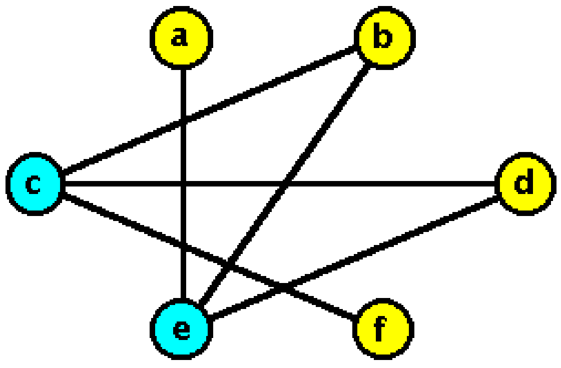
Vertex Cover (VC)

Instance: **Given an *n*-node undirected graph *G(V,E)* with node set *V* and edge set *E*; a positive integer *k* with *k*<=*n*.**

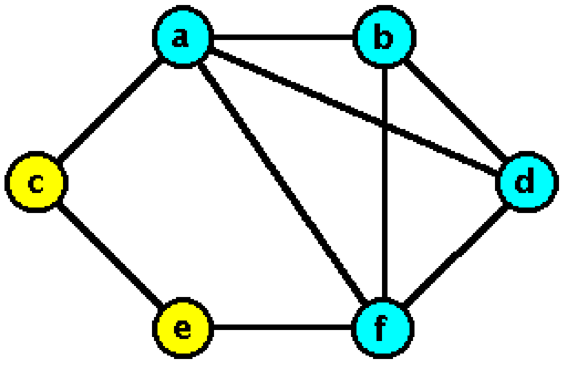
Question: **Is there a subset *W* of *V* having size at most *k* and such that for every edge *{u,v}* in *E* at least one of *u* and *v* belongs to *W*?**



CLIQUE

***Instance*:** ***Given an n*-node undirected graph *G(V,E)* with node set *V* and edge set *E*; a positive integer *k* with *k*<=*n*.**

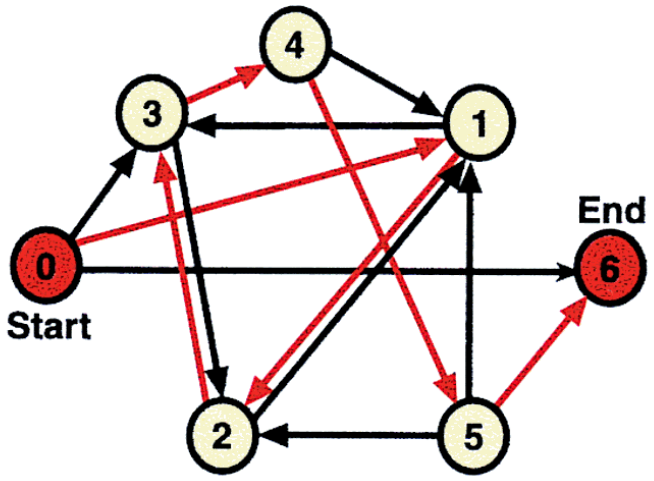
Question: **Does *G* contain a *k*-clique, i.e. a subset *W* of the nodes *V* such that *W* has size *k* and for each distinct pair of nodes *u*, *v* in *W*, *{u,v}* is an edge of *G*?**



Hamiltonian Path

Instance: Given an *n*-node undirected graph *G(V,E)*.

Question: Is there a simple path of edges in *G* that contains every node in *V*, and thus contains exactly *n-1* edges?



Longest Path

Instance: **Given an *n*-node undirected graph *G(V,E)*; nodes *s* and *t* in *V*; a positive integer *k*.**

Question: **Is there a *simple* path between *s* and *t* in *G* that contains at least *k* edges?**

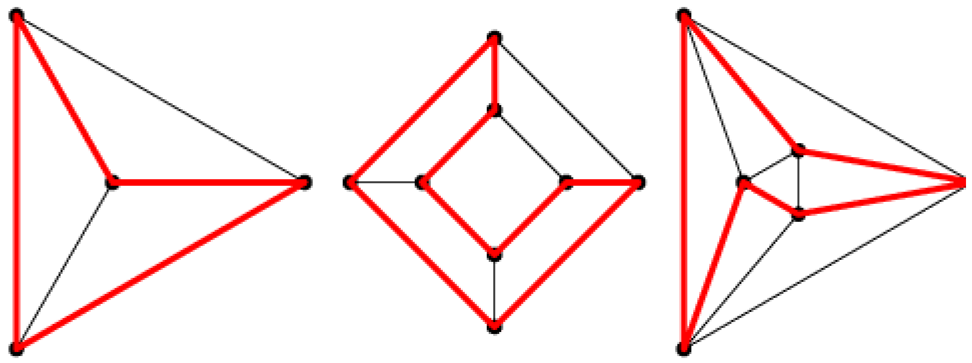
A *simple path* in a graph is just one without any repeated edges or vertices.

