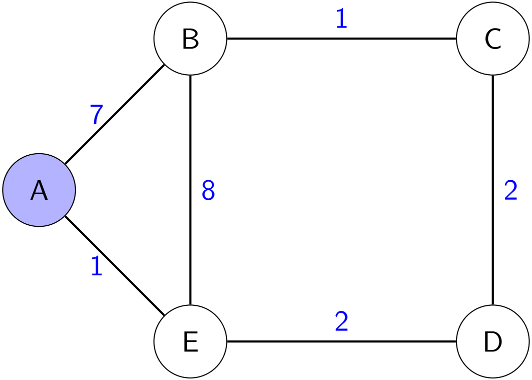
$\hookrightarrow \mathrm{d}_X(Y) = \min_V\{c(X, V) + d_V(Y)\}$



			TO		
FROM	A	B	C	D	E
A	0	7	-	-	1
B	7	0	1	-	8
C	-	1	0	2	-
D	-	-	2	0	2
E	1	8	-	2	0

What Are We Dealing With?

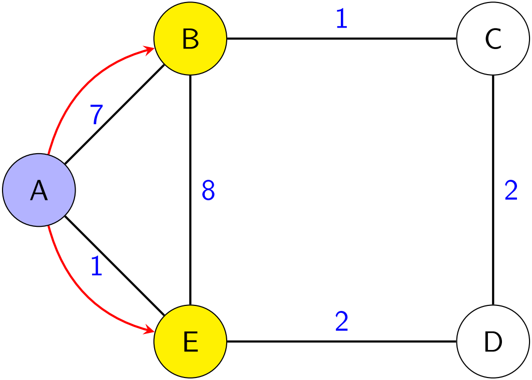
Calculate shortest paths from each node X_i to each other node Y_i !



			TO		
FROM	A	B	C	D	E
A	0	7	-	-	1
B	7	0	1	-	8
C	-	1	0	2	-
D	-	-	2	0	2
E	1	8	-	2	0

What Are We Dealing With?

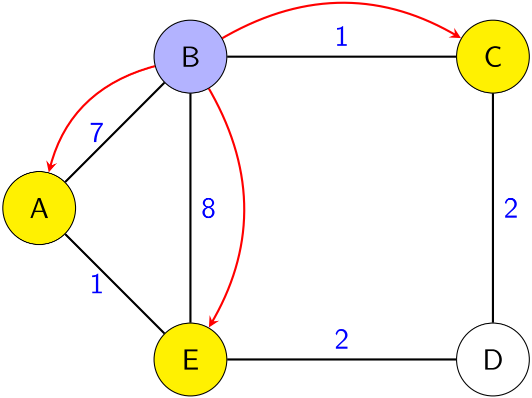
Works on the principle of “tell your neighbours how you see the world“



	TO				
FROM	A	B	C	D	E
A	0	7	-	-	1
B	7	0	1	-	8
C	-	1	0	2	-
D	-	-	2	0	2
E	1	8	-	2	0

What Are We Dealing With?

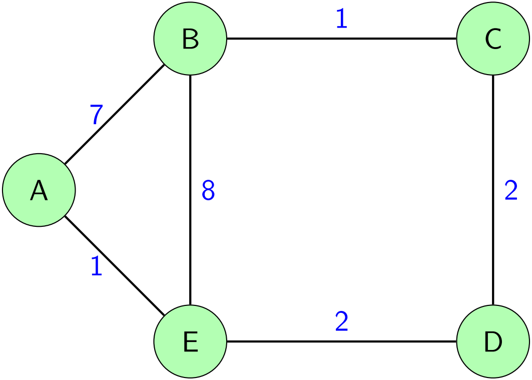
B communicates its current distance vector to neighbours A, C and E



	TO				
FROM	A	B	C	D	E
A	0	7	8	-	1
B	7	0	1	-	8
C	8	1	0	2	9
D	-	-	2	0	2
E	1	8	9	2	0

What Are We Dealing With?

