COMP 761: Lecture 33 – Computational Complexity

David Rolnick

November 20, 2020

Problem

Is there an efficient algorithm that determines whether an undirected, unweighted graph has a cycle that includes every vertex exactly once?

(Please don't post your ideas in the chat just yet, we'll discuss the problem soon in class.)

Course Announcements

Decision problems

Decision problems

- Today, we'll mostly be looking at *decision problems* – i.e. yes/no questions like "is this number prime?" or "does this graph have a cycle that hits every vertex exactly once?"

Decision problems

- Today, we'll mostly be looking at *decision problems* – i.e. yes/no questions like "is this number prime?" or "does this graph have a cycle that hits every vertex exactly once?"
- The *computational complexity* of a problem refers to how hard it is to solve that problem.

Decision problems

- Today, we'll mostly be looking at *decision problems* – i.e. yes/no questions like "is this number prime?" or "does this graph have a cycle that hits every vertex exactly once?"
- The *computational complexity* of a problem refers to how hard it is to solve that problem.
- Proving that something can be solved fast generally means building an algorithm that solves it exactly that fast.

Decision problems

- Today, we'll mostly be looking at *decision problems* – i.e. yes/no questions like "is this number prime?" or "does this graph have a cycle that hits every vertex exactly once?"
- The *computational complexity* of a problem refers to how hard it is to solve that problem.
- Proving that something can be solved fast generally means building an algorithm that solves it exactly that fast.
- Proving that something *cannot* be solved fast is sometimes harder.

The class P

The class P

- We have seen a lot of algorithms that run in time $\Theta(n)$ or $\Theta(n \log n)$ or $\Theta(n^2)$ or $\Theta(n^3)$, where n is the size of the input.

The class P

- We have seen a lot of algorithms that run in time $\Theta(n)$ or $\Theta(n \log n)$ or $\Theta(n^2)$ or $\Theta(n^3)$, where n is the size of the input.
- All of these algorithms are *polynomial time*.

The class P

- We have seen a lot of algorithms that run in time $\Theta(n)$ or $\Theta(n \log n)$ or $\Theta(n^2)$ or $\Theta(n^3)$, where n is the size of the input.
- All of these algorithms are *polynomial time*.
- More formally, we say that an algorithm runs in *polynomial time* if the running time is $O(n^d)$ for some constant d.

The class P

- We have seen a lot of algorithms that run in time $\Theta(n)$ or $\Theta(n \log n)$ or $\Theta(n^2)$ or $\Theta(n^3)$, where n is the size of the input.
- All of these algorithms are *polynomial time*.
- More formally, we say that an algorithm runs in *polynomial time* if the running time is $O(n^d)$ for some constant d.
- Note that, for example, $\Theta(n \log n)$ algorithms are polynomial time because they are $O(n^2)$.

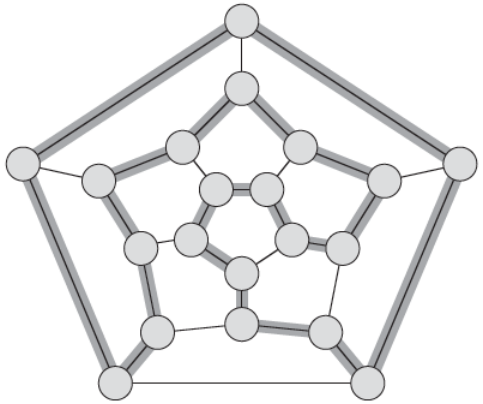
The class P

- We have seen a lot of algorithms that run in time $\Theta(n)$ or $\Theta(n \log n)$ or $\Theta(n^2)$ or $\Theta(n^3)$, where n is the size of the input.
- All of these algorithms are *polynomial time*.
- More formally, we say that an algorithm runs in *polynomial time* if the running time is $O(n^d)$ for some constant d.
- Note that, for example, $\Theta(n \log n)$ algorithms are polynomial time because they are $O(n^2)$.
- We write the set of decision problems (yes/no problems) that can be solved in polynomial time as P.

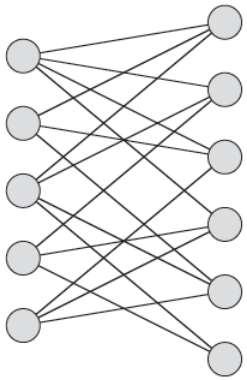
Hamiltonian cycles

Hamiltonian cycles

- A *Hamiltonian cycle* in an undirected, unweighted graph is a cycle in the graph that visits every vertex exactly once.



