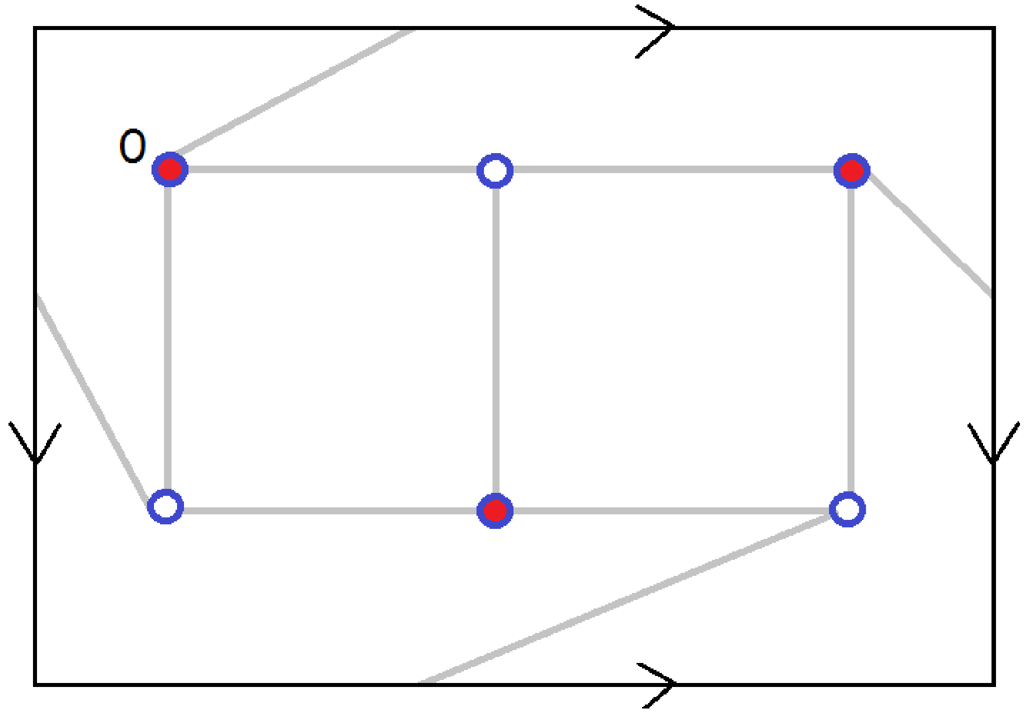
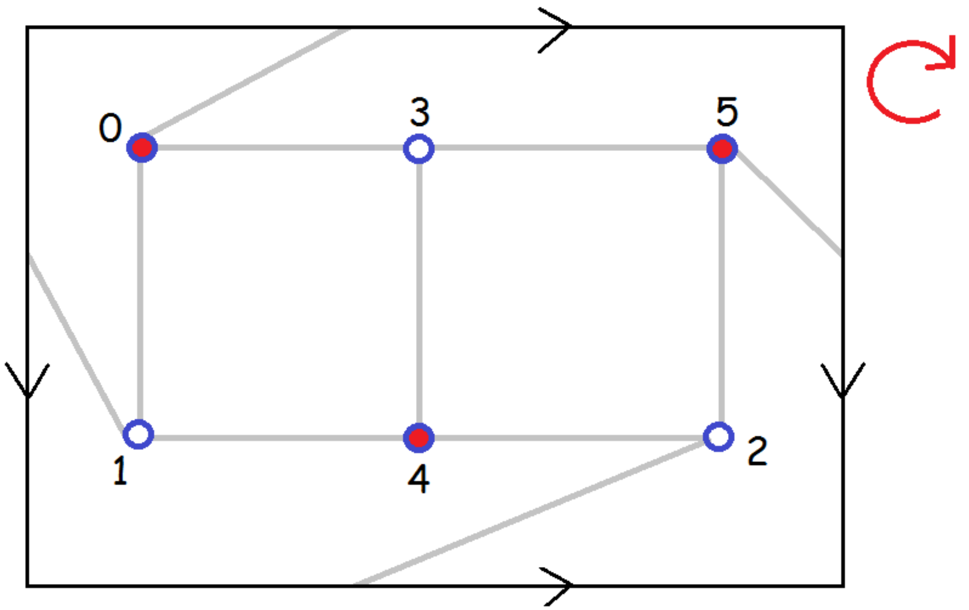
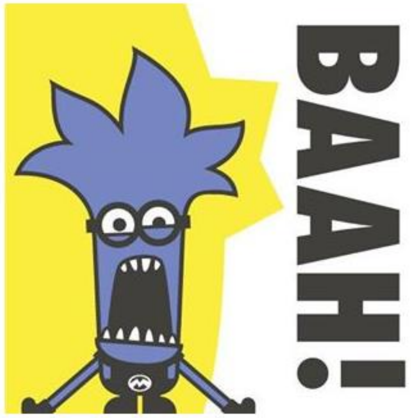
Do clockwise BFS on this graph using vertex 0 as the root, each choice of first child and each direction. Which case results in the lexicographically smallest adjacency list?



