# Compact Complete Inverted Files for Texts and Directed Acyclic Graphs Based on Sequence Binary Decision Diagram

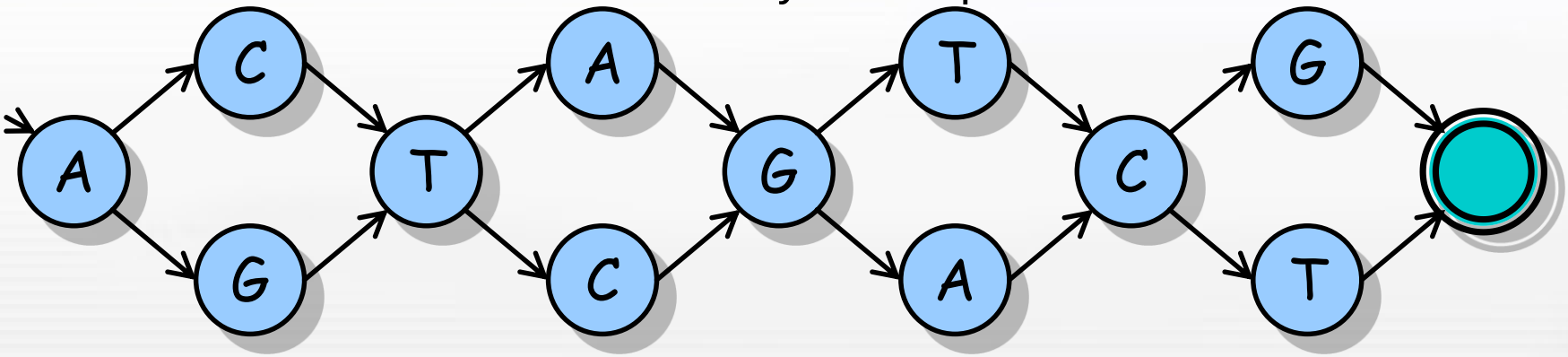
Shuhei Denzumi[1], Koji Tsuda[2, 3], Shin-ichi Minato[1,2], and Hiroki Arimura[1]

1) Hokkaido University
2) JST ERATO Minato Discrete Structure Manipulation System Project
3) AIST CBRC

# Background

- Text index is an important data structure for sequence mining
  - Many text indices have been proposed
  - Most of their inputs are ordinary linear text

- Index for directed acyclic graph (DAG)
  - Regular expression without infinite repeat, genome with mutations
  - DAG can represent huge number of strings
  - Construct indices after expanding it is nonsense
  - We want to make indices directly from input DAGs



# Complete Inverted Files

- Factor automata of automata [Mohri et al., 2007]
  - Automata for all factors of strings represented by a given automata
  - Determine whether a pattern occurs as a factor or not

- Complete Inverted Files
  - find(p): Return TRUE if a pattern p occurs as a substring of the input
  - freq(p): Frequency of p in the input
  - locate(p): Positions of occurrences of p in the input

- General Compressed Suffix Array (GCSA) [Siren et al., 2011]
  - Complete inverted file for DAG
  - Very compact because of using succinct data structure
  - Require special property for the input DAG

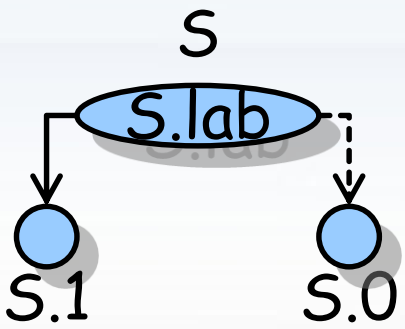
# Sequence BDD (SeqBDD)

- Sequence Binary Decision Diagram [Loekito et al., 2009]
  - Acyclic graphs for finite sets of strings
  - One kind of binarized automata
  - Member of Binary Decision Diagram (BDD) family
  - BDD is a graph structure for Boolean functions

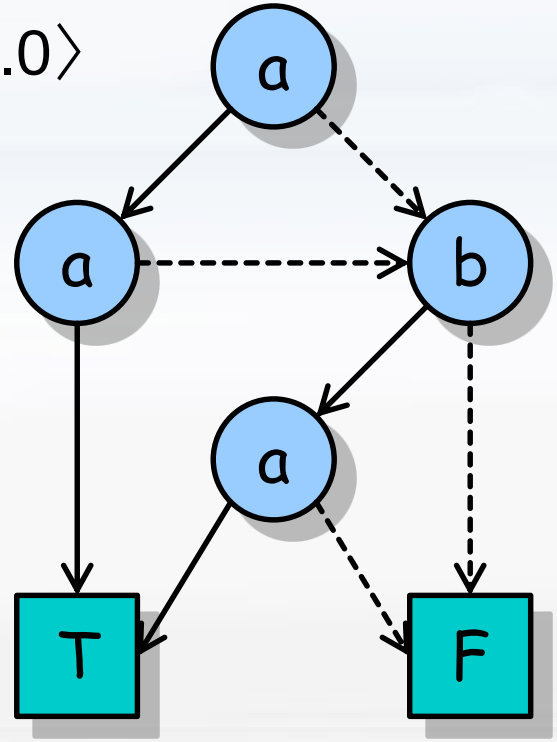
- Characteristic
  - Using hash tables
  - Automatically share all equivalent subgraphs
  - String set operations by simple recursive algorithms
    - Union, Intersection, Difference, …
    - Enumerate all prefixes, suffixes, substrings, and subsequences
    - Analyzing time/space complexity is difficult

# Definition

- Σ: alphabet (totally ordered by $\prec$)
- Internal node: ⓐ, ⓑ, …, ⓩ, T/F - terminal node: T / F
- 1/0 - edge: ⟶ / ⇢
- SeqBDD: directed acyclic graph
- Internal node S, $\tau(S) \mapsto \langle S.lab, S.1, S.0 \rangle$
  - S.lab: label
  - S.1: 1-child
  - S.0: 0-child