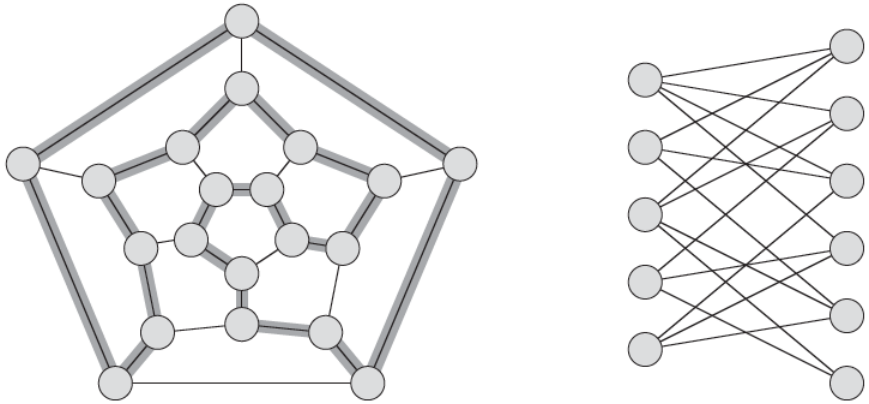
(a)



(b)

Hamiltonian cycles

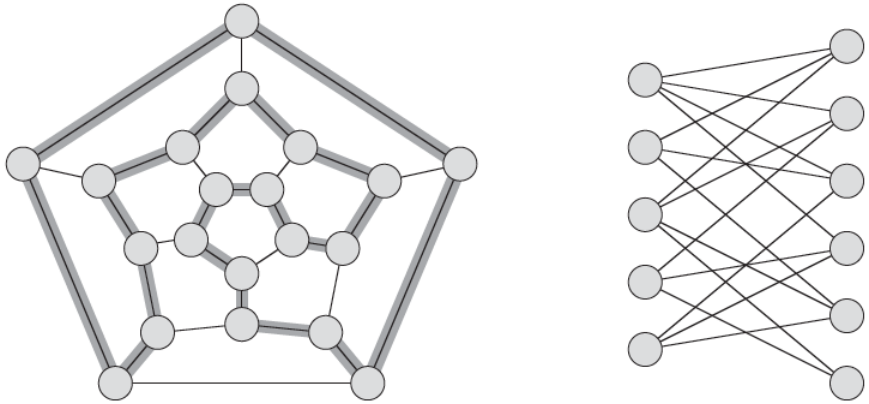
- A *Hamiltonian cycle* in an undirected, unweighted graph is a cycle in the graph that visits every vertex exactly once.



- The graph on the left has a Hamiltonian cycle, the graph on the right (bipartite with odd number of vertices) doesn't.

Hamiltonian cycles

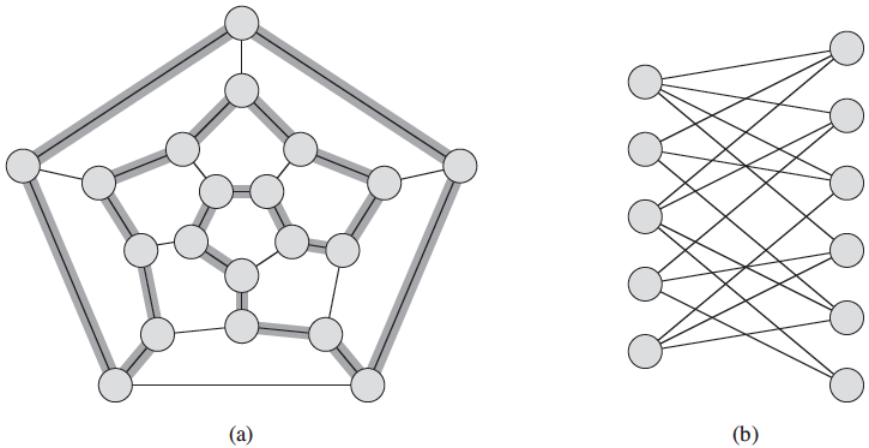
- A *Hamiltonian cycle* in an undirected, unweighted graph is a cycle in the graph that visits every vertex exactly once.



- The graph on the left has a Hamiltonian cycle, the graph on the right (bipartite with odd number of vertices) doesn't.
- This is different from the *Eulerian cycles* we saw before which have to cross every *edge* exactly once.

