蟻本 P.335

たのしい Suffix Array

by **Suu0313**

目次

やりたいこと

例題:

文字列 S, T が与えられます S の(連続する)部分文字列で T と一致するものはいくつありますか？

制約:

$1 \leq |S| \leq 1000000$
$1 \leq |T| \leq 10000$

愚直解

```
cnt = 0
for i in range(|S|):
  if S[i:i+|T|] == T:
    cnt++
```

$\Theta(|S||T|)$ くらい？

高速化

Suffix-Array (接尾辞配列)

文字通り、文字列のSuffix (接尾辞) の配列

を、辞書順に並び替えてえられるもの

例: "atcoder"

	接尾辞	ソート後
0	atcoder	atcoder
1	tcoder	coder
2	coder	der
3	oder	er
4	der	oder
5	er	r
6	r	tcoder

何が嬉しいのか

Suffix-Arrayをうまく使って部分文字列の探索を高速化したい

ん？

ソート済み配列 + 探索 → フッフッフw

伝家の宝刀†二分探索† が使えそう！

S の部分文字列と T の比較で lower_bound と upper_bound を出せば、そのインデックスの差が答え！

$\Theta(|T| \log |S|)$

Suffix-Arrayの構築

では、Suffix-Arrayはどうやって作りましょう

愚直:

愚直に全部切り出してソート

Suffixの数・長さはどちらも $\Theta(|S|)$

ということは、計算量は $\Theta(|S|^2 \log |S|)$

これでは本末転倒

考え方: ダブリングをしよう！

どの文字列も 1 文字だけなら普通にソートして $\Theta(\log |S|)$ ですね

その結果を利用して 2 文字、その結果を利用して 4 文字 $\ldots$ と高速にできないだろうか？

$rank_{k,i} :=$ Suffixの前 2^k 文字までだけの情報でソートした結果、$S[i:]$ は辞書順で何番目か (全く同じ文字列は同じ順位)

とりあえず $rank_{0,i} := S[i]$ とします

もし $rank_{k,i} < rank_{k,j}$ なら、

辞書順比較は前の文字で差があったらそれが絶対なので

$rank_{k+1,i} < rank_{k+1,j}$ となりますね

では、$rank_{k,i} == rank_{k,j}$ ならどうしましょう

答えは、

$rank_{k,i+2^k}$ と $rank_{k,j+2^k}$ を比べればいいです

前半 2^k 文字は一致したのでじゃあ無視して後半 2^k 文字を比べればいいというだけですね

2^{k+1} 文字だけ見て比べるので、ここでも一致した場合は $rank_{k+1,i} == rank_{k+1,j}$ となります

最後まで同率が残ってしまったらどうするんだ！

という気持ちになるかもしれませんが、Suffixの長さはすべて異なるので最終的にはすべて異なる数字が入ります

ということで、ダブリングの中でソートをするので計算量は $\Theta(|S| \log^2 |S|)$ となります

Library Checker: Suffix Array

問題文

長さ N の文字列 S が与えられます。S の suffix array a_i を求めてください。

$1 \leq N \leq 500,000$

$\Theta(N \log^2 N) : 745ms$

https://judge.yosupo.jp/submission/47223

まだ遅い！！！

蟻本「より高速なアルゴリズムも存在しますが、ほとんどの場合でこのアルゴリズムで十分です」

これはちょっと嘘です

確かに構築自体は間に合うことが多いですが、ほかのところの定数倍が少し大きくなるとTLEしてしまう… ということが割とよくあるようです

衝撃の事実:

$\log$ は定数じゃない

線形とは言わなくても $\log$ を一個落とせると結構変わります

ということで、 $\Theta(|S|\log^2|S|)$ を $\Theta(|S|\log|S|)$ にする方法を考えてみましょう

さきほどの方針で $\log$ が付いたのは二か所、ダブリングとソートです

ダブリングはこのアルゴリズムのキーとなる部分なので、ここは仕方がない感じがします…

ということで、ソートを線形にしましょう！

２要素の大小比較に基づくソートアルゴリズムは $\Omega(n \log n)$ の時間計算量を必要とする ということが知られているので、そうでないアルゴリズムを使いましょう

