STMC HKOI Training

Lesson 7: String Manipulation (I)

Chan Yan Mong

March 23, 2022

Goal today

String and list are similar so we can do similar operations (Slice, membership, elementwise access etc.) on it.

- Relation between string and list
- ASCII encoding, `ord()` function
- Length, slicing, substring and membership
- Using `split` and `join`
- Knuth–Morris–Pratt (KMP) algorithm

Character Encoding

- In computers, all data are stored as 0s and 1s
- Hence, all data stored in computer are fundamentally just numbers
- However, similar to how a string of integers like 21 can represent either your class number or the money in your pocket, *what a given string of numbers mean are up to us*.
- A scheme that assigns numbers to graphical characters is called a **character encoding**

Analogy: Morse Code

- To illustrate the idea, consider the **Morse Code**
- Can't send characters directly via telegraph
- So characters are turned into sequences of two different signal durations, called dots and dashes, that can be sent
- To retrieve the characters, people only need to look up what these sequences mean

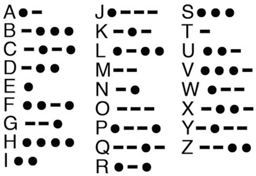


Figure 1: Morse Code (Source)

Character Encoding

- A character encoding does the same thing, except it associate a character with an integer
- For example, the ASCII encoding encode `'A'` with 65, `'B'` with 66 and etc.
- Commonly used character encodings include: **ASCII**, **Unicode**, **Big5**, **GB**
- Here we will focus on **ASCII**, one of the oldest but also simplest character encoding scheme

Here is the ASCII Table (Source: alpahrithms)

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051	)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[	123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	|
29	1d	035	GS	61	3d	075	=	93	5d	135	]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

www.alpharithms.com

Getting ASCII value using `ord()`

- In Python, we can get the ASCII value of a character using `ord()`
- For example:

```
ord('A') # Return 65
ord(' ') # Return 32
ord('a') # Return 97
```

- **Exercise:** Using `ord()`, find the ASCII value of `'B'`,`'c'`,`'0'`,`'#'`,`'!'`. Compare the results with the ASCII table above

String and List

- Now we can understand what actually is a string
- Fundamentally, strings are list of characters
- Take the string `"Hello World!"` for example
- It can be thought as a list of 12 characters:
  `['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd','!']`
- Therefore, almost all the operations we did with list can be transfered to string

Length of String

- Like list, we can obtain the length of string using `len()` function

```
len('Hello!') # Return 6
len('Python is fun! (Nope)') # Return 21
len('13+19=32') # Return 8
```

Looping Through String

- Using this, we can easy loop over strings like we did with list

```
mySTR = "This is some long long sentence"
for i in range(len(mySTR)):
  print(mySTR[i]) # Print the string character by character
```

- For example, we the following code search for the indicies of the character `'o'`

```
mySTR = "This is some long long sentence"

for i in range(len(mySTR)):
  if mySTR[i] == 'o':
    print(f'Find an "o" at: {i}!')
```

Finding substrings

- **Substring** is a continguous sequence of characters within a string
- For example, `"the best of"` is a substring of `"it was the best of time"`
- In python, we can find the substring using the keyword `in`. For example:

```python
"Hire" in "Hire the top freelancers" # True
"Ben" in "Kelvin is handsome" # False
"Free" in "I am not Free" # True
"may" in "May is a good person" # False
```

Finding substrings

- Furthermore, we can get the starting index of the substring using `find` method

```python
# Indices  0123456789012345678901234567890123456789012345
message = 'Python is a fun programming language'

message.find('fun') # 12
message.find('program') # 16
message.find('language') # 28
```

Slicing string

- We can also slice string using similar syntax as that in string

```
  myString[start:end]
```

where the `end` is the index for the end of the slice, with `end` excluded

- For examples:

```
# Indices  012345678901234567890123456789012345
message = 'Python is a fun programming language'

message[0:6] # Python
message[12:15] # fun
message[16:23] # program
```

Editing characters in strings

- Strings are **immutable** in python (unlike in C/C++)
- This means the following is invalid:

```python
myStr = "Hello World"
myStr[0] = 'h'
# TypeError: 'str' object does not support item assignment
```

- In general, you cannot modify characters in string directly
- To modify string, one can first convert a string to list and convert it back to string

