Problem of the Day:

Draw pictures of all the caterpilars that have 6 vertices.

Announcements:

Assignment #1 has 3 parts:

Part A: Literature Review [30 points]:

Upload your work to connex under Assignment #1A.

Part A is due: Wednesday Sept. 28 at 11:55pm.

Part B: Programming Question [30 points]:

Upload your work to connex under Assignment #1B.

Part B is due: Wednesday Sept. 28 at 11:55pm.

Part C: Written questions [40 points]:

Part C is due: Wednesday Sept. 28 at the beginning of class.

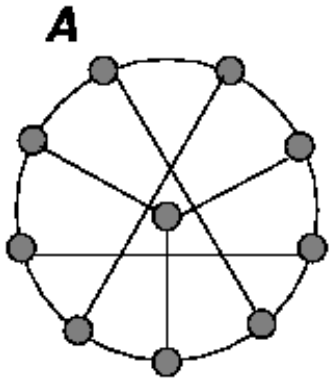
Graph Isomorphism

The graph isomorphism problem has no known polynomial time algorithm which works for an arbitrary graph.

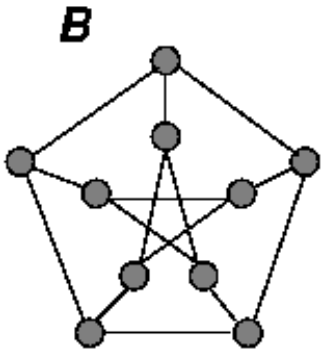
Canonical form: If two graphs are isomorphic, their canonical forms must be the same, otherwise, they must be different.

For trees and planar graphs, a canonical form can be computed in polynomial time.

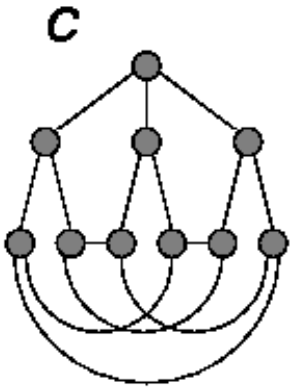
Which graphs are isomorphic to graph B?



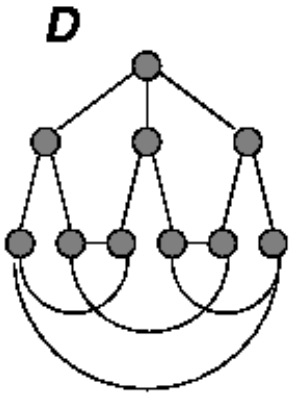
?



