PS1 Recap Slides

Stanford CS145 Fall 2015

PS1: What you learned

- This was a **tough** problem set- congratulations on doing so well!

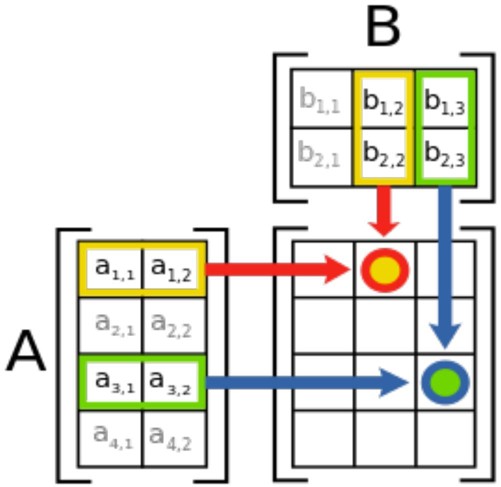
- You used a **declarative** programming language (SQL) to
    - do *linear algebra*
    - answer *questions* about data
    - do *graph* operations
        - Cool stuff! However the point is not these <u>specific</u> applications…

- **Less tricky** versions of these same types of queries will be fair game for exams

Linear Algebra, Declaratively

- Matrix multiplication & other operations = just **joins**!

- The shift from **procedural** to **declarative** programming



$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$$

```
C = [[0]*p for i in range(n)]

for i in range(n):
  for j in range(p):
    for k in range(m):
      C[i][j] += A[i][k] * B[k][j]
```

Proceed through a series of instructions

```
SELECT A.i, B.j, SUM(A.x * B.x)
FROM A, B
WHERE A.j = B.i
GROUP BY A.i, B.j;
```

Declare a desired output set

Common SQL Query Paradigms

GROUP BY / HAVING + Aggregators + Nested queries

```
SELECT station_id,
       COUNT(day) AS nbd
FROM precipitation,
    (SELECT day, MAX(precip)
     FROM precipitation
     GROUP BY day) AS m
WHERE day = m.day AND precip = m.precip
GROUP BY station_id
HAVING COUNT(day) > 1
ORDER BY nbd DESC;
```

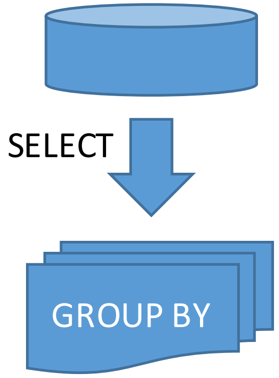
Think about **order*!**

**of the semantics, not the actual execution*

Common SQL Query Paradigms

GROUP BY / HAVING + Aggregators + Nested queries

```
SELECT station_id,
       COUNT(day) AS nbd
FROM precipitation,
  (SELECT day, MAX(precip)
   FROM precipitation
   GROUP BY day) AS m
WHERE day = m.day AND precip = m.precip
GROUP BY station_id
HAVING COUNT(day) > 1
ORDER BY nbd DESC;
```

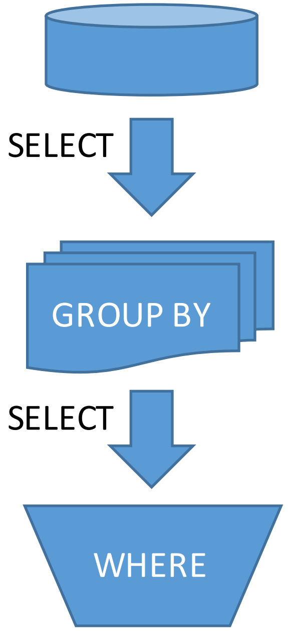


Get the max precipitation **by day**

Common SQL Query Paradigms

GROUP BY / HAVING + Aggregators + Nested queries

```
SELECT station_id,
       COUNT(day) AS nbd
FROM precipitation,
    (SELECT day, MAX(precip)
     FROM precipitation
     GROUP BY day) AS m
WHERE day = m.day AND precip = m.precip
GROUP BY station_id
HAVING COUNT(day) > 1
ORDER BY nbd DESC;
```



