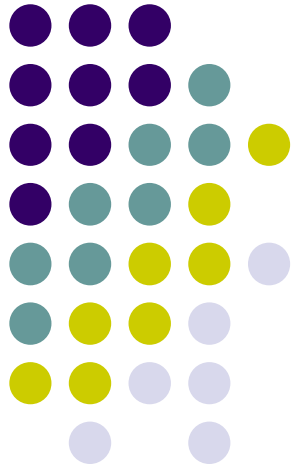
Early Profile Pruning on XML-Aware Publish-Subscribe Systems

Authors:
Mirella M. Moro
Petko Bakalov
Vassilis J. Tsotras

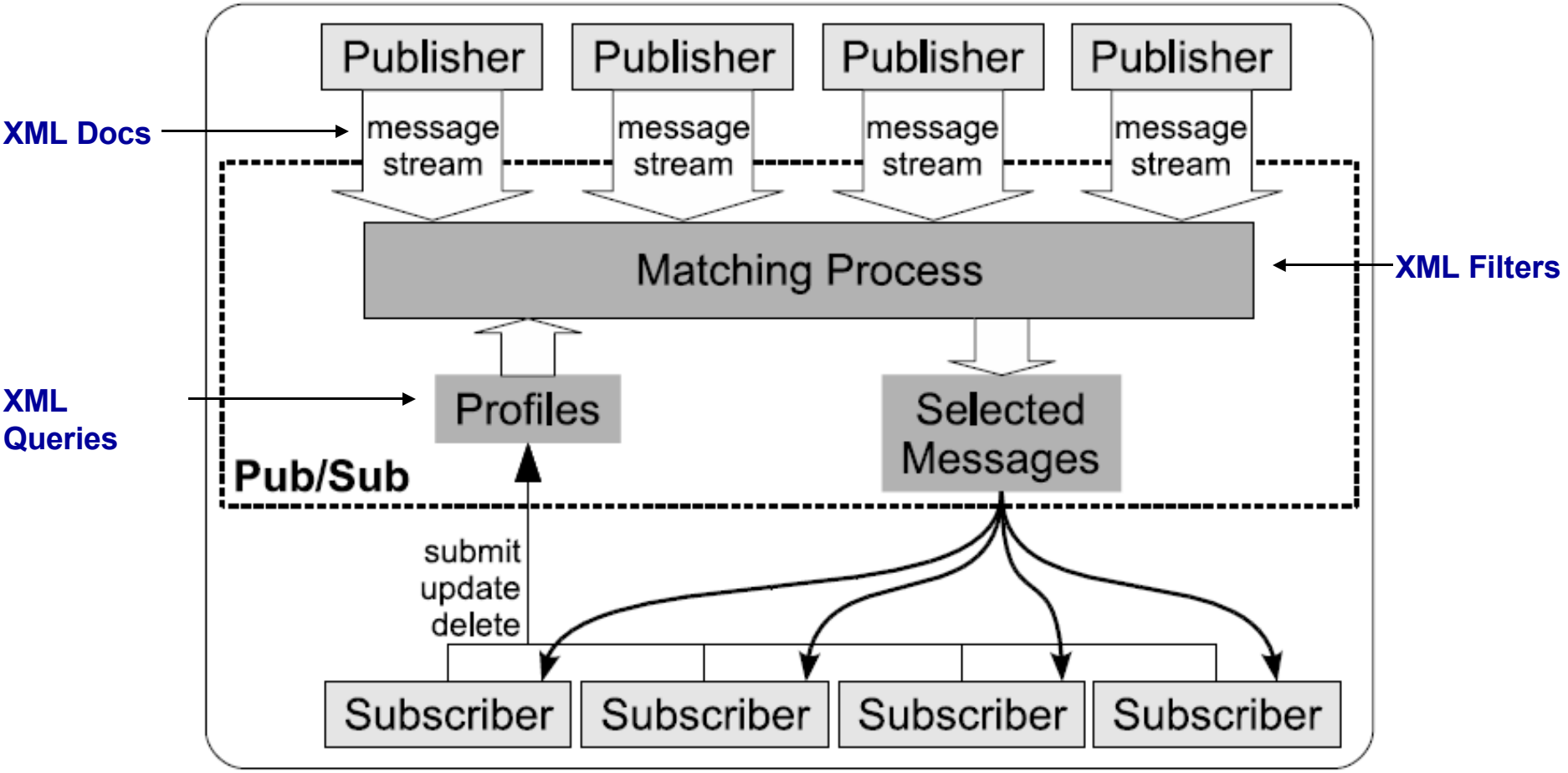


Presented by:
Sanjay Kulhari

Outline

- Publish-Subscribe Systems
- Bottom-Up Filtering FSM (BUFF)
- Bounding-Based XML Filtering (BoXFilter)
- Experimental Results
- Conclusion

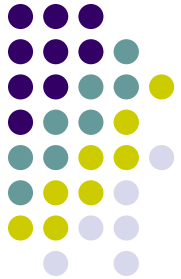
Architecture of a Pub-Sub System



Filter Strategies for XML Pub-Sub

- Relational Model
  - Join operations on relational models.
- Aggregate XML Queries using indexing
  - Sequence matching (FiST)
    - Bottom up evaluation of both queries and document.
    - More selective elements at leaf nodes.
- Finite state machines (X-Filter, Y-Filter)
  - Process common parts of queries at once.
  - Typically top-down fashion.

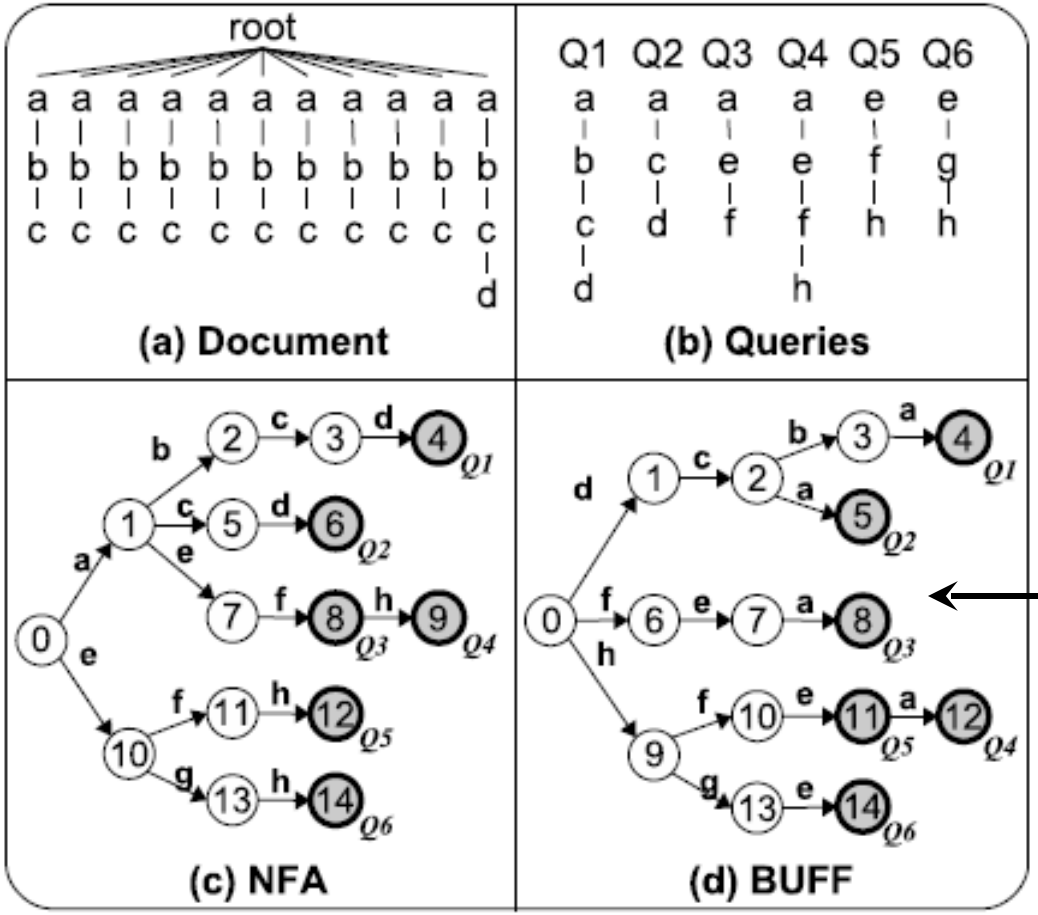
Need for new strategy



- Provision of *early pruning* to discard queries that are bound not to match any documents.

- *Early pruning* will save processing time on the matching process because it reduces the number of queries to be considered.

Bottom-Up Filtering FSM (BUFF)



//a/b/c//d (NFA)

//d//c/b/a (BUFF)

Groups queries according to common suffixes

BUFF Processing algorithm

Algorithm 2 BUFF processing

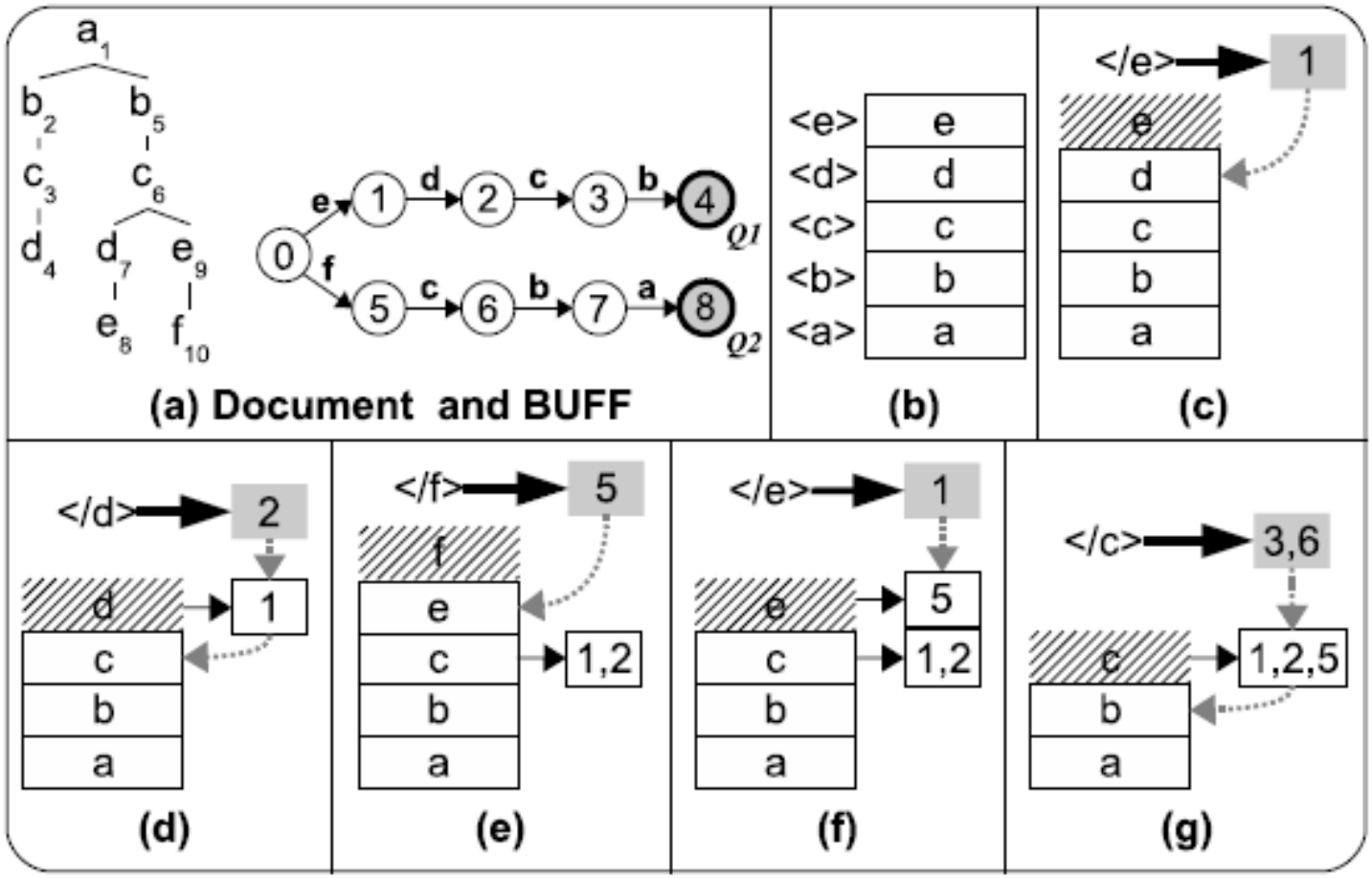
Require: D: document, B: BUFF
Ensure: The set of profiles S which are satisfied by D

```
1:  saxParser.parse(D)
2:  forward D to the subscriber of each query within S
3:  procedure STARTELEMENT        ▷ SAX parameters omitted for clarity
4:      curList ← B.initialState
5:      RS.push(node, curList)            ▷ node is the element read
6:  procedure ENDELEMENT          ▷ SAX parameters omitted for clarity
7:      curList ← RS.top().stateList
8:      newList ← B.move(node, curList)          ▷ BUFF transition
9:      RS.pop()
10:     RS.top().stateList.add(newList)
11: procedure MOVE(node,stlist)          ▷ BUFF machine transitions
12:     for all states s ∈ stlist do
13:         newState ← B.nextState(node, s)
14:         result.add(newState)
15:         if newState is final state then S.add(queriesIdentifiers)
16:     return result
```

