CS234 Project Demo

Baoju Wang
SID: 862325133

The project

In this project I implemented the construction of suffix trees in linear-time using Ukkonen's algorithm.

Ukkonen's algorithm is a linear-time algorithm for constructing suffix trees, proposed by Esko Ukkonen in 1995.

Ukkonen's Algorithm

Time : O(n)
Space: O(n)

Loop the string once, then we call inserting suffix S[i+1] (for i = 1 to n) to the existing suffix tree as a phase. Then for every phase, we need to follow three rules:

- Rule 1: For phase i+1, if S[j...i] ends at last character of leaf edge, then append S[i+1] to the end.
- Rule 2: For phase i+1, if S[j..i] ends in the middle of the edge and S[j..i+1] does not overlap with one of the existing edges around the node, create a new edge with label S[i+1]
- Rule 3: For phase i+1, if S[j..i] ends in the middle of the edge and S[j...i+1] overlaps with one of the existing edges around the node, do nothing.

Ukkonen's Algorithm

The definition of these variables are described below:

- Active point: Consists of active position, active node and active edge.
- Active position: A point where next phase or next rule extension starts (Active position always starts from the root)
- Active node: The node where active point starts
- Active edge: The edge we use around the active node. Each edge is represented by its start index around the node.
- Active length: How far we go in active edge

Ukkonen's Algorithm

For every extension:

1. If case = rule 3, increment active length by 1 if active length is less than the length of the edge (not skip over next node)
2. If case = rule 3 and active length gets greater than the length of the edge, change the active node to the next internal node, active edge is none, active length is 0 (skip over next node)
3. If active length is 0, start from the root
4. If case = rule 2 and active node = root, increment active edge by one(pick the next edge around the root) and decrement active length by one.
5. If case = rule 2 and active node is not the root, change the active node to the node that suffix link is pointing to

One example – Phase0,Extension0 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6

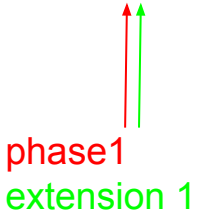
phase0
extension 0

```
remainingSuffixCount = 0
activeNode = None
activeEdge = -1
activeLength = 0
globalLeafEnd = -1
```

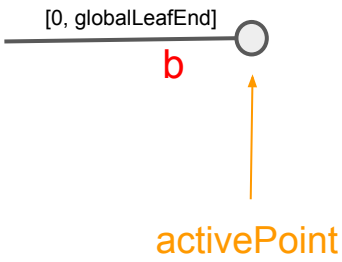
One example – Phase0,Extension0 End
Phase1, Extension1 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6




```
remainingSuffixCount = 0
activeNode = Root
activeEdge = -1
activeLength = 0
globalLeafEnd = 0
```

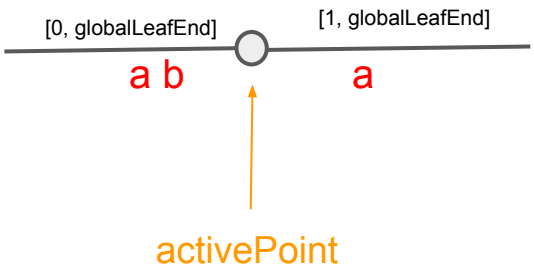


One example – Phase1,Extension1 End
Phase2, Extension2 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6


```
remainingSuffixCount = 0
activeNode = Root
activeEdge = -1
activeLength = 0
globalLeafEnd = 1
```

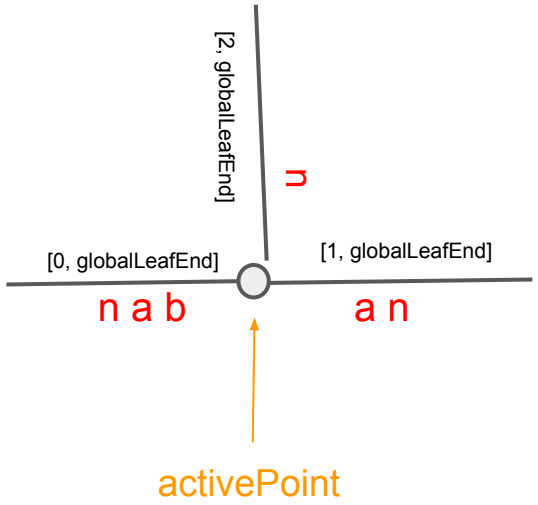


One example – Phase2,Extension2 End
Phase3, Extension3 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6

