

۱. [۳۲نمره] درست یا نادرست بودن سوالات زیر را همراه با دلیل مشخص کنید.

- (a) T The Burrows Wheeler Transform is used for pattern matching.

True. See 2nd lecture in the String module.

- (b) T The "Trie" data structure can be used to match  $n$  patterns of maximum length  $P$  against a string of length  $M$  in  $O(M \times P)$  independent of  $n$ .

True. Matching a string against a Trie is in the order of the depth of the tree which is  $P$ . Repeating this for each position is  $O(M \times P)$ .

- (c) T The Burrows Wheeler Transform is used for data compression.

True. The

$BW$

transforms brings repeating symbols/characters together (as much as possible) which enables better data compression.

- (d) F A Suffix Trie can be stored with  $O(N \times \log N)$  memory consumption.

False. For a string of length  $N$ , all suffixes have  $O(N^2)$  number of characters together. Memory consumption in a Trie is order of the total number of characters in the Trie which is  $O(N^2)$ .

- (e) F Storing a Suffix Trie requires more memory than a Suffix Tree, but Suffix Trie is faster for pattern matching.

False. While suffix tree consumes less memory due to the use of indices into the string but Suffix Tree is just as fast as a Suffix Trie.

- (f) F A SuffixArray cannot be used for pattern matching directly. However for a string of length  $T$  it can be used to build a Suffix Tree in  $O(T)$  and then used for pattern matching against a pattern of length  $P$  in  $O(P)$

True. A Suffix Array can be used for pattern matching. See Q3 in assignment 7.

- (g) F Given algorithms learned in class, for a very long string and many short patterns, using the Knuth-Morris-Pratt algorithm is the most efficient way to find all occurrences of the short patterns in the long string.

KMP order is  $O(P + T)$  for a single matching. For  $n$  patterns with max length of  $P$ , it is  $O(n \times (P + T))$ . Building a Suffix Tree is  $O(T \log T)$ . Matching a single pattern against suffix tree is  $O(P)$ . So matching all  $n$  patterns against the suffix tree would be  $O(P \times n)$ . Therefore the total order using a Suffix Tree would be  $O((T \log T) + P \times n)$ . Since  $n$  and  $T$  are both large numbers, the KMP solution is less efficient since it contains a  $n \times T$  term.

۲. [۳۰نمره] رشته mississippi را در نظر گرفته و در رابطه با آن به سوالات زیر پاسخ دهید.

(آ) [۵نمره] تبدیل را برای این رشته بدست آورید و در کادر زیر وارد کنید.

Answer:  
ipssm\$piissii

Rotations:  
mississippi\$  
ississippi\$m  
ssissippi\$mi  
sissippi\$mis  
issippi\$miss  
ssippi\$missi  
sippi\$missis  
ippi\$mississ  
ppi\$mississi  
pi\$mississip  
i\$mississipp  
\$mississippi