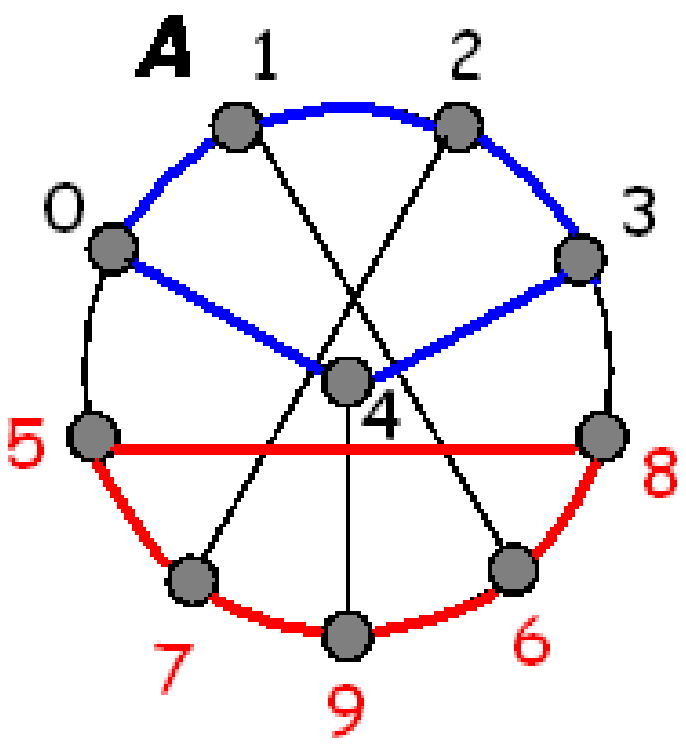
Petersen Graph

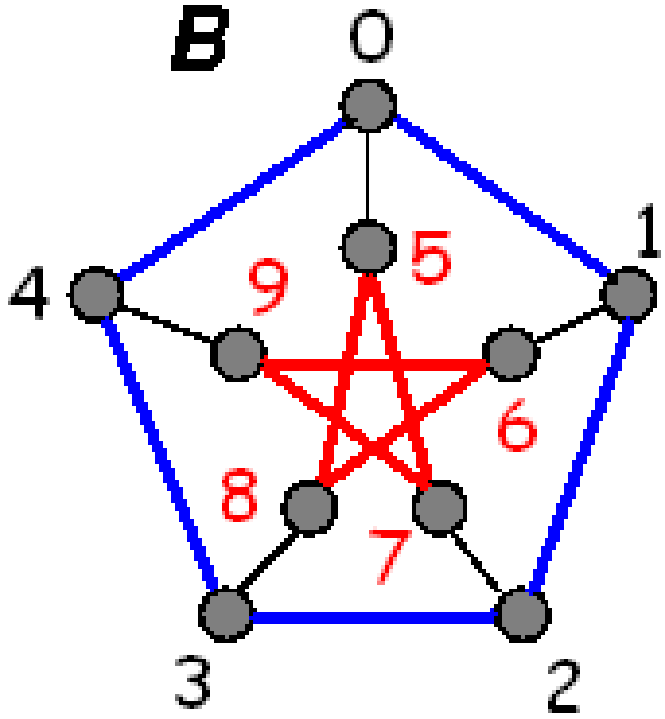


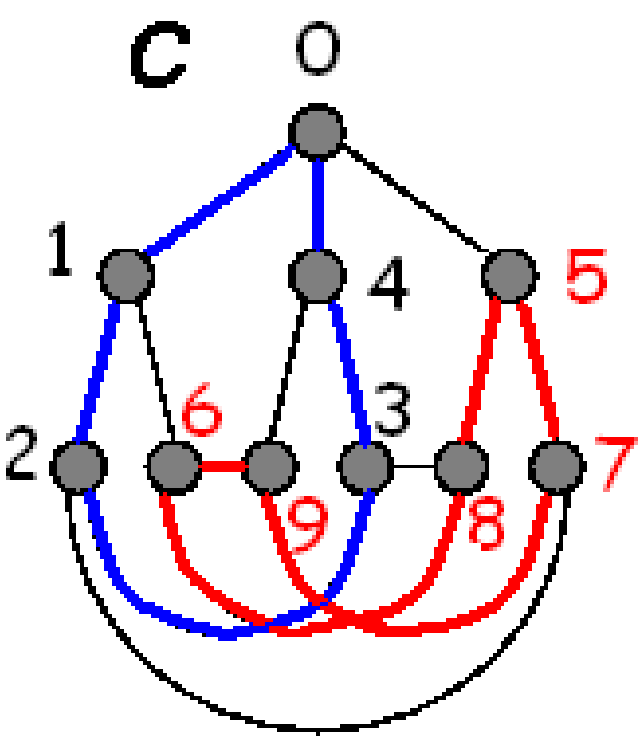
?

