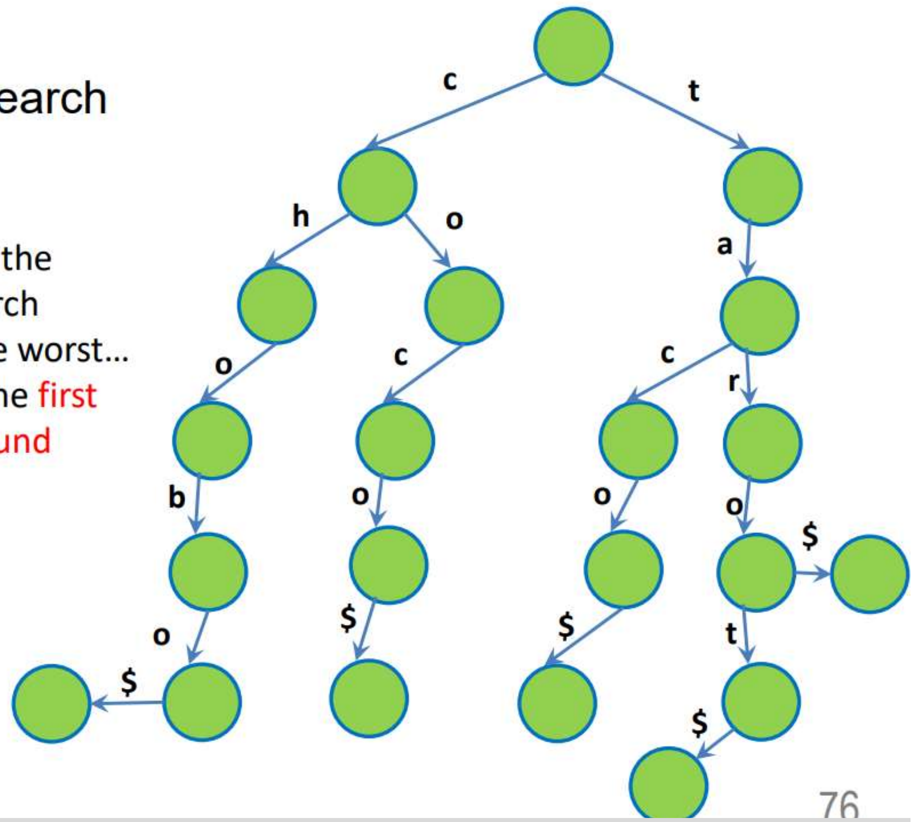
Suffix Trie

Wednesday, 1 June, 2022 22:02

Tries
Efficient string retrieval



- So how do we search for retrieval?
  - Complexity?
    O(M) where M is the length of the search string… This is the worst…
    O(1) best when the first character isn't found

76

Tries
Efficient string retrieval

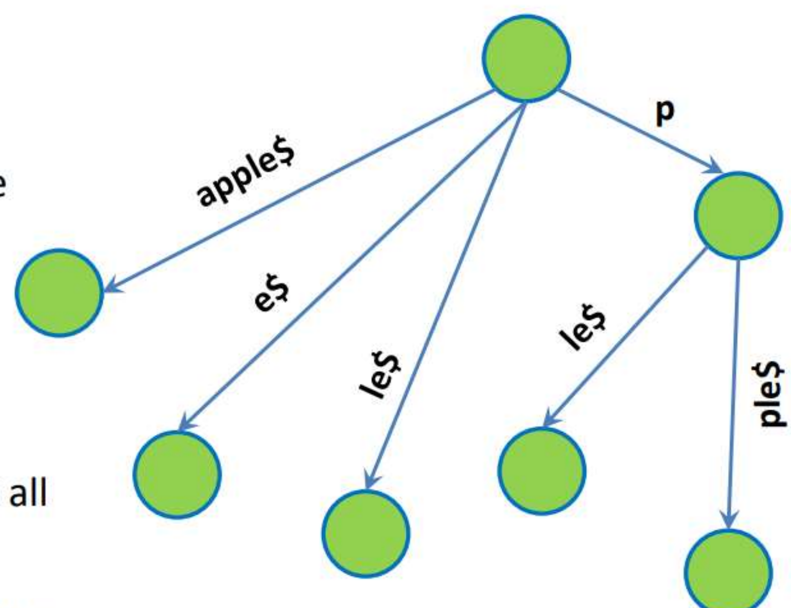
- Benefits?
  - Better for string search than BST/ AVL
  - More versatile than hash table
  - Search is O(M), where M is length of string
  - Can sort very quickly by traversing the string
    - The edges/ links are in-order (from a to z)
    - This is O(MN)
- Disadvantage?
  - At times can be slower than hash table
  - Wasted space if the self.link array is left empty most of the time

Tries
Efficient string retrieval

- Benefits?
  - Better for string search than BST/ AVL
  - More versatile than hash table
  - Search is O(M), where M is length of string
  - Can sort very quickly by traversing the string
    - The edges/ links are in-order (from a to z)
    - This is O(MN)
- Disadvantage?
  - At times can be slower than hash table
  - Wasted space if the self.link array is left empty most of the time
- Space complexity?
  - O(N^2)
  - N suffixes, longest suffix is N character
  - Have N number of leaves!

