「千葉工業大学先進工学部未来ロボティクス学科数値解析学 II」

大レポート課題

学籍番号：20C1119
氏名：森田　大雅
作成日 : 2022 年 06 月 11 日

[1] 目的

DP マッチングのプログラムを書き、それをもとに小語彙の単語音声認識実験を行う.

※Github に載せていますのでよろしければご参照になってください.
https://github.com/hirotamorimasa/report

[2] ファイルの処理

データセットのファイルを読み込むのに、標準ライブラリ関数の fscanf 関数を使用した. 使用するには、ファイルなのデータの型が同じでないと上手く読み込めないため、(これ以外の方法が浮かばなかったので、)以下のような処理を施してからプログラムを実行した.

まずデータセット最初の三行がファイルを読み込むのに邪魔だったので、sed コマンドとリダイレクトを使い音響特徴ベクトルのみのサブファイルを全部作った. 作ったサブファイルはサブディレクトリにそれ移動させておく.

次の例は、city_011_001.txt を場合の例である.

~/report/ $ mkdir sub.city011　　//サブディレクトリを作成
~/report/city011 $ sed '1,3d' city011_001.txt > ../sub.city011/city011_001.txt　　// サブファイルを作成

これを city011 ディレクトリ 100 回分、そして city011~city022 まで 400 回分繰り返す.

メインルーチンは次の組み合わせでそれぞれ 6 つに分けた.

~/pattern1.c（city011, city021）
~/pattern2.c（city012, city022）
~/pattern3.c（city011, city012）
~/pattern4.c（city012, city021）

~/pattern5.c（city011, city022）
~/pattern6.c（city021, city022）

［3］ヘッダファイルの仕様

メインルーチンにのみプログラムを書くと、あまりにも読みにくくなるので、ヘッダファイルを作った．例えば、city011 に対して二つのヘッダファイルがある．つまり city011、city012、city021、city022 の 4× 2 のヘッダファイルを全部でつくった．それぞれ動作は全く同じで、ヘッダファイル内の関数名がそれぞれ異なっている．

例えば city011 の場合だと、sub.city011 のディレクトリにある city011.h は、主に city011_001.txt から city011_100.txt までのデータの読み込みと格納を行っている．

また report ディレクトリの下にある city011_line.h は city011 内にあるファイル全ての（フレーム数）行数の情報を格納している．なおこのヘッダファイルは処理が完全に被っているので説明は省略する．

① city011.h の説明

city011_file_line()で各ファイルのフレーム数（行数）をカウントする．これは 2 次元配列に（データを格納）するときにファイルのフレーム数が分かっていれば、for 文のカウントに使ってデータを最初から最後まできちんと格納することができる．

city011_file_read()で一回データを全て一次元化して読み込んでいる．ファイルが（city011_001.txt から city011_100.txt まで）100 個分あるので実際は 2 次元配列となっているが、最初に手順を書き出したときにファイル一つ分で考えてから、その次に 100 個分に拡張したのでこうなっている．