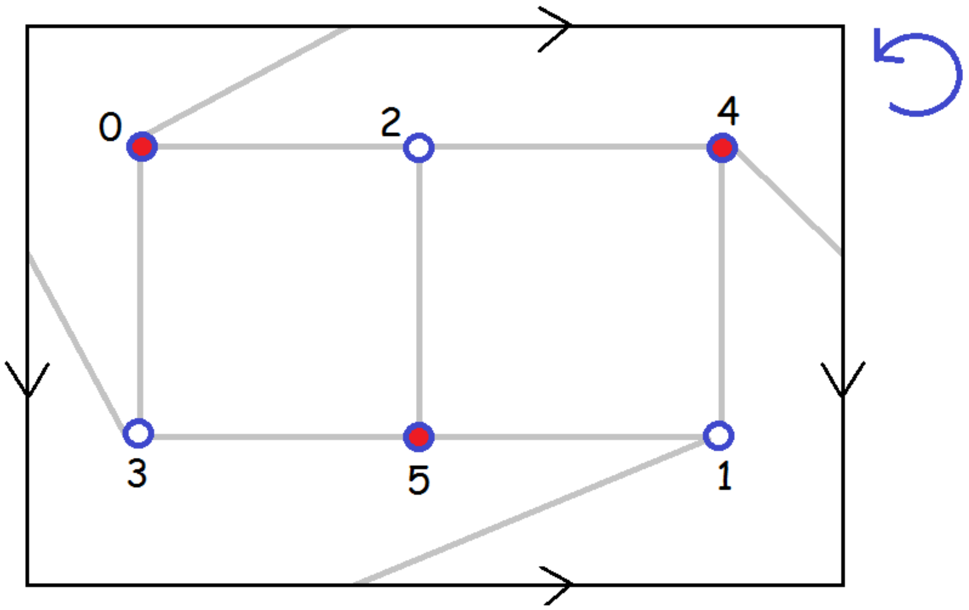
Then smallest ones in each direction:



0:	1 2 3
1:	0 4 5
2:	0 4 5
3:	0 5 4
4:	1 3 2
5:	1 2 3



These are different!



0:	1 2 3
1:	0 4 5
2:	0 4 5
3:	0 5 4
4:	1 2 3
5:	1 3 2

Assignment 3, Question 2 had an error in my assertions:

2. (d) Argue in terms of the embedding symmetries why it is that we do not have to consider other choices for r, f, or the direction in order to ensure the lexicographic minimum has been found.

We used one choice of vertex per orbit for r, each possible first child and direction clockwise. The assignment problem is a counterexample to this being sufficient.
You need to consider both directions (but only one vertex per orbit is sufficient).

Grading scheme:
If you followed instructions: full marks.
If you realized I made an error- bonus marks.

