Return of the Antijoins

Conjunctive Queries with Negation and Aggregation: A Linear Time Characterization (PODS'24)

Hangdong Zhao, Austen Fan, Xiating Ouyang and Paraschos Koutris

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Agenda

- What's an Antijoin?
- How to make Antijoins faster?
- A Short Demo

Agenda

- What's an Antijoin?
- How to make Antijoins faster?
- A Short Demo

What is an Antijoin?

Students

StudentId	StudentName
WISC001	John
WISC002	Charlie
WISC003	Sarah
WISC004	Elizabeth
WISC005	George

Enrollments

EnrollNo	ClassCode	StudentId
001	CS577	WISC002
002	CS537	WISC003
003	CS739	WISC002
004	CS564	WISC005
005	CS739	WISC005
006	CS537	WISC001
007	CS537	WISC005

List all student that have NOT yet enrolled in any class


```
SELECT *
FROM Students AS S
WHERE NOT EXISTS (
        SELECT *
        FROM Enrollments AS E
        WHERE S. StudentId = E.StudentId
)
```

What is an Antijoin?

Students

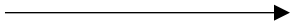
StudentId	StudentName
WISC001	John
WISC002	Charlie
WISC003	Sarah
WISC004	Elizabeth
WISC005	George

Enrollments

Hash Build

EnrollNo	ClassCode	StudentId
001	CS577	WISC002
002	CS537	WISC003
003	CS739	WISC002
004	CS564	WISC005
005	CS739	WISC005
006	CS537	WISC001
007	CS537	WISC005

List all student that have NOT yet enrolled in any class



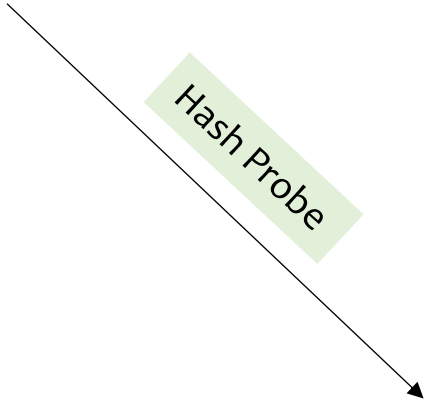
```
Enroll = buildHashtable(Enrollments)
```

```
for (StudentId, StudentName) in Students:
    if StudentId not in Enroll:
        print StudentName
```

What is an Antijoin?

Students

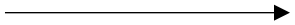
StudentId	StudentName
WISC001	**John**
WISC002	Charlie
WISC003	Sarah
WISC004	Elizabeth
WISC005	George



Enrollments

EnrollNo	ClassCode	StudentId
001	CS577	WISC002
002	CS537	WISC003
003	CS739	WISC002
004	CS564	WISC005
005	CS739	WISC005
006	**CS537**	**WISC001**
007	CS537	WISC005

List all student that have NOT yet enrolled in any class



```
Enroll = buildHashtable(Enrollments)
```

```
for (StudentId, StudentName) in Students:
    if StudentId not in Enroll:
        print StudentName
```

What is an Antijoin?

Students

StudentId	StudentName
WISC001	John
WISC002	**Charlie**
WISC003	Sarah
WISC004	Elizabeth
WISC005	George

Enrollments

Hash Probe

EnrollNo	ClassCode	StudentId
001	**CS577**	**WISC002**
002	CS537	WISC003
003	CS739	WISC002
004	CS564	WISC005
005	CS739	WISC005
006	CS537	WISC001
007	CS537	WISC005

List all student that have NOT yet enrolled in any class

```
Enroll = buildHashtable(Enrollments)

for (StudentId, StudentName) in Students:
    if StudentId not in Enroll:
        print StudentName
```

What is an Antijoin?

