2011 BIOSTAT 615/815 Homework #4

Due is Tuesday March 8th, 08:30AM

Problem 1. Floyd-Warshall algorithm

Write a program that computes all-pair shortest path by filling out the code from the skeleton below. It is okay to use your own implementation from scratch if you prefer, but the input/output interface should be compatible. Submit your code by email to the instructor, and print the hard copy of your code with example output screenshot.

First, you may use the following implementation of `Matrix615` class to represent a matrix allowing missing values. Note that `boost` library is required to compile the class above.

```
#ifndef __BIOSTAT615_MATRIX_H  // avoid including the same header twice
#define __BIOSTAT615_MATRIX_H

#include <iostream>
#include <fstream>
#include <boost/tokenizer.hpp>
#include <boost/lexical_cast.hpp>
#include <boost/foreach.hpp>

// a generic class for Matrix
template<class T>
class Matrix615 {
protected:   // internal data
  std::vector<T> data;    // using std::vector - object copy is now possible
  int nr, nc;             // # rows and cols
  bool hasMissing;
  T valueMissing;
  std::string strMissing;
public:
  // default constructor
 Matrix615() : nr(0), nc(0), hasMissing(false) {}

  // Allow missing value as a pair of actual value and string value
  void enableMissingValue(const T value, const char* string) {
    hasMissing = true;
    valueMissing = value;
    strMissing = string;
  }

  // resize the dimension of the matrix
  void resize(int nrows, int ncols) {
    nr = nrows;
    nc = ncols;
    data.resize(nr*nc);
  }

  // fill the content
  void fill(T defaultValue) {
    std::fill( data.begin(), data.end(), defaultValue );
  }

  // access individual element
  T& at(int r, int c) { return data[r*nc+c]; }

  int numRows() { return nr; }
  int numCols() { return nc; }

  // print the content of the matrix
  void print(std::ostream& o) {
    for(int i=0; i < nr; ++i) {
      for(int j=0; j < nc; ++j) {
        if ( j > 0 ) o << "\t";
        if ( hasMissing && ( valueMissing == at(i,j) ) )
          o << strMissing;
        else
          o << at(i,j);
      }
      o << std::endl;
    }
  }

  // opens a file to fill the matrix
  void readFromFile(const char* file) {
    std::ifstream ifs(file);
    if ( ! ifs.is_open() ) {
      std::cerr << "Cannot open file " << file << std::endl;
      abort();
    }

    std::string line;
    boost::char_separator<char> sep(" \t");
    typedef boost::tokenizer< boost::char_separator<char> > wsTokenizer;

    nr = nc = 0;
    while( std::getline(ifs, line) ) {
      wsTokenizer t(line,sep);
      for(wsTokenizer::iterator i=t.begin(); i != t.end(); ++i) {
        // if hasMissing is set, convert string "Missing" into special value for Missing
        if ( hasMissing && ( i->compare(strMissing) == 0 ) )
            data.push_back(valueMissing);
        // Otherwise, convert the string to a particular type
        else
          data.push_back(boost::lexical_cast<T>(i->c_str()));
        if ( nr == 0 ) ++nc;  // count # of columns at the first row
      }
      ++nr;
      // when reading each line, make sure that the # of columns match to expectation;
      if ( (int)data.size() != nr*nc ) {
        std::cerr << "The input file is not rectangle at line " << nr << std::endl;
        abort();
      }
    }
  }
};
#endif
```