SM is triggered for closing tag

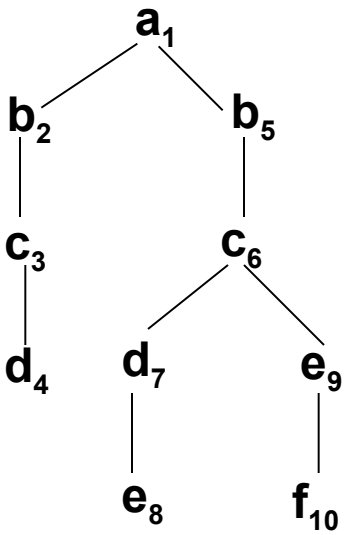
List of states per element

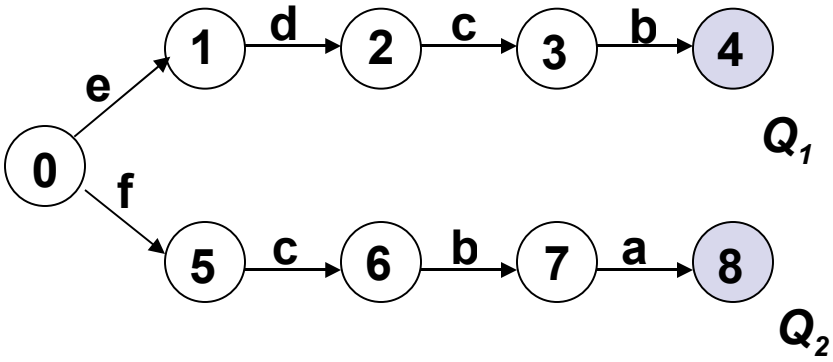
Example of BUFF matching

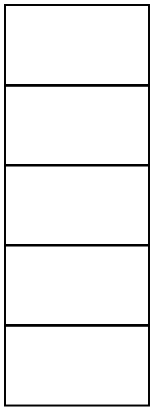


BUFF matching (Runtime Stack)

Initial stage







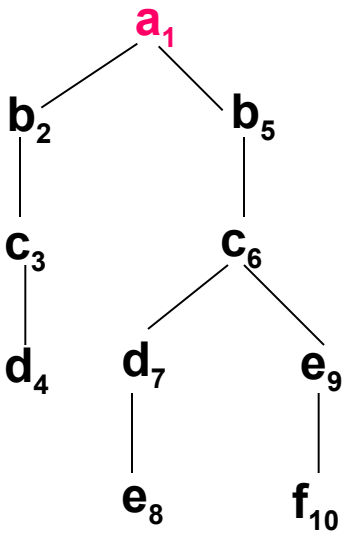
Document

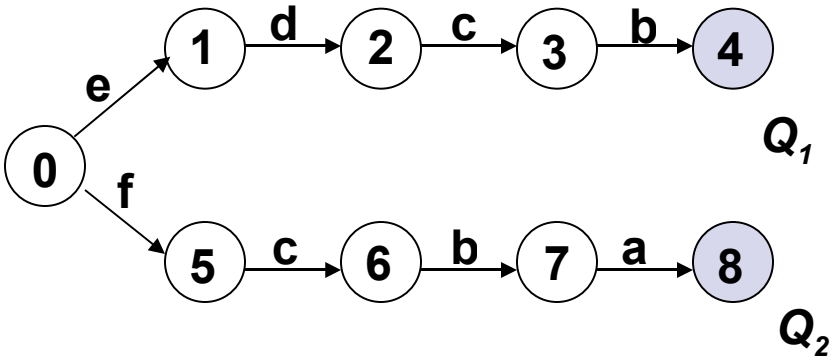
BUFF

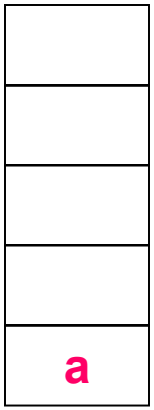
Runtime stack

BUFF matching (Runtime Stack)

Insert a







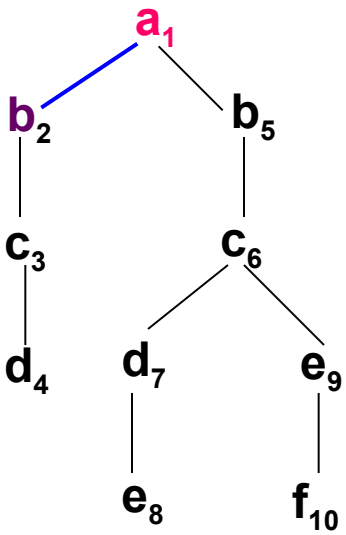
Document

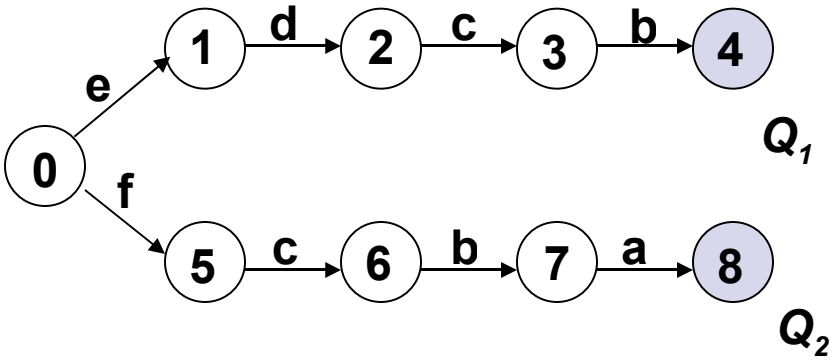
BUFF

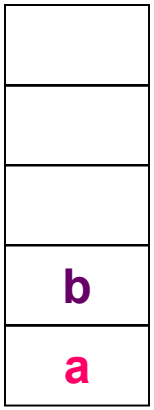
Runtime stack

BUFF matching (Runtime Stack)

Insert b







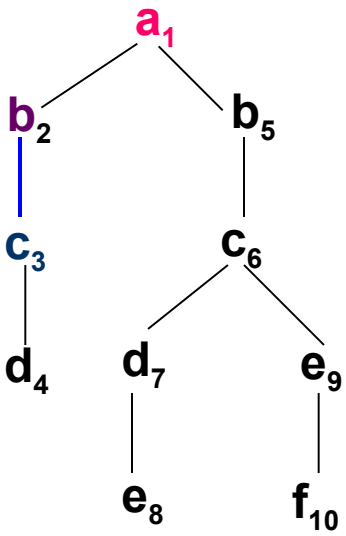
Document

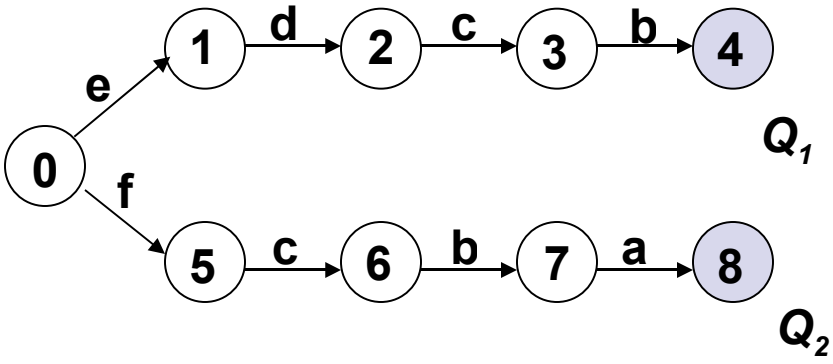
BUFF

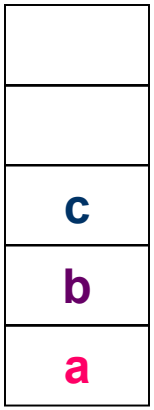
Runtime stack

BUFF matching (Runtime Stack)

Insert c







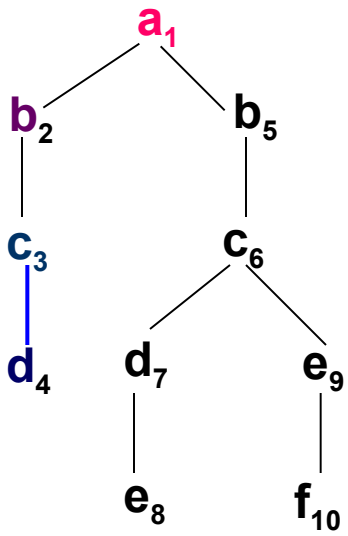
Document

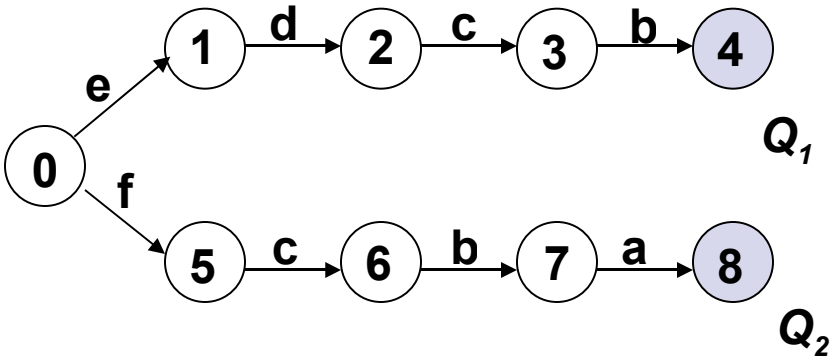
BUFF

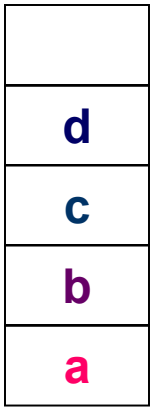
Runtime stack

BUFF matching (Runtime Stack)

Insert d







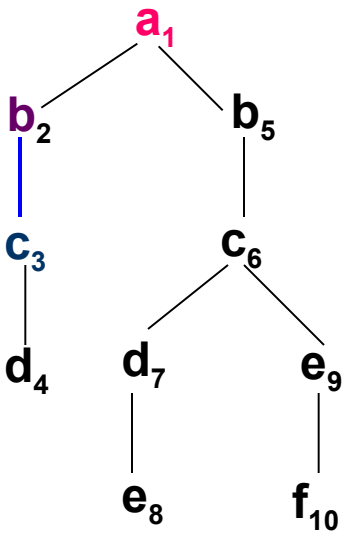
Document

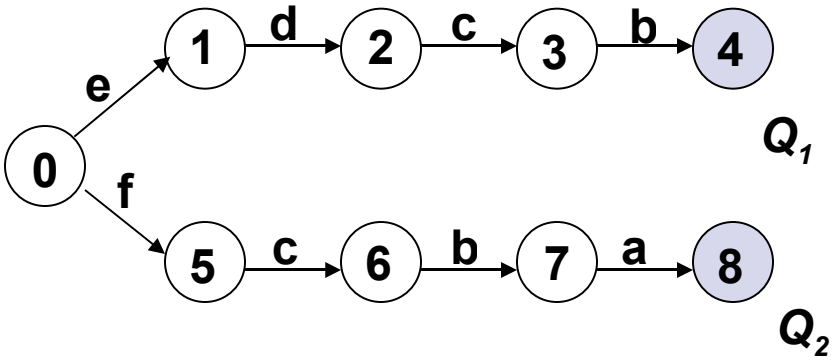
BUFF

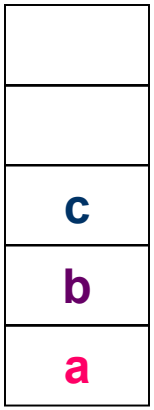
Runtime stack

BUFF matching (Runtime Stack)

Pop d (no transition defined)