Students

StudentId	StudentName
WISC001	John
WISC002	Charlie
WISC003	**Sarah**
WISC004	Elizabeth
WISC005	George

Enrollments

EnrollNo	ClassCode	StudentId
001	CS577	WISC002
002	**CS537**	**WISC003**
003	CS739	WISC002
004	CS564	WISC005
005	CS739	WISC005
006	CS537	WISC001
007	CS537	WISC005

Hash Probe

List all student that have NOT yet enrolled in any class

```
Enroll = buildHashtable(Enrollments)
```

```
for (StudentId, StudentName) in Students:
    if StudentId not in Enroll:
        print StudentName
```

What is an Antijoin?

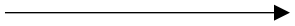
Students

StudentId	StudentName
WISC001	John
WISC002	Charlie
WISC003	Sarah
WISC004	**Elizabeth**
WISC005	George

Enrollments

EnrollNo	ClassCode	StudentId
001	CS577	WISC002
002	CS537	WISC003
003	CS739	WISC002
004	CS564	WISC005
005	CS739	WISC005
006	CS537	WISC001
007	CS537	WISC005

List all student that have NOT yet enrolled in any class



```
Enroll = buildHashtable(Enrollments)
```

```
for (StudentId, StudentName) in Students:
    if StudentId not in Enroll:
        print StudentName
```

What is an Antijoin?

Students

StudentId	StudentName
WISC001	John
WISC002	Charlie
WISC003	Sarah
WISC004	Elizabeth
WISC005	**George**

Hash Probe

Enrollments

EnrollNo	ClassCode	StudentId
001	CS577	WISC002
002	CS537	WISC003
003	CS739	WISC002
004	**CS564**	**WISC005**
005	CS739	WISC005
006	CS537	WISC001
007	CS537	WISC005

List all student that have NOT yet enrolled in any class

```
Enroll = buildHashtable(Enrollments)
```

```
for (StudentId, StudentName) in Students:
    if StudentId not in Enroll:
        print StudentName
```

What is an Antijoin?

Students

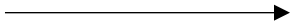
StudentId	StudentName
WISC001	John
WISC002	Charlie
WISC003	Sarah
WISC004	Elizabeth
WISC005	George

Enrollments

EnrollNo	ClassCode	StudentId
001	CS577	WISC002
002	CS537	WISC003
003	CS739	WISC002
004	CS564	WISC005
005	CS739	WISC005
006	CS537	WISC001
007	CS537	WISC005

List all student that have NOT yet enrolled in any class

Elizabeth



```
Enroll = buildHashtable(Enrollments)
```

```
for (StudentId, StudentName) in Students:
    if StudentId not in Enroll:
        print StudentName
```

Linear Time :)

Antijoins are useful to…

Express queries with ‘any’, ‘for all’ semantics

- list all customers who have not placed any order over the past month

Antijoins are useful to…

Express queries with ‘any’, ‘for all’ semantics

- list all customers who have not placed any order over the past month

Find graph patterns

- list all author pairs who have not collaborated with each other, but have a common co-author

$$\Pi_{X,Z}\,\mathrm{CoAuthor}(X,Y) \bowtie \mathrm{CoAuthor}(Y,Z) - \mathrm{CoAuthor}(X,Z)$$

Antijoins are useful to…

Express queries with ‘any’, ‘for all’ semantics

- list all customers who have not placed any order over the past month

Find graph patterns

- list all author pairs who have not collaborated with each other, but have a common co-author

$$\Pi_{X,Z}\ \mathrm{CoAuthor}(X,Y) \bowtie \mathrm{CoAuthor}(Y,Z) - \mathrm{CoAuthor}(X,Z)$$

Take `Differences`

- ″ – ″ operator used in data processing pipelines
- determine if two queries Q1, Q2 yield identical outputs <=> check if both Q1 – Q2 and Q2 – Q1 are empty

Another Antijoin

Students

StudentId	StudentName
WISC001	John
WISC002	Charlie
WISC003	Sarah
WISC004	Elizabeth
WISC005	George

CSRequirements

ClassCode	ClassName
CS577	Intro to Algorithms
CS564	Databases
CS536	Compilers and PL
CS537	Intro to OS
CS739	Distributed Systems