phase2
extension 2



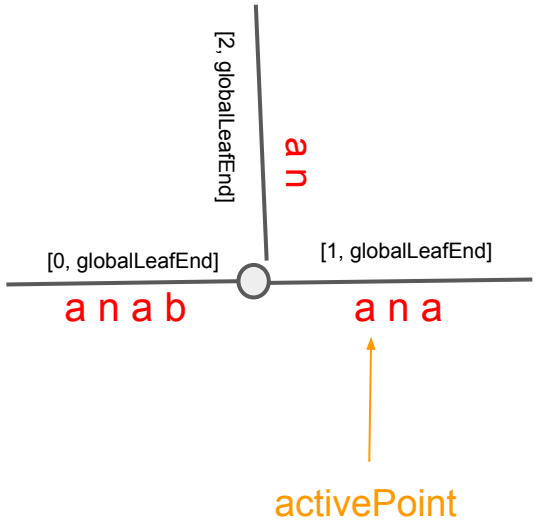

```
remainingSuffixCount = 0
activeNode = Root
activeEdge = -1
activeLength = 0
globalLeafEnd = 2
```

One example – Phase3,Extension3 End
Phase4, Extension4 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6

phase3
extension 3



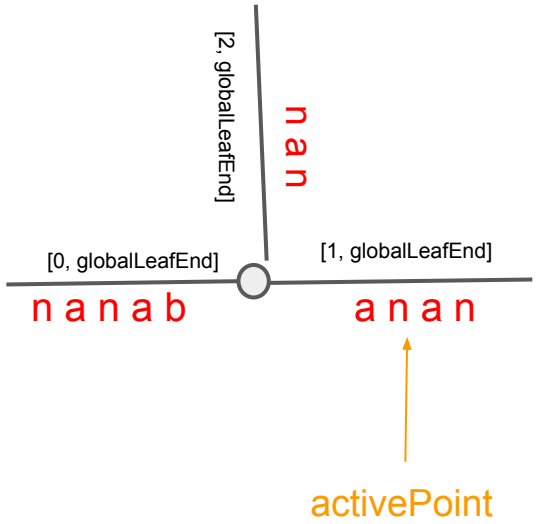

```
remainingSuffixCount = 1
activeNode = Root
activeEdge = 1
activeLength = 1
globalLeafEnd = 3
```

One example – Phase4,Extension4 End
Phase5, Extension5 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6

phase4
extension 4




```
remainingSuffixCount = 2
activeNode = Root
activeEdge = 1
activeLength = 2
globalLeafEnd = 4
```

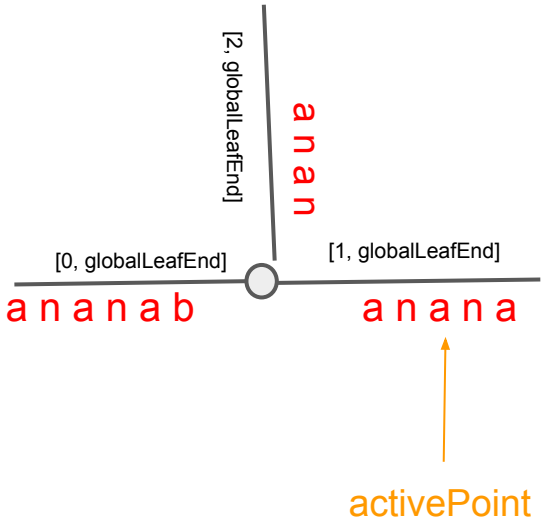
One example – Phase5,Extension5 End
Phase6, Extension0 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6

phase5
extension 5

```
remainingSuffixCount = 3
activeNode = Root
activeEdge = 1
activeLength = 3
globalLeafEnd = 5
```



