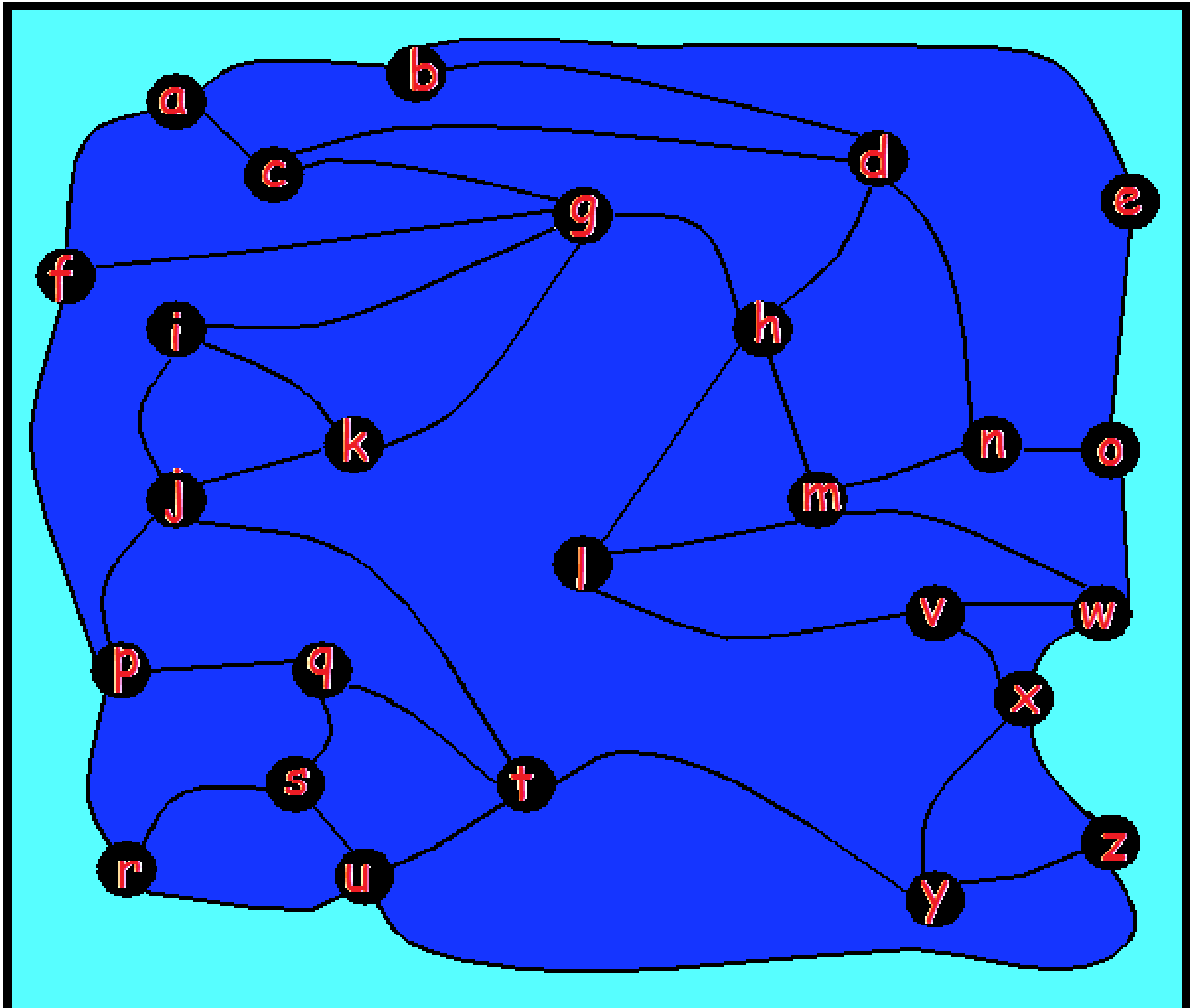
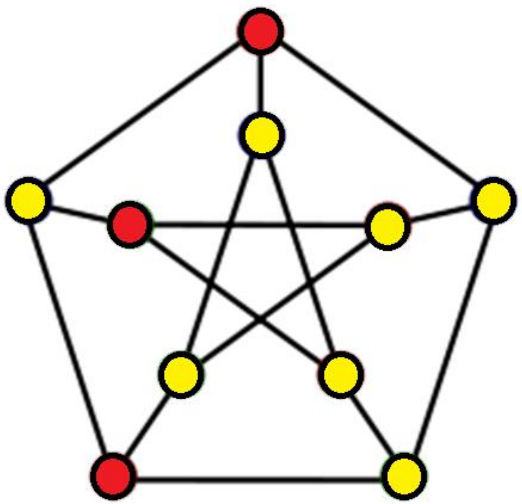
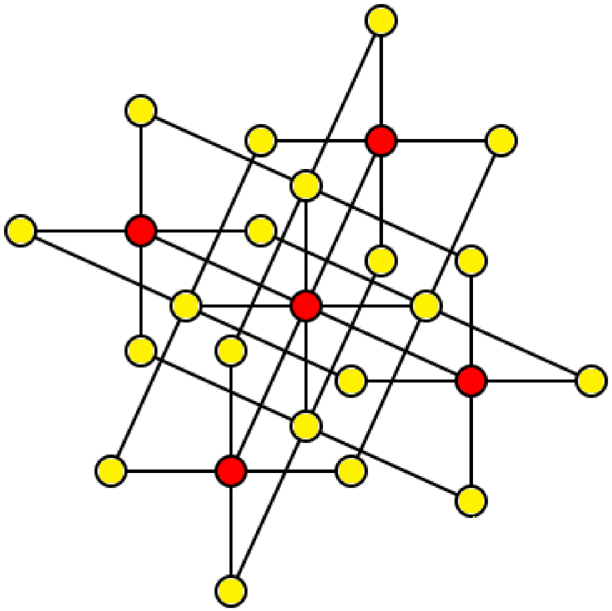


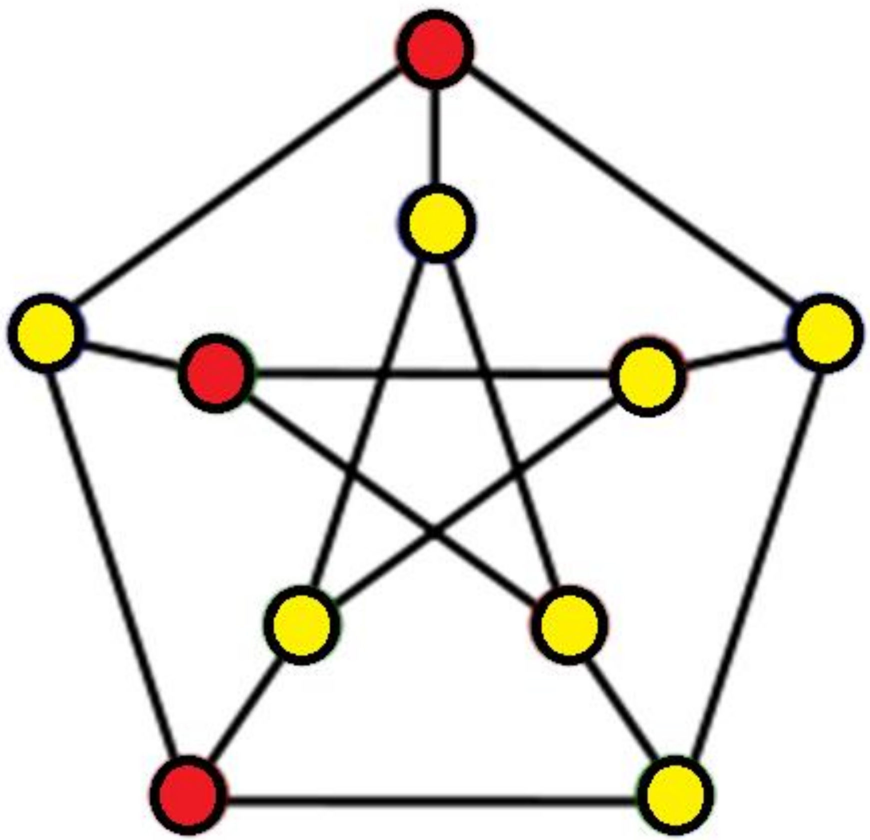
A Map of the Town of Iceberg

A Simple Algorithm for Dominating Set



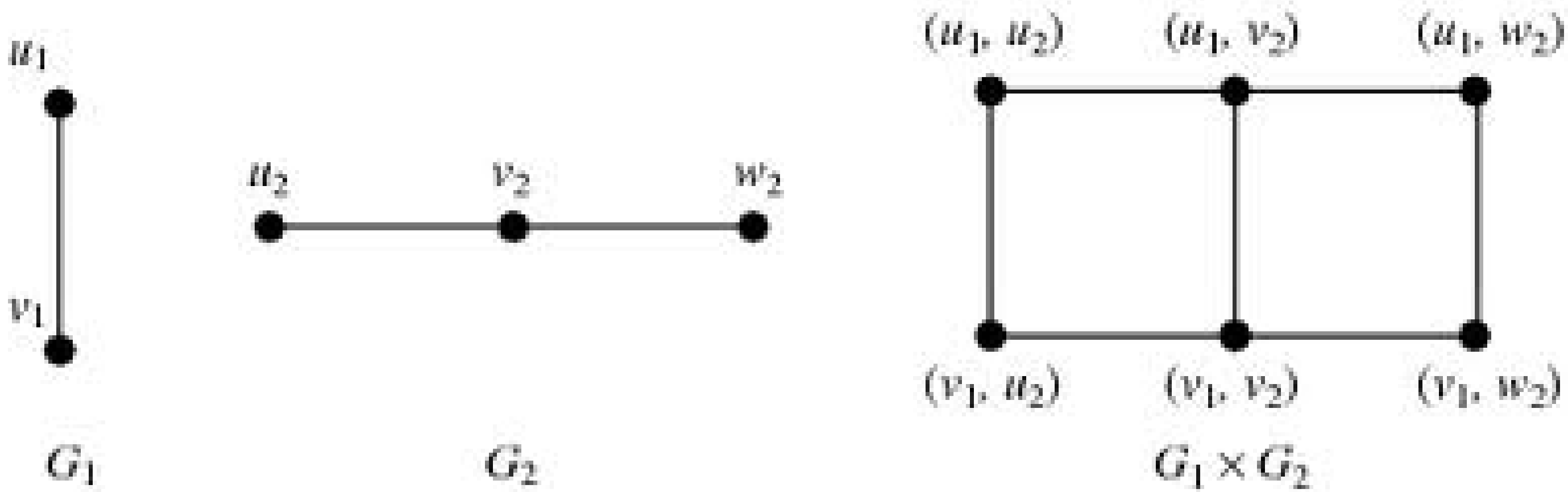


A *dominating set* of a graph G is a subset D of the vertices of G such that every vertex v of G is either in the set D or v has at least one neighbour that is in D.