Enrollments

EnrollNo	ClassCode	StudentId
001	CS577	WISC002
002	CS537	WISC003
003	CS739	WISC002
004	CS564	WISC005
005	CS739	WISC005
006	CS537	WISC001
007	CS537	WISC005

Find the list of required classes that each student has NOT yet enrolled in

Another Antijoin

Students

StudentId	StudentName
…	…

CSRequirements

ClassCode	ClassName
…	…

Enrollments

EnrollNo	ClassCode	StudentId
…	…	…

Find the list of required classes that each student has NOT yet enrolled in

```
SELECT S.StudentId, C.ClassCode
FROM Students AS S,
        CSRequirements AS C
WHERE NOT EXISTS (
      SELECT *
      FROM Enrollments AS E
      WHERE E. ClassCode = C.ClassCode
      AND     E. StudentId = S.StudentId
)
```

Another Antijoin

Students

StudentId	StudentName
…	…

CSRequirements

ClassCode	ClassName
…	…

Enrollments

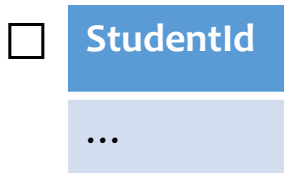
EnrollNo	ClassCode	StudentId
…	…	…

Find the list of required classes that each student has NOT yet enrolled in

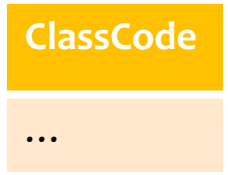
```
SELECT S.StudentId, C.ClassCode
FROM Students AS S,
        CSRequirements AS C
WHERE NOT EXISTS (
      SELECT *
      FROM Enrollments AS E
      WHERE E. ClassCode = C.ClassCode
      AND     E. StudentId = S.StudentId
)
```

$$\Pi_{\text{StudentId}}\text{Students} \times \Pi_{\text{ClassCode}}\text{CSRequirements} - \Pi_{\text{StudentId,ClassCode}}\text{Enrollment}$$

Another Antijoin (Simplified)



C



E

ClassCode	StudentId
…	…

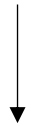
Find the list of required classes that each student has NOT yet enrolled in

```
SELECT *
FROM S,
        C
WHERE NOT EXISTS (
        SELECT *
        FROM E
        WHERE E. ClassCode = C.ClassCode
        AND     E. StudentId = S.StudentId
)
```

$$\mathrm{S}(\mathrm{StudentId}) \times \mathrm{C}(\mathrm{ClassCode}) - \mathrm{E}(\mathrm{StudentId}, \mathrm{ClassCode})$$

Another Antijoin (Simplified)

$$\mathrm{S}(\mathrm{StudentId}) \times \mathrm{C}(\mathrm{ClassCode}) - \mathrm{E}(\mathrm{StudentId}, \mathrm{ClassCode})$$



```
E = buildHashtable(E)

for StudentId in S:
    for ClassCode in C: // cross-product
        if (StudentId, ClassCode) not in E:
            print (StudentId, ClassCode)
```

Quadratic Time :(

Our Theme
Is there a linear-time algorithm?

Yes :)

Agenda

- What's an Antijoin?
- How to make Antijoins faster?
- A Short Demo

(1/2) Linear-time Preprocessing

```
C_array = buildArray(C)

HashTables = { WISCXXX : {} } // init empty hashtables for StudentId in S

for (ClassCode, StudentId) in E:
    skip(ClassCode, HashTables[StudentId])
```

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	CS537
5	CS739

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

ClassCode	StudentId
CS577	WISC002
CS537	WISC003
CS739	WISC002
CS564	WISC005
CS739	WISC005
CS537	WISC001
CS537	WISC005

(1/2) Linear-time Preprocessing

```
C_array = buildArray(C)

HashTables = { WISCXXX : {} } // init empty hashtables for StudentId in S

for (ClassCode, StudentId) in E:
    skip(ClassCode, HashTables[StudentId])
```