Hamiltonian cycles

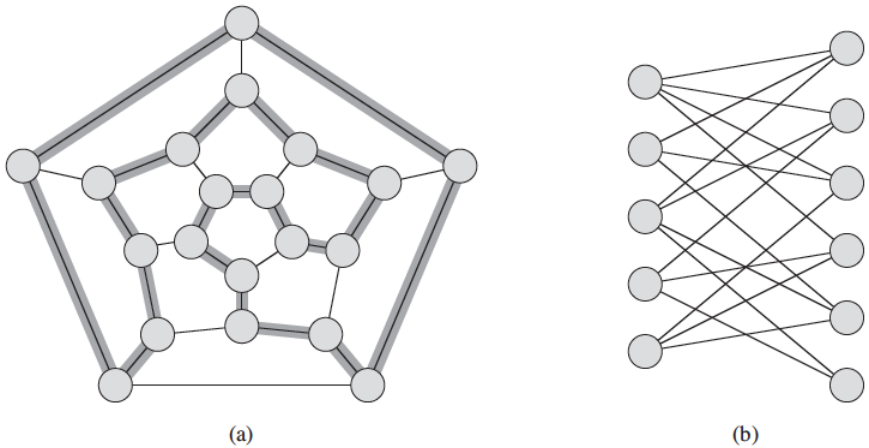
- A *Hamiltonian cycle* in an undirected, unweighted graph is a cycle in the graph that visits every vertex exactly once.



- The graph on the left has a Hamiltonian cycle, the graph on the right (bipartite with odd number of vertices) doesn't.
- This is different from the *Eulerian cycles* we saw before which have to cross every *edge* exactly once.
- Any easy way to work out whether a graph contains a Hamiltonian cycle?

Hamiltonian cycles

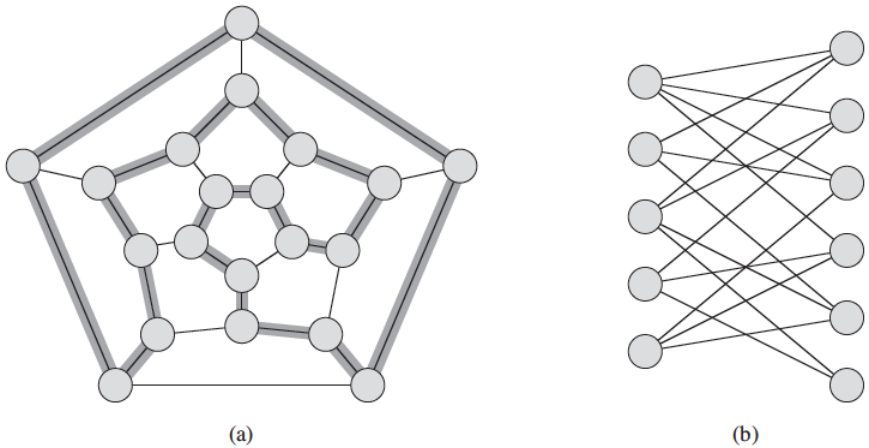
- A *Hamiltonian cycle* in an undirected, unweighted graph is a cycle in the graph that visits every vertex exactly once.



- The graph on the left has a Hamiltonian cycle, the graph on the right (bipartite with odd number of vertices) doesn't.
- This is different from the *Eulerian cycles* we saw before which have to cross every *edge* exactly once.
- Any easy way to work out whether a graph contains a Hamiltonian cycle?
- Turns out there (probably) aren't any.

Hamiltonian cycles

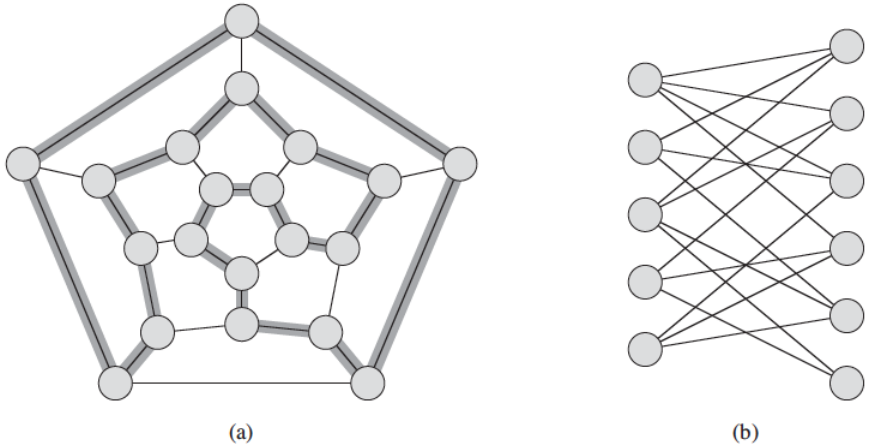
- A *Hamiltonian cycle* in an undirected, unweighted graph is a cycle in the graph that visits every vertex exactly once.



- The graph on the left has a Hamiltonian cycle, the graph on the right (bipartite with odd number of vertices) doesn't.
- This is different from the *Eulerian cycles* we saw before which have to cross every *edge* exactly once.
- Any easy way to work out whether a graph contains a Hamiltonian cycle?
- Turns out there (probably) aren't any.

