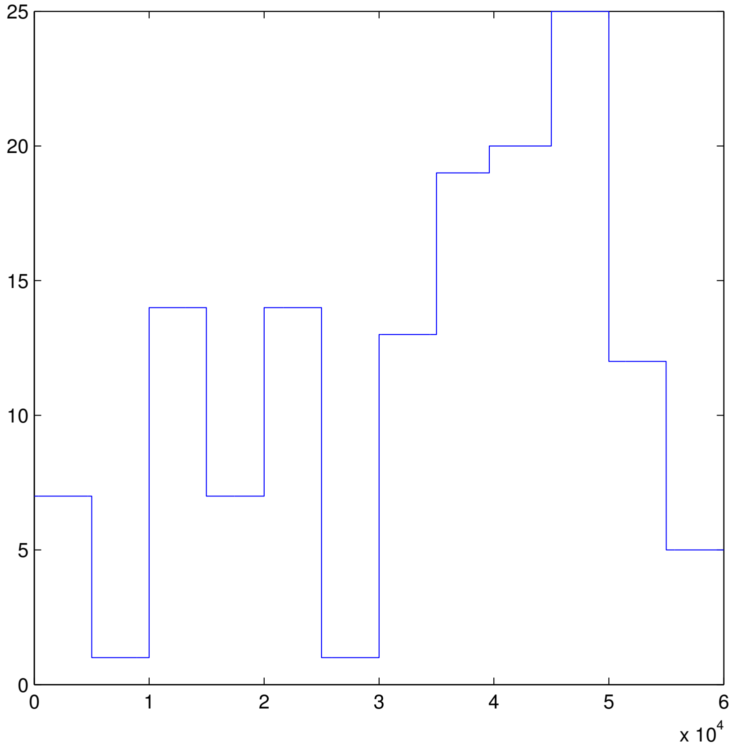

Figure 1: Most likely sequence

The sequence is **gangnamstyle**
The following is the Matlab source code

```
//
//  main.cpp
//  cs_hw5_verbiti
//
//  Created by Ning Ma on 2/23/13.
//  Copyright (c) 2013 Ning Ma. All rights reserved.
//
#include <iomanip>
#include <iostream>
#include <fstream>
#include <sstream>// for istringstream
#include <vector>
#include <set>
#include <algorithm>
#include <cmath>

using namespace std;
```

```
//read initial data from the txt files
void initialize(string a, vector<vector<double> >& matrix)
{
    string line;    vector<double> row;
    double number = 0;
    ifstream myfile;
    myfile.open(a);
    if(!myfile)
    {
        cout << "can't open or find the file" << endl;
        exit(1);
    }
    while(getline(myfile,line))//get next line from the file
    {

        if(line.empty())
            continue;
        istringstream inputstream(line);//separtae the line based on the spaces betwe
        row.clear();

        while( inputstream >> number )
        {
            //cout << number << " ";
            row.push_back(number);//get a row of matrix
        }
        //cout << endl;
        matrix.push_back(row);//get the matrix
    }
    myfile.close();

}


int main(int argc, const char * argv[])
{

    //L_matrix : compute the matrix L_it;
    //Theta_matrix : the matrix which record the most likely transition
    vector<vector<double> >  Ini_state, A_transition, B_emission, O_observation;
    vector<double> temp_column;
    vector<vector<double> >::iterator iter_r;
    vector<double>::iterator iter;
    double max_value;

    typedef const vector<double>::size_type CONST_vec_sz;
    typedef const vector<vector<double> >::size_type CONST_vec_vec_sz;
    typedef vector<double>::size_type vec_sz;
    typedef vector<vector<double> >::size_type vec_vec_sz;
```

```
vec_vec_sz j, i;
vec_sz t, max_index;

initialize("emissionMatrix.txt", B_emission);
initialize("initialStateDistribution.txt", Ini_state);
initialize("transitionMatrix.txt", A_transition);
initialize("observations.txt",O_observation);

CONST_vec_vec_sz N_state = Ini_state.size();//number of states
CONST_vec_sz T = O_observation[0].size(); //number of obervations
cout << N_state << " " << T << endl;
vector<vector<double> >L_matrix(N_state, vector<double>(T));
vector<vec_sz> S_star(T);
vector<vector<vec_sz> > Theta_matrix(N_state, vector<vec_sz>(T));

temp_column.clear();
for(i = 0; i < N_state; i++)
{
    L_matrix[i][0] = log( Ini_state[i][0] ) + log( B_emission[i][O_observation[0]
    //cout << L_matrix[i][0] << endl;
}
for( t = 1; t < T; t++ )
{
    for ( j = 0; j < N_state; j++)
    {

        for( i = 0; i < N_state; i++)
        {
            temp_column.push_back(L_matrix[i][t-1] + log(A_transition[i][j]));
            //cout << temp_column[i] <<endl;
        }

        iter = max_element( temp_column.begin(),temp_column.end() );
        max_index = iter - temp_column.begin();
        max_value = *iter;
        L_matrix[j][t] = max_value + log(B_emission[j][O_observation[0][t]]);
        Theta_matrix[j][t] = max_index;
        temp_column.clear();
        //cout << Theta_matrix[j][t] << endl;

    }


}
//cout << Theta_matrix[16][2] << endl;
for(i = 0; i < N_state; i++)
    temp_column.push_back(L_matrix[i][T-1]);
iter = max_element( temp_column.begin(),temp_column.end() );
max_index = iter - temp_column.begin();
```

```
    S_star[T-1] = max_index+1;
    cout << S_star[T-1] << endl;

    for (t = T-2; t > 0; t--)
    {
        S_star[t] = Theta_matrix[S_star[t+1]-1][t+1]+1;
        //cout << S_star[t] << " " << t << endl;
    }
    S_star[0] = Theta_matrix[S_star[1]][1];
    /*for(i = 0; i < O_observation.size(); i++)
    {
        row = O_observation[i];
        for (j = 0; j < row.size(); j++)
        {
            cout << row[j] << " ";
        }
        cout << endl;

    }*/
    ofstream outfile;
    outfile.open("S_star.txt");
    if(!outfile)
        cout << "can't open a file" <<endl;
    for (t = 0;t < T;t++)
        outfile << S_star[t] <<endl;
    outfile.close();

    return 0;
}
```