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	CS537
5	CS739

HashTables

```
{
    WISC001: {}
    WISC002: {}
    WISC003: {}
    WISC004: {}
    WISC005: {}
}
```

ClassCode	StudentId
CS577	WISC002
CS537	WISC003
CS739	WISC002
CS564	WISC005
CS739	WISC005
CS537	WISC001
CS537	WISC005

(1/2) Linear-time Preprocessing

```
C_array = buildArray(C)

HashTables = { WISCXXX : {} } // init empty hashtables for StudentId in S
```

```
for (ClassCode, StudentId) in E:
    skip(ClassCode, HashTables[StudentId])
```

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	**CS577**
2	CS564
3	CS536
4	CS537
5	CS739

HashTables

```
{
    WISC001: {}
    WISC002: {0 : 2}
    WISC003: {}
    WISC004: {}
    WISC005: {}
}
```

skip (1, WISC002)

ClassCode	StudentId
CS577	**WISC002**
CS537	WISC003
CS739	WISC002
CS564	WISC005
CS739	WISC005
CS537	WISC001
CS537	WISC005

(1/2) Linear-time Preprocessing

```
C_array = buildArray(C)

HashTables = { WISCXXX : {} } // init empty hashtables for StudentId in S
```

```
for (ClassCode, StudentId) in E:
    skip(ClassCode, HashTables[StudentId])
```

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	**CS537**
5	CS739

HashTables

```
{
    WISC001: {}
    WISC002: {0 : 2}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {}
}
```

skip (4, WISC003)

ClassCode	StudentId
CS577	WISC002
CS537	**WISC003**
CS739	WISC002
CS564	WISC005
CS739	WISC005
CS537	WISC001
CS537	WISC005

(1/2) Linear-time Preprocessing

```
C_array = buildArray(C)

HashTables = { WISCXXX : {} } // init empty hashtables for StudentId in S

for (ClassCode, StudentId) in E:
    skip(ClassCode, HashTables[StudentId])
```

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	CS537
5	**CS739**

HashTables

```
{
    WISC001: {}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {}
}
```

skip (5, WISC002)

ClassCode	StudentId
CS577	WISC002
CS537	WISC003
CS739	**WISC002**
CS564	WISC005
CS739	WISC005
CS537	WISC001
CS537	WISC005

(1/2) Linear-time Preprocessing

```
C_array = buildArray(C)

HashTables = { WISCXXX : {} } // init empty hashtables for StudentId in S
```

```
for (ClassCode, StudentId) in E:
    skip(ClassCode, HashTables[StudentId])
```

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	**CS564**
3	CS536
4	CS537
5	CS739

HashTables

```
{
    WISC001: {}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3}
}
```

skip (2, WISC005)

ClassCode	StudentId
CS577	WISC002
CS537	WISC003
CS739	WISC002
CS564	**WISC005**
CS739	WISC005
CS537	WISC001
CS537	WISC005

(1/2) Linear-time Preprocessing

```
C_array = buildArray(C)

HashTables = { WISCXXX : {} } // init empty hashtables for StudentId in S

for (ClassCode, StudentId) in E:
    skip(ClassCode, HashTables[StudentId])
```

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	CS537
5	**CS739**

HashTables

```
{
    WISC001: {}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 4 : 6}
}
```

skip (5, WISC005)

ClassCode	StudentId
CS577	WISC002
CS537	WISC003
CS739	WISC002
CS564	WISC005
CS739	**WISC005**
CS537	WISC001
CS537	WISC005

(1/2) Linear-time Preprocessing

```
C_array = buildArray(C)

HashTables = { WISCXXX : {} } // init empty hashtables for StudentId in S

for (ClassCode, StudentId) in E:
    skip(ClassCode, HashTables[StudentId])
```

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	**CS537**
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 4 : 6}
}
```

skip (4, WISC001)

ClassCode	StudentId
CS577	WISC002
CS537	WISC003
CS739	WISC002
CS564	WISC005
CS739	WISC005
CS537	**WISC001**
CS537	WISC005

(1/2) Linear-time Preprocessing

