Discussion Section 7 (CS145)

2015-11-13

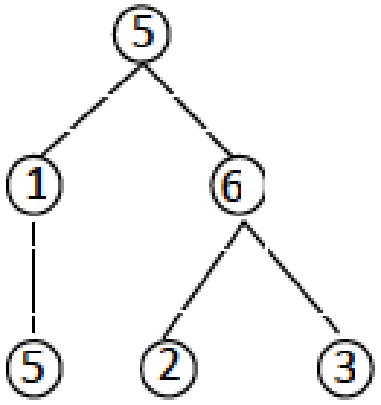
Week 07

Outline

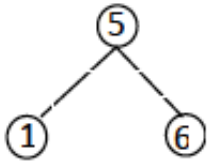
- Review:
  - Tree Mining
    - (TreeMiner)
      - Candidate generation
      - Frequency Computation

Tree Mining

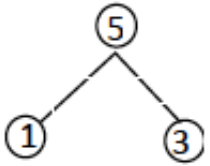
- Goal: Given a database of trees, and a minimum support, find all frequent **subtrees**
- Subtrees: Induced and Embedded Subtrees



T_1



T_2

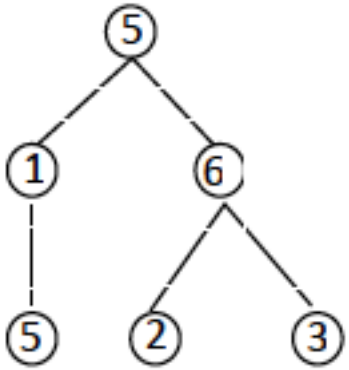


T_3

- *T2* is an induced subtree of *T1*
- *T2* is an embedded subtree of *T1*
- *T3* is an embedded subtree of *T1*
- *Note: An induced subtree is a special case of embedded subtree*

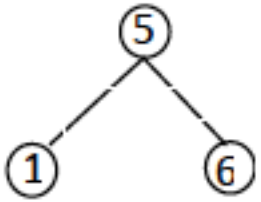
Tree Mining

- String Representation of Trees
  - DFS traversal
  - Use “-1” to represent backtrack



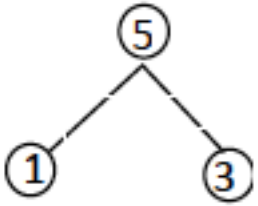
T_1

T1: 515-1-162-13-1-1



T_2

T2: 51-16-1



T_3

T3: 51-13-1

Tree Mining

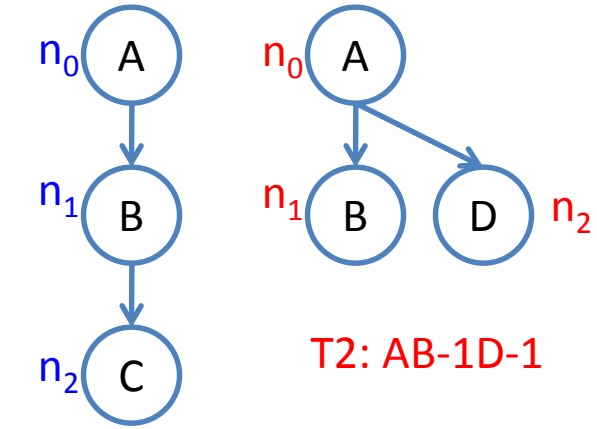
- Apriori Algorithm: TreeMiner

TREEMINER (D, *minsup*):
$F_1 = \{$ frequent 1-subtrees $\}$;
$F_2 = \{$ classes $[P]_1$ of frequent 2-subtrees $\}$;
for all $[P]_1 \in E$ **do** *Enumerate-Frequent-Subtrees*$([P]_1)$;

