参考論文：Nong, Zhang, and Chan.
Two Efficient Algorithms for Linear Suffix Array Construction.　(2009)
Appendix のプログラムの解説

プログラムを Microsoft Visual C++ の環境で動かす際の注意点

先頭で使いそうなライブラリを指定

```
#include <iostream>          cout, cin, cerr や cout << などの演算子
#include <fstream>           ファイルの生成、open, close, getline, eof など
#include <string>            文字列の代入、比較、部分文字列の切り出しなど
#include <cmath>             sin, cos, ceil, log, sqrt など
#include <ctime>             clock_t, clock() 時間計測
using namespace std;         名前空間を std にする。Std::cout を cout と略記できる
```

ファイルからのデータ読み込み

有田先生の演習のプログラムを拡張。
まずファイル中の塩基の総数を計算した後で、すべての塩基を格納できる配列を生成し、再度、ファイルを読み込みながら、塩基を配列に格納してゆく。
（参考：次のスライドのプログラム）

プログラムの呼び出し方

```
int slen;
int* s = read_fasta_and_create_int_array("/tmp/chr21.fa", &slen); // 配列の長さを入れる slen の場所 &slen を渡す
```

プログラム本体

```
int* read_fasta_and_create_int_array(char* filename, int *slen){
      int bufsize = 100;
      char *line = new char[bufsize];                        // 100文字までを一度に読むバッファを用意
      ifstream fin;

      // Determine the number of nucleotides denoted by "size".
      fin.open(filename);
      if (!fin){ cout << "File not found" << endl; exit(0); }          // fin==0 ならばファイルは存在しない
      int size=0;                    // 塩基の総数を size に格納
      fin.getline(line, bufsize);  // Skip the first line. fasta 形式では最初の行が "> chr21" のようにメモ書き
      while (!fin.getline(line, bufsize).eof()){ // 1行読み込む. UCSC の染色体の塩基は1行50文字. 100文字バッファへ.
            for(int j=0; line[j] != 0; j++) {     // eof はファイルの終了
                  size++;           // バッファに格納された文字を改行 0 に達するまでチェックし、各文字ごとに size を1つ増やす
                  if (j >= bufsize){ cout << "Error: bufsize overflow" << endl;  exit(0);}
                                    // 万が一、バッファを超える行を読み込んだらエラーメッセージを出して終了
            }
      }
      fin.close();      // ファイルを一旦閉じる

      // Generate an int array of "size" and feed characters.
      fin.open(filename);           // ふたたびファイルを開く
      if (!fin){ cout << "File not found" << endl; exit(0); }
      fin.getline(line, bufsize);  // skip the first line.
      size++;                       // 配列の最後に 0 を入れるため、size を1つ増やす
      int *s = new int[size];       // 入力ファイルの全塩基を格納する配列を生成
      int i = 0;
      while (!fin.getline(line, bufsize).eof()){
            for(int j=0; line[j] != 0; j++) {
                  s[i++] = encode_char(line[j]); //  A,C,G,T を 1,2,3,4 へ書き換える
                  if (j >= bufsize){ cout << "Error: bufsize overflow" << endl; exit(0); }
            }
      }
      fin.close();
      s[size-1] = 0;    // 最後の元に 0 を追加
      *slen = size;     // 入力ファイルの塩基数を slen に戻す
      return s;         // 配列を返す
  }
```

読み込んだ文字列は、たとえば以下のように整数にコー化ドして suffix array を構築

```
int encode_char(char c){
        switch (c){
        case '$':               return 0;
        case 'A': case 'a':     return 1;
        case 'C': case 'c':     return 2;
        case 'G': case 'g':     return 3;
        case 'T': case 't':     return 4;
        default:                return 1;
        }
}
```

註：ゲノム中の N および大文字と小文字の扱い方
ゲノム配列中には大文字の A, C, G, T (コード領域)および小文字の a, c, g, t（コード領域以外）が現れていますが、今回の演習では区別しないので、すべて大文字に変換してください。その他、ゲノム配列には、A, C, G, T 以外に N という文字列が現れます。N は塩基を確定できなかったことを意味しています。たとえばテロメアやセントロメアのように繰返し配列が連続している領域では、解読することが諦められ N が続いています。

N の扱い方ですが、A, C, G, T に 1, 2, 3, 4 の番号を割り当てて、N には 5 を割り当てる方法が自然です。また、多少変則的ですが、N に A, C, G, T を割り当てることで、4文字を 2 bit 表現して記憶領域を圧縮する方式もあります。主記憶や二次記憶装置が高価で潤沢に使えなかった時代に愛用されました。しかしN に A,C,G,Tを割り当てると、その位置に問合せ配列が誤って写像されることが起こります。しかしこのような悪影響はある程度避けられることもわかっています。ゲノム中には poly-A (A が連続して現れる配列)が多く存在します。この性質を利用して、N に A を割り当てます。するともしpoly-Aを含む問合せ配列が問い合わされても、元からある多数の poly-A に写像されてしまい、位置を確定することはできません。したがって N に A を割り当てても、問合せ配列の位置を確定できない状況には変化が無いと言えるからです。もうひとつ愛用されているのは N に A,C,G,T をランダムに割り当てる考え方です。問合せ配列がある程度の長さ（長さ 20 以上）の時には、ランダムに生成された場所に誤って当たる確率は非常に低くなるため、間違いは起こりにくいからです。4文字に圧縮したい人は、自分の好きな方式で N を扱ってください。