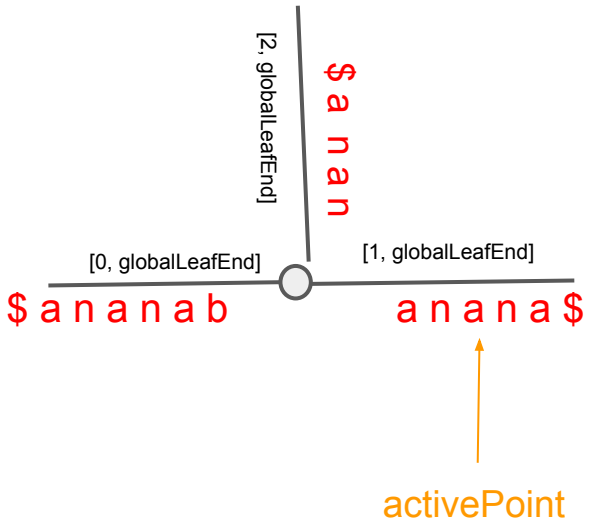
One example – Phase6,Extension3 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6

phase6

extension 2




```
remainingSuffixCount = 4
activeNode = Root
activeEdge = 1
activeLength = 3
globalLeafEnd = 6
```

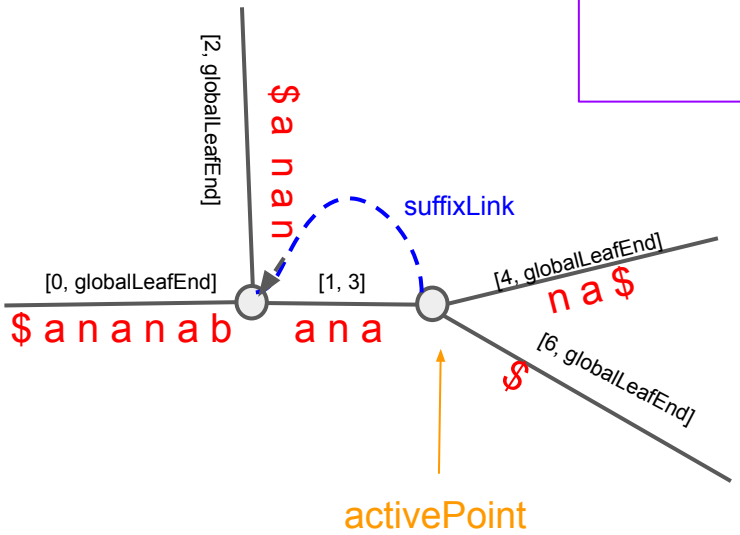
One example – Phase6,Extension3 End
Phase6, Extension4 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6

phase6
extension 3

```
remainingSuffixCount = 3
activeNode = Root
activeEdge = 1
activeLength = 3
globalLeafEnd = 6
```

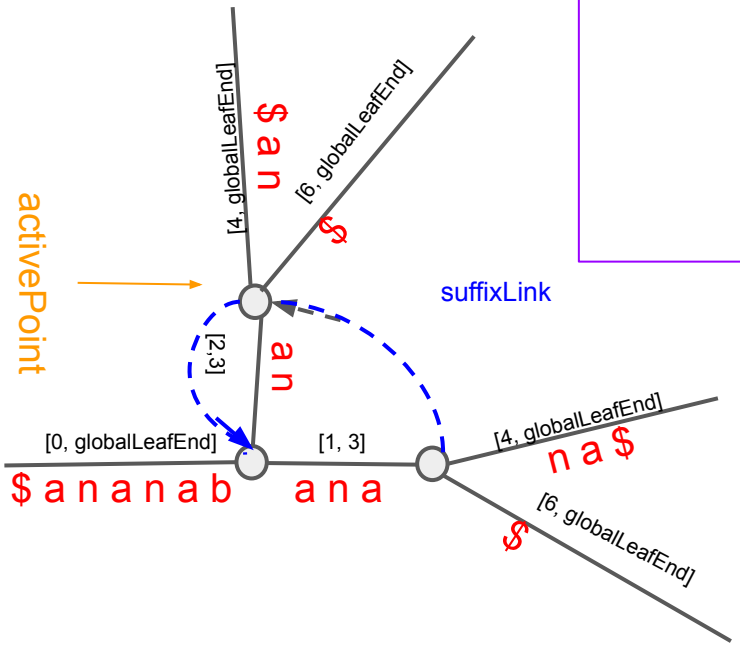


One example – Phase6,Extension4 End
Phase6, Extension5 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6