city011_file_processing()で 2 次元配列に変換して扱いやすくしている．授業の説明を参考に、フレーム数（数はファイルに依存する）と各フレームの要素（15 個の dimension）を data という配列変数にデータを格納している．これも実際は 3 次元配列ではあるが、理由は前と同様である．

```c
#include <stdio.h>
#include <stdlib.h>

#define FRAME 150
#define DIMENSION 15
#define STRING 30       //city0xx_001.txt~city0xx_100.txt
#define NUMBER 1500      //city011_001.txt~city011_100.txt
#define DATA_SET 100   //データセット100個分

//ファイルの行数を確認する
int city011_file_line(char fname[])
{
        int ch;
        FILE *fp;
        int line = 0;
        int count = 0;

        fp = fopen(fname, "r");
        while((ch = fgetc(fp)) != EOF)
                if(ch == 10)
                        count++;
        fclose(fp);
        return count;
}

/*
void city011_file_print(char city011[][STRING], double data[DATA_SET][FRAME][DIMENSION])
{
        int line;
        for(int i = 0; i < DATA_SET; i++){
                line = city011_file_line(city011[i]);
                for(int j = 0; j < line; j++)
                        for(int j = 0; j < line; j++)
                                for(int k = 0; k < DIMENSION; k++)
                                        printf("data[%d][%d][%d]:%lf\n", i+1, j+1, k+1, data[i][j][k]);
                printf("\n\n");
        }
}
*/

//一次元配列でデータを読み込む（実際は２次元配列）
void city011_file_read(char city011[][STRING], double data[DATA_SET][NUMBER])
{
        FILE *fp;
        int count = 0;
        int result;
        int frame = 0, dimension = 0;   //frame:61行  dimension:15要素(列)
        int line;

        for(int i = 0; i < DATA_SET; i++)
        {
                fp = fopen(city011[i], "r");
                if(fp == NULL)
                {
                        printf("%s file don't  open.\n", city011[i]);
                        return;
                }
                line = city011_file_line(city011[i]);
                for(int j = 0; j < DIMENSION * line; j++)
                {
                        result = fscanf(fp, "%lf", &data[i][count]);
                        if(result == EOF)       break;
                        count++;
```

```
                        count++;
                }
                count = 0;
        }
        fclose(fp);
        return;
}

//二次元配列に変換する（実際は３次元配列）
void city011_file_processing(char city011[][STRING], double data[DATA_SET][FRAME][DIMENSION], double d
{
        int count = 0;
        int line;
        for(int i = 0; i < DATA_SET; i++)
        {
                line = city011_file_line(city011[i]);
                for(int j = 0; j < line; j++)
                {
                        for(int k = 0; k < DIMENSION; k++)
                                data[i][j][k] = data915[i][count++];
                }
        count = 0;
        }
}
```

```
                "./sub.city011/city011_004.txt", "./sub.city011/city011_005.txt", "./sub.city011/city011_006.txt",
                "./sub.city011/city011_007.txt", "./sub.city011/city011_008.txt", "./sub.city011/city011_009.txt",
                "./sub.city011/city011_010.txt", "./sub.city011/city011_011.txt", "./sub.city011/city011_012.txt",
                "./sub.city011/city011_013.txt", "./sub.city011/city011_014.txt", "./sub.city011/city011_015.txt",
                "./sub.city011/city011_016.txt", "./sub.city011/city011_017.txt", "./sub.city011/city011_018.txt",
                "./sub.city011/city011_019.txt", "./sub.city011/city011_020.txt", "./sub.city011/city011_021.txt",
                "./sub.city011/city011_022.txt", "./sub.city011/city011_023.txt", "./sub.city011/city011_024.txt",
                "./sub.city011/city011_025.txt", "./sub.city011/city011_026.txt", "./sub.city011/city011_027.txt",
                "./sub.city011/city011_028.txt", "./sub.city011/city011_029.txt", "./sub.city011/city011_030.txt",
                "./sub.city011/city011_031.txt", "./sub.city011/city011_032.txt", "./sub.city011/city011_033.txt",
                "./sub.city011/city011_034.txt", "./sub.city011/city011_035.txt", "./sub.city011/city011_036.txt",
                "./sub.city011/city011_037.txt", "./sub.city011/city011_038.txt", "./sub.city011/city011_039.txt",
                "./sub.city011/city011_040.txt", "./sub.city011/city011_041.txt", "./sub.city011/city011_042.txt",
                "./sub.city011/city011_043.txt", "./sub.city011/city011_044.txt", "./sub.city011/city011_045.txt",
                "./sub.city011/city011_046.txt", "./sub.city011/city011_047.txt", "./sub.city011/city011_048.txt",
                "./sub.city011/city011_049.txt", "./sub.city011/city011_050.txt", "./sub.city011/city011_051.txt",
                "./sub.city011/city011_052.txt", "./sub.city011/city011_053.txt", "./sub.city011/city011_054.txt",
                "./sub.city011/city011_055.txt", "./sub.city011/city011_056.txt", "./sub.city011/city011_057.txt",
                "./sub.city011/city011_058.txt", "./sub.city011/city011_059.txt", "./sub.city011/city011_060.txt",
                "./sub.city011/city011_061.txt", "./sub.city011/city011_062.txt", "./sub.city011/city011_063.txt",
                "./sub.city011/city011_064.txt", "./sub.city011/city011_065.txt", "./sub.city011/city011_066.txt",
                "./sub.city011/city011_067.txt", "./sub.city011/city011_068.txt", "./sub.city011/city011_069.txt",
                "./sub.city011/city011_070.txt", "./sub.city011/city011_071.txt", "./sub.city011/city011_072.txt",
                "./sub.city011/city011_073.txt", "./sub.city011/city011_074.txt", "./sub.city011/city011_075.txt",
                "./sub.city011/city011_076.txt", "./sub.city011/city011_077.txt", "./sub.city011/city011_078.txt",
                "./sub.city011/city011_079.txt", "./sub.city011/city011_080.txt", "./sub.city011/city011_081.txt",
                "./sub.city011/city011_082.txt", "./sub.city011/city011_083.txt", "./sub.city011/city011_084.txt",
                "./sub.city011/city011_085.txt", "./sub.city011/city011_086.txt", "./sub.city011/city011_087.txt",
                "./sub.city011/city011_088.txt", "./sub.city011/city011_089.txt", "./sub.city011/city011_090.txt",
                "./sub.city011/city011_091.txt", "./sub.city011/city011_092.txt", "./sub.city011/city011_093.txt",
                "./sub.city011/city011_094.txt", "./sub.city011/city011_095.txt", "./sub.city011/city011_096.txt",
                "./sub.city011/city011_097.txt", "./sub.city011/city011_098.txt", "./sub.city011/city011_099.txt",
```

```
120                 "./sub.city011/city011_091.txt", "./sub.city011/city011_092.txt", "./sub.city011/city011_093.txt",
121                 "./sub.city011/city011_094.txt", "./sub.city011/city011_095.txt", "./sub.city011/city011_096.txt",
122                 "./sub.city011/city011_097.txt", "./sub.city011/city011_098.txt", "./sub.city011/city011_099.txt",
123                 "./sub.city011/city011_100.txt"
124         };
125         city011_file_read(city011, city011_data1300);
126         city011_file_processing(city011, city011_data, city011_data1300);
127 //      city011_file_print(city011, city011_data);
128         return;
129 }
130
```