# Classify the type of each character

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S$ | T | A | T | A | A | T | A | A | T | A | T | A | A | T | A | $ |
| $t$ | L | S | L | S | S | L | S | S | L | S | L | S | S | L | L | S |

S = 1, L = 0 ② 0 1 ①

0 1 0 1 1 0 1 1 0 1 0 1 1 0 0 1

1-byte (8 bits) で表現

```
void SA_IS(unsigned char *s, int *SA, int n, int K, int cs) {
   //           int
   // 元のプログラムでは、入力文字列が char か int の配列かを、cs の値で分けている
   // 簡単のため、 s を int 配列に固定し、 cs を使わなくてよい
   int i, j;
   unsigned char *t = (unsigned char *)malloc(n/8+1); // LS-type array in bits
                  //   new unsigned char[n/8 + 1]; でもよい
   // Classify the type of each character
① tset(n-2, 0); tset(n-1, 1); // the sentinel must be in s1, important!!!
② for(i=n-3; i>=0; i--)
     tset(i, (chr(i)<chr(i+1) || (chr(i)==chr(i+1) && tget(i+1)==1))?1:0);
        // chr(i) を s[i] へ
```

If $S[i] < S[i+1]$, or $S[i] = S[i+1]$ and $S[i+1, n-1] < S[i+2, n-1]$, $S[i]$ is S-type, else L-type.

## サブルーチンの解説

```
unsigned char mask[]={0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};
```

16進数にするには 0x をつける。 mask[] の元は二進数では以下のようになる。

```
0x80 10000000
0x40 01000000
 :
0x02 00000010
0x01 00000001
```

```
#define tset(i, b) t[(i)/8] = (b)?(mask[(i)%8]|t[(i)/8]):((~mask[(i)%8])&t[(i)/8])
```

プログラムを読みやすくするためマクロではなく関数にしてもよい

```
void tset(unsigned char* t, int i, bool b){
    if(b) t[i/8] = (mask[i%8]|t[i/8]); else t[i/8] = (~mask[i%8]&t[i/8]);
}
```

t の初期状態の例　t[0]= 00000000, t[1] = 00000001

```
i=14,b=0
i/8=1    t[i/8] = t[1]       = 00000001
i%8=6    ~mask[i%8] = ~0x02 = 11111101      ~ は 0 と 1 を反転
((~mask[i%8])&t[i/8])        = 00000001     ビットごとの AND 演算 （0を追加）

i=12,b=1
i/8=1    t[i/8] = t[1]       = 00000001
i%8=4    mask[i%8] = 0x08    = 00001000
((mask[(i)%8])|t[(i)/8])     = 00001001     ビットごとの OR 演算 （1を追加）
```

サブルーチンの解説

```
#define tget(i) ( (t[(i)/8]&mask[(i)%8]) ? 1 : 0 )
```

プログラムを読みやすくするためマクロではなく関数にしてもよい

```
   bool tget(unsigned char* t, int i){
        if((t[i/8]&mask[i%8]) != 0x00) return true; else return false;}
```

t を設定後の例　t[0]=0101 1011, t[1]= 01011001

i=11の時

```
i/8=1    t[i/8] = t[1]     = 01011001
i%8=3    mask[i%8] = 0x10 = 00010000
(t[(i)/8] & mask[i%8])     = 00010000
```

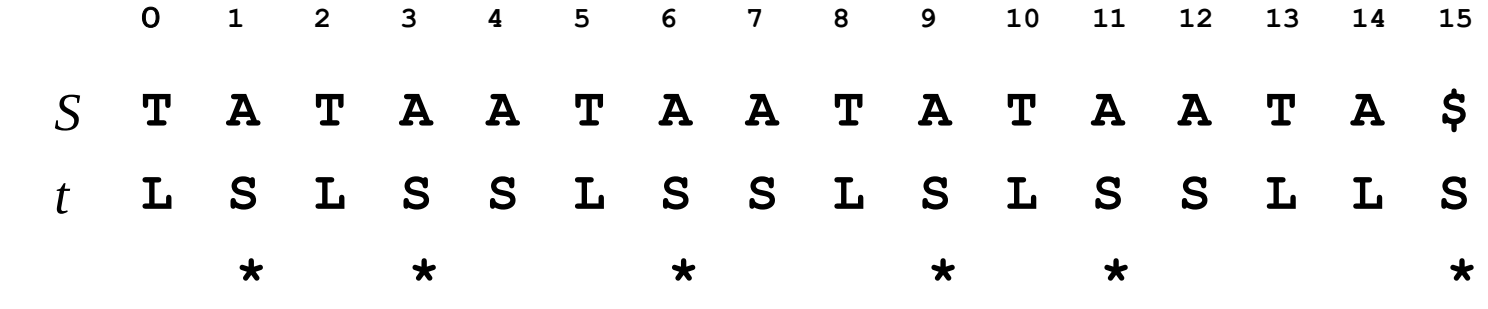
ビットごとの AND 演算 （3番目を取出す）

```
#define chr(i) (cs==sizeof(int)?((int*)s)[i]:((unsigned char *)s)[i])
```