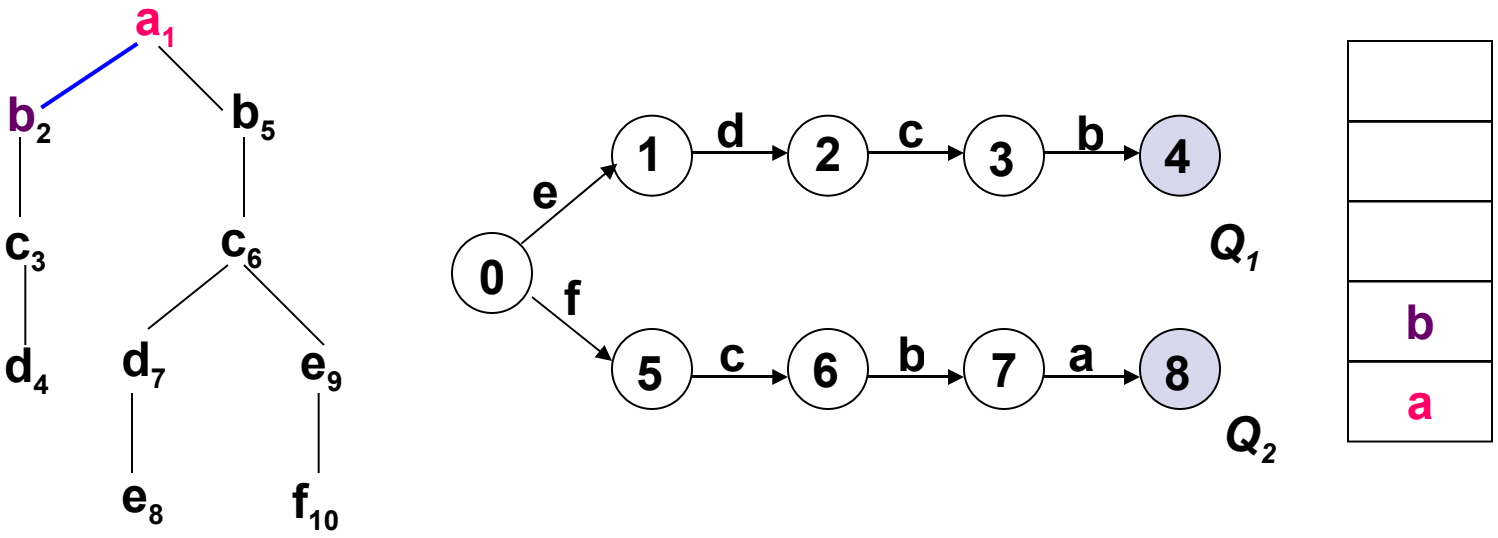
Document

BUFF

Runtime stack

BUFF matching (Runtime Stack)

Pop c (no transition defined)



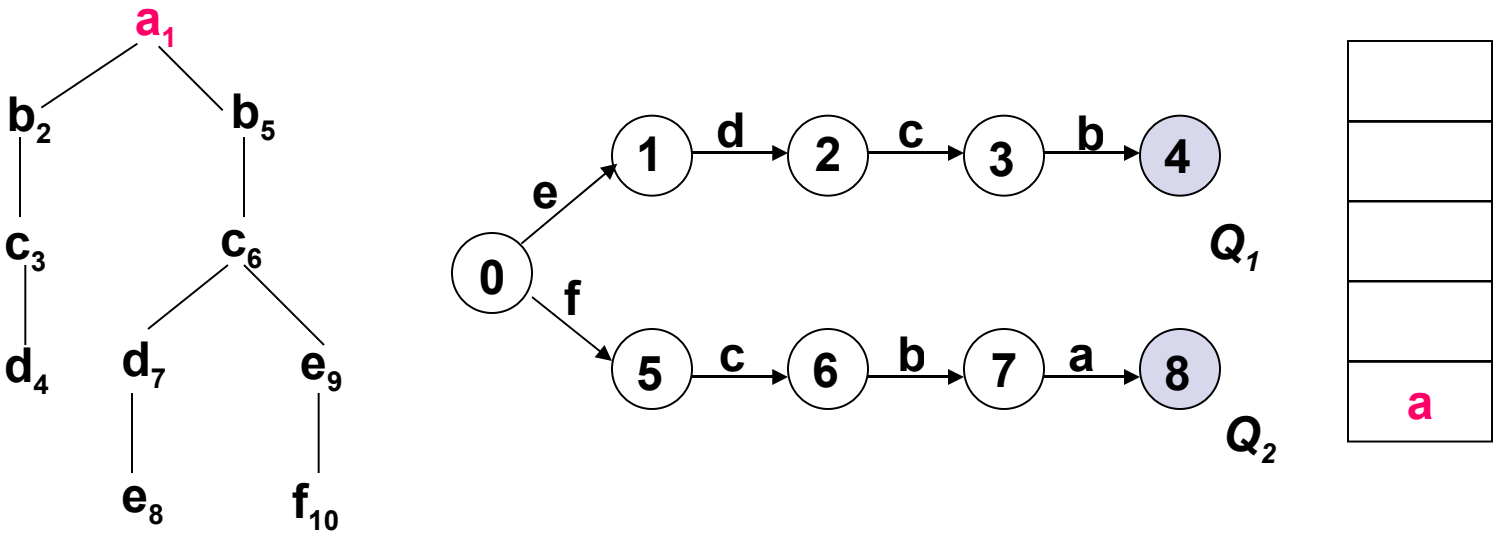
Document

BUFF

Runtime stack

BUFF matching (Runtime Stack)

Pop b (no transition defined)



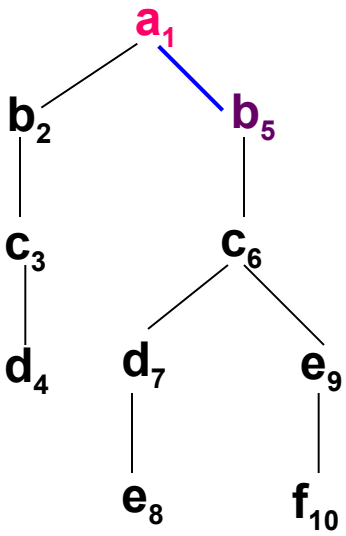
Document

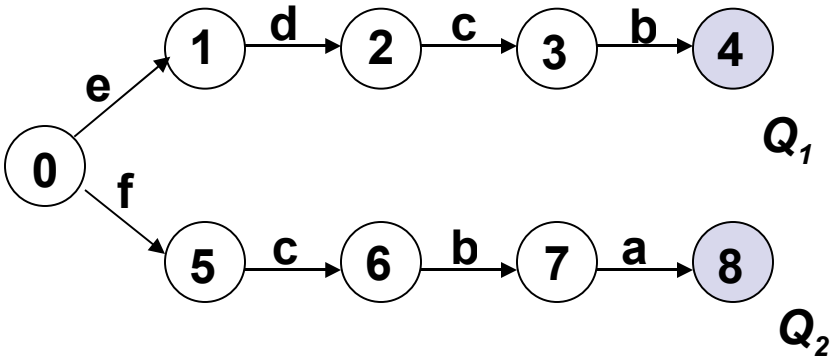
BUFF

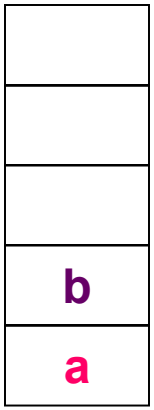
Runtime stack

BUFF matching (Runtime Stack)

Insert b







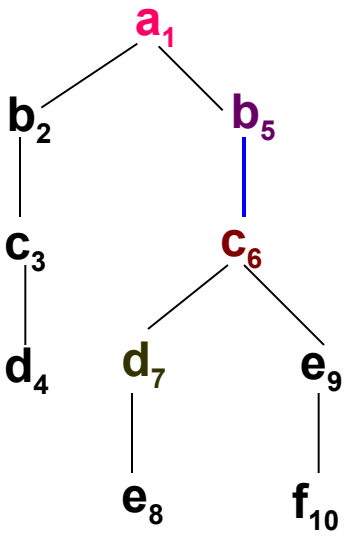
Document

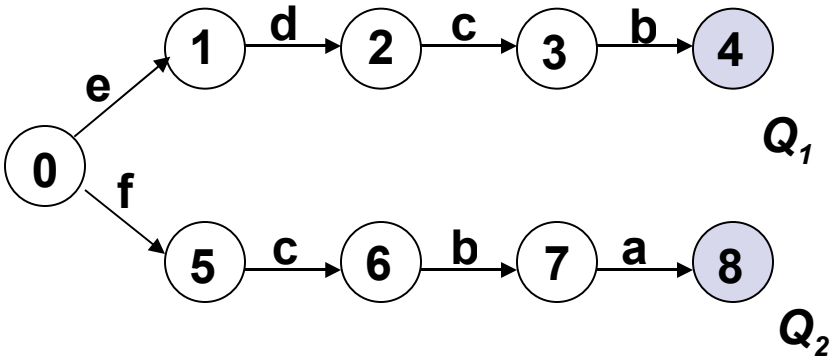
BUFF

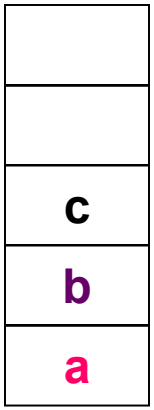
Runtime stack

BUFF matching (Runtime Stack)

Insert c







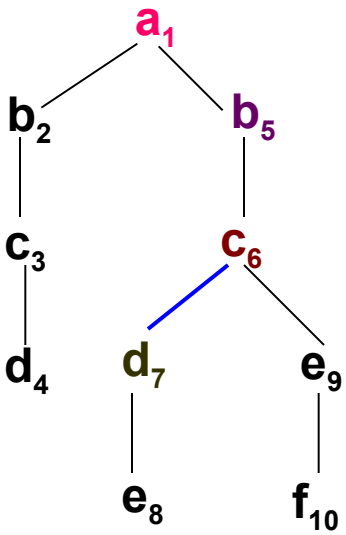
Document

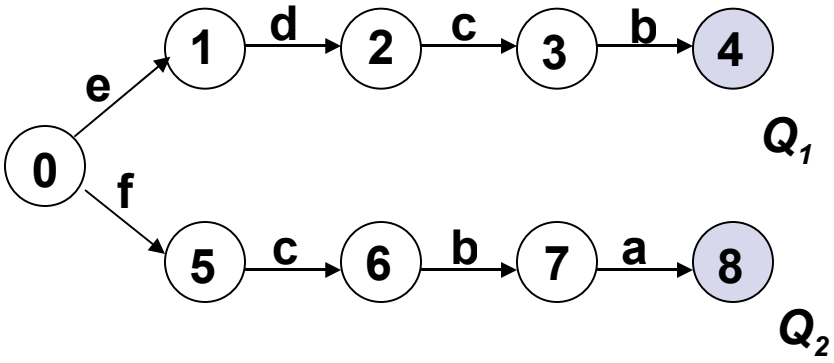
BUFF

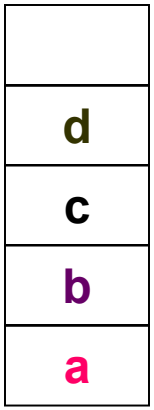
Runtime stack

BUFF matching (Runtime Stack)

Insert d







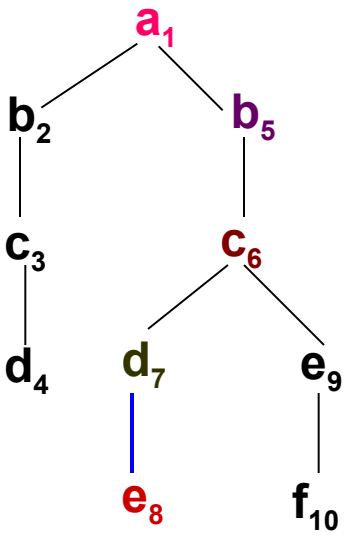
Document

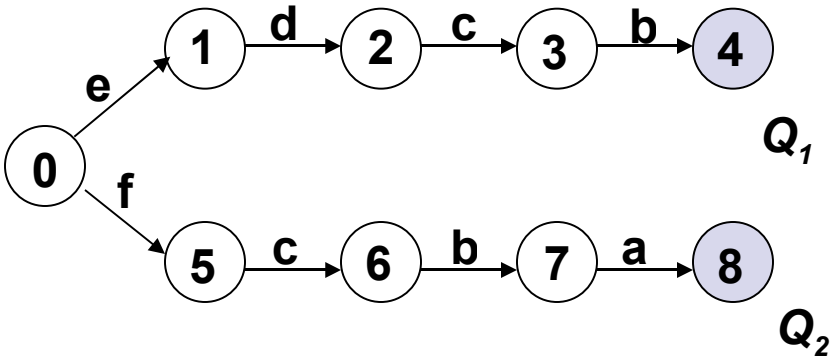
BUFF

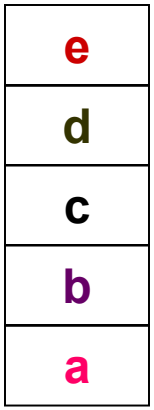
Runtime stack

BUFF matching (Runtime Stack)