Hamiltonian cycles

- A *Hamiltonian cycle* in an undirected, unweighted graph is a cycle in the graph that visits every vertex exactly once.



- The graph on the left has a Hamiltonian cycle, the graph on the right (bipartite with odd number of vertices) doesn't.
- This is different from the *Eulerian cycles* we saw before which have to cross every *edge* exactly once.
- Any easy way to work out whether a graph contains a Hamiltonian cycle?
- Turns out there (probably) aren't any.
- We write HAM-CYCLE(G) as the problem of determining whether or not there is a Hamiltonian cycle in the graph G.

The class NP

The class NP

- We saw before that a problem was in the class P if the answer (yes/no) can be found in polynomial time.

The class NP

- We saw before that a problem was in the class P if the answer (yes/no) can be found in polynomial time.
- We say that a problem is in *NP* if for every "yes" answer, there exists a *certificate* that proves it is indeed "yes" and can be checked in polynomial time.

The class NP

- We saw before that a problem was in the class P if the answer (yes/no) can be found in polynomial time.
- We say that a problem is in *NP* if for every "yes" answer, there exists a *certificate* that proves it is indeed "yes" and can be checked in polynomial time.
- I.e. it is possible to convince someone that the answer is "yes" in polynomial time.

The class NP

- We saw before that a problem was in the class P if the answer (yes/no) can be found in polynomial time.
- We say that a problem is in *NP* if for every "yes" answer, there exists a *certificate* that proves it is indeed "yes" and can be checked in polynomial time.
- I.e. it is possible to convince someone that the answer is "yes" in polynomial time.
- So HAM-CYCLE is an example of a problem in NP – the Hamiltonian cycle is itself a certificate that such a cycle exists.

The class NP

- We saw before that a problem was in the class P if the answer (yes/no) can be found in polynomial time.
- We say that a problem is in *NP* if for every "yes" answer, there exists a *certificate* that proves it is indeed "yes" and can be checked in polynomial time.
- I.e. it is possible to convince someone that the answer is "yes" in polynomial time.
- So HAM-CYCLE is an example of a problem in NP – the Hamiltonian cycle is itself a certificate that such a cycle exists.
- It is not known whether HAM-CYCLE is also in P.

The class NP

- We saw before that a problem was in the class P if the answer (yes/no) can be found in polynomial time.
- We say that a problem is in *NP* if for every "yes" answer, there exists a *certificate* that proves it is indeed "yes" and can be checked in polynomial time.
- I.e. it is possible to convince someone that the answer is "yes" in polynomial time.
- So HAM-CYCLE is an example of a problem in NP – the Hamiltonian cycle is itself a certificate that such a cycle exists.
- It is not known whether HAM-CYCLE is also in P.
- We say that a problem is in *co-NP* if for every "no" answer, there is a certificate that proves it is indeed "no" and can be checked in polynomial time.

The class NP

- We saw before that a problem was in the class P if the answer (yes/no) can be found in polynomial time.
- We say that a problem is in *NP* if for every "yes" answer, there exists a *certificate* that proves it is indeed "yes" and can be checked in polynomial time.
- I.e. it is possible to convince someone that the answer is "yes" in polynomial time.
- So HAM-CYCLE is an example of a problem in NP – the Hamiltonian cycle is itself a certificate that such a cycle exists.
- It is not known whether HAM-CYCLE is also in P.
- We say that a problem is in *co-NP* if for every "no" answer, there is a certificate that proves it is indeed "no" and can be checked in polynomial time.
- I.e. it is possible to convince someone that the answer is "no" in polynomial time.

The class NP

- We saw before that a problem was in the class P if the answer (yes/no) can be found in polynomial time.
- We say that a problem is in *NP* if for every "yes" answer, there exists a *certificate* that proves it is indeed "yes" and can be checked in polynomial time.
- I.e. it is possible to convince someone that the answer is "yes" in polynomial time.
- So HAM-CYCLE is an example of a problem in NP – the Hamiltonian cycle is itself a certificate that such a cycle exists.
- It is not known whether HAM-CYCLE is also in P.
- We say that a problem is in *co-NP* if for every "no" answer, there is a certificate that proves it is indeed "no" and can be checked in polynomial time.
- I.e. it is possible to convince someone that the answer is "no" in polynomial time.
- Is HAM-CYCLE also in co-NP?