Editing characters in strings

- To convert string to list, we can use `list`
- To convert list to string, we can use `join`
- Here is an example:

```
myStr = "Hello"
myStr = list(myStr) # ['H','e','l','l','o']
myStr[0] = 'h' # ['h','e','l','l','o']
myStr = ''.join(myStr) # 'hello'
```

More on `join` method

- More generally, the `join` method takes all items in a list into a string, using `'sep'` as separator:

```
'sep'.join(myList)
```

- For example:

```
','.join(['May','John','Tim']) # 'May,John,Tim'
' '.join(['May','John','Tim']) # 'May John Tim'
'!!'.join(['May','John','Tim']) # 'May!!John!!Tim'
```

split method

- We can also split string into list with separators using the `split` method
- For example:

```
'Tim,Amy,John'.split(',') # ['Tim','Amy','John']
'These are some words'.split(' ') # ['These','are','some','words']
'apple#banana#cherry#orange'.split('#') # ['apple', 'banana', '
    cherry', 'orange']
```

- This is sometimes useful when processing text
- Those who are interested in more advanced text processing should search for **regex** (regular expression)

Practicing String Handling

Exercise: King movement

HKOI Online Judge (D110): `https://judge.hkoi.org/task/D110`

Exercise: Phone number

HKOI Online Judge (D101): `https://judge.hkoi.org/task/D101`

Exercise: Bus fare

HKOI Online Judge (D102): `https://judge.hkoi.org/task/D102`

Exercise: String length and words

HKOI Online Judge (D302): `https://judge.hkoi.org/task/D302`

String-Search Algorithms (Optional)

- The method `find` is surely a useful string utility provided by python
- But *how do they work?*
- Here we will introduce a commonly used string search algorithm called **Knuth–Morris–Pratt (KMP) algorithm**
- But first let's try to see how we would approach the problem naively

Algorithm 1 Naive String Search

Input: $T[1 \cdots n], P[1 \cdots m]$ - String and pattern to be searched with length n and m respectively

Program:

for all $i = 1$ to $n - m + 1$ **do**
  TotalMatch $\leftarrow$ True
  for all $j = 1$ to m **do**
    if $T[i + j - 1]$ not equal $P[j]$ **then**
      TotalMatch $\leftarrow$ False
    end if
  end for
  if *TotalMatch* is True **then**
    Print "A match is found at $s = i$
  end if
end for

Naive String Search

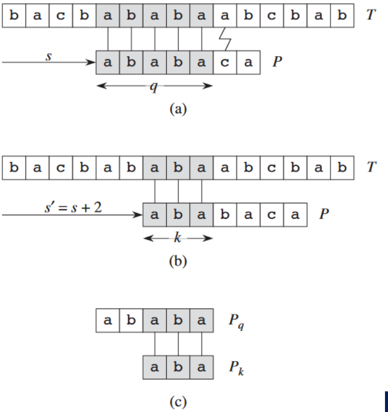
- As one can see, the naive algorithm will run for $(n - m + 1)m$ times
- So the time complexity is $\mathcal{O}(m(n - m + 1))$
- As one can see, the algorithm is linear in n but quadratic in m, so this is most inefficient when $m \approx n/2$
- However, it can be shown that for randomly chosen patterns and text from a set of d alphabets the *expected* number of character-to-character comparison is $\leq 2(n - m + 1)$ (See Introduction to Algorithms - Problem 32.1-3)
- Thus, for randomly chosen strings, the naive algorithm is quite efficient

Why naive algorithm is slow

- However, we can do better. To see how we can improve, let's first see why the naive algorithm is slow
- Our discussion will follow closely that of Ch.32.4 of *Introduction to Algorithms* by Cormen et. al.
- The images below are extracted from the book

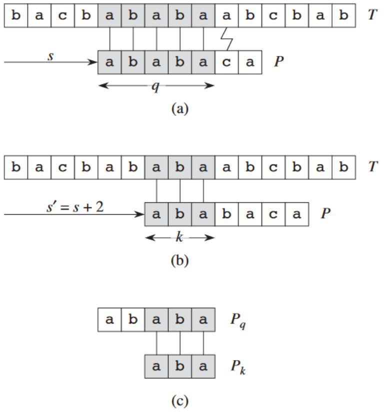
Why naive algorithm is slow

- Let $P =$ "ababaca" be the pattern and it aligns with the text T so that the first $q = 5$ characters match
- Now we found that the 6th character does not match (Connected by zig-zag lines)
- We must now shift to the right to find another match
- But shift by how much?