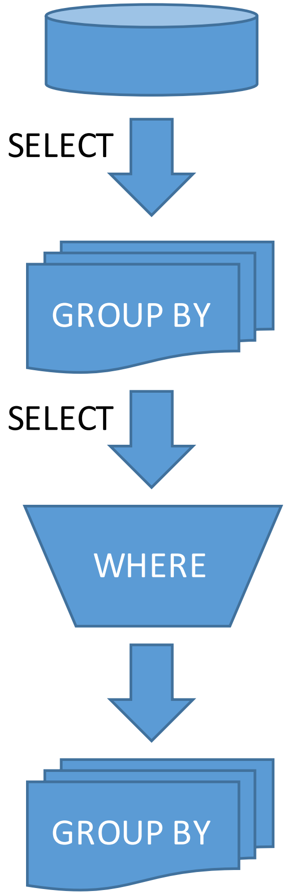
Get the max precipitation **by day**

Get the station, day pairs where / when this happened

Common SQL Query Paradigms

GROUP BY / HAVING + Aggregators + Nested queries

```
SELECT station_id,
       COUNT(day) AS nbd
FROM precipitation,
    (SELECT day, MAX(precip)
     FROM precipitation
     GROUP BY day) AS m
WHERE day = m.day AND precip = m.precip
GROUP BY station_id
HAVING COUNT(day) > 1
ORDER BY nbd DESC;
```



SELECT

GROUP BY

SELECT

WHERE

GROUP BY

Get the max precipitation **by day**

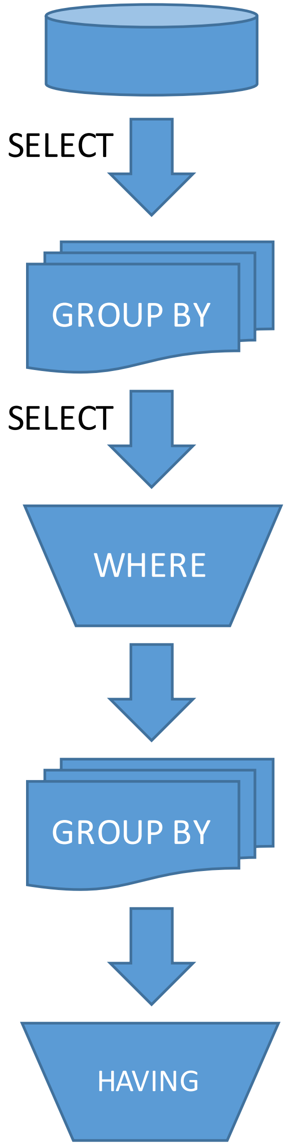
Get the station, day pairs where / when this happened

Group by stations

Common SQL Query Paradigms

GROUP BY / HAVING + Aggregators + Nested queries

```
SELECT station_id,
       COUNT(day) AS nbd
FROM precipitation,
    (SELECT day, MAX(precip)
     FROM precipitation
     GROUP BY day) AS m
WHERE day = m.day AND precip = m.precip
GROUP BY station_id
HAVING COUNT(day) > 1
ORDER BY nbd DESC;
```



Get the max precipitation **by day**

Get the station, day pairs where / when this happened

Group by stations

Having > 1 such day

Common SQL Query Paradigms

Complex correlated queries

```
SELECT x1.p AS median
FROM x AS x1
WHERE
      (SELECT COUNT(*)
       FROM X AS x2
       WHERE x2.p > x1.p)
      =
      (SELECT COUNT(*)
       FROM X AS x2
       WHERE x2.p < x1.p);
```

This was a tricky problem- but good practice in thinking about things declaratively