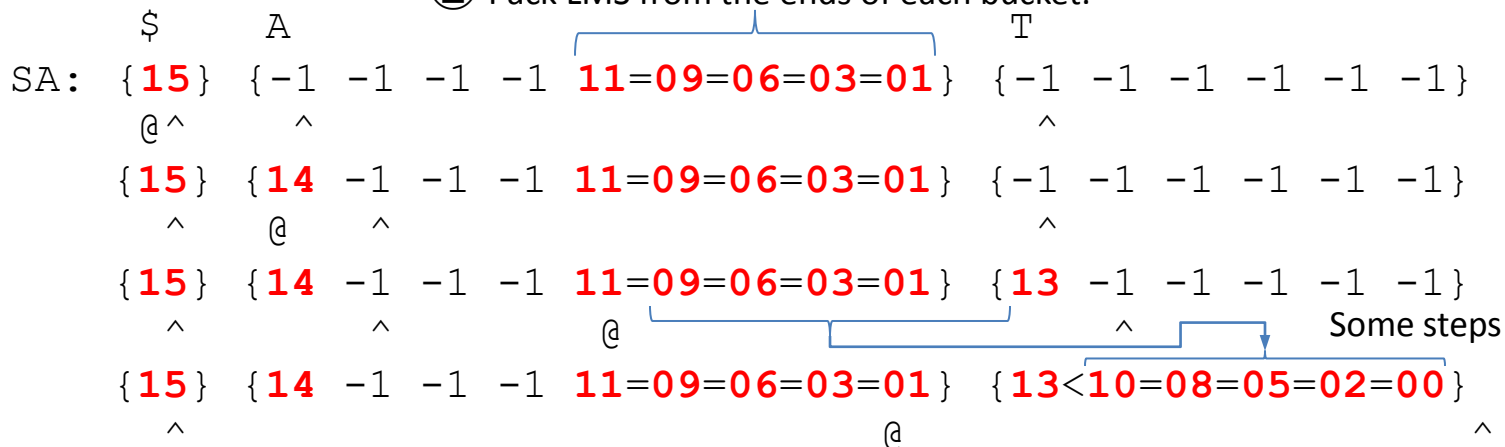
s を整数型にするか、文字型にするかという丁寧な判断をしている

メモリ消費は増えますが、簡単のため、プログラム中の chr(i) を s[i] に書き換えて構いません

# Sort all the S-substrings




② Pack LMS from the ends of each bucket.




③ Some steps are omitted.

① bkt (size)

| $ | A | C | G | T |
|---|---|---|---|---|
| 1 | 9 | 0 | 0 | 6 |

② bkt (end position)

| $ | A | C | G | T |
|---|---|---|---|---|
| 1 | 10 | 10 | 10 | 16 |

**SA_IS の内部**

```
int *bkt = (int *)malloc(sizeof(int)*(K+1));
  // K は文字の数．最初，$,A,C,G,T の場合，たとえば 0,1,2,3,4 を割り当て，K=5
① getBuckets(s, bkt, n, K, cs, true); // true は ends of buckets を計算
for(i=0; i<n; i++) SA[i]=-1;
② for(i=1; i<n; i++)  if(isLMS(i)) SA[--bkt[chr(i)]]=i; // chr(i) を s[i]へ

③ induceSA1(t, SA, s, bkt, n, K, cs, false); // false は starts of buckets
```

# サブルーチンの解説

① 
```
// find the start or end of each bucket
void getBuckets(unsigned char *s, int *bkt, int n, int K, int cs, bool end)
{                        int
   int i, sum=0;
   for(i=0; i<=K; i++) bkt[i]=0;              // clear all buckets
   for(i=0; i<n;  i++) bkt[chr(i)]++;         // chr(i) を s[i] へ
   for(i=0; i<=K; i++) { sum+=bkt[i]; bkt[i]=end ? sum : sum-bkt[i]; }
}
```

