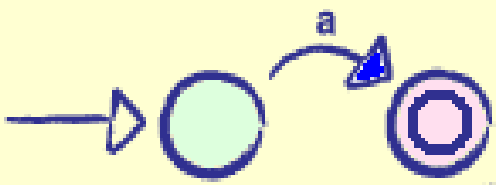
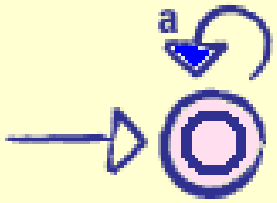
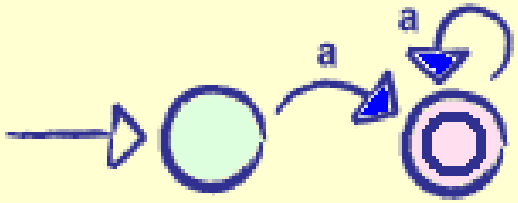
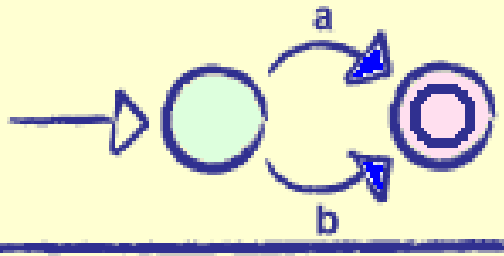
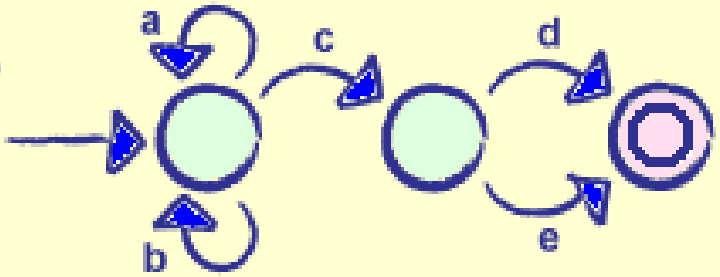
Give two proofs that L is not regular.

$L = \{ w \in \{0,1\}^* : w = 0^r 1^s \text{ where } r \leq s \leq 2r \}$

For your two proofs:
Let k be the number of states in a DFA accepting L.

(a) Use $w = 0^k 1^{2k}$.

(b) Use $w = 0^k 1^k$.

regular expression	finite state machine
a	
a*	
a+	
a/b	
(a/b)*c(d/e)	

http://www.cgl.ucsf.edu/Outreach/bmi280/slides/swc/lec/re.html

~L 2006

Algorithms to Answer Questions about Regular Languages

```
if (if23==23)
   x= -23.2e23-6;
```

The first step of a compiler is to break your program into tokens.
Tokens:

Keywords: if

Brackets: ( )

Variables: if23 x

Assignment: =

Math Operator: -

```
if (if23==23)
   x= -23.2e23-6;
```

Logical: ==

Delimiter: ;

Double:-23.2e23

Integers: 23 6

Keywords:

if U while U int U double U switch U ….

Variables: Not a Keyword but of the form:

(a-z U A-Z)(a-z U A-Z U 0-9 U _)*

Non-negative Integers: N= (0 U (1-9)(0-9)*)

Numeric values:

(Φ* U -) U N (Φ* U . (0-9)*)

(Φ* U (eUE)(Φ* U + U -) N)

The pumping lemma and also closure properties are used to prove languages are not regular.

Regular languages are closed under:

- union
- concatenation
- Kleene star
- complement
- intersection
- exclusive or
- difference
- reversal

There are algorithms for the following questions about regular languages:

1. Given a DFA M and a string w, is $w \in L(M)$?

2. Given a DFA M, is $L(M) = \Phi$?

3. Given a DFA M, is $L(M) = \Sigma^*$?

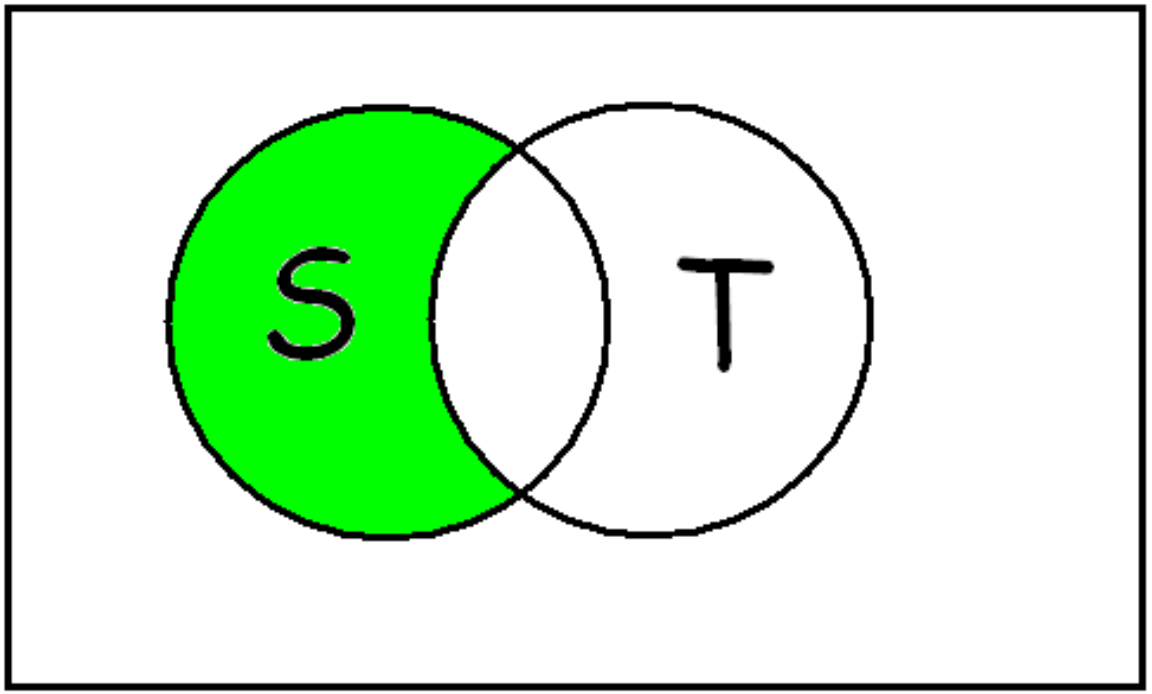
4. Given DFA's M_1 and M_2, is $L(M_1) \subseteq L(M_2)$?

