```
/************************************************************/
/*        GetInter.cpp                                      */
/*          Hui Jin                                         */
/************************************************************/
/*
   file description:
       Get appropriate permutation for LDGM (LDPC) code for given
       degree sequence L and R.
       In first version. R sequence is constant.
   Author: Hui Jin
   date:   3/12/2000

   modification:  3/12/2000   first version.
                              One important thing is, L and R should
be
                              in decreasing order.
                  4/5/2000    Second version. Leave GetInter() there.
                              implement new version newGetInter()
using
                              Gallager idea, similar to his ensemble
but has
                              variable and check nodes reversed. See
page 38
                              Book III.
                  4/9/2000    Method II has problems, return to
GetInter(),
                              but change the way the next number is
                              generated.
                                L and R should be
                              in decreasing order !!!!
                              And we should use the same .prm in
IRAsimu.
                  4/12/2000   We are going to arrange the degree 2
nodes on
                              the left, because those are causing
problems of
                              short cycles in decoding.
                              The main idea is to arrange them as a
path on
                              nodes 0, t, 2t, 3t, ... (n-1)t, where t
is an
                              integer that gcd(t,n)=1, and t is around
                              sqrt(n). So this is like the right part
except
                              in different order.
                              In this way, We can make sure the
                              shortest cycle consisting only degree 2
node is
                              around 2t. Which is good.
                              Degree sequence is still in decreasing
```

```
order,
                                    but we take care of degree 2 nodes
first. The
                                    rest the same as 4/9/2000 version.
*/




#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

#include "vector.h"
#include "ldpc.h"
#include "paramfio.h"
#include "datatype.hh"
#include "random.hh"



void printusage()
{
        printf("Usage: GetInter filename.prm infolen \n");
        exit(1);
}

void getLRdegree(degree_sequence& L, degree_sequence &R, char *
filename)
{
        ivector deg;
        vector fe;
        paramfio P;
        P.set_filename(filename);
        P.read("variable degree", deg);
        P.read("variable fraction", fe);
        L.init(deg.getsize());
        L.d = deg;
        L.fe = fe;
        P.read("check degree", deg);
        P.read("check fraction", fe);
        R.init(deg.getsize());
        R.d = deg;
        R.fe = fe;
        L.edge_to_node();
        R.edge_to_node();
}
```

```
/*
delete the index term from a list with length=len.
  index is in [0, len-1].
*/

void list_del(int list[], int len, int index)
{
  for(int i=index; i<len-1; i++)
     list[i]=list[i+1];
}

/*
3/12/2000   Hui Jin
get an interleaver for LDGM code.
Any two edge in the same variable node won't be in the same check
node, we
garantee the second part by imposing if two edges are adjacent to the
same
variable node, their permutation must be with distance >=k, which can
be
adjacent to the same check node.
help function:  void list_del(int list[], int len, int index)
modification:   4/9/2000
        only check distance property in the same node.
*/
void getInter(int *_interleaver, degree_sequence & LV, int nv[], int
_len, int check_deg)
{

   //how many numbers left in the pool.
   int remain_len = _len;
   int k=0;
   int distance;

   int s_ch;

   //the set of all the numbers.
   int list[_len];
   int list_index;
   for(int m=0; m< _len; m++) list[m]=m;


   RandomGenerator rand;

   for(int i=0; i< LV.n; i++)
      for(int l=0; l< nv[i]; l++)
         for(int j=0; j<LV.d[i]; j++)
           {
           //randomly get a number in the remaining numbers.
           list_index = (int) floor(rand.UNI()*remain_len);
```

```
          _interleaver[k]=list[list_index];
         distance=2*check_deg;
          //check if all the close positions have a permutation  with
          // appropriate distance.
         s_ch=k-j;
         while(s_ch<k)
           { //if degree 2 nodes, special care.
             if(LV.d[i]==2)  distance = 200;
             if(abs(_interleaver[s_ch]-_interleaver[k])>= distance)
s_ch++;
             else
                 {
                  list_index = (int) floor(rand.UNI()*remain_len);
                  _interleaver[k]=list[list_index];
                  s_ch=k-j;
                  }
            }
       //we found the appropriate number, now delete that element
from the
       //  remain list.
       list_del(list, remain_len, list_index);
        k++;
        remain_len--;
      }


}




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



        if (argc !=3)
        {
                printusage();
        }


        degree_sequence LV;
        degree_sequence LC;
        int argcount = 1;

       getLRdegree(LV, LC, argv[argcount++]);

       int info_len= atoi(argv[argcount++]);
```

```
        int nv[LV.n];
        int LEdge_num=0;
        double s=0;
        for(int i=0; i< LV.n-1; i++)
              {
              nv[i]= int(floor((s+LV.fn[i])*info_len) -
floor(s*info_len));
              s+=LV.fn[i];
              LEdge_num += nv[i] * LV.d[i];
               }

         nv[LV.n-1]= info_len -(int) floor(s*info_len);
         LEdge_num += nv[LV.n-1] * LV.d[LV.n-1];


        //assume here LC.av_degree() is constant
        int check_deg= (int) floor(LC. av_degree_node());

        int _len= LEdge_num;

        cout << "Total length permutation " << _len << endl;

        int _Interleaver[_len];



        getInter(_Interleaver, LV, nv, _len, check_deg);


        for(int i=0; i< _len; i++)
           cout << _Interleaver[i]<< endl;




}
```

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.