ends of buckets　starts of buckets

② 
```
#define isLMS(i) (i>0 && tget(i) && !tget(i-1))
```

S-type　L-type

③ 
```
void induceSA1(unsigned char *t, int *SA, unsigned char *s, int *bkt,
         int n, int K, int cs, bool end) {      int
   int i, j;
   getBuckets(s, bkt, n, K, cs, end); // end = false で呼ばれると start を計算
   for(i=0; i<n; i++) {  // @ を動かす
      j=SA[i]-1;  // 前の元を埋めてゆく
      if(j>=0 && !tget(j)) SA[bkt[chr(j)]++]=j; } // s[j]
}      // j が L-type ならば SA に値をいれ、^ をひとつ先へ動かす
```

| bkt (start position) | | | | |
|---|---|---|---|---|
| $ | A | C | G | T |
| 0 | 1 | 10 | 10 | 10 |

```
       $     A                                   T
③ SA: {15} {-1 -1 -1 -1 11=09=06=03=01} {-1 -1 -1 -1 -1 -1}
       @^     ^                           ^
      {15} {14 -1 -1 -1 11=09=06=03=01} {-1 -1 -1 -1 -1 -1}
        ^    @    ^                      ^
      {15} {14 -1 -1 -1 11=09=06=03=01} {13 -1 -1 -1 -1 -1}
        ^         ^          @                ^
      {15} {14 -1 -1 -1 11=09=06=03=01} {13<10=08=05=02=00}
        ^                                 @                 ^
```

Some steps are omitted.

# Sort all the S-substrings

```
        0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
   S    T  A  T  A  A  T  A  A  T  A  T  A  A  T  A  $
   t    L  S  L  S  S  L  S  S  L  S  L  S  S  L  L  S
           *     *        *        *     *           *
```

* = LMS

```
        $     A                                 T
SA: {15} {14 -1 -1 -1 11=09=06=03=01} {13<10=08=05=02=00}
       ^                                    @     ^
```

Some steps are omitted.

```
    {15} {14 -1 -1 -1 12<09=07=04=01} {13<10=08=05=02=00}
      ^              ^          @                   ^
    {15} {14 -1 -1 03<12<09=07=04=01} {13<10=08=05=02=00}
      ^           ^          @                      ^
    {15} {14 11<06=03<12<09=07=04=01} {13<10=08=05=02=00}
      ^     ^          @                            ^
```

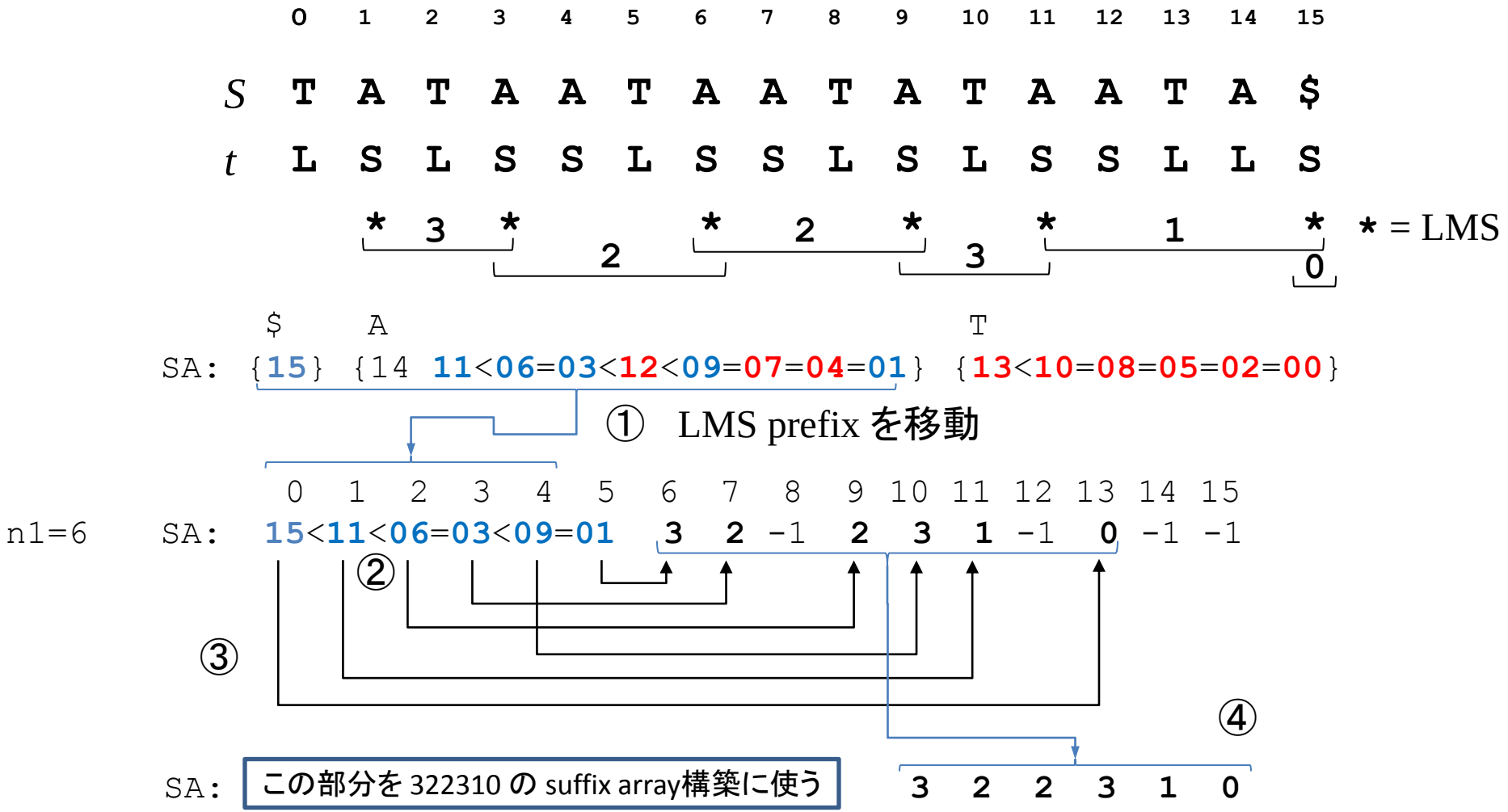
| Bkt (ending position) | | | | |
|---|---|---|---|---|
| $ | A | C | G | T |
| 1 | 10 | 10 | 10 | 16 |

