Today:

– NP-Completeness (con’t.)

COSC 581, Algorithms

April 22, 2014

Many of these slides are adapted from several online sources

Reading Assignments

- Today's class:
  - Chapter 34.5 (con't.)

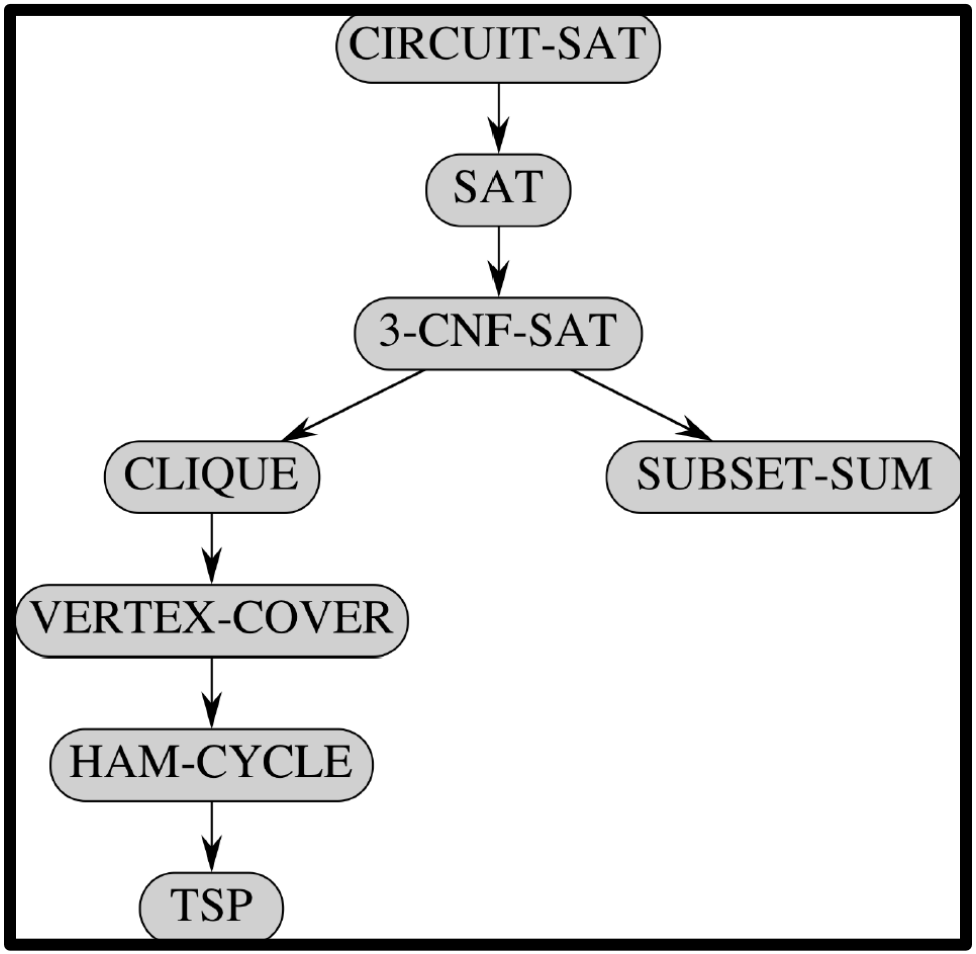
Recall: Proving NP-Completeness In Practice

1) Prove that the new problem B $\in$ NP
2) Show that **one known** NP-Complete problem, A, can be transformed to B in polynomial time (i.e., A $\leq_p$ B)

- Conclude that B is NP-Complete

Once one problem (SAT) shown to be NP-Complete, can show many others…

Example reductions (From CLRS, Ch. 34):



Subset-Sum is NP Complete

Subset-Sum Problem:

- Given: finite set S of positive integers and integer target t > 0

  (That is, input instance is <S, t>)

- Question: Is there a subset S' $\subseteq$ S such that $t = \sum_{s \in S'} s$

Proving NP-Completeness:

- Step 1: Subset-Sum $\in$ NP
  - Argue that, given a certificate, you can verify that the certificate provides a solution in polynomial time
- Step 2: Show that some known NP-Complete problem is reducible in poly-time to Subset-Sum (i.e., A $\leq_p$ Subset-Sum)
  - What known NP-Complete problem do we choose?

3-CNF-SAT $\leq_p$ Subset-Sum

- What do we have to do?
  1) Given an instance $< \phi >$ of 3-CNF-SAT, define poly-time function *f* that converts $< \phi >$ to instance $< S, t >$ of Subset-Sum
  2) Argue that *f* is poly-time
  3) Argue that *f* is correct (i.e., ϕ is satisfiable iff S has a subset that sums to t)

3-CNF-SAT $\leq_p$ Subset-Sum

Reduction function *f* from 3-CNF-SAT to Subset-Sum:

- Given a 3-CNF formula $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ with literals $x_1, x_2, \ldots, x_n$, construct a Subset-Sum instance as follows:
  - Two assumptions about ϕ (WLOG): (1) no clause contains both a literal and its negation; (2) every variable appears in at least one clause.
  - The numbers in S are in base 10 and have $n+k$ digits; each digit corresponds to (or is labeled by) a literal or a clause.
  - Target t=1…1||4…4 (i.e., n 1's and k 4's)
  - For each variable x_i, create two integers:
    - v_i =0…$01_{(i)}$0…0||0…$01_{(l)}$0…$01_{(w)}$0…0, where x_i appears in $C_l,\ldots,C_w$.
    - v_i'=0…$01_{(i)}$0…0||0…$1_{(m)}$0……$01_{(p)}$0…0, where $\neg x_i$ appears in $C_m,\ldots,C_p$.
    - Clearly, v_i and v_i' can not be equal in right k digits; moreover all v_i and v_i' in S are distinct.
  - For each clause C_j, create two integers:
    - s_j=0…0||0…$01_{(j)}$0…0,
    - s_j'=0…0||0…$02_{(j)}$0…0.
    - all s_j and s_j' are called "slack variables". Clearly, all s_j and s_j' in S are distinct.
  - Note: the sum of digits in any one digit position is 2 or 6, so there will never be carries when adding any subset of the above integers.

Example:

$\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$
$C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$
$C_3 = (\neg x_1 \vee \neg x_2 \vee x_3)$
$C_4 = (x_1 \vee x_2 \vee x_3)$

Satisfying assignment for ϕ :
$x_1 = 0, x_2 = 0, x_3 = 1$

S= {all 14 numbers in table}

t = 1114444

Satisfying assignment for <S, t>
is lightly shaded numbers from S

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v'_1	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v'_2	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v'_3	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

Subset-Sum is NP Complete

Now, prove correctness of *f:*

First, prove reduction is poly time:

Should be apparent – set contains $2n + 2k$ values; each has $n + k$ digits; time to produce each is polynomial in $n + k$.

Then, prove reduction is correct:

- The 3-CNF formula ϕ is satisfiable iff there is a subset S' of S whose sum is t.

$\Rightarrow$ Suppose ϕ has a satisfying assignment.

- Then for i=1,…,n, if x_i=1 in the assignment, then v_i is put in S', otherwise, v_i' is put in S'.
- The digits labeled by literals will sum to 1.
- Moreover, for each digit labeled by a clause C_j , there may be 1, 2, or 3 assignments that are 1.
- Correspondingly, both s_j and s_j' , or s_j', or $s_{j,}$ is added to S' to make the sum of the digits be 4.
- So S' will sum to 1…1 4…4.

$\Leftarrow$ Suppose there is a S' which sums to 1…1 4…4.

- Then S' contains exactly one of v_i and v_i' for i =1,…,n. If $v_i \in$ S', then set x_i=1, otherwise, $v_i' \in$ S', so set x_i = 0.
- It can be seen that this assignment makes each clause of ϕ evaluate to 1. So ϕ is satisfiable.

