2011 BIOSTAT 615/815 Homework #3

- Due is Tuesday November 1st, 08:40AM (before the class starts).
- You need to both (1) hand-in your hard copy of the source code (using smaller font is fine), and (2) submit your source codes (ready to compile and run) in a zip or tar.gz compressed format by email.

Problem 1. Dynamic programming

Write a program that calculates binomial coefficients, based on the following recursive rule.

$$\begin{aligned} \binom{n}{k} &= \binom{n-1}{k} + \binom{n-1}{k-1} \\ \binom{n}{0} &= \binom{n}{n} = 1 \end{aligned}$$

A suggested skeleton of the program is given below. Write down the full function `binom()`, and compute the value when $n = 30, k = 15$ using the implemented program.

```
#include <iostream>

int binom(int n, int k, int** stored) {
  // fill in the function
}

int main(int argc, char** argv) {
  if ( argc != 3 ) {
    std::cerr << "Usage: " << argv[0] << " [n] [k] " << std::endl;
    return -1;
  }
  int n = atoi(argv[1]);
  int k = atoi(argv[2]);
  if ( k > n ) {
    std::cerr << "n = " << n << " is smaller than k = " << k << std::endl;
    return -1;
  }

  int** v = new int* [n+1];
  for(int i=0; i < n+1; ++i) {
    v[i] = new int[k+1]();
  }

  int binomCoefficient = binom(n, k, v);
  std::cerr << "choose(" << n << "," << k << ") = " << binomCoefficient << std::endl;

  for(int i=0; i < n+1; ++i) {
    delete[] v[i];
  }
  delete[] v;
  return 0;
}
```

Problem 2 - Edit Distance Problem

Fill out the printEdits() function editDistance problem.

```
#include <iostream>
#include <climits>
#include <string>
#include <vector>

template <class T>
class Matrix615 {
public:
  std::vector< std::vector<T> > data;
  Matrix615(int nrow, int ncol, T val = 0) {
    data.resize(nrow); // make n rows
    for(int i=0; i < nrow; ++i) {
      data[i].resize(ncol,val); // make n cols with default value val
    }
  }
  int rowNums() { return (int)data.size(); }
  int colNums() { return ( data.size() == 0 ) ? 0 : (int)data[0].size(); }
};

int editDistance(std::string& s1, std::string& s2, Matrix615<int>& cost, Matrix615<int>& move, int r, int c) {
  int iCost = 1, dCost = 1, mCost = 1; // insertion, deletion, mismatch cost

  if ( cost.data[r][c] == INT_MAX ) {
    if ( r == 0 && c == 0 ) { cost.data[r][c] = 0; }
    else if ( r == 0 ) {
      move.data[r][c] = 0;  // only insertion is possible
      cost.data[r][c] = editDistance(s1,s2,cost,move,r,c-1) + iCost;
    }
    else if ( c == 0 ) {
      move.data[r][c] = 1;  // only deletion is possible
      cost.data[r][c] = editDistance(s1,s2,cost,move,r-1,c) + dCost;
    }
    else {  // compare 3 different possible moves and take the optimal one
      int iDist = editDistance(s1,s2,cost,move,r,c-1) + iCost;
      int dDist = editDistance(s1,s2,cost,move,r-1,c) + dCost;
      int mDist = editDistance(s1,s2,cost,move,r-1,c-1) + (s1[r-1] == s2[c-1] ? 0 : mCost);
      if ( iDist < dDist ) {
        if ( iDist < mDist ) { // insertion is optimal
          move.data[r][c] = 0;
          cost.data[r][c] = iDist;
        }
        else {
          move.data[r][c] = 2; // match is optimal
          cost.data[r][c] = mDist;
        }
      }
      else {
        if ( dDist < mDist ) {
          move.data[r][c] = 1; // deletion is optimal
          cost.data[r][c] = dDist;
        }
        else {
          move.data[r][c] = 2; // match is optimal
          cost.data[r][c] = mDist;
        }
      }
    }
```

```
  }
  return cost.data[r][c];
}

int printEdits(std::string& s1, std::string& s2, Matrix615<int>& move) {
  // *** FILL THIS FUNCTION ***









}

int main(int argc, char** argv) {
  if ( argc != 3 ) {
    std::cerr << "Usage: editDistance [str1] [str2]" << std::endl;
    return -1;
  }
  std::string s1(argv[1]);
  std::string s2(argv[2]);

  Matrix615<int> cost(s1.size()+1, s2.size()+1, INT_MAX);
  Matrix615<int> move(s1.size()+1, s2.size()+1, -1);

  int optDist = editDistance(s1, s2, cost,move, cost.rowNums()-1, cost.colNums()-1);

  std::cout << "EditDistance is " << optDist << std::endl;
  printEdits(s1, s2, move);

  return 0;
}
```