```
C_array = buildArray(C)

HashTables = { WISCXXX : {} } // init empty hashtables for StudentId in S

for (ClassCode, StudentId) in E:
    skip(ClassCode, HashTables[StudentId])
```

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	**CS537**
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

skip (4, WISC005)

ClassCode	StudentId
CS577	WISC002
CS537	WISC003
CS739	WISC002
CS564	WISC005
CS739	WISC005
CS537	WISC001
CS537	**WISC005**

(1/2) Linear-time Preprocessing

```
C_array = buildArray(C)

HashTables = { WISCXXX : {} } // init empty hashtables for StudentId in S

for (ClassCode, StudentId) in E:
    skip(ClassCode, HashTables[StudentId])
```

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	**CS537**
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

Linear Time
i.e. runs in O(|*Input*|) time

ClassCode	StudentId
CS577	WISC002
CS537	WISC003
CS739	WISC005
CS537	WISC001
CS537	**WISC005**

Now we don't need the Enrollment table

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	CS537
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

Output

StudentId	ClassCode
WISC001	**C_array[1]**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	**CS577**
2	CS564
3	CS536
4	CS537
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

Output

StudentId	ClassCode
WISC001	C_array[1]
WISC001	**C_array[2]**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	**CS564**
3	CS536
4	CS537
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

Output

StudentId	ClassCode
WISC001	C_array[1]
WISC001	C_array[2]
WISC001	**C_array[3]**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	**CS536**
4	CS537
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

Output

StudentId	ClassCode
WISC001	C_array[1]
WISC001	C_array[2]
WISC001	C_array[3]
WISC001	**C_array[5]**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	CS537
5	**CS739**

HashTables

```
{
    WISC001: {3 : 5} // (WISC001, C_array[4]) skipped
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

Output

StudentId	ClassCode
WISC001	C_array[1]
WISC001	C_array[2]
WISC001	C_array[3]
WISC001	C_array[5]
WISC002	**C_array[2]**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	**CS564**
3	CS536
4	CS537
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6} // (WISC002, C_array[1])
                                skipped
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

Output

StudentId	ClassCode
WISC001	C_array[1]
WISC001	C_array[2]
WISC001	C_array[3]
WISC001	C_array[5]
WISC002	C_array[2]
WISC002	**C_array[3]**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	**CS536**
4	CS537
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

Output

StudentId	ClassCode
WISC001	C_array[1]
WISC001	C_array[2]
WISC001	C_array[3]
WISC001	C_array[5]
WISC002	C_array[2]
WISC002	C_array[3]
WISC002	**C_array[4]**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	**CS537**
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6} // (WISC002, C_array[5])
                               skipped
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

Output

StudentId	ClassCode
WISC001	C_array[1]
WISC001	C_array[2]
WISC001	C_array[3]
WISC001	C_array[5]
WISC002	C_array[2]
WISC002	C_array[3]
WISC002	C_array[4]
WISC003	**C_array[1]**
☐	**{2, 3, 5}**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	**CS577**
2	**CS564**
3	**CS536**
4	CS537
5	**CS739**

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
}
```

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

Output

StudentId	ClassCode
WISC001	C_array[1]
WISC001	C_array[2]
WISC001	C_array[3]
WISC001	C_array[5]
WISC002	C_array[2]
WISC002	C_array[3]
WISC002	C_array[4]
WISC003	C_array[1]
…	{2, 3, 5}
WISC004	**C_array[1]**
□	**{2, 3, 4, 5}**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

C_array

idx	ClassCode
1	CS577
2	CS564
3	CS536
4	CS537
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {} // no skips
    WISC005: {1 : 3, 3 : 6}
}
```