Suffix Tree
A tree, not a trie

- What is a suffix tree?
  - Using our same example
- What is our space complexity?
  - O(N^2) still because we still store the characters all
- When asked in the exam…
  - Draw a suffix trie

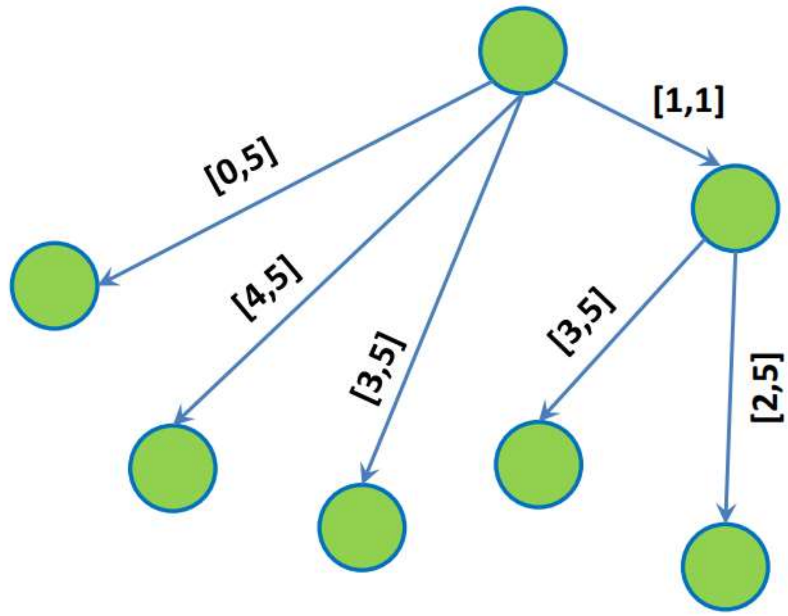


- When asked in the exam…
  - Draw a suffix trie
  - Then compress to suffix tree
- Note: Some like to separate out the $ node

Suffix Tree

A tree, not a trie

- Space complexity?
  - O(N)
  - N leaves
  - Each non-leaf node has at least 2 children
  - Total number of node = O(N + N/2 + N/4 + …) = O(N)
- Time complexity remains O(N^2) as we still need to insert every suffix with N character max



a	p	p	l	e	$
0	1	2	3	4	5

Suffix Array

Thursday, 2 June, 2022 23:41

- What is the complexity of this?
  - N is the number of character
  - O(N^2) to generate the suffixes
  - O(N^2 log N) to sort the suffixes with merge sort
    - Note O(N) for string comparison
    - We can reduce this to O(N^2) with radix sort
      - N passes (columns)

Time complexity:
Merge sort = O(N^2 log N) (* n for string comparison)
Radix sort = O(N^2) (N passes (columns))
Prefix doubling = O(N^2 log N) (* n for string comparison)
Prefix doubling O(1) comparison = O(N log^2 N)

Suffix Array
With prefix doubling with O(1) comparison

MONASH University

- Time complexity is O(N^2) with radix sort
  - Can we do better?
  - Yes with prefix doubling!
    - K character is sorted
    - So when we calculate the next K character (total of 2K)
      - We reuse the sorted first K
      - This is possible since they are suffixes of the same string!
  - But complexity still the same O(N^2 log N), even slower than radix sort
    - Due to the O(N) comparison
    - We use the rank table to get O(1) comparison!
    - So we can use this to sort very quickly within the same rank
    - So complexity now is O(N log^2 N)
- Space complexity is O(N) with suffix ID

Past Year 2021

Trie, Suffix Tree and Suffix Arrays

Question 12

Assume that we are constructing the **suffix array** for a string *S* using the **prefix doubling approach**. We have **already sorted** the suffixes for string *S* according to their **first 2 characters**; with the corresponding **rank array** shown below:

2 Marks

Suffix ID	1	2	3	4	5	6	7	8	9	10	11
Rank	4	6	5	7	4	6	3	5	7	2	1

We are now **sorting on the first 4 characters.**

a) Provide an example of two suffix IDs whose relative order has not been determined at this point. Justify your example.

b) Describe how this situation is resolved in the current iteration of prefix doubling.

c) State the resulting order between the suffixes in your example, after this resolution.

a) ID2 and ID6 since they both have the same rank (6)
b) Since both ID2 and ID6 has same rank we check for rank[2+2] (7) and rank[6+2] (5). ID8 has a higher rank than ID4.
c) ID6 before ID2

Past Year 2020 Sem1

Suffix Array
Question 16

Assume that we are constructing the **suffix array** for a **string** *S* using the **prefix doubling** approach. We have already sorted the suffixes for string *S* according to their first *4* **characters**; with the corresponding **rank array** shown below:

3 Marks

Suffix ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Rank	9	8	3	14	5	15	6	13	3	12	7	9	11	2	1

We are now sorting on the first **8** characters.