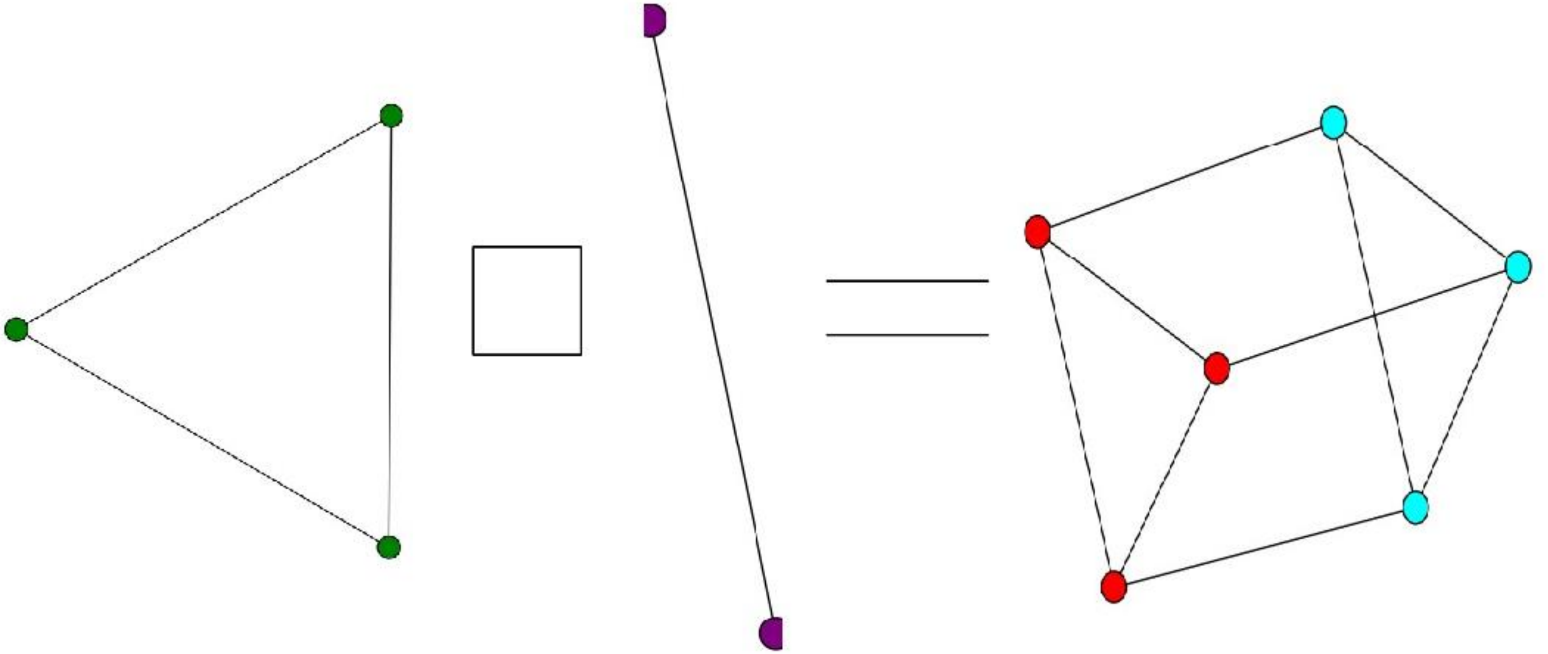


The Cartesian product, $G \square H$, of graphs G and H is a graph F such that

1. V(F) = V(G) × V(H); and
2. any two vertices (u,u') and (v,v') are adjacent in F if and only if either:
   u = v and u' is adjacent with v' in H, or
   u' = v' and u is adjacent with v in G.

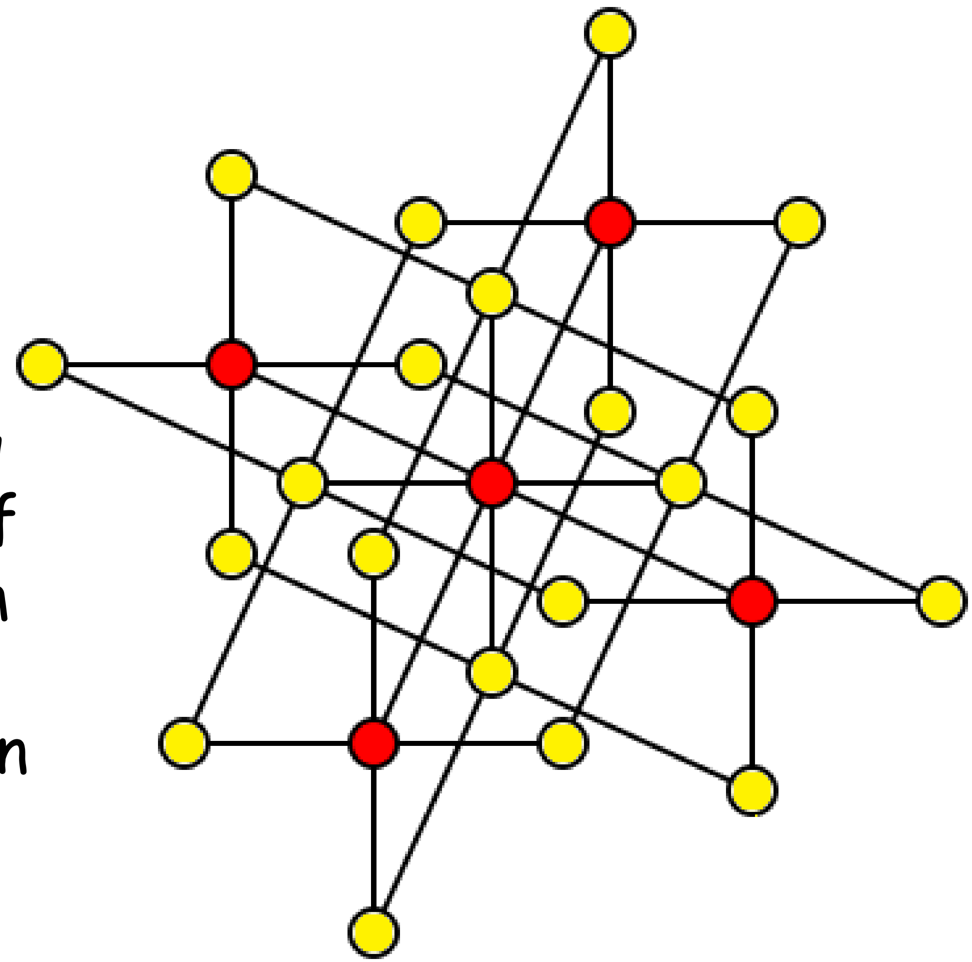


http://mathworld.wolfram.com/GraphCartesianProduct.html



http://cnx.org/content/m34835/latest/?collection=col10523/latest

Vizing's conjecture concerns a relation between the domination number and the cartesian product of graphs. This conjecture was first stated by Vadim G. Vizing (1968), and states that, if γ(G) denotes the minimum number of vertices in a dominating set for G, then γ(G H) ≥ γ(G)γ(H).

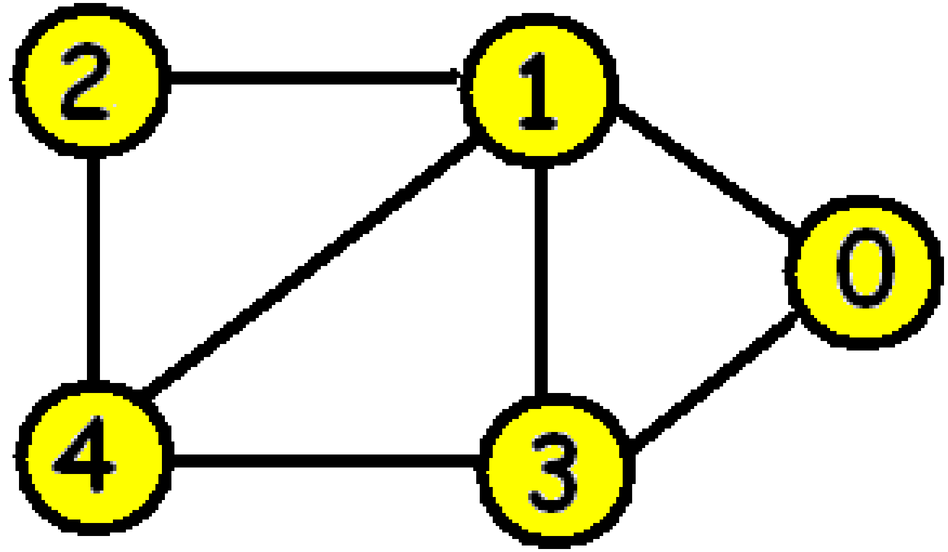


Conjecure predicts ≥ 1 for this graph so it is not tight.