5. Given DFA's M_1 and M_2, is $L(M_1) = L(M_2)$?

How can we use these to check correctness of student answers for the java tutorial?

Java Regular expression tutorial:

S= student answer, T= teacher answer

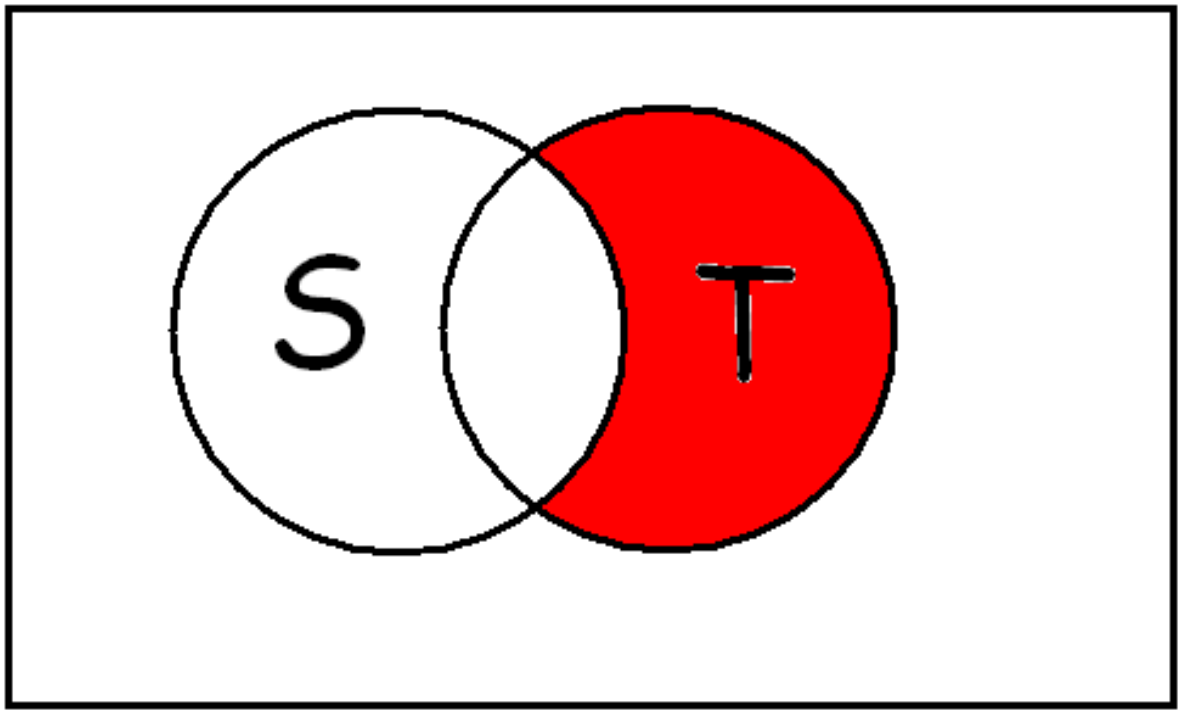


Strings student generates but should not.

Is S intersect the complement of T = Φ?

Java Regular expression tutorial:

S= student answer, T= teacher answer



Strings student should generate but does not.

Is T intersect the complement of S = Φ?

The Pumping Lemma for Regular Languages:

If L is a language accepted by a DFA with k states, and $w \in L$, $|w| \geq k$, then $\exists$ x, y, z such that

1. $w = x\ y\ z$,
2. $y \neq \varepsilon$,
3. $|\ x\ y\ | \leq k$, and
4. $x\ y^n\ z$ is in L for all $n \geq 0$.

The pumping lemma is NOT strong enough to work directly to prove that certain languages are not regular.

Let L be a language which has a constant k such that for all $w \in L$, $|w| \geq k$, $\exists$ x, y, z such that

1. $w = x\ y\ z$,
2. $y \neq \varepsilon$,
3. $|\ x\ y\ | \leq k$, and
4. $x\ y^n\ z$ is in L for all $n \geq 0$.

This is necessary but not sufficient for a language to be regular.

Then you CANNOT conclude that L is regular.

Counterexample: See assignment 3.

$L_1 = \{ u\ u^R\ v : u, v \text{ in } \{a, b\}^+ \}$

Table 1: Comparing polynomial and exponential time complexity.
Assume a problem of size one takes 0.000001 seconds (1 microsecond).