② pattern1.c の説明

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <math.h>
 4
 5 #include "./sub.city011/city011.h"
 6 #include "./sub.city021/city021.h"
 7 #include "./city011_line.h"
 8 #include "./city021_line.h"
 9
10 #define FRAME 150
11 #define DIMENSION 15
12 #define STRING 30       //city0xx_001.txt~city0xx_100.txt
13 #define NUMBER 1500     //city011_001.txt~city011_100.txt
14 #define DATA_SET 100    //データセット100個分
15
16 //局所間の距離を計算
17 void dp_processing(int city011_line_data[DATA_SET], int city021_line_data[DATA_SET], double dp
   [DATA_SET][DATA_SET], double city011_data[DATA_SET][FRAME][DIMENSION], double city021_data[DAT
   A_SET][FRAME][DIMENSION])
18 {
19         int line;
20         //city011のファイルを100個分ソート
21         for(int city_011 = 0; city_011 < DATA_SET; city_011++)
22         {
23                 //city021のファイルを100個分ソート
24                 for(int city_021 = 0; city_021 < DATA_SET; city_021++)
25                 {
26                         if(city011_line_data[city_011] < city021_line_data[city_021])
27                                 line = city021_line_data[city_021];
28                         else
29                                 line = city011_line_data[city_011];
30
```

使用した標準ヘッダは、stdio.h、stdlib.h、math.h である.
標準ライブラリ関数 pow を使って累乗計算を行うため、math.h はインクルードしている.

先ほど作ったヘッダファイル#include “./city011_line.h”などを必要な分インクルードしている.
オブジェクト形式マクロ#define FRAME 150 でフレーム数を、また#define DIMENSION 15 で各フレームの要素を定義している.

dp_processing で dp という配列変数に city011_001.txt から city011_100.txt までと、city021_001.txt から city021_100.txt までの局所間の距離を計算し、その情報を格納している.

```
27                                      line = city021_line_data[city_021];
28                              else
29                                      line = city011_line_data[city_011];
30
31                              for(int i = 0; i < line; i++)
32                                      for(int k = 0; k < DIMENSION; k++)
33                                              dp[city_011][city_021] += pow(city011_data[city_011][i
   ][k] - city021_data[city_021][i][k], 2.0);
34
35                              dp[city_011][city_021] = pow(dp[city_011][city_021], 0.5);
36                      }
37              }
38 }
39
40 //最小値を計算
41 void dp_minimum(int store[DATA_SET], double min[DATA_SET], double dp[DATA_SET][DATA_SET])
42 {
43              for(int i = 0; i < DATA_SET; i++)
44              {
45                      for(int j = 0; j < DATA_SET; j++)
46                      {
47                              //最初は0番目を格納
48                              if(j == 0)
49                                      min[i] = dp[i][j];
50                              //0~99番までの最小値を探す
51                              else
52                              {
53                                      if(min[i] >= dp[i][j])
54                                      {
55                                              min[i] = dp[i][j];
56                                              store[i] = j + 1;
57                                      }
```

dp_minimum()で city011 の N 番目の単語（ファイル）と city021 の n 番目のどの単語が最も距離が近いかを計算し、n 番目の番号を store という配列変数に格納し、その時の最小距離の値を double 型の min という配列変数に格納している.

Proportion()で先ほど求めた、n 番目の番号と N番目の番号と対応させ、番号が一致していたら ratio という変数を+1 する. ratio の合計を 100（全体のファイル数）で除したものが割合をなる. この結果をもとに認識率を判定する.