…update distance vectors of all routers in the network until convergence!



	TO				
FROM	A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	2	0

Plan Of Attack With SQL

1. Schema: $graph\,(from, to, via, cost)$

Plan Of Attack With SQL

1. Schema: $graph\,(from, to, via, cost)$
2. Perform a recursive query on a given graph

Plan Of Attack With SQL

1. Schema: $graph\,(from, to, via, cost)$
2. Perform a recursive query on a given graph
3. Explore all possible paths from a node

Plan Of Attack With SQL

1. Schema: *graph* (*from*, *to*, *via*, *cost*)
2. Perform a recursive query on a given graph
3. Explore all possible paths from a node
4. Cut off non-lucrative paths using window functions ☺

Plan Of Attack With SQL

1. Schema: *graph* (*from*, *to*, *via*, *cost*)
2. Perform a recursive query on a given graph
3. Explore all possible paths from a node
4. Cut off non-lucrative paths using window functions ☺
5. Make sure that no cycles are created

Plan Of Attack With SQL

1. Schema: $graph\,(from, to, via, cost)$
2. Perform a recursive query on a given graph
3. Explore all possible paths from a node
4. Cut off non-lucrative paths using window functions ☺
5. Make sure that no cycles are created
6. Keep track of the summed edge costs

Plan Of Attack With SQL

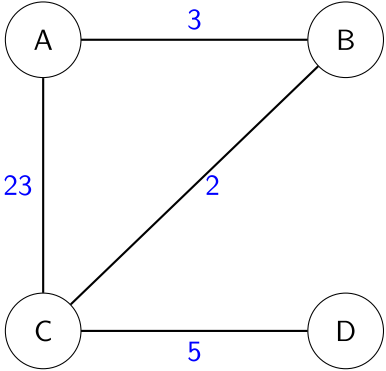
1. Schema: *graph* (*from*, *to*, *via*, *cost*)
2. Perform a recursive query on a given graph
3. Explore all possible paths from a node
4. Cut off non-lucrative paths using window functions ☺
5. Make sure that no cycles are created
6. Keep track of the summed edge costs
7. $d_X(Y) = \min_V \underset{\text{step 2}}{\boxed{\underbrace{\{c(X, V)}_{\textit{neighbour } V}}} + \underset{\text{step 1}}{\boxed{\underbrace{d_V(Y)\}}_{\textit{recursive part}}}}$

Plan Of Attack With SQL

Sample input table:

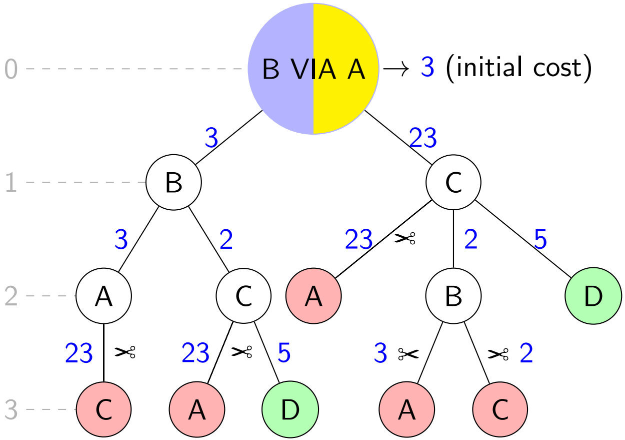
from	to	via	cost
A	B	B	3
A	B	C	Inf
A	C	B	Inf
A	C	C	23
A	D	B	Inf
A	D	C	Inf
⋮	⋮	⋮	⋮
D	C	C	5