Give the binary values for a, b, c, d, e, f, g, h.

```
int a, b, c, d, e, f, g, h;

a= 07205;
b= 05314;
c= a & b;
d= a | b;
e= ~a;

f= a && b;
g= a || b;
h= !a;
```

Give the binary values for a, b, c, d, e, f, g, h.

```
a= 07205;
b= 05314;
c= a & b;
d= a | b;
e= ~a;

f= a && b;
g= a || b;
h= !a;
```

a = 00 000 000 000 000 000 000 111 010 000 101

b = 00 000 000 000 000 000 000 101 011 001 100

c = 00 000 000 000 000 000 000 101 010 000 100

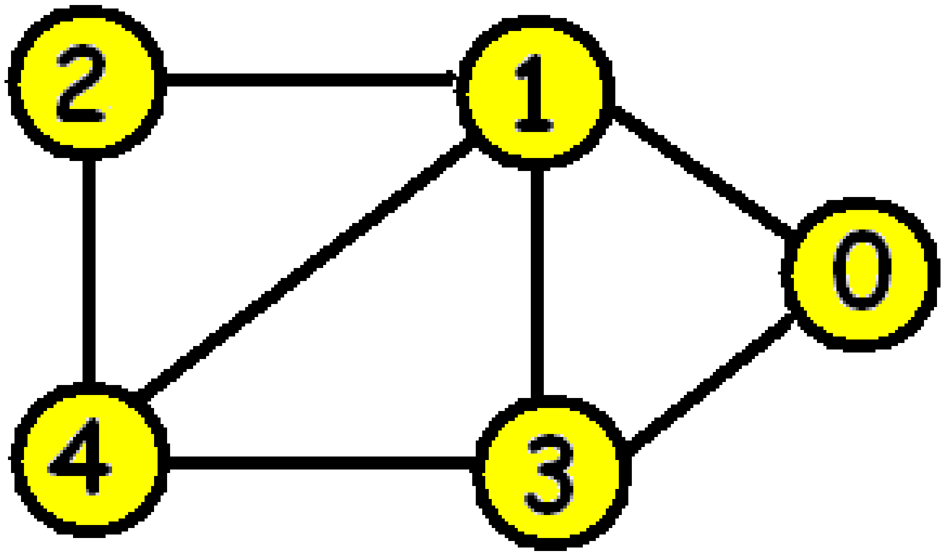
d = 00 000 000 000 000 000 000 111 011 001 101

e = 11 111 111 111 111 111 111 000 101 111 010

f = 00 000 000 000 000 000 000 000 000 000 001

g = 00 000 000 000 000 000 000 000 000 000 001

h = 00 000 000 000 000 000 000 000 000 000 000



Adjacency matrix:

	0	1	2	3	4
0	0	1	0	1	0
1	1	0	1	1	1
2	0	1	0	0	1
3	1	1	0	0	1
4	0	1	1	1	0

Uncompressed adjacency matrix in C or C++:

```
#define NMAX 128

int G[NMAX][NMAX];
```

Compressed Adjacency Matrices

A compressed adjacency matrix for a graph will use 1/32 the space of an adjacency matrix (assuming 32 bit integers are used).

Bit twiddling to implement it can result in some operations for algorithms such as dominating set and clique being about 32 times faster.