phase6
extension 4

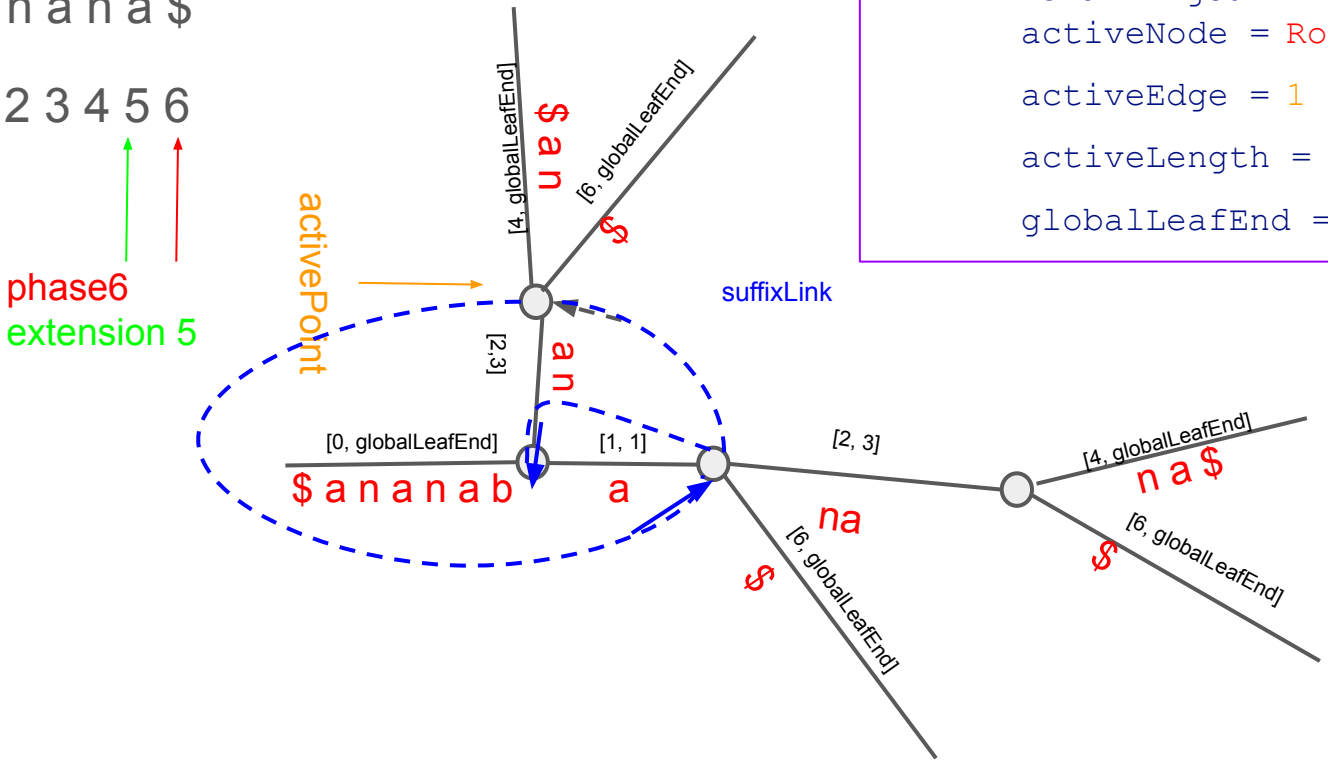



```
remainingSuffixCount = 2
activeNode = Root
activeEdge = 2
activeLength = 2
globalLeafEnd = 6
```