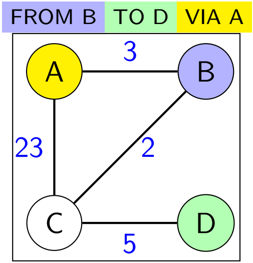
Routing tables A, B, C, D



Plan Of Attack With SQL

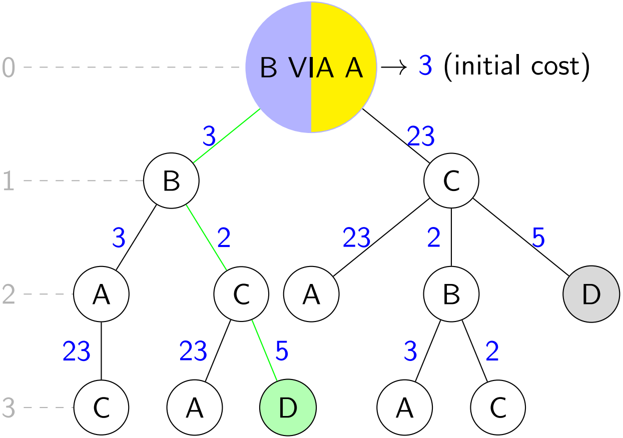
Cut unnecessary paths with branch and bound:



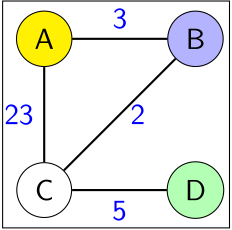


Plan Of Attack With SQL

Do branch and bound for all FROM X_i TO Y_i VIA V_i



FROM B TO D VIA A



Non-Recursive Term

This way we know the start and destination of the track during the recursion steps

Copy those 3 columns

Finished iff to = via

from	to	via	cost next	track	total cost	b & b
A	B	B	3	{A,B,FINISHED}	3	3
A	B	C	23	{A,C}	23	Inf
A	C	C	23	{A,C, FINISHED}	23	23
A	D	B	3	{A,B}	3	Inf
A	D	C	23	{A,C}	23	Inf
⋮	⋮	⋮	⋮	⋮	⋮	⋮
D	A	C	5	{D,C}	5	Inf
D	B	C	5	{D,C}	5	Inf
D	C	C	5	{D,C,FINISHED}	5	5

Non-Recursive Term

```
CREATE OR REPLACE FUNCTION array_smallest(anyarray) RETURNS anyelement
LANGUAGE SQL AS $$
  SELECT min(elements) FROM unnest($1) elements
$$;
```

```
WITH RECURSIVE exploration as (
    SELECT
         d.origin as initialization,         --from (static)
         d.destination as final_destination, --to   (static)
         d.via as first_stopover,            --via  (static)
         d.origin,                           --from (dynamic)
         d.destination,                      --to   (dynamic)
         d.via,                              --via  (dynamic)
         e.cost as cost_next_hop,
         CASE
             WHEN d.destination = d.via
             THEN array[d.origin] || array[d.via] || array['FINISHED']::VARCHAR[]
             ELSE array[d.origin] || array[d.via] END as track,
         e.cost as total_cost,
         CASE
             WHEN d.destination = d.via
             THEN d.cost                                              --initialize upper bounds
             ELSE 'infinity' END as branch_and_bound
    FROM graph as d, graph as e
    WHERE d.origin = e.origin AND d.via = e.via AND e.destination = e.via  --look at neighbour cost only

  UNION ALL

  ...
```

Bellman-Ford perspective:

$$d_X(Y) = \min_V \{\underbrace{c(X,V)}_{\textit{neighbour } V} + \boxed{\underbrace{d_V(Y)\}}_{\textit{recursive part}}}$$

Recursive Term

Recursion depth: 0 FROM A TO B VIA E (Non-Recursive Term)

from	to	via	cost next	track	total cost	b & b
A	B	E	1	{A,E}	1	Inf
E	D	D	2	{A,E,D}	3	9
E	C	C	5	{A,E,C}	6	9
E	B	B	8	{A,E,B,FINISHED}	9	9
E	A	A	1	{A,E,A}	2	9
A	B	B	7	{A,E,A,B,FINISHED}	9	7
D	C	C	2	{A,E,D,C}	5	7
C	D	D	2	{A,E,C,D}	8	7
C	B	B	1	{A,E,C,B,FINISHED}	7	7
C	B	B	1	{A,E,D,C,B,FINISHED}	6	6