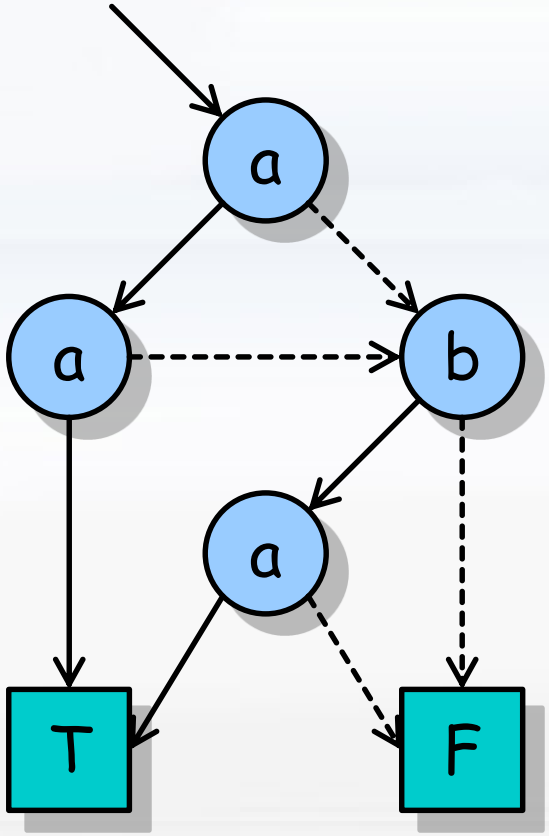

- Ordering rule
  - N.lab $\prec$ (N.0).lab

# Semantics

- A path from the root node to the True-terminal node correspond to a strings in the set that SeqBDD represent

- Example: SeqBDD for {aa, aba, ba}




- Each edge has different meaning
  - 1-edge means choosing the symbol
  - 0-edge means ignoring the symbol
  - Remember 0-ordering rule

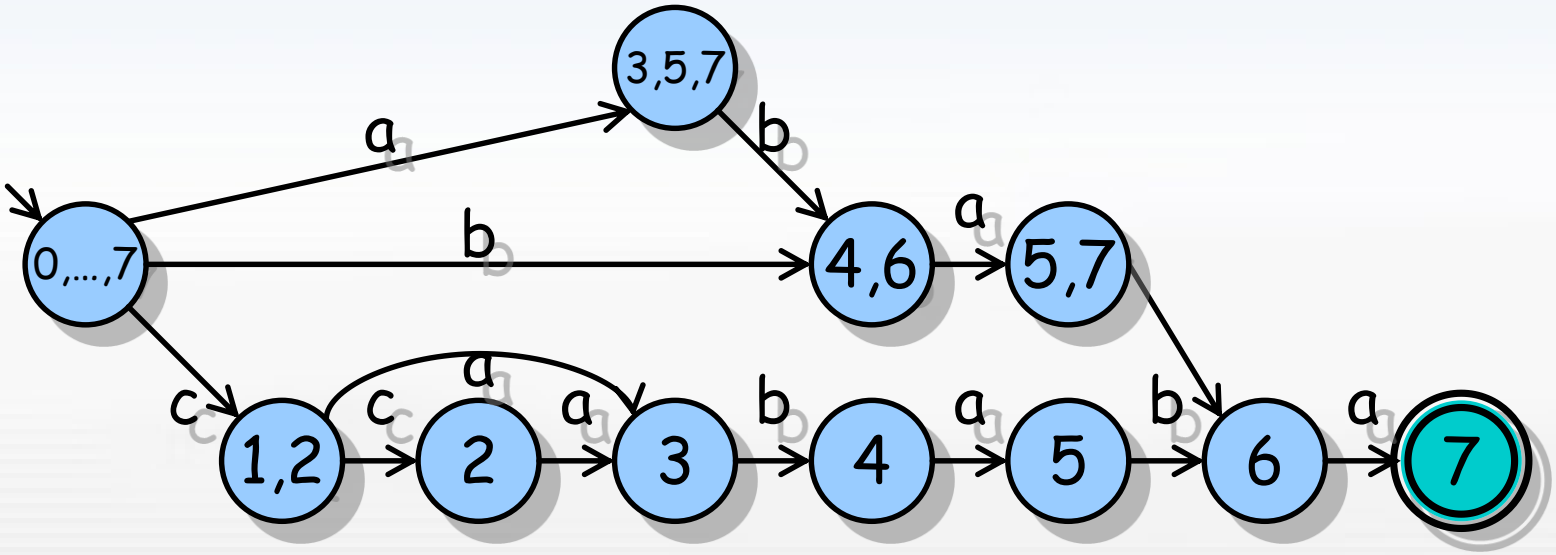
- Comparison to automata
  - T-terminal is finite state
  - F-terminal is garbage state

# Index for text

# DAWG

- We use end positions (In *ccababa*, *aba* occurs at 5 and 7)
- Directed Word Acyclic Graph (DAWG) [Blumer et al., 1987]
  - Complete inverted file for text
  - Each DAWG node has occurrence information