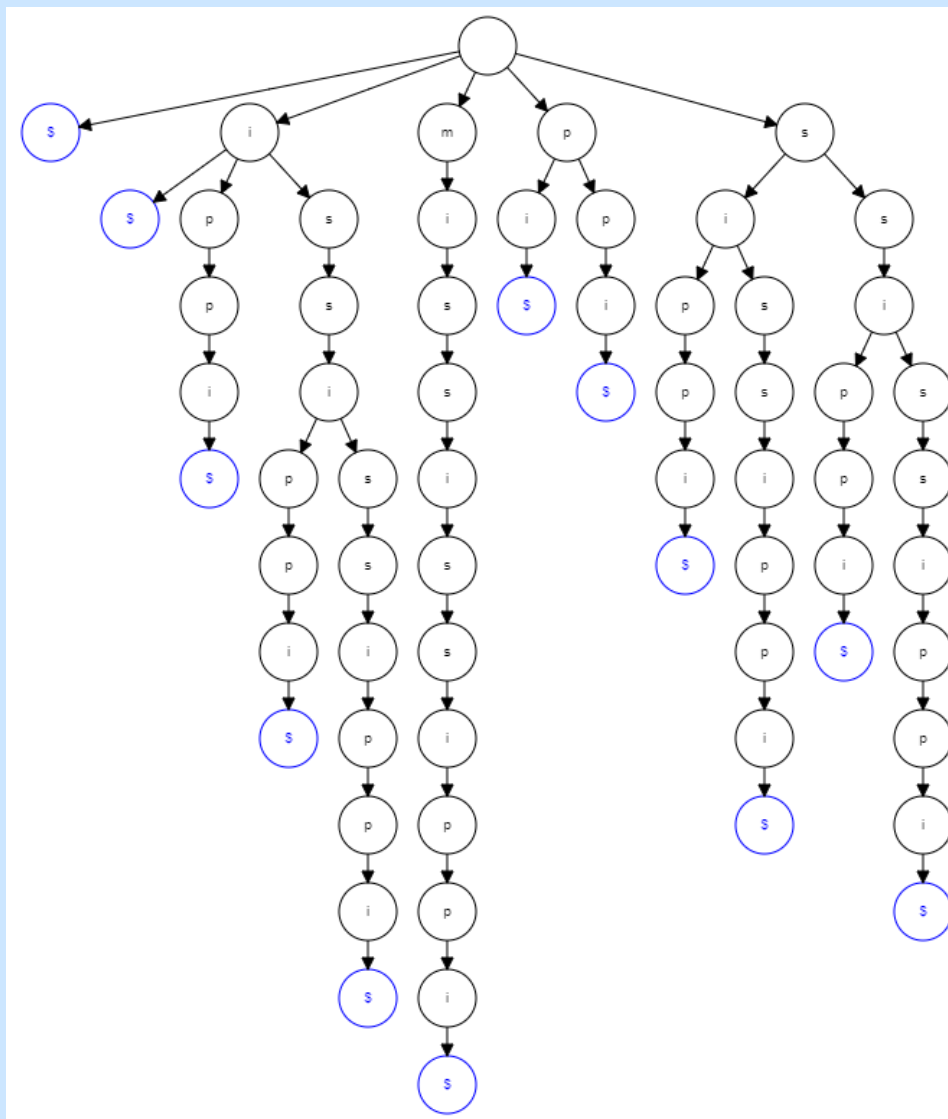
Sorted Rotations:

\$mississippi  
i\$mississipp  
ippi\$mississ  
issippi\$miss  
ississippi\$m  
mississippi\$  
pi\$mississip  
ppi\$mississi  
sippi\$missis  
sissippi\$mis  
ssippi\$missi  
ssissippi\$mi

Transform (Last Column):

ipssm\$piissii

(ب) [۵نمره] Suffix Trie متناظر با این رشته را رسم کنید.



شکل ۱: Generated By: <https://people.ok.ubc.ca/ylocet/DS/Trie.html>

(ج) [۵نمره] Suffix Tree متناظر با این رشته در حالتی که پیچیدگی حافظه برای آن  $O(|Text|)$

باشد را رسم کنید.

```

      | (1:12) | leaf
tree: |
      |      |      | (6:12) | leaf
      |      | (3:5) |
      |      |      | (9:12) | leaf
      | (2:2) |
      |      | (9:12) | leaf
      |      |
      |      | (12:12) | leaf
      |
      |      |      | (6:12) | leaf
      |      | (4:5) |
      |      |      | (9:12) | leaf
      | (3:3) |
      |      |      | (6:12) | leaf
      |      | (5:5) |
      |      |      | (9:12) | leaf
      |
      |      | (10:12) | leaf
      | (9:9) |
      |      | (11:12) | leaf
      |
      | (12:12) | leaf

```

Generated By: <http://www.allisons.org/ll/AlgDS/Tree/Suffix/>

(د) [۱۰نمره] با توجه به الگوریتم  $O(n \log n)$  آرایه شده در درس جهت بدست آوردن SuffixArray جداول زیر را پر کرده و سپس آرایه LCP را محاسبه کنید.