Recursive Term

Recursion depth: 1 FROM A TO B VIA E

from	to	via	cost next	track	total cost	b & b
A	B	E	1	{A,E}	1	Inf
E	D	D	2	{A,E,D}	3	9
E	C	C	5	{A,E,C}	6	9
E	B	B	8	{A,E,B,FINISHED}	9	9
E	A	A	1	{A,E,A}	2	9
A	B	B	7	{A,E,A,B,FINISHED}	9	7
D	C	C	2	{A,E,D,C}	5	7
C	D	D	2	{A,E,C,D}	8	7
C	B	B	1	{A,E,C,B,FINISHED}	7	7
C	B	B	1	{A,E,D,C,B,FINISHED}	6	6

Recursive Term

Recursion depth: 2 FROM A TO B VIA E

from	to	via	cost next	track	total cost	b & b
A	B	E	1	{A,E}	1	Inf
E	D	D	2	{A,E,D}	3	9
E	C	C	5	{A,E,C}	6	9
E	B	B	8	{A,E,B,FINISHED}	9	9
E	A	A	1	{A,E,A}	2	9
A	B	B	7	{A,E,A,B,FINISHED}	9	7
D	C	C	2	{A,E,D,C}	5	7
C	D	D	2	{A,E,C,D}	8	7
C	B	B	1	{A,E,C,B,FINISHED}	7	7
C	B	B	1	{A,E,D,C,B,FINISHED}	6	6

Recursive Term

Recursion depth: 2 FROM A TO B VIA E

from	to	via	cost next	track	total cost	b & b
A	B	E	1	{A,E}	1	Inf
E	D	D	2	{A,E,D}	3	9
E	C	C	5	{A,E,C}	6	9
E	B	B	8	{A,E,B,FINISHED}	9	9
E	A	A	1	{A,E,A}	2	9
A	B	B	7	{A,E,A,B,FINISHED}	9	7
D	C	C	2	{A,E,D,C}	5	7
C	D	D	2	{A,E,C,D}	8	7
C	B	B	1	{A,E,C,B,FINISHED}	7	7
C	B	B	1	{A,E,D,C,B,FINISHED}	6	6

Recursive Term

Recursion depth: 2 FROM A TO B VIA E

from	to	via	cost next	track	total cost	b & b
A	B	E	1	{A,E}	1	Inf
E	D	D	2	{A,E,D}	3	9
E	C	C	5	{A,E,C}	6	9
E	B	B	8	{A,E,B,FINISHED}	9	9
E	A	A	1	{A,E,A}	2	9
A	B	B	7	{A,E,A,B,FINISHED}	9	7
D	C	C	2	{A,E,D,C}	5	7
C	D	D	2	{A,E,C,D}	8	7
C	B	B	1	{A,E,C,B,FINISHED}	7	7
C	B	B	1	{A,E,D,C,B,FINISHED}	6	6

Recursive Term

Recursion depth: 3 FROM A TO B VIA E

from	to	via	cost next	track	total cost	b & b
A	B	E	1	{A,E}	1	Inf
E	D	D	2	{A,E,D}	3	9
E	C	C	5	{A,E,C}	6	9
E	B	B	8	{A,E,B,FINISHED}	9	9
E	A	A	1	{A,E,A}	2	9
A	B	B	7	{A,E,A,B,FINISHED}	9	7
D	C	C	2	{A,E,D,C}	5	7
C	D	D	2	{A,E,C,D}	8	7
C	B	B	1	{A,E,C,B,FINISHED}	7	7
C	B	B	1	{A,E,D,C,B,FINISHED}	6	6