Since (1) Subset-Sum $\in$ NP, and (2) 3-CNF-SAT $\leq_p$ Subset-Sum, then Subset-Sum is NP-Complete

Hamiltonian Cycle is NP-Complete

Hamiltonian Cycle Problem: Given a graph <G>, does it contain a cycle that includes all of the vertices in G?

Proving NP-Completeness:

Step 1: Ham-Cycle $\in$ NP

Argue that, given a certificate, you can verify that the certificate provides a solution in polynomial time

Step 2: Show that some known NP-Complete problem is reducible in poly-time to Ham-Cycle (i.e., A $\leq_p$ Ham-Cycle)

What known NP-Complete problem do we choose?

Vertex-Cover $\leq_p$ Ham-Cycle

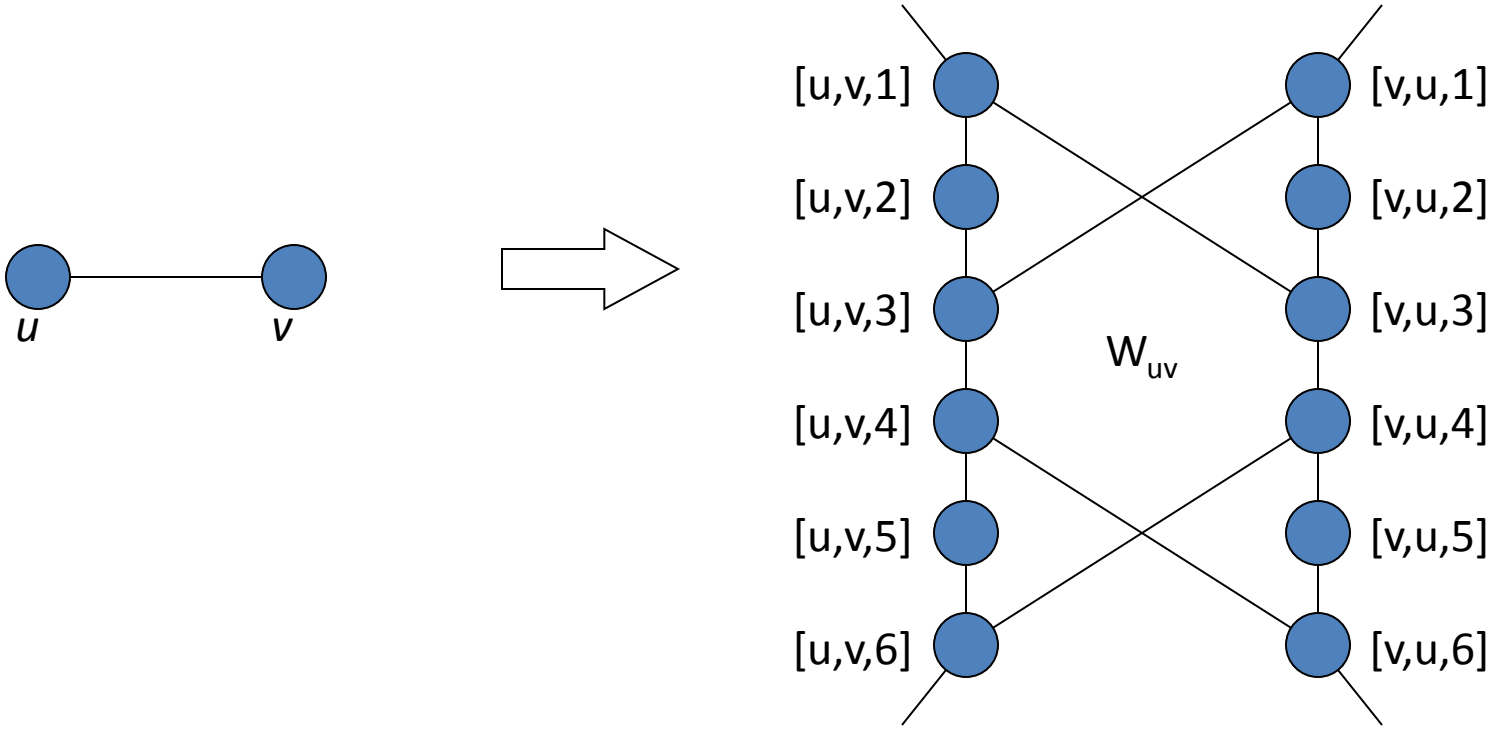
What do we have to do?

1) Given an instance < G, k > of Vertex-Cover, define poly-time function *f* that converts < G, k > to instance < G' > of Ham-Cycle
2) Argue that *f* is poly-time
3) Argue that *f* is correct (i.e., G has a vertex cover of size k iff G' has a Hamiltonian Cycle)

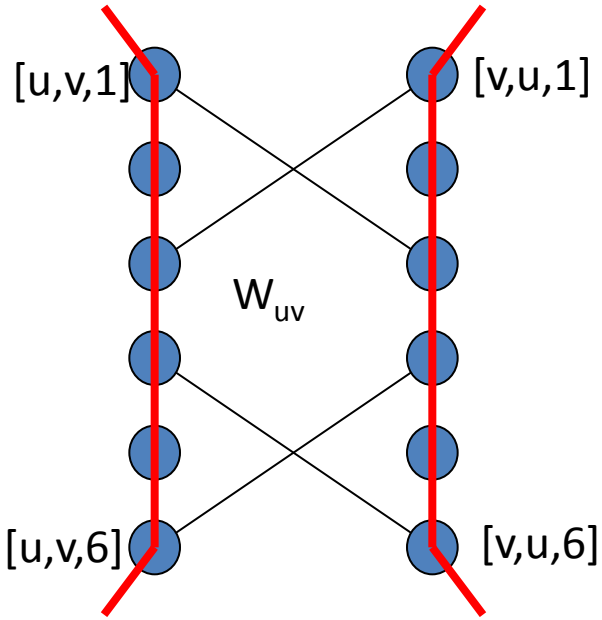
Vertex-Cover $\leq_p$ Ham-Cycle

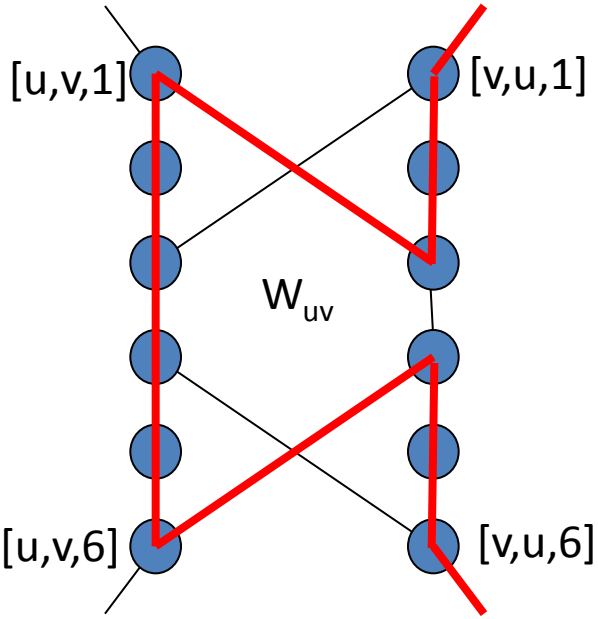
Reduction function *f* from Vertex-Cover to Ham-Cycle:

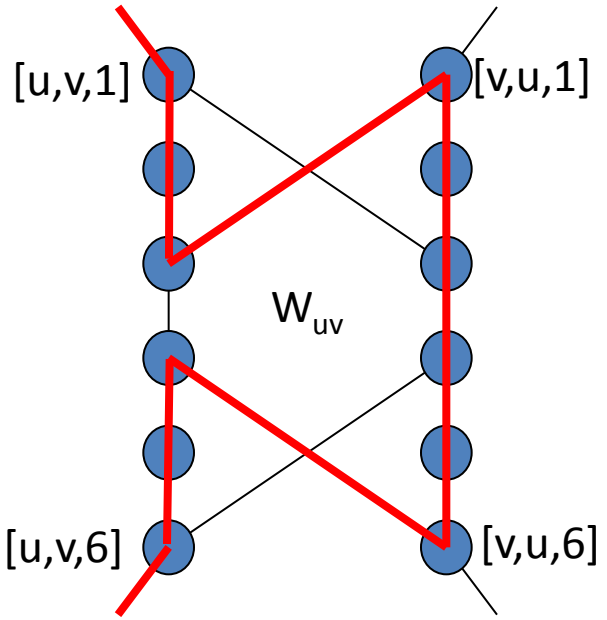
For every edge in the Vertex Cover problem (in G), we must reduce it to a "widget" in the Hamiltonian Cycle Problem (in G'):



Observations….