One example – Phase6,Extension5 End
Phase6, Extension6 Start

Input: b a n a n a $

Index 0 1 2 3 4 5 6




```
remainingSuffixCount = 1
activeNode = Root
activeEdge = 1
activeLength = 1
globalLeafEnd = 6
```

One example – Phase6,Extension6 End

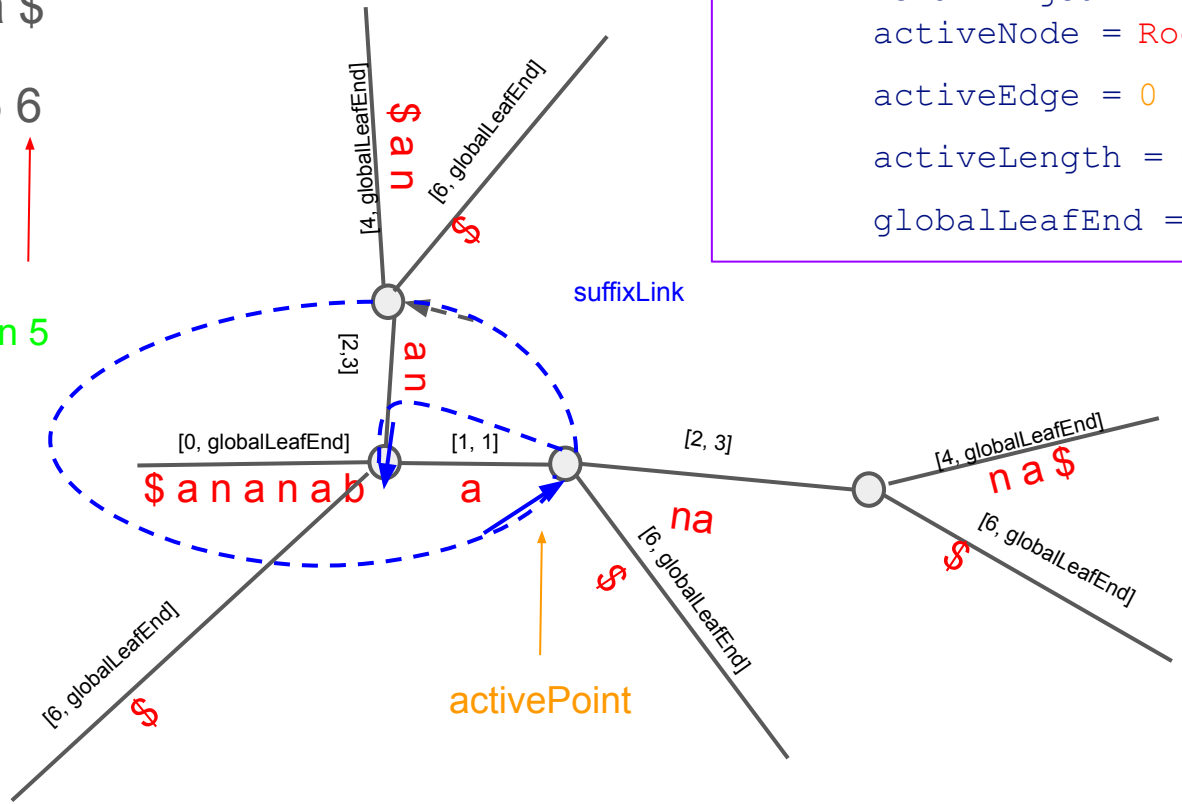
Input: b a n a n a $

Index 0 1 2 3 4 5 6

phase6

extension 5

```
remainingSuffixCount = 0
activeNode = Root
activeEdge = 0
activeLength = 0
globalLeafEnd = 6
```



My implementation performance

Input Size	1000	2000	5000	10000	100000
My implementation (time in seconds)	0.0259	0.0564	0.1508	0.3264	3.5807
The library (time in seconds)	0.0249	0.0556	0.1753	0.3064	3.3367

Most Challenging Part

- Following the rules are not very easy
- How to implement the extensions for each phase
- Need to deal all the suffixLinks correctly