	Size *n*					
	10	**20**	**30**	**40**	**50**	**60**
n	0.00001 second	0.00002 second	0.00003 second	0.00004 second	0.00005 second	0.00006 second
n^2	0.0001 second	0.0004 second	0.0009 second	0.0016 second	0.0025 second	0.0036 second
n^3	0.001 second	0.008 second	0.027 second	0.064 second	0.125 second	0.216 second
n^5	0.1 second	3.2 second	24.3 second	1.7 minutes	5.2 minutes	13.2 minutes
2^n	0.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	0.059 second	58 minutes	6.5 years	3855 centuries	$2*10^8$ centuries	$1.3*10^{13}$ centuries

(from M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, New York, 1979.)

Table 2: Effect of improved technology on several polynomial and exponential time algorithms. The following table represents the size of the largest problem instance solvable in 1 hour.

Time Complexity function	With present computer	With computer 100 times faster	With computer 1000 times faster
n	N1	100 N1	1000 N1
n^2	N2	10 N2	31.6 N2
n^3	N3	4.46 N3	10 N3
n^5	N4	2.5 N4	3.98 N4
2^n	N5	N5+6.64	N5+9.97
3^n	N6	N6+4.19	N6+6.29

(from M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, New York, 1979.)

Theorem 1:

If Hamilton Cycle has an algorithm which is polynomial time, then so does Hamilton Path.

Theorem 2:

If Hamilton Path has an algorithm which is polynomial time, then so does Hamilton Cycle.

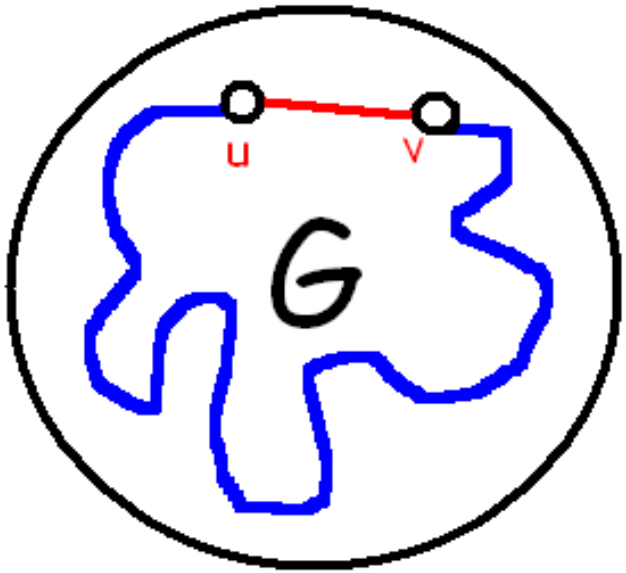
Currently nobody knows a polynomial time algorithm for either problem.

Hamilton Path

If (G has a Hamilton Cycle) return(yes)

For each missing edge (u, v), if G+(u,v) has a Hamilton Cycle return(yes)

Return(no)



Time:

$O(n^2 * p(n, m+1))$

What is inefficient about this code?

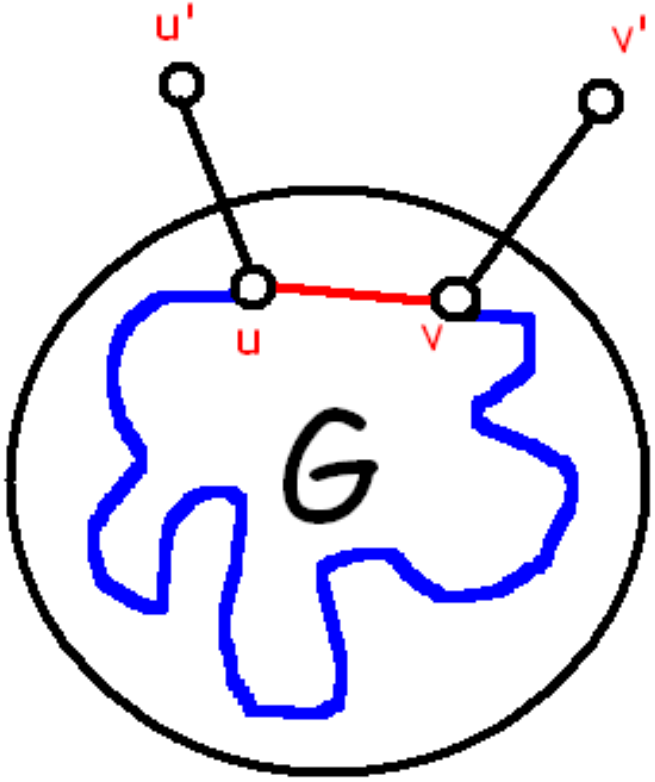
```
for (i=0; i < n; i++)
    for (j=0; j < n; j++)
    {
        tmp= G[i][j]; G[i][j]=1; G[j][i]=1;
        if (hamilton_cycle(n, G)) return(1);
        G[i][j]=tmp; G[j][i]=tmp;
     }
return(0);
```

My solution:

```
if (hamilton_cycle(n, G)) return(1);
for (i=0; i < n; i++)
      for (j=i+1; j < n; j++)
            if (G[i][j] == 0)
            {
                  G[i][j]=1; G[j][i]=1;
                  found= hamilton_cycle(n, G);
                  G[i][j]=0; G[j][i]=0;
                  if (found) return(1);
             }
    return(0);
```

Hamilton Cycle:

For each edge (u,v) if G + u′ + v′ + (u, u′) + (v, v′) has a Hamilton Path return (yes)