ENUMERATE-FREQUENT-SUBTREES$([P])$:
for each element $(x, i) \in [P]$ **do**
$[P_x] = \emptyset$;
for each element $(y, j) \in [P]$ **do**
$\mathbf{R} = \{(x, i) \otimes (y, j)\}$; Class Extension - Candidates Generation
$\mathcal{L}(\mathbf{R}) = \{\mathcal{L}(x) \cap_{\otimes} \mathcal{L}(y)\}$; Scope List Join - Frequency Computation
if for any $R \in \mathbf{R}$, R **is frequent then**
$[P_x] = [P_x] \cup \{R\}$;
Enumerate-Frequent-Subtrees$([P_x])$;

Tree Miner: Candidates Generation

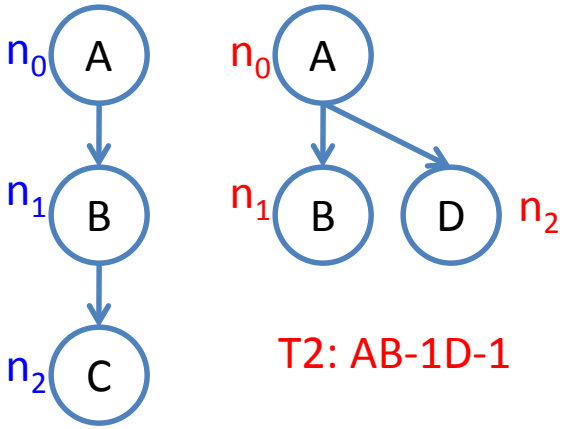
- Equivalence Class
  - Two k-subtrees are in the same class iif they share a common prefix up to (k-1) node



K = 3

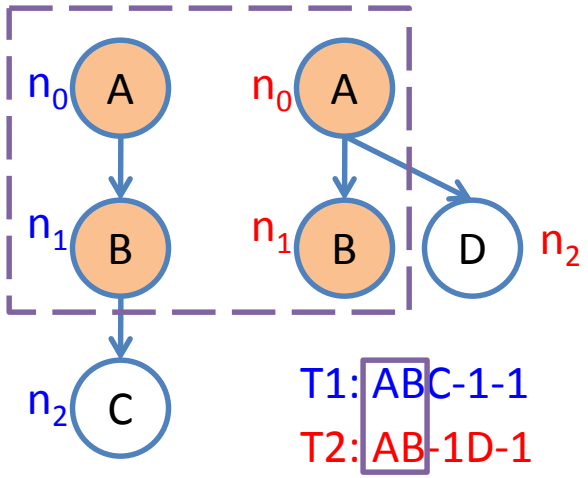
Tree Miner: Candidates Generation

- Equivalence Class
  - Two k-subtrees are in the same class iif they share a common prefix up to (k-1) node



T1: ABC-1-1

K = 3



Prefix String: AB
Element List:

(C, 1) => From T1, C attach to node 1
(D, 0) => From T2, D attach to node 0

Tree Miner: Candidates Generation

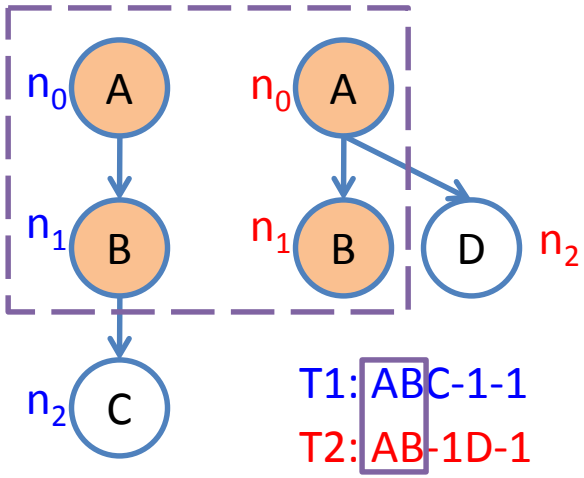
- Class Extension
  - Let
    - *P* = prefix class with encoding P
    - *(x, i)* and *(y, j)* = any two elements in the class
    - P_x = the class representing extensions of element *(x, i)*
  - Joining any two elements, *(x, i)* ⊗ *(y, j)* is defined as
    - **case I – (i = j):**

      1. If P != **Ø,** add (y, j) and (y, ni) to class [Px], where ni is the depth-first number for node (x, i) in tree Px.
      2. If P = **Ø,** add (y, j + 1) to [Px].