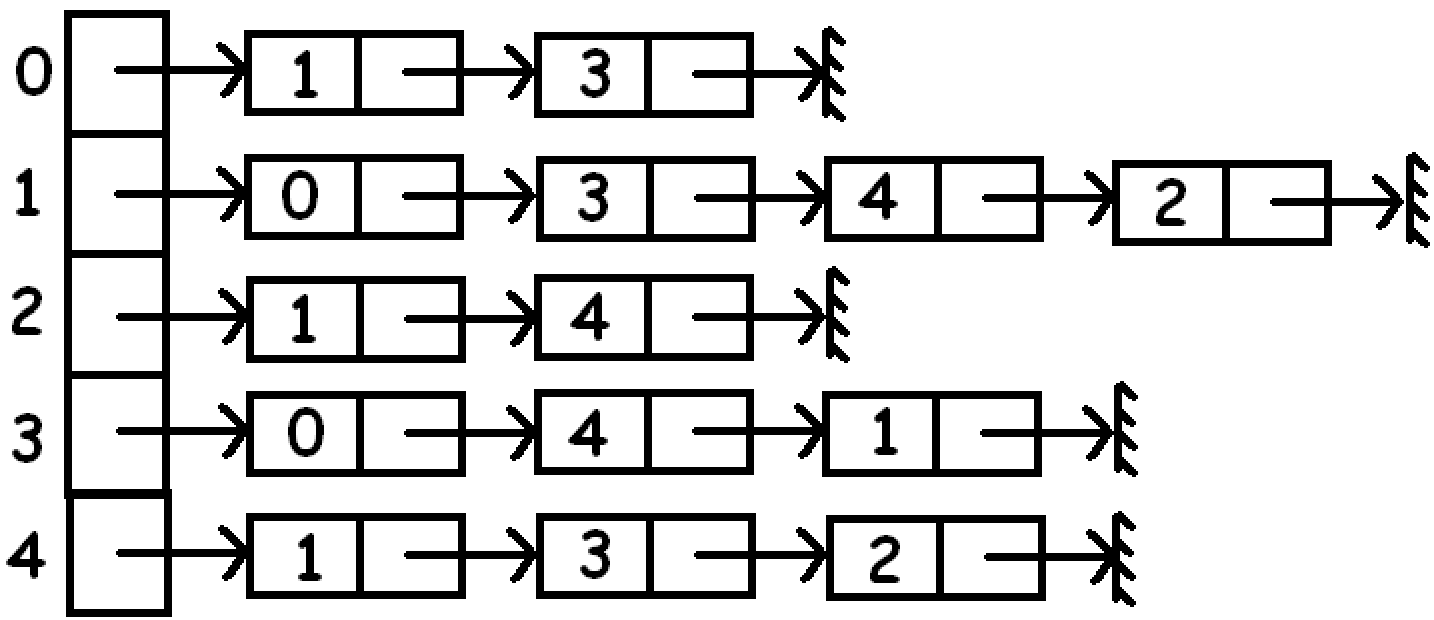
http://en.wikipedia.org/wiki/Vizing's_conjecture

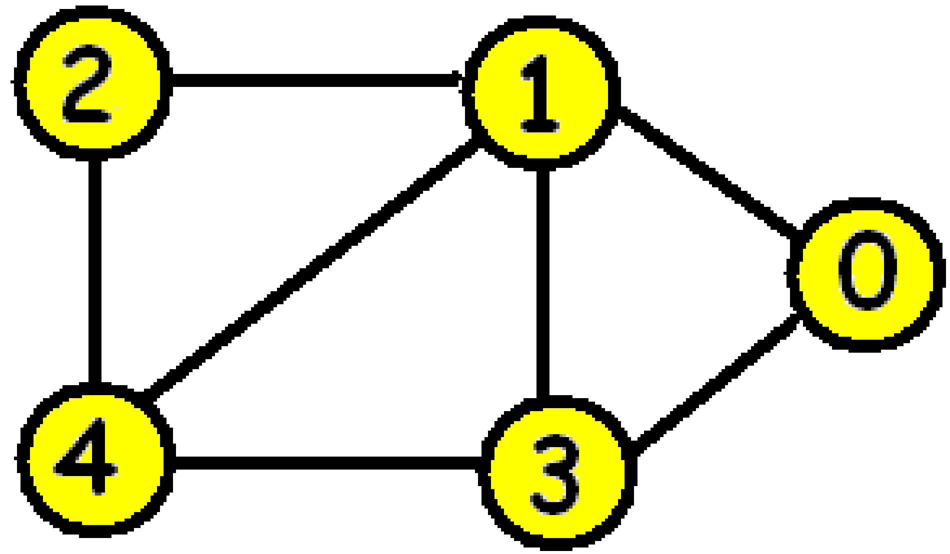
A recent survey paper:

Brešar, Boštjan; Dorbec, Paul; Goddard, Wayne; Hartnell, Bert L.; Henning, Michael A.; Klavžar, Sandi; Rall, Douglas F. Vizing's conjecture: a survey and recent results. J. Graph Theory 69 (2012), no. 1, 46-76.



Adjacency list:





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

Input as:

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

A simple but reasonable fast dominating set algorithm (you can implement this for milestone 1):

Data structures:

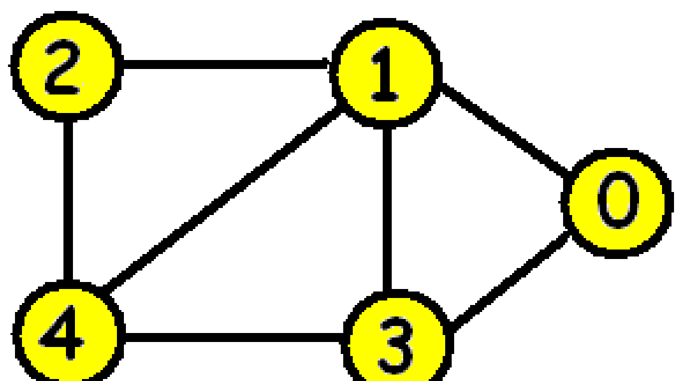
The graph:

n = number of vertices

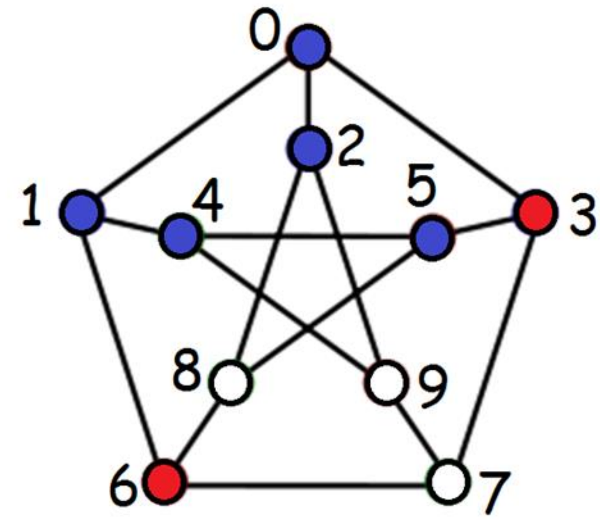
A[0..(n-1)][0..(n-1)] = adjacency matrix, but I changed the diagonal so that the values are all 1's (because a vertex dominates itself).

DELTA= maximum degree of a vertex v

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



Each vertex has a status:



white: not decided

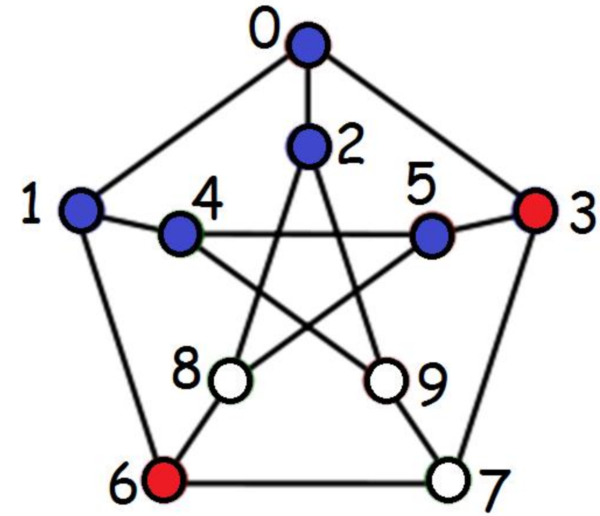
blue: excluded from dominating set

red: included in dominating set

I did not actually record these explicitly although it could be useful in algorithm variants.

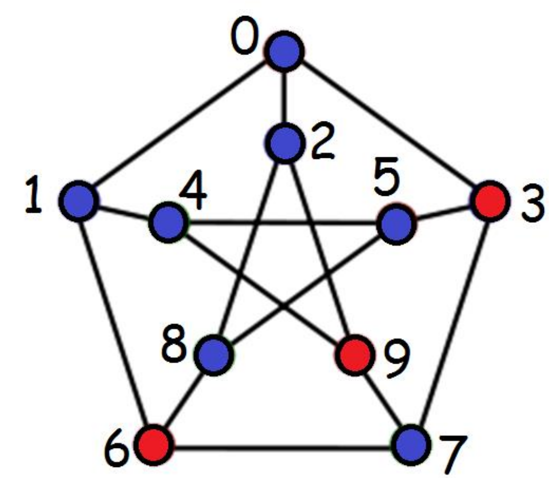
To record a partial dominating set:

size
dom[0..(n-1)]

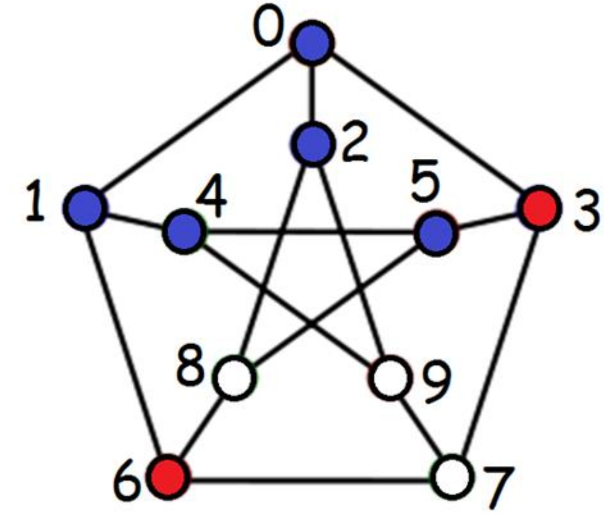


or the minimum dominating set found so far:

min_size
min_dom[0..(n-1)]



n_dominated= number of dominated vertices.