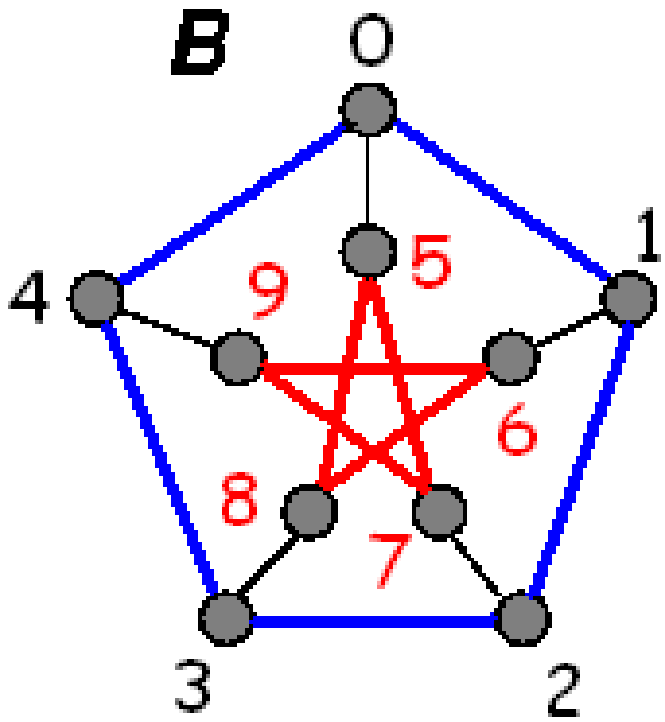


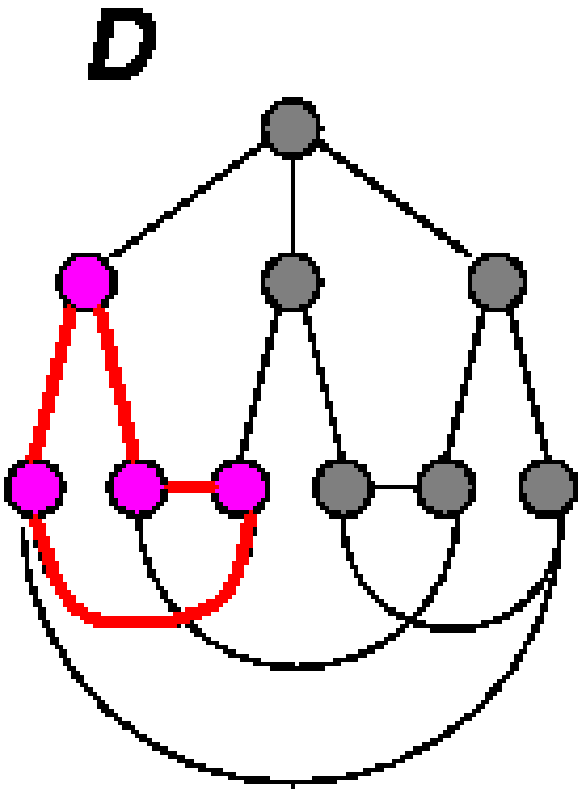
?

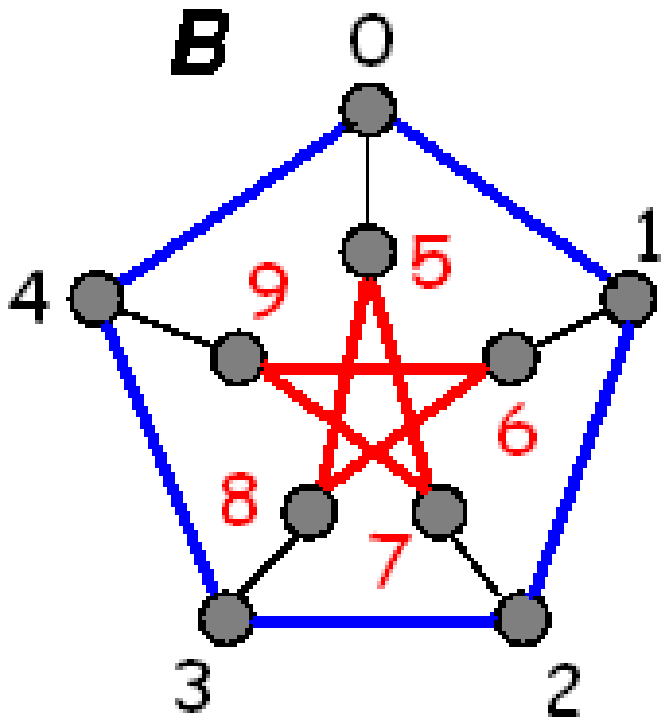

A
1
2
0
3
4
5
8
7
9
6


B
0
1
4
5
9
6
8
7
3
2


C
0
1
4
5
2
6
3
7
9
8


B
0
1
4
9
5
6
8
7
3
2


D


B
0
1
2
3
4
5
6
7
8
9

Brute force canonical form for graphs:

Consider all n! permutations of the vertices.