Return(no)

Time:
$O(n^2\, q(n+2, m+2))$

My solution:

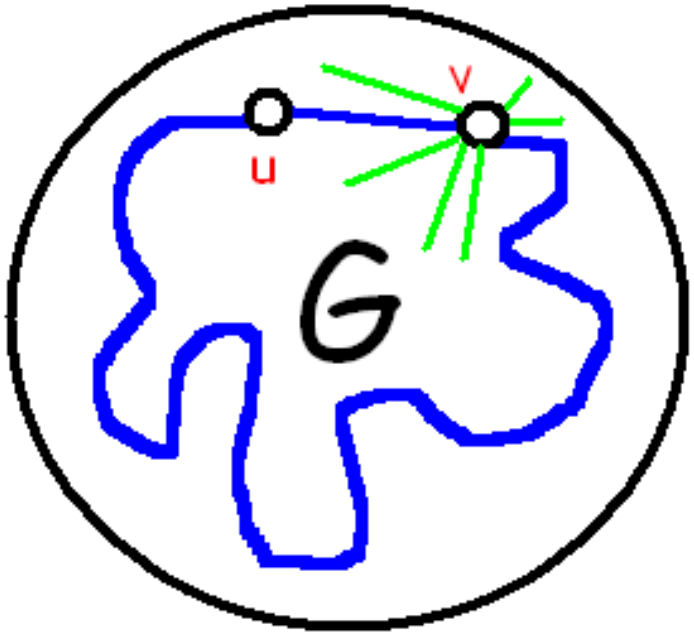
```
for (i=0; i < n+2; i++)
    for (j=n; j < n+2; j++) { G[i][j]=0; G[j][i]=0;}

for (i=0; i < n; i++)
  for (j=i+1; j<n; j++)
    if (G[i][j])
    {
       G[i][n]=1; G[n][i]=1; G[j][n+1]=1;G[n+1][j]=1;
       if (hamilton_path(n+2, G)) return(1);
       G[i][n]=0; G[n][i]=0;G[j][n+1]=0;G[n+1][j]=0;
     }
return(0);
```

Hamilton Cycle: A better solution

For each neighbour of v if G + u′ + v′ + (u, u′) + (v, v′) has a Hamilton Path return (yes)

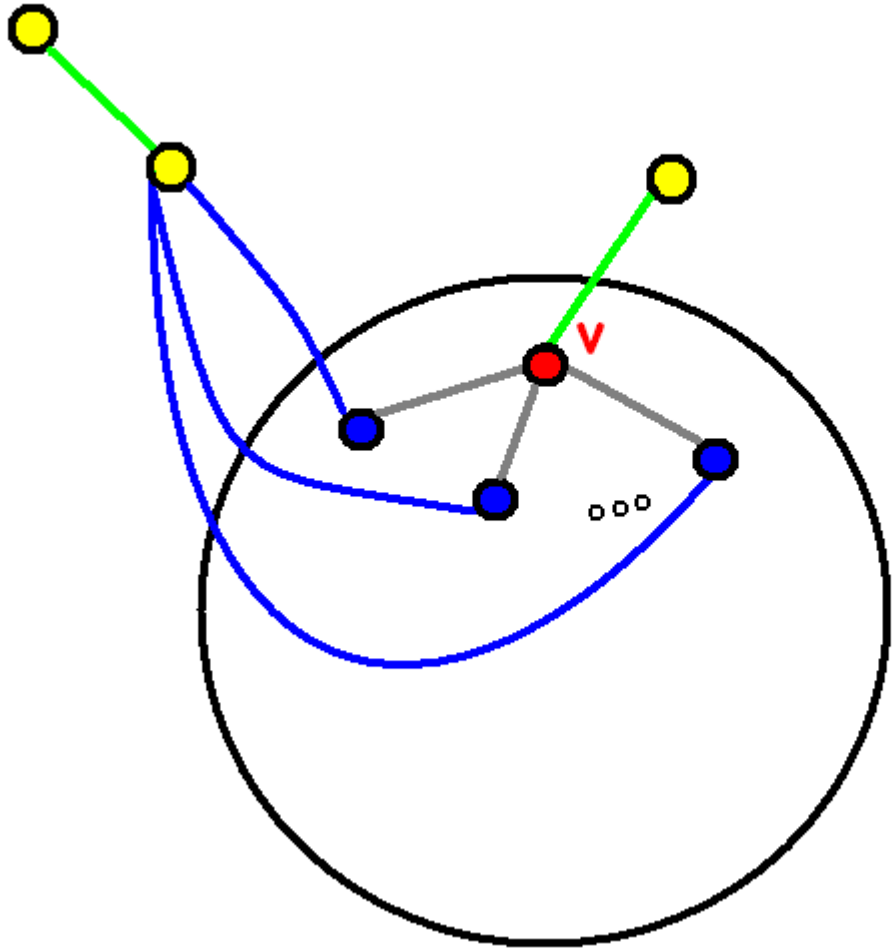
Return(no)