**SA_IS の内部**

④

```
induceSAs(t, SA, s, bkt, n, K, cs, true);
        // process from the end
void induceSAs(unsigned char *t, int *SA, unsigned char *s, int *bkt,
        int n, int K, int cs, bool end) {
   int i, j;
   getBuckets(s, bkt, n, K, cs, end); // find ends of buckets
   for(i=n-1; i>=0; i--) {   // @ を動かす
      j=SA[i]-1;   // 前の元を埋めてゆく
      if(j>=0 && tget(j)) SA[--bkt[chr(j)]]=j; }
}        //  j が S-type ならは SA を更新し、^ をひとつ前へ動かす
```

# Find the lexicographic names of all substrings




① LMS prefix を順序を保存しながら SA の前半に移動する.
② ひとつ前の LMS prefix と等しいか大きいかをチェック .
③ SA後半に詰める. LMS は少なくとも1つ置きに出現することを利用しコンパクトに.
④ -1 を除きながら、さらに SA の後ろ側に詰めてゆく.

# SA_IS の内部

①

```
int n1=0;                    // LMS prefix の総数は n1 で、n の半分以下
for(i=0; i<n; i++)           // LMS prefix を順序を保存しながら SA の前半に移動する
   if(isLMS(SA[i])) SA[n1++]=SA[i];
```

②

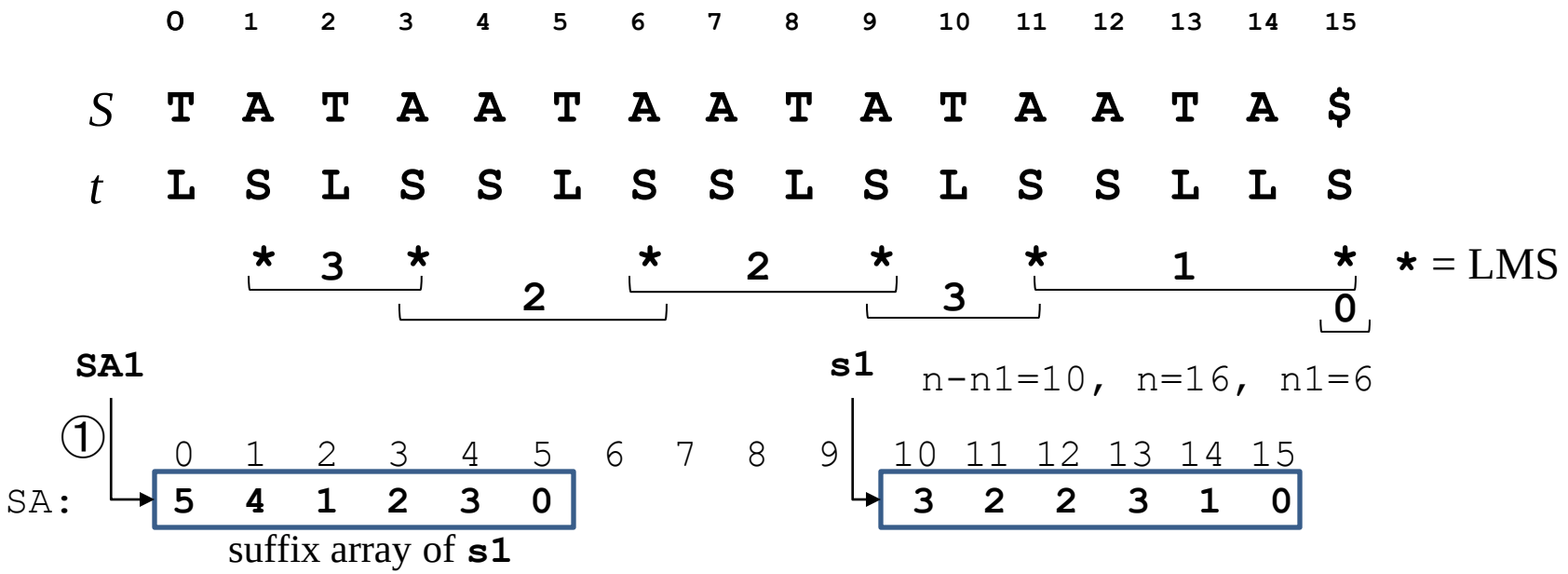
```
for(i=n1; i<n; i++) SA[i]=-1;       // n1 番目以降の要素を -1 で初期化
int name=0, prev=-1;                // name は LMS prefix に付与する順番

for(i=0; i<n1; i++) {               // 元の配列 S 中の LMS prefix に順序を付与
   int pos=SA[i];                   // S 中での prefix の位置
   bool diff=false;                 // ひとつ前の LMS prefix より大きければ true
   for(int d=0; d<n; d++)           // ひとつ前の LMS prefix と一致するかどうかチェック
      if(prev==-1 || chr(pos+d)!=chr(prev+d) || tget(pos+d)!=tget(prev+d))
         { diff=true; break; }      // 一致しなければ diff を true にして抜ける

      else if(d>0 && isLMS(pos+d)) break;
         // 次の LMS prefix まで到達し、一致することがわかる
         // ここに分岐してくるときにはisLMS(pos+d)とisLMS(prev+d)は同値なので一方をチェック
```