Finally, you need to fill out the source code below.

```
#include "Matrix615.h"

#define NIL 999999

int main(int argc, char** argv) {
  if ( argc != 2 ) {
    std::cerr << "Usage: floydWarshall [weight matrix]" << std::endl;
    abort();
  }
  Matrix615<int> W;
  W.enableMissingValue(NIL,"NIL");
  W.readFromFile(argv[1]);
  if ( W.numRows() != W.numCols() ) {
    std::cerr << "The input file is not exactly square" << std::endl;
    abort();
  }

  int n = W.numRows();
  Matrix615<int> P;
  P.enableMissingValue(NIL,"NIL");
  P.resize(n,n);

  // Fill out the code in the part marked as *** [FILL HERE] ***

  // *** FILL HERE ** initialize P and W matrix as described in page 695-696 of the textbook

  Matrix615<int> D = W;  // object copy is valid because no pointer is involved

  // print initial D and W matrix
  std::cout << "---------- Initial D  Matrix ---------------------------------" << std::endl;
  D.print(std::cout);
  std::cout << "---------- Initial PI Matrix ---------------------------------" << std::endl;
  P.print(std::cout);

  for(int k=0; k < n; ++k) {
    // *** FILL HERE *** Run floyd-Warshall algorithm for each k and

    // print out D and PI matrix for each iteration
    std::cout << "---------- D  Matrix after covering node " << k << "----------" << std::endl;
    D.print(std::cout);
    std::cout << "---------- PI Matrix after covering node " << k << "----------" << std::endl;
    P.print(std::cout);
  }

  // *** EXTRA POINTS ***  Extra points (10\%) will be given if the optimal path
  //                       for every pair of node is printed below

  return 0;
}
```

An example input and output should look like

```
user@host:~/> cat ./sampleInput.txt
0       10      NIL     5       NIL
NIL     0       1       2       NIL
NIL     NIL     0       NIL     4
NIL     3       9       0       2
7       NIL     6       NIL     0
user@host:~/> ./floydWarshall ./sampleInput.txt
---------- Initial D  Matrix ---------------------------------
0       10      NIL     5       NIL
NIL     0       1       2       NIL
NIL     NIL     0       NIL     4
NIL     3       9       0       2
7       NIL     6       NIL     0
---------- Initial PI Matrix ---------------------------------
NIL     0       NIL     0       NIL
NIL     NIL     1       1       NIL
NIL     NIL     NIL     NIL     2
NIL     3       3       NIL     3
4       NIL     4       NIL     NIL
---------- D  Matrix after covering node 0----------
0       10      NIL     5       NIL
NIL     0       1       2       NIL
NIL     NIL     0       NIL     4
NIL     3       9       0       2
7       17      6       12      0
---------- PI Matrix after covering node 0----------
NIL     0       NIL     0       NIL
NIL     NIL     1       1       NIL
NIL     NIL     NIL     NIL     2
NIL     3       3       NIL     3
4       0       4       0       NIL
---------- D  Matrix after covering node 1----------
0       10      11      5       NIL
NIL     0       1       2       NIL
NIL     NIL     0       NIL     4
NIL     3       4       0       2
7       17      6       12      0
---------- PI Matrix after covering node 1----------
NIL     0       1       0       NIL
NIL     NIL     1       1       NIL
NIL     NIL     NIL     NIL     2
NIL     3       1       NIL     3
4       0       4       0       NIL
---------- D  Matrix after covering node 2----------
0       10      11      5       15
NIL     0       1       2       5
NIL     NIL     0       NIL     4
NIL     3       4       0       2
7       17      6       12      0
---------- PI Matrix after covering node 2----------
NIL     0       1       0       2
NIL     NIL     1       1       2
NIL     NIL     NIL     NIL     2
NIL     3       1       NIL     3
4       0       4       0       NIL
---------- D  Matrix after covering node 3----------
0       8       9       5       7
NIL     0       1       2       4
NIL     NIL     0       NIL     4
NIL     3       4       0       2
7       15      6       12      0
---------- PI Matrix after covering node 3----------
NIL     3       1       0       3
NIL     NIL     1       1       3
NIL     NIL     NIL     NIL     2
NIL     3       1       NIL     3
4       3       4       0       NIL
---------- D  Matrix after covering node 4----------
0       8       9       5       7
11      0       1       2       4
11      19      0       16      4
9       3       4       0       2
7       15      6       12      0
---------- PI Matrix after covering node 4----------
NIL     3       1       0       3
4       NIL     1       1       3
4       3       NIL     0       2
```

```
4       3       1       NIL     3
4       3       4       0       NIL
```