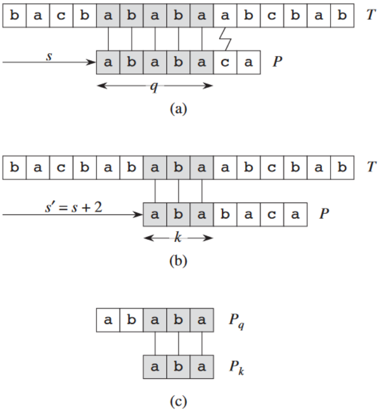
Why naive algorithm is slow

- In the *naive* algorithm, we always shift by 1 character ($s' = s + 1$)
- But as shown in (a), this is inefficient
- If we shift $s' = s + 1$, the substring in T will be `"babaabcbab..."`. In other words, it starts with the 2nd character in P
- But the 2nd character in P is `"b"` while the first is `"a"`. There is no way they can match



Why naive algorithm is slow

- Instead, we can be more efficient by shifting $s' = s + 2$ (See (b))
- Shifting more allow us to check less before hitting the end of the string. This speeds up the algorithm
- Furthermore we also found that the first k has been matched, so we can sart checking from $s' = s + k + 2$
- This also speeds up the computation

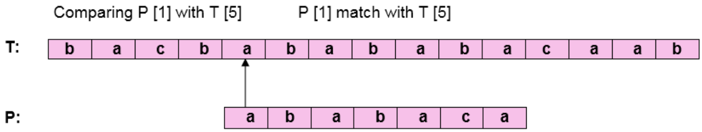


How many characters to skip?

- Now comes the question: *How many character should I skip?*
- More importantly: *Can I precompute them using only the pattern string P*?
- To answer these problems we can consider some examples.

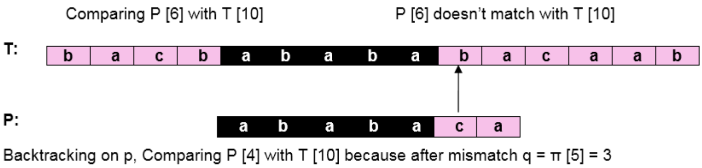
How many characters to skip

Example: $T = bacbabababacaab$; $P = ababaca$



Extracted from javatpoint (KMP algorithm)

How many characters to skip



Extracted from javatpoint (KMP algorithm)

How many characters to skip

- This example lead us to the following observations
  - Given that the pattern characters $P[1 \cdots q]$ match text characters $T[s+1 \cdots s+q]$, then:
  - Naively, to skip the most characters, we would like to mentally shift $P[1 \cdots q]$ to the end of $T[s+1 \cdots s+q]$
  - But this is wrong, because we might skip useful characters
  - So we go backward, trying to match the back of $P[1 \cdots q]$ with the head of $P[1 \cdots q]$ as much as possible
  - That is, we want to shift by an amount $\pi[q]$ such that:

$$\boxed{\pi[q] = \max\{k : k < q \quad \text{and} \quad P[1 \cdots k] \text{ is a suffix of } P[1 \cdots q]\}}$$

  - This is the key to **Knuth–Morris–Pratt algorithm**

Knuth–Morris–Pratt algorithm (Optional)

- The KMP algorithm was developed by James H. Morris and independently discovered by Donald Knuth. Morris and Vaughan Pratt published a technical report in 1970 ("Knuth–Morris–Pratt algorithm", Wikipedia)
- The worse case time complexity is $\Theta(m) + \Theta(n)$

Algorithm 2 KMP Match

$T[1 \cdots n], P[1 \cdots m]$
$\pi, q \leftarrow$ Compute-Prefix-Function(P)$, 0$
for all $i = 1$ to n **do**
    while $q > 0$ and $P[q+1] \neq T[i]$ **do**
        $q \leftarrow \pi[q]$
    end while
    if $P[q+1] == T[i]$ **then**
        $q \leftarrow q + 1$
    end if
    if $q == m$ **then**
        Print "Pattern occurs with shift " $i - m$
        $q \leftarrow \pi[q]$
    end if
end for

Algorithm 3 Compute Prefix Function

```
P[1 ··· m], π[1 ··· m] ← Empty List
π[1] = 0
k = 0
for all q = 2 to m do
    while k > 0 and P[k + 1] ≠ P[q] do
        k ← π[k]
    end while
    if P[k + 1] == P[q] then
        k ← k + 1
    end if
    π[q] ← k
end for
Return π
```