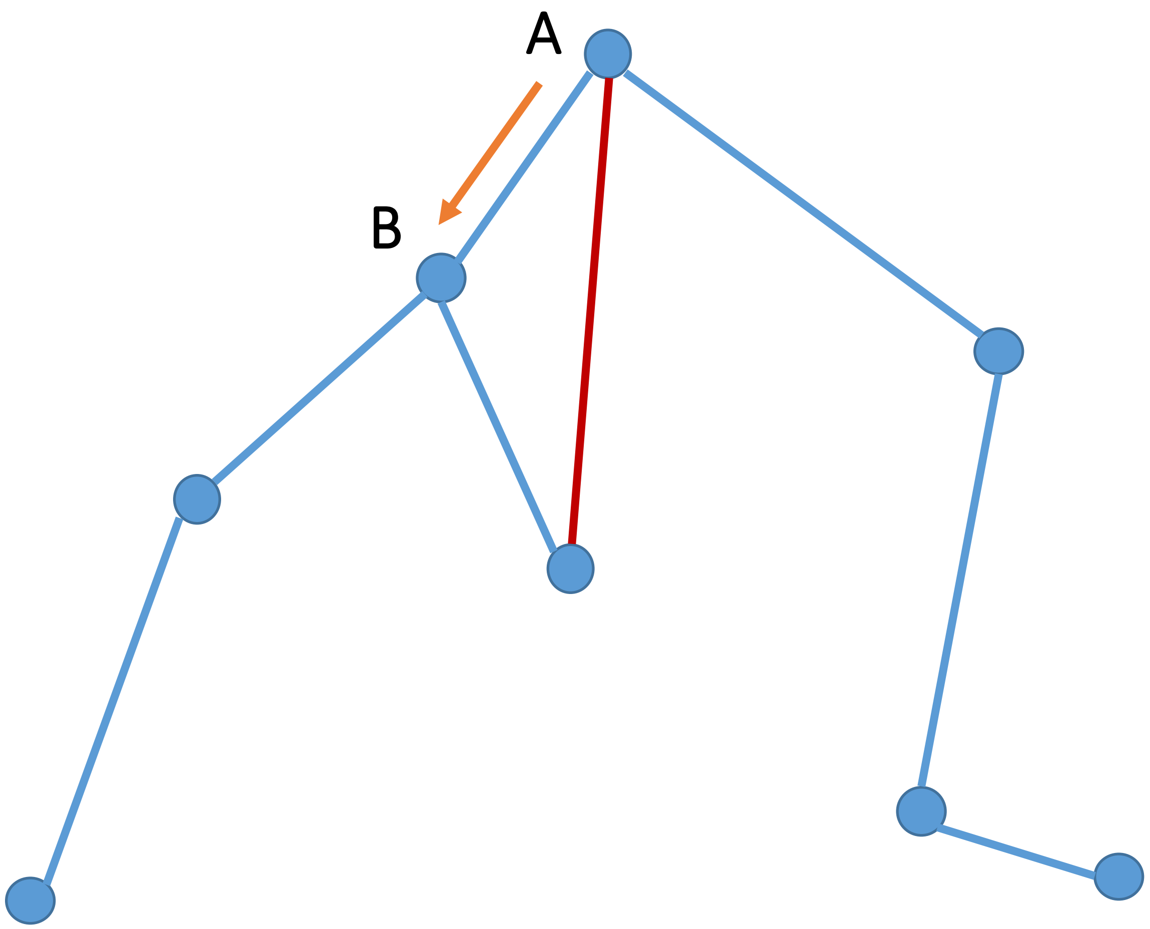
Common SQL Query Paradigms

Nesting + EXISTS / ANY / ALL

```
SELECT sid, p3.precip
FROM (
      SELECT sid, precip
      FROM precipitation AS p1
      WHERE precip > 0 AND NOT EXISTS (
            SELECT p2.precip
            FROM precipitation AS p2
            WHERE p2.sid = p1.sid
              AND p2.precip > 0
              AND p2.precip < p1.precip)) AS p3
WHERE NOT EXISTS (
      SELECT p4.precip
      FROM precipitation AS p4
      WHERE p4.precip – 400 > p3.precip);
```