For the extra-points problem, example output is below.

```
Optimal path 1 <- 0 (d = 8) : 1 <-(3)-- 3 <-(5)-- 0
Optimal path 2 <- 0 (d = 9) : 2 <-(1)-- 1 <-(3)-- 3 <-(5)-- 0
Optimal path 3 <- 0 (d = 5) : 3 <-(5)-- 0
Optimal path 4 <- 0 (d = 7) : 4 <-(2)-- 3 <-(5)-- 0
Optimal path 0 <- 1 (d = 11) : 0 <-(7)-- 4 <-(2)-- 3 <-(2)-- 1
Optimal path 2 <- 1 (d = 1) : 2 <-(1)-- 1
Optimal path 3 <- 1 (d = 2) : 3 <-(2)-- 1
Optimal path 4 <- 1 (d = 4) : 4 <-(2)-- 3 <-(2)-- 1
Optimal path 0 <- 2 (d = 11) : 0 <-(7)-- 4 <-(4)-- 2
Optimal path 1 <- 2 (d = 19) : 1 <-(3)-- 3 <-(5)-- 0 <-(7)-- 4 <-(4)-- 2
Optimal path 3 <- 2 (d = 16) : 3 <-(5)-- 0 <-(7)-- 4 <-(4)-- 2
Optimal path 4 <- 2 (d = 4) : 4 <-(4)-- 2
Optimal path 0 <- 3 (d = 9) : 0 <-(7)-- 4 <-(2)-- 3
Optimal path 1 <- 3 (d = 3) : 1 <-(3)-- 3
Optimal path 2 <- 3 (d = 4) : 2 <-(1)-- 1 <-(3)-- 3
Optimal path 4 <- 3 (d = 2) : 4 <-(2)-- 3
Optimal path 0 <- 4 (d = 7) : 0 <-(7)-- 4
Optimal path 1 <- 4 (d = 15) : 1 <-(3)-- 3 <-(5)-- 0 <-(7)-- 4
Optimal path 2 <- 4 (d = 6) : 2 <-(6)-- 4
Optimal path 3 <- 4 (d = 12) : 3 <-(5)-- 0 <-(7)-- 4
```

Problem 2. Biased coin example of Hidden Markov Model

Implement a forward-backward algorithm and Viterbi algorithm for the biased coin example described in the class. The hidden state $S = \{0, 1\}$ represents fair (F) and biased (B) states respectively, and the observation $O = \{0, 1\}$ represents head (H) and tail (T), respectively. The parameters are

$$\begin{aligned}
\pi &= (0.9 0.1)^T \\
\Pr(S_t = 0|S_{t-1} = 0) &= 0.95 \\
\Pr(S_t = 1|S_{t-1} = 1) &= 0.8 \\
\Pr(O_t = 0|S_t = 0) &= 0.5 \\
\Pr(O_t = 0|S_t = 1) &= 0.9
\end{aligned}$$

You may want to complete the code from the skeleton below. Use the `simulCoinInput.txt` file in the class web page as a sample input. An example output files are attached `simulCoinOutput.txt` for your reference.

