ソフトウェア演習 IV 手順書（３週目）　　2020/01/14 梅村恭司

ペアプログラミングによる Suffix Array の実装。Suffix Array は前処理をして、出現数を高速に求めることを可能とするデータ構造である。

本日の概要、Git Pull Request, ペアプログラミングを経験する。

-- 以下は全体で行う。--
(1) 指定した時間まで、ペアプログラミング，Suffix Array のビデオ視聴せよ。

(2) 別ページに記載されている「Suffix Array の内部設計情報」についての教員の説明を受けよ。
　なお、この設計情報のファイルは、演習用のリポジトリの umemura/Frequencer.java に配置してあるので利用してもよい。ただし、利用するときにパッケージの指定を適切に変更する必要がある。

(3)Google meet の自分の会議室を作り、その URL をスプレッドシートに記入せよ。さらに、ペア相手と作成した会議室で会話できることと、および、その会議室でプログラミングを行っている画面の共有ができることを確認せよ。

ペア情報シートの URL
https://docs.google.com/spreadsheets/d/1tSjt0DEAWhgnEA2WxN22qgM9ah9OzoYEzjksajahuhs/edit?usp=sharing

(4)以下の手順で、ペア相手の共有リポジトリに自分の共有リポジトリの pull request を送れ。GitHub に login し、”new pull request”のボタンを押すと、 演習用のリポジトリが送り先の規定値となるので、”base repository:TUT3Software4/…”のところで、ペア相手のリポジトリを選択し、リクエストの送り先を変更すること。そののち、”Create pull Request”を押す。

(5)以下の手順で、相手からから送られてきた pull request を受け入れよ。GitHub に login した後にあらわれる “pull requests”をクリックすることで、相手からの pull request が表示される。そのリクエストの名前をクリックすると、内容を判断できるようになる。たとえば、”Files changed”の場所をクリックすると、差分が表示される。Review changes のボタンで、判断をする。ここでは、問題のない場所のファイルのはずであるので、approve を押す。そして、submit review を押す。画面が変わるので、そこであらわれる “Merge pull request”を押すことで、GitHub 上での merge が行われる。

(6)以下の手順で、ローカルで作業している work tree に変更を反映させよ。Work tree のディレクトリ(readme. md のある場所)で git pull を行え。

(7) 内部設計仕様にしたがったコーディングを行え。今週は SuffixArray の作成までを，入力・観察に分かれてペアプログラミングをせよ。最初に、作業をするディレクトリをどちらのものにするかをまず決めて作業せよ。時間内で確実に動作するプログラムを作ることに目標に作業を行う。したがって、ソートはバブルソート、下記の内部設計仕様は java ですでに用意されているソートは使いにくい設計（ソートを自分で書くの求められる設計）になっているので、すでに用意されているソートは使うことを試みないほうがよい。ペア間の役割の交代は 30 分をめやすに行え。コードを追加するところは、コメントで示してある。デバッグのためには、Frequencer クラスの main を実行することになる。実行するには、Makefile の「#java … Frequencer」の行のコメントを外せばよい。ソートの順番を決めるための文字列比較については、文字列を比較するというより、設定されたバイト配列の指定した範囲を対象に比較をおこなう設計になる。そして、バイト列の比較で実装し、1 バイトごとに比較を行うことを期待している。コーディングでは終わりの条件の比較が正しいことを確認することにとくに注意してほしい。

(8)ペア指定のシートに，ペアプログラミングについて，経験の有無，および，一人でおこなう作業との比較を，最後に気がついたところを記入せよ。

(9)作業結果をペア間で共有せよ。まず、お互いに pull request を送り合うことで行う。そして、作業場所が自分の番号の配下でないばあいは、自分の番号の配下にコピーして、パッケージを適切に変更せよ。

(10) これまでと違い，リファクタリングとダイナミックプログラミングのビデオは，来週の授業まえに視聴しておくことを求める。

(11)今回は、共有リポジトリに push することは求めないが、ペアプログラミングで Suffix Array の構築が正しくできている場合は、ビデオをみたあとに、別ブランチを作成し、ソートのアルゴリズムを高速なものにとりかえる作業を自分だけで試みてみよ。