The canonical form is the relabelled graph that has the lexicographically minimum upper triangular adjacency matrix.

Time: $O(n! \quad n^2)$

I have implemented this approach recursively to give an example of C code that is similar to what you need for the assignment.

Ideally, your programs should include detailed comments.

You should also include error messages if the input is not correct.

You do not need an "error message" at the end of the input.

These are not included in order to show the most important details of the code (and you have me here for comments).

```
#include <stdio.h>
#include <stdlib.h>
#define NMAX 32

#define DEBUG 1

void copy(int n, int A[NMAX][NMAX],
                 int B[NMAX][NMAX]);

void print_upper(int n, int A[NMAX][NMAX]);

void get_permuted_matrix(int n, int p[NMAX],
     int A[NMAX][NMAX], int p_A[NMAX][NMAX]);

void get_can_form(int level, int n,
    int A[NMAX][NMAX], int p_A[NMAX][NMAX],
    int can_A[NMAX][NMAX], int used[NMAX],
    int p[NMAX], int verbose);
```

```
int main(int argc, char *argv[])
{
  int n;
  int A[NMAX][NMAX];
  int p_A[NMAX][NMAX];
  int can_A[NMAX][NMAX];
  int used[NMAX];
  int p[NMAX];
  int n_graph;
  int verbose;

  if (argc!= 2)
  {
      printf("Usage: %s <verbose>\n", argv[0]);
      exit(0);
   }
   verbose= atoi(argv[1]);
```

The code in red is for the command line parameter.

```
n_graph=0;
while (read_upper(&n, A))
{
    n_graph++;
    if (verbose)
    {
        printf("Input graph %3d:\n", n_graph);
        print_upper(n, A);
    }
    get_can_form(0, n, A, p_A, can_A, used,
                 p, verbose);
    if (verbose)
    {
       printf("Canonical form for graph %3d\n",
                    n_graph);
    }
    print_upper(n, can_A);
}
```

At the end of the main, I have a normal termination message. It is critical to use these and check to ensure your jobs all finish properly if you are running on multiple CPU's and/or the computations take a long time. You do not need this for the assignment.

```
 printf("Normal termination: %10d graphs\n",
                    n_graph);
}
```

```
int read_upper(int *n, int A[NMAX][NMAX])
{
   int i, j;

   if (scanf("%d", n)!=1) return(0);

   for (i=0; i < *n; i++)
   {
       A[i][i]=0;
       for (j= i+1; j < *n; j++)
       {
          if (scanf("%1d", &A[i][j])!= 1)
                    return(0);
          A[j][i]= A[i][j];
       }
   }
   return(1);
}
```

scanf reads from standard input.

```
void print_upper(int n,
                 int A[NMAX][NMAX])
{
    int i, j;

    printf("%3d ", n);
    for (i=0; i < n; i++)
        for (j=i+1; j< n; j++)
            printf("%1d", A[i][j]);
    printf("\n");
}
```