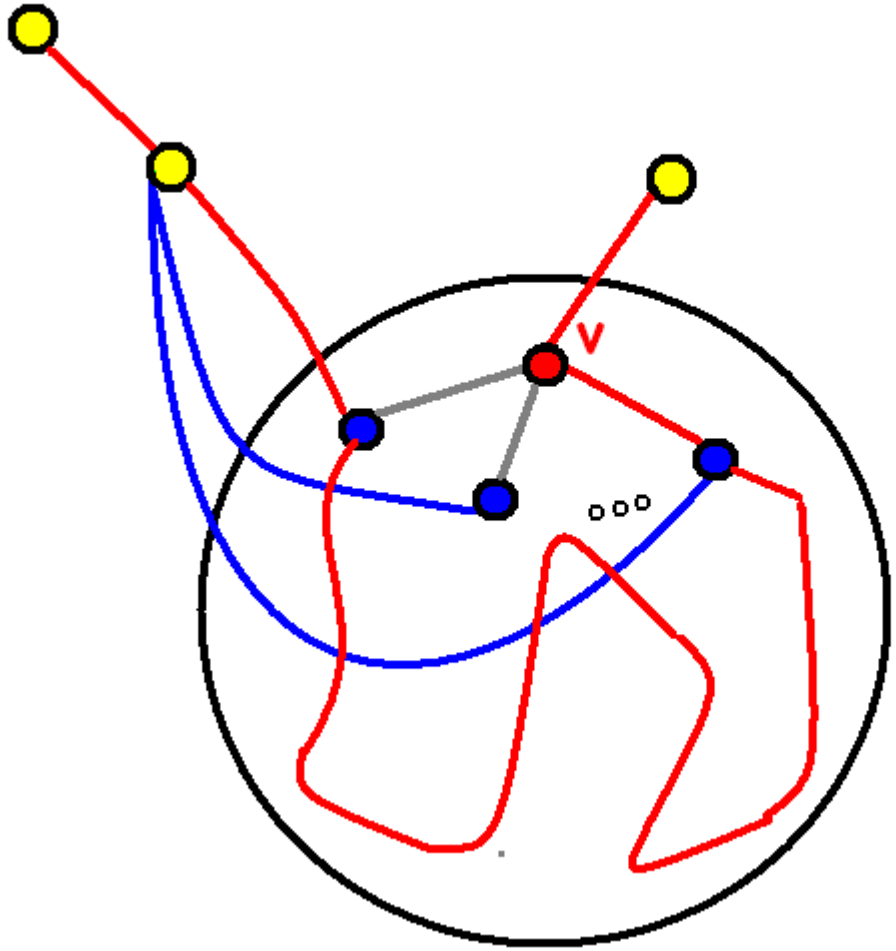
Time:

O(n * q(n+2, m+2))

A creative solution:



A creative solution:

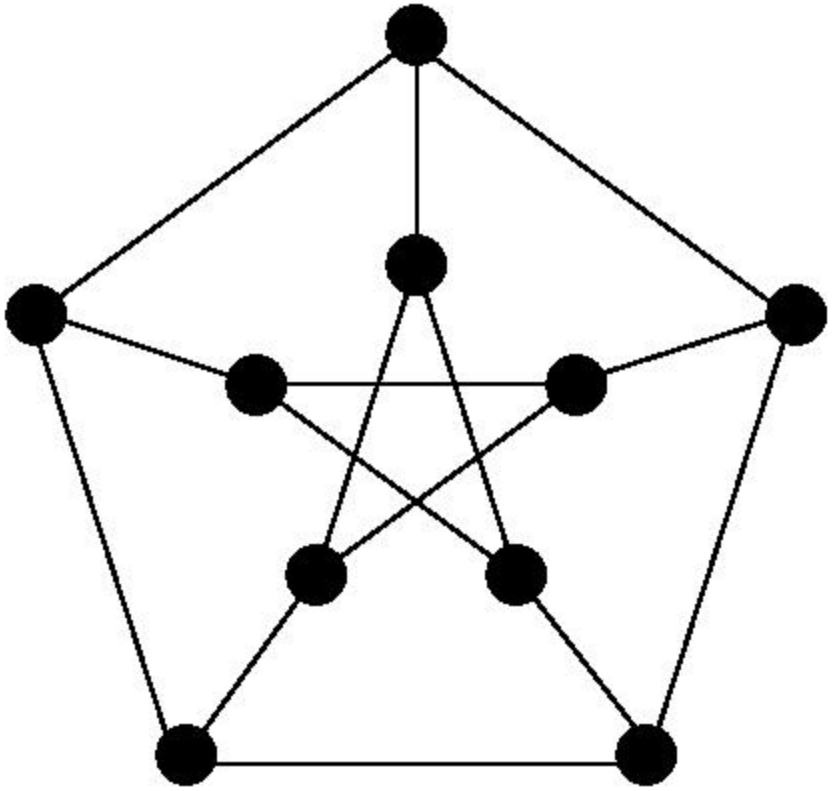


Some algorithms which do not work:

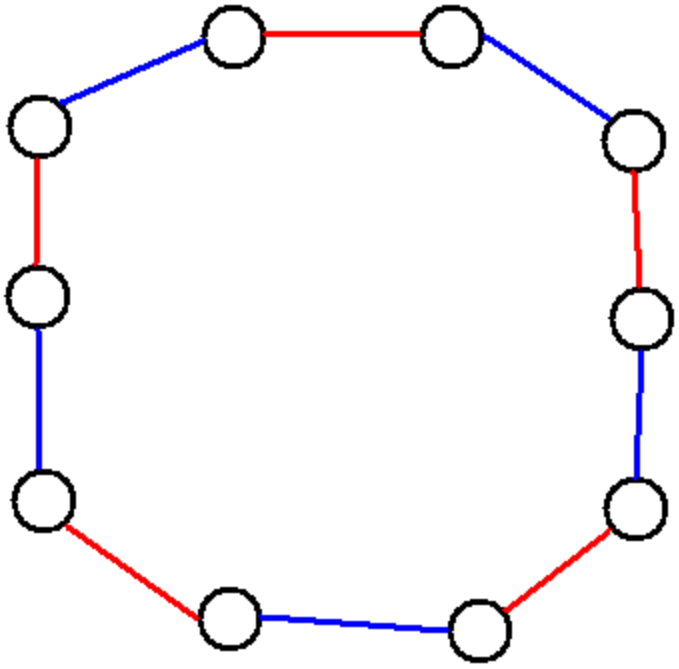
If $G-v$ has a Hamilton path for all vertices v this does NOT mean G has a Hamilton cycle.