Note that a naive implementation may suffer from precision problem, but you may ignore them. If your implementation is robust against precision issue with large-sized input (e.g. 1 million), you will get an extra credit. Submit your full source code to instructor by E-mail, and attach the hard copy of your source code in your submission.

```
#include <cstdlib>
#include <iomanip>
#include <iostream>

#include "Matrix615.h"

#define N_STATES 2
#define N_DATA 2

const char* stateLabels[N_STATES] = {"F","B"};
const char* dataLabels[N_DATA] = {"H","T"};

class BiasedCoinHMM {
```

```
protected:
  int T;
  // HMM initial parameters
  std::vector<double> pi; // N_STATES * 1 vector
  Matrix615<double> A;    // N_STATES * N_STATES matrix
                          //  : A_{ij} is \Pr(q_t=i|q_{t-1}=j)
  Matrix615<double> B;    // N_OBS * N_STATES matrix
                          //  : B_{ij} = b_j(o_i) = \Pr(o_t=i|q_t=j)

  // Data (observations)
  std::vector<int> o; // vector observations

  // forward-backward probability
  Matrix615<double> alphas; // T * N_STATES : alphas[i,j] = alpha_i(i)
  Matrix615<double> betas;  // T * N_STATES : betas[i,j]  = beta_j(i)
  Matrix615<double> gammas; // T * N_STATES : gammas[i,j] = gammas_j(i) = Pr(q_t=i|o_t=i)

  // viterbi probability and paths
  Matrix615<double> deltas; // T * N_STATES
  Matrix615<int> phis;   // T * N_STATES
  std::vector<int> mleStates; // vector of MLE states

public:
  BiasedCoinHMM(double priors[N_STATES], double trans[N_STATES][N_STATES], double emis[N_DATA][N_STATES]) {
    for(int i=0; i < N_STATES; ++i) {
      pi.push_back(priors[i]);
    }

    A.resize(N_STATES,N_STATES);

    for(int i=0; i < N_STATES; ++i) {
      for(int j=0; j < N_STATES; ++j) {
        A.at(i,j) = trans[i][j];
      }
    }

    B.resize(N_DATA,N_STATES);
    for(int i=0; i < N_DATA; ++i) {
      for(int j=0; j < N_STATES; ++j) {
        B.at(i,j) = emis[i][j];
      }
    }

    T = 0;
  }

  void loadObservations(std::vector<int>& obs) {
    o = obs;
    T = o.size();

    alphas.resize(T,N_STATES);
    betas.resize(T,N_STATES);
    gammas.resize(T,N_STATES);
    deltas.resize(T,N_STATES);
    phis.resize(T,N_STATES);
    mleStates.resize(T);
  }

  void computeForwardBackward() {
    // *** FILL OUT *** to compute
    // forward and backward probabilities into alphas, betas, and gammas
  }

  void computeViterbiPath() {
    // *** FILL OUT *** to compute
    // viterbi path and likelihood into deltas, phis, and mleStates,
  }

  void outputResults(std::ostream& os, std::vector<int>& trueStates) {
    if ( T != (int)trueStates.size() ) {
      std::cerr << "True states are in different length with HMM" << std::endl;
      abort();
    }

    os << "#SEQ\tTRUE_S\tOBS\tP(q|o)\tMLE_S" << std::endl << std::fixed << std::setprecision(4);
    for(int t = 0; t < T; ++t) {
      os << t+1 << "\t"
         << stateLabels[trueStates[t]] << "\t"
         << dataLabels[o[t]] << "\t"
         << gammas.at(t,1) << "\t"  // prints Pr(q_t=1|o)
         << stateLabels[mleStates[t]] << "\n";
    }
  }
};

int main(int argc, char** argv) {
  if ( argc != 2 ) {
    std::cerr << "Usage: coinHMM [inputFile]" << std::endl;
    return -1;
  }

  std::vector<int> trueStates;
  std::vector<int> observations;
  std::ifstream ifs(argv[1]);

  if ( !ifs.is_open() ) {
    std::cerr << "Cannot open file " << argv[1] << std::endl;
    return -1;
  }

  std::string tok;
  for(int i=0; ifs >> tok; ++i) {
    if ( i % 3 == 1 ) {
      if ( tok.compare("F") == 0 ) {
        trueStates.push_back(0);
      }
      else if ( tok.compare("B") == 0 ) {
        trueStates.push_back(1);
      }
      else {
        std::cerr << "Cannot recognize state " << tok << std::endl;
      }
    }
    else if ( i % 3 == 2 ) {
      if ( tok.compare("H") == 0 ) {
        observations.push_back(0);
      }
      else if ( tok.compare("T") == 0 ) {
        observations.push_back(1);
      }
      else {
        std::cerr << "Cannot recognize observation " << tok << std::endl;
      }
    }
  }

  std::cout << "Finished reading " << trueStates.size() << " states and observations.." << std::endl;

  double trans[N_STATES][N_STATES] = { {0.95,0.2}, {0.05,0.8} };
  double emis[N_DATA][N_STATES] = { {0.5,0.9}, {0.5,0.1} };
  double pi[N_STATES] = {0.9,0.1};

  BiasedCoinHMM bcHMM(pi, trans, emis);

  bcHMM.loadObservations(observations);
  bcHMM.computeForwardBackward();
  bcHMM.computeViterbiPath();
  bcHMM.outputResults(std::cout, trueStates);
  return 0;
}
```