(正確には、$\Omega(n \log n)$ の時間計算量を必要とする入力例が存在するということを示せます)

ボゴソート

ボゴソートを知っていますか？私は知っています

数列をランダムにシャッフルします 運が良ければソートされていますね

シャッフルに $\Theta(|S|)$ かかるとして、運が良い人なら定数回で当ててくれるということを信じて $\Theta(|S|)$ でソートができました！

さすがにつらいものがあります

バケットソート

バケットソート(ビンソート)を知っていますか？私は知っています (どちらの呼び方が一般的なのでしょうか…)

整列したいデータの数が n 個、種類が k 個として $\Theta(n+k)$ でソートをすることができます

今回、数も $rank$ の種類も $|S|$ 個なので、$\Theta(|S|)$ でソートができるはずです

ただし、今回は $(rank_{k,i}, rank_{k,i+2^k})$ という順序対のソートがしたいので、よくあるバケットソートがそのままはできません

そこで、バケットソートを二回することを考えます

まずは $rank_{k,i+2^k}$ の方、つまり第二要素をもとにソートします

その次に $rank_{k,i}$ の方、つまり第一要素をもとに **安定ソート** します

すると、第一要素が同じなら第二要素の順で並んでいるはずなのでソートが完了します

ほかの部分は $\Theta(|S|\log^2|S|)$ の方法と同じようにすれば、$\log$ を落として $\Theta(|S|\log|S|)$ とすることができます！

Library Checker: Suffix Array

問題文

長さ N の文字列 S が与えられます。S の suffix array a_i を求めてください。

$$1 \leq N \leq 500,000$$

$\Theta(N \log^2 N) : 745ms$

https://judge.yosupo.jp/submission/47223

$\Theta(N \log N) : 293ms$

https://judge.yosupo.jp/submission/47257

この記事 を参考にさせていただきました

が、ソートを線形でやる発想は同じですが方針・実装は全く違うのでメモリ効率とか定数倍とかはあまりよくないかもしれないです

おまけ

まだ遅いですか？

$\Theta(|S| \log |S|)$ まで高速化できて、実際これで困ることはないんじゃないかと思いますが、実は $\Theta(|S|)$ で構築することができます

SA-IS という単語をTLなどで見かけることもあったかもしれません

SA-IS

Suffix Array Induced Sorting の略です

お気持ち

それっぽい線形ソートを何回かやったら全体がほんとにソートされてくれたらうれしいな

induced sort

さっきと同じでソートを線形でやりたいので、バケットソートのようなことをします

しかし、ちゃんとやると大変なのでだいたいあってればオッケーということにします

S 型と L 型

とりあえず先頭の文字だけでソート... だとさすがに弱すぎるので、少し性質を探してみます

$S[i:]$ と $S[i+1:]$ を辞書順比較します

$S[i:] < S[i+1:]$ なら $S[i:]$ は S 型
$S[i:] > S[i+1:]$ なら $S[i:]$ は L 型

と呼ぶことにします (長さが異なるので、等しくはなりません)

0	1	2	3	4	5	6	7
a	b	c	b	c	b	a	$
S	S	L	S	L	L	L	S

とりあえずこれが求めたいですが、部分文字列を本当に比較したらここで $\Theta(|S|^2)$ かかってしまいます

これを高速化するアイデアとして、後ろから順に埋めていく というものがあります

$S[i:]$ と $S[i+1:]$ を比較するとして、もし $S[i] \neq S[i+1]$ ならその先頭一文字だけで決まりますね

では $S[i] == S[i+1]$ のときは？

先頭一文字が同じなので、二文字目以降の辞書順と同じです

と、いうことは、$S[i+1:]$ と $S[i+2:]$ の比較結果と同じになるわけですね

後ろから埋めていけば、一個後ろを見ればいいのでこれば $\mathrm{O}(1)$ でわかります

ということで、$\Theta(|S|)$ で各Suffixが何型なのかがわかりました

それではこの型の性質を考えてみましょう

$S[i] = S[j]$ のとき、$S[i:]$ が S 型、 $S[j:]$ が L 型 なら $S[i:]$ と $S[j:]$ どちらが先に来るでしょうか？