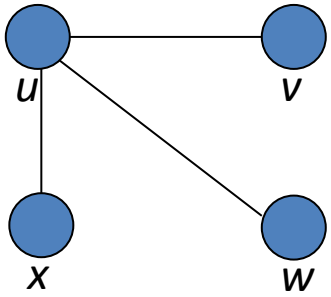






There are only three possible ways that a cycle can include all of the vertices in this widget.

Joining Widgets

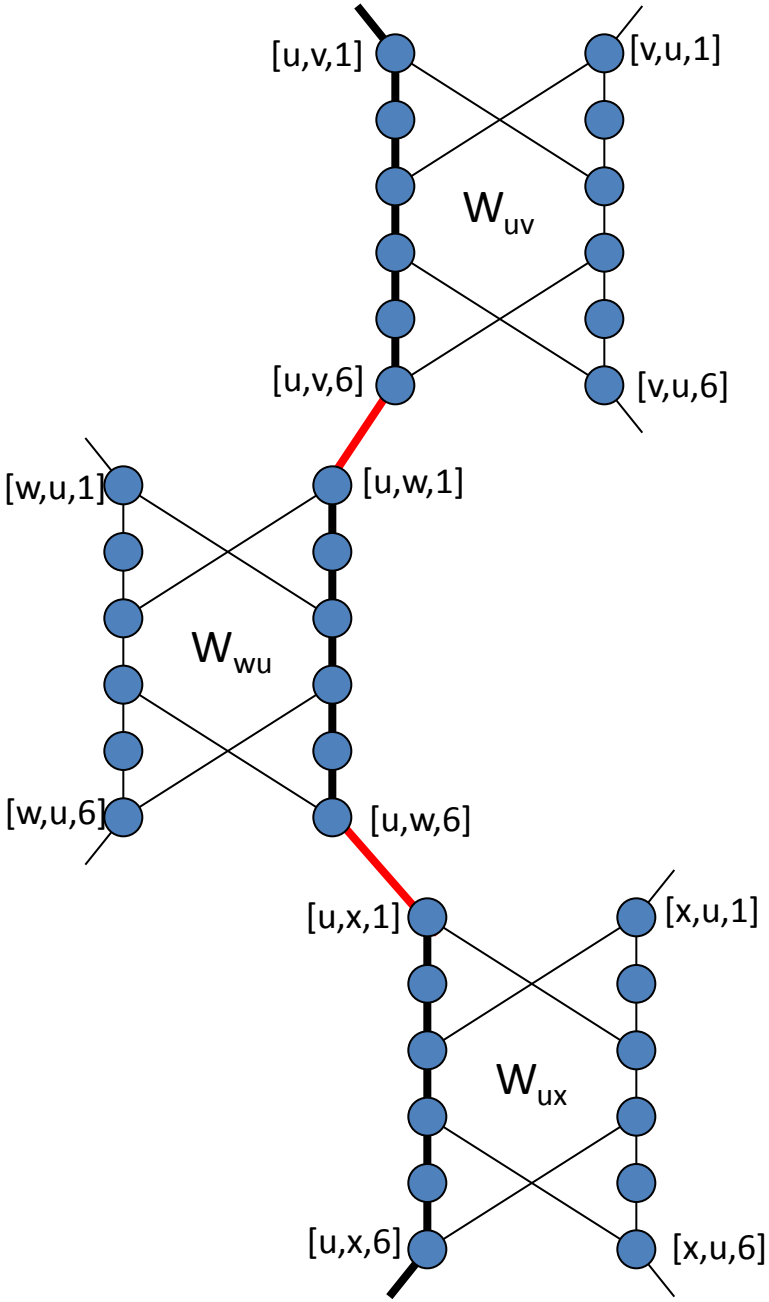


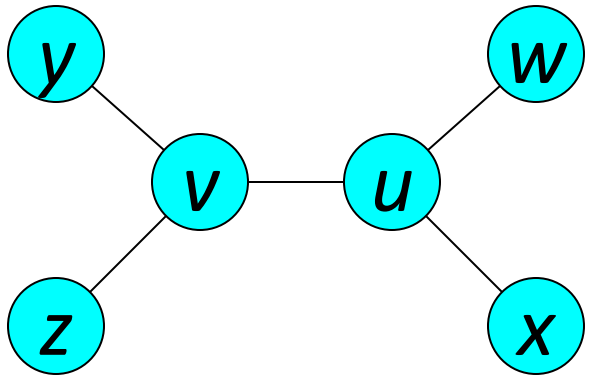
Adjacency:
u → v, w, x

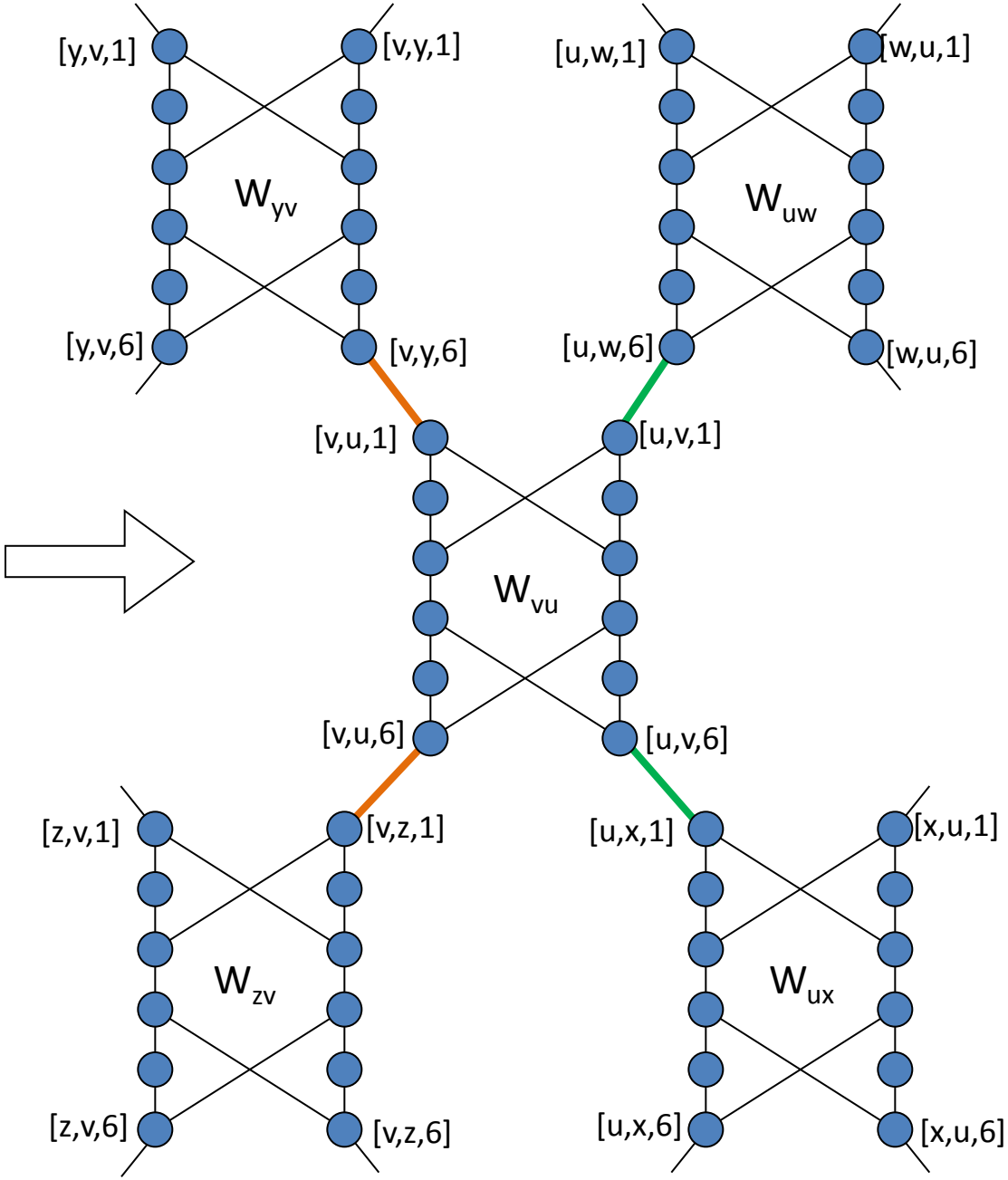
Means we connect:
[u,v,6] with [u,w,1],
[u,w,6] with [u,x,1]

All widgets that represent edges adjacent to *u* are strung together into a chain, according to an arbitrary ordering of the adjacent vertices.

We also have to connect chains to selector vertices (coming up soon…)







Adjacencies:

y → v (no additional edge needed)

v → y, u, z

u → w, v, x

w → u (no additional edge needed)

x → u (no additional edge needed)

z → v (no additional edge needed)

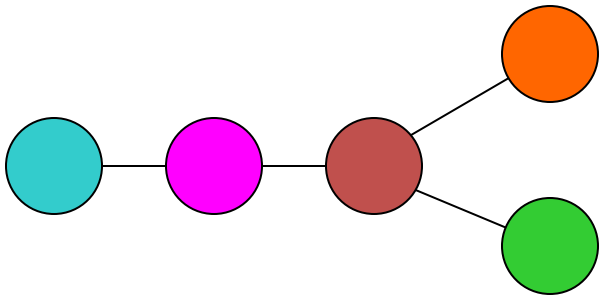
Tying the Chains Together

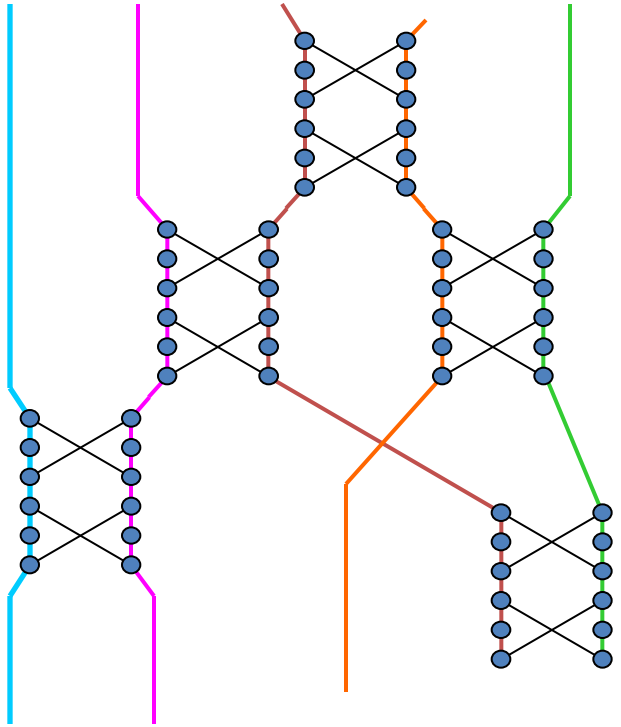
Since we want to know if its possible to cover the original graph using only *k* vertices, this would be the same as seeing if we can include all of the vertices using only *k* chains.

How can we include exactly *k* chains in the Hamiltonian Cycle problem?

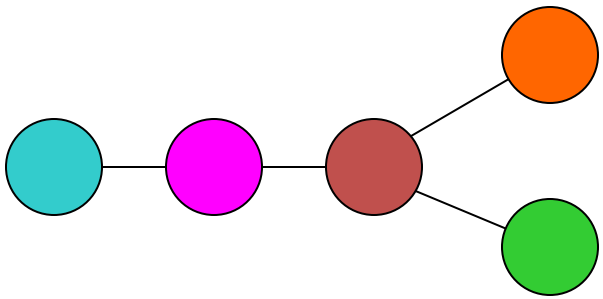
We must add *k* extra vertices and connect each of them to the beginning and end of every chain. Since each vertex can only be included once, this allows *k* chains in the final cycle.

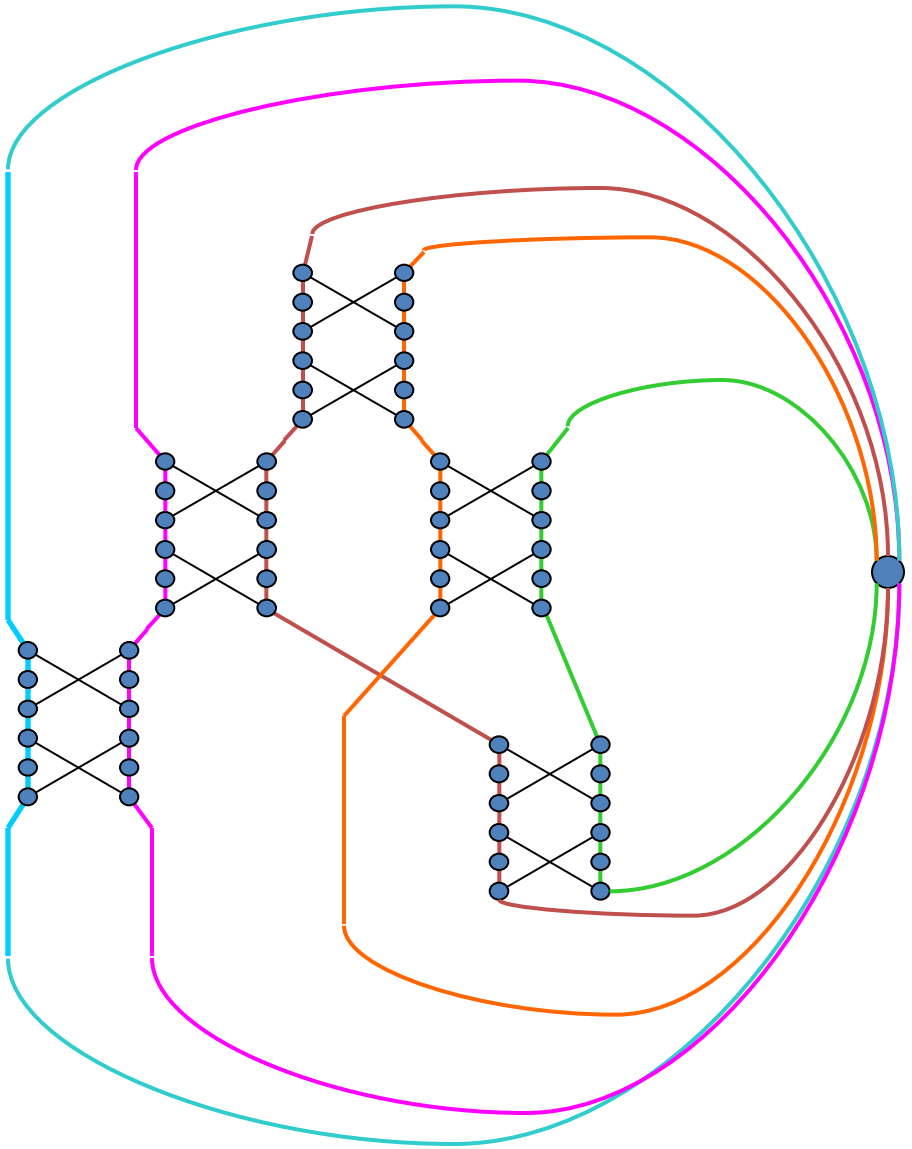
Beginning a Transform

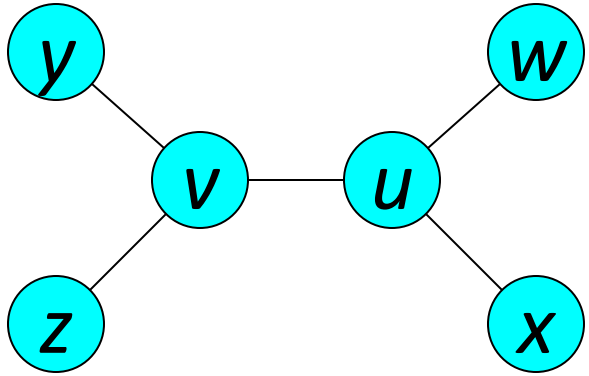


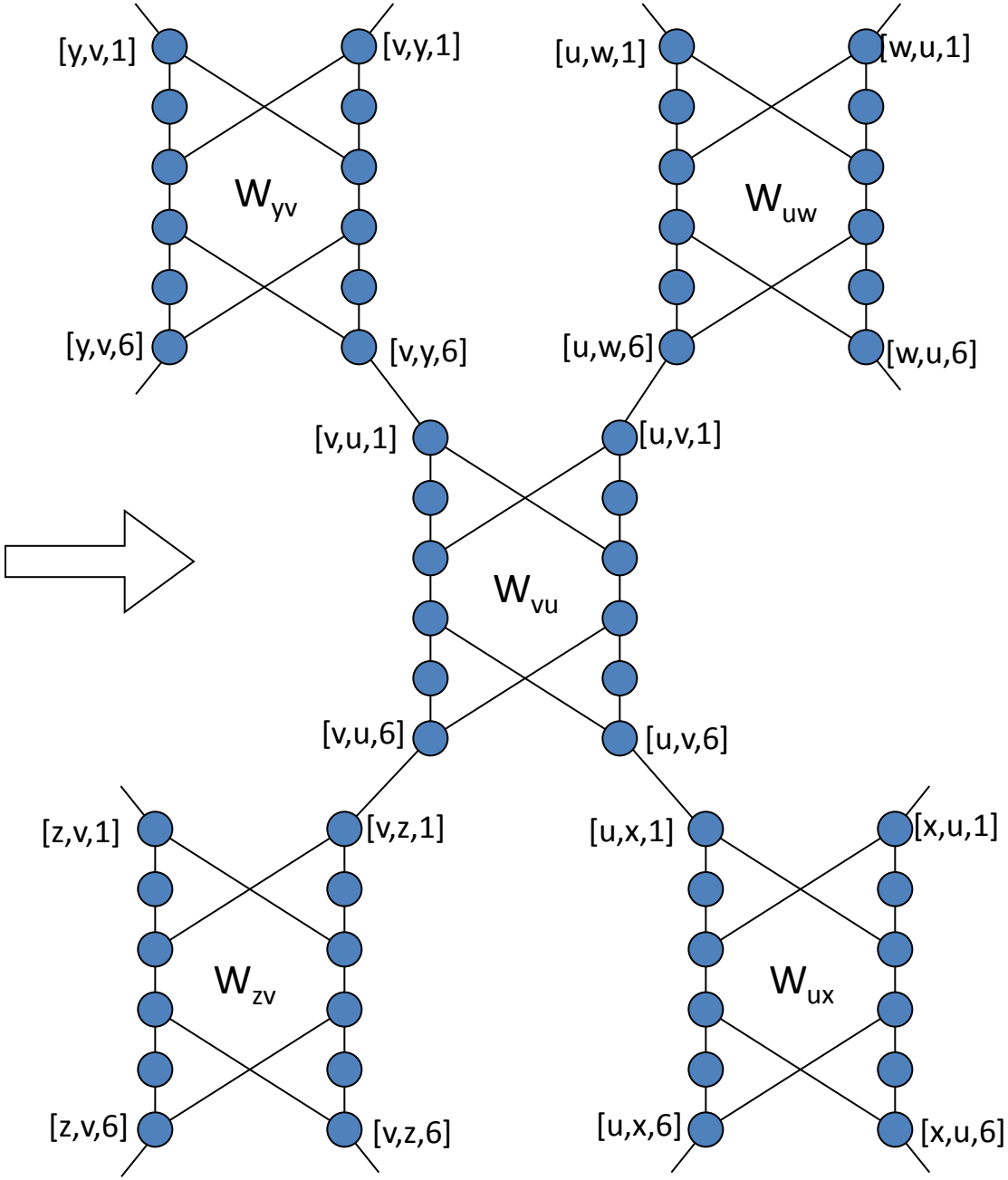


The Final Transform



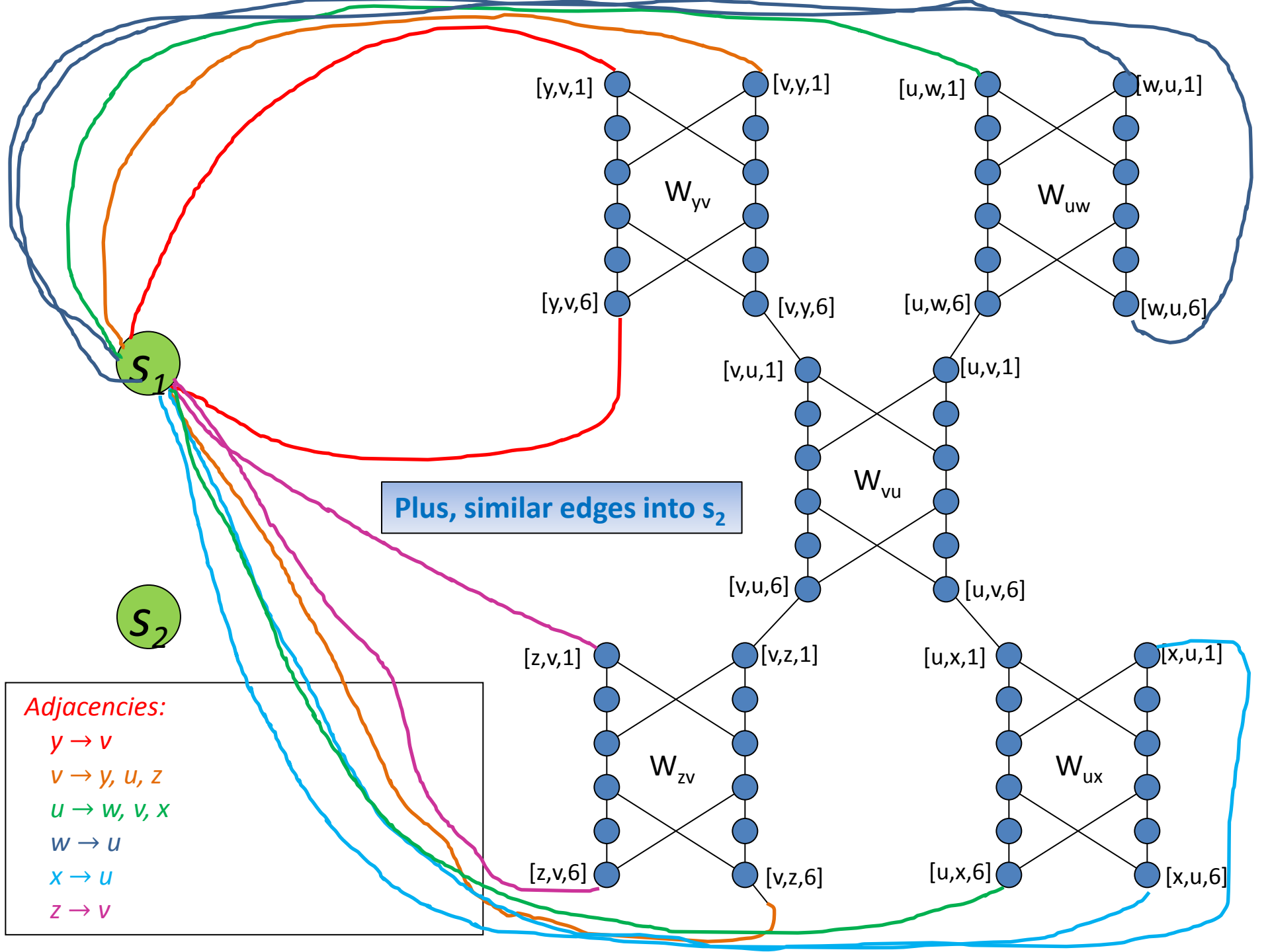


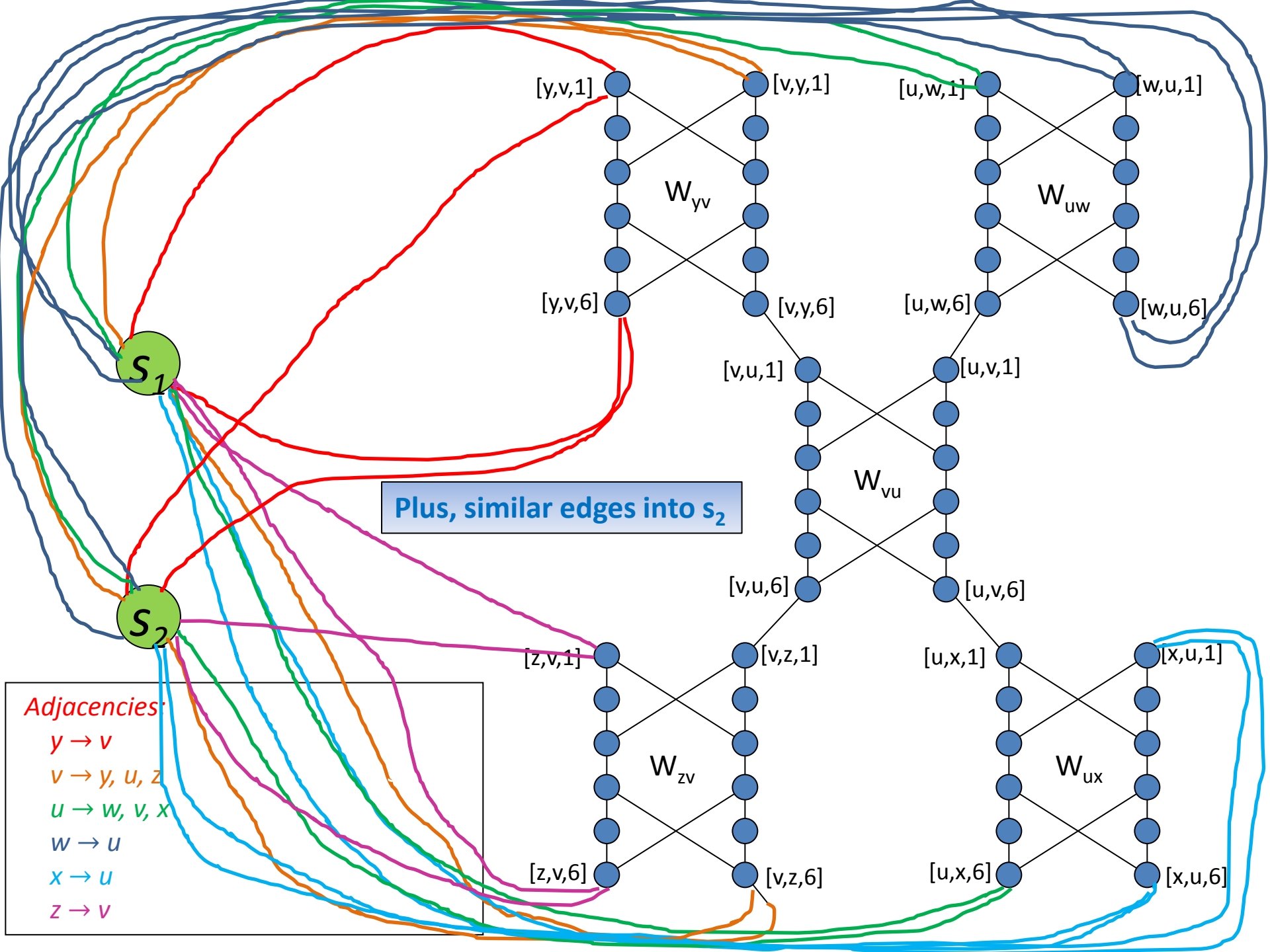




Adjacencies:

- *y* → *v*
- *v* → *y, u, z*
- *u* → *w, v, x*
- *w* → *u*
- *x* → *u*
- *z* → *v*


[y,v,1]
[v,y,1]
[u,w,1]
[w,u,1]
W_{yv}
W_{uw}
[y,v,6]
[v,y,6]
[u,w,6]
[w,u,6]
s_1
[v,u,1]
[u,v,1]
W_{vu}
Plus, similar edges into s_2
[v,u,6]
[u,v,6]
s_2
[z,v,1]
[v,z,1]
[u,x,1]
[x,u,1]
W_{zv}
W_{ux}
[z,v,6]
[v,z,6]
[u,x,6]
[x,u,6]
Adjacencies:
y → v
v → y, u, z
u → w, v, x
w → u
x → u
z → v


[y,v,1]
[v,y,1]
[u,w,1]
[w,u,1]
W_yv
W_uw
[y,v,6]
[v,y,6]
[u,w,6]
[w,u,1]
[v,u,1]
[u,v,1]
W_vu
Plus, similar edges into s_2
[v,u,6]
[u,v,6]
s_1
s_2
[z,v,1]
[v,z,1]
[u,x,1]
[x,u,1]
W_zv
W_ux
[z,v,6]
[v,z,6]
[u,x,6]
[x,u,6]
Adjacencies:
y → v
v → y, u, z
u → w, v, x
w → u
x → u
z → v

Vertex-Cover $\leq_p$ Ham-Cycle

Now, prove correctness of *f*:

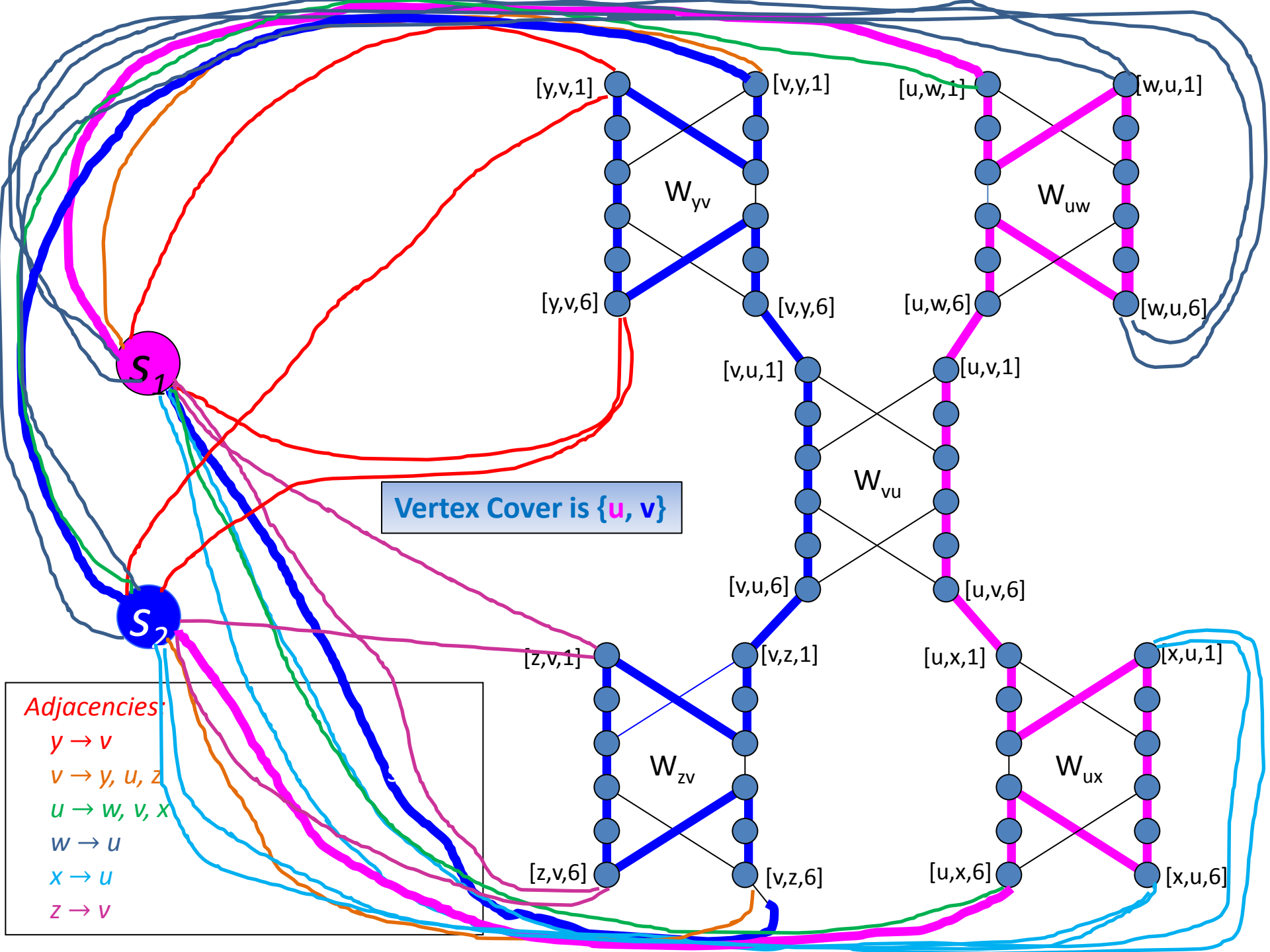
- First, prove reduction is poly time:
  - # vertices in G':
    - There are 12 vertices for each edge in G, plus k selector vertices (where k < |V|)
    - So, we have $|V'| \leq 12|E| + k$
  - # edges in G':
    - $14|E|$ for widgets
    - $\sum_{u \in V}(\text{degree}(u) - 1) = 2|E| - |V|$ edges between widgets
    - 2 edges connecting each selector vertex and each vertex
    - So, we have $|E'| \leq 16|E| + (2|V| - 1)|V|$

Vertex-Cover $\leq_p$ Ham-Cycle

Then, prove reduction is correct:

$\Rightarrow$ If G has a vertex cover of size k, then *G'* has a Hamiltonian Cycle.

Given the vertex cover, we can find the Hamiltonian Cycle – see next slide.

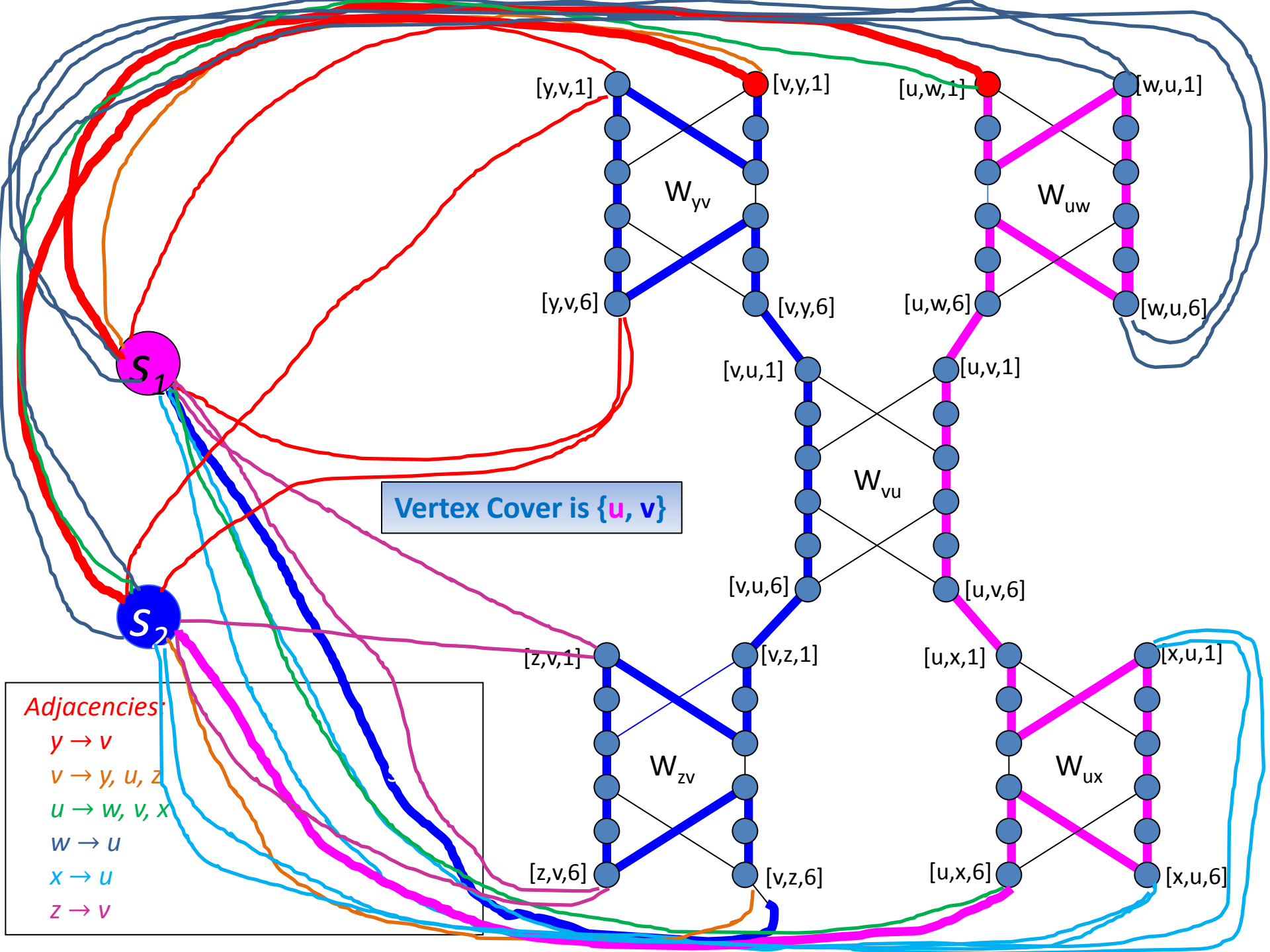

[y,v,1]
[v,y,1]
[u,w,1]
[w,u,1]
W_{yv}
W_{uw}
[y,v,6]
[v,y,6]
[u,w,6]
[w,u,6]
S_1
[v,u,1]
[u,v,1]
W_{vu}
Vertex Cover is {u, v}
[v,u,6]
[u,v,6]
S_2
[z,v,1]
[v,z,1]
[u,x,1]
[x,u,1]
W_{zv}
W_{ux}
[z,v,6]
[v,z,6]
[u,x,6]
[x,u,6]
Adjacencies:
y → v
v → y, u, z
u → w, v, x
w → u
x → u
z → v