Recursive Term

Recursion End: The most cost-effective path wins

from	to	via	cost next	track	total cost	b & b
A	B	E	1	{A,E}	1	Inf
E	D	D	2	{A,E,D}	3	9
E	C	C	5	{A,E,C}	6	9
E	B	B	8	{A,E,B,FINISHED}	9	9
E	A	A	1	{A,E,A}	2	9
A	B	B	7	{A,E,A,B,FINISHED}	9	7
D	C	C	2	{A,E,D,C}	5	7
C	D	D	2	{A,E,C,D}	8	7
C	B	B	1	{A,E,C,B,FINISHED}	7	7
C	B	B	1	{A,E,D,C,B,FINISHED}	6	6

Recursive Term

```
...

UNION ALL

  SELECT
       e.initialization,
       e.final_destination,
       e.first_stopover,
       d.origin,
       d.destination,
       d.via,
       d.cost,
       CASE
           WHEN e.final_destination = d.via
           THEN e.track || array[d.via, 'FINISHED']::VARCHAR[]
           ELSE e.track || array[d.via] END as track,
       e.total_cost+d.cost as total_cost,
       array_smallest(array[e.branch_and_bound]::float[] ||
       array[min(CASE WHEN e.final_destination = d.via
                      THEN e.total_cost+d.cost
                      ELSE 'infinity' END) over win]::float[]) as branch_and_bound
  FROM exploration as e, graph as d
  WHERE e.via = d.origin                                                 AND
        d.destination = d.via                                            AND
        'FINISHED' <> ALL(e.track)                                       AND
        (SELECT cardinality(array_positions(e.track[2:],d.via)) < 1)     AND
        e.total_cost+d.cost <= e.branch_and_bound
  WINDOW win as (PARTITION BY e.initialization, e.final_destination, e.first_stopover
                 ORDER BY e.total_cost RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
)
```

Bellman-Ford perspective:

$$d_X(Y) = \min_V \{\underbrace{c(X,V)}_{\textit{neighbour } V} + \boxed{\underbrace{d_V(Y)\}}_{\textit{recursive part}}}$$

Recursive Term

```
SELECT

     .
     .
     .

     array_smallest(array[e.branch_and_bound]::float[] ||
     array[min(CASE WHEN e.final_destination = d.via                       --min[b_and_b(t-1),b_and_b(t)]
                    THEN e.total_cost+d.cost                               --across finished paths only
                    ELSE 'infinity' END) over win]::float[])
                    as branch_and_bound
FROM exploration as e, graph as d
WHERE e.via = d.origin                                              AND    --link last via with new origin
      d.destination = d.via                                         AND    --look one step ahead
      'FINISHED' <> ALL(s.track)                                    AND    --only unfinished paths
      (SELECT cardinality(array_positions(e.track[2:],d.via)) < 1)  AND    --recognize loops
      e.total_cost+d.cost <= s.branch_and_bound                            --do not violate upper bound
WINDOW win as (PARTITION BY e.initialization,
                            e.final_destination,                           --define window function
                            e.first_stopover                               --partitions:
               ORDER BY s.total_cost                                       --FROM X TO Y VIA V
               RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
```

After recursion...

Find path over minimum neighbour VIA V_i for all FROM X_i TO Y_i

```
SELECT
     d.initialization,      --from
     d.final_destination,   --to
     d.minimal_cost,        --min cost d(x_i,y_i)
     e.track                --protocol of the itinerary
FROM
   (SELECT e.initialization,
         e.final_destination,
         e.first_stopover,
         min(e.total_cost) as minimal_cost
    FROM exploration as e
    WHERE  'FINISHED' = e.track[array_upper(e.track,1)]
    GROUP BY e.initialization, e.final_destination
    ) as d, exploration as e
WHERE  d.initialization    =    e.initialization      AND
       d.final_destination =    s.final_destination   AND
       e.total_cost        =    d.minimal_cost        AND
       'FINISHED' = e.track[array_upper(s.track,1)]
ORDER BY d.initialization, d.final_destination;
```