Compressed adjacency matrix

G[0][0]=

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

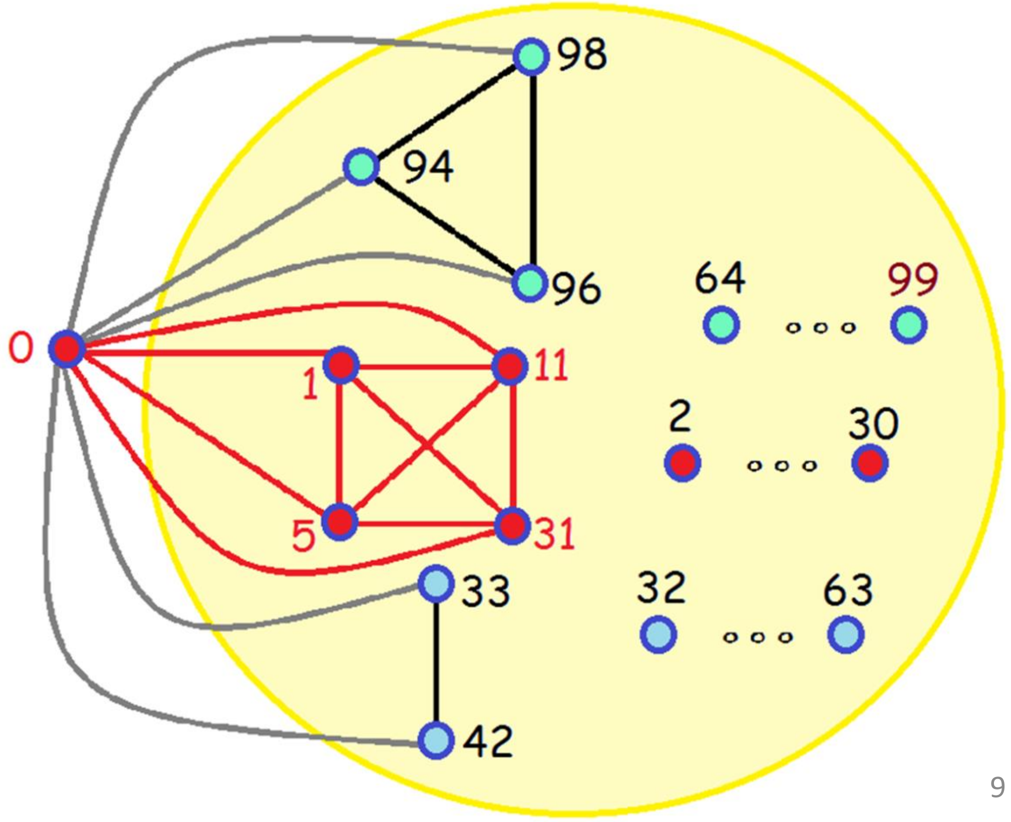
$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 2 & 0 & 1 & 0 & 0 & 1 \\ 3 & 1 & 1 & 0 & 0 & 1 \\ 4 & 0 & 1 & 1 & 1 & 0 \end{array}$$

```
#define NMAX 100
#define MMAX (NMAX+31)/32
// MMAX = ⌈NMAX/32⌉
int G[NMAX][MMAX];
```

Pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0

Pos	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

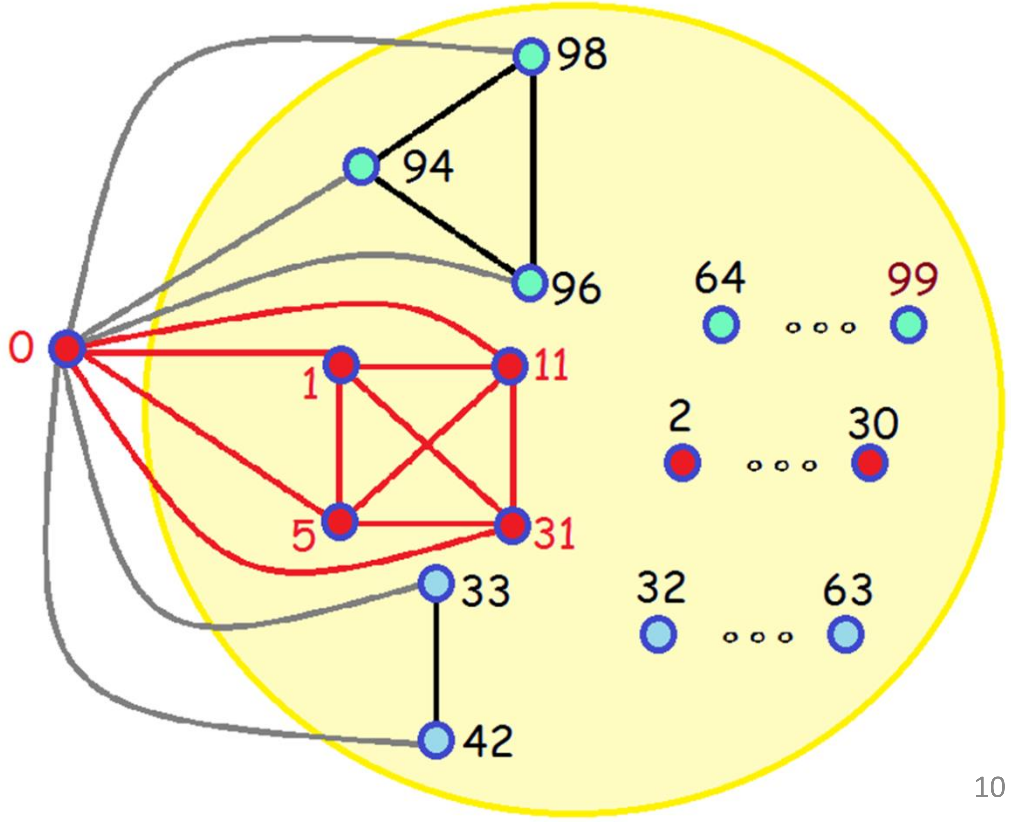
G[0][0]