Hamiltonian Cycle

An undirected graph has a *Hamiltonian cycle* if there exists a cycle in the graph which visits each vertex in the graph exactly once.

Instance: **Undirected graph $G = (V,E)$.**

Question: **Does G contain a Hamiltonian cycle?**



Longest Circuit

Instance: **Given an n-node undirected graph $G(V,E)$; a positive integer k.**

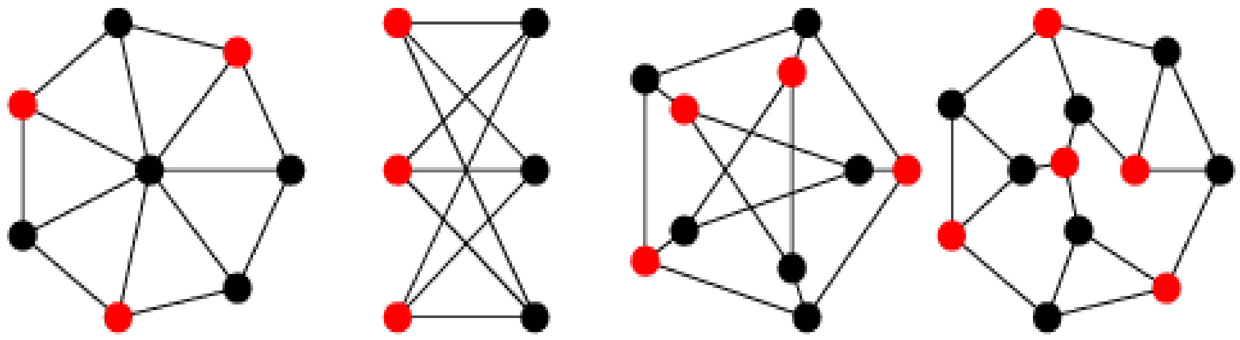
***Question*:** **Does G contain a simple cycle containing at least k nodes?**

***Comments*:** **Special cases of this are the famous Travelling Salesman Problem and Hamiltonian Circuit Problem. The latter corresponds to the cases $k = n$; the former to the case with each graph edge being weighted and also having $k = n$.**

Independent Set

Instance: ***Given an n*-node undirected graph *G(V,E)*; a positive integer *k* <= *n*.**

***Question*: Does *G* have an independent set of size at least *k*, i.e. a subset *W* of at least *k* nodes from *V* such that no pair of nodes in *W* is joined by an edge in *E*?**



Reduction

Formally, NP-completeness is defined in terms of "reduction" which is just a complicated way of saying one problem is easier than another.

We say that <u>A is easier than B</u> (A is reducible to B), and write A < B, if we can write down an algorithm for solving A that uses a small number of calls to a subroutine for B (with everything outside the subroutine calls being fast, polynomial time). There are several minor variations of this definition depending on the detailed meaning of "small" -- it may be a polynomial number of calls, a fixed constant number, or just one call. It is possible for the algorithms for A to be slower than those for B, even though A < B.

How to prove NP-completeness in practice

Most proofs of NP-completeness are based on the observation that if $A < B$ and $B < C$, then $A < C$.

As a consequence of this observation, if A is NP-complete, B is in NP, and $A < B$, B is NP-complete. In practice that's how we prove NP-completeness: We start with one specific problem that we prove NP-complete, and we then prove that it's easier than lots of others which must therefore also be NP-complete.

<u>Example</u>:

Consider the Hamiltonian cycle problem. Does a given graph have a cycle visiting each vertex exactly once? Here's a solution, using longest path as a subroutine:

```
for each edge (u,v) of G
if there is a simple path of length n-1 from u to v
return yes      // path + edge form a cycle
return no
```

This algorithm makes m calls to a longest path subroutine, and does O(m) work outside those subroutine calls, so it shows that Hamiltonian cycle < longest path.

So e.g. since Hamiltonian cycle is known to be NP-complete, and Hamiltonian cycle < longest path, we can deduce that longest path is also NP-complete.

