生物情報ソフトウエア論　（担当　森下）

試験日時　２０１４年６月５日　木曜日

１０：３０　～　１２：３０

途中退室して構いません。

問題 1 （DNA 解読の高速化）

2007 年以降ゲノム解読が高速化した。特に 約 100 塩基の長さの短い DNA 断片を約 1 週間で 100 億本近くも解読できる Illumina 社の DNA 解読装置は普及している。未知の DNA 配列の決定という狭い範囲の問題にとどまらず、生物学と医学の多様な問題に応用されている。下記の問題に対して、DNA 解読装置がどのように使われているか 200 〜 500 文字（正確に数える必要はない）で答えよ。

(1)　パーソナルゲノム解析　体質と病気に関連する遺伝子変異の検出

(2)　遺伝子発現量の定量化

(3)　CpG メチル化状態の観測

(4)　ヌクレオソームコア位置の推定

問題２　(sorting)

sorting は様々な計算を高速化する際に利用されており、我々の計算機の中でも常に何らかの形で動作している。生物情報処理においても、遺伝子発現量で遺伝子をランキング、多次元空間内での遺伝子をクラスタリングなど多様な応用例で要となる。sorting を実装する方法はいくつか提案されている。最も使われているのが以下にしめす randomized quick sort の問題点を改善した introspective sort である。

```
void randomQuickSort1(int* target, int left, int right ) {
     if (left < right) {
          int i = partition(target, left, right);
               // i indicates the position of the pivot.
          randomQuickSort1(target, left, i - 1);
          randomQuickSort1(target, i + 1, right);
     }
}
```

(partition の実装は以下では変更しないため、プログラムを示していない)

プログラムを設計する際には、stack 領域を無駄に消費しないように、再帰的呼び出し(recursive call) を減らす工夫をするが、この観点からすると randomized quick sort は様々な問題をかかえている。

(1) この randomized quick sort の中には、再帰的呼び出しをしなくてよい部分がある。この無駄を回避する tail recursion elimination を施したプログラムを示せ。

(2) partition による配列の分割がうまくゆかないと、stack を消費し尽してしまう現象 stack overflow が生じる。どのような分割が進行すると stack overflow が起こるか？これを避けるため、入力配列の長さが N の場合でも、プログラム実行中のいかなる時点でも、再帰的呼び出しが高々 $\log_2 N$ 以下しか生じないようにプログラムを変形せよ。

(3) 上の工夫をした時の最悪計算量は $O(N^2)$ もしくは $O(N \log N)$ のどちらか？
理由とともに答えよ。

(4) 長さ N の入力配列の各要素 x が $0 \leq x < N^2$の範囲にある特殊な状況を考える。この場合、最悪計算量が $O(N)$ となるアルゴリズムを設計できる。どのようなアルゴリズムか説明せよ。

問題３　(suffix array, doubling)

Suffix array は文字列検索を高速化するために 1990 年に Manber と Myers が提唱したデータ構造である。当初は構築に時間がかかり、主記憶を大量に消費するなどの理由で DNA 配列の検索にはあまり利用されていなかった。しかし 1999 年に Larsson と Sadakane が $O(n \log n)$ 時間のアルゴリズムを発表し、2003 年には線形時間で構築するアルゴリズムがいくつか提案された。2007 年以降、高速な DNA 解読装置が普及したため、生命科学でも利用が広がっている。

記号列 S の i 番目の記号を $S[i]$ と表記する。$|S|$ は S の長さを示す。記号列および配列は 0-origin indexing で表現する。記号列 S は **$** で終わるものとし、**$** は他の位置には出現しないとする。記号列の辞書式順序を決める際には **$** はどの記号より小さいと約束する。次の各問に答えよ。

(1)　S = **TCCTCCTCCTA$**　に対する suffix array SA を示せ。

(2)　inverse suffix array ISA を示せ

(3)　SA を構築する Larsson-Sadakane の doubling algorithm とはなにか？　その動作を S = **TCCTCCTCCTA$**　を例として使って説明せよ。その際、$h$-group, h-rank, SA_h, approximate inverse suffix array ISA_h 等の記号を用いること。