For each vertex v:

num_dominated[v]= number of times it is dominated by a red vertex.

num_choice[v]= number of times it could be dominated if all white vertices were red ones. If num_choice[v] is 0 for some vertex, we can back up (solution cannot be completed to a dominating set).

When programming recursive algorithms, it helps in debugging to have a variable level representing the level of recursion.

Initial call:

```
min_dom_set(0, …
```

Declaration of function:

```
int min_dom_set(int level, …
```

Recursive calls:

```
min_dom_set(level+1, …
```

At the top:

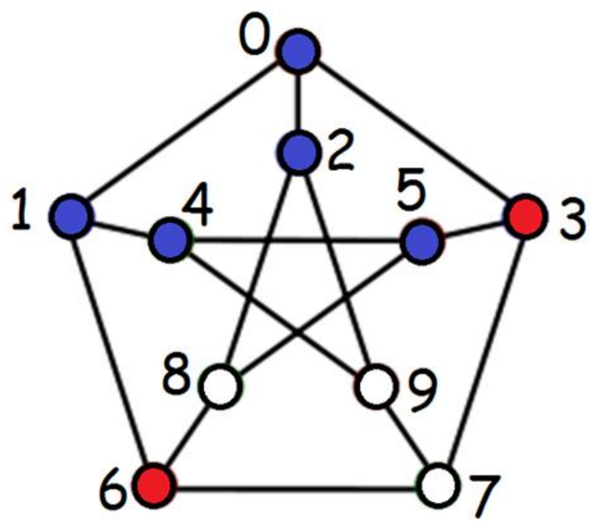
```
#include <stdio.h>

#include <stdlib.h>

#define NMAX 500

#define DEBUG 1
```

Debugging:

```
#if DEBUG
   printf("Level %3d: ", level);
   print_vector(size, dom);
   printf("Number of vertices dominated: %3d\n",
              n_dominated);
   printf("Number of choices per vertex:\n");
   print_vector(n, num_choice);
   printf("Number of times dominated:\n");
   print_vector(n, num_dominated);
#endif
```

At a given level, I decide the status of vertex number *level*.

I first try making it blue and then red. Before returning: change it back to white.



At level 0 we initialize the data structures first:

Implicit: all vertices are white.
n_dominated=0
num_choice[v]= degree of v + 1
num_dominated[v]= 0

size=0
dom[i]= no values assigned since size is 0
min_size= n
min_dom[i]= i for i= 0 to n-1

Tests used to check if we should backtrack (not completable to a dominating set smaller than the min so far).

If for any vertex v, num_choice[v] is 0 then return.

Set n_extra= $\left\lceil \frac{u}{\Delta+1} \right\rceil$
u= number of undominated vertices
Δ = maximum degree of a vertex
If size + n_extra ≥ min then return.

Termination condition:

At level n (all vertices v have a status, and num_choice[v] is at least 1 so dominated) or if all vertices are dominated:

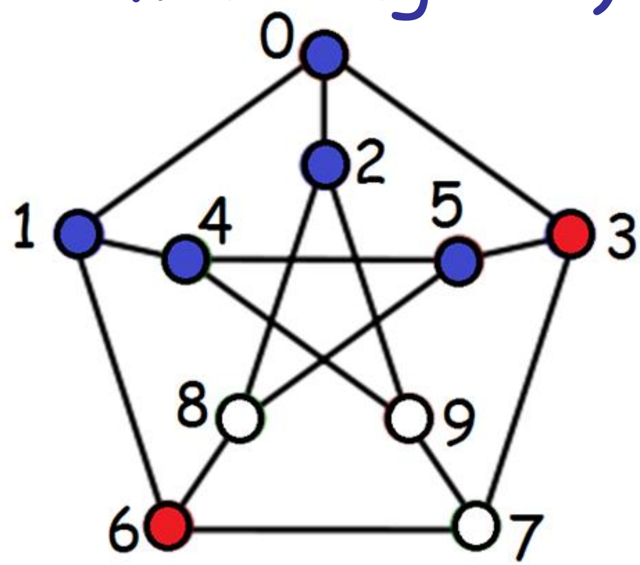
```
If size < min_size
      copy the current dominating
      set dom to min_dom
      and set min_size= size.
End if

return
```

The exhaustive backtrack:

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

Set u= level.

Try vertex u as blue (excluded from dominating set):



For each neighbour v of u as recorded in *A*, decrement num_choice[v].

Call the routine recursively.

Recursive routines should restore data structures to avoid need to copy them.

For each neighbour v of u as recorded in *A*, increment num_choice[v].

Try vertex u as red (in dominating set):

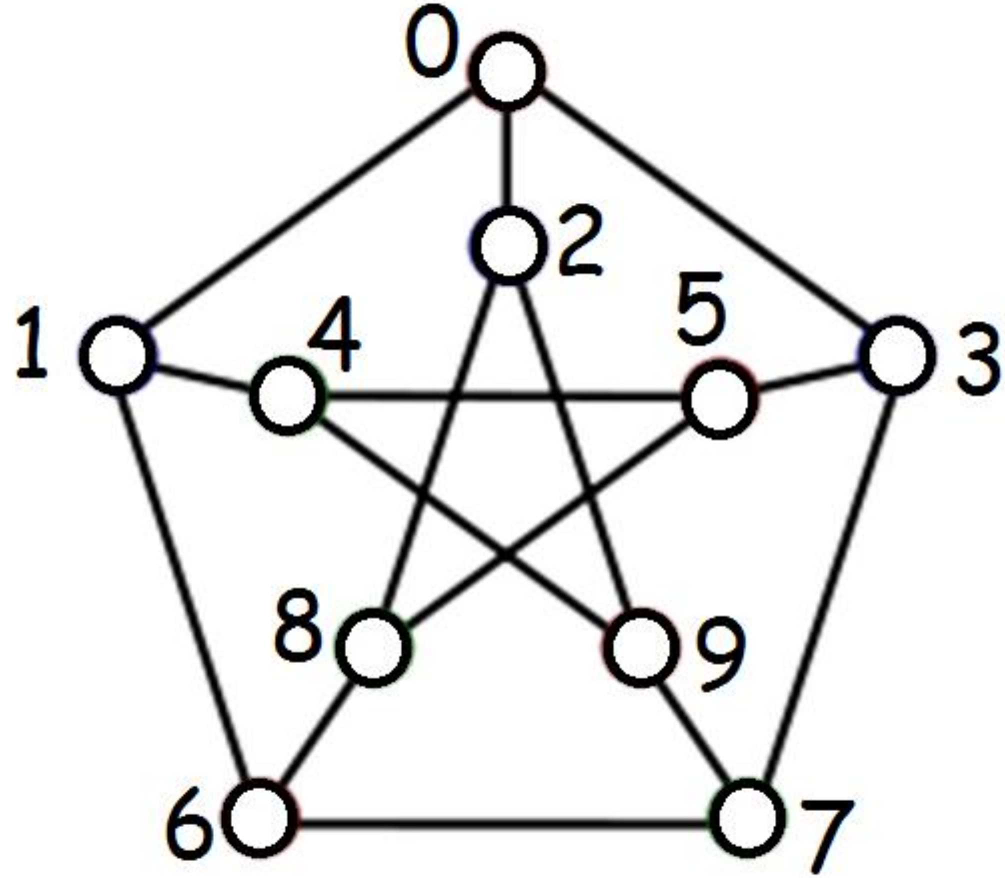
Add u to dom.
For each neighbour v of u as recorded in A, increment num_dominated[v].
Update n_dominated.

Call the routine recursively.