- DAWG for a text *ccababa*

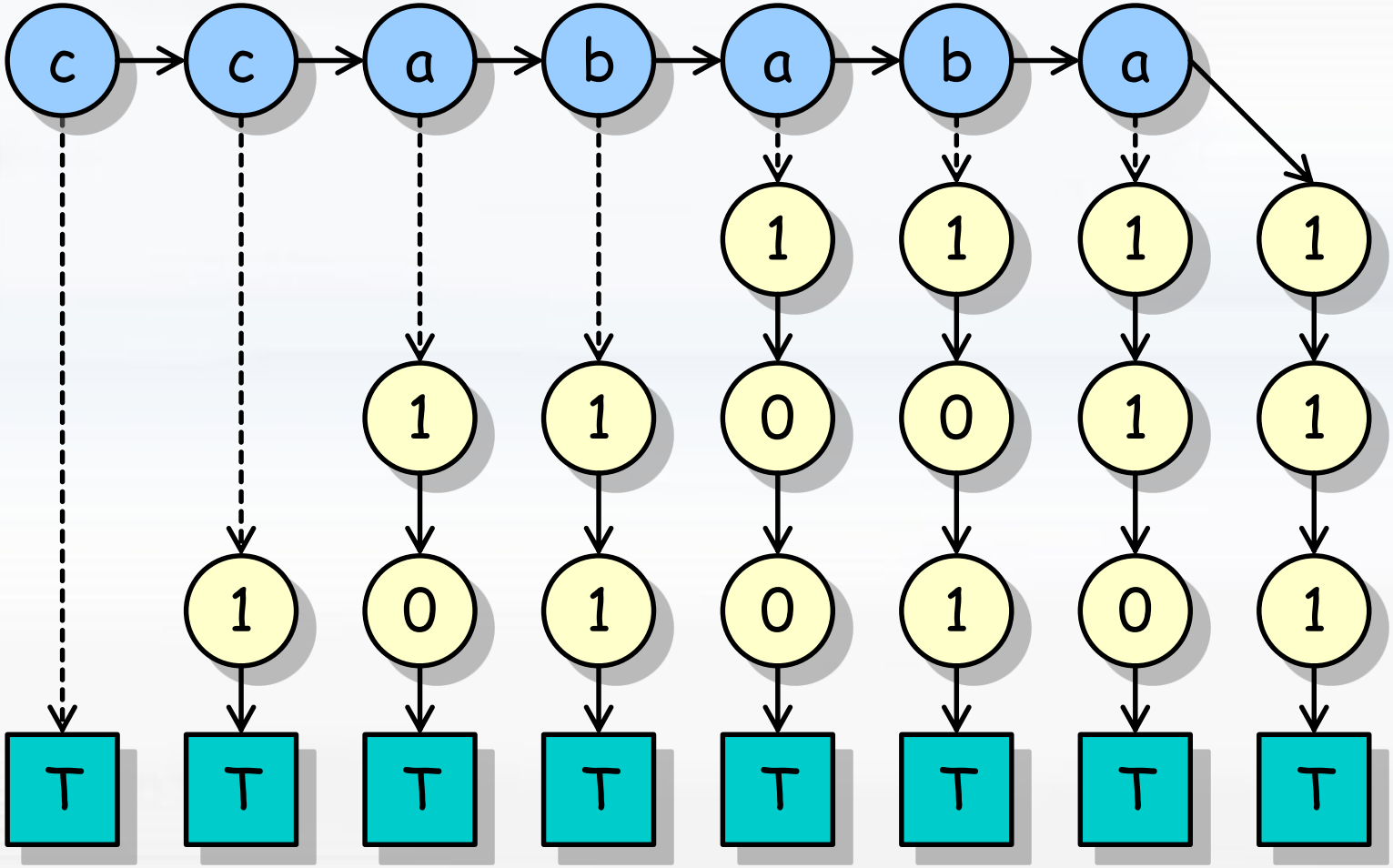


# Main idea

- Consider integers as symbols in the alphabet
  - We can deal with positions as a string
  - If *aba* occurs at 5 and 7, the index include *aba5* and *aba7* as strings
- Size complexity of DAG index is not analyzed
  - Size of DAWG is linear
  - However, size of occurrence information of DAG index is unknown
  - We want to reduce the size
- Use binary representation of integers
  - SeqBDD can share equivalent subgraphs automatically
  - Lists of raw integers are difficult to be shared
  - We define a new alphabet $\Sigma_b=\Sigma\cup\{0, 1\}$, $\forall a\in\Sigma, a \prec 0 \prec 1$.