The class NP

- We saw before that a problem was in the class P if the answer (yes/no) can be found in polynomial time.
- We say that a problem is in *NP* if for every "yes" answer, there exists a *certificate* that proves it is indeed "yes" and can be checked in polynomial time.
- I.e. it is possible to convince someone that the answer is "yes" in polynomial time.
- So HAM-CYCLE is an example of a problem in NP – the Hamiltonian cycle is itself a certificate that such a cycle exists.
- It is not known whether HAM-CYCLE is also in P.
- We say that a problem is in *co-NP* if for every "no" answer, there is a certificate that proves it is indeed "no" and can be checked in polynomial time.
- I.e. it is possible to convince someone that the answer is "no" in polynomial time.
- Is HAM-CYCLE also in co-NP?
- This is also unknown! Just not having found a cycle isn't a good certificate that there isn't one.

The class NP

The class NP

- Class P: the answer (yes/no) can be found in polynomial time.

The class NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every “yes” answer, there is a *certificate* that proves it is indeed “yes” and can be *checked* in polynomial time.

The class NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every “yes” answer, there is a *certificate* that proves it is indeed “yes” and can be *checked* in polynomial time.
- Is a problem in P necessarily in NP?

The class NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every "yes" answer, there is a *certificate* that proves it is indeed "yes" and can be *checked* in polynomial time.
- Is a problem in P necessarily in NP?
- Yes, since finding the answer takes polynomial time, even ignoring the certificate.

The class NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every "yes" answer, there is a *certificate* that proves it is indeed "yes" and can be *checked* in polynomial time.
- Is a problem in P necessarily in NP?
- Yes, since finding the answer takes polynomial time, even ignoring the certificate.
- The reverse is not known though: Is every problem in NP also in P?

The class NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every "yes" answer, there is a *certificate* that proves it is indeed "yes" and can be *checked* in polynomial time.
- Is a problem in P necessarily in NP?
- Yes, since finding the answer takes polynomial time, even ignoring the certificate.
- The reverse is not known though: Is every problem in NP also in P?
- This is the famous P vs. NP problem. Most people believe the answer is *no*, that there are problems in NP that aren't in P.

The class NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every "yes" answer, there is a *certificate* that proves it is indeed "yes" and can be *checked* in polynomial time.
- Is a problem in P necessarily in NP?
- Yes, since finding the answer takes polynomial time, even ignoring the certificate.
- The reverse is not known though: Is every problem in NP also in P?
- This is the famous P vs. NP problem. Most people believe the answer is *no*, that there are problems in NP that aren't in P.
- Note: There are also problems which aren't in either P or NP – i.e. hard to find a solution, and also hard to verify a solution.

Reducibility

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Is B polynomial-time reducible to A?

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Is B polynomial-time reducible to A?
- Yes. Suppose we already know the solutions to A.

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Is B polynomial-time reducible to A?
- For (H, e) the input to B, let's use A to solve B.

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Is B polynomial-time reducible to A?
- For (H, e) the input to B, let's use A to solve B.
- Let H' be the graph obtained by deleting e from H.

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Is B polynomial-time reducible to A?
- For (H, e) the input to B, let's use A to solve B.
- Let H' be the graph obtained by deleting e from H.
- We have access to the answers $A(H)$ and $A(H')$.

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Is B polynomial-time reducible to A?
- For (H, e) the input to B, let's use A to solve B.
- Let H' be the graph obtained by deleting e from H.
- We have access to the answers $A(H)$ and $A(H')$.
- What happens if $A(H)$ is "yes" and $A(H')$ is "no"?

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Is B polynomial-time reducible to A?
- For (H, e) the input to B, let's use A to solve B.
- Let H' be the graph obtained by deleting e from H.
- We have access to the answers $A(H)$ and $A(H')$.
- What happens if $A(H)$ is "yes" and $A(H')$ is "no"?
- We know that $B(H, e)$ must be "yes", since there were Hamiltonian cycles in H but now there are none if we take out e.

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Is B polynomial-time reducible to A?
- For (H, e) the input to B, let's use A to solve B.
- Let H' be the graph obtained by deleting e from H.
- We have access to the answers $A(H)$ and $A(H')$.
- What happens if $A(H)$ is "yes" and $A(H')$ is "no"?
- We know that $B(H, e)$ must be "yes", since there were Hamiltonian cycles in H but now there are none if we take out e.
- Likewise, if $A(H)$ is "yes" and $A(H')$ is "yes", then $B(H, e)$ is "no".

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Is B polynomial-time reducible to A?
- For (H, e) the input to B, let's use A to solve B.
- Let H' be the graph obtained by deleting e from H.
- We have access to the answers $A(H)$ and $A(H')$.
- What happens if $A(H)$ is "yes" and $A(H')$ is "no"?
- We know that $B(H, e)$ must be "yes", since there were Hamiltonian cycles in H but now there are none if we take out e.
- Likewise, if $A(H)$ is "yes" and $A(H')$ is "yes", then $B(H, e)$ is "no".
- What if $A(H)$ is "no"?