    - **case II – (i > j):** add (y, j) to class [Px].
    - **case III – (i < j):** no new candidate is possible in this case.

Tree Miner: Candidates Generation

- Class Extension

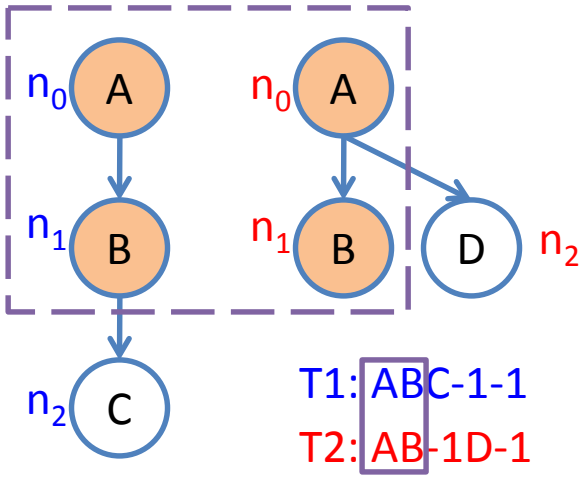


Prefix String: AB
Element List:
(C, 1) => From T1, C attach to node 1
(D, 0) => From T2, D attach to node 0

- When *x* = *C*, new candidates $[P_x]$ are:
  - (C, 1) $\otimes$ (C, 1)
  - (C, 1) $\otimes$ (D, 0)

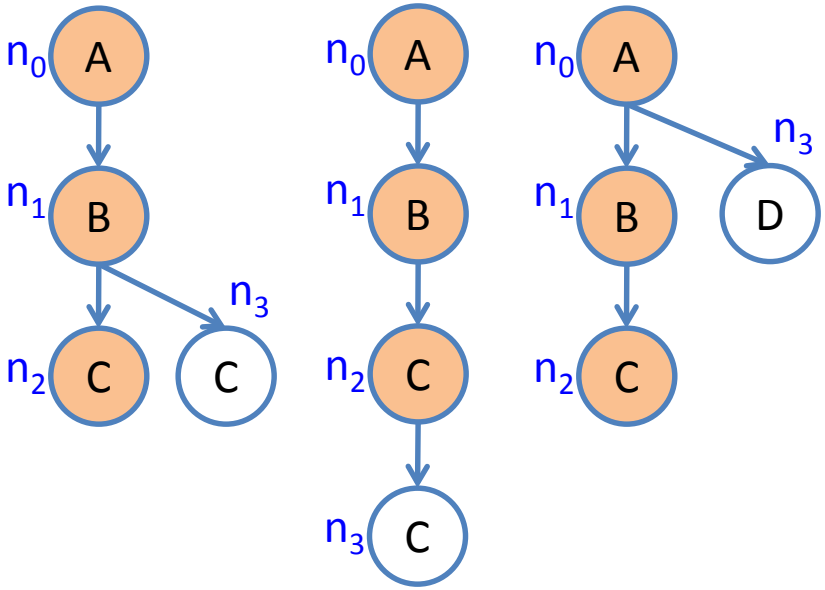
Tree Miner: Candidates Generation

- Class Extension



Prefix String: AB
Element List:
(C, 1) => From T1, C attach to node 1
(D, 0) => From T2, D attach to node 0

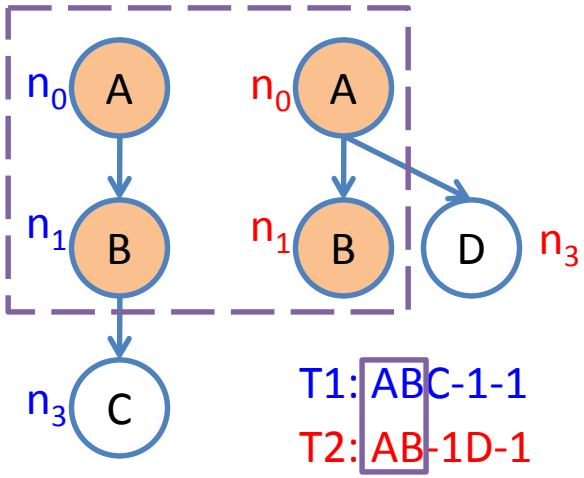
- When *x* = *C*, new candidates [P_x] are:



New Equivalence Class
Prefix String: ABC
Element List:
(C, 1), (C, 2), (D, 0)

Tree Miner: Candidates Generation

- Class Extension

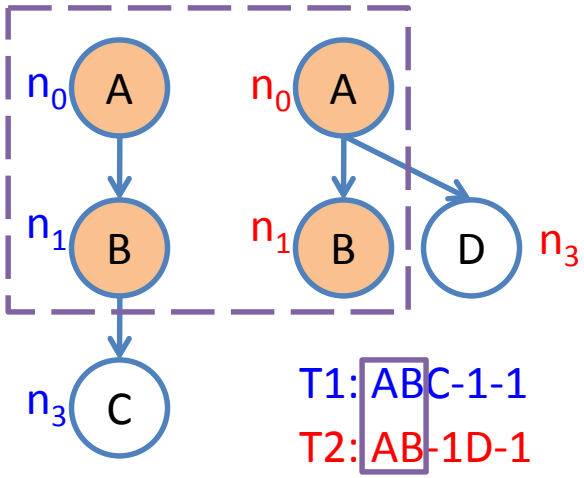


Prefix String: AB
Element List:
(C, 1) => From T1, C attach to node 1
(D, 0) => From T2, D attach to node 0

- When *x* = *D*, candidates for $[P_x]$:
  - (D, 0) $\otimes$ (C, 1)
  - (D, 0) $\otimes$ (D, 0)

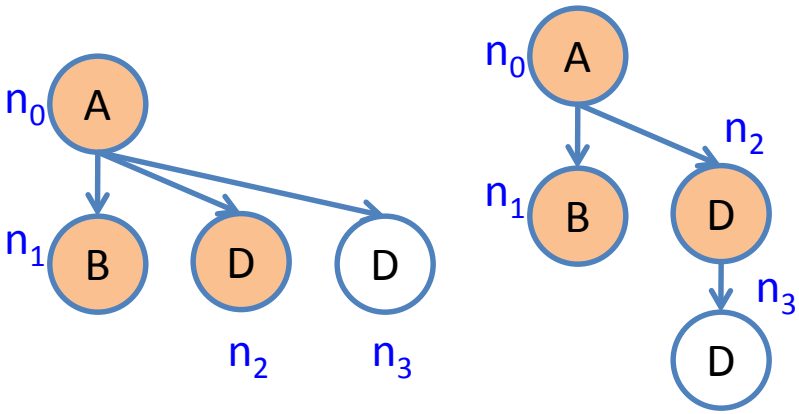
Tree Miner: Candidates Generation

- Class Extension



Prefix String: AB
Element List:
(C, 1) => From T1, C attach to node 1
(D, 0) => From T2, D attach to node 0

- When *x* = *D*, candidates for $[P_x]$:



New Equivalence Class

Prefix String: ABD
Element List:
(D, 0), (D, 2)