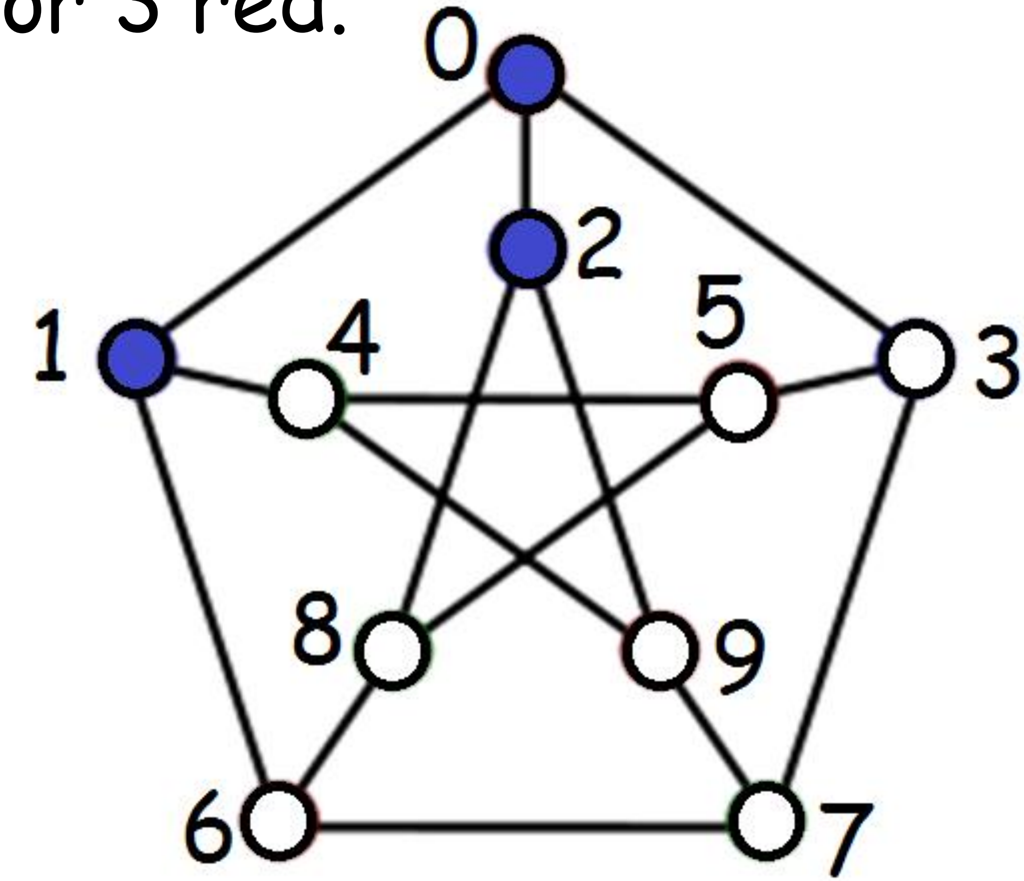
Restore data structures and return.

Recursive routines should restore data structures to avoid need to copy them.

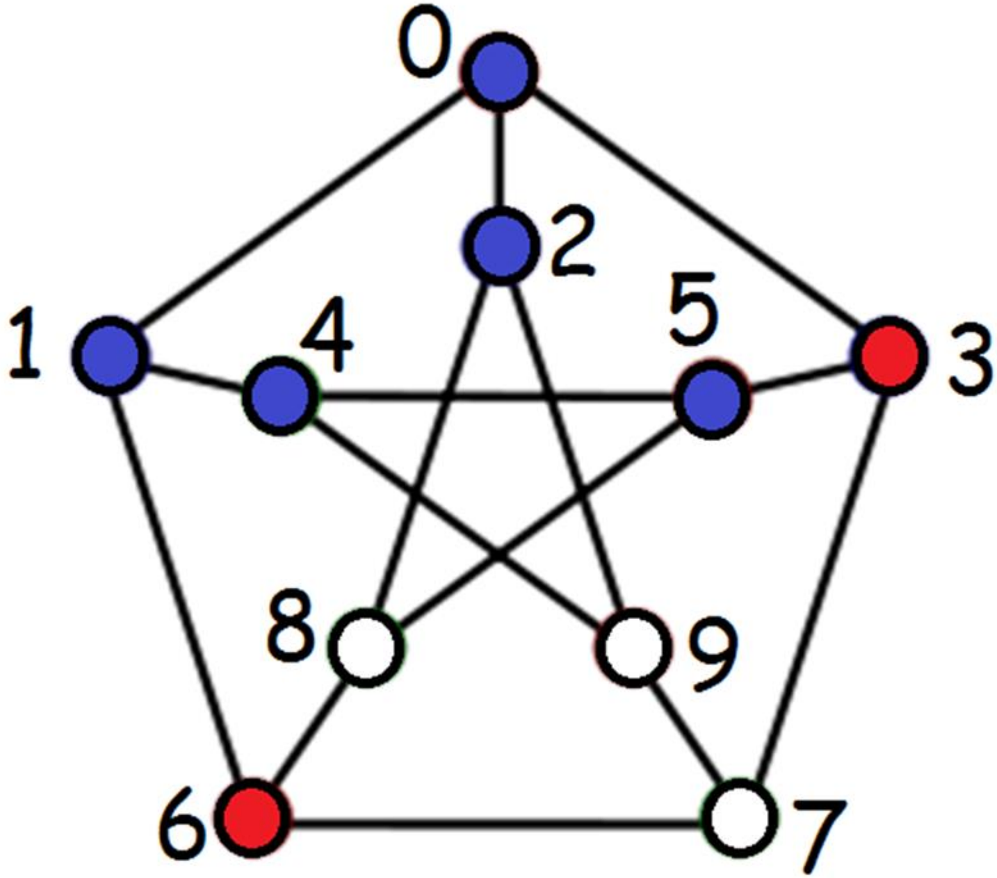
Level 0: initially all vertices are white:



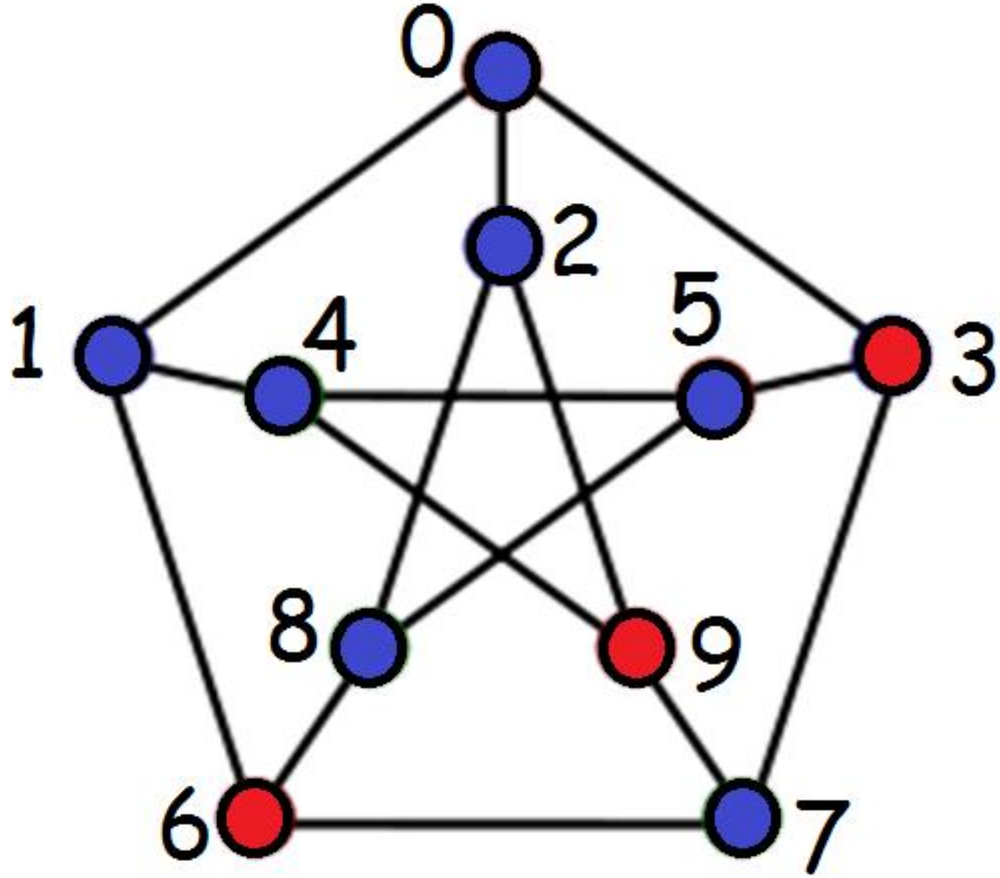
Levels 0, 1, 2: vertices are set to blue initially. Level 3: try blue and then 0 has num_choice[0]=0 so back up from level 4 then color 3 red.



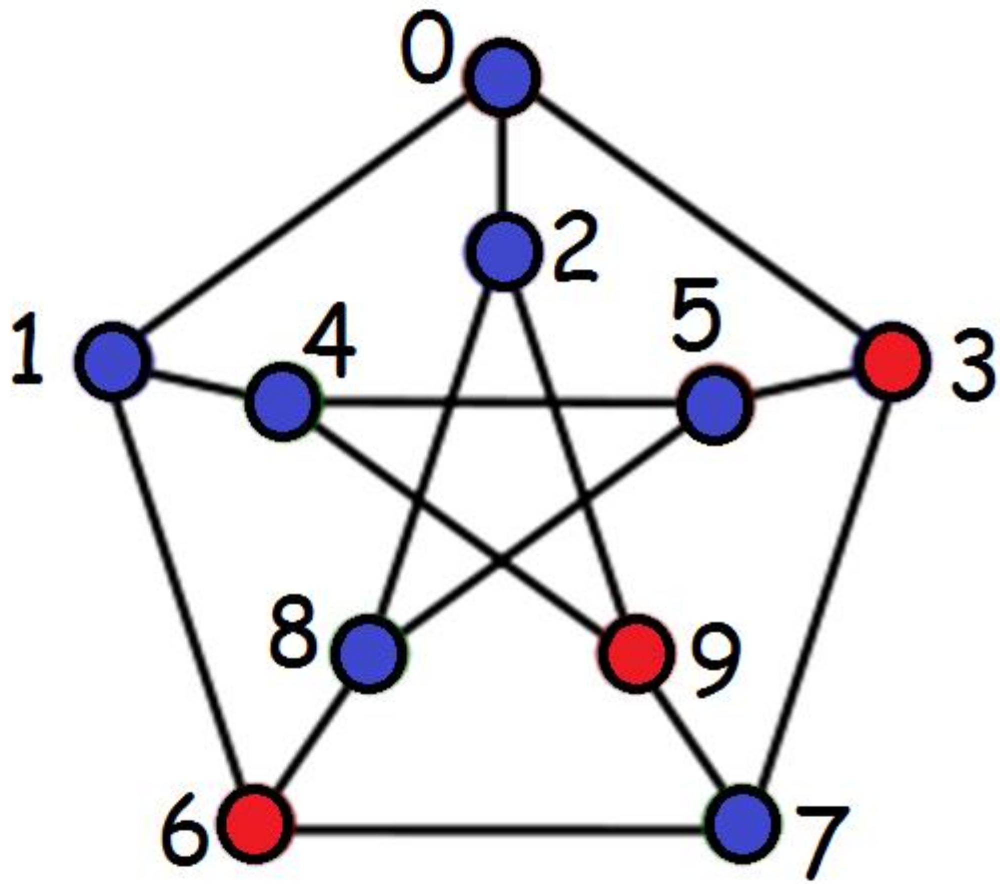
Levels 4, 5 choose blue initially, level 6 tries blue (but then num_choice[1]=0 at level 7) and then red.