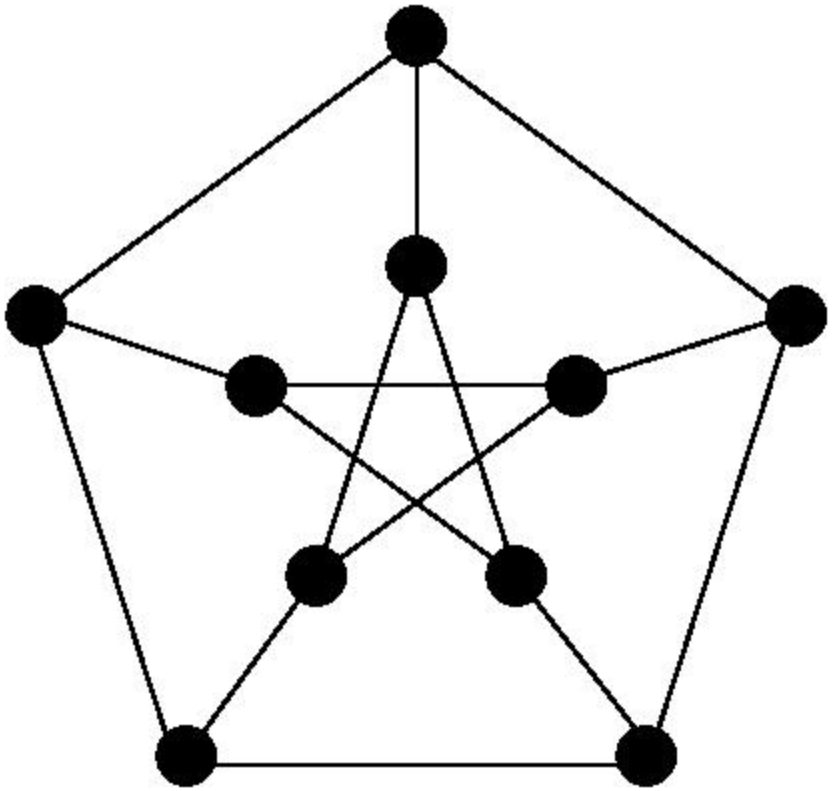
If $G-e$ has a Hamilton path for all edges e this does NOT mean G has a Hamilton cycle.

Theorem: The Petersen graph has no Hamilton cycles.