Tree Mining

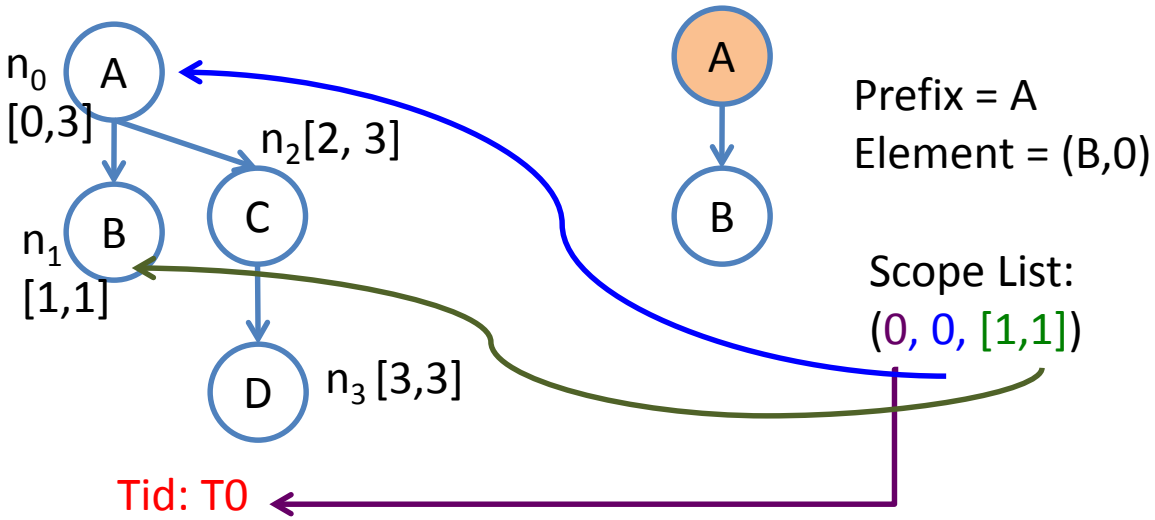
- Apriori Algorithm: TreeMiner

TREEMINER (D, *minsup*):
$F_1 = \{$ frequent 1-subtrees $\}$;
$F_2 = \{$ classes $[P]_1$ of frequent 2-subtrees $\}$;
for all $[P]_1 \in E$ **do** *Enumerate-Frequent-Subtrees*$([P]_1)$;

ENUMERATE-FREQUENT-SUBTREES$([P])$:
for each element $(x, i) \in [P]$ **do**
$[P_x] = \emptyset$;
for each element $(y, j) \in [P]$ **do**
$\mathbf{R} = \{(x, i) \otimes (y, j)\}$; Class Extension - Candidates Generation
$\mathcal{L}(\mathbf{R}) = \{\mathcal{L}(x) \cap_{\otimes} \mathcal{L}(y)\}$; **Scope List Join - Frequency Computation**
if for any $R \in \mathbf{R}$, R **is frequent then**
$[P_x] = [P_x] \cup \{R\}$;
Enumerate-Frequent-Subtrees$([P_x])$;

Tree Mining: Frequency Computation

- Scope
  - Scope for each node can be represented as (i, j)
    - i = the node id of itself
    - j = the last node id it can reach
- Scope List
  - Scope list of each element of a class can be represented as *(t, m, s)*
    - t = tree id,
    - m = matched label of the prefix,
    - s = scope of the last item