Pos	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Pos	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

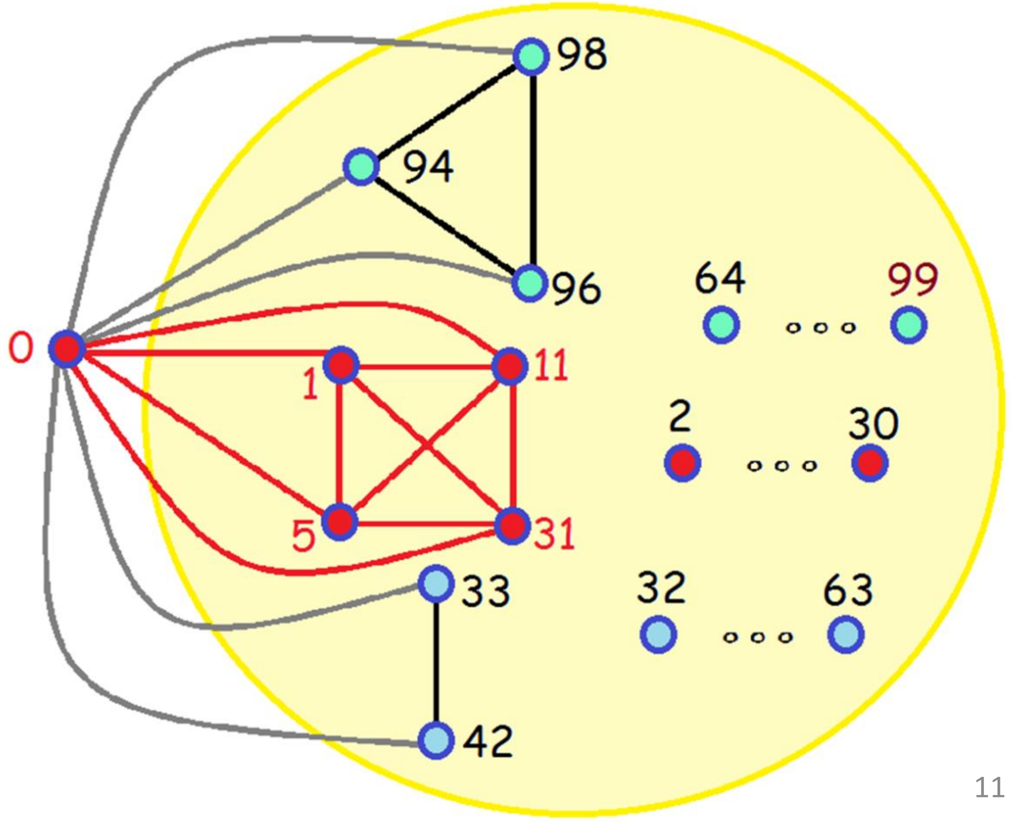
G[0][1]



Pos	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Pos	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