正解はL型です

なぜ？

S型の文字列は $S[i:] < S[i+1:]$
L型の文字列は $S[j:] > S[j+1:]$

であることを考えます

$S[i:]$ の一文字目以降で初めて現れる $S[i]$ と異なる文字 $S[k]$ は、$S[i] < S[k]$ となっているはずです

$S[j:]$ の一文字目以降で初めて現れる $S[j]$ と異なる文字 $S[l]$ は、$S[j] > S[l]$ となっているはずです

そうでないと S 型 ・ L 型 にならないためです

bbbbc なら bbbbc < bbbc, bbbba なら bbbba > bbba

$S[l] < S[j] = S[i] < S[k]$ となるため、L 型 $<$ S 型 となります

LMS

ここで、~~連続する S 型のインデックスに対し一番左のものを LMS と呼ぶことにします~~
$S[i-1]$ が L 型で $S[i]$ が S 型 であるようなSuffixを LMS と呼ぶことにします

0	1	2	3	4	5	6	7
a	b	c	b	c	b	a	$
S	S	L	S	L	L	L	S
			↑				↑

ソートの手順

(1) LMSのインデックスをバケツの対応する先頭の文字の場所の後ろから挿入する

$	a	a	b	b	b	c	c
7					3		

頭文字がかぶってたら順番は何でもいいです (どうしようもないので)

ソートの手順

(2) バケツを前から見ていって、今見ているところに入っているインデックスの一つ前が指す場所が L 型 だったらそのインデックスをバケツに前から入れる

$	a	a	b	b	b	c	c
7					3		
↑							

$S[6:]$ は L 型で a から始まるので

$	a	a	b	b	b	c	c
7	6				3		
↑							

こうする

$	a	a	b	b	b	c	c
7	6		5		3	4	2

進めていくとこうなる

実は、こうすると L 型のインデックスがすべて埋まります

バケツへの挿入位置は必ず今見ているところより後に来ます

なぜ？

L 型 の文字列は $S[i-1:] > S[i:]$ であることを考えます

まず、頭文字が異なるとすると $S[i] < S[i-1]$ のはずです

頭文字が同じなら、 $S[i-1:]$ と $S[i:]$ がどちらも L 型とすると L 型は左から埋めているので今より右のものしか空いていません

$S[i:]$ が LMS だとすると、$S[i] = S[i-1]$ だと $S[i-1:]$ は S 型になってしまい、$S[i:]$ が LMS であることと矛盾してしまいます

よって、$S[i] < S[i-1]$ のはずです

ということで、L 型 のインデックスがすべて埋まりました

ソートの手順

(3) バケツを後ろから見ていって、今見ているところに入っているインデックスの一つ前が指す場所が S 型 だったらそのインデックスをバケツに後ろから入れる

(2) をいろいろ逆にした感じです

LMS も S 型 なので上書きして詰めなおします

\$	a	a	b	b	b	c	c
7	6		5		3	4	2
							↑

$S[1:]$ は S 型 で b から始まるので

\$	a	a	b	b	b	c	c
7	6		5		1	4	2
							↑

こうする

$	**a**	**a**	**b**	**b**	**b**	**c**	**c**
7	6	0	5	3	1	4	2

進めていくとこうなる
サンプルが悪いですが、LMS の場所も変わります

こちらも、 S 型のインデックスがすべて埋まります

バケツへの挿入位置は必ず今見ているところより前に来ます

なぜ？

理由はさっきと同様なので省略します

完成したバケツを見ると次のようになります

	頭文字	S[i:]
7	$	$
6	a	a$
0	a	abcbcba$
5	b	ba$
3	b	bcba$
1	b	bcbcba$
4	c	cba$
2	c	cbcba$

なんかもう完成してますね

実は、最初の LMS の入れ方が正しければこれでソートが完了します

(1) では、とりあえず頭文字だけで比べてるのでこれは正しくソートできないかもしれませんね

(2) では、今入っているSuffixの一つ前を入れていく ということは長さが $+1$ されたものを入れていきます

ここで、今入っているSuffixが正しい場所に入っていれば新しく挿入したものも正しい場所に入ることが保証されます

そこに入るようなもっと小さいものがあったとして、そのインデックスを i、実際に挿入したもののインデックスを j とします

すると、$S[i+1:] < S[j+1]$ となるような $S[i+1:]$ が存在することになりますが、いまバケツを前から見ているのでそれについては先に処理をしているはずです