Insert e







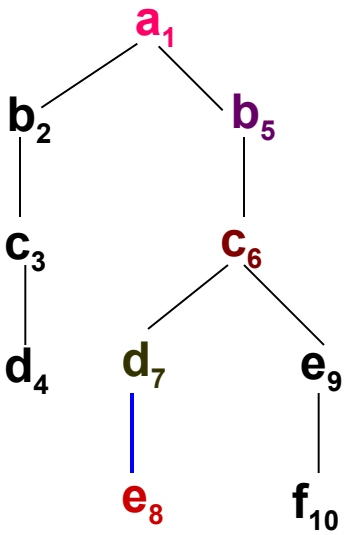
Document

BUFF

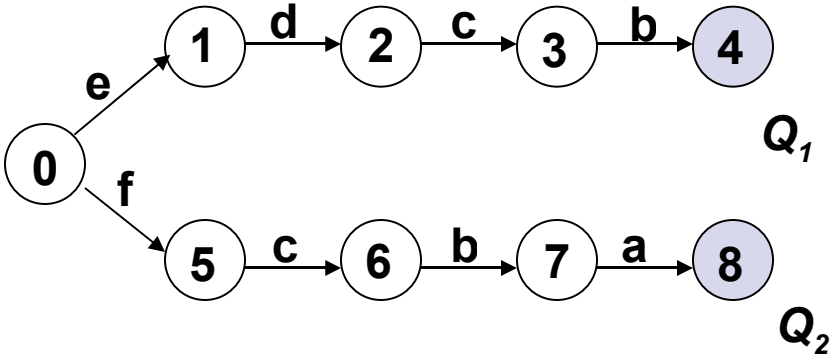
Runtime stack

BUFF matching (Runtime Stack)

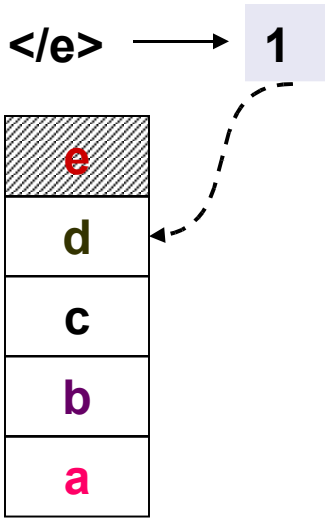
Pop e (State moves to transition 1)



Document



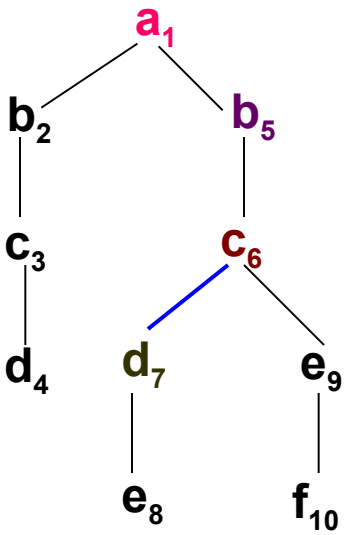
BUFF



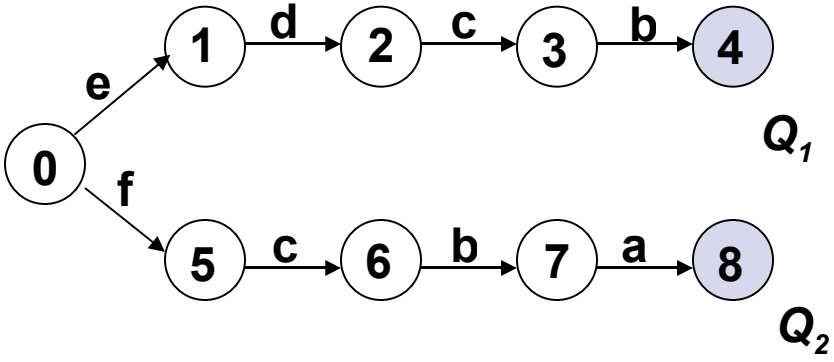
Runtime stack

BUFF matching (Runtime Stack)

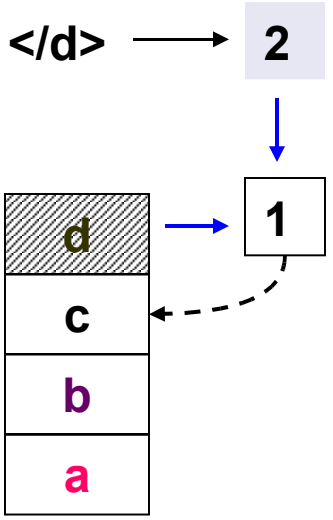
Pop d (State moves to transition 2)



Document



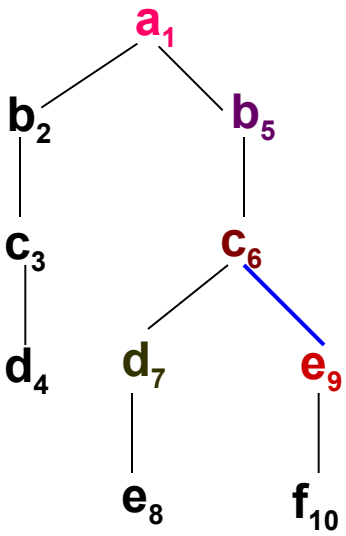
BUFF



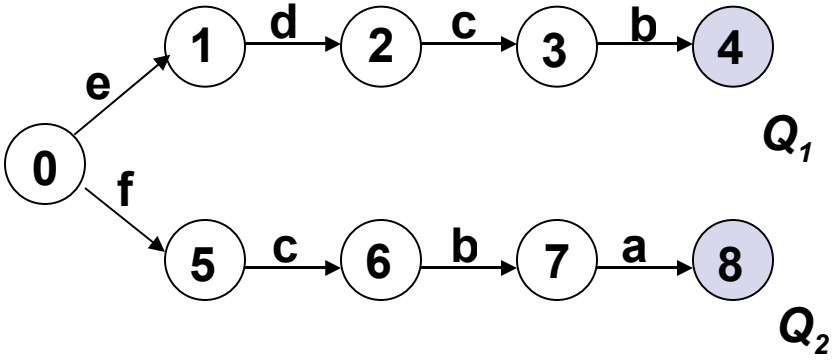
Runtime stack

BUFF matching (Runtime Stack)

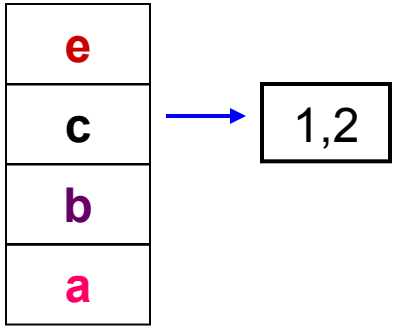
Insert e



Document



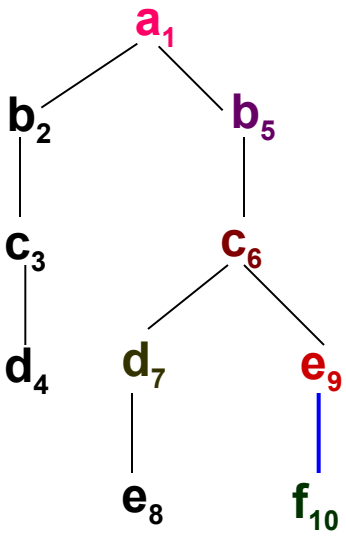
BUFF



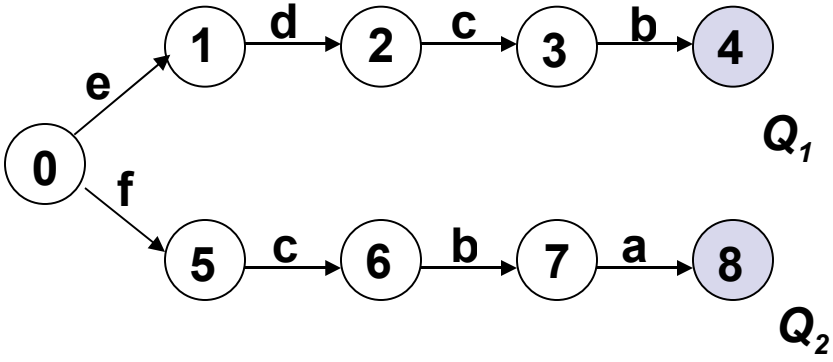
Runtime stack

BUFF matching (Runtime Stack)

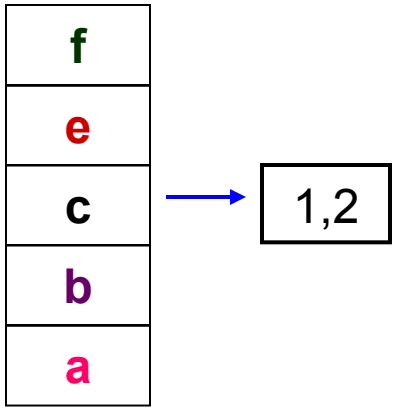
Insert f



Document



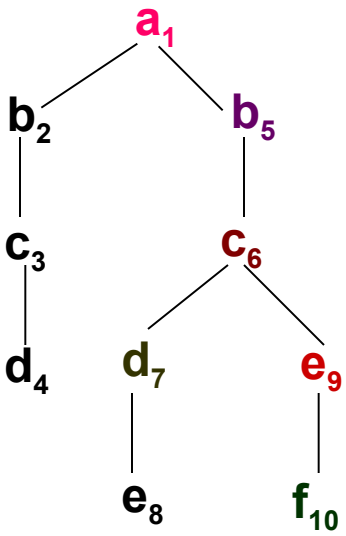
BUFF



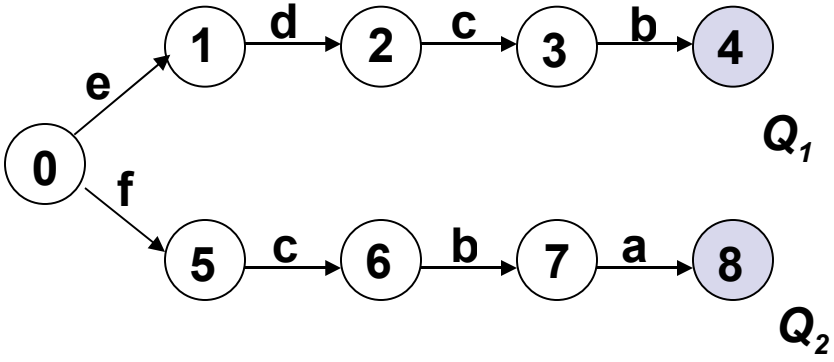
Runtime stack

BUFF matching (Runtime Stack)

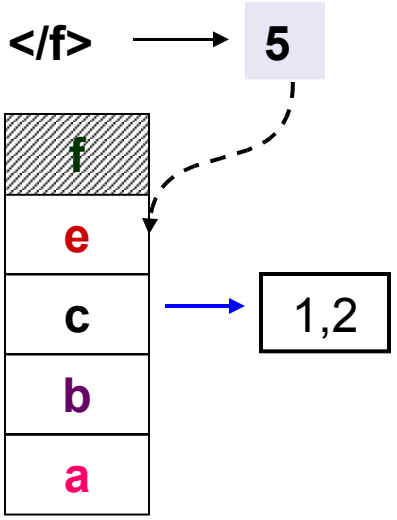
Pop f (State moves to transition 5)



Document



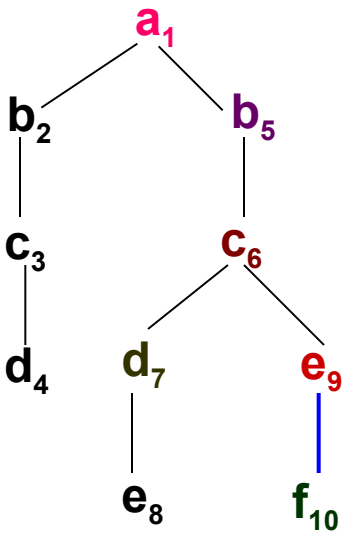
BUFF



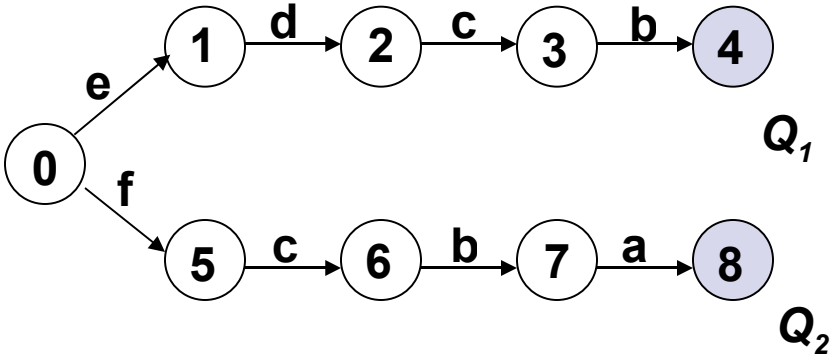
Runtime stack

BUFF matching (Runtime Stack)