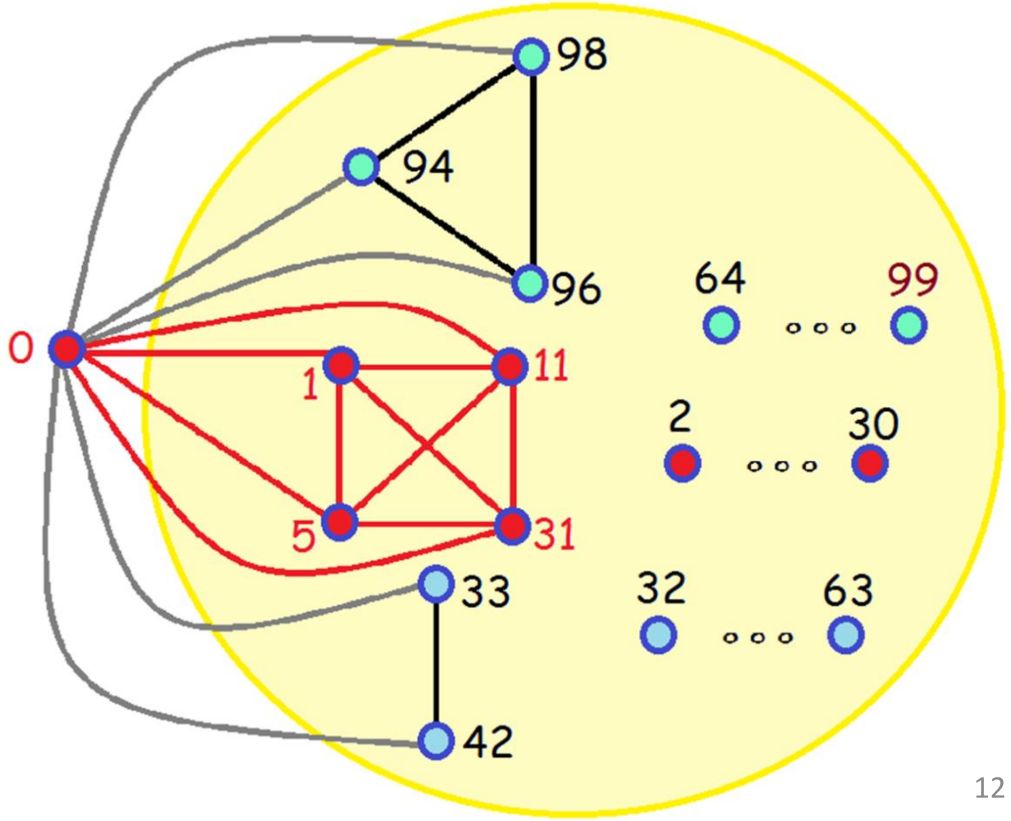
G[0][2]



Pos	96	97	98	99												
	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Pos																
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

G[0][3]



```
int bit[] = {
             020000000000,010000000000,
 04000000000, 02000000000, 01000000000,
  0400000000,  0200000000,  0100000000,
   040000000,   020000000,   010000000,
    04000000,    02000000,    01000000,
     0400000,     0200000,     0100000,
      040000,      020000,      010000,
       04000,       02000,       01000,
        0400,        0200,        0100,
         040,         020,         010,
          04,          02,          01};
```

Putting a 0 in front of a number means the value is represented base 8.

```
// pos/32 is word number.

#define SETWD(pos) ((pos)>>5

// pos % 32 is bit number.

#define SETBT(pos) ((pos)&037)

// This sets bit number pos to 1.
// Bits are numbered 0, 1, … , n-1.

#define ADD_ELEMENT(setadd,pos)
    ((setadd)[SETWD(pos)] |=
                     bit[SETBT(pos)])
```

```
// This sets bit number pos to 0.

#define DEL_ELEMENT(setadd,pos)
       ((setadd)[SETWD(pos)] &=
                    ~bit[SETBT(pos)])

// Tests if bit number pos is 1.

#define IS_ELEMENT(setadd,pos)
      ((setadd)[SETWD(pos)] &
                bit[SETBT(pos)])
```

```
#define NMAX 128
#define MMAX ((NMAX +31)/ 32)

int read_graph(int *n, int *m,
               int G[NMAX][MMAX])
{

    int i, j, u, d;

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

    *m= (*n + 31)/32; // $m=\left\lceil \frac{n}{32} \right\rceil$
```

```
// Initialize the graph to
// have no edges.

    for (i=0; i < *n; i++)
    {
        for (j=0; j < *m; j++)
        {
            G[i][j]= 0;
        }
    }
```

```
// Read in the adjacency list info.

for (i=0; i < *n; i++)
{
   if (scanf("%d", &d)!=1) return(0);
   for (j=0; j < d; j++)
   {
         if (scanf("%d", &u)!=1)
               return(0);
         // Add both (i, u) and (u,i).
         ADD_ELEMENT(G[i], u);
         ADD_ELEMENT(G[u], i);
    }
}
return(1);
```