③

```
   if(diff) { name++; prev=pos; }  // 一致しなければ、新しい順番を name に設定
   pos=(pos%2==0)?pos/2:(pos-1)/2; // LMS は少なくとも1つ置きに出現．SA後半に詰める．
   SA[n1+pos]=name-1;              // 更新した name の1つ前の順番を付与
}
```

④

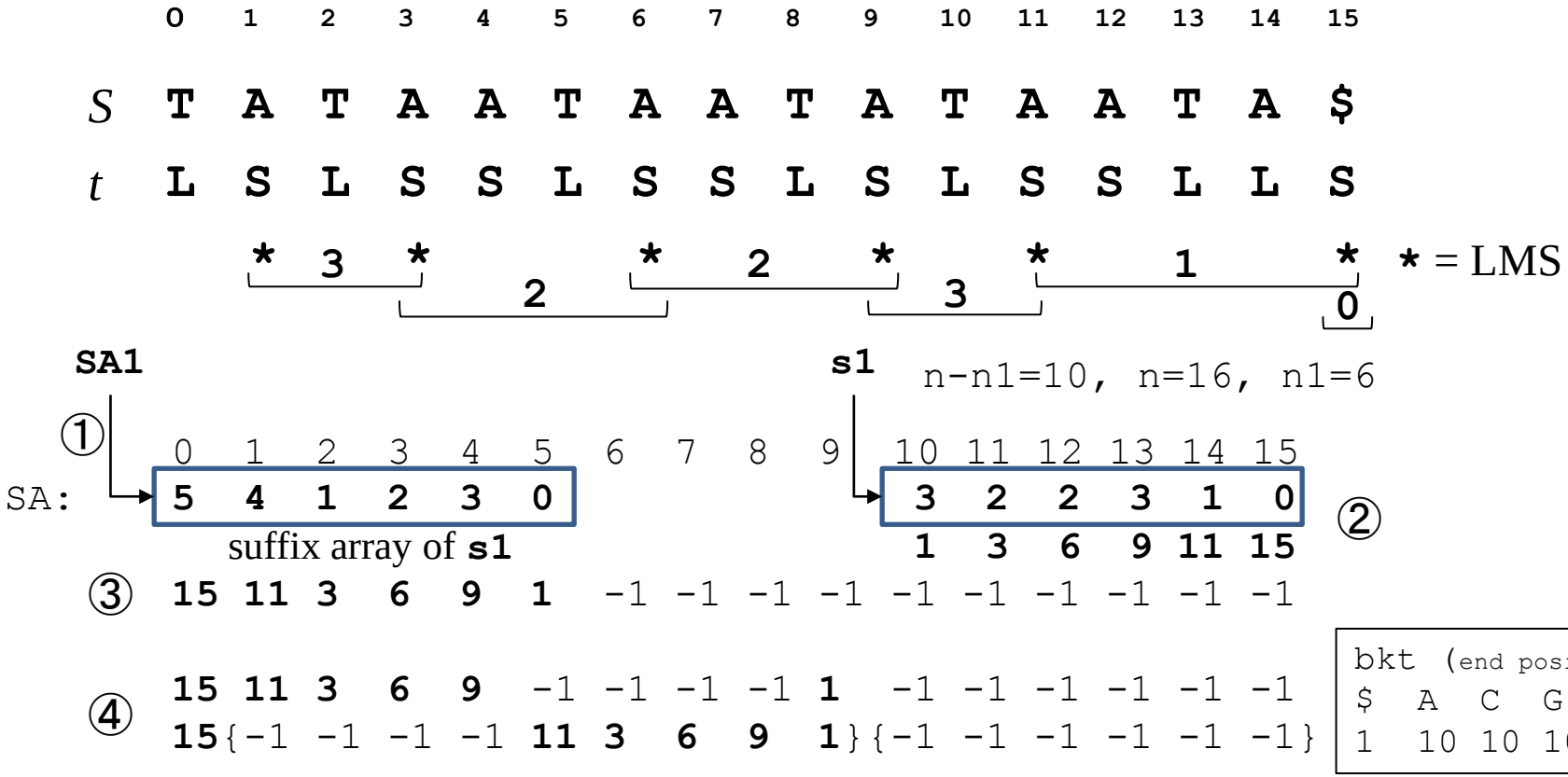
```
for(i=n-1, j=n-1; i>=n1; i--)       // -1 を除きながら、さらに SA の後ろ側に詰めてゆく
   if(SA[i]>=0) SA[j--]=SA[i];
```