More complex, but again just think about **order!**

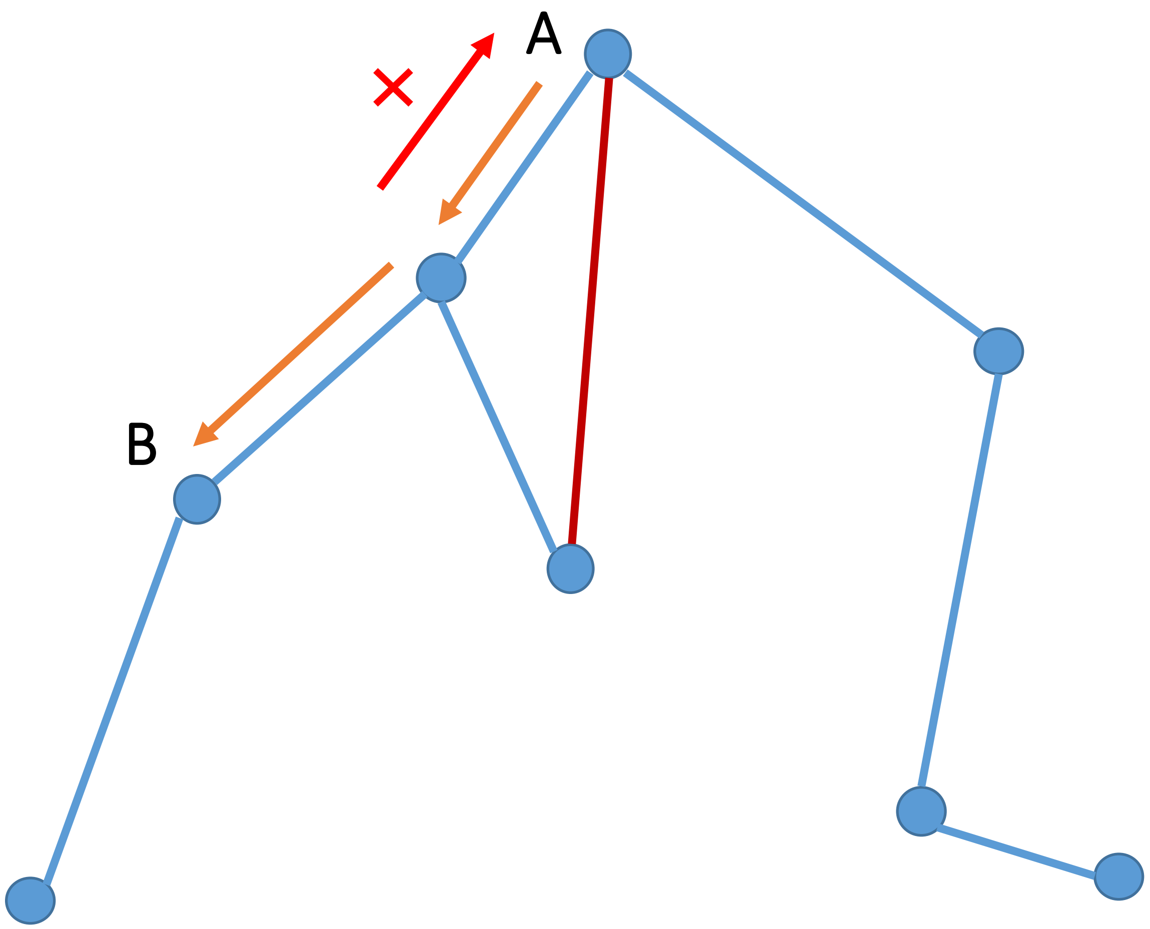
Graph traversal & recursion



For fixed-length paths

```
SELECT A, B, d
FROM edges
```

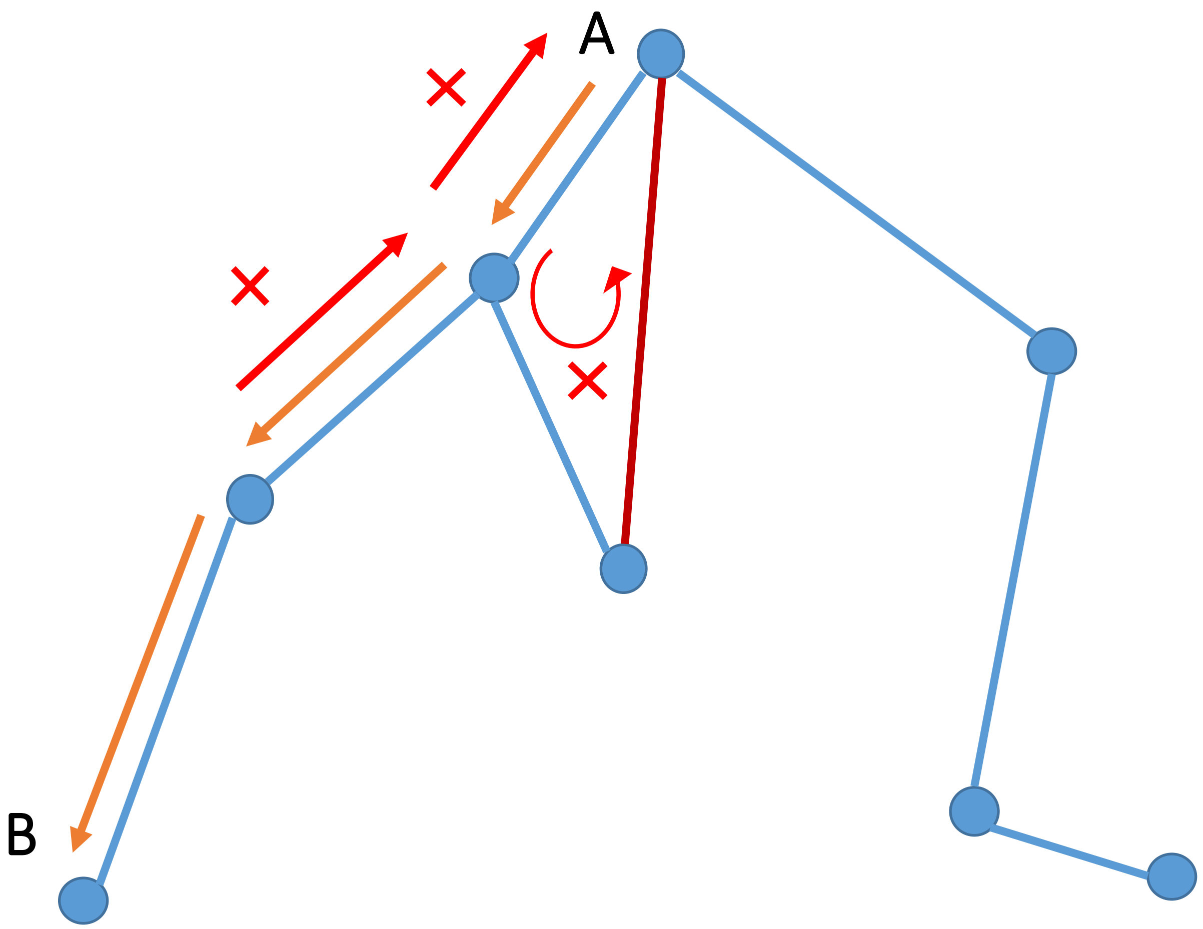
Graph traversal & recursion



For fixed-length paths

```
SELECT A, B, d
FROM edges
UNION
SELECT e1.A, e2.B,
        e1.d + e2.d AS d
FROM edges e1, edges e2
WHERE e1.B = e2.A
  AND e2.B <> e1.A
```