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

C_array

idx	ClassCode
1	**CS577**
2	CS564
3	**CS536**
4	CS537
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
    // {2, 4, 5} skipped
}
```

Output

StudentId	ClassCode
WISC001	C_array[1]
WISC001	C_array[2]
WISC001	C_array[3]
WISC001	C_array[5]
WISC002	C_array[2]
WISC002	C_array[3]
WISC002	C_array[4]
WISC003	C_array[1]
…	{2, 3, 5}
WISC004	C_array[1]
□	{2, 3, 4, 5}
WISC005	**C_array[1]**
WISC005	**C_array[3]**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

(2/2) Constant-time Enumeration

```
for StudentId in S:
    for idx in range(len(C)): // 0..6
        if idx == 0 or len(C): // 0 or 6
            continue
        print (StudentId, C_array[idx])
        if idx in HashTables[StudentId]:
            // skip to the next output tuple
            idx = HashTables[StudentId][idx]
```

C_array

idx	ClassCode
1	**CS577**
2	CS564
3	**CS536**
4	CS537
5	CS739

HashTables

```
{
    WISC001: {3 : 5}
    WISC002: {0 : 2, 4 : 6}
    WISC003: {3 : 5}
    WISC004: {}
    WISC005: {1 : 3, 3 : 6}
    // {2, 4, 5} skipped
}
```

Output

StudentId	ClassCode
WISC001	C_array[1]
WISC001	C_array[2]