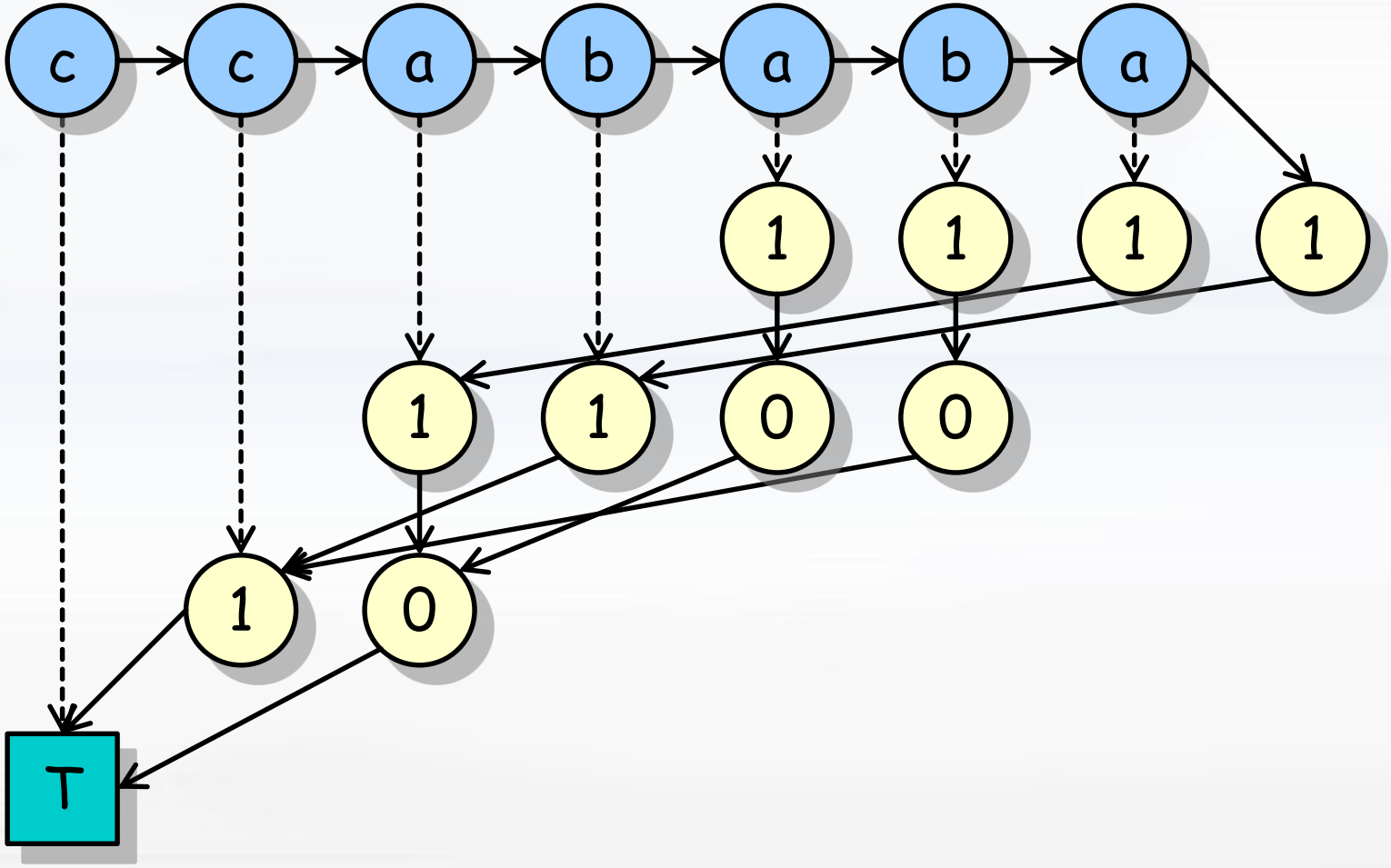
# Example

- Input text: *ccababa*



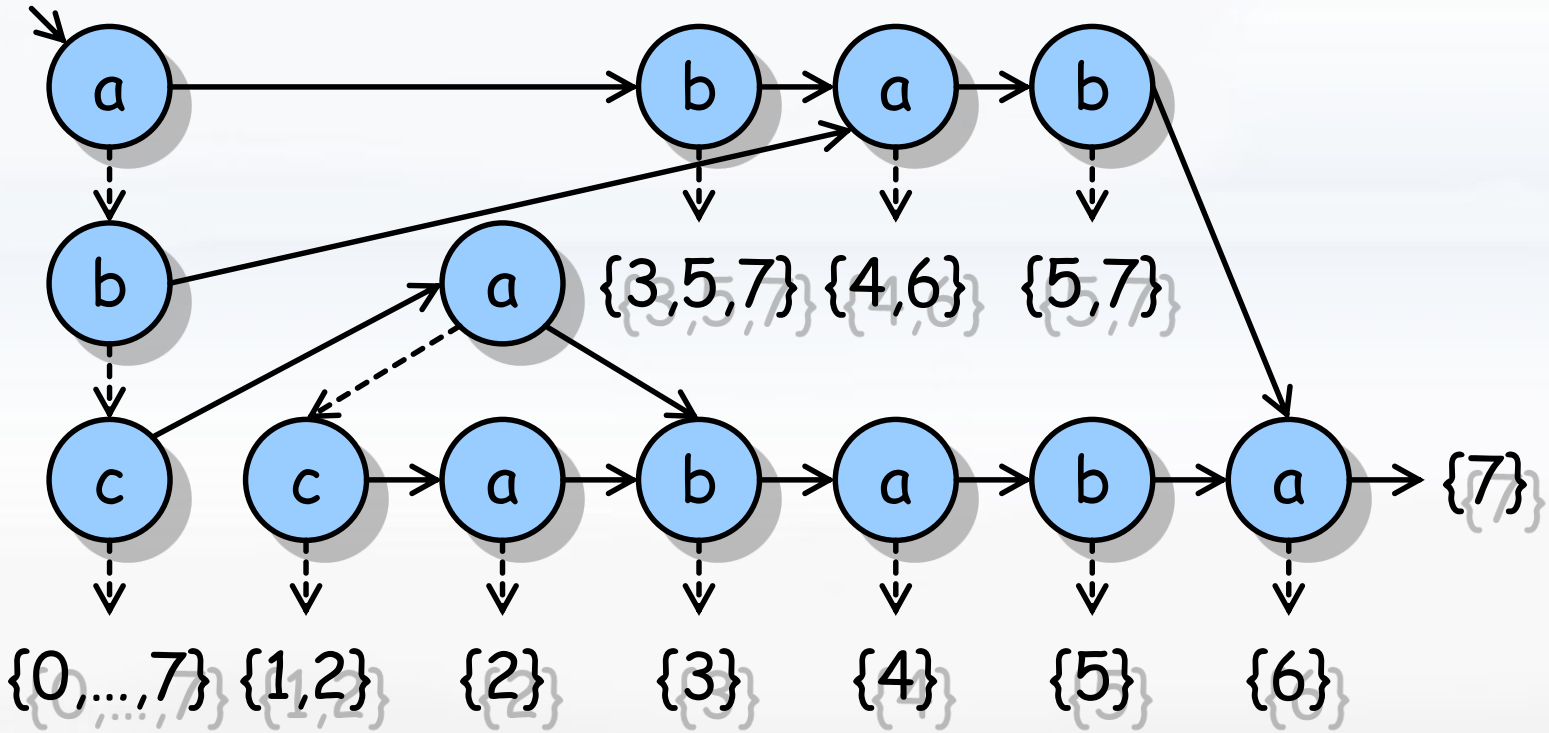
# Example

- Input text: *ccababa*



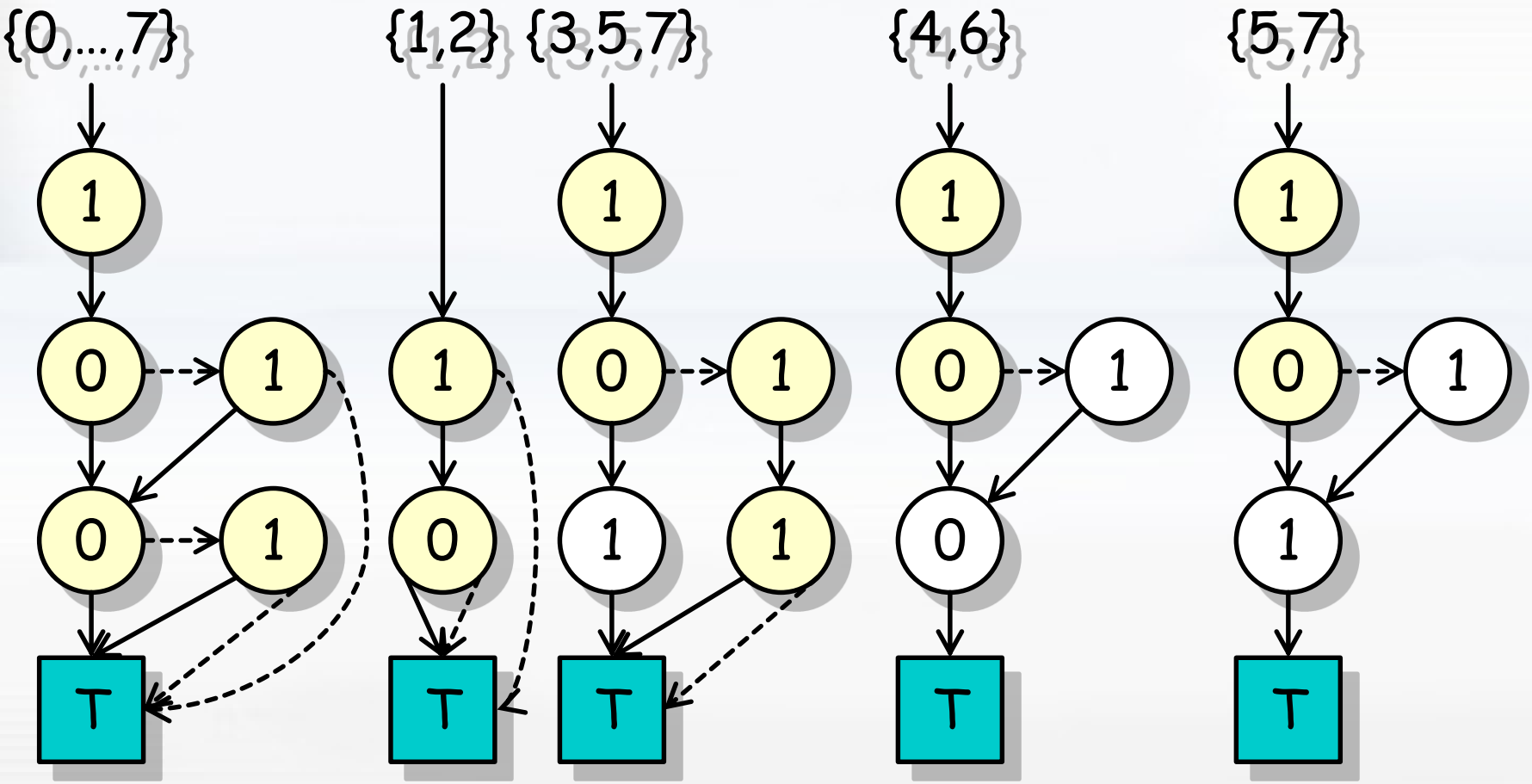
# Example: text index

- Input text: *ccababa*



# Example: integer part

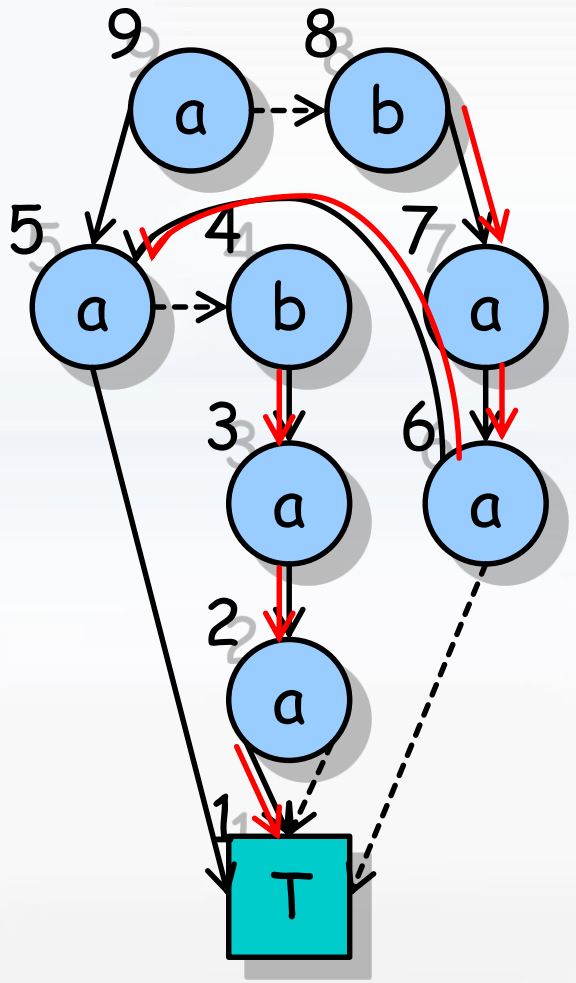
- Input text: *ccababa*