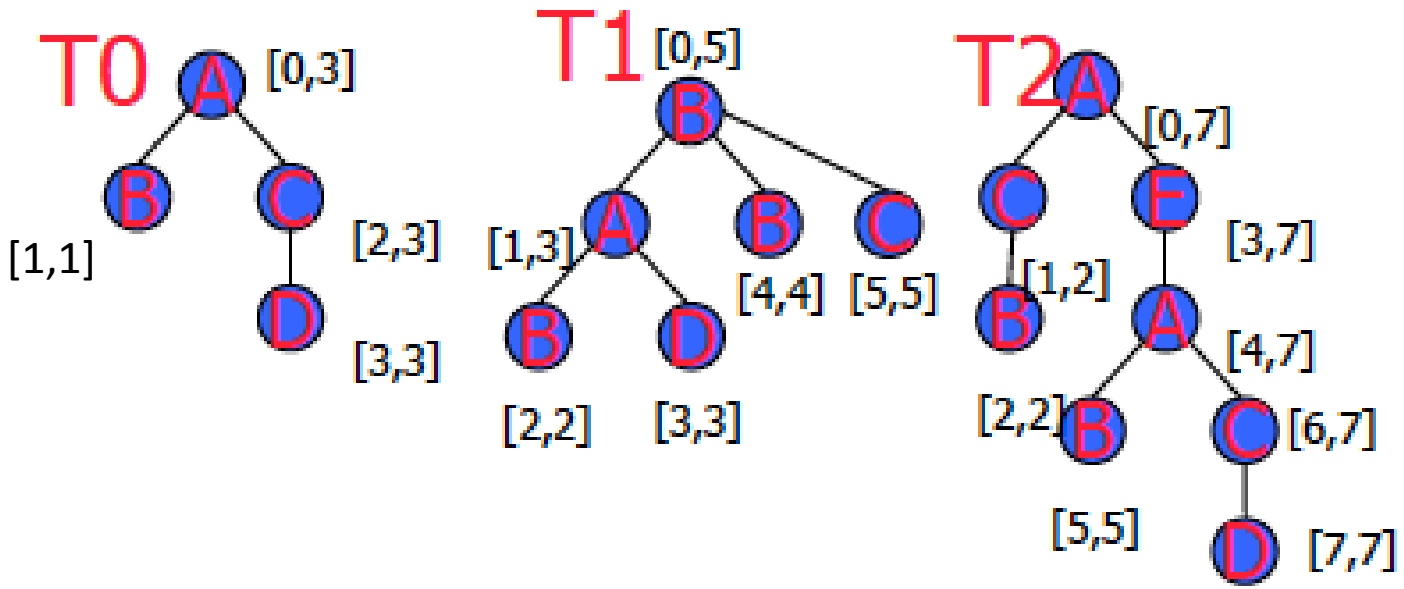
Tree Mining: Frequency Computation

- Joining (x, i) and (y, j)
- In-Scope Test
  - Add (y, j+1) when i = j
  - Add y as a child of x
  - Check whether there exist (t_x, m_x, s_x) and (t_y, m_y, s_y) s.t.
    - $t_x = t_y$
    - $m_x = m_y$
    - $s_y \subseteq s_x$,

- Joining (x, i) and (y, j)
- Out-Scope Test
  - Add (y, j)
  - Add y as a sibling of x
  - Check whether there exist (t_x, m_x, s_x) and (t_y, m_y, s_y) s.t.
    - $t_x = t_y$
    - $m_x = m_y$
    - $s_y > s_x$,

Tree Mining: Frequency Computation

- Example

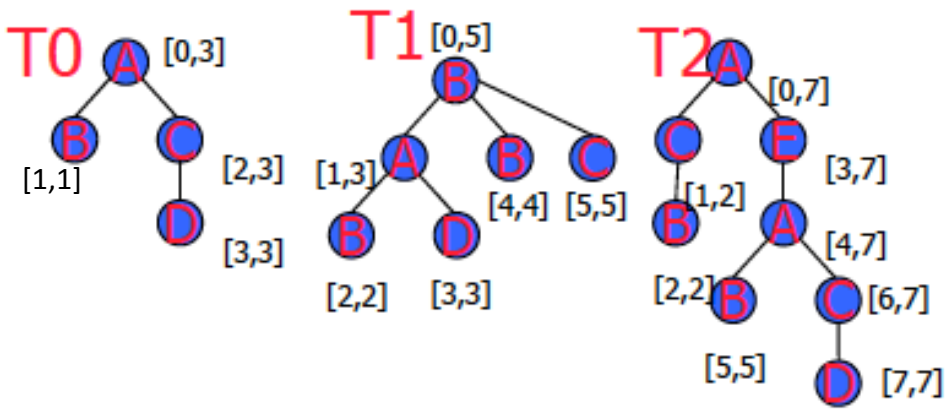


Min_support = 3 (100%)

Tree Mining: Frequency Computation

- Example

Min_support = 3 (100%)



Scope Lists:

A	B	C	D	E
0, [0,3]	0, [1,1]	0, [2,3]	0, [3,3]	2, [3,7]
1, [1,3]	1, [0,5]	1, [5,5]	1, [3,3]	
2, [0,7]	1, [2,2]	2, [1,2]	2, [7,7]	
2, [4,7]	1, [4,4]	2, [6,7]		
	2, [2,2]			
	2, [5,5]			