ということで、(1) が正しければ (2) も正しいことがわかります

(3) も同様で、(2) までが正しければ正しくソートされるはずです

ということで、(1) のときに LMS を正しくソートする必要があります

しかし、そこで普通にソートしてしまうととても線形には間に合いません

LMS部分文字列

LMS と LMS の間 (閉区間) の部分文字列を LMS部分文字列 と呼ぶことにします

abcbcba$ は bcba$ $ の二つの LMS部分文字列 に分解できます

これらは、(S+)(L+)(S) という形になっています (最後の番兵以外)

ここで、["bcba$", "$"] という文字列の配列を考え、これのSuffix Arrayを考えてみます

Suffix	対応する元の文字列
["bcba$", "$"]	bcba$
["$"]	$

これをソートすると

Suffix	対応する元の文字列
["$"]	$
["bcba$", "$"]	bcba$

こうなって、正しくソートされた LMS の関係がわかります

なんかこっちのほうが時間かかりそうに見えますよね
しかし、LMS部分文字列どうしの比較は $O(1)$ でできます

なぜ？

最初にやった (間違った) induced sort の結果を使うことができます

	頭文字	S[i:]
7	$	$
6	a	a$
0	a	abcbcba$
5	b	ba$
3	b	bcba$
1	b	bcbcba$
4	c	cba$
2	c	cbcba$

サンプルが悪すぎて説明できないので、mmiissiissiippii という文字列について考えます

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
m	m	i	i	s	s	i	i	s	s	i	i	p	p	i	i	$
L	L	S	S	L	L	S	S	L	L	S	S	L	L	L	L	S

LMS部分文字列 は iissi iissi iippii$ $ の四つです

["iissi", "iissi", "iippii$", "$"] という配列のSuffix Arrayを考えます

これを induced sort した結果を LMS の部分だけ見るとこうなります

	頭文字	S[i:]
16	$	$
10	i	iippii$
2	i	iissiissiippii$
6	i	iissiippii$

LMS部分文字列 だけ取り出すとこうなります

	頭文字	S[i:]	rank
16	$	$	0
10	i	iippii$	1
2	i	iissii	2
6	i	iissii	2

ここで、この rank のように番号を振れば、以降このrankをみて比較すればいいことがわかります

実は、頭文字が同じでも LMS部分文字列 の順には正しくソートができています

なぜ？

induced sort の (1) は適当でした

(2) では、LMS から初めてその前にある L 型を挿入、(3) ではその L 型から前にある S 型を挿入 としました
要は、LMS からその一つ前の LMS までの範囲がソートされます

これによって、LMS部分文字列の分については正しくソートされてるようです

rankの振り方

一つ目の LMS部分文字列 ($=$ "$") のrankは 0 とします

そこから次の LMS を見ていって、もし一つ前と同じなら同じrank、異なれば $+1$ したrankを振ります

ここでの文字列比較は愚直に行いますが、合計の長さが $\Theta(|S|)$ となるので、線形で抑えられます

こうすることで、

["iissi", "iissi", "iippii$", "$"]

をrankで置き換えて

$$[2, 2, 1, 0]$$

という数列が求まりました

もはや文字列ではなくなってしまいましたが、気にせずこの数列の Suffix Arrayを求めると LMS の正しい順序がわかります

さて、計算量はいくつになったでしょうか

$\Theta(|S|)$ の処理の中で再帰的に処理を行っているので、線形じゃなくなってしまった気分になりますが、新たに計算する配列の長さは元の長さの半分以下になります (LMS だけを考えるので、連続した S は消える -> S と L が交互になるのが最長)