# Index for DAG

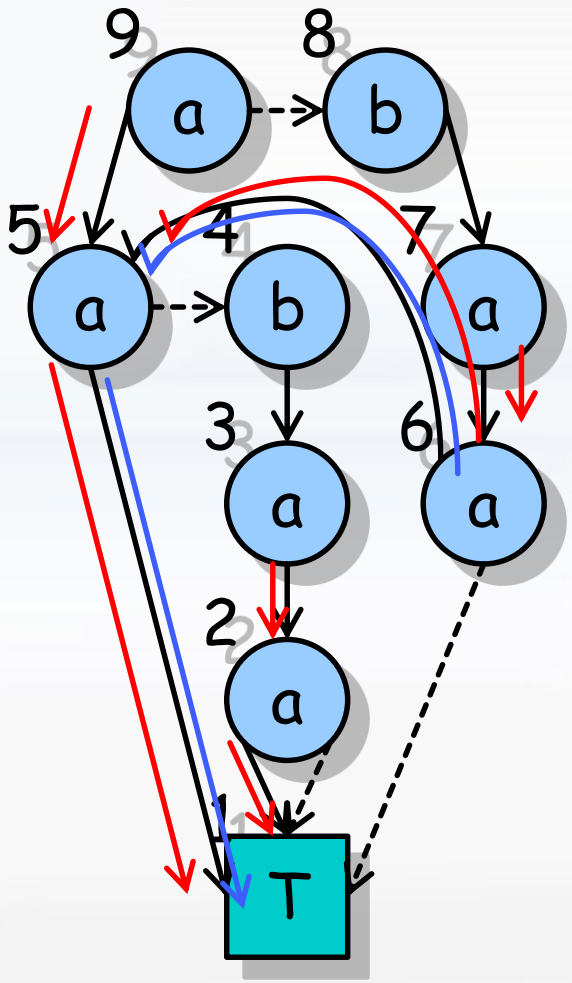
# Definition of location

- Each node has unique node ID
  - Use node IDs as well as positions
- For a pattern p
  - There are some paths that represent p
  - Location of p is
    the end nodes of the paths
  - Frequency of p is
    the number of such end nodes
- Example
  - baa: 2 paths and 2 end nodes
  - aa: 4 paths and 2 end nodes