Levels 7, 8 try blue initially. Level 9 try blue (but then num_choice[2]= 0 at level 10) and then red.



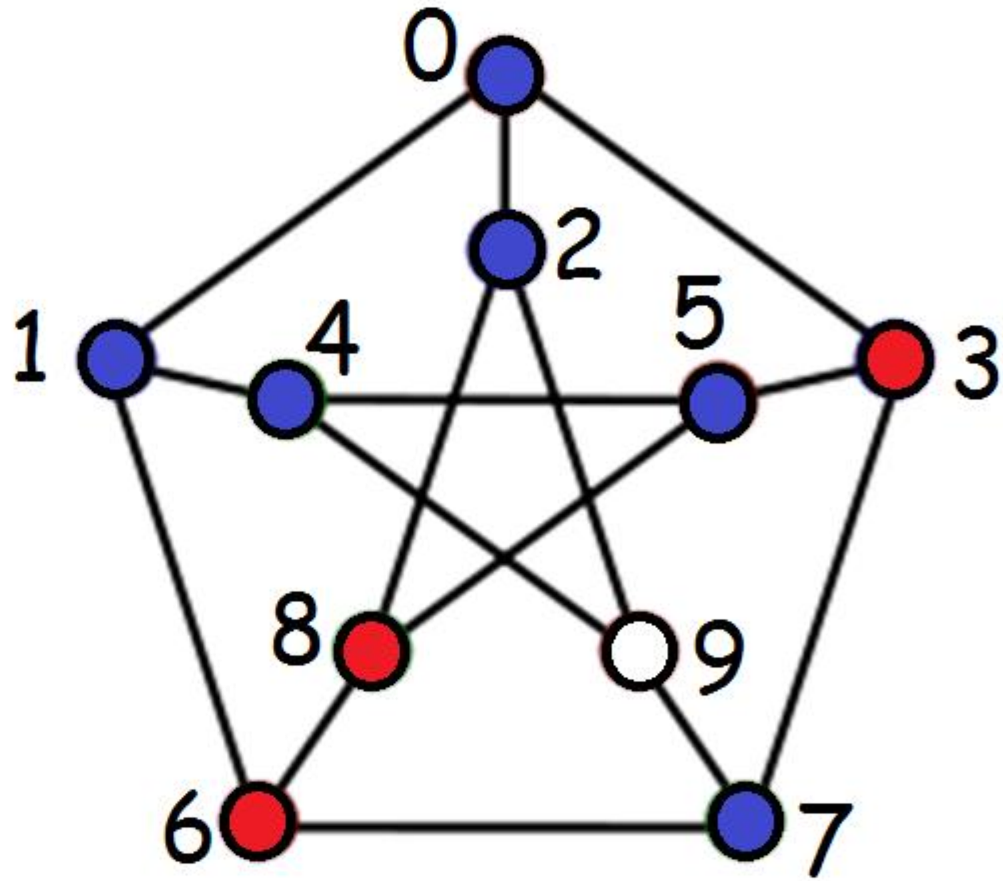
Record this (min_size, min_dom) since better than best so far (10) and return.



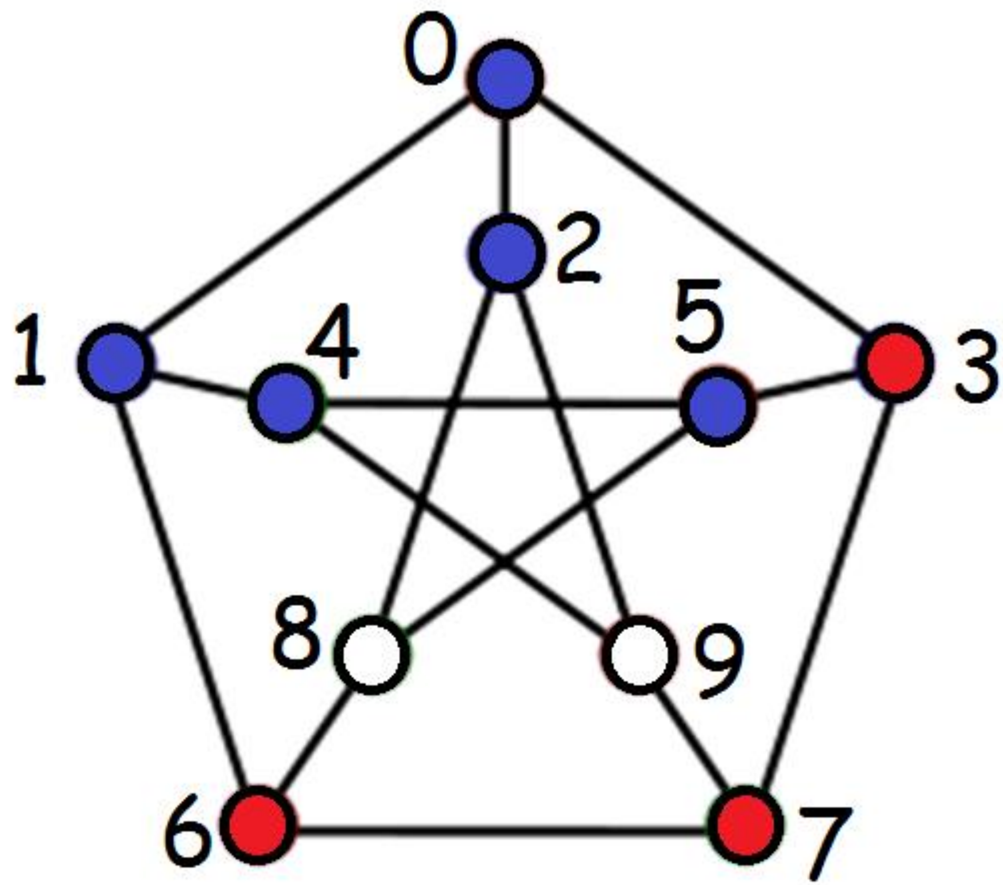
At level 8, try vertex 8 as red. At level 9:

size=3, n_extra= $\lceil 2/(3+1) \rceil$ =1

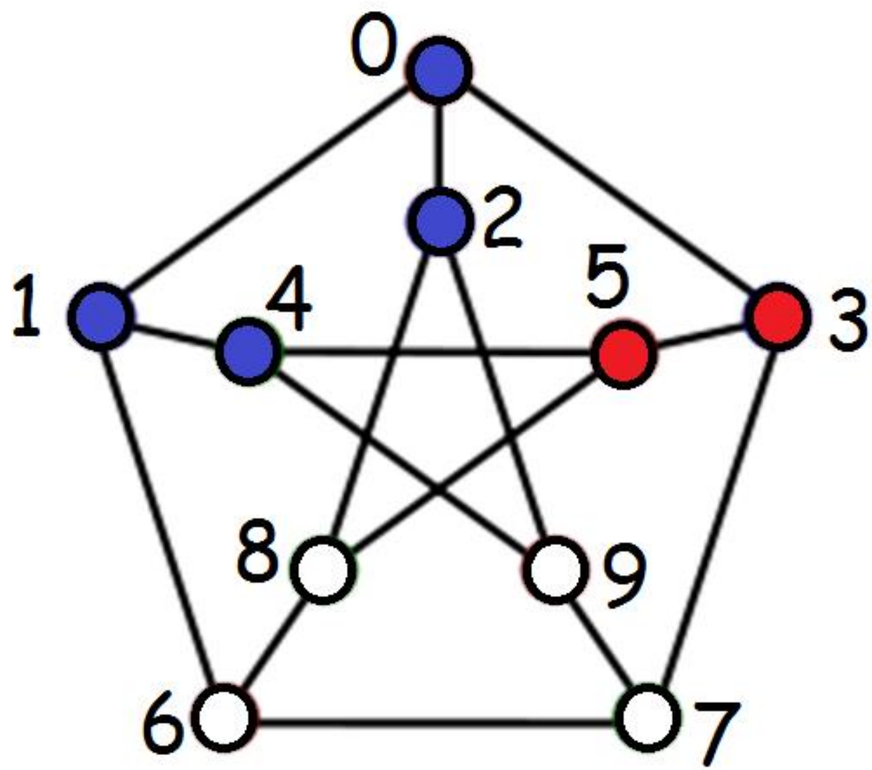
size + n_extra = 4 $\geq$ 3 = min_size so return.