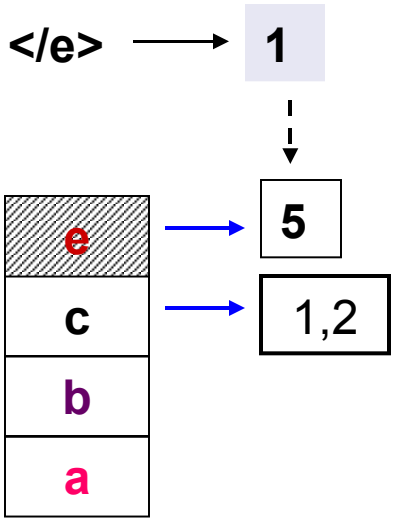
Pop e (State moves to transition 1)



Document



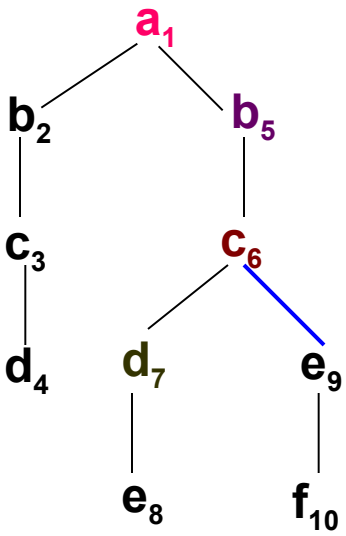
BUFF



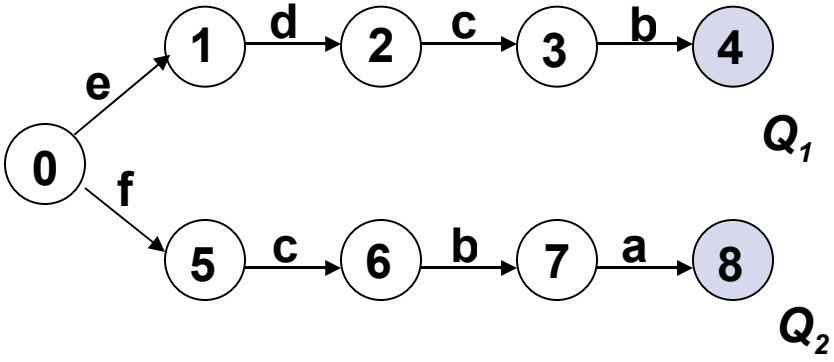
Runtime stack

BUFF matching (Runtime Stack)

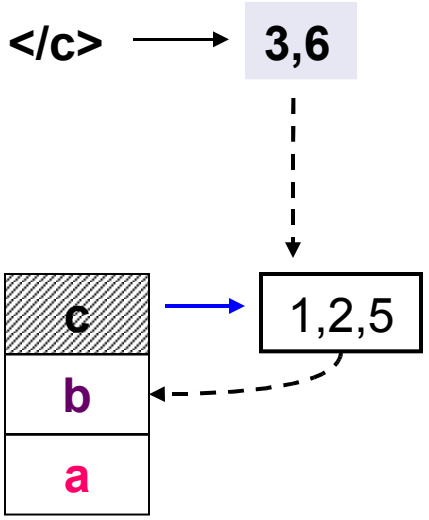
Pop c (State moves to transition 3,6)



Document



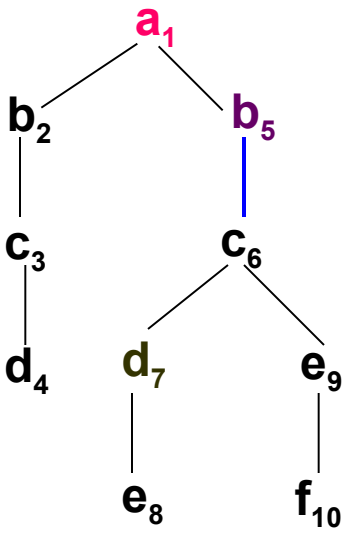
BUFF



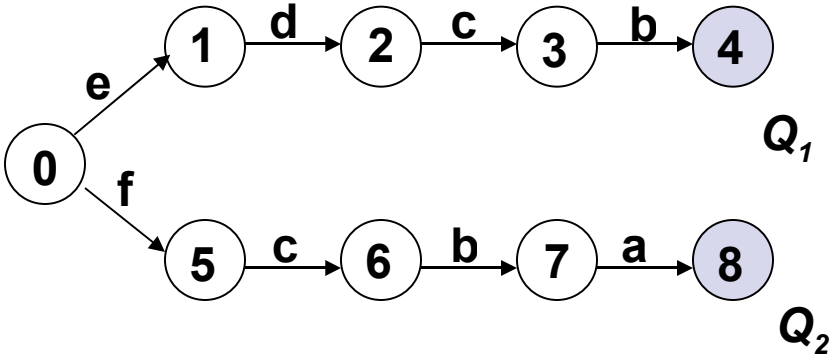
Runtime stack

BUFF matching (Runtime Stack)

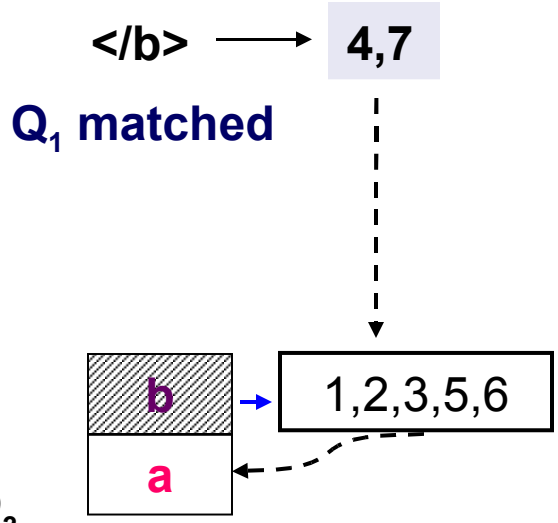
Pop b (State moves to transition 4,7)



Document



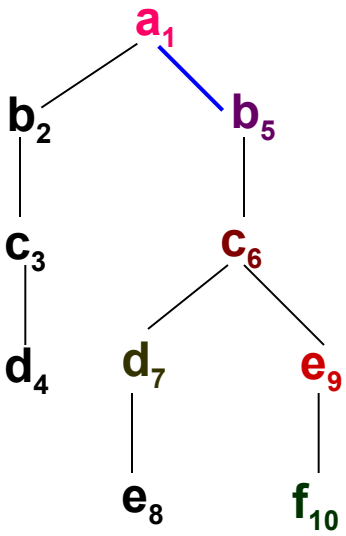
BUFF



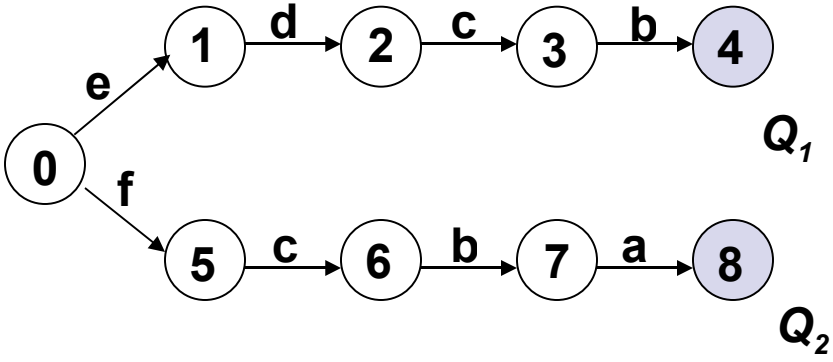
Runtime stack

BUFF matching (Runtime Stack)

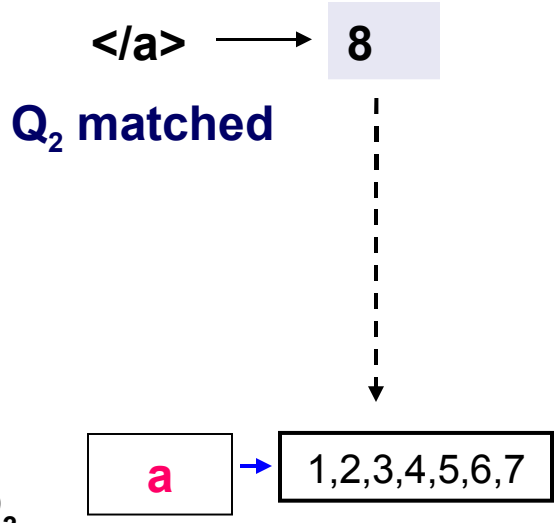
Pop a (State moves to transition 8)