PROVING NP–COMPLETENESS

Given a problem Π.

Question: $\Pi \in P$?

$\Pi \in NP$?

<u>Method</u>

1. **Check: $\Pi \in P$? $\rightarrow$ Look for and find a polytime algorithm $\Rightarrow \Pi \in P$–class.**

   $\rightarrow$ Ask someone else.

If we are not able to find a polytime algorithm (and nobody else), then $\rightarrow$ 2.

2. **We suspect that $\Pi \in NP$.**

   Check: $\Pi \in NP$?

   a) **Verify that checking is polytime $\overset{TRUE}{\Rightarrow}$ $\Pi \in NP$**

      (‘hint’ solution: choose at random & check whether it is a solution or not)

   b) **Π can be <u>reduced</u> to a known NP–complete problem $\Rightarrow \Pi \in NP$–complete class.**

      ! Reduction can be done in polytime.

Write or formulate the <u>*known*</u> problem in terms of the <u>*unknown*</u> problem. Not vice versa! The unknown problem cannot be harder than the known problem.

PROVE THAT THE MPS IS NP COMPLETE BY REDUCING PARTITION TO 2PS AND THEN REDUCING 2PS TO MPS

Partition
Instance: **a multiset A and a measurement of the size of element in A s:A-->N;**
Question: **can A be partitioned into subsets A0 and A1 such that the sum over elements a in A0 of s(a) is equal to the sum over element a in A1 of s(a)? (That is: can A be partitioned into equally sized subsets?)**
Multi-processor scheduling (mPS)
Instance: **A multiset A of tasks, a measurement of the time required for each task l: A-->N, a deadline (real number) D;**
Question: **is there a partition of A into m disjoint sets such that the total time (sum of l(a)) for every element in a partition is always at most D?**
Two Processor Scheduling (2PS) Same as mPS, with m=2.

Theorem: 2PS is NP complete
Proof: The problem is clearly in NP, since it is easy to verify that the total time for the tasks in each of two partitions is at most D, given the partitions as "hints".

Now, we show that Partition <= 2PS in order to conclude that the problem is NP complete.

Let (A,s) be an instance of partition. Let D be half the sum of s(a) over all a's in a. Then (A,s,D) is an instance of 2PS.

Now, if (A,s) is a "yes" instance of partition, then let A0 and A1 be the partition such that the sum of s(a)'s in each are the same. For these same partitions A0 and A1, their sum is less than or equal to (actually, equal to) D for this D. So, (A,s,D) is a "yes" instance of 2PS.

Conversely, if (A,s,D) is a "yes" instance of 2PS, then let A0, A1 be the schedules of tasks on processors 0 and 1. We have that the sum of the times s(a) over all a in A0 is at most D, and the same is true

for the sums over A1. So the sum of these two sums is at most 2D which is the sum of all s(a)'s over all a in A. But this cannot exceed the sum of all values, so 2D is exactly the sum of all weights in A, which implies that the sum of weights in both A0 and A1 is exactly half of D. Consequently, these two sums are equal and so (A,s) is a "yes" instance of partition.

In other words, partition reduces to 2PS, and so 2PS is NP complete.

Now we show that mPS is NP complete, by reducing 2PS to it.

<u>Theorem:</u> mPS is NP complete
<u>Proof:</u> That mPS is in NP is obvious, by a similar argument to the above one: the "hint" is a schedule, the verification is just adding up times on each processor and comparing to D.

Now, we reduce an arbitrary instance of 2PS, (A,s,D) to an instance of mPS.

For instance (A,s,D), form an m-processor instance (A',s',D) by letting A' = A union {v} union ... union {v}, where:

> **m-2 copies of {v} are added in the unions**
> **v is not in A**
> **s'(v) = D, and s'(x) = s(x) for all x in A**

Now, if (A,s,D) is a "yes" instance, then the same partition of A that works in the two processor case will work in the m processor case, with the other m-2 processors being assigned {v}, with a cost of at most D for all processors.

Conversely, any schedule for the m-processor case can be transformed into one for the two-processor case. Take any such schedule A1, ... Am. None of these sets can contain both v and x, for any non-v x, since the weight would then be s'(v) + s'(x) = D + s(x) > D. Furthermore, exactly m-2 of these must contain a {v}, since there are m-2 copies of v. So, let Ai and Aj be the two remaining sets (m-(m-2)) which contain only non-v elements. By construction, these are precisely the elements in A, and so they form a "yes" instance to the 2PS instance (A,s,D).