printf prints to standard output.

```
int compare(int n, int A[NMAX][NMAX],
                   int B[NMAX][NMAX])
{   int i, j;

    for (i=0; i < n; i++)
    {
        for (j=i+1; j < n; j++)
        {
            if (A[i][j] < B[i][j])
                return(-1); // A < B
            if (A[i][j] > B[i][j])
                return( 1); // A > B
        }
    }
    return(0); // A = B
}
```

```
void get_permuted_matrix(int n,
    int p[NMAX], int A[NMAX][NMAX],
                 int p_A[NMAX][NMAX])
{
    int i, j;

    for (i=0; i < n; i++)
    {
        for (j=0; j < n; j++)
        {
            p_A[p[i]][p[j]]= A[i][j];
        }
    }
}
```

```
void copy(int n, int A[NMAX][NMAX],
                 int B[NMAX][NMAX])
{
    int i, j;

    for (i=0; i < n; i++)
    {
        for (j=0; j < n; j++)
        {
            B[i][j]= A[i][j];
        }
    }
}
```

```
void get_can_form(int level,
   int n, int A[NMAX][NMAX],
   int p_A[NMAX][NMAX],
   int can_A[NMAX][NMAX],
   int used[NMAX],
   int p[NMAX], int verbose)
{
      int i;
      int cmp;
```

```
// Initial level is level 0.
// Variables are initialized here.

    if (level ==0)
    {
        for (i=0; i < n; i++)
        {
            used[i]=0;
        }
        copy(n, A, can_A);
    }
```

```
// At level n, check if the matrix
// permuted by p is smaller.

if (level == n)
{
   get_permuted_matrix(n, p, A, p_A);
   cmp= compare(n, p_A, can_A);
   if (cmp < 0)
   {
       copy(n, p_A, can_A);
   }
   return;
}
```

```
if (level == n)
{
    get_permuted_matrix(n, p, A, p_A);
#if DEBUG
    printf("The matrix permuted by ");
    for (i=0; i < n; i++)
        printf("%1d ", p[i]);
    printf(" : ");
    print_upper(n, p_A);
#endif
    cmp= compare(n, p_A, can_A);
```

It's good practice to use lots of debugging messages. Put these in a try printing every step before asking me for help with buggy programs.

```
    if (cmp < 0)
    {
#if DEBUG
        printf(
          "Smaller adjacency matrix:\n");
        printf("Before:");
        print_upper(n, can_A);
        printf("After :");
        print_upper(n, p_A);
#endif
        copy(n, p_A, can_A);
    }
    return;
}
```

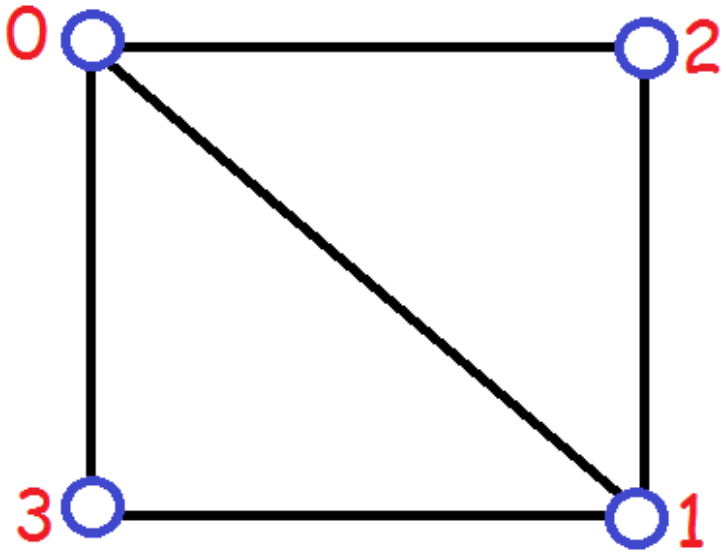
More debugging messages!

```
    for (i=0; i < n; i++)
    {
        if (! used[i])
        {
            p[level]= i;

            used[i]= 1;
            get_can_form(level+1, n, A,
                    p_A, can_A, used,
                        p, verbose);

            used[i]= 0;
         }
     }
}
```

It is good practice to include the level of recursion in debugging messages. For example:

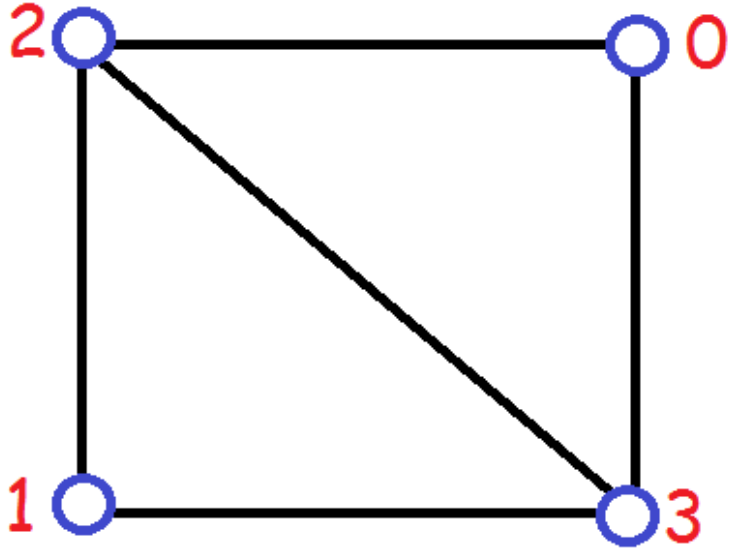
```
for (i=0; i < n; i++)
{
    if (! used[i])
    {
#if DEBUG
        printf(
          "Level %2d: Set p[%2d]=%2d]\n",
              level, level, i);
#endif
```

Graph 1:



4 1 1 1 1 1 0

Input



4 0 1 1 1 1 1

Canonical form

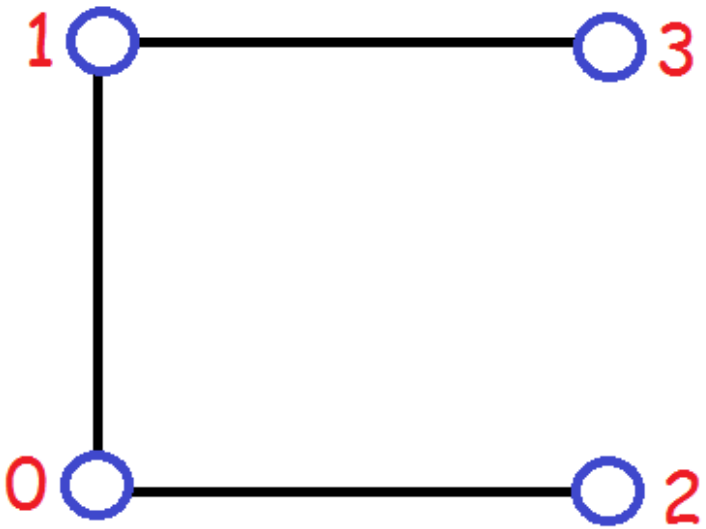
```
Input graph   1:
  4 111110
The matrix permuted by 0 1 2 3  :   4 111110
The matrix permuted by 0 1 3 2  :   4 111110
The matrix permuted by 0 2 1 3  :   4 111101
Smaller adjacency matrix:
Before:  4 111110
After :  4 111101
The matrix permuted by 0 2 3 1  :   4 111101
The matrix permuted by 0 3 1 2  :   4 111011
Smaller adjacency matrix:
Before:  4 111101
After :  4 111011
The matrix permuted by 0 3 2 1  :   4 111011
```

```
The matrix permuted by 1 0 2 3  :   4 111110
The matrix permuted by 1 0 3 2  :   4 111110
The matrix permuted by 1 2 0 3  :   4 110111
Smaller adjacency matrix:
Before:  4 111011
After :  4 110111
The matrix permuted by 1 2 3 0  :   4 110111
The matrix permuted by 1 3 0 2  :   4 101111
Smaller adjacency matrix:
Before:  4 110111
After :  4 101111
The matrix permuted by 1 3 2 0  :   4 101111
```

```
The matrix permuted by 2 0 1 3  :   4 111101
The matrix permuted by 2 0 3 1  :   4 111101
The matrix permuted by 2 1 0 3  :   4 110111
The matrix permuted by 2 1 3 0  :   4 110111
The matrix permuted by 2 3 0 1  :   4 011111
Smaller adjacency matrix:
Before:  4 101111
After :  4 011111
The matrix permuted by 2 3 1 0  :   4 011111
```