よって、計算量は $n = |S|$ として

$$\Theta(n + n/2 + n/4 + \ldots) = \Theta(2n) = \Theta(n)$$

と、線形のままです！

Library Checker: Suffix Array

問題文

長さ N の文字列 S が与えられます。S の suffix array a_i を求めてください。

$1 \leq N \leq 500,000$

$\Theta(N \log^2 N) : 745ms$

https://judge.yosupo.jp/submission/47223

$\Theta(N \log N) : 293ms$

https://judge.yosupo.jp/submission/47257

$\Theta(N) : 80ms$

https://judge.yosupo.jp/submission/47335

ALDS1_14_B: 文字列検索

問題文

文字列 T の中から文字列 P と一致する部分を探してください。P と一致する部分について、文字列 T の左端の位置 i を順番にすべて報告してください。

$$1 \leq |T| \leq 10^6$$
$$1 \leq |P| \leq 10^4$$

Time Limit : **1 sec**

$\Theta(N \log^2 N) : 00.76s$

https://judge.yosupo.jp/submission/47223

$\Theta(N \log N) : 00.45s$

https://judge.yosupo.jp/submission/47257

$\Theta(N) : 00.08s$

https://judge.yosupo.jp/submission/47335

応用問題

蟻本 P.338

POJ 3581: Sequence

問題文

長さ N の数列 $A = (A_1, A_2, \ldots, A_N)$ が与えられる
これを空でない三つの連続した部分列に分割し、それぞれを反転してから結合しなおす
このとき作れる辞書順最小の列を求めなさい

制約

$N \leq 200000$
A_1 はほかの要素より大きい

N の下限とか A_i の制約が足りないゴミ

とりあえず

- $3 \leq N$
- $1 \leq A_i \leq 200000$
- 入力はすべて整数

くらいを仮定しておきます

考察

A を S_1, S_2, S_3 に分割するとします

「A_1 はほかの要素より大きい」という制約から、
S_1 は S_2, S_3 を考えず貪欲に一番小さくなるものにすればよい という
ことがわかります

とりあえずこれを探しましょう

S_i をリバースしたものを R_i とします

S_i はリバースするので、A もリバースして考えてみます

すると、A をリバースしたもののSuffixの中で一番小さいものが R_1 になることがわかります

これは、A をリバースしたもののSuffix Arrayを求めれば簡単にわかります

さて、これで S_1 が求まったのでのこりは S_2 と S_3 です

今回は追加の制約がないので貪欲に S_2 を最小にすることができません

例:

$[2, 1, 2, 1, 3]$ は 貪欲に S_2 を選ぶと
$[2, 1], [2, 1, 3]$ となってリバースすると $[1, 2, 3, 1, 2]$ となるが、

$[2, 1, 2, 1], [3]$ として $[1, 2, 1, 2, 3]$ のほうが辞書順で小さい

A から S_1 の部分を取り除いたものを B とします

すると、R_2R_3 を最小化したいわけですが、

ここで B を二つ並べた BB という列を考えてみます

これは $S_2S_3S_2S_3$ というような分割になりますが、これもリバースしてみると $R_3R_2R_3R_2$ という列になります

これの真ん中の部分が答えになるので、この列のSuffix Arrayを求めれば適切な範囲内で最小のものが答えになることがわかります

実装

```
int N;
vector<int> A(N), R(N-2);

// 後ろ二文字は残さないといけないので -2
reverse_copy(A.begin(), A.end()-2, R.begin());

SuffixArray sa1(R);
// sa[0]が、最小値のインデックスを返す
// 最初の切れ目
int p1 = N - sa1[0] - 2, M = N - p1;

vector<int> B(M * 2 -1);
// 後ろ一個は残さないといけないので -1
reverse_copy(A.begin()+p1, A.end()-1, B.begin());
reverse_copy(A.begin()+p1, A.end(), B.begin()+M-1);
```

実装

```
    SuffixArray sa2(B);
    // 二個目の切れ目
    int p2 = -1;

    for(int i = 0; i < M*2-1; i++){
        int p = sa2[i];
        if(p < M-1){ // 真ん中より前じゃないとR_3の分がなくなる
            p2 = p1 + M - p - 1;
            break;
        }
    }

    reverse(A.begin(), A.begin()+p1);
    reverse(A.begin()+p1, A.begin()+p2);
    reverse(A.begin()+p2, A.end());
}
```