To count the number of 1 bits in a 32-bit integer:

```
#define POP_COUNT(x)
        bytecount[(x)>>24 & 0377] +
        bytecount[(x)>>16 & 0377] +
        bytecount[(x)>> 8 & 0377] +
        bytecount[(x)     & 0377]
```

```
int bytecount[] =
          {0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4,
           1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,
           1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,
           2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
           1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,
           2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
           2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
           3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
           1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,
           2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
           2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
           3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
           2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
           3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
           3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
           4,5,5,6,5,6,6,7,5,6,6,7,6,7,7,8};
```

```
// Find a set size.

int set_size(int n, int set[])
{
    int j, m, d;

    m= (n+31)/ 32;

    d=0;
    for (j=0; j < m; j++)
    {
        d+= POP_COUNT(set[j]);
    }
    return(d);
}
```

```
// Prints a set.

void print_set(int n, int set[])
{
   int i;

   for (i=0; i < n; i++)
      if (IS_ELEMENT(set, i))
         printf("%3d", i);
   printf("\n");
}
```

```
// Prints a graph.
void print_graph(int n,
                 int G[NMAX][MMAX])
{
    int i, j, d, m;


    for (i=0; i < n; i++)
    {
        d= set_size(n, G[i]);
        printf("%3d(%1d):", i, d);
        print_set(n, G[i]);
    }
}
```

```
// Read graphs in and print them.

main(int argc, char *argv[])
{
   int n,m;
   int G[NMAX][MMAX];


   while (read_graph(&n, &m, G))
   {
      printf("The input graph:\n");
      print_graph(n, G);
   }

}
```

```
The input file contains:

4
2 1 2
3 0 2 3
3 0 1 3
2 1 2

The input graph:
  0(2):  1  2
  1(3):  0  2  3
  2(3):  0  1  3
  3(2):  1  2
```

Templates for design of clique and dominating set routines:

```
int vertex_set[MMAX];
int answer[MMAX];
int k=5;// arbitrary parameter choice.

while (read_graph(&n, &m, G))
{
  find_clique(0, k, n, m, G,
                vertex_set, answer);

  find_dom_set(0, k, n, m, G,
             vertex_set, answer);
}
```

```
#define DEBUG 1
int find_clique(
   int level, int k,
   int n, int m, int G[NMAX][MMAX],
   int candidates[MMAX],
   int clique[MMAX])
{
    int new_candidates[MMAX];
    int i, j, u;
if (level == 0)
{
    // Initialize candidates and
    // clique to have no vertices.
    for (j=0; j < m; j++)
    {
        candidates[j]=0;
        clique[j]=0;
     }
     // Every vertex is a candidate.
     for (i=0; i < n; i++)
          ADD_ELEMENT(candidates, i);
     }
}
```

```
// When k is 0 we have found a
// clique of the desired order.

if (k == 0)
{
    printf("Clique: ");
    print_set(n, clique);
    return(1);
}

// No candidates remaining.
if (set_size(n, candidates) == 0)
        return(0);
```

Print often to make sure your code is correct. Including the level will help with recursive function debugging.

```
#if DEBUG

printf(
    "Level %3d: The candidates are:",
                                             level);
print_set(n, candidates);

#endif
```

I will leave the remaining logic for a clique routine up to you to discover.

The next snippets show how to add a vertex u to the clique currently being constructed.

```
ADD_ELEMENT(clique, u);
for (j=0; j < m; j++)
{
   new_candidates[j]= candidates[j]
                          & G[u][j];
}
if (find_clique(level+1, k-1,
    n, m, G, new_candidates, clique))
          return(1);

//  Take u out again.

DEL_ELEMENT(clique, u);
```

For dominating set:

```
if (level == 0)
{   // Set diagonal entries to 1.
    for (i=0; i < n; i++)
        ADD_ELEMENT(G[i], i);

// Initialize dominated and
// dom_set to have no vertices.

    for (j=0; j < m; j++)
    {
        dominated[j]=0;
        dom_set[j]=0;
    }
}
```

```
// Try adding u to dominating set.
ADD_ELEMENT(dom_set, u);
for (j=0; j < m; j++)
{
    new_dominated[j]= dominated[j]
                         | G[u][j];
}
if (find_dom_set(level+1, k-1,
    n, m, G, new_dominated, dom_set))
       return(1);

//  Take u out again.

DEL_ELEMENT(dom_set, u);
```