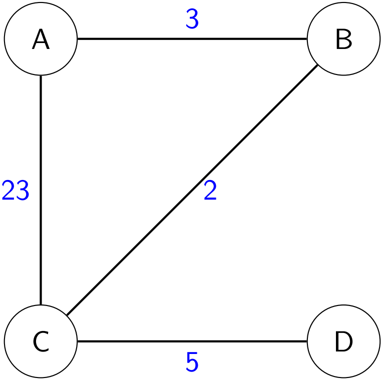
$$d_X(Y) = \min_V \boxed{\underbrace{\{c(X, V)}_{\textit{neighbour } V}} + \underbrace{d_V(Y)\}}_{\textit{recursive part}}$$

From recursive CTE find:

FROM X_i	TO Y_i	VIA V_1	COST C_1
FROM X_i	TO Y_i	VIA V_2	COST C_2
...	...	...	...
FROM X_i	TO Y_i	VIA V_n	COST C_n

$$\min \{c_1, c_2, \dots, c_n\} \ \forall \ X_i, \ Y_i$$

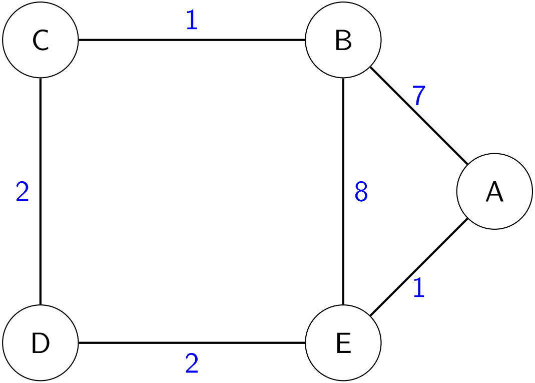
Live Demo With PostgreSQL



Final output:

from	to	min cost	track
A	B	3	{A,B,FINISHED}
A	C	5	{A,B,C,FINISHED}
A	D	10	{A,B,C,D,FINISHED}
B	A	3	{B,A,FINISHED}
B	C	2	{B,C,FINISHED}
B	D	7	{B,C,D,FINISHED}
C	A	5	{C,B,A,FINISHED}
C	B	2	{C,B,FINISHED}
C	D	5	{C,D,FINISHED}
D	A	10	{D,C,B,A,FINISHED}
D	B	7	{D,C,B,FINISHED}
D	C	5	{D,C,FINISHED}

Live Demo With PostgreSQL



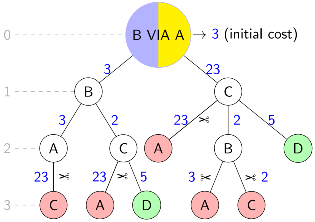
Final output:

from	to	min cost	track
A	B	6	{A,E,D,C,B,FINISHED}
A	C	5	{A,E,D,C,FINISHED}
A	D	3	{A,E,D,FINISHED}
A	E	1	{A,E,FINISHED}
B	A	6	{B,C,D,E,A,FINISHED}
B	C	1	{B,C,FINISHED}
B	D	3	{B,C,D,FINISHED}
B	E	5	{B,C,D,E,FINISHED}
C	A	5	{C,D,E,A,FINISHED}
C	B	1	{C,B,FINISHED}
C	D	2	{C,D,FINISHED}
C	E	4	{C,D,E,FINISHED}
D	A	3	{D,E,A,FINISHED}
D	B	3	{D,C,B,FINISHED}
D	C	2	{D,C,FINISHED}
D	E	2	{D,E,FINISHED}
E	A	1	{E,A,FINISHED}
E	B	5	{E,D,C,B,FINISHED}
E	C	4	{E,D,C,FINISHED}
E	D	2	{E,D,FINISHED}

Any Questions? ☺

Key takaways:

- Perform a recursive query on a given graph
- Keep track of the summed edge costs
- Explore lucrative paths only:
  Build in cycle detections & upper bounds


0
B VIA A
→ 3 (initial cost)
3
23
1
B
C
3
2
23 ✂
2
5
2
A
C
A
B
D
23 ✂
23 ✂ 5
3 ✂
✂ 2
3
C
A
D
A
C