Longest Common Prefix Array

これはなに

最長共通接頭辞配列

字面的には Suffix Array の逆

Suffix Array の隣り合う Suffix の先頭何文字が共通しているのか

を表す配列

構築方法

Suffix Array ほど天才をしなくても線形で構築できます

尺取り法のようなことをします

ここでは

$$lcp_i = \mathrm{LCP}(\mathrm{sa_{i-1}}, \mathrm{sa_i})$$

とします

元の文字列での先頭から考えていきます

	0	1	2	3	4	5	6	
rank	0	2	4	6	1	3	5	7
	a	b	c	d	a	b	c	e
	←		→		←		→	
lcp	0				3			

これは愚直に比較して求めます
求まった lcp は 3 です

さて、次の文字を見ます

	0	1	2	3	4	5	6	
rank	0	2	4	6	1	3	5	7
	a	b	c	d	a	b	c	e
		←	→			←	→	
lcp	0				3	2		

これは、先頭の一文字を削っただけなので簡単に求まります！

うそです

いやいや、右側のやつは全然違う場所を指すかもしれないじゃん

それはそうなのですが、一致する範囲は長くはなっても短くなることはありません

よって、後ろに伸ばす回数と前を縮める回数がどちらも $|S|$ 回となり、線形で構築できます

拡張

今作ったものを利用して、任意の二つのSuffix、$S[i:]$ と $S[j:]$ のlcpも求めることができます

$l = rank[i], r = rank[j]$ とすると、l, r の prefix が共通しているということは、$l, l+1, r$ の prefix も共通しているはずです

純粋に、辞書順で近いほうが lcp も長くなるためです

つまり、$\mathrm{LCP(l, r)} = \min_{l<i\leq l} lcp_i$ となります

よって、Segment Tree や Sparse Table などによって高速に RmQ を求めることができれば、この離れた lcp も高速に求めることができます

これを利用して、文字列検索をさらに高速化することができます

簡単に言うと、既に一致している部分は飛ばして求めることで枝刈りができます

そのどれだけ飛ばせるかというのを lcp で求めることができそうですね

いい感じに実装すると、$\Theta(|T| + \log |S|)$ になりそうです

が、

Segment Tree ならクエリに、Sparse Table なら構築に $\log$ がついてしまいます (セグ木でうまくやると付かないかも)

定数倍などの関係もあり $\Theta(|T| \log |S|)$ の方法のほうが速いことが多いようです

LCS

LCS と聞くと EDPC-F のようなものを想像するかもしれませんが、この問題は共通する部分列

そうではなく 共通した **連続する** 部分文字列です

同じ文字列の ということなら話は単純で lcp の最大値です

では二つの文字だったら？

適当な区切り文字で区切って二つを連結した文字列を作ります

その文字列の lcp のうち、それぞれが別の文字列に属するようなものの中での最大値が答えになります

よって、$\Theta(|S| + |T|)$ で求まります

第 7 回日本情報オリンピック 本選
B - 共通部分文字列

2 個の文字列が与えられたとき, 両方の文字列に含まれる文字列のうち最も長いも
のを探し, その長さを答えるプログラムを作成せよ.

(さすがに想定は $\Theta(|S||T|)$)

ということで ACしました

最長回文

Manacherのアルゴリズムを知っていますか？僕は知っています

snukeさんの記事とかわかりやすいので見てください

ということで、$\Theta(|S|)$ で求めることができました

一応 LCP を利用して解くこともできます

S の i を中心とした回文 とは、$rev(S[:i])$ と $S[i+1:]$ の lcp になりそうです

さきほどの LCS の考え方を利用して、S と $rev(S)$ を適当な文字で区切って連結したものの Suffix Array を作ってみます

すると$rev(S[:i])$ と $S[i+1:]$ のどちらも Suffix Array に登場するので、それらの lcp を求めるとよいです

真ん中がない (偶数長の) 回文でも同じように求められます

構築は一回すれば各 i に対して求められるので、 $\Theta(|S| \log |S|)$ とかでしょうか

(ごめんなさい実装はしてません)

おしまい！

参考文献

http://wk1080id.hatenablog.com/entry/2018/12/25/005926

https://mametter.hatenablog.com/entry/20180130/p1#fn-c89519d1

https://shogo82148.github.io/homepage/memo/algorithm/suffix-array/sa-is.html

https://github.com/drken1215/algorithm/blob/master/String/suffix_array.cpp

https://niuez.hatenablog.com/entry/2019/12/16/203739

https://snuke.hatenablog.com/entry/2014/12/02/235837