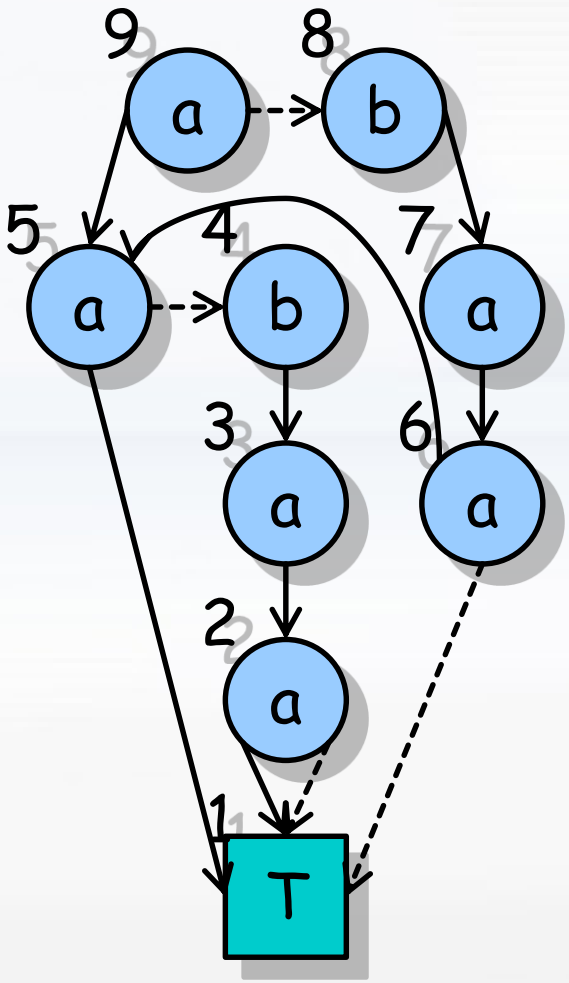
# Definition of location

- Each node has unique node ID
  - Use node IDs as well as positions
- For a pattern p
  - There are some paths that represent p
  - Location of p is
    the end nodes of the paths
  - Frequency of p is
    the number of such end nodes
- Example
  - baa: 2 paths and 2 end nodes
  - aa: 4 paths and 2 end nodes



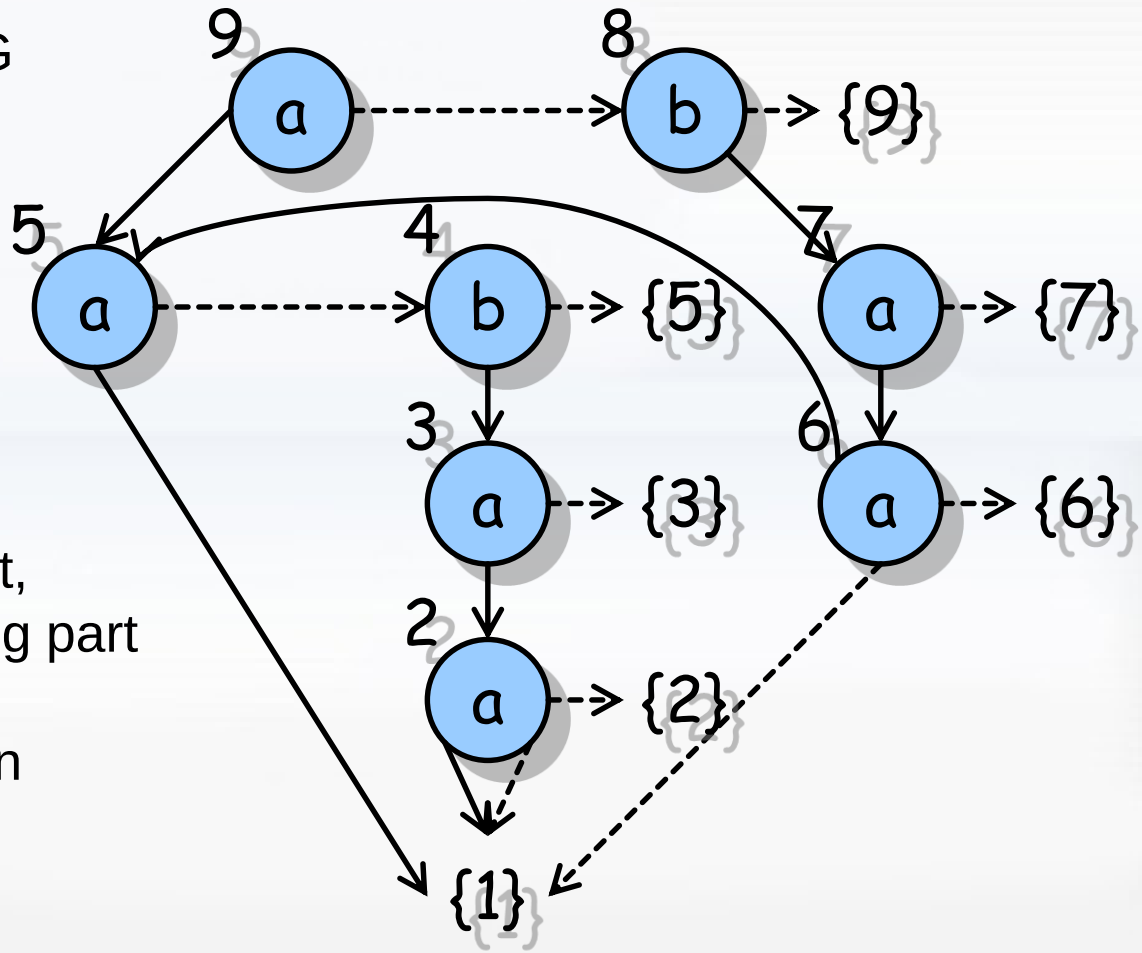
# Construct DAG index

- Compute union of all nodes
  - All substrings in a DAG are the union of sets of strings that are represented by all nodes
- Append node ID
  - As well as ordinary text, we append binary string part to each node