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Let H' be the graph obtained by deleting e from H.
- We have access to the answers $A(H)$ and $A(H')$.
- What happens if $A(H)$ is "yes" and $A(H')$ is "no"?
- We know that $B(H, e)$ must be "yes", since there were Hamiltonian cycles in H but now there are none if we take out e.
- Likewise, if $A(H)$ is "yes" and $A(H')$ is "yes", then $B(H, e)$ is "no".
- If $A(H)$ is "no", then $B(H, e)$ is "yes", since technically every Hamiltonian cycle in H contains e, because there aren't any Hamiltonian cycles :)

Reducibility

- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- For example, let $A(G)$ be the problem HAM-CYCLE(G) for a graph G.
- Let $B(H, e)$ be the problem of working out whether every Hamiltonian cycle in the graph H *contains the particular edge* e.
- Let H' be the graph obtained by deleting e from H.
- We have access to the answers $A(H)$ and $A(H')$.
- What happens if $A(H)$ is "yes" and $A(H')$ is "no"?
- We know that $B(H, e)$ must be "yes", since there were Hamiltonian cycles in H but now there are none if we take out e.
- Likewise, if $A(H)$ is "yes" and $A(H')$ is "yes", then $B(H, e)$ is "no".
- If $A(H)$ is "no", then $B(H, e)$ is "yes", since technically every Hamiltonian cycle in H contains e, because there aren't any Hamiltonian cycles :)
- This shows also how if we know the answer to A, we can actually work out *exactly where a Hamiltonian cycle is* since we can one-by-one find edges e that aren't necessary and delete them.

NP-complete

NP-complete

- Since we don't know if P = NP, we would like to have a way of saying "This problem is in NP and is really hard unless P = NP."

NP-complete

- Since we don't know if P = NP, we would like to have a way of saying "This problem is in NP and is really hard unless P = NP."
- This is made more formal by the notion of NP-completeness.

NP-complete

- Since we don't know if P = NP, we would like to have a way of saying "This problem is in NP and is really hard unless P = NP."
- This is made more formal by the notion of NP-completeness.
- We say a problem *A* is *NP-complete* if (a) it is in NP, and (b) every problem *B* in NP is polynomial-time reducible to *A*.

NP-complete

- Since we don't know if P = NP, we would like to have a way of saying "This problem is in NP and is really hard unless P = NP."
- This is made more formal by the notion of NP-completeness.
- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- So in some sense A is just as hard as every other problem in NP.

NP-complete

- Since we don't know if P = NP, we would like to have a way of saying "This problem is in NP and is really hard unless P = NP."
- This is made more formal by the notion of NP-completeness.
- We say a problem *A* is *NP-complete* if (a) it is in NP, and (b) every problem *B* in NP is polynomial-time reducible to *A*.
- So in some sense *A* is just as hard as every other problem in NP.
- You may have also heard the term *NP-hard*. That is just condition (b) without condition (a) needing to be true. So a problem that is NP-hard might or might not itself be in NP, but it is possible to reduce any problem in NP to it.

NP-complete

- Since we don't know if P = NP, we would like to have a way of saying "This problem is in NP and is really hard unless P = NP."
- This is made more formal by the notion of NP-completeness.
- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- So in some sense A is just as hard as every other problem in NP.
- You may have also heard the term *NP-hard*. That is just condition (b) without condition (a) needing to be true. So a problem that is NP-hard might or might not itself be in NP, but it is possible to reduce any problem in NP to it.
- (So every NP-complete problem is also NP-hard.)

NP-complete

- Since we don't know if P = NP, we would like to have a way of saying "This problem is in NP and is really hard unless P = NP."
- This is made more formal by the notion of NP-completeness.
- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- So in some sense A is just as hard as every other problem in NP.
- You may have also heard the term *NP-hard*. That is just condition (b) without condition (a) needing to be true. So a problem that is NP-hard might or might not itself be in NP, but it is possible to reduce any problem in NP to it.
- (So every NP-complete problem is also NP-hard.)
- HAM-CYCLE is an example of a problem that is NP-complete (we won't prove this, but the proof isn't too bad).

NP-complete

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- Suppose that A is NP-complete and B is in NP.

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- Suppose that A is NP-complete and B is in NP.
- We know that there is a polynomial-time reduction from B to A.

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- Suppose that A is NP-complete and B is in NP.
- We know that there is a polynomial-time reduction from B to A.
- What happens if there is a polynomial-time reduction from A to B?