Document

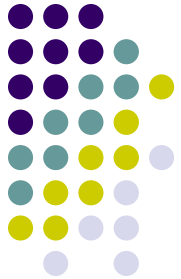


BUFF



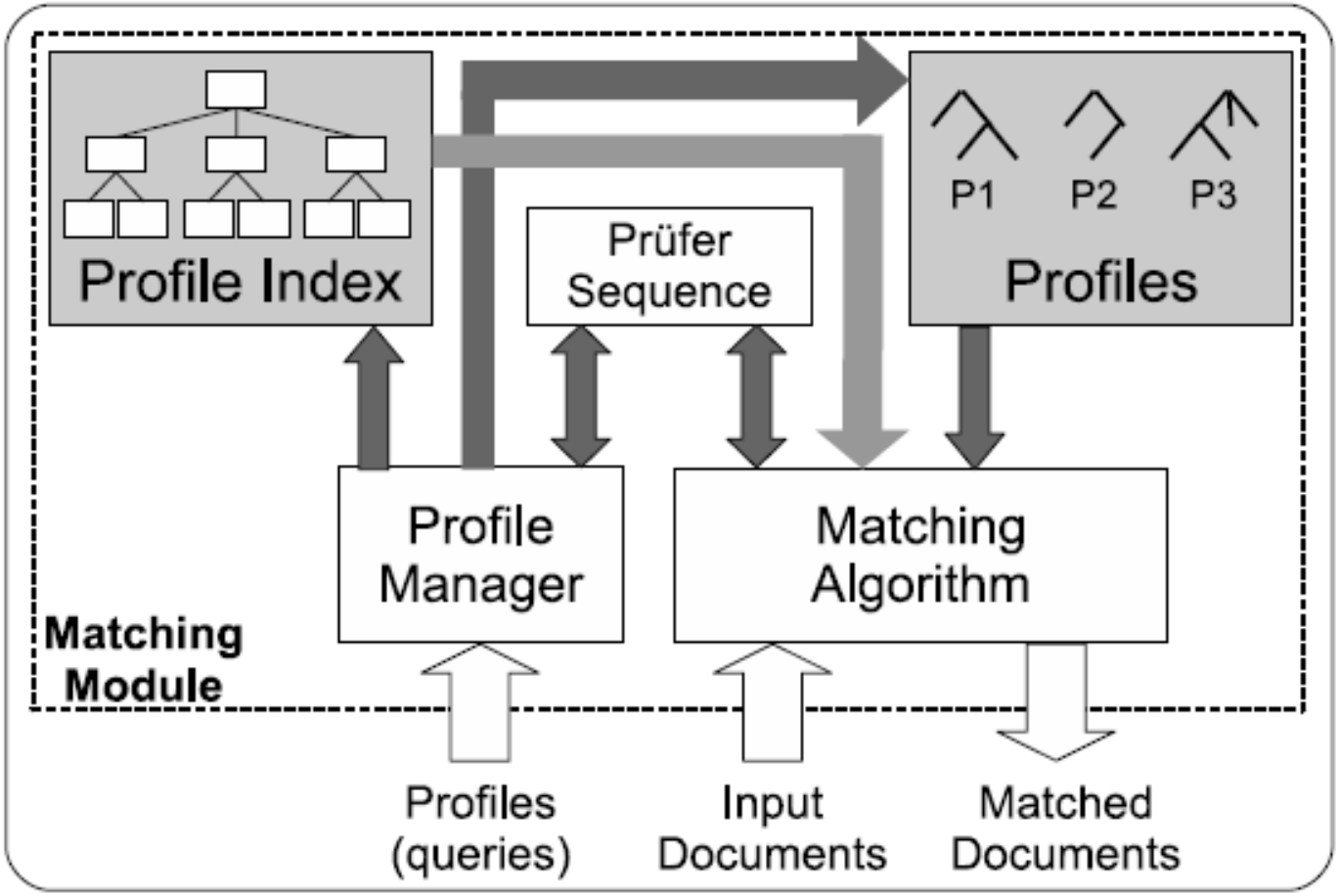
Runtime stack

Bounding Based XML Filtering

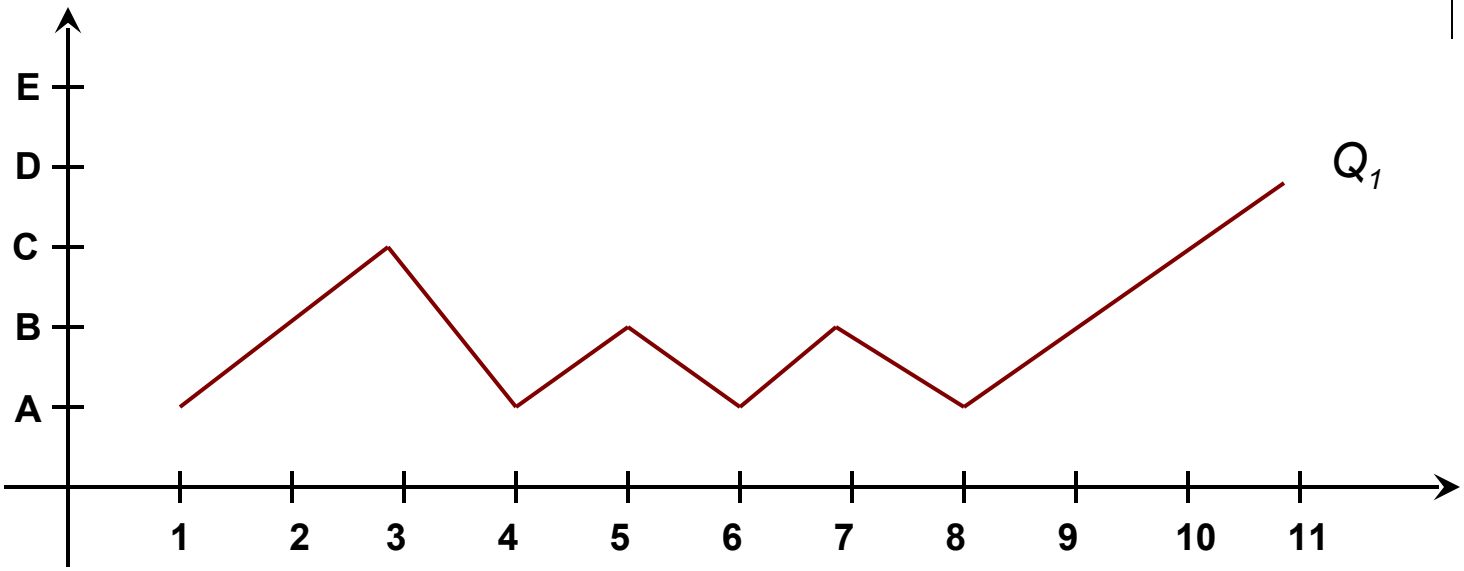


- Translates documents and queries to Prüfer sequences.
- Inserts the Prüfer encoding of the profile in a tree-based indexing structure.
- Performs subsequence matching.
- Verifies candidate profiles to guarantee query structural constraints
- Employs *query pruning* technique based on lower and upper bound estimates.
- Stores original profiles and destination address in a routing table.

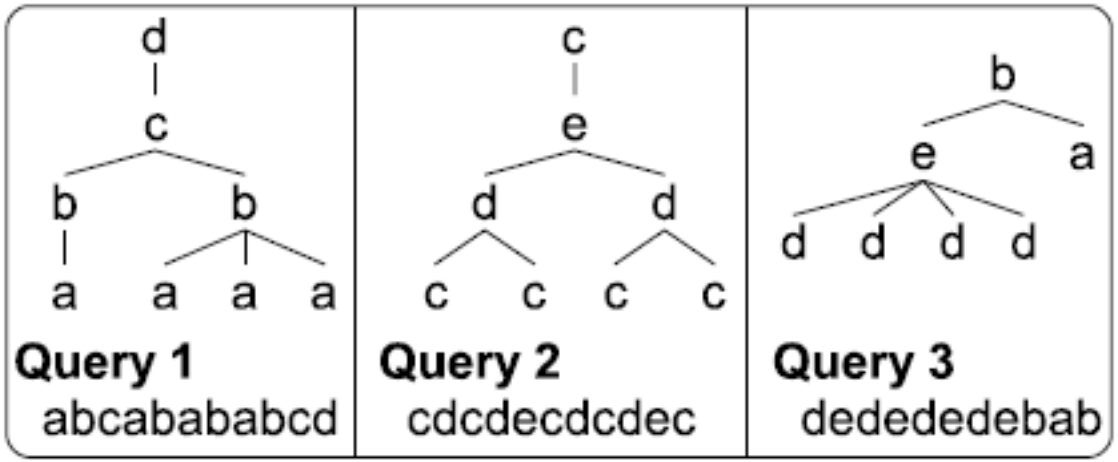
BoXFilter Core Modules



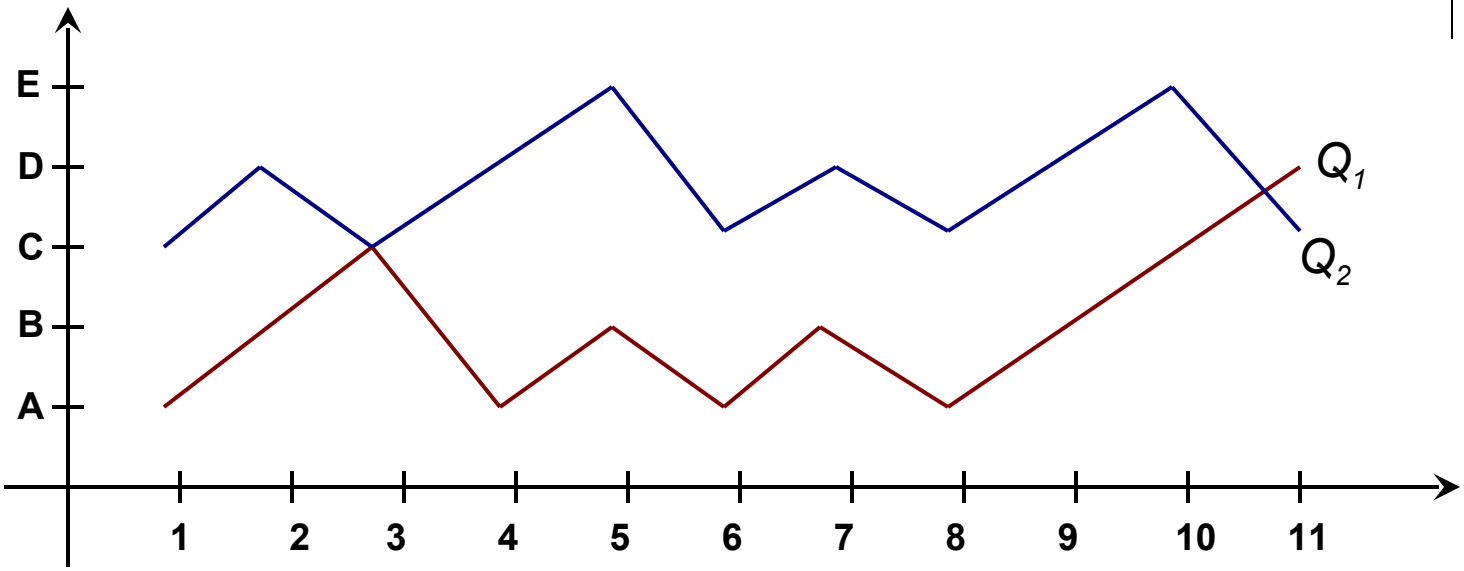
Plot (Prüfer sequence pos, element)



Q_1 = *abcababababcd*

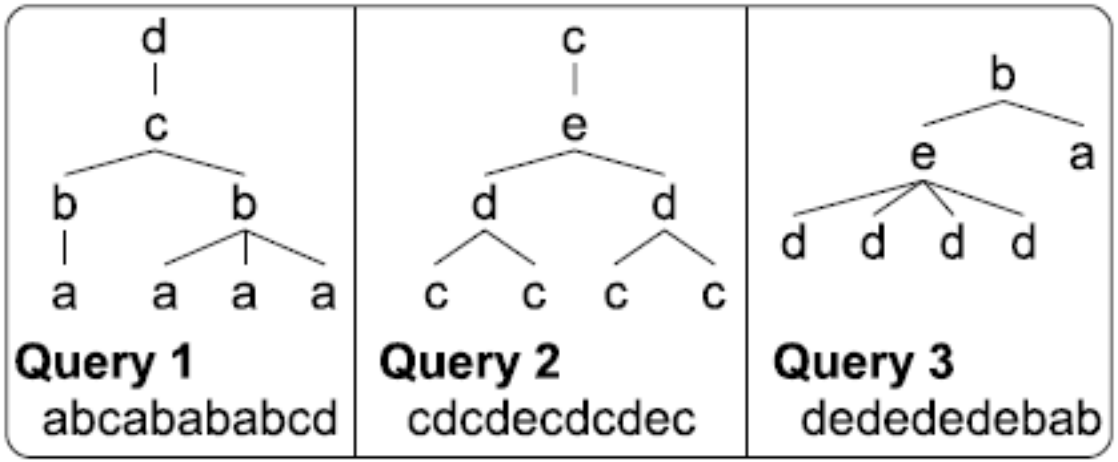


Plot (Prüfer sequence pos, element)

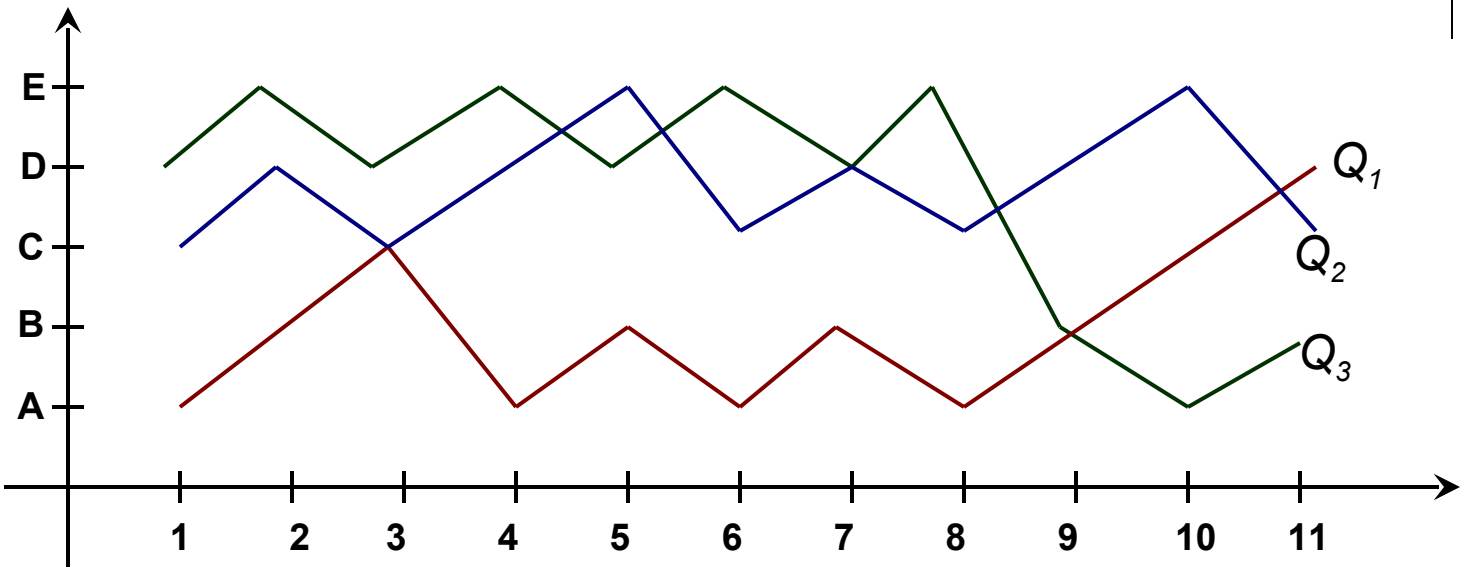


Q_1 = *abcababababcd*

Q_2 = *cdcdecdcdec*



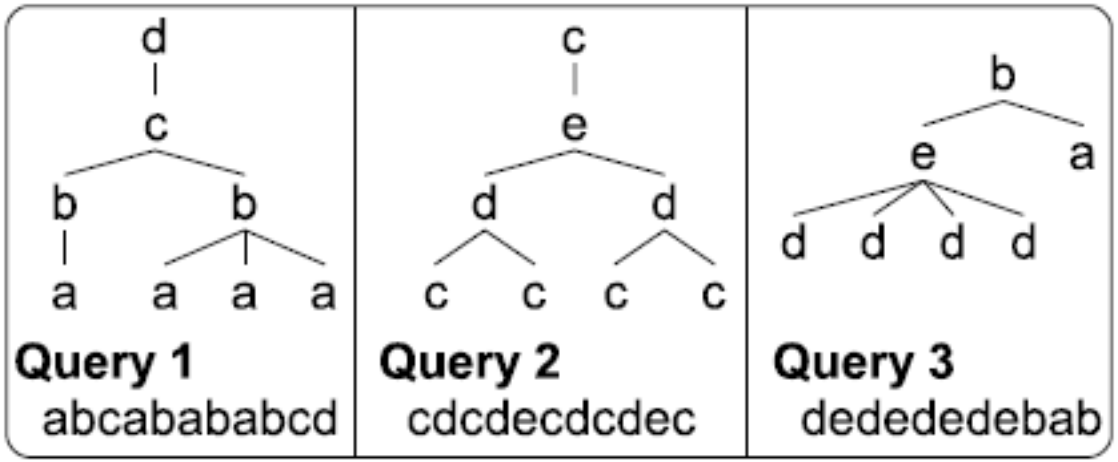
Plot (Prüfer sequence pos, element)