WISC001	C_array[3]
WISC001	C_array[5]
WISC002	C_array[2]
WISC002	C_array[3]
WISC002	C_array[4]
WISC003	C_array[1]
…	{2, 3, 5}
WISC004	C_array[1]
□	{2, 3, 4, 5}
WISC005	**C_array[1]**
WISC005	**C_array[3]**

StudentId
WISC001
WISC002
WISC003
WISC004
WISC005

×

ClassCode
CS577
CS564
CS536
CS537
CS739

O(|*Output*|) Time

Our Algorithm…

is *optimal* for any given input

- runs in O(|*Input*| + |*Output*|) time

can be generalized to *join-aggregate / join-project* queries via lens of *semirings*

- reductions to Range Sum Queries

works when the query graph is *free-connex signed-acyclic*

- we proved that linear time beyond this class is **unattainable** (so linear-time if and only if free-connex signed-acyclic)

$$\Pi_{X,Z}\ \mathrm{CoAuthor}(X,Y) \bowtie \mathrm{CoAuthor}(Y,Z) - \mathrm{CoAuthor}(X,Z)$$

- but similar ideas can still be used beyond this class to get faster algorithms

Agenda

- What's an Antijoin?
- How to make Antijoins faster?
- A Short Demo

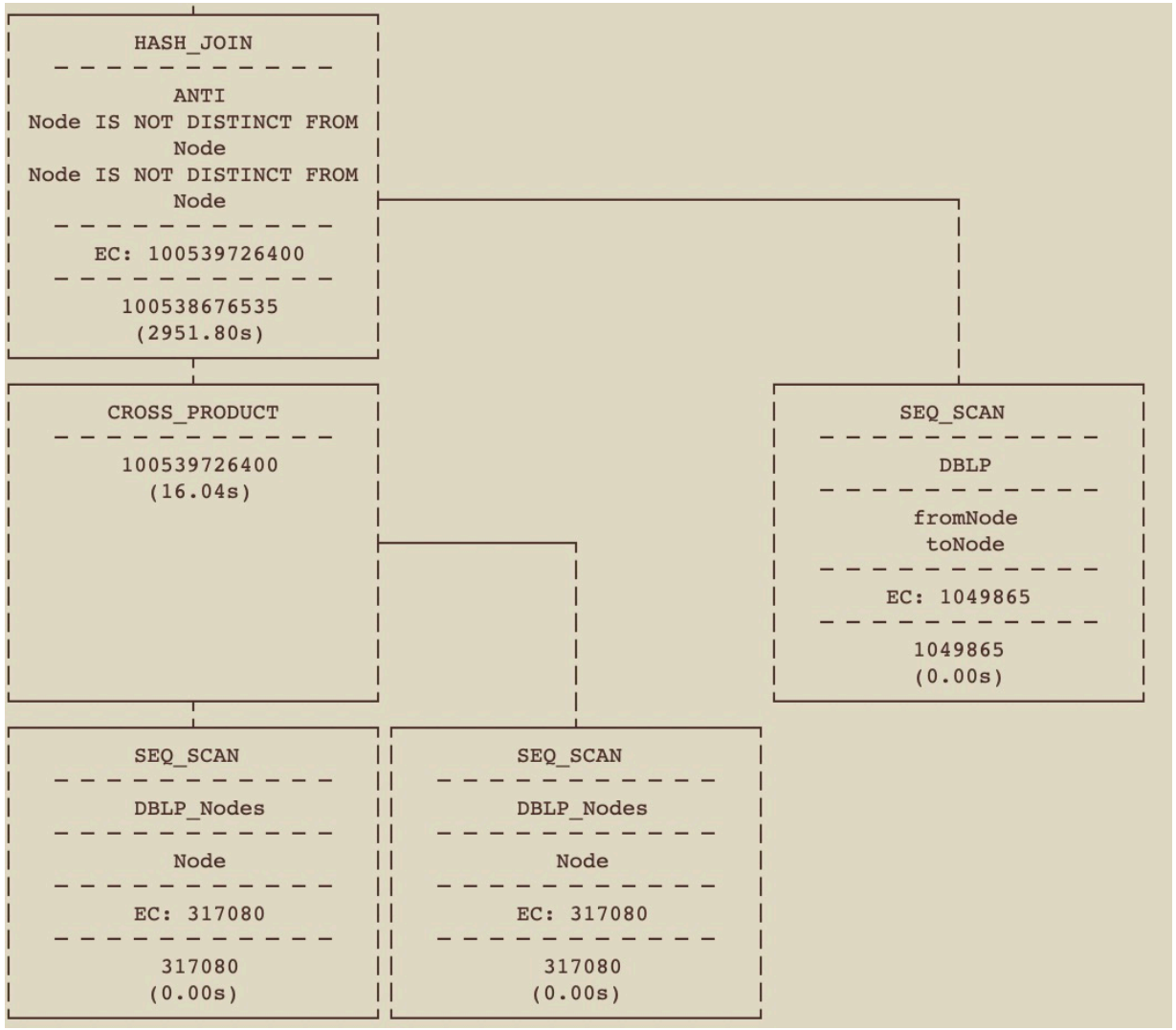
Practicality (Ongoing)

DBLP Datasets: contain co-author info (~300K nodes, ~1M edges)

Query: // list all author pairs that haven't yet collaborated