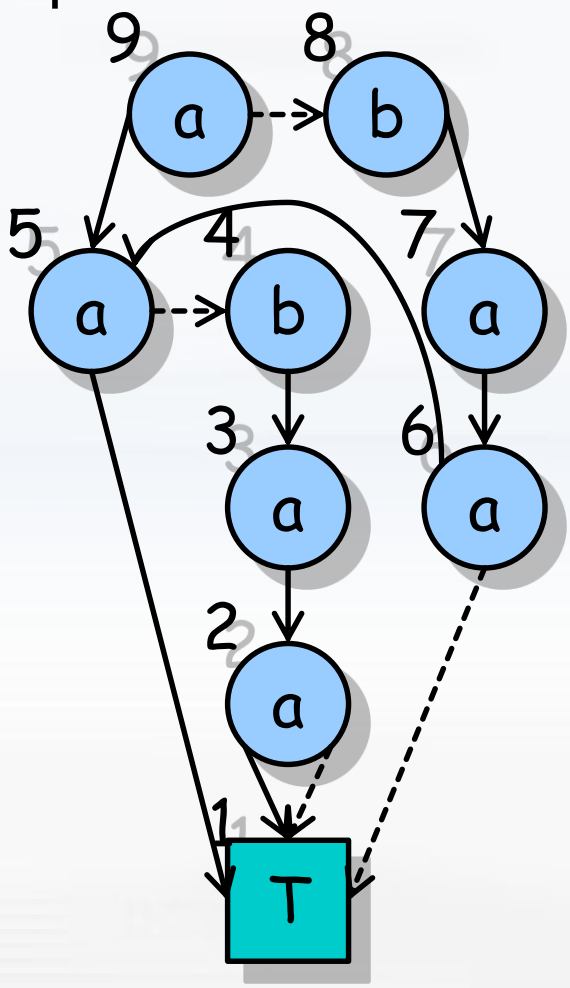
# Construct DAG index

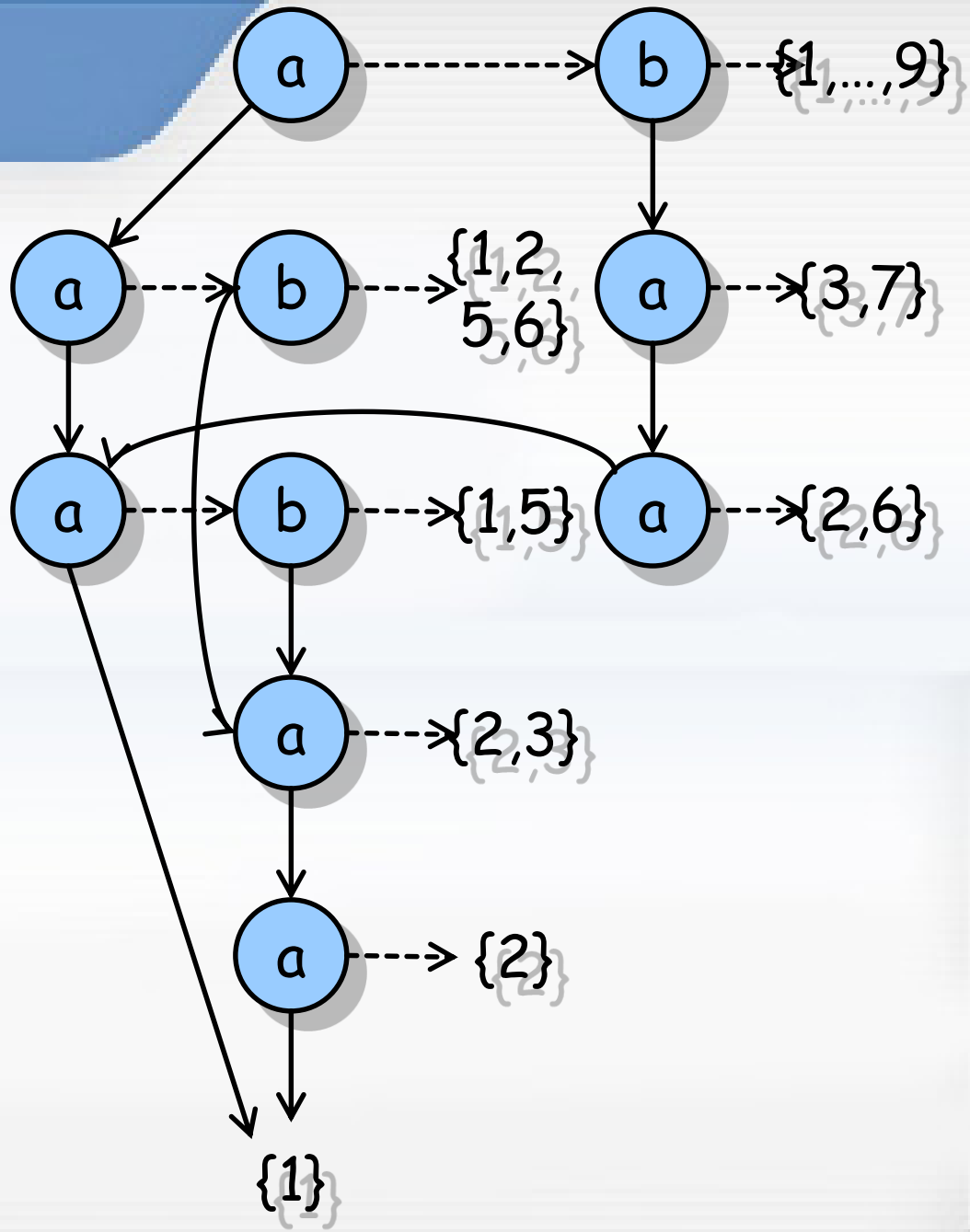
- Compute union of all nodes
  - All substrings in a DAG are the union of sets of strings that are represented by all nodes
- Append node ID
  - As well as ordinary text, we append binary string part to each node before computing union



# Example

- Input:



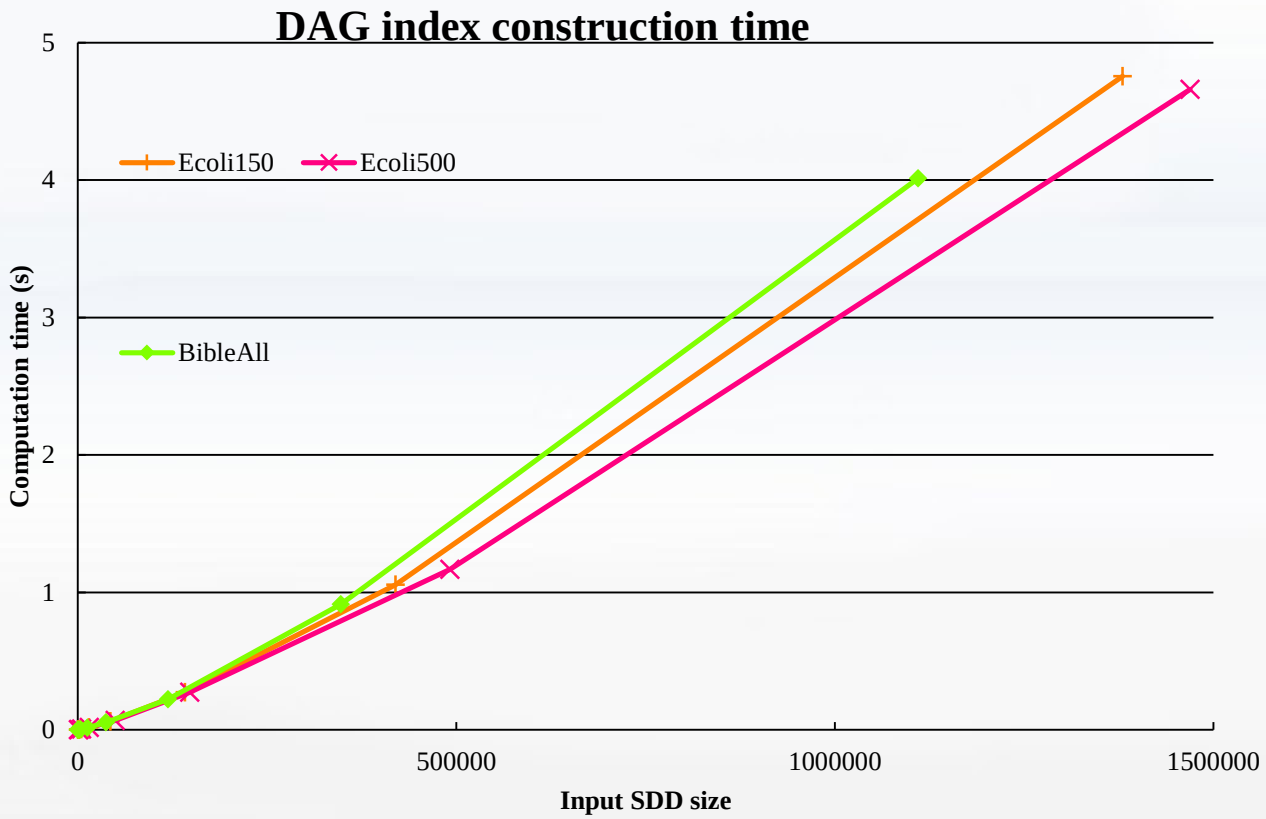
# Experiment

# Setting

- Data sets
  - SeqBDD for word bigrams from bible.txt
  - SeqBDD for genome sequences truncated from original sequence every 150/500 symbols
- Environment
  - 3.1 GHz Intel Xeon CPU
  - 1 TB DDR2 memory
- On SAPPOROBDD package [unpublished]
  - Each node use 50 ~ 55 bytes including space for hash tables
  - Construct DAG index for given SeqBDDs

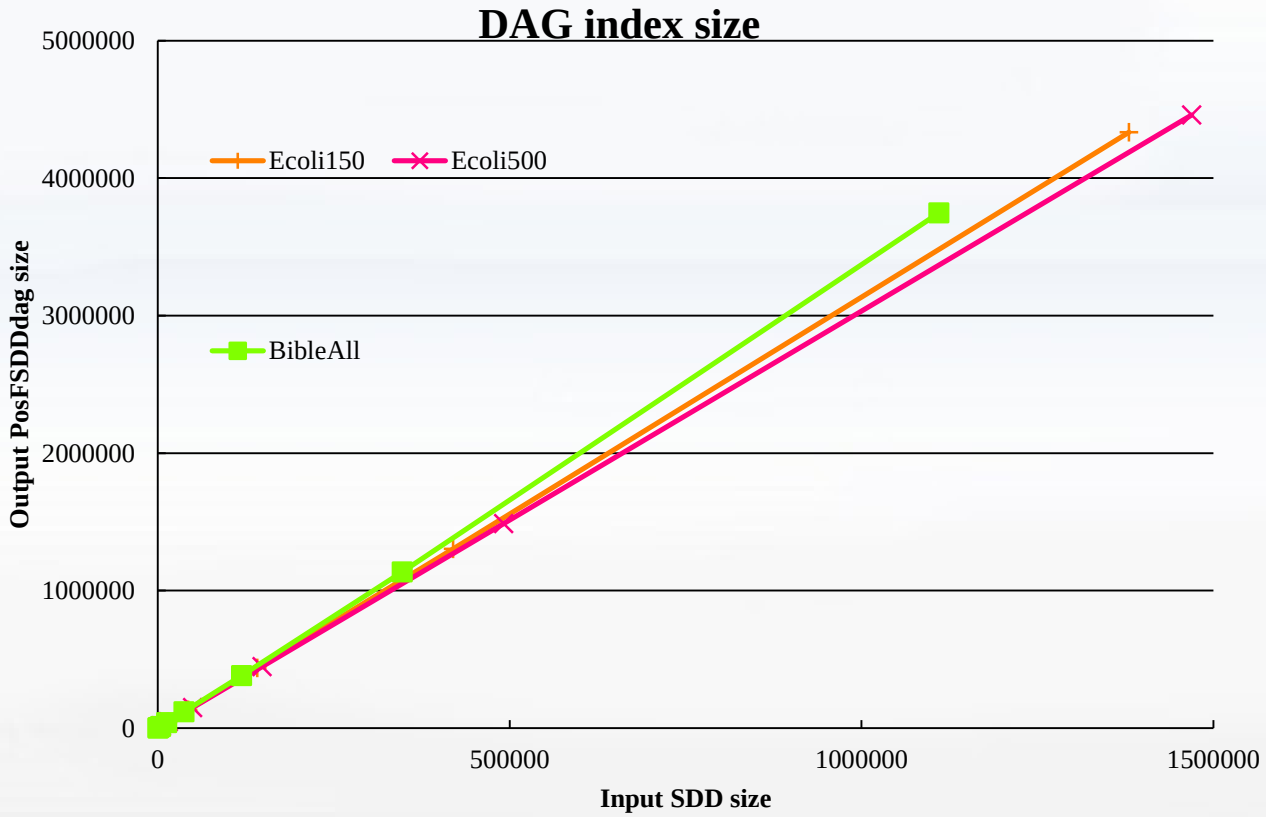
# Construction time

- Looks O(n log n) time
  - Roughly three seconds per 1 million input nodes



# Index size

- Linear or O(n log n) order
  - Three times larger than the input SeqBDDs



# Conclusion

- Complete inverted files for directed acyclic graph
  - Manipulate occurrence information as strings
  - Use binary representation for integers
  - Construct on Sequence BDD environment
  - Share equivalent subgraphs

- Result
  - Time complexity of construction looks like O(n log n)
  - Space complexity of index is linear or O

- Future work
  - Use more precise definition of frequency and location
  - Compare with other data structures

# Thank you!