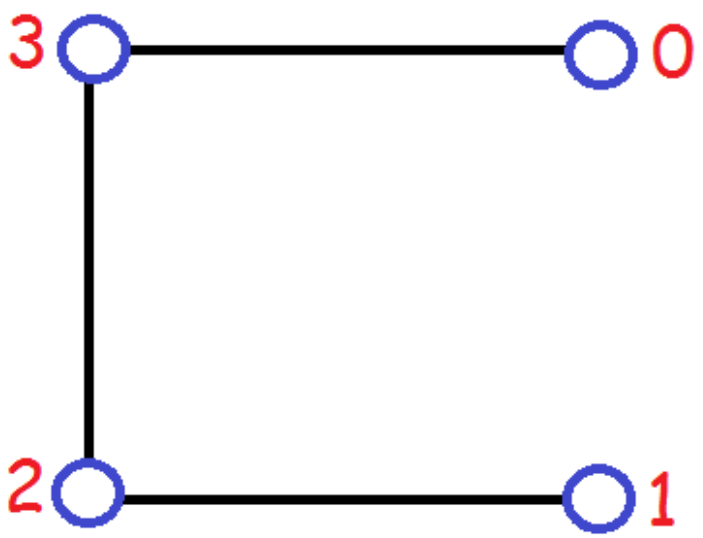
```
The matrix permuted by 3 0 1 2  :    4 111011
The matrix permuted by 3 0 2 1  :    4 111011
The matrix permuted by 3 1 0 2  :    4 101111
The matrix permuted by 3 1 2 0  :    4 101111
The matrix permuted by 3 2 0 1  :    4 011111
The matrix permuted by 3 2 1 0  :    4 011111
Canonical form for graph    1
  4 011111
```

Graph 2:



4 1 1 0 0 1 0

Input



4 0 0 1 1 0 1

Canonical form

```
Input graph   2:
  4 110010
The matrix permuted by 0 1 2 3  :   4 110010
The matrix permuted by 0 1 3 2  :   4 101100
Smaller adjacency matrix:
Before:  4 110010
After :  4 101100
The matrix permuted by 0 2 1 3  :   4 110001
The matrix permuted by 0 2 3 1  :   4 011100
Smaller adjacency matrix:
Before:  4 101100
After :  4 011100
The matrix permuted by 0 3 1 2  :   4 101001
The matrix permuted by 0 3 2 1  :   4 011010
Smaller adjacency matrix:
Before:  4 011100
After :  4 011010
```

```
The matrix permuted by 1 0 2 3  :   4 101100
The matrix permuted by 1 0 3 2  :   4 110010
The matrix permuted by 1 2 0 3  :   4 100101
The matrix permuted by 1 2 3 0  :   4 010110
Smaller adjacency matrix:
Before:  4 011010
After :  4 010110
The matrix permuted by 1 3 0 2  :   4 100011
The matrix permuted by 1 3 2 0  :   4 001110
Smaller adjacency matrix:
Before:  4 010110
After :  4 001110
```

```
The matrix permuted by 2 0 1 3  :   4 011100
The matrix permuted by 2 0 3 1  :   4 110001
The matrix permuted by 2 1 0 3  :   4 010110
The matrix permuted by 2 1 3 0  :   4 100101
The matrix permuted by 2 3 0 1  :   4 010011
The matrix permuted by 2 3 1 0  :   4 001101
Smaller adjacency matrix:
Before:  4 001110
After :  4 001101
```

```
The matrix permuted by 3 0 1 2  :    4 011010
The matrix permuted by 3 0 2 1  :    4 101001
The matrix permuted by 3 1 0 2  :    4 001110
The matrix permuted by 3 1 2 0  :    4 100011
The matrix permuted by 3 2 0 1  :    4 001101
The matrix permuted by 3 2 1 0  :    4 010011
Canonical form for graph   2
  4 001101
Normal termination:          2 graphs
```

With verbose set to 0 and
#define DEBUG 0

Input:
4 111110
4 110010

Output:
  4 011111
  4 001101
Normal termination: 2 graphs

With verbose set to 1 and
#define DEBUG 0

Input:
4 111110
4 110010

Output:
Input graph   1:
  4 111110
Canonical form for graph   1
  4 011111
Input graph   2:
  4 110010
Canonical form for graph   2
  4 001101
Normal termination: 2 graphs