



JOI Summer Seminar 2020

SA-IS の実装

高速文字列解析班 define

Hello World !



- 筑駒中 3 年の define です。
- 最近 Twitter してない間に忘れ去られてないか心配な今日この頃です。
- 先程の blackyuki 君の SA-IS の理論系の話の続きで SA-IS の実装の話をしてします！



SA-IS とは？

- 先程の blackyuki 君の解説の通り
- Suffix Array を $O(N)$ で構築するアルゴリズム
- 文字を Type-S , Type-L に分けて Induced Sorting
- 実装が割と重い（全文検索抜いても 100 行くらい）



実装を、しよう！

- 絶対に \log を付けないという信念を持つ
- **\log は定数ではありません**（宗教戦争）
- 理論の話ではなく、実装の話がメインです。



用語の確認

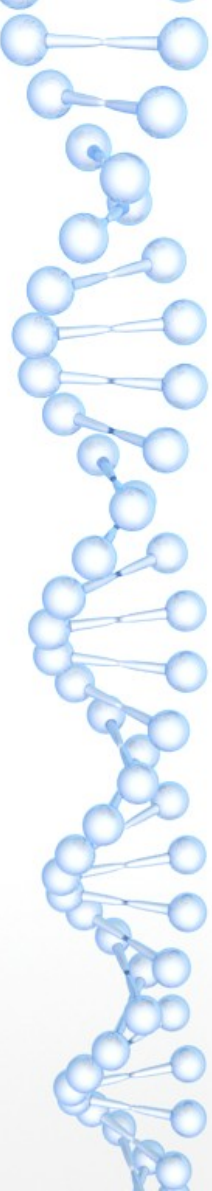
- ▶ Type-S ... $SA[i] < SA[i+1]$ となる接尾辞及び i 文字列の末端 ('\$') も Type-S とする
- ▶ Type-L ... $SA[i] > SA[i+1]$ (以下同文)
- ▶ LMS ... $SA[i]=\text{Type-S}$ かつ $SA[i-1]=\text{Type-L}$ (以下同文)
- ▶ LMS-substring
... ある LMS ～次の LMS までの文字列



実装の流れ 1

- ▶ Step 1
各 index に Type-S , Type-L , LMS を割り当てる
- ▶ Step 2
LMS-substring を Type-L , Type-S のソートにより辞書順に並び替え、
番号をつける
- ▶ Step 3
LMS-substring が同じものがある場合は番号をくっつけたものの
辞書順を再帰で求める

実装の流れ 2

- 
- ▶ Step 4
LMS の順序から Type-L の順序を求める
 - ▶ Step 5
Type-L の順序から Type-S (及び LMS) の順序を求めて完成！



Step 1 . 各 index に Type を割り当てる (1)

- これは簡単で、後ろから文字の比較をすれば OK。

	0	1	2	3	4	5	6	7	8	9	10	11
S	a	b	r	a	c	a	d	a	b	r	a	\$
type	S	S	L	LMS	L	LMS	L	LMS	S	L	L	LMS

- この場合、LMS-substring は {"aca","ada","abra\$","\$"} となります。



Step 1 . 各 index に Type を割り当てる (2)

- 実装は、後ろからやっていくだけなので簡単です。
- 私は `pair<bool,bool>` で管理しました。

```
#define typeS make_pair(false,false)
#define LMS make_pair(false,true)
#define typeL make_pair(true,true)
using TYPE=pair<bool,bool>;
vector<TYPE>assignType(vector<int>&S){
    vector<TYPE>type(len(S));
    type[len(S)-1]=LMS;
    for(int i=len(S)-2;i>=0;i--){
        if(S[i]<S[i+1])type[i]=typeS;
        else if(S[i]>S[i+1]){
            type[i]=typeL;
            if(type[i+1]==typeS)type[i+1]=LMS;
        }else type[i]=type[i+1];
    }
    return type;
}
```

Step 2. LMS-substring を並び替える (1)

- LMS を対応するバケットに位置の逆順で入れます。
- この状態で Type-L , Type-S の順にソートすると、LMS-substring が辞書順になります。

No.	pos	suffix	type
0	11	\$	LMS
1			L
2			S
3	7	abra\$	LMS
4	5	adabra\$	LMS
5	3	acadabra\$	LMS
6			S
7			S
8			L
9			L
10			L
11			L

Step 2. LMS-substring を並び替える (2)

No.	pos	suffix	type
0	11	\$	LMS
1		a???	
2		a???	
3	7	abra\$	LMS
4	5	adabra\$	LMS
5	3	acadabra\$	LMS
6		b???	
7		b???	
8		c???	
9		d???	
10		r???	
11		r???	