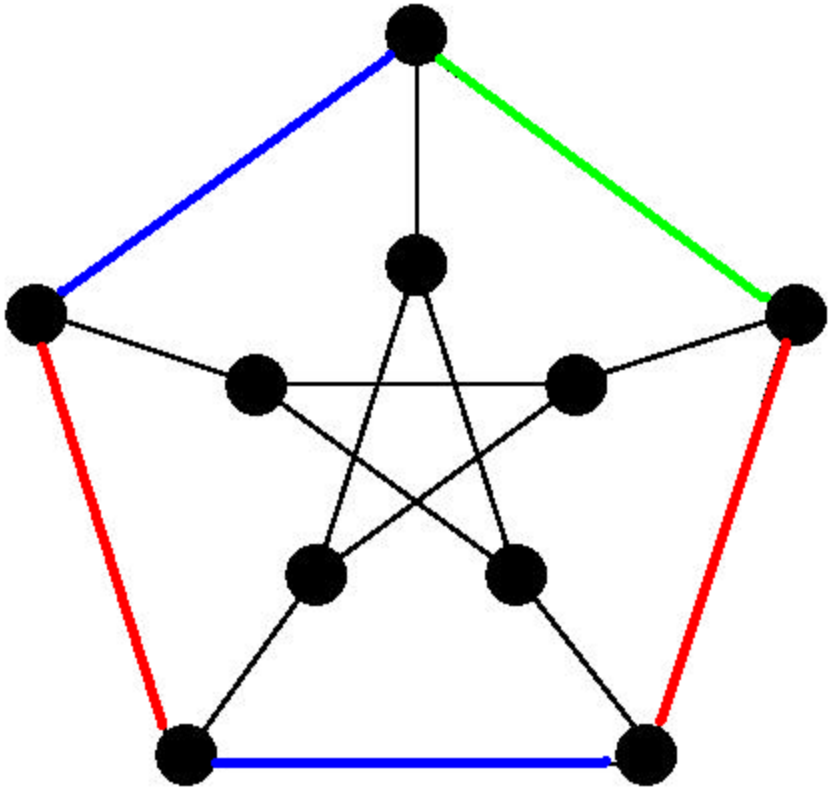


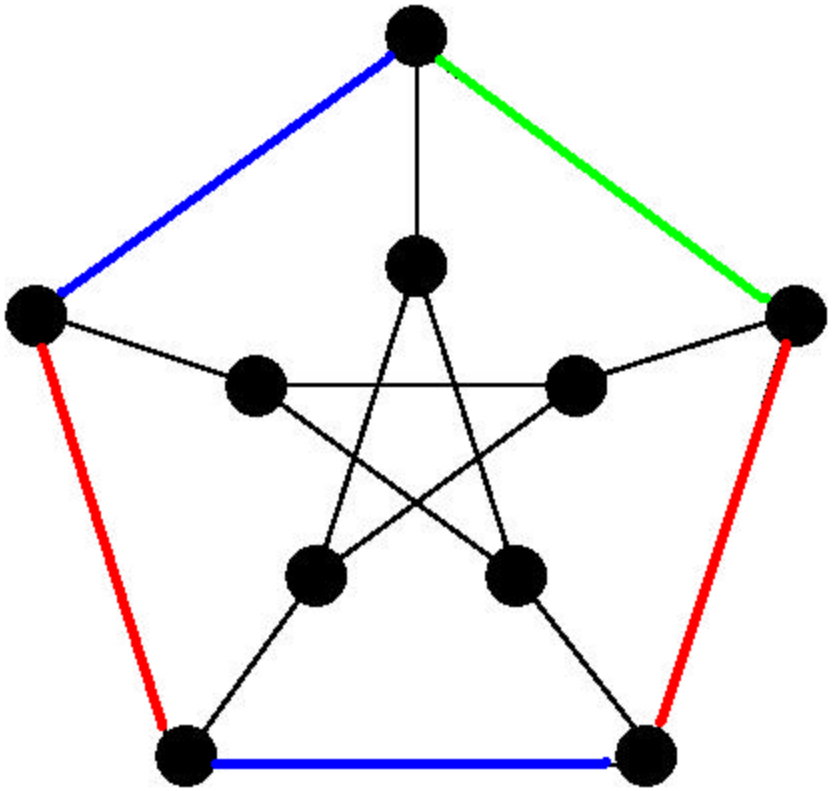
If it did, the edges would be 3-edge colourable.



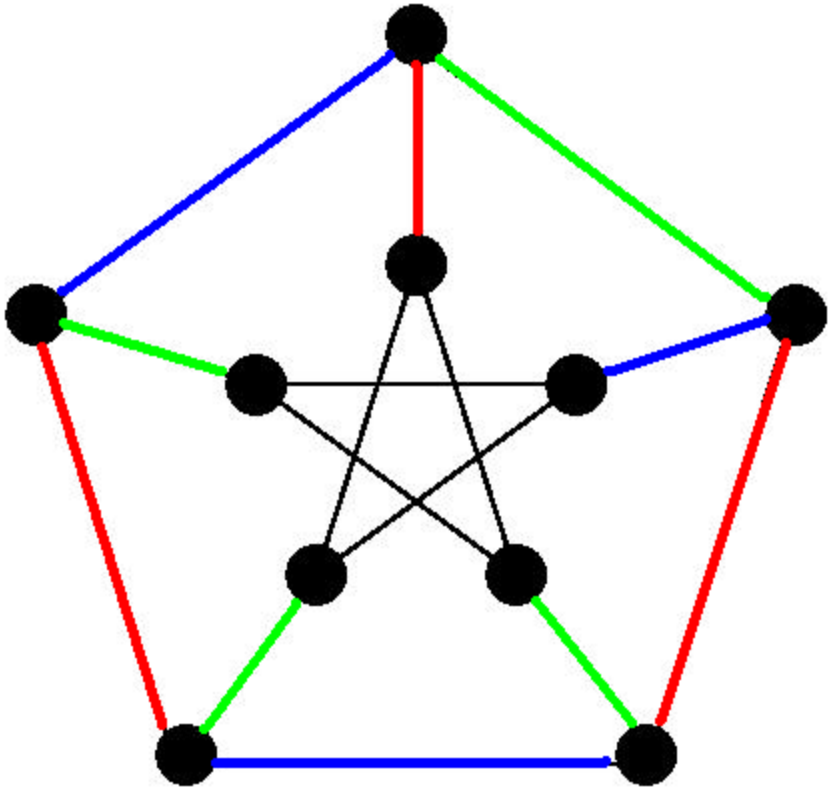


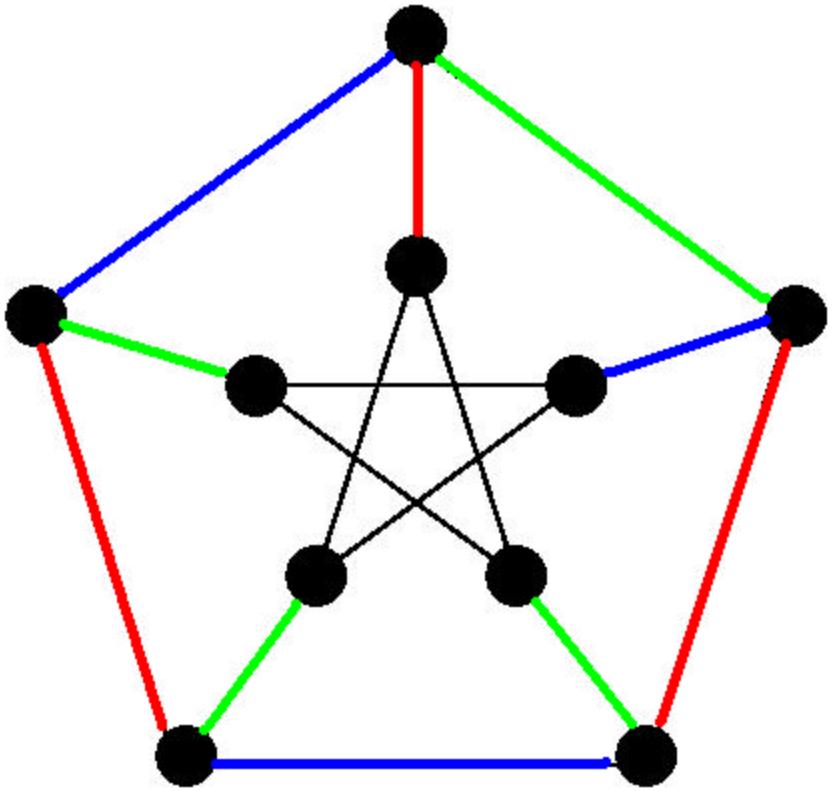
On outside cycle: one colour used once and others used twice:



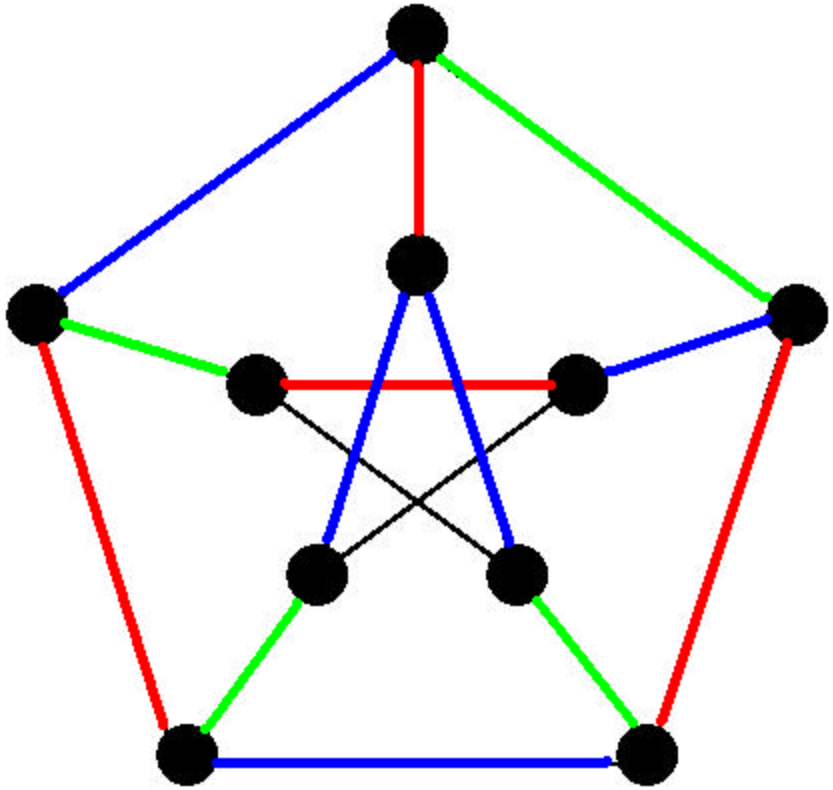


Colours of spoke edges are forced:

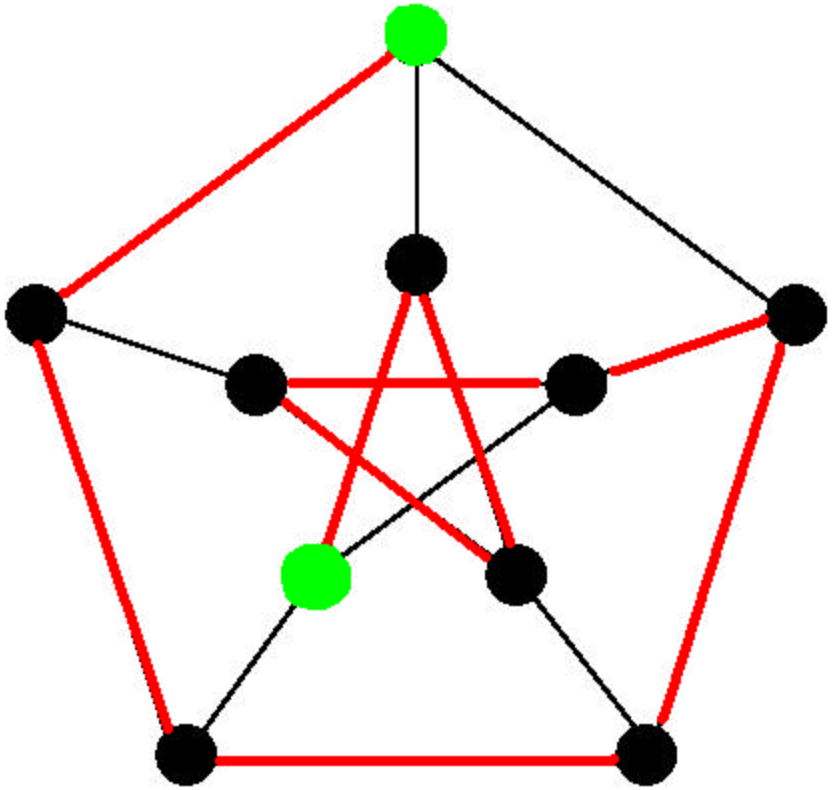




More forcings
then stuck:



G-v has a Hamilton Path for all v.



G-e has a Hamilton Path for all e.