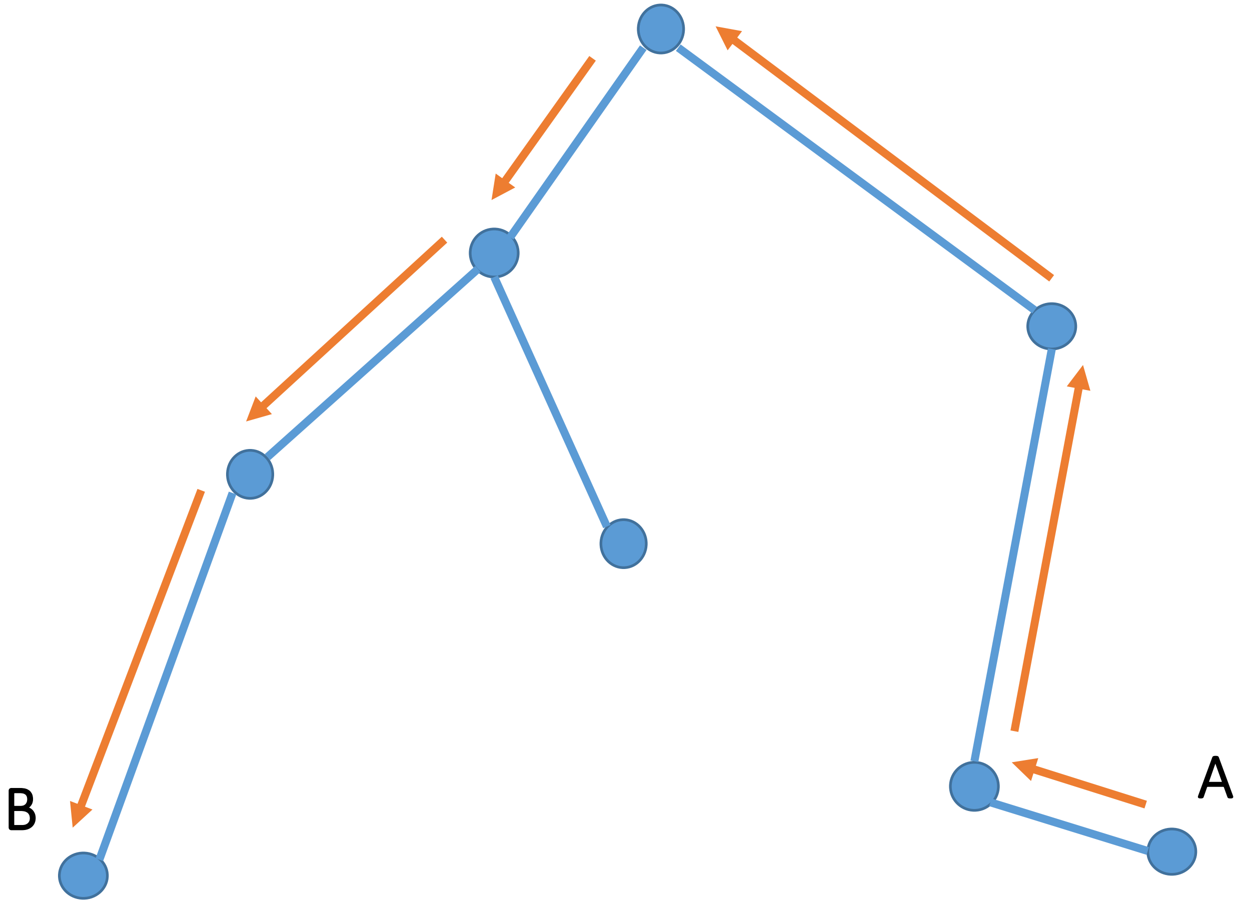
Graph traversal & recursion



For fixed-length paths

```
SELECT A, B, d
FROM edges
UNION
SELECT e1.A, e2.B,
        e1.d + e2.d AS d
FROM edges e1, edges e2
WHERE e1.B = e2.A
  AND e2.B <> e1.A
UNION
SELECT e1.A, e3.B,
        e1.d + e2.d + e3.d AS d
FROM edges e1, edges e2, edges e3
WHERE e1.B = e2.A
  AND e2.B = e3.A
  AND e2.B <> e1.A
  AND e3.B <> e2.A
  AMD e3.B <> e1.A
```