(12) 来週は、Suffix Array の検索の実装を行う。

-- 以下は suffix array をつかった Frequencer の内部設計情報である。
　このファイルを参照しながらペアプログラミングすることで，ペアプログラミングの前提となっている条件：「作成するコードに対する意思統一ができていること」を確保することを意図している。

```
-----------------------------------------------
package s4.umemura;
import java.lang.*;
import s4.specification.*;


/*package s4.specification;
  ここは、１回、２回と変更のない外部仕様である。
  public interface FrequencerInterface {     // This interface provides the design for
frequency counter.
  void setTarget(byte  target[]); // set the data to search.
  void setSpace(byte  space[]);  // set the data to be searched target from.
  int frequency(); //It return -1, when TARGET is not set or TARGET's length is zero
  //Otherwise, it return 0, when SPACE is not set or SPACE's length is zero
  //Otherwise, get the frequency of TAGET in SPACE
  int subByteFrequency(int start, int end);
  // get the frequency of subByte of taget, i.e target[start], taget[start+1], ... ,
target[end-1].
  // For the incorrect value of START or END, the behavior is undefined.
  }
*/



public class Frequencer implements FrequencerInterface{
    // Code to start with: This code is not working, but good start point to work.
    byte [] myTarget;
    byte [] mySpace;
```

```
    boolean targetReady = false;
    boolean spaceReady = false;

    int []  suffixArray; // Suffix Array の実装に使うデータの型を int []とせよ。


    // The variable, "suffixArray" is the sorted array of all suffixes of mySpace.
    // Each suffix is expressed by a integer, which is the starting position in mySpace.

    // The following is the code to print the contents of suffixArray.
    // This code could be used on debugging.

    private void printSuffixArray() {
        if(spaceReady) {
            for(int i=0; i< mySpace.length; i++) {
                int s = suffixArray[i];
                for(int j=s;j<mySpace.length;j++) {
                    System.out.write(mySpace[j]);
                }
                System.out.write('\n');
            }
        }
    }

    private int suffixCompare(int i, int j) {
        // suffixCompare はソートのための比較メソッドである。
        // 次のように定義せよ。
        // comparing two suffixes by dictionary order.
        // suffix_i is a string starting with the position i in "byte [] mySpace".
        // Each i and j denote suffix_i, and suffix_j.
        // Example of dictionary order
        // "i"      <  "o"        : compare by code
        // "Hi"     <  "Ho"       ; if head is same, compare the next element
        // "Ho"     <  "Ho "      ; if the prefix is identical, longer string is big
        //
        //The    return    value    of    "int    suffixCompare"    is    as    follows.
\


        // if suffix_i > suffix_j, it returns 1
        // if suffix_i < suffix_j, it returns -1
        // if suffix_i = suffix_j, it returns 0;


        // ここにコードを記述せよ
        //
        return 0; // この行は変更しなければいけない。
    }

    public void setSpace(byte []space) {
        // suffixArray の前処理は、setSpace で定義せよ。
        mySpace = space; if(mySpace.length>0) spaceReady = true;
        // First, create unsorted suffix array.
        suffixArray = new int[space.length];
        // put all suffixes in suffixArray.
```

```
        for(int i = 0; i< space.length; i++) {
            suffixArray[i] = i; // Please note that each suffix is expressed by one
integer.
        }
        //
        // ここに、int suffixArray をソートするコードを書け。
        //   順番は suffixCompare で定義されるものとする。
    }

    // Suffix Array を用いて、文字列の頻度を求めるコード
    // ここから、指定する範囲のコードは変更してはならない。

    public void setTarget(byte [] target) {
        myTarget = target; if(myTarget.length>0) targetReady = true;
    }

    public int frequency() {
        if(targetReady == false) return -1;
        if(spaceReady == false) return 0;
        return subByteFrequency(0, myTarget.length);
    }



    public int subByteFrequency(int start, int end) {
        /* This method be work as follows, but much more efficient
           int spaceLength = mySpace.length;
           int count = 0;
           for(int offset = 0; offset< spaceLength - (end - start); offset++) {
           boolean abort = false;
           for(int i = 0; i< (end - start); i++) {
           if(myTarget[start+i] != mySpace[offset+i]) { abort = true; break; }
           }
           if(abort == false) { count++; }
           }
        */
        int first = subByteStartIndex(start, end);