```
SELECT A.fromNode, B.toNode
FROM DBLPNodes A,
        DBLPNodes B
WHERE NOT EXISTS (
        SELECT *
        FROM DBLP R
        WHERE A. Node = R.fromNode
        AND B. Node = R. toNode
)
```

Baselines: DuckDB runs in ~50 min



Practicality (Ongoing)

DBLP Datasets: contain co-author info (~300K nodes, ~1M edges)

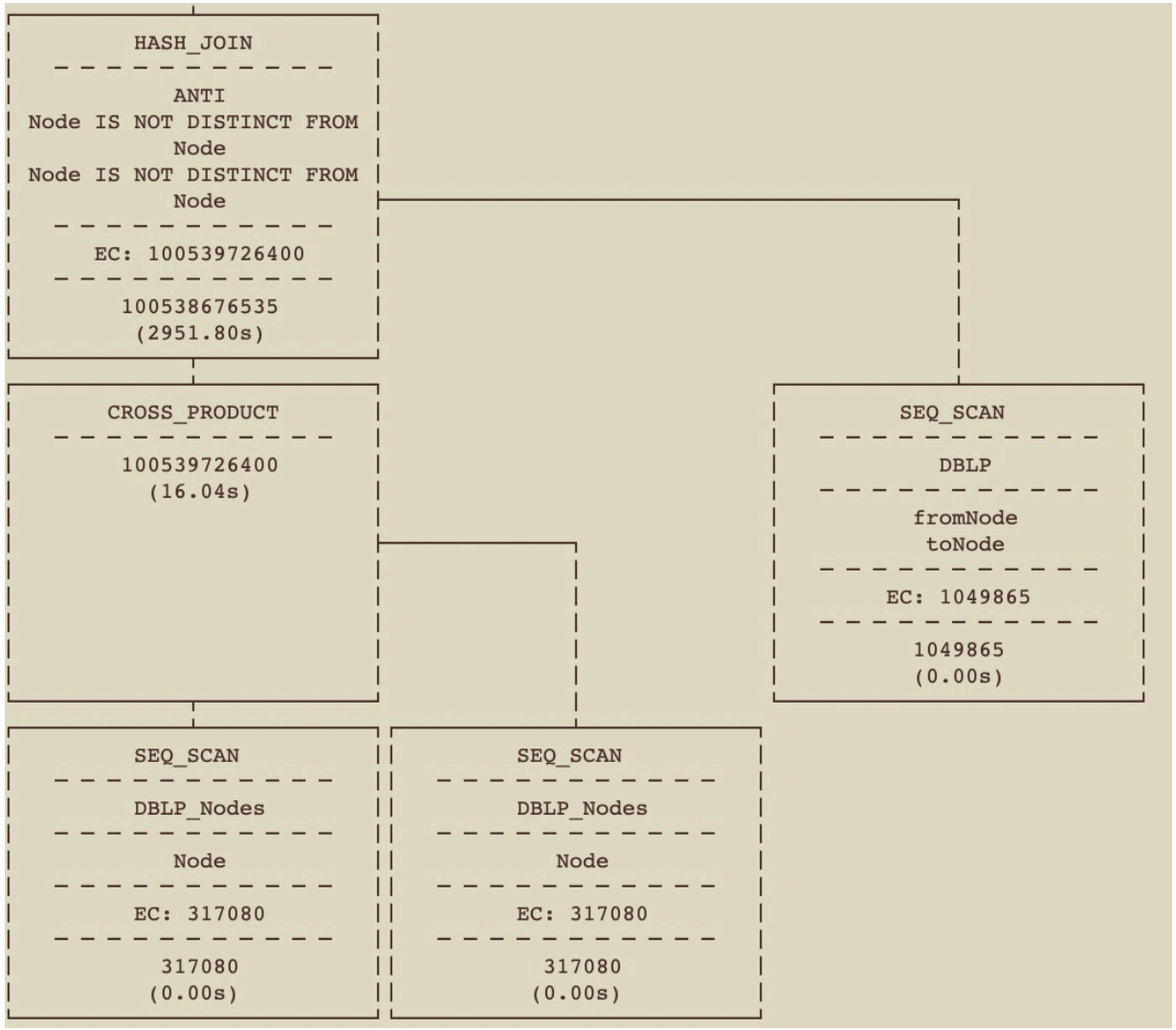
Query: // list all author pairs that haven't yet collaborated

```
SELECT A.fromNode, B.toNode
FROM DBLPNodes A,
        DBLPNodes B
WHERE NOT EXISTS (
        SELECT *
        FROM DBLP R
        WHERE A. Node = R.fromNode
        AND B. Node = R. toNode
)
```

Baselines: DuckDB and `cross-product.rs` runs in ~50 min

Our Rust Implementation: `join.rs` runs in ~15 min

```
real    14m44.804s
user    14m44.398s
sys     0m0.296s
[1]+  Done                    time ./target/release/join &> join.txt
^C
[hangdong@rockhopper-06] (16)$ cat join.txt
count: 100538676537
time: 884.412510477s
```



Look into the Future

- SQL queries with `NOT EXIST (or NOT IN / EXCEPT)` clauses

  often exhibit poor performances

- Antijoins are not meant to be rigid and slow!
- Can we implement a practical, efficient ANTIJOIN operator?

Thank you!