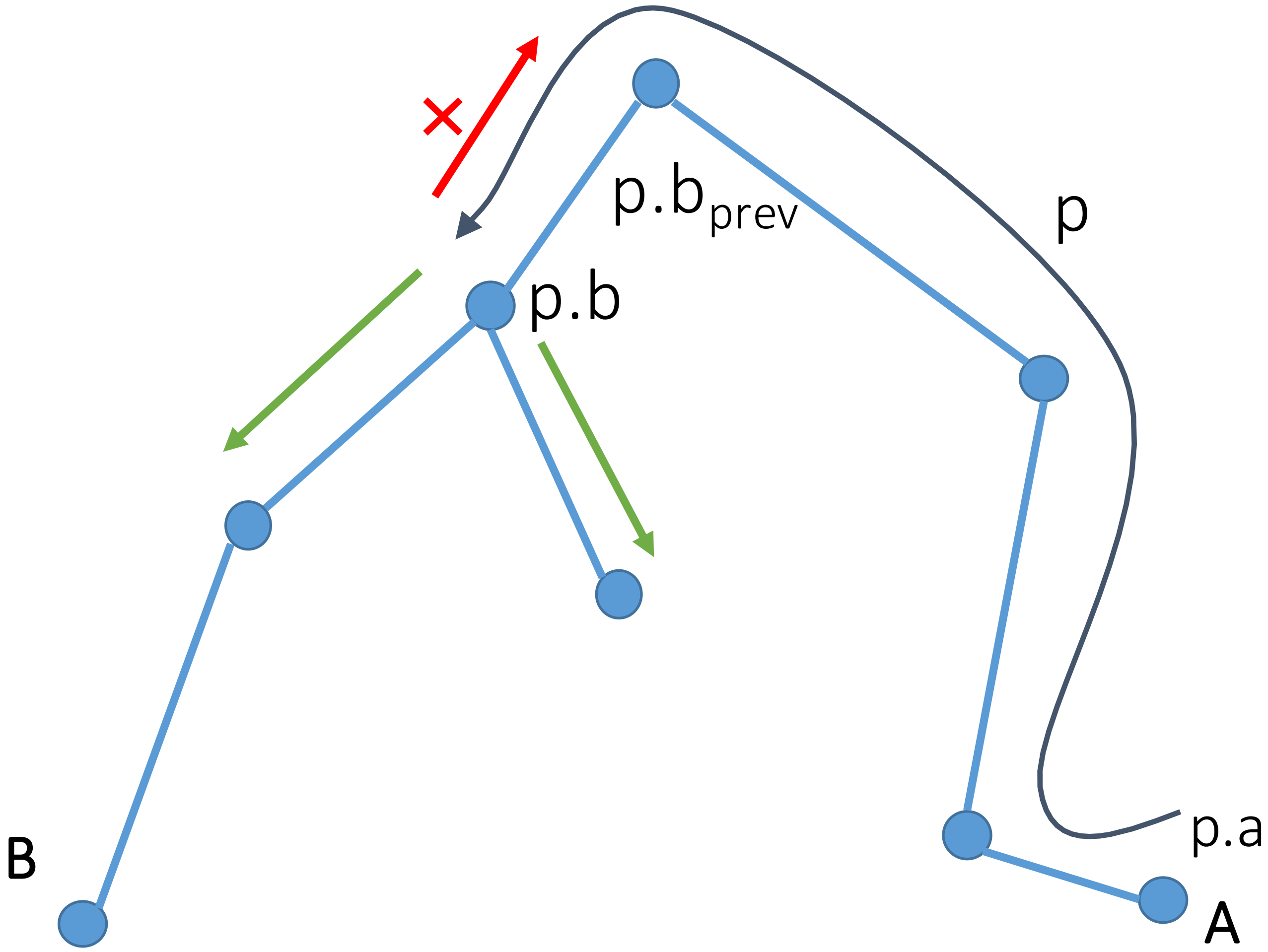
Graph traversal & recursion



For variable-length paths on trees

```
WITH RECURSIVE
paths(a, b, b_prev, d) AS (
      SELECT A, B, A
      FROM edges
      UNION
      SELECT p.a, e.B, e.A,
             p.d + e.d
      FROM paths p, edges e
      WHERE p.b = e.A
        AND s.B <> p.b_prev)
SELECT a, b, MAX(d)
FROM paths;
```

Graph traversal & recursion



For variable-length paths on trees

```
WITH RECURSIVE
paths(a, b, b_prev, d) AS (
      SELECT A, B, A
      FROM edges
      UNION
      SELECT p.a, e.B, e.A,
             p.d + e.d
      FROM paths p, edges e
      WHERE p.b = e.A
        AND e.B <> p.b_prev)
SELECT a, b, MAX(d)
FROM paths;
```