Example outputs are given below.

```
$ ./editDistance FOOD MONEY
EditDistance is 4
*-I**
FO-OD
MONEY

$ ./editDistance ALGORITHM ALTRUISTIC
EditDistance is 6
--***-I-**
ALGORI-THM
ALTRUISTIC

$ ./editDistance AAACTAG AACTAGGG
EditDistance is 3
D-----II-
AAACTA--G
-AACTAGGG
```

Problem 3. Hidden Markov Model

```
#ifndef __HMM_615_H
#define __HMM_615_H

#include "Matrix615.h"

class HMM615 {
 public:
  // parameters
  int nStates;  // n : number of possible states
  int nObs;     // m : number of possible output values
  int nTimes;   // t : number of time slots with observations
  std::vector<double> pis; // initial states
  std::vector<int> outs;   // observed outcomes
  Matrix615<double> trans;   // trans[i][j] corresponds to A_{ij}
  Matrix615<double> emis;

  // storages for dynamic programming
  Matrix615<double> alphas;
  Matrix615<double> betas;
  Matrix615<double> gammas;
  Matrix615<double> deltas;
  Matrix615<int> phis;
  std::vector<int> path;

 HMM615(int states, int obs, int times) : nStates(states), nObs(obs), nTimes(times), trans(states, states, 0),
     emis(states, obs, 0), alphas(times, states, 0), betas(times, states, 0),
     gammas(times, states, 0), deltas(times, states, 0), phis(times, states, 0)
  {
    pis.resize(nStates);
    path.resize(nTimes);
  }

  void forward() {
    for(int i=0; i < nStates; ++i) {
      alphas.data[0][i] = pis[i] * emis.data[i][outs[0]];
    }
    for(int t=1; t < nTimes; ++t) {
      for(int i=0; i < nStates; ++i) {
        alphas.data[t][i] = 0;
        for(int j=0; j < nStates; ++j) {
          alphas.data[t][i] += (alphas.data[t-1][j] * trans.data[j][i] * emis.data[i][outs[t]]);
        }
      }
    }
  }

  void backward() {
    // fill betas
```

```
  }

  void forwardBackward() {
    forward();
    backward();

    for(int t=0; t < nTimes; ++t) {
      double sum = 0;
      for(int i=0; i < nStates; ++i) {
        sum += (alphas.data[t][i] * betas.data[t][i]);
      }
      for(int i=0; i < nStates; ++i) {
        gammas.data[t][i] = (alphas.data[t][i] * betas.data[t][i])/sum;
      }
    }
  }

  void viterbi() {
    // fill deltas and phi









    // backtrack viterbi path by filling path









  }
};
#endif // __HMM_615_H
```

1. Complete the HMM615 class below to run the biasedCoin example given in the class note.
2. Modify the `biasedCoin.cpp` to allow 3 states, "FAIR", "HEAD-BIASED", "TAIL-BIASED", using the following parameters:
   - $\pi = (5/7, 1/7, 1/7)$
   - $A = \begin{pmatrix} 0.96 & 0.02 & 0.02 \\ 0.10 & 0.80 & 0.10 \\ 0.10 & 0.10 & 0.80 \end{pmatrix}$
   - $B = \begin{pmatrix} 0.5 & 0.5 \\ 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$

   Run the program with example inputs given below and report the output

3. (815 only) The HMM algorithm above does suffer from precision problem with large number of observations (e.g. 10,000). Where does the the precision problem come from? Modify the HMM615 class to overcome the precision problem with large number of inputs. (You may need to modify multiple places of the class)