```
/* FILE: tournament.cpp   last change: 28-July-2012   authors: Marco Palama', Francesco Martinelli, Romeo Rizzi
 * This program is a solver for problem 1 (tournament) in 25-06-2012 exam in Algorithms
 */
/*
 * APPROACH: noi costruiamo il cammino in modo greedy prendendo sempre come prossimo nodo da visitare un qualsiasi nodo di grado
 * residuo massimo preso tra quelli presenti nell'out-neighboorhood del nodo corrente nel tournament residuo.
 * (Ad ogni passo, il tournament residuo consta dei nodi non ancora visitati piu' il nodo corrente.)
 * Dimostreremo che tale algoritmo Greedy costruisce sempre un cammino che visita tutti i nodi (cammino hamiltoniano).
 * La dimostrazione si basera' sul seguente lemma e sul di lui ovvio corollario.
 * Lemma: se in un tournament non vi e' un cammino da u a v allora l'out-degree di v eccede quello di u.
 * proof: se ci fosse un qualche nodo x nell'out-neighbourhood di u ma non in quello di v  allora avremmo il cammino u-> x-> v.
 *        Inoltre l'arco tra u e v e' diretto da v ad u altrimenti avremmo il cammino di un solo arco u-> v. QED
 * Corollario: ogni nodo di un tournament e' raggiungibile da un qualsiasi nodo di grado massimo. (proof: e' dura a superare il massimo. QED)
 */

// #include <iostream>  // NOTA: ho commentato questa riga per essere sicuro di non aver lasciato operazioni di input/output di debug nella fretta di consegnare, se compila cosi' non ci sono ed io non perdo stupidamente i miei punti-esame.
#include <fstream>
#include <cassert>

using namespace std;

const int MAX_N = 2000;
int n;   // Numero giocatori (numero di nodi nel tournament, un tournament e' un grafo diretto completo).

int T[MAX_N +1][MAX_N +1];   // i nodi sono numerati da 1 ad n
#define NONE 0               // e quindi lo 0 e' libero come flag di "non ancora trovato" o "nessuno"

// Matrice vittorie/sconfitte:
// T[i][j] = 1 se i batte j, T[i][j] = 0 in caso contrario (ossia se e' j a battere i)
// T[i][0] contiene l'out-degree del nodo i nel tournament residuo
//         ed e' messo a 0 per quegli i che non appartengono al tournament residuo.


ofstream fout;

/* Procedura ricorsiva che stampa un cammino hamiltoniano (del tournament residuo) partente dal nodo <nodo>
 * @param <nodo> nodo da cui il cammino deve dipartire, ed implicitamente il tournament residuo memorizzato nella variabile globale T[][].
 * @precond  nel tournament residuo ogni nodo e' raggiungibile partendo da <nodo>.
 */
void print_greedy_path_from_node(int nodo) {
  int maxOutDeg = -1, maxNodo = NONE; // usate per individuare un nodo <v> di out-degree massimo nell'out-neighboorhood di <nodo>
  for(int v=1; v<=n; v++) {  // scelta del successore e rimozione dal tournament T del nodo corrente <nodo>
      if( T[nodo][v] ) // se <nodo> batte <v> potremmo fare step su v
         if( T[v][0] > maxOutDeg ) { // di tutti questi candidati su cui fare step ne prendo uno di out-degree massimo
            maxOutDeg = T[v][0];
            maxNodo = v;
         }
```



```
      // rimozione del nodo corrente <nodo>
      if( T[v][nodo] ) {
         T[v][nodo] = 0; // non c'e' piu' l'arco da <v> a <nodo>, il cammino non dovra' piu' tornare in <nodo>
         T[v][0]--;      // maintenance dei gradi dei nodi nel tournament residuo
      }
   }

   // Se da <nodo>, nodo corrente del cammino, non ci sono piu' archi uscenti, allora il cammino termina.
   if (maxNodo==NONE) return;  // abbiamo infatti finito: il tournament residuo consta ora del solo nodo <nodo> !
                               // proof: da <nodo> non possiamo uscire, mentre ogni altro nodo sarebbe raggiungibile
                               //        partendo da <nodo> in virtu' della precondizione. QED
   fout << nodo << " " << maxNodo << endl; // Fai step da nodo a maxNodo

   print_greedy_path_from_node(maxNodo);   // E avanti cosi' ...
      /* KEY FACT: Questa chiamata rispetta la precondizione.
       * proof: si noti che nel nuovo tournament residuo ogni nodo v e' raggiungibile da maxNodo: v e' infatti raggiungibile
       * da un qualche nodo dell'out-neighboorhood ON(nodo) di nodo in quanto raggiungibile da nodo nel vecchio tournament;
       * inoltre ogni nodo di ON(nodo) e' raggiungibile da maxNode in virtu' del Lemma. -QED                              */
}

int main() {
  ifstream fin("input.txt");  assert( fin );
  fin >> n;
  int maxOutDeg = -1, maxNodo = NONE; // usate per individuare un nodo nell'out-neighboorhood di v con out-degree massimo
  for(int i=1; i<=n; i++) {
        for(int j=1; j<=n; j++) {
           fin >>T[i][j];
           if(T[i][j]==1) T[i][0]++; // T[i][0] = Grado uscente del nodo i
        }
        if( T[i][0] > maxOutDeg ) {
           maxOutDeg = T[i][0];
           maxNodo = i;
        }
  }
  fin.close();

  fout.open("output.txt");  assert( fout );
  fout << n << endl; // Teorema: ogni tournament ammette un cammino Hamiltoniamo (come dimostrato nel presente documento)

  print_greedy_path_from_node(maxNodo); // Ogni nodo e' raggiungibile da maxNode in virtu' del Corollario. Questa prima chiamata rispetta la precondizione!
  fout.close();

  return 0;
}
```