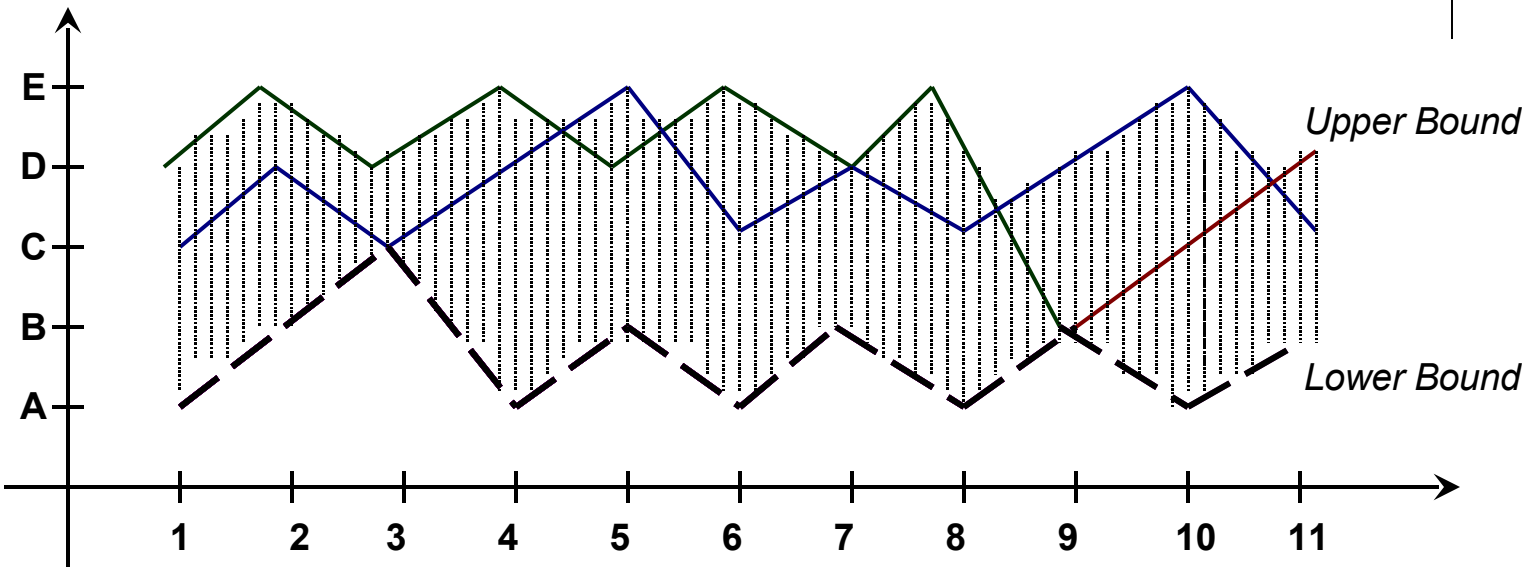
Q_1 = *abcababababcd*

Q_2 = *cdcdecdcdec*

Q_3 = *dedededebab*



Sequence Envelope



$L_i = min(S1_i, \ldots, Sk_i)$

$U_i = min(S1_i, \ldots, Sk_i)$

Lower Bound (L)= abcababababab

Upper Bound (U) = dedeeededed

Min and Max is determined based on alphabetical order

Q_1 = *abcababababcd*

Q_2 = *cdcdecdcdec*

Q_3 = *dedededebab*

Properties of a sequence envelope

- Can be used as an aggregation of set of sequences such that the Prüfer encoding of profiles (S1,…,S*k*) can be combined into a single sequence envelope *se* and the document can be matched against *se*.

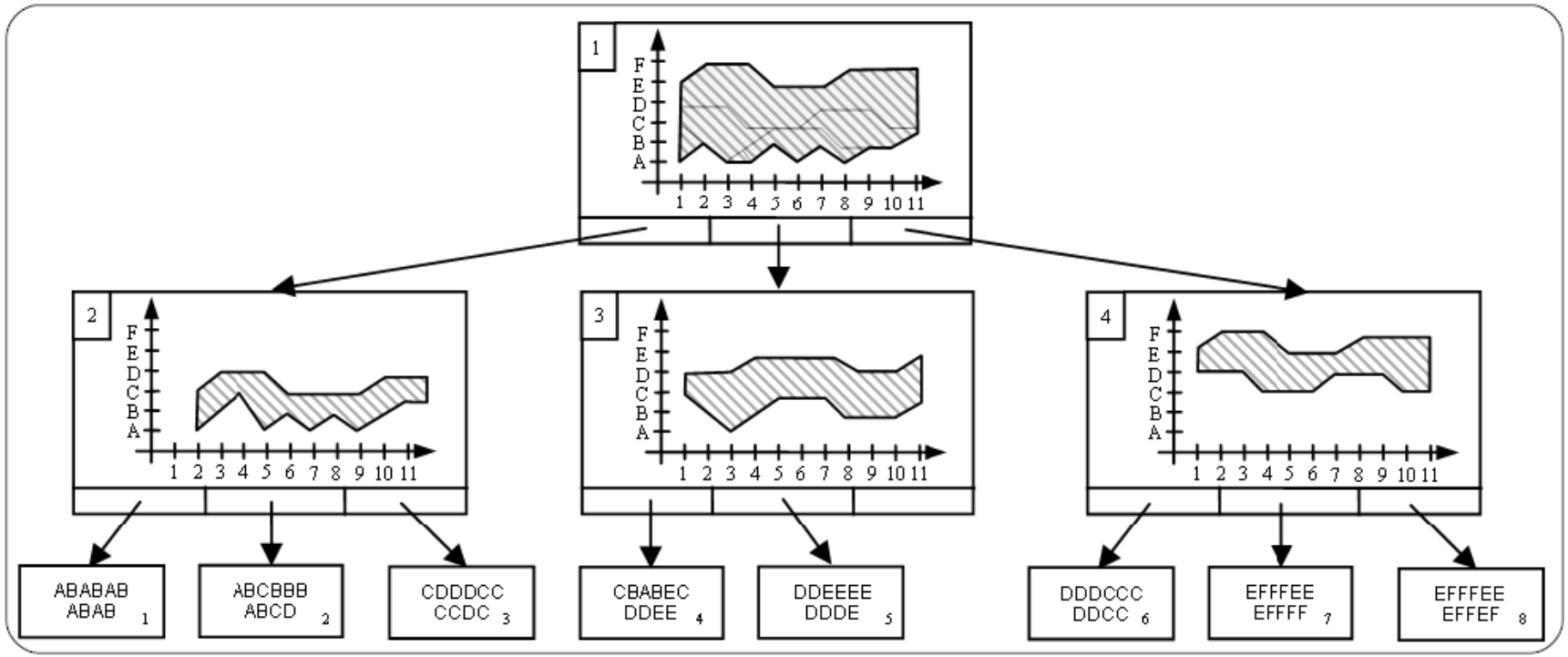
- If there is no subsequence in document for which every element in the subsequence is between the lower and upper bound of the on the corresponding position in the sequence envelope then there is no subsequence matching between document any of the profile encodings *S1*,…,*Sk.*

Properties of a sequence envelope

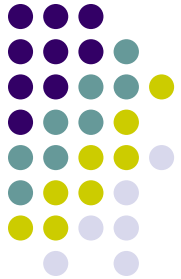
- *Lemma: If there is no subsequence matching between the Prüfer sequence of the document D and a sequence envelope SE, then there is no matching between D and any of the queries whose sequence are within SE.*

- Sequence envelopes can be nested.
  - This property allows the creation of the hierarchical index structure over the envelopes, where a parent node in the index tree has an envelope that contains all the envelopes of its children.

Sequence envelope tree

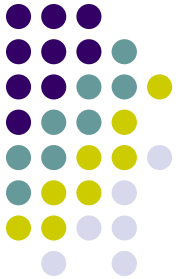


Sequence envelope tree



- Tree search
  - Traverse index tree from root to the leaf node.
  - Checks the bounding envelope at the current node.
    - If there is a subsequence matching, algorithm is executed recursively for the corresponding sub-tree
- Advantages
  - Dynamic nature
    - Insert and delete operations can be intermixed with search operation.
  - Scalability
    - In case main memory is not sufficient to accommodate all entries of tree, nodes can be paged secondary storage and loaded upon request

Filtering algorithm (sequential mode)



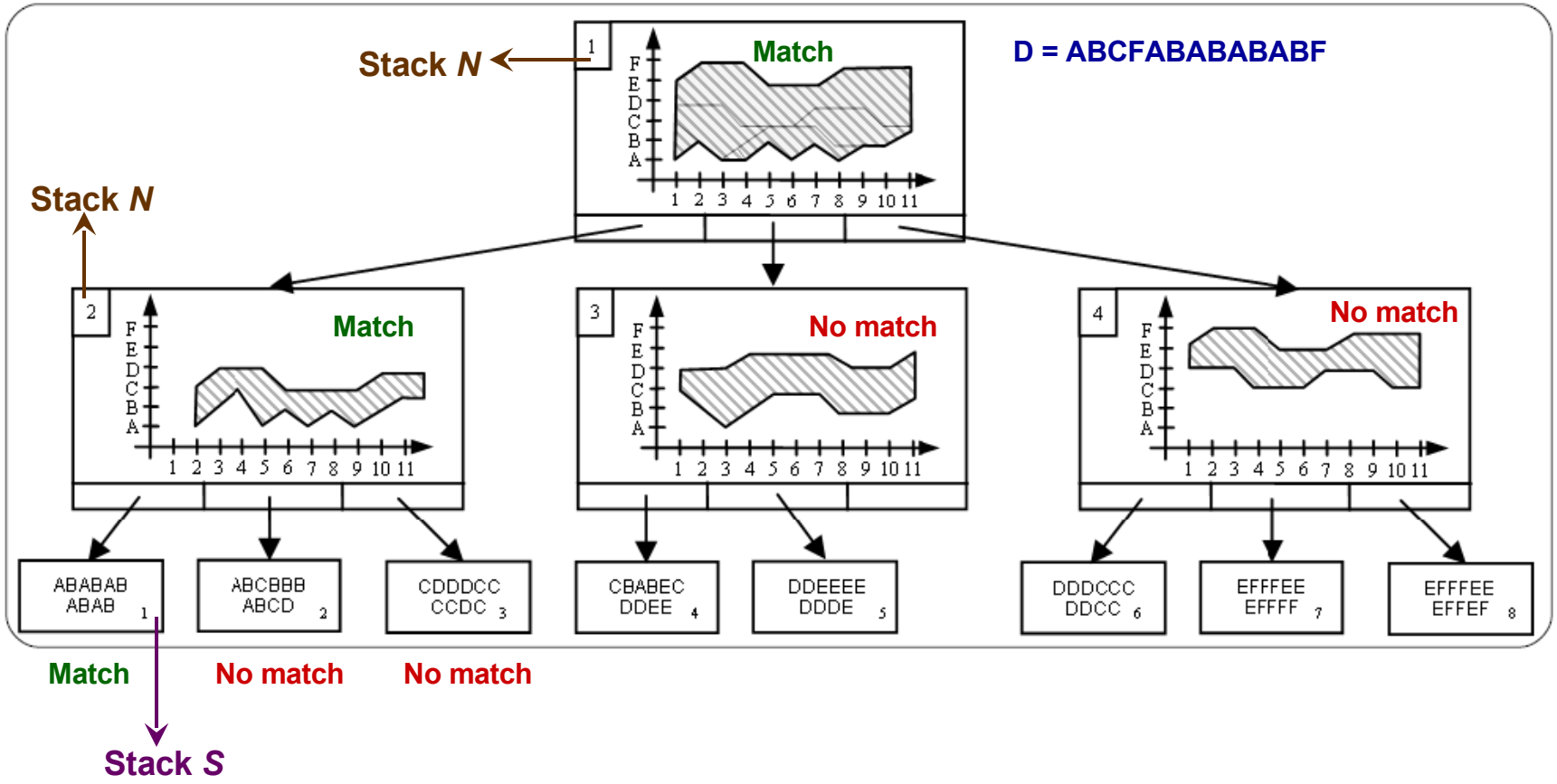
Algorithm 3 Document Filtering (sequential mode)

Require: $\bar{D}$: document, R: BoXFilter tree root, T: routing table
Ensure: The set of profiles $\bar{S}$ which are satisfied

```
1:  Set S ← ∅; N ← R,
2:  D = ConvertToPruferSeq(D̄)
3:  while N is not empty do                    ▷ BoXFilter tree traversal
4:      C = N.pop()
5:      for each child E of node C do
6:          if E is a leaf then
7:              if E's sequence is a substring of D then
8:                  S.push(E)
9:          else
10:             if E's sequence envelope is a substring of D then
11:                 N.push(E)
12:     end for
13: end while
14: while S is not empty do                    ▷ Verification step
15:     S = S.pop()
16:     S̄ = T.LookUp(S)
17:     if D̄ satisfies S̄ then
18:         S̄.push(S̄)
19: end while
```

1. Profiles are encoded in Prüfer sequences and organized in a BoXFilter structure.
2. Document is also encoded in Prüfer sequence (D).
3. Input to the algorithm is the root of the tree and the document (D).
4. BoXFilter tree is traversed and two stacks are used, one stack (S) stores the pointer to the leaf nodes that contain Prüfer encoding of the profile and is a subsequence of D. Another stack (N) keeps a list of internal nodes that have a sequence envelope that is subsequence of D.
5. Candidate profiles in S are verified.