## SA_IS の内部

①

```
// stage 2: solve the reduced problem
// recurse if names are not yet unique
int *SA1=SA, *s1=SA+n-n1;
if(name<n1)         // 同一の順序の LMS prefix が存在するので
                    // suffix array を構築
   SA_IS((unsigned char*)s1, SA1, n1, name-1, sizeof(int));

else                // LMS prefix には、異なる順番が付与されている
                    // 順番をそのまま suffix array に使える
   for(i=0; i<n1; i++) SA1[s1[i]] = i;
```




**SA_IS の内部**

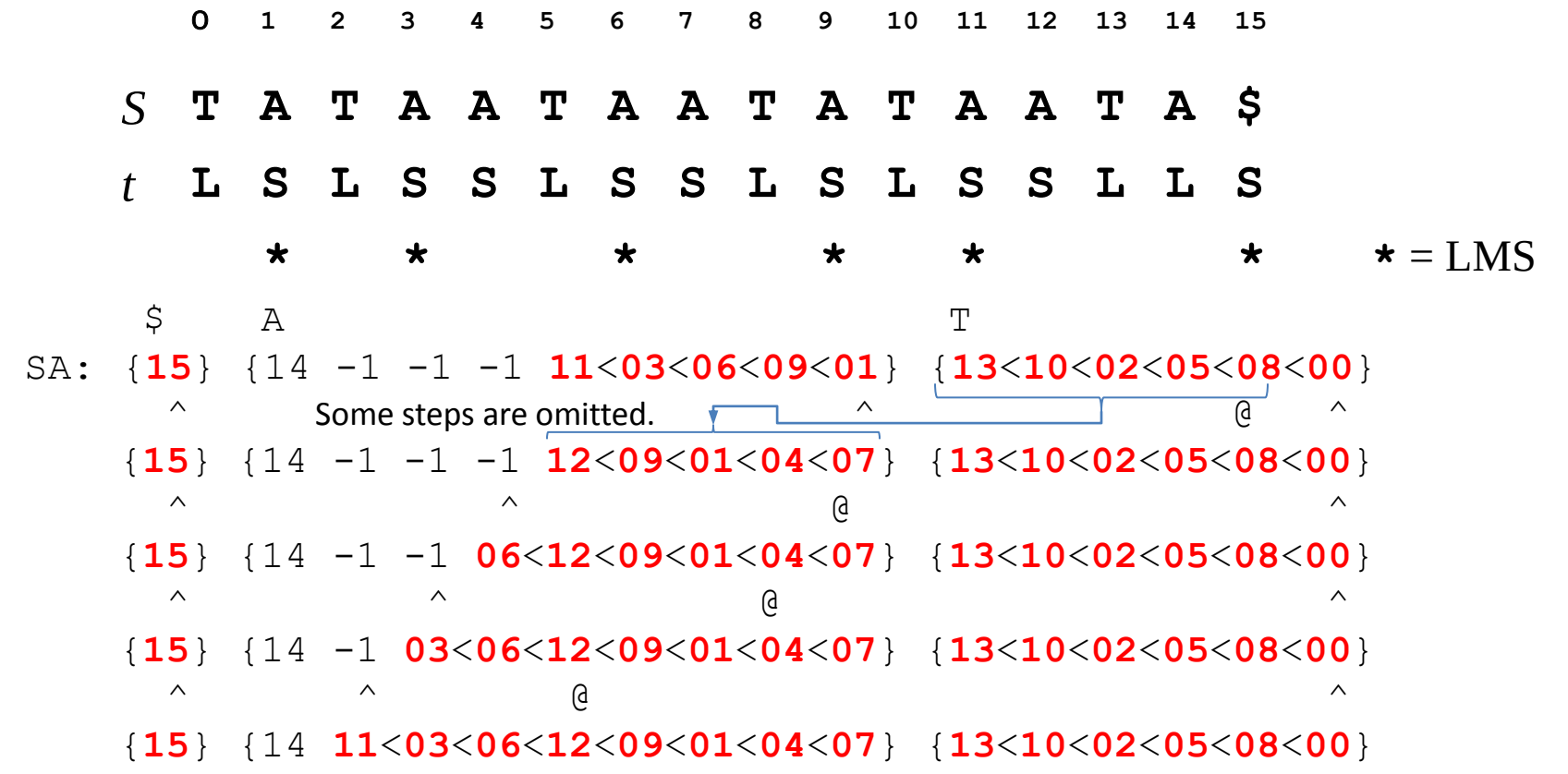
```
// stage 3: induce the result for the original problem
bkt = (int *)malloc(sizeof(int)*(K+1));     // バケットは④で使用
// put all left-most S characters into their buckets
getBuckets(s, bkt, n, K, cs, true);         // バケットは④で使用
② for(i=1, j=0; i<n; i++)                   // 各 LMS prefix の元の位置を確認
     if(isLMS(i)) s1[j++]=i;                // s1 の suffix array は構築済
                                            // s1 はもう使わないので上書き
③ for(i=0; i<n1; i++) SA1[i]=s1[SA1[i]]; // LMS prefix の順序を求める
④ for(i=n1; i<n; i++) SA[i]=-1;             // SA 後半部分を-1で初期化
   for(i=n1-1; i>=0; i--) {                 // induced sorting の準備のため、
     j=SA[i]; SA[i]=-1;                     // 順番がついた LMS prefix を配置
     SA[--bkt[chr(j)]]=j;
   }
```

```
          0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      S   T  A  T  A  A  T  A  A  T  A  T  A  A  T  A  $
      t   L  S  L  S  S  L  S  S  L  S  L  S  S  L  L  S
             *     *        *        *     *           *     * = LMS
       $     A                                 T
SA: {15} {-1 -1 -1 -1<11<03<06<09<01} {-1 -1 -1 -1 -1 -1}
     @^     ^                           ^
    {15} {14 -1 -1 -1<11<03<06<09<01} {-1 -1 -1 -1 -1 -1}
      ^    @   ^                      ^
    {15} {14 -1 -1 -1<11<03<06<09<01} {13 -1 -1 -1 -1 -1}
      ^        ^         @     途中経過省略      ^
    {15} {14 -1 -1 -1<11<03<06<09<01} {13<10<02<05<08<00}
      ^                          @                    ^
```