Index	CyclicShifts1	Index	SortdCyclicShifts1	CyclicShifts1-idx	CyclicShifts2	CyclicShifts1-idx	Index	SortdCyclicShifts2	CyclicShifts2-idx	CyclicShifts4	CyclicShifts2-idx	Index	SortdCyclicShifts4
0 m		11 \$		11 \$ m		0	11 \$	m		11 \$ m i s	1	11 \$	m i s
1 i		1 i		1 i s	2		10 i	\$		10 i \$ m i	0	10 i	\$ m i
2 s		4 i		4 i s	5		7 i	p		7 i p p i	9	7 i	p p i
3 s		7 i		7 i p	8		1 i	s		1 i s s i	3	1 i	s s i
4 i		10 i		10 i \$	11		4 i	s		4 i s s i	6	4 i	s s i
5 s		0 m		0 m i	1		0 m	i		0 m i s s	2	0 m	i s s
6 s		8 p		8 p p	9		9 p	i		9 p i \$ m	11	9 p	i \$ m
7 i		9 p		9 p i	10		8 p	p		8 p p i \$	10	8 p	p i \$
8 p		2 s		2 s s	3		3 s	i		3 s i s s	5	6 s	i p p
9 p		3 s		3 s i	4		6 s	i		6 s i p p	8	3 s	i s s
10 i		5 s		5 s s	6		2 s	s		2 s s i s	4	5 s	s i p
11 \$		6 s		6 s i	7		5 s	s		5 s s i p	7	2 s	s i s

CyclicShifts4-idx	CyclicShifts8	CyclicShifts4-idx
11	\$ m i s s i s s	3
10	i \$ m i s s i s s	2
7	i p p i \$ m i s s	11
1	i s s i s s i p s	5
4	i s s i p p i \$ s	8
0	m i s s i s s i s	4
9	p i \$ m i s s i s	1
8	p p i \$ m i s s s	0
6	s i p p i \$ m i s	10
3	s i s s i p p i s	7
5	s s i p p i \$ m s	9
2	s s i s s i p p s	6

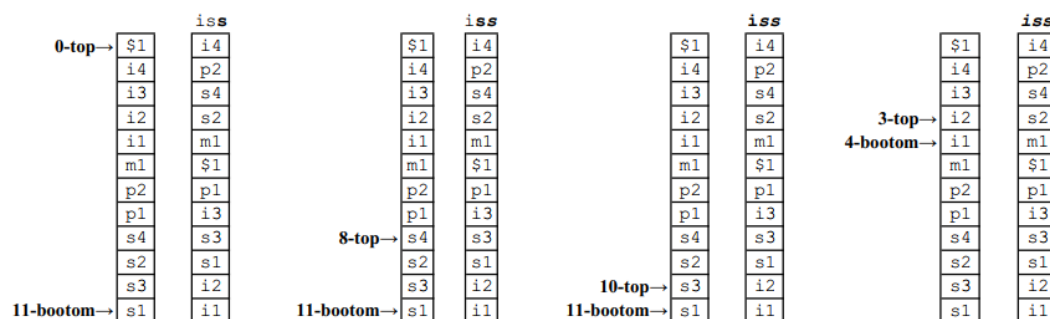
Index	CyclicShifts8
11	\$ m i s s i s s
10	i \$ m i s s i s s
7	i p p i \$ m i s s
4	i s s i p p i \$ s
1	i s s i s s i p s
0	m i s s i s s i s
9	p i \$ m i s s i s
8	p p i \$ m i s s s
6	s i p p i \$ m i s
3	s i s s i p p i s
5	s s i p p i \$ m s
2	s s i s s i p p s

CyclicShifts8-idx	CyclicShifts16	CyclicShifts8-idx
11	\$ m i s s i s s i p p i \$ m i s	7
10	i \$ m i s s i s s i p p i \$ m i	6
7	i p p i \$ m i s s i s s i p p i	3
4	i s s i p p i \$ m i s s i s s i	0
1	i s s i s s i p p i \$ m i s s i	9
0	m i s s i s s i p p i \$ m i s s s	8
9	p i \$ m i s s i s s i p p i \$ m	5
8	p p i \$ m i s s i s s i p p i \$	4
6	s i p p i \$ m i s s i s s i p p	2
3	s i s s i p p i \$ m i s s i s s	11
5	s s i p p i \$ m i s s i s s i p	1
2	s s i s s i p p i \$ m i s s i s	10

