```
#include <stdlib.h>
#include <stdio.h>
#include "graph.h"

// An implementation of Bron-Kerbosch  algorithm
// From Algorithm 457 of the Collected Algorithms from CACM
// http://www.netlib.org/tomspdf/457.pdf

void Graph::findMaxClique( void ) {
      int i;
      int *all = (int *) malloc(N*sizeof(int));
      for (i=0; i<N; i++)
            all[i] = i;
      bkv2( all, 0, N );
      free( all );
}

// recursive function version 2 of Bron-Kerbosch  algorithm
void Graph::bkv2( int* oldSet, int ne, int ce ) {
      int *newSet = (int *)malloc(ce*sizeof(int));
      int nod, fixp;
      int newne, newce, i, j, count, pos, p, s, sel, minnod;

      minnod = ce;
      nod = 0;

      // Determine each counter value and look for minimum

      for ( i = 0 ; i <ce && minnod != 0; i++) {
            p = oldSet[i];
            count = 0;

            // Count disconnections
            for (j = ne; j < ce && count < minnod; j++)
                  if (!connected[p][oldSet[j]]) {
                        count++;
                        // Save position of potential candidate
                        pos = j;
                  }

            // Test new minimum
            if (count < minnod) {
                  fixp = p;
                  minnod = count;
                  if (i<ne)
                        s = pos;
                  else {
                        s = i;
                        // pre-increment
                        nod = 1;
                  }
            }
      }
      // If fixed point initially chosen from candidates then
      // number of diconnections will be preincreased by one

      // Backtrackcycle
      for (nod=minnod+nod; nod>=1; nod--) {
            // Interchange
            p = oldSet[s];
            oldSet[s] = oldSet[ne];
            sel = oldSet[ne] = p;

            // Fill new set "not"
            newne = 0;
            for ( i = 0 ; i < ne ; i++)
                  if ( connected[sel][oldSet[i]] )
                        newSet[newne++] = oldSet[i];


            // Fill new set "cand"
            newce = newne;
            for (i=ne+1; i<ce; i++)
                  if ( connected[sel][oldSet[i]] )
                        newSet[newce++] = oldSet[i];

            // Add to compsub
            compsub.add( sel );

            if (newce == 0) {
                  // found a max clique
                  compsub.print();

          } else if (newne < newce)
                  bkv2( newSet, newne, newce );

            // Remove from compsub
            compsub.remove();

            // Add to "not"
            ne++;
            if (nod > 1)
                  // Select a candidate disconnected to the fixed point
                  for ( s = ne ; connected[fixp][oldSet[s]] ; s++)
                        ;
                  // nothing but finding s

      } /* Backtrackcycle */
      free( newSet );
}
```