```
                                if(min[i] >= dp[i][j])
                                {
                                        min[i] = dp[i][j];
                                        store[i] = j + 1;
                                }
                        }
                }
        }
}

//ヒットしたファイルと照らし合わせる
void dp_minimum_print(int store[DATA_SET], double min[DATA_SET])
{
        for(int i = 0; i < DATA_SET; i++)
                printf("city011_%03d.txt---->city021_%03d.txt:%lf\n", i+1, store[i], min[i]);
}

//割合を計算
void proportion(int store[DATA_SET])
{
        int ratio = 0;
        for(int i = 0; i < DATA_SET; i++)
                if((i+1) == store[i])
                        ratio++;
        printf("ratio=%lf\n", (double)ratio / 100.0);
}

int main(void)
{
        double city011_data[DATA_SET][FRAME][DIMENSION];
        double city021_data[DATA_SET][FRAME][DIMENSION];

```

```
77          printf("ratio=%lf\n", (double)ratio / 100.0);
78  }
79
80  int main(void)
81  {
82          double city011_data[DATA_SET][FRAME][DIMENSION];
83          double city021_data[DATA_SET][FRAME][DIMENSION];
84
85          double city011_data1300[DATA_SET][NUMBER];
86          double city021_data1300[DATA_SET][NUMBER];
87
88          int city011_line_data[DATA_SET];        //行数を格納する
89          int city021_line_data[DATA_SET];
90
91          double dp[DATA_SET][DATA_SET];
92          double min[DATA_SET];
93          int store[DATA_SET];
94
95          city011_File_Line(city011_line_data);
96          city021_File_Line(city021_line_data);
97
98          city011_main(city011_data, city011_data1300);
99          city021_main(city021_data, city021_data1300);
100
101         dp_processing(city011_line_data, city021_line_data, dp, city011_data, city021_data);
102         dp_minimum(store, min, dp);
103 //      dp_minimum_print(store, min);
104
105         proportion(store);
106         return 0;
107 }
```

[4] 実験結果

~/pattern1.c（city011, city021）→ 0.290000
~/pattern2.c（city012, city022）→ 0.310000
~/pattern3.c（city011, city012）→ 0.860000
~/pattern4.c（city012, city021）→ 0.300000
~/pattern5.c（city011, city022）→ 0.310000
~/pattern6.c（city021, city022）→ 0.850000

[5] 考察

同一話者の場合は認識率が 8 割を超えているが、非同一話者の場合は 3 割程度しか出せていない. 考えられる原因はデータの読み込みと格納のところで一部正常にいっていない（プログラムが適切に書けていない）ところがあると考えられる. 同一話者の場合、認識率が 8 割を超えているので、局所間の距離の計算や最小値の計算するプログラムの部分に間違いがある可能性は低いと考えられる.
また、今回はデータ量が多いので可読性を意識して書く場合、ファイル分割しいくつか処理

ごとに区切り、メインのファイルにインクルードする形が適切であると考えられる．インクルードが上手くいっていないと考えることもできるが、自分が#define FRAME 150 で設定したフレーム数を超えるファイルがあり、必要なデータが一部読み込めていないケースなども考えられる．またこれから、今回と同じように扱うデータ数が増えてくるとなると、より最適化された効率の良いプログラムを書くべきであり、アルゴリズムをこれら勉強する必要があると感じた．

［6］追加

一つのメインルーチンに city011~city022 までの全てのデータを(ラズパイ 8G の 4B+で)実行したら core dumped になった．これはデータ量が多すぎてオーバーフローになってしまったと考えられる．

［7］追加

反省点としては同じ処理をする関数を作っているので、それを再利用してプログラムの行数を減らして可読性を上げるべきであった．プログラムの行数が減ればその分、処理も軽くなる.普段あまりプログラムを書くことがないので、プログラム全体が動くまでにかなりの労力を費やしたので、時間がなくて改変するまでできなかった．インデントは深くしないように、一つの関数にはなるべく一つの処理というのを意識してプログラムを書いていた．