Index	CyclicShifts16	LCP
11	\$ m i s s i s s i p p i \$ m i s	
10	i \$ m i s s i s s i p p i \$ m i	0
7	i p p i \$ m i s s i s s i p p i	1
4	i s s i p p i \$ m i s s i s s i	1
1	i s s i s s i p p i \$ m i s s i	4
0	m i s s i s s i p p i \$ m i s s s	0
9	p i \$ m i s s i s s i p p i \$ m	0
8	p p i \$ m i s s i s s i p p i \$	1
6	s i p p i \$ m i s s i s s i p p	0
3	s i s s i p p i \$ m i s s i s s	2
5	s s i p p i \$ m i s s i s s i p	1
2	s s i s s i p p i \$ m i s s i s	3

suffixArray      11 10 7 4 1 0 9 8 6 3 5 2

(ه) [۵نمره] با استفاده از تبدیل  $BW$  که در قسمت الف بدست آوردید الگوی iss را درون رشته پیدا کنید. سپس با کمک ArraySuffix ایندکس وقوع الگو در رشته را مشخص کنید.

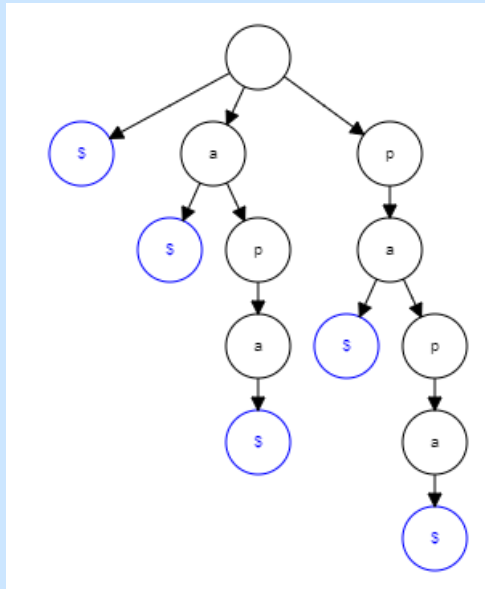


ایندکس وقوع 1, 4

ایندکس‌های وقوع الگو در متن را با توجه به SuffixArray مشخص کنید:

با الگو برداری از نحوه ساخت رشته به کمک تبدیل  $BW$  در ابتدا اندیس هر کاراکتر را در کنار خود کاراکتر ذخیره می‌کنیم. سپس کاراکترها را مرتب کرده در یک لیست قرار می‌دهیم. از ابتدای لیست اندیس هر کاراکتر را بازیابی کرده و کاراکتر موجود در مقدار اندیس منهای یک را در خروجی قرار می‌دهیم. خروجی تبدیل  $BW$  است.

مثال درخت پسوندی برای رشته papa یکبار بدون \$ و یکبار با \$ رسم شود. مشاهده می شود برای رشته هایی که یک زیررشته از خودشان درونشان هست به مشکل می خوریم و مثلاً برای تطبیق رشته بدون ذخیره اطلاعات بیشتر در کمک درخت نمی توانیم این کار را به درستی انجام دهیم. همچنین روش های تولید SuffixArray هم نیاز به بازبینی پیدا می کند.



```

graph TD
    Root(( )) --> a1((a))
    Root --> p1((p))
    a1 --> p2((p))
    p1 --> a2((a))
    p2 --> a3((a))
    a2 --> p3((p))
    p3 --> a4((a))
    style Root fill:#fff,stroke:#000
    style a1 fill:#add8e6,stroke:#000
    style p1 fill:#fff,stroke:#000
    style p2 fill:#fff,stroke:#000
    style a2 fill:#add8e6,stroke:#000
    style a3 fill:#add8e6,stroke:#000
    style p3 fill:#fff,stroke:#000
    style a4 fill:#add8e6,stroke:#000
  
```

$P + T$	a	b	a	b	a	b
$\pi$	0	0	1	2	3	4

با تشکر ویژه از خانم مهسا سادات رضوی و آقای امیر خاکپور که در طراحی و آماده سازی سوال‌ها و پاسخنامه و همچنین برگزاری امتحان نقش بسزایی داشتند.