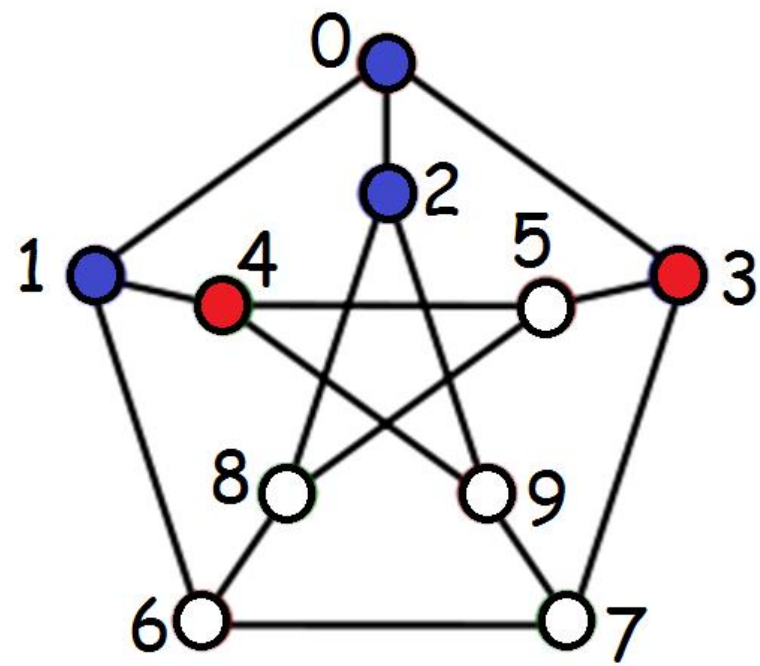
At level 7, try vertex 7 as red. At level 8:
size=3, n_extra= $\lceil 2/(3+1) \rceil$ =1
size + n_extra = 4 $\geq$ 3 = min_size so return.



At level 5, try vertex 5 as red. At level 6:
size=2, n_extra= $\lceil 4/(3+1) \rceil$ =1
size + n_extra = 3 ≥ 3 = min_size so return.

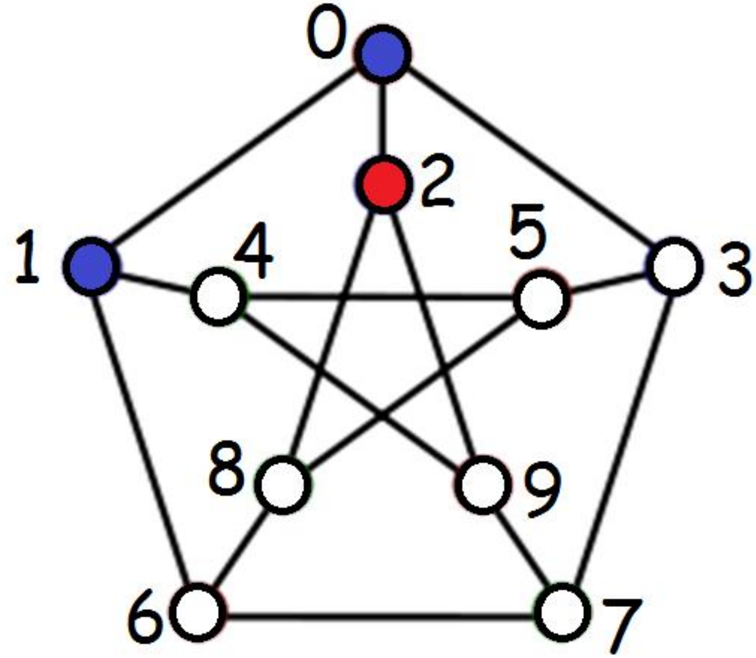


At level 4, try vertex 4 as red. At level 5:
size=2, n_extra= $\lceil 3/(3+1) \rceil$ =1
size + n_extra = 3 ≥ 3 = min_size so return.



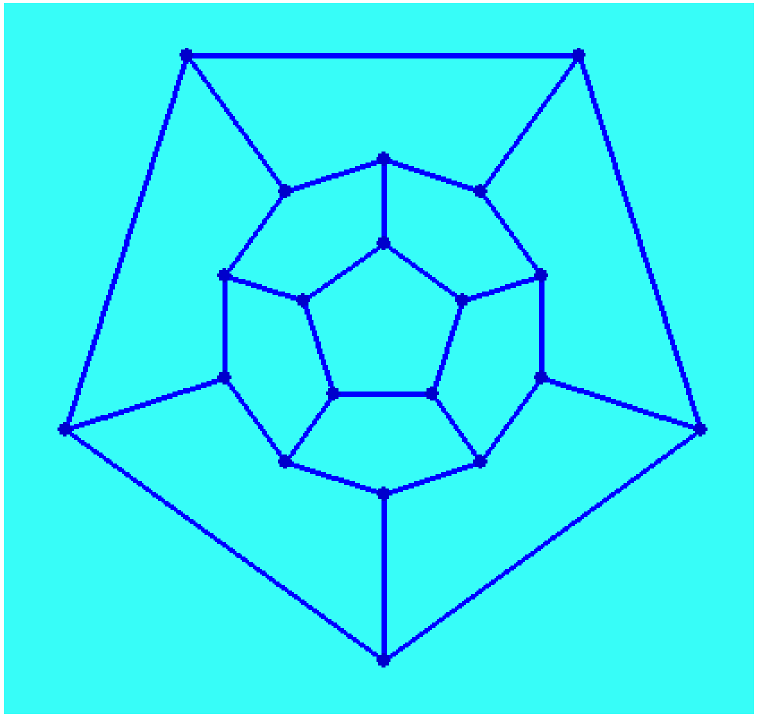
At level 42 try vertex 2 as red. At level 3:
size=1, n_extra= $\lceil 6/(3+1) \rceil = 2$
size + n_extra = 3 ≥ 3 = min_size so return.

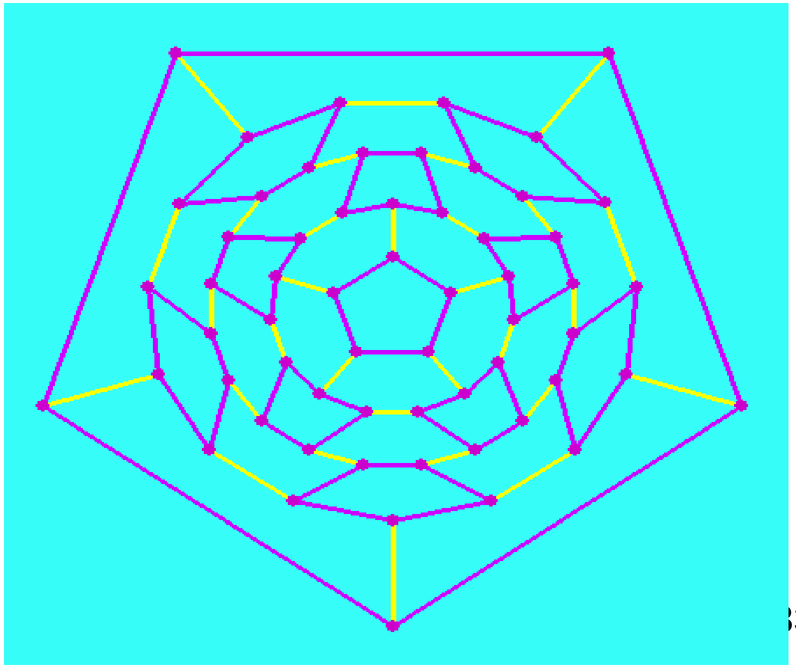
Similarly, we return at levels 1 and 0 after trying red.