(4)　$|S| = n$ のとき上の doubling algorithm の計算量が $O(n \log n)$ であることを説明せよ。

問題４　(suffix array, induced sorting)

suffix array を構築する線形時間アルゴリズムの中で、induced sorting は比較的理解がしやすい方法である。そこで、induced sorting を利用して suffix array を線形時間で構築するアルゴリズムを設計することを考える。次の各問に答えよ。

(1) $S[i]$ からはじまる suffix が、S[i+1]からはじまる suffix より辞書式順序に関して小さい（大きい）とき、記号 $S[i]$ と $S[i]$ からはじまる suffix を S-タイプ (L-タイプ) と呼ぶ。各 suffix が S もしくは L-タイプどちらであるかを $O(|S|)$ で判定するアルゴリズムを述べ、その動作を S = **TCCTCCTCCTA$** を例に説明せよ。

(2) $S[i]$ ($i > 0$) が S-タイプで、$S[i\text{-}1]$ が L-タイプの時、$S[i]$ を LMS (Left-Most S-type) 記号と呼ぶ。また LMS 記号からはじまる suffix を LMS suffix とよぶ。すべての LMS suffix の辞書式順番が判明していると仮定して、suffix array を $O(|S|)$ で計算するアルゴリズムの動作を、S = **TCCTCCTCCTA$** を例に示せ。

(3) 長さ 2 以上の部分記号列 $S[i,k]$ の中で $S[i]$ の後ろで最初に出現する LMS 記号が $S[k]$ となるとき、$S[i,k]$ を LMS prefix とよぶ。なお単一の LMS 記号も LMS prefix と呼ぶ。LMS prefix の辞書式順序を $O(|S|)$ で計算するアルゴリズムの動作を、S = **TCCTCCTCCTA$** を例に示せ。LMS prefix の辞書式順序から LMS suffix の辞書式順序を計算するアルゴリズムを簡潔に説明せよ。

(4) $|S| = n$ とするとき、上の induced sorting algorithm の計算量が $O(n)$ であることを説明せよ。

問題５　（Burrows-Wheeler Transform）

suffix array を使って文字列検索する初期の方法は、記号列 S の長さを n, 問合せ記号列 W の長さ m に対して $O(m \log n)$ 時間、もしくは少し最適化して $O(m + \log n)$ 時間かかっていた。Ferragina と Manzini は 2005 年に発表した論文で、Burrows と Wheeler が文字列圧縮のために考案した変換法（Burrows-Wheeler Transform, 1994 年）を suffix array から $O(n)$ 時間で構築し、検索を $O(m)$ 時間で実行するためのインデックスを提案し注目される。このアイデアは近年さまざまな DNA アラインメントツール(たとえば BWA)で採用されるようになる。Burrows-Wheeler Transform を使った検索技術について以下の問いに答えよ。

(1) 記号列 S の suffix array を SA、 Burrows-Wheeler Transform を BWT と表記する。S = **TCCTCCTCCTA$** の BWT を求めよ。

(2) BWT から元の記号列 S を再構築するために、

$$S[\,(|S|-2)-k\,] = BWT[\,T^k[0]\,] \quad (k = 0, \ldots, |S|-2)$$

を満たす写像 T を構築することを考える（$T^0[x]=x,\ T^{k+1}[x]=T[T^k[x]]$ ）。T を $O(n)$ 時間で計算するために、SA を利用する方法を説明せよ。SA を利用せずに T を $O(n)$ 時間で計算する方法を説明せよ。また S = **TCCTCCTCCTA$** のとき、対応する T を求めよ。

(3) 上記の方法を拡張し、問合せ記号列 W が出現する位置を $O(m)$ 時間で全て列挙するアルゴリズムの動作を、S = **TCCTCCTCCTA$** および W = **CCT** を例にして示せ。また、そのアルゴリズムが使用する主記憶サイズを軽減する方法について述べよ。