        int last1 = subByteEndIndex(start, end);
        return last1 - first;
    }
    // 変更してはいけないコードはここまで。

private int targetCompare(int i, int j, int k) {
        // suffixArray を探索するときに使う比較関数。
        // 次のように定義せよ
        // suffix_i is a string in mySpace starting at i-th position.
        // target_i_k is a string in myTarget start at j-th postion ending k-th position.
        // comparing suffix_i and target_j_k.
        // if the beginning of suffix_i matches target_i_k, it return 0.
        // The behavior is different from suffixCompare on this case.
        // if suffix_i > target_i_k it return 1;
        // if suffix_i < target_i_k it return -1

        // It should be used to search the appropriate index of some suffix.
```

```
    // Example of search
    // suffix          target
    // "o"       >     "i"
    // "o"       <     "z"
    // "o"       =     "o"
    // "o"       <     "oo"
    // "Ho"      >     "Hi"
    // "Ho"      <     "Hz"
    // "Ho"      =     "Ho"
    // "Ho"      <     "Ho "   : "Ho " is not in the head of suffix "Ho"
    // "Ho"      =     "H"     : "H" is in the head of suffix "Ho"
    //
    // ここに比較のコードを書け
    //
    return 0; // この行は変更しなければならない。
}


private int subByteStartIndex(int start, int end) {
    //suffix array のなかで、目的の文字列の出現が始まる位置を求めるメソッド
    // 以下のように定義せよ。
    /* Example of suffix created from "Hi Ho Hi Ho"
       0: Hi Ho
       1: Ho
       2: Ho Hi Ho
       3:Hi Ho
       4:Hi Ho Hi Ho
       5:Ho
       6:Ho Hi Ho
       7:i Ho
       8:i Ho Hi Ho
       9:o
       A:o Hi Ho
    */

    // It returns the index of the first suffix
    // which is equal or greater than target_start_end.
    // Assuming the suffix array is created from "Hi Ho Hi Ho",
    // if target_start_end is "Ho", it will return 5.
    // Assuming the suffix array is created from "Hi Ho Hi Ho",
    // if target_start_end is "Ho ", it will return 6.
    //
    // ここにコードを記述せよ。
    //
    return suffixArray.length; //このコードは変更しなければならない。
}

private int subByteEndIndex(int start, int end) {
    //suffix array のなかで、目的の文字列の出現しなくなる場所を求めるメソッド
    // 以下のように定義せよ。
    /* Example of suffix created from "Hi Ho Hi Ho"
       0: Hi Ho
       1: Ho
       2: Ho Hi Ho
```

```
           3:Hi Ho
           4:Hi Ho Hi Ho
           5:Ho
           6:Ho Hi Ho
           7:i Ho
           8:i Ho Hi Ho
           9:o
           A:o Hi Ho
        */
        // It returns the index of the first suffix
        // which is greater than target_start_end; (and not equal to target_start_end)
        // Assuming the suffix array is created from "Hi Ho Hi Ho",
        // if target_start_end is "Ho", it will return 7 for "Hi Ho Hi Ho".
        //  Assuming the suffix array is created from "Hi Ho Hi Ho",
\

        // if target_start_end is"i", it will return 9 for "Hi Ho Hi Ho".
        //
        //  ここにコードを記述せよ
        //
        return suffixArray.length; // この行は変更しなければならない、
    }


    // Suffix Array を使ったプログラムのホワイトテストは、
    // private なメソッドとフィールドをアクセスすることが必要なので、
    // クラスに属する static main に書く方法もある。
    // static main があっても、呼びださなければよい。
    // 以下は、自由に変更して実験すること。
    // 注意：標準出力、エラー出力にメッセージを出すことは、
    // static main からの実行のときだけに許される。
    // 外部から Frequencer を使うときにメッセージを出力してはならない。
    // 教員のテスト実行のときにメッセージがでると、仕様にない動作をするとみなし、
    // 減点の対象である。
    public static void main(String[] args) {
        Frequencer frequencerObject;
        try {
            frequencerObject = new Frequencer();
            frequencerObject.setSpace("Hi Ho Hi Ho".getBytes());
            frequencerObject.printSuffixArray(); // you may use this line for DEBUG
            /* Example from "Hi Ho Hi Ho"
               0: Hi Ho
               1: Ho
               2: Ho Hi Ho
               3:Hi Ho
               4:Hi Ho Hi Ho
               5:Ho
               6:Ho Hi Ho
               7:i Ho
               8:i Ho Hi Ho
               9:o
               A:o Hi Ho
            */
```

```
            frequencerObject.setTarget("H".getBytes());
            //
            // ****  Please write code to check subByteStartIndex, and subByteEndIndex
            //

            int result = frequencerObject.frequency();
            System.out.print("Freq = "+ result+" ");
            if(4     ==     result)     {     System.out.println("OK");     }     else
{System.out.println("WRONG"); }
        }
        catch(Exception e) {
            System.out.println("STOP");
        }
    }
}
```