## SA_IS の内部とサブルーチン

```
bkt_start
$  A  C  G  T
0  1  10 10 10
```

```
induceSA1(t, SA, s, bkt, n, K, cs, false);

サブルーチン
void induceSA1(unsigned char *t, int *SA, unsigned char *s, int *bkt,
                  int n, int K, int cs, bool end) {
   int i, j;
   getBuckets(s, bkt, n, K, cs, end); // find starts of buckets
   for(i=0; i<n; i++) {  // @ を動かす
      j=SA[i]-1;
      if(j>=0 && !tget(j)) SA[bkt[chr(j)]++]=j; }
}     // j が L-type ならば SA に値をいれ、^ をひとつ先へ動かす
```




## SA_IS の内部とサブルーチン

| bkt_end | | | | |
|---|---|---|---|---|
| $ | A | C | G | T |
| 1 | 10 | 10 | 10 | 16 |

```
induceSAs(t, SA, s, bkt, n, K, cs, true);

void induceSAs(unsigned char *t, int *SA, unsigned char *s, int *bkt,
         int n, int K, int cs, bool end) {
   int i, j;
   getBuckets(s, bkt, n, K, cs, end); // find ends of buckets
   for(i=n-1; i>=0; i--) {   // @ を動かす
      j=SA[i]-1;
      if(j>=0 && tget(j)) SA[--bkt[chr(j)]]=j; }
}        //  j が S-type ならは SA を更新し、^ をひとつ前へ動かす
```

■ 計算時間の計測の例

先頭にインクルード

```
#include <ctime>

clock_t start_time, end_time, etime;
start_time = clock();
```

（計測したいプログラムの一部）

```
end_time = clock();
etime = (end_time - start_time); // CLOCKS_PER_SEC;
cout << "何かコメントを書く" << etime << " msec" << endl;
```

■ 擬似的にランダムな塩基配列を生成する例

```
void generateRandomNumbers(int* target, int targetLen){
        // Generate an array in which elements are selected
        // from {1,2,3,4} at random.
        for(int i=0; i<targetLen-1; i++)   target[i] = rand()%4 + 1;
        // Set the last element to 0 !
        target[targetLen-1] = 0;  // Set the last element to 0.
}
```

**よりランダム性の高い配列を生成したい場合には、数学科の松本眞先生が開発され、広く利用されている Mersenne Twister を使ってください。**

**http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html**