Finding a match in an envelope tree



Filtering algorithm (batch mode)

Algorithm 4 Document Filtering (batch mode)

Require: R: query BoXFilter root, M: document BoXFilter root, T: routing table, H document table
Ensure: The set of profiles $\bar{S}$ which are satisfied

```
1: Set S ← ∅; N ← (R, M),
2: while N is not empty do                                  ▷ Tree Join step
3:     (C1,C2) = N.pop()
4:     for each child E1 of node C1 do
5:         for each child E2 of node C2 do
6:             if both E1 and E2 are leafs then
7:                 if E1's sequence is a substring of E2 then
8:                     S.push(E1,E2)
9:             else
10:                if E1 is leaf then
11:                    LSearch(E1,E2,S)
12:                else
13:                if E2 is leaf then
14:                    RSearch(E1,E2,S)
15:                else
16:                if E1's sequence envelope is a substring of E2 sequence envelope then
17:                    N.push(E1,E2)
18:        end for
19:    end for
20: end while
21: while S is not empty do                                 ▷ Verification step
22:     (S,D) = S.pop()
23:     S̄ = T.LookUp(S)
24:     S̄ = H.LookUp(D)
25:     if D̄ satisfies S̄ then
26:         S̄.push(S̄)
27: end while
```

1. Documents are organized in a BoXFilter tree, same way as profiles.

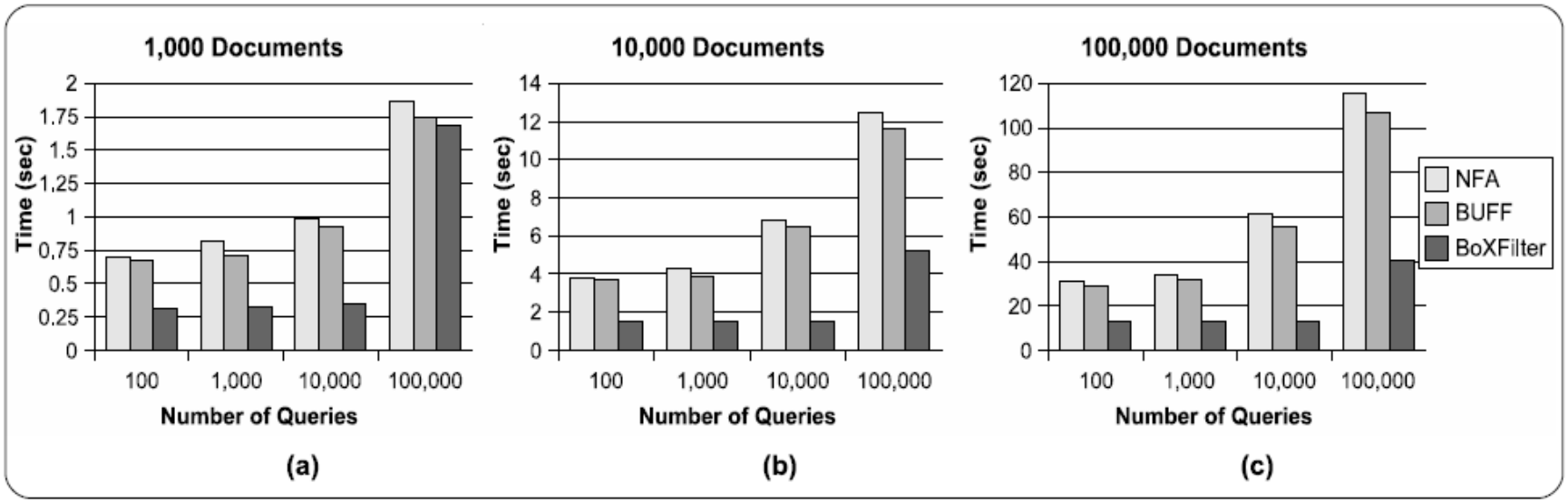
2. Join BoxFilter tree with the document tree in top-down manner

3. Checks if sequence envelope of query node is a substring of the sequence envelope of the document node.

4. Continue until the leaf level in one of the tree is reached.

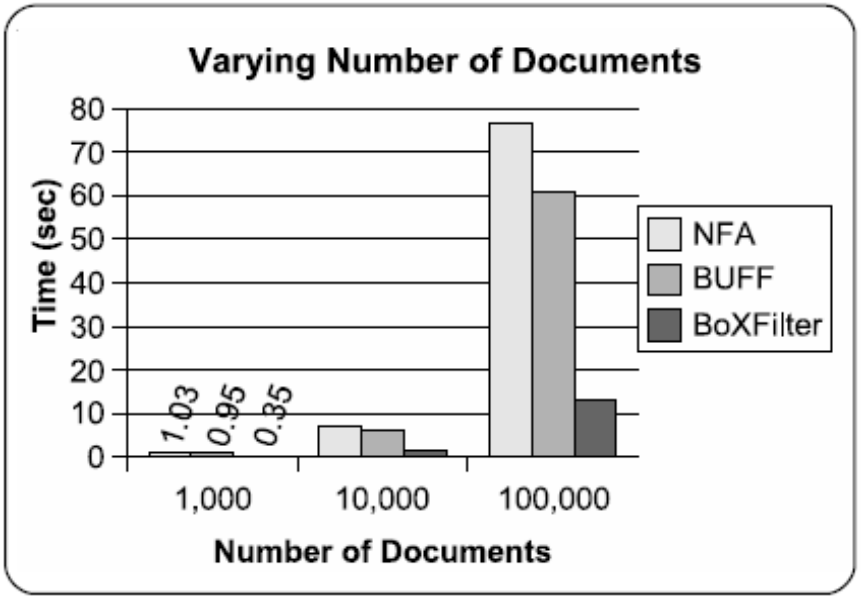
5. Use the leaf node as the key to search in the remaining subtree.

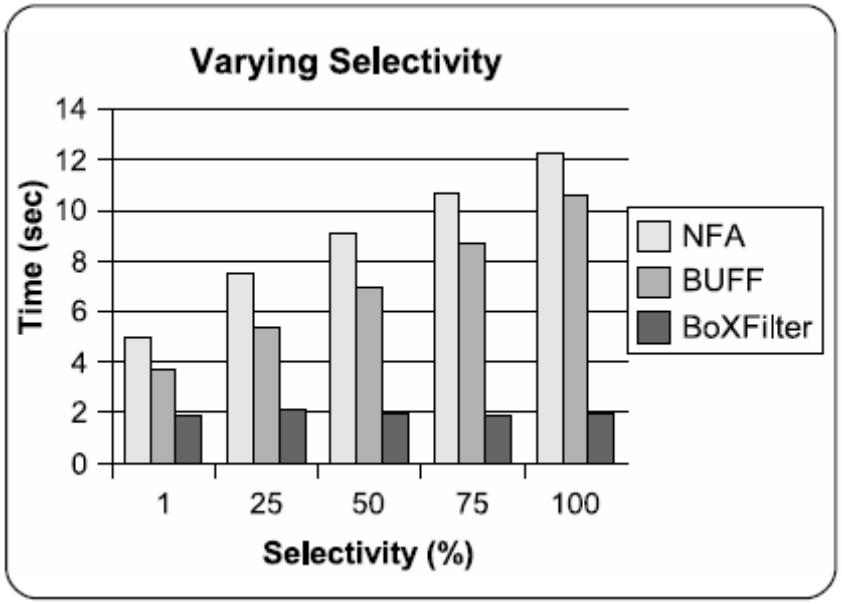
Experimental results



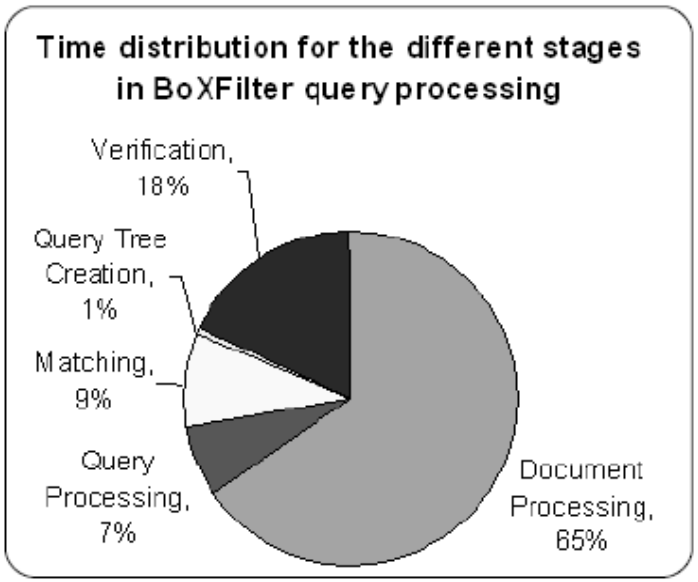
Performance results when varying the number of queries

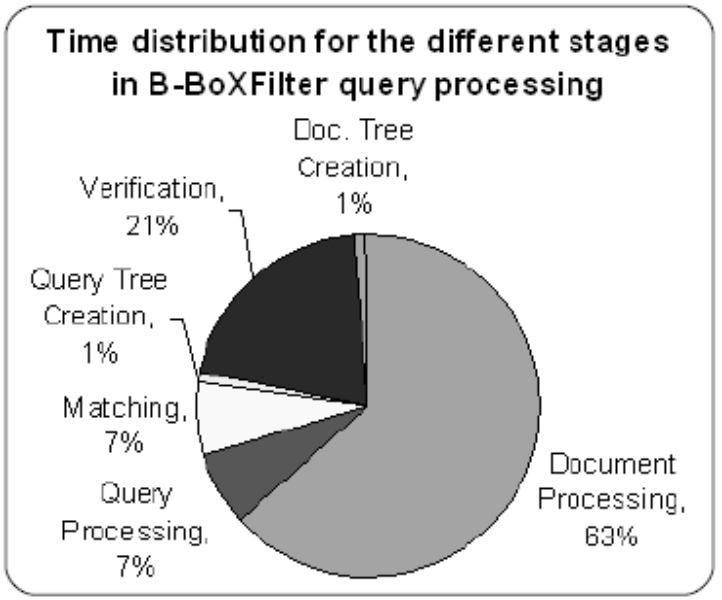
Experimental results

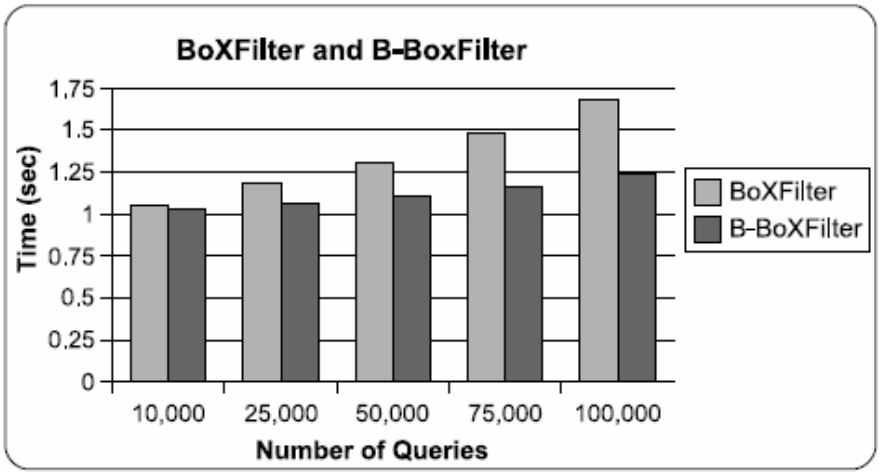




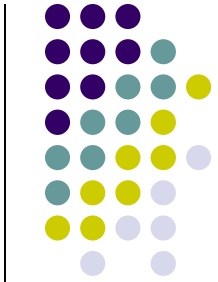
Experimental results







Conclusion



- Queries can be pruned out by evaluating the lower and upper bounds of their envelopes.
- Bottom-Up Filtering FSM (BUFF) offers performance advantages over the traditional FSM-based approach.
- Pruning offered by BoXFilter provides drastic performance improvement.