Not frequent

Equivalence Class

Prefix String: ∅

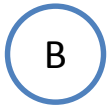
Element List:

(A, -1), (B, -1), (C, -1), (D, -1)

Tree Mining: Frequency Computation

- Example

0, [0,3]
1, [1,3]
2, [0,7]
2, [4,7]



0, [1,1]
1, [0,5]
1, [2,2]
1, [4,4]
2, [2,2]
2, [5,5]

0, [2,3]
1, [5,5]
2, [1,2]
2, [6,7]

0, [3,3]
1, [3,3]
2, [7,7]

Equivalence Class

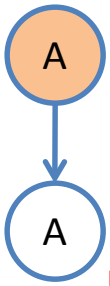
Prefix String: ∅
Element List:
(A, -1), (B, -1), (C, -1), (D, -1)

Min_support = 3 (100%)

2-subtrees candidates:

For x = A, y = A, B, C, D; $(x, i) \otimes (y, j)$

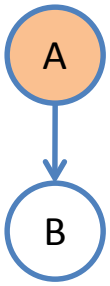
(A, -1) ⊗ (A, -1)



2, 0, [4,7]

In-Scope Test

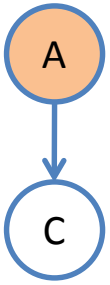
(A, -1) ⊗ (B, -1)



0, 0, [1,1]
1, 1, [2,2]
2, 0, [2,2]
2, 0, [5,5]
2, 4, [5,5]

In-Scope Test

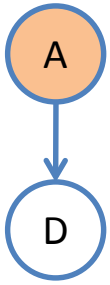
(A, -1) ⊗ (C, -1)



0, 0, [2,3]
2, 0, [1,2]
2, 0, [6,7]
2, 4, [6,7]

In-Scope Test

(A, -1) ⊗ (D, -1)



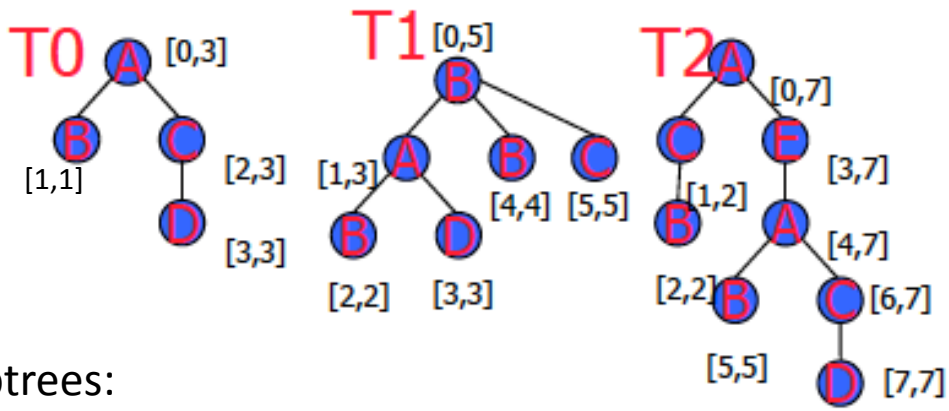
0, 0, [3,3]
1, 1, [3,3]
2, 0, [7,7]
2, 4, [7,7]

In-Scope Test

Tree Mining: Frequency Computation

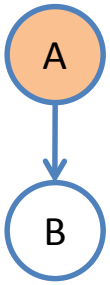
- Example

Min_support = 3 (100%)



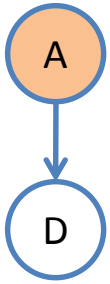
Frequent 2-subtrees:

(A, -1) ⊗ (B, -1)



0, 0, [1,1]
1, 1, [2,2]
2, 0, [2,2]
2, 0, [5,5]
2, 4, [5,5]

(A, -1) ⊗ (D, -1)