Describe how will you compare the following two suffixes on their first *8* characters in *O(1)*:

- Suffixes with ID 2 and ID 3
- Suffixes with ID 3 and ID 9

- ID3 is before ID2 since rank[3] is 3 and rank[2] is 8.
- ID3 and ID9 has the same rank which is 3. So we compare rank[3+4] and rank[9+4] since we have already sorted first 4 characters. Rank of ID7 is 6 and rank of ID13 is 11. So ID7 is higher than ID13. So ID3 before ID9.

Past Year 2020 Sem2

Suffix Array
Question 16

Assume that we are constructing the **suffix array** for a string *S* using the **prefix doubling approach**. We have **already sorted** the suffixes for string S according to their **first 2 characters**; with the corresponding **rank array** shown below:

3 Marks

Suffix ID	1	2	3	4	5	6	7	8	9	10	11
Rank	4	6	5	7	4	6	3	5	7	2	1

We are now **sorting on the first 4 characters.**

For the following pairs of suffixes, describe how will you **compare them on their first 4 characters in O(1)**, and **what the resulting order would be.**

1. Suffixes with ID 1 and ID 5
2. Suffixes with ID 3 and ID 5

- ID1 has rank[1] of 4 and ID5 has rank[5] of 4. So we check for rank[1+2] and rank[5+2]. Which ID7 is before ID3. Which means ID5 is before ID1.
- ID3 has rank[3] of 5 and ID5 has rank[5] of 4. ID5 before ID3.

Suffix Array Studio12 Q6

Friday, 3 June, 2022 00:31

Problem 6. Consider the prefix doubling algorithm applied to computing the suffix array of JARARAKA$. Write the partially-sorted suffix array after the length two prefixes have been sorted. Write the corresponding rank array and perform the next iteration of prefix doubling, showing the partially-sorted suffix array for the length four prefixes.

$O(1)$

if rank[a] = rank[b]

↓

rank[a+k] vs rank[b+k]

Text	J	A	R	A	R	A	K	A	$
	1	2	3	4	5	6	7	8	9

J	A	R	A	R	A	K	A	$
1	2	3	4	5	6	7	8	9

ID	Text[ID]
1	J
2	A
3	R
4	A
5	R
6	A
7	K
8	A
9	$

counting sort =>

k=1

Sort(text[ID])	ID	rank
$	9	1
A	2	2
A	4	2
A	6	2
A	8	2
J	1	3
K	7	4
R	3	5
R	5	5

merge sort =>

k=2

Sort(text[ID])	ID	rank
$	9	1
A$	8	2
AK	6	3
AR	2	4
AR	4	4
JA	1	5
KA	7	6
RA	3	7
RA	5	7

ID2 vs ID6 (k = 1)
rank[2] = 2 vs rank[6] = 2
rank[6+1] = 5 rank[6+1] = 4
ID6 before ID2

ID4 vs ID6
rank[4] = 2 vs rank[6] = 2
rank[4+1] = 5 vs rank[6+1] = 4
ID6 before ID4

ID2 vs ID4
rank[2] = 2 vs rank[4] = 2
rank[4+1] = 5 rank[4+1] = 5
ID2 == ID4

ID3 vs ID5
rank[3] = 5 vs rank[5] = 5
rank[3+1] = 2 rank[5+1] = 2
ID3 == ID5

=>

k=4

Sort(text[ID])	ID	rank
$	9	1
A$	8	2
AKA$	6	3
ARAK	4	4
ARAR	2	5
JARA	1	6
KA$	7	7
RAKA	5	8
RARA	3	9

ID2 vs ID4 (k = 2)
rank[2] = 4 vs rank[4] = 4
rank[2+2] = 4 rank[4+2] = 3
ID4 before ID2

ID3 vs ID5
rank[3] = 7 vs rank[5] = 7
rank[3+2] = 7 vs rank[5+2] = 6
ID5 before ID3

=>

k=8

Sort(text[ID])	ID	rank
$	9	1
A$	8	2
AKA$	6	3
ARAKA$	4	4
ARARAKA$	2	5
JARARAKA	1	6
KA$	7	7
RAKA$	5	8
RARAKA$	3	9

=>

k=16

Sort(text[ID])	ID	rank
$	9	1
A$	8	2
AKA$	6	3
ARAKA$	4	4
ARARAKA$	2	5
JARARAKA$	1	6
KA$	7	7
RAKA$	5	8
RARAKA$	3	9