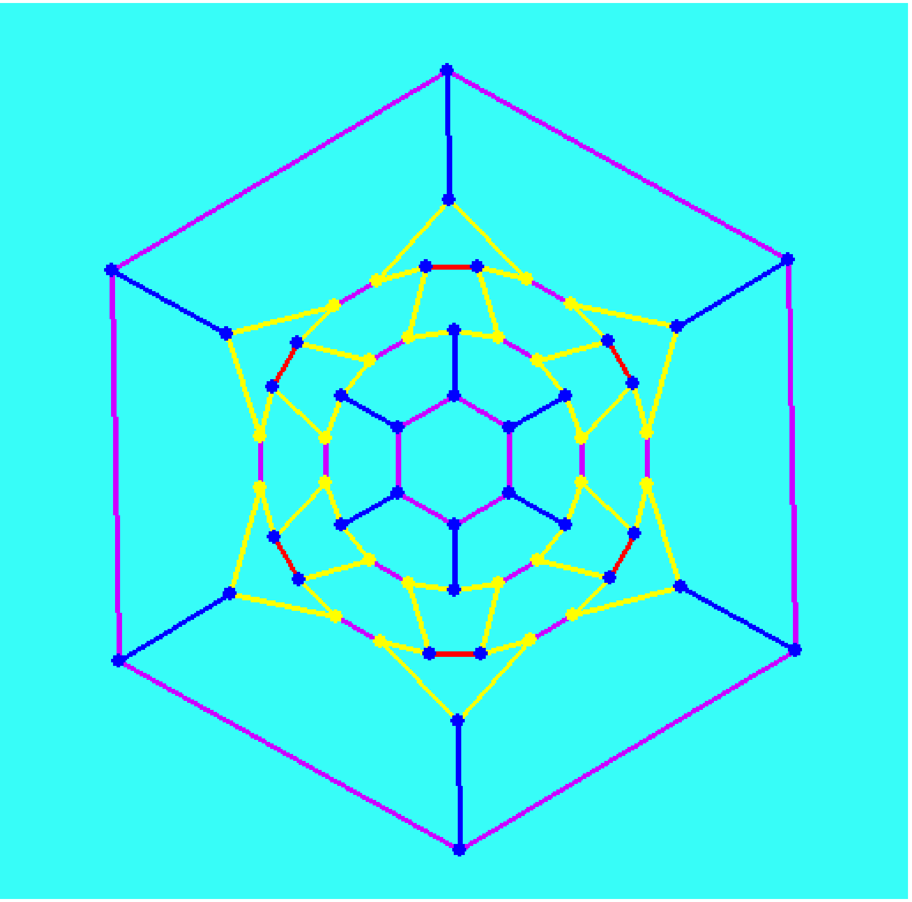


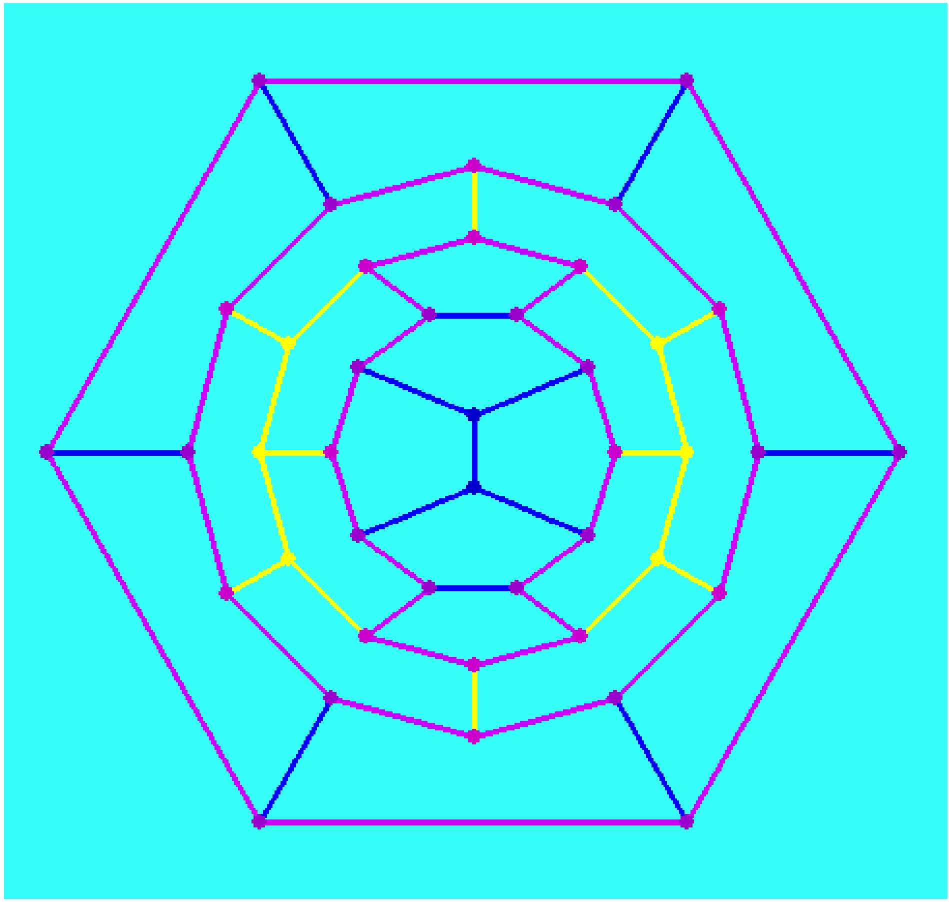
Fullerenes

- Correspond to 3-regular planar graphs.
- All faces are size 5 or 6.
- Euler's formula: exactly 12 pentagons.









Command file for running on small fullerenes (run_com):

```
time  a.out  1 < c020 > o020
time  a.out  1 < c024 > o024
time  a.out  1 < c026 > o026
time  a.out  1 < c028 > o028
time  a.out  1 < c030 > o030
time  a.out  1 < c032 > o032
time  a.out  1 < c034 > o034
time  a.out  1 < c036 > o036
time  a.out  1 < c038 > o038
time  a.out  1 < c040 > o040
time  a.out  1 < c042 > o042
time  a.out  1 < c044 > o044
time  a.out  1 < c046 > o046
time  a.out  1 < c048 > o048
time  a.out  1 < c050 > o050
time  a.out  1 < c052 > o052
time  a.out  1 < c054 > o054
time  a.out  1 < c056 > o056
time  a.out  1 < c058 > o058
time  a.out  1 < c060 > o060
```

To run this:
source run_com

Timing data for all small fullerenes:

n	#	time
20	1	0
24	1	0
26	1	0.004
28	2	0
30	3	0.004
32	6	0.02
34	6	0.016
36	15	0.076
38	17	0.092
40	40	0.672

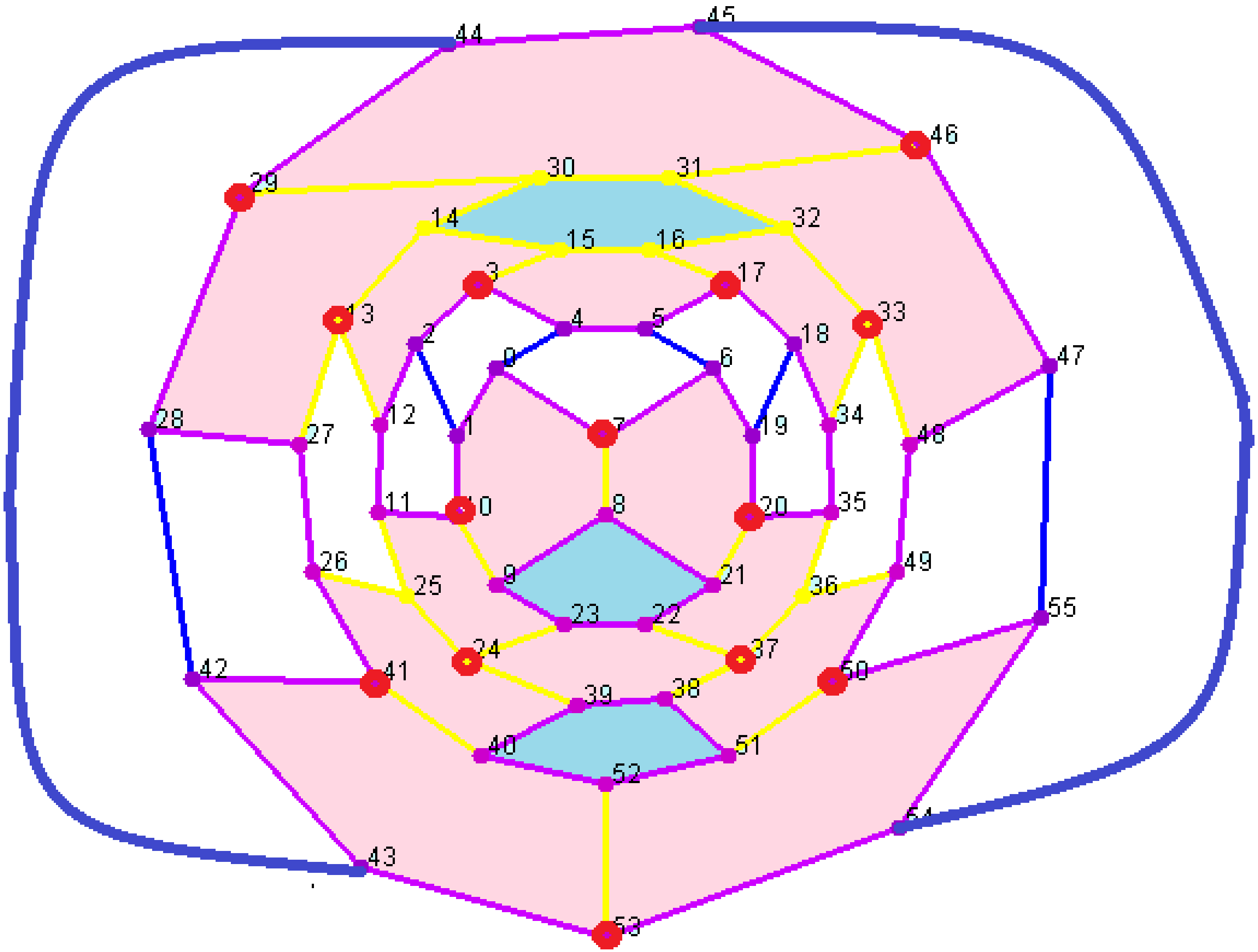
n	lb	#	time
42	11	45	0.504
44	11	89	2.6
46	12	116	2.728
48	12	299	13.66
50	13	271	13.592
52	13	437	58.023
54	14	580	58.44
56	14	924	295.042
58	15	1205	248.143
60	15	1812	1109.341

= 4.9 minutes

= 18.5 minutes

For 40: 0.672u 0.000s 0:00.67 100.0% 0+0k 0+24io 0pf+0w

n	lb	#	time	Adj. list
42	11	45	0.504	0.104
44	11	89	2.6	0.532
46	12	116	2.728	0.544
48	12	299	13.66	2.62
50	13	271	13.592	2.54
52	13	437	58.023	10.58
54	14	580	58.44	10.38
56	14	924	295.042	51.46
58	15	1205	248.143	42.11
60	15	1812	1109.341	183.7



Only fullerene isomer C_56:649 has dominating set order 14.

n	LB	#LB	#LB+1	#LB+2	Adj. list
40	10	1	21	18	0.156
42	11	1	44	0	0.104
44	11	0	55	34	0.532
46	12	6	110	0	0.544
48	12	1	109	89	2.62
50	13	6	265	0	2.54
52	13	0	270	167	10.58
54	14	19	561	0	10.38
56	14	1	470	453	51.46
58	15	23	1182	0	42.11
60	15	0	1014	798	183.7

Important: all computations should be carefully double checked by at least 2 different people with independent programs.

I have NOT double checked these results.
But you can double check them for me.

Some published papers were buggy:
Initial proof of 4-color theorem.
Lam, C. W. H.; Thiel, L.; and Swiercz, S. "The Nonexistence of Finite Projective Planes of Order 10." Canad. J. Math. 41, 1117-1123, 1989.

Some conjectures for fullerenes:

If n is divisible by 4 then the minimum dominating set order is either n/4, n/4 +1, or n/4+2. Can we characterize the cases that are n/4?

If n is not divisible by 4 (n is congruent to 2 mod 4) then the minimum dominating set order is
$\left\lceil \frac{n}{4} \right\rceil$ or $\left\lceil \frac{n}{4} \right\rceil$ +1.

There is a linear time (or maybe $O(n^2)$ time) algorithm for finding a minimum size dominating set of a fullerene.