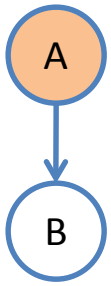
0, 0, [3,3]
1, 1, [3,3]
2, 0, [7,7]
2, 4, [7,7]

Equivalence Class

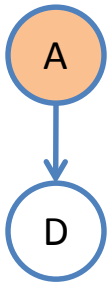
Prefix String: A
Element List: (B, 0), (D, 0)

Tree Mining: Frequency Computation

- Example



0, 0, [1,1]
1, 1, [2,2]
2, 0, [2,2]
2, 0, [5,5]
2, 4, [5,5]



0, 0, [3,3]
1, 1, [3,3]
2, 0, [7,7]
2, 4, [7,7]

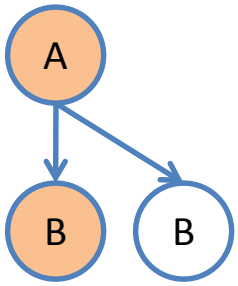
Equivalence Class

Prefix String: A
Element List: (B, 0), (D, 0)

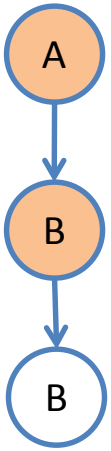
3-subtrees candidates:

For x = B, y = B,D; $(x, i) \otimes (y, j)$

(B, 0) ⊗ (B, 0)



2, 02, [5,5]

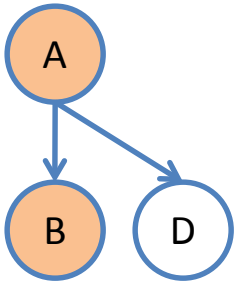


Ø

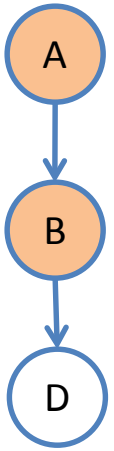
Out-Scope Test

In-Scope Test

(B, 0) ⊗ (D, 0)



0, 01, [3,3]
1, 12, [3,3]
2, 02, [7,7]
2, 05, [7,7]
2, 45, [7,7]



Ø

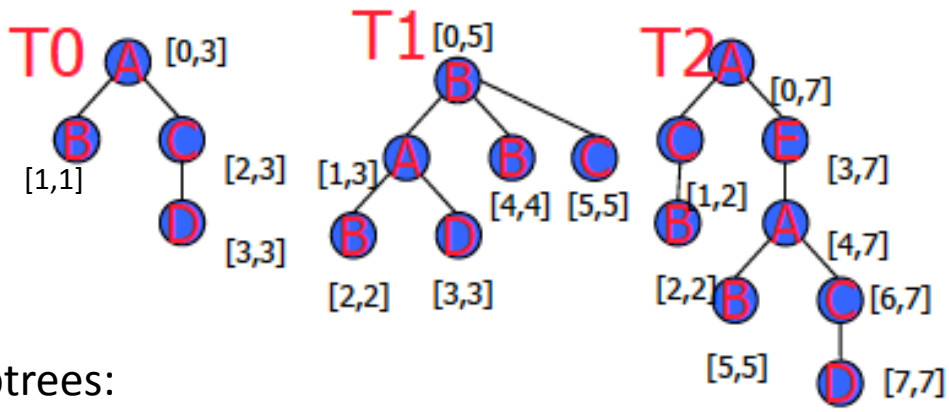
Out-Scope Test

In-Scope Test

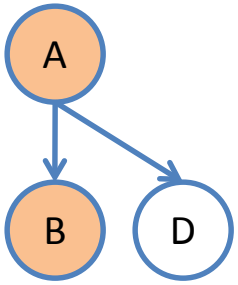
Tree Mining: Frequency Computation

- Example

Min_support = 3 (100%)



Frequent 3-subtrees:



0, 01, [3,3]
1, 12, [3,3]
2, 02, [7,7]
2, 05, [7,7]
2, 45, [7,7]

Equivalence Class

Prefix String: AB
Element List: (D, 0)