No.	pos	suffix	type
0	11	\$	LMS
1	10	a\$	L
2		a???	
3	7	abra\$	LMS
4	5	adabra\$	LMS
5	3	acadabra\$	LMS
6		b???	
7		b???	
8	4	cadabra\$	L
9	6	dabra\$	L
10	9	ra\$	L
11	2	racadabra\$	L

No.	pos	suffix	type
0	11	\$	LMS
1	10	a\$	L
2	7	abra\$	S
3	0	abracadabra\$	LMS
4	3	acadabra\$	LMS
5	5	adabra\$	LMS
6	8	bra\$	S
7	1	bracadabra\$	S
8	4	cadabra\$	L
9	6	dabra\$	L
10	9	ra\$	L
11	2	racadabra\$	L

	0	1	2	3	4	5	6	7	8	9	10	11
S	a	b	r	a	c	a	d	a	b	r	a	\$
type	S	S	L	LMS	L	LMS	L	LMS	S	L	L	LMS

Step 2. LMS-substring を並び替える (3)

- まずは、先程の表のようにバケットを分け、LMS を各バケットに下から入れていきます。

ついでに、各 LMS において次の LMS の位置と index 順の LMS 集合も持っておきます。

```
vector<int>getBucket(vector<int>&S,int alph){  
    vector<int>bucket(alph);  
    for(int i:S)bucket[i]++;  
    rep(i,len(bucket)-1)bucket[i+1]+=bucket[i];  
    return bucket;  
}
```

```
vector<int>InducedSorting(vector<int>&S,int alph){  
    vector<int>SA(len(S),-1);  
    vector<TYPE>type=assignType(S);  
    vector<int>bucket=getBucket(S,alph);  
    vector<int>nextlms(len(S),-1),ordered_lms;  
    int lastlms=-1;  
    rep(i,len(S))if(type[i]==LMS){  
        SA[--bucket[S[i]]]=i;  
        if(lastlms!=-1)nextlms[lastlms]=i;  
        lastlms=i;  
        ordered_lms.emplace_back(i);  
    }  
    nextlms[lastlms]=lastlms;
```

Step 2. LMS-substring を並び替える (4)

- LMS(既に決まった Type-L も) を上から見ていって、1 つ前が Type-L なら対応するバケットの一番上に入れます。

```
void sortTypeL(vector<int>&S,vector<int>&SA,vector<TYPE>&type,int alph){  
    vector<int>bucket=getBucket(S,alph);  
    for(int i:SA){  
        if(i>0&&type[i-1]==typeL)SA[bucket[S[i-1]-1]++]=i-1;  
    }  
}
```



Step 2. LMS-substring を並び替える (5)

- ▶ 同様に、Type-L についてもやります。
- ▶ \$ の 1 つ前の文字が Type-L なので、\$ 以外の LMS は全て書き換えられます。

```
void sortTypeS(vector<int>&S,vector<int>&SA,vector<TYPE>&type,int alph){  
    vector<int>bucket=getBucket(S,alph);  
    rev(j,len(S)){  
        int i=SA[j];  
        if(i>0&&(type[i-1]==typeS||type[i-1]==LMS))SA[--bucket[S[i-1]]]=i-1;  
    }  
}
```

Step 2. LMS-substring を並び替える (6)

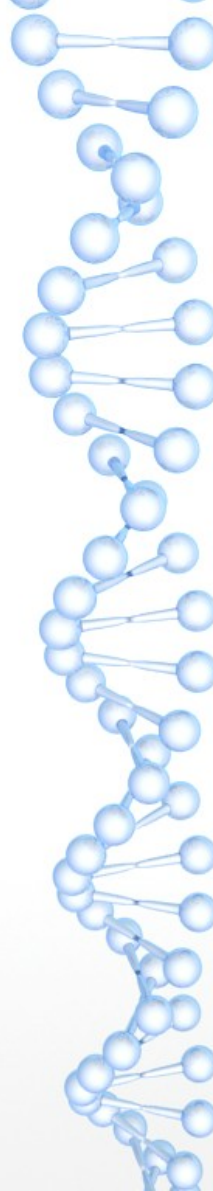
- 辞書順が分かった所で、LMS-substring を1つずつ比較して同じかどうかチェックしながら番号をつけていきます。

```
sortTypeL(S,SA,type,alph);
sortTypeS(S,SA,type,alph);
vector<int>lmses;
for(int i:SA)if(type[i]==LMS)lmses.emplace_back(i);
int nowrank=0;
vector<int>newS={0};
REP(i,len(lmses)){
    int pre=lmses[i-1],now=lmses[i];
    if(nextlms[pre]-pre!=nextlms[now]-now)newS.emplace_back(++nowrank);
    else {
        bool flag=false;
        rep(j,nextlms[pre]-pre+1){
            if(S[pre+j]!=S[now+j]){flag=true;break;}
        }
        if(flag)newS.emplace_back(++nowrank);
        else newS.emplace_back(nowrank);
    }
}
```

Step 3. LMS-substring \neq distinct の場合

- LMS-substring の順位をつなげて新しい文字列を作り、再帰的に同じアルゴリズムを適用して辞書順を求めます。
- `lmses` は LMS の辞書順、`ordered_lms` は LMS の index 順です。