Vertex-Cover $\leq_p$ Ham-Cycle

Then, prove reduction is correct:

$\Rightarrow$ If G has a vertex cover of size k, then *G'* has a Hamiltonian Cycle. Given the vertex cover, we can find the Hamiltonian Cycle

$\Leftarrow$ If G' has a Hamiltonian Cycle, then G has vertex cover of size k. We can select the vertex cover as the vertices u connected from s_j to [u, _, 1] (see next slide)


[y,v,1]
[v,y,1]
[u,w,1]
[w,u,1]
W_{yv}
W_{uw}
[y,v,6]
[v,y,6]
[u,w,6]
[w,u,6]
S_1
[v,u,1]
[u,v,1]
W_{vu}
Vertex Cover is {u, v}
[v,u,6]
[u,v,6]
S_2
[z,v,1]
[v,z,1]
[u,x,1]
[x,u,1]
W_{zv}
W_{ux}
[z,v,6]
[v,z,6]
[u,x,6]
[x,u,6]
Adjacencies:
y → v
v → y, u, z
u → w, v, x
w → u
x → u
z → v

Vertex-Cover $\leq_p$ Ham-Cycle

Then, prove reduction is correct:

$\Rightarrow$ If G has a vertex cover of size k, then *G'* has a Hamiltonian Cycle. Given the vertex cover, we can find the Hamiltonian Cycle

$\Leftarrow$ If G' has a Hamiltonian Cycle, then G has vertex cover of size k. We can select the vertex cover as the vertices u connected from s_j to [u, _, 1] (see next slide)

Since (1) Ham-Cycle $\in$ NP, and (2) Vertex-Cover $\leq_p$ Ham-Cycle, then Ham-Cycle is NP-Complete. [WHEW!]

Traveling-salesman problem is NP Complete

TSP Problem:

Instance of TSP: <G,*c*,*k*>, where:

G=(V,E) is a complete graph,

c is a function from $V \times V \rightarrow Z$,

$k \in Z$, and G has a traveling salesman tour with cost at most *k*.}

Proving NP-Completeness:

- Step 1: TSP $\in$ NP
  - Argument: given a certificate of a sequence of vertices in the tour, the verifying algorithm checks whether each vertex appears once, and that the sum of the costs is at most *k*. Can be done in poly-time.

- Step 2: Show that some known NP-Complete problem is reducible in poly-time to TSP (i.e., $A \leq_p$ TSP)

  - What known NP-Complete problem do we choose?

Ham-Cycle $\leq_p$ TSP

- What do we have to do?
  1) Given an instance < G > of Ham-Cycle, define poly-time function *f* that converts < G > to instance < G', c, k > of TSP
  2) Argue that *f* is poly-time
  3) Argue that *f* is correct (i.e., G has a Hamiltonian Cycle iff G has a traveling salesman tour of weight at most k, according to cost function c)

Ham-Cycle $\leq_p$ TSP

Reduction function *f* from Ham-Cycle to TSP:

- Given an instance G=(V,E) of HAM-CYCLE, construct a TSP instance <G',c,0) as follows:
  - G'=(V,E'), where E'={<*i*,*j*>: *i*,*j*$\in$ V and *i*$\neq$*j*}
  - Cost function *c* is defined as:

$$c(i,j) = \begin{cases} 0 & \text{if } (i,j) \in E \\ 1 & \text{if } (i,j) \notin E \end{cases}$$

Ham-Cycle $\leq_p$ TSP

Now, prove correctness of *f*:

- First, prove reduction is poly time:
  - Size of G' is polynomial in G. Cost function is polynomial in G.

Then, prove reduction is correct:

$\Rightarrow$ If G has a Hamiltonian cycle *h*, then *h* is also a tour in G' with cost at most 0.

$\Leftarrow$ If G' has a tour *h*' of cost at most 0, then each edge in *h*' has weight 0. This means that each edge belongs to E, so *h*' is also a Hamiltonian cycle in G.

Since (1) TSP $\in$ NP, and (2) Ham-Cycle $\leq_p$ TSP then TSP is NP-Complete.

The Art of Proving Hardness

Proving that problems are hard is a skill. Once you get the hang of it, it becomes much more straightforward and intuitive.

Indeed, the hidden secret of NP-completeness proofs is that they are usually easier to recreate than explain, in the same way that it is usually easier to rewrite old code than to try to understand it.

Guideline 1

Make your source problem as simple as possible.

Never try to reduce the general *Traveling Salesman Problem* to prove hardness. Better, use *Hamiltonian Cycle*. Even better, don't worry about closing the cycle, and use *Hamiltonian Path*.

If you are aware of simpler NP-Complete problems, you should always use them instead of their more complex variants. When reducing Hamiltonian Path, you could actually demand the graph to be directed, planar or even 3-regular if any of these make an easier reduction (since all these variants are also NP-Complete).

Guideline 2

Make your target problem as hard as possible.

Don't be afraid to add extra constraints or freedoms in order to make your problem more general.

Perhaps you are trying to prove a problem NP-Complete on an undirected graph. If you can prove it using a directed graph, do so, and then come back and try to simplify the target, modifying your proof. Once you have one working proof, it is often (but not always) much easier to produce a related one.

Guideline 3

Select the right source problem for the right reason.

3-SAT: The old reliable. When none of the other problems seem to work, this is the one to come back to.

Vertex Cover: This is the answer for any graph problems whose hardness depends upon *selection*.

Hamiltonian Path: This is the proper choice for most problems whose answer depends upon *ordering*.

Integer Partition: This is the primary choice for problems whose hardness requires using large numbers.

> [In the Integer Partition problem, you're given a set of positive integers S, and you want to know whether S can be partitioned into 2 subsets S_1 and S_2, such that the sums of S_1 and S_2 are the same.]

Guideline 4

Amplify the penalties for making the undesired selection.

If you want to remove certain possibilities from being considered, it is usually possible to assign extreme values to them, such as zero or infinity.

For example, we can show that the Traveling Salesman Problem is still hard on a complete graph by assigning a weight of infinity to those edges that we don't want used.

Guideline 5

Think strategically at a high level, and then build gadgets to enforce tactics.

You should be asking yourself the following types of questions: "How can I force that either A or B, but not both are chosen?" "How can I force that A is taken before B?" "How can I clean up the things that I did not select?"

After you have an idea of what you want your gadgets to do, you can start to worry about how to craft them. The reduction to Hamiltonian Path is a perfect example.

Guideline 6

When you get stuck, alternate between looking for an algorithm or a reduction.

Sometimes the reason you cannot prove hardness is that there exists an efficient algorithm that will solve your problem! Techniques such as dynamic programming or reducing to polynomial time graph problems sometimes yield surprising polynomial time algorithms.

Whenever you can't prove hardness, it likely pays to alter your opinion occasionally to keep yourself honest.

Reading Assignments

- Next class:
  - Chapter 34.3
    - Circuit-SAT proof of NP-Completeness