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- Suppose that A is NP-complete and B is in NP.
- We know that there is a polynomial-time reduction from B to A.
- What happens if there is a polynomial-time reduction from A to B?
- Then B is NP-complete too!

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- Suppose that A is NP-complete and B is in NP.
- We know that there is a polynomial-time reduction from B to A.
- What happens if there is a polynomial-time reduction from A to B?
- Then B is NP-complete too!
- We can see this by bringing in a new problem C in NP.

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- Suppose that A is NP-complete and B is in NP.
- We know that there is a polynomial-time reduction from B to A.
- What happens if there is a polynomial-time reduction from A to B?
- Then B is NP-complete too!
- We can see this by bringing in a new problem C in NP.
- To prove B is NP-complete, we need to show C is polynomial-time reducible to B. How can we do it?

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- Suppose that A is NP-complete and B is in NP.
- We know that there is a polynomial-time reduction from B to A.
- What happens if there is a polynomial-time reduction from A to B?
- Then B is NP-complete too!
- We can see this by bringing in a new problem C in NP.
- To prove B is NP-complete, we need to show C is polynomial-time reducible to B. How can we do it?
- Since A is NP-complete, we know C is polynomial-time reducible to A.

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- Suppose that A is NP-complete and B is in NP.
- We know that there is a polynomial-time reduction from B to A.
- What happens if there is a polynomial-time reduction from A to B?
- Then B is NP-complete too!
- We can see this by bringing in a new problem C in NP.
- To prove B is NP-complete, we need to show C is polynomial-time reducible to B. How can we do it?
- Since A is NP-complete, we know C is polynomial-time reducible to A.
- And we also know A is polynomial-time reducible to B.

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- Suppose that A is NP-complete and B is in NP.
- We know that there is a polynomial-time reduction from B to A.
- What happens if there is a polynomial-time reduction from A to B?
- Then B is NP-complete too!
- We can see this by bringing in a new problem C in NP.
- To prove B is NP-complete, we need to show C is polynomial-time reducible to B. How can we do it?
- Since A is NP-complete, we know C is polynomial-time reducible to A.
- And we also know A is polynomial-time reducible to B.
- So we can use solutions to B to get solutions to A, and then use those to get solutions to C.

NP-complete

- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- Suppose that A is NP-complete and B is in NP.
- We know that there is a polynomial-time reduction from B to A.
- What happens if there is a polynomial-time reduction from A to B?
- Then B is NP-complete too!
- We can see this by bringing in a new problem C in NP.
- To prove B is NP-complete, we need to show C is polynomial-time reducible to B. How can we do it?
- Since A is NP-complete, we know C is polynomial-time reducible to A.
- And we also know A is polynomial-time reducible to B.
- So we can use solutions to B to get solutions to A, and then use those to get solutions to C.
- Since each of those reductions takes polynomial time, the overall reduction must take polynomial time – hence C is polynomial-reducible to B, so B is NP-complete.

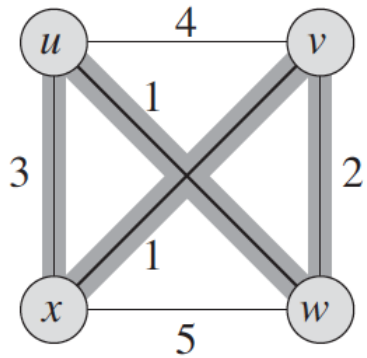
Traveling salesman

Traveling salesman

- In a weighted graph, the *weight* of a Hamiltonian cycle is the sum of the weights on the edges (just as with a path).

Traveling salesman

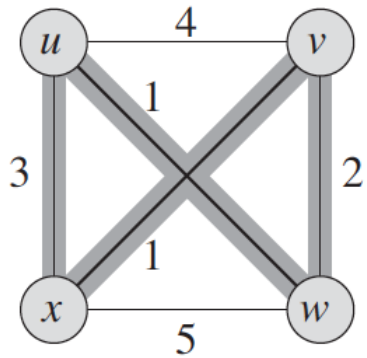
- In a weighted graph, the *weight* of a Hamiltonian cycle is the sum of the weights on the edges (just as with a path).
- In the *traveling salesman problem* (TSP), we are given an undirected, weighted graph and would like to find the smallest weight of a Hamiltonian cycle in the graph.



- In order to make TSP into a decision problem, we can frame it as a yes-no question: Is there a Hamiltonian cycle with weight at most k? (We could repeat this question with different k to work out the smallest weight.)

Traveling salesman

- In a weighted graph, the *weight* of a Hamiltonian cycle is the sum of the weights on the edges (just as with a path).
- In the *traveling salesman problem* (TSP), we are given an undirected, weighted graph and would like to find the smallest weight of a Hamiltonian cycle in the graph.