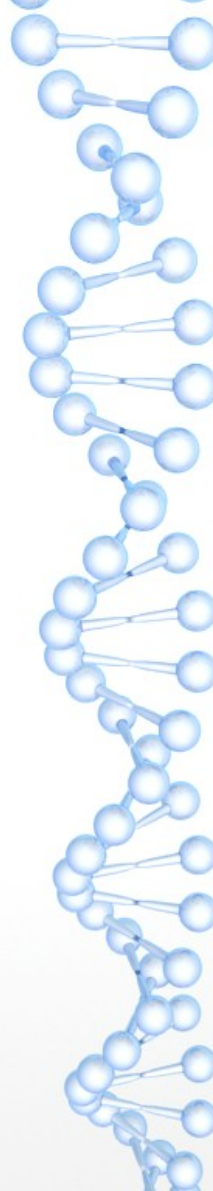
```
if(nowrank+1!=len(lmses)){  
    vector<int>V(len(S),-1);  
    rep(i,len(lmses)){  
        V[lmses[i]]=newS[i];  
    }  
    vector<int>newnewS;  
    rep(i,len(S))if(V[i]!=-1)newnewS.emplace_back(V[i]);  
    vector<int>SA_=InducedSorting(newnewS,nowrank+1);  
    rep(i,len(SA_)){  
        lmses[i]=ordered_lms[SA_[i]];  
    }  
}
```

Step 4. Type-L の順序を求める

- LMS の順序が分かった所で Step 2 と同じ事をします。
- 一旦 Suffix Array を初期化した後、LMS を辞書順で後ろから、各バケットの後ろに入れていきます。

```
SA.assign(len(S), -1);  
bucket=getBucket(S, alph);  
rev(i, len(lmses)){  
    SA[--bucket[S[lmses[i]]]]=lmses[i];  
}  
sortTypeL(S, SA, type, alph);
```



Step 5. Type-S の順序を求める

- ▶ Type-L の順序が分かった所で Type-S(LMS 含む) の順序を決定します。
- ▶ LMS を消していなくてバグるんじゃないか？
と思いますが, "\$" 以外は必ず更新されるので
大丈夫です (2 回目)

```
sortTypeS(S, SA, type, alph);  
return SA;
```

おまけ：全文検索 (1)

- Suffix Array は、文字列 T がいくつ / どこにあるかを二分探索で求める事ができます。
- 計算量は $O(|T| \log N)$ ですが、実際には判定が最後まで残る事は少なく、FM-index より速い事もあります。

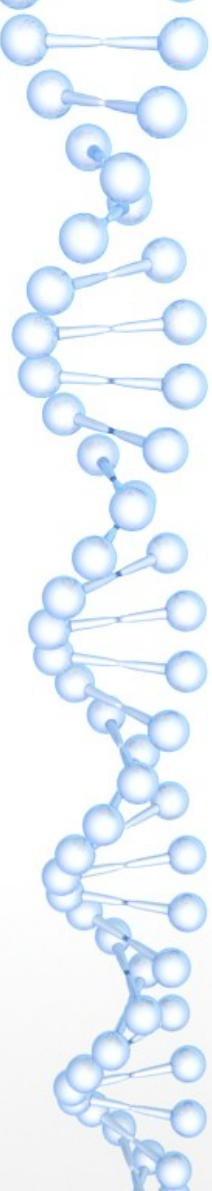
```
int ismatch(T &S,int index){
    rep(i,len(S)){
        if(i+index>=len(ST))return 1;
        if(ST[i+index]<S[i])return 1;
        if(ST[i+index]>S[i])return -1;
    }
    return 0;
}
```

おまけ：全文検索 (2)

- 実装は簡単です。
ただし、`locate` で結果をソートすると
 $O((|T|+N) \log N)$ になるので注意です。

```
P occ(T &S){
    int okl=len(ST)+1,ngl=0;
    while(okl-ngl>1){
        int mid=(okl+ngl)/2;
        if(ismatch(S,SA[mid])<=0)okl=mid;
        else ngl=mid;
    }
    int okr=len(ST)+1,ngr=0;
    while(okr-ngr>1){
        int mid=(okr+ngr)/2;
        if(ismatch(S,SA[mid])<0)okr=mid;
        else ngr=mid;
    }
    return P(okl,okr);
}

vector<int> locate(T &S){
    vector<bool> v(len(ST)+1);
    P range=occ(S);
    for(int i=range.first;i<range.second;i++)v[SA[i]]=true;
    vector<int> res;
    rep(i,len(ST)+1)if(v[i])res.emplace_back(i);
    return res;
}
```



スピード比較

	Suffix Array	FM-index	Rolling Hash
N は 10^6 以下 長さ 10000 以下の全文位置検索 1 件	80ms	130ms	60ms
N は 10^6 以下 長さ 1000 以下の存在判定 10000 件以下	550ms	500ms	不可能

FM-index は Wavelet Matrix の Bit Vector の popcount で QCFium 法をすると 2 倍くらい速くなりました。



おしまい

- ソースコードはこちら
<https://github.com/defineProgram/JOISS-2020>
- ご清聴ありがとうございました。