- In order to make TSP into a decision problem, we can frame it as a yes-no question: Is there a Hamiltonian cycle with weight at most k? (We could repeat this question with different k to work out the smallest weight.)
- Applications of TSP in logistics and transportation, DNA sequencing, even astronomy (efficient ways of moving a telescope to look at different points in the sky).

Traveling salesman

Traveling salesman

- TSP(G, k): is there a Hamiltonian cycle in G with weight at most k?

Traveling salesman

- TSP(G, k): is there a Hamiltonian cycle in G with weight at most k?
- How can we prove TSP is NP-complete?

Traveling salesman

- TSP(G, k): is there a Hamiltonian cycle in G with weight at most k?
- How can we prove TSP is NP-complete?
- First, we need to prove it is in NP. Why is it in NP?

Traveling salesman

- TSP(G, k): is there a Hamiltonian cycle in G with weight at most k?
- How can we prove TSP is NP-complete?
- First, we need to prove it is in NP. Why is it in NP?
- Any Hamiltonian cycle of weight at most k is a valid certificate – it can easily be checked in polynomial time.

Traveling salesman

- TSP(G, k): is there a Hamiltonian cycle in G with weight at most k?
- How can we prove TSP is NP-complete?
- First, we need to prove it is in NP. Why is it in NP?
- Any Hamiltonian cycle of weight at most k is a valid certificate – it can easily be checked in polynomial time.
- What can we do now?

Traveling salesman

- TSP(G, k): is there a Hamiltonian cycle in G with weight at most k?
- How can we prove TSP is NP-complete?
- First, we need to prove it is in NP. Why is it in NP?
- Any Hamiltonian cycle of weight at most k is a valid certificate – it can easily be checked in polynomial time.
- What can we do now?
- We know that if an NP-complete problem is polynomial-time reducible to TSP, then TSP must also be NP-complete.

Traveling salesman

- TSP(G, k): is there a Hamiltonian cycle in G with weight at most k?
- How can we prove TSP is NP-complete?
- First, we need to prove it is in NP. Why is it in NP?
- Any Hamiltonian cycle of weight at most k is a valid certificate – it can easily be checked in polynomial time.
- What can we do now?
- We know that if an NP-complete problem is polynomial-time reducible to TSP, then TSP must also be NP-complete.
- HAM-CYCLE is NP-complete – can we reduce it to TSP (and if so how)?

Traveling salesman

- TSP(G, k): is there a Hamiltonian cycle in G with weight at most k?
- How can we prove TSP is NP-complete?
- First, we need to prove it is in NP. Why is it in NP?
- Any Hamiltonian cycle of weight at most k is a valid certificate – it can easily be checked in polynomial time.
- What can we do now?
- We know that if an NP-complete problem is polynomial-time reducible to TSP, then TSP must also be NP-complete.
- HAM-CYCLE is NP-complete – can we reduce it to TSP (and if so how)?
- Absolutely! Given a graph H where we want to compute HAM-CYCLE(H), we can just set all the weights equal to 1 to give a weighted graph G.

Traveling salesman

- TSP(G, k): is there a Hamiltonian cycle in G with weight at most k?
- How can we prove TSP is NP-complete?
- First, we need to prove it is in NP. Why is it in NP?
- Any Hamiltonian cycle of weight at most k is a valid certificate – it can easily be checked in polynomial time.
- What can we do now?
- We know that if an NP-complete problem is polynomial-time reducible to TSP, then TSP must also be NP-complete.
- HAM-CYCLE is NP-complete – can we reduce it to TSP (and if so how)?
- Absolutely! Given a graph H where we want to compute HAM-CYCLE(H), we can just set all the weights equal to 1 to give a weighted graph G.
- Then, HAM-CYCLE(H) equals TSP(G, ∞), where we have set $k = \infty$.

Traveling salesman

- TSP(G, k): is there a Hamiltonian cycle in G with weight at most k?
- How can we prove TSP is NP-complete?
- First, we need to prove it is in NP. Why is it in NP?
- Any Hamiltonian cycle of weight at most k is a valid certificate – it can easily be checked in polynomial time.
- What can we do now?
- We know that if an NP-complete problem is polynomial-time reducible to TSP, then TSP must also be NP-complete.
- HAM-CYCLE is NP-complete – can we reduce it to TSP (and if so how)?
- Absolutely! Given a graph H where we want to compute HAM-CYCLE(H), we can just set all the weights equal to 1 to give a weighted graph G.
- Then, HAM-CYCLE(H) equals TSP(G, ∞), where we have set $k = \infty$.
- This reduction